Foundations of Programming

– Concurrency –

Session 13 – April 29, 2002

Uwe Nestmann

EPFL-LAMP

Goals

- Session 13
  - encodings in π
  - towards implementation: asynchrony
  - towards distribution: from π to join
  - from λ to join ?
  - encodings between π & join
- Session 14
  - back to funnel / functional nets

Encoding Tuples

$$\llbracket\, \overline{y}\langle\vec{z}\rangle.P \,\rrbracket \stackrel{\text{def}}{=}$$
$$\llbracket\, y(\vec{x}).P \,\rrbracket \stackrel{\text{def}}{=}$$

Think about:

$$\overline{y}\langle z_1, z_2\rangle.P \mid y(x_1, x_2, x_3).Q \;\longrightarrow$$
$$\overline{y}\langle z_1, z_2\rangle.P \mid y(x_1, x_2).Q \mid \overline{y}\langle w_1, w_2\rangle.R \;\longrightarrow$$

$$\llbracket\, \overline{y}\langle\vec{z}\rangle.P \,\rrbracket \stackrel{\text{def}}{=}$$
$$\llbracket\, y(\vec{x}).P \,\rrbracket \stackrel{\text{def}}{=}$$

Implementing the Pi-Calculus

- goal: design of a programming language on top of π (just like functional languages on top of λ)
- observation: certain constructs are both
  - difficult to implement
  - misinterpretable for the purpose of verification
  - replacable by more primitive notions
- result: a simpler, but (almost) equally expressive pi-calculus
  - only asynchronous output
  - no choice/summation

The Asynchronous Pi-Calculus

$$P, Q \quad ::= \quad \underset{\text{restriction}}{(\nu y)\, P} \quad \Big| \quad \underset{\text{output}}{\overline{y}\langle \tilde{z} \rangle} \quad \Big| \quad \underset{\text{input}}{y(\tilde{x}).P} \quad \Big| \quad \underset{\text{parallel}}{P|Q} \quad \Big| \quad \underset{\text{replication}}{*P}$$

+ large amount of theory & techniques
  equivalences, (sub-) types, polymorphism, tools ...
+ enormous expressive power
  functions, ADTs, objects, classes, constraints ...
+ efficient, type-safe implementation **(Nomadic) Pict**
  [Pierce, Turner '93–'97], [Wojciechowski '98–'02]
– **Pict** on monoprocessor: only quasi-parallel

Encoding Synchrony

$$[\![P_1 \mid P_2]\!] \stackrel{\text{def}}{=} [\![P_1]\!] \mid [\![P_2]\!]$$

$$\vdots$$

$$[\![\overline{y}\langle z \rangle.P]\!] \stackrel{\text{def}}{=}$$

$$[\![y(x).P]\!] \stackrel{\text{def}}{=}$$

Encoding Summation

- ☐ (only shown on demand)

Encoding Lambda-Calculus

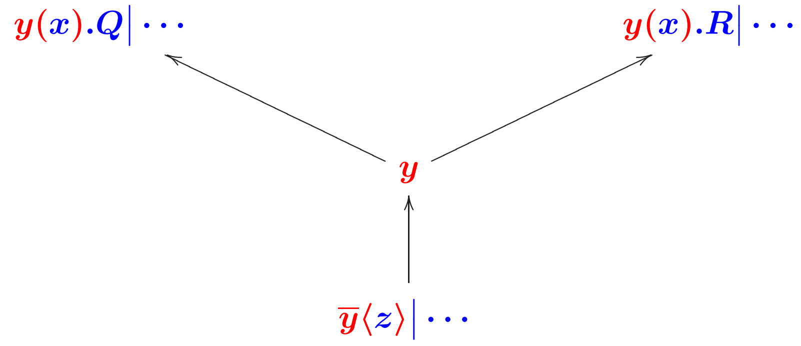
$$\llbracket\, x \,\rrbracket(\, u \,) \quad \stackrel{\text{def}}{=} \quad \overline{x}\langle u \rangle$$

$$\llbracket\, \lambda x\, M \,\rrbracket(\, u \,) \quad \stackrel{\text{def}}{=} \quad u(x, v).\llbracket\, M \,\rrbracket\langle\, v \,\rangle$$

$$\begin{aligned} \llbracket\, (M N) \,\rrbracket(\, u \,) \quad \stackrel{\text{def}}{=} \quad (\nu v)\ \big(\ & \llbracket\, M \,\rrbracket\langle\, v \,\rangle \\ \mid\ & (\nu x)\, \overline{v}\langle x, u \rangle \\ \mid\ & *x(u).\llbracket\, N \,\rrbracket\langle\, u \,\rangle\ \big) \end{aligned}$$

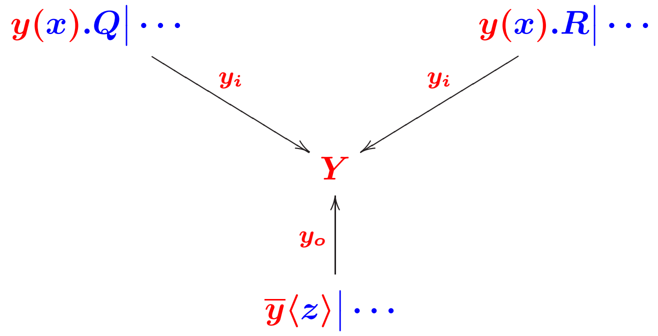
Try to evaluate/encode $(\cdots((M_0 N_1) N_2) \cdots)$

Distributed Implementation



- nearly every communication requires to solve a global consensus problem

Solution: Channel Managers



- **LOCALITY**: at most 1 receiver per channel
- $Y \stackrel{\text{def}}{=} *y_i(a).y_o(z).\overline{a}\langle z\rangle$

Locality syntactically: π^{def}

channel manager

residence site = creation site

avoids unnecessary global communications

$$\boxed{\begin{array}{r}(\nu y)\ P\\ y(x).P\\ *P\end{array}}$$

$$\textbf{def}\ y(x)=P\ \textbf{in}\ Q$$

$$(\nu y)\ (\ *\,y(x).P\ \mid\ Q\ )$$

channel managers are like function **def**initions
unique (replicated) receivers for the defined channels

Consequences (I)

. . . restricted π-notation . . .

REPLICATION

$$y(x).P\ \mid\ \overline{y}\langle z\rangle \qquad \textbf{non}$$

$$(\nu y)\ (\ *y(x).P\ \mid\ \overline{y}\langle z\rangle\ ) \qquad \textbf{oui}$$

SHARING

$$*y(x).P_1\ \mid\ *y(x).P_2\ \mid\ \overline{y}\langle z\rangle \qquad \textbf{non}$$

$$(\nu y)\ (\ *y(x).P\ \mid\ \overline{y}\langle z_1\rangle\ \mid\ \overline{y}\langle z_2\rangle\ ) \qquad \textbf{oui}$$

Consequences (II)

... restricted π-notation ...

PREFIX-NESTING

$(\nu y)\ (\ *y(x).u(v).P\ )$ **non**

$(\nu y)\ (\ *y(x).\overline{u}\langle v\rangle\ )$ **oui**

$\longrightarrow$ **in particular: INVERSION-OF-POLARITY**

$(\nu y)\ (\ *y(x).x(u).P\ )$ **non**

$(\nu y)\ (\ *y(x).\overline{x}\langle u\rangle\ )$ **oui**

$(\nu y)\ (\ *y(x).\overline{u}\langle x\rangle\ )$ **oui**

Definition π^{def}

CORE SYNTAX with y, x channel (names):

$$\begin{array}{rcl} D & ::= & y(x){=}P \\ P, Q & ::= & \textbf{def } D \textbf{ in } Q \quad | \quad y(x) \quad | \quad P|Q \end{array}$$

REDUCTION SEMANTICS

1 computational rule: $\textbf{def } y(x){=}P \textbf{ in } (Q|y(z))$

$\longrightarrow \textbf{def } y(x){=}P \textbf{ in } (Q|P[^z/_x])$

+ structural rules ...

Examples π^{def}

FORWARDER

$$\textbf{def } y(x)=u(x) \textbf{ in } y(z)$$
$$\longrightarrow \textbf{def } y(x)=u(x) \textbf{ in } u(z)$$

APPLICATOR

$$\textbf{def } \mathsf{eval}(f,x)=f(x) \textbf{ in } \quad \mathsf{eval}(\mathsf{square}, 5) \quad | \cdots$$
$$\longrightarrow \textbf{def } \mathsf{eval}(f,x)=f(x) \textbf{ in } \quad \mathsf{square}(5) \quad | \cdots$$

Expressiveness?

π^{def} is not expressive enough:

$$\textbf{def } D \textbf{ in } (P|Q) \triangleq \textbf{def } D \textbf{ in } P \;|\; \textbf{def } D \textbf{ in } Q$$

- no synchronization over parallel composition
- only local/functional computations
  by sending and receiving individual messages

Join-Synchronization: π_j

CORE SYNTAX with y, x, u, w channel (names):

$$D \quad ::= \quad \overbrace{y(x)|u(w)}^{J} = P$$

$$P, Q \quad ::= \quad \textbf{def } D \textbf{ in } Q \quad | \quad y(x) \quad | \quad P|Q$$

REDUCTION SEMANTICS generalization of π^{def}:

1 computational rule: $\quad \textbf{def } J{=}P \textbf{ in } (Q|J\sigma)$

$\longrightarrow \textbf{def } J{=}P \textbf{ in } (Q|P\sigma)$

+ structural rules $\quad \ldots$

Examples π_j (I)

Let D be defined as $y_1(x_1)|y_2(x_2) \; = \; P$.

$a)$ $\textbf{def } D \textbf{ in } a(z_1)|b(z_2)|c(z_3)$

$\longrightarrow$

$b)$ $\textbf{def } D \textbf{ in } y_1(z)$

$\longrightarrow$

$c)$ $\textbf{def } D \textbf{ in } y_1(z_1)|y_2(z_2)|Q$

$\longrightarrow$

Examples π_j (II)

MULTIPLEXER

$\textbf{def}\ y(x)|u(w) = z(u,w)\ \textbf{in}\ \ldots$

APPLICATOR

$\textbf{def}\ \mathsf{apply}(f)|\mathsf{args}(\tilde{w}) = f(\tilde{w})\ \textbf{in}\ \ldots$

PRINTER-SPOOLER

$$\begin{array}{ll} \textbf{def} & \mathsf{ready}(\mathsf{printer})\ |\ \mathsf{job}(\mathsf{doc}) \quad = \mathsf{printer}(\mathsf{doc}) \\ \textbf{in} & \mathsf{ready}(\mathsf{laser})\ |\ \mathsf{job}(\mathsf{ps})\ |\ \mathsf{job}(\mathsf{pdf}) \\ \longrightarrow & \ldots \end{array}$$

Expressiveness!

$$\llbracket\, \textbf{def}\ y(x)|u(w){=}P\ \textbf{in}\ Q\,\rrbracket \stackrel{\text{def}}{=} (\nu y,u)\ (y(x).u(w).\llbracket P\rrbracket \mid \llbracket Q\rrbracket)$$

$$\llbracket\, x(u)\,\rrbracket \stackrel{\text{def}}{=} \overline{x}\langle u\rangle$$

$$\llbracket\, P \mid Q\,\rrbracket \stackrel{\text{def}}{=} \llbracket P\rrbracket \mid \llbracket Q\rrbracket$$

$$\llbracket\, (\nu y)\ P\,\rrbracket \stackrel{\text{def}}{=} \textbf{def}\ y_o(x_o,x_i)|y_i(\kappa){=}\kappa(x_o,x_i)\ \textbf{in}\ \llbracket P\rrbracket$$

$$\llbracket\, \overline{y}\langle z\rangle\,\rrbracket \stackrel{\text{def}}{=} y_o(z_o,z_i)$$

$$\llbracket\, y(x).P\,\rrbracket \stackrel{\text{def}}{=} \textbf{def}\ \kappa(x_o,x_i){=}\llbracket P\rrbracket\ \textbf{in}\ y_i(\kappa)$$

π_j-calculus “ $\approx$ ” π_a-calculus