

The PI-CALCULUS: an introduction

Matthew Hennessy, U of Sussex

- Process calculi
- CCS with values
- Why CCS is not powerful enough
- The PI-CALCULUS
- Types for PI-CALCULUS
- Reasoning in presence of types
- Extensions mobile agents in a distributed world

Process calculi

Formal languages for

- describing concurrent processes
- reasoning about concurrent processes

Process calculi

Formal languages for

- describing concurrent processes
- reasoning about concurrent processes

Typical process calculus:

- Small set of operators for describing structure of processes
- Algebraic laws governing process behaviour
- proof techniques for establishing process correctness

Process calculi

Formal languages for

- describing concurrent processes
- reasoning about concurrent processes

Typical process calculus:

- Small set of operators for describing structure of processes
- Algebraic laws governing process behaviour
- proof techniques for establishing process correctness

Many schools of thought:

- CCS - Edinburgh
- CSP - Oxford
- ACP - Amsterdam

Describing concurrent processes, communicating *values* over *channels*

- $c!\langle e \rangle P$ - output value of expression e along channel c and execute P
- $c?(X) T$ - input from channel c and execute T with X instantiated with values input
- operators for *defining* and *combining* processes

Buffers

Example: one time buffer

$$\text{BUFF}_{single}(\text{in}, \text{out}) \Leftarrow \text{in?}(x) \text{out!}\langle x \rangle . \text{stop}$$

Buffers

Example: one time buffer

$$\text{BUFF}_{single}(\text{in}, \text{out}) \Leftarrow \text{in?}(x) \text{out!}\langle x \rangle . \text{stop}$$

Example: repeated buffer

$$\text{BUFF}(\text{in}, \text{out}) \Leftarrow * \text{in?}(x) \text{out!}\langle x \rangle . \text{stop}$$

Buffers

Example: one time buffer

$$\text{BUFF}_{single}(\text{in}, \text{out}) \Leftarrow \text{in?}(x) \text{out!}\langle x \rangle . \text{stop}$$

Example: repeated buffer

$$\text{BUFF}(\text{in}, \text{out}) \Leftarrow * \text{in?}(x) \text{out!}\langle x \rangle . \text{stop}$$

Connecting buffers:

$$\text{BUFF}_2 \Leftarrow (\text{new priv})(\text{BUFF}(\text{in}, \text{priv}) \mid \text{BUFF}(\text{priv}, \text{out}))$$

Syntax of CCS with values

Read from channel c : $c?(x) R$

Write to channel c : $c!\langle e \rangle R$ - expressions e are *integer, booleans, etc*

Private channels: $(\text{new } n) R$

Parallelism: $R_1 \mid R_2$

Iteration: $* R$

Choice: $R_1 + R_2$

Stop: stop

Value-testing: $\text{if } v_1 = v_2 \text{ then } R_1 \text{ else } R_2$

Formal semantics

Reduction semantics:

- Describes computations which processes may perform
- Proto-typical implementation

Behavioural semantics:

- Describes how processes behaves as part of larger systems
- Determines semantic equivalence between process descriptions

Formal semantics

Reduction semantics:

- Describes computations which processes may perform
- Proto-typical implementation

$P \longrightarrow Q$: one step of the computation of P leads to Q

Behavioural semantics:

- Describes how processes behaves as part of larger systems
- Determines semantic equivalence between process descriptions

$P \xrightarrow{\mu} Q$: P may interact with environment using data in μ

$S_1 \approx S_2$: no observer can distinguish between S_1 and S_2

Behavioural semantics

You can build a two-place buffer using two one-place Buffers

$$\text{SPEC} \Leftarrow \text{in?}(x) S_1(x)$$

$$S_1(x) \Leftarrow \text{in?}(y) S_2(x, y) + \text{out!}\langle x \rangle \text{SPEC}$$

$$S_2(x, y) \Leftarrow \text{out!}\langle x \rangle S_1(y)$$

Behavioural semantics

You can build a two-place buffer using two one-place Buffers

$$\text{SPEC} \Leftarrow \text{in?}(x) S_1(x)$$

$$S_1(x) \Leftarrow \text{in?}(y) S_2(x, y) + \text{out!}\langle x \rangle \text{SPEC}$$

$$S_2(x, y) \Leftarrow \text{out!}\langle x \rangle S_1(y)$$

Statement of correctness:

$$\text{BUFF}_2 \approx \text{SPEC}$$

Example system

PRIMES accepts requests at req and produces answers at reply:

PRIMES \Leftarrow * req?(x) let $a = isprime(x)$ in reply! $\langle a \rangle$

Example system

PRIMES accepts requests at req and produces answers at reply:

$$\text{PRIMES} \Leftarrow * \text{req?}(x) \text{ let } a = \text{isprime}(x) \text{ in reply!}\langle a \rangle$$

Clients: send requests and prints the results:

$$\text{CLIENT}_1 \Leftarrow \text{req!}\langle 27 \rangle \text{ reply?}(x) \text{ print}_1!\langle x \rangle$$
$$\text{CLIENT}_2 \Leftarrow \text{req!}\langle 29 \rangle \text{ reply?}(x) \text{ print}_2!\langle x \rangle$$

Example system

PRIMES accepts requests at req and produces answers at reply:

$$\text{PRIMES} \Leftarrow * \text{req?}(x) \text{ let } a = \text{isprime}(x) \text{ in reply!}\langle a \rangle$$

Clients: send requests and prints the results:

$$\text{CLIENT}_1 \Leftarrow \text{req!}\langle 27 \rangle \text{ reply?}(x) \text{ print}_1!\langle x \rangle$$
$$\text{CLIENT}_2 \Leftarrow \text{req!}\langle 29 \rangle \text{ reply?}(x) \text{ print}_2!\langle x \rangle$$

* P acts like $P \mid * P$

The server works

PRIMES | CLIENT₁ \longrightarrow^* PRIMES | print₁**\langle**false**\rangle**

The server works

PRIMES | CLIENT₁ \longrightarrow^* PRIMES | print₁ \langle false \rangle

With two clients:

PRIMES | CLIENT₁ | CLIENT₂ \longrightarrow^*
PRIMES | print₁ \langle false \rangle | print₂ \langle true \rangle

The server does not work

PRIMES | CLIENT₁ | CLIENT₂ \longrightarrow^*

PRIMES | print₁**<true>** | print₂**<>false>**

The server does not work

PRIMES | CLIENT₁ | CLIENT₂ \longrightarrow^*

PRIMES | print₁**<true>** | print₂**<false>**

- Clients need their own reply channel
- Reply channels need to be dynamically created
- Dynamically related channels need to be properly managed

Syntax of PI-CALCULUS: API

Read from channel u : $u?(x) R$

Write to channel u : $u!\langle v \rangle$ - values v may be channels

Channel creation: $(\text{new } n) R$

Parallelism: $R_1 \mid R_2$

Iteration: $* R$

Stop: stop

Value-testing: $\text{if } v_1 = v_2 \text{ then } R_1 \text{ else } R_2$

Example: a server

A prime-checking server PRIMES:

- receives a request at port req
 - an integer,
 - a return *address*
- analyzes the integer
- returns true or false on address

$$\text{PRIMES} \leftarrow *req?(x, y) \text{ let } a = \text{isprime}(x) \text{ in } y!\langle a \rangle$$

Example: Clients

- generates a *return address* reply
- sends a value and reply to server
- awaits answer on return address
- prints answer when it comes

$$\text{CLIENT}_1 \leftarrow (\text{new reply})(\text{req!}\langle 27, \text{reply} \rangle \\ | \text{reply?}(x) \text{print}_1!\langle x \rangle)$$
$$\text{CLIENT}_2 \leftarrow (\text{new reply})(\text{req!}\langle 29, \text{reply} \rangle \\ | \text{reply?}(x) \text{print}_2!\langle x \rangle)$$

The server works

PRIMES | CLIENT₁ | CLIENT₂ \longrightarrow^*

PRIMES | print₁**<false>** | print₂**<>true>**

The server works

PRIMES | CLIENT₁ | CLIENT₂ \longrightarrow^*

PRIMES | print₁**<false>** | print₂**<>true>**

If PRIMES | CLIENT₁ | CLIENT₂ \longrightarrow^* P

and P is stable

then

P is more or less PRIMES | print₁**<false>** | print₂**<>true>**

The server works

PRIMES | CLIENT₁ | CLIENT₂ \longrightarrow^*

PRIMES | print₁**<false>** | print₂**<>true>**

If PRIMES | CLIENT₁ | CLIENT₂ \longrightarrow^* P

and P is stable

then

P is *more or less* PRIMES | print₁**<false>** | print₂**<>true>**

Depends on intricacies of reduction semantics \longrightarrow

Specifying reduction semantics: $P \longrightarrow Q$

Using a set of rewrite rules:

Axioms: eg

$$c?(X) R \mid c!(V) \longrightarrow R\{X/V\}$$

Contextual rules:

$$\begin{array}{c} \text{(R-PAR)} \\ \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \\ \\ \text{(R-NEW)} \\ \frac{P \longrightarrow P'}{(\text{new } n) P \longrightarrow (\text{new } n) P'} \end{array} \qquad \frac{P \longrightarrow P'}{Q \mid P \longrightarrow Q \mid P'}$$

Specifying reduction semantics: $P \longrightarrow Q$

Up to structural equivalence:

$$\frac{\text{(R-STRUCT)} \quad P \equiv P', P \longrightarrow Q, Q \equiv Q'}{P' \longrightarrow Q'}$$

Specifying reduction semantics: $P \longrightarrow Q$

Up to structural equivalence:
$$\frac{\text{(R-STRUCT)} \quad P \equiv P', P \longrightarrow Q, Q \equiv Q'}{P' \longrightarrow Q'}$$

Structural equivalence axioms:

(S-COM)

$$P \mid Q = Q \mid P$$

(S-ASSOC)

$$(P \mid Q) \mid R = P \mid (Q \mid R)$$

(S-STOP)

$$P \mid \text{stop} = P$$

$$(\text{new } n) \text{ stop} = \text{stop}$$

(S-FLIP)

$$(\text{new } n) (\text{new } m) P = (\text{new } m) (\text{new } n) P$$

Specifying reduction semantics: $P \longrightarrow Q$

Up to structural equivalence:
$$\frac{\text{(R-STRUCT)} \quad P \equiv P', P \longrightarrow Q, Q \equiv Q'}{P' \longrightarrow Q'}$$

Structural equivalence axioms:

(S-COM)

$$P \mid Q = Q \mid P$$

(S-ASSOC)

$$(P \mid Q) \mid R = P \mid (Q \mid R)$$

(S-STOP)

$$P \mid \text{stop} = P$$

$$(\text{new } n) \text{ stop} = \text{stop}$$

(S-FLIP)

$$(\text{new } n) (\text{new } m) P = (\text{new } m) (\text{new } n) P$$

And

$$\text{(S-EXTR)} \quad (\text{new } n)(P \mid Q) = P \mid (\text{new } n) Q \quad \text{if } n \notin \text{fn}(P)$$

Using reduction semantics

(R-ITER) $\text{PRIMES} \longrightarrow \text{req?}(x, y) \dots \mid \text{PRIMES}$

(R-CONXT) $(\text{new reply})(\dots \mid \text{PRIMES}) \longrightarrow$
 $(\text{new reply})(\dots \mid \text{req?}(x, y) \dots \mid \text{PRIMES})$

Using reduction semantics

(R-ITER) $\text{PRIMES} \longrightarrow \text{req?}(x, y) \dots \mid \text{PRIMES}$

(R-CONXT) $(\text{new reply})(\dots \mid \text{PRIMES}) \longrightarrow$
 $(\text{new reply})(\dots \mid \text{req?}(x, y) \dots \mid \text{PRIMES})$

(S-EXTR) $(\text{new reply})(\dots \mid \text{PRIMES})$
 \equiv
 $(\text{new reply})(\dots) \mid \text{PRIMES}$

Using reduction semantics

(R-ITER) $\text{PRIMES} \longrightarrow \text{req?}(x, y) \dots \mid \text{PRIMES}$

(R-CONXT) $(\text{new reply})(\dots \mid \text{PRIMES}) \longrightarrow$
 $(\text{new reply})(\dots \mid \text{req?}(x, y) \dots \mid \text{PRIMES})$

(S-EXTR) $(\text{new reply})(\dots \mid \text{PRIMES})$
 \equiv
 $(\text{new reply})(\dots) \mid \text{PRIMES}$

Therefore

$\text{CLIENT}_1 \mid \text{PRIMES} \longrightarrow$
 $(\text{new reply}) (\text{CLIENT}_1 \mid \text{req?}(x, y) \dots \mid \text{PRIMES})$

Using reduction semantics

$\text{CLIENT}_1 \mid \text{PRIMES} \longrightarrow^* (\text{new reply}) (\text{print}_1 \langle \mathbf{false} \rangle \mid \text{PRIMES})$

Using reduction semantics

$\text{CLIENT}_1 \mid \text{PRIMES} \longrightarrow^* (\text{new reply}) (\text{print}_1 \langle \mathbf{false} \rangle \mid \text{PRIMES})$

Derived law:

$(\text{new } n) P \equiv P \quad \text{if } n \text{ not free in } P$

Using reduction semantics

$$\text{CLIENT}_1 \mid \text{PRIMES} \longrightarrow^* (\text{new reply}) (\text{print}_1 \langle \mathbf{false} \rangle \mid \text{PRIMES})$$

Derived law:

$$(\text{new } n) P \equiv P \quad \text{if } n \text{ not free in } P$$

Therefore

$$\text{CLIENT}_1 \mid \text{PRIMES} \longrightarrow^* \text{print}_1 \langle \mathbf{false} \rangle \mid \text{PRIMES}$$

Does the server work?

- Will CLIENT₁ print **false**? under all circumstances?

Does the server work?

- Will CLIENT_1 print **false**? under all circumstances?
- Is SYS behaviourally equivalent to SPEC where

$$\begin{array}{l} \text{SYS} \Leftarrow \text{PRIMES} \quad | \quad \text{CLIENT}_1 \\ \text{SPEC} \Leftarrow \text{PRIMES} \quad | \quad \text{print}_1! \langle \text{false} \rangle \end{array}$$

Does the server work?

- Will CLIENT_1 print **false**? under all circumstances?
- Is SYS **behaviourally equivalent** to SPEC where

$$\begin{array}{l} \text{SYS} \leftarrow \text{PRIMES} \quad | \quad \text{CLIENT}_1 \\ \text{SPEC} \leftarrow \text{PRIMES} \quad | \quad \text{print}_1! \langle \text{false} \rangle \end{array}$$

- $S1$ **behaviourally equivalent** to $S2$ means no observer/user can tell them apart, in any context
 - formalise using
 - *bisimulation equivalence*
 - *contextual equivalence*

A bad server

A rogue server `BADPRS`:

- receives a request at port `req`
 - an integer,
 - a *return address*
- always returns `true` on return address

$$\text{BADPRS} \Leftarrow *req?(x, y) y!\langle \text{true} \rangle$$

Does the server work?

Does the server work?

SYS **not** behaviourally equivalent to SPEC

because

SYS | BADPRS **not** behaviourally equivalent to SPEC | BADPRS

Does the server work?

SYS **not** behaviourally equivalent to SPEC

because

SYS | BADPRS **not** behaviourally equivalent to SPEC | BADPRS

recall:

SYS \Leftarrow PRIMES | CLIENT₁

SPEC \Leftarrow PRIMES | print₁!**⟨false⟩**

BADPRS \Leftarrow *req?(x, y) y!**⟨true⟩**

Does the server work?

SYS **not** behaviourally equivalent to SPEC

because

SYS | BADPRS **not** behaviourally equivalent to SPEC | BADPRS

recall:

SYS \Leftarrow PRIMES | CLIENT₁

SPEC \Leftarrow PRIMES | print₁!**⟨false⟩**

BADPRS \Leftarrow *req?(x, y) y!**⟨true⟩**

SPEC | BADPRS **always prints false**

SYS | BADPRS **may print true or false**

The server works

- under certain assumptions about the *working context*

The server works

- under certain assumptions about the *working context*
- assuming there is no other server in the environment

The server works

- under certain assumptions about the *working context*
- assuming there is no other server in the environment

What is a server?

The server works

- under certain assumptions about the *working context*
- assuming there is no other server in the environment

What is a server?

A process which has the *capability* to read from request channel req

The server works

- under certain assumptions about the *working context*
- assuming there is no other server in the environment

What is a server?

A process which has the *capability* to read from request channel req

Approach: use **types** to manage capabilities

Types for the PI-CALCULUS

Types constrain the valid use of channels/resources

Types represent sets of capabilities

- **int, bool**: for basic values
- $r\langle T \rangle$: channels which can only be **read** from
- $w\langle T \rangle$: channels which can only be **written** to
- $rw\langle T \rangle$: channels which can be both **read** from and **written** to

subtyping used

Types for the PI-CALCULUS

Types constrain the valid use of channels/resources

Types represent sets of capabilities

- **int, bool**: for basic values
- $r\langle T \rangle$: channels which can only be **read** from
- $w\langle T \rangle$: channels which can only be **written** to
- $rw\langle T \rangle$: channels which can be both **read** from and **written** to

subtyping used

- In $\alpha\langle T \rangle$ the type T represents the capabilities of the values received

Typing the server

PRIMES \Leftarrow *req?(x, y) let $a = isprime(x)$ in $y!\langle a \rangle$

- receives at port req
 - an integer,
 - a return *address*
- writes boolean answer on address

Typing the server

$\text{PRIMES} \Leftarrow *req?(x, y) \text{ let } a = isprime(x) \text{ in } y!\langle a \rangle$

- receives at port req
 - an integer,
 - a return *address*
- writes boolean answer on address

$\Gamma \vdash \text{PRIMES} \quad \text{if } \Gamma(\text{req}) = r\langle \mathbf{int}, w\langle \mathbf{bool} \rangle \rangle$

Typing the server

$\text{PRIMES} \Leftarrow *req?(x, y) \text{ let } a = isprime(x) \text{ in } y!\langle a \rangle$

- receives at port req
 - an integer,
 - a return *address*
- writes boolean answer on address

$\Gamma \vdash \text{PRIMES} \quad \text{if } \Gamma(\text{req}) = r\langle \mathbf{int}, w\langle \mathbf{bool} \rangle \rangle$

Γ – a mapping from resource names to types

Typing clients

$$\text{CLIENT}_1 \leftarrow (\text{new reply} : \mathbf{R})(\text{req!}\langle 27, \text{reply} \rangle \\ | \text{reply?}(x) \text{print}_1!\langle x \rangle)$$

- generates a *return address* `reply` with required capabilities \mathbf{R}
- sends a value and `reply` to server
- prints boolean answer when it comes

Typing clients

$$\text{CLIENT}_1 \leftarrow (\text{new reply} : \mathbf{R})(\text{req}!\langle 27, \text{reply} \rangle \\ | \text{reply}?(x) \text{print}_1!\langle x \rangle)$$

- generates a *return address* `reply` with required capabilities \mathbf{R}
- sends a value and reply to server
- prints boolean answer when it comes

$$\Gamma \vdash \text{CLIENT}_1 \quad \text{if } \mathbf{R} = \text{rw}\langle \mathbf{bool} \rangle \\ \Gamma(\text{req}) = \text{w}\langle \mathbf{int}, \text{w}\langle \mathbf{bool} \rangle \rangle \\ \Gamma(\text{print}_1) = \text{w}\langle \mathbf{bool} \rangle$$

Typing the Π -CALCULUS - issues

- Typechecking processes
 - handling accumulation of capabilities
 - recursive processes need recursive types

- Type inference
 - processes may not have *most general* types

Contextual behaviour

$$\mathcal{I} \models P \approx Q$$

P and Q are behaviourally equivalent in all contexts containing knowledge \mathcal{I} of the capabilities/potential of P and Q

Contextual behaviour

$$\mathcal{I} \models P \approx Q$$

P and Q are behaviourally equivalent in all contexts containing knowledge \mathcal{I} of the capabilities/potential of P and Q

- $\mathcal{I} \models \text{req?}(x) T \approx \text{stop}$
if \mathcal{I} does not contain the capability to write to req
- $\mathcal{I} \models (\text{new } s)(\text{dist!}\langle s \rangle \mid s?(x) T) \approx (\text{new } s) \text{dist!}\langle s \rangle$
if in \mathcal{I} dist can not read *write capabilities*

The server works

$$\mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_1 \approx \text{PRIMES} \mid \text{print}_1! \langle \text{false} \rangle$$

provided

\mathcal{I} does not allow reading on req

The server works

$$\mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_1 \approx \text{PRIMES} \mid \text{print}_1! \langle \text{false} \rangle$$

provided

\mathcal{I} does not allow reading on req

recall:

$\text{PRIMES} \Leftarrow * \text{req?}(x, y) \text{ let } a = \text{isprime}(x) \text{ in } y! \langle a \rangle$

$\text{CLIENT}_1 \Leftarrow (\text{new reply}) \text{ req!} \langle 27, \text{reply} \rangle$
 $\quad \quad \quad \mid \text{reply?}(x) \text{ print}_1! \langle x \rangle$

Formalising $\mathcal{I} \models P \approx Q$

Using bisimulations over *actions-in-context*

$$\mathcal{I} \triangleright P \xrightarrow{\mu} \mathcal{I}' \triangleright P'$$

where

- \mathcal{I} is current knowledge of context
- P is current system
- μ is description of possible interaction with context
- \mathcal{I}' is new knowledge of context after the interaction
- P' is resulting system after the interaction

Actions-in-context: examples

- standard action:

$$\text{req?}(x) T \xrightarrow{(n)\text{req?}n} T \{n/x\}$$

Actions-in-context: examples

- standard action:

$$\text{req?}(x) T \xrightarrow{(n)\text{req?}n} T \{n/x\}$$

- action-in-context:

$$\mathcal{I} \triangleright \text{req?}(x) T \xrightarrow{(n:E)\text{req?}n} \mathcal{I}, n : E \triangleright T \{n/x\}$$

provided allows writing n on req at type E

Actions-in-context: examples

- standard action:

$$\text{req?}(x) T \xrightarrow{(n)\text{req?}n} T \{n/x\}$$

- action-in-context:

$$\mathcal{I} \triangleright \text{req?}(x) T \xrightarrow{(n:E)\text{req?}n} \mathcal{I}, n : E \triangleright T \{n/x\}$$

provided allows writing n on req at type E

- standard action:

$$(\text{new } n : E) \text{req!}\langle n \rangle \xrightarrow{(n)\text{req!}n} \mathbf{0}$$

Actions-in-context: examples

- standard action:

$$\text{req?}(x) T \xrightarrow{(n)\text{req?}n} T \{n/x\}$$

- action-in-context:

$$\mathcal{I} \triangleright \text{req?}(x) T \xrightarrow{(n:E)\text{req?}n} \mathcal{I}, n : E \triangleright T \{n/x\}$$

provided \mathcal{I} allows writing n on req at type E

- standard action:

$$(\text{new } n : E) \text{req!}\langle n \rangle \xrightarrow{(n)\text{req!}n} \mathbf{0}$$

- action-in-context:

$$\mathcal{I} \triangleright (\text{new } n : E) \text{req!}\langle n \rangle \xrightarrow{(n)\text{req!}n} \mathcal{I}, n : R_{\text{req}} \triangleright \mathbf{0}$$

provided \mathcal{I} allows reading on req

Contextual reasoning

Contextual reasoning principle:

- $\mathcal{I} \models P \approx Q$
- R conforms to \mathcal{I}

implies

$$\mathcal{I} \models P \mid R \approx Q \mid R$$

Contextual reasoning

Contextual reasoning principle:

- $\mathcal{I} \models P \approx Q$
- R conforms to \mathcal{I}

implies

$$\mathcal{I} \models P \mid R \approx Q \mid R$$

eg if \mathcal{I} is a type environment then R conforms to \mathcal{I} means $\mathcal{I} \vdash R$

Contextual reasoning - an example

$$\mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_1 \mid \text{CLIENT}_2$$
$$\approx$$
$$\text{PRIMES} \mid \text{print}_1!\langle \mathbf{false} \rangle \mid \text{print}_2!\langle \mathbf{true} \rangle$$

provided \mathcal{I} does not allow reading on req

Contextual reasoning - an example

$$\begin{aligned} \mathcal{I} &\models \text{PRIMES} \mid \text{CLIENT}_1 \mid \text{CLIENT}_2 \\ &\approx \\ &\text{PRIMES} \mid \text{print}_1!\langle\mathbf{false}\rangle \mid \text{print}_2!\langle\mathbf{true}\rangle \end{aligned}$$

provided \mathcal{I} does not allow reading on req

$$\begin{aligned} \text{PRIMES} &\Leftarrow * \text{req?}(x, y) \text{ let } a = \text{isprime}(x) \text{ in } y!\langle a \rangle \\ \text{CLIENT}_1 &\Leftarrow (\text{new reply}) \text{ req!}\langle 27, r \rangle \\ &\quad \mid \text{reply?}(x) \text{ print}_1!\langle x \rangle \\ \text{CLIENT}_2 &\Leftarrow (\text{new reply}) \text{ req!}\langle 19, r \rangle \\ &\quad \mid \text{reply?}(x) \text{ print}_2!\langle x \rangle \end{aligned}$$

Contextual reasoning - an example

- $\mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_1 \approx \text{PRIMES} \mid \text{print}_1! \langle \text{false} \rangle$
- a direct proof

Contextual reasoning - an example

- $\mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_1 \approx \text{PRIMES} \mid \text{print}_1! \langle \text{false} \rangle$
- a direct proof
- CLIENT_2 conforms to \mathcal{I}
So $\mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_1 \mid \text{CLIENT}_2 \approx$
 $\text{PRIMES} \mid \text{print}_1! \langle \text{false} \rangle \mid \text{CLIENT}_2$

Contextual reasoning - an example

- $\mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_1 \approx \text{PRIMES} \mid \text{print}_1!\langle\text{false}\rangle$
- a direct proof
- CLIENT_2 conforms to \mathcal{I}
So $\mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_1 \mid \text{CLIENT}_2 \approx$
 $\text{PRIMES} \mid \text{print}_1!\langle\text{false}\rangle \mid \text{CLIENT}_2$
- $\mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_2 \approx \text{PRIMES} \mid \text{print}_2!\langle\text{true}\rangle$
- a similar direct proof
- $\text{print}_1!\langle\text{false}\rangle$ conforms to \mathcal{I}
So $\mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_2 \mid \text{print}_1!\langle\text{false}\rangle \approx$
 $\text{PRIMES} \mid \text{print}_2!\langle\text{true}\rangle \mid \text{print}_1!\langle\text{false}\rangle$

Contextual reasoning - an example

- $\mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_1 \approx \text{PRIMES} \mid \text{print}_1!\langle\text{false}\rangle$
- a direct proof
- CLIENT_2 conforms to \mathcal{I}
So $\mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_1 \mid \text{CLIENT}_2 \approx$
 $\text{PRIMES} \mid \text{print}_1!\langle\text{false}\rangle \mid \text{CLIENT}_2$
- $\mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_2 \approx \text{PRIMES} \mid \text{print}_2!\langle\text{true}\rangle$
- a similar direct proof
- $\text{print}_1!\langle\text{false}\rangle$ conforms to \mathcal{I}
So $\mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_2 \mid \text{print}_1!\langle\text{false}\rangle \approx$
 $\text{PRIMES} \mid \text{print}_2!\langle\text{true}\rangle \mid \text{print}_1!\langle\text{false}\rangle$
- Therefore
 $\mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_1 \mid \text{CLIENT}_2 \approx$
 $\text{PRIMES} \mid \text{print}_1!\langle\text{false}\rangle \mid \text{print}_2!\langle\text{true}\rangle$

How general is all this ?

$$\mathcal{I} \models M \approx N$$

- M, N mobile agents in a distributed world:
 \mathcal{I} contains
 - known agents
 - agents rights
 - ...
- M, N mobile agents in a distributed world:
 \mathcal{I} contains
 - known sites
 - state of network - *failed sites, broken links*
 - migration rights
 - ...

DPI- a typed PI-CALCULUS with explicit locations

Computational model underlying DPI

locations/sites: where computations occur. Locations have a flat structure but may be generated dynamically.

In DPI all communication is local.

mobile agents: perform computations; move from site to site.

Described by augmented PI-CALCULUS terms.

resources: (communication channels) available at specific locations.

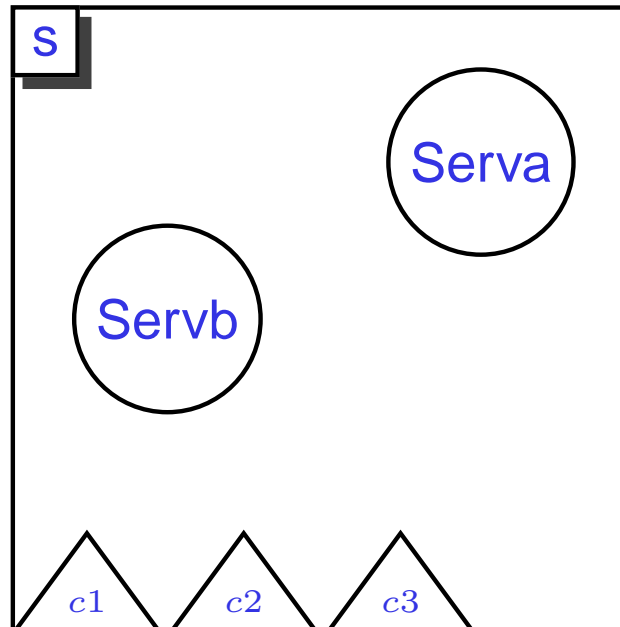
In DPI names of resources only have local significance.

types:

guarantee controlled access to resources.

Agents should only use resources in accordance with capabilities/permissions they have acquired.

DPI processes



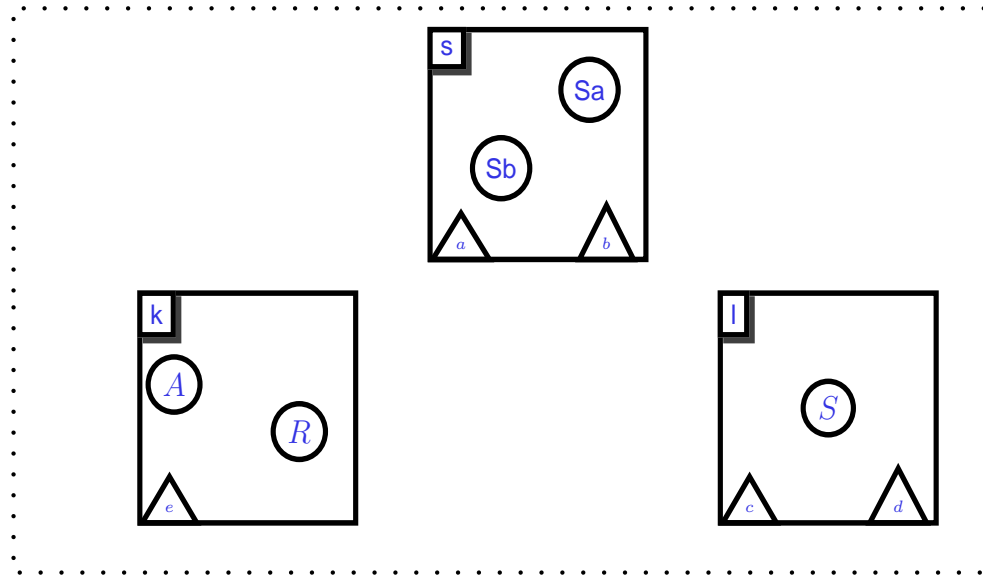
s - site name

c1, *c2*, *c3* -resources available at **s**

Serva, Servb - anonymous agents

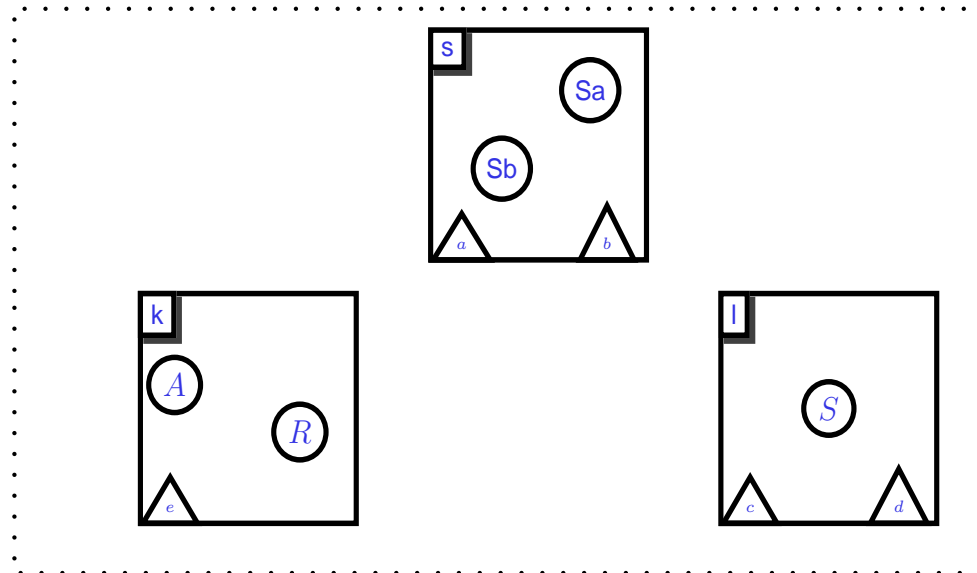
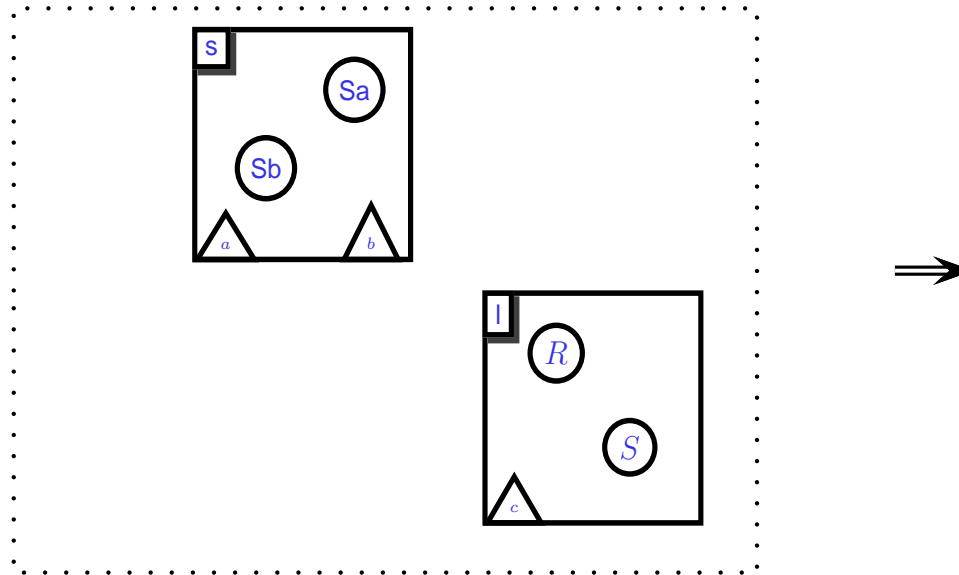
Agents may move autonomously from site to site

Distributed Systems

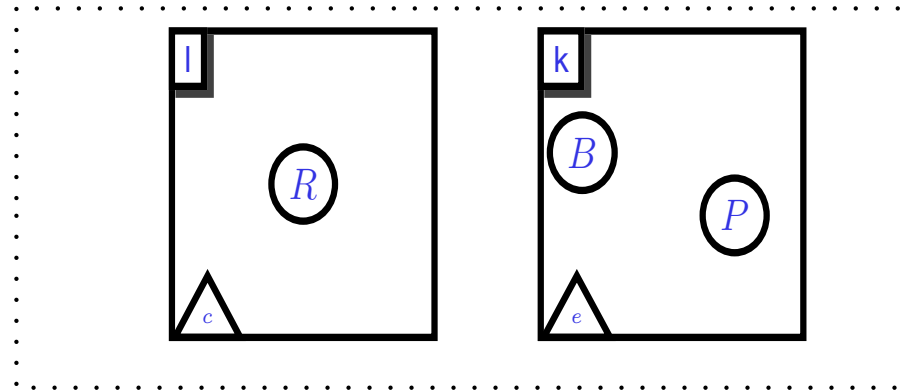
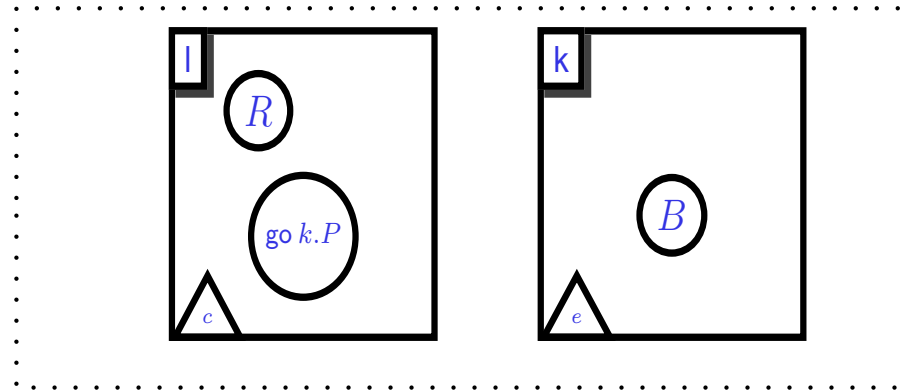


- A collection of independent distributed *sites* offering **services/resources** to migrating **agents**.
- **Resources:** modelled by picalculus channels
- **Agents:** modelled by (augmented) PI-CALCULUS processes

System Evolution



Migration



DPI— a typed PI-CALCULUS with explicit locations

A **System** is a collection of autonomous **agents** each running at explicit locations

Syntax of Systems $M - N$:

- $l[[P]]$ - the thread P running at site l
- $M \mid N$ - agents running in parallel
- $(\text{new } e : \mathbf{E}) M$ - sharing information
- $\mathbf{0}$ - the empty system

Example Systems:

$$l[[P]] \mid (\text{new } a @ k)(l[[Q]] \mid k[[R]])$$

$$l[[d?(x @ z) T]] \mid k[(\text{new } a) \text{ go } l.d!\langle a @ k \rangle]$$

Agents in DPI

- $u?(X : T) R$ - *local* input on channel u
- $u!\langle V \rangle$ - *local* output on channel u
- **go** $u.R$ - code movement to site u
- **if** $v_1 = v_2$ **then** R **else** U - testing of names
- **name creation**:
 - $(\text{newc } c : A) R$ - local channels/resources
 - $(\text{newloc } k : K) R$ - site names
- $R \mid U$ - concurrent code
- **rec** $x. R$ - iteration
- **stop** - finished

Example: a distributed compute server

$$s \llbracket \text{DPRIMES} \rrbracket \quad | \quad h \llbracket \text{CLIENT}(v) \rrbracket$$

where

$$\text{DPRIMES} \Leftarrow \text{rec } z. \text{in}_p?(x, y@w) \text{ go } w.y! \langle \text{isprime}(x) \rangle \quad | \quad z$$
$$\text{CLIENT}(v) \Leftarrow (\text{newc } r : \mathbf{R}) (\text{go } s.\text{in}_p! \langle v, r@h \rangle \quad | \quad r?(x) \text{ print!} \langle x \rangle)$$

$r@h$ is an *address* to which the result should be sent

Example: distributed forwarder

- maintains a forwarding service between two addresses, $b@k_1, c@k_2$
- initially at site h

$$h \llbracket \text{for}(b@k_1, c@k_2) \rrbracket$$

where

$$\text{for}(b@k_1, c@k_2) \Leftarrow \text{go } k_1. * b?(x, y) \text{go } k_2.$$
$$(\text{new ack})(c!\langle x, \text{ack} \rangle \mid$$
$$\text{ack?go } k_1.y!)$$

Routed Forwarding

$\text{Fwd}(\text{in}@h, \text{s}@d)$ sets up a communication link between in , local to h and s , local to d .

$$h[[* \text{in?}(x) \text{ go } d.\text{s}!\langle x \rangle]]$$

Routed Forwarding

$\text{Fwd}(\text{in}@h, \text{s}@d)$ sets up a communication link between in , local to h and s , local to d .

$$h[[* \text{in}?(x) \text{go } d.\text{s}!\langle x \rangle]]$$

Assuming no direct connection from h to d but instead a function $\text{Route}(d)$ which returns the neighbour nearest to d :

Routed Forwarding

$\text{Fwd}(\text{in}@h, \text{s}@d)$ sets up a communication link between in , local to h and s , local to d .

$$h \llbracket * \text{in?}(x) \text{ go } d.\text{s}!\langle x \rangle \rrbracket$$

Assuming no direct connection from h to d but instead a function $\text{Route}(d)$ which returns the neighbour nearest to d :

$$\begin{aligned} \text{Fwd}(\text{in}@h, \text{s}@d) &\Leftarrow \text{if } h = d \text{ then } * \text{in?}(x) \text{ s}!\langle x \rangle \\ &\text{else} \\ &\quad \text{let } n \leftarrow \text{Route}(d) \\ &\quad \text{in } \text{go } n. (\text{new } c) \text{Fwd}(c@n, \text{s}@d) \\ &\quad \quad | * \text{in?}(x) \text{ go } n.c!\langle x \rangle \end{aligned}$$

DPI - issues

- Formal semantics
- Types
- Reasoning techniques
- Applications

DPI - issues

- Formal semantics
 - reduction semantics
 - behavioural semantics
- Types
 - capability types
 - access control to sites
 - agent and site protection
- Reasoning techniques
 - bisimulations in the presence of types
 - contextual reasoning
- Applications
 - only trivial examples