Foundations of Distributed Programming Languages

Peter Sewell

University of Cambridge

July 6, 1998

Overview

1. - Background: π calculus reduction semantics and programming
   - Problems arising from distribution
2. - Background: π calculus labelled transition semantics and bisimulations
   - Observational equivalences for concurrent languages
   - Distributed infrastructure for mobile agents

Background: π calculus reduction semantics and programming

π-calculus: example

π-calculus: syntax

Names of channels $a, b, \ldots$

Process terms $P, Q ::=$

0	nil
$P \mid Q$	parallel composition
$c!v$	output v on channel c
$c?w.P$	input from channel c
$*c?w.P$	replicated input from channel c
$\mathbf{new}\ c\ \mathbf{in}\ P$	new channel name creation

w binding in inputs, c binding in new

Reduction

$$P \longrightarrow Q$$

π-calculus: example reductions

Communication:

$$x!7 \mid x?u.y!u \;\longrightarrow\; \{7/u\}(y!u) = y!7$$

Many-to-one communication, nondeterminism:

$$x!7 \mid x!8 \mid x?u.y!u \quad \begin{array}{l} \longrightarrow\; x!8 \mid y!7 \\ \\ \longrightarrow\; x!7 \mid y!8 \end{array}$$

Restriction:

$$x!7 \mid \textbf{new}\ x\ \textbf{in}\ (x!8 \mid x?u.y!u) \;\longrightarrow\; x!7 \mid \textbf{new}\ x\ \textbf{in}\ y!8$$

Name passing:

$$x!y \mid x?u.u!5 \;\longrightarrow\; y!5$$

Scope extrusion:

$$\begin{aligned}(\textbf{new}\ y\ \textbf{in}\ x!y \mid y?v.P) \mid x?u.u!5 &\;\longrightarrow\; \textbf{new}\ y\ \textbf{in}\ y?v.P \mid y!5 \\ &\;\longrightarrow\; \textbf{new}\ y\ \textbf{in}\ \{5/v\}P\end{aligned}$$

Replication:

$$x!7 \mid *x?u.y!u \;\longrightarrow\; y!7 \mid *x?u.y!u$$

π-calculus: reduction semantics

Structural congruence:

$$\begin{aligned}
P \mid 0 &\equiv P \\
P \mid Q &\equiv Q \mid P \\
P \mid (Q \mid R) &\equiv (P \mid Q) \mid R \\
P \mid \textbf{new } x \textbf{ in } Q &\equiv \textbf{new } x \textbf{ in } (P \mid Q) \quad x \notin \mathrm{fn}(P)
\end{aligned}$$

Reduction:

$$c!v \mid c?w.P \longrightarrow \{v/w\}P$$

$$c!v \mid *c?w.P \longrightarrow (\{v/w\}P) \mid *c?w.P$$

$$\frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}$$

$$\frac{P \longrightarrow P'}{\textbf{new } x \textbf{ in } P \longrightarrow \textbf{new } x \textbf{ in } P'}$$

$$\frac{P \equiv P' \longrightarrow P'' \equiv P'''}{P \longrightarrow P'''}$$

π-calculi

Modelling *vs* Programming

This calculus is

- *asynchronous* – it has only $c!v$ not $c!v.P$.
- it is *choice-free* – no $P + Q$
- it has only *replicated input* $*c?w.P$, not general replication $*P$

I'll freely use *polyadic* and *matching* forms:

$c![v_1 \,..\, v_n]$	output of a tuple
$c?(w_1 \,..\, w_n).P$	input of a tuple
$\textbf{if } a = b \textbf{ then } P \textbf{ else } Q$	name equality testing

Programming idioms

Process abstractions:

$$
\begin{array}{l}
\textbf{new } \mathit{plustwo} \textbf{ in} \\
\quad *\mathit{plustwo}?(x\ r).r!(x+2) \\
\mid \textbf{new } r \textbf{ in} \\
\quad \mathit{plustwo}![56\ r] \mid r?z.\mathit{printi}!z
\end{array}
$$

Locks and methods:

$$
\begin{array}{l}
\textbf{new } \mathit{lock} \textbf{ in} \\
\quad \mathit{lock}![] \\
\quad \mid *\mathit{method1}?\mathit{arg}. \\
\qquad \mathit{lock}?(). \\
\qquad \ldots \\
\qquad \mathit{lock}![] \\
\\
\quad \mid *\mathit{method2}?\mathit{arg}. \\
\qquad \mathit{lock}?(). \\
\qquad \ldots \\
\qquad \mathit{lock}![]
\end{array}
$$

Objects:

$$[\mathit{method1}\ \ \mathit{method2}]$$

Lambda calculus encodings...

π Programming: Pict

Pict developed since 1992 by Pierce and Turner to:

- Experiment with programming in the π-calculus

Sequential (functional)	Concurrent
λ-calculus	π-calculus
functions	processes
function application	parallel composition
beta reduction	communication
LISP, ML, Haskell, etc.	Pict

- Experiment with rich type systems for communicating concurrent objects
- Experiment with efficient implementation techniques

Documentation and an implementation available [PT98, PT97].

Problems arising from distribution

Distribution

Only concurrency so far – good for certain applications (e.g. GUIs) but the real interest is in *distribution*.

π fits in some ways:

- concurrency
- π async reliable message passing $\sim$ reliable datagrams – not far above IP
- naming. Analogies:
  - communication channels (read/write)
  - references (deref/assign)
  - cryptographic keys (decrypt/encrypt)

  π-style semantics provide a tractable and compositional way of describing systems that can locally generate fresh names

π doesn't fit in other ways:

- local/non-local performance issues
- many-to-many communication
- failure (machines and communication links), time
- code and agent migration
- security
- quality of service

These cannot be abstracted away – there are many language design/semantics problems, and in some cases delicate protocol design problems.

Some have been studied in the setting of particular calculi...

Modelling Facile failure semantics – π_l – Amadio and Prasad [AP94]

Basis for a mobile agent language – **DJoin** – Fournet et al [FGL+96],

Reasoning about security protocols – **Spi** – Abadi and Gordon [AG97]

Typing for locality enforcement of capabilities – **dpi** – Sewell [Sew97]

Semantics of failure; typing for resource control – **D**π – Riely and Hennessy[RH97, RH98]

Modelling administrative domains – **Ambient** – Cardelli and Gordon [CG98]

Design and reasoning for mobile agent infrastructures – **Dot** – Sewell, Wojciechowski and Pierce [SWP98]

Resource access control for mobile agents – **Seal** – Vitek and Castagna [VC98]

Units of X

Suppose we have processes

$$P \stackrel{def}{=} P_1 \,|\, P_2$$
$$Q \stackrel{def}{=} Q_1 \,|\, Q_2$$

then in π we might have

$$\begin{aligned} P \,|\, Q \quad &\equiv \quad (P_2 \,|\, Q_1) \,|(Q_2 \,|\, P_1) \\ &\longrightarrow \ldots \longrightarrow \; R_1 \,|\, .. \,|\, R_n \end{aligned}$$

– the identity of the components is lost.

Therefore must add primitives for grouping process terms, into units of:

- failure (eg machines or runtime system instances)
- migration (eg mobile agents)
- trust (eg large admin domain or small secure critical regions)

Uniquely named, non-uniquely named, or anonymous

flat, two-level, or hierachical

dynamically generable or not

DJoin locations

Many-to-one communication

A straightforward combination of π channel communication and join locations would allow inputs on a channel x in any location. How to implement?

Need a queue for x – must be located – then an output, input and channel may all be on different machines.

Join patterns combine restriction, replicated input and linearity –

$$\mathbf{def}\ x?w.P\ \mathbf{in}\ Q$$

is similar to

$$\mathbf{new}\ x\ \mathbf{in}\ Q \,|\, {*}x?w.P$$

Join patterns:

$$\mathbf{def}\ (x_1?w_1 \wedge\ ..\ \wedge x_n?w_n).P\ \mathbf{in}\ Q$$

Example reduction

$$\begin{array}{l}\mathbf{def}\ (x_1?w_1 \wedge x_2?w_2).P\ \mathbf{in}\ Q \,|\, x_1!3 \,|\, x_2!7 \\ \longrightarrow \\ \mathbf{def}\ (x_1?w_1 \wedge x_2?w_2).P\ \mathbf{in}\ Q \,|\{3,7/w_1,w_2\}P\end{array}$$

Background: π calculus labelled transition semantics and bisimulations

Operational Congruences

What is the behaviour of a sub-process?

How should the semantics relate to the behaviour of an implementation?

- when is an abstract machine correct?
- when is a program transformation correct?

Several congruence relations have been defined....

Broadly:

1. Bisimulations over some *labelled transition system*
2. Bisimulations induced by some *barbs*
3. Relations induced by some *observations*

Labelled Transition Semantics

Early Transitions:

output	input	internal
$A \vdash P \xrightarrow{x!\vec{z}} Q$	$A \vdash P \xrightarrow{x?\vec{z}} Q$	$A \vdash P \xrightarrow{\tau} Q$

where $\mathrm{fn}(P) \subseteq A$.

Output

$$\{x, z\} \vdash x!z \xrightarrow{x!z} 0$$

Input

$$\{x\} \vdash x?y.P \xrightarrow{x?z} \{z/y\}P$$

if $\mathrm{fn}(P) \subseteq \{x\}$.

Bound output

$$\{x\} \vdash \mathbf{new}\ z\ \mathbf{in}\ x!z \xrightarrow{x!w} 0$$

for any $w \neq x$.

Communication rule

$$\frac{A \vdash P \xrightarrow{x!\vec{z}} P' \qquad A \vdash Q \xrightarrow{x?\vec{z}} Q'}{A \vdash P \,|\, Q \xrightarrow{\tau} \mathbf{new}\ \vec{z} - A\ \mathbf{in}\ P' \,|\, Q'}$$

SOS

An inductive-on-terms definition of the labelled transitions:

Out	$A \vdash x!\vec{z} \xrightarrow{x!\vec{z}} 0$
In	$A \vdash x?\vec{y}.P \xrightarrow{x?\vec{z}} \{\vec{z}/\vec{y}\}P$
Rep	$A \vdash *x?\vec{y}.P \xrightarrow{x?\vec{z}} (\{\vec{z}/\vec{y}\}P) \mid *x?\vec{y}.P$
Par-com	$\dfrac{A \vdash P \xrightarrow{x!\vec{z}} P' \qquad A \vdash Q \xrightarrow{x?\vec{z}} Q'}{A \vdash P \mid Q \xrightarrow{\tau} \mathbf{new}\ \vec{z} - A\ \mathbf{in}\ P' \mid Q'}$ and sym.
Par-cong	$\dfrac{A \vdash P \xrightarrow{\alpha} P'}{A \vdash P \mid Q \xrightarrow{\alpha} P' \mid Q}$ and sym.
New-open	$\dfrac{A, x \vdash P \xrightarrow{w!\vec{z}} P' \qquad w \neq x \in \vec{z}}{A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{w!\vec{z}} P'}$
New-cong	$\dfrac{A, x \vdash P \xrightarrow{\alpha} P' \qquad x \notin \mathrm{n}(\alpha)}{A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{\alpha} \mathbf{new}\ x\ \mathbf{in}\ P'}$
Struct-Cong-Right	$\dfrac{A \vdash P \xrightarrow{\alpha} P' \quad P' \equiv P''}{A \vdash P \xrightarrow{\alpha} P''}$

Basic Properties

1 Struct-Cong-Left

$$\frac{P \equiv P' \qquad A \vdash P' \xrightarrow{\alpha} P''}{A \vdash P \xrightarrow{\alpha} P''}$$

is admissible.

2 If $\mathrm{fn}(P) \subseteq A$ then $A \vdash P \xrightarrow{\tau} Q$ iff $P \longrightarrow Q$.

Strong bisimulation

Bisimulation $\dot{\sim}_A$ is the largest family of symmetric relations such that if $P \mathrel{\dot{\sim}_A} Q$ then $\mathrm{fn}(P) \cup \mathrm{fn}(Q) \subseteq A$ and

$A \vdash P \xrightarrow{\alpha} P'$ implies $\exists Q' \,.\, A \vdash Q \xrightarrow{\alpha} Q' \wedge P' \mathrel{\dot{\sim}_{A \cup \mathrm{n}(\alpha)}} Q'$

Bisimulation congruence $\sim$ is defined by closing under name substitutions:

$$P \sim Q \overset{\mathit{def}}{\Leftrightarrow} \forall \sigma \,.\, \sigma P \mathrel{\dot{\sim}_A} \sigma Q$$

where $A \overset{\mathit{def}}{=} \mathrm{fn}(\sigma P) \cup \mathrm{fn}(\sigma Q)$.

3 $\sim$ is a congruence relation

I've not checked any of these for exactly this π-calculus!

Barbed Bisimulation

Say

$$P{\downarrow}_{a!} \overset{def}{\Leftrightarrow} \exists A, \vec{z}, Q \;.\; A \vdash P \xrightarrow{a!\vec{z}} Q$$
$$P{\downarrow}_{a?} \overset{def}{\Leftrightarrow} \exists A, \vec{z}, Q \;.\; A \vdash P \xrightarrow{a?\vec{z}} Q$$

Barbed bisimulation $\dot{\sim}_{\text{barbed}}$ is the largest symmetric relation such that if $P \;\dot{\sim}_{\text{barbed}}\; Q$ then

- $P{\downarrow}_\beta$ implies $Q{\downarrow}_\beta$
- $P \longrightarrow P'$ implies $\exists Q' \;.\; Q \longrightarrow Q' \wedge P' \;\dot{\sim}_{\text{barbed}}\; Q'$

It is very coarse – eg $x?y.P \;\dot{\sim}_{\text{barbed}}\; x?y.Q$

Barbed bisimulation congruence $\sim_{\text{barbed}}$ is the largest congruence contained in $\dot{\sim}_{\text{barbed}}$:

$$P \sim_{\text{barbed}} Q \overset{def}{\Leftrightarrow} \forall C \;.\; C[P] \;\dot{\sim}_{\text{barbed}}\; C[Q]$$

For some calculi barbed bisimulation congruence coincides nicely with a bisimulation congruence defined over a LTS.

Observational equivalences for concurrent languages – Pict case study

Pict

$$
\begin{tikzcd}
\text{High Level Language} \arrow[d, "\text{Type Inference and CPS translation}"'] & \\
\text{Core Pict} \approx \pi \arrow[r, "\text{Structural Operational Semantics}"] \arrow[d, "\text{Code Generation}"'] & \pi\text{-LTS} \\
\text{C} &
\end{tikzcd}
$$

Implementation Correctness:

1. strong enough for programmers
2. weak enough for implementers
3. mathematically tractable for correctness proofs

Implementation

Uniprocessor

How to resolve nondeterminism?

Have state:

- run queue, of processes waiting to be scheduled (round robin)
- channel queues, of pending input or output processes on each channel (FIFO)

e.g.

$$\textbf{new } x \textbf{ in } (x!3 \mid x?y.P) \xrightarrow{\tau} \textbf{new } x \textbf{ in } \{3/y\}P$$

Branching or Linear time

e.g. $P \stackrel{def}{=} \textbf{new}\ x\ \textbf{in}\ (x! \mid x?.\text{print!``This''} \mid x?.\text{print!``That''})$

SOS	Implementation
$P \xrightarrow{\tau}\ \xrightarrow{\text{print!``This''}}$ $P \xrightarrow{\tau}\ \xrightarrow{\text{print!``That''}}$	unspecified...

Termination

e.g. 0

SOS	Implementation
$0 \not\longrightarrow$	Returns to Unix shell

Divergence

e.g. $D \stackrel{def}{=} \textbf{new}\ x\ \textbf{in}\ (x! \mid *x?.x!)$

SOS	Implementation
$D \xrightarrow{\tau} D$	Never returns

Congruence

External Input/Output

Library primitives for interaction with Unix and X windows.

$$\begin{aligned}\Gamma_l = \qquad\qquad \mathrm{print}&: ![\mathrm{String}]\\ \mathrm{pr}&: ![\mathrm{String}, \updownarrow[]]\\ \mathrm{f}&: ![\mathrm{ArgType}, \updownarrow[\mathrm{ResultType}]]\\ &\dots\\ \mathrm{getXEvent}&: ?[\mathrm{XEvent}]\end{aligned}$$

Programs are processes P such that $\Gamma_l \vdash P$.

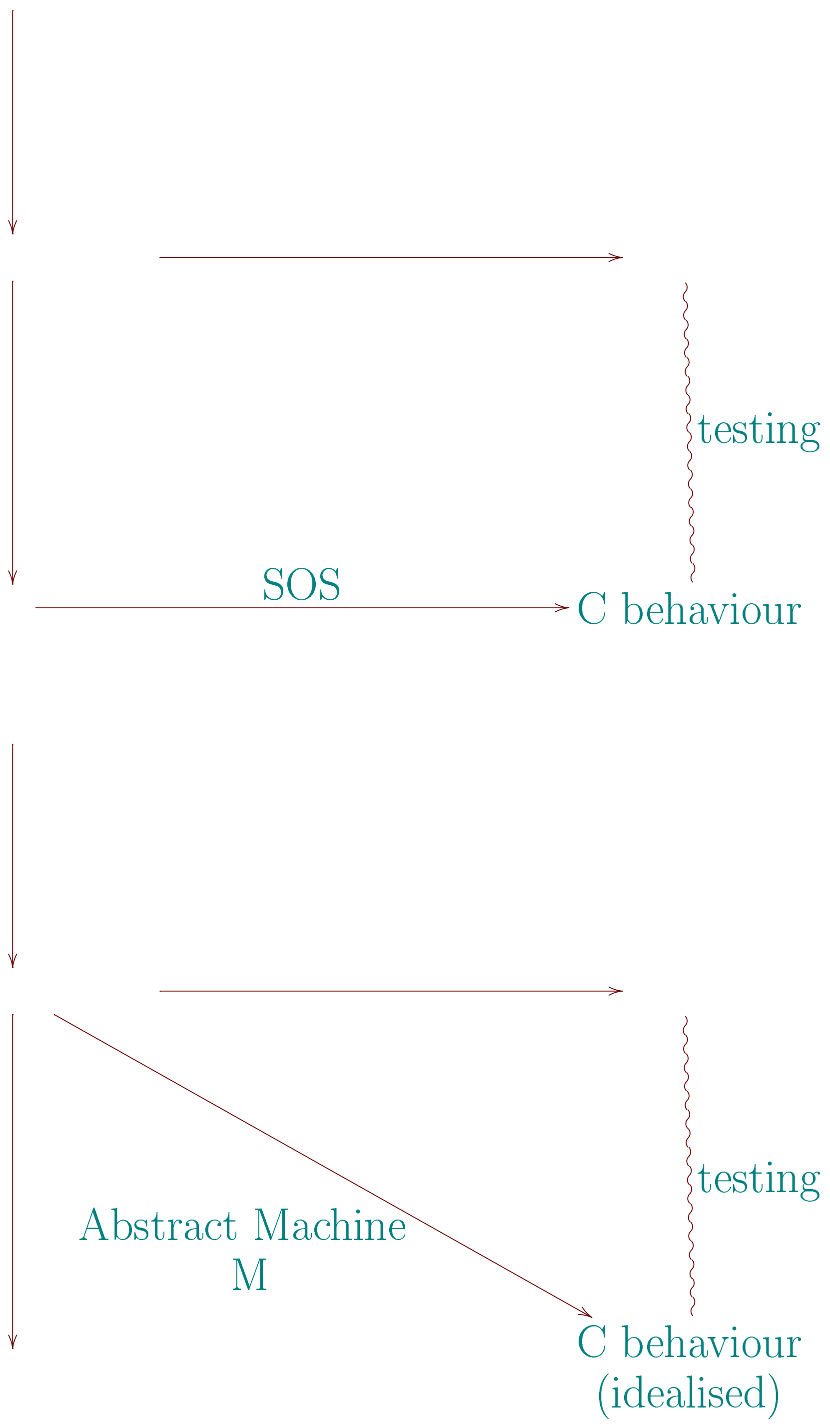
SOS	Implementation
$\Gamma_l \vdash P \xrightarrow{\mathrm{print}!\text{“Hello”}} P'$	If print!. . . is at the front of the run queue the C library routine `printf(...)` is called

$$
\begin{tikzcd}
\text{High Level Language} \arrow[d, swap, "\begin{array}{c}\text{Type Inference}\\ \text{and CPS translation}\end{array}"] & \\
\text{Core Pict} \approx \ \pi \arrow[r, "\text{SOS}"] \arrow[d, swap, "\text{Code Generation}"] & \pi\text{-LTS} \\
\text{C} &
\end{tikzcd}
$$

A C-behaviour is an LTS with labels

$$
\begin{array}{lll}
\mathrm{print}\langle s, \bullet \rangle & \mathrm{return}\langle \bullet, \bullet \rangle & \tau \\
\mathrm{f}\langle \mathrm{arg}, \mathrm{res} \rangle & & \\
\ldots & & \\
\mathrm{XPending}\langle \bullet, n \rangle & & \\
\mathrm{XNextEvent}\langle \bullet, e \rangle & &
\end{array}
$$

satisfying ...


testing
SOS
C behaviour
testing
Abstract Machine
M
C behaviour
(idealised)

SOS	C-behaviour
$\Gamma_l \vdash P \xrightarrow{\text{print!“Hello”}} P'$	$C \xrightarrow{\text{print}\langle\text{“Hello”},\bullet\rangle} C'$

SOS	C-behaviour
$\Gamma_l \vdash P \not\longrightarrow$	
SOS in test harness $P \xrightarrow{\text{return}\langle\bullet,\bullet\rangle}$	$C \xrightarrow{\text{return}\langle\bullet,\bullet\rangle}$

Non-refusable communication

C behaviours must satisfy

1. If $c \xrightarrow{\text{f}\langle\text{arg},\text{res}\rangle} c_1$ then $\forall \text{res}' \;.\; \exists c' \;.\; c \xrightarrow{\text{f}\langle\text{arg},\text{res}'\rangle} c'$.

No external nondeterminism

1. If

$$\begin{tikzcd} & c_1 \\ c \arrow[ur, "l_1"] \arrow[dr, "l_2"'] & \nparallel \\ & c_2 \end{tikzcd}$$

   then either $l_1 = l_2 = \tau$

   or $\exists f, a, r_1, r_2 \;.\; l_1 = \text{f}\langle a, r_1\rangle \wedge l_2 = \text{f}\langle a, r_2\rangle \wedge r_1 \neq r_2$.

2. ...

Blocking Communication and X events

Some C library routines can block:

`XPending(...)`	returns number of events in buffer
`XNextEvent(...)`	returns next event, blocking if the buffer is empty

$$\text{getXEvent}?e.0 \mid \text{print}!\text{“Ping”}$$

To compare SOS and abstract machine must incorporate buffer into test harnesses.

Asynchrony

SOS	C-behaviour
$\Gamma_l \vdash P \xrightarrow{\text{f}![\text{arg}\ x]} Q$	
SOS in test harness	
$P \xrightarrow{\text{f}\langle\text{arg},\text{res}\rangle} \textbf{new}\ x\ \textbf{in}\ (x!\text{res} \mid Q)$	$C \xrightarrow{\text{f}\langle\text{arg},\text{res}\rangle} C'$

Overview

$SOS : \mathrm{Programs} \rightarrow \pi\text{-LTS}$	$M : \mathrm{Programs} \rightarrow \text{C-behaviours}$
Test harness $\mathrm{H}_{\mathrm{sos}}(\_)$: • detects termination • includes event buffer with $\mathrm{getXEvent}$ interface • generates result outputs	Test harness $\mathrm{H}_{\mathrm{am}}(\_)$ • includes event buffer with $\mathrm{XPending}, \mathrm{XNextEvent}$ interface

Enviroments E modelling user and Unix/X etc. (cannot refuse communications)

Composition $\_ \,||\, \_$

Now executions of $E \,||\, \mathrm{H}_{\mathrm{sos}}(SOS(P))$ and $E \,||\, \mathrm{H}_{\mathrm{am}}(M(P))$ can be compared...

Distributed infrastructure for mobile agents

Distributed programming languages

- Static
- Mobile Code
- Mobile Agents

Focus on one problem – the design of communication primitives for agents to interact.

Communication primitives (e.g. async. messages):

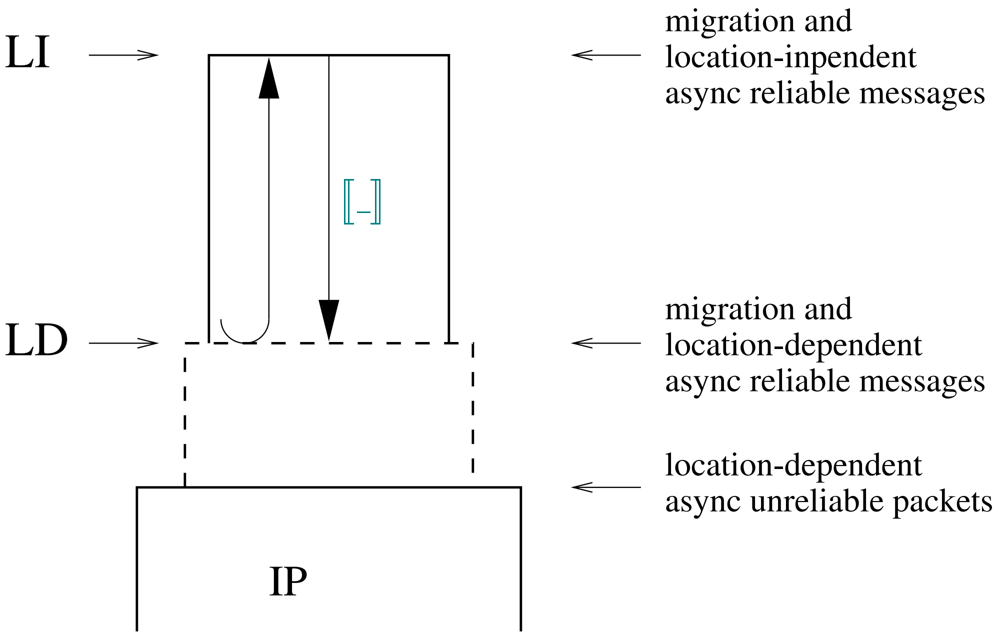
- Location Independent
- Location Dependent
- Adjacent

LI primitives potentially simplify application development

...BUT...

they need distributed infrastructure

Two Levels of Abstraction



define levels by giving process calculi (with reduction semantics)

- Understanding of the algorithms
- Experimental implementation
- Correctness and robustness results
- Heterogeneous infrastructures

Joint work with Pierce, Unyapoth, Wojciechowski

Low- and High-Level Calculi

Names of sites s, of agents a, b

Low-Level:

$\textbf{create}\ a = P\ \textbf{in}\ Q$	agent creation
$\textbf{migrate to}\ s.P$	agent migration
$P \mid Q$	parallel composition
0	nil
$?w.P$	input
$\langle a@s \rangle\ !v$	LD output to agent a on site s
$\textbf{iflocal}\ \langle a \rangle\ !v.P\ \textbf{else}\ Q$	test-and-send

High-Level:

$\langle a@? \rangle\ !v$	LI output to agent a

Low- and High-Level Calculi

Names of sites s, of agents a, b, and of channels c

Low-Level: Agents

create $a = P$ **in** Q	agent creation
migrate to $s.P$	agent migration
$P \mid Q$	parallel composition
0	nil

π-calculus style communication within agents

new c **in** P	new channel name creation
$c!v$	output v in the current agent
$c?w.P$	input from channel c
$*c?w.P$	replicated input from channel c
if $a = b$ **then** P **else** Q	name equality testing

Inter-agent communication

$\langle a@s \rangle c!v$	LD output to agent a on site s
iflocal $\langle a \rangle c!v.P$ **else** Q	test-and-send

High-Level:

$\langle a@? \rangle c!v$	LI output to agent a

Reduction Semantics

Extend syntax:

$@_a\, P$	P as part of agent a
$\mathbf{new}\ a@s\ \mathbf{in}\ P$	new agent name a, currently at site s

Take π structural congruence with

$$\begin{array}{rcll} @_a\,(P \,|\, Q) & \equiv & @_a\, P \,|\, @_a\, Q & \\ @_a\, \mathbf{new}\ c\ \mathbf{in}\ P & \equiv & \mathbf{new}\ c\ \mathbf{in}\ @_a\, P & \text{if } c \neq a \end{array}$$

Take *location contexts* Γ mapping agent names to site names

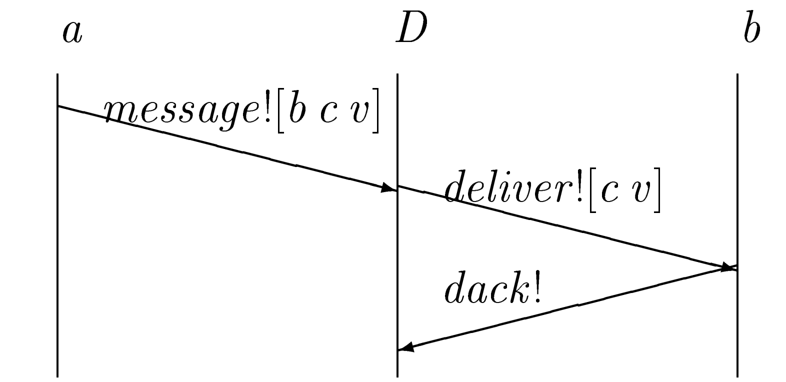
Take reduction axioms

$$\begin{array}{llll} \Gamma, @_a\, \mathbf{migrate\ to}\ s.P & \longrightarrow & (\Gamma \oplus a \mapsto s), @_a\, P & \\ \Gamma, @_a\, \mathbf{iflocal}\ \langle b \rangle c!v.P\ \mathbf{else}\ Q & \longrightarrow & \Gamma, @_b\ c!v | @_a\, P & \text{if } \Gamma(a) = \Gamma(b) \\ & \longrightarrow & \Gamma, @_a\, Q & \text{if } \Gamma(a) \neq \Gamma(b) \\ \Gamma, @_a\,(c!v | c?p.P) & \longrightarrow & \Gamma, @_a\, \{v/p\}P & \end{array}$$

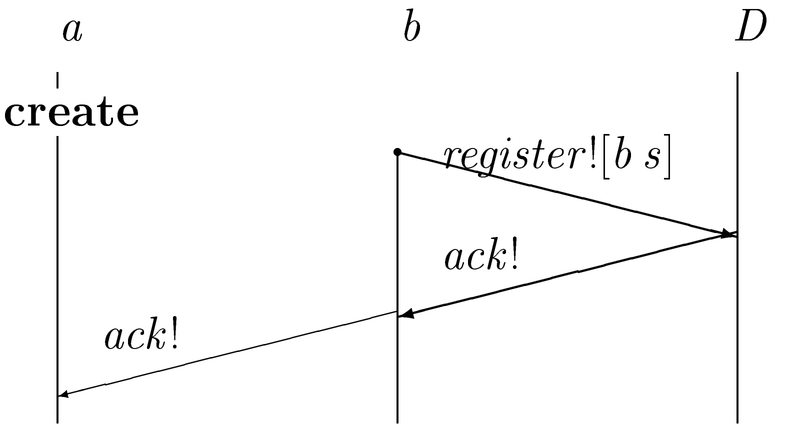
together with 3 others and 4 closure rules...

Example Encoding – Central Daemon

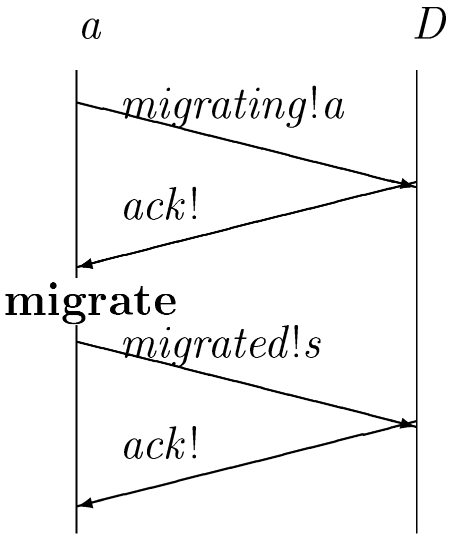
Location-independent output



Creation



Migration



Example Encoding – Central Daemon

$[\![P]\!]$ is roughly $[\![P]\!]_a \mid \mathit{DAEMON}$

$$[\![\langle b@?\rangle c!v]\!]_a \quad = \quad \langle D@Dsite\rangle message![b\ c\ v]$$

$$\begin{array}{lll}
[\![\textbf{create}\ b = P\ \textbf{in}\ Q]\!]_a & = & currentloc?s. \\
& & \quad \textbf{create}\ b = \\
& & \qquad\quad *deliver?c\ v.(\langle D@Dsite\rangle dack![] \mid c!v) \\
& & \qquad \mid \langle D@Dsite\rangle register![b\ s] \\
& & \qquad \mid ack?.(\langle a@s\rangle ack![] \mid currentloc!s \mid [\![P]\!]_b) \\
& & \quad \textbf{in} \\
& & \qquad ack?.(currentloc!s \mid [\![Q]\!]_a)
\end{array}$$

$$\begin{array}{lll}
[\![\langle b@?\rangle c!v]\!]_a & = & \langle D@Dsite\rangle message![b\ c\ v] \\
[\![\textbf{create}\ b = P\ \textbf{in}\ Q]\!]_a & = & currentloc?s. \\
& & \quad \textbf{create}\ b = \\
& & \qquad\quad *deliver?c\ v.(\langle D@Dsite\rangle dack![] \mid c!v) \\
& & \qquad \mid \langle D@Dsite\rangle register![b\ s] \\
& & \qquad \mid ack?.(\langle a@s\rangle ack![] \mid currentloc!s \mid [\![P]\!]_b) \\
& & \quad \textbf{in} \\
& & \qquad ack?.(currentloc!s \mid [\![Q]\!]_a) \\
[\![\textbf{migrate to}\ u.P]\!]_a & = & currentloc?_. \\
& & \qquad \langle D@Dsite\rangle migrating!a \\
& & \quad \mid ack?. \\
& & \qquad \textbf{migrate to}\ u. \\
& & \qquad\qquad \langle D@Dsite\rangle register![a\ u] \\
& & \qquad\quad \mid ack?.(currentloc!u \mid [\![P]\!]_a) \\
[\![0]\!]_a & = & \\
[\![P \mid Q]\!]_a & = & \\
[\![c?w.P]\!]_a & = & \\
[\![*c?w.P]\!]_a & = & \text{ALL HOMOMORPHIC} \\
[\![\textbf{iflocal}\ \langle b\rangle c!v.P\ \textbf{else}\ Q]\!]_a & = & \\
[\![\textbf{new}\ c\ \textbf{in}\ P]\!]_a & = & \\
[\![\textbf{if}\ a = b\ \textbf{then}\ P\ \textbf{else}\ Q]\!]_a & = &
\end{array}$$

$$\begin{array}{l}
\mathit{DAEMON} = \textbf{new}\ lock\ \textbf{in} \\
\quad lock!\textbf{emptymap} \\
\mid *register?a\ s. \\
\quad lock?m. \\
\quad \textbf{let}\ m' = (m\ \textbf{with}\ a \mapsto s)\ \textbf{in} \\
\qquad lock!m' \mid \langle a@s\rangle ack![] \\
\mid *migrating?a. \\
\quad lock?m. \\
\quad \textbf{lookup}\ a\ \textbf{in}\ m\ \textbf{with} \\
\qquad \textbf{found}(s). \\
\qquad\qquad \langle a@s\rangle ack![] \\
\qquad\quad \mid migrated?s. \\
\qquad\qquad \textbf{let}\ m' = (m\ \textbf{with}\ a \mapsto s')\ \textbf{in} \\
\qquad\qquad lock!m' \mid \langle a@s'\rangle ack![] \\
\qquad \textbf{notfound}.0 \\
\mid *message?a\ c\ v. \\
\quad lock?m. \\
\quad \textbf{lookup}\ a\ \textbf{in}\ m\ \textbf{with} \\
\qquad \textbf{found}(s). \\
\qquad\qquad \langle a@s\rangle deliver![c\ v] \\
\qquad\quad \mid dack?.lock!m \\
\qquad \textbf{notfound}.0
\end{array}$$

$$\begin{array}{lll}
[\![@_a\ P]\!]_s & = & @_a\ \textbf{new}\ register, migrating, message, dack, deliver, ack, currentloc\ \textbf{in} \\
& & \quad \textbf{create}\ D = \mathit{DAEMON}\ \textbf{in} \\
& & \quad\quad \textbf{let}\ Dsite = s\ \textbf{in} \\
& & \qquad\qquad *deliver?c\ v.(\langle D@Dsite\rangle dack![] \mid c!v) \\
& & \qquad\quad \mid \langle D@Dsite\rangle register![a\ s] \\
& & \qquad\quad \mid ack?.(currentloc!s \mid [\![P]\!]_a)
\end{array}$$

Language Design/Semantics Problems:

- Adjacency
- Secrecy
- Control of resource use
- Integration with rich language features – higher order functions/objects
- Failure and reconfiguration
- Distributed observational equivalences
- Fairness
- Time
- Quality of service

Web pointers

In the first lecture I shall give a rapid introduction to the Pi-calculus, discuss its use as a foundation for concurrent programming languages, and talk about problems that arise in the distributed setting. Background on the Pi calculus can be found in the `http://www.dcs.ed.ac.uk/lfcsreps/EXPORT/89/ECS-LFCS-89-85/` introductory papers (in 2 parts; by Robin Milner, Joachim Parrow, and David Walker) and a more recent tutorial : `ftp://ftp.cl.cam.ac.uk/users/rm135/ppi.ps.Z` The polyadic pi-calculus: a tutorial, Robin Milner, LFCS technical report ECS-LFCS-91-180. Chapters 1-3 of this are particularly relevant.

The use of the Pi-calculus as the foundation for the concurrent programming language Pict is discussed in `http://www.cs.indiana.edu/hyplan/pierce/pierce/ftp/pict-design.ps.gz` Pict: A Programming Language Based on the Pi-Calculus, Benjamin C. Pierce and David N. Turner.

Some distributed issues are discussed in the main paper on the distributed join calculus: `http://pauillac.inria.fr/ fournet/papers/concur-96.ps.gz` A calculus of mobile agents Cedric Fournet and Georges Gonthier and Jean-Jacques Levy and Luc Maranget and Didier Remy.

The second lecture will be divided into two parts. One part will address the problem of defining observational equivalences for concurrent programming languages, based on

`http://www.cl.cam.ac.uk/users/pes20/index.html#ObsOnPict` On Implementations and Semantics of a Concurrent Programming Language Peter Sewell.

In the other part I will discuss how concurrent calculi and semantics can be used in the design of mobile agent infrastructure algorithms. Some material on this can be found in

`http://www.cl.cam.ac.uk/users/pes20/index.html#LIMA` Location Independence for Mobile Agents, Peter Sewell, Pawel Wojciechowski and Benjamin Pierce

Useful Links

The `http://www.cs.auc.dk/mobility/` Mobility page maintained by Uwe Nestmann.

The material on `http://www.cs.indiana.edu/hyplan/pierce/pierce/629/papers.html` Languages for Programming the Web collected by Benjamin Pierce

My home page `http://www.cl.cam.ac.uk/users/pes20`

References

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the Fourth ACM Conference on Computer and Communications Security, Zürich*, pages 36–47. ACM Press, April 1997.

[AP94] R. M. Amadio and S. Prasad. Localities and failures. In P. S. Thiagarajan, editor, *Proceedings of* 14^{th} *FST and TCS Conference, FST-TCS'94. LNCS 880*, pages 205–216. Springer-Verlag, 1994.

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *Proc. of Foundations of Software Science and Computation Structures (FoSSaCS), ETAPS'98*, March 1998.

[FGL+96] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In *Proceedings of CONCUR '96. LNCS 1119*, pages 406–421. Springer-Verlag, August 1996.

[PT97] Benjamin C. Pierce and David N. Turner. Pict: A programming language based on the pi-calculus. Technical Report CSCI 476, Computer Science Department, Indiana University, 1997. To appear in *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, MIT Press.

[PT98] Benjamin C. Pierce and David N. Turner. Pict, release 4.1. Available from `http://www.cs.indiana.edu/hyplan/pierce/pierce/ftp/pict/Html/Pict.html`, March 1998.

[RH97] James Riely and Matthew Hennessy. Distributed processes and location failures. In *Proceedings of ICALP '97. LNCS 1256*, pages 471–481. Springer-Verlag, July 1997.

[RH98] James Riely and Matthew Hennessy. A typed language for distributed mobile processes. In *Proceedings of the 25th POPL*, January 1998.

[Sew97] Peter Sewell. Global/local subtyping for a distributed π-calculus. Technical Report 435, University of Cambridge, August 1997. Available from `http://www.cl.cam.ac.uk/users/pes20/`.

[SWP98] Peter Sewell, Paweł T. Wojciechowski, and Benjamin C. Pierce. Location independence for mobile agents. In *Workshop on Internet Programming Languages, Chicago*, May 1998.

[VC98] Jan Vitek and Guiseppe Castagna. Towards a calculus of mobile computations. In *Workshop on Internet Programming Languages, Chicago*, May 1998.