

Fondamenti: Alcune Proprietà di Calcolabilità e dei Linguaggi per Funzioni Calcolabili

Sommario: 17-18 marzo, 2021

- Funzioni Parziali ed Esecuzioni Terminanti di Programmi
- Problemi Semidecidibili: Halting, Equivalenza, Ambiguità
- Linguaggi per \mathcal{F} : Lambda Calcolo (Church 1936)
- Logica Combinatoria (Shonfinkel 1924)
- Macchina a Stati (Minsky 1956, Wang 1957)
- Calcolo di Processi Mobili: π -Calcolo (Milner et al. 1992)

Lecture e Approfondimenti:

- [Barendregt90] H.P. Barendregt, Functional Programming and Lambda Calculus, in Handbook of Theoretical Computer Science, vol. B, Chapter 7, pp. 321-363, Elsevier Science Publishers, 1990
- [Minsky72] M. Minsky, Computation: Finite and Infinite Machines, Chapter 11, pp. 199-216, Prentice-Hall International, 1972.
- [Milner92] R. Milner, J. Parrow, D. Walker, A Calculus of Mobile Processes, Information and Computations 100, pp. 1-9, 41-49, 1992.

- **Funzione di Decisione** È una funzione booleana, g , totale, ovvero:
 - $g \in \mathcal{D} \rightarrow \{\text{true}, \text{false}\}$ per due valori $\text{true}, \text{false} \in \mathcal{D}$
 - $\forall x \in \mathcal{D}, g(x) \in \{\text{true}, \text{false}\}$
- **Funzione di semi-Decisione** È una funzione parziale g , ovvero:
 - $g \in \mathcal{D} \rightarrow \{\text{true}\}$ per $\text{true} \in \mathcal{D}$
 - $\forall x \in \mathcal{D}, g(x) = \text{true}$ oppure $g(x) = \uparrow$ ¹
- **Decidibilità** Le funzioni di decisione e semi-decisione, sono utilizzate per descrivere problemi calcolabili con caratteristiche diverse.

¹↑ significa *indefinito*: Ogni programma che esprime la funzione, calcolato su tale valore x è non terminante.

- **Ambiguità di un grammatica Libera.** È semi-decidibile se ambigua, ovvero:
 - esiste semi-decisione g , s.t. $g(x)=\text{true}$ sse x è ambigua
- **Equivalenza di due grammatiche Libere.** È semi-decidibile se diverse, ovvero:
 - esiste semi-decisione g , s.t. $g(x,y)=\text{true}$ sse $\mathcal{L}(x) \neq \mathcal{L}(y)$
 - Equivalenza di grammatiche regolari è decidibile
- **Appartenenza a $\mathcal{L}(G)$ per Libera G** È decidibile
 - esiste decisione g_G , s.t. $g_G(x) = \begin{cases} \text{true} & x \in \mathcal{L}(G) \\ \text{false} & x \notin \mathcal{L}(G) \end{cases}$
- **Terminazione Esecuzione Programma p .** È semi-decidibile se termina, ovvero:
 - esiste semi-decisione g , s.t. $g(p,d)=\text{true}$ sse p termina su input d
- **Equivalenza di Programmi.** È non decidibile
 - eccetto che per programmi terminanti (vedi esercizio in Esercizi5L5)

- **Sintassi** (ovviamente, astratta)

$$\Lambda = X \mid \Pi \mid \lambda X. \Lambda \mid \Lambda \Lambda$$

- **Termini:** Λ, g, t, t_1 . Insieme (numerabile) dei termini.
 - **Variabili:** X, x, y, z . Insieme (numerabile) di simboli detti variabili.
 - **Costanti:** Π, c, \dots . Insieme (numerabile) di simboli detti costanti (distinguibili: $X \cap \Pi = \{\}$)
 - **Op. Astrazione:** $\lambda x. t$. Operazione binaria che esprime il valore (funzionale) ottenuto da t per *astrazione* (generalizzazione) rispetto alla variabile (libera) x . Il termine è anche detto, funzione di t nella variabile x .
 - **Op. Applicazione:** $t_1 t_2$. Operazione binaria che esprime il valore ottenuto per *applicazione* del termine t_1 al termine t_2 . Il termine è anche detto, applicazione di t_1 a t_2 . Ha notazione implicita via giustapposizione degli argomenti.
- **Esempi**

$[\@ - ([x], [y])]$
 $[\lambda - ([x], [\lambda - ([y], [\@ - ([x], [y])]])]$ (grafica migliora la lettura!)

$$\Lambda = X \mid \Pi \mid \lambda X. \Lambda \mid \Lambda \Lambda$$

• Sintassi Concreta e Variabili Libere

- **Applicazione.** è associativa a sinistra:
 $x y x$ significa $((x y) x)$
- **Applicazione.** ha priorità sull'Astrazione:
 $\lambda y. x y$ significa $\lambda y. (x y)$
- **Variabile Legata, Libera, Occorrenze.** Le variabili che occorrono in un termine t sono raccolte in $\text{Var}(t)$, e si dividono Legate, $\text{BV}(t)$, o Libere, $\text{FV}(t)$, a seconda che siano state astratte o meno.

$t \in \Lambda$	$\text{BV}(t)$	$\text{FV}(t)$
$x \in X$	$\{\}$	$\{x\}$
$c \in \Pi$	$\{\}$	$\{\}$
$\lambda x. t_1$	$\{x\} \cup \text{BV}(t_1)$	$\text{FV}(t_1) \setminus \{x\}$
$t_1 t_2$	$\text{BV}(t_1) \cup \text{BV}(t_2)$	$\text{FV}(t_1) \cup \text{FV}(t_2)$

$$\text{Var}(t) = \text{BV}(t) \cup \text{FV}(t)$$

• Esempi

Sia $t \equiv \lambda y. \lambda x. y(y x x)$. Possiamo omettere le parentesi?

Sia $t \equiv \lambda y. \lambda x. x z$. Abbiamo: $\text{Var}(t) = \{x, y, z\}$, $\text{FV}(t) = \{z\}$, $\text{BV}(t) = \{x, y\}$.

Sia $t \equiv \lambda y. \lambda x. (\lambda x. x + 5)(x + y)$. Abbiamo: ...

$$\Lambda = X \mid \Pi \mid \lambda X. \Lambda \mid \Lambda \Lambda$$

• Semantica

- Una relazione \rightarrow definita da 3 regole ed estesa in una congruenza su Λ .

- α – red.

$$\lambda x. t \rightarrow_{\alpha} \lambda y. t[y/x] \quad \text{per } y \notin \text{Var}(t)$$

- β – red.

$$(\lambda x. t)t_2 \rightarrow_{\beta} t[t_2/x] \quad \text{per } \text{FV}(t_2) \cap \text{BV}(t) = \{\}$$

- η – red.

$$\lambda x. (t x) \rightarrow_{\eta} t \quad \text{per } x \notin \text{FV}(t)$$

- Sostituzione. $t_1[t_2/x]$ è l'operazione che rimpiazza, in t_1 , ogni occorrenza libera di x con il termine t_2 .

$$x[t/x] = t$$

$$y[t/x] = y \text{ con } x \neq y; \quad c[t/x] = c$$

$$(\lambda x. t)[t_2/x] = \lambda x. t; \quad (\lambda y. t)[t_2/x] = \lambda y. (t[t_2/x]) \text{ con } x \neq y$$

$$(t_1 t_2)[t_3/x] = (t_1[t_3/x])(t_2[t_3/x])$$

• Esempi

Sia $t \equiv \lambda y. \lambda x. y(y(x))$. Quali regole sono applicabili a t ?

$$\Lambda = X \mid \Pi \mid \lambda X. \Lambda \mid \Lambda \Lambda$$

- **Valutazione, Computazione** di $t \in \Lambda$

- Una sequenza:

$$t_0 \rightarrow_{r_1} t_1 \quad \dots \quad \rightarrow_{r_n} t_n$$

tale che:

- $(t \equiv t_0) \wedge (n \geq 0)$
- annotazioni $r_i \in \{\alpha, \beta, \eta\}$
- $t_n \rightarrow_{r_{n+1}} t_{n+1}$ solo se $r_{n+1} \equiv \alpha$

- **Esempi**

Sia $t \equiv \lambda x. (\lambda y. x y) (\lambda x. y x)$. Vediamo una valutazione di t :

$$t \rightarrow_{\eta} \lambda x. x (\lambda x. y x) \rightarrow_{\eta} \lambda x. x y$$

Non è unica:

$$t \rightarrow_{\beta} \lambda x. x (\lambda x. y x) \rightarrow_{\eta} \lambda x. x y$$

$$\Lambda = X \mid \Pi \mid \lambda X. \Lambda \mid \Lambda \Lambda$$

- **Programmi** Sono tutti i termini chiusi, i.e.

$$\mathcal{P} = \{t \in \Lambda \mid FV(t) = \{\}\}$$

- **Aritmetica**

- $[0] \equiv \lambda y. \lambda x. x$
- $[n + 1] \equiv \lambda y. \lambda x. y([n]yx)$
- $\text{succ} \equiv \lambda z. \lambda y. \lambda x. y(zyx)$
- plus, prod, minus, div, ...

- **Conditional e booleani**

- $\text{if} \equiv \lambda b. \lambda x. \lambda y. b \ x \ y$
- $\text{true} \equiv \lambda x. \lambda y. x$
- $\text{false} \equiv \lambda x. \lambda y. y$
- zero, and, or, eq, ...

- **Recursion-FixedPoint**

- $\Psi \equiv \lambda g. (\lambda x. g(xx))(\lambda x. g(xx))$

$$\Lambda = X \mid \Pi \mid \lambda X. \Lambda \mid \Lambda \Lambda$$

- **Aritmetica**
- **Programmi**
- **Conditional e costanti booleane**
- **Recursion-FixedPoint**

- $\Psi \equiv \lambda g. (\lambda x. g(xx)) (\lambda x. g(xx))$

- $\forall h \in \mathcal{F}, \forall d \in \mathcal{D},$

$$\Psi(h)(d) = h(\Psi(h))(d) \quad (\text{vedi esercizio L5.7})$$

- Sia $g = \lambda x. t$ una definizione per una funzione g ²:


$$g \equiv \Psi(\lambda g. \lambda x. t)$$

- **Esempi**

$$\text{fact} = \lambda x. \text{if } (\text{eq } x \text{ [0]}) \text{ [1]} (\text{prod } x \text{ (fact (minus } x \text{ [1]))})$$

è una sintassi concreta per il termine

$$\Psi(\lambda \text{fact}. \lambda x. \text{if } (\text{eq } x \text{ [0]}) \text{ [1]} (\text{prod } x \text{ (fact (minus } x \text{ [1]))}))$$

²(possibilmente ricorsiva, i.e. g occorre libera in t) 

- **Recursion-FixedPoint**

- $\Psi \equiv \lambda g. (\lambda x. g(xx)) (\lambda x. g(xx))$

- $\forall h \in \mathcal{F}, \forall d \in \mathcal{D},$

$$\Psi(h)(d) = h(\Psi(h))(d) \quad (\text{vedi esercizio L5.7})$$

- **Esempi**

`fact = $\lambda x. \text{if (eq x [0]) [1] (prod x (fact (minus x [1])))$`

è una sintassi concreta per il termine

`$\Psi(\lambda \text{fact}. \lambda x. \text{if (eq x [0]) [1] (prod x (fact (minus x [1]))))$`

Calcoliamo: `fact[0]` ovvero: $\Psi(h)[0]$, dove $h \equiv \lambda \text{fact}. \lambda x. \text{if}..$

$$\Psi(h)[0] = h(\Psi(h))[0] \rightarrow_{\beta} (\lambda x. \text{if}..)[0] \rightarrow_{\beta}$$

$$\rightarrow_{\beta} \text{if (eq [0] [0]) [1] (prod x ((\Psi(h)) (minus x [1]))))$$

$$\rightarrow_{\beta} \dots \rightarrow_{\beta} [1] \quad (\text{vedi. esercizio L5.10})$$

- 1 (a) Si fornisca una grammatica per la sintassi astratta del λ -Calcolo in accordo alla notazione utilizzata nell'albero astratto $[\lambda - ([x], [\lambda - ([y], [@ - ([x], [y])])])]$. (b) Si mostri poi, che tale albero è ottenuto dalla grammatica data.
- 2 (a) Si fornisca una grammatica per la sintassi concreta del λ -Calcolo in accordo alle proprietà date per associatività e precedenza tra operatori. (b) Si mostri poi, la sintassi concreta dell'albero astratto $[\lambda - ([x], [\lambda - ([y], [@ - ([x], [y])])])]$. (c) Si mostri, infine il parse tree del termine ottenuto al punto (b) precedente.
- 3 Si completino i calcoli indicati con '...' nelle slides precedenti.
- 4 Si mostri la sequenza di α -red applicate per ridurre il termine dato sotto, ad un termine contenente sempre, identificatori diversi per variabili diverse:
$$\lambda x. \lambda y. (\lambda x. y (\lambda y. x) x) (\lambda y. x (\lambda x. y x) y)$$
- 5 (a) Si mostri la valutazione di $\lambda x. \lambda y. (\lambda x. y (\lambda y. x) x) (\lambda y. x (\lambda x. y x) y)$. (b) Nell'ipotesi di aver usato α -red nella valutazione fornita se ne giustifichi l'uso.
- 6 Si mostri la valutazione di $(\lambda x. (\lambda y. \lambda x. x y) (\lambda y. y x)) y$
- 7 Si dimostri che: $\forall F, \Psi F = F(\Psi F)$
- 8 (a) Si scriva, in Lambda-Calcolo, un programma per la funzione plus introdotta nell'aritmetica data per tale linguaggio. (b) Si mostri la computazione di `plus[2][1]`
- 9 Si scriva, in Lambda-Calcolo, un programma per la funzione zerop che calcola true quando applicata a [0], false altrimenti. (b) Si mostri la computazione di `zerop[2]`.

Calcolo di Funzioni: Logica Combinatoria (Schonfinkel 1924)

- **Sintassi** (ovviamente, astratta)

$$\mathcal{C} = X \mid \Pi \mid S \mid K \mid I \mid \mathcal{C}\mathcal{C}$$

- **Termini:** \mathcal{C}, r, r_1 . Insieme (numerabile) dei termini.
 - **Variabili:** X, x, y, z . Insieme (numerabile) di simboli detti variabili libere.
 - **Costanti:** Π, c, \dots . Insieme (numerabile) di simboli detti costanti (distinguibili: $X \cap \Pi = \{\}$)
 - **S, K, I** Sono tre identificatori, i.e. $S, K, I \in \Pi$, detti combinatori fondamentali
 - **op. Applicazione.** $r_1 r_2$ termine ottenuto per *applicazione* del termine r_1 al termine r_2 . Il termine è anche detto, applicazione di r_1 a r_2 .
- **Esempi**
 $S(KX)I$
 $S(S(KS)(S(KK)I))(KI)$

$$\mathcal{C} = X \mid \Pi \mid S \mid K \mid I \mid \mathcal{CC}$$

- **Sintassi Concreta e Variabili Libere**

- **op. Applicazione.** è associativa a sinistra:

$x y x$ significa $((x y) x)$

- **Variabile Libera.** Le variabili che occorrono in un termine sono tutte libere.

- **Esempi**

$S(Kx)I$

$S(S(K+)I)(K5)$

$$\mathcal{C} = X \mid \Pi \mid S \mid K \mid I \mid \mathcal{CC}$$

• Semantica

- Una relazione \rightarrow definita da 1 regola per combinatore ed estesa in una congruenza su \mathcal{C} ..
 - $S \ r_1 \ r_2 \ r \rightarrow r_1 \ r \ (r_2 \ r)$
 - $K \ r_1 \ r_2 \rightarrow r_1$
 - $I \ r \rightarrow r$

• Esempi

$S(Kx)I \ 3 \rightarrow \dots$

$S(S(K+)I)(K \ 5)7 \rightarrow \dots$

$S(S(KS)(S(KK)I))(KI)3 \ 5 \rightarrow \dots$

$$\mathcal{C} = \mathbf{X} \mid \Pi \mid \mathbf{S} \mid \mathbf{K} \mid \mathbf{I} \mid \mathcal{CC}$$

- Sia \mathcal{U} un generico calcolo con termini F per esprimere funzioni e valori, variabili X per e. parametri ³, applicazione di termini per e. l'applicazione di funzione.⁴

Definition (Astratto, Proprietà di Astrazione)

L'astratto di un termine $u \in F$, rispetto ad una variabile $x \in X$, è un termine che indichiamo con $\{x\}u$ avente la seguente proprietà di astrazione:

$$\forall w \in F, (\{x\}u)w = u[w/x]$$

dove $(\{x\}u)w$ è l'applicazione di $\{x\}u$ a w , $u[w/x]$ è la sostituzione, in u , di x con w .

- Nel Lambda-Calcolo, $\{x\}u$ è il termine $\lambda x.u$ e fa parte di una classe speciale di termini.
- \mathcal{C} non ha una classe speciale di termini.

Proposition (Astratto in \mathcal{C})

\mathcal{C} esprime l'astratto mediante una composizione di applicazioni dei suoi termini, ovvero: $\forall r \in \mathcal{C}, \forall x \in X, \{x\}r \in \mathcal{C}$

³ si assuma per semplicità, che variabili diverse abbiano sempre nomi diversi in uno stesso termine

⁴ il calcolo può contenere ulteriori operazioni e strutture

$$\mathcal{C} = X \mid \Pi \mid S \mid K \mid I \mid \mathcal{CC}$$

Proposition (Astratto in \mathcal{C})

\mathcal{C} esprime l'astratto mediante una composizione di applicazioni dei suoi termini, ovvero: $\forall r \in \mathcal{C}, \forall x \in X, \{x\}r \in \mathcal{C}$

- **Astrazione.** Dato un qualunque termine r e una qualunque variabile x , mostriamo un metodo ⁵ per ottenere $\{x\}r \in \mathcal{C}$.
 - $\{x\}x = I$
 - $\{x\}r = K r$ se $x \notin \text{Var}(r)$
 - $\{x\}(r_1 r_2) = S(\{x\}r_1)(\{x\}r_2)$
- **Esempi**
 $\{x\}(\{y\}x) = \dots$

⁵Si noti che $\{x\}r$ può essere espresso con termini equivalenti ma diversi, pertanto esistono altri metodi per calcolare per l'astratto in \mathcal{C}

$$\mathcal{C} = \mathbf{X} \mid \Pi \mid \mathbf{S} \mid \mathbf{K} \mid \mathbf{I} \mid \mathcal{CC}$$

- **Programmi** Sono tutti i termini chiusi, i.e. senza variabili

$$\mathcal{P} = \{r \in \mathcal{C} \mid \text{Var}(r) = \{\}\}$$

- **Conditional e booleani**

- $\text{if} \equiv \{\mathbf{b}\}(\{\mathbf{x}\}(\{\mathbf{y}\}\mathbf{bxy}))$
- $\text{true} \equiv \mathbf{S}(\mathbf{K}\mathbf{K})\mathbf{I}$
- $\text{false} \equiv \mathbf{K}\mathbf{I}$
- and, or, eq, ...

- **Coppie e liste**

- $\text{pair} \equiv \{\mathbf{x}\}(\{\mathbf{y}\}(\{\mathbf{s}\}\mathbf{sxy}))$
- $\text{first } r \equiv r \text{ true}$
- $\text{snd } r \equiv r \text{ false}$

$$\mathcal{C} = X \mid \Pi \mid S \mid K \mid I \mid \mathcal{CC}$$

- **Programmi** Sono tutti i termini chiusi, i.e. senza variabili.
- **Conditional e booleani**
- **Coppie e liste**
- **Aritmetica**
 - $[0] \equiv \text{pair true false}$
 - $[n + 1] \equiv \text{pair false } [n]$
 - $\text{zerop } r \equiv r \text{ true}$
 - $\text{succ, plus, prod, minus, div}$
- **Recursion-FixedPoint**
 - $\Psi \equiv \dots$

$$\mathcal{C} = X \mid \Pi \mid S \mid K \mid I \mid \mathcal{CC}$$

- **Programmi** Sono tutti i termini chiusi, i.e. senza variabili
- **Aritmetica e Liste**
- **Conditional e booleani**
- **Recursion-FixedPoint**
- **Un monoide** con applicazione come unica operazione
- **Il più compatto** formalismo di calcolo
- **Il più compatto** linguaggio di programmazione
 - esecutore semplice da realizzare (no sostituzione)
 - valori e operazioni espressi nel linguaggio (inclusa ricorsione) possono essere realizzati come primitive della sua Macchina Astratta
- **Espressività:** In grado di esprimere i meccanismi (valori, condizionali, ricorsione,..) di uso comune in programmazione.

Macchina a Stato (Minsky 1956, Wang 1957)

- **Sintassi** (ovviamente, astratta)

$\mathcal{Z} = \text{Zero } R \mid \text{Inc } R \mid \text{DJ0 } R \ N \mid \text{Halt}$

- **Termini:** \mathcal{Z} Insieme (numerabile) degli Statements, z, z_i .
 - **Azzeramento** registro R ;
 - **Incremento** registro R ;
 - **Decremento o salto** a N se R contiene 0;
 - **Arresto** esecuzione;
- **Registri:** R_i . Insieme finito di Registri a *capacità infinita*.
- **N:** interi nonnegativi
- **Programmi, \mathcal{P} :**
 - Ogni sequenza finita di \mathcal{Z} ;
 - Ogni statement accessibile attraverso la propria posizione nella sequenza

- **Esempi**

Zero R_0 ; Halt

Zero R_0 ; DJ0 R_1 4; Inc R_2 ; DJ0 R_0 1; Halt

$\mathcal{Z} = \text{Zero R} \mid \text{Inc R} \mid \text{DJO RN} \mid \text{Halt}$

- Semantica SOS

- Definizione di **stato**

- Registri (variabili con nomi predefiniti) rappresentati da una sequenza di coppie, ρ , della forma: $(R_0, n_0) \dots (R_k, n_k)$
 - Program Counter: Posizione PC del successivo statement da eseguire (inizialmente 0).
 - Uno stato è rappresentato da una coppia $\{\text{PC}, \rho\}$

- Definizione di **Transizione** \rightarrow

- Esecuzione di Statement z nello stato σ della macchina
 - $\langle z, \sigma \rangle \rightarrow \sigma'$, indicante ...
 - $\langle z, \sigma \rangle \rightarrow \langle z', \sigma' \rangle$, indicante ...

$\mathcal{Z} = \text{Zero R} \mid \text{Inc R} \mid \text{DJ0 R N} \mid \text{Halt}$

- Semantica SOS

- Definizione di **stato**
- Definizione di **Transizione**

- Definizione di **Computazione** di un programma di $p \in \mathcal{P}$:

- sequenza componente registro degli stati attraversati nell'esecuzione di p .

$$\langle z, \{n, \rho\} \rangle \rightarrow \langle z_1, \{n_1, \rho_1\} \rangle \rightarrow \langle z_2, \{n_2, \rho_2\} \rangle \rightarrow \dots$$

- La semantica di un programma è la sua computazione:

$$\text{Sem}(p) \equiv \rho, \rho_1, \rho_2, \dots,$$

- **Notazione.**

- $\{n, (R_1, n_1), \dots, (R_k, n_k)\}$ stato con PC $\equiv n$ e k registri
- ρ, τ (anche con pedici) sono componente registro dello stato
- $\rho(R_i) = 0$ quando $\rho = (R_1, n_1), \dots, (R_i, 0), \dots, (R_k, n_k)$,
- $\rho(R_i) \neq 0$ quando $\rho = (R_1, n_1), \dots, (R_i, n_i \neq 0), \dots, (R_k, n_k)$,
- $\rho[R_i \leftarrow 0] = (R_1, n_1), \dots, (R_i, 0), \dots, (R_k, n_k)$
- $\rho[R_i \leftarrow R_i + 1] = (R_1, n_1), \dots, (R_i, n_i + 1), \dots, (R_k, n_k)$
quando $\rho = (R_1, n_1), \dots, (R_i, n_i), \dots, (R_k, n_k)$
- $P[n]$ n -esimo statement del programma P
- $\#P$ numero di statements del programma (posizionati da 0 a $\#P - 1$)

$\mathcal{Z} = \text{Zero R} \mid \text{Inc R} \mid \text{DJO R N} \mid \text{Halt}$

$$\langle \text{Halt}, \{n, \rho\} \rangle \rightarrow \{n, \rho\}$$

$$\frac{0 \leq n < \#P, \quad z_n = P[n]}{\langle \text{Zero R}_i, \{n, \rho\} \rangle \rightarrow \langle z_n, \{n+1, \rho[R_i \leftarrow 0]\} \rangle}$$

$$\frac{0 \leq n < \#P, \quad z_n = P[n]}{\langle \text{Inc R}_i, \{n, \rho\} \rangle \rightarrow \langle z_n, \{n+1, \rho[R_i \leftarrow R_i + 1]\} \rangle}$$

$$\frac{0 \leq m < \#P, \quad z_m = P[m], \quad \rho(R_i) = 0}{\langle \text{DJO R}_i m, \{n, \rho\} \rangle \rightarrow \langle z_m, \{m+1, \rho\} \rangle} \quad \frac{0 \leq n < \#P, \quad z_n = P[n], \quad \rho(R_i) \neq 0}{\langle \text{DJO R}_i m, \{n, \rho\} \rangle \rightarrow \langle z_n, \{n+1, \rho[R_i \leftarrow R_i - 1]\} \rangle}$$

- 1 Si mostri la computazione di $\langle P, \{0, \rho_0\} \rangle$, dove $\rho_0 \equiv \{0, (R_0, 5), (R_1, 3), (R_2, 4)\}$
 $P \equiv \text{Zero } R_0; \text{DJ0 } R_1 \ 4; \text{Inc } R_2; \text{DJ0 } R_0 \ 1; \text{Halt}$. Si assuma il primo statement di P alla posizione 0.
- 2 Si mostri la computazione del programma P sopra, nello stato $\{0, \rho\}$, dove ρ_0 sia un'arbitraria configurazione dei registri R_0, R_1, R_2 .
- 3 Si scriva un programma che calcolato in uno stato iniziale arbitrario ρ_0 termini avendo copiato nel registro R_1 il contenuto del registro R_2 .
- 4 Lo stesso esercizio precedente salvo che nello stato finale ρ_k deve valere:
$$\rho_k(R_1) = \rho_0(R_1) = \rho_k(R_2).$$
- 5 Si scriva un programma che calcolato in uno stato iniziale ρ_0 termini scrivendo 0 nel registro R_0 se $\rho_0(R_1) \neq \rho_0(R_2)$. Il programma termina scrivendo 1, in caso contrario.
- 6 Si mostri la computazione del programma P sotto, nello stato $\{0, \rho_0\}$, dove ρ_0 sia un arbitraria configurazione dei registri R_0, R_1, R_2 .
$$P \equiv \text{Zero } R_0; \text{Inc } R_1; \text{DJ0 } R_0 \ 1; \text{Halt}$$
.
- 7 Si scriva un programma che calcolato in uno stato iniziale ρ_0 termini in uno stato ρ_k tale che $\rho_k(R_2) = \rho_0(R_2) - \rho_0(R_1)$.
- 8 Cosa implica l'assunzione che i registri della Macchina a Stato sebbene in numero finito abbiano capacità illimitata?

● Calcolo di Processi, Distribuito

Definisce *grafi* con *nodi* come *processi* (= *agenti* = *produttori di azioni*) che comunicano esclusivamente attraverso *canali* rappresentati come *archi*.

Le azioni possono: 1) comunicare (segnali, valori, ...); 2) avere *effetti laterali* (i.e. modificare lo stato [memoria condivisa e/o privata, sensori, attuatori,...]); 3) produrre nuovi processi (figli), nuovi canali (per questi), riconfigurare canali di comunicazione, generando *raffinamenti* ed anche *nuove strutture* di grafo.

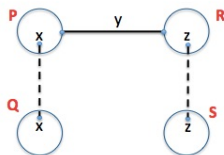


Fig. 5.1 Un sistema con 4 processi, 1 canale pubblico, con nome y , 2 canali privati, di nome x e z , per la comunicazione protetta tra P e Q, e R ed S rispettivamente.

... Concorrente

Processi *autonomi* che sono in esecuzione nello stesso istante.
producendo azioni *atomiche/non-atomiche*, *sincrone/asincrone*

- ... **Mobile**

I Processi comunicano attraverso *canali* che possono essere *riconfigurati dinamicamente, attraverso opportune comunicazioni tra i processi* interessati.

1) nuovi canali possono affiancarsi a e/o sostituire vecchi; e soprattutto, 2) vecchi canali possono essere ri-utilizzati per nuove comunicazioni con differenti e/o nuovi processi.

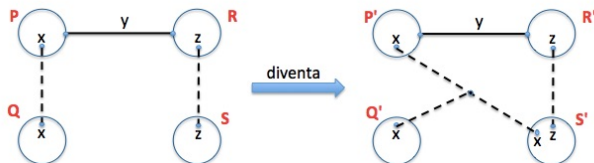


Fig. 5.2 Se il sistema permette mobilità allora il nome x del canale che a sinistra è privato (per comunicazione protetta tra P e Q) può essere comunicato ad S , conducendo al sistema a destra. Il nuovo sistema permette di avviare comunicazione protetta tra P , Q ed S

- **Modelli di Calcolo di Processi,...**

Sono formalismi (eseguibili o no) per definire il comportamento di un sistema (*calcolabile*) di processi allo scopo di:

- 1) Studiarne e Provare proprietà
(*Protocollo di comunicazione* = come avviene la comunicazione, regole;
Sistema di Processi = cosa calcola il sistema, quali valori si scambiano)
- 2) Progettare linguaggi per programmare tali sistemi
(rendendoli eseguibili ovvero attivi, concreti, effettivi, utilizzabili,...)

- **Esempi di Uso**

- **Quesito1.** Fornire una definizione del sistema mostrato in Fig.5.1;
- **Quesito2.** In accordo alla definizione data in (risposta al) Q.1, si può affermare che il sistema è stato ottenuto come evoluzione di un sistema formato inizialmente da un solo processo?
- **Quesito 3.** Fornire una definizione di un sistema che evolve in un sistema come quello mostrato in Fig.5.2;
- **Quesito 4.** Studiare la terminazione del sistema fornito in risposta al Q.1;

Calcolo di Processi, Distribuito, Concorrente, Mobile: π -Calcolo (Milner 1992)

- **Sintassi** (ovviamente, astratta)

$$\pi = \pi + \pi \mid \pi \parallel \pi \mid (X)\pi \mid \alpha.\pi \mid 0 \mid [X = X]\pi \mid A(\vec{X})$$

$$\alpha = \bar{X}X \mid X(X) \mid \tau$$

- **Processi/Agenti:** π, P, Q, P_i, Q_i . Insieme (numerabile) delle espressioni per sistemi di processi.
- **Nomi:** X, x, y, x_i, y_i, \dots . Insieme (numerabile) di identificatori per canali di comunicazione e/o dei valori atomici comunicati.
- **Prefissi:** α . Espressioni di comunicazione.
- **Agente Definito.** $A(\vec{X})$, dove $\vec{X} \equiv x_1, \dots, x_n$, con $x_i \in X$ ed $n \geq 0$, ed un'unica (anche **ricorsiva**) equazione della forma $A(y_1, \dots, y_n) \stackrel{\text{def}}{=} P$ sia data per il simbolo ausiliario $A \notin X$.
- **Programmi:** Ogni espressione π *chiusa*.

- **Esempi**

$\text{in}(c).c(n).c(m).c(r).\bar{r}n.0$ -- espressione π *non chiusa*

$(c)\text{in}(c).c(n).c(m).c(r).\bar{r}m.0$ -- un programma

$$\begin{aligned}\pi &= \pi + \pi \mid \pi \parallel \pi \mid (X)\pi \mid \alpha.\pi \mid 0 \mid [X = X]\pi \mid A(\vec{X}) \\ \alpha &= \bar{X}X \mid X(X) \mid \tau\end{aligned}$$

- **Sum:** $P_1 + P_2$. Processo *nondeterministico*.
- **Composition:** $P_1 \parallel P_2$. Sistema di processi in *parallelo*.
- **Restriction:** $(x)P$. Nome x è dichiarato locale in P .
- **Prefix Form:** $\alpha.P$. Comunicazione con l'esterno.
- **Inaction:** 0 . Processo *neutro*.
- **Match:** $[x = y]P$. Processo *condizionato*.
- **Defined:** $A(x_1, \dots, x_n)$. Uso di definizione ausiliaria.

Esempi

$\text{fst}(\text{in}) \stackrel{\text{def}}{=} \text{in}(c).c(n).c(m).c(r).\bar{r}n.0$ -- una definizione ausiliaria
 $(c)(\text{fst}(c) \parallel \text{snd}(c) \parallel P(c))$ -- un programma (sopra le sue ausiliarie)
 $x(c).(\text{fst}(c) \parallel \text{snd}(c) \parallel P(c)).0$ -- un sistema di processi che forma un processo

$$\begin{aligned}\pi &= \pi + \pi \mid \pi \parallel \pi \mid (\mathbf{X})\pi \mid \alpha.\pi \mid 0 \mid [\mathbf{X} = \mathbf{X}]\pi \mid \mathbf{A}(\vec{\mathbf{X}}) \\ \alpha &= \bar{\mathbf{X}}\mathbf{X} \mid \mathbf{X}(\mathbf{X}) \mid \tau\end{aligned}$$

• Sintassi Concreta: Convenzioni

- **Sum, Composition:** associatività left/right, e commutatività.
 $P_1 + \dots + P_n$ sta per ogni permutazione e raggruppamento dei suoi termini
- **Precedenza.** tra operatori:

$$\left. \begin{array}{l} \text{Restriction} \\ \text{Prefix} \\ \text{Match} \end{array} \right\} > \text{Composition} > \text{Sum}$$

- **Parameter Grouping** $(\vec{\mathbf{X}})P, \bar{\mathbf{X}}\vec{\mathbf{X}}, \mathbf{X}(\vec{\mathbf{X}})$

• Esempi

- $P_1 + P_2 + P_3 + P_4 \parallel Q$ *equivalente* $P_2 + (P_4 \parallel Q) + P_3 + P_1$ *equivalente ...*
- $(p)(u) \bar{o} u p.P \parallel Q + R$ *equivalente* $((p, u) \bar{o} u p.P) \parallel Q + R$ *equivalente...*

$$\begin{aligned}\pi &= \pi + \pi \mid \pi \parallel \pi \mid (X)\pi \mid \alpha.\pi \mid 0 \mid [X = X]\pi \mid A(\vec{X}) \mid \alpha \star \pi \\ \alpha &= \bar{X}X \mid X(X) \mid \tau\end{aligned}$$

- **Nomi: $n(Q)$.** I nomi che occorrono in un termine $P \in \pi$ sono raccolti in $n(P) = \text{bn}(P) \cup \text{fn}(P)$, e si dividono in nomi Legati, $\text{bn}(P)$, e nomi Liberi, $\text{fn}(P)$.

Esempio: Sia $U \equiv \bar{x}y.(z)(\bar{x}z.P + y(w).Q)$.

$$n(U) = \{x, y, z, w\} \cup n(P) \cup n(Q).$$

- **Nomi Legati: $\text{bn}(Q)$.** Introdotti dal binder (y) : $y \in \text{bn}(x(y).P)$, $y \in \text{bn}((y).P)$.
 P è lo *scope* di (y) : Ogni occorrenza libera di y in P diventa legata all'occorrenza y nel binder.

Esempio: Sia $U \equiv \bar{x}y.(z)(\bar{x}z.P + y(w).Q)$.

$$\text{bn}(U) = \{z, w\} \cup \text{bn}(P) \cup \text{bn}(Q).$$

$$\begin{aligned}\pi &= \pi + \pi \mid \pi \parallel \pi \mid (\mathbf{X})\pi \mid \alpha.\pi \mid 0 \mid [\mathbf{X} = \mathbf{X}]\pi \mid \mathbf{A}(\overrightarrow{\mathbf{X}}) \mid \alpha \star \pi \\ \alpha &= \overline{\mathbf{X}}\mathbf{X} \mid \mathbf{X}(\mathbf{X}) \mid \tau\end{aligned}$$

- **Nomi:** $n(\mathbf{Q})$.
- **Nomi Legati:** $bn(\mathbf{Q})$.
- **Nomi Liberi:** $fn(\mathbf{Q})$. Tutti i nomi in \mathbf{Q} che non sono nello scope di un binder con tale nome.

Esempio: Sia $U \equiv \overline{xy}.(z)(\overline{xz}.P + y(w).Q)$.

$$fn(U) = \{x, y\} \cup (fn(P) \setminus \{z\}) \cup (fn(Q) \setminus \{z, w\}).$$

- **Occorrenze.** Un identificatore legato più volte può condurre ad occorrenze indicanti differenti nomi legati.

Esempio: Sia $U \equiv \overline{xy}.(z)(\overline{xz}.P + y(z).Q)$.

Le occorrenze di z libere in Q e quelle libere in P sono nello scope di differenti binder in U e sono quindi nomi differenti.

π -Calcolo: Definizioni di $n(Q)$, $bn(Q)$, $fn(Q)$

$$\pi = \pi + \pi \mid \pi \parallel \pi \mid (X)\pi \mid \alpha.\pi \mid 0 \mid [X = X]\pi \mid A(\vec{X})$$

$$\alpha = \bar{X}X \mid X(X) \mid \tau$$

$Q \in \pi$	$bn(Q)$	$fn(Q)$
$P_1 + P_2$	$bn(P_1) \cup bn(P_2)$	$fn(P_1) \cup fn(P_2)$
$P_1 \parallel P_2$	$bn(P_1) \cup bn(P_2)$	$fn(P_1) \cup fn(P_2)$
$(y)P$	$\{y\} \cup bn(P)$	$fn(P) \setminus \{y\}$
$\bar{x}y.P$	$bn(P)$	$\{x, y\} \cup fn(P)$
$x(y).P$	$\{y\} \cup bn(P)$	$\{x\} \cup (fn(P) \setminus \{y\})$
$\tau.P$	$bn(P)$	$fn(P)$
0	$\{\}$	$\{\}$
$[x = y].P$	$bn(P)$	$\{x, y\} \cup fn(P)$
$A(x_1, \dots, x_n)$	$\{\}$	$\{x_1, \dots, x_n\}$
$\alpha * P$	$bn(\alpha.P)$	$fn(\alpha.P)$

$$n(P) = fn(P) \cup bn(P)$$

Esempi

Applichiamo le regole per calcolare:

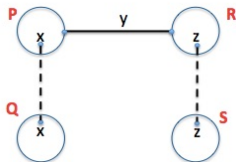
$$\begin{aligned} &bn(\bar{x}y.(z)(\bar{x}z.P + y(w).Q)) \\ &= bn((z)(\bar{x}z.P + y(w).Q)) \\ &= \{z\} \cup bn(\bar{x}z.P + y(w).Q) \\ &= \{z\} \cup bn(\bar{x}z.P) \cup bn(y(w).Q) \\ &= \{z\} \cup bn(P) \cup bn(y(w).Q) \\ &= \{z, w\} \cup bn(P) \cup bn(Q) \end{aligned}$$

π -Calcolo: Free, bound ed Espressioni chiuse – Uso

$$\begin{aligned}\pi &= \pi + \pi \mid \pi \parallel \pi \mid (\mathbf{X})\pi \mid \alpha.\pi \mid 0 \mid [\mathbf{X} = \mathbf{X}]\pi \mid \mathbf{A}(\overrightarrow{\mathbf{X}}) \\ \alpha &= \overline{\mathbf{X}}\mathbf{X} \mid \mathbf{X}(\mathbf{X}) \mid \tau\end{aligned}$$

- **Binders e Nomi Legati.** I Binder sono utilizzati in un processo per introdurre nuovi nomi/canali:
 - + interni attraverso cui i (sotto-)processi di cui è costituito potranno comunicare tra loro;
 - + esterni da trasmettere per rinnovare (*mobilità*) i canali di comunicazione esterna.
- **Nomi Liberi.** Rappresentano la conoscenza che un processo deve avere del contesto in cui opera.

Esempi: Una soluzione al Quesito1 (vedi anche esercizio...).



$\mathbf{P}(\mathbf{y}) \parallel \mathbf{R}(\mathbf{y})$

dove:

$\mathbf{P}(\mathbf{z}) = (\mathbf{n})(\mathbf{z} \mathbf{n}.\mathbf{P}_1 + \mathbf{z}(\mathbf{v}).\mathbf{P}_2 + (\mathbf{x})(\mathbf{P}_3 \mid \mathbf{Q}))$

$\mathbf{Q}() = \dots$

$\mathbf{R}(\mathbf{z}) = \dots$

...

π -Calcolo - Semantica: La relazione $\xrightarrow{\alpha}$ (tra π -termini)

$$\pi = \pi + \pi \mid \pi \parallel \pi \mid (X)\pi \mid \alpha.\pi \mid 0 \mid [X = X]\pi \mid A(\vec{X})$$

$$\alpha = \bar{X}X \mid X(X) \mid \tau$$

input: $\frac{w \notin \text{fn}((y)P)}{x(y).P \xrightarrow{x(w)} P\{w/y\}}$		output: $\frac{}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$	tau: $\frac{}{\tau.P \xrightarrow{\tau} P}$
res: $\frac{P \xrightarrow{\alpha} P' \quad x \notin \text{n}(\alpha)}{(x)P \xrightarrow{\alpha} (x)P'}$	open: $\frac{P \xrightarrow{\bar{x}y} P' \quad x \neq y \quad w \notin \text{fn}((y)P)}{(y)P \xrightarrow{\bar{x}(w)} P'\{w/y\}}$		
sum: $\frac{P_1 \xrightarrow{\alpha} P'_1}{P_1 + P_2 \xrightarrow{\alpha} P'_1}$	match: $\frac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'}$		
def: $\frac{P\{\vec{Y}/\vec{X}\} \xrightarrow{\alpha} P' \quad A(\vec{X}) \stackrel{\text{def}}{=} P}{A(\vec{Y}) \xrightarrow{\alpha} P'}$			
par: $\frac{P_1 \xrightarrow{\alpha} P'_1 \quad \text{bn}(\alpha) \cap \text{fn}(P_2) = \{\}}{P_1 \parallel P_2 \xrightarrow{\alpha} P'_1 \parallel P_2}$	com: $\frac{P_1 \xrightarrow{\bar{x}y} P'_1 \quad P_2 \xrightarrow{x(y)} P'_2}{P_1 \parallel P_2 \xrightarrow{\tau} P'_1 \parallel P'_2\{w/y\}}$	close: $\frac{P_1 \xrightarrow{\bar{x}(y)} P'_1 \quad P_2 \xrightarrow{x(y)} P'_2}{P_1 \parallel P_2 \xrightarrow{\tau} (y)(P'_1 \parallel P'_2)}$	

Osservazioni.

- Associatività+Commutatività: Le regole *sum*, *par*, *com* e *close* hanno la variante in cui i ruoli delle espressioni P_1 e P_2 sono tra loro scambiati (ved. esercizio...)

Notazione.

- $P\{w/y\}$ rimpiazzamento di occorrenze libere di y con w (vedi esercizio ...)

π -Calcolo - Applichiamo la riduzione $\xrightarrow{\alpha}$ / 1

input: $\frac{w \notin \text{fn}((y)P)}{x(y).P \xrightarrow{x(y)} P\{w/y\}}$		output: $\frac{}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$	tau: $\frac{}{\tau.P \xrightarrow{\tau} P}$
res: $\frac{P \xrightarrow{\alpha} P' \quad x \notin \text{n}(\alpha)}{(x)P \xrightarrow{\alpha} (x)P'}$	open: $\frac{P \xrightarrow{\bar{x}y} P' \quad x \neq y \quad w \notin \text{fn}((y)P)}{(y)P \xrightarrow{\bar{x}(w)} P'\{w/y\}}$		
sum: $\frac{P_1 \xrightarrow{\alpha} P'_1}{P_1 + P_2 \xrightarrow{\alpha} P'_1}$	match: $\frac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'}$		
def: $\frac{P\{\bar{Y}/\bar{X}\} \xrightarrow{\alpha} P' \quad A(\bar{X}) \stackrel{\text{def}}{=} P}{A(\bar{Y}) \xrightarrow{\alpha} P'}$			
par: $\frac{P_1 \xrightarrow{\alpha} P'_1 \quad \text{bn}(\alpha) \cap \text{fn}(P_2) = \{\}}{P_1 \parallel P_2 \xrightarrow{\alpha} P'_1 \parallel P_2}$	com: $\frac{P_1 \xrightarrow{\bar{x}w} P'_1 \quad P_2 \xrightarrow{x(y)} P'_2}{P_1 \parallel P_2 \xrightarrow{\tau} P'_1 \parallel P'_2\{w/y\}}$	close: $\frac{P_1 \xrightarrow{\bar{x}(y)} P'_1 \quad P_2 \xrightarrow{x(y)} P'_2}{P_1 \parallel P_2 \xrightarrow{\tau} (y)(P'_1 \parallel P'_2)}$	


Esercizi. Applichiamo le regole al seguente sistema:

$(z)\bar{y}z.P \parallel y(x).Q$ – dove notiamo che S sta per $(z)(\bar{y}z.P) \parallel (y(x).Q)$ (vedi esercizio ...)

Step 1: Renaming

$$R(\text{close})^6 \frac{(z)\bar{y}z.P \parallel y(x).Q \xrightarrow{\tau} (w)(P' \parallel Q')}{(z)\bar{y}z.P \xrightarrow{\bar{y}(w)} P' \quad y(x).Q \xrightarrow{y(w)} Q'}$$

– continua

⁶R(c) si legge reversed-c ed indica l'uso invertito della regola c: Dalle conclusioni alle sufficienti premesse.  36/60

π -Calcolo - Applichiamo la riduzione $\xrightarrow{\alpha}$ / 2

input: $\frac{w \notin \text{fn}((y)P)}{x(y).P \xrightarrow{x(w)} P\{w/y\}}$		output: $\frac{}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$	tau: $\frac{}{\tau.P \xrightarrow{\tau} P}$
res: $\frac{P \xrightarrow{\alpha} P' \quad x \notin \text{n}(\alpha)}{(x)P \xrightarrow{\alpha} (x)P'}$	open: $\frac{P \xrightarrow{\bar{x}y} P' \quad x \neq y \quad w \notin \text{fn}((y)P)}{(y)P \xrightarrow{\bar{x}(w)} P'\{w/y\}}$		
sum: $\frac{P_1 \xrightarrow{\alpha} P'_1}{P_1 + P_2 \xrightarrow{\alpha} P'_1}$	match: $\frac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'}$		
def: $\frac{P\{\bar{Y}/\bar{X}\} \xrightarrow{\alpha} P' \quad A(\bar{X}) \stackrel{\text{def}}{=} P}{A(\bar{Y}) \xrightarrow{\alpha} P'}$			
par: $\frac{P_1 \xrightarrow{\alpha} P'_1 \quad \text{bn}(\alpha) \cap \text{fn}(P_2) = \{\}}{P_1 \parallel P_2 \xrightarrow{\alpha} P'_1 \parallel P_2}$	com: $\frac{P_1 \xrightarrow{\bar{x}w} P'_1 \quad P_2 \xrightarrow{x(y)} P'_2}{P_1 \parallel P_2 \xrightarrow{\tau} P'_1 \parallel P'_2\{w/y\}}$	close: $\frac{P_1 \xrightarrow{\bar{x}(y)} P'_1 \quad P_2 \xrightarrow{x(y)} P'_2}{P_1 \parallel P_2 \xrightarrow{\tau} (y)(P'_1 \parallel P'_2)}$	

Esercizi. Applichiamo le regole al seguente sistema:

$(z)\bar{y}z.P \parallel y(x).Q$ – dove notiamo che S sta per $(z)(\bar{y}z.P) \parallel (y(x).Q)$ (vedi esercizio ...)

Step 1: renaming

Step 2: open

$$R(\text{close}) \frac{(z)\bar{y}z.P \parallel y(x).Q \xrightarrow{\tau} (w)(P' \parallel Q')}{(z)\bar{y}z.P \xrightarrow{\bar{y}(w)} P' \equiv P''\{w/z\}} \quad y(x).Q \xrightarrow{y(w)} Q'$$

$$R(\text{open}) \frac{}{\bar{y}z.P \xrightarrow{\bar{y}z} P'' \quad y \neq z \quad w \notin \text{fn}((z)P)}$$

– continua

π -Calcolo - Applichiamo la riduzione $\xrightarrow{\alpha}$ / 3

input: $\frac{w \notin \text{fn}((y)P)}{x(y).P \xrightarrow{x(w)} P\{w/y\}}$		output: $\frac{}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$	tau: $\frac{}{\tau.P \xrightarrow{\tau} P}$
res: $\frac{P \xrightarrow{\alpha} P'}{(x)P \xrightarrow{\alpha} (x)P'}$	$x \notin \text{n}(\alpha)$	open: $\frac{P \xrightarrow{\bar{x}y} P' \quad x \neq y \quad w \notin \text{fn}((y)P)}{(y)P \xrightarrow{\bar{x}(w)} P'\{w/y\}}$	
sum: $\frac{P_1 \xrightarrow{\alpha} P'_1}{P_1 + P_2 \xrightarrow{\alpha} P'_1}$		match: $\frac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'}$	
def: $\frac{P\{\bar{Y}/\bar{X}\} \xrightarrow{\alpha} P' \quad A(\bar{X}) \stackrel{\text{def}}{=} P}{A(\bar{Y}) \xrightarrow{\alpha} P'}$			
par: $\frac{P_1 \xrightarrow{\alpha} P'_1 \quad \text{bn}(\alpha) \cap \text{fn}(P_2) = \{\}}{P_1 \parallel P_2 \xrightarrow{\alpha} P'_1 \parallel P_2}$	com: $\frac{P_1 \xrightarrow{\bar{x}w} P'_1 \quad P_2 \xrightarrow{x(y)} P'_2}{P_1 \parallel P_2 \xrightarrow{\tau} P'_1 \parallel P'_2\{w/y\}}$	close: $\frac{P_1 \xrightarrow{\bar{x}(y)} P'_1 \quad P_2 \xrightarrow{x(y)} P'_2}{P_1 \parallel P_2 \xrightarrow{\tau} (y)(P'_1 \parallel P'_2)}$	

Esercizi. Applichiamo le regole al seguente sistema: $(z)\bar{y}z.P \parallel y(x).Q$.

Step 1: renaming

Step 2: open

Step 3: Input

$$\begin{array}{l}
 \text{R(close)} \frac{(z)\bar{y}z.P \parallel y(x).Q \xrightarrow{\tau} (w)(P' \parallel Q')}{(z)\bar{y}z.P \xrightarrow{\bar{y}(w)} P' \equiv P''\{w/z\}} \\
 \text{R(open)} \frac{(z)\bar{y}z.P \xrightarrow{\bar{y}z} P'' \quad y \neq z \quad w \notin \text{fn}((z)P)}{(z)\bar{y}z.P \xrightarrow{\bar{y}(w)} P' \equiv P''\{w/z\}} \\
 \text{R(input)} \frac{y(x).Q \xrightarrow{y(w)} Q' \equiv Q\{w/x\} \quad w \notin \text{fn}(x)Q}{(z)\bar{y}z.P \xrightarrow{\bar{y}(w)} P' \equiv P''\{w/z\}}
 \end{array}$$

— Continua

π -Calcolo - Applichiamo la riduzione $\xrightarrow{\alpha}$ / 4

input: $\frac{w \notin \text{fn}((y)P)}{x(y).P \xrightarrow{x(w)} P\{w/y\}}$		output: $\frac{}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$	tau: $\frac{}{\tau.P \xrightarrow{\tau} P}$
res: $\frac{P \xrightarrow{\alpha} P' \quad x \notin \text{n}(\alpha)}{(x)P \xrightarrow{\alpha} (x)P'}$	open: $\frac{P \xrightarrow{\bar{x}y} P' \quad x \neq y \quad w \notin \text{fn}((y)P)}{(y)P \xrightarrow{\bar{x}(w)} P'\{w/y\}}$		
sum: $\frac{P_1 \xrightarrow{\alpha} P'_1}{P_1 + P_2 \xrightarrow{\alpha} P'_1}$	match: $\frac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'}$		
def: $\frac{P\{\bar{Y}/\bar{X}\} \xrightarrow{\alpha} P' \quad A(\bar{X}) \stackrel{\text{def}}{=} P}{A(\bar{Y}) \xrightarrow{\alpha} P'}$			
par: $\frac{P_1 \xrightarrow{\alpha} P'_1 \quad \text{bn}(\alpha) \cap \text{fn}(P_2) = \{\}}{P_1 \parallel P_2 \xrightarrow{\alpha} P'_1 \parallel P_2}$	com: $\frac{P_1 \xrightarrow{\bar{x}w} P'_1 \quad P_2 \xrightarrow{x(y)} P'_2}{P_1 \parallel P_2 \xrightarrow{\tau} P'_1 \parallel P'_2\{w/y\}}$	close: $\frac{P_1 \xrightarrow{\bar{x}(y)} P'_1 \quad P_2 \xrightarrow{x(y)} P'_2}{P_1 \parallel P_2 \xrightarrow{\tau} (y)(P'_1 \parallel P'_2)}$	

Esercizi. Applichiamo le regole al seguente sistema: $(z)\bar{y}z.P \parallel y(x).Q$.

Step 1: renaming

Step 2: open

Step 3: Input

Step 4: Output

$$\begin{array}{l}
 R(\text{close}) \frac{(z)\bar{y}z.P \parallel y(x).Q \xrightarrow{\tau} (w)(P' \parallel Q')}{(z)\bar{y}z.P \parallel y(x).Q} \\
 R(\text{open}) \frac{(z)\bar{y}z.P \xrightarrow{\bar{y}(w)} P' \equiv P''\{w/z\}}{(z)\bar{y}z.P \parallel y(x).Q \xrightarrow{\tau} (w)(P' \parallel Q')} \\
 R(\text{input}) \frac{y(x).Q \xrightarrow{y(w)} Q' \equiv Q\{w/x\}}{(z)\bar{y}z.P \parallel y(x).Q \xrightarrow{\tau} (w)(P' \parallel Q')} \\
 R(\text{output}) \frac{\bar{y}z.P \xrightarrow{\bar{y}z} P'' \equiv P \quad y \neq z \quad w \notin \text{fn}((z)P)}{(z)\bar{y}z.P \parallel y(x).Q \xrightarrow{\tau} (w)(P' \parallel Q')} \quad w \notin \text{fn}((x)Q)
 \end{array}$$

Questo calcolo fornisce la seguente riduzione:

$$(z)\bar{y}z.P \parallel y(x).Q \xrightarrow{\tau} (w)(P\{w/z\} \parallel Q\{w/x\}) \text{ con vincoli: } w \notin \text{fn}((z)P), w \notin \text{fn}((x)Q)$$

Esercizi. Applichiamo le regole al seguente sistema: $(z)\bar{y}z.P \parallel y(x).Q$.

Step 1: renaming

Step 2: open

Step 3: Input

Step 4: Output

$$\begin{array}{c} (z)\bar{y}z.P \parallel y(x).Q \xrightarrow{\tau} (w)(P' \parallel Q') \\ \hline \text{R(close)} \frac{}{} \\ \hline \begin{array}{c} \text{R(open)} \frac{(z)\bar{y}z.P \xrightarrow{\bar{y}(w)} P' \equiv P'\{w/z\}}{} \\ \text{R(input)} \frac{y(x).Q \xrightarrow{y(w)} Q' \equiv Q\{w/x\}}{w \notin \text{fn}((x)Q)} \\ \text{R(output)} \frac{\bar{y}z.P \xrightarrow{\bar{y}z} P'' \equiv P}{y \neq z \quad w \notin \text{fn}((z)P)} \end{array} \end{array}$$

Questo calcolo fornisce la seguente riduzione:

$$(z)\bar{y}z.P \parallel y(x).Q \xrightarrow{\tau} (w)(P\{w/z\} \parallel Q\{w/x\}) \text{ con vincoli: } w \notin \text{fn}((z)P), w \notin \text{fn}((x)Q)$$

- **Pura Comunicazione.** Il π -Calcolo definisce sistemi di Pura Comunicazione
 - **Protocolli di Comunicazione.** In tal modo Il π -Calcolo è adatto a descrivere Protocolli di Comunicazione dove l'interesse è descrivere le regole con cui i processi comunicano piuttosto che il perchè e/o quali valori comunicano.
 - **λ -Calcolo.** Analogamente, il λ -Calcolo descrive Calcoli di Funzione dove l'interesse è descrivere la struttura computazionale (i.e. da quali funzioni è ottenuta) della funzione calcolabile piuttosto che il suo uso e/o i valori manipolati.
- **Alta Espressività.** Il π -Calcolo ha alta espressività ovvero è in grado di descrivere (direttamente⁷) una grande varietà di meccanismi di calcolo e strutture e costrutti linguistici quali dati strutturati, puntatori, sharing, λ -Calcolo, la Logica Combinatoria (vedi [Milner92] pag....).
- **Aggiungere Valori.** Si può procedere in modi diversi:
 - **Espressività - 1.** Come abbiamo fatto nel λ -Calcolo, anche qui esprimiamo nel calcolo stesso i valori che vogliamo aggiungere;
 - **Espressività - 2.** Utilizziamo i λ -termini che esprimono i valori e le operazioni che vogliamo aggiungere e compiliamo il tutto in π -Calcolo;
 - **Estendere.** Estendiamo la struttura del linguaggio, aggiungendo i valori e le operazioni come nuove primitive

⁷senza ricorrere ad una emulazione

π -Calcolo. Estendiamo con Valori: π^N

- **Estendiamo** la struttura del linguaggio, aggiungendo i valori e le operazioni come nuove primitive
- **Valori** sono i naturali N (in notazione decimale) con le quattro operazioni indicate $\oplus, \ominus, \otimes, \oslash$ e il predicato \emptyset (test sullo 0).

$$\pi = \pi + \pi \mid \pi \parallel \pi \mid (X)\pi \mid \alpha.\pi \mid 0 \mid [D = D]\pi \mid A(\vec{X})$$

$$\alpha = \bar{X}E \mid X(X) \mid \tau \quad D = X \mid N \quad E = D \mid D \oplus D \mid D \ominus D \mid D \otimes D \mid D \oslash D$$

input: $\frac{w \notin \text{fn}((y)P)}{x(y).P \xrightarrow{x(y)} P\{w/y\}}$		output: $\frac{}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$	tau: $\frac{}{\tau.P \xrightarrow{\tau} P}$
res: $\frac{P \xrightarrow{\alpha} P' \quad x \notin n(\alpha)}{(x)P \xrightarrow{\alpha} (x)P'}$	open: $\frac{P \xrightarrow{\bar{x}y} P' \quad x \neq y, w \notin \text{fn}((y)P), y \notin N}{(y)P \xrightarrow{\bar{x}(w)} P'\{w/y\}}$		
sum: $\frac{P_1 \xrightarrow{\alpha} P'_1}{P_1 + P_2 \xrightarrow{\alpha} P'_1}$	match: $\frac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'}$		
def: $\frac{P\{\vec{Y}/\vec{X}\} \xrightarrow{\alpha} P' \quad A(\vec{X}) \stackrel{\text{def}}{=} P}{A(\vec{Y}) \xrightarrow{\alpha} P'}$			
par: $\frac{P_1 \xrightarrow{\alpha} P'_1 \quad \text{bn}(\alpha) \cap \text{fn}(P_2) = \{\}}{P_1 \parallel P_2 \xrightarrow{\alpha} P'_1 \parallel P_2}$	com: $\frac{P_1 \xrightarrow{\bar{x}w} P'_1 \quad P_2 \xrightarrow{x(y)} P'_2}{P_1 \parallel P_2 \xrightarrow{\tau} P'_1 \parallel P'_2\{w/y\}}$	close: $\frac{P_1 \xrightarrow{\bar{x}(y)} P'_1 \quad P_2 \xrightarrow{x(y)} P'_2}{P_1 \parallel P_2 \xrightarrow{\tau} (y)(P'_1 \parallel P'_2)}$	
\emptyset -false: $\frac{d \notin X, d \neq 0}{\emptyset d \rightarrow 0}$	\emptyset -true: $\frac{}{\emptyset 0 \rightarrow 1}$	+: $\frac{d_1, d_2 \notin X, d_1 + d_2 = n}{d_1 \oplus d_2 \rightarrow n}$	
\cdot : $\frac{d_1, d_2 \notin X, d_1 \geq d_2, d_1 - d_2 = n}{d_1 \ominus d_2 \rightarrow n}$	\times : $\frac{d_1, d_2 \notin X, d_1 \times d_2 = n}{d_1 \otimes d_2 \rightarrow n}$	div: $\frac{d_1, d_2 \notin X, d_1 \text{ div } d_2 = n}{d_1 \oslash d_2 \rightarrow n}$	

- **Programma** per il calcolo dell'espressione: $E \equiv (3+5) \times 7 \text{ div } 3$.

Soluzione:

- Individuiamo i sotto-termini (non valori):

$$E_1 \equiv (3+5)$$

$$E_2 \equiv E_1 \times 7$$

$$E \equiv E_2 \text{ div } 3$$

- e compiliamo separatamente, introducendo: 1 canale di input per ogni sotto-termini che non sia un valore, e 1 canale di output ove inserire il valore calcolato

$$E1(r_1) \stackrel{\text{def}}{=} \overline{r_1} 3 \oplus 5.0$$

$$E2(x_2, r_2) \stackrel{\text{def}}{=} x_2(n). \overline{r_2} n \otimes 7.0$$

$$E3(x_3, r_3) \stackrel{\text{def}}{=} x_3(n). \overline{r_3} n \oslash 3.0$$

- e componiamo:

$$(x, y)(E1(x) \parallel E2(x, y) \parallel E3(y, z))$$

π^N -Calcolo. Programmare in π^N : Un esercizio di Progr. Sequenziale - La funzione fattoriale

- **Programma** per il calcolo del fattoriale: $f(n) \equiv \text{if } n=0 \text{ then } 1 \text{ else } n \times f(n-1)$.

Soluzione:

- Traduciamo in un processo Defined Agent di nome F ;
- L'agent avrà un canale x per l'input, ed un canale r per l'output:

$$F(x, r) \stackrel{\text{def}}{=} E$$

- Traduciamo l'espressione condizionale in un match sul predicato $\emptyset n$
 $E \equiv x(n).(y)(\bar{y} \emptyset n.0 \parallel y(t).([t = 1]E_1 \parallel [t = 0]E_2))$

Legge il valore input n sul canale x , genera un canale interno y su cui è scritto il valore del test $\emptyset n$ e, in modo concorrente, letto per il match da E_1 e da E_2

- Generiamo l'espressione E_1

$$E_1 \equiv \bar{r} 1.0$$

Corrisponde alla traduzione del ramo then del condizionale: Scrive il valore 1 nel canale di output r e termina

- Generiamo l'espressione E_2

$$E_2 \equiv (r_1)((r_1(m).\bar{r} n \otimes m.0) \parallel (z)(\bar{z} n \ominus 1.0 \parallel F(z, r_1)))$$

Corrisponde alla traduzione del ramo else del condizionale. Introduce un canale interno r_1 utilizzato in scrittura da $(z)(\bar{z} n \ominus 1.0 \parallel F(z, r_1))$, come canale di output di $F(z, r_1)$. Quest'ultimo legge dal canale z dove il processo $\bar{z} n \ominus 1.0$ avrà scritto, prima di terminare, il valore calcolato per il decremento di n .

π^N -Calcolo. Programmare in π^N : Programmazione Sequenziale in π -Calcolo

- Possiamo Esprimere la programmazione sequenziale (e con essa, gli algoritmi sequenziale implementati) in π -Calcolo.

$f(n) \equiv \text{if } n=0 \text{ then } 1 \text{ else } n \times f(n-1).$

può essere espressa in π^N , da:

$F(x, r) \stackrel{\text{def}}{=} x(n).(y)(\bar{y} \emptyset n.0 \parallel y(t).([t = 1]E_1 \parallel [t = 0]E_2))$

- Possiamo Compilare il λ -Calcolo in π -Calcolo in modo estremamente semplice (vedi [Milner92] Example 9, pag. 24-27)
- Possiamo Compilare la Logica Combinatoria in π -Calcolo in modo estremamente semplice (vedi [Milner92] Example 8, pag. 21-24)
- Cosa ci guadagnamo?
 - Migliora la Complessità? - NO: L'algoritmo implementato rimane sequenziale!
 - Migliora l'Efficienza? - Il ridotto costo di alcune operazioni (che potrebbero essere avvantaggiate dalla concorrenza) è compensato dal costo della comunicazione;
 - **Integrazione** di algoritmi sequenziali con concorrenti, in un ambiente di calcolo distribuito: Molti sistemi che operano in rete utilizzano algoritmi di entrambi i tipi.

π^N -Calcolo. Programmare in π^N : Un esercizio di Progr. Distribuita - Il Consenso /1

- **Programma** per il calcolo del Consenso: Una popolazione finita di agenti richiede di concordare su un valore tra un insieme S finito possibile. Assumeremo $S \equiv \{p, n, i\}$. Gli agenti possono comunicare in coppie il valore scelto da ciascuno ed eventualmente cambiare valore in accordo ad un dato protocollo. Assumeremo un protocollo a 3 regole così definito: Se i due agenti hanno scelto
 - (1) stesso valore, mantengono tale valore;
 - (2) uno ha scelto p (per positivo), l'altro n (per negativo), entrambi cambiano valore in i (per indeciso);
 - (3) uno solo ha scelto i , quell'uno cambia valore nel valore scelto dall'altro.

Soluzione:

- Definiamo 1 defined agent $A(s, x)$: L'argomento s è lo stato, l'argomento x indica il canale su cui il processo è comunicante con l'esterno. Lo stato interno indica il valore correntemente scelto dall'agente.

$$A(s, x) \stackrel{\text{def}}{=} (y)((\bar{x}y.y(v).\bar{y}s.E(s, v, x) + (x(w).\bar{w}s.w(v).E(s, v, x)))$$
$$\text{where } E(s, v, x) \stackrel{\text{def}}{=} [v = s].A(s, x) \parallel [s = i]A(v, x)$$
$$\parallel [s = p][v = n]A(i, x) \parallel [s = n][v = p]A(i, x)$$

— continua

π^N -Calcolo. Programmare in π^N : Un esercizio di Progr. Distribuita - Il Consenso /2

- **Programma** per il calcolo del Consenso: ...

Soluzione:

- Definiamo 1 defined agent $A(s, x)$: L'argomento s lo stato interno e l'argomento x indica il canale su cui il processo è comunicante con l'esterno. Lo stato interno indica il valore dall'agente.

$$A(s, x) \stackrel{\text{def}}{=} (y)((\bar{x}y.y(v).\bar{y}s.E(s, v, x) + (x(w).\bar{w}s.w(v).E(s, v, x)))$$
$$\text{where } E(s, v, x) \stackrel{\text{def}}{=} [v = s].A(s, x) \parallel [s = i]A(v, x)$$
$$\parallel [s = p][v = n]A(i, x) \parallel [s = n][v = p]A(i, x)$$

- Definiamo i 3 agenti base $P(x)$, $N(x)$, $I(x)$: L'argomento x indica il canale su cui il processo è comunicante con l'esterno. Lo stato interno indica il valore dall'agente.

$$P(x) \stackrel{\text{def}}{=} A(p, x); \quad N(x) \stackrel{\text{def}}{=} A(n, x); \quad I(x) \stackrel{\text{def}}{=} A(i, x)$$

- Il programma per il Consenso:

$$\underbrace{P(x) \parallel \dots \parallel P(x)}_{k_p} \parallel \underbrace{N(x) \parallel \dots \parallel N(x)}_{k_n} \parallel \underbrace{I(x) \parallel \dots \parallel I(x)}_{k_i}$$

π^N -Calcolo. Programmare in π^N : Un esercizio di Progr. Distribuita - Numero di Comunicazioni /1

- **Programma** per il Calcolo delle Comunicazione tra Processi. Un sistema distribuito richiede di rilevare su un canale pubblico ma dedicato z il numero di comunicazioni tra i processi del sistema indipendentemente dal canale utilizzato (il canale z non si considera). Si discuta come potrebbe essere integrato il codice del sistema scritto in π^N .

Soluzione:

- Definiamo 2 Defined Agents. Il primo che chiameremo `Count`, resta in attesa sul canale z per ogni avviso di fine di comunicazione. Ad ogni ricezione di fine comunicazione, `Count` incrementa un parziale (inizializzato a 0) di 1 unità. Il secondo che chiameremo `End`, è attivato dai processi del sistema originale, al termine di ogni comunicazione, per emettere il relativo avviso.

$$\text{Count}(s) \stackrel{\text{def}}{=} z(m).(x)(x(v).\text{Count}(v) \parallel \bar{x}s \oplus 1.0)$$

$$\text{End}() \stackrel{\text{def}}{=} \bar{z}1^8$$

— continua

⁸L'uso di 1 in `End` è inessenziale ma potrebbe essere quello di un codice indicante la fine di una comunicazione.

π^N -Calcolo. Programmare in π^N : Un esercizio di Progr. Distribuita - Numero di Comunicazioni /2

- **Programma** per il Calcolo delle Comunicazione tra Processi. Un sistema distri-

Soluzione:

- Definiamo 2 Defined Agents. Il primo che chiameremo Count, resta ...

$$\begin{aligned}\text{Count}(s) &\stackrel{\text{def}}{=} z(m).(x)(x(v).\text{Count}(v) \parallel \bar{x}s \oplus 1.0) \\ \text{End}() &\stackrel{\text{def}}{=} \bar{z}\text{end}\end{aligned}$$

- Indichiamo con S il sistema iniziale. Il nuovo sistema diventa:

$$S' \parallel \text{Count}(0)$$

dove:

- S' è S in cui dopo il codice relativo ad ogni comunicazione terminata è inserito $\text{End}()$.
- $\text{Count}(0)$ sta per: $(w)(w(s).\text{Count}(s) \parallel \bar{w}0.0)$.

Esercizio

Si completino i calcoli indicati con '...' nelle slides precedenti.

Soluzione

Esercizio

Si mostri la sequenza di α - red applicate per ridurre il termine dato ad un termine contenente sempre, identificatori diversi per variabili diverse: $\lambda x.\lambda y.(\lambda x.y(\lambda y.x)x)(\lambda y.x(\lambda x.yx)y)$

Soluzione

Esercizio

Si dimostri che: $\forall F, \Psi F = F(\Psi F)$

Soluzione

Esercizio

- *Si scriva, in Lambda-Calcolo, un programma per la funzione `plus` introdotta nell'aritmetica data per tale linguaggio.*
- *Si mostri la computazione di `plus[2][1]`*

Soluzione

Esercizio

Si scriva, in Lambda-Calcolo, un programma per la funzione zero_{op} che calcola il test su zero nell'aritmetica data per tale linguaggio.

Soluzione

Esercizio

Si dia una semantica SOS per la Logica Combinatoria

Soluzione

Esercizio

Si dimostri che $SKK = I$

Soluzione

Esercizio

Si scriva, in Logica Combinatoria, un programma per la funzione plus introdotta nell'aritmetica data per tale linguaggio.

Soluzione

Esercizio

Si scriva, in Logica Combinatoria, un programma per la funzione minus introdotta nell'aritmetica data per tale linguaggio.

Soluzione

Esercizio

In π -Calcolo, il termine $P_1 + \dots + P_n$ equivale ad ogni termine corrispondente ad un albero binario con nodi interni $+$ e frontiera ogni permutazione di P_1, \dots, P_n . Si mostri il significato dell'affermazione, discutendo i seguenti punti:

- 1) Fornire una definizione della corrispondenza π -termine/albero binario, richiamata nell'affermazione;*
- 2) Indicare quali proprietà del π -Calcolo giustificano tale affermazione;*
- 3) Esprimere il numero di tali, diversi, alberi binari;*
- 4) Dire se, e quali, altri operatori del π -Calcolo mostrano analoga proprietà.*

Soluzione