Part 2

Ambient Calculus

Luca Cardelli
Andy Gordon
Approach

- We want to capture in an abstract way, notions of locality, of mobility, and of ability to cross barriers.

- An ambient is a place, delimited by a boundary, where computation happens.

- Ambients have a name, a collection of local processes, and a collection of subambients.

- Ambients can move in and out of other ambients, subject to capabilities that are associated with ambient names.

- Ambient names are unforgeable (as in $\pi$ and spi).
Basic Assumptions

- Mobile processes are not data. They move, they are not moved.
  - (It might be tempting to move processes by sending them over channels.)

- Mobile computation is the **dynamic local rearrangement of labeled trees**.
  - (Cf.: in $\pi$, it is dynamic propagation of channel names.)

- The choice of primitives for tree rearrangement depends strongly on the design principles one adopts.
  - Are these trees in-memory? (No, they are distributed)
  - Are they just passive data that gets globally transformed? (No, they are full of active local processes with a will of their own.)
  - Do mobile processes have any guarantees?
    - Can they get killed, robbed, poisoned, kidnapped? (In Classical Ambients, only if they are stupid: talk too much, eat bad food, step in dark alleys.)
    - Can they get infected? (Not in Safe Ambients, if they are careful.)
    - How do they talk to each other? (Richer options in Boxed Ambients.)
Folder Metaphor

- An ambient can be graphically represented as a folder:
  - Consisting of a folder name $n$,
  - And active contents $P$, including:
    - Hierarchical data, and computations ("gremlins").
    - Primitives for mobility and communication.
Example: Message from $a$ to $b$
Example: Message from $a$ to $b$

```
2003-03-17 16:36
Talk 6
```

- **Message from $a$ to $b$**: The message $x \in P\{x\}$ is opened in folder $b$.

- **Folder $a$** is linked to the message $x \in P\{x\}$ in folder $b$.

- Enter: Proceed to the next step.
Example: Message from \( a \) to \( b \)

Open
Example: Message from $a$ to $b$
Example: Message from $a$ to $b$
Example: Agent Authentication

- **home**
  - **n**
  - **open n**
  - **n**

- **g**
  - **out home. in home**
  - **x**
  - **out g. open g**
  - **P**

---

2003-03-17 16:36
Talk 10
Example: Agent Authentication

```
home

n
open n

n

n

out home.

in home

x

out g.

open g

P
```
Example: Agent Authentication
Example: Agent Authentication

home

open n

in home

out g.
open g

P

in home

out g.
open g

P
Example: Agent Authentication
Example: Agent Authentication
Example: Agent Authentication
Example: Agent Authentication
# The Ambient Calculus

<table>
<thead>
<tr>
<th>Processes</th>
<th>Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P \in \Pi \ ::= )</td>
<td>( M ::= )</td>
</tr>
<tr>
<td>((\forall n)P) restriction</td>
<td>( n ) name</td>
</tr>
<tr>
<td>( 0 ) inactivity</td>
<td>( in M ) entry capability</td>
</tr>
<tr>
<td>( P \parallel P' ) parallel</td>
<td>( out M ) exit capability</td>
</tr>
<tr>
<td>( M[P] ) ambient</td>
<td>( open M ) open capability</td>
</tr>
<tr>
<td>( !P ) replication</td>
<td>( \epsilon ) empty path</td>
</tr>
<tr>
<td>( M.P ) exercise a capability</td>
<td>( M.M' ) composite path</td>
</tr>
<tr>
<td>( (n).P ) input locally, bind to ( n )</td>
<td>Actions</td>
</tr>
<tr>
<td>( \langle M \rangle ) output locally (async)</td>
<td>( Temporal )</td>
</tr>
</tbody>
</table>

\[ n[] \triangleq n[0] \]

\[ M \triangleq M.0 \] (where appropriate)
Reduction Semantics

- A structural congruence relation $P \equiv Q$:
  - On spatial expressions, $P \equiv Q$ iff $P$ and $Q$ denote the same tree. So, the syntax modulo $\equiv$ is a notation for spatial trees.
  - On full ambient expressions, $P \equiv Q$ if in addition the respective threads are “trivially equivalent”.
  - Prominent in the definition of the logic.

- A reduction relation $P \rightarrow^* Q$:
  - Defining the meaning of mobility and communication actions.
  - Closed up to structural congruence:
    
    $P \equiv P', \ P' \rightarrow^* Q', \ Q' \equiv Q \ \Rightarrow \ \ P \rightarrow^* Q$
Composition

- Parallel execution is denoted by a binary operator:
  \[ P \parallel Q \]
- It is commutative and associative:
  \[ P \parallel Q \equiv Q \parallel P \]
  \[ (P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R) \]
- It obeys the reduction rule:
  \[ P \rightarrow Q \Rightarrow P \parallel R \rightarrow Q \parallel R \]
Replication

• Replication is a technically convenient way of representing iteration and recursion.

\[ !P \]

• It denotes the unbounded replication of a process \( P \).

\[ !P \equiv P | !P \]

\[ !(P | Q) \equiv !P | !Q \quad !0 \equiv 0 \quad !P \equiv !!P \]

• There are no reduction rules for \( !P \); in particular, the process \( P \) under \(!\) cannot begin to reduce until it is expanded out as \( P | !P \).
Restriction

• The restriction operator creates a new (forever unique) ambient name \( n \) within a scope \( P \).

\[(\forall n)P\]

• As in the \( \pi \)-calculus, the \( \forall n \) binder can float as necessary to extend or restrict the scope of a name. E.g.:

\[(\forall n)(P \mid Q) \equiv P \mid (\forall n)Q \quad \text{if} \; n \notin fn(P)\]

\[(\forall n)m[P] \equiv m[(\forall n)P] \quad \text{if} \; n \neq m\]

• Reduction rule:

\[P \rightarrow Q \quad \Rightarrow \quad (\forall n)P \rightarrow (\forall n)Q\]
Inaction

- The process that does nothing:

\[
0
\]

- Some garbage-collection equivalences:

\[
P \mid 0 \equiv P
\]

\[
!0 \equiv 0
\]

\[
(\forall n)0 \equiv 0
\]

- This process does not reduce.
Ambients

- An ambient is written as follows, where \( n \) is the name of the ambient, and \( P \) is the process running inside of it.

\[
n[P]
\]

- In \( n[P] \), it is understood that \( P \) is actively running:

\[
P \rightarrow Q \implies n[P] \rightarrow n[Q]
\]

- Multiple ambients may have the same name, (e.g., replicated servers).
Actions and Capabilities

- Operations that change the hierarchical structure of ambients are sensitive. They can be interpreted as the crossing of firewalls or the decoding of ciphertexts.

- Hence these operations are restricted by capabilities.

\[ M, P \]

- This executes an action regulated by the capability \( M \), and then continues as the process \( P \).

- The reduction rules for \( M, P \) depend on \( M \).
Entry Capability

- An entry capability, \textit{in} \textit{m}, can be used in the action:

\[ \text{in } m. \ P \]

- The reduction rule (non-deterministic and blocking) is:

\[ n[\text{in } m. \ P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R] \]
Exit Capability

• An exit capability, $out \ m$, can be used in the action:

\[
out \ m. \ P
\]

• The reduction rule (non-deterministic and blocking) is:

\[
m[n[\text{out} \ m. \ P | Q] | R] \rightarrow n[P | Q] | m[R]
\]
Open Capability

- An opening capability, \textit{open m}, can be used in the action:
  \[ \text{open n. } P \]

- The reduction rule (non-deterministic and blocking) is:
  \[ \text{open n. } P \mid n[Q] \rightarrow P \mid Q \]

- An \textit{open} operation may be upsetting to both \( P \) and \( Q \) above.
  - From the point of view of \( P \), there is no telling in general what \( Q \) might do when unleashed.
  - From the point of view of \( Q \), its environment is being ripped open.

- Still, this operation is relatively well-behaved because:
  - The dissolution is initiated by the agent \textit{open n. } P, so that the appearance of \( Q \) at the same level as \( P \) is not totally unexpected;
  - \textit{open n} is a capability that is given out by \( n \), so \( n[Q] \) cannot be dissolved if it does not wish to be.
Design Principle

• An ambient should not get killed or trapped unless:
  • It talks too much. (Making its capabilities public.)
  • It poisons itself. (Opening an untrusted intruder.)
  • Doesn’t look where it’s going. (Entering an untrusted ambient.)

• Some natural primitives violate this principle. E.g.:

\[
\text{n[burst } n. \ P \mid Q] \rightarrow P \mid Q
\]

• Then a mere \textit{in} capability gives a kidnapping ability:

\[
\text{entrap}(M) \equiv (\forall k m) \ (m[M. \ burst \ m. \ in \ k] \mid k[])
\]

\[
\text{entrap}(\text{in } n) \mid n[P] \rightarrow^* (\forall k) \ (n[\text{in } k \mid P] \mid k[])
\]

\[
\rightarrow^* (\forall k) \ k[n[P]]
\]

• One can imagine lots of different mobility primitives, but one must think hard about the “security” implications of combinations of these primitives.
Ambient I/O

• Local anonymous communication within an ambient:

\[(x). P \quad \text{input action}\]
\[\langle M \rangle \quad \text{async output action}\]

• We have the reduction:

\[(x). P \mid \langle M \rangle \rightarrow P\{x\leftarrow M\}\]

• This mechanism fits well with the ambient intuitions.

  • Long-range communication, like long-range movement, should not happen automatically because messages may have to cross firewalls and other obstacles.
  
  • Still, this is sufficient to emulate communication over named channels, etc.
## Reduction

\[
\begin{align*}
\text{Red In: } & \quad n \in m \cdot P \rightarrow Q \mid m[R] & \rightarrow m[n[P \rightarrow Q] \mid R] \\
\text{Red Out: } & \quad m[n \in \text{out } m \cdot P \rightarrow Q] \mid R & \rightarrow n[P \rightarrow Q] \mid m[R] \\
\text{Red Open: } & \quad \text{open } m \cdot P \rightarrow m[Q] & \rightarrow P \rightarrow Q \\
\text{Red Comm: } & \quad (n).P \rightarrow \langle M \rangle & \rightarrow P \{n \leftarrow M\} \\
\text{Red Res: } & \quad P \rightarrow Q \quad \Rightarrow \quad (\forall n)P \rightarrow (\forall n)Q \\
\text{Red Amb: } & \quad P \rightarrow Q \quad \Rightarrow \quad n[P] \rightarrow n[Q] \\
\text{Red Par: } & \quad P \rightarrow Q \quad \Rightarrow \quad P \rightarrow R \rightarrow Q \rightarrow R \\
\text{Red } \equiv: & \quad P', P \equiv Q', Q \equiv Q' \quad \Rightarrow \quad P' \rightarrow Q' \\
\end{align*}
\]

\[\rightarrow^* \quad \text{is the reflexive-transitive closure of } \rightarrow\]
### Structural Congruence

\[
P \equiv P
\]  
(Struct Refl)

\[
P \equiv Q \implies Q \equiv P
\]  
(Struct Symm)

\[
P \equiv Q, Q \equiv R \implies P \equiv R
\]  
(Struct Trans)

\[
P \equiv Q \implies (\forall n)P \equiv (\forall n)Q
\]  
(Struct Res)

\[
P \equiv Q \implies P \mid R \equiv Q \mid R
\]  
(Struct Par)

\[
P \equiv Q \implies !P \equiv !Q
\]  
(Struct Repl)

\[
P \equiv Q \implies M[P] \equiv M[Q]
\]  
(Struct Amb)

\[
P \equiv Q \implies M.P \equiv M.Q
\]  
(Struct Action)

\[
P \equiv Q \implies (n).P \equiv (n).Q
\]  
(Struct Input)

\[
\varepsilon.P \equiv P
\]  
(Struct \(\varepsilon\))

\[
(M.M').P \equiv M.M'.P
\]  
(Struct .)
\[(\forall n)0 \equiv 0\]  
\[(\forall n)(\forall m)P \equiv (\forall m)(\forall n)P\]  
\[(\forall n)(P \mid Q) \equiv P \mid (\forall n)Q\]  \(\text{if } n \notin \text{fn}(P)\)  
\[(\forall n)(m[P]) \equiv m[(\forall n)P]\]  \(\text{if } n \neq m\)  
\[P \mid Q \equiv Q \mid P\]  
\[(P \mid Q) \mid R \equiv P \mid (Q \mid R)\]  
\[P \mid 0 \equiv P\]  
\[!(P \mid Q) \equiv !P \mid !Q\]  
\[!0 \equiv 0\]  
\[!P \equiv P \mid !P\]  
\[!P \equiv !!P\]  

\begin{itemize}
  \item These axioms (particularly the ones for \(!\)) are sound and complete with respect to equality of spatial trees; edge-labeled finite-depth unordered trees, with infinite-branching but finitely many distinct labels under each node.
\end{itemize}
The packet `msg` moves from `a` to `b`, mediated by the capabilities `out a` (to exit `a`), `in b` (to enter `b`), and `open msg` (to open the `msg` envelope).
Noticeable Inequivalences

- Replication creates new names:

\[ !(\forall n)P \not\equiv (\forall n)!P \]

- Multiple \( n \) ambients have separate identity:

\[ n[P] \parallel n[Q] \not\equiv n[P \parallel Q] \]
Safe Ambients [Levi, Sangiorgi]

- “Each action has an equal and opposite coaction.”

- In Ambient Calculus it is difficult to count reliably the number of visitors to an ambient. The fix:

\[
\begin{align*}
n[\text{in } m. P \mid Q] \mid m[\text{in } m. R \mid S] & \longrightarrow m[n[P \mid Q] \mid R \mid S] \quad \text{(In)} \\
m[\text{out } m. P \mid Q] \mid \text{out } m. R \mid S & \longrightarrow n[P \mid Q] \mid m[R \mid S] \quad \text{(Out)} \\
\text{open } n. P \mid n[\text{open } n.Q \mid R] & \longrightarrow P \mid Q \mid R \quad \text{(Open)}
\end{align*}
\]

\[
(m).P \mid \langle M \rangle.Q
\longrightarrow P\{m\leftarrow M\} \mid Q \quad \text{(Comm)}
\]

- The Ambient Calculus is recovered by sprinkling !\text{in } n, !\text{out } n, !\text{open } n appropriately.
Each ambient contains a list of channels $c$ that are used for named communication within the ambient. They are restricted as usual.

\[
\begin{align*}
n[D, c; c(M).P \parallel c(m).Q \parallel R] & \quad \rightarrow \quad n[D, c; P \parallel Q\{m \leftarrow M\} \parallel R] \\
n[D; \text{in } m. P \parallel Q] \parallel m[E; R] & \quad \rightarrow \quad m[E; n[D; P \parallel Q] \parallel R] \\
m[E; n[D; \text{out } m. P \parallel Q] \parallel R] & \quad \rightarrow \quad n[D; P \parallel Q] \parallel m[E; R] \\
m[D; \text{open } n. P \parallel n[E; Q] \parallel R] & \quad \rightarrow \quad m[D; P \parallel Q \parallel R]
\end{align*}
\]
Boxed Ambients [Bugliesi, Castagna, Crafa]

- I/O to parents/children is tricky to encode reliably in Ambient Calculus, but is a very natural basic primitive.
- Boxed Ambients provide it directly (simplifying Seal):

\[
\begin{align*}
\text{in} & \quad n[m[P \mid Q] \mid m[R]] \quad \rightarrow \quad m[n[P \mid Q] \mid R] & \quad \text{(In)} \\
\text{out} & \quad m[n[out \mid m].P \mid Q] \mid R] \quad \rightarrow \quad n[P \mid Q] \mid m[R] & \quad \text{(Out)} \\
\text{m} & \quad (m).P \mid \langle M \rangle.Q \quad \rightarrow \quad P\{m \leftarrow M\} \mid Q & \quad \text{(Local)} \\
\text{Open} & \quad (m)^n.P \mid n[\langle M \rangle.Q \mid R] \quad \rightarrow \quad P\{m \leftarrow M\} \mid n[Q \mid R] & \quad \text{(Input n)} \\
\text{m} & \quad \langle M \rangle^n.P \mid n[(m).Q \mid R] \quad \rightarrow \quad P \mid n[Q\{m \leftarrow M\} \mid R] & \quad \text{(Output n)} \\
\text{m} & \quad \langle M \rangle.P \mid n[(m)^\uparrow.Q \mid R] \quad \rightarrow \quad P \mid n[Q\{m \leftarrow M\} \mid R] & \quad \text{(Input \uparrow)} \\
\text{m} & \quad (m).P \mid n[\langle M \rangle^\uparrow.Q \mid R] \quad \rightarrow \quad P\{m \leftarrow M\} \mid n[Q \mid R] & \quad \text{(Output \uparrow)}
\end{align*}
\]
Ambjects [Bugliesi, Castagna]

- [CG] Ambient Calculus + [AC] Object Calculus =

\[
\begin{align*}
\text{n.a}(M).P & \mid n[D; a(m).Q; R] \\
\rightarrow & \quad P \mid Q\{m \leftarrow M, \text{self} \leftarrow n\} \mid n[D; a(m).Q; R] \\
\text{n}[D; \text{in } m. P \mid Q] & \mid m[E; R] \quad \rightarrow \quad m[E; n[D; P \mid Q] \mid R] \\
\text{m}[E; n[D; \text{out } m. P \mid Q] \mid R] & \quad \rightarrow \quad n[D; P \mid Q] \mid m[E; R] \\
\text{m}[E; \text{open n. P} \mid n[D; Q] \mid R] & \quad \rightarrow \quad m[E; D; P \mid Q \mid R]
\end{align*}
\]
**Joinbients** [Anonymous]

- Ambient Calculus + Join Calculus =

<table>
<thead>
<tr>
<th>Expression</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>???  (n[D; P])</td>
<td>(Join)</td>
</tr>
<tr>
<td>(n[D; \text{in } m. P \mid Q] \mid m[E; R])</td>
<td>(In)</td>
</tr>
<tr>
<td>(m[E; n[D; \text{out } m. P \mid Q] \mid R])</td>
<td>(Out)</td>
</tr>
<tr>
<td>(m[E; \text{open } n. P \mid n[D; Q]])</td>
<td>(Open)</td>
</tr>
</tbody>
</table>
Expressiveness: Encoding Old Concepts

- Synchronization and communication mechanisms.
- Turing machines. (Natural encoding, no I/O required.)
- Arithmetic. (Tricky, no I/O required.)
- Data structures.
  - $\pi$-calculus. (Easy: channels are ambients.)
  - $\lambda$-calculus. (Hard: different than encoding $\lambda$ in $\pi$.)
- Spi-calculus concepts. (?)
Expressiveness: Encoding New Concepts

- Named machines and services on complex networks.
- Agents, applets, RPC.
- Encrypted data and firewalls.
- Data packets, routing, active networks.
- Dynamically linked libraries, plug-ins.
- Mobile devices.
- Public transportation.
Expressiveness: New Challenges

- The combination of mobility and security in the same formal framework is novel and intriguing.
- E.g., we can represent both mobility and security aspects of “crossing a firewall”.
- The combination of mobility and local communication raises questions about suitable synchronization models and programming constructs.
We can use *open* to encode locks:

\[
\begin{align*}
\text{release } n. \ P & \triangleq n[] \mid P \\
\text{acquire } n. \ P & \triangleq \text{open } n. \ P
\end{align*}
\]

This way, two processes can “shake hands” before proceeding with their execution:

\[
\text{acquire } n. \ \text{release } m. \ P \mid \text{release } n. \ \text{acquire } m. \ Q
\]
Turing Machines

\[
\begin{align*}
\text{end} & \mid \text{extendLft} \mid S_0 \mid \\
\text{square} & \mid S_1 \mid \\
\text{square} & \mid S_2 \mid \\
\ldots & \\
\text{square} & \mid S_i \mid \text{head} \mid \\
\ldots & \\
\text{square} & \mid S_{n-1} \mid \\
\text{square} & \mid S_n \mid \text{extendRht} \mid ] \ldots ] \ldots ]]]
\end{align*}
\]

- Exercise: code up \textit{extendLft}, \textit{extendRht}, and (an example of) \textit{head}. You will probably need to use restriction.
Random Access Machines [Busi]

- A finite set of registers: they can hold arbitrary natural numbers.
- A program is a sequence of numbered operations:
  - `succ(r_j)`: add 1 to the contents of register `r_j` and continue.
  - `decjmp(r_j, s)`: if the contents of `r_j` is non-zero, decrease it by 1 and continue, otherwise jump to instruction `s`.
- To stop: jump to nowhere; answer is the content of registers.

\[
\begin{align*}
[r_i = 0] &= z_i[...] & & \ldots = \text{some clever code} \\
[r_i = n+1] &= s_i[...] | [r_i = n] \\
[i : succ(r_j)] &= ![p_i][inc-req_j ![in s_i | in z_i. \ldots] \mid open inc-ack_j. open p_{i+1}] \\
[i : decjmp(r_j, s)] &= ![p_i][dec-req_j [in s_i] | zero-req_j [in z_i] \mid \ldots open ok-dec_j. \ldots open p_{i+1} | \ldots open ok-zero_j. \ldots open p_s] \\
\end{align*}
\]

To start the program: `open p_1`
- Turing-completeness even without restriction and I/O.
Ambients as Mobile Processes

\( \text{tourist} \triangleq (x). \text{joe}[x. \text{enjoy}] \)

\( \text{ticket-desk} \triangleq ! (\text{in AF81SFO. out AF81CDG}) \)

\[ SFO[\text{ticket-desk} \mid \text{tourist} \mid \text{AF81SFO}[\text{route}]] \]

\[ \rightarrow^* SFO[\text{ticket-desk} \mid \]

\[ \text{joe}[\text{in AF81SFO. out AF81CDG. enjoy}] \mid \text{AF81SFO}[\text{route}]] \]

\[ \rightarrow^* SFO[\text{ticket-desk} \mid \]

\[ \text{AF81SFO}[\text{route} \mid \text{joe}[\text{out AF81CDG. enjoy}]]] \]
Assume that the shared key $k$ is already known to the firewall and the client, and that $w$ is the secret name of the firewall.

$$Wally \triangleq (\forall w r) \ (\langle in r \rangle \ | \ r[opw k. in w] \ | \ w[opw r. P])$$

$$Cleo \triangleq (x). \ k[x. C]$$

**Cleo | Wally**

$$\rightarrow^* (\forall w r) \ ( (x). \ k[x. C] \ | \ \langle in r \rangle \ | \ r[opw k. in w] \ | \ w[opw r. P])$$

$$\rightarrow^* (\forall w r) \ ( k[\langle in r. C \rangle] \ | \ r[opw k. in w] \ | \ w[opw r. P])$$

$$\rightarrow^* (\forall w r) \ ( r[k[C] \ | \ open k. in w] \ | \ w[opw r. P])$$

$$\rightarrow^* (\forall w r) \ ( r[C \ | \ in w] \ | \ w[opw r. P])$$

$$\rightarrow^* (\forall w r) \ ( w[r[C] \ | \ open r. P])$$

$$\rightarrow^* (\forall w) \ ( w[C \ | \ P])$$

• Prone to a “stowaway attack”.

Firewall Crossing (buggy)
Firewall Crossing

- Assume that the shared key $k$ is already known to the firewall and the client, and that $w$ is the secret name of the firewall.

$$\text{Wally} \triangleq (\forall w) (k[\text{in } k. \text{ in } w] | w[\text{open } k. \text{ P}])$$

$$\text{Cleo} \triangleq k[\text{open } k. \text{ C}]$$

$$\text{Cleo} | \text{Wally}$$

$$\rightarrow^* (\forall w) (k[\text{open } k. \text{ C}] | k[\text{in } k. \text{ in } w] | w[\text{open } k. \text{ P}])$$

$$\rightarrow^* (\forall w) (k[k[\text{in } w] | \text{open } k. \text{ C}] | w[\text{open } k. \text{ P}])$$

$$\rightarrow^* (\forall w) (k[\text{in } w | \text{C}] | w[\text{open } k. \text{ P}])$$

$$\rightarrow^* (\forall w) w[k[C] | \text{open } k. \text{ P}]$$

$$\rightarrow^* (\forall w) w[C | P]$$
The Asynchronous $\pi$-Calculus

- A named channel is represented by an ambient.
  - The name of the channel is the name of the ambient.
  - Communication on a channel is becomes local I/O inside a channel-ambient.
  - A conventional name, $io$, is used to transport I/O requests into the channel.

$$(ch \ n)P \triangleq (\forall n) (n[!open \ io] \parallel P)$$

$$n(x).P \triangleq (\forall p) (io[in \ n. \ (x). \ p[\text{out} \ n. \ P]] \parallel open \ p)$$

$$n(M) \triangleq io[in \ n. \ \langle M \rangle]$$

- These definitions satisfy the expected reduction in presence of a channel for $n$:

$$n(x).P \parallel n(m) \rightarrow^* P\{x \leftarrow m\}$$
• **Full translation**

\[
\begin{align*}
\langle (\forall n) P \rangle & \triangleq (\forall n) (n[!\text{open } \text{io}] \mid \langle P \rangle) \\
\langle n(x). P \rangle & \triangleq (\forall p) (\text{io}[\text{in } n. (x). p[\text{out } n. \langle P \rangle)] \mid \text{open } p) \\
\langle n\langle m \rangle \rangle & \triangleq \text{io}[\text{in } n. \langle m \rangle] \\
\langle P \mid Q \rangle & \triangleq \langle P \rangle \mid \langle Q \rangle \\
\langle !P \rangle & \triangleq !\langle P \rangle
\end{align*}
\]

• The choice-free synchronous π-calculus, can be encoded within the asynchronous π-calculus.

• The λ-calculus can be encoded within the asynchronous π-calculus.
‘Bigger’

• Ambients is certainly “bigger” than $\pi$.

• We initially strived for the smallest possible set of primitives, compatibly with our design principles. in-out-open are Turing-complete (even without I/O). Hard to find a smaller such set for tree operations.

• Several new versions of the Ambient Calculus primitives have been proposed:
  • They each have their merits in terms of design principles that the original Ambient Calculus does not capture or enforce.
  • They lead to even “bigger” calculi. But the features provided by Safe Ambients and Boxed Ambients (and probably more) are certainly needed in a programming language.
  • Nobody has proposed a variation that is “smaller” than the original Ambient Calculus.
The Tram Protocol

• Example:
  • A tram goes back and forth along a line with several stops.
  • A tram leaves a stop whenever it feels like.
  • A passenger can jump on any available tram.
  • A passenger cannot enter or leave a tram between stations.

• Exercise:
  • Code this in the Ambient Calculus.
The Golf Cart Protocol

• Example:
  • A golf cart carries at most one passenger. When empty, it moves randomly between “holes”.
  • A passenger can hail a golf cart. An empty golf cart will not ignore a passenger.
  • The passenger can then tell the golf cart where to go. The golf cart will then go there (without leaving the passenger behind).
  • The passenger cannot exit the golf cart until the destination.
  • The golf cart cannot leave again until the passenger has disembarked.

• Exercise:
  • Try coding this example in Ambients, Safe Ambients, and Boxed Ambients.
Think!

- To what extent is the Ambient Calculus (or its variations) WAN-sound and WAN-complete?