Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 104 (2004) 217–234
www.elsevier.com/locate/entcs

A Filter Model for Safe Ambients

Ines Margaria [1,2]

Dipartimento di Informatica
Università di Torino
Torino, Italy

Maddalena Zacchi[3]

Dipartimento di Informatica
Università di Torino
Torino, Italy

Abstract

The definition of filter model is extended to a variant of Ambient Calculus: the Safe Ambient Calculus. The types are constructed by means of elementary and higher-order actions, that define the moves processes can do. Entailment rules for types allow to translate the parallel composition of moves into a non-deterministic choice of sequences of interleaved actions, providing a normal form for types assigned to processes. In the filter model obtained via the introduced type system, any process is interpreted as the set of all its types. The type assignment system results to be sound and complete with respect to the given semantics. Moreover the partial order relation induced by the filter model is compared with observational equivalence: the model is proved adequate, but it fails to be fully abstract.

Keywords: Ambient Calculus, Safe Ambient Calculus, Types, Filter Models

1 Introduction

One of the fundamental aspects of Wide Area Networks is that of barriers: the notions of locality, communication, mobility and security assume partic-

[1] Partially supported by IST-2001-33477 Project, Cofin '01 CoMeta Project, Cofin '01 Napoli Project and IST-2001-33477 DART Project. The funding bodies are not responsible for any use that might be made of the results presented here.
[2] Email: ines@di.unito.it
[3] Email: zacchi@di.unito.it

1571-0661
doi:10.1016/j.entcs.2004.08.027

ular importance due to the necessity of crossing barriers. The calculus of *Mobile Ambients* (MA) [5] is a process calculus for describing mobile computations, that is computations that cross barriers. Unit of the movement is the ambient $n[P]$ that represents a bounded space named n enclosing a multi-threaded process P. Ambients can be nested and can run concurrently. Inside ambients, processes can make computations and interact with other parallel processes of the same ambient, but not with processes running inside other ambients. To interact with processes of different ambients, a process can exercise the movement capabilities: *in* m, *out* m and *open* m to enter or exit other named ambients or dissolve ambient boundaries. The ambient object of the movement undergoes the action, because it has no means to control if and when the movement occurs. To provide processes with tools to protect themselves from unwanted movements, a variant of MA: the calculus of *Safe Ambients* (SA) [12] has been proposed. SA is obtained from MA by adding to the three mobility actions three corresponding coactions: $\overline{in}\ m$, $\overline{out}\ m$ and $\overline{open}\ m$. In SA to cross a barrier is always the result of a handshaking between two ambients. So ambient behaviors result from a subjective control exerted by the migrating ambient and an agreement given by the ambient where the coaction is consumed. The introduction of coactions is explicitly motivated by the aim of studying a dangerous form of interferences, situations where "the activity of a process is damaged or corrupted because of the activities of the other processes" [12]. An interesting topics in Ambient Calculi is the study of an appropriate notion of semantics equivalence and of the methods for establishing such equivalences [6], [14]; the principal equivalence relation proposed for the Ambient Calculi is a contextual equivalence based on the observability of ambients [5]. In [9] the equivalence between processes of a variant of MA is studied by means of a filter model, that results to be fully abstract with respect to the contextual equivalence $\cong_{obs}$. The model is designed via a type system, where types represent properties of processes. This paper is devoted to Safe Ambients and provides processes with types having a normal form, an intersection of *"sequential"* types. In this way a process is described as the set of all possible traces of its behaviours. Moreover the type system, inspired by the labelled transition system of [13], is used to define a filter model in which the SA processes are interpreted, as usual, as the set of their types. The inclusion relation between set of types induces an ordering $\subseteq_F$ on processes. Soundness and completeness of the type assignment system with respect to the given semantics is proved; moreover the model is proved to be adequate in the sense that : $P \subseteq_F Q$ implies $P \subseteq_{obs} Q$.

- *Names*: $n \in N$;
- *Capabilities*: $c \in C$

$$c \ ::= \ in\ m \mid out\ m \mid open\ m \mid \overline{in}\ m \mid \overline{out}\ m \mid \overline{open}\ m;$$

- *Processes*: $P \in \mathcal{P}$

$$\mathcal{P} \ ::= \ \mathbf{0} \mid c.\mathcal{P} \mid \ \mathcal{P}_1 \mid \mathcal{P}_2 \ \mid n[\mathcal{P}] \mid (\nu n)P \mid !\mathcal{P}.$$

Fig. 1. Process syntax

2 The Calculus

The syntax of the calculus is given in Fig. 1. For simplicity in this paper, we omit communication. N denotes the set of *ambient names*, ranged over by n, m, ...; C the set of *capabilities*, ranged over by c, d, ... and P the set of *processes*, ranged over by P, Q .... The operator of *restriction* (νn) is a binder for ambient names and leads to the usual notions of free occurrences of names for a process P ($fn(P)$) and of α-conversion. In the sequel the prefixing operator . takes precedence over the parallel composition |; hence $c.P \mid Q$ is read as $(c.P) \mid Q$. The nil process $\mathbf{0}$ is often omitted, so c can be a shorthand for $c.\mathbf{0}$.

Structural congruence and reduction rules defined in Fig. 2, give the operational semantics of the calculus. For the out-reduction rule we follow the variant proposed in [13], requiring that the co-capability $\overline{out}\ m$ for the emigration outside of the ambient m is exercised by the target computation rather then by the ambient m. This choice is due to the aim of giving to the receiving ambient the control of the movement.

In process calculus, a standard way to define behavior equivalence for processes is the may-testing equivalence: two processes are contextually equivalent if they satisfy the same observation predicates, when they are placed in the same contexts. For MA the observation predicate proposed [5] is the occurrence, at top level of a process, of an ambient whose name is not restricted. Actually the presence of an ambient n at top level of a process, represents the possibility for P of interacting with the environment via n. In [12], for SA, where to cross a boundary requires an authorization, the "exhibition of an ambient" requires that the ambient brought at top level can exercise the $\overline{in}$ or the $\overline{open}$ capability. We use in that follows a simplification of the definition in [12], as introduced in [13], requiring only the presence of the co-capability $\overline{open}$.

$\equiv$ is the least equivalence relation that:
(i) includes α-conversion
(ii) is preserved by all operators
(iii) satisfies the following rules:

- $P \mid Q \equiv Q \mid P$ (Struct Par Comm)
- $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$ (Struct Par Ass)
- $P \mid \mathbf{0} \equiv P$ (Struct Zero Par)
- $(\nu n)\mathbf{0} \equiv \mathbf{0}$ (Struct Zero Res)
- $!P \equiv !P \mid P$ (Struct Repl Par)
- $!\mathbf{0} \equiv \mathbf{0}$ (Struct Zero Repl)
- $(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$ (Struct Res Res)
- $n \notin fn(P)$ *implies* $(\nu n)(P \mid Q) \equiv P \mid (\nu n)Q$ (Struct Res Par)
- $n \neq m$ *implies* $(\nu n)(m[P]) \equiv m[(\nu n)P]$ (Struct Res Amb)

$\rightarrow$ is the least preorder relation that:
(i) is preserved by all operators, except prefixing
(ii) satisfies the rules below:

- $m[in\ n.P \mid Q] \mid n[\overline{in}\ n.R \mid S] \rightarrow n[m[P \mid Q] \mid R \mid S]$ (Red-In)
- $m[n[out\ m.P \mid Q] \mid R] \mid \overline{out}\ m.S \rightarrow n[P \mid Q] \mid m[R] \mid S$ (Red-Out)
- $open\ n.P \mid n[\overline{open}\ n.Q \mid R] \rightarrow P \mid Q \mid R$ (Red-Open)
- $P \equiv Q,\ Q \rightarrow R,\ R \equiv S \Rightarrow P \rightarrow S$ (Red-Struct)

Fig. 2. Structural Congruence and Reduction Relation

Definition 2.1 (i) A process P *exhibits an ambient* n: $P \Downarrow n$ if $P \rightarrow^* \rightarrow^* (\nu\overrightarrow{m})(n[\overline{open}\ n.Q \mid R] \mid S)$ for some processes Q, R, S $(n \notin \{\overrightarrow{m}\})$.
(ii) $P \subseteq_{obs} Q$ if for all context $\mathcal{C}[\,]$ and ambients n: $\mathcal{C}[P] \Downarrow n \Rightarrow \mathcal{C}[Q] \Downarrow n$.
(iii) $P \cong_{obs} Q$ if $P \subseteq_{obs} Q$ and $Q \subseteq_{obs} P$.

Remark 2.2 Note that $P \rightarrow^* Q$ implies $Q \subseteq_{obs} P$, but in general it does not imply that $P \cong Q$; for example, let $P = open\ n.\mathbf{0} \mid n[\overline{open}\ n.\mathbf{0} \mid m[\mathbf{0}]]$ and $Q = m[\mathbf{0}]$, $P \rightarrow^* Q$, but for the context [-], $P \Downarrow n$ whereas Q does not converge to any ambient n.

3 Types

Type systems have been proposed for Mobile Calculi essentially with the goal to provide a tool of control: to control the mobility of ambients [4], the values exchanged [7], the absence of grave interferences [12]. Different is the aim of the type system proposed in [9] for MA: to characterize the process behavior in order to provide a tool for giving its semantics. This is also our goal for SA; the process semantics we look for, is a kind of *trace semantics*, in which a process is characterized by means of all sequences of actions it may exercise. Therefore in the definition of types we want to capture the idea of *action*.

In [13] the labels of the transition system are actions, defined as an extension of the original definition of capability. As a matter of fact each capability gives rise to an action, but, when inserted in ambients, it can induce further higher order actions. In fact, as the process $in\ n.P$, when inserted in an appropriate ambient, can exercise the action $in\ n$ and then its behavior is that one of the process P, in the same way the process $m[in\ n.P]$, placed in a suitable ambient, has the capability *to move the ambient m into the ambient* n, and then it continues with some behavior. To describe this continuation Merro and Hennessy use the *concretion* $(\nu\overrightarrow{m})(< P >_n Q)$ (see also [6], [12]); a concretion $(\nu\overrightarrow{m})(< P >_n Q)$ models the behavior of a process that, after exercising an action, leaves inside the ambient n the process P, and outside the ambient n the process Q; $\overrightarrow{m}$ represents the set of private names shared by P and Q.

We follow this suggestion, so our set of types T contains besides the six actions induced by the capabilities and co-capabilities (elementary actions), five higher order actions, precisely the action $enter_m n$ induced by $m[in\ n]$, the action $exit_m n$ induced by the $m[out\ n]$, the action $\overline{enter}\ n$ induced by $n[\overline{in}\ n]$, the action $pop_m n$ induced by $n[m[out\ n]]$ and the action $free\ n$ induced by $n[\overline{open}\ n]$.

The pairs $(enter_m n,\ \overline{enter}\ n)$, $(pop_m n,\ \overline{out}\ n)$, $(free\ n,\ open\ n)$ are said *matching* pairs. Notice that the definition of $\overline{out}$ semantics here considered allows to have the actions of the matching pair $(pop_m n,\ \overline{out}\ n)$ at the same level of ambient nesting.

The formal definition of the set of types T is given in Fig. 3. *Prefixes1* define the actions requiring as continuation a standard type, whereas *Prefixes2* define the actions that must be followed by a concretion $(\nu\overrightarrow{m})(< \sigma >_n \tau)$, where σ is the type of the process that is into the ambient n, τ is the type of the process that is outside n and $\overrightarrow{m}$ are the private names shared by σ and τ. In the sequel $(\nu\overrightarrow{m})(< \sigma >_n \tau) \restriction \varrho$ indicates the concretion $(\nu\overrightarrow{m})(< \sigma >_n (\tau \mid \varrho))$, whereas $(\nu\overrightarrow{m})(< \sigma >_n \tau) \upharpoonright \varrho$ indicates the concretion $(\nu\overrightarrow{m})(< (\sigma \mid$

- *Prefixes1*: $\mu \ ::= \ in\ n | out\ n | open\ n | \overline{in}\ n | \overline{out}\ n | \overline{open}\ n | pop_m n | free\ n$;
- *Prefixes2*: $\alpha \ ::= \ enter_m n | exit_m n | \overline{enter}\ n$;
- *Actions*: $\gamma \ ::= \ \mu | \alpha$;
- *Types*: $\sigma \ ::= \ \omega \mid \mu.\sigma \mid \alpha.(\nu \overrightarrow{m})(< \sigma_1 >_n \ \sigma_2) \mid n[\sigma] \mid (\nu n)\sigma \mid \mid \sigma_1 \mid \sigma_2 \mid \sigma_1 \wedge \sigma_2$

Fig. 3. Type Definition

$\varrho) >_n \tau$). Besides actions, as type constructors we consider the ambient, the restriction, the parallel composition, and the intersection $\wedge$. Type ω represents a property true for all processes, whereas the intersection $\wedge$ models "may" nondeterminism: a process having type $\sigma \wedge \tau$ can possibly exhibit, in different reduction paths, both property σ and τ.

An action γ is said *compatible with the ambient* n if $\gamma \in \{in\ m,\ \ out\ m,\ \ \overline{in}\ n,\ \overline{open}\ n,\ \ exit_m n\}$, *not compatible* otherwise. If γ is not compatible with n, the type $n[\gamma.\sigma]$ is said *deadlocked*.

The notion of *free name* of a type is usual; remember that the occurrence of a subscript name of a prefix must not be considered a free occurrence.

On the set of types T is defined a partial order relation $\leq$; $\sigma \leq \tau$ means that the property σ entails property τ; $\sigma \simeq \tau$ iff $\sigma \leq \tau$ and $\tau \leq \sigma$. Type Entailment Rules are shown in Fig. 4 and Fig. 5. The *Action rules* define the higher order actions, whereas *Reduction rules* formalize the fact that the execution of an action corresponds to a loss of capabilities. As usual, the right hand side of a reduction rule is called the *contractum*. Of particular relevance for our goal are the *Sequentialization rules*, that can be interpreted as a first step toward the translation of parallel composition of actions into nondeterministic choice between sequences of interleaved actions. The first four sequentialization rules say that the type $\gamma.(\sigma \mid \tau)$ has fewer capabilities than the type $\gamma.\sigma \mid \tau$ because it can offer to its environment, as first move, only the action γ, whereas $\gamma.\sigma \mid \tau$ besides the action γ, can possibly offer other moves risen by the type τ. The last two rules say that the parallel composition of two prefix types $\gamma_1.\sigma \mid \gamma_2.\tau$ is equivalent to the nondeterministic choice between different paths; if the actions γ_1 and γ_2 do not match the paths are two: one starting with the move γ_1, the other one starting with the move γ_2. If the two actions γ_1 and γ_2 match, there is a third choice: to execute the reduction. To sake of simplicity in Fig. 5 the notation of a prefix type has been stretched so that $\gamma.\sigma$ denotes both *Prefixes1* types (hence σ represents a type) and *Prefixes2* types (hence σ represents a concretion). So in the expressions

of the shape $\gamma_1.(\sigma \mid \gamma_2.\tau)$, $(\sigma \mid \gamma_2.\tau)$ indicates the usual parallel composition between types if γ_1 is a elementary action, it denotes $(\sigma \restriction (\gamma_2.\tau))$ if γ_1 is the action $\overline{enter}\ n$, $(\sigma \upharpoonright (\gamma_2.\tau))$ otherwise. Types are considered modulo $\simeq$. $\simeq$ is preserved by both intersection and parallel composition with ω. Parallel composition of types is considered modulo permutations, and intersection of types is considered modulo permutations and repetitions. $\bigwedge_{i\in[1\cdots n]}\sigma_i$ denotes the intersection $\sigma_1 \wedge \sigma_2 \wedge \sigma_n$;. $\tau \propto \bigwedge_{i\in[1\cdots n]}\sigma_i$ denotes that $\tau \equiv \sigma_i$ for some σ_i.

A crucial notion is that of sequential type. A sequential type models the behavior of a process performing a sequence of actions.

Definition 3.1 (i) The set $S \subset T$ of *sequential types* is defined inductively in the following way: $\varphi \ ::= \ \omega \ \mid \ \mu.\varphi \ \mid \ \alpha.(\nu\overrightarrow{m})(<\varphi_1>_n \ \varphi_2)$

(ii) The *weight* of a *sequential type* is defined as follows:

$$\begin{aligned}
|\omega| &= 0\\
|\mu.\varphi| &= 1+|\varphi| \ \ if \ \ \mu \neq free\ n \ \ or \ \ \mu \neq pop_m n\\
&= 2\ +|\varphi| \ \ otherwise\\
|\alpha.(\nu\overrightarrow{m})(<\varphi_1>_n \ \varphi_2)| &= 2\ +|\varphi_1|+|\varphi_2|
\end{aligned}$$

To prove that every type can be expressed by a nondeterministic choice of sequential types, we use the functions *res*, *unfold* and *ser*. Their formal definition is rather complex and can be found in Appendix. Here we give only an informal description.
The function *res*, defined by structural induction on sequential type definition, takes as arguments a sequence of names $\overrightarrow{n}$ and a sequential type ξ and returns a sequential type such that:

$$res\ (\overrightarrow{n},\xi) \ \simeq \ (\nu\overrightarrow{n})\xi.$$

The functions *ser* and *unfold* are defined by simultaneous induction on the weight. *ser* takes as arguments two sequential types ξ, χ and returns a set of sequential types such that :

$$\bigwedge_{\zeta\in\ ser(\xi,\chi)}\zeta \ \simeq \ \xi \mid \chi$$

The function *unfold* takes as arguments a name n and a sequential type ξ, and returns a set of sequential types such that :

$$\bigwedge_{\zeta\in\ unfold(n,\xi)}\zeta \ \simeq \ n[\xi]$$

We can now prove that every type has a unique normal form modulo permutations and parallel composition with ω.

- Axioms for ω

$$- \ \sigma \leq \omega \qquad\qquad - \ \sigma \simeq \sigma \mid \omega$$
$$- \ (\nu n)\omega \simeq \omega \qquad\qquad - \ n[\omega] \simeq \omega$$

- Commutativity and Associativity of parallel composition $\mid$

$$- \ \sigma \mid \tau \simeq \tau \mid \sigma \qquad\qquad - \ (\sigma \mid \tau) \mid \varrho \simeq \sigma \mid (\tau \mid \varrho)$$

- Intersection $\wedge$

$$- \ \sigma \wedge \tau \leq \sigma \quad \sigma \wedge \tau \leq \tau \qquad\qquad - \ \sigma \leq \sigma \wedge \sigma$$
$$- \ \sigma \leq \sigma' \ and \ \tau \leq \tau' \Rightarrow \sigma \wedge \tau \leq \sigma' \wedge \tau' \qquad - \ \rho \mid (\sigma \wedge \tau) \ \simeq \ (\rho \mid \sigma) \wedge (\rho \mid \tau)$$
$$- \ n[\sigma \wedge \tau] \ \simeq \ n[\sigma] \wedge n[\tau] \qquad\qquad - \ \mu.(\sigma \wedge \tau) \ \simeq \ \mu.\sigma \wedge \mu.\tau$$
$$- \ \alpha.(\nu\overrightarrow{m})(< \sigma \wedge \tau >_n \rho) \ \simeq \ \alpha.(\nu\overrightarrow{m})(< \sigma >_n \rho) \ \wedge \ \alpha.(\nu\overrightarrow{m})(< \tau >_n \rho)$$
$$- \ \alpha.(\nu\overrightarrow{m})(< \sigma >_n \rho \wedge \tau) \ \simeq \ \alpha.(\nu\overrightarrow{m})(< \sigma >_n \rho) \ \wedge \ \alpha.(\nu\overrightarrow{m})(< \sigma >_n \tau)$$

- Action

$$- \ m[in \ n.\sigma] \simeq enter_m n.(< m[\sigma] >_n \omega)$$
$$- \ m[out \ n.\sigma] \simeq exit_m n.(< \omega >_n m[\sigma])$$
$$- \ n[\overline{in} \ n.\sigma] \simeq \overline{enter} \ n.(< \sigma >_n \omega)$$
$$- \ n[\overline{open} \ n.\sigma] \ \simeq \ free \ n.\sigma$$
$$- \ n[exit_m n.(< \sigma >_n \tau) \ \simeq \ pop_m n.(n[\sigma] \mid \tau)$$
$$- \ n[\gamma.\sigma] \ \simeq \ \omega \qquad \text{if} \ \ \gamma \ \ \text{not compatible with n}$$

- Reduction

$$- \ enter_m n.(\nu\overrightarrow{p})(< \sigma_1 >_n \sigma_2) \mid \overline{enter} \ n.(\nu\overrightarrow{q})(< \tau_1 >_n \tau_2) \ \leq$$
$$\leq \ (\nu\overrightarrow{p})(\nu\overrightarrow{q})(n[\sigma_1 \mid \tau_1] \mid \sigma_2 \mid \tau_2)$$
$$- \ pop_m n.\sigma \mid \overline{out} \ n.\tau \ \leq \ \sigma \mid \tau$$
$$- \ open \ n.\sigma \mid free \ n.\tau \ \leq \ \sigma \mid \tau$$

Fig. 4. Type Entailment Rules (part I)

- Restriction
  - $(\nu m)n[\sigma] \simeq n[(\nu m)\sigma]$ $\quad m \neq n$
  - $(\nu n)(\nu m)\sigma \simeq (\nu m)(\nu n)\sigma$
  - $(\nu m)(\sigma \mid \tau) \simeq \sigma \mid (\nu m)\tau$ $\quad m \notin fn(\sigma)$
  - $(\nu m)(\sigma \wedge \tau) \simeq (\nu m)\sigma \wedge (\nu m)\tau$
  - $(\nu m)\gamma.\sigma \simeq \omega$ $\quad$ if $m \in fn(\gamma)$
  - $(\nu m)\gamma.\sigma \simeq \gamma.(\nu m)\sigma$ $\quad$ if $m \notin fn(\gamma)$
- Sequentialization
  - $\mu.\sigma \mid \tau \leq \mu.(\sigma \mid \tau)$
  - $enter_m n.(\nu \overrightarrow{h})(< \sigma >_n \tau) \mid \rho \leq enter_m n.(\nu \overrightarrow{h})(< \sigma >_n (\tau \mid \rho))^*$
  - $exit_m n.(\nu \overrightarrow{h})(< \sigma >_n \tau) \mid \rho \leq exit_m n.(\nu \overrightarrow{h})(< \sigma \mid \rho >_n \tau)^*$
  - $\overline{enter}\ n.(\nu \overrightarrow{h})(< \sigma >_n \tau) \mid \rho \leq \overline{enter}\ n.(\nu \overrightarrow{h})(< \sigma >_n (\tau \mid \rho))^*$
  - $\gamma_1.\sigma \mid \gamma_2.\tau \simeq \gamma_1.(\sigma \mid \gamma_2.\tau) \wedge \gamma_2.(\gamma_1.\sigma \mid \tau)$ if γ_1 and γ_2 do not match
    $\simeq \varrho \wedge \gamma_1.(\sigma \mid \gamma_2.\tau) \wedge \gamma_2.(\gamma_1.\sigma \mid \tau)$
    if γ_1 and γ_2 match and ρ is the contractum
- Congruence
  - $\sigma \leq \tau \Rightarrow n[\sigma] \leq n[\tau]$
  - $\sigma \leq \tau \Rightarrow \mu.\sigma \leq \mu.\tau$
  - $\sigma \leq \tau \Rightarrow (\nu m)\sigma \leq (\nu m)\tau$
  - $\sigma \leq \tau \Rightarrow \sigma \mid \rho \leq \tau \mid \rho$
- Transitivity
  - $\sigma \leq \tau \ \& \ \tau \leq \rho \Rightarrow \sigma \leq \rho$

(⋆) If $fn(\rho) \cap \{\overrightarrow{h}\} = \Phi$

Fig. 5. Type Entailment Rules (part II)

Lemma 3.2 *For all $\sigma \in T$ there is a unique type $\bigwedge_{i\in[1\cdots n]} \xi_i$, where ξ_i are sequential types, such that $\sigma \simeq \bigwedge_{i\in[1\cdots n]} \xi_i$. We call it the* normal form *of σ, denoted by $nf(\sigma)$.*

Proof. The proof is by structural induction on types, using the functions *res*, *ser* and *unfold*.

(i) $nf(\omega) = \omega$.

(ii) $nf(\mu.\sigma) = \bigwedge_{\zeta \propto nf(\sigma)} \mu.\zeta$.

(iii) $nf(\alpha.(< \sigma >_n \tau)) = \bigwedge_{\zeta \propto nf(\sigma),\ \chi \propto nf(\tau)} (\alpha.(< \zeta >_n \chi))$.

(iv) $nf(n[\sigma]) = \bigwedge_{\zeta \in unfold(n,\xi),\ \xi \propto nf(\sigma)} \zeta$.

(v) $nf(\sigma \mid \tau) = \bigwedge_{\zeta \in ser(\chi,\xi), \chi \propto nf(\sigma),\ \xi \propto nf(\tau)} \zeta$.

(vi) $nf(\sigma \wedge \tau) = nf(\sigma) \wedge nf(\tau)$

(vii) $nf((\nu \overrightarrow{n})\sigma) = \bigwedge_{\zeta = res(\overrightarrow{n},\xi),\ \xi \propto nf(\sigma)} \zeta$.

□

Example 3.3 $\sigma \equiv m[out\ n.\overline{in}\ m.\omega \mid \overline{open}\ m.\omega]$

$$nf(\sigma) = exit_m n.\ (< \omega >_n\ (\overline{enter}\ m.(< \overline{open}\ m.\omega >_m \omega))) \wedge \\ \wedge\ exit_m n.\ (< \omega >_n (free\ m.\overline{in}\ m.\omega)) \wedge free\ m.out\ n.\overline{in}\ m.\omega$$

$$nf(n[\sigma]) = pop_m n.(\overline{enter}\ m.(< \overline{open}\ m.\omega >_m \omega)) \wedge pop_m n.free\ m.\overline{in}\ m.\omega$$

The following Lemma, proved by induction on the definition of $\leq$ relates entailment relations between types and normal forms.

Lemma 3.4 (i) *$\bigwedge_{i \in I} \xi_i \leq \bigwedge_{j \in J} \chi_j$ implies that for every $j \in J$ there is a $i \in I$ such that $\xi_i \leq \chi_j$.*

(ii) *Let $\sigma \leq \tau$. Then for every $\chi \propto nf(\tau)$, there is a $\xi \propto nf(\sigma)$ such that $\xi \leq \chi$.*

4 Type Inference

Types are associated with processes by means of a type assignment system $\vdash$, defined by the rules of Fig. 6.

We can prove by simple induction on deduction the following Lemma:

Lemma 4.1 (Generation Lemma) (i) $\vdash \mathbf{0} : \sigma$ *iff* $\sigma \simeq \omega$;

(ii) $\vdash c.P : \sigma$ *iff* $\vdash P : \tau$ *and* $c.\tau \leq \sigma$ *for some* τ;

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

(iv) $\vdash (\nu n)P : \sigma$ *iff* $\vdash P : \tau$ *and* $(\nu n)\tau \leq \sigma$ *for some* τ;

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

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

$$(\omega) \quad \vdash P : \omega \qquad\qquad (prefix) \quad \frac{\vdash P : \sigma \quad c \in C}{\vdash c.P : c.\sigma}$$

$$(amb) \quad \frac{\vdash P : \sigma \quad n \in N}{\vdash n[P] : n[\sigma]} \qquad\qquad (|) \quad \frac{\vdash P_1 : \sigma \quad \vdash P_2 : \tau}{\vdash P_1 \mid P_2 : \sigma \mid \tau}$$

$$(res) \quad \frac{\vdash P : \sigma}{\vdash (\nu n)P : (\nu n)\sigma} \qquad\qquad (!) \quad \frac{\vdash P : \sigma \quad \vdash\ !P : \tau}{\vdash\ !P : \sigma \mid \tau}$$

$$(\wedge) \quad \frac{\vdash P : \sigma \quad \vdash P : \tau}{\vdash P : \sigma \wedge \tau} \qquad\qquad (\leq) \quad \frac{\vdash P : \sigma \quad \sigma \leq \tau}{\vdash P : \tau}$$

Fig. 6. Type Inference Rules

Lemma 4.1 with the definitions of $\equiv$ and of $\rightarrow$ allows us to state Subject Congruence Property (congruent processes have the same types) and Subject Expansion Property (types are preserved under subject expansion). The proofs are by induction on deduction.

Lemma 4.2 (Subject Congruence) $\vdash P : \sigma$ *and* $P \equiv Q \Rightarrow \vdash Q : \sigma.$

Lemma 4.3 (Subject Expansion) $\vdash Q : \sigma$ *and* $P \rightarrow^* Q \Rightarrow \vdash P : \sigma.$

5 The Filter Model

The construction of the filter model via type system is an approach widely used for λ-calculus and its extensions [1], [2], [3]. A filter model has been also used for higher order concurrent processes [10], [11], and in [9] for Mobile Ambients. Let us recall the filter definition. Let $< D, \leq >$ be a preorder. A non-empty subset L of D is a *filter* if it is an upper set, i.e. $d \in L$ and $d \leq d'$ imply $d' \in L$, and every finite subset of L has a greatest lower bound in L. Domain of our model is $< F(T), \subseteq >$ where $F(T)$ is the set of filters over $< T, \leq >$ and $\subseteq$ is the set inclusion relation. Note that the intersection operator plays an important role on $< T, \leq >$, because the greatest lower bound of a finite set of types is the intersection of the types in the set. It is standard to prove that $< F(T), \subseteq >$ is a complete algebraic lattice, so every continuous function f has the least fixed point $fix(f)$.

Lemma 5.1 $< F(T), \subseteq >$ *is a complete algebraic lattice.*

If $A \subseteq T$ then $\uparrow A$ denotes the filter generated by A, obtained by closing

A under finite intersection and by (upper closing A under) $\leq$.

Let par: $F(T) \times F(T) \to F(T)$ be the function defined by $par(F,G) = \uparrow \{\sigma \mid \tau \mid \sigma \in F \text{ and } \tau \in G\}$.
It is standard to prove that the function par is continuous.

The interpretation of a type in the model is done by means of the function $\| - \|$, defined by structural induction on types.

Definition 5.2 The function $\| - \|$: $\mathcal{P} \to F(T)$ is defined as follows:

- $\| \mathbf{0} \| = \uparrow \{\omega\}$
- $\| c.P \| = \uparrow \{c.\sigma \mid \sigma \in \| P \|\}$
- $\| n[P] \| = \uparrow \{n[\sigma] \mid \sigma \in \| P \|\}$
- $\| P \mid Q \| = par(\| P \|, \| Q \|)$
- $\| (\nu n)P \| = \uparrow \{(\nu n)\sigma \mid \sigma \in \| P \|\}$
- $\|!P \| = fix(\lambda X \in F(T).\ par(\| P \|, X))$

By Lemma 4.2 we can prove that the interpretation of a process is the filter of all types that can be derived for it.

Theorem 5.3 $\| P \| = \{\sigma \mid \vdash P : \sigma\}$

The inclusion on filters gives rise to an order relation $\subseteq_F$ on processes, in the sense that $P \subseteq_F Q$ if and only if $\| P \| \subseteq \| Q \|$. Obviously, by Theorem 5.3, follows that $P \subseteq_F Q$ if and only if $\vdash P : \sigma$ implies $\vdash Q : \sigma$, for all σ.

A crucial question is the relationship between the order relation $\subseteq_F$ and the observational relation $\subseteq_{obs}$. We prove that $P \subseteq_F Q$ implies $P \subseteq_{obs} Q$, whereas a counter-example shows that the converse is not true; so the model is *adequate*, but it fails to be *fully abstract*.

The proof of adequacy is done via a type interpretation, defined in a quite standard way. This interpretation allows to prove easily that the type assignment system is sound and complete with respect to the obtained semantics.

We associate with every type a set of filters of $F(T)$ (*type interpretation*) and we show that the interpretation of a process P belongs to the interpretation of a type σ if and only if σ can be derived for P. For the definition of type interpretation, we need a stronger notion of reduction over processes $\rightsquigarrow$, that does not modify the notion of ambient convergency.

Definition 5.4 The reduction relation $\rightsquigarrow$ over $\mathcal{P}$ is defined by adding to the reduction rules of Fig. 2 the following rule:

$$c.P \mid Q \rightsquigarrow c.(P \mid Q) \qquad \text{(Red-Seq)}$$

Lemma 5.5 (i) $P \rightsquigarrow^\star Q$ *and* $\vdash Q : \sigma \Rightarrow \vdash P : \sigma$

(ii) $P \Downarrow n$ *iff* $P \rightsquigarrow^\star (\nu\overrightarrow{m})(n[\overline{open}\ n.Q \mid R] \mid S) \quad (n \notin \{\overrightarrow{m}\})$ *for some processes* Q, R, S.

The interpretation of the types is done in two steps: first we define the interpretation of sequential types, then we use the definition of normal form to construct the interpretation of generic types.

Definition 5.6 The interpretation of sequential types is defined by structural induction as follows:

- $\| \omega \| \ = \ \mathcal{P}$
- $\| c.\xi \| \ = \ \{P \mid P \rightsquigarrow^\star c.Q \ \text{ and } \ Q \in \| \xi \|\}$
- $\| \mathit{free}\ n.\xi \| \ = \ \{P \mid P \rightsquigarrow^\star n[\overline{open}\ n.Q \mid R] \ \text{ and } \ Q \mid R \in \| \xi \|\}$
- $\| pop_m n.\xi \| \ = \ \{P \mid P \rightsquigarrow^\star n[m[out\ n.Q]] \mid R \ \text{ and } \ m[Q] \mid R \in \| \xi \|\}$
- $\| enter_m n.(\nu\overrightarrow{h})(<\xi_1>_n \xi_2) \| \ = \ \{P \mid P \rightsquigarrow^\star (\nu\overrightarrow{h})(m[in\ n.Q] \mid R)$ and and $m[Q] \in \| \xi_1 \| \ \text{ and } \ R \in \| \xi_2 \|\}$
- $\| exit_m n.(\nu\overrightarrow{h})(<\xi_1>_n \xi_2) \| \ = \ \{P | P \rightsquigarrow^\star (\nu\overrightarrow{h})(m[out\ n.Q] | R)$ and and $m[Q] \in \| \xi_2 \| \ \text{ and } \ R \in \| \xi_1 \|\}$
- $\| \overline{enter}\ n.(\nu\overrightarrow{h})(<\xi_1>_n \xi_2) \| \ = \ \{P \mid P \rightsquigarrow^\star (\nu\overrightarrow{h})(n[\overline{in}\ n.Q] \mid R)$ and and $Q \in \| \xi_1 \| \ \text{ and } \ R \in \| \xi_2 \|\}$

Definition 5.7 The interpretation of generic types is defined by:

$$\| \sigma \| \ = \ \bigcap_{\xi \propto nf(\sigma)} \| \xi \|$$

In order to prove the soundness of type assignment we need some lemmas, in particular we must prove that the type interpretation agrees with the entailment relation.

Lemma 5.8 (i) $P \in \| \xi \|$ *implies* $(\nu n)P \in \| \zeta \|$ *for all* $\zeta \propto nf((\nu n)\xi)$.

(ii) *Let* ξ *and* χ *be two sequential types. Then* $P \in \| \xi \|$ *and* $Q \in \| \chi \|$ *imply* $P \mid Q \in \| \zeta \|$ *for all* $\zeta \propto nf(\xi \mid \chi)$ *and* $m[P] \in \| \vartheta \|$ *for all* $\vartheta \propto nf(m[\xi])$.

(iii) $\sigma \leq \tau$ *implies* $\| \sigma \| \ \subseteq \ \| \tau \|$

Proof.

(i) By structural induction on sequential types.

(ii) By simultaneous induction on the weight.

(iii) $\sigma \leq \tau \ \Rightarrow \ nf(\sigma) \leq nf(\tau)$
$\Rightarrow$ for every $\chi \propto nf(\tau)$ there is a $\xi \propto nf(\sigma)$ such that $\xi \leq \chi$
$\Rightarrow \ P \in \| \sigma \|$ implies $P \in \| \tau \|$.

□

Theorem 5.9 (Soundness and completeness of $\vdash$) $\vdash P : \sigma$ *iff* $P \in\!\| \sigma \|$.

Proof. Soundness is proved by induction on deduction using Lemma 5.8(iv) for rule ($\leq$). As for completeness, it is sufficient to prove for sequential types: $\xi\ P \in\!\| \xi \|\ \Rightarrow\ \vdash P : \xi$. This can be proved by induction on weight, using Definition 5.7, Lemma 5.5 and Lemma 4.1. □

We can now prove that there is a type characterizing the convergency to an ambient.

Lemma 5.10 $\vdash P : \mathit{free}\ n.\omega$ *iff* $P \Downarrow n$

Proof.

$$
\begin{array}{lll}
(\Longrightarrow) & \vdash P : \mathit{free}\ n.\omega \Rightarrow P \in\!\| \mathit{free}\ n.\omega \| & \\
 & \Rightarrow P \rightsquigarrow^{\star} (n[\overline{\mathit{open}}\ n.Q] \mid R) \text{ and } Q \mid R \in\!\| \omega \| & \\
 & \qquad\qquad \text{by Definition 5.6} & \\
 & \Rightarrow P \Downarrow n \ \text{ by Lemma 5.5(ii)} & \\
(\Longleftarrow) & P \Downarrow n \qquad \Rightarrow P \rightsquigarrow^{\star} (\nu \overrightarrow{m})(n[\overline{\mathit{open}}\ n.Q \mid R] \mid S) \text{ by Lemma 5.5(ii)} & \\
 & \Rightarrow \vdash P : n[\overline{\mathit{open}}\ n.\omega] \mid \omega \quad \text{by subject expansion} & \\
 & \Rightarrow \vdash P : \mathit{free}\ n.\omega \ \text{ by } (\leq) \text{ rule.} &
\end{array}
$$

□

Theorem 5.11 (Adequacy) *If* $P \subseteq_F Q$ *then* $P \subseteq_{obs} Q$.

Proof.

$$
\begin{array}{lll}
C[P] \Downarrow n \Rightarrow & C[P] : \mathit{free}\ n.\omega & \text{by Lemma 5.10} \\
\Rightarrow & C[Q] : \mathit{free}\ n.\omega & \text{since } Q \text{ has all the types of } P \\
\Rightarrow & C[Q] \Downarrow n & \text{by Lemma 5.10.}
\end{array}
$$

□

To show that the model is not fully abstract, let consider the processes P and Q:
$P = n[\overline{in}\ n.\mathbf{0}]$
$Q = n[m[out\ n.\overline{open}\ m.\mathbf{0} \mid n[\overline{in}\ n.\mathbf{0}]]] \mid open\ m.\overline{out}\ n.\mathbf{0}$

They are not comparable in the order relation $\subseteq_F$ because for P is derivable the type $\overline{enter}\ n.(< \omega >_n \omega)$, whereas the same type is not derivable for Q and vice versa the type $open\ m.\overline{out}\ n.(< \omega >_n \omega)$ is derivable for Q,

but not for P. In the order relation $\subseteq_{obs}$ however $P \subseteq_{obs} Q$, in fact for a non-trivial context $C[-]$, if $C[P] \Downarrow h$ for some ambient h, $C[-]$ must allow the emigration from n by exhibiting the co-capability $\overline{out}\ n$, hence it must have the form $C[- \mid \overline{out}\ n]$, but $C[Q \mid \overline{out}\ n] \rightarrow^{\star} C[P \mid \overline{out}\ n]$ and so there is no way to find a distinguishing context between P and Q. This fact is not surprising: Merro and Hennessy [13] already noticed the difficulties in conceiving a distinguishing context for action $\overline{enter}\ n$.

6 Conclusion

We have constructed a filter model via a type system for Safe Ambients, following the line of the filter model defined in [9] for Mobile Ambients. Basic elements of our types are the actions that can be considered as the atomic moves of an ambient. We proved that every type has a normal form that is an intersection of sequential types; this fact allows to express a parallel composition of actions as a nondeterministic choice of the sequences of interleaved actions.

The model turns out to be adequate, but not fully abstract. In [9] it is a new capability, the *self-open* n, that makes the filter model fully abstract. We conjecture that also in SA the addition of the *self-open* n capability should permit to obtain a fully abstract filter model, but it should distort the spirit of the calculus. On the other hand the *self-open* n with its corresponding co-action in the ambient outside does not modify the calculus. In the future we wish to study a way to obtain a fully abstract model or by means of a stronger notion of type inclusion or by adding new features to capabilities (cfr. the password in [13]). Other interesting arguments of study are the application of this type system to the problem of graves interferences [12] and the connections between this system and the logics for ambient calculi, [8], [14].

References

[1] S. Abramsky and C. H. L. Ong. Full abstraction in the lazy lambda calculus. *Information and Computation*, 105(2):159–267, 1993.

[2] H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *The Journal of Symbolic Logic*, 48(4):931–940, 1983.

[3] G. Boudol. Lambda-calculi for (strict) parallel functions. *Information and Computation*, 108(1):51–127, 1994.

[4] L. Cardelli, G. Ghelli, and A. D. Gordon. Mobility types for mobile ambients. In J. Wiederman, P. van Emde Boas, and M.Nielsen, editors, *ICALP'99*, volume 1644 of *LNCS*, pages 230–239, Berlin, 1999. Springer-Verlag.

[5] L. Cardelli and A. D. Gordon. Mobile ambients. In *FoSSaCS'98*, volume 1378 of *LNCS*, pages 140–155, Berlin, 1998. Springer-Verlag.

[6] L. Cardelli and A. D. Gordon. Equational properties of mobile ambients. In *FoSSaCS'98*, volume 1578 of *LNCS*, pages 212–226, Berlin, 1999. Springer-Verlag.

[7] L. Cardelli and A. D. Gordon. Types for mobile ambients. In *POPL'99*, pages 79–92, New York, 1999. ACM Press.

[8] L. Cardelli and A. D. Gordon. Anytime, anywhere. modal logics for mobile ambients. In *POPL'00*, pages 365–377. ACM Press, 2000.

[9] M. Coppo and M. Dezani-Ciancaglini. A fully abstract model for higher-order mobile ambients. In *VMCAI'02*, volume 2294 of *LNCS*, pages 255–271, Berlin, 2002. Springer-Verlag.

[10] C. Hartonas and M. Hennessy. Full abstractness for a functional/concurrent language with higher-order value-passing. *Information and Computation*, 145(1):64–106, 1998.

[11] M. Hennessy. A fully abstract denotational model for higher-order processes. *Information and Computation*, 112(1):55–95, 1994.

[12] F. Levi and D. Sangiorgi. Controlling interferences in ambients. In *POPL'00*, pages 352–364, New York, 2000. ACM Press.

[13] M. Merro and M. Hennessy. Bisimulation congruences in safe ambients. In *SIGPLAN Notices, 31(1)'02*, pages 71–80, New York, 2002. ACM Press.

[14] D. Sangiorgi. Extensionality and intensionality of the ambient logics. In *POPL'01*, pages 4–13, New York, 2001. ACM Press.

7 Appendix

Definition 7.1 (i) The function res: $(N^\star \times S) \to S$ is defined by structural induction on sequential types definition, as follows:

- $res\ (\overrightarrow{h}, \omega\ ) = \omega$
- $res\ (\overrightarrow{h}, \mu.\psi) = \mu.res\ (\overrightarrow{h}, \psi)$ if $fn(\mu) \cap \{\overrightarrow{h}\} = \Phi$
- $res\ (\overrightarrow{h}, \mu.\psi) = \omega$ otherwise
- $res\ (\overrightarrow{h}, \alpha.(\nu\overrightarrow{m})(< \varphi_1 >_n \varphi_2)) =$
  $= \alpha.(\nu\overrightarrow{m})(\nu\overrightarrow{h})(< \varphi_1 >_n \varphi_2)$ if $fn(\alpha) \cap \{\overrightarrow{h}\} = \Phi$
- $res\ (\overrightarrow{h}, \alpha.(\nu\overrightarrow{m})(< \varphi_1 >_n \varphi_2)) = \omega$ otherwise

(ii) The functions *ser*: $(S \times S) \to 2^S$ and *unfold*: $(N \times S) \to 2^S$ are defined by simultaneous induction on the weight.

a) For $|\varphi| + |\psi| = w$, we define $ser\ (\phi, \psi)$ in such a way that for every $\zeta \in ser\ (\phi, \psi) :\ |\zeta| \leq w$;

b) For $|\phi| = w$, we define $unfold\ (n, \phi)$ in such a way that for every $\zeta \in unfold\ (n, \phi) :\ |\zeta| \leq 1 + w$.

Base step:
a) $ser\ (\omega, \omega) = \omega$
b) $unfold\ (n, \omega) = \omega$

Inductive step:
a) We distinguish the following cases:

- $\varphi = \mu.\phi_1, \psi = \omega :\quad ser(\varphi, \psi) = \{\varphi\}$
- $\varphi = \mu.\phi_1, \psi = \nu.\psi_1,\quad \mu, \nu$ not matching:
  $ser(\varphi, \psi) = \{\mu.\zeta \mid \zeta \in ser(\varphi_1, \nu.\psi_1)\} \cup \{\nu.\xi \mid \xi \in ser(\mu.\phi_1, \psi_1)\}$
- $\varphi = \mu.\phi_1, \psi = \nu.\psi_1,\quad \mu, \nu$ matching :
  $ser(\varphi, \psi) = \{\chi \mid \chi \in ser(\varphi_1, \psi_1)\} \cup \{\mu.\zeta \mid \zeta \in ser(\phi_1, \nu.\psi_1)\} \cup$
  $\cup \{\nu.\xi \mid \xi \in ser(\mu.\phi_1, \psi_1)\}$
- $\varphi = \mu.\phi_1, \psi = \alpha.(\nu\overrightarrow{m})(< \psi_1 >_n \psi_2)$:
  $ser(\varphi, \psi) = \{\mu.\zeta \mid \zeta \in ser(\varphi_1, \psi)\} \cup$
  $\cup \{\alpha.(\nu\overrightarrow{m})(< \psi_1 >_n \chi) \mid \chi \in ser(\mu.\phi_1, \psi_2)\}$
- $\varphi = \alpha.(\nu\overrightarrow{m})(< \varphi_1 >_n \varphi_2), \psi = \beta.(\nu\overrightarrow{q})(< \psi_1 >_m \psi_2),$
  α, β not matching:
  $ser(\varphi, \psi) = \{\alpha.(\nu\overrightarrow{m})(< \varphi_1 >_n \xi) \mid \xi \in ser(\varphi_2, \psi)\} \cup$
  $\cup \{\beta.(\nu\overrightarrow{q})(< \chi_1 >_m \chi) \mid \chi \in ser(\varphi_2, \psi_2)\}$
- $\varphi = enter_p n.(\nu\overrightarrow{m})(< \varphi_1 >_n \varphi_2)$ and
  $\psi = \overline{enter}\ n.(\nu\overrightarrow{q})(< \psi_1 >_m \psi_2)$:
  $ser(\varphi, \psi) = \{enter_p n.(\nu\overrightarrow{m})(< \varphi_1 >_n \psi) \mid \xi \in ser(\phi_2, \psi)\} \cup$
  $\cup \{\overline{enter}\ n.(\nu\overrightarrow{q})(< \psi_1 >_n \chi) \mid \chi \in ser(\phi, \psi_2)\} \cup$
  $\cup \{\zeta \mid \zeta = res(\overrightarrow{m}\ \overrightarrow{q}, \zeta'), \zeta' \in ser(\zeta_1, \zeta_2) \mid$
  $\mid \zeta_1 \in unfold(n, \chi), \chi \in ser(\phi_1, \psi_1)$ and $\zeta_2 \in ser(\phi_2, \psi_2)\}$

b) We distinguish the following cases:

$-\ \varphi = \mu.\psi$:

$$\begin{aligned}
\mathit{unfold}(n,\varphi) &= \{\omega\} && \text{if } \mu \text{ is not compatible with } n\\
&= \{\mathit{enter}_n m.(<\zeta>_m \omega) \mid \zeta \in \mathit{unfold}(n,\psi)\} && \text{if } \mu = \mathit{in}\ n\\
&= \{\mathit{exit}_n m.(<\omega>_m \zeta) \mid \zeta \in \mathit{unfold}(n,\psi)\} && \text{if } \mu = \mathit{out}\ m\\
&= \{\overline{\mathit{enter}}\ n.(<\zeta>_n \omega) \mid \zeta \in \mathit{unfold}(n,\psi)\} && \text{if } \mu = \overline{\mathit{in}}\ n\\
&= \{\mathit{free}\ n.\psi\} && \text{if } \mu = \overline{\mathit{open}}\ n
\end{aligned}$$

$-\ \varphi = \alpha.(<\varphi_1>_n \varphi_2)$:

$$\begin{aligned}
\mathit{unfold}(n,\varphi) &= \{\omega\} && \text{if } \alpha \text{ is not compatible with } n\\
&= \{\mathit{pop}_m n.\zeta \mid \zeta \in \mathit{ser}(\chi,\varphi_2) \ \text{ with } \chi \in \mathit{unfold}(n,\varphi_1)\}\\
&&& \text{if } \ \alpha = \mathit{exit}_m n
\end{aligned}$$

Lemma 7.2 (i) *If ξ is a sequential type, $(\nu\overrightarrow{m})\xi \simeq \mathit{res}\ (\overrightarrow{m},\xi)$*

(ii) *If ξ is a sequential type, $n[\xi] \simeq \bigwedge_{\zeta\in\mathit{unfold}(n,\xi)} \zeta$*

(iii) *If ξ and χ are sequential types, $\xi \mid \chi \simeq \bigwedge_{\zeta\in\mathit{ser}(\xi,\chi)} \zeta$*

Proof. Obvious by definitions of $\simeq$, *res*, *ser*, and *unfold*. □