Processi Sequenziali

- D'ora in poi rappresenteremo gli stati $p, q, \ldots$ di un sistema attraverso espressioni algebriche, denotate con $P, Q, \ldots$, che identificano **processi.**

- Tali espressioni conterranno informazioni sia sul **comportamento** che sulla **struttura** di un sistema.

- Nel caso di processi sequenziali, tale informazione indica esplicitamente le azioni che possono essere eseguite.

Seguiremo la seguente notazione:

Id	identificatori di processo	$A, B, \ldots, P, Q, \ldots$
$\mathcal{N}$	nomi di canale	$a, b, c, \ldots$
$\overline{\mathcal{N}}$	co-nomi	$\overline{a}, \overline{b}, \overline{c}, \ldots$
$\mathcal{L}$	etichette (o pulsanti)	$:= \mathcal{N} \cup \overline{\mathcal{N}}$
$\mathcal{A}$	actions	metavariabili $\alpha, \beta, \gamma, \ldots$

Importante: I *nomi di canali* non sono altro che comode abbreviazioni per denotare gli indirizzi dove sono memorizzati i *canali* di comunicazione. (Ad esempio: a può essere visto come un'abbreviazione per un indirizzo a 32 bit associato al canale). A volte scriveremo $\vec{a}$ come abbreviazione per una *sequenza finita di nomi* $a_1, \ldots, a_n$.

Definizione 11 *L'insieme* $\mathcal{P}^{\mathrm{seq}}$ *dei* **processi sequenziali** *è dato dalla seguente sintassi:*

$$P \quad ::= \quad \sum_{i \in I} \alpha_i.P_i \quad \Big| \quad A\langle a_1, \ldots, a_n \rangle$$

dove I *è un insieme finito di indici.*

Vediamo più in dettaglio questi due costrutti:

Scelta non deterministica

- $\sum_{i \in I} \alpha_i.P_i$ **somma** o **scelta non deterministica**. Intuitivamente, questo processo ha la possibilità di eseguire l'azione α_i, al variare di $i \in I$, evolvendo nel processo P_i. In tal senso, il processo può *scegliere* tra $\mid I \mid$ differenti modi di evolvere. La scelta è non deterministica perchè alcune di queste azioni α_i possono essere uguali tra loro, cioè è possibile che $\alpha_i = \alpha_j$ per qualche $i, j \in I$.

- Ad esempio, per $I = \{1, 2, 3\}$, il processo $\sum_{i \in I} \alpha_i.P_i$ può essere scritto in forma estesa come

$$\alpha_1.P_1 + \alpha_2.P_2 + \alpha_3.P_3.$$

- Nel caso in cui $\alpha_1 = \overline{a}$, $\alpha_2 = b$ e $\alpha_3 = \overline{a}$ il processo sopra non è altro che

$$\overline{a}.P_1 + b.P_2 + \overline{a}.P_3.$$

  Intuitivamente, questo è un processo che può fare un input sul canale b, continuando secondo quanto indicato nel processo P_2; oppure può eseguire un output $\overline{a}$ e continuare o come indicato da P_1 oppure come da P_3.

- Quando $I = \emptyset$ il processo $\sum_{i \in I} \alpha_i.P_i$ è la **somma vuota**, che abbrevieremo con $\mathbf{0}$.

Processi costanti

- $A\langle a_1, \ldots a_n\rangle$ viene chiamato **processo costante**.
- Intuivamente, A è il nome di una procedura, mentre $A\langle a_1, \ldots a_n\rangle$ è l'invocazione della procedura A con parametri $a_1, \ldots a_n$.
- Per ogni costante (o procedura) A deve esistere una equazione che la definisce:
$$A(\vec{x}) \stackrel{\text{def}}{=} P_A$$
dove il corpo P_A è un processo somma, i nomi $\vec{x}$ sono **tutti distinti** e contengono tutti i **nomi liberi** di P_A.
Ad esempio:
$$A(x, y, z) \stackrel{\text{def}}{=} y.\overline{z}.\mathbf{0} + \overline{x}.\overline{y}\ .$$
- Non confondete la definizione di una costante (o procedura), $A(\vec{x})$, in cui $\vec{x}$ sono i *parametri formali*, con l'invocazione, $A\langle\vec{b}\rangle$, a cui vengono passati i *parametri attuali* $\vec{b}$. Il risultato di tale applicazione consiste nel sostituire nel corpo della costante i parametri formali

con quelli attuali (come nell'invocazione di una procedura), e lo scriviamo $\{\vec{b}/\vec{x}\}P_A$.

- **Sostituzione:** $\{\vec{b}/\vec{x}\}P$ sostituisce tutte le occorrenze *libere*[1] di x_i in P con b_i. Nota che per essere ben definita $|\ \vec{x}\ |=|\ \vec{b}\ |$. Nota inoltre che la sostituzione deve essere **simultanea**. Ad esempio: $\{^{b,x}/_{c,b}\}x.\overline{b}.c.\mathbf{0} = x.\overline{x}.b.\mathbf{0}$ e non $x.\overline{x}.x.\mathbf{0}$.

- La definizione formale data in maniera induttiva è:

$$
\begin{array}{lcl}
\{\vec{b}/\vec{c}\}\alpha & \stackrel{\text{def}}{=} & \left[\begin{array}{ll} b_i & \text{if } \alpha = c_i \\ \overline{b_i} & \text{if } \alpha = \overline{c_i} \\ \alpha & \text{altrimenti} \end{array}\right] \\
\{\vec{b}/\vec{c}\}\vec{\alpha} & \stackrel{\text{def}}{=} & \{\vec{b}/\vec{c}\}\alpha_1, \ldots, \{\vec{b}/\vec{c}\}\alpha_n \\
\{\vec{b}/\vec{c}\}A\langle\vec{\alpha}\rangle & \stackrel{\text{def}}{=} & A\langle\{\vec{b}/\vec{c}\}\vec{\alpha}\rangle \\
\{\vec{b}/\vec{c}\}\sum_{i\in I}\alpha_i.P_i & \stackrel{\text{def}}{=} & \sum_{i\in I}\{\vec{b}/\vec{c}\}\alpha_i.\{\vec{b}/\vec{c}\}P_i
\end{array}
$$

[1] Spiegheremo meglio cosa si intenda con ciò.

Semantica dei processi sequenziali

Diamo adesso la *semantica operazionale*, attraverso un LTS, del nostro piccolo linguaggio per processi sequenziali.

Definizione 12 *Due processi sequenziali P e Q sono* **strutturalmente congruenti**, *scritto $P \equiv Q$, se possiamo trasformare l'uno nell'altro sostituendo occorrenze di $A\langle\vec{b}\rangle$ con $\{\vec{b}/\vec{x}\}P_A$, o viceversa, per A arbitrario definito dall'equazione $A(\vec{x}) \stackrel{def}{=} P_A$.*

Ad esempio se:

$$
\begin{array}{rcl}
A(x,y) & \stackrel{\mathrm{def}}{=} & x.A\langle x,y\rangle + y.B\langle x,x\rangle \\
B(w,z) & \stackrel{\mathrm{def}}{=} & w.z.\mathbf{0}
\end{array}
$$

Allora:

$$
\begin{array}{rcl}
B\langle a, a\rangle & \equiv & a.a.\mathbf{0} \\
A\langle d, e\rangle & \equiv & d.A\langle d, e\rangle + e.B\langle d, d\rangle \\
 & \equiv & d.(d.A\langle d, e\rangle + e.B\langle d, d\rangle) + e.d.d.\mathbf{0} \\
 & \equiv & \ldots
\end{array}
$$

L'evoluzione dei processi sequenziali la modelliamo attraverso un LTS.

Definizione 13 (LTS per processi sequenziali) *L'LTS di processi sequenziali è dato dall'insieme di stati $\mathcal{P}^{\mathrm{seq}}$ e dall'insieme di transizioni così definite: se $P \equiv \sum_{i \in I} \alpha_i.P_i$ allora, per ogni $j \in I$, $P \xrightarrow{\alpha_j} P_j$.*

Ad esempio:

$$
\begin{array}{rcl}
\alpha_1.P_1 + \alpha_2.P_2 + \alpha_3.P_3 & \xrightarrow{\alpha_1} & P_1 \\
\alpha_1.P_1 + \alpha_2.P_2 + \alpha_3.P_3 & \xrightarrow{\alpha_2} & P_2 \\
\alpha_1.P_1 + \alpha_2.P_2 + \alpha_3.P_3 & \xrightarrow{\alpha_3} & P_3
\end{array}
$$

Esempi di processi sequenziali

1. Guardiamo l'esempio a pag. 22. Diamo la specifica di un piccolo server che gestisce un buffer di booleani di capacità due, $B^{(2)}$, ini-

zialmente vuoto. Si noti che tutte le strutture dati che descriveremo nei nostri linguaggi sono sempre entità attive, che interagiscono con l'ambiente esterno. Il buffer di questo esempio è parametrico sui nomi di canale in_i e out_i per $i \in \{0,1\}$, ovvero vi sono quattro canali. Il buffer riceve booleani e li ritrasmette rispettando l'ordine di ricezione (politica FIFO). La ricezione di uno 0 è modellato attraverso un'azione in_0, ovvero un input sul canale in_0. L'output di un 1 è modellato attraverso un'azione $\overline{out_1}$, ovvero un output sul canale out_1. Similmente per gli altri due casi. Il buffer può trovarsi in uno stato $B^{(2)}_s$ al variare di $s \in \{0,1,00,01,10,11\}$. Così esistono sette identificatori di processi: $B^{(2)}, B^{(2)}_0, B^{(2)}_1, B^{(2)}_{00}, \ldots$ Diamo la definizione formale del buffer:

$$
\begin{array}{lcl}
B^{(2)} & \stackrel{\text{def}}{=} & \sum_{i\in\{0,1\}} in_i.B^{(2)}_i \\
B^{(2)}_i & \stackrel{\text{def}}{=} & \overline{out_i}.B^{(2)} + \sum_{j\in\{0,1\}} in_j.B^{(2)}_{ji} \\
B^{(2)}_{ij} & \stackrel{\text{def}}{=} & \overline{out_j}.B^{(2)}_i
\end{array}
$$

- Vedremo in seguito come sia possibile combinare più buffer per ottenere un buffer di capacità superiore.

- Vedremo anche come decomporre $B^{(2)}$ in processi più semplici.

2. Scrivere un buffer $B^{(3)}$ analogo a quello visto a pag. 23 ma di capacità 3.

$$
\begin{array}{lcl}
B^{(3)} & \stackrel{\mathrm{def}}{=} & \sum_{i\in\{0,1\}} in_i.B^{(3)}_i \\
B^{(3)}_i & \stackrel{\mathrm{def}}{=} & \overline{out_i}.B^{(3)} + \sum_{j\in\{0,1\}} in_j.B^{(3)}_{ji} \\
B^{(3)}_{ij} & \stackrel{\mathrm{def}}{=} & \overline{out_j}.B^{(3)}_i + \sum_{k\in\{0,1\}} in_k.B^{(3)}_{kji} \\
B^{(3)}_{ijk} & \stackrel{\mathrm{def}}{=} & \overline{out_k}.B^{(3)}_{ij}
\end{array}
$$

3. Modificare il buffer $B^{(2)}$ in modo da consentire che i valori ricevuti siano inviati senza tener necessariamente conto dell'ordine di

arrivo:

$$\begin{array}{lcl} B^{(2)} & \stackrel{\text{def}}{=} & \sum_{i\in\{0,1\}} in_i.B_i^{(2)} \\ B_i^{(2)} & \stackrel{\text{def}}{=} & \overline{out_i}.B^{(2)} + \sum_{j\in\{0,1\}} in_j.B_{ji}^{(2)} \\ B_{ij}^{(2)} & \stackrel{\text{def}}{=} & \overline{out_i}.B_j^{(2)} + \overline{out_j}.B_i^{(2)} \end{array}$$

Scheduler

1. Un insieme di agenti P_i, per $1 \leq i \leq n$, deve essere schedulato.
2. Ciascun P_i desidera eseguire un certo task ripetutamente e lo scheduler si occupa di far sì che gli agenti **inizino** i loro task in ordine ciclico, iniziando da P_1.
3. Ogni agente, quando termina il proprio task, lo segnala allo scheduler.
4. Più processi possono eseguire il proprio task allo stesso tempo.
5. Comunque, lo scheduler non consente a P_i di iniziare un nuovo task se prima non ha terminato il task precedente.

6. Ogni processo P_i richiede di iniziare il proprio task "premendo" il tasto a_i; dopodichè segnalerà la fine del task "premendo" il tasto b_i.

Guardate la figura a pag. 23 del libro. Informalmente la specifica dello scheduler è la seguente:

1. I pulsanti $a_1, \ldots, a_n$ possono essere premuti ciclicamente partendo da a_1.

2. Per ciascun processo P_i i tasti a_i e b_i devono essere premuti in modo alternato.

3. ogni pulsante può essere premuto, purchè si rispettino i precedenti due punti.

Scriviamo la specifica come un processo sequenziale.

- A tal fine definiamo uno stato $Sched_{i,X}$ per ogni $i \in \{1, \ldots, n\}$ e $X \subseteq \{1, \ldots, n\}$.

- Il parametro i indica che è il turno del processo P_i di iniziare.

- Il parametro X rappresenta l'insieme di agenti che in quel momento stanno eseguendo il loro task.

- Così in uno stato $Sched_{i,X}$:

  1. solo P_i può iniziare (purchè $i \notin X$)
  2. ciascun P_j, con $j \in X$, può completere il proprio task.

La **specifica formale** è data allora dalle seguenti equazioni:

$$
\begin{aligned}
Scheduler &\stackrel{\text{def}}{=} Sched_{1,\emptyset} \\
Sched_{i,X} &\stackrel{\text{def}}{=} \begin{pmatrix} \sum_{j \in X} b_j.Sched_{i,X \setminus j} & (i \in X) \\ \sum_{j \in X} b_j.Sched_{i,X \setminus j} + a_i.Sched_{i+1,X \cup i} & (i \notin X) \end{pmatrix}
\end{aligned}
$$

dove $i+1$ è interpretato modulo n, ed abbreviamo $X \cup \{i\}$ e $X \setminus \{i\}$, con $X \cup i$ e $X \setminus i$, rispettivamente.
Esercizio 3.15(a) mostrare che lo scheduler non è mai in stallo.
Esercizio 3.15(b) Tracciare il grafo delle transizioni per $n = 2$.
Esercizio 3.15(c) Da quanti stati è composto lo scheduler?

Esercizio 3.15(d) Se avessimo definito lo scheduler come descritto sotto cosa sarebbe cambiato?

$$Sched_{i,X} \stackrel{\text{def}}{=} \sum_{j \in X} b_j.Sched_{i,X\setminus j} + a_i.Sched_{i+1,X\cup i}$$

indipendentemente dal fatto che i sia in X o meno.

Counter

- Fino adesso abbiamo visto processi con un numero **finito** di stati. Vediamo adesso un semplice processo contatore in grado di mantenere in memoria un numero naturale grande a piacere.
- Un po' come avviene nella programmazione ad oggetti, pensate ad un numero come un oggetto capace di interagire.

$$\begin{aligned} Count &\stackrel{\text{def}}{=} Count_0 \\ Count_0 &\stackrel{\text{def}}{=} inc.Count_1 + \overline{zero}.Count_0 \\ Count_{n+1} &\stackrel{\text{def}}{=} inc.Count_{n+2} + \overline{dec}.Count_n \end{aligned}$$

Stack

L'idea del contatore si può facilmente generalizzare per definire la specifica di una **pila** di booleani. Con $Stack_s$, dove $s \in \{0,1\}^\star$, denoteremo una pila contenente la stringa s. Nel seguito $i, j \in \{0,1\}$.

$$
\begin{array}{rcl}
Stack & \stackrel{\text{def}}{=} & Stack_\epsilon \\
Stack_\epsilon & \stackrel{\text{def}}{=} & \sum_{i\in\{0,1\}} push_i.Stack_i + \overline{empty}.Stack_\epsilon \\
Stack_{is} & \stackrel{\text{def}}{=} & \overline{pop}_i.Stack_s + \sum_{j\in\{0,1\}} push_j.Stack_{jis}
\end{array}
$$

Similmente è possibile modellare una coda.

Queue

$$
\begin{array}{rcl}
Queue & \stackrel{\text{def}}{=} & Queue_\epsilon \\
Queue_\epsilon & \stackrel{\text{def}}{=} & \sum_{i\in\{0,1\}} put_i.Queue_i + \overline{empty}.Queue_\epsilon \\
Queue_{si} & \stackrel{\text{def}}{=} & \overline{get}_i.Queue_s + \sum_{j\in\{0,1\}} put_j.Queue_{jsi}
\end{array}
$$