Part III: The Ambient Calculus

In this part:

- Ambients as units of mobility and security
- The untyped ambient calculus
- Types for regulating ambient behaviours

A calculus to describe the movement of processes and devices, including movement through administrative domains, and to suggest flexible ways of programming mobility.

Background: Mobile Hardware and Software

Computation is getting more mobile than it used to be:

- devices: laptops, palmtops, smartcards, smartphones, . . .
- protocols: Mobile IP, WAP, Bluetooth, . . .
- code: Java applets, ECMAscript, WAPscript, . . .
- computation: Facile, Telescript, Obliq, Voyager, . . .

Consequently, programming is getting more complicated, and security risks loom larger than ever, such as:

- secrecy risks: e.g., protect login credentials from smartcard reader
- integrity risks: e.g., prevent malicious applet from formatting the hard drive

Two Underlying Problems

First, security problems arise not so much from mobility itself (after all, most code is mobile), but from careless or malicious crossing of **administrative domains**.

Administrative domains are second-class citizens; for example, security policies for untrusted code in Java are defined outside the language in terms of stack walking.

What would **first-class** administrative domains look like?

Second, in Telescript or Obliq it is easy to move either a whole application or a single object, but problematic to move a **cluster** of logically related objects and threads.

The Idea of an Ambient

An ambient is a bounded place where computation happens, with an inside and an outside.

An ambient is both a unit of mobility—of either software and hardware—and an administrative domain.

An ambient may contain other ambients, to model related clusters of objects, or hierarchical administrative domains.

An ambient has an unforgeable **name**.

An ambient's security rests on the controlled distribution of suitable credentials, or **capabilities**, derived from its name.

Our Aims

To study mobile computation we formalize ambients within a process calculus, the **ambient calculus**.

Calculi of functions, processes, and objects clarify existing styles of computation. Sometimes they suggest better programming habits too.

Our goal is that the theory and implementation of the ambient calculus will do the same for mobile computation.

Specifically, in this part of the course, we use ambients to develop type systems for mobility, adaptable for use in a bytecode verifier, for example.

The Untyped Ambient Calculus

Formalising Ambients

Our starting point, Milner, Parrow, and Walker's π-calculus:

- groups processes in a **single**, **contiguous**, **centralised** collection
- enables interaction by **shared names**, used as communication channels
- has no direct account of access control

Our ambient calculus:

- groups processes in **multiple**, **disjoint**, **distributed** ambients
- enables interaction by **shared position**, with no action at a distance
- uses **capabilities**, derived from ambient names, for access control

Mobile Ambients: a packet from A to B

$$\overbrace{A[\underbrace{msg[\mathit{out}\ A.\mathit{in}\ B \mid \langle M\rangle]}_{A \to B : M}]}^{\text{Machine } A} \mid \overbrace{B[\underbrace{\mathit{open}\ msg.(x).P}_{\text{receive } x;\ P}]}^{\text{Machine } B}$$

- Ambients may model both machines and packets
- Ambients are mobile: $msg[\cdots]$ 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

Exiting an Ambient

The capability *out* A allows the ambient *msg* to exit the ambient A:

$$A[\mathit{msg}[\mathit{out}\ A.\mathit{in}\ B \mid \langle M \rangle]]$$
$$\longrightarrow\ A[] \mid \mathit{msg}[\mathit{in}\ B \mid \langle M \rangle]$$

Ambient *msg* is the unit of mobility, which crosses the perimeter A.

Entering an Ambient

The capability *in* B allows the ambient *msg* to enter the ambient B:

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

Ambient *msg* is the unit of mobility, which crosses the perimeter B.

Opening an Ambient

The capability *open* *msg* dissolves the boundary around ambient *msg*:

$$\begin{aligned} & \mathit{msg}[\langle M \rangle] \mid \mathit{open}\ \mathit{msg}.(x).P \\ \rightarrow\ & \langle M \rangle \mid (x).P \end{aligned}$$

The ambient *msg* is the unit of mobility in that as its perimeter is breached, its subprocesses become subprocesses of the top-level.

Exchanging a Message

If there is no intervening boundary, messages may be exchanged:

$$\langle M \rangle \mid (x).P \;\rightarrow\; P\{x \leftarrow M\}$$

In the processes below, the boundary n prevents exchange of M:

$$n[\langle M \rangle] \mid (x).P$$

$$\langle M \rangle \mid n[(x).P]$$

Ambient Behaviour, By Example

Altogether, we have:

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

Syntax of the Untyped Ambient Calculus:

$M ::=$	expression
$\quad n$	$\quad$ ambient name
$\quad$ *in* M	$\quad$ can enter into M
$\quad$ *out* M	$\quad$ can exit out of M
$\quad$ *open* M	$\quad$ can open M
$P, Q, R ::=$	process
$\quad$ *new*$(n)P$	$\quad$ restriction
$\quad$ *stop*	$\quad$ inactivity
$\quad P \mid Q$	$\quad$ composition
$\quad$ *repeat* P	$\quad$ replication
$\quad M[P]$	$\quad$ ambient
$\quad M.P$	$\quad$ action
$\quad (x_1, \ldots, x_k).P$	$\quad$ input action
$\quad \langle M_1, \ldots, M_k \rangle$	$\quad$ asynchronous output action

Variation: Subjective versus Objective Moves

Subjective: “I move. I become a child ambient.”

$$n[\mathit{in}\ m.P \mid Q] \mid m[R] \;\rightarrow\; m[n[P \mid Q] \mid R]$$

$$m[n[\mathit{out}\ m.P \mid Q] \mid R] \;\rightarrow\; n[P \mid Q] \mid m[R]$$

Objective: “I make you move. You become a local process.”

$$\mathit{mv\ in}\ n.P \mid n[Q] \;\rightarrow\; n[P \mid Q]$$

$$n[\mathit{mv\ out}\ n.P \mid Q] \;\rightarrow\; P \mid n[Q]$$

Variation: Open versus Acid

Ambient acid: “I dissolve my own boundary.”

$$n[\mathit{acid}.P \mid Q] \;\rightarrow\; P \mid Q$$

Objective moves derivable:

$$\begin{aligned}\mathit{mv\ in}\ n.P &\triangleq \mathit{new}(q)q[\mathit{in}\ n.\mathit{acid}.P]\\ \mathit{mv\ out}\ n.P &\triangleq \mathit{new}(q)q[\mathit{out}\ n.\mathit{acid}.P]\end{aligned}$$

But the risk is that objective moves allow ambient kidnap:

$$\begin{aligned}\mathit{entrap}\ m &\triangleq \mathit{new}(k)(k[] \mid \mathit{mv\ in}\ m.\mathit{in}\ k)\\ \mathit{entrap}\ m \mid m[P] &\rightarrow^{*} \mathit{new}(k)k[m[P]]\end{aligned}$$

Examples in the Untyped Calculus

- Locks
- Objective Moves and Dissolution
- Booleans
- Numerals
- Turing Machines
- The Choice-Free Asynchronous π-calculus
- The λ-calculus
- Mutable cells
- Routable packets and active networks

Example: Boolean Flags

$$\textit{flag}\ n \triangleq n[]$$

$$\begin{aligned}
&\textit{if}\ tt\ P, \textit{if}\ ff\ Q \triangleq \\
&\quad \textit{new}(k)(k[] \mid \\
&\qquad\qquad \textit{open}\ tt.\textit{open}\ k.\textit{new}(t)(ff[t[]] | \textit{open}\ t.P) \mid \\
&\qquad\qquad \textit{open}\ ff.\textit{open}\ k.\textit{new}(f)(tt[f[]] | \textit{open}\ f.Q))
\end{aligned}$$

We have:

$$\textit{flag}\ tt \mid \textit{if}\ tt\ P, \textit{if}\ ff\ Q \ \longrightarrow^* \approx \ P$$

$$\textit{flag}\ ff \mid \textit{if}\ tt\ P, \textit{if}\ ff\ Q \ \longrightarrow^* \approx \ Q$$

Example: Encoding Objective Moves I

$$\begin{aligned}
n^{\downarrow}[P] &\triangleq n[P \mid \mathit{allow}\ in] \\
\mathit{allow}\ n &\triangleq \mathit{repeat\ open}\ n \\
\mathit{mv\ in}\ n.P &\triangleq \mathit{new}(k)k[\mathit{in}\ n.in[\mathit{out}\ k.P]]
\end{aligned}$$

We get:

$$\begin{aligned}
\mathit{mv\ in}\ n.P \mid n^{\downarrow}[Q] &\rightarrow n^{\downarrow}[\mathit{new}(k)k[in[\mathit{out}\ k.P]] \mid Q] \\
&\rightarrow n^{\downarrow}[\mathit{new}(k)k[] \mid in[P] \mid Q] \\
&\rightarrow n^{\downarrow}[\mathit{new}(k)k[] \mid P \mid Q] \\
&\approx n^{\downarrow}[P \mid Q]
\end{aligned}$$

Example: Encoding Objective Moves II

Deriving objective exit:

$$
\begin{aligned}
n^{\uparrow}[P] &\triangleq n[P] \mid \textit{allow}\ \mathrm{out} \\
\textit{mv out}\ n.P &\triangleq \textit{new}(k)k[\textit{out}\ n.\mathrm{out}[\textit{out}\ k.P]]
\end{aligned}
$$

We get:

$$
n^{\uparrow}[\textit{mv out}\ n.P \mid Q] \ \rightarrow^{*}\approx\ P \mid n^{\uparrow}[Q]
$$

Ambient allowing both objective entry and exit:

$$
n^{\updownarrow}[P] \triangleq n[P \mid \textit{allow}\ \mathrm{in}] \mid \textit{allow}\ \mathrm{out}
$$

Example: Turing Machines

Idea: tape looks like $\mathrm{end}^{\updownarrow}[\mathrm{ff}[] \mid \mathrm{sq}^{\updownarrow}[\mathrm{ff}[] \mid \mathrm{sq}^{\updownarrow}[\mathrm{ff}[] \mid \mathrm{sq}^{\updownarrow}[\cdots]]]]$.

$$
\begin{array}{rcl}
\mathit{head} & \triangleq & \mathrm{head}^{\updownarrow}[\mathit{repeat\ open}\ S_1.\mathit{mv\ out}\ \mathrm{head}. \\
 & & \quad \mathit{if}\ \mathrm{tt}\ (\mathrm{ff}[] \mid \mathit{mv\ in}\ \mathrm{head}.\mathit{in}\ \mathrm{sq}.S_2[]), \\
 & & \quad \mathit{if}\ \mathrm{ff}\ (\mathrm{tt}[] \mid \mathit{mv\ in}\ \mathrm{head}.\mathit{out}\ \mathrm{sq}.S_3[]) \mid \\
 & & \ldots \mid \\
 & & S_1[]] \\
\mathit{stretchRht} & \longrightarrow^{*} & \mathrm{sq}^{\updownarrow}[\mathrm{ff}[] \mid \mathit{stretchRht}] \\
\mathit{machine} & \triangleq & \mathrm{end}^{\updownarrow}[\mathrm{ff}[] \mid \mathit{head} \mid \mathit{stretchRht}]
\end{array}
$$

Extended Example:

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:

$$
\begin{array}{l}
\textit{node}\ a\ [\textit{channel}\ c\ \mid \\
\qquad \textit{thread}[\textit{output}\ c(b)]\ \mid \\
\qquad \textit{thread}[\textit{input}\ c(x);\ \textit{go}\ x]]\ \mid \\
\textit{node}\ b\ []
\end{array}
$$

Heteregeneous models like this underly several distributed programming systems, and several distributed forms of the π-calculus.

An Encoding $[\![-]\!]$ in the Ambient Calculus

Ambients model nodes, channels, and threads. For example:

$$\begin{aligned}
&a[[\![\textit{channel}\ c]\!]_a \mid \\
&\quad [\![\textit{thread}[\textit{output}\ c(b)]]\!]_a \mid \\
&\quad [\![\textit{thread}[\textit{input}\ c(x); \textit{go}\ x]]\!]_a] \mid \\
&b[]
\end{aligned}$$

A channel consists of a buffer ambient c^b that opens up any packets named c^p sent into it:

$$[\![\textit{channel}\ c]\!]_a = c^b[\textit{repeat open}\ c^p.\textit{stop}]$$

A thread is an anonymous ambient, with a fresh name.

An output is a packet that exits its thread, and enters a channel buffer:

$$[\![\textit{thread}[\textit{output}\ c(b)]]\!]_a = \textit{new}(t)t[\textit{go}(\textit{out}\ t.\textit{in}\ c^b).c^p[\langle b, b^p\rangle]]$$

In the untyped calculus, *go* $M.n[P]$ is short for:

$$\textit{go}\ M.n[P] \triangleq \textit{new}(k)k[M.n[\textit{out}\ k.P]]$$

An input is a packet that exits its thread, enters the buffer, gets opened, inputs a message, then returns to its thread. A move to x executes capabilities to exit the current node, then enter the destination node x.

$$
\begin{aligned}
&[\![\mathit{thread}[\mathit{input}\ c(x); \mathit{go}\ x]]\!]_a = \\
&\quad \mathit{new}(t)t[\mathit{new}(s)(\mathit{go}(\mathit{out}\ t.\mathit{in}\ c^b).c^p[(x, x^p). \\
&\qquad\qquad\qquad\qquad \mathit{go}(\mathit{out}\ c^b.\mathit{in}\ t).s[\mathit{open}\ s.\mathit{out}\ a.\mathit{in}\ x.\mathit{stop}]] \mid \\
&\qquad\qquad\qquad \mathit{open}\ s.s[])]
\end{aligned}
$$

The name s is for synchronisation ambients $s[]$, used to delay the move until the input has completed.

A fragment of a distributed programming language:

Net $::=$	network
node n $[$*Cro*$]$	node
Net $\mid$ *Net*	composition of networks
Cro $::=$	crowd of channels and threads
channel c	channel
thread$[$*Th*$]$	thread
Cro $\mid$ *Cro*	composition of crowds
Th $::=$	thread
go n; *Th*	migration
output $c(n_1, \ldots, n_k)$	output to a channel
input $c(x_1, \ldots, x_k)$; *Th*	input from a channel
$\cdots$	imperative features (omitted)

Summary of the Untyped Calculus

The core calculus (without I/O) is Turing complete. The full calculus (with I/O) can naturally model the π-calculus.

It offers a simple, abstract description of classical distributed languages, where ambients model both the unit of mobility (threads) and security perimeters (network nodes).

This description of security and mobility is more direct and explicit than possible in most other process calculi.

Ambient Types I:

Exchange Types

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 *new*$(n)n.P$ 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 *stop* exchanges messages of any type, since it exchanges none.

stop : T for all T.

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

If $\mathrm{P} : \mathrm{T}$ and $\mathrm{Q} : \mathrm{T}$ then $\mathrm{P} \mid \mathrm{Q} : \mathrm{T}$.
If $\mathrm{P} : \mathrm{T}$ then *repeat* $\mathrm{P} : \mathrm{T}$.

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
$\quad$ *Amb*$[T]$	ambient name allowing T exchanges
$\quad$ *Cap*$[T]$	capability unleashing T exchanges
$S, T ::=$	exchange types
$\quad$ *Shh*	no exchange
$\quad W_1 \times \cdots \times W_k$	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 $E = x_1{:}W_1, \ldots, x_k{:}W_k$.

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

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

Typing the Packet Example

Packet from A to B:

If $A : \mathit{Amb}[\mathit{Shh}]$, B, $\mathit{msg} : \mathit{Amb}[W]$, and $M, P : W$ then

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

Example: The Distributed Language

Each name has a type Ty, either Node or $\mathit{Ch}[\mathit{Ty}_1, \ldots, \mathit{Ty}_k]$.

Two ambient names represent each source name; e.g., each channel name is represented by a buffer name and a packet name.

We translate these to ambient types so that $[\![\mathit{Node}]\!] = \mathit{Amb}[\mathit{Shh}]$ and $[\![\mathit{Ch}[\mathit{Ty}_1, \ldots, \mathit{Ty}_k]]\!] = \mathit{Amb}[[\![\mathit{Ty}_1]\!] \times [\![\mathit{Ty}_1]\!] \times \cdots \times [\![\mathit{Ty}_k]\!] \times [\![\mathit{Ty}_k]\!]]$.

We can prove that if a program in the distributed language is well-typed, so is its translation to the ambient calculus.

Example using Exchange Types

Assuming $c{:}\mathit{Ch}[\mathit{Node}]$, the translation of $\mathit{thread}[\mathit{input}\ c(x); \mathit{go}\ x]$,

$$\begin{aligned}
&\mathit{new}(t)t[\mathit{new}(s)(\mathit{go}(\mathit{out}\ t.\mathit{in}\ c^b).c^p[(x, x^p).\\
&\qquad\qquad \mathit{go}(\mathit{out}\ c^b.\mathit{in}\ t).s[\mathit{open}\ s.\mathit{out}\ a.\mathit{in}\ x.\mathit{stop}]] \mid\\
&\qquad \mathit{open}\ s.s[])]
\end{aligned}$$

has type Shh assuming that:

$$\begin{array}{ll}
a : \mathit{Amb}[\mathit{Shh}], & t : \mathit{Amb}[\mathit{Shh}],\\
c^b, c^p : \mathit{Amb}[[\![\mathit{Node}]\!], [\![\mathit{Node}]\!]], & s : \mathit{Amb}[\mathit{Shh}]
\end{array}$$

Ambient Types II: Mobility and Locking Annotations

Regulating Mobility and Persistence

We decorate ambient types with annotations

$$Amb^{Y}[^{Z}T]$$

The locking annotation ^{Y} is either **locked** ($^{\bullet}$) or **unlocked** ($^{\circ}$).

The mobility annotation ^{Z} is either **mobile** ($^{\curvearrowright}$) or **immobile** ($^{\underline{\vee}}$).

Opening a locked ambient or moving an immobile ambient once its running is an execution error. Our type system prevents such errors.

Modifying the Type System

Let an **effect** of a process be a pair ${}^{Z}T$, where T is the type of exchanged messages, and ${}^{Z} = {}^{\veebar}$ only if no *in* or *out* capabilities are exercised.

Types and judgments acquire the form:

Message type $W ::= \mathit{Amb}^{Y}[F] \mid \mathit{Cap}[F]$

Exchange type $T ::= \mathit{Shh} \mid (W_1 \times \cdots \times W_k)$

Good expression $E \vdash M : W$

Good process $E \vdash P : F$

As before, any state reachable from a good process is a good process.

If n : *Amb*$^{Y}[F]$ then *in* n : *Cap*$[\curvearrowright T]$

If n : *Amb*$^{Y}[F]$ then *out* n : *Cap*$[\curvearrowright T]$

If n : *Amb*$^{\circ}[F]$ then *open* n : *Cap*$[F]$

If $M : W$ then $\langle M \rangle : {}^{Z}W$

If $P : {}^{Z}W$ then $(x{:}W).P : {}^{Z}W$

If M : *Amb*$^{Y}[F]$ and $P : F$ then $M[P] : F'$

If M : *Cap*$[F]$ and $P : F$ then $M.P : F$

If M : *Cap*$[F]$ and $N[P] : F'$ then *go* $M.N[P] : F'$

If $P : F$ then *new*$(n{:}W)P : F$

If $P : F$ and $Q : F$ then $P \mid Q : F$

If $P : F$ then *repeat* $P : F$

stop : F

Examples of Type Errors

You cannot open a locked ambient:

$$\mathit{new}(n{:}\mathit{Amb}^{\bullet}[F])(n[] \mid \langle n \rangle \mid (x{:}\mathit{Amb}^{\bullet}[F]).\mathit{open}\ x)$$

You cannot move an immobile ambient once its running:

$$(x{:}\mathit{Amb}^{Y}[\veebar T]).x[\mathit{out}\ m]$$

Example: Encoding Distribution, Again

Assuming $c{:}\mathit{Ch}[\mathit{Node}]$, the translation of $\mathit{thread}[\mathit{input}\ c(x); \mathit{go}\ x]$,

$$\begin{aligned}&\mathit{new}(t)t[\mathit{new}(s)(\mathit{go}(\mathit{out}\ t.\mathit{in}\ c^{b}).c^{p}[(x, x^{p}).\\ &\qquad\qquad \mathit{go}(\mathit{out}\ c^{b}.\mathit{in}\ t).s[\mathit{open}\ s.\mathit{out}\ a.\mathit{in}\ x.\mathit{stop}]] \mid \\ &\qquad \mathit{open}\ s.s[])]\end{aligned}$$

has effect $\veebar \mathit{Shh}$ assuming that:

$$\begin{array}{ll} a : \mathit{Amb}^{\bullet}[\veebar \mathit{Shh}], & t : \mathit{Amb}^{\bullet}[\curvearrowright \mathit{Shh}], \\ c^{b} : \mathit{Amb}^{\bullet}[\veebar [\![\mathit{Node}]\!]^{b} \times [\![\mathit{Node}]\!]^{p}], & s : \mathit{Amb}^{\circ}[\curvearrowright \mathit{Shh}], \\ c^{p} : \mathit{Amb}^{\circ}[\veebar [\![\mathit{Node}]\!]^{b} \times [\![\mathit{Node}]\!]^{p}] & \end{array}$$

Ambient Types III:

Ambient Groups

Motivating Ambient Groups

We may wish to express that an ambient n can enter the ambient m.

To formalise this, we introduce type-level groups of names G, H, as we did for the π-calculus, and express the property as:

> The name n belongs to group G; the name m belongs to group H. Any ambient of group G can enter any ambient of group H.

Generalizing Locking and Immobility Annotations

We decorate an ambient type with its group G, the set **G** of groups it may cross once its running, the set **H** of groups it may open, and the type T of exchanges within it:

$$G[\curvearrowright \mathbf{G}, {}^{\circ}\mathbf{H}, T]$$

Moreover, a new operation, *new*$(G)P$, creates a new group G. Within P, new names of group G can be created. In a well-typed situation, scoping rules dictate that such names may only be handled within P.

Adding Groups to the Type System

Types and judgments acquire the form:

Effect $F ::= \curvearrowright\mathbf{G}, {}^{\circ}\mathbf{G}, T$ where $\mathbf{G} ::= \{G_1, \ldots, G_n\}$

Message type $W ::= G[F] \mid \mathit{Cap}[F]$

Exchange type $T ::= \mathit{Shh} \mid (W_1 \times \cdots \times W_k)$

Good expression $E \vdash M : W$

Good process $E \vdash P : F$

As before, any state reachable from a good process is a good process.

The effect of a good process is an upper bound on the ambients it may cross or open, and the messages it may exchange.

If $n : G[F]$ and $G \in \mathbf{G}$ then *in* n : *Cap*$[{\curvearrowright}\mathbf{G}, {\circ}\mathbf{H}, T]$

If $n : G[F]$ and $G \in \mathbf{G}$ then *out* n : *Cap*$[{\curvearrowright}\mathbf{G}, {\circ}\mathbf{H}, T]$

If $n : G[{\curvearrowright}\mathbf{G}, {\circ}\mathbf{H}, T]$ and $G \in \mathbf{H}$ then *open* n : *Cap*$[{\curvearrowright}\mathbf{G}, {\circ}\mathbf{H}, T]$

If $M : W$ then $\langle M \rangle : {\curvearrowright}\mathbf{G}, {\circ}\mathbf{H}, W$

If $P : {\curvearrowright}\mathbf{G}, {\circ}\mathbf{H}, W$ then $(x{:}W).P : {\curvearrowright}\mathbf{G}, {\circ}\mathbf{H}, W$

If M : *Amb*$[F]$ and $P : F$ then $M[P] : F'$

If M : *Cap*$[F]$ and $P : F$ then $M.P : F$

If M : *Cap*$[F]$ and $N[P] : F'$ then *go* $M.N[P] : F'$

If $P : F$ then *new*$(n{:}W)P : F$

If $P : F$ and $Q : F$ then $P \mid Q : F$

If $P : F$ then *repeat* $P : F$

stop $: F$

Example: Encoding Distribution, with Groups

Assuming $c{:}\mathit{Ch}[\mathit{Node}]$, the translation of $\mathit{thread}[\mathit{input}\ c(x); \mathit{go}\ x]$,

$$
\begin{aligned}
&\mathit{new}(\mathit{Sync})\,\mathit{new}(t)\\
&\quad t[\mathit{new}(s)(\mathit{go}(\mathit{out}\ t.\mathit{in}\ c^{b}).c^{p}[(x, x^{p}).\\
&\qquad\qquad \mathit{go}(\mathit{out}\ c^{b}.\mathit{in}\ t).s[\mathit{open}\ s.\mathit{out}\ a.\mathit{in}\ x.\mathit{stop}]] \mid\\
&\qquad\quad \mathit{open}\ s.s[])]
\end{aligned}
$$

has effect $\curvearrowright\varnothing, {}^{\circ}\varnothing, \mathit{Shh}$ assuming that:

$$
\begin{array}{ll}
a : \mathit{Node}[\curvearrowright\varnothing, {}^{\circ}\mathit{Aux}, \mathit{Shh}], & t : \mathit{Thr}[\curvearrowright\mathit{Node}, {}^{\circ}\mathit{Sync}, \mathit{Shh}],\\
c^{b} : \mathit{Ch}[\curvearrowright\varnothing, {}^{\circ}\mathit{Pkt}, [\![\mathit{Node}]\!]^{b} \times [\![\mathit{Node}]\!]^{p}], & s : \mathit{Sync}[\curvearrowright\mathit{Node}, {}^{\circ}\mathit{Sync}, \mathit{Shh}],\\
c^{p} : \mathit{Pkt}[\curvearrowright\varnothing, {}^{\circ}\mathit{Pkt}, [\![\mathit{Node}]\!]^{b} \times [\![\mathit{Node}]\!]^{p}] &
\end{array}
$$

Conclusions,

Related Work

Related Work

Several process calculi model distribution and mobility (Boudol; Amadio and Prasad; Hennessy and Riely; Sewell; Fournet, Gonthier, and Lévy).

Zimmer has proposed algorithms for our system with mobility and locking annotations. Few other type systems regulate process mobility.

The idea of groups is related to Milner's sorts for π, to channels and binders found in flow analyses for π, and to the regions used for memory management in ML.

Some Papers Related to this Lecture

Untyped ambient calculus (Cardelli and Gordon, FoSSaCS'98)

Abstractions for mobile computation (Cardelli, ICALP'99)

Equational properties (Gordon and Cardelli, FoSSaCS'99)

Safe ambients (Levi and Sangiorgi, POPL'00)

Modal logics (Cardelli and Gordon, POPL'00)

Exchange types (Cardelli and Gordon, POPL'99)

Mobility types (Cardelli, Ghelli, and Gordon, ICALP'99)

Subtyping and algorithms for mobility types (Zimmer, FOSSACS'00)

Ambient groups (Cardelli, Ghelli, and Gordon, TCS'00)

Implementations of Ambients

Ambit applet (Cardelli)

Ambient language design (Cardelli and Torgersen)

Ambients in Jocaml (Fournet, Lévy, Schmitt)

Reactive ambients (Sangiorgi and Boussinot)

Ambients in Haskell (Peyton Jones)

Summary

A goal of 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.

Type systems like these could be checked by a bytecode verifier to better constrain mobile code.

Papers and software available from:

http://www.luca.demon.co.uk/Ambit/Ambit.html

http://research.microsoft.com/users/adg/Publications

http://go.163.com/ mobileambient