

# Types for Mobile Ambients

Andy Gordon, Microsoft Research

(based on joint work with Luca Cardelli, Microsoft Research)

## Orientation: Ambients

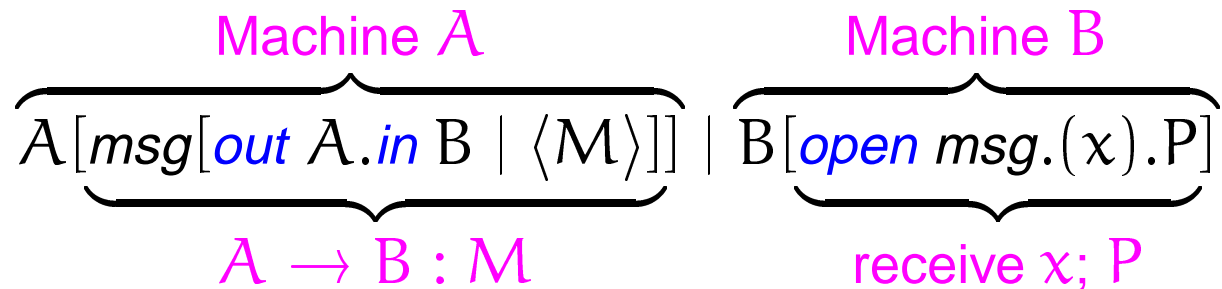
Ambients describe the mobility of software and hardware, including the passage across administrative boundaries.

Ambients are named, bounded places where computation happens.

Ambient security rests on the controlled distribution of suitable credentials, or **capabilities**, derived from unforgeable **names**.

One goal of this work is to develop a flexible, precise, secure, and typeful programming model for mobile software components.

## Mobile Ambients: a packet from $A$ to $B$



- Ambients may model both machines and packets
- Ambients are mobile:  $\text{msg}[\cdot \cdot \cdot]$  moves out of  $A$  and into  $B$
- Ambients are boundaries: passage is regulated by **capabilities**

You need capability *out*  $A$  to exit  $A$ ; you need capability *in*  $B$  to enter  $B$

## Ambient Behaviour, By Example

There are four basic reduction rules in the calculus:

$$\begin{aligned}
 & A[msg[*out* A.in B \mid \langle M \rangle]] \mid B[*open* msg.(x).P] \\
 & \rightarrow A[] \mid msg[*in* B \mid \langle M \rangle] \mid B[*open* msg.(x).P] \\
 & \rightarrow A[] \mid B[msg[\langle M \rangle] \mid *open* msg.(x).P] \\
 & \rightarrow A[] \mid B[\langle M \rangle \mid (x).P] \\
 & \rightarrow A[] \mid B[P\{x \leftarrow M\}]
 \end{aligned}$$

## Mobility and Communication Primitives

$M ::=$	expression
$n$	ambient name
$in M$	can enter into $M$
$out M$	can exit out of $M$
$open M$	can open $M$
$P, Q, R ::=$	process
$(\nu n)P$	restriction
$0$	inactivity
$P \mid Q$	composition
$!P$	replication
$M[P]$	ambient
$M.P$	action
$(n_1, \dots, n_k).P$	input action
$\langle M_1, \dots, M_k \rangle$	async output action

## Subjective versus Objective Moves

We base ambient mobility on **subjective** moves:

$$\begin{aligned} n[in\ m.P \mid Q] \mid m[R] &\rightarrow m[n[P \mid Q] \mid R] \\ m[n[out\ m.P \mid Q] \mid R] &\rightarrow n[P \mid Q] \mid m[R] \end{aligned}$$

Instead, we might have adopted primitives for **objective** moves:

$$\begin{aligned} mv\ in\ n.P \mid n[Q] &\rightarrow n[P \mid Q] \\ n[mv\ out\ n.P \mid Q] &\rightarrow P \mid n[Q] \end{aligned}$$

But objective moves only move still ambients, and they allow kidnap:

$$m[P] \mid (\nu k)(k[] \mid mv\ in\ m.in\ k) \rightarrow^* (\nu k)k[m[P]]$$

## Objective Ambient Moves

The special case of objective movement of an ambient is safe, convenient, and derivable from subjective movement:

$$\textit{move } M.n[P] \stackrel{\Delta}{=} (\nu k)k[M.n[\textit{out } k.P]] \quad \text{for } k \text{ not free in } M.P$$

For example:

$$\begin{aligned} \textit{move in } m.n[P] \mid m[R] &\rightarrow (\nu k)m[k[n[\textit{out } k.P]] \mid R] \\ &\rightarrow (\nu k)m[k[] \mid n[P] \mid R] \\ &\simeq m[n[P] \mid R] \end{aligned}$$

The relation  $\simeq$  is a semantic equivalence used for garbage collection.

# Regulating Input/Output

## Related Work

Our work borrows ideas from previous treatments of types for the  $\lambda$ -calculus and the  $\pi$ -calculus.

In particular, Milner's sorts for  $\pi$ , Pierce and Sangiorgi's type system for  $\pi$ , and Kobayashi, Pierce, and Turner's linear type system for  $\pi$ .

Some quite sophisticated type systems are being investigated for mobile computation, e.g., by Sewell, Hennessy and Riely, Jeffrey, . . .

Our approach is to investigate several simple systems that regulate exchanges, mobility, security levels, etc., and attempt to integrate them into a coherent whole.

## Motivation for Exchange Types

In the untyped calculus, certain processes arise that make no sense:

- Process  $in\ n[P]$  uses a capability as an ambient name
- Process  $n.P \mid n[Q]$  uses an ambient name as a capability

In an implementation, these processes are execution errors.

To avoid these errors, we regulate the types of messages a process may **exchange**, that is, input or output.

## Typing Input and Output

If a message  $M$  has message type  $W$ , then  $\langle M \rangle$  is a process that exchanges  $W$  messages.

If  $M : W$  then  $\langle M \rangle : W$ .

If  $P$  is a process that exchanges  $W$  messages, then  $(x:W).P$  is also a process that exchanges  $W$  messages.

If  $P : W$  then  $(x:W).P : W$ .

## Typing Parallelism

Process  $\mathbf{0}$  exchanges messages of any type, since it exchanges none.

$\mathbf{0} : T$  for all  $T$ .

If  $P$  and  $Q$  are processes that exchange  $T$  messages, so is  $P \mid Q$ .

If  $P : T$  and  $Q : T$  then  $P \mid Q : T$ .

If  $P : T$  then  $!P : T$ .

These rules ensure matching of the types of inputs and outputs from processes running in parallel.

## Typing Ambients

An expression of type  $Amb[T]$  names an ambient inside which  $T$  messages are exchanged.

If  $M$  is such an expression, and  $P$  is a process that exchanges  $T$  messages, then  $M[P]$  is correctly typed.

If  $M : Amb[T]$  and  $P : T$  then  $M[P] : S$  for all  $S$ .

An ambient exchanges no messages, so it may be assigned any type.

## Typing Capabilities

An expression of type  $Cap[T]$  is a capability that may unleash exchanges of type  $T$ .

If  $M : Cap[T]$  and  $P : T$  then  $M.P : T$ .

If ambients named  $n$  exchange  $T$  messages, then the capability  $open\ n$  may unleash these exchanges.

If  $n : Amb[T]$  then  $open\ n : Cap[T]$ .

Capabilities  $in\ n$  and  $out\ n$  unleash no exchanges.

If  $n : Amb[S]$  then  $in\ n : Cap[T]$  for all  $T$ .

If  $n : Amb[S]$  then  $out\ n : Cap[T]$  for all  $T$ .

## Exchange Types

### Types:

$W ::=$	message types
$Amb[T]$	ambient name allowing $T$ exchanges
$Cap[T]$	capability unleashing $T$ exchanges
$S, T ::=$	exchange types
$Shh$	no exchange
$W_1 \times \dots \times W_k$	tuple exchange

- A quiet ambient,  $Amb[Shh]$ , and a harmless capability,  $Cap[Shh]$
- An ambient allowing exchange of harmless capabilities:  $Amb[Cap[Shh]]$
- A capability unleashing exchanges of names of quiet ambients:  $Cap[Amb[Shh]]$

## Properties of Exchange Types

Formally, we base our type system on judgments  $E \vdash M : W$  and  $E \vdash P : T$ , where  $E = x_1:W_1, \dots, x_k:W_k$ .

**Theorem** (Soundness) If  $E \vdash P : T$  and  $P \rightarrow Q$  then  $E \vdash Q : T$ .

Hence, execution errors like *in*  $n[P]$  and  $n.P \mid n[Q]$  cannot arise during a computation, since they are not typeable.

## Examples of Typing

Packet from  $A$  to  $B$ :

If  $A : \mathit{Amb}[\mathit{Shh}]$ ,  $B$ ,  $\mathit{msg} : \mathit{Amb}[\mathit{W}]$ , and  $M, P : \mathit{W}$  then  
 $A[\mathit{msg}[\underbrace{\mathit{out} A.\mathit{in} B}_{\mathit{Cap}[\mathit{W}]} \mid \langle M \rangle]] : \mid B[\underbrace{\mathit{open} \mathit{msg} . (x:\mathit{W}).P}_{\mathit{Cap}[\mathit{W}]}] : \mathit{Shh}.$

Objective ambient move:

If  $M : \mathit{Cap}[\mathit{T}]$  and  $n[P] : S$  then  $\mathit{move} M.n[P] : S.$

# Typed Semantics of a Distributed Language

## Programming Model

There is a flat collection of named nodes (or locations), each of which contains a group of named channels and anonymous threads:

*node* A [*thread* [*go* B.*b*(*node*, *ch*).*go* *node*. $\overline{ch}\langle A \rangle$ ]] |

*node* B [*channel* *b* | *thread* [ $\overline{b}\langle C, c \rangle$ ]] |

*node* C [*channel* *c*]

Heterogeneous models like this underly several distributed programming systems, and several distributed forms of the  $\pi$ -calculus.

### A fragment of a typed, distributed programming language:

$Net ::=$	network
$(\forall n:Ty) Net$	restriction
$Net \mid Net$	composition of networks
$node\ n\ [Grp]$	node
$Grp ::=$	group of channels and threads
$(\forall n:Ty) Grp$	restriction
$Grp \mid Grp$	composition of groups
$channel\ c$	channel
$thread\ [Th]$	thread
$Th ::=$	thread
$go\ n.Th$	migration
$\bar{c}\langle n_1, \dots, n_k \rangle$	output to a channel
$c(x_1:Ty_1, \dots, x_k:Ty_k).Th$	input from a channel
$fork(Grp).Th$	fork a group

## A Typed Semantics

Much as in distributed forms of  $\pi$ , each name has a type,  $Ty$ .

**Types of names:**

$Ty ::=$	type
<i>Node</i>	name of a node
<i>Ch</i> $[Ty_1, \dots, Ty_k]$	name of a channel

**Translation of a type  $Ty$ ,  $\llbracket Ty \rrbracket$**

$$\begin{aligned} \llbracket \textit{Node} \rrbracket &\triangleq \textit{Amb}[\textit{Shh}] \\ \llbracket \textit{Ch}[Ty_1, \dots, Ty_k] \rrbracket &\triangleq \textit{Amb}[\llbracket Ty_1 \rrbracket \times \dots \times \llbracket Ty_k \rrbracket] \end{aligned}$$

### Translation of a network $Net$ , $\llbracket Net \rrbracket$

$$\llbracket (\nu n:Ty) Net \rrbracket \stackrel{\Delta}{=} (\nu n:\llbracket Ty \rrbracket) \llbracket Net \rrbracket$$

$$\llbracket Net \mid Net \rrbracket \stackrel{\Delta}{=} \llbracket Net \rrbracket \mid \llbracket Net \rrbracket$$

$$\llbracket \text{node } n \text{ } [Grp] \rrbracket \stackrel{\Delta}{=} n[!\text{open } n \mid \llbracket Grp \rrbracket_n]$$

### Translation of a group $Grp$ located at $n$ , $\llbracket Grp \rrbracket_n$

$$\llbracket (\nu m:Ty) Grp \rrbracket_n \stackrel{\Delta}{=} (\nu m:\llbracket Ty \rrbracket) \llbracket Grp \rrbracket_n \quad \text{for } m \neq n$$

$$\llbracket Grp \mid Grp \rrbracket_n \stackrel{\Delta}{=} \llbracket Grp \rrbracket_n \mid \llbracket Grp \rrbracket_n$$

$$\llbracket \text{channel } c \rrbracket_n \stackrel{\Delta}{=} c[!\text{open } c]$$

$$\llbracket \text{thread } Th \rrbracket_n \stackrel{\Delta}{=} (\nu t:\text{Amb}[\text{Shh}])t[\llbracket Th \rrbracket_n^t] \quad \text{for } t \neq n \text{ not free in } Th$$

Translation of a thread  $Th$  named  $t$  located at  $n$ ,  $\llbracket Th \rrbracket_n^t$

$$\begin{aligned}
 \llbracket go\ m.\ Th \rrbracket_n^t &\triangleq out\ n.\ in\ m.\ \llbracket Th \rrbracket_m^t \\
 \llbracket \bar{c}\langle n_1, \dots, n_k \rangle \rrbracket_n^t &\triangleq move\ (out\ t.\ in\ c).\ c[\langle n_1, \dots, n_k \rangle] \\
 \llbracket c(x_1:Ty_1, \dots, x_k:Ty_k).\ Th \rrbracket_n^t &\triangleq (\nu s:Amb[Shh]) \\
 &\quad (move\ (out\ t.\ in\ c).\ \\
 &\quad\quad c[(x_1:\llbracket Ty_1 \rrbracket, \dots, x_k:\llbracket Ty_k \rrbracket)]. \\
 &\quad\quad move\ (out\ c.\ in\ t).\ s[\llbracket Th \rrbracket_n^t] \mid \\
 &\quad\quad open\ s) \\
 \llbracket fork\ (Grp).\ Th \rrbracket_n^t &\triangleq (\nu m:Amb[Shh]) \\
 &\quad (move\ out\ t.\ n[open\ m.\ \llbracket Grp \rrbracket_n] \mid \\
 &\quad\quad move\ out\ t.\ m[move\ in\ t.\ t[]] \mid \\
 &\quad\quad open\ t.\ \llbracket Th \rrbracket_n^t)
 \end{aligned}$$

## Exchange Types Support a Typed Semantics

Let  $\llbracket E \rrbracket \triangleq n_1:\llbracket Ty_1 \rrbracket, \dots, n_k:\llbracket Ty_k \rrbracket$  if  $E = n_1:Ty_1, \dots, n_k:Ty_k$ .

(1) If  $E \vdash Net$  then  $\llbracket E \rrbracket \vdash \llbracket Net \rrbracket : Shh$ .

(2) If  $E \vdash Grp$  and  $E \vdash n : Node$  then  $\llbracket E \rrbracket \vdash \llbracket Grp \rrbracket_n : Shh$ .

(3) If  $E \vdash Th$  and  $E \vdash n : Node$  and  $t \notin dom(E)$ ,  
then  $\llbracket E \rrbracket, t:Amb[Shh] \vdash \llbracket Th \rrbracket_n^t : Shh$ .

# Regulating Mobility

## Motivation for Mobility Types

Although in general ambients should be mobile, some important classes of ambients (like network nodes or channels) should be immobile.

We annotate the syntax of ambients to draw the distinction:

- $n^{\checkmark}[P]$  may move, whereas
- $n^{\times}[P]$  may not.

The purpose of the mobility type system is to rule out execution errors such as  $n^{\times}[in\ m]$  or  $n^{\times}[out\ m]$ .

## Extending the System of Exchange Types

### Types:

 $Z ::= \checkmark \mid \times$ 

mobility annotations

 $W ::= \mathit{Amb}^Z[T] \mid \mathit{Cap}^Z[T]$ 

message types

 $T ::= \mathit{Shh} \mid W_1 \times \cdots \times W_k$ 

exchange types

- A quiet immobile ambient,  $\mathit{Amb}^\times[\mathit{Shh}]$
- A quiet mobile ambient  $\mathit{Amb}^\checkmark[\mathit{Shh}]$ .
- A capability unleashing no mobility effects and no exchanges  $\mathit{Cap}^\times[\mathit{Shh}]$
- A capability that may unleash mobility effects, but no exchanges,  $\mathit{Cap}^\checkmark[\mathit{Shh}]$

**Judgments:** $E \vdash M : W$ good expression of message type  $W$  $E \vdash P :^Z T$ process with mobility  $Z$  exchanging  $T$ **Good expressions:** $E \vdash n : \mathit{Amb}^Z[T]$  $E \vdash n : \mathit{Amb}^Z[T]$  $E \vdash n : \mathit{Amb}^Z[T]$  $E \vdash \mathit{in} n : \mathit{Cap}^\checkmark[T']$  $E \vdash \mathit{out} n : \mathit{Cap}^\checkmark[T']$  $E \vdash \mathit{open} n : \mathit{Cap}^Z[T]$  $E \vdash \diamond$  $E \vdash M : \mathit{Cap}^Z[T] \quad E \vdash M' : \mathit{Cap}^Z[T]$  $E \vdash \epsilon : \mathit{Cap}^Z[T]$  $E \vdash M.M' : \mathit{Cap}^Z[T]$ 

A capability for moving a mobile ambient in and out of an immobile ambient:

 $n : \mathit{Amb}^x[\mathit{Shh}] \vdash \mathit{in} n . \mathit{out} n : \mathit{Cap}^\checkmark[T]$  for all  $T$ .

**Good processes:**

$$\begin{array}{c}
 \frac{E \vdash M : \mathit{Amb}^Z[T] \quad E \vdash P :^Z T}{E \vdash M^Z[P] :^{Z'} T'} \quad \frac{E \vdash M : \mathit{Cap}^Z[T] \quad E \vdash P :^Z T}{E \vdash M.P :^Z T} \\
 \\
 \frac{E \vdash P :^Z T \quad E \vdash Q :^Z T}{E \vdash P \mid Q :^Z T} \quad \frac{E, p : \mathit{Amb}^Z[T] \vdash P :^{Z'} T'}{E \vdash (\nu p)P :^{Z'} T'} \quad \frac{E \vdash \diamond}{E \vdash \mathbf{0} :^Z T} \\
 \\
 \frac{E, n_1 : W_1, \dots, n_k : W_k \vdash P :^Z W_1 \times \dots \times W_k}{E \vdash (n_1, \dots, n_k).P :^Z W_1 \times \dots \times W_k} \\
 \\
 \frac{E \vdash M_1 : W_1 \quad \dots \quad E \vdash M_k : W_k}{E \vdash \langle M_1, \dots, M_k \rangle :^Z W_1 \times \dots \times W_k}
 \end{array}$$

## An Inflexibility

This basic system is sound, but immobile ambients cannot easily interact with mobile ambients.

For example, we cannot type a packet  $p$  being sent to an immobile ambient  $q$ , whether we make the packet  $\checkmark$  or  $x$ :

- $p^{\checkmark}[in\ q.\langle M \rangle] \mid q^x[open\ p.(x).R]$
- $p^x[in\ q.\langle M \rangle] \mid q^x[open\ p.(x).R]$

But, in fact, no execution error can arise in either case.

## Objective Ambient Movement as Primitive

We add *move*  $M.N^Z[P]$  to the syntax of typed processes.

The reduction rules are derived from the untyped encoding  $(\nu k)k[M.N[out\ k.P]]$ . For example:

$$move\ in\ q.p^x[\langle M \rangle] \mid q^x[open\ p.(x).R] \rightarrow q^x[p^x[\langle M \rangle] \mid open\ p.(x).R]$$

An objective ambient move exchanges no messages, and has no mobility effects, so we have:

If  $M : Cap^v[S]$  and  $N : Amb^Z[T]$  and  $P :^Z T$   
 then *move*  $M.N^Z[P] :^{Z'} T'$ .

A subject reduction theorem holds for this extended system.

## Threads are Mobile; Nodes, Channels are Immobile

A more accurate typed semantics of the distributed language:

$$\begin{aligned}
 \llbracket \text{Node} \rrbracket &\triangleq \text{Amb}^{\times}[\text{Shh}] \\
 \llbracket \text{Ch}[Ty_1, \dots, Ty_k] \rrbracket &\triangleq \text{Amb}^{\times}[\llbracket Ty_1 \rrbracket \times \dots \times \llbracket Ty_k \rrbracket] \\
 \llbracket \text{node } n \text{ [Grp]} \rrbracket &\triangleq n^{\times}[\text{!open } n \mid \llbracket \text{Grp} \rrbracket_n] \\
 \llbracket \text{channel } c \rrbracket_n &\triangleq c^{\times}[\text{!open } c] \\
 \llbracket \text{thread } Th \rrbracket_n &\triangleq (\forall t: \text{Amb}^{\checkmark}[\text{Shh}]) t^{\checkmark}[\llbracket Th \rrbracket_n^t]
 \end{aligned}$$

If  $E \vdash \text{Net}$  then  $\llbracket E \rrbracket \vdash \llbracket \text{Net} \rrbracket :^{\times} \text{Shh}$ .

If  $E \vdash \text{Grp}$  and  $E \vdash n : \text{Node}$  then  $\llbracket E \rrbracket \vdash \llbracket \text{Grp} \rrbracket_n :^{\times} \text{Shh}$ .

If  $E \vdash Th$ ,  $E \vdash n : \text{Node}$ ,  $t \notin \text{dom}(E)$  then  $\llbracket E \rrbracket, t: \text{Amb}^{\checkmark}[\text{Shh}] \vdash \llbracket Th \rrbracket_n^t :^{\checkmark} \text{Shh}$ .

## Summary

A goal of developing our calculus is to prototype a flexible, precise, secure, and typeful programming model for mobile software components.

Types regulate aspects of mobile computation such as exchanging messages and exercising capabilities for mobility.

Hence, the ambient calculus serves as a typed metalanguage for describing mobile computation.