



What are Polymorphically-typed Ambients ?

**Torben Amtoft, Assaf J. Kfoury, Santiago Pericas-G.
Boston University**

**April 5, 2001
ESOP'01**

The Ambient Calculus

- Proposed by Cardelli & Gordon to model notions of

Location: ambients are nested, forming a dynamic tree structure

Mobility: ambients can enter and exit other ambients

Communication: inside an ambient, values can be exchanged between processes

- We propose to extend this with
 - a polymorphic type system
 - a functional layer

Purpose of Type Systems

Processes should be guaranteed not to receive values of wrong type:

$$m [\langle 7 \rangle \mid (x). \text{if } x \text{ then } \dots \text{ else } \dots]$$

- Solution in previous type systems [Cardelli & Gordon, etc]: allow each ambient to host only one topic of conversation throughout.
- Our solution: allow for an ambient to hold different topics of conversation *consecutively*

$$m [\langle 7 \rangle \mid (x). \text{open } n. \langle x = 42 \rangle \mid n [(y). \mathcal{P}]]$$

cf. session types in the π -calculus [Gay & Hole]

- Also we allow an ambient to hold different topics of conversation *simultaneously*, if of different arity.

Example: Packet Routing

- Packet Routing:

```
router [[:route {in packet.(dst).open hop.⟨lookup-route(dst)⟩}]] |
```

```
packet [in router.open route.⟨“genoa”⟩ | hop{(x).x}]]
```

- we use $n\{\mathcal{P}\}$ as an abbreviation for

$$n\{\mathcal{P}\} \triangleq n [\text{coopen } n \mid \mathcal{P}]$$

in the style of Levi & Sangiorgi’s co-capabilities.

AC+

- Expressions:

$$M \in \text{Exp} ::= n \mid c \mid \lambda n : \sigma. M \mid M_1 M_2 \mid \times (M_1, \dots, M_k)$$

| if M_0 then M_1 else M_2 | ... | $\epsilon \mid M_1.M_2$

| in M | out M | open M | coopen M

- Processes:

$$P \in \text{Proc} ::= \mathbf{0} \mid P_1 \mid P_2 \mid !P \mid (\nu n : \sigma). P \mid M.P \mid M[P]$$

| $(n_1 : \sigma_1, \dots, n_k : \sigma_k). P \mid \langle M \rangle$

Operational Semantics

- Mobility and Communication:

$$n[\text{in } m.P \mid Q] \mid m[R] \xrightarrow{\epsilon} m[n[P \mid Q] \mid R] \quad (\text{Red In})$$

$$m[n[\text{out } m.P \mid Q] \mid R] \xrightarrow{\epsilon} n[P \mid Q] \mid m[R] \quad (\text{Red Out})$$

$$\text{open } n.P \mid n[\text{coopen } n.Q \mid R] \xrightarrow{\epsilon} P \mid Q \mid R \quad (\text{Red Open})$$

$$\begin{aligned} & (n_1 : \sigma_1, \dots, n_k : \sigma_k).P \mid \langle \times (V_1, \dots, V_k) \rangle \\ & \xrightarrow{\text{comm}(\tau)} P[n_i := V_i] \text{ if } \tau = \times (\sigma_1, \dots, \sigma_k) \quad (\text{Red Comm}) \end{aligned}$$

Operational Semantics

- Expressions (standard CBV)

$$(\lambda n : \sigma.M)V \longrightarrow M[n := V] \quad (\text{Red Beta})$$

- Context

$$\text{If } M_1 \longrightarrow M_2 \text{ then } \mathcal{E}[M_1] \longrightarrow \mathcal{E}[M_2] \quad (\text{Red MctxtM})$$

$$\text{If } M_1 \longrightarrow M_2 \text{ then } \mathcal{P}[M_1]_e \xrightarrow{\epsilon} \mathcal{P}[M_2]_e \quad (\text{Red MctxtP})$$

$$\text{If } P \xrightarrow{\ell} Q \text{ then } \mathcal{P}[P]_p \xrightarrow{\ell} \mathcal{P}[Q]_p \quad (\text{Red PctxtP})$$

$$\text{If } P \xrightarrow{\ell} Q \text{ then } n[P] \xrightarrow{\epsilon} n[Q] \quad (\text{Red Amb})$$

$$\text{If } P' \equiv P, P \xrightarrow{\ell} Q, Q \equiv Q' \text{ then } P' \xrightarrow{\ell} Q' \quad (\text{Red } \equiv)$$

- Replication

$$!P \xrightarrow{\epsilon} P \mid !P \quad (\text{Red Repl})$$

Types and Behaviors

Behaviors

| | | |
|------------------------|---------------|--|
| $b \in \text{Beh} ::=$ | ε | no traceable action |
| $b_1.b_2$ | | first b_1 then b_2 |
| $b_1 \mid b_2$ | | parallel composition |
| $\text{put}(\sigma)$ | | output of type σ |
| $\text{get}(\sigma)$ | | input of type σ |
| diss | | ambient dissolution |
| \dots | | other behaviors according to need; |
| $\text{fromnow } T$ | | in this paper we have settled for: unordered communication of T |

$T \in \text{Topics} = \{ \{ \tau_1, \dots, \tau_m \} \mid m \geq 0 \text{ and } \text{arity}(\tau_i) \neq \text{arity}(\tau_j) \text{ for } i \neq j \}$

Types and Behaviors

Types

| | | |
|-----------------------------------|--|-----------------------------|
| $\sigma, \tau \in \text{Typ} ::=$ | <code>bool</code> <code>int</code> <code>real</code> <code>string</code> \dots | type constant |
| | $\sigma \rightarrow \tau$ | function type |
| | $\times(\sigma_1, \dots, \sigma_k)$ | tuple with arity $k \geq 0$ |
| | \dots | other type constructors |
| | <code>amb</code> [b, b'] | type of ambient name |
| | <code>cap</code> [B] | type of capability |

Shorthand: `amb`[b] for `amb`[b, b'] if $b = b'$ (cf. [Zimmer])

$B \in \text{BehCont} ::= \square$ | $(b.B)$ | $(B.b)$ | $(b \mid B)$ | $(B \mid b)$ behavior context

Example: Orderly Communication

By assigning n the type $\text{amb}[\text{get}(\text{bool}).b]$ (with b the behavior of P), we can construct a type derivation for

$$m[\langle 7 \rangle \mid (x).\text{open } n.\langle x = 42 \rangle \mid n\{ (y).\mathcal{P} \}]$$

where the process inside m has behavior

$$\text{put}(\text{int}) \mid \text{get}(\text{int}).(\text{put}(\text{bool}) \mid \boxed{\text{get}(\text{bool}).b}) \mid \varepsilon$$

which is *safe*, in that the only “well-formed” trace is

$$\text{put}(\text{int}) \text{ get}(\text{int}) \text{ put}(\text{bool}) \text{ get}(\text{bool})$$

Example: Routing

$$\begin{aligned} &router\ [!route\ \{\text{in}\ packet.\ (dst).\ \text{open}\ hop.\ \langle\text{lookup-route}(dst)\rangle\}\}] \mid \\ &packet\ [\text{in}\ router.\ \text{open}\ route.\ \langle\text{“genoa”}\rangle \mid hop\ \{(x).\ x\}] \end{aligned}$$

A type derivation for this process can be constructed as follows:

- Process inside hop : $get(\text{cap}[\square]).\varepsilon \mid \text{diss}$
- hop : $\text{amb}[get(\text{cap}[\square]).\varepsilon]$
- Process inside $route$: $b \mid \text{diss}$ where

$$b = get(\text{string}).(\text{get}(\text{cap}[\square]).\varepsilon \mid \text{put}(\text{cap}[\square]))$$

- $route$: $\text{amb}[b]$
- Process inside $packet$: $b \mid \text{put}(\text{string})$

Typing Rules

(Proc Par)

$$\frac{E \vdash P_1 : b_1 \quad E \vdash P_2 : b_2}{E \vdash P_1 \mid P_2 : b_1 \mid b_2}$$

(Proc Output)

$$\frac{E \vdash M : \times(\tau_1, \dots, \tau_k)}{E \vdash \langle M \rangle : \text{put}(\times(\tau_1, \dots, \tau_k))}$$

(Proc Input)

$$\frac{E, n_1 : \tau_1, \dots, n_k : \tau_k \vdash P : b}{E \vdash (n_1 : \tau_1, \dots, n_k : \tau_k).P : \text{get}(\times(\tau_1, \dots, \tau_k)).b}$$

Typing Rules

(Proc Action)

$$\frac{E \vdash M : \text{cap}[B] \quad E \vdash P : b}{E \vdash M.P : B[b]}$$

(Exp In)

$$\frac{E \vdash M : \text{amb}[b, b']}{E \vdash \text{in } M : \text{cap}[\square]}$$

(Exp Coopen)

$$\frac{E \vdash M : \text{amb}[b, b']}{E \vdash \text{coopen } M : \text{cap}[\text{diss.}\square]}$$

(Exp Open)

$$\frac{E \vdash M : \text{amb}[b, b']}{E \vdash \text{open } M : \text{cap}[b' \mid \square]}$$

Subtyping and Behavior Subsumption

- On base types, we have $\text{int} \leq \text{real}$. On composite types, the ordering is defined using the following polarity rules:

$$\ominus \rightarrow \oplus \quad (\oplus, \dots, \oplus) \quad \text{amb}[\ominus, \oplus] \quad \text{cap}[\oplus]$$

- We write $b_1 \leq b_2$ when b_2 is more “permissive” than b_1 , as in

$$\begin{aligned} \text{put}(\text{int}) &\leq \text{put}(\text{real}) \\ \text{get}(\text{real}) &\leq \text{get}(\text{int}) \\ \text{put}(\text{int}) &\leq \text{fromnow} \{ \text{int} \} \end{aligned}$$

Thus “descriptive” rather than “prescriptive” point of view.

- The ordering on behaviors gives rise to an ordering on behavior contexts: $B_1 \leq B_2$ holds iff for all b we have $B_1[b] \leq B_2[b]$.

Traces = Behavior Semantics

1. A trace $tr \in \text{Trace}$ is a finite sequence of actions, where an action $a \in \text{Act}$ is either $\text{put}(\tau)$, $\text{get}(\tau)$, or diss .
2. The semantics $\llbracket b \rrbracket$ of a behavior b belongs to the powerset $\mathcal{P}(\text{Trace})$, and is given by

$$\llbracket \varepsilon \rrbracket = \{\bullet\} \quad \llbracket \text{diss} \rrbracket = \{\text{diss}\}$$

$$\llbracket b_1.b_2 \rrbracket = \llbracket b_1 \rrbracket \diamond \llbracket b_2 \rrbracket \quad \llbracket b_1 \mid b_2 \rrbracket = \llbracket b_1 \rrbracket \parallel \llbracket b_2 \rrbracket$$

$$\llbracket \text{put}(\tau) \rrbracket = \{\text{put}(\tau)\} \quad \llbracket \text{get}(\tau) \rrbracket = \{\text{get}(\tau)\}$$

$$\llbracket \text{fromnow } T \rrbracket = \{tr \mid \forall a \in tr : a \in \{\text{put}(\tau), \text{get}(\tau)\} \text{ for } \tau \in T\}$$

3. $b_1 \leq b_2$ iff $\llbracket b_1 \rrbracket \leq \llbracket b_2 \rrbracket$, that is for all $tr_1 \in \llbracket b_1 \rrbracket$ there exists $tr_2 \in \llbracket b_2 \rrbracket$ with $tr_1 \leq tr_2$.

Typing Rules

$$\frac{E \vdash P : b}{E \vdash !P : b} \text{ (Proc Repl)} \quad (\text{if } (b \mid b) \leq b)$$

$$\text{(Proc Amb)} \quad \frac{E \vdash M : \text{amb}[b, b'] \quad E \vdash P : b_1}{E \vdash M[P] : \varepsilon} \quad (\text{if } b_1 \text{ safe and } b_1 \rightsquigarrow b \text{ and } b \leq b')$$

Here b is safe if no trace $tr \in \llbracket b \rrbracket$ is of the form ($n \geq 0$ and tr arbitrary)

$$\text{put}(\tau_1)\text{get}(\sigma_1) \dots \text{put}(\tau_n)\text{get}(\sigma_n)\text{put}(\tau)\text{get}(\sigma) \diamond tr$$

with $\text{arity}(\tau) = \text{arity}(\sigma)$ but $\tau \not\leq \sigma$, and $\forall i \in \{1 \dots n\} : \tau_i \leq \sigma_i$.

Lemmas and Theorems

- **[Subject reduction for processes]** Suppose that $P \xrightarrow{\ell} Q$. If with b safe it holds that $E \vdash P : b$ then there exists safe b' and b_0 with $\ell \sim b_0$ such that $E \vdash Q : b'$ and $b_0.b' \leq b$.
Here $\epsilon \sim \varepsilon$ and $\text{comm}(\tau) \sim \text{put}(\tau^-).\text{get}(\tau^+)$ if $\tau^- \leq \tau \leq \tau^+$.
- **[Decidability issues]** Using finite automata, we can decide
 - **[subtyping]** whether $\tau \leq \sigma$
 - **[subsumption]** whether $b_1 \leq b_2$
 - **[type checking]** whether a purported derivation of $E \vdash M : \tau$ or $E \vdash P : b$ is in fact valid.

Future work

- An algorithm for type *inference*?
- Extend traces/behaviors to record ambient movements (useful for security)
- Explore relation to other type systems, such as the single-threaded types of Levi & Sangiorgi
- Denotational semantics for the ambient calculus by means of traces?