Type Assignement for Mobile Objects

Franco Barbanera [1,3]

Dipartimento di Matematica
Università di Catania
Catania, Italy

Ugo de'Liguoro [2,4]

Dipartimento di Informatica
Università di Torino
Torino, Italy

Abstract

We address the problem of formal reasoning about mobile code. We consider an Ambient Calculus, where process syntax includes constructs for object-oriented sequential programming. For the sake of concreteness, and because of practical relevance, we consider objects using message exchange to implement method invocation and overriding. The main contribution of the paper is an intersection type assignment system, obtained by a combination of systems introduced for the Calculus of Mobile Ambients and for the ς-calculus. We exploit, in the mobility framework, a typical feature of the intersection type discipline for object calculi, namely late typing of self. The proposed system is then checked against standard properties of related systems, establishing type invariance and a completeness theorem.

Key words: objects, ambients, mobility, intersection, types.

1 Introduction

Many theoretical models have been proposed to attack the urgent problem of understanding "global computing", as it has been recently dubbed the scenario in which programmers have to cope with a plethora of computing devices

[1] Partially supported by MURST Cofin'01 NAPOLI
[2] Partially supported by MURST Cofin'01 COMETA and Cofin'02 PROTOCOLLO Projects, IST-2001-33477 DART Project and IST-2001-322222 MIKADO Project. The funding bodies are not responsible for any use that might be made of the results presented here.
[3] Email: barba@dmi.unict.it
[4] Email: deliguoro@di.unito.it

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

and a great variety of interconnections, messages and code exchanges. These models, however, stemming from Milner π-calculus or from the Ambient Calculus of [CG00], do not directly take into account sequential procedures. Any reasonable language for designing and programming mobile systems needs to have a sequential component. As a matter of fact actual languages designed for the network have a sequential core and a set of primitives for communication, concurrency and mobility. In the models mentioned above, sequential procedures have to be simulated via heavy encodings.

A possible reaction is the one pursued e.g. in [BCC01], were the formalization of object-oriented concepts, borrowed from calculi like the ς-calculus of [AC96], are strongly tied to the formalizations of distributed and mobile aspects, coming from the calculus of Mobile Ambients (MA). Another possible approach is to keep ambients and sequential code each other apart, as it is the case of the λ-terms inside ambients in [AKPG01]. The latter choice has the advantage of conceptual clarity, which is reflected by the type system and (denotational) semantics.

We shall describe a calculus where the sequential and the mobile/distributed components are loosely coupled, that is where objects and ambients have different syntax and operational semantics; but, much as for "concurrent objects" in [Lan01], objects are allowed to interact with processes and to drive mobility of ambients.

The contribution of the paper, however, is not the calculus itself, but rather a type assignment system. This is mainly because what we intend to focus on is partial (semantical) information that one can get about programs, rather than ensuring global properties of the system (e.g. well-formedness, error freeness, etc.), which is typical of typed calculi a' la Church. Therefore our calculus is type free, and types, which are better understood as predicates, are deduced for terms a posteriori.

As a matter of fact we also try to partially reflect in the type system the "loosely coupling" aspect of distributed/mobile and sequential features of the calculus, borrowing from type assignment systems for ς-calculus objects proposed in [dL01] and from the system for ambients introduced in [CD01]. The same approach should apply in the case of other sequential components of ambients, e.g. functional programs.

From the work on type assignment for sequential objects, developed in works by the second author of the present paper [dL01,vBdL03] we retain the idea of "late typing of the self". This consists in typing objects as records of methods; methods are typed by recording pre and post-conditions (the antecedent and the consequent of an arrow type respectively); it is only when a method is invoked that the system checks if the object satisfies the precondition of that particular method, which is actually a precondition of the self variable. This is contrasted to what we call the "early typing of the self", consisting in the typing policy adopted in all type systems for objects, like those in [AC96], where all relevant information about the self variables needs

to be checked prior to any possible method invocation. Early typing of the self leads to difficulties that have been faced in the literature by means of polymorphic types, as it is the case of the subtyping and matching polymorphism in [BCC01].

The main result in the paper is that types catch computational properties of terms because the system defines a model of the calculus. It is a logical model (namely a filter model), where each term denotes the sets of its properties, that is of its types; dually, types denote sets of terms sharing certain relevant properties. This is the contents of our completeness theorem, which is also the basic tool to characterize the behavior of terms via their typings.

2 A calculus of Mobile Objects

Definition 2.1 The set $\mathcal{P}$ of process terms of the *Mobile Ambients* calculus *MA* [CG00] (omitting restriction $(\nu n)P$) is defined by the grammar:

$$P, Q ::= \mathbf{0} \mid (P \mid Q) \mid \mathbf{!}P \mid n[\, P \,] \mid M \,.\, P$$

where n ranges over a denumerable set of names, and

$$M ::= \mathbf{in}\, n \mid \mathbf{out}\, n \mid \mathbf{open}\, n$$

The operational semantics of the MA calculus is given in terms of a structural congruence relation $\equiv$, making $\mid$ into an associative and commutative operator with unit $\mathbf{0}$ and such that $\mathbf{!}P \equiv P \mid \mathbf{!}P$, and a reduction relation.

Definition 2.2 Over $\mathcal{P}$ a reduction relation $\longrightarrow$ is defined by the following clauses:

$$(R - \mathbf{in}) \quad n[\, \mathbf{in}\, m \,.\, P \mid Q \,] \mid m[\, R \,] \longrightarrow m[\, n[\, P \mid Q \,] \mid R \,]$$

$$(R - \mathbf{out}) \quad m[\, n[\, \mathbf{out}\, m \,.\, P \mid Q \,] \mid R \,] \longrightarrow n[\, P \mid Q \,] \mid m[\, R \,]$$

$$(R - \mathbf{open}) \quad \mathbf{open}\, n \,.\, P \mid n[\, Q \,] \longrightarrow P \mid Q$$

$$(R - Par) \quad P \longrightarrow Q \implies P \mid R \longrightarrow Q \mid R$$

$$(R - Amb) \quad P \longrightarrow Q \implies n[\, P \,] \longrightarrow n[\, Q \,]$$

$$(R - \equiv) \quad P \equiv P', P \longrightarrow Q, Q \equiv Q' \implies P' \longrightarrow Q'$$

In [CG00] the extension of MA with communication primitives amounts to the redefinition of the action part M in the prefix operator by adding send and receive actions of the asynchronous π-calculus. Besides, we need to add the proper clauses to the definition of the reduction relation.

In order to embody objects in MA, several possibilities are open. In [BCC01] objects are ambients with two fields: the first one is a list of methods, the second is a process. Overriding is then implemented by means of the **open** primitive.

To the other extreme one might follow [AKPG01], where MA is extended by redefining values that can be communicated to be (first order typed) λ-terms instead of plain names. It is not difficult to do the same using terms from any ς-calculus in [AC96].

The calculus considered here sits in between: objects are considered as special processes, and we use prefixing to exchange messages representing method invocation and overriding. Nevertheless objects retain their own syntax and operational semantics from the ς-calculus, and are not ambients. Instead, to make the calculus more expressive and allowing mobility primitives in the syntax of the "sequential" part, method bodies are processes themselves. This choice is closer to the "concurrent objects" in [Lan01].

Definition 2.3 Terms $\mathcal{P}^+$ of the MA calculus extended with objects are obtained by adding to Definition 2.1 the following clauses:

$$\begin{aligned} a &::= x \mid o \mid a.l \mid a.l \Leftarrow \varsigma(x)b \\ b &::= a \mid P \\ o &::= [l_i = \varsigma(x_i)b_i \ ^{i\in I}] \\ P &::= \ldots \mid o \mid \langle l \rangle \mid \langle l \Leftarrow \varsigma(x)b \rangle \end{aligned}$$

The operational semantics of the calculus is then given by extending Definition 2.2 by:

$$\begin{array}{lccc} (R-1) & o \mid \langle l \rangle & \longrightarrow & o.l \\ (R-2) & o \mid \langle l \Leftarrow \varsigma(x)b \rangle & \longrightarrow & o.l \Leftarrow \varsigma(x)b \\ (R-3) & [l_i = \varsigma(x_i)b_i \ ^{i\in I}].l_j & \longrightarrow & b_j\{[l_i = \varsigma(x_i)b_i \ ^{i\in I}]/x_j\} \\ (R-4) & [l_i = \varsigma(x_i)b_i \ ^{i\in I}].l_j \Leftarrow \varsigma(y)b & \longrightarrow & [l_i = \varsigma(x_i)b_i \ ^{i\in I\setminus\{j\}}, l_j = \varsigma(y)b] \end{array}$$

where in $(R-3)$ and $(R-4)$ we assume that $j \in I$.

3 A Type Assignment System

Our *types*, which could be more appropriately called *predicates*, are basically intersection types (see e.g. [DGdL98]). Their syntax comes from [CD01] for the MA terms, and from [dL01] for the ς-terms.

$$\frac{\Gamma \vdash P : \sigma}{\Gamma \vdash M\,.\,P : M\,.\,\sigma}\,(M) \qquad \frac{\Gamma \vdash P : \sigma \quad \Gamma \vdash Q : \tau}{\Gamma \vdash P \mid Q : \sigma \mid \tau}\,(|)$$

$$\frac{\Gamma \vdash P : \sigma}{\Gamma \vdash n[\,P\,] : n[\,\sigma\,]}\,(n[\ ]) \qquad \frac{\Gamma \vdash P : \sigma \quad \Gamma \vdash !P : \tau}{\Gamma \vdash !P : \sigma \mid \tau}\,(|)$$

Fig. 1. Type Inference Rules for Ambients

Definition 3.1 The set $\mathcal{T}$ of *types* is the union of *process types* $\mathcal{T}_{\mathcal{P}}$ and *functional types* $\mathcal{T}_{\mathcal{F}}$, which are defined by mutual induction according to the grammars:

$$\mathcal{T}_{\mathcal{P}} : \ \sigma, \tau ::= \omega \mid M\,.\,\sigma \mid n[\,\sigma\,] \mid (\sigma \mid \tau) \mid$$

$$\sigma \wedge \tau \mid \{l_i : \varphi_i \ ^{i \in I}\} \mid \langle l \rangle \mid \langle l \Leftarrow \ \varphi \rangle,$$

$$\mathcal{T}_{\mathcal{F}} : \varphi, \psi ::= \sigma \to \tau \mid \varphi \wedge \psi.$$

The intended meaning of types is that of predicates, namely sets of terms sharing certain relevant properties. More precisely, types are seen as partial information about the "future" of process terms, so that whenever we say: "the meaning of the type σ is the set of processes such and such", one should understand "the meaning of the type σ is the set of processes *eventually reducing* to something such and such" (among other possibilities).

$M\,.\,\sigma$ is the type of processes that may exhibit the capability M and then continue with something of type σ; $n[\ \sigma\ ]$ is the type of processes that are ambients named n, enclosing some process in σ; finally $(\sigma \mid \tau)$ is the set of processes (equivalent to) $(P \mid Q)$, where P and Q are in σ and τ respectively. The formal definition of our type inference rules for ambients is given in Figure 1, where the statements $\Gamma \vdash P : \sigma$, for $\Gamma = \{x_1 : \sigma_1, \ldots, x_k : \sigma_k\}$, have the standard meaning: P has type σ under the assumptions in the basis Γ.

Types for the object part of the calculus have a similar meaning to types in [dL01] or predicates in [vBdL03]. $\{l_i : \sigma_i \to \tau_i \ ^{i \in I}\}$ is the type of objects having at least methods labelled l_i such that, if the object itself satisfies σ_i (hence a precondition about the self), then the body will return a value satisfying τ_i (a postcondition of the method). We remark that this is a conditional statement, not involving any assumption about the actual properties of the object as a whole. This is also the basic tool to resolve the recursive nature of objects into simpler concepts. The rules for typing objects are in Figure 2, where $l_j = \varsigma(x_j)b_j \in a$ is defined as follows:

- $j \in I \implies l_j = \varsigma(x_j)b_j \ \in \ [l_i = \varsigma(x_j)b_i \ ^{i \in I}]$
- $l = \varsigma(x)b \ \in \ a.l \Leftarrow \varsigma(x)b$

$$\frac{\Gamma, x : \sigma \vdash b : \tau \qquad l = \varsigma(x)b \ \in \ a}{\Gamma \vdash a : \{l : \sigma \to \tau\}} (\{\ \}\text{-I})$$

$$\frac{\Gamma \vdash a : \{l : \sigma \to \tau\} \qquad \Gamma \vdash a : \sigma}{\Gamma \vdash a.l : \tau} (\{\ \}\text{-E})$$

Fig. 2. Type Inference Rules for objects

- $l = \varsigma(x)b \ \in \ a, l \neq l' \implies l = \varsigma(x)b \ \in \ a.l' \Leftarrow \varsigma(x')b'$

Types $\langle l \rangle$ and $\langle l \Leftarrow \varphi \rangle$ are about communication; in particular if an update message of the shape $\langle l \Leftarrow \varsigma(x)b \rangle$ is sent, then the type $\langle l \Leftarrow \sigma \to \tau \rangle$ records the pre and post conditions of the method implemented by the new body b. Rules for typing messages are in Figure 4.

The type ω, arrow types and intersection types have their standard meaning: ω is the trivial predicate "true", namely the whole set of terms; $\sigma \to \tau$ is the type of functions (in the present case methods) sending anything of type σ into something of type τ (which is our understanding of pre and postconditions); $\sigma \wedge \tau$ is the property of any terms satisfying both σ and τ, so that extensionally it is the intersection of them.

The typing rules dealing with intersection are given in the set of Logical Rules in Figure 3 in the style of the BCD type system [BCD83], where $\sigma \leq \tau$ is logical implication of predicates, hence set inclusion extensionally. The relation $\leq$ is axiomatized in Figures 5 and 6.

4 Late versus Early Typing of Self

Before embarking in the technical treatment, let us pause on some relevant aspects of the assignment system we have proposed.

The most apparent feature of the calculus is that at first glance we regard objects as records of methods. This is the content of the following rule, which

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma} (\text{Var}) \qquad \frac{}{\Gamma \vdash a : \omega} (\omega)$$

$$\frac{\Gamma \vdash a : \sigma \quad \Gamma \vdash a : \tau}{\Gamma \vdash a : \sigma \wedge \tau} (\wedge\text{-I}) \qquad \frac{\Gamma \vdash a : \sigma \quad \sigma \leq \tau}{\Gamma \vdash a : \tau} (\leq)$$

Fig. 3. Logical Type Inference Rules

$$\frac{}{\Gamma \vdash \langle l \rangle : \langle l \rangle}\ (\text{MsgSel}) \qquad \frac{\Gamma, x : \sigma \vdash b : \tau}{\Gamma \vdash \langle l \Leftarrow \varsigma(x)b \rangle : \langle l \Leftarrow \sigma \to \tau \rangle}\ (\text{MsgUpdate})$$

Fig. 4. Type Inference Rules for Communication

is admissible in the system:

$$\frac{x_j : \sigma_j \vdash b_i : \tau_j \quad \forall j \in J}{\Gamma \vdash [l_i = \varsigma(x_i)b_i \ ^{i\in I}] : \{l_j : \sigma_j \to \tau_j \ ^{j\in J}\}}\ (J \subseteq I)$$

Note that the σ_j do not need to be the same, not even equivalent each other. The admissibility follows since $\varsigma(x_j)b_j \in [l_i = \varsigma(x_i)b_i \ ^{i\in I}]$ for all $j \in J$, so that, by using Rules ($\{\ \}$-intro) and ($\wedge$-I) repeatedly, we eventually get $\Gamma \vdash [l_i = \varsigma(x_i)b_i \ ^{i\in I}] : \bigwedge_{j\in J} \sigma_j \to \tau_j$, which is equivalent to $\{l_j : \sigma_j \to \tau_j \ ^{j\in J}\}$. So we conclude by Rule ($\leq$).

Reasoning in the same way we see that also the following rule is admissible:

$$\frac{\Gamma \vdash a : \{l_i : \varphi_i \ ^{i\in I}\} \quad x : \sigma \vdash b : \tau}{\Gamma \vdash a.l_k \Leftarrow \varsigma(x)b : \{l_i : \varphi_i \ ^{i\in I\setminus\{k\}}, l_k : \sigma \to \tau\}}$$

This has the obvious advantage that we can treat uniformly the typing of an object and of the overriding operation, so that e.g. we can give the same types, say $\{l_0 : \sigma_0 \to \tau_0, l_1 : \sigma_1 \to \tau_1, l_2 : \sigma_2 \to \tau_2\}$, both to

$$o = [l_0 = \varsigma(x_0)b_0, l_1 = \varsigma(x_1)b_1, l_2 = \varsigma(x_2)b_2]$$

and to

$$o' = ([l_0 = \varsigma(x_0)b_0, l_1 = \varsigma(x_1)b'_1,]\langle l_1 \Leftarrow \varsigma(x_1)b_1 \rangle)\langle l_2 \Leftarrow \varsigma(x_2)b_2 \rangle$$

where $o' \xrightarrow{*} o$, given that $x_i : \sigma_i \vdash b_i : \tau_i$ for $i = 0, 1, 2$. Besides, observe that we may allow for the overriding of a method which is not in the object, namely for object extension, which is not possible in the ς-calculus.

It would be surely incorrect to step from $a : \{l_i : \sigma_i \to \tau_i \ ^{i\in I}\}$ to $a.l_i : \tau_i$, since we have never checked that $a : \sigma_i$, that is a (the object itself) satisfies the precondition of its method l_i. In [dL01,vBdL03] the same difficulty is solved by means of a *late typing of the self*, that is by checking the object against the precondition of the selected method just before method invocation:

$$\frac{\Gamma \vdash a : \{l_i : \sigma_i \to \tau_i \ ^{i\in I}\} \quad \Gamma \vdash a : \sigma_k}{\Gamma \vdash a.l_k : \tau_k}\ (k \in I)$$

This should be contrasted to the type systems of the ς-calculus in [AC96], where e.g. the rule to type an object $[l_i = \varsigma(x_i)b_i \ ^{i\in I}]$ by $A = [l_k : B_k \ ^{k\in I}]$ is

- Commutativity and distributivity of $|$

$$(|\,1)\ \ \sigma \mid \tau \simeq \tau \mid \sigma \qquad\qquad (|\,2)\ \ (\sigma \mid \tau) \mid \gamma \simeq \sigma \mid (\tau \mid \gamma)$$

- Axioms for ω

$$(\omega 1)\ \sigma \leq \omega \qquad (\omega 2)\ \sigma \simeq \sigma \mid \omega \qquad (\omega 3)\ \sigma \to \omega \leq \omega \to \omega$$

- Distributivity of $\wedge$

$$(a[\ \ ]\wedge)\ \ a[\,\sigma \wedge \tau\,] \simeq a[\,\sigma\,] \wedge a[\,\tau\,] \qquad (|\wedge)\ \ \sigma \mid (\tau \wedge \gamma) \simeq (\sigma \mid \tau) \wedge (\sigma \mid \gamma)$$

$$(.\wedge)\ \ \mathsf{m}.(\sigma \wedge \tau) \simeq \mathsf{m}.\sigma \wedge \mathsf{m}.\tau$$

$$(\{\ \}\wedge 1)\ \ \{l:\sigma\} \wedge \{l:\tau\} \leq \{l:\sigma \wedge \tau\} \qquad (\{\ \}\wedge 2)\ \ \{l_i:\sigma_i\ ^{i\in I}\} \simeq \textstyle\bigwedge_{i\in I}\{l_i:\sigma_i\}$$

$$(\langle\ \rangle\wedge)\ \ \langle l \Leftarrow \sigma\rangle \wedge \langle l \Leftarrow \tau\rangle \leq \langle l \Leftarrow \sigma \wedge \tau\rangle$$

- Sequentialization

$$(.\,|_1)\ \ \mathsf{m}.\sigma \mid \tau \leq \mathsf{m}.(\sigma \mid \tau) \qquad (.\,|_2)\ \ \mathsf{m}.\sigma \mid \mathsf{n}.\tau \simeq \mathsf{m}.(\sigma \mid \mathsf{n}.\tau) \wedge \mathsf{n}.(\mathsf{m}.\sigma \mid \tau)$$

- Reduction

$$(in)\ \ a[\,\mathbf{in}\,b.\sigma \mid \tau\,] \mid b[\,\gamma\,] \leq b[\,a[\,\sigma \mid \tau\,] \mid \gamma\,]$$

$$(|)\ \ a[\,b[\,\mathbf{out}\,a.\sigma \mid \tau\,] \mid \gamma\,] \leq a[\,\gamma\,] \mid b[\,\sigma \mid \tau\,]$$

$$(open)\ \mathbf{open}\,a.\sigma \mid a[\,\tau\,] \leq \sigma \mid \tau$$

$$(out\text{-}in)\ \mathbf{in}\,a.\mathbf{out}\,a.\mathbf{in}\,a.\sigma \leq \mathbf{in}\,a.\sigma$$

$$(in\text{-}out)\ \mathbf{out}\,a.\mathbf{in}\,a.\mathbf{out}\,a.\sigma \leq \mathbf{out}\,a.\sigma$$

$$(comm\text{-}1)(\{l:\sigma \to \tau\} \wedge \sigma) \mid \langle l\rangle \leq \tau$$

$$(comm\text{-}2)\{l_i:\varphi_i\ ^{i\in I}\} \mid \langle l_k \Leftarrow\ \varphi\rangle \leq \{l_i:\varphi_i\ ^{i\in I\setminus\{k\}}, l_k:\varphi\}$$

Fig. 5. Type Entailment Rules, Part I

$$\frac{E, x_i : A \vdash b_i : B_i \quad \forall i \in I}{E \vdash [l_i = \varsigma(x_i)b_i\ ^{i\in I}] : [l_k : B_k\ ^{k\in I}]}$$

That is all relevant information has to be assumed for the self variables x_i, and it is coded in their type A which has to be the same as that of the object

- Congruence

$$(cg - M[\ ]) \quad \frac{\sigma \leq \tau}{a[\,\sigma\,] \leq a[\,\tau\,]} \qquad (cg - act) \quad \frac{\sigma \leq \tau}{\mathsf{m}\,.\sigma \leq \mathsf{m}\,.\tau}$$

$$(cg - |) \quad \frac{\sigma \leq \gamma \quad \tau \leq \gamma}{\sigma \mid \tau \leq \gamma \mid \gamma} \qquad (cg - \langle\ \rangle) \quad \frac{\sigma \leq \tau}{\langle l \Leftarrow \sigma \rangle \leq \langle l \Leftarrow \tau \rangle}$$

$$(cg - \{\ \}) \quad \frac{\sigma \leq \tau}{\{l : \sigma\} \leq \{l : \tau\}}$$

- Transitivity

$$(trans) \quad \frac{\sigma \leq \tau \quad \tau \leq \gamma}{\sigma \leq \gamma}$$

- Logical

$$(\wedge - id) \quad \sigma \leq \sigma \wedge \sigma \qquad (\wedge - l) \quad \sigma \wedge \tau \leq \sigma$$

$$(\wedge - r) \quad \sigma \wedge \tau \leq \tau \qquad (\rightarrow -) \quad (\sigma \rightarrow \tau) \wedge (\sigma \rightarrow \tau') \leq \sigma \rightarrow (\tau \wedge \tau')$$

$$(\wedge - \leq) \quad \frac{\sigma \leq \sigma' \quad \tau \leq \tau'}{\sigma \wedge \tau \leq \sigma' \wedge \tau'} \qquad (\rightarrow -) \quad \frac{\sigma \geq \sigma' \quad \tau \leq \tau'}{\sigma \rightarrow \tau \leq \sigma' \rightarrow \tau'}$$

Fig. 6. Type Entailment Rules, Part II

(hence the same for all i). We call this an *early typing of the self*.

This choice makes it clear what the self variables stay for, and that objects are in fact recursive records, but causes lots of troubles when dealing with object modification and extension, which are essential features of object based languages in practice. Such problems become even more apparent when mobility is involved, since we can know in advance neither in which ambient the methods of an object will be actually invoked, nor whether the object will be modified before the invocation. Consequently, the impact of early typing of the self in such cases would be very restrictive.

As it is the case of [CD01], the essential properties of typing are invariance under structural equivalence and, more importantly, subject expansion. We have that $a \mid \langle l \rangle \longrightarrow a.l$. Assume now that $a.l : \tau$ since $a : \{l : \sigma \rightarrow \tau\} \wedge \sigma$. This implies that we can type $a \mid \langle l \rangle$ by $(\{l : \sigma \rightarrow \tau\} \wedge \sigma) \mid \langle l \rangle$. This forces to us the type inequality:

$$(\{l : \sigma \rightarrow \tau\} \wedge \sigma) \mid \langle l \rangle \leq \tau$$

which implies $a \mid \langle l \rangle : \tau$ by $(\leq)$. Similarly for overriding we have the schema:

$$\{l_i : \varphi_i \ ^{i \in I}\} \mid \langle l_k \Leftarrow \ \varphi \rangle \leq \{l_i : \varphi_i \ ^{i \in I \setminus \{k\}}, l_k : \varphi\}.$$

5 Invariance under Congruence and Expansion

To establish the invariance of types under congruence and expansion we need a first lemma, essentially reversing the derivation rules.

Lemma 5.1 (Generation Lemma)

(i) $\Gamma \vdash \mathbf{0} : \sigma$ *iff* $\sigma \simeq \omega$;

(ii) $\Gamma \vdash \mathsf{m} \,.\, P : \sigma$ *iff* $\Gamma \vdash P : \tau$ *and* $\mathsf{m} \,.\, \tau \leq \sigma$ *for some* τ;

(iii) $\Gamma \vdash a[\, P \,] : \sigma$ *iff* $\Gamma \vdash P : \tau$ *and* $a[\, \tau \,] \leq \sigma$ *for some* τ;

(iv) $\Gamma \vdash P \mid Q : \sigma$ *iff* $\Gamma \vdash P : \tau$, $\Gamma \vdash Q : \rho$ *and* $\tau \mid \rho \leq \sigma$ *for some* τ, ρ;

(v) $\Gamma \vdash !P : \sigma$ *iff* $\Gamma \vdash P : \tau_i$ *for* $(1 \leq i \leq n)$ *and* $\tau_1 \mid \ldots \mid \tau_n \leq \sigma$ *for some* $\tau_1, \ldots, \tau_n$.

(vi) $\Gamma \vdash [l_i = \varsigma(x_i) b_i \ ^{i \in I}] : \sigma$ *iff* $\Gamma, x_j : \sigma_j \vdash b_j : \tau_j$, *and* $\{l_j : \sigma_j \to \tau_j \ ^{j \in J}\} \leq \sigma$, *for some* $J \subseteq I$, σ_j *and* τ_j;

(vii) $\Gamma \vdash a.l : \tau$ *iff* $\Gamma \vdash a : \{l : \sigma \to \tau'\} \wedge \sigma$ *and* $\tau' \leq \tau$, *for some* σ *and* τ';

(viii) $\Gamma \vdash a.l_k \Leftarrow \varsigma(x) b : \sigma$ *iff* $\Gamma \vdash a : \{l_i : \sigma_i \ ^{i \in I}\}$, $\Gamma, x : \sigma' \vdash b : \tau$ *and* $\{l_i : \sigma_i \ ^{i \in I \setminus \{k\}}, l_k : \sigma' \to \tau\} \leq \sigma$, *for some* I, σ_i, σ' *and* τ;

(ix) $\gamma \vdash \langle l \rangle : \sigma$ *iff* $\langle l \rangle \leq \sigma$

(x) $\gamma \vdash \langle l \Leftarrow \varsigma(x) b \rangle : \sigma$ *iff* $\Gamma, x : \sigma' \vdash b : \tau$ *where* $\langle l \Leftarrow \sigma' \to \tau \rangle \leq \sigma$.

Lemma 5.1 is proved using similar techniques as in the λ-calculus case, e.g. in [BCD83], because rule $(\leq)$ does not commute with the arrow introduction in rule ($\{\ \}$-I). By the way, we cannot have in the present setting a conjunctive normal form of normal types as it happens in [CD01], where only the ambient part of the calculus is dealt with.

Lemma 5.2 (Subject Congruence)
If $P \equiv Q$ *and* $\Gamma \vdash P : \sigma$ *then* $\Gamma \vdash Q : \sigma$

Proof. By induction on the definition of $\equiv$, using Lemma 5.1. □

Theorem 5.3 (Subject Expansion)
If $P \xrightarrow{*} Q$ *and* $\Gamma \vdash Q : \sigma$ *then* $\Gamma \vdash P : \sigma$.

Proof. By induction on the definition of $\longrightarrow$, using Lemma 5.1 and Lemma 5.2.

□

We observe that subject reduction does not hold: $n[\, \mathbf{0} \,] \mid \mathbf{open}\, n \,.\, \mathbf{0}$ has type $n[\, \omega \,]$ (among others), but it reduces to $\mathbf{0} \mid \mathbf{0} \equiv \mathbf{0}$ which is only typable by ω or equivalent types.

6 Type Semantics and Completeness Theorem

Given that $\mathcal{P}^0$ is the set of closed terms, and $\mathcal{P}[x]$ the set of terms whose free variables are included in $\{x\}$, we define a type interpretation which is essentially based on the concept of *saturated sets* used e.g. in [Kri90]. They are subsets of $\mathcal{P}^0$, closed under expansion. This happens to be the same interpretation of types given in [dL01] to characterize convergence in the ς-calculus, and in [CD01] to model the MA calculus. The following definition is a bit more involved because we do not have λ-abstractions in the syntax to interpret arrow types, namely types in $\mathcal{T}_{\mathcal{F}}$, which occur in the typing of objects.

Definition 6.1 We define by mutual induction the maps $[\![\ ]\!]_{\mathcal{P}} : \mathcal{T}_{\mathcal{P}} \to \wp(\mathcal{P}^0)$ and $[\![\ ]\!]_{\mathcal{F}} : \mathcal{T}_{\mathcal{F}} \times \mathit{Var} \to \wp(\bigcup_{x\in \mathit{Var}} \mathcal{P}[x])$ as follows:

(i) $[\![\omega]\!]_{\mathcal{P}} = \mathcal{P}^0$

(ii) $[\![\sigma \wedge \tau]\!]_{\mathcal{P}} = [\![\sigma]\!]_{\mathcal{P}} \cap [\![\tau]\!]_{\mathcal{P}}$

(iii) $[\![M\,.\,\sigma]\!]_{\mathcal{P}} = \{P \in \mathcal{P}^0 : \exists Q \in [\![\sigma]\!]_{\mathcal{P}}.\ P \xrightarrow{*} M\,.\,Q\}$

(iv) $[\![n[\ \sigma\ ]]\!]_{\mathcal{P}} = \{P \in \mathcal{P}^0 : \exists Q \in [\![\sigma]\!]_{\mathcal{P}}, R \in \mathcal{P}^0.\ P \xrightarrow{*} n[\ Q\ ] \mid R\}$

(v) $[\![(\sigma \mid \tau)]\!]_{\mathcal{P}} = \{P \in \mathcal{P}^0 : \exists Q \in [\![\sigma]\!]_{\mathcal{P}}, R \in [\![\tau]\!]_{\mathcal{P}}.\ P \xrightarrow{*} Q \mid R\}$

(vi) $[\![\langle l\rangle]\!]_{\mathcal{P}} = \{P \in \mathcal{P}^0 :\ P \xrightarrow{*} \langle l\rangle\}$

(vii) $[\![\langle l \Leftarrow\ \varphi\rangle]\!]_{\mathcal{P}} = \{P \in \mathcal{P}^0 : \exists\, \varsigma(x)b.\ P \xrightarrow{*} \langle l \Leftarrow \varsigma(x)b\rangle, b \in [\![\varphi]\!]_{\mathcal{F}}(x)\}$

(viii) $[\![\{l_i : \varphi_i\ ^{i\in I}\}]\!]_{\mathcal{P}}$
$= \{P \in \mathcal{P}^0 : \exists o.\ P \xrightarrow{*} o\ \&\ \forall\, i \in I\ \exists\ l_i = \varsigma(x_i)b_i \in o.\ b_i \in [\![\varphi_i]\!]_{\mathcal{F}}(x_i)\}$

(ix) $[\![\sigma \to \tau]\!]_{\mathcal{F}}(x) = \{b \in \mathcal{P}[x] : \forall P \in [\![\sigma]\!]_{\mathcal{P}}.\ b\{P/x\} \in [\![\tau]\!]_{\mathcal{P}}\}$

(x) $[\![\varphi \wedge \psi]\!]_{\mathcal{F}}(x) = [\![\varphi]\!]_{\mathcal{F}}(x) \cap [\![\psi]\!]_{\mathcal{F}}(x)$

Lemma 6.2

(i) *For all $\sigma, \tau \in \mathcal{T}_{\mathcal{P}}$, if $\sigma \leq \tau$ then $[\![\sigma]\!]_{\mathcal{P}} \subseteq [\![\tau]\!]_{\mathcal{P}}$.*

(ii) *For all $\varphi, \psi \in \mathcal{T}_{\mathcal{F}}$, if $\varphi \leq \psi$ then $[\![\varphi]\!]_{\mathcal{F}}(x) \subseteq [\![\psi]\!]_{\mathcal{F}}(x)$, for all $x \in \mathit{Var}$.*

Definition 6.3 Let $\Gamma = \{x_1 : \sigma_1, \ldots, x_k : \sigma_k\}$, $P \in \mathcal{P}$. $\sigma \in \mathcal{T}_{\mathcal{P}}$ and $\vartheta : \mathit{Var} \to \mathcal{P}^0$ is a substitution, and $P\vartheta$ is the result of substituting $\vartheta(x)$ for x in P for each free variable x of P, avoiding variable clashes by renaming bound variables. Then we define:

(i) $\vartheta \models \Gamma$ if $\vartheta(x_i) \in [\![\sigma_i]\!]_{\mathcal{P}}$, for all $1 \leq i \leq k$;

(ii) $\Gamma \models P : \sigma$ if for all ϑ such that $\vartheta \models \Gamma$, it is the case that $P\vartheta \in [\![\sigma]\!]_{\mathcal{P}}$.

Lemma 6.4 (Soundeness) *If $\Gamma \vdash P : \sigma$ then $\Gamma \models P : \sigma$.*

Proof. By induction on the derivation of $\Gamma \vdash P : \sigma$, using theorem 5.3 and lemma 6.2. □

Theorem 6.5 (Completeness)

$$\Gamma \vdash P : \sigma \Leftrightarrow \Gamma \models P : \sigma.$$

Proof. The only if part is lemma 6.4. To prove the if part we reason by induction on the size of σ. We report some interesting cases only (the others are either immediate consequence of the induction hypothesis, or they are similar to the proofs of analogous results in [dL01,CD01])

Case $\langle l \Leftarrow \ \varphi\rangle$: given any closed substitution ϑ such that $\vartheta \models \Gamma$, we know that $P\vartheta \xrightarrow{*} \langle l \Leftarrow \varsigma(x)b\rangle$, for some $b \in [\![\varphi]\!]_{\mathcal{F}}(x)$. Now $\varphi = \bigwedge_{j\in J}(\sigma_j \to \tau_j)$, and clearly

$$[\![\varphi]\!]_{\mathcal{F}}(x) = \bigcap_{j\in J} [\![\sigma_j \to \tau_j]\!]_{\mathcal{F}}(x),$$

being J finite, so that $b \in [\![\sigma_j \to \tau_j]\!]_{\mathcal{F}}(x)$ for all $j \in J$. Let $Q \in [\![\sigma_J]\!]_{\mathcal{P}}$ be arbitrarily chosen, and ϑ' be the same as ϑ but for $\vartheta'(x) = Q$. Then we have that $\vartheta' \models \Gamma, x : \sigma_j$ and that $b\vartheta' = b\{Q/x\} \in [\![\tau_J]\!]_{\mathcal{P}}$, namely that $\Gamma, x : \sigma_j \vdash b : \tau_j$ by induction, since the size of τ_j has to be smaller than the size of φ. By this we deduce $\Gamma \vdash \langle l \Leftarrow \varsigma(x)b\rangle : \langle l \Leftarrow \ \sigma_j \to \tau_j\rangle$ for all $j \in J$ by rule (MsgUpdate), and eventually $\Gamma \vdash \langle l \Leftarrow \varsigma(x)b\rangle : \langle l \Leftarrow \ \varphi\rangle$ by rules ($\wedge$-I) and ($\leq$). The thesis then follows by theorem 5.3.

Case $\{l_i : \varphi_i \ ^{i\in I}\}$: in this case we have that $P\vartheta \xrightarrow{*} o$ and that $b_i \in [\![\varphi_i]\!]_{\mathcal{F}}(x_i)$ for all $l_i = \varsigma(x_i)b_i \in o$. By reasoning in a similar way as in the previous case, this time using rule ($\{\ \}$-I), we conclude that $\Gamma \vdash o : \{l_i : \varphi_i\}$ for all $i \in I$, end hence $\Gamma \vdash o : \{l_i : \varphi_i \ ^{i\in I}\}$ by rules ($\wedge$-I) and ($\leq$). Again the thesis follows by 5.3.

□

The completeness theorem has relevant consequences. First we have a model, properly a filter model, of the calculus. Closed terms $P \in \mathcal{P}^0$ denote filters of types, namely upward sets closed under $\wedge$, by setting

$$[\![P]\!] = \{\sigma \in \mathcal{T}_{\mathcal{P}} : P \in [\![\sigma]\!]_{\mathcal{P}}\}.$$

By theorem 6.5 we have that

$$[\![P]\!] = \{\sigma \in \mathcal{T}_{\mathcal{P}} : \ \vdash P : \sigma\}$$

By introducing the notion of environments ξ, assigning filters to term variables, and setting $\xi \models \Gamma$ if $\sigma \in \xi(x)$ whenever $x : \sigma \in \Gamma$, we have for arbitrary terms $P \in \mathcal{P}$:

$$[\![P]\!]_{\xi} = \{\sigma \in \mathcal{T}_{\mathcal{P}} : \exists\Gamma.\ \xi \models \Gamma \ \&\ \Gamma \vdash P : \sigma\}.$$

Second we have a framework to logically characterize the capabilities that a process term P exhibits in one of its reducts by inspecting its typings. In fact it is not difficult to define a convergence predicate $\Downarrow$ by combining

the definitions in [dL01] and [CD01], and then use theorem 6.5 to show that $\vdash P : n[\ \omega\ ]$ if and only if $P \Downarrow n$ (namely it reduces to a parallel of processes having at top level an ambient named n), or that $\vdash P : \{l : \omega \to \omega\}$ if and only if $P \Downarrow l$ (that is $P \xrightarrow{*} o$ and o is an object that reacts to the message $\langle l \rangle$).

7 Conclusion

The type assignment systems for ς-calculus and the MA calculus smoothly combine into a system for a calculus of mobile objects. Essential properties, namely type invariance under expansion of subjects, completeness and adequacy of the assignments system with respect to an observational semantics based on capabilities of reducts, the construction of a logical model of properties, are retained by the resulting system. Full abstraction remains to be investigated (but then we need something like the self-open capability of [CD01], at least). More interesting would be a study of general methods to compose sequential and mobile/concurrent calculi in such a way that their assignment systems compose, namely their semantics.

Acknowledgements

The authors are very grateful to Pia Barbanera and Daniela Ferrarello for their valuable support.

References

[AC96] M. Abadi, L. Cardelli, *A Theory of Objects*, Springer 1996.

[AKPG01] T. Amtoft, A.J. Kfoury, S.M. Pericas-Geertsen, "What are Polymorphically-Typed Ambients?", *LNCS* 2028, 2001, pp. 206–224

[BCC01] M. Bugliesi, G. Castagna, S. Crafa, "Subtyping and Matching for Mobile Objects", *LNCS* 2202, 2001, pp 235–255.

[BCD83] H.P. Barendregt, M. Coppo, M. Dezani, "A Filter Lambda Model and the Completeness of Type Assignment", *JSL* 48, 1983, 931-940.

[vBdL03] S. van Bakel, U. de'Liguoro, "Logical Semantics for the First Order ς-Calculus", *LNCS* 2841, 2003, 202–215.

[CD01] M. Coppo and M. Dezani-Ciancaglini "A fully abstract model for mobile ambients" *ENTCS 62*, 2002.

[CG00] L. Cardelli, A.D. Gordon, "Mobile Ambients", *TCS* 240/1, 2000, pp 177–213.

[DGdL98] M. Dezani, E. Giovannetti, U. de' Liguoro, "Intersection types, λ-models and Böhm trees", in M. Takahashi, M. Okada, M. Dezani eds., *Theories of Types and Proofs*, Mathematical Society of Japan, vol. 2, 1998.

[Lan01] C. Laneve, "Inheritance in Concurrent Objects", in *Formal Methods for Distributed Processing, An Object Oriented Approach* H. Bowman, J. Derrick eds. Cambridge University Press, 2001.

[dL01] U. de' Liguoro "Characterizing convergent terms in object calculi via intersection types" *LNCS* 2044, 2001, 315–328.

[Kri90] J.L. Krivine, *Lambda-calcul, types et modèles*, Masson 1990.