MPRI Concurrency (course number 2-3) 2004-2005: π-calculus

25 November 2004

http://pauillac.inria.fr/~leifer/teaching/mpri-concurrency-2004/

James J. Leifer
INRIA Rocquencourt

James.Leifer@inria.fr

Today's plan

- exercises from last week
- data structures
- coding definitions in terms of replication
- bisimulation theorems

Adding sum

$P ::=$	M	sum
	$P \mid P$	parallel (par)
	$\boldsymbol{\nu} x.P$	restriction (new) (x binds in P)
	$!P$	replication (bang)
$M ::=$	$\overline{x}y.P$	output
	$x(y).P$	input (y binds in P)
	$M + M$	sum
	$\mathbf{0}$	

Changes:

- structural congruence: $+$ is associative and commutative with identity $\mathbf{0}$.
- reduction: $(\overline{x}y.P + M) \mid (x(u).Q + N) \longrightarrow P \mid \{u/y\}Q$.
- labelled transition: $M + \overline{x}y.P + N \xrightarrow{\overline{x}y} P$
  $M + x(y).P + N \xrightarrow{xz} \{y/z\}P$

Process abstractions

We don't need CCS-style "definitions" for infinite behaviour since we have replication, $!P$, as shown later. Nonetheless, they are convenient. In π-calculus, we call them process abstractions:

$$F = (u_1, ..., u_k).P$$

Instantiation takes an abstraction and a vector of names and gives back a process:

$$F\langle x_1, ..., x_k\rangle = \{x_1/u_1, ..., x_k/u_k\}P$$

Booleans

In Ocaml,

```
type bool = True | False;;
let cases b t f = match b with True -> t | False -> f;;
let not b = cases b False True;;
```

In π-calculus,

$$\begin{aligned} \mathit{True} &= (l).l(t,f).\bar{t} \\ \mathit{False} &= (l).l(t,f).\overline{f} \\ \mathit{cases}(P,Q) &= (l).\boldsymbol{\nu} t.\boldsymbol{\nu} f.\bar{l}\langle t,f\rangle.(t.P + f.Q) \\ \mathit{not} &= (l,k).\mathit{cases}(\mathit{False}\langle k\rangle,\ \mathit{True}\langle k\rangle)\langle l\rangle \end{aligned}$$

Example: show that

$$\boldsymbol{\nu} l.(\mathit{True}\langle l\rangle \mid \mathit{not}\langle l,k\rangle) \longrightarrow^* \mathit{False}\langle k\rangle$$

From linear to replicated data

Can we reuse a boolean? No...

Example: show that we don't have

$$\boldsymbol{\nu} l.(\mathit{True}\langle l\rangle \mid \mathit{not}\langle l,k_0\rangle \mid \mathit{not}\langle l,k_1\rangle) \longrightarrow^* \mathit{False}\langle k_0\rangle \mid \mathit{False}\langle k_1\rangle$$

Why? After we use $\mathit{True}\langle l\rangle$ once, we "exhaust" it. The solution is to use replication:

$$\begin{aligned} \mathit{True}' &= (l).!l(t,f).\bar{t} \\ \mathit{False}' &= (l).!l(t,f).\overline{f} \end{aligned}$$

Lists

In Ocaml,

```
type 'a list = Nil | Cons of 'a * 'a list;;
let cases xs n c =
  match xs with Nil -> n | Cons (y, ys) -> c y ys;;
```

In π-calculus,

$$\begin{aligned} \mathit{Nil} &= (l).!l(n,c).\overline{n} \\ \mathit{Cons}(H,T) &= (l).\boldsymbol{\nu} h,t.(!l(n,c).\bar{c}\langle h,t\rangle \mid H\langle h\rangle \mid T\langle t\rangle) \\ \mathit{cases}(P,F) &= (l).\boldsymbol{\nu} n,c.(\bar{l}\langle n,c\rangle \mid (n.P + c(h,t).F\langle h,t\rangle)) \\ \mathit{copy} &= (l,m).\mathit{cases}(\mathit{Nil}\langle m\rangle, \\ &\qquad (h,t).\boldsymbol{\nu} t'.(!m(n,c).\bar{c}\langle h,t'\rangle \mid \mathit{copy}\langle t,t'\rangle) \\ &\qquad)\langle l\rangle \end{aligned}$$

Example: show that for all lists L made from Nil and $\mathit{Cons}(-,-)$,

$$\boldsymbol{\nu} l.(L\langle l\rangle \mid \mathit{copy}\langle l,m\rangle) \approx L\langle m\rangle$$

Note that it's cheating to use copy recursively...

Interlude: encoding recursive definitions in terms of replication

Consider the recursive abstraction ("definition" in CCS):

$$F = (\vec{x}).P$$

where P may well contain recursive calls to F of the form $F\langle\vec{z}\rangle$.

We can replace the RHS with the following process abstraction containing no mention of F:

$$(\vec{x}).\boldsymbol{\nu} f.(\overline{f}\langle\vec{x}\rangle \mid !f(\vec{x}).\{\overline{f}/F\}P)$$

provided that f is fresh.

Example: compare the transitions of $F\langle u,v\rangle$, where $F = (x,y).\overline{x}y.F\langle y,x\rangle$ to those of its encoding. Notice the extra τ steps.

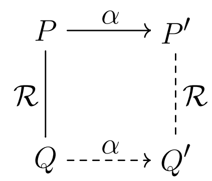
List append

```
let rec append xs zs =
  cases xs zs (fun y -> fun ys -> Cons(y, append ys zs));;
```

$$\begin{aligned} \mathit{append} = (k, l, m).\mathit{cases}(&\mathit{copy}\langle l, m\rangle, \\ &(h, t).\boldsymbol{\nu} t'.(!m(n, c).\overline{c}\langle h, t'\rangle \mid \mathit{append}\langle t, l, t'\rangle) \\ &)\langle k\rangle \end{aligned}$$

Strong bisimulation

A relation $\mathcal{R}$ is a strong bisimulation if for all $(P, Q) \in \mathcal{R}$ and $P \xrightarrow{\alpha} P'$, where $\mathsf{bn}(\alpha) \cap \mathsf{fn}(Q) = \varnothing$, there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{R}$, and symmetrically.



Strong bisimilarity $\sim$ is the largest strong bisimulation.

Bisimulation proofs

Theorem: $P \equiv Q$ implies $P \sim Q$.
Can you think of a counterexample to the converse?

Some easy results:

1. $P \mid \mathbf{0} \sim P$
2. $\overline{x}y.\boldsymbol{\nu} z.P \sim \boldsymbol{\nu} z.\overline{x}y.P$, if $z \notin \{x, y\}$
3. $x(y).\boldsymbol{\nu} z.P \sim \boldsymbol{\nu} z.x(y).P$, if $z \notin \{x, y\}$
4. $!\boldsymbol{\nu} z.P \not\sim \boldsymbol{\nu} z.!P$ for some P

More difficult:

1. $\boldsymbol{\nu} x.P \mid Q \sim \boldsymbol{\nu} x.(P \mid Q)$, for $x \notin \mathsf{fn}(Q)$
2. $P \sim Q$ implies $P \mid S \sim Q \mid S$
3. $!P \mid !P \sim !P$
4. $!!P \sim !P$

Congruence with respect to parallel

Theorem: $P \sim Q$ implies $P \mid S \sim Q \mid S$

Proof: Consider $\mathcal{R} = \{(P \mid S, Q \mid S) \ / \ P \sim Q\}$. If we can show $\mathcal{R} \subseteq \sim$ then we're done: if $P \sim Q$, then $(P \mid S, Q \mid S) \in \mathcal{R}$, thus $P \mid S \sim Q \mid S$.

Claim: $\mathcal{R}$ is a bisimulation. Suppose $P \sim Q$ and $P \mid S \xrightarrow{\alpha} P_0$, where $\mathsf{bn}(\alpha) \cap \mathsf{fn}(Q \mid S) = \varnothing$.

What are the cases to consider?

Congruence with respect to parallel: case analysis

P is solely responsible:

- $P \xrightarrow{\alpha} P'$ and $P_0 = P' \mid S$ and $\mathsf{bn}(\alpha) \cap \mathsf{fn}(S) = \varnothing$

S is solely responsible:

- $S \xrightarrow{\alpha} S'$ and $P_0 = P \mid S'$ and $\mathsf{bn}(\alpha) \cap \mathsf{fn}(P) = \varnothing$

P and S are jointly responsible:

- $P \xrightarrow{\overline{x}y} P'$ and $S \xrightarrow{xy} S'$ and $P_0 = P' \mid S'$ and $\alpha = \tau$
- $P \xrightarrow{xy} P'$ and $S \xrightarrow{\overline{x}y} S'$ and $P_0 = P' \mid S'$ and $\alpha = \tau$
- $P \xrightarrow{\overline{x}(y)} P'$ and $S \xrightarrow{xy} S'$ and $P_0 = \boldsymbol{\nu}y.(P' \mid S')$ and $\alpha = \tau$ and $y \notin \mathsf{fn}(S)$
- $P \xrightarrow{xy} P'$ and $S \xrightarrow{\overline{x}(y)} S'$ and $P_0 = \boldsymbol{\nu}y.(P' \mid S')$ and $\alpha = \tau$ and $y \notin \mathsf{fn}(P)$: careful!

Congruence with respect to parallel: the tricky case

Case: $P \xrightarrow{xy} P'$ and $S \xrightarrow{\overline{x}(y)} S'$ and $P_0 = \boldsymbol{\nu}y.(P' \mid S')$ and $\alpha = \tau$ and $y \notin \mathsf{fn}(P)$. The following lemmas can help:

1. If $P \xrightarrow{xy} P'$ and $y \notin \mathsf{fn}(P)$ then $P \xrightarrow{xy'} \{y'/y\}P'$.
2. If $S \xrightarrow{\overline{x}(y)} S'$ and $y' \notin \mathsf{fn}(S)$ then $S \xrightarrow{\overline{x}(y')} \{y'/y\}S'$.

Now, let y' be fresh. We can apply both lemmas. By alpha-conversion, $P_0 = \boldsymbol{\nu}y'.(\{y'/y\}P' \mid \{y'/y\}S')$

Since $P \sim Q$, there exists Q'' such that $Q \xrightarrow{xy'} Q''$ and $\{y'/y\}P' \sim Q''$. Since y' is fresh,

$$Q \mid S \xrightarrow{\tau} \boldsymbol{\nu}y'.(Q'' \mid \{y'/y\}S')$$

Our bisimulation isn't big enough! Take instead:

$$\mathcal{R} = \{(\boldsymbol{\nu}\vec{z}.(P \mid S), \boldsymbol{\nu}\vec{z}.(Q \mid S)) \;/\; P \sim Q\}$$

Exercises for next lecture

1. I gave an imprecise argument that $!\boldsymbol{\nu}z.P \sim \boldsymbol{\nu}z.!P$ is not generally true.
   (a) Make the argument precise by giving a concrete process P and a sequence of labelled transitions showing that bisimulation doesn't hold.
   (b) Let us say that a process Q has a weak barb b, written $Q \Downarrow b$ if Q is eventually able to output on b, i.e. there exists Q_0, Q_1, and $\vec{y}$ such that $Q \longrightarrow^* \boldsymbol{\nu}\vec{y}.(\overline{b}u.Q_0 \mid Q_1)$ with $b \notin \vec{y}$.
   Find a context C that can distinguish the two processes above, i.e. such that $C[\boldsymbol{\nu}z.!P] \Downarrow b$ but not $C[!\boldsymbol{\nu}z.P] \Downarrow b$.
   (c) Give an example of a general class of processes P for which the bisimulation would hold?

2. Recall the encoding of recursive abstractions in terms of replication.
   (a) Write the process $F\langle x, y\rangle$ in terms of replication, where the abstraction F is defined as follows:
   $$F = (u, v).u.F\langle u, v\rangle$$
   (b) Consider the pair of mutually recursive definition
   $$G = (u, v).(u.H\langle u, v\rangle \mid k.H\langle u, v\rangle)$$
   $$H = (u, v).v.G\langle u, v\rangle$$
   Write the process $G\langle x, y\rangle$ in terms of replication. (Note that we didn't discuss the coding of mutually recursive definitions so you have to invent the technique yourself!)

3. Help the lecturer to get his lists right! Fix my broken result about lists (corrected in the slides) by showing:

$$\boldsymbol{\nu} l.(L\langle l\rangle \mid copy\langle l, m\rangle) \approx L\langle m\rangle$$

4. Write a process abstraction rev such that $rev\langle l, m\rangle$ takes the list located at l and produces a new list at m with the elements reversed. It may help to consider the definition of `rev` (and that of the auxiliary function `rev’`) in Ocaml:

```
let rec rev’ xs ys =
  match ys with Nil -> xs
      | Cons (z,zs) -> rev’ (Cons (z,xs)) zs;;
let rev ys = rev’ Nil ys;;
```


5. Prove $!P \mid !P \sim !P$. To make the problem easier, replace the labelled transition rule for replication by the following ones that make the analysis much easier:

$$\frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \text{if } \mathsf{bn}(\alpha) \cap \mathsf{fn}(P) = \varnothing \quad \text{(lab-bang-simple)}$$

$$\frac{P \xrightarrow{\overline{x}y} P' \qquad P \xrightarrow{xy} P''}{!P \xrightarrow{\tau} (P' \mid P'') \mid !P} \quad \text{(lab-bang-comm)}$$

$$\frac{P \xrightarrow{\overline{x}(y)} P' \qquad P \xrightarrow{xy} P''}{!P \xrightarrow{\tau} \boldsymbol{\nu} y.(P' \mid P'') \mid !P} \text{if } y \notin \mathsf{fn}(P) \quad \text{(lab-bang-close)}$$

Furthermore, feel free to use structural congruence (e.g. $!P \equiv P \mid !P$) instead of process equality anywhere you need it in the proof.