Il π-calcolo: Sintassi e Semantica

- Iniziamo col definire il π-calcolo **monadico**, in cui viene trasmesso un nome di canale alla volta.
- Si supponga di avere un insieme infinito $\mathcal{N}$ di nomi (di canali): $a, b, c, \ldots, x, y, z$.
- I **prefissi**, denotati dalla metavariabile π, sono una generalizzazione di quelli già viste precedentemente:

$$
\begin{array}{lcll}
\pi & ::= & x(y) & \text{ricevi sul canale } x \text{ un qualche nome } b\text{, ed istanzia } y \text{ con } b \\
 & & \overline{x}\langle b\rangle & \text{spedisci } b \text{ sul canale } x \\
 & & \tau & \text{azione interna non osservabile.}
\end{array}
$$

Definizione 78 *L'insieme* $\mathcal{P}_\pi$ *dei* **processi π-calcolo** *è definito dalla seguente sintassi:*

$$P ::= \sum_{i\in I} \pi_i.P_i \quad \Big| \quad P_1 \mid P_2 \quad \Big| \quad (\nu a)P \quad \Big| \quad !P$$

dove I è un insieme finito di indici.

- Nella sommatoria $\sum_{i \in I} \pi_i.P_i$ diremo che P_i è guardato da π_i.
- Al solito $\mathbf{0}$ è la somma vuota e $\overline{x}\langle y \rangle$ è un'abbreviazione per $\overline{x}\langle y \rangle.\mathbf{0}$.
- Entrambi gli operatori (νy) e $x(y)$ legano il nome y; al contrario y è libero in $\overline{x}\langle y \rangle$. In generale useremo parentesi tonde per contenere nomi legati.
- Nota che nella sintassi non abbiamo più identificatori di processi (a cui associavamo delle equazioni) ma il costrutto $!P$, detto **replicazione**. Intutivamente $!P$ può fornire un numero infinito di copie di P in parallelo tra loro:

$$!P \equiv P \mid P \mid \ldots \mid P \mid !P.$$

- In seguito vedremo che la replicazione, insieme al passaggio di nomi, consente di codificare facilmente processi definiti tramite equazioni ricorsive.

Nomi liberi e nomi legati in π-calcolo

I **nome liberi**, $\mathsf{fn}(P)$, per un processo P in π-calcolo sono così definiti:

$$
\begin{aligned}
\mathsf{fn}(a(x).P) &\stackrel{\text{def}}{=} \{a\} \cup \mathsf{fn}(P) \setminus \{x\} \\
\mathsf{fn}(\overline{a}\langle b\rangle.P) &\stackrel{\text{def}}{=} \{a, b\} \cup \mathsf{fn}(P) \\
\mathsf{fn}(\textstyle\sum_{i\in I} \alpha_i.P_i) &\stackrel{\text{def}}{=} \cup_{i\in I} \mathsf{fn}(\alpha_i.P_i) \\
\mathsf{fn}(P_1 \mid P_2) &\stackrel{\text{def}}{=} \mathsf{fn}(P_1) \cup \mathsf{fn}(P_2) \\
\mathsf{fn}((\nu a)P) &\stackrel{\text{def}}{=} \mathsf{fn}(P) \setminus \{a\} \\
\mathsf{fn}(!P) &\stackrel{\text{def}}{=} \mathsf{fn}(P)
\end{aligned}
$$

Definiamo l'insieme dei **nomi legati**, $\mathsf{bn}(P)$, di un processo P, in cui a meno di α-conversione tutti i nomi legati sono differenti.

$$
\begin{array}{lcl}
\mathsf{bn}(a(x).P) & \stackrel{\text{def}}{=} & \{x\} \cup \mathsf{bn}(P) \\
\mathsf{bn}(\overline{a}\langle b\rangle.P) & \stackrel{\text{def}}{=} & \mathsf{bn}(P) \\
\mathsf{bn}(\sum_{i\in I} \alpha_i.P_i) & \stackrel{\text{def}}{=} & \cup_{i\in I}\mathsf{bn}(\alpha_i.P_i) \\
\mathsf{bn}(P_1 \mid P_2) & \stackrel{\text{def}}{=} & \mathsf{bn}(P_1) \cup \mathsf{bn}(P_2) \\
\mathsf{bn}((\nu a)P) & \stackrel{\text{def}}{=} & \{a\} \cup \mathsf{bn}(P)
\end{array}
$$

Qualcuno sa dire a cosa corrisponde $\mathsf{bn}(!P)$?
Esercizio: Calcolare i seguenti termini:

1. $\{{}^{s}\!/\!{}_{h}\}\{{}^{h}\!/\!{}_{r}\}((\nu h)\overline{r}.h.\mathbf{0} \mid h.r.\mathbf{0})$
2. $\{{}^{s}\!/\!{}_{h}\}\{{}^{h}\!/\!{}_{r}\}((\nu h)r(s).h(r).\overline{r}\langle s\rangle.\mathbf{0} \mid h(a).r(h).\overline{h}\langle\rangle.\mathbf{0})$
3. $\{{}^{s}\!/\!{}_{h}\}\{{}^{h}\!/\!{}_{r}\}((\nu h)r(s).(h(r).\overline{r}\langle s\rangle.0 \mid \overline{r}\langle s\rangle) \mid h(a).r(h).\overline{h}\langle r\rangle.0)$

Esempio 79 *Si consideri il processo*

$$P \stackrel{def}{=} (\nu z)((\overline{x}\langle y\rangle + z(w).\overline{w}\langle y\rangle) \mid x(u).\overline{u}\langle v\rangle \mid \overline{x}\langle z\rangle)$$

- *Nota che* x, y *e* v *sono i soli nomi* **liberi** *in* P.
- *Due azioni* $\overline{x}\langle y\rangle$ *e* $x(u)$ *sono dette* **complementari**.
- *Se due azioni complementari non sono guardate e non stanno nella stessa sommatoria, allora costituiscono un* **redex**.
- *Un redex può dar luogo ad una* **reazione** *del tipo* $P \xrightarrow{\tau} P'$ *da cui scaturisce una sostituzione del tipo* $\{y/u\}$.
- *Il processo* P *ha due redex: la coppia* $\overline{x}\langle y\rangle, x(u)$ *e* $x(u), \overline{x}\langle z\rangle$. *Così vi sono due possibili riduzioni* $P \xrightarrow{\tau} P_1$ *e* $P \xrightarrow{\tau} P_2$, *dove*

$$\begin{array}{rcl} P_1 & \stackrel{def}{=} & (\nu z)(\mathbf{0} \mid \overline{y}\langle v\rangle \mid \overline{x}\langle z\rangle) \\ P_2 & \stackrel{def}{=} & (\nu z)((\overline{x}\langle y\rangle + z(w).\overline{w}\langle y\rangle) \mid \overline{z}\langle v\rangle \mid \mathbf{0}) \end{array}$$

P_1 *non contiene ulteriori redex, mentre* P_2 *ne contiene un altro, la coppia* $z(w), \overline{z}\langle v\rangle$. *Così* $P_2 \xrightarrow{\tau} P_3$ *con*

$$P_3 = (\nu z)(\overline{v}\langle y\rangle \mid \mathbf{0} \mid \mathbf{0})$$

Congruenza strutturale

Come fatto per i processi concorrenti privi di mobilità dobbiamo dare la definizione di congruenza strutturale, $\equiv$, e prima di ciò generalizzare la nozione di contesto.

Definizione 80 *Un* **contesto** *è dato dalla seguente grammatica:*

$$\begin{array}{rcl} C[\cdot] & ::= & [\cdot] \quad | \quad \pi.C[\cdot] + M \quad | \quad M + \pi.C[\cdot] \quad | \quad (\nu a)C[\cdot] \\ & | & C[\cdot] \mid P \quad | \quad P \mid C[\cdot] \quad | \quad !C[\cdot]. \end{array}$$

I **contesti elementari** *sono:*

$$[\cdot], \quad \pi.[\cdot] + M, \quad M + \pi.[\cdot], \quad (\nu a)[\cdot], \quad [\cdot] \mid P, \quad P \mid [\cdot], \quad ![\cdot]$$

Definizione 81 (Congruenza tra processi) *Sia $\cong$ una relazione di equivalenza sull'insieme di processi $\mathcal{P}^\pi$. Diremo che la relazione $\cong$ è una congruenza se è preservata da tutti i contesti elementari; cioè, se* *ogni qual volta che $P \cong Q$ allora per ogni α, M ed R:*

$$\begin{array}{rcl} \pi.P + M & \cong & \pi.Q + M \\ M + \pi.P & \cong & M + \pi.Q \\ (\nu a)P & \cong & (\nu a)Q \\ P \mid R & \cong & Q \mid R \\ R \mid P & \cong & R \mid Q \\ !P & \cong & !Q \end{array}$$

Il seguente risultato è una facile conseguenza.

Proposizione 82 *Un'arbitraria relazione di equivalenza $\cong$ è una congruenza* **se e solo se** *per ogni contesto $C[\cdot]$, se $P \cong Q$ allora $C[P] \cong C[Q]$.*

Definizione 83 *La* **congruenza strutturale**, *scritta $\equiv$, è la congruenza su $\mathcal{P}^\pi$ determinata dalle seguenti equazioni:*

1. *$P =_\alpha Q$ implica $P \equiv Q$*
2. *Possiamo permutare gli addendi di una somma*

3. $(\mathcal{P}, \mid, \mathbf{0})$ *è un monoide commutativo, cioè*

$$P \mid \mathbf{0} \equiv P, \quad P \mid Q \equiv Q \mid P, \quad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$$

4. $(\nu a)(P \mid Q) \equiv P \mid (\nu a)Q$ *se* $a \notin \mathsf{fn}(P)$

   $(\nu a)\mathbf{0} \equiv \mathbf{0}, \quad (\nu ab)P \equiv (\nu ba)P$

5. $!P \equiv P \mid !P$ *(**nota questa regola**)*

La congruenza strutturale è come al solito la chiusura riflessiva, simmetrica, transitiva e per contesti dell'insieme di equazioni viste nella Definizione precedente

$$(\mathrm{Ide})\ \frac{-}{P \equiv P} \qquad (\mathrm{Sym})\ \frac{P \equiv Q}{Q \equiv P} \qquad (\mathrm{Tra})\ \frac{P \equiv Q \qquad Q \equiv R}{P \equiv R}$$

$$(\mathrm{Cxt})\ \frac{P \equiv Q}{C[P] \equiv C[Q]}$$

Fare Esercizi 9.8 – 9.11.

Definizione 84 *Un processo in* $\mathcal{P}^\pi$ *della forma*

$$(\nu\vec{a})(M_1 \mid \ldots \mid M_m \mid !Q_1 \mid \ldots \mid !Q_n)$$

è detto essere in **standard form** *se ciascun* M_i *è una somma non vuota e ciascun* Q_j *è esso stesso in standard form.*

Proposizione 85 *Ogni processo è strutturalmente congruente ad uno in standard form.*

Forniamo adesso la semantica operazionale per il π-calcolo. A tale scopo dobbiamo definire la sintassi delle azioni che possono essere eseguite dai processi:

$$\alpha \;\; ::= \;\; ab \quad \Big| \quad \overline{a}\langle b\rangle \quad \Big| \quad \overline{a}(b) \quad \Big| \quad \tau \;.$$

L'azione ab denota la ricezione sul canale a di un canale b. L'azione $\overline{a}\langle b\rangle$ (free output) denota la spedizione sul canale a del canale *globale* b. L'azione $\overline{a}(b)$ (bound output) denota la spedizione sul canale a del canale *privato* b. τ modella la sincronizzazione tra componenti parallele asincrone con scambio di valori (ovvero canali) attraverso un qualche canale.[3]

Si noti come in π-calcolo, diversamente dal CCS, le azioni α non coincidano con i prefissi π (visti precedentemente). Infatti, i prefissi π sono pezzi della sintassi del linguaggio, mentre le azioni α sono parte della semantica (operazionale) del linguaggio, ovvero dell'LTS.

[3]Un ringraziamento agli studenti che hanno fatto emergere l'inconsistenza presente nella versione precedente.

$$(\mathrm{INP_t}) \qquad a(x).P \xrightarrow{ab} \{b/x\}P \qquad\qquad (\mathrm{OUT_t}) \qquad \overline{a}\langle b\rangle.P \xrightarrow{\overline{a}\langle b\rangle} P$$

$$(\mathrm{SUM_t}) \ \frac{M_1 \xrightarrow{\alpha} M_2}{M + M_1 + N \xrightarrow{\alpha} M_2} \qquad\qquad (\mathrm{REACT_t}) \ \frac{P \xrightarrow{ab} P' \qquad Q \xrightarrow{\overline{a}\langle b\rangle} Q'}{\begin{array}{c} P \mid Q \xrightarrow{\tau} P' \mid Q' \\ Q \mid P \xrightarrow{\tau} Q' \mid P' \end{array}}$$

$$(\mathrm{L\text{-}PAR_t}) \ \frac{P \xrightarrow{\alpha} P' \quad \mathsf{bn}(\alpha)\cap\mathsf{fn}(Q)=\emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad (\mathrm{R\text{-}PAR_t}) \ \frac{Q \xrightarrow{\alpha} Q' \quad \mathsf{bn}(\alpha)\cap\mathsf{fn}(P)=\emptyset}{P \mid Q \xrightarrow{\alpha} P \mid Q'}$$

$$(\mathrm{REP_t}) \ \frac{P \mid !P \xrightarrow{\alpha} A}{!P \xrightarrow{\alpha} A}$$

Tabella 2: LTS per il π-calcolo – 1 parte

$$(\mathrm{RES}_{\mathsf{t}})\ \frac{P \xrightarrow{\alpha} P' \quad a \notin \mathsf{fn}(\alpha)}{(\nu a)P \xrightarrow{\alpha} (\nu a)P'} \qquad (\mathrm{OPEN}_{\mathsf{t}})\ \frac{P \xrightarrow{\overline{a}\langle b\rangle} P' \quad a \neq b}{(\nu b)P \xrightarrow{\overline{a}(b)} P'}$$

$$(\mathrm{CLOSE}_{\mathsf{t}})\ \frac{P \xrightarrow{ab} P' \qquad Q \xrightarrow{\overline{a}(b)} Q' \qquad b \notin \mathsf{fn}(P)}{\begin{array}{c} P \mid Q \xrightarrow{\tau} (\nu b)(P' \mid Q') \\ Q \mid P \xrightarrow{\tau} (\nu b)(Q' \mid P') \end{array}}$$

$$(\mathrm{ALPHA}_{\mathsf{t}})\ \frac{P =_\alpha P' \qquad P' \xrightarrow{\alpha} Q' \qquad Q =_\alpha Q'}{P \xrightarrow{\alpha} Q}$$

Tabella 3: LTS per il π-calcolo – 2 parte

Importante: Si noti che:

- la regola ($\mathrm{INP_t}$) è ben definita, ovvero non è necessario richiedere $b \notin \mathsf{bn}(P)$. Qualcuno sa dire perchè?
- Si noti che la condizione nelle regole ($\mathrm{L\text{-}PAR_t}$) e ($\mathrm{R\text{-}PAR_t}$) impedisce di mescolare nomi free con nomi bound nel caso in cui α sia un un bound output. Nel caso in cui α contenga un nome bound presente nella componente parallela posso sempre utilizzare la regola ($\mathrm{ALPHA_t}$) per eseguire un bound output con un diverso nome bound non presente nella componente parallela.
- L'introduzione di tale condizione garantisce che la regola ($\mathrm{CLOSE_t}$) non catturi erroneamente nomi liberi del processo Q.

π-calcolo poliadico vs π-calcolo monadico

- In molti esempi, come quello della telefonia mobile, è utile avere comunicazioni poliadiche, in cui cioè vengono trasmessi più nomi alla volta.

- Comunque, la comunicazione poliadica non introduce nuovo potere espressivo ma può essere codificata in termini di quella monadica.

- È possibile cioè codificare i costrutti di input ed output poliadici

$$x(y_1, \ldots, y_n).P \text{ and } \overline{x}\langle z_1, \ldots, z_n\rangle.Q$$

  in termini di π-calcolo monadico.

- Il primo tentativo che viene in mente sarebbe di tradurli in

$$x(y_1).\ldots.x(y_n).\widehat{P} \text{ and } \overline{x}\langle z_1\rangle.\ldots.\overline{x}\langle z_n\rangle.\widehat{Q}$$

  dove $\widehat{P}$ e $\widehat{Q}$ denotano l'encoding di P e Q, rispettivamente. Comunque, una tale traduzione sarebbe scorretta. Infatti si consideri il processo poliadico

$$x(y_1, y_2).P \ | \ \overline{x}\langle z_1, z_2\rangle.Q \ | \ \overline{x}\langle z_1', z_2'\rangle.Q'$$

in cui i nomi y_1 e y_2 vengono sostituiti da z_1 e z_2 oppure z_1' e z_2'.

- Ma allora la sua traduzione nel π-calcolo monadico produrrebbe

$$x(y_1).x(y_2).\widehat{P} \ | \ \overline{x}\langle z_1\rangle.\overline{x}\langle z_2\rangle.\widehat{Q} \ | \ \overline{x}\langle z_1'\rangle.\overline{x}\langle z_2'\rangle.\widehat{Q'}$$

In cui è possibile mescolare i valori forniti dai due output. Ad esempio y_1 e y_2 potrebbero essere sostituiti da z_1' e z_1.

- Per implementare l'encoding correttamente è necessario garantire che non vi siano **interferenze** sul canale x. Un modo per implementare ciò è quello di stabilire fra i due partner un **canale privato** w per trasmettere (indisturbatamente) l'n-pla di valori in sequenza. Poichè il canale è privato, non vi saranno pericoli di interferenze.

$$\begin{array}{rcl} x(y_1, \ldots y_n).P & \longmapsto & x(w).w(y_1).\ldots.w(y_n).\widehat{P} \\ \overline{x}\langle z_1, \ldots, z_n\rangle.Q & \longmapsto & (\nu w)(\overline{x}\langle w\rangle.\overline{w}\langle z_1\rangle.\ldots.\overline{w}\langle z_n\rangle.\widehat{Q}) \end{array}$$

L'encoding è quindi un omomorfismo su tutti gli operatori eccetto per input ed output. L'encoding completo è fornito nella seguente tabella. Si noti che nell'encoding dell'input e dell'output, w è scelto fresh; ovvero, $w \notin \mathsf{fn}(P, Q)$.

$$\begin{aligned}
[\![\mathbf{0}]\!] &\stackrel{\text{def}}{=} \mathbf{0} \\
[\![(\nu a)P]\!] &\stackrel{\text{def}}{=} (\nu a)[\![P]\!] \\
[\![P \mid Q]\!] &\stackrel{\text{def}}{=} [\![P]\!] \mid [\![Q]\!] \\
[\![M + N]\!] &\stackrel{\text{def}}{=} [\![M]\!] + [\![N]\!] \\
[\![!P]\!] &\stackrel{\text{def}}{=} ![\![P]\!] \\
[\![x(y_1, \ldots y_n).P]\!] &\stackrel{\text{def}}{=} x(w).w(y_1). \ldots .w(y_n).[\![P]\!] \\
[\![\overline{x}\langle z_1, \ldots, z_n\rangle.Q]\!] &\stackrel{\text{def}}{=} (\nu w)(\overline{x}\langle w\rangle.\overline{w}\langle z_1\rangle. \ldots .\overline{w}\langle z_n\rangle.[\![Q]\!])
\end{aligned}$$

Tabella 4: Encoding del π-calcolo poliadico in quello monadico

- Questo encoding soddisfa una proprietà importante che si chiama **corrispondenza operazionale** (sulle azioni interne). Intuitivamente, il processo poliadico P ed il suo encoding $[\![P]\!]$ nel calcolo monadico hanno la medesima evoluzione interna, a meno di $\approx$.

  Teorema 86 (Corrispondenza operazionale di $[\![\cdot]\!]$) *Sia P un processo del π-calcolo poliadico,*

  - *Se esiste un processo poliadico P' tale che $P \xrightarrow{\tau} P'$, allora esiste un processo monadico P_1 tale che $[\![P]\!] \xrightarrow{\tau} P_1 \approx [\![P']\!]$;*
  - *Se esiste un processo monadico P_1 tale che $[\![P]\!] \xrightarrow{\tau} P_1$, allora esiste un processo poliadico P' tale che $P \xrightarrow{\tau} P'$ e $P_1 \approx [\![P']\!]$.*

Un esempio:

$$
\begin{array}{rl}
\multicolumn{2}{l}{[\![x(y_1, y_2).P \mid \overline{x}\langle z_1, z_2\rangle.Q]\!]} \\
\xrightarrow{\tau} & (\nu w)\big(w(y_1).w(y_2).[\![P]\!] \mid \overline{w}\langle z_1\rangle.\overline{w}\langle z_2\rangle.[\![Q]\!]\big) \\
\sim & \tau.(\nu w)\big(w(y_2).\{z_1/y_1\}[\![P]\!] \mid \overline{w}\langle z_2\rangle.[\![Q]\!]\big) \\
\sim & \tau.\tau.(\nu w)\big(\{z_2/y_2\}\{z_1/y_1\}[\![P]\!] \mid [\![Q]\!]\big) \\
\approx & \{z_2/y_2\}\{z_1/y_1\}[\![P]\!] \mid [\![Q]\!]
\end{array}
$$

Si noti che poichè $w \notin \mathsf{fn}(P \mid Q)$, possiamo buttare via (νw).

- L'encoding è **adeguato**, ovvero non introduce errori. In pratica, se l'encoding di due processi P e Q dá luogo a processi equivalenti, allora P e Q sono equivalenti. Cioè non accade mai che due processi non equivalenti vengano trasformati dall'encoding in processi equivalenti.

Teorema 87 (Adeguatezza di $[\![\cdot]\!]$) *Siano P e Q due processi poliadici. Allora*

$$[\![P]\!] \approx [\![Q]\!] \textit{ implica } P \approx Q\ .$$

- L'adeguatezza è sicuramente una proprietà fondamentale per un encoding. Ma è solo in presenza della corrispondenza operazionale che tale proprietà ha senso.

- Un encoding da un linguaggio ad un altro è detto **completo** se vale anche il viceversa, cioè se

$$P \approx Q \text{ implica } [\![P]\!] \approx [\![Q]\!] \ .$$

- **Domanda:** L'encoding del poliadico nel monadico non è completo. Qualcuno di voi è in grado di mostrarlo con un controesempio?

- Concludendo, l'encoding del π-calcolo poliadico nel monadico giustifica la scelta di considerare come calcolo fondamentale il calcolo monadico, chiaramente più semplice.

- Il π-calcolo monadico è anche stato usato come base per sviluppare un modelllo algebrico per **protocolli di comunicazione crittografici**, dove i nomi ristretti consentono di modellare **chiavi segrete**. A tal scopo Abadi e Gordon hanno sviluppato un'estensione del π-calcolo chiamata **Spi-calcolo**.

- Leggere **attentamente** la discussione sulla mobilità a pag. 92.

π-calcolo sincrono vs π-calcolo asincrono

- Nel π-calcolo visto finora la comunicazione è *sincrona*. In pratica, il processo che spedisce un messaggio e quello che lo riceve si sincronizzano, ovvero si aspettano per poter eseguire insieme la trasmissione del messaggio. Solo dopo che la comunicazione ha avuto luogo, i due processi possono continuare la loro evoluzione.

- Però quando si ha a che fare con *sistemi concorrenti distribuiti*, spedizione e ricezione debbono necessariamente avvenire in maniera asimmetrica, ovvero non allo stesso tempo.

- Infatti, la ricezione di messaggi è facile da determinare, mentre, in assenza di un meccanismo di acknowledgment, non si può stabilire se un messaggio spedito verso l'esterno è stato veramente consegnato al destinatario. Un esempio molto calzante è lo scambio di messaggi via email.

- Il π-*calcolo asincrono* non è altro che un sottocalcolo del π-calcolo in cui gli output hanno sempre la forma

$$\overline{a}\langle b\rangle.\mathbf{0}$$

abbreviato, $\overline{a}\langle b\rangle$, ovvero con continuazione nulla. La continuazione è nulla proprio perchè in un sistema distribuito un processo di output non è in grado di sapere quando un messaggio inviato sia stato effettivamente ricevuto dal destinatario. E quindi, non ha senso mettere una continuazione se non si è in grado di dire quando va eseguita.

- Inoltre, nel π-calcolo asincrono le somme sono prefissate solo da input.

- Comunque, così come avviene per SMS e email, possiamo sempre *simulare la trasmissione sincrona in termini di quella asincrona*, usando proprio dei meccanismi di acknowledgment.

- Mostriamo ciò fornendo un encoding del π-calcolo sincrono in termini di quello asincrono.

- Anche in questo caso l'encoding è un omomorfismo, eccezion fatta per gli operatori di input ed output. Nell'encoding di tali operatori i nomi u, w sono fresh; ovvero $\{u, w\} \cap \mathsf{fn}(P, Q) = \emptyset$. Si noti come

$$
\begin{aligned}
[\![\mathbf{0}]\!] &\stackrel{\text{def}}{=} \mathbf{0} \\
[\![(\nu a)P]\!] &\stackrel{\text{def}}{=} (\nu a)[\![P]\!] \\
[\![P \mid Q]\!] &\stackrel{\text{def}}{=} [\![P]\!] \mid [\![Q]\!] \\
[\![M + N]\!] &\stackrel{\text{def}}{=} [\![M]\!] + [\![N]\!] \\
[\![!P]\!] &\stackrel{\text{def}}{=} ![\![P]\!] \\
[\![x(y).P]\!] &\stackrel{\text{def}}{=} x(u).(\nu w)(\overline{u}\langle w\rangle \mid w(y).[\![P]\!]) \\
[\![\overline{x}\langle z\rangle.Q]\!] &\stackrel{\text{def}}{=} (\nu u)\big(\overline{x}\langle u\rangle \mid u(w).(\overline{w}\langle z\rangle \mid [\![Q]\!])\big)
\end{aligned}
$$

Tabella 5: Encoding del π-calcolo sincrono in quello asincrono nella codifica dell'output la continuazione $[\![Q]\!]$ viene eseguita solo dopo che il protocollo si acknowlegment ha avuto luogo.

- Anche questo encoding soddisfa una corrispondenza operazionale sulle τ-action, è adeguato ma non completo.

Definizioni ricorsive vs replicazione

- Negli esempi della telefonia mobile e dei buffers elastici abbiamo potuto apprezzare la comodità nell'avere anche in π-calcolo definizioni di processi ricorsivi della forma

$$A(\vec{x}) \stackrel{\mathrm{def}}{=} Q_A$$

dove A è un identificatore di processo e Q_A un processo che può contenere più invocazioni di A, eventualmente con parametri differenti. La semantica di un processo così definito deriva dalla legge $A\langle\vec{y}\rangle \equiv \{\vec{y}/\vec{x}\}Q_A$.

- Comunque, abbiamo visto che nel π-calcolo le definizioni ricorsive sono state sostituite con un costrutto nettamente più semplice: la replicazione.

- La giustificazione di ciò deriva dal fatto che le definizioni ricorsive possono essere facilmente codificate usando la replicazione.

- I punti chiave di tale traduzione sono:

  – Il **passaggio dei parametri** viene implementato attraverso il passaggio di nomi;

  – attraverso la replicazione si modellano **attivazione multiple** del processo A;

  – la restrizione modella lo **scope della definizione** di A.

- Vediamo come funziona. Supponiamo che si voglia codificare la definizione

$$A(\vec{x}) \stackrel{\text{def}}{=} Q_A, \text{ dove } Q_A = \ldots A\langle\vec{u}\rangle \ldots A\langle\vec{v}\rangle \ldots$$

e supponiamo che lo scope della definizione sia un processo P che contenga un certo numero di invocazioni di A:

$$P = \ldots A\langle\vec{y}\rangle \ldots A\langle\vec{z}\rangle \ldots$$

- Allora possiamo tradurre P nel seguente modo:
  - Inventiamo un nome (di canale) a, che verrà usato per invocare l'attivazione del processo A;
  - Dato un qualsiasi processo R, denotiamo con $\widehat{R}$ il processo ottenuto sostituendo in R tutte le invocazioni $A\langle\vec{w}\rangle$ con $\overline{a}\langle\vec{w}\rangle$;
  - Rimpiazzamo il processo P e le definizioni di A con il processo seguente:
$$\widehat{\widehat{P}} \stackrel{\text{def}}{=} (\nu a)(\widehat{P} \mid !a(\vec{x}).\widehat{Q_A})$$

Questa traduzione si estende facilmente al caso di definizioni mutualmente ricorsive.

Esercizio 88 (9.23 pag. 95)

Consideriamo la definizione di buffer vista nella sezione 8.3:

$$\begin{array}{rcl} B(l,r) & \stackrel{\text{def}}{=} & l(x).C\langle x,l,r\rangle \\ C(x,l,r) & \stackrel{\text{def}}{=} & \overline{r}\langle x\rangle.B\langle l,r\rangle \end{array}$$

e chiamiamo i due membri destri Q_B e Q_C, rispettivamente. Scriviamo $\widehat{Q_B}$ e $\widehat{Q_C}$.

$$\begin{array}{rcl} \widehat{Q_B} & \stackrel{\text{def}}{=} & l(x).\overline{c}\langle x,l,r\rangle \\ \widehat{Q_C} & \stackrel{\text{def}}{=} & \overline{r}\langle x\rangle.\overline{b}\langle l,r\rangle \end{array}$$

Supponiamo adesso che lo scope delle due definizioni sia il seguente processo:

$$P \stackrel{\text{def}}{=} \overline{l}\langle x_1\rangle.\overline{l}\langle x_2\rangle.\overline{l}\langle x_3\rangle \mid B_2 \mid r(y_1).r(y_2).r(y_3).\mathbf{0}$$

Dove B_2 è il buffer di capacità due così definito:

$$B_2 \stackrel{\text{def}}{=} (\nu m)(B\langle l,m\rangle \mid B\langle m,r\rangle)$$

Diamo allora la traduzione $\widehat{\widehat{P}}$.

$$\begin{aligned}\widehat{\widehat{P}} \quad \stackrel{\text{def}}{=} \quad (\nu b, c)\big(\quad & \bar{l}\langle x_1\rangle.\bar{l}\langle x_2\rangle.\bar{l}\langle x_3\rangle \mid (\nu m)(\bar{b}\langle l, m\rangle \mid \bar{b}\langle m, r\rangle) \\ & \mid r(y_1).r(y_2).r(y_3).\mathbf{0} \\ & \mid !b(l, r).\widehat{Q_B} \\ & \mid !c(x, l, r).\widehat{Q_C} \quad \big)\end{aligned}$$

- Se pensate a come avviene l'implementazione di procedure di un linguaggio ad alto livello, potete vedere $!a(\vec{x}).\widehat{Q_A}$ come il codice compilato corrispondente alla procedura A, mentre ogni output $\overline{a}\langle\vec{y}\rangle$, all'interno di $\widehat{Q_A}$ o altrove, come l'invocazione della procedura che dà luogo alla creazione di un nuovo **record di attivazione**.