Concurrency, Pi, and Semantics for Distributed Systems
(a high-speed overview!)

Peter Sewell

University of Cambridge

October 16, 2001

References mostly in *Applied π – a Brief Tutorial*. Tech Report 498.

Why?

Distributed systems are tricky:

- parallelism $\Longrightarrow$
  - large state spaces ('combinatorial explosion'), but more seriously *complex* state spaces)
  - nondeterminism (and deadlock, divergence,...)
- naming
- partial failure
- malicious (mailicious?) attack
- mobility (of code, of running executables, of devices)
- accumulated historical junk

and rigorous semantics can help, both to analyse what's there and (increasingly? we hope) to guide future system building.

What

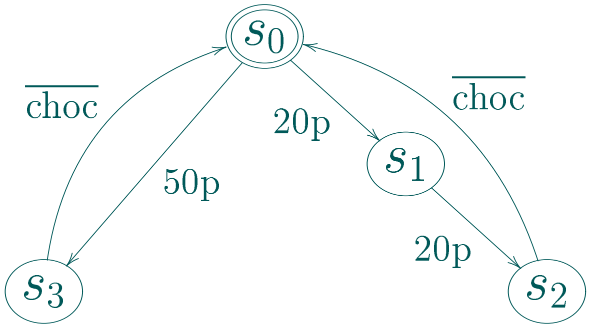
Focus on primitives and operational semantics, highlighting some choices in as simple a setting as possible, and assuming some general familiarity.

No time to talk about proof techniques; not much for particular examples.

1. Concurrency
   (a) labelled transition systems
   (b) equivalences and congruences
   (c) pure CCS-like calculi, with labelled-transition and reduction semantics
   (d) value-passing and name-passing
   (e) π-calculus, with reduction and labelled-transition semantics
2. Distribution
   (a) Naming
   (b) Grouping
   (c) Computation Mobility
   (d) Secure Encapsulation
   (e) Crypto
   (f) Failure

Parallelism and Nondeterminacy – Labelled Transition Systems

$\mathcal{S}_{\text{vend}} \;=$

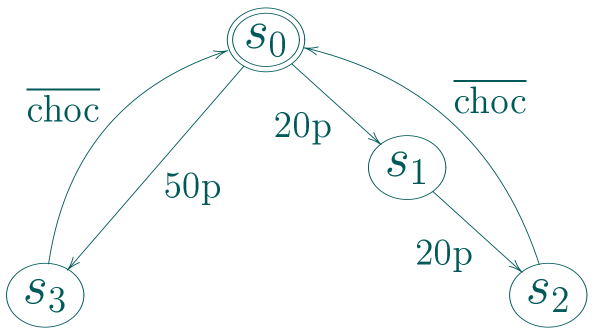


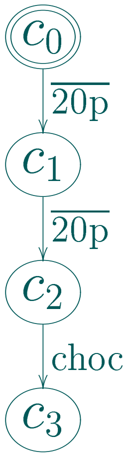
A *labelled transition system* $\mathcal{S}$ is a tuple $\langle L, S, \longrightarrow, s_0 \rangle$

- L is a set, of *labels* $\qquad L = \{20\text{p},\ 50\text{p},\ \overline{\text{choc}}\}$
- S is a set, of *states* $\qquad S = \{s_0,\ s_1,\ s_2,\ s_3\}$
- $\longrightarrow\ \subseteq S \times L \times S$ is the *transition relation* $\qquad$ as above
- $s_0 \in S$ is the *initial state* $\qquad s_0 = s_0$

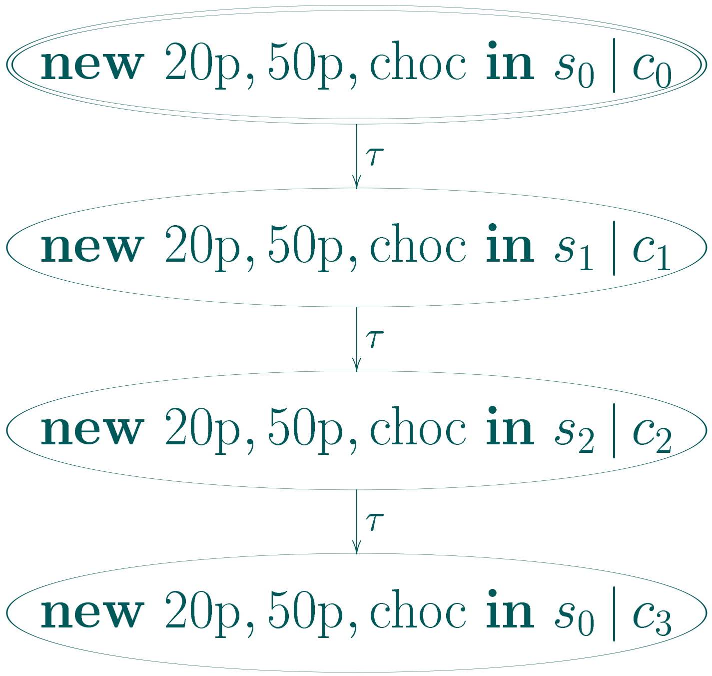
Operations on LTSs – Parallel Composition and Restriction

The vending machine interacting with a sample customer $\mathcal{S}_{\text{cust}}$





$\textbf{new } \text{20p}, \text{50p}, \text{choc } \textbf{in } \mathcal{S}_{\text{vend}} \,|\, \mathcal{S}_{\text{cust}} \ =$



Parallel Composition and Restriction – Precise Definitions

Fix labels $L = \{\, \overline{n} \mid n \in \mathcal{N} \,\} \cup \{\, n \mid n \in \mathcal{N} \,\} \cup \{\tau\}$ (pure CCS labels).

Define $\mathcal{S} \mid \mathcal{T}$ to have states the pairs $s \mid t$, initial state $s_0 \mid t_0$, and transition relation the least such that:

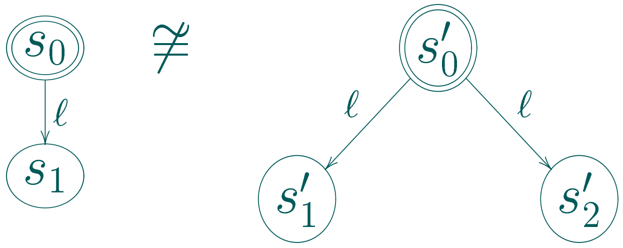
$$\textsc{Par}\ \frac{s \xrightarrow{\ell} s'}{s \mid t \xrightarrow{\ell} s' \mid t} \qquad \textsc{Com}\ \frac{s \xrightarrow{n} s' \quad t \xrightarrow{\overline{n}} t'}{s \mid t \xrightarrow{\tau} s' \mid t'}$$

and symmetric rules.

Define $\textbf{new}\ n\ \textbf{in}\ \mathcal{S}$ to have states $\textbf{new}\ n\ \textbf{in}\ s$ for $s \in S$, initial state $\textbf{new}\ n\ \textbf{in}\ s_0$, and transition relation the least such that:

$$\textsc{Res}\ \frac{s \xrightarrow{\ell} s' \quad n \notin \mathrm{fn}(\ell)}{\textbf{new}\ n\ \textbf{in}\ s \xrightarrow{\ell} \textbf{new}\ n\ \textbf{in}\ s'}$$

LTS isomorphism but



Partial traces

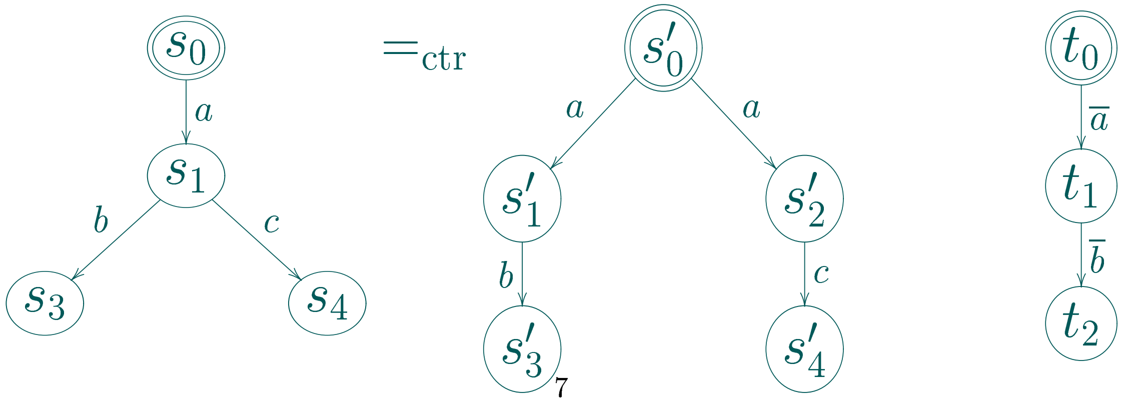
$$\mathrm{ptr}(\mathcal{S}) = \{\, \ell_1 \ldots \ell_n \mid \exists s_1, \ldots, s_n \,.\, s_0 \xrightarrow{\ell_1} s_1 \cdots \xrightarrow{\ell_n} s_n \,\}$$

and say $\mathcal{S} \simeq_{\mathrm{ptr}} \mathcal{S}' \iff \mathrm{ptr}(\mathcal{S}) = \mathrm{ptr}(\mathcal{S}')$, but care about termination/deadlock...

Completed traces

$$\begin{aligned} \mathrm{ctr}(\mathcal{S}) \;=\; & \{\, \ell_1 \ldots \ell_n \mid \exists s_1, \ldots, s_n \,.\, s_0 \xrightarrow{\ell_1} s_1 \cdots \xrightarrow{\ell_n} s_n \not\longrightarrow \,\} \\ \cup \; & \{\, \ell_1 \ell_2 \ldots \mid \exists s_1, s_2, \ldots \,.\, s_0 \xrightarrow{\ell_1} s_1 \xrightarrow{\ell_2} \cdots \,\} \end{aligned}$$

but not a congruence for CCS parallel:



Strong Bisimulation Say $\sim \subseteq S \times T$ is *a* strong bisimulation if for all (s, t) such that $s \sim t$ we have

- if $s \xrightarrow{\ell} s'$ then $\exists t' \,.\, t \xrightarrow{\ell} t' \wedge s' \sim t'$
- if $t \xrightarrow{\ell} t'$ then $\exists s' \,.\, s \xrightarrow{\ell} s' \wedge s' \sim t'$

henceforth write $\sim$ for the largest such.

Theorem 1 (Congruence of $\sim$ for $|$)

$$\frac{s \sim s' \quad t \sim t'}{s \,|\, t \sim s' \,|\, t'}$$

$$\begin{array}{c} \cong \\ | \\ \sim \\ | \\ =_{\mathrm{ctr}} \\ | \\ =_{\mathrm{ptr}} \end{array}$$

Diversion: Alternatives

Other synchronisation primitives For example, take labels $L = \mathcal{N}$ and $A \subseteq \mathcal{N}$

$$\textsc{Com}\ \frac{s \xrightarrow{a} s' \quad t \xrightarrow{a} t' \quad a \in A}{s \,||_A\, t \xrightarrow{a} s' \,||_A\, t'}$$

(consider $\mathcal{S} \,||_A\, \mathcal{T} \,||_A\, \mathcal{U}$). cf modelling vs programming.

Noninterleaving If $\mathcal{S} = \left(s_0 \xrightarrow{a} s_1 \right)$ and $\mathcal{T} = \left(t_0 \xrightarrow{b} t_1 \right)$ then

$$\mathcal{S} \,|\, \mathcal{T} = \begin{tikzcd} & s_1 \,|\, t_0 \arrow[dr, "b"] & \\ s_0 \,|\, t_0 \arrow[ur, "a"] \arrow[dr, "b"'] & & s_1 \,|\, t_1 \\ & s_0 \,|\, t_1 \arrow[ur, "a"'] & \end{tikzcd}$$

emphasising that LTSs and the definition of $|$ reduce parallelism to nondeterministic choices between all interleavings. Sometimes convenient to build more data into the model.

Shared variable models $\{x := 3\} \,||\, \{y := x;\ z := x + 1\}$

Calculi and SOS (Structural Operational Semantics)

So far, been *model-theoretic* – LTSs could have arbitrary sets and relations.

Instead, can consider processes $\mathcal{P}$ defined by a syntax, e.g.

$$
\begin{array}{lll}
P, Q & ::= & 0 & \text{nil} \\
& & P \mid Q & \text{parallel composition of } P \text{ and } Q \\
& & \ell.P & \text{do } \ell \text{ then be } P \\
& & \textbf{new } n \textbf{ in } P & \text{restrict } n \\
& & P + Q & \text{nondeterministic choice between the actions of } P \text{ and } Q
\end{array}
$$

where the prefixes ℓ are n, $\overline{n}$, or τ (exactly the labels from before).

Now define transition relations inductively:

$$
\textsc{Par}\ \frac{P \xrightarrow{\ell} P'}{P \mid Q \xrightarrow{\ell} P' \mid Q}
\qquad
\textsc{Com}\ \frac{P \xrightarrow{n} P' \quad Q \xrightarrow{\overline{n}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}
\qquad
\textsc{Pre}\ \frac{}{\ell.P \xrightarrow{\ell} P}
$$

$$
\textsc{Res}\ \frac{P \xrightarrow{\ell} P' \quad n \notin \mathrm{fn}(\ell)}{\textbf{new } n \textbf{ in } P \xrightarrow{\ell} \textbf{new } n \textbf{ in } P'}
\qquad
\textsc{Sum}\ \frac{P \xrightarrow{\ell} P'}{P + Q \xrightarrow{\ell} P'}
$$

and symmetric rules. Now $\langle L, \mathcal{P}, \longrightarrow, P\rangle$ is an LTS for any process term $P \in \mathcal{P}$.

Reduction Semantics

A different way to define the τ transition relation. Take just

$$
\begin{array}{rcll}
P, Q & ::= & 0 & \text{nil} \\
 & & P \mid Q & \text{parallel composition of } P \text{ and } Q \\
 & & \ell.P & \text{do } \ell \text{ then be } P
\end{array}
$$

then instead of

$$
\textsc{Pre}\,\frac{}{\ell.P \xrightarrow{\ell} P} \qquad \textsc{Par}\,\frac{P \xrightarrow{\ell} P'}{P \mid Q \xrightarrow{\ell} P' \mid Q} \qquad \textsc{Com}\,\frac{P \xrightarrow{n} P' \quad Q \xrightarrow{\overline{n}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}
$$

we define *structural congruence*

$$
\begin{array}{rcl}
P \mid 0 & \equiv & P \\
P \mid Q & \equiv & Q \mid P \\
P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R
\end{array}
$$

and *reduction*

$$
\textsc{Par}\,\frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \qquad \textsc{Com}\,\frac{}{n.P \mid \overline{n}.Q \longrightarrow P \mid Q} \qquad \textsc{Str}\,\frac{P \equiv P' \longrightarrow P'' \equiv P'''}{P \longrightarrow P'''}
$$

Theorem 2 $P \longrightarrow Q$ *iff* $P \xrightarrow{\tau} \equiv Q$.

Value Passing

Allow channels to carry values, so instead of pure outputs $\overline{n}.P$ and inputs $n.Q$ allow e.g. $\overline{n}\langle 15, 3\rangle.P$ and $n\langle x_1, x_2\rangle.Q$.

Value 6 being sent along channel x:

$$\overline{x}6 \mid xu.\overline{y}u \xrightarrow{\tau} \{6/u\}(\overline{y}u) = \overline{y}6$$

Tuple values (and tuple patterns):

$$\overline{x}\langle 8, 3\rangle \mid x\langle z_1, z_2\rangle.\overline{y}z_1 \xrightarrow{\tau} \{\langle 8, 3\rangle/\langle z_1, z_2\rangle\}(\overline{y}z_1) = \overline{y}8$$

Many outputs on the same channel competing for the same input:

$$\begin{tikzcd}
& \overline{x}6 \mid \overline{y}5 \\
\overline{x}5 \mid \overline{x}6 \mid xu.\overline{y}u \arrow[ur] \arrow[dr] & \\
& \overline{x}5 \mid \overline{y}6
\end{tikzcd}$$

Many inputs on the same channel competing for an output:

$$\begin{tikzcd}
& \overline{y}5 \mid xu.\overline{z}u \\
\overline{x}5 \mid xu.\overline{y}u \mid xu.\overline{z}u \arrow[ur] \arrow[dr] & \\
& xu.\overline{y}u \mid \overline{z}5
\end{tikzcd}$$

Restricted names are different from all others:

$$\begin{tikzcd}
\overline{x}5 \mid \textbf{new } x \textbf{ in } (\overline{x}6 \mid xu.\overline{y}u) \arrow[r] \arrow[d, equal] & \overline{x}5 \mid \textbf{new } x \textbf{ in } \overline{y}6 \arrow[d, equal] \\
\overline{x}5 \mid \textbf{new } x' \textbf{ in } (\overline{x'}6 \mid x'u.\overline{y}u) \arrow[r] & \overline{x}5 \mid \textbf{new } x'' \textbf{ in } \overline{y}6
\end{tikzcd}$$

Name passing

Now allow those values to include channel names.

A name received on a channel can then be used itself as a channel name for output or input – here y is received on x and then used to output 7:

$$\overline{x}y \mid xu.\overline{u}7 \longrightarrow \overline{y}7$$

Finally, a restricted name can be sent outside its original scope. Here y is sent on channel x outside the scope of the $\textbf{new } y \textbf{ in}$ binder, which must therefore be moved (with care, to avoid capture of free instances of y). This is *scope mobility*:

$$\begin{array}{rl} (\textbf{new } y \textbf{ in } \overline{x}y \mid yv.P) \mid xu.\overline{u}7 & \longrightarrow \textbf{new } y \textbf{ in } yv.P \mid \overline{y}7 \\ & \longrightarrow \textbf{new } y \textbf{ in } \{7/v\}P \end{array}$$

$$\begin{array}{rcl} (\textbf{new } y \textbf{ in } \overline{x}y \mid yv.P) \mid xu.\overline{u}7 & \equiv & (\textbf{new } y \textbf{ in } \overline{x}y \mid yv.P \mid xu.\overline{u}7) \\ & \longrightarrow & \textbf{new } y \textbf{ in } yv.P \mid \overline{y}7 \\ & \longrightarrow & \textbf{new } y \textbf{ in } \{7/v\}P \end{array}$$

The Simplest π-Calculus: Reduction Semantics

Syntax

P, Q	$::=$	0	nil
		$P \mid Q$	parallel composition of P and Q
		$\overline{c}v$	output v on channel c
		$cw.P$	input from channel c
		$\mathbf{new}\ c\ \mathbf{in}\ P$	new channel name creation

Structural Congruence

$$
\begin{array}{rcll}
P \mid 0 & \equiv & P & \\
P \mid Q & \equiv & Q \mid P & \\
P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R & \\
\mathbf{new}\ x\ \mathbf{in}\ \mathbf{new}\ y\ \mathbf{in}\ P & \equiv & \mathbf{new}\ y\ \mathbf{in}\ \mathbf{new}\ x\ \mathbf{in}\ P & \\
P \mid \mathbf{new}\ x\ \mathbf{in}\ Q & \equiv & \mathbf{new}\ x\ \mathbf{in}\ (P \mid Q) & \qquad x \notin \mathrm{fn}(P)
\end{array}
$$

Reduction

$$
\textsc{Com}\quad \frac{}{\overline{c}v \mid cw.P \longrightarrow \{v/w\}P}
\qquad\qquad
\textsc{Par}\quad \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}
$$

$$
\textsc{Res}\quad \frac{P \longrightarrow P'}{\mathbf{new}\ x\ \mathbf{in}\ P \longrightarrow \mathbf{new}\ x\ \mathbf{in}\ P'}
\qquad\qquad
\textsc{Struct}\quad \frac{P \equiv P' \longrightarrow P'' \equiv P'''}{P \longrightarrow P'''}
$$

Modelling vs Programming: Choices of Primitives

- Monadic
- No replicated input $!xy.P$ or full replication $!P \equiv P \,|!P$
- No name (in)equality testing (natural to have in a programming lang)
- No $+$ (don't seem to need arbitrary $P + Q$ for programming; can code some choice)
- Asynchronous (fits with asychronous messaging. Can encode synchronous communication – in some sense)
- No process-passing, or higher order values

Facile, CML, Pict, ...

Programming: Examples and Pictish Idioms

Process abstractions:

$$\begin{array}{l}
\textbf{new}\ \mathit{plustwo}\ \textbf{in} \\
\quad !\mathit{plustwo}\langle x\ r\rangle.\overline{r}(x+2) \\
\quad \mid\ \textbf{new}\ r\ \textbf{in} \\
\qquad \overline{\mathit{plustwo}}\langle 56\ r\rangle \mid rz.\overline{\mathit{printi}}\,z
\end{array}$$

Locks and methods:

$$\begin{array}{l}
\textbf{new}\ \mathit{lock}\ \textbf{in} \\
\quad \overline{\mathit{lock}}\langle\rangle \\
\quad \mid\ !\mathit{method1}\ \mathit{arg}. \\
\qquad \mathit{lock}\langle\rangle. \\
\qquad\quad \ldots \\
\qquad\quad \overline{\mathit{lock}}\langle\rangle \\
\\
\quad \mid\ !\mathit{method2}\ \mathit{arg}. \\
\qquad \mathit{lock}\langle\rangle. \\
\qquad\quad \ldots \\
\qquad\quad \overline{\mathit{lock}}\langle\rangle
\end{array}$$

Objects:

$$\langle \mathit{method1}\ \mathit{method2}\rangle$$

The Simplest π-Calculus: Labelled Transition Semantics

The labelled transition relation has the form

$$A \vdash P \xrightarrow{\ell} Q$$

where A is a finite set of names, $\mathrm{fn}(P) \subseteq A$, and ℓ is from

$$\begin{array}{lll} \ell & ::= & \tau & \text{internal action} \\ & & \overline{x}v & \text{output of } v \text{ on } x \\ & & xv & \text{input of } v \text{ on } x \end{array}$$

Output of a free name:

$$\{x, y\} \vdash \overline{x}y \xrightarrow{\overline{x}y} 0$$

Output of a new name (for *any* $w \neq x$):

$$\{x\} \vdash \mathbf{new}\ \hat{y}\ \mathbf{in}\ \overline{x}\hat{y} \xrightarrow{\overline{x}w} 0$$

Input of a name:

$$A \vdash xu.P \xrightarrow{xw} \{w/u\}P$$

Scope extrusion (example from Slide 13):

$$
\textsc{Com}\;\dfrac{\textsc{Open}\;\dfrac{\textsc{Par}\;\dfrac{\textsc{Out}\;\dfrac{}{\{x,y\} \vdash \overline{x}y \xrightarrow{\overline{x}y} 0}}{\{x,y\} \vdash \overline{x}y \mid yv.P \xrightarrow{\overline{x}y} 0 \mid yv.P}}{\{x\} \vdash \textbf{new } \hat{y} \textbf{ in } \overline{x}\hat{y} \mid \hat{y}v.P \xrightarrow{\overline{x}y} 0 \mid yv.P} \qquad \textsc{In}\;\dfrac{}{\{x\} \vdash xy.\overline{u}7 \xrightarrow{xy} \{y/u\}\overline{u}7 = \overline{y}7}}{\{x\} \vdash (\textbf{new } \hat{y} \textbf{ in } \overline{x}\hat{y} \mid \hat{y}v.P) \mid (xy.\overline{u}7) \xrightarrow{\tau} \textbf{new } \hat{\hat{y}} \textbf{ in } 0 \mid \hat{\hat{y}}v.P \mid \overline{\hat{\hat{y}}}7}
$$

SOS rules

$$
\textsc{Out}\;\dfrac{}{A \vdash \overline{x}v \xrightarrow{\overline{x}v} 0} \qquad \textsc{In}\;\dfrac{}{A \vdash xp.P \xrightarrow{xv} \{{}^{v}\!/\!_{p}\}P}
$$

$$
\textsc{Par}\;\dfrac{A \vdash P \xrightarrow{\ell} P'}{A \vdash P \mid Q \xrightarrow{\ell} P' \mid Q} \qquad \textsc{Com}\;\dfrac{A \vdash P \xrightarrow{\overline{x}v} P' \quad A \vdash Q \xrightarrow{xv} Q'}{A \vdash P \mid Q \xrightarrow{\tau} \textbf{new } \{v\} - A \textbf{ in } (P' \mid Q')}
$$

$$
\textsc{Open}\;\dfrac{A, x \vdash P \xrightarrow{\overline{y}x} P' \quad y \neq x}{A \vdash \textbf{new } x \textbf{ in } P \xrightarrow{\overline{y}x} P'} \qquad \textsc{Res}\;\dfrac{A, x \vdash P \xrightarrow{\ell} P' \quad x \notin \mathrm{fn}(\ell)}{A \vdash \textbf{new } x \textbf{ in } P \xrightarrow{\ell} \textbf{new } x \textbf{ in } P'}
$$

$$
\textsc{Struct Right}\;\dfrac{A \vdash P \xrightarrow{\ell} P' \quad P' \equiv P''}{A \vdash P \xrightarrow{\ell} P''}
$$

Structural congruence on the left of a transition is admissible.

Theorem 3 *If* $P' \equiv P$ *then* $A \vdash P' \xrightarrow{\ell} Q$ *iff* $A \vdash P \xrightarrow{\ell} Q$.

The reduction and transition semantics give exactly the same internal steps.

Theorem 4 *If* $\mathrm{fn}(P) \subseteq A$ *then* $P \longrightarrow Q$ *iff* $A \vdash P \xrightarrow{\tau} Q$.

Partial traces

$$\mathrm{ptr}_A(P) = \{\, \ell_1 \ldots \ell_n \mid A \vdash P \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_n} \,\}$$

Strong Bisimulation Take *bisimulation* $\dot{\sim}$ to be the largest family of relations indexed by finite sets of names such that each $\dot{\sim}_A$ is a relation over $\{\, P \mid \mathrm{fn}(P) \subseteq A \,\}$ and for all $P \mathrel{\dot{\sim}_A} Q$,

- if $A \vdash P \xrightarrow{\ell} P'$ then $\exists Q' \,.\, A \vdash Q \xrightarrow{\ell} Q' \wedge P' \mathrel{\dot{\sim}_{A \cup \mathrm{fn}(\ell)}} Q'$
- if $A \vdash Q \xrightarrow{\ell} Q'$ then $\exists P' \,.\, A \vdash P \xrightarrow{\ell} P' \wedge P' \mathrel{\dot{\sim}_{A \cup \mathrm{fn}(\ell)}} Q'$

Theorem 5 *Bisimulation $\dot{\sim}$ is an indexed congruence.*

Define $\sim$ by $P \sim_A Q$ iff for all substitutions σ with $\mathrm{dom}(\sigma) \cup \mathrm{ran}(\sigma) \subseteq A$ we have $\sigma P \mathrel{\dot{\sim}_A} \sigma Q$.

enough!

Semantics for Distributed Systems

2. Distribution

Asynchronous π – a good fit to distributed systems?

+ clear treatment of concurrency
+ asynchronous π communication not far above real comms
+ π-style naming is widely applicable

but doesn't address:

– point-to-point or multicast comms

– failure (machines and comms); timeouts, transactions

– security properties (secrecy, integrity, authenticity, non-repudiation, anonymity) and their implementation using crypto

– secure encapsulation; policy managment

– code, computation and device mobility

– performance

and

– proofs of concurrent algorithms are still difficult

– asynchronous π communication is far above real comms

New

- communication channels (with read/write operations)
- cryptographic keys (with decrypt/encrypt)
- reference cells (with deref/assign)
- boxes
- nonces
- type ids

Use a syntax with binding, and a structural congruence (or other rules) for moving the binders, instead of global name generation.

Pure names.

Grouping

π has *names* and *processes*.

Must add primitives for *grouping* process terms, into units of:

- failure (e.g. machines or runtime system instances);
- migration (e.g. mobile computations);
- trust (e.g. large administrative domains or small secure critical regions);
- synchronisation (i.e. regions within which an output and an input on the same channel name can interact).

Computation Mobility – DJoin

Take a tree of *locations*, each uniquely named. Generable.

Migration allows any location ℓ to move to become a child of any non-descendant.

Join patterns combine restriction, replicated input and linearity – $\textbf{def}\ x(w).P\ \textbf{in}\ Q$ is similar to $\textbf{new}\ x\ \textbf{in}\ Q \mid !xw.P$. An example reduction:

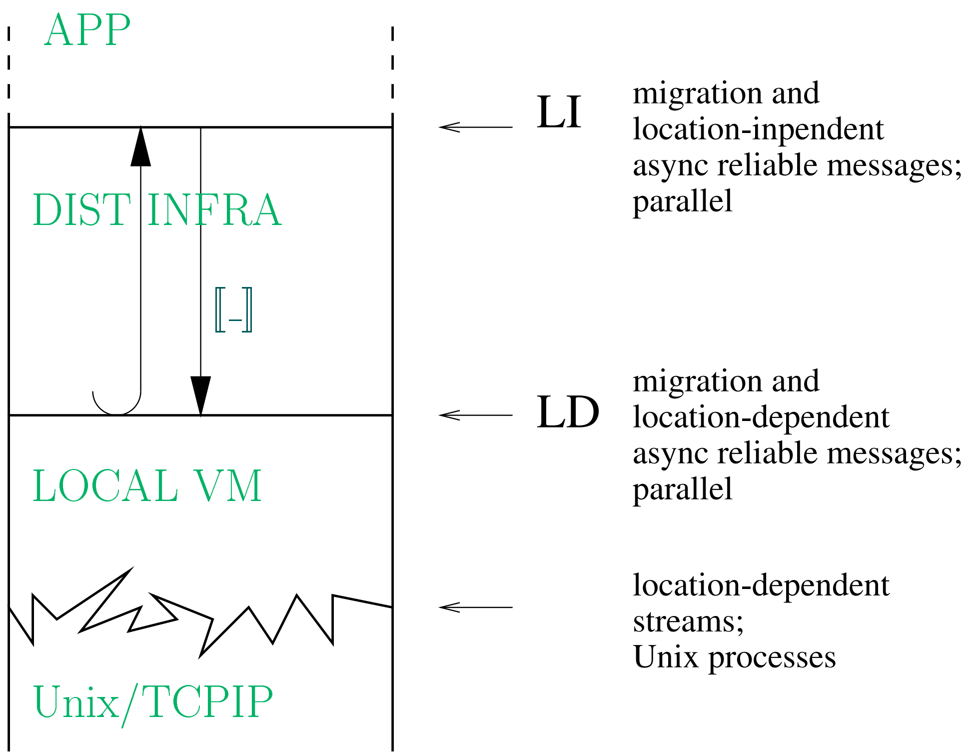
$$\begin{array}{l}\textbf{def}\ (x_1(w_1) \wedge x_2(w_2)).P\ \textbf{in}\ Q \mid \overline{x_1}3 \mid \overline{x_2}7 \\ \longrightarrow \\ \textbf{def}\ (x_1(w_1) \wedge x_2(w_2)).P\ \textbf{in}\ Q \mid \{3,7/w_1,w_2\}P\end{array}$$

Computation Mobility – Nomadic π

Focus on communication infrastructure for mobile computation.

Take a short tree of *sites* and *agents*, each uniquely named. Agents are located at sites, and are generable.

Migration allows any agent a to move to become a child of any site s.



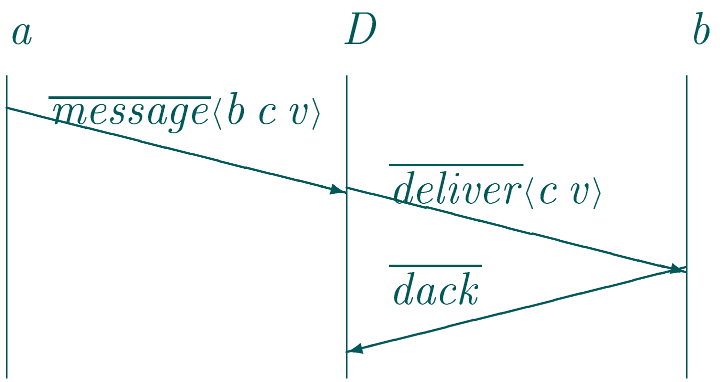
π-style communication, but synchronisation only within the same agent. Write $@_a P$ for P as part of agent a.

Record the current site of each agent in Γ.

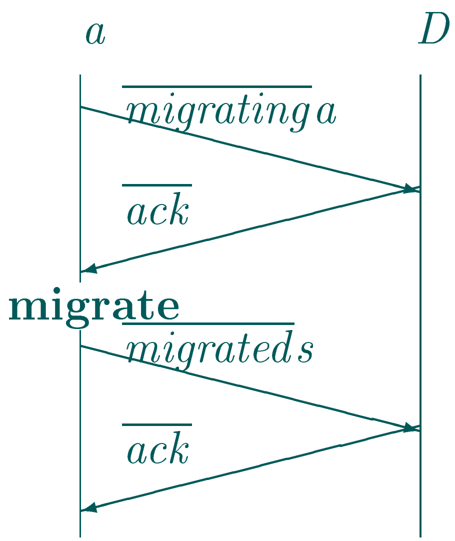
$$
\begin{array}{llll}
\Gamma, @_a \,\textbf{agent}\; b = P \;\textbf{in}\; Q & \longrightarrow & \Gamma, \textbf{new}\; b@\Gamma(a) \;\textbf{in}\; (@_b P | @_a Q) & \\[2ex]
\Gamma, @_a \,\textbf{migrate to}\; s.P & \longrightarrow & (\Gamma \oplus a \mapsto s), @_a P & \\[2ex]
\Gamma, @_a \overline{\langle b@?\rangle c} v & \longrightarrow & \Gamma, @_b \overline{c} v & \\[2ex]
\Gamma, @_a \langle b@s\rangle c!v & \longrightarrow & \Gamma, @_b \overline{c} v & \text{if } \Gamma(b) = s \\[2ex]
\Gamma, @_a \langle b@s\rangle c!v & \longrightarrow & \Gamma, 0 & \text{if } \Gamma(b) \neq s \\[2ex]
\Gamma, @_a (\overline{c} v | cp.P) & \longrightarrow & \Gamma, @_a \{v/p\}P & \\[2ex]
\Gamma, @_a \,\textbf{iflocal}\; \overline{\langle b\rangle c} v.P \;\textbf{else}\; Q & \longrightarrow & \Gamma, @_b \overline{c} v | @_a P & \text{if } \Gamma(a) = \Gamma(b) \\
 & \longrightarrow & \Gamma, @_a Q & \text{if } \Gamma(a) \neq \Gamma(b)
\end{array}
$$

Computation Mobility – Example Encoding – Central Daemon

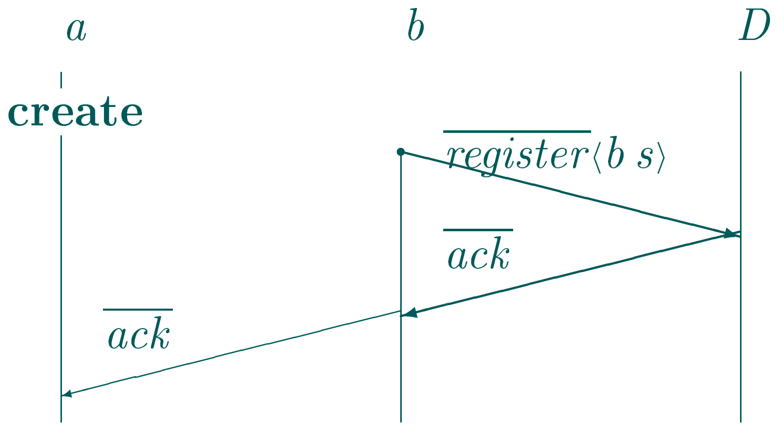
Location-independent output



Migration



Creation



Computation Mobility – Example Encoding – Central Daemon

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

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

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

but in an untrusted world?

Two scenarios:

1. running untrusted code in a trusted environment. Encapsulate, controlling interactions.
2. interacting with (partially) trusted code across an untrusted environment. Crypto protocols.

For both:

- if calculus reasoning is to be useful it must accurately model what an attacker could do

- slippery properties

Secure Encapsulation

$$P \ ::= \ \ldots$$

$a[P]$ ambient/seal/box named a containing P

more-or-less Draconian. Problem of escaping capabilities...

Box-π

$P \ ::= \ \overline{c}^{\star}v$	output v on channel c within this group
$\overline{c}^{\uparrow}v$	output v on channel c to parent
$\overline{c}^{\downarrow n}v$	output v on channel c to child n

$$\begin{array}{llcl}
\text{Up} & n[\overline{x}^{\uparrow}v \mid Q] & \longrightarrow & \overline{x}^{\bar{n}}v \mid n[Q] \\
\text{Down} & \overline{x}^{n}v \mid n[Q] & \longrightarrow & n[\overline{x}^{\bar{\uparrow}}v \mid Q] \\
\text{Comm} & \overline{x}^{\bar{\iota}}v \mid x^{\iota}p.P & \longrightarrow & \{^{v}/_{p}\}P
\end{array}$$

Pipeline Wrapper

$$\begin{aligned}
\mathcal{W}_2[\ _1,\ _2] \stackrel{def}{=} \textbf{new}\ a_1, a_2\ \textbf{in}\ (\ & a_1[\ _1] \mid a_2[\ _2] \\
& \mid !in_1{}^{\uparrow}y.\overline{in_1}^{a_1}y \\
& \mid !in_2{}^{\uparrow}y.\overline{in_2}^{a_2}y \\
& \mid !out_1{}^{a_1}y.\overline{out_1}^{\uparrow}y \\
& \mid !out_2{}^{a_2}y.\overline{out_2}^{\uparrow}y \\
& \mid !mid^{a_1}y.\overline{mid}^{a_2}y\ )
\end{aligned}$$

Now can state encapsulation properties (coloured semantics, tracking causal flow) and use a fancy type system to prove them.

Other primitives:

$\overline{c}^{\to n} v$	output v on channel c to sibling n
$\overline{c}^{\uparrow n} v$	output v on channel c to parent if a child of n
$\overline{c}^{path} v$	follow path p then output v on channel c

where paths are sequences of local movements:

$$
\begin{array}{lll}
path & ::= \star & \text{output within this group} \\
 & \quad \uparrow.path & \text{go to parent and then follow } path \\
 & \quad \downarrow n.path & \text{go to child } n \text{ and then follow } path
\end{array}
$$

Problem: reconcile need for boxing primitives (to limit interaction) and need for communication primitives (to get things done).

Ultimately, need *policy management*.

Crypto (Spi, SJoin, ...)

Extend values with encryption $\{v\}_k$, and add a decrypt form. Use $\textbf{new } k \textbf{ in } P$.

Calculus

$$
\begin{array}{rcl}
v & ::= & x \;\Big|\; \langle v, .., v \rangle \;\Big|\; \{v\}_v \\
p & ::= & x \;\Big|\; \langle p, .., p \rangle \\
P & ::= & 0 \;\Big|\; \overline{x}v \;\Big|\; xp.P \;\Big|\; P \,|\, P \;\Big|\; \textbf{new } x \textbf{ in } P \\
 & & \textbf{decrypt } v \textbf{ as } \{p\}_v \textbf{ in } P \textbf{ else } P
\end{array}
$$

Example Protocol

$$
\begin{array}{rcl}
A & = & \textbf{new } K_{ab} \textbf{ in } \overline{s}\{K_{ab}\}_{K_{as}} \,|\, \overline{b}\{m\}_{K_{ab}} \\
S & = & sv.\textbf{decrypt } v \textbf{ as } \{y\}_{K_{as}} \textbf{ in } \overline{b}\{y\}_{K_{bs}} \textbf{ else } 0 \\
B & = & bv.\textbf{decrypt } v \textbf{ as } \{K\}_{K_{bs}} \textbf{ in } bw.\textbf{decrypt } w \textbf{ as } \{n\}_K \textbf{ in } \overline{bprint}\,n \textbf{ else } 0 \textbf{ else } 0
\end{array}
$$

Consider $Sys = A|S|B|C$ for an attacker C without $\{K_{as}, K_{bs}, m, bprint\}$ free.

Failure

First attempt: message loss

$$\overline{c}v \longrightarrow 0$$

and crash fail

$$m[P] \longrightarrow 0$$

All very well, but need (at a low level) fairness and/or time, and (at a high level) transactions....?

Realistic Semantics for Low-Level Network Programming

UDP calculus...

Conclusion

Focussed on programming language side – design of primitives and proofs of executable descriptions of algorithms.

Some progress on each aspect, but not short of problems...