(Calculi and Types for) Global Computing

FOUNDATIONS OF SECURITY
Oregon (23.6.03)

Vladimiro Sassone

University of Sussex,

Foundations of Global Computing

Global Computing:

computation over a global network of mobile, bounded resources shared among mobile entities which move between highly dynamic, largely unknown, untrusted networks.

Difficulties:

Extreme dynamic reconfigurability; lack of coordination and trust; limited capabilities; partial knowledge . . .

Issues:

Protection and management of resources; privacy and confidentiality of data; . . .

Mobility & Agent Migration
Globality & Variability
Resource Sharing & Control
Safety & Protection

Overview of the Lectures

Aim:

Illustrate calculi which formalise these ideas and pave the ground for the development of foundations solid enough to underpin future applications.

Approach:

Present tools – essentially type systems – to guarantee safety, security and in particular resource access control.

What:

- *Name Mobility*
- *Types for Safety & Control*
- *Asynchrony & Distribution*
- *Ambient Mobility*
- *Resource Control*

The π Calculus:

- Basic calculus
- Variations
- Bisimulation
- Properties

Overview of the Lectures

Aim:

Illustrate calculi which formalise these ideas and pave the ground for the development of foundations solid enough to underpin future applications.

Approach:

Present tools – essentially type systems – to guarantee safety, security and in particular resource access control.

What:

- *Name Mobility*
- *Types for Safety & Control*
- *Asynchrony & Distribution*
- *Ambient Mobility*
- *Resource Control*

Typed π Calculi:

- Sorts
- Simply Typed π
- I/O Types
- Secrecy Types
- Group Types

Overview of the Lectures

Aim:

Illustrate calculi which formalise these ideas and pave the ground for the development of foundations solid enough to underpin future applications.

Approach:

Present tools – essentially type systems – to guarantee safety, security and in particular resource access control.

What:

- *Name Mobility*
- *Types for Safety & Control*
- *Asynchrony & Distribution*
- *Ambient Mobility*
- *Resource Control*

Asynchronous π Calculi:

- π_A
- Asynchronous $\approx$
- π_L
- Dπ
- Join calculus

Overview of the Lectures

Aim:

Illustrate calculi which formalise these ideas and pave the ground for the development of foundations solid enough to underpin future applications.

Approach:

Present tools – essentially type systems – to guarantee safety, security and in particular resource access control.

What:

- *Name Mobility*
- *Types for Safety & Control*
- *Asynchrony & Distribution*
- *Ambient Mobility*
- *Resource Control*

Ambient Calculi:

- Mobile Ambients
- Ambient Types
- Boxed Ambients
- Types for Access Control

Overview of the Lectures

Aim:

Illustrate calculi which formalise these ideas and pave the ground for the development of foundations solid enough to underpin future applications.

Approach:

Present tools – essentially type systems – to guarantee safety, security and in particular resource access control.

What:

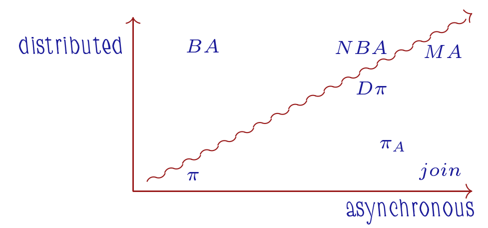
- *Name Mobility*
- *Types for Safety & Control*
- *Asynchrony & Distribution*
- *Ambient Mobility*
- *Resource Control*

Resource Control

- Interferences
- Secrecy in Ambients
- Sizes & Capacities

Roadmap

- Another way to look at the plan:
  Start from π and move towards asynchrony and distribution.



- What will we ignore?
  An enormous amount! However, what we'll do will be sufficient to be able to follow the literature and the current developments.

Global Computing

— Lecture I —

Name Mobility

———

Roadmap for Lecture I

- The π calculus' basic mechanisms
- Examples
- Variations
  - Polyadic π
  - Summation
  - Match & mismatch
  - Recursion
  - Higher order
- Barbs & bisimulation
- LTS & bisimulation

The π calculus

The π calculus is:

- A formal model to describe and analyse systems of interacting (communicating) processes, with dynamic (re)configuration;
- Terms are processes, that is computational activities running in parallel with each other and possibly containing several independent subprocesses.
- Currently the canonical model of concurrent computation, as the λ-calculus for functional computation:
  - computation in the λ-calculus is the result of function application; computation is the process of applying functions to arguments and yielding results:
  - computation in the π-calculus arises from process interaction/reaction (based on communication).

Names in the π calculus

Naming is a pervasive notion in π:

- It is a prerequisite to communication and, therefore, interaction and computation.
- It presupposes independence: the namer and the named are independent (concurrent) entities.

Names 'name' *communication channels*

not agents

The Syntax

- An infinite set of names: $\mathcal{N} = \{x, y, z, \ldots\}$.
- Action prefixes:

$$\pi ::= x(y) \ | \ \overline{x}\langle y \rangle \ | \ \tau$$

- Processes:

$$P ::= \sum_{i \in I} \pi_i.P_i \ | \ P \,|\, P \,|\, (\boldsymbol{\nu} a)P \ | \ !P$$

where
  - I finite;
  - input $x(y).P$ and new $(\nu y).P$ bind y in P. Terms are taken up to α-conversion. That is: for z not free in P

$$x(y).P \equiv x(z).P\{z/y\} \qquad (\nu y)P \equiv (\nu z)P\{z/y\};$$

  - commonly used shorthands: $\mathbf{0}$ for the empty sum $\sum_{i \in \emptyset}$; $P + P$ for binary sums; $\overline{x}$ or x when the message is irrelevant

Reductions

$$(\nu z)(\bar{x}\langle y \rangle + z(w).\bar{w}\langle y \rangle \,|\, x(u).\bar{u}\langle v \rangle \,|\, \bar{x}\langle z \rangle)$$

$$\swarrow \qquad \searrow$$

$$(\nu z)(0 \,|\, \bar{y}\langle v \rangle \,|\, \bar{x}\langle z \rangle) \qquad (\nu z)(\bar{x}\langle y \rangle + z(w).\bar{w}\langle y \rangle \,|\, \bar{z}\langle v \rangle \,|\, 0)$$

$$\downarrow$$

$$(\nu z)(\bar{v}\langle y \rangle \,|\, 0 \,|\, 0)$$

Contexts and Congruences

Process Contexts:

$$\mathscr{C} ::= [\,.\,] \ | \ \pi.\mathscr{C} + P \ | \ (\boldsymbol{\nu} a)\mathscr{C} \ | \ \mathscr{C} \,|\, P \ | \ !\mathscr{C}$$

Congruences:

A relation $\bowtie$ is a congruence if it is preserved by all contexts, that is $P \bowtie Q$ implies:

$$\begin{array}{rcl} \pi.P + R \bowtie \pi.Q + R & & (\boldsymbol{\nu} a)P \bowtie (\boldsymbol{\nu} a)Q \\ P \,|\, R \bowtie Q \,|\, R & & R \,|\, P \bowtie R \,|\, Q \\ & !P \bowtie !Q & \end{array}$$

Structural Congruence

$P \equiv Q$ if they can be transformed into each other using

- rearrangement of terms in summations;
- commutative monoidal laws for $|$ (with 0 as unit);
-

$$\begin{array}{ll} (\boldsymbol{\nu} z)(P \,|\, Q) \equiv (\boldsymbol{\nu} z)P \,|\, Q, & \text{if } z \notin \mathsf{fn}(Q); \\ (\boldsymbol{\nu} z)\mathbf{0} \equiv \mathbf{0}; & \\ (\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P \equiv (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P. & \end{array}$$

- $!P \equiv P \,|\, !P$

Standard Form

A process

$$(\nu\vec{a})(M_1 \mid \cdots \mid M_m \mid !Q_1 \mid \cdots \mid !Q_n)$$

is in standard form if

1. each M_i is a sum and
2. each Q_i is itself in standard form.

Thm: Every processes is structurally congruent to a process in standard form.

proof: Easy, by structural induction.

Reaction Rules

Tau: $(\tau.P + M) \longrightarrow P$

React: $(M + x(y).P) \mid (\bar{x}\langle z\rangle.Q + M') \longrightarrow \{z/y\}P \mid Q$

Par: $\dfrac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}$ Res: $\dfrac{P \longrightarrow P'}{(\boldsymbol{\nu} a)P \longrightarrow (\boldsymbol{\nu} a)P'}$

Struct: $\dfrac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q}$

Evaluation contexts: $\mathscr{E} ::= [\,.\,] \mid \mathscr{E} + P \mid (\boldsymbol{\nu} a)\mathscr{E} \mid \mathscr{E} \mid P$

An example

Although it may appear not obvious, the term

$$P = x(z).\bar{y}\langle z\rangle \mid !(\nu y)\bar{x}\langle y\rangle.Q$$

has a redex. Let us use $\equiv$ to uncover it.

$$\begin{aligned} P &\equiv x(z).\bar{y}\langle z\rangle \mid (\nu y)(\bar{x}\langle y\rangle.Q) \mid !(\nu y)\bar{x}\langle y\rangle.Q \\ &\equiv x(z).\bar{y}\langle z\rangle \mid (\nu a)(\bar{x}\langle a\rangle.Q) \mid !(\nu y)\bar{x}\langle y\rangle.Q \\ &\equiv (\nu a)(x(z).\bar{y}\langle z\rangle \mid \bar{x}\langle a\rangle.Q) \mid !(\nu y)\bar{x}\langle y\rangle.Q \\ &\to (\nu a)(\bar{y}\langle a\rangle \mid Q) \mid !(\nu y)\bar{x}\langle y\rangle.Q \end{aligned}$$

Scope extrusion

The following rule enlarges the scope of a:

$$(\boldsymbol{\nu} a)(P \mid Q) \equiv P \mid (\boldsymbol{\nu} a)Q \quad \text{if } a \notin \mathsf{fn}(P)$$

- left-to-right reading: no surprise
- right-to-left reading: enables export of private names.

$$c(x).P \mid (\boldsymbol{\nu} a)\bar{c}\langle a\rangle.Q$$

In such form, the processes may not communicate.
However:

$$\begin{aligned} c(x).P \mid (\boldsymbol{\nu} a)\bar{c}\langle a\rangle.Q &\equiv (\boldsymbol{\nu} a)(c(x).P \mid \bar{c}\langle a\rangle.Q) \\ &\longrightarrow (\boldsymbol{\nu} a)(P\{a/x\} \mid Q) \end{aligned}$$

the name a, private to Q, has been communicated to P.

As in the previous slide, it may be necessary to perform an α-conversion on a.

Scope extrusion, continued

The reduction

$$c(x).P \mid (\nu a)\overline{c}\langle a\rangle.Q \longrightarrow (\nu a)(P\{a/x\} \mid Q)$$

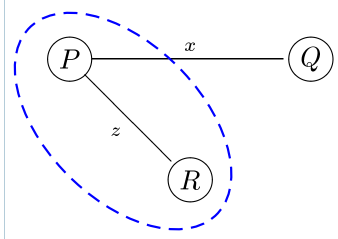
establishes a new communication link between P and Q, viz. a.

The new link is now private to P and Q, and will remain so until one of them communicates it to third parties.

Scope extrusion and channel-based communication provide an elementar, yet powerful mechanism for:

- Name mobility: dynamically changing the topological structure of a system of processes, by creating new communication links.
- Secrecy: establishing private, hence secret channels.

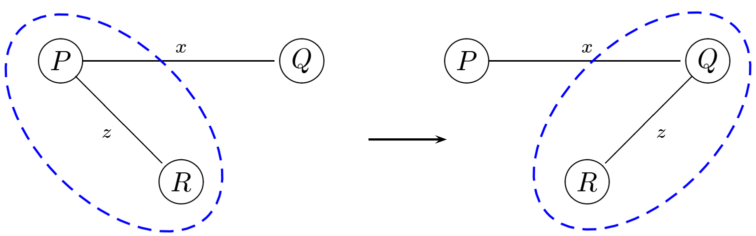
The essence of name mobility



$(\nu z)(P \mid R) \mid Q$

Suppose $P = \bar{x}\langle z\rangle.P'$ (x not in P') and $Q = x(y).Q'$

The essence of name mobility



$(\nu z)(P \mid R) \mid Q$ $\qquad$ $P' \mid (\nu z)(R \mid Q')$

New scope of z

Name mobility and secret channels

A simple *security protocol*:

- Alice and Bob want to exchange secret M, Server mediates. A and B share private channels c_{AS} and c_{BS} with S
- A sends B a secret channel c_{AB} via S.

  Msg 1: $A \to S$ $\quad c_{AB}$ on c_{AS}

  Msg 2: $S \to B$ $\quad c_{AB}$ on c_{BS}
- Now A and B communicate via c_{AB}.

  Msg 3: $A \to B$ $\quad M$ on c_{AB}

π-calculus *specification* of the protocol

$$A \triangleq (\nu c_{AB})\overline{c_{AS}}\langle c_{AB}\rangle.\overline{c_{AB}}\langle M\rangle$$

$$S \triangleq c_{AS}(x).\overline{c_{BS}}\langle x\rangle$$

$$B \triangleq c_{BS}(x).x(y).P\{y\}$$

$$\mathsf{SYS} \triangleq (\nu c_{AS})(\nu c_{BS})(A \mid B \mid S)$$

A run of the protocol

$$\mathsf{SYS} = c_{BS}(x).x(y).P\{y\} \mid c_{AS}(x).\overline{c_{BS}}\langle x\rangle \mid (\boldsymbol{\nu} c_{AB})(\overline{c_{AS}}\langle c_{AB}\rangle.\overline{c_{AB}}\langle M\rangle)$$

$$\equiv c_{BS}(x).x(y).P\{y\} \mid (\boldsymbol{\nu} c_{AB})(c_{AS}(x).\overline{c_{BS}}\langle x\rangle \mid \overline{c_{AS}}\langle c_{AB}\rangle.\overline{c_{AB}}\langle M\rangle)$$

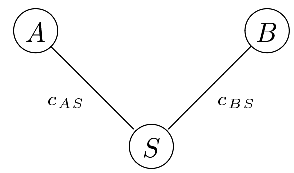
$$\longrightarrow c_{BS}(x).x(y).P\{y\} \mid (\boldsymbol{\nu} c_{AB})\overline{c_{BS}}\langle c_{AB}\rangle \mid \overline{c_{AB}}\langle M\rangle)$$

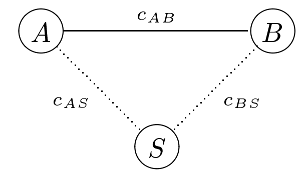
$$\equiv (\boldsymbol{\nu} c_{AB})(c_{BS}(x).x(y).P\{y\} \mid \overline{c_{BS}}\langle c_{AB}\rangle \mid \overline{c_{AB}}\langle M\rangle)$$

$$\longrightarrow (\boldsymbol{\nu} c_{AB})(c_{AB}(y).P\{y\} \mid \overline{c_{AB}}\langle M\rangle)$$

The run, conceptually

$$A = (\boldsymbol{\nu} c_{AB})\overline{c_{AS}}\langle c_{AB}\rangle.\overline{c_{AB}}\langle M\rangle \qquad\qquad A = \overline{c_{AB}}\langle M\rangle$$
$$S = c_{AS}(x).\overline{c_{BS}}\langle x\rangle \qquad \Longrightarrow \qquad S = \mathbf{0}$$
$$B = c_{BS}(x).x(y).P\{y\} \qquad\qquad B = c_{AB}(y).P\{y\}$$





We will revisit this example with cryptographic primitives

But of course ...

- This is an ideal picture, as private channels are an abstraction

$$(\boldsymbol{\nu} n)(\overline{n}\langle a\rangle Q \mid n(x).P)$$

- What if the two processes are located at remote sites?
- In practice, one needs cryptography

$$(\boldsymbol{\nu} n)(\overline{p}\langle\{a\}_n\rangle Q \mid p(y).\text{decrypt } y \text{ as } \{x\}_n \text{ in } P)$$

- that's the idea behind the spi calculus

Polyadic π

The idea:

$$x(y_1,\ldots,y_n).P \mid \bar{x}\langle z_1,\ldots,z_n\rangle.Q \rightarrow \{\vec{z}/\vec{y}\}P \mid Q$$

Is it a more expressive paradigm? Or can it be encoded?

Encoding? $\llcorner\bar{x}\langle z_1,z_2\rangle.P\lrcorner = \bar{x}\langle z_1\rangle.\bar{x}\langle z_2\rangle.\llcorner P\lrcorner.$

Right idea:

$$\ulcorner\bar{x}\langle z_1,z_2\rangle.P\urcorner = (\nu z)\bar{x}\langle z\rangle.\bar{z}\langle z_1\rangle.\bar{z}\langle z_2\rangle.\ulcorner P\urcorner$$
$$\ulcorner x(z_1,z_2).P\urcorner = x(y).y(y_1).y(y_2).\ulcorner P\urcorner$$

Thm. The translation is 'sound' (i.e., if $\ulcorner P\urcorner$ behaves like $\ulcorner Q\urcorner$, then P behaves like Q). Is it 'correct' (and therefore 'fully abstract')? (Left for exercise).

Example: memory cells

$$\mathsf{cell}(n) \triangleq (\boldsymbol{\nu} s)(\overline{s}\langle n\rangle \mid$$
$$!\mathsf{get}(y).s(x).(\overline{s}\langle x\rangle \mid \overline{y}\langle x\rangle) \mid$$
$$!\mathsf{put}(y,v).s(x)(\overline{s}\langle v\rangle \mid \overline{y}\langle\rangle))$$

- a private channel s 'stores' the value n (it represents the state of the memory cell),
- two handlers serving the 'get' and 'put' requests.
  - both implemented as replicated processes, to serve multiple requests.
  - each request served by first spawning a fresh copy of the handler by means of the congruence $!P \equiv P \mid !P$.

Cell: put and get

$$\mathsf{get}(y).s(x).(\overline{s}\langle x\rangle \mid \overline{y}\langle x\rangle)$$

- receive on get the name y of a channel where to send back the result
- upon receiving the channel name, consume the current cell value, and then reinstate it while copying it to the channel y;

$$\mathsf{put}(y,v).s(x)(\overline{s}\langle v\rangle \mid \overline{y}\langle\rangle)$$

- similar situation, with a further subtlety: also expect an "ack" channel (y) from the user, and use it to signal the completion of the protocol.

Cell and User

A sample user of the cell:

$$\mathsf{client}(v) = (\boldsymbol{\nu}\mathsf{ack})(\boldsymbol{\nu}\mathsf{ret})$$
$$\overline{\mathsf{put}}\langle \mathsf{ack}, v\rangle.\mathsf{ack}().\overline{\mathsf{get}}\langle \mathsf{ret}\rangle.\mathsf{ret}(x).\overline{\mathsf{print}}\langle x\rangle$$

- declare private return and ack channels
- first write a new value, wait for ack, and then read the cell contents to print the returned value.

Let us look at the system:

$$\mathsf{cell}(0) \mid \mathsf{client}(v) \quad \equiv \quad (\boldsymbol{\nu} s)(\boldsymbol{\nu}\mathsf{ack})(\boldsymbol{\nu}\mathsf{ret})(\ldots)_{cell} \mid (\ldots)_{\mathsf{user}}$$

Cell & user: reduction

$$(\overline{s}\langle 0\rangle \mid (\mathsf{put}(y,v).s(x).\ldots \mid \ldots))_{cell} \mid (\overline{\mathsf{put}}\langle \mathsf{ack}, 1\rangle.\ldots)_{user}$$
$$\longrightarrow (\overline{s}\langle 0\rangle \mid s(x).(\overline{s}\langle 1\rangle \mid \overline{\mathsf{ack}}\langle\rangle) \mid \ldots))_{cell} \mid (\mathsf{ack}().\ldots)_{user}$$
$$\longrightarrow (\overline{s}\langle 1\rangle \mid \overline{\mathsf{ack}}\langle\rangle \mid \ldots)_{cell} \mid (\mathsf{ack}().\ldots)_{user}$$
$$\longrightarrow (\overline{s}\langle 1\rangle \mid (\mathsf{get}(y).s(x).\ldots))_{cell} \mid (\overline{\mathsf{get}}\langle \mathsf{ret}\rangle.\ldots)_{user}$$
$$\longrightarrow (\overline{s}\langle 1\rangle \mid (s(x).(\overline{s}\langle x\rangle \mid \overline{\mathsf{ret}}\langle x\rangle)\ldots))_{cell} \mid \mathsf{ret}(x).\overline{\mathsf{print}}\langle x\rangle$$
$$\longrightarrow (\overline{s}\langle 1\rangle \mid \overline{\mathsf{ret}}\langle 1\rangle \ldots)_{cell} \mid \mathsf{ret}(x).\overline{\mathsf{print}}\langle x\rangle$$
$$\longrightarrow (\overline{s}\langle 1\rangle \mid \ldots)_{cell} \mid \overline{\mathsf{print}}\langle 1\rangle$$

Note: $\equiv$-steps omitted.

Summation

The original calculus has unguarded sums:

$$P ::= \ldots \mid P + P$$

This makes the theory more complex while producing little gains in expressiveness.

Input guarded sum: $\sum_{i \in I} x_i(y).P$

Rejects things like $(x.P + \bar{y}.Q) \mid (\bar{x}.P' + y.Q')$.

No sums at all:
Still, purely internal choice $\tau.P + \tau.Q$ can be defined:

$$(\nu a)(\bar{a} \mid a.P \mid a.Q)$$

Matching and Mismatching

The original calculus has

$$\pi ::= \ldots \mid [x = y]\pi \mid [x \neq y]\pi$$

$$[x = x]P \equiv P \qquad [x \neq y]P \equiv \mathbf{0}$$

Useful in programming, it has an impact on the theory, and somehow complicates it. A general encoding is impossible, but in some cases its effect can be recovered to a certain extent:

$$\llcorner a(x).(\textstyle\sum_i [x = k_i]P_i) \lrcorner = a(x).(\bar{x}\langle\rangle \mid \textstyle\sum_i k_i().\llcorner P_i \lrcorner)$$

Though this works only provided nobody 'interferes' with k_i.

Recursive Definitions

It is useful to be able to write

$$A(\vec{x}) \stackrel{def}{=} Q_A, \quad \text{where } Q_A = \cdots A\langle\vec{w}\rangle \cdots A\langle\vec{w}\rangle \cdots$$

It can be obtained using replication as follows

1. Choose a_A to stand for A;
2. $\widehat{R}$ = replace $A\langle\vec{w}\rangle$ with $\bar{a}_A\langle\vec{w}\rangle$ in R;
3. $\widehat{\widehat{P}} = (\nu a_A)(\widehat{P} \mid !a_A(\vec{x}).\widehat{Q_A})$.

On the other hand, replication can be defined from recursive defs.

$$A \stackrel{def}{=} P \mid A.$$

Higher Order π

A most natural suggestion for process mobility:

$$\pi ::= \cdots \mid x(X) \mid \bar{x}\langle P\rangle$$
$$P ::= \cdots \mid X$$

$$x(X).P \mid \bar{x}\langle R\rangle.Q \to \{R/X\}P \mid Q$$

Thm. Encoding 'fully abstract'. Assume a name an unused name x associated to each X.

$$[\bar{a}\langle P\rangle.Q] = (\nu p)\bar{a}\langle p\rangle.([Q] \mid \mathbf{!}p.[P]), \quad p \text{ fresh}$$

$$[a(X).P] = a(x).[P])$$

$$[X] = \bar{x}$$

Other variants

- Asynchronous π: Disallow continuation on sending $\overline{a}\langle x\rangle.P$.
- Local π: Disallow inputs on x in the body of $a(x).P$.
- Private π: Disallow output of free names: Processes can only pass names their own private names.
- Distributed π: Several interesting calculi based on π_A: Dpi, Join, Blue, Seal, Nomadic Pict,...
- Spi, applied pi,...

Observations and Bisimulation

Observations: $P{\downarrow_a}$ if P can engage in action involving a

$$P{\downarrow_{\overline{a}}} \triangleq P \equiv (\nu\vec{x})(\overline{a}\langle z\rangle.P' + \cdots) \qquad a \notin \vec{x}$$

$$P{\downarrow_a} \triangleq P \equiv (\nu\vec{x})(a(z).P' + \cdots) \qquad a \notin \vec{x}$$

$$P{\Downarrow_\alpha} \triangleq P \Longrightarrow{\downarrow_\alpha} \qquad (\Longrightarrow \triangleq \longrightarrow^*)$$

Barbed Bisimulation $\dot{\approx}$: is the largest equivalence relation $\pitchfork$ s.t. for all $P \pitchfork Q$

- If $P \longrightarrow P'$ then $Q \Longrightarrow Q'$ for some such that $P' \pitchfork Q'$;
- If $P{\downarrow_x}$, then $Q{\Downarrow_x}$

Barbed bisimulation is very weak, pretty useless:

$$\overline{a}\langle u\rangle \quad \dot{\approx} \quad \overline{a}\langle v\rangle \quad \dot{\approx} \quad (\boldsymbol{\nu} u)\overline{a}\langle u\rangle$$

Consider e.g. $\pitchfork = \{(\overline{a}\langle u\rangle, \overline{a}\langle v\rangle)\}$ and ...

Observations and Contexts

Barbed equivalence is very weak, but 'contexts' are very powerful enquirers: Consider $\mathscr{C} = (\nu a)([\;] \mid a(x).x)$. Then

$$\mathscr{C}[\bar{a}\langle u\rangle] \to (\nu a)u{\downarrow_u} \qquad \mathscr{C}[\bar{a}\langle v\rangle] \to (\nu a)v{\downarrow_v} \qquad \mathscr{C}[(\nu u)\bar{a}\langle u\rangle] \to (\boldsymbol{\nu} au)u \not\downarrow$$

Therefore $\quad \mathscr{C}[\bar{a}\langle u\rangle] \quad \dot{\not\approx} \quad \mathscr{C}[\bar{a}\langle v\rangle] \quad \dot{\not\approx} \quad \mathscr{C}[(\nu u)\bar{a}\langle u\rangle]$

Barbed Congruence $\cong^c$: Is the largest congruence in $\dot{\approx}$, that is

$$P \cong^c Q \quad \text{iff } \mathscr{C}[P] \dot{\approx} \mathscr{C}[Q], \text{ for all } \mathscr{C}.$$

Context Lemma: $P \cong^c Q$ iff $P\sigma \mid R \dot{\approx} Q\sigma \mid R$, for all R and substitutions σ.

Exercise: Only non-injective substitutions are interesting here...

The problem with aliasing

Role of σ is account for aliasing occurring from rebinding of names after input.

Consider that $\quad \overline{x} \mid y \dot{\approx} \overline{x}.y + y.\overline{x}$.

But, in $\mathscr{C} = a(y).[\,.\,] \mid \overline{a}\langle x\rangle$, since x is received for y, $\overline{x} \mid x \dot{\not\approx} \overline{x}.x + x.\overline{x}$.
The effect of such contexts is captured by the context lemma $\sigma(x) = \sigma(y) = z$.

Similar problems for matching:

$$[x = y]c \dot{\approx} 0 \text{ but } [x = x]c \dot{\not\approx} 0.$$

Labelled Transition System

- Barbed bisimulation derives naturally from the reduction rules.
- Congruences are brought about by engineering, mathematical and logical considerations.
- But barbed congruence is hard to work with.
- Desiderata: Characterise it in terms of:
  1. bisimulations (easy to reason with – coinduction)
  2. labelled transition systems (describe interactions with environment explicitly, help intuition, bag of tools, ...)

π Actions

$$\alpha ::= \tau \mid \overline{x}y \mid \nu\overline{x}y \mid xy$$

- $\mathsf{n}(\alpha)$: names in α
- $\mathsf{bn}(\alpha)$: names bound in α, that is bound output.

Desiderata:

Thm: Establish a compositional $\xrightarrow{\alpha}$ such that $\xrightarrow{\tau} \equiv = \longrightarrow$

Thm: Establish a proof technique for $\cong^c$ using bisimulation on $\xrightarrow{\alpha}$

Labelled Transition System

(Out) $$\frac{}{\overline{x}\langle y\rangle.P \xrightarrow{\overline{x}y} P}$$

(Inp) $$\frac{}{x(z).P \xrightarrow{xy} P\{y/z\}}$$

(Tau) $$\frac{}{\tau.P \xrightarrow{\tau} P}$$

(Open) $$\frac{P \xrightarrow{\overline{x}z} P'}{(\boldsymbol{\nu}z)P \xrightarrow{\nu\overline{x}z} P'}\ x{\neq}z$$

(CloseL) $$\frac{P \xrightarrow{\nu\overline{x}z} P' \quad Q \xrightarrow{xz} Q'}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}z)(P' \mid Q')}\ z{\notin}\mathsf{fn}(Q)$$

(CommL) $$\frac{P \xrightarrow{\overline{x}z} P' \quad Q \xrightarrow{xz} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

(SumL) $$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

(ParL) $$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathsf{bn}(\alpha)\cap\mathsf{fn}(Q)=\emptyset$$

(Res) $$\frac{P \xrightarrow{\alpha} P'}{(\boldsymbol{\nu}z)P \xrightarrow{\alpha} (\boldsymbol{\nu}z)P'}\ z{\notin}\mathsf{n}(\alpha)$$

(Rep) $$\frac{P \mid !P \xrightarrow{\alpha} Q}{!P \xrightarrow{\alpha} Q}$$

An example

Let us show how to prove that

$$x(y).P \mid (\boldsymbol{\nu}a)\overline{x}\langle a\rangle.Q \xrightarrow{\tau} (\boldsymbol{\nu}a)(P\{a/y\} \mid Q)$$

$$\frac{x(y).P \xrightarrow{xa} P\{a/y\} \qquad \dfrac{\overline{x}\langle a\rangle.Q \xrightarrow{\overline{x}a} Q}{(\boldsymbol{\nu}a)\overline{x}\langle a\rangle.Q \xrightarrow{\nu\overline{x}a} Q}}{x(y).P \mid (\boldsymbol{\nu}a)\overline{x}\langle a\rangle.Q \xrightarrow{\tau} (\boldsymbol{\nu}a)(P\{a/y\} \mid Q)}$$

The role of Open, Close, and Comm:

$$(y).P \xleftrightarrow{x} \langle a\rangle.Q = \quad (P\{a/y\} \mid Q)$$
$$(y).P \xleftrightarrow{x} (\boldsymbol{\nu}a)\langle a\rangle.Q = \quad (\boldsymbol{\nu}a)(P\{a/y\} \mid Q)$$

Bisimulation

Bisimulation $\approx$: is the largest equivalence relation $\pitchfork$ s.t. for all $P \pitchfork Q$

$$P \xrightarrow{\alpha} P' \text{ then } Q \stackrel{\hat{\alpha}}{\Longrightarrow} \pitchfork P'$$

Note that:

in $a(x).P \approx Q$, term $P\{y/x\}$ must be matched for all y (well, almost...);

$\nu\overline{x}z$ must be matched only for an appropriately fresh z.

Again, $\approx$ is not preserved by substitutions

Full Bisimulation $P \approx^c Q$ iff $P\sigma \approx Q\sigma$ for all substitutions σ (non injective).

Thm. $\approx \subset \cong$ and $\approx^c \subset \cong^c$.

On finite-image processes, i.e., such all sets of α-derivated $\{P' \mid P \stackrel{\alpha}{\Longrightarrow} P'\}$ are finite, with matching $\approx^c = \cong^c$.

Exercise: Prove that matching is necessary, by showing that $P \not\approx Q$, but $\mathscr{C}[P] \approx \mathscr{C}[Q]$ for all contexts without matching, where

$$P = \overline{a}\langle x\rangle \mid !x \mid !\overline{x} \mid !y \mid !\overline{y} \quad \text{and} \quad Q = \overline{a}\langle y\rangle \mid !x \mid !\overline{x} \mid !y \mid !\overline{y}$$

Comparisons and Other bisimulations

$$\begin{tikzcd}
\cong^c \arrow[r, hook] & \cong \arrow[dr, hook] & \\
& & \dot{\approx} \\
\approx^c \arrow[uu, hook] \arrow[r, hook] & \approx \arrow[uu, hook] \arrow[ur, hook] &
\end{tikzcd}$$

Differences in the treatment of input actions $\xrightarrow{xy}$ give rise to different notions of bisimulation:

- late: $a(x).P \approx a(x).Q$ iff $P\{y/x\} \approx Q\{y/x\}$ for all y;
- ground: $a(x).P \approx a(x).Q$ iff $P\{y/x\} \approx Q\{y/x\}$ for a fresh y;
- open: obtained using substitutions explicitly in the bisimulation game.

Exercises

Exercises:

- Prove $\equiv \xrightarrow{\tau} = \xrightarrow{\tau} \equiv$;
- Prove $\equiv \subset \approx$;
- Prove $\equiv \subset \dot{\approx}$ and $\equiv \subset \cong^c$
- Prove $\overline{x}\langle a\rangle \not\cong^c (\boldsymbol{\nu} z)\overline{x}\langle z\rangle$.
- Prove $\tau.P \approx^c P$.
- Prove $\approx \subseteq \dot{\approx}$ and $\approx^c \subseteq \cong^c$

Summary of Lecture I

- This lecture introduced the π calculus' mechanisms and the fundamentals of its semantic theory, ie barbed congruence and full bisimilarity.
- *Further Reading:*

  A complete list would be enourmous. Luckily, two references for all can take you a long way

  - Communication and Mobile Systems: the π-calculus (Milner)
  - The π-calculus: A theory of mobile processes (Sangiorgi, Walker)
  - The mobility home page http://lamp.epfl.ch/mobility/

Global Computing

— Lecture II —

Types for Safety and Control

Roadmap of Lecture II

- Milner's sorting system
- Simply typed π calculus
- IO types and subtypes
- Typed barbed congruence
- Types for secrecy
- Group types

Why Types?

Consider the following terms of the polyadic π-calculus.

$$\overline{a}\langle b, c\rangle.P \mid a(x).Q$$
$$\overline{a}\langle \mathsf{true}\rangle.P \mid a(x).x(y).Q$$

Both terms are ill-formed, make no sense, and must therefore be ruled out. This is one of the role of types. In general, types establish invariants of computation that we use to guarantee safety in many varieties.

- Types protect from errors and therefore provide guarantees of consistent process interaction.
- Types convey logical structure: the untyped π-calculus is too weak to prove some expected properties of processes arising from implicit discipline of name usage.

  Types bring the intended structure back into light, and enhance formal reasoning on process terms: for instance, typed behavioral equivalences are more generous, and can be easier to prove, as only typed contexts need to be looked at.

Milner's sorting system

Memory cells

$$\mathsf{cell}(n) \triangleq (\boldsymbol{\nu} s)(\overline{s}\langle n\rangle \mid !\mathsf{get}(y).s(x).(\overline{s}\langle x\rangle \mid \overline{y}\langle x\rangle) \mid$$
$$!\mathsf{put}(y, v).s(x)(\overline{s}\langle v\rangle \mid \overline{y}\langle\rangle))$$

- ret used to communicate integers,
- get and ack used to communicate another channel

Sorting:

ret	:	S_i		S_i	$\mapsto$	(int)
get	:	S_g		S_g	$\mapsto$	(S_i)
ack	:	S_a		S_a	$\mapsto$	()
put	:	S_p		S_p	$\mapsto$	$(\mathsf{S}_a, \mathsf{S}_i)$

Sorting, formally

A Sorting System

- A function $\Sigma : \mathsf{S} \longrightarrow \mathsf{S}^*$ describes the tuples allowed on channels of each sort. $\Sigma(\gamma)$ is the object sort of γ.
- Object sort of $\gamma \in S$ must follow the sorting discipline $\Sigma(\gamma)$.
- P respects Σ if in each subterm $\overline{x}\langle\vec{y}\rangle.P'$ or $x(\vec{y}).P'$, if $x : \gamma$, then $\vec{y} : \Sigma(\gamma)$

Subject Reduction: If P respects Σ and $P \longrightarrow Q$, then Q respects Σ.

It follows that $P \xrightarrow{\overline{x}\vec{y}}$ implies that $\vec{y} : \Sigma(\gamma)$, for $x : \gamma$.

Therefore, these cannot happen:

$$\overline{a}\langle b, c\rangle.P \mid a(x).Q$$

$$\overline{a}\langle \mathsf{true}\rangle.P \mid a(x).x(y).Q$$

Simply Typed π calculus

S, T	$::=$	B	types of basic values
	$\mid$	$(T_1, \ldots, T_k)$	tuple type, $k \geqslant 0$
	$\mid$	$\sharp T$	link type (channel)

Channel types

- inform on the type of the value they carry

Examples

- $\sharp(\mathsf{int})$: channel carrying values of type int.
- $\sharp(\mathsf{unit})$: channel carrying $\star$, the only value of type unit.
- $\sharp(\sharp\mathsf{int})$: channel whose values are channels carrying integers.

Type system

- Initial idea: types assigned only to channels, processes are either well typed under a particular set of assumptions for their bound and free names, or they are not.
- two judgement forms:
  - $\Gamma \vdash v : T$ $\quad v$ has type T
  - $\Gamma \vdash P$ $\quad P$ is well-typed
- Γ type environment: a set of type assumptions for names and variables (equivalently, a finite map from names and variables to types)

Different approach possible, based on assigning more informative types to processes to describe various forms of process behavior.

Typing rules: Values and Messages

- Values

(Base)
$$\frac{}{\Gamma \vdash bv : \mathsf{B}}$$

(Name)
$$\frac{}{\Gamma, u : T \vdash u : T}$$

(Tuple)
$$\frac{\Gamma \vdash v_i : T_i \quad i = 1..k}{\Gamma \vdash v_1, \ldots, v_k : (T_1, \ldots, T_k)}$$

- Processes I

(Input)
$$\frac{\Gamma \vdash u : \sharp(\vec{T}) \qquad \Gamma, \vec{x} : \vec{T} \vdash P}{\Gamma \vdash u(\vec{x}).P}\ \vec{x} \cap \mathsf{Dom}(\Gamma) = \varnothing$$

(Output)
$$\frac{\Gamma \vdash u : \sharp(\vec{T}) \qquad \Gamma \vdash \vec{v} : \vec{T} \qquad \Gamma \vdash P}{\Gamma \vdash \overline{u}\langle\vec{v}\rangle.P}$$

Typing rules: Process

Processes II

(Zero)
$$\frac{}{\Gamma \vdash \mathbf{0}}$$

(Par)
$$\frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q}$$

(Repl)
$$\frac{\Gamma \vdash P}{\Gamma \vdash !P}$$

(Restr)
$$\frac{\Gamma, a : T \vdash P}{\Gamma \vdash (\boldsymbol{\nu} a : T)P}$$

Type System Properties I

Subject Reduction:

- reduction preserves well-typedness.
- if $\Gamma \vdash P$ and $P \longrightarrow Q$, then $\Gamma \vdash Q$.
- needs
  - Substitution Lemma
    if $\Gamma \vdash u : T$ and $\Gamma, x : T \vdash P$, then $\Gamma \vdash P\{u/x\}$.
  - Subject Congruence
    if $\Gamma \vdash P$ and $P \equiv Q$, then $\Gamma \vdash Q$.

Type System Properties II

Type Safety

- well-typed processes communicate in type-consistent ways.
- Let $\Gamma, c : \sharp(T_1, \ldots, T_n) \vdash P$. If P contains

$$\vdash c(x_1, \ldots, x_h).Q_1 \mid \overline{c}\langle v_1, \ldots, v_k\rangle.Q_2$$

  then c is a name (not a basic value), $k = h = n$ and $v_i : T_i$.
- Subject reduction guarantees that this property holds of all derivatives of P.
- We will describe richer notions of type safety that provide security guarantees

Why Types? II

- Types help resource access control

  In the untyped π-calculus, resources (channels) are protected by *hiding* them

  Often too coarse a policy: protection is lost when the channel name is transmitted, as no assumption can be made on how the recipient of the name will use it.

  Types come to the rescue: enforce constraints on use of channels by associating them with *read* and/or *write* capabilities.

Example: printer

- printer P and two clients C_1 and C_2.
- P provides a request channel p carrying data to be printed
- π-calculus representation:

$$(\boldsymbol{\nu} p)(P \mid C_1 \mid C_2)$$

- if $C_1 \triangleq \overline{p}\langle j_1 \rangle.\overline{p}\langle j_2 \rangle.\ldots$, we expect that the jobs $j_1, j_2, \ldots$ are received and processed, in that order.
- Not necessarily true: C_2 might compete with P to "steal" the jobs sent by C_1 and throw them away: $C_2 \triangleq !\, p(j).\mathbf{0}$.

- Let's fix this with *Types*!

IO Types and Subtypes

$$
\begin{array}{rcll}
S, T & ::= & \ldots & \\
& \mid & \mathsf{i}T & \text{input capability on a channel of } T \text{ values} \\
& \mid & \mathsf{o}T & \text{output capability on a channel of } T \text{ values} \\
& \mid & \sharp T & \text{link type (channel)}
\end{array}
$$

Channel types:

- inform on the type of the value they carry
- offer capabilities to their users

Examples:

- i(int) : input-only channel carrying values of type int.
- ♯i(int) : channel carrying input-only integer channels.

Subtyping kicks in: any channel can be used in only one of its capabilities. . .

$\mathsf{i}T$ $\mathsf{o}T$

$\sharp T$

$\boxed{\text{Subsumption: if } x : \sharp T \text{, then } x : \mathsf{i}T \text{ and } x : \mathsf{o}T \text{ too}}$

Subtyping

The core of resource access control by typing: restrict capabilities in certain contexts to protect channels from misuse.

$$p : \sharp T, a : \sharp \mathsf{i}T \vdash \overline{a}\langle p \rangle.P \mid a(x).Q$$

P knows p as a read/write channel. Q receives it on a and therefore knows it only as a input-only channel. Name p can travel on a because of Subtyping ans subsumption.

(Subs Refl) $\dfrac{}{T \leqslant T}$ (Subs Tran) $\dfrac{T \leqslant T' \quad T' \leqslant T''}{T \leqslant T''}$

(Subs IO/I) $\dfrac{}{\sharp T \leqslant \mathsf{i}T}$ (Sub IO/O) $\dfrac{}{\sharp T \leqslant \mathsf{o}T}$

Subtyping, II

- subtyping applies also to argument types

(Sub I) $\dfrac{S \leqslant T}{\mathsf{i}S \leqslant \mathsf{i}T}$ (Sub O) $\dfrac{T \leqslant S}{\mathsf{o}S \leqslant \mathsf{o}T}$ (Sub IO) $\dfrac{T \leqslant S \quad S \leqslant T}{\sharp S \leqslant \sharp T}$

covariant contravariant invariant

- intuition: Assume $c : \sharp T$.
  - c can safely be used to read values at type T or higher, ...
  - provided that only values at type T, or lower, are written to c

  As for invariance, suppose nat $\leqslant$ int $\leqslant$ real and $a : \sharp(\text{int})$.
  If ♯ was variant, either this $P_1 = \overline{a}\langle 3.5 \rangle$ or $P_2 = a(x).\overline{\log}\langle x \rangle$ would be typable.

  Also $Q = a(x).\mathsf{succ}\ x \mid \overline{a}\langle -2 \rangle$ is obviously alright.
  But $P_1 \mid Q$ and $P_2 \mid Q$ are both flawed.

New typing rules

Processes

(Input)
$$\frac{\Gamma \vdash u : \mathsf{i}\vec{T} \qquad \Gamma, \vec{x} : \vec{T} \vdash P}{\Gamma \vdash u(\vec{x}).P}$$

(Output)
$$\frac{\Gamma \vdash u : \mathsf{o}\vec{T} \qquad \Gamma \vdash v_i : T_i \qquad \Gamma \vdash P}{\Gamma \vdash \overline{u}\langle\vec{v}\rangle.P}$$

(Subsumption)
$$\frac{\Gamma \vdash u : S \qquad S \leqslant T}{\Gamma \vdash u : T}$$

Subject Reduction: if $\Gamma \vdash P$ and $P \longrightarrow Q$, then $\Gamma \vdash Q$.

Typed printer

- use types to make sure the printer only reads from p, and the clients only write on p.
- initialize the system with two channels a and b, to send the name p to P and to C_1 and C_2 restricting the use of p.

$$\begin{aligned} S \quad &\triangleq \quad (\boldsymbol{\nu} p : \sharp T)\overline{a}\langle p\rangle.\overline{b}\langle p\rangle | a(x : \mathsf{i}T).P \mid b(y : \mathsf{o}T).(C_1 \mid C_2) \\ &\longrightarrow \quad (\boldsymbol{\nu} p : \sharp T)P\{p/x\} \mid (C_1 \mid C_2)\{p/y\} \end{aligned}$$

- typing ensures that P only reads, and C_i's only write on p
- With appropriate definitions for P and C_i's

$$a, b : \sharp(\sharp T) \vdash S$$

Typed printer II

Main steps of the typing derivation of $\Gamma \vdash \overline{a}\langle p\rangle.\overline{b}\langle p\rangle \mid a(x).P \mid b(y).(C_1 \mid C_2)$, where $\Gamma = \{a, b : \sharp(\sharp T), p : \sharp T\}$.

- $$\frac{\Gamma \vdash p : \sharp T \qquad \dfrac{\Gamma \vdash a, b : \sharp(\sharp T) \qquad \sharp(\sharp T) \leqslant \mathsf{o}(\sharp T)}{\Gamma \vdash a, b : \mathsf{o}(\sharp T)}}{\Gamma \vdash \overline{a}\langle p\rangle.\overline{b}\langle p\rangle}$$
- $$\frac{\dfrac{\sharp(\sharp T) \leqslant \mathsf{i}(\sharp T) \qquad \dfrac{\sharp T \leqslant \mathsf{i}T}{\mathsf{i}(\sharp T) \leqslant \mathsf{i}(\mathsf{i}T)}}{\Gamma \vdash a : \mathsf{i}(\mathsf{i}T)} \qquad \Gamma, x : \mathsf{i}T \vdash P}{\Gamma \vdash a(x).P}$$
- $$\frac{\dfrac{\sharp(\sharp T) \leqslant \mathsf{i}(\sharp T) \qquad \dfrac{\sharp T \leqslant \mathsf{o}T}{\mathsf{i}(\sharp T) \leqslant \mathsf{i}(\sharp T)}}{\Gamma \vdash b : \mathsf{i}(\mathsf{o}T)} \qquad \Gamma, y : \mathsf{o}T \vdash C_1 \mid C_2}{\Gamma \vdash b(y).(C_1 \mid C_2)}$$

Limitations

The simply typed λ calculus has only finite computation. Not so the simply type π calculus. There however limitations, due with the fact that terms must have a finite bound on the “nesting” of channels.

Thm. No well typed term can produce a sequence of actions such as

$$x_1(x_2).x_2(x_3).x_3(x_4).x_4(x_5).\cdots$$

'Cause, what would the type of x_1 be? $\sharp(\sharp(\sharp(\ldots)))$

The problem can be avoided with recursive types (Pierce, Sangiorgi).

Then $x_1 : \mu X.\sharp X$. The untyped calculus can be encoded satisfactorily in the recursively typed π using such type.

Advanced Type Systems

The work on types has push forward towards greater refinement and control of resources. We will see examples of types for secrecy and capability types.
A list of things we will not see:

- linear type systems, trying to control how many times a resource is used (Pierce, Kobayashi, Yoshida,)
- types for deadlocks avoidance (Kobayashi,...)
- Polymorphic types:
  $a : \sharp\langle X | \sharp X \times X\rangle \vdash \overline{a}\langle Int; c, s\rangle \mid a(x).\text{open } x \text{ as } (X; z, y) \text{ in } \overline{z}\langle y\rangle$ (Pierce, Sangiorgi)
- ...

The Case for Typed Behavioural Equivalences

What is the effect of types on behavioural equivalences?
Firstly, it makes no sense to compare processes with different types. Also, we know that

$$P = a(b).(\overline{b}\langle v\rangle \mid c(z)) \quad \not\cong^c \quad a(b).(\overline{b}\langle v\rangle.c(z) + c(z).\overline{b}\langle v\rangle) = Q$$

But what if we know, say, $\Gamma = \{a : \sharp\sharp S, c : \sharp T\} \vdash P, Q$ for $S \neq T$?

Then $\{a : \sharp\sharp S, c : \sharp T\} \triangleright P \cong^c Q$

This is because no legit context will be able to alias b an c.

Example: $P = (\boldsymbol{\nu} x)(\overline{a}\langle x\rangle \mid \overline{x}\langle b\rangle) \not\cong^c Q = (\boldsymbol{\nu} x)\overline{a}\langle x\rangle$.
P and Q are distinguished by $\mathscr{C} = (\boldsymbol{\nu} a)(a(x).x(z).\overline{c} \mid [.])$. However,

$$\Delta = \{a : \sharp\text{o}S, b : S\} \triangleright P_1 \cong^c P_2.$$

This is because no well-typed environment will be able to verify the presence of the output $\overline{x}\langle b\rangle$. Types make equivalence coarser, as they limit the "power of observer," that is the number of contexts.

Typed Barbed Bisimulation

Definition: A (Γ/Δ)-context is a Γ-context with a Δ-hole. That is, $\mathscr{C}$ such that whenever $\Delta \vdash P$, then $\Gamma \vdash \mathscr{C}(P)$.

Barbed Congruence. For $\Delta \vdash P, Q$ we say $\Delta \triangleright P \cong^c Q$ if for all closed Γ and all (Γ/Δ)-contexts $\mathscr{C}$, we have $\mathscr{C}(P) \dot{\approx} \mathscr{C}(Q)$.

Typed substitutions σ is a Δ/Γ substitution if for all $x \in \text{Dom}(\Delta)$, we have $\Gamma \vdash \sigma(x) : \Delta(x)$.

Typed Context Lemma $\Delta \triangleright P \cong^c Q$ if and only if all closed Γ which extend Γ, for all Δ/Γ substitutions, and all $\Gamma \vdash R$ we have $P\sigma \mid R \dot{\approx} Q\sigma \mid R$.

Typed Bisimulation

Typed notions of $\approx$ are know for most typed π calculi, but the issue can be problematic.

Consider

$$P = (\boldsymbol{\nu} xy)(\overline{a}\langle x\rangle \mid \overline{a}\langle y\rangle \mid !x.Q \mid !y.Q) \qquad Q = (\boldsymbol{\nu} x)(\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle \mid !x.Q).$$

These are $\cong^c$. A distinguishing context is $\mathscr{C} = a(z_1).a(z_2).(z_1().\overline{c} \mid \overline{z_2})$.
But if $\Gamma \vdash a : \text{oounit}$, then $\Gamma \triangleright P \cong^c Q$, because no context will be able to input and, therefore, tell y apart from x.

Matching actions is not trivial, though. Observe that

$$P \xrightarrow{\nu\overline{a}x} \xrightarrow{\nu\overline{a}y} \xrightarrow{yv}$$

$$Q \xrightarrow{\nu\overline{a}x} \xrightarrow{\overline{a}x} \xrightarrow{xv}$$

So, they are easily too fine. Need to refine with system/environment point of view of an action.
As a proof technique sometimes the context lemma works much better.

Types for Secrecy

An application in which types guarantee that secrets are not leaked by programs.
Expressed in the spi calculus.
Remember the wide mouth frog protocol? Let's add explicit encryption:

$$A \longrightarrow S : \{K_{AB}\}_{K_{AS}}$$
$$S \longrightarrow B : \{K_{AB}\}_{K_{BS}}$$
$$A \longrightarrow B : \{M\}_{K_{AB}}$$

The protocol now runs as

$$A(M) \triangleq (\nu K_{AB})\overline{c_{AS}}\langle\{K_{AB}\}_{K_{AS}}\rangle.\overline{c_{AB}}\langle\{M\}_{K_{AB}}\rangle$$
$$S \triangleq c_{AS}(x).\mathsf{case}\ x\ \mathsf{of}\ \{y\}_{K_{AS}}\ \mathsf{in}\ \overline{c_{SB}}\langle\{y\}_{K_{SB}}\rangle$$
$$B \triangleq c_{SB}(x).\mathsf{case}\ x\ \mathsf{of}\ \{y\}_{K_{SB}}\ \mathsf{in}\ c_{AB}(z).\mathsf{case}\ z\ \mathsf{of}\ \{w\}_y\ \mathsf{in}\ \mathsf{F}(w)$$
$$\mathsf{Inst}(M) \triangleq (\nu K_{AS}, K_{SB})(A(M) \mid S \mid B)$$

Secrecy of M: $\mathsf{Inst}(M) \cong \mathsf{Inst}(M')$, for all M'. (Similar notion available for authenticity)

Spi: an applied π calculus

$L, M, N ::= \ldots$

$\mid 0$	zero
$\mid \mathsf{succ}(M)$	successor
$\mid (M, N)$	pair
$\mid \{M_1, \ldots, M_k\}_N$	shared-key encryption

$P, Q, R ::= \ldots$

$\mid [M\ \mathsf{is}\ N]P$	match
$\mid \mathsf{let}\ (x, y) = M\ \mathsf{in}\ P$	pair splitting
$\mid \mathsf{case}\ M\ \mathsf{of}\ 0 : P,\ \mathsf{succ}(x) : Q$	integer case
$\mid \mathsf{case}\ L\ \mathsf{of}\ \{x_1, \ldots, x_k\}_N\ \mathsf{in}\ P$	shared-key decryption

Spi calculus: Semantics

$$\ldots \equiv \ldots$$
$$[M\ \mathsf{is}\ M]P \equiv P$$
$$\mathsf{let}\ (x, y) = (M, N)\ \mathsf{in}\ P \equiv P\{M/x, N/y\}$$
$$\mathsf{case}\ 0\ \mathsf{of}\ 0 : P\ \mathsf{succ}(x) : Q \equiv P$$
$$\mathsf{case}\ \mathsf{succ}(M)\ \mathsf{of}\ 0 : P,\ \ \mathsf{succ}(x) : Q \equiv Q\{M/x\}$$
$$\mathsf{case}\ \{M\}_N\ \mathsf{of}\ \{x\}_N\ \mathsf{in}\ P \equiv P\{M/x\}$$

(Comm)
$$\frac{}{n(x_1, \ldots, x_k)P \mid \overline{n}\langle M_1, \ldots, M_k\rangle.Q \longrightarrow P\{M_1/x_1, \ldots, M_k/x_k\}}$$

(Par)
$$\frac{P \to P'}{P \mid Q \to P' \mid Q}$$

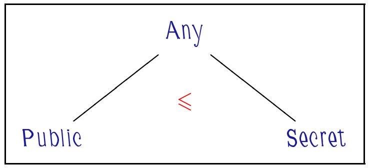
(New)
$$\frac{P \to P'}{(\nu n)P \to (\nu n)P'}$$

(Cong)
$$\frac{P \equiv P' \quad P' \to Q' \quad Q' \equiv Q}{P \to Q}$$

Secrecy

Data into three security classes, formalised as types:

- Public, which can be communicated
- Secret, which should not be leaked;
- Any, which is arbitrary data



Encryption keys are data. Only the following combination are reasonable

- encrypting v with a Public key has the same level as the data
  encrypting v with a Secret key can be made Public;
- only public data can be sent on public channels, while all kinds of data may be sent on secret channels.

Aim: Design a type system to guarantee the secrecy of parameters of type Any.

Messages and Confounders

To avoid confusion on the format of encrypted data, we adopt common one.

Secret confounder

$\{M_1, M_2, M_3, n\}_K$

Any Public

Message 1 $\quad B \to A: \quad N_B$

Message 2 $\quad A \to B: \quad \{M, N_B\}_{K_{AB}}$

This does not guarantee the secrecy of M. If an attacker sends a nonce N_C twice, A replies with ciphertexts $\{M, N_C\}_{K_{AB}}$ and $\{M', N_C\}_{K_{AB}}$. The attacker gets to know whether M and M' are the same message by just comparing the two ciphertexts.

Message 1 $\quad B \to A: \quad N_B$

Message 2 $\quad A \to B: \quad \{M, N_B, N_A\}_{K_{AB}}$

The confounder N_A is a fresh number that A creates for every encryption and prevents the information flow illustrated above.

The Guarantees

The confounder N_A is a fresh number for every encryption. This prevents the information flow arising from encrypting the same data repeatedly. The protocol with confounders:

Message 1 $\quad B \to A: \quad N_B$

Message 2 $\quad A \to B: \quad \{M, N_B, N_A\}_{K_{AB}}$

Expressed in the spi calculus, A's part of the protocol looks like this:

$$m(n_B)(\nu K)(\nu n_A)\overline{c}\langle\{n_B, x, *, n_A\}_K\rangle$$

where c is a public channel and x has level Any.

It is possible to show that this typechecks. Thus it does not leak the value of x, in the sense that $A[M/x]$ and $A[N/x]$ are equivalent for all closed M and N.

The Types

The Types:

- $\vdash E$ well formed means that environment E is well-formed.
- $E \vdash M : T$ means that term M is of level T in E.
- $E \vdash P$ means that process P type-checks in E.

Environments

(Env Empty)
$$\frac{}{\vdash \emptyset \text{ well formed}}$$

(Env Variable)
$$\frac{\vdash E \text{ well formed}}{\vdash E, x : T \text{ well formed}} \; x \notin dom(E)$$

(Env Name)
$$\frac{\vdash E \text{ well formed} \quad E \vdash M_1 : T_1 \ldots E \vdash M_k : T_k \quad E \vdash N : R}{\vdash E, n : T :: \{M_1, \ldots, M_k, n\}_N \text{ well formed}} \; n \notin dom(E)$$

Typing – Values

(Level Sub)
$$\frac{E \vdash M : T \quad T \leqslant R}{E \vdash M : R}$$

(Level Var)
$$\frac{\vdash E \text{ well formed} \quad x : T \text{ in } E}{E \vdash x : T}$$

(Level Name)
$$\frac{\vdash E \text{ well formed} \quad E \vdash n : T :: \{M_1, \ldots, M_k, n\}_N}{E \vdash n : T}$$

(Level Zero)
$$\frac{\vdash E \text{ well formed}}{E \vdash 0 : \text{Public}}$$

(Level Succ)
$$\frac{E \vdash M : T}{E \vdash \text{succ}(M) : T}$$

(Level Pair)
$$\frac{E \vdash M : T \quad E \vdash N : T}{E \vdash (M, N) : T}$$

Typing – Values II

(Level Enc Public)

$$\frac{E \vdash M_1 : T \ldots M_k : T \quad E \vdash N : \mathsf{Public}}{E \vdash \{M_1, \ldots, M_k\}_N : T} \; T{=}\mathsf{Public} \text{ if } k{=}0$$

(Level Enc Secret)

$$\frac{\begin{array}{l} E \vdash n : T :: \{M_1, M_2, M_3, n\}_N \\ E \vdash M_1 : \mathsf{Secret} \quad E \vdash M_2 : \mathsf{Any} \quad E \vdash M_3 : \mathsf{Public} \quad E \vdash N : \mathsf{Secret} \end{array}}{E \vdash \{M_1, M_2, M_3, n\}_N : \mathsf{Public}}$$

Typing – Processes

(Level Output Public)

$$\frac{E \vdash M : \mathsf{Public} \quad E \vdash M_1 : \mathsf{Public}, \ldots, E \vdash M_k : \mathsf{Public} \quad E \vdash P}{E \vdash \overline{M}\langle M_1, \ldots, M_k\rangle.P}$$

(Level Output Secret)

$$\frac{E \vdash M : \mathsf{Secret} \quad E \vdash M_1 : \mathsf{Secret} \quad E \vdash M_2 : \mathsf{Any} \quad E \vdash M_3 : \mathsf{Public} \quad E \vdash P}{E \vdash \overline{M}\langle M_1, M_2, M_3\rangle.P}$$

(Level Input Public)

$$\frac{E \vdash M : \mathsf{Public} \quad E, x_1 : \mathsf{Public}, \ldots, x_k : \mathsf{Public} \vdash P}{E \vdash M(x_1, \ldots, x_k).P}$$

(Level Input Secret)

$$\frac{E \vdash M : \mathsf{Secret} \quad E, x_1 : \mathsf{Secret}, x_2 : \mathsf{Any}, x_3 : \mathsf{Public} \vdash P}{E \vdash M(x_1, x_2, x_3).P}$$

(Level Nil) $\dfrac{\vdash E \text{ well formed}}{E \vdash \mathbf{0}}$ (Level Par) $\dfrac{E \vdash P \quad E \vdash Q}{E \vdash P \mid Q}$ (Level Rep) $\dfrac{E \vdash P}{E \vdash !P}$ (Level Res) $\dfrac{E, n : T :: L \vdash P}{E \vdash (\nu n)P}$

Typing – Processes II

(Level Match)

$$\frac{E \vdash M : T \quad E \vdash N : R \quad E \vdash P}{E \vdash [M \text{ is } N].P}$$

here and below T, R not Any

(Level Pair Split)

$$\frac{E \vdash M : T \quad E, x : T, y : T \vdash P}{E \vdash \text{ let } (x, y) = M \text{ in } P}$$

(Level Int Case)

$$\frac{E \vdash M : T \quad E \vdash P \quad E, x : T \vdash Q}{E \vdash \text{ case } M \text{ of } 0 : P, \text{ succ}(x) : Q}$$

(Level Dec Public)

$$\frac{E \vdash L : T \quad E \vdash N : \mathsf{Public} \quad E, x_1 : T, \ldots, x_k : T \vdash P}{E \vdash \text{ case } L \text{ of } \{x_1, \ldots, x_k\}_N \text{ in } P}$$

(Level Dec Secret)

$$\frac{E \vdash L : T \quad E \vdash N : \mathsf{Secret} \quad E, x_1 : \mathsf{Public}, x_2 : \mathsf{Any}, x_3 : \mathsf{Secret}, x_4 : \mathsf{Any} \vdash P}{E \vdash \text{ case } L \text{ of } \{x_1, x_2, x_3, x_4\}_N \text{ in } P}$$

Secrecy by Typing

Thm: Secrecy

Let E be an environment with only variables of level Any and names of level Public in Dom(E). Let σ, σ' be substitutions of values for the variables which respect E.

If $E \vdash P$, then $P\sigma \cong P\sigma'$

In other words, if P is well typed, then no observer that can tell $P\sigma$ apart from $P\sigma'$, so it cannot detect differences in the value of any parameter of type Any.

Groups for Secrecy: Scope extrusion revisited

$$p(x).O \mid (\boldsymbol{\nu} s)\ \overline{p}\langle s\rangle.P$$

- The name s is initially private to P. One step of reduction passes it over to O. This may be desirable, as we have seen.
- But we may instead want to keep s from escaping its initial scope. Eg, s could be a secret, and O an opponent.
- How can we do that?
- one could say: $\overline{p}\langle s\rangle$ should not occur in P, ie s should not be sent on a channel known to the opponent.
- But this is not easily enforced: p may be obtained dynamically from some other channel, and may not occur at all in P.

Controlling scope extrusion

- The problem must be approached carefully: scope extrusion is a fundamental mechanism.
- Idea: classify names into groups, and isolate a group G for names that should be secret. Then declare $(\boldsymbol{\nu} s : G)\overline{p}\langle s\rangle.P$.
- Global groups are of no use. Leakage can be made to typecheck:

$$p(y : G).O \mid (\boldsymbol{\nu} s : G)\overline{p}\langle s\rangle.P$$

- Groups themselves should be secret, so that P cannot output values of group G on public channels.
- A scope mechanism for groups:

$$p(y : T).O \mid (\boldsymbol{\nu} G)(\boldsymbol{\nu} s : G)\overline{p}\langle s\rangle.P$$

This will not typecheck if one tries to imply $T = G$, as G has local scope.

Pi Calculus with Groups

- Groups can be created dynamically

P	$::=$	$\dots$	as before
	$\mid$	$(\boldsymbol{\nu} G)P$	group creation

- Additional reduction
  - $(\boldsymbol{\nu} G)P \longrightarrow (\boldsymbol{\nu} G)Q$ if $P \longrightarrow Q$
- Additional congruence rules

$$(\boldsymbol{\nu} G_1)(\boldsymbol{\nu} G_2)P \equiv (\boldsymbol{\nu} G_2)(\boldsymbol{\nu} G_1)P$$

$$(\boldsymbol{\nu} G)(P \mid Q) \equiv P \mid (\boldsymbol{\nu} G)Q \qquad \text{if } G \notin \text{fg}(P)$$

$$(\boldsymbol{\nu} G)(\boldsymbol{\nu} a : T)P \equiv (\boldsymbol{\nu} a : T)(\boldsymbol{\nu} G)P \qquad \text{if } G \notin \text{fg}(T)$$

Groups and reduction

- Groups have no computational impact:

$$\mathit{erase}((\boldsymbol{\nu} G)P) \triangleq \mathit{erase}(P)$$

$$\mathit{erase}(a(\vec{x} : \vec{T}).P) \triangleq a(\vec{x}).\mathit{erase}(P)$$

$$\mathit{erase}((\boldsymbol{\nu} x : T)P) \triangleq (\boldsymbol{\nu} x)\mathit{erase}(P)$$

$$\dots \triangleq \dots$$

- $P \longrightarrow Q$ if and only if $\mathit{erase}(P) \longrightarrow R$, for some $R \equiv \mathit{erase}(Q)$.
- They do, however, affect typing

Types and Judgements

- Channel Types
  - $T, U ::= G[T_1, \ldots, T_n]$: polyadic channel in group G
- Type Environments
  - $\Gamma ::= \varnothing \mid \Gamma, G \mid \Gamma, u : T$ lists, not sets (!)
- Additional judgements
  - $\Gamma \vdash \diamond$: good environments
  - $\Gamma \vdash T$: good types
  - Intuition:

    $\Gamma \vdash T$ iff all group names in T are declared in Γ.

    $\Gamma \vdash \diamond$ iff all types in Γ are well-formed

Typing Rules: formation rules

Good Environments

$$\text{(Empty)}\ \frac{}{\varnothing \vdash \diamond} \qquad \text{(Env } u\text{)}\ \frac{\Gamma \vdash T \quad u \notin \mathsf{Dom}(\Gamma)}{\Gamma, u : T \vdash \diamond} \qquad \text{(Env } G\text{)}\ \frac{G \notin \mathsf{Dom}(\Gamma)}{\Gamma, G \vdash \diamond}$$

Good Types

$$\text{(Type Chan)}\ \frac{G \in dom(\Gamma) \quad \Gamma \vdash T_1 \ldots \Gamma \vdash T_n}{\Gamma \vdash G[T_1, \ldots, T_n]}$$

Typing Rules: processes

Process typing as before, e.g.

$$\text{(Input)}\ \frac{\Gamma \vdash u : G[T_1, \ldots, T_n] \quad \Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash P}{\Gamma \vdash u(x_1 : T_1, \ldots, x_n : T_n)P}$$

Rule for group creation

$$\text{(GRes)}\ \frac{\Gamma, G \vdash P}{\Gamma \vdash (\boldsymbol{\nu} G)P}$$

Properties of the type system

- Subject Reduction

  If $\Gamma \vdash P : T$ and $P \longrightarrow Q$, then $\Gamma \vdash Q : T$.
- Secrecy

  Let $S = p(y : U).O \mid (\boldsymbol{\nu} G)(\boldsymbol{\nu} x : G[\ldots])P$, and assume $\Gamma \vdash S$.
  - Then no process deriving from S outputs x along p.
  - Formally, for all processes Q, S' and S'', and contexts $\mathscr{C}[\cdot]$ such that

    $$S \equiv (\boldsymbol{\nu} G)(\boldsymbol{\nu} x : G[\ldots])S', \qquad S' \longrightarrow S'' \quad \text{and} \quad S'' \equiv \mathscr{C}[\overline{p}\langle x\rangle.Q],$$

    it is the case that p is bound by $\mathscr{C}[\cdot]$

Proof of secrecy

Assume:

$$\Gamma \vdash p(y:U).O \mid (\boldsymbol{\nu} G)(\boldsymbol{\nu} x:G[\dots])P$$
$$\longrightarrow (\boldsymbol{\nu} G)(\boldsymbol{\nu} x:G[\dots])\mathscr{C}[\overline{p}\langle x\rangle.Q]$$

with p not bound in $\mathscr{C}[\cdot]$.

- By subject reduction $\Gamma, G, x:G[\dots] \vdash \mathscr{C}[\overline{p}\langle x\rangle.Q]$.
- This implies that $\Gamma, G, x:G[\dots], \Gamma' \vdash \overline{p}\langle x\rangle.Q$ for some Γ'
- Then $\Gamma, G, x:G[\dots], \Gamma' \vdash p : H[G[\dots]]$ for some H.
- Impossible:
  - $\Gamma \vdash p(y:U).O \mid (\boldsymbol{\nu} G)(\boldsymbol{\nu} x:G[\dots])P$ implies $p \in Dom(\Gamma)$
  - Thus we would have $\Gamma \vdash p : H[G[\dots]]$, but this judgement is not derivable because $G \notin \mathsf{Dom}(\Gamma)$.

Untyped Opponents

Secrecy Thm generalises to the case of untyped/ill-typed opponents

- Idea (simplified): extend the type system so that all processes type check trivially:

$$\frac{\Gamma \vdash n : \mathsf{Un} \quad \Gamma, \vec{x} : \tilde{\mathsf{Un}} \vdash P}{\Gamma \vdash n(\vec{x} : \tilde{\mathsf{Un}}).P} \qquad \frac{\Gamma \vdash n : \mathsf{Un} \quad \Gamma \vdash M_i : \mathsf{Un} \quad \Gamma \vdash P}{\Gamma \vdash \overline{n}\langle M_1, \dots, M_k\rangle P}$$

- Untyped opponents can be made to typecheck by annotating all their free/bound names/variables with the type Un.
- Prove subject reduction for the new system
- Derive generalized secrecy

Summary of Lecture II

- We studied the use of (elementary) types in the π calculus, starting simple sorts to protect tupling for programmers' errors, arriving to types to protect secrecy.
- We introduced subtyping as a natural way to manage capabilities and disclose different 'views' of the same object to different users.
- We considered the why and how of typed equivalences.
- *Further Reading:*

  Again, the best starting points are:
  - The π-calculus: A theory of mobile processes (Sangiorgi, Walker)
  - The mobility home page `http://lamp.epfl.ch/mobility/`

  Consider also
  - The Spi Calculus (Abadi, Gordon)
  - Secrecy by Typing in Security Protocols (Abadi)
  - Secrecy and Group Creation (Cardelli, Ghelli, Gordon)

Global Computing

— Lecture III —

Asynchrony and Distribution

———

The Case for Asynchrony

Let us move a step towards realistic networks, trying to embed locations in our calculi. As a first step, let us build a case for:

Asynchrony. Channels in π are 'global' high-level, somehow unrealistic. Everybody can send and receive on them, regardless of location. The handshaking

$$\overline{x}.P \longleftrightarrow x.Q$$

represents an instantaneous action at a distance unfeasible in distributed networks, where localities, delays, and failures play a fundamental role (distributed consensus (Lynch)).

Also summation can be criticised especially in 'mixed' forms like

$$x.P + \overline{x}.Q \mid \overline{x}.Q + x.P' + k$$

Synchronised choice at a distance very hard to implement.

Also, $\overline{x}\langle z\rangle + y(z)$ makes little sense in general.

Let us abandon synchronous remote communication. . .

Roadmap of Lecture III

- The asynchronous π calculus
- The localised π calculus
- The distributed π calculus
- The join calculus

Asynchronous π

No continuation on output: $\bar{x}\langle y\rangle.\cancel{P}$
Continuations can be simulated as: $\tau.(\bar{x}\langle y\rangle \mid P)$.

How can P know when and if the output is received? Well, in general it can't. But . . . explicit continuations:

$$\text{sender: } \tau.(\overline{x}\langle y\rangle \mid c_x.P') \quad \text{receiver: } x(y).(Q' \mid \bar{c}_x).$$

The synchronisation is now much looser, and widely accepted to be a better base for distributed systems

- Action prefixes $\pi ::= \cancel{\overline{x}\langle y\rangle} \mid x(y) \mid \cancel{\tau}$
- Processes $P ::= \overline{x}\langle y\rangle \mid \cancel{\sum_{i\in I}} \pi_i.P_i \mid P \mid P \mid (\boldsymbol{\nu} a)P \mid \boldsymbol{!}P$

Asynchronous calculi often do not consider sums nor τ.

π_A: Expressiveness

Polyadic π:

$$\{\overline{x}\langle y_1, y_2\rangle\} \triangleq (\nu w)(\,\overline{x}\langle w\rangle \mid w(v).(\overline{v}\langle y_1\rangle \mid w(v).\overline{v}\langle y_2\rangle)\,)$$

$$\{x(z_1, z_2).P\} \triangleq x(w)(\nu v)(\overline{w}\langle v\rangle \mid v(z_1)(\overline{w}\langle v\rangle \mid v(z_2).\{P\}))$$

Synchronisation:

$$[\overline{x}\langle y\rangle.P] \triangleq (\nu a)\overline{x}\langle y, a\rangle \mid a.[P]$$

$$[x(z).P] \triangleq x(z, a).(\overline{a} \mid [P])$$

Exercise: Compose $\{\ \cdot\ \}$ and $[\ \cdot\ ]$ above. Prove that it takes $2n + 5$ monadic asynchronous communications to exchange one n-pla.

π_A: Expressiveness, II

Summation

$$\mathsf{Ok} \triangleq l(p,f).\overline{p} \qquad \mathsf{Fail} \triangleq l(p,f).\overline{f}$$

Terms in the summation will compete to grab $\overline{p}$ from Ok. The loosers will either hang forever, or grab $\overline{f}$ from Fail.

$$[x_1(y_1).P_1 + x_2(y_2).P_2] \triangleq$$
$$(\nu l)(\mathsf{Ok} \mid {}_{i=1,2}\, x_i(z).(\nu p f)(\overline{l}\langle p,f\rangle \mid p.(\mathsf{Fail} \mid [P_i]) \mid f.(\mathsf{Fail} \mid \overline{x_i}\langle z\rangle)))$$

Mixed choice cannot be encoded (satisfactorily).

π_L: The Local π calculus

Only the output capability can be received on names (either by typing or by syntactic restrictions). It makes a lot of sense in practice (eg printer, objects, distributed environment, ...). As a consequence, all the 'receivers' for a channel are local to the scope. One of the semantics consequence:

$$\pi_L \triangleright (\nu z)\overline{x}\langle z\rangle \cong^c (\nu z)(\overline{x}\langle z\rangle \mid \overline{z}\langle y\rangle)$$

Simulating read capabilities.
We represent a name x of π_A with a pair $\langle x^o, x\rangle$ of π_L, whose first component represent the (lost) read capability on x.

$$x^o \hookrightarrow x \triangleq !x^o(z).x(s,t).\overline{z}\langle s,t\rangle$$

$$[\overline{a}\langle x\rangle]_{\mathscr{E}} \triangleq (\nu x^o)(\overline{a}\langle x^o, x\rangle \mid x^o \hookrightarrow x)$$

$$[a(x).P]_{\mathscr{E}} \triangleq \begin{cases} a(z^o,z).[P]_{\mathscr{E}\cup\{z\}} & \text{if } a \notin \mathscr{E} \\ (\nu w)(\overline{a}^o\langle w\rangle \mid w(z^o,z).[P]_{\mathscr{E}\cup\{z\}} & \end{cases}$$

We will see later the Join calculus, which originated the idea of unique receptors.

On Asynchronous Bisimulation

Will the hypothesis of asynchrony bear consequences on bisimulation?

$$x(z).\overline{x}\langle z\rangle \stackrel{?}{=} \mathbf{0}$$

Matter of fact, with only asynchronous contexts, $\pi_A \triangleright !x(z).\overline{x}\langle z\rangle \cong^c \mathbf{0}$

This can be captured a LTS definition.
Asynchronous Bisimulation $\approx_a$: the largest equivalence s.t. for all $P \approx_a Q$

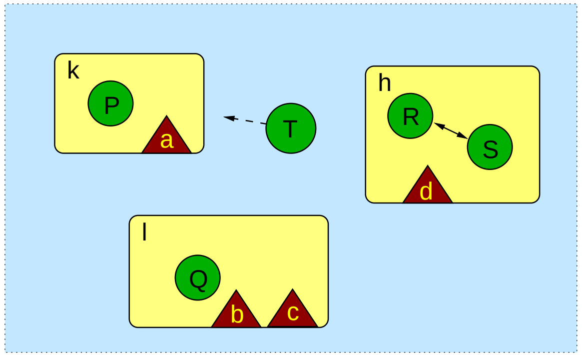
$$P \xrightarrow{\alpha} P' \text{ for } \alpha \in \{\overline{x}y, \nu\overline{x}z, \tau\} \text{ then } Q \stackrel{\hat{\alpha}}{\Longrightarrow}\approx_a P'$$

$$P \xrightarrow{xy} P' \text{then } Q \stackrel{xy}{\Longrightarrow}\approx_a P' \text{ or } Q \Longrightarrow Q' \text{ for } P' \approx_a Q' \mid \overline{x}\langle y\rangle$$

Note: Alternatively the 2nd clause: if $P{\downarrow_x}$, then $P \mid \overline{x}\langle y\rangle \approx_a Q \mid \overline{x}\langle y\rangle$ for all y.

$$\begin{tikzcd} & & \cong_a \\ & \cong_a^c \arrow[ur] & \\ \cong^c \arrow[ur] & & \approx_a \arrow[ul] \end{tikzcd}$$

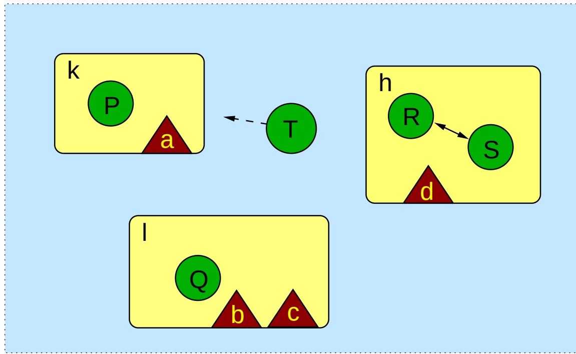
The Case for Distribution



Distributed Systems consist of:

- A collection of independent distributed sites offering services/resources to migrating agents.
- Resources: All sort of things a agent may long for (CPU time, space, printers, ...); we will model them as π-calculus channels for now.
- Agents: mobile processes of general nature; we will model them by augmented π-calculus processes.

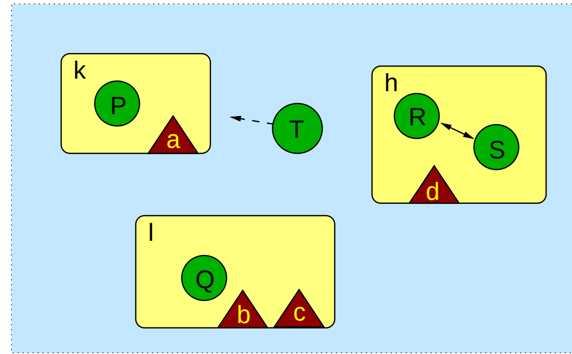
The Case for Distribution



More precisely... Add locations or sites, remove direct remote communication.
The basic elements are:

- Locations are sites containing processes: $l[\![P]\!]$
- Communication is only local: $l[\![\overline{x}\langle z\rangle.P \mid x(y).Q]\!] \longrightarrow l[\![P \mid Q\{z/y\}]\!]$
- Agents travel between locations $l[\![\text{goto}\, k\,.P \mid Q]\!] \mid k[\![R]\!] \longrightarrow l[\![Q]\!] \mid k[\![P \mid R]\!]$

The Case for Distribution



Agent Migration
This opens a lot of interesting "global" issues

- Which resource are available at a given location?
- How do we make sure their are used accordingly?

The use of types to control resource access and usage is an important current topic.

The distributed π calculus

- Values: $V ::= l \mid c \mid c@l$
- Sites: $M ::= \mathbf{0} \mid l[\![P]\!] \mid M \mid M \mid (\boldsymbol{\nu} e@l : T)\, M$

Example: $h[\![P]\!] \mid (\boldsymbol{\nu} a@l)(k[\![\ldots a@l \ldots]\!] \mid l[\![\ldots a \ldots]\!])$

- Threads:

$u(\vec{x} : \vec{T})Q$	*local* input on channel u
$\overline{u}\langle V\rangle Q$	*local* output on channel u
$\text{goto}\, u\, P$	code movement to site u
$[u = v]P\,;Q$	testing of names
$(\boldsymbol{\nu} e : T)\, P$	generation of new names (channels or locations)
$P \mid Q$	composition
$!P$	replication
$\mathbf{0}$	finished

Example: $l[\![d(x@z)P]\!] \mid k[\![(\boldsymbol{\nu} a)\ \text{goto}\, l\,.\overline{d}\langle a@k\rangle]\!]$

Semantics

Structural congruence

- $l[\![P \mid Q]\!] \equiv l[\![P]\!] \mid k[\![Q]\!]$
- $l[\![(\boldsymbol{\nu} a : T)\, P]\!] \equiv (\boldsymbol{\nu} a@l : T)\, l[\![P]\!]$
- ...

Reduction Semantics

- $k[\![\overline{c}\langle v\rangle.Q \mid c(x : T)P]\!] \longrightarrow k[\![Q \mid P\{v/x\}]\!]$
- $k[\![\text{goto}\, l\,.P]\!] \longrightarrow l[\![P]\!]$

The famous Cell (yet again)

Let us reconsider the the cell and write a distributed version.

- g – inputs a location; the location must have a channel called ret on which to return value
- p – inputs a location and new value; location must have a channel ack on which to send acknowledgement.

$$\mathsf{Cell}(n) \triangleq (\boldsymbol{\nu} s)(\overline{s}\langle n\rangle \mid\ !g(y).s(x).(\overline{s}\langle x\rangle \mid \mathsf{goto}\, y\, .\overline{\mathsf{ret}}\langle x\rangle) \mid !p(y,v).s(x)(\overline{s}\langle v\rangle \mid \mathsf{goto}\, y\, .\overline{\mathsf{ack}}\langle\rangle)$$

$$\mathsf{User} \triangleq \mathsf{goto}\, l\, .\overline{p}\langle h, 0\rangle \mid \mathsf{ack}().\,\mathsf{goto}\, l\, .\overline{g}\langle h\rangle \mid \mathsf{ret}(x)\ \overline{\mathsf{print}}\langle x\rangle$$

$$\mathsf{System} \triangleq l[\![\mathsf{Cell}(v)]\!] \mid h[\![\mathsf{User}]\!]$$

A refined cell

In order to use the cell, users have to have a ack and a reply channel. Here is a version where these are generated on purpose and communicated from the client to the cell.

$$\mathsf{System} \triangleq l[\![\mathsf{Cell}(v)]\!] \mid h[\![\mathsf{User}]\!]$$

$$\mathsf{User} \triangleq (\boldsymbol{\nu} ar)\, l :: \overline{p}\langle a@h, 0\rangle \mid a()(l :: \overline{g}\langle r@h\rangle \mid r(x).\overline{\mathsf{print}}\langle x\rangle)$$

$$\mathsf{Cell}(n) \triangleq (\boldsymbol{\nu} s)(\overline{s}\langle n\rangle \mid !g(z@y).s(x).(\overline{s}\langle x\rangle \mid y :: \overline{z}\langle x\rangle) \mid !p(z@y, v).s(x).(\overline{s}\langle v\rangle \mid y :: \overline{z}\langle\rangle)$$

Notation: calling a method at a location: $l :: \overline{a}\langle v\rangle$ shorthand for $\mathsf{goto}\, l\, .\overline{a}\langle v\rangle.\mathbf{0}$.

A Cell Factory

Here is a (final) version, where a new private cell is created for each user.

$$\mathsf{System} \triangleq s[\![\mathsf{S}]\!] \mid h_1[\![\mathsf{User}_1]\!] \mid h_2[\![\mathsf{User}_2]\!]$$

$$\mathsf{S} \triangleq !\mathsf{req}(z@y).\,(\boldsymbol{\nu} c)(y :: \overline{z}\langle c\rangle \mid \mathsf{goto}\, c\, .\mathsf{Cell}(0))$$

$$\mathsf{User}_i \triangleq (\boldsymbol{\nu} r)(s :: \overline{\mathsf{req}}\langle r@h_i\rangle \mid r(z).\mathsf{User}_i(z))$$

Evolution of the System

$$
\begin{aligned}
& s[\![\mathsf{S}]\!] \mid h_1[\![\mathsf{User}_1]\!] \mid h_2[\![\mathsf{User}_2]\!] \\
& \quad \longrightarrow s[\![\mathsf{S}]\!]\ (\boldsymbol{\nu} c_1)(c_1[\![\mathsf{Cell}(0)]\!] \mid h_1[\![\mathsf{User}_1]\!]) \mid h_2[\![\mathsf{User}_2]\!] \\
& \quad \longrightarrow s[\![\mathsf{S}]\!]\ (\boldsymbol{\nu} c_1)(c_1[\![\mathsf{Cell}(0)]\!] \mid h_1[\![\mathsf{User}_1]\!]) \mid (\boldsymbol{\nu} c_2)(c_2[\![\mathsf{Cell}(0)]\!] \mid h_2[\![\mathsf{User}_2]\!]) \\
& \quad \longrightarrow \ldots
\end{aligned}
$$

Exercise:

- Program a remote channel creation and a newloc construct.
- Program a forwarder from $in@a$ to $out@b$

Types for Resource Access Control

Location types: Key notion: describe the services available at a site.

$$\mathsf{loc}[s_1 : T_1, \ldots, s_n : T_n]$$

The purpose of the type system is to guarantee that incoming agents access only the resources granted to them, in the way granted to them. This is called

Resource Access Control.

Example: $\mathsf{comp\_f}[\![!\mathsf{req}(x : \mathsf{int}, \mathsf{ret}@l : \sharp(\mathsf{int})@\mathsf{loc}).l :: \overline{\mathsf{ret}}\langle f x\rangle]\!]$. Then,

$$\mathsf{comp\_f} : \mathsf{loc}[\mathsf{req} : \mathsf{i}(\mathsf{int}, \sharp(\mathsf{int})@\mathsf{loc})]$$

Subtyping:

Subtyping plays a central role in access control policies:

$$\frac{\vec{S} \leqslant \vec{T}}{\mathsf{loc}[\vec{s} : \vec{S}, s_k : S_k] \leqslant \mathsf{loc}[\vec{s} : \vec{T}]}\ \text{(Sub Loc)}$$

The receiver gains capabilities according to the type of the location. Using subtyping the sender can control this.
For instance, a receiver of l over a channel carrying $\mathsf{loc}[a : \mathsf{i}V, b : \mathsf{o}W]$ will be able to read V-values over a and write W-values to b, but not to use any other resource possibly present at l (and hidded by subtyping).

$$\frac{A \leqslant A' \quad L \leqslant L'}{A@L \leqslant A'@L'}\ \text{(Sub Remote l)}$$

Let us study the types of the previous examples, and unveil some of the issues involved.

The Types, more precisely

$V ::= A \mid A@L$	value types
$A ::= \sharp(T) \mid \mathsf{i}(T) \mid \mathsf{o}(T)$	local channels types
$L ::= \mathsf{loc}[s_1 : T_1, \ldots, s_n : T_n]$	location types
$T ::= (V, \ldots, V)$	transmission types
$P ::= @u$	process types
$N ::= \Diamond$	network types

A feel for the rules

$$\frac{\Gamma \vdash v : V \quad \Gamma \vdash V \leqslant V'}{\Gamma \vdash u : V'}\ \text{(Sub Val)} \qquad \frac{\Gamma \vdash \ell : L \quad \Gamma \vdash L \leqslant L'}{\Gamma \vdash \ell : L'}\ \text{(Sub Loc)}$$

$$\frac{\Gamma \vdash u : A \quad \Gamma \vdash L \leqslant \mathsf{loc}}{\Gamma \vdash u@L : A@L}\ \text{(Val Rem)}$$

$$\frac{\Gamma \vdash \ell : \mathsf{loc}[a : \mathsf{o}T] \quad \Gamma \vdash \vec{u} : T \quad \Gamma \vdash P : @\ell}{\Gamma \vdash \overline{a}\langle\vec{u}\rangle.P : @\ell}\ \text{(Out)}$$

$$\frac{\Gamma \vdash \ell : \mathsf{loc}[a : \mathsf{i}\vec{T}] \quad \Gamma, \vec{x} : \vec{T} \vdash P : @\ell}{\Gamma \vdash a(\vec{x} : \vec{T}).P : @\ell}\ \text{(In)}$$

$$\frac{\Gamma \vdash P : @\ell}{\Gamma \vdash \mathsf{goto}\,\ell\,.P : @k}\ \text{(Go)} \qquad \frac{\Gamma \vdash \ell : \mathsf{loc} \quad \Gamma \vdash P : @\ell}{\Gamma \vdash \ell[\![P]\!]}\ \text{(Loc)}$$

Typing the cell

The simple cell

$$\mathsf{Cell}(n) \triangleq (\boldsymbol{\nu} s : \sharp V)(\overline{s}\langle n\rangle \mid !g(z@y : \mathsf{o}V@\mathsf{loc}).s(x : V).(\overline{s}\langle x\rangle \mid y :: \overline{z}\langle x\rangle)$$
$$\mid !p(z@y : \mathsf{o}(\mathsf{unit})@c, v : V).s(x : V).(\overline{s}\langle v\rangle \mid y :: \overline{z}\langle\rangle)$$

$$\mathsf{cell} : \mathsf{CELL} = \mathsf{loc}[g : \sharp(\mathsf{o}V@\mathsf{loc}), p : \sharp(\mathsf{o}(\mathsf{unit})@\mathsf{loc}, V)]$$

The cell factory

$$\mathsf{S} \triangleq !\mathsf{req}(z@y : \mathsf{o}(\mathsf{CELL})@\mathsf{loc}).\,(\boldsymbol{\nu} c : \mathsf{CELL})(y :: \overline{z}\langle c\rangle \mid \mathsf{goto}\,c\,.\mathsf{Cell}(0))$$

$$\mathsf{s} : \mathsf{loc}[\mathsf{req} : \sharp(\mathsf{o}(\mathsf{CELL})@\mathsf{loc})]$$

Dynamic Types

Consider a generic system

$$\mathsf{server}[\![\ldots \mid !\mathsf{quest}(v : V, x@l : \mathsf{o}(\mathsf{ANSW})@\mathsf{loc}) \ldots l :: \overline{x}\langle \mathsf{answ}\rangle]\!]$$

The type of server:

$$\mathsf{Serv} \triangleq \mathsf{loc}[\mathsf{quest} : \sharp(V, \mathsf{o}(\mathsf{ANSW})@\mathsf{loc}), \ldots]$$

Now, suppose that client wants to setup a 'private' service, so that answers to quest's can reach only it.

$$\mathsf{client}[\![\mathsf{server} :: \overline{\mathsf{setup}}\langle \mathsf{reply}@\mathsf{client}\rangle.\mathsf{reply}(s : \mathsf{Serv}) \ldots]\!]$$

$$\mathsf{server}[\![!\mathsf{setup}(r@z : \mathsf{Serv}).\,(\boldsymbol{\nu} s : \mathsf{Serv}_z)$$

$$\mathsf{goto}\, s\,.(z :: \overline{\mathsf{reply}}\langle s\rangle \mid !\mathsf{quest}(v, x@z).z :: \overline{y}\langle f(x)\rangle \ldots)]\!]$$

$$\mathsf{Serv}_z \triangleq \mathsf{loc}[\mathsf{quest} : \sharp(V, \mathsf{o}(\mathsf{ANSW})@z), \ldots]$$

$$\mathsf{server} \xrightarrow{\nu\, \overline{\mathsf{setup}}\, k} (\boldsymbol{\nu} s : \mathsf{loc}[\mathsf{quest} : \sharp(V, \mathsf{o}(\mathsf{ANSW})@k)])\; s[\![\ldots]\!]$$

Dynamic Types, Resolved

$$V ::= A \mid A@L \mid A@v \qquad \text{value types}$$

(Sub Remote II)
$$\frac{A \leqslant A'}{A@u \leqslant A'@u}$$

(Sub Remote III)
$$\frac{\Gamma \vdash v : L}{A@v \leqslant A@L}$$

(Val Rem II)
$$\frac{\Gamma \vdash u : A@v \quad \Gamma \vdash v : L \leqslant \mathsf{loc}}{\Gamma \vdash u@v : A@v}$$

The move capability

Add a new component of types which allows to control movements. Consider

$$M ::= \ldots \mathsf{loc}[\mathsf{move}_\ell, \vec{x} : \vec{T}]$$

The obvious typing rule is

(Move)
$$\frac{\Gamma \vdash k : \mathsf{loc}[\mathsf{move}_\ell] \qquad \Gamma \vdash P : @k}{\Gamma \vdash \mathsf{goto}\, k\,.P : @\ell}$$

Similarly, one can add a capability new_ℓ to allow processes from ℓ to create new channels at a given location k. And more.

Example: A confidential account

$$\mathsf{bank}[\![!\mathsf{new\_account}(y@z : \sharp W@\mathsf{loc}, \_@a : \mathsf{i}()@\mathsf{Agent})$$

$$(\boldsymbol{\nu}\mathsf{loc} : [\mathsf{move}_a, \mathsf{withdraw} : \ldots, \mathsf{deposit} : \ldots]) \ldots]\!]$$

Location Typed Bisimulation

Question: Does this affect the behavioural equivalences?

$$k[\![\overline{b}\langle\rangle]\!] \overset{?}{\cong^c} k[\![\mathbf{0}]\!]$$

Of course. The impact of types with locations is even greater than before. Whether the equivalence above holds depends on Γ. Precisely, can we move to k to observe b?

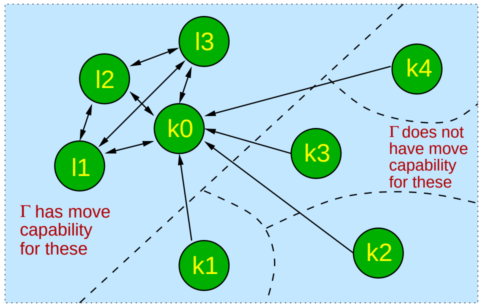
Consider

$$(\boldsymbol{\nu} k : \mathsf{loc}[a : \sharp(), \mathsf{move}_{h_1}])\; h_1[\![\overline{d}\langle k\rangle]\!] \mid h_2[\![\overline{c}\langle k\rangle]\!] \mid k[\![\overline{a}\langle\rangle]\!]$$

$$(\boldsymbol{\nu} k : \mathsf{loc}[a : \sharp(), \mathsf{move}_{h_1}])\; h_1[\![\overline{d}\langle k\rangle]\!] \mid h_2[\![\overline{c}\langle k\rangle]\!] \mid k[\![\mathbf{0}]\!]$$

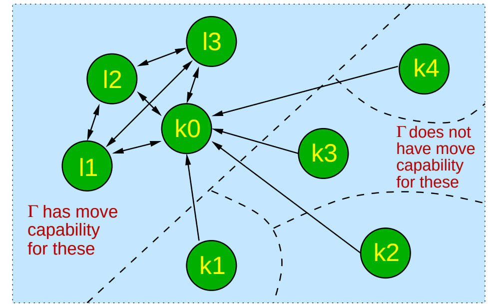
They are equivalent if Γ contains: $d : \mathsf{i}(\mathsf{loc}[\mathsf{move}])$ and $c : \mathsf{i}(\mathsf{loc}[a : \sharp()])$ and it is not possible to move between h_1 and h_2.

Location Typed Bisimulation



Let $\mathscr{T}$ be the collection of locations (*e.g.* $\{k1, k2, k3, k4\}$) for which Γ does not have move capability but the environment may *a priori* have code running at.

Location Typed Bisimulation



Transition of the form: $(\overline{\Gamma} \triangleright M) \xrightarrow{\alpha} (\overline{\Gamma}' \triangleright M')$ where $\overline{\Gamma}$ is of the form $(\Gamma_{k0}, \Gamma_{k1}, \ldots, \Gamma_{kn})$ for $\mathscr{T} = \{k1, \ldots, kn\}$ such that

$$\Gamma_{k0} \leqslant \Gamma_{ki} \text{ for each } ki \in \mathscr{T}$$

- Knowledge gained at any li is learned in Γ_{k0}
- Knowledge gained at ki is learned in Γ_{ki} and Γ_{k0}

Labelled transition system example rules

If

- Γ_{k0} contains move capability for l
- and Γ_{k0} contains read capability for a at l for values of type T
- then $(\overline{\Gamma} \triangleright l[\![\overline{a}\langle v\rangle.P]\!]) \xrightarrow{l::\overline{a}v} (\overline{\Gamma} \sqcap_{k0} v : T@l \triangleright P)$

If

- Γ_{k0} does not contain move capability at $k \in \mathscr{T}$
- and Γ_k contains read capability for a at k for values of type T
- then $(\overline{\Gamma} \triangleright k[\![\overline{a}\langle v\rangle.P]\!]) \xrightarrow{k::\overline{a}v} (\overline{\Gamma} \sqcap_k v : T@l \sqcap_{k0} v : T@l \triangleright P)$

Full abstraction

Thm

With the simplifying assumption in place that all move capabilities are of the form move_* (that is everybody is allowed in, or nobody is) then, for all systems of Dπ and all $\mathscr{T}$ we have

$$\Gamma \triangleright M \cong^c N \qquad \text{iff} \qquad \overline{\Gamma}_\triangledown \triangleright M \approx^{\mathscr{T}} N$$

where $\overline{\Gamma}_\triangledown = (\Gamma, \Gamma, \ldots, \Gamma)$ and $\approx^{\mathscr{T}}$ is the bisimulation equivalence induced by the labelled transition system.

Open issues:

- Remove the simplifying assumption about move capabilities
- Other capabilities: permission to exit, permission to create new channels, ...

The Join Calculus

The Join calculus (aka the Reflexive Chemical Abstract Machine): a version of asynchronous π combining restriction, reception, and replication in one construct:

Join receptor: $J \triangleright P$.

For example the definition

$$\mathtt{def\ apply}\langle \mathtt{f}, \mathtt{x}\rangle \triangleright \mathtt{f}\langle \mathtt{x}\rangle$$

defines apply that receives two arguments and applies the first to the second, as shown by the reduction:

$$\mathtt{def\ apply}\langle \mathtt{f},\mathtt{x}\rangle \triangleright \mathtt{f}\langle \mathtt{x}\rangle\ \mathtt{in\ apply}\langle \mathtt{g},\mathtt{y}\rangle \longrightarrow \mathtt{def\ apply}\langle \mathtt{f},\mathtt{x}\rangle \triangleright \mathtt{f}\langle \mathtt{x}\rangle\ \mathtt{in\ g}\langle \mathtt{y}\rangle$$

Notice:

- Definition persistence and locality
- This is very similar to $(\nu\mathtt{apply})(!\mathtt{apply}(x,y).\overline{x}\langle y\rangle \mid \overline{\mathtt{apply}}\langle \mathtt{g},\mathtt{y}\rangle)$ in π_A. In general, def D in P corresponds to a π form of the kind $(\nu\vec{d})(!\ulcorner D\urcorner \mid \ulcorner P\urcorner)$.

The syntax

Processes	P, Q	$::=$	$x\langle\widetilde{V}\rangle$	Asynchronous message on x
			def D in P	Definition of D in P
			$P \mid Q$	Parallel Composition
			0	Empty Process
Join patterns	J, J'	$::=$	$x\langle\tilde{y}\rangle$	Asynchronous reception on x
			$J \mid J'$	Joining messages
Definition	D, E	$::=$	$J \triangleright P$	Elementary clause
			$D \wedge E$	Simultaneous definition
Values	V, V'	$::=$	$\tilde{x}$	Names

The only synchronisation primitive is the Join pattern:

$$\mathtt{def}\ x\langle z_1\rangle \mid y\langle z_2\rangle \triangleright Q\ \mathtt{in}\ P$$

The RCHAM Semantics

Reflexive Chemical Abstract Machine: Structural rules $\rightleftharpoons$ plus reduction $\rightarrow$: States of the RCHAM are expression of the form $D \models P$, where P are the running processes and D are the (chemical) reactions.

(str-join)	$\models P \mid Q$	$\rightleftharpoons$	$\models P, Q$	
(str-def)	$\models \mathtt{def}\ D\ \mathtt{in}\ P$	$\rightleftharpoons$	$D_{\sigma_{dv}} \models P_{\sigma_{dv}}$	σ_{dv} instantiates $dv(D)$ freshly
(red)	$J \triangleright P \models J_{\sigma_{rv}}$	$\rightarrow$	$J \triangleright P \models P_{\sigma_{rv}}$	σ_{rv} substitutes for $rv(J)$

$$\models \mathtt{def}\ x\langle z\rangle \triangleright x\langle z,z\rangle\ \mathtt{in}\ x\langle a\rangle \mid \mathtt{def}\ x\langle z\rangle \triangleright x\langle x,z,z\rangle\ \mathtt{in}\ x\langle b\rangle$$
$$\rightleftharpoons k\langle z\rangle \triangleright k\langle z,z\rangle \models k\langle a\rangle, \mathtt{def}\ x\langle z\rangle \triangleright x\langle x,z,z\rangle\ \mathtt{in} x\langle b\rangle$$
$$\rightleftharpoons k\langle z\rangle \triangleright k\langle z,z\rangle, r\langle z\rangle \triangleright r\langle r,z,z\rangle \models k\langle a\rangle, r\langle b\rangle$$
$$\rightarrow k\langle z\rangle \triangleright k\langle z,z\rangle, r\langle z\rangle \triangleright r\langle r,z,z\rangle \models k\langle a,a\rangle, r\langle b\rangle$$
$$\rightarrow k\langle z\rangle \triangleright k\langle z,z\rangle, r\langle z\rangle \triangleright r\langle r,z,z\rangle \models k\langle a,a\rangle, r\langle r,b,b\rangle$$
$$\rightleftharpoons\ \models \mathtt{def}\ x\langle z\rangle \triangleright x\langle z,z\rangle\ \mathtt{in}\ x\langle x,a\rangle \mid \mathtt{def}\ x\langle z\rangle \triangleright x\langle x,z,z\rangle\ \mathtt{in}\ x\langle x,b,b\rangle$$

Example: Join pattern

$$\mathtt{def\ ready}\langle \mathtt{printer}\rangle \mid \mathtt{print}\langle \mathtt{file}\rangle \triangleright \mathtt{printer}\langle \mathtt{file}\rangle\ \mathtt{in}\ P$$

reduces only in the presence of messages on both ready and print, concurrently

$$\mathtt{def\ ready}\langle \mathtt{printer}\rangle \mid \mathtt{print}\langle \mathtt{file}\rangle \triangleright \mathtt{printer}\langle \mathtt{file}\rangle$$
$$\mathtt{in\ ready}\langle \mathtt{gutenberg}\rangle \mid \mathtt{print}\langle \text{"}\mathtt{slides.ps}\text{"}\rangle \mid Q$$
$$\longrightarrow \mathtt{def\ ready}\langle \mathtt{printer}\rangle \mid \mathtt{print}\langle \mathtt{file}\rangle \triangleright \mathtt{printer}\langle \mathtt{file}\rangle$$
$$\mathtt{gutenberg}\langle \text{"}\mathtt{slides.ps}\text{"}\rangle \mid Q$$

The same behavior is obtained by composing the definitions of apply and printer:

$$\mathtt{def\ apply}\langle \mathtt{f},\mathtt{x}\rangle \triangleright \mathtt{f}\langle \mathtt{x}\rangle \ \wedge\ \mathtt{ready}\langle p\rangle \mid \mathtt{print}\langle f\rangle \triangleright \mathtt{apply}\langle p, f\rangle$$

Derived constructs

P, Q	$::=$	$f(\widetilde{V}); P$	Sequential Composition
		$\mathtt{let}\ \tilde{x} = \widetilde{V} \mathtt{in}\ P$	Named Values
		$\mathtt{reply}\ \widetilde{V} \Rightarrow f$	Implicit Continuation
J, J'	$::=$	$f(\tilde{x})$	Synchronous reception on f
V, V'	$::=$	$f(\widetilde{V})$	Synchronous Call

$\lfloor f(\tilde{u}) \rfloor$	$\triangleq$	$f\langle \tilde{u}, \kappa_f \rangle$	(join pattern)
$\lfloor \mathtt{reply}\ \widetilde{V} \Rightarrow f \rfloor$	$\triangleq$	$\kappa_f \langle \widetilde{V} \rangle$	(process)
$\lfloor x\langle \widetilde{V} \rangle \rfloor$	$\triangleq$	$\mathtt{let}\ \tilde{u} = \widetilde{V}\ \mathtt{in}\ x\langle \tilde{u} \rangle$	(asynchronous call)
$\lfloor \mathtt{let}\ \tilde{u} = f(\widetilde{V})\ \mathtt{in}\ P \rfloor$	$\triangleq$	$\mathtt{def}\ \kappa\langle \tilde{u} \rangle \triangleright P\ \mathtt{in}\ f\langle \widetilde{V}, \kappa \rangle$	(synchronous call)
$\lfloor \mathtt{let}\ \tilde{u} = \tilde{v}\ \mathtt{in}\ P \rfloor$	$\triangleq$	$P\{\tilde{v}/\tilde{u}\}$	
$\lfloor f(\widetilde{V}); P \rfloor$	$\triangleq$	$\mathtt{def}\ \kappa\langle\rangle \triangleright P\ \mathtt{in}\ f\langle \widetilde{V}, \kappa \rangle$	(sequencing)

Distributed Join Calculus

The distributed reflexive chemical abstract machine (DRCHAM) is a multiset of "located" RCHAMs

$$D_1 \models_{\rho_1} P_1 \| \cdots \| D_k \models_{\rho_k} P_k$$

Each of these is a running location, and they are related to each other by a sublocation relation:

$\models_\phi$ is a sublocation of $\models_\rho$ if ρ is a prefix of ϕ.

(str-loc)	$a[D : P] \models_\rho$	$\rightleftharpoons$	$\models_\rho \| D \models_{\rho a} P$	(a frozen)
(comm)	$\models_\phi x\langle\tilde{v}\rangle \ \|\ J \triangleright P \models$	$\longrightarrow$	$\models_\phi \ \|\ J \triangleright P \models x\langle\tilde{v}\rangle$	$(x \in dv(J))$
(move)	$a[D : P \| go\langle b, \kappa\rangle] \models_\phi \ \|\ \models_{\psi b}$	$\longrightarrow$	$\models_\phi \ \|\ a[D : P \| \kappa\langle\rangle] \models_{\psi b}$	

Notice:

- Remove messages are forwarded to the unique solution where they are defined. There the usual (red) applies.
- The entire location moves, not the thread.

The cell in Join

$$\models \mathtt{def}\ \mathsf{cell}[s\langle x\rangle \mid \mathsf{get}\langle r\rangle \triangleright r\langle x\rangle \mid s\langle x\rangle \wedge$$
$$s\langle x\rangle \mid \mathsf{put}\langle v\rangle \triangleright s\langle v\rangle : s\langle 0\rangle]\ \mathtt{in}\ \mathsf{get}\langle x\rangle; \mathsf{put}\langle 2\rangle$$

$$\longrightarrow \ \mathsf{cell}[D : s\langle 0\rangle] \models \mathsf{get}\langle x\rangle; \mathsf{put}\langle 2\rangle \ \longrightarrow \ D \models_{\mathsf{cell}} s\langle 0\rangle \| \models \mathsf{get}\langle x\rangle; \mathsf{put}\langle 2\rangle$$
$$\longrightarrow \ D \models_{\mathsf{cell}} s\langle 0\rangle \mid \mathsf{get}\langle x\rangle \| \models \mathsf{put}\langle 2\rangle \ \longrightarrow \ D \models_{\mathsf{cell}} s\langle 0\rangle \mid x\langle 0\rangle \| \models \mathsf{put}\langle 2\rangle$$
$$\longrightarrow \ D \models_{\mathsf{cell}} s\langle 0\rangle \mid x\langle 0\rangle \mid \mathsf{put}\langle 2\rangle \ \longrightarrow \ D \models_{\mathsf{cell}} s\langle 2\rangle \mid x\langle 0\rangle \longrightarrow$$

$$\models \mathtt{def}\ \mathsf{cell}[s\langle x\rangle \mid \mathsf{get}\langle r\rangle \triangleright r\langle x\rangle \mid s\langle x\rangle \wedge$$
$$s\langle x\rangle \mid \mathsf{put}\langle v\rangle \triangleright s\langle v\rangle : s\langle 2\rangle]\ \mathtt{in}\ x\langle 0\rangle$$

Exercise: Program a mobile cell.

Summary of Lecture III

- We studied asynchrony in the π calculus and its consequences on behavioural equivalences.
- We considered a distributed π calculus and a notion of typed equivalence.
- We introduced the distributed join calculus.
- *Further Reading:*

  Again, the best starting points are:
  - The π-calculus: A theory of mobile processes (Sangiorgi, Walker)
  - The mobility home page http://lamp.epfl.ch/mobility/

  Consider also
  - Asynchronous π (Boudol), (Honda, Tokoro)
  - Comparing Synchronous and Asynchronous π (Palamidessi)
  - Decoding Choice Encodings (Pierce, Nestmann)
  - On Asynchronous Bisimulation (Amadio, Castellani, Sangiorgi)
  - Access and Mobility Control in Distributed Systems (Hennessy, Rathke, et al)
  - Join Calculus (Fournet, Gonthier, et al) http://pauillac.inria.fr

— Lecture VI —

Ambient Mobility

- Distribution over wide-area networks introduces new issues and breaks many of assumptions usually made in concurrent systems. We have already discussed asynchrony.
- Other examples:
  - existence of explicit physical locations determines latency in communication,
  - existence of virtual locations enforcing security policies that can restrict access to resources or even the visibility of locations.
- Mobile Ambients: a calculus of locations with migration primitives sufficiently expressive to encode the π-calculus, and presenting a new idea:

Modelling Environment Mobility

- Mobile Ambients
- Typing ambients
- Boxed Ambients
- Types for access control in ambients

Mobile Ambients:

- Are named agents: $n[\,P\,]$
- Represent computational environments, containing data and live computations: $n[\,P \mid \langle M\rangle\,]$
- Can be nested: $n[\,m[\,P\,] \mid k[\,Q\,]\,]$
- Move under the control of their enclosed processes: $n[\,\mathsf{in}\ m.P\,] \mid m[\,Q\,]$
  This is dubbed (subjective mobility).

Provide a direct representation of the structured nature of physical or administrative domains as well as of mobile computational environments.

The Essence of Mobile Ambients

- Subjective movements

$$n[\,\mathsf{in}\ m.P \mid Q\,] \mid m[\,R\,] \longrightarrow m[\,n[\,P \mid Q\,] \mid R\,]$$
$$m[\,n[\,\mathsf{out}\ m.P \mid Q\,] \mid R\,] \longrightarrow n[\,P \mid Q\,] \mid m[\,R\,]$$

- Process interaction

$$n[\,\langle M\rangle.P \mid (x).Q\,] \longrightarrow n[\,P \mid Q\{M/x\}\,],$$

- Boundary dissolver

$$\mathsf{open}\ n.P \mid n[\,Q\,] \longrightarrow P \mid Q.$$

Syntax

Expressions				Processes			
M	$::=$	$a,\ldots,q$	names	P	$::=$	$\mathbf{0}$	stop
	$\mid$	$x,\ldots,z$	variables		$\mid$	$(\boldsymbol{\nu} a : W)P$	restriction
	$\mid$	$\mathsf{in}\ M$	enter M		$\mid$	$P \mid P$	composition
	$\mid$	$\mathsf{out}\ M$	exit M		$\mid$	$!P$	replication
	$\mid$	$\mathsf{open}\ M$	open M		$\mid$	$M[\,P\,]$	ambient
	$\mid$	$(M_1,\ldots,M_k)$	tuple		$\mid$	$M.P$	action
	$\mid$	$M.M$	path		$\mid$	$(\vec{x} : \vec{W})P$	input
					$\mid$	$\langle M\rangle.P$	output

Note: Channel are local and anonymous. Output can be synchronous.

Structural Congruence

$P \equiv Q$

- $(M.M').P \equiv M.(M'.P)$
- $(\boldsymbol{\nu} a : W)b[\,P\,] \equiv b[\,(\boldsymbol{\nu} a : W)P\,]$ for $a \neq b$
- $(\boldsymbol{\nu} a : W)\mathbf{0} \equiv \mathbf{0}$
- $(\boldsymbol{\nu} a : W)a[\,\mathbf{0}\,] \equiv \mathbf{0}$
- $(\boldsymbol{\nu} a : W)(\boldsymbol{\nu} b : W')P \equiv (\boldsymbol{\nu} b : W')(\boldsymbol{\nu} a : W)P$ for $a \neq b$
- $(\boldsymbol{\nu} a; W)(P \mid Q) \equiv P \mid (\boldsymbol{\nu} a : W)Q$ for $a \notin \mathsf{fn}(P)$
- $!P \equiv !P \mid P,$
- usual rules for symmetricity, associativity and identity

Mobility: An example

Firewalls

- An agent crosses a firewall by means of passwords k, k_1 and k_2. The firewall, with secret name w, sends out the pilot ambient k to guide the agent inside.

$$\mathit{Firewall} \triangleq (\boldsymbol{\nu} w)w[\,k[\,\mathsf{out}\ w.\mathsf{in}\ k_1.\mathsf{in}\ w\,] \mid \mathsf{open}\ k_1.\mathsf{open}\ k_2.P\,]$$
$$\mathit{Agent} \triangleq k_1[\,\mathsf{open}\ k.k_2[\,Q\,]\,]$$

- Agent exhibits the passwd k_1 using k_1 as wrapper ambient
- Firewall verifies that agent knows the passwd with in k_1
- in w carries the agent into the firewall
- k_2 prevents Q from interfering with the protocol

A run of the protocol

$\text{Firewall} \triangleq (\boldsymbol{\nu} w)w[\,k[\,\text{out } w.\text{in } k_1.\text{in } w\,] \mid \text{open } k_1.\text{open } k_2.P\,]$

$\text{Agent} \triangleq k_1[\,\text{open } k.k_2[\,Q\,]\,]$

$\text{Firewall} \mid \text{Agent}$

$\longrightarrow (\boldsymbol{\nu} w)k[\,\text{in } k_1.\text{in } w\,] \mid w[\,\text{open } k_1.\text{open } k_2.P\,] \mid k_1[\,\text{open } k.k_2[\,Q\,]\,]$

$\longrightarrow (\boldsymbol{\nu} w)w[\text{open } k_1.\text{open } k_2.P] \mid k_1[k[\text{in } w] \mid \text{open } k.k_2[Q]]$

$\longrightarrow (\boldsymbol{\nu} w)w[\text{open } k_1.\text{open } k_2.P] \mid k_1[\text{in } w \mid k_2[Q]]$

$\longrightarrow (\boldsymbol{\nu} w)w[\,\text{open } k_1.\text{open } k_2.P \mid k_1[\,k_2[\,Q\,]\,]\,]$

$\longrightarrow (\boldsymbol{\nu} w)w[\text{open } k_2.P \mid k_2[Q]]$

$\longrightarrow (\boldsymbol{\nu} w)w[P \mid Q]$

Reduction: communication

(Comm) $\quad (x)P \mid \langle M\rangle \longrightarrow P\{M/x\}$

- local, asynchronous and anonymous
- Remote communication must be encoded.
  - To realise:

$$a[\,\text{send } M \text{ to } b \mid P\,] \mid b[\,\text{get}(x) \text{ from } a.Q \mid R\,]$$

we use a 'taxi' ambient to transport a packet with M:

$$(\boldsymbol{\nu} n)(a[\,n[\,\text{out } a.\text{in } b \mid \langle M\rangle\,] \mid P\,] \mid b[\,\text{open } n \mid (x)Q \mid R\,])$$
$$\longrightarrow (\boldsymbol{\nu} n)(a[\,P\,] \mid n[\,\text{in } b \mid \langle M\rangle\,] \mid b[\,\text{open } n \mid (x)Q \mid R\,])$$
$$\longrightarrow (\boldsymbol{\nu} n)(a[\,P\,] \mid b[\,\text{open } n \mid n[\,\langle M\rangle\,] \mid (x)Q \mid R\,])$$
$$\longrightarrow a[\,P\,] \mid b[\,\langle M\rangle \mid (x)Q \mid R\,]$$
$$\longrightarrow a[\,P\,] \mid b[\,Q\{M/x\} \mid R\,]$$

Reduction: structural rules

No surprises ...

(Par) $\dfrac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}$ (New) $\dfrac{P \longrightarrow P'}{(\boldsymbol{\nu} a : W)P \longrightarrow (\boldsymbol{\nu} a : W)P'}$

(Amb) $\dfrac{P \longrightarrow P'}{a[\,P\,] \longrightarrow a[\,P'\,]}$ (Cong) $\dfrac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q}$

... but note that processes inside ambients reduce

- ambients are active while moving

Types: overview

- Idea: ambients are places of conversation
- multiple processes within an ambient can freely execute input and output actions:
- since messages are not directed to a particular channel, it is possible for a process to output a message that is not appropriate for the receivers active inside the current ambient.

$$(x : W)x[\,P\,] \mid \langle \text{in } n\rangle$$

- type systems must statically detect such errors, keeping track of the

topic of conversation

appropriate within a given ambient

Exchange Types

Expression Types

W	$::=$	B	type constants
	$\mid$	Amb$[E]$	ambient: allow E exchanges
	$\mid$	Cap$[E]$	capability: when exercised unleash E

Exchange Types

E, F	$::=$	Shh	no exchange
	$\mid$	$(W_1, \ldots, W_n)$	tuple $n \geq 0$

Process Types

T	$::=$	$[E]$	process exchanging E

Typing Rules

Type Judgements

- $\Gamma \vdash M : W$ expression M has exchange type W;
- $\Gamma \vdash P : T$ process P has type T.

Note: processes have types, describing effects (i.e. their exchanges)

Typing Processes I

Input-Output

- $$\frac{\Gamma, x : W \vdash P : [W]}{\Gamma \vdash (x : W)P : [W]}$$
  - if $x : W$, and P also exchanges W, then $(x : W).P$ exchanges W.
  - Only one topic of conversation.
- $$\frac{\Gamma \vdash M : W}{\Gamma \vdash \langle M \rangle : [W]}$$
  - M is an expression of type W: hence a process that outputs M exchanges W, i.e. has type $[W]$.

Typing Processes II

Ambients

- $$\frac{\Gamma \vdash a : \text{Amb}[E] \quad \Gamma \vdash P : [E]}{\Gamma \vdash a[P] : T}$$
  - an ambient exchanges nothing at its own level, hence it may be thought of as potentially exchanging any type.
  - Type safety requires consistency between the type that a declares and the type of exchanges held by P inside a.
  - We can check that processes inside that ambient behave consistently.
  - Further consequence: can tell what types are exchanged with ambient is opened ... useful for typing open

Typing Expressions I

Open

- $$\frac{\Gamma \vdash M : \mathsf{Amb}[E]}{\Gamma \vdash \mathsf{open}\ M : \mathsf{Cap}[E]}$$
  - if $n : \mathsf{Amb}[E]$, opening n unleashes processes enclosed in n, which have exchanges of type E.
  - by our intended interpretation of $\mathsf{Cap}[E]$, this implies $\mathsf{open}\ n : \mathsf{Cap}[E]$.
  - Further consequence: process exercises $\mathsf{open}\ n : \mathsf{Cap}[E]$ may end-up running in parallel with processes which exchange E.
  - any such process must be itself prepared to exchange E that types ... useful for typing $M.P$

Typing Expressions II

In and Out

- $$\frac{\Gamma \vdash M : \mathsf{Amb}[F]}{\Gamma \vdash \mathsf{in}\ M : \mathsf{Cap}[E]}$$
- $$\frac{\Gamma \vdash M : \mathsf{Amb}[F]}{\Gamma \vdash \mathsf{out}\ M : \mathsf{Cap}[E]}$$
- need not track type information
- all the required consistency checks on the local exchanges within an ambient are already accounted for by the typing of open.

Typing Processes III

- (Prefix) $$\frac{\Gamma \vdash M : \mathsf{Cap}[E] \quad \Gamma \vdash P : [E]}{\Gamma \vdash M.P : [E]}$$
  Explained earlier
- (Parallel) $$\frac{\Gamma \vdash P : T \quad \Gamma \vdash Q : T}{\Gamma \vdash P \mid Q : T}$$
- Remember: just one topic of conversation
- Remaining rules present no difficulties

Mobility Types

- Exchange types are effective in imposing a discipline on the exchanges within ambients.
- but communication is just one aspect of the computation of the ambient calculus, and not the core one.
- basic properties of mobile ambients are those related to their mobility, to the possibility of opening an ambient and exposing its contents,
- want a type system to control/characterize these aspects of ambient behavior.

Boxing Ambients

- The open capability is
  - essential for communication, but
  - potentially dangerous

$$a[\,\text{in safe\_amb}.\text{NastyCode}\,] \mid \text{safe\_amb}[\,\text{open } a.Q\,]$$
$$\longrightarrow\longrightarrow \text{safe\_amb}[\,\text{NastyCode} \mid Q\,]$$

  - complicates the typing of mobility

- Drop the open capability of Mobile Ambients

- Provide new constructs for communication across boundaries
  - exchanges towards children: $(x)^n.P$, $\langle M\rangle^n.P$
  - exchanges towards parent: $(x)^{\uparrow}.P$, $\langle M\rangle^{\uparrow}.P$

Boxed Ambients: overview

- Mobile Ambients \ open + parent-child I/O

Communication primitives

- $(\vec{x})^\eta P$: input from ambient η
- $\langle M\rangle^\eta P$: output to ambiet η

$\eta =$ the location where the communication happens

- $\eta = \star$: local, as in Mobile Ambients
- $\eta = n$: from parent to child
- $\eta = \uparrow$: from child to parend

Remote communication, between siblings, still requires mobility

Typed Boxed Ambients: Syntax

η	$::=$	n	names
	$\mid XS$	$\uparrow$	enclosing amb
	$\mid XS$	$\star$	local

P	$::=$	$\mathbf{0}$	nil process
	$\mid XS$	$P_1 \mid P_2$	composition
	$\mid XS$	$(\nu n{:}A)P$	restriction
	$\mid XS$	$!P$	replication
	$\mid XS$	$V[\,P\,]$	ambient
	$\mid XS$	$V.P$	prefixing
	$\mid XS$	$(\boldsymbol{x} : \boldsymbol{W})^\eta.P$	input
	$\mid XS$	$\langle \boldsymbol{V}\rangle^\eta.P$	output
	$\mid XS$	$(\nu \mathsf{G})P$	group creation

V, U	$::=$	n	name
	$\mid XS$	$\mathtt{in}\, V$	may enter V
	$\mid XS$	$\mathtt{out}\, V$	may exit V
	$\mid XS$	~~open V~~	~~may open V~~
	$\mid XS$	$V_1.V_2$	path

Reduction Semantics

Mobility:

$$n[\,\mathtt{in}\, m.P|Q\,] \mid m[\,R\,] \rightarrow m[\,n[\,P \mid Q\,] \mid R\,] \qquad \text{(In)}$$

$$m[\,n[\,\mathtt{out}\, m.P \mid Q\,] \mid R\,] \rightarrow n[\,P \mid Q\,] \mid m[\,R\,] \qquad \text{(Out)}$$

Communication:

$$(\boldsymbol{x}).P \mid \langle \boldsymbol{V}\rangle.Q \rightarrow P\{\boldsymbol{V}/\boldsymbol{x}\} \mid Q \qquad \text{(Comm Local)}$$

$$(\boldsymbol{x})^n.P \mid n[\,\langle \boldsymbol{V}\rangle.Q \mid R\,] \rightarrow P\{\boldsymbol{V}/\boldsymbol{x}\} \mid n[\,Q \mid R\,] \qquad \text{(Comm Input } n\text{)}$$

$$(\boldsymbol{x}).P \mid n[\,\langle \boldsymbol{V}\rangle^{\uparrow}.Q \mid R\,] \rightarrow P\{\boldsymbol{V}/\boldsymbol{x}\} \mid n[\,Q|R\,] \qquad \text{(Comm Output } \uparrow\text{)}$$

$$\langle \boldsymbol{V}\rangle^n.P \mid n[\,(\boldsymbol{x}).Q \mid R\,] \rightarrow P \mid n[\,Q\{\boldsymbol{V}/\boldsymbol{x}\} \mid R\,] \qquad \text{(Comm Output } n\text{)}$$

$$\langle \boldsymbol{V}\rangle.P \mid n[\,(\boldsymbol{x})^{\uparrow}.Q \mid R\,] \rightarrow P \mid n[\,Q\{\boldsymbol{V}/\boldsymbol{x}\} \mid R\,] \qquad \text{(Comm Input } \uparrow\text{)}$$

Discussion

- Reductions yield clear notions of resources and access requests
  - each ambient has an allocated resource, its local (anonymous) channel;
  - $\langle M \rangle^\eta \approx$ write access to the ambient η towards which the request is directed (equivalently, its anonymous channel).
  - $(x)^\eta \approx$ read access
- Resource access control policies expressed easily and directly
- Static analysis, by typing, eased and more accurate, given the absence of open

Still, we will criticise it tomorrow. For now let us look at access control.

Group Types for Mobility

Aim: *Resource Access Control*

- Detect and prevent unwanted access to resources.
- We focus on static approaches based on enforcing type disciplines.

Groups: Sets of processes with common access rights. (Cardelli-Ghelli-Gordon)

Constraints like k : CanEnter(n) are modelled as:

n belongs to group G

k may cross the border of any ambient of group G.

For instance, the system:

$$k[\,\mathtt{in}\, n \mid l[\,\mathtt{out}\, k\,]\,] \mid n[\,\_\,]$$

is well-typed under assumptions of the form:

$$k : \mathrm{amb}[\mathtt{K}, \mathrm{cross}(\mathtt{N})]$$

$$l : \mathrm{amb}[\mathtt{L}, \mathrm{cross}(\mathtt{K})] \qquad n : \mathrm{amb}[\mathtt{N}, \ldots]$$

Indirect Border Crossing in MA

Troyan Horses: The system

$$\mathrm{Odysseus}[\,\mathtt{in}\,\mathrm{Horse}.\mathtt{out}\,\mathrm{Horse}.\mathit{Destroy}\,] \mid \mathrm{Horse}[\,\mathtt{in}\,\mathrm{Troy}\,] \mid \mathrm{Troy}[\,\mathit{Trojans}\,]$$

is well-typed under assumptions:

$$\mathrm{Odysseus} : \mathrm{amb}[\mathtt{Achaean}, \mathrm{cross}(\mathtt{Toy})]$$

$$\mathrm{Horse} : \mathrm{amb}[\mathtt{Toy}, \mathrm{cross}(\mathtt{City})]$$

$$\mathrm{Troy} : \mathrm{amb}[\mathtt{City}, \_]$$

However, the system may evolve to

$$\mathrm{Troy}[\,\mathit{Trojans} \mid \mathrm{Horse}[\,\_\,] \mid \mathrm{Odysseus}[\,\mathit{Destroy}\,]\,]$$

where Odysseus got inside Troy's Walls taking by surprise the *Trojans*.

Types

Groups: $\mathtt{G}, \mathtt{H}, \ldots$

Sets of groups: $\mathscr{G}, \mathscr{D}, \mathscr{S}, \ldots$ $\quad \mathscr{U}$ The universal set of groups

Ambients types:

$A \;::=\; \mathrm{amb}_\chi[\mathtt{G}, M, C]$ — amb of group G, good for actions $\chi \subseteq \{\mathtt{i}, \mathtt{o}, \mathtt{c}, \mathtt{r}, \mathtt{w}\}$, with mobility type M, and communication type C

Process types:

$\Pi \;::=\; \mathrm{proc}[\mathtt{G}, M, C]$ — process that can be enclosed in an ambient of group G, may drive to ambients whose groups are in M, and communicates as described by type C

Capability types:

$K \;::=\; \mathrm{cap}[\mathtt{G}, M, F]$ — capability that can appear in an ambient of group G, may drive it to ambients whose groups are in M, with exchange type F for local communication

Types (cont.)

Mobility types:

$M \quad ::= \quad \mathsf{mob}[\mathscr{G}]$ — mobility specs

Communication types:

$C \quad ::= \quad \mathsf{com}[E, F]$ — E for local and F for upward e

Exchange types:

$E, F \quad ::= \quad \mathsf{rw}[I, O]$ — read/write values (valid if $O \prec$

Message types:

$I, O \quad ::= \quad \bot \;\mid XS\;\; W_1 \times \ldots \times W_k \;\mid XS\;\; \top$ — bottom, tuple, top

Value types:

$W, Y \quad ::= \quad A$ — ambient name

$\mid XS \quad K$ — capability

Subtyping

(sAmb)
$$\frac{\chi_1 \subseteq \chi_0 \subseteq \{\mathtt{i}, \mathtt{o}, \mathtt{c}, \mathtt{r}, \mathtt{w}\}}{\mathsf{amb}_{\chi_0}[\mathtt{G}, M, C] \prec \mathsf{amb}_{\chi_1}[\mathtt{G}, M, C]}$$

(sProc)
$$\frac{M_0 \prec M_1; \quad C_0 \prec C_1}{\mathsf{proc}[\mathtt{G}, M_0, C_0] \prec \mathsf{proc}[\mathtt{G}, M_1, C_1]}$$

(sCap)
$$\frac{M_0 \prec M_1; \quad F_0 \prec F_1}{\mathsf{cap}[\mathtt{G}, M_0, F_0] \prec \mathsf{cap}[\mathtt{G}, M_1, F_1]}$$

(sMob)
$$\frac{\mathscr{G}_0 \subseteq \mathscr{G}_1}{\mathsf{mob}[\mathscr{G}_0] \prec \mathsf{mob}[\mathscr{G}_1]}$$

(sCom)
$$\frac{E_0 \prec E_1; \quad F_0 \prec F_1}{\mathsf{com}[E_0, F_0] \prec \mathsf{com}[E_1, F_1]}$$

(sExc)
$$\frac{I_1 \prec I_0; \quad O_0 \prec O_1}{\mathsf{rw}[I_0, O_0] \prec \mathsf{rw}[I_1, O_1]}$$

(sMsg)
$$\frac{-}{\bot \prec W_1 \times \ldots \times W_k \prec \top}$$

(sTuple)
$$\frac{W_i \prec T_i; \quad i \in 1..k}{W_1 \times \ldots \times W_k \prec T_1 \times \ldots \times T_k}$$

Good Values

(Val n)
$$\frac{\Gamma, n : W, \Gamma' \vdash \diamond}{\Gamma, n : W, \Gamma' \vdash n : W}$$

(Val pfx)
$$\frac{\Gamma \vdash V_0 : K; \quad \Gamma \vdash V_1 : K}{\Gamma \vdash V_0.V_1 : K}$$

(Val in)
$$\frac{\Gamma \vdash V : \mathsf{amb}_{\mathtt{i}}[\mathtt{G}, M, \mathsf{com}[E, F]]}{\Gamma \vdash \mathbf{in}\, V : \mathsf{cap}[\mathtt{H}, \mathsf{mob}[\{\mathtt{G}\}], E]} \; \mathtt{H} \in \Gamma$$

(Val sub)
$$\frac{\Gamma \vdash V : W; \quad W \prec W'}{\Gamma \vdash V : W'}$$

(Val out)
$$\frac{\Gamma \vdash V : \mathsf{amb}_{\mathtt{o}}[\mathtt{G}, M, \mathsf{com}[E, F]]}{\Gamma \vdash \mathbf{out}\, V : \mathsf{cap}[\mathtt{H}, M, F]} \; \mathtt{H} \in \Gamma$$

Good Processes – Mobility

(Pro pfx)
$$\frac{\Gamma \vdash V{:}\mathsf{cap}[\mathtt{G}, M, F]; \quad \Gamma \vdash P{:}\mathsf{proc}[\mathtt{G}, M, \mathsf{com}[E, F]]}{\Gamma \vdash V.P{:}\mathsf{proc}[\mathtt{G}, M, \mathsf{com}[E, F]]}$$

(Pro amb)
$$\frac{\Gamma \vdash V{:}\mathsf{amb}_{\mathtt{c}}[\mathtt{H}, \mathsf{mob}[\mathscr{S}], \mathsf{com}[E, F]]; \; \Gamma \vdash P{:}\mathsf{proc}[\mathtt{H}, \mathsf{mob}[\mathscr{S}], \mathsf{com}[E, F]]}{\Gamma \vdash V[\,P\,]{:}\mathsf{proc}[\mathtt{G}, \mathsf{mob}[\emptyset], \mathsf{com}[F, \mathsf{zero}]]} \; \mathtt{G} \in \mathscr{S}$$

(Pro res)
$$\frac{\Gamma, n : A \vdash P : \Pi}{\Gamma \vdash (\nu n : A)P : \Pi}$$

(Pro gres)
$$\frac{\Gamma, \mathtt{G} \vdash P : \Pi}{\Gamma \vdash (\nu \mathtt{G})P : \Pi} \quad \mathtt{G} \notin \mathsf{fg}(\Pi)$$

(Pro $\mathbf{0}$)
$$\frac{\mathtt{G} \in \mathsf{dom}(\Gamma)}{\Gamma \vdash \mathbf{0} : \mathsf{proc}[\mathtt{G}, \mathsf{mob}[\emptyset], \mathsf{com}[\mathsf{zero}, \mathsf{zero}]]}$$

(Pro par)
$$\frac{\Gamma \vdash P : \Pi; \quad \Gamma \vdash Q : \Pi}{\Gamma \vdash P \mid Q : \Pi}$$

(Pro rep)
$$\frac{\Gamma \vdash P : \Pi}{\Gamma \vdash !P : \Pi}$$

(Pro sub)
$$\frac{\Gamma \vdash P : \Pi \quad \Pi \prec \Pi'}{\Gamma \vdash P : \Pi'}$$

Good Processes – Communication

(inp $\star$)
$$\frac{\Gamma, \boldsymbol{x} : \boldsymbol{W} \vdash P : \mathsf{proc}[\mathtt{G}, M, \mathsf{com}[\mathsf{rw}[I, O], F]]}{\Gamma \vdash (\boldsymbol{x} : \boldsymbol{W}).P : \mathsf{proc}[\mathtt{G}, M, \mathsf{com}[\mathsf{rw}[I, O], F]]}\ I \prec \boldsymbol{W}$$

(out $\star$)
$$\frac{\Gamma \vdash \boldsymbol{V} : \boldsymbol{W}; \quad \Gamma \vdash P : \mathsf{proc}[\mathtt{G}, M, \mathsf{com}[\mathsf{rw}[I, \boldsymbol{W}], F]]}{\Gamma \vdash \langle \boldsymbol{V} \rangle.P : \mathsf{proc}[\mathtt{G}, M, \mathsf{com}[\mathsf{rw}[I, \boldsymbol{W}], F]]}$$

(inp $\uparrow$)
$$\frac{\Gamma, \boldsymbol{x} : \boldsymbol{W} \vdash P : \mathsf{proc}[\mathtt{G}, M, \mathsf{com}[E, \mathsf{rw}[I, O]]]}{\Gamma \vdash (\boldsymbol{x} : \boldsymbol{W}_k)^{\uparrow}.P : \mathsf{proc}[\mathtt{G}, M, \mathsf{com}[E, \mathsf{rw}[I, O]]]}\ I \prec \boldsymbol{W}$$

(output $\uparrow$)
$$\frac{\Gamma \vdash \boldsymbol{V} : \boldsymbol{W}; \quad \Gamma \vdash P : \mathsf{proc}[\mathtt{G}, M, \mathsf{com}[E, \mathsf{rw}[I, \boldsymbol{W}]]]}{\Gamma \vdash \langle \boldsymbol{V} \rangle^{\uparrow}.P : \mathsf{proc}[\mathtt{G}, M, \mathsf{com}[E, \mathsf{rw}[I, \boldsymbol{W}]]]}$$

...and so on analogously

Properties

Communication properties:

- If $\Gamma \vdash (\boldsymbol{x} : \boldsymbol{W}).P \mid \langle \boldsymbol{V} \rangle.Q : \Pi$ then $\Gamma \vdash \boldsymbol{V} : \boldsymbol{Y}$ with $\boldsymbol{Y} \prec \boldsymbol{W}$.

Mobility properties:

- If $\Gamma \vdash n[\,\mathtt{in}\, m.P \mid Q\,] \mid m[\,R\,] : \Pi$, then
$$\Gamma \vdash m : \mathsf{amb}_{\chi_0}[\mathtt{M}, \_, \_] \quad \text{and} \quad \Gamma \vdash n : \mathsf{amb}_{\chi_1}[\_, \mathsf{mob}[\mathscr{S}], \_]$$
with $\mathtt{M} \in \mathscr{S}$, $\mathtt{i}, \mathtt{c} \in \chi_0$ and $\mathtt{c} \in \chi_1$.
- If $\Gamma \vdash m[\,n[\,\mathtt{out}\, m.P \mid Q\,] \mid R\,] : \Pi$, then
$$\Gamma \vdash m : \mathsf{amb}_{\chi_0}[\mathtt{M}, \mathsf{mob}[\mathscr{S}_m], \_] \text{ and } \Gamma \vdash n : \mathsf{amb}_{\chi_1}[\mathtt{N}, \mathsf{mob}[\mathscr{S}_n], \_]$$
with $\mathtt{o}, \mathtt{c} \in \chi_0$, $\mathtt{c} \in \chi_1$, $\mathtt{M} \in \mathscr{S}_n$ and $\mathscr{S}_m \subseteq \mathscr{S}_n$.

Subject reduction:

- If $\Gamma \vdash P : \Pi$ and $P \equiv Q$ or $P \to Q$, then there exist groups $\mathtt{G}_0, \ldots \mathtt{G}_k$ such that $\mathtt{G}_0, \ldots, \mathtt{G}_k, \Gamma \vdash Q : \Pi$.

Detecting Odysseus' intentions

Now, in order to assign a type to

$$\mathsf{Odysseus}[\,\mathtt{in}\,\mathsf{Horse}.\mathtt{out}\,\mathsf{Horse}.\mathit{Destroy}\,] \mid \mathsf{Horse}[\,\mathtt{in}\,\mathsf{Troy}\,] \mid \mathsf{Troy}[\,\mathit{Trojans}\,]$$

we need assumptions of the form:

$$\mathsf{Odysseus} : \mathsf{amb}_{\mathtt{c}}[\mathtt{Achaean}, \mathsf{mob}[\{\mathtt{Ground}, \mathtt{Toy}, \mathtt{City}\}], \_]$$
$$\mathsf{Horse} : \mathsf{amb}_{\mathtt{ioc}}[\mathtt{Toy}, \mathsf{mob}[\{\mathtt{Ground}, \mathtt{City}\}], \_]$$
$$\mathsf{Troy} : \mathsf{amb}_{\mathtt{ioc}}[\mathtt{City}, \_, \_]$$

representing that Odysseus is an $\mathtt{Achaean}$ intentioned to move into a $\mathtt{City}$!
On the other hand, under assumptions of the form

$$\mathsf{Odysseus} : \mathsf{amb}_{\mathtt{c}}[\mathtt{Achaean}, \mathsf{mob}[\{\mathtt{Ground}, \mathtt{Toy}\}], \_]$$

the *Trojans* should not fear any attack from Odysseus.

But what if Odysseus is lying about his intentions (i.e. type)?

Adding co-capabilities

Reduction Semantics:

$$n[\,\mathtt{in}\, m.P \mid Q\,] \mid m[\,\overline{\mathtt{in}}\,\alpha.R \mid S\,] \to m[\,n[\,P \mid Q\,] \mid R \mid S\,] \quad \text{for } \alpha \in \{\star, n\}$$

$$m[\,n[\,\mathtt{out}\, m.P \mid Q\,] \mid R\,] \mid \overline{\mathtt{out}}\,\alpha.S \to n[\,P \mid Q\,] \mid m[\,R\,] \mid S \quad \text{for } \alpha \in \{\star, n\}$$

Mobility Types: (extended: $\mathscr{C}$ tells which processes are allowed in.)

$$M \quad ::= \quad \mathsf{mob}[\mathscr{S}, \mathscr{C}]$$

Subtyping Relation: (extended)

(sMob)
$$\frac{\mathscr{G}_0 \subseteq \mathscr{G}_1, \quad \mathscr{C}_0 \subseteq \mathscr{C}_1}{\mathsf{mob}[\mathscr{G}_0, \mathscr{C}_0] \prec \mathsf{mob}[\mathscr{G}_1, \mathscr{C}_1]}$$

Good Values in BSA

(Val in)
$$\frac{\Gamma \vdash V{:}\mathsf{amb_i}[\mathtt{G}, M, \mathsf{com}[E,F]]}{\Gamma \vdash \mathbf{in}\, V{:}\mathsf{cap}[\mathtt{H}, \mathsf{mob}[\{G\}, \emptyset], E]}$$

(Val out)
$$\frac{\Gamma \vdash V{:}\mathsf{amb_o}[\mathtt{G}, \mathsf{mob}[\mathscr{S}, \mathscr{C}], \mathsf{com}[E,F]]}{\Gamma \vdash \mathbf{out}\, V{:}\mathsf{cap}[\mathtt{H}, \mathsf{mob}[\mathscr{S}, \emptyset], F]}$$

(Val coin)
$$\frac{\Gamma \vdash V{:}\mathsf{amb_i}[\mathtt{G}, M, \mathsf{com}[E,F]]}{\Gamma \vdash \overline{\mathbf{in}}\, V{:}\mathsf{cap}[\mathtt{H}, \mathsf{mob}[\emptyset, \{\mathtt{G}\}], F]}$$

(Val coout)
$$\frac{\Gamma \vdash V{:}\mathsf{amb_o}[\mathtt{G}, M, \mathsf{com}[E,F]]}{\Gamma \vdash \overline{\mathbf{out}}\, V{:}\mathsf{cap}[\mathtt{H}, \mathsf{mob}[\emptyset, \{\mathtt{G}\}], F]}$$

(Val coin $\star$)
$$\frac{\mathtt{G} \in \mathsf{dom}(\Gamma)}{\Gamma \vdash \overline{\mathbf{in}}\, \star{:}\mathsf{cap}[\mathtt{G}, \mathsf{mob}[\emptyset, \mathscr{U}], \mathsf{zero}]}$$

(Val coout $\star$)
$$\frac{\mathtt{G} \in \mathsf{dom}(\Gamma)}{\Gamma \vdash \overline{\mathbf{out}}\, \star{:}\mathsf{cap}[\mathtt{G}, \mathsf{mob}[\emptyset, \mathscr{U}], \mathsf{zero}]}$$

In (Val in), (Val out), (Val coin), (Val coin), assume $\mathtt{H} \in \Gamma$

Good Processes – Mobility in BSA

(Pro $\mathbf{0}$)
$$\frac{\mathtt{G} \in \mathsf{dom}(\Gamma)}{\Gamma \vdash \mathbf{0}{:}\mathsf{proc}[\mathtt{G}, \mathsf{mob}[\emptyset, \emptyset], \mathsf{com}[\mathsf{zero}, \mathsf{zero}]]}$$

(Pro pfx)
$$\frac{\Gamma \vdash V{:}\mathsf{cap}[\mathtt{G}, M, F]; \quad \Gamma \vdash P{:}\mathsf{proc}[\mathtt{G}, M, \mathsf{com}[E,F]]}{\Gamma \vdash V.P{:}\mathsf{proc}[\mathtt{G}, M, \mathsf{com}[E,F]]}$$

(Pro amb)
$$\frac{\begin{array}{c}\Gamma \vdash V{:}\mathsf{amb_c}[\mathtt{H}, \mathsf{mob}[\mathscr{S}, \mathscr{C}], \mathsf{com}[E,F]] \\ \Gamma \vdash P{:}\mathsf{proc}[\mathtt{H}, \mathsf{mob}[\mathscr{S}, \mathscr{C}], \mathsf{com}[E,F]]\end{array}}{\Gamma \vdash V[\,P\,]{:}\mathsf{proc}[\mathtt{G}, \mathsf{mob}[\emptyset, \{\mathtt{H}\}], \mathsf{com}[F, \mathsf{zero}]]}\ \mathtt{G} \in \mathscr{S}$$

Control Properties in BSA

Access Control Theorem:

Whenever

$$\Gamma \vdash m[\overline{\mathbf{in}}\,\alpha.P \mid Q] : \Pi \quad \text{or} \quad \Gamma \vdash m[\overline{\mathbf{out}}\,\alpha.P \mid Q] : \Pi,$$

with $\alpha \in \{\star, n\}$, then

- $\Gamma \vdash m : \mathsf{amb}_{\chi_0}[\_, \mathsf{mob}[\_, \mathscr{C}], \_]$, and
  - either $\alpha = \star$ and $\mathscr{C} = \mathscr{U}$,
  - or $\alpha = n$ with $\Gamma \vdash n : \mathsf{amb}_{\chi_1}[\mathtt{N}, \_, \_]$ and $\mathtt{N} \in \mathscr{C}$.

Using co-capabilities to defend Troy

Our running example in BSA:

$$\begin{array}{rl} \mathit{The\ Trojan\ War} \triangleq & \mathsf{Odysseus}[\,\mathbf{in}\,\mathsf{Horse}.\mathbf{out}\,\mathsf{Horse}.\mathit{Destroy}\,] \\ & \mid \mathsf{Horse}[\,\overline{\mathbf{in}}\,\star.\mathbf{in}\,\mathsf{Troy}\,] \\ & \mid \mathsf{Troy}[\,\overline{\mathbf{in}}\,\mathsf{Horse}.\mathit{Trojans} \mid \overline{\mathbf{out}}\,\mathsf{Odysseus}.\mathit{Sinon}\,] \end{array}$$

which can be well-typed only if

$$\Gamma \vdash \mathsf{Troy} : \mathsf{amb_{ioc}}[\mathtt{City}, \mathsf{mob}[\_, \{\mathtt{Toy}, \mathtt{Achaean}\}], \_]$$

That is if Troy (in suicidal mood) allowed Achaeans in.

Using co-capabilities to defend Troy (ctd)

Consider now the system:

$$\mathit{The\ Trojan\ Trap} \triangleq \mathsf{Odysseus}[\,\mathtt{in}\,\mathsf{Horse}.\mathtt{out}\,\mathsf{Horse}.\mathit{Destroy}\,]$$
$$\mid \mathsf{Horse}[\,\overline{\mathtt{in}}\ \star.\mathtt{in}\,\mathsf{Troy}\,]$$
$$\mid \mathsf{Troy}[\,\overline{\mathtt{in}}\,\mathsf{Horse}.\mathit{Trojans}\,]$$

This situation would be perfectly safe for Troy (but dangerous for Odysseus!) provided we can type it under the assumptions of the form

$$\mathsf{Odysseus} : \mathsf{amb}_{\mathtt{c}}[\mathtt{Achaean}, \_, \_]$$
$$\mathsf{Horse} : \mathsf{amb}_{\mathtt{ioc}}[\mathtt{Toy}, \_, \mathsf{com}[E, \mathbf{0}]]$$
$$\mathsf{Troy} : \mathsf{amb}_{\mathtt{ioc}}[\mathtt{City}, \mathsf{mob}[\emptyset, \mathscr{C}], \_]$$

with $\mathtt{Achaean} \notin \mathscr{C}$.

Summary of Lecture IV

We focused on ambient mobility, introducing ambients and their exchange types, boxed ambients and their mobility types.

Further Reading

Mobile Ambients, a "hot topic": lots of papers. References for this lecture are:

- Mobile ambients (Cardelli, Gordon)
- Mobility types (Cardelli, Ghelli, Gordon)
- Polymorphic Typing (Amtoft, Kfoury, Pericas)
- Safe Ambients (Levi, Sangiorgi)
- Boxed Ambients (Bugliesi, Crafa, Castagna)
- Typing and Subtyping Mobility (Merro, Sassone)
- . . . a long list, . . .

Global Computing

— Lecture V —

Towards Ambient Resource Control

———

The Case for Resource Usage Control

Global Computing involve scenarios where mobile devices enter and exit domains and networks.

Typical Devices:

Today: Smart Cards, Embedded devs (e.g. in cars), Mobile phones, PDAs, Sat navigators, . . .
Tomorrow: PAN, VAN, D-ME, P-COM, . . .

Requirements:

- Security: Authentication, Privacy, Non Repudiation
- Trust Formation and Management
- Context (e.g. Location) Awareness
- Dynamic Learning and Adaptability
- Policies of Access Control and their Enforcement
- Negotiation of Access, Access Rights, Resource Acquisition
- Protection of Resource Bounds . . .

Central Notion: Resource Usage

Roadmap for Lecture V

- Control of Interference in Mobile Ambients: manageable vs expressive: how do you want your calculus?
- Secrecy in Untrusted Networks: crypto-primitives for mobile agents
- Ambients with Bounded Capacity: make realistic hypothesis on ambient capacities and processes' space consumption.

Interferences in Mobile Ambients

- The inherent nondeterminism of movement may go wild: Grave Interferences.

$$k[\ n[\ \mathsf{in}\ m.P \mid \mathsf{out}\ k.R\ ] \mid m[\ Q\ ]\ ]$$

- Introducing Safe Ambients

$$n[\ \mathsf{in}\ m.P \mid Q\ ] \mid m[\ \overline{\mathsf{in}}\ m.R \mid S\ ] \longrightarrow m[\ n[\ P \mid Q\ ] \mid R \mid S\ ]$$

- Co-capabilities and single-threadedness rule out grave interferences

Mobile Boxed Ambients

- open's nature of ambient dissolver is a potential source of problems.
- Direct communication as alternative source of expressiveness: Mobile Boxed Ambients. Perform I/O on a subambient n's local channel (viz. $(x)^n$) as well as from the parent's local channel (viz. $(x)^\uparrow$)

$$(x)^n.P \mid n[\ \langle M\rangle.Q \mid R\ ] \longrightarrow P\{M/x\} \mid n[\ Q \mid R\ ]$$

$$\langle M\rangle.P \mid n[\ (x)^\uparrow.Q \mid R\ ] \longrightarrow P \mid n[\ Q\{M/x\} \mid R\ ].$$

- But it is a great source of non-local nondeterminism and communication interference.

$$m[\ (x)^n.P \mid n[\ \langle M\rangle \mid (x).Q \mid k[\ (x)^\uparrow.R\ ]\ ]\ ]$$

Introducing NBA: Communication

NBA: a fresh foundation based on: each ambient comes equipped with two mutually non-interfering channels, for local and upward communications.

$$(x)^n.P \mid n[\langle M\rangle^{\hat{}}.Q \mid R\ ] \longrightarrow P\{M/x\} \mid n[\ Q \mid R\ ]$$

$$\langle M\rangle^n.P \mid n[(x)^{\hat{}}.Q \mid R\ ] \longrightarrow P \mid n[\ Q\{M/x\} \mid R\ ]$$

- Expressiveness??
- Hmm, rather poor: $n[P]$ cannot, for instance, communicate with children it doesn't know statically. It can never learn about incoming ambients, and will never be able to talk to them.

Introducing NBA: Mobility

- Let us introduce co-actions of the form $\overline{\mathsf{enter}}(x)$ which have the effect of binding the variable x.

- Such a purely binding mechanism does not provide a way control of access, but only to registers it. As a (realistic) access protocol where newly arrived agents must register themselves to be granted access to local resources.

- Need a finer mechanism of access control:

$$a[\ \mathsf{enter}\langle b, k\rangle.P \mid R\ ] \mid b[\ \overline{\mathsf{enter}}(x, k).Q \mid S\ ] \longrightarrow b[\ a[\ P \mid R\ ] \mid Q\{a/x\} \mid S\ ]$$

This represent an access protocol where the credentials of incoming processes (k in the rule above) are controlled, as a preliminary step to the registration protocol.

NBA: Syntax

Names : $a, b, \ldots n, x, y, \ldots \in \mathbf{N}$

Locations:

η	$::=$	a	nested names
	$\mid XS$	$\hat{}$	enclosing amb
	$\mid XS$	$\star$	local

Messages:

M, N	$::=$	a	name
	$\mid XS$	$\mathsf{enter}\langle M, N\rangle$	may ent
	$\mid XS$	$\mathsf{exit}\langle M, N\rangle$	may exit
	$\mid XS$	$M.N$	path

Processes:

P	$::=$	$\mathbf{0}$	nil process
	$\mid XS$	$P_1 \mid P_2$	composition
	$\mid XS$	$(\boldsymbol{\nu} n)P$	restriction
	$\mid XS$	$!\pi.P$	replication
	$\mid XS$	$M[\,P\,]$	ambient
	$\mid XS$	$\pi.P$	prefixing

Prefixes:

π	$::=$	M	message
	$\mid XS$	$(x_1, \ldots, x_k)^\eta$	input
	$\mid XS$	$\langle M_1, \ldots, M_k\rangle^\eta$	output
	$\mid XS$	$\overline{\mathsf{enter}}(x, M)$	allow ent
	$\mid XS$	$\overline{\mathsf{exit}}(x, M)$	allow exi

NBA: Reduction Semantics

mobility

$$n[\,\mathsf{enter}\langle m, k\rangle.P \mid R\,] \mid m[\,\overline{\mathsf{enter}}(x, k).Q \mid S\,] \longrightarrow m[\,n[\,P \mid R\,] \mid Q\{n/x\} \mid S\,]$$

$$n[\,m[\,\mathsf{exit}\langle n, k\rangle.P \mid R\,] \mid S\,] \mid \overline{\mathsf{exit}}(x, k).Q \longrightarrow m[\,P \mid R\,] \mid n[\,S\,] \mid Q\{m/x\}$$

communication

$$(\vec{x}).P \mid \langle\vec{M}\rangle.Q \longrightarrow P\{\vec{M}/\vec{x}\} \mid Q$$

$$(\vec{x})^n.P \mid n[\,\langle\vec{M}\rangle^{\hat{}}.Q \mid R\,] \longrightarrow P\{\vec{M}/\vec{x}\} \mid n[\,Q \mid R\,]$$

$$\langle\vec{M}\rangle^n.P \mid n[\,(\vec{x})^{\hat{}}.Q \mid R\,] \longrightarrow P \mid n[\,Q\{\vec{M}/\vec{x}\} \mid R\,]$$

structural congruence

$$P \equiv Q,\ \ Q \longrightarrow R,\ \ R \equiv S \Longrightarrow P \longrightarrow S$$

A one-to-one communication server

- Let $\mathsf{w}(k)$ be a bidirectional forwarder for any pair of incoming ambients.

$$\mathsf{w}(k) \triangleq \mathsf{w}[\ \overline{\mathsf{enter}}(x, k).\overline{\mathsf{enter}}(y, k).(!(z)^x.\langle z\rangle^y \mid !(z)^y.\langle z\rangle^x)\ \ ]$$

An agent can be defined as: $A(a, k, P, Q) \triangleq a[\,\mathsf{enter}\langle \mathsf{w}, k\rangle.P \mid \mathsf{exit}\langle \mathsf{w}, k\rangle.Q\,]$ and a communication server as:

$$\mathsf{o2o}(k) \ \ = \ \ (\boldsymbol{\nu} r)\ (\ r[\,\langle\,\rangle^{\hat{}}\ ] \mid !\,(\,)^r.(\mathsf{w}(k) \mid \overline{\mathsf{exit}}(\_, k).\overline{\mathsf{exit}}(\_, k).r[\,\langle\,\rangle^{\hat{}}\ ])\ )$$

- It can be proved that once two agents engage in communication no other agent knowing the key k can interfere with their completing the exchange. In formulas:

$$(\boldsymbol{\nu} k)(\ \mathsf{o2o}(k) \mid A(k, a_1, \langle M\rangle^{\hat{}}.P_1, Q_1) \mid A(k, a_2, (x)^{\hat{}}.P_2\{x\}, Q_2) \mid \Pi_{i\in I} A(K, a_i, R_i, S_i)\ )$$

$$\Longrightarrow \cong^c\ \ (\boldsymbol{\nu} k)(\ \mathsf{o2o}(k) \mid a_1[\,P_1 \mid Q_1\,] \mid a_2[\,P_1\{M/x\} \mid Q_2\,] \mid \Pi_{i\in I} A(K, a_i, R_i, S_i)\ )$$

A print server

- The following process assigns a progressive number to incoming jobs.

$$\mathsf{enqueue}_k \triangleq (\boldsymbol{\nu} c)\ (\ c[\ \langle 1\rangle^{\hat{}}\ ]\ |\ !(n)^c.\overline{\mathsf{enter}}(x,k).\langle n\rangle^x.c[\ \langle n+1\rangle^{\hat{}}\ ])$$

- We can turn it into a print server (which consumes such numbers).

$$\begin{array}{lll} \mathsf{prtsrv}(k) & \triangleq & k[\ \mathsf{enqueue}_k \mid \mathsf{print}\ ] \\ \mathsf{print} & \triangleq & (\boldsymbol{\nu} c)\ (\ c[\ \langle 1\rangle^{\hat{}}\ ]\ |\ !(n)^c.\overline{\mathsf{exit}}(x,n).(dat)^x.(P\{dat\} \mid c[\ \langle n+1\rangle^{\hat{}}\ ]) \end{array}$$

- A client then acts as:

$$\mathsf{job}(M,k) \triangleq (\boldsymbol{\nu} p)p[\ \mathsf{enter}\langle k,k\rangle.(n)^{\hat{}}.(\boldsymbol{\nu} q)q[\mathsf{exit}\langle p,n\rangle.\langle M\rangle^{\hat{}}\ ]\ ]$$

It enters the server $\mathsf{prtsrv}(k)$ (using enqueue), it is assigned a number that it uses as a password to carry job M to print (which eventually will bind it to dat in P. (Dynamic name discovery and passwords are fundamental here.)

Goodies of NBA over BA

- A good set of equational laws
- A simpler type system
- A sound LTS characterisation of barbed congruence.
- No significant loss of expressive power

Some Equational Laws

Garbage Collection laws

- $l[\ (\vec{x}_i)^n.P \mid (\vec{x}).Q \mid \langle \vec{M}\rangle^m.R\ ]\ \cong^c\ \mathbf{0}$
- $l[\ (\vec{x})^n.P \mid \langle \vec{M}\rangle.P \mid \langle \vec{M}\rangle^m.P\ ]\ \cong^c\ \mathbf{0}$

Communication laws

- $l[\ \langle \tilde{M_0}\rangle^{\hat{}} \mid \langle \tilde{M_1}\rangle^{\hat{}}\ ]\ \cong^c\ l[\langle \tilde{M_0}\rangle^{\hat{}}\ ] \mid l[\langle \tilde{M_1}\rangle^{\hat{}}\ ]$
- $l[(\vec{x}).P \mid \langle \vec{M}\rangle.Q]\ \cong^c\ l[P\{\vec{M}/\vec{x}\} \mid Q]$
- $(\boldsymbol{\nu} l)(\ (\vec{x})^l.P \mid l[\langle \vec{M}\rangle^{\hat{}}.Q]\ )\ \cong^c\ (\boldsymbol{\nu} l)(\ P\{\vec{M}/\vec{x}\} \mid l[Q]\ )$
- $m[(\vec{x})^l.P \mid l[\langle \vec{M}\rangle^{\hat{}}.Q]]\ \cong^c\ m[P\{\vec{M}/\vec{x}\} \mid l[Q]]$

Mobility laws

- $(\boldsymbol{\nu} p)(m[\mathsf{enter}\langle n,p\rangle.P] \mid n[\overline{\mathsf{enter}}(x,p).Q])\ \cong^c\ (\boldsymbol{\nu} p)(n[Q\{m/x\} \mid m[P]])$
- $l[m[\mathsf{enter}\langle n,p\rangle.P] \mid n[\overline{\mathsf{enter}}(x,p).Q]]\ \cong^c\ l[n[Q\{m/x\} \mid m[P]]]$

A Type System for NBA

- Types

Message Types	W	$::=$	$\mathsf{N}[E]$	ambient/password
		$\mid$	$\mathsf{C}[E]$	capability
Exchange Types	E, F	$::=$	Shh	no exchange
		$\mid$	$W_1 \times \ldots \times W_k$	tuples ($k \geq 0$)
Process Types	T	$::=$	$[E, F]$	local/upward exchange

$\mathsf{N}[E]$ types both ambients and passwords; Shh is the silent type; $\mathsf{N}[\mathsf{Shh}]$ is an ambient with no upward exchanges or a password that reveal the visitor's name.

- Type Environments

$$\text{(Env Empty)}\ \frac{}{\varnothing \vdash \diamond} \qquad \text{(Env name)}\ \frac{\Gamma \vdash \diamond \quad a \notin \mathrm{Dom}(\Gamma)}{\Gamma, a : W \vdash \diamond}$$

Typing Rules

➤ Messages

(Projection)
$$\frac{\Gamma, a : W, \Gamma' \vdash \diamond}{\Gamma, a : W, \Gamma' \vdash a : W}$$

(Path)
$$\frac{\Gamma \vdash M_1 : \mathsf{C}[E_1] \quad \Gamma \vdash M_2 : \mathsf{C}[E_2]}{\Gamma \vdash M_1.M_2 : \mathsf{C}[E_1 \sqcup E_2]}$$

(Enter)
$$\frac{\Gamma \vdash M : \mathsf{N}[E] \quad \Gamma \vdash N : \mathsf{N}[F] \quad (F \leqslant G)}{\Gamma \vdash \mathsf{enter}\langle M, N\rangle : \mathsf{C}[G]}$$

(Exit)
$$\frac{\Gamma \vdash M : \mathsf{N}[E] \quad \Gamma \vdash N : \mathsf{N}[F] \quad (F \leqslant G)}{\Gamma \vdash \mathsf{exit}\langle M, N\rangle : \mathsf{C}[G]}$$

➤ Processes

(Par)
$$\frac{\Gamma \vdash P : [E, F] \quad \Gamma \vdash Q : [E, F]}{\Gamma \vdash P \mid Q : [E, F]}$$

(Repl)
$$\frac{\Gamma \vdash P : [E, F]}{\Gamma \vdash !P : [E, F]}$$

(Dead)
$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathbf{0} : [E, F]}$$

(New)
$$\frac{\Gamma, n : \mathsf{N}[G] \vdash P : [E, F]}{\Gamma \vdash (\boldsymbol{\nu} n{:}\mathsf{N}[G])P : [E, F]}$$

Typing Rules: II

➤ Processes: mobility

(Amb)
$$\frac{\Gamma \vdash M : \mathsf{N}[E] \quad \Gamma \vdash P : [F, E]}{\Gamma \vdash M[\,P\,] : [G, H]}$$

(Prefix)
$$\frac{\Gamma \vdash M : \mathsf{C}[F] \quad \Gamma \vdash P : [E, G] \quad (F \leqslant G)}{\Gamma \vdash M.P : [E, G]}$$

(Co-enter)
$$\frac{\Gamma \vdash M : \mathsf{N}[\tilde{W}] \quad \Gamma, x : \mathsf{N}[\tilde{W}] \vdash P : [E, F]}{\Gamma \vdash \overline{\mathsf{enter}}(x, M).P : [E, F]}$$

(Co-exit)
$$\frac{\Gamma \vdash M : \mathsf{N}[\tilde{W}] \quad \Gamma, x : \mathsf{N}[\tilde{W}] \vdash P : [E, F]}{\Gamma \vdash \overline{\mathsf{exit}}(x, M).P : [E, F]}$$

(Co-enter-silent)
$$\frac{\Gamma \vdash M : \mathsf{N}[\mathsf{Shh}] \quad \Gamma \vdash P : [E, F] \quad x \notin \mathsf{fv}(P)}{\Gamma \vdash \overline{\mathsf{enter}}(x, M).P : [E, F]}$$

(Co-exit-silent)
$$\frac{\Gamma \vdash M : \mathsf{N}[\mathsf{Shh}] \quad \Gamma \vdash P : [E, F] \quad x \notin \mathsf{fv}(P)}{\Gamma \vdash \overline{\mathsf{exit}}(x, M).P : [E, F]}$$

Typing Rules: II

➤ Processes: I/O

(Input)
$$\frac{\Gamma, \vec{x}{:}\vec{W} \vdash P : [\vec{W}, E]}{\Gamma \vdash (\vec{x}{:}\vec{W}).P : [\vec{W}, E]}$$

(Input $\hat{}$)
$$\frac{\Gamma, \vec{x}{:}\vec{W} \vdash P : [E, \vec{W}]}{\Gamma \vdash (\vec{x}{:}\vec{W})^{\hat{}}.P : [E, \vec{W}]}$$

(Input M)
$$\frac{\Gamma \vdash M : \mathsf{N}[\vec{W}] \quad \Gamma, \vec{x}{:}\vec{W} \vdash P : [G, H]}{\Gamma \vdash (\vec{x}{:}\vec{W})^{M}.P : [G, H]}$$

(Output)
$$\frac{\Gamma \vdash \vec{M} : \vec{W} \quad \Gamma \vdash P : [\vec{W}, E]}{\Gamma \vdash \langle \vec{M}\rangle.P : [\vec{W}, E]}$$

(Output $\hat{}$)
$$\frac{\Gamma \vdash \vec{M} : \vec{W} \quad \Gamma \vdash P : [E, \vec{W}]}{\Gamma \vdash \langle \vec{M}\rangle^{\hat{}}.P : [E, \vec{W}]}$$

(Output N)
$$\frac{\Gamma \vdash N : \mathsf{N}[\vec{W}] \quad \Gamma \vdash \vec{M} : \vec{W} \quad \Gamma \vdash P : [G, H]}{\Gamma \vdash \langle \vec{M}\rangle^{N}.P : [G, H]}$$

➤ Subject Reduction. If $\Gamma \vdash P : T$ and $P \longrightarrow Q$, then $\Gamma \vdash Q : T$.

Encoding: BA in NBA

We can encode BA into NBA enriched with a focused form of nondeterminism.

$$\begin{aligned}
\{\!|\, P \,|\!\}_n &= \overline{\mathsf{cross}} \mid \langle\!\langle\, P \,\rangle\!\rangle_n \\
\langle\!\langle\, m[\,P\,] \,\rangle\!\rangle_n &= m[\,\{\!|\, P \,|\!\}_m\,] \\
\langle\!\langle\, (x)^a P \,\rangle\!\rangle_n &= (x)^a \,\langle\!\langle\, P \,\rangle\!\rangle_n \\
\langle\!\langle\, (x) P \,\rangle\!\rangle_n &= (x)\,\langle\!\langle\, P \,\rangle\!\rangle_n + (x)^{\hat{}}\,\langle\!\langle\, P \,\rangle\!\rangle_n + \overline{\mathsf{exit}}(y, \mathsf{pw})(x)^y \,\langle\!\langle\, P \,\rangle\!\rangle_n \\
\langle\!\langle\, (x)^{\uparrow} P \,\rangle\!\rangle_n &= (\boldsymbol{\nu} p)p[\,\mathsf{exit}\langle n, \mathsf{pr}\rangle.(x)^{\hat{}}.\mathsf{enter}\langle n, p\rangle.\langle x\rangle^{\hat{}}\,] \mid \overline{\mathsf{enter}}(y, p)(x)^y \,\langle\!\langle\, P \,\rangle\!\rangle_n \\
\langle\!\langle\, \langle M\rangle^a P \,\rangle\!\rangle_n &= \langle M\rangle^a \,\langle\!\langle\, P \,\rangle\!\rangle_n \\
\langle\!\langle\, \langle M\rangle P \,\rangle\!\rangle_n &= \langle M\rangle\,\langle\!\langle\, P \,\rangle\!\rangle_n + \langle M\rangle^{\hat{}}\,\langle\!\langle\, P \,\rangle\!\rangle_n + \overline{\mathsf{exit}}(y, \mathsf{pr})\langle M\rangle^y \,\langle\!\langle\, P \,\rangle\!\rangle_n \\
\langle\!\langle\, \langle M\rangle^{\uparrow} P \,\rangle\!\rangle_n &= (\boldsymbol{\nu} p)p[\,\mathsf{exit}\langle n, \mathsf{pw}\rangle.\langle M\rangle^{\hat{}}.\mathsf{enter}\langle n, p\rangle.\langle\cdot\rangle^{\hat{}}\,] \mid \overline{\mathsf{enter}}(y, p)(_)^y \,\langle\!\langle\, P \,\rangle\!\rangle_n
\end{aligned}$$

where $\overline{\mathsf{cross}} = !\overline{\mathsf{enter}}(x, \mathsf{mv}) \mid !\overline{\mathsf{exit}}(x, \mathsf{mv})$, $\mathsf{in}\ n = \mathsf{enter}\langle n, \mathsf{mv}\rangle$, and $\mathsf{out}\ n = \mathsf{exit}\langle n, \mathsf{mv}\rangle$ and $p, y \notin \mathsf{fn}(P)$.

➤ Thm. The encoding is operationally sound. If P and Q are single-threaded, then it is equationally sound, that is $\{\!|\, P \,|\!\}_n \cong^c \{\!|\, Q \,|\!\}_n$ implies $P \cong^c Q$.

Secrecy in the pi calculus

- exchange messages over private channels

$$(\boldsymbol{\nu} n)(\ \overline{n}\langle m\rangle \mid n(x).P\ )$$

- No third process can
  - discover m by interacting with the process
  - cause a different message to be sent on n
- Or is it?

Secrecy in the spi calculus

Not the spi calculus way, as we saw. In a distributed system

$$(\boldsymbol{\nu} n)(\ \underbrace{\overline{n}\langle m\rangle}_{\text{at site A}} \mid \underbrace{n(x).P}_{\text{at site B}}\ )$$

- Link between A and B may be physically insecure, regardless of the privacy of n
- use private keys to encrypt connections over public channels

$$(\boldsymbol{\nu} n)(\ \overline{p}\langle\{m\}_n\rangle \mid p(y).\mathsf{case}\ y\ \mathsf{of}\ \{x\}_n\ \mathsf{in}\ P\ )$$

  - anybody can read on p
  - only the intended recipients know n and will read m

Secrecy in the Mobile Ambients

Names $\approx$ Cryptokeys: Carrying messages inside private ambients preserves message integrity and privacy. Or, does it?

$$(\boldsymbol{\nu} n)(\ a[\ n[\ \mathsf{out}\ a.\mathsf{in}\ b.\langle M\rangle\ ]\ ] \mid b[\ \mathsf{open}\ n.(x)P\ ]\ )$$

It actually offers poor guarantees, as n must be revealed along the move.

How to provide stronger protection?

- Commit to agents their own security, with co-capabilities

$$(\boldsymbol{\nu} n)(a[\,n[\,\mathsf{out}\ a.\mathsf{in}\ b.\overline{\mathsf{open}}\ n.\langle M\rangle\,]\,] \mid b[\,\overline{\mathsf{in}}\ b.\mathsf{open}\ n.(x)P\,])$$

  No one can open n and read M before n reaches b.

- Protect ambients by encapsulating them

$$(\boldsymbol{\nu} n)(a[\,p[\,\mathsf{out}\ a.\mathsf{in}\ b \mid n[\,\langle M\rangle\,]\,]\,] \mid b[\,\mathsf{open}\ p.\mathsf{open}\ n.(x)P\,])$$

  A public ambient p carries a private ambient n, which need not reveal its name to move.

Secrecy: Need new primitives?

Case I: Physical devices:

- the first proposal is all we need: physical devices can easily perform access control, such as that encompassed by co-capabilities

Case II: Soft agents

- the first proposal is pointless in "untrusted" networks. Similarly, the second proposal presupposes encryption for data and code and applies only partially to active agents, which may not move autonomously when encrypted.

A crypto-primitive: subjective access control using co-capabilities + data encryption to preserve secrecy of data while agents move autonomously

$$n[\,\mathsf{seal}\ k.P \mid Q\,] \ \longrightarrow\ n\{\!|\ P \mid Q\ |\!\}_k$$

sealed under k
crypto-key

Effects:

- blocks message exchanges and encrypts their contents;
- the sealed ambient cannot communicate, but it may move.

Sealed Ambients

➤ The mechanism to resume to a fully operational state is associated to movements and co-capabilities containing keys

$$n\{\!|\, \text{in } m.P \mid Q \,|\!\}_k \mid m\{\, \overline{\text{in}}\ \{x\}_k.R \mid R' \,\} \longrightarrow m\{\, n[\,P \mid Q\,] \mid R\{n/x\} \mid R' \,\}$$

Example:

$$(\boldsymbol{\nu} k)a[\ n[\,\text{seal } k.\text{out } a.\text{in } b.\langle M\rangle^{\hat{}}\,]\ ] \mid b[\,\overline{\text{in}}\ \{x\}_k.(y)^x.P\,]$$

$$\longrightarrow (\boldsymbol{\nu} k)a[\ n\{\!|\,\text{out } a.\text{in } b.\langle M\rangle^{\hat{}}\,|\!\}_k\ ] \mid b[\,\overline{\text{in}}\ \{x\}_k.(y)^x.P\,]$$

$$\longrightarrow (\boldsymbol{\nu} k)a[\ \ ] \mid n\{\!|\,\text{in } b.\langle M\rangle^{\hat{}}\,|\!\}_k \mid b[\,\overline{\text{in}}\ \{x\}_k.(y)^x.P\,]$$

$$\longrightarrow (\boldsymbol{\nu} k)a[\ \ ] \mid b[\,n[\,\langle M\rangle^{\hat{}}\,] \mid (y)^n.P\{n/x\}\,]$$

CBA: Syntax

Expressions

M, N	$::=$	$k, \ldots, q$	names
	$\mid$	$x, \ldots, z$	variables
	$\mid$	in M	enter M
	$\mid$	out M	exit M
	$\mid$	$\overline{\text{in}}$	let enter
	$\mid$	$\overline{\text{out}}$	let exit
	$\mid$	$M.M$	path

Locations

η	$::=$	a	child
	$\mid$	$\uparrow$	parent
	$\mid$	$\star$	local

Prefixes

π	$::=$	M	path
	$\mid$	$(x_1, \ldots, x_k)^\eta$	input
	$\mid$	$\langle M_1, \ldots, M_k\rangle^\eta$	output
	$\mid$	$\overline{\text{in}}\ \{x\}_M,$	let in & unseal
	$\mid$	$\overline{\text{out}}\ \{x\}_M$	let out & unseal
	$\mid$	seal M	sealing

Processes

P	$::=$	$\mathbf{0}$
	$\mid$	$\pi.P$
	$\mid$	$(\boldsymbol{\nu} n)P$
	$\mid$	$P \mid P$
	$\mid$	$!\pi.P$
	$\mid$	$M[\,P\,]$
	$\mid$	$M\{\!|\,P\,|\!\}_N$

Silent Reduction

Silent Evaluation Context: $SE ::= [-] \mid (\boldsymbol{\nu} n)SE \mid SE \mid P \mid n[\,SE\,] \mid n\{\!|\,SE\,|\!\}_k$

Mobility I

$$n\{\,\text{in } m.P \mid Q\,\} \mid m\{\,\overline{\text{in}}\ .R \mid S\,\} \xrightarrow{\text{shh}} m\{\,n\{\,P \mid Q\,\} \mid R \mid S\,\} \qquad \text{(enter)}$$

$$m\{\,n\{\,\text{out } m.P \mid Q\,\} \mid R\,\} \mid \overline{\text{out}}\ .S \xrightarrow{\text{shh}} n\{\,P \mid Q\,\} \mid m\{\,R\,\} \mid S \qquad \text{(exit)}$$

Mobility II

$$n\{\!|\,\text{in } m.P \mid Q\,|\!\}_k \mid m\{\,\overline{\text{in}}\ \{x\}_k.R \mid S\,\} \xrightarrow{\text{shh}} m\{\,n[\,P \mid Q\,] \mid R\{n/x\} \mid S\,\} \qquad \text{(K-enter)}$$

$$m\{\,P \mid n\{\!|\,\text{out } m.Q \mid R\,|\!\}_k\,\} \mid \overline{\text{out}}\ \{x\}_k.S \xrightarrow{\text{shh}} m\{\,P\,\} \mid n[\,Q \mid R\,] \mid S\{n/x\} \qquad \text{(K-exit)}$$

$$n[\,\text{seal } k.P \mid Q\,] \xrightarrow{\text{shh}} n\{\!|\,P \mid Q\,|\!\}_k \qquad \text{(seal)}$$

Structural Rules

$$P \equiv Q,\ Q \xrightarrow{\text{shh}} R,\ R \equiv S \Longrightarrow P \xrightarrow{\text{shh}} S \qquad \text{(struct)}$$

$$P \xrightarrow{\text{shh}} Q \Longrightarrow SE[P] \xrightarrow{\text{shh}} SE[Q] \qquad \text{(context)}$$

Reduction

Evaluation Context $E ::= [-] \mid (\boldsymbol{\nu} n)E \mid P \mid E \mid E \mid P \mid n[\,E\,]$

Communication

(local)	$(\vec{x})P \mid \langle\vec{M}\rangle Q$	$\longrightarrow$	$P\{\vec{M}/\vec{x}\} \mid Q$
(input n)	$(\vec{x})^n P \mid n[\,\langle\vec{M}\rangle^{\uparrow} Q \mid R\,]$	$\longrightarrow$	$P\{\vec{M}/\vec{x}\} \mid n[\,Q \mid R\,]$
(output n)	$\langle\vec{M}\rangle^n P \mid n[\,(\vec{x})^{\uparrow} Q \mid R\,]$	$\longrightarrow$	$P \mid n[\,Q\{\vec{M}/\vec{x}\} \mid R\,]$

Structural Rules

(silent)	$P \xrightarrow{\text{shh}} Q$	$\Rightarrow$	$P \longrightarrow Q$
(struct)	$P \equiv Q, Q \longrightarrow R,\ R \equiv S$	$\Rightarrow$	$P \longrightarrow S$
(context)	$P \longrightarrow Q$	$\Rightarrow$	$E[P] \longrightarrow E[Q]$

Remarks

- No explicit data encryption (delegate it to implementation).
- At most one level of sealing.
- Silent reductions do not apply under prefix.
- Reductions do not apply under prefix and sealed ambients.
- No computation, except mobility, for sealed ambients

Encoding of spi

- Idea: represent an encrypted message with a sealed ambient which contains the message.

  Communicating the encrypted messages is communicating the name of the corresponding ambient.

- Use three translation maps, with $[\![\,\cdot\,]\!]$ leading (and p a name):

$$\langle\!\langle\,\cdot\,\rangle\!\rangle_p : \text{Expressions} \longmapsto \text{Expressions}$$
$$[\![\,\cdot\,]\!]_p : \text{Expressions} \longmapsto \text{Processes}$$
$$[\![\,\cdot\,]\!] \ : \text{Processes} \longmapsto \text{Processes}$$

- $\langle\!\langle M \rangle\!\rangle_p$ returns p, the name of the ambient that stores M. Correspondingly, $[\![M]\!]_p$ stores M into an ambient named p.

Encoding of spi

- The encoding (monadic case)

$$\langle\!\langle a \rangle\!\rangle_p \triangleq a, \quad \langle\!\langle \{M\}_k \rangle\!\rangle_p \triangleq p$$

$$[\![a]\!]_p \triangleq \mathbf{0}$$

$$[\![\{M\}_k]\!]_p \triangleq (\boldsymbol{\nu} q)([\![M]\!]_q \mid p[\,(x)^{\hat{}}.\mathsf{seal}\ k.\mathsf{in}\ x.\langle\,\langle\!\langle M \rangle\!\rangle_q\rangle^{\hat{}}\,])$$

$$[\![\mathbf{0}]\!] \triangleq \mathbf{0}, \quad [\![(\boldsymbol{\nu} n)P]\!] \triangleq (\boldsymbol{\nu} n)[\![P]\!], \quad [\![P \mid Q]\!] \triangleq [\![P]\!] \mid [\![Q]\!]$$

$$[\![\bar{b}\langle M\rangle]\!] \triangleq (\boldsymbol{\nu} q)([\![M]\!]_q \mid b[\,\langle\,\langle\!\langle M \rangle\!\rangle_q\rangle^{\hat{}}\,])$$

$$[\![b(x)P]\!] \triangleq (x)^b[\![P]\!]$$

$$[\![\mathsf{case}\ M\ \mathsf{of}\ \{x\}_k\ \mathsf{in}\ P]\!] \triangleq \mathsf{let}\ z = \langle\!\langle M \rangle\!\rangle_p\ \mathsf{in}$$
$$(\boldsymbol{\nu} p)([\![M]\!]_p \mid (\boldsymbol{\nu} c)(\langle c\rangle^z \mid c[\,\overline{\mathsf{in}}\ \{y\}_k.(x)^y\langle x\rangle^{\hat{}}\,] \mid (x)^c P))$$

- Thm. The encoding is equationally sound.

Secrecy, by typing

Typing System: Secrecy is captured by a type system $\vdash$ which may classify processes as untrusted and data as public if it can be exchanged with untrusted process.

- Types split the world in two: TRUSTED vs UNTRUSTED.

	TRUSTED	UNTRUSTED
Message types W	$\mathsf{N}[E]$, $\mathsf{Key}[E]$	Public
Exchange types E	$(W_1, \ldots, W_k)$, Shh	
Process Types T	$[E, F]$	Un

- The type system
  - allows interactions between the two components
  - preserves the desired secrecy invariants on the trusted components.

Types for Secrecy

Message Types

- Public: messages that may be exchanged with untrusted processes. Includes movement (co-)capabilities
- $\mathsf{N}[E]$: ambients with upward E exchanges (as usual)
- $\mathsf{Key}[E]$: keys that may apply to ambients of type $\mathsf{N}[E]$

Process Types

- Un: unknown processes
- [E,F]: processes with local E exchanges + upward F exchanges

Trusted and Untrusted

The rules of the game: How do TRUSTED and UNTRUSTED interact?

- mobility for free: (un)trusted ambients may traverse (un)trusted sites;
- NO local exchanges between trusted and untrusted processes;
- YES hierarchical exchanges of public values allowed between trusted and untrusted processes.

Typing Rules: A Manichean view of the world.

- Each process form has two typing rules, depending on whether is trusted or untrusted .
- Trusted systems exchanging public values with the untrusted components become themselves untrusted

Sample typing rules

(Co-In Key)

$$\frac{\Gamma \vdash M : \mathsf{Key}[E] \qquad \Gamma, x{:}\mathsf{N}[E] \vdash P : [G,H]}{\Gamma \vdash \overline{\mathsf{in}}\ \{x\}_M.P : [G,H]}$$

(Untrusted Co-In)

$$\frac{\Gamma \vdash M : \mathsf{Public} \qquad \Gamma, x{:}\mathsf{Public} \vdash P : \mathsf{Un}}{\Gamma \vdash \overline{\mathsf{in}}\ \{x\}_M.P : \mathsf{Un}}$$

(Amb Seal)

$$\frac{\Gamma \vdash N : \mathsf{Key}[E] \qquad \Gamma \vdash M : \mathsf{N}[E] \qquad \Gamma \vdash P : [F,E]}{\Gamma \vdash M\{\!|\, P \,|\!\}_N : T}$$

(Untrusted Amb Seal)

$$\frac{\Gamma \vdash N : \mathsf{Public} \qquad \Gamma \vdash M : \mathsf{Public} \qquad \Gamma \vdash P : \mathsf{Un}}{\Gamma \vdash M\{\!|\, P \,|\!\}_N : T}$$

(Input M Amb)

$$\frac{\Gamma \vdash M : \mathsf{N}[W_1,\ldots,W_n] \qquad \Gamma, x_1 : W_1, \ldots, x_n : W_n \vdash P : [E,F]}{\Gamma \vdash (x_1,\ldots,x_n)^M P : [E,F]}$$

(Untrusted Input M)

$$\frac{\Gamma \vdash M : \mathsf{Public} \qquad \Gamma, x_i : \mathsf{Public} \vdash P : \mathsf{Un}}{\Gamma \vdash (x_1,\ldots,x_k)^M P : \mathsf{Un}}$$

Properties of the Type System

- *Subject Reduction*
  If $\Gamma \vdash P : T$ and $P \longrightarrow Q$, then $\Gamma \vdash Q : T$.
- *Typability*
  Let $\{a_1,..,a_n\} = \mathsf{fn}(P)$ and $\{x_1,..,x_m\} = \mathsf{fv}(P)$ then

  $$a_1 : \mathsf{Public}, \ldots, a_n : \mathsf{Public}, x_1 : \mathsf{Public}, \ldots, x_m : \mathsf{Public} \vdash P : \mathsf{Un}$$

- and *Secrecy*...

Secrecy and Adversaries

Intuitively:

> A process preserves the secrecy of a piece of data M if it does not publish M, or anything that would permit the computation of M.

S-Adversary: A context $\mathbf{A}(-)$ which initially knows all names in S.

Revealing Names: P may reveal n to S if there exists an S-adversary $\mathbf{A}(-)$, a context $\mathbf{C}(-)$, and a name $c \in S$ not bound by $\mathbf{C}(-)$ such that:

$$\mathbf{A}(P) \Longrightarrow \mathbf{C}(c[\,\langle n\rangle^{\hat{}}\mid Q\,]).$$

This captures two kinds of attacks

- an hostile context enclosing a trusted process, as in $a[\,Q \mid (-)\,]$,
- a malicious agent mounting an attack to a remote host, as in $a[\,\mathsf{in}\ p.\mathsf{in}\ q.Q \mid Q'\,] \mid (-)$.

An example: $P = c[\,\langle a\rangle^{\hat{}}\,] \mid a[\,\langle k\rangle^{\hat{}}\,]$ may reveal k to $\{c\}$. In fact, for $\mathbf{A}(-)$ the $\{c\}$-adversary $(x)^c.(y)^x.c[\,\langle y\rangle^{\hat{}}\,] \mid (-)$, we have $\mathbf{A}(P) \Longrightarrow c[\,\langle k\rangle^{\hat{}}\,] \mid c[\,] \mid a[\,]$.

Secrecy and Adversaries

Intuitively:

> A process preserves the secrecy of a piece of data M if it does not publish M, or anything that would permit the computation of M.

S-Adversary: A context $\mathbf{A}(-)$ which initially knows all names in S.

Revealing Names: P may reveal n to S if there exists an S-adversary $\mathbf{A}(-)$, a context $\mathbf{C}(-)$, and a name $c \in S$ not bound by $\mathbf{C}(-)$ such that:

$$\mathbf{A}(P) \Longrightarrow \mathbf{C}(c[\,\langle n\rangle^{\hat{}}\mid Q\,]).$$

This captures two kinds of attacks

- an hostile context enclosing a trusted process, as in $a[\,Q \mid (-)\,]$,
- a malicious agent mounting an attack to a remote host, as in $a[\,\mathsf{in}\ p.\mathsf{in}\ q.Q \mid Q'\,] \mid (-)$.

Secrecy Theorem: Well-typed processes do not reveal their secrets publicly. Formally, if $\Gamma \vdash P : \mathsf{Un}$ and $\Gamma \vdash s : W \neq \mathsf{Public}$, then P preserves the secrecy of s from all public channels, i.e. from $\{a \mid \Gamma \vdash a : \mathsf{Public}\}$.

Questions

- Sealing/unsealing is a rather flexible mechanism.
- How and how efficiently can the underlying mechanism of selective encryption be implemented ?
- Sealing provides for secrecy of messages. It would be nice to have more, eg hiding part of the agent structure.
- Can guarantees of data integrity be established along similar lines.

Dimensions, Capacities, Mobility

$\boxed{\text{Focus: Capacity Bounds Awareness.}}$

BoCa: Bounded Capacities

- Subjective Mobility
- Bounded Capacity Ambients
- Space as a linear co-capability.
- Fine control of capacity.

$$a[\,\mathbf{in}\,b\,.\,P \mid Q\,] \mid b[\,\blacksquare \mid R\,] \quad \longrightarrow \quad \blacksquare \mid b[\,a[\,P \mid Q\,] \mid R\,]$$

subjective move capability

space co-capability

Minimal Desiderata

- Realistic about space occupation. Bigger processes take more space.

$$n[\,\mathbf{in}\,m\,.\,\text{big and fat P}\,] \mid m[\,\rule{0.6em}{0.3em}\,] \mid n[\,\mathbf{in}\,m\,.\,\text{small and slim P}\,]$$

- Replication must be handled appropriately

$$a[\,!P\,] = a[\,!P \mid P\,] = a[\,!P \mid P \mid P\,] = a[\,!P \mid P \mid P \mid P\,] = \dots$$

Allow an analisys of variation in space occupation

- More precisely, control process spawning.

Computation takes space, dynamically, and we'd like to model it.

A Calculus of Bounded Capacities: Movement

Fundamentals: Space Conscious Movement

$$\begin{aligned} a[\,\mathbf{in}\,b\,.\,P \mid Q\,] \mid b[\,\rule{0.6em}{0.3em} \mid R\,] &\longrightarrow \rule{0.6em}{0.3em} \mid b[\,a[\,P \mid Q\,] \mid R\,] \\ \rule{0.6em}{0.3em} \mid b[\,a[\,\mathbf{out}\,b\,.\,P \mid Q\,] \mid R\,] &\longrightarrow a[\,P \mid Q\,] \mid b[\,\rule{0.6em}{0.3em} \mid R\,] \end{aligned}$$

Example: Travelling needs but consumes no space.

$$\begin{aligned} & a[\,\mathbf{in}\,b\,.\,\mathbf{in}\,c\,.\,\mathbf{out}\,c\,.\,\mathbf{out}\,b\,.\,\mathbf{0}\,] \mid b[\,\rule{0.6em}{0.3em} \mid c[\,\rule{0.6em}{0.3em}\,]\,] \\ & \searrow\searrow \rule{0.6em}{0.3em} \mid b[\,\rule{0.6em}{0.3em} \mid c[\,a[\,\mathbf{out}\,c\,.\,\mathbf{out}\,b\,.\,\mathbf{0}\,]\,]\,] \\ & \searrow\searrow a[\,\mathbf{0}\,] \mid b[\,\rule{0.6em}{0.3em} \mid c[\,\rule{0.6em}{0.3em}\,]\,] \end{aligned}$$

Term Well-formedness

Fundamentals: Space Conscious Movement

- But the size of travellers matters!

$$\begin{aligned} a^k[\,\mathbf{in}\,b\,.\,P \mid Q\,] \mid b[\overbrace{\rule{0.6em}{0.3em} \mid \dots \mid \rule{0.6em}{0.3em}}^{k\text{ times}} \mid R\,] &\longrightarrow \overbrace{\rule{0.6em}{0.3em} \mid \dots \mid \rule{0.6em}{0.3em}}^{k\text{ times}} \mid b[\,a^k[\,P \mid Q\,] \mid R\,] \\ \underbrace{\rule{0.6em}{0.3em} \mid \dots \mid \rule{0.6em}{0.3em}}_{k\text{ times}} \mid b[\,a^k[\,\mathbf{out}\,b\,.\,P \mid Q\,] \mid R\,] &\longrightarrow a^k[\,P \mid Q\,] \mid b[\underbrace{\rule{0.6em}{0.3em} \mid \dots \mid \rule{0.6em}{0.3em}}_{k\text{ times}} \mid R\,] \end{aligned}$$

What is the a^k? A well-formedness annotation measuring the size of P.

It counts spaces: $\text{weight}(\rule{0.6em}{0.3em}) = 1$, $\text{weight}(a^k[\,P\,]) = k$ if $\text{weight}(P) = k$, $\perp$ otherwise.

Reduction only for well-formed terms: (1) weights appear as conditions on reductions; (2) the calculus' operators make only sense with type annotations.

Notation. We use $\rule{0.6em}{0.3em}^k$ as a shorthand for $\underbrace{\rule{0.6em}{0.3em} \mid \dots \mid \rule{0.6em}{0.3em}}_{k\text{ times}}$.

A Calculus of Bounded Capacities: Open

Fundamentals: Space Conscious Opening

$$\mathbf{opn}\,a\,.\,P \mid a^k[\,\overline{\mathbf{opn}}\,.\,Q \mid R\,] \longrightarrow P \mid Q \mid R$$

Example: Recovering Mobile Ambients.

$$\begin{aligned} [\![\, a[\,P\,] \,]\!] &\triangleq a^0[\,!\overline{\mathbf{opn}} \mid [\![\, P \,]\!]\,] \\ [\![(\boldsymbol{\nu} a)P]\!] &\triangleq (\boldsymbol{\nu} a^0)[\![P]\!] \\ & \dots \end{aligned}$$

A Calculus of Bounded Capacities: Spawning

Fundamentals: Space Conscious Process Activation

passive process: weighs 0

$$\triangleright^k P \mid \rule{1ex}{.5ex}^k \quad\equiv\quad P$$

P weighs k

Example: Replication: $!^k \triangleq\, !\triangleright^k$

$$!\triangleright^k P \mid \rule{1ex}{.5ex}^k \quad\longrightarrow\quad !\triangleright^k P \mid P$$

Types ensure only 0-weighted processes are replicable: One must use spawning, so that replication needs space proportional to the process' weight.

Example: Recursion (well, almost):

$$\mathsf{rec}(X^k)P \;\triangleq\; (\nu X^k)(!\mathbf{opn}\, X\,.\triangleright^k \widehat{P} \mid X[\,\rule{1ex}{.5ex}^k\,]), \qquad \text{where } \widehat{P} \triangleq P\{X[\,\rule{1ex}{.5ex}^k\,]/X\}$$

BoCa: Examples (Open)

Example: Ambient Spawning

$$\mathsf{spw}^k b[\,P\,] \triangleq \mathsf{exp}^0[\,\mathbf{out}\, a\,.\,\overline{\mathbf{opn}}\,.\triangleright^k b[\,P\,]\,]$$

Then,

$$a[\,\mathsf{spw}^k b[\,P\,] \mid Q\,] \mid \rule{1ex}{.5ex}^k \mid \mathbf{opn}\,\mathsf{exp} \quad\longrightarrow\quad a[\,Q\,] \mid b[\,P\,].$$

The father must provide enough space for the activation, of course.

Example: Ambient Renaming

$$\mathsf{a\_be\_b}^k\,.\,P \triangleq \mathsf{spw}^k_a b[\,\rule{1ex}{.5ex}^k \mid \mathbf{opn}\, a\,] \mid \mathbf{in}\, b\,.\,\overline{\mathbf{opn}}\,.\,P.$$

Then,

$$\rule{1ex}{.5ex}^k \mid \mathbf{opn}\, x \mid a^k[\,\mathsf{a\_be\_b}^k\,.\,P \mid Q\,] \quad\longrightarrow\quad b[\,P \mid Q\,] \mid \rule{1ex}{.5ex}^k.$$

Ambient a needs to borrow space to rename itself.

A Calculus of Bounded Capacities: Transfer

Fundamentals: Space Acquisition and Release

$$a^{\hat{}}\,.\,P \mid \rule{1ex}{.5ex} \mid a^k[\,{}^{\check{}}\,.\,Q \mid R\,] \quad\longrightarrow\quad P \mid a^{k+1}[\,Q \mid \rule{1ex}{.5ex} \mid R\,]$$

$$a^{k+1}[\,\text{«}\,.\,P \mid \rule{1ex}{.5ex} \mid S\,] \mid b^h[\,a\text{»}\,.\,Q \mid R\,] \quad\longrightarrow\quad a^k[\,P \mid S\,] \mid b^{h+1}[\,Q \mid \rule{1ex}{.5ex} \mid R\,]$$

Transfer from Child:

$$\mathsf{get\_from\_child}\ a\,.\,P \triangleq (\nu n)(\mathbf{opn}\, n\,.\,P \mid n[\,a\text{»}.\overline{\mathbf{opn}}\,])$$

Example: A Memory Module

$$\mathsf{memMod} \triangleq \mathsf{mem}[\,\rule{1ex}{.5ex}^{256MB} \mid !\text{«} \mid !\mathsf{free}\text{»}\,]$$

$$\mathsf{malloc} \triangleq \mathsf{m}[\,!\mathsf{mem}\text{»}\,.\,\mathsf{free}[\,\mathbf{out}\,\mathsf{m}\,.\,\mathsf{m}\text{»}\,.\,\text{«}\,] \mid !\text{«}\,]$$

$$\mathsf{memMod} \mid \mathsf{malloc} \longrightarrow^{256MB} \mathsf{mem}[\,!\text{«} \mid !\mathsf{free}\text{»}\,] \mid \mathsf{m}[\,\rule{1ex}{.5ex}^{256MB} \mid \ldots\,] \longrightarrow^{2\times 256MB}$$

$$\mathsf{mem}[\,!\text{«} \mid !\mathsf{free}\text{»}\,] \mid \mathsf{malloc} \mid \mathsf{free}^{256MB}[\,\rule{1ex}{.5ex} \mid \text{«}\,] \longrightarrow^{256MB} \mathsf{memMod} \mid \mathsf{malloc} \mid \ldots$$

On the nature of space

An economic vehicle for multiple concepts

- Available space: $a[\,\rule{1ex}{.5ex} \mid P\,]$
- Occupied space: $M\,.\,\rule{1ex}{.5ex}$. (Notation: $M\,.\,\blacktriangle$.)
- Lost space: $(\nu a)a^k[\,\rule{1ex}{.5ex}^k\,]$. (Notation: $\mathbf{0}^k$.)

$$\mathsf{destroy}^k \triangleq (\nu a)(\underbrace{a^{\hat{}}\,.\,\ldots\,.\,a^{\hat{}}}_{k\ \text{times}}.\mathbf{0} \mid a^0[\,\underbrace{{}^{\check{}}\,.\,\ldots\,.\,{}^{\check{}}}_{k\ \text{times}}.\mathbf{0}\,])$$

$$\mathsf{destroy}^k \mid \rule{1ex}{.5ex}^k \longrightarrow^k \mathbf{0}^k$$

A Calculus of Bounded Capabilities: Syntax

Syntax

$$P ::= \rule{0.6em}{0.25em} \mid \mathbf{0} \mid M\,.\,P \mid P \mid P \mid M[\,P\,] \mid \,!P \mid \triangleright^k P \mid (\boldsymbol{\nu} n : \pi)P \mid (x:\chi)P \mid \langle M \rangle P$$

$$C ::= \mathbf{in}\, M \mid \mathbf{out}\, M \mid \mathbf{opn}\, M \mid M^{\hat{}} \mid \,«$$

$$\overline{C} ::= \overline{\mathbf{opn}} \mid {}^{\check{}} \mid M»$$

$$M ::= \varepsilon \mid x \mid C \mid \overline{C} \mid M\,.\,M$$

Structural Congruence:

$(\mid, \mathbf{0})$ is a commutative monoid.

$(\boldsymbol{\nu} a)(P \mid Q) \equiv (\boldsymbol{\nu} a)P \mid Q$	if $a \notin \mathsf{fn}(Q)$
$(\boldsymbol{\nu} a)\mathbf{0} \equiv \mathbf{0}$	
$(\boldsymbol{\nu} a)\langle M \rangle P \equiv \langle M \rangle (\boldsymbol{\nu} a)P$	if $a \notin \mathsf{fn}(P)$
$(\boldsymbol{\nu} a)(\boldsymbol{\nu} b)P \equiv (\boldsymbol{\nu} b)(\boldsymbol{\nu} a)P$	
$a[\,(\boldsymbol{\nu} b)P\,] \equiv (\boldsymbol{\nu} b)a[\,P\,]$	if $a \neq b$
$!P \equiv \,!P \mid P$	

BoCa: Reduction Semantics

(enter)	$a^k[\,\mathbf{in}\, b\,.\,P \mid Q\,] \mid b[\,\rule{0.6em}{0.25em}^k \mid R\,] \longrightarrow \rule{0.6em}{0.25em}^k \mid b[\,a[\,P \mid Q\,] \mid R\,]$
(exit)	$\rule{0.6em}{0.25em}^k \mid b[\,a^k[\,\mathbf{out}\, b\,.\,P \mid Q\,] \mid R\,] \longrightarrow a^k[\,P \mid Q\,] \mid b[\,\rule{0.6em}{0.25em}^k \mid R\,]$
(open)	$\mathbf{opn}\, a\,.\,P \mid a[\,\overline{\mathbf{opn}}\,.\,Q \mid R\,] \longrightarrow P \mid Q \mid R$
(tranD)	$a^{\hat{}}\,.\,P \mid \rule{0.6em}{0.25em} \mid a^k[\,{}^{\check{}}\,.\,Q \mid R\,] \longrightarrow P \mid a^{k+1}[\,Q \mid \rule{0.6em}{0.25em} \mid R\,]$
(trans)	$a^{k+1}[\,«\,.\,P \mid \rule{0.6em}{0.25em} \mid S\,] \mid b^h[\,a»\,.\,Q \mid R\,] \longrightarrow a^k[\,P \mid S\,] \mid b^{h+1}[\,Q \mid \rule{0.6em}{0.25em} \mid R\,]$
(spawn)	$\triangleright^k P \mid \rule{0.6em}{0.25em}^k \longrightarrow P$
(comm)	$(x:\chi)P \mid \langle M \rangle Q \longrightarrow P\{M/x\} \mid Q$

A System of Capacity Types

Capacity Types: $\phi, \ldots$ are pairs of nats $[n, N]$, with $n \leq N$.

Effect Types $\mathcal{E}, \ldots$ are pairs of nats (d, i), representing decs and incs.

Exchange Types: $\chi ::= \mathsf{Shh} \mid \mathsf{Amb}\langle \sigma, \chi \rangle \mid \mathsf{Cap}\langle \mathcal{E}, \chi \rangle$

Process and **Ambient** and **Capability Types**:

$a : \mathsf{Amb}\langle \phi, \chi \rangle$	a has no less than ϕ_{m} and no more than ϕ_{M} spaces
$P : \mathsf{Proc}\langle k, \mathcal{E}, \chi \rangle$	P weighs k and produces the effect $\mathcal{E}$ on ambients
$C : \mathsf{Cap}\langle \mathcal{E}, \chi \rangle$	C transforms processes adding $\mathcal{E}$ to their effects

Effects and capacities componentwise and are ordered as follows:

$$\sigma \lessdot \phi \equiv \phi_{\mathsf{m}} \leq \sigma_{\mathsf{m}} \text{ and } \sigma_{\mathsf{M}} \leq \phi_{\mathsf{M}},$$

A Typing System: Capabilities

(Axiom)
$$\frac{}{\Gamma, a : \mathsf{Amb}\langle \phi, \chi \rangle \vdash a : \mathsf{Amb}\langle \phi, \chi \rangle}$$

(Empty)
$$\frac{}{\Gamma \vdash \varepsilon : \mathsf{Cap}\langle (0,0), \chi \rangle}$$

(In)
$$\frac{\Gamma \vdash M : \mathsf{Amb}\langle \phi, \chi' \rangle}{\Gamma \vdash \mathbf{in}\, M : \mathsf{Cap}\langle (0,0), \chi \rangle}$$

(Out)
$$\frac{\Gamma \vdash M : \mathsf{Amb}\langle \phi, \chi' \rangle}{\Gamma \vdash \mathbf{out}\, M : \mathsf{Cap}\langle (0,0), \chi \rangle}$$

(TranD)
$$\frac{\Gamma \vdash M : \mathsf{Amb}\langle \phi, \chi' \rangle}{\Gamma \vdash M^{\hat{}} : \mathsf{Cap}\langle (0,0), \chi \rangle}$$

(TranS)
$$\frac{}{\Gamma \vdash « : \mathsf{Cap}\langle (1,0), \chi \rangle}$$

(Open)
$$\frac{\Gamma \vdash M : \mathsf{Amb}\langle [n,N], \chi \rangle}{\Gamma \vdash \mathbf{opn}\, M : \mathsf{Cap}\langle (N-n, N-n), \chi \rangle}$$

A Typing System: CoCapabilities and Processe

(coTranD)
$$\frac{}{\Gamma \vdash \check{\check{}} : \mathsf{Cap}\langle(0,1),\chi\rangle}$$

(coTranS)
$$\frac{\Gamma \vdash M : \mathsf{Amb}\langle\phi,\chi'\rangle}{\Gamma \vdash M\text{»} : \mathsf{Cap}\langle(0,1),\chi\rangle}$$

(coOpen)
$$\frac{}{\Gamma \vdash \overline{\mathbf{opn}} : \mathsf{Cap}\langle(0,0),\chi\rangle}$$

(Composition)
$$\frac{\Gamma \vdash M : \mathsf{Cap}\langle\mathcal{E},\chi\rangle \quad \Gamma \vdash M' : \mathsf{Cap}\langle\mathcal{E}',\chi\rangle}{\Gamma \vdash M.M' : \mathsf{Cap}\langle\mathcal{E}+\mathcal{E}',\chi\rangle}$$

(Slot)
$$\frac{}{\Gamma \vdash \rule{1em}{0.4ex} : \mathsf{Proc}\langle 1,(0,0),\chi\rangle}$$

(Zero)
$$\frac{}{\Gamma \vdash \mathbf{0} : \mathsf{Proc}\langle 0,(0,0),\chi\rangle}$$

(Input)
$$\frac{\Gamma, x:\chi \vdash P : \mathsf{Proc}\langle k,\mathcal{E},\chi\rangle}{\Gamma \vdash (x:\chi)P : \mathsf{Proc}\langle k,\mathcal{E},\chi\rangle}$$

(Output)
$$\frac{\Gamma \vdash M : \chi \quad \Gamma \vdash P : \mathsf{Proc}\langle k,\mathcal{E},\chi\rangle}{\Gamma \vdash \langle M\rangle P : \mathsf{Proc}\langle k,\mathcal{E},\chi\rangle}$$

A Typing System: Processes

(Prefix)
$$\frac{\Gamma \vdash M : \mathsf{Cap}\langle\mathcal{E},\chi\rangle \quad \Gamma \vdash P : \mathsf{Proc}\langle k,\mathcal{E}',\chi\rangle}{\Gamma \vdash M\,.\,P : \mathsf{Proc}\langle k,\mathcal{E}+\mathcal{E}',\chi\rangle}$$

(Replication)
$$\frac{\Gamma \vdash P : \mathsf{Proc}\langle 0,(0,0),\chi\rangle}{\Gamma \vdash !P : \mathsf{Proc}\langle 0,(0,0),\chi\rangle}$$

(New)
$$\frac{\Gamma, a:\mathsf{Amb}\langle\phi,\chi\rangle \vdash P : \mathsf{Proc}\langle k,\mathcal{E},\chi'\rangle}{\Gamma \vdash (\boldsymbol{\nu} a:\mathsf{Amb}\langle\phi,\chi\rangle)P : \mathsf{Proc}\langle k,\mathcal{E},\chi'\rangle}$$

(Spawn)
$$\frac{\Gamma \vdash P : \mathsf{Proc}\langle k,\mathcal{E},\chi\rangle}{\Gamma \vdash \triangleright^k P : \mathsf{Proc}\langle 0,\mathcal{E},\chi\rangle}$$

(Parallel)
$$\frac{\Gamma \vdash P : \mathsf{Proc}\langle k,\mathcal{E},\chi\rangle \quad \Gamma \vdash Q : \mathsf{Proc}\langle k',\mathcal{E}',\chi\rangle}{\Gamma \vdash P \mid Q : \mathsf{Proc}\langle k+k',\mathcal{E}+\mathcal{E}',\chi\rangle}$$

(Ambient)
$$\frac{\Gamma \vdash M : \mathsf{Amb}\langle[n,N],\chi\rangle \quad \Gamma \vdash P : \mathsf{Proc}\langle k,(d,i),\chi\rangle \quad n \le k-d \quad k+i \le N}{\Gamma \vdash M^k[\,P\,] : \mathsf{Proc}\langle k,(0,0),\chi'\rangle}$$

A Calculus of Bounded Capabilities

Thm: Subject Reduction

If $\Gamma \vdash P : \mathsf{Proc}\langle k,\mathcal{E},\chi\rangle$ and $P \longrightarrow Q$ then $\Gamma \vdash Q : \mathsf{Proc}\langle k,\mathcal{E}',\chi\rangle$ for some $\mathcal{E}' \lessdot \mathcal{E}$.

The missing bit:

Grave interferences in the use of spaces

$$a[\,\mathbf{in}\,b\,] \mid b[\,\triangleright P \mid \rule{1em}{0.4ex} \mid a[\,c[\,\mathbf{out}\,a\,]\,]\,]$$

$$\mathsf{rec}(X^k)P \triangleq (\boldsymbol{\nu} X^k)(!\mathbf{opn}\,X\,.\triangleright^k\widehat{P} \mid X[\,\rule{1em}{0.4ex}^k\,])$$
$$\longrightarrow (\boldsymbol{\nu} X^k)(!\mathbf{opn}\,X\,.\triangleright^k\widehat{P} \mid \mathbf{opn}\,X\,.\triangleright^k\widehat{P} \mid X[\,\rule{1em}{0.4ex}^k\,])$$
$$\longrightarrow (\boldsymbol{\nu} X^k)(!\mathbf{opn}\,X\,.\triangleright^k\widehat{P} \mid \triangleright^k\widehat{P}) \mid \rule{1em}{0.4ex}^k \quad \text{Oooops}$$

Control Space Usage: Named Slots

$$P ::= \rule{1em}{0.4ex}_a \mid a\triangleright^k P \mid \cdots \qquad (\text{spawn}) \quad a\triangleright^k P \mid \rule{1em}{0.4ex}^k_a \;\longrightarrow\; P$$

Example: Renaming slots

$$\{{}^x/_y\}_k\,.\,P \triangleq y\triangleright^k(\rule{1em}{0.4ex}^k_x \mid P)$$

Then, $\rule{1em}{0.4ex}^k_y \mid \{{}^x/_y\}_k\,.\,P \longrightarrow \rule{1em}{0.4ex}^k_x \mid P$

Example: Recursion (now right):

$$\mathsf{rec}(X^k)P \triangleq (\boldsymbol{\nu} X)(!X\triangleright^k\widehat{P} \mid \rule{1em}{0.4ex}^k_X), \quad \text{where } \widehat{P} \triangleq P\{\rule{1em}{0.4ex}^k_X/X\}$$

Example: Deriving Named Slots

$$\rule{1em}{0.4ex}_a \triangleq a[\,\text{«} \mid \rule{1em}{0.4ex}\,]$$
$$a\triangleright^k P \triangleq (\boldsymbol{\nu} n)(n[\,a^k\text{»}\,.\triangleright^k\overline{\mathbf{opn}}\,.\,P\,] \mid \mathbf{opn}\,n)$$

Discussion

This is just a start.

Yet to be done:

- In the large: Develop a theory a resources, including quantitative bounds negotiation and enforcement in GC, which goes beyond space.
  Develop languages and logics to express policies and properties. ...
- In the small: Expressiveness of BoCa; Equational theory; Smarter types; ...
- In general: A lot to be done...

Summary of Lecture V

We illustrated some initial ideas about resource control in ambient-like environment. In particular, access control based on passwords and dynamic learning about the environment; data secrecy data for migrating agents; control of space usage for mobile mobile ambients.

Further Reading: This lecture was based on

- Safe Ambients (Levi, Sangiorgi).
- Boxed Ambients (Bugliesi, Castagna, Crafa)
- NBA (Bugliesi, Crafa, Sassone)
- Secrecy in Untrusted Networks (Bugliesi, Crafa, Sassone)
- Calculus of Bounded Capacities (Barbanera, Bugliesi, Dezani, Sassone)

Related work include

- Finite Control Ambients (Gordon et al)
- Resource Control in the Ambient Calculus (Teller et al)
- Resource Usage Analysis (Igarashi, Kobayashi)
- Typed Assembly Languages (Morrisett) ...and many more...

Drawing conclusions

- Global Computing is about computation over a global, highly distributed, swiftly changing network of bounded resources.
- Central problems are (third-party) resource usage, usage analysis, and protection.
- These lectures have focused on foundational calculi (arising also from work on concurrency), useful to represent and understand issues in GC, and on types systems which guarantee properties of relevance.
- What we discussed:
  - *Name Mobility*
  - *Types for Safety & Control*
  - *Asynchrony & Distribution*
  - *Ambient Mobility*
  - *Resource Control*
- GC is a moving target, and very much alive and kicking. There are many open issues and challenging problems, spanning (almost) all grades from theoretical to practical.

  Certainly a good topic for a PhD...