Lezione 14 MSC
Value-passing CCS

Roberto Gorrieri

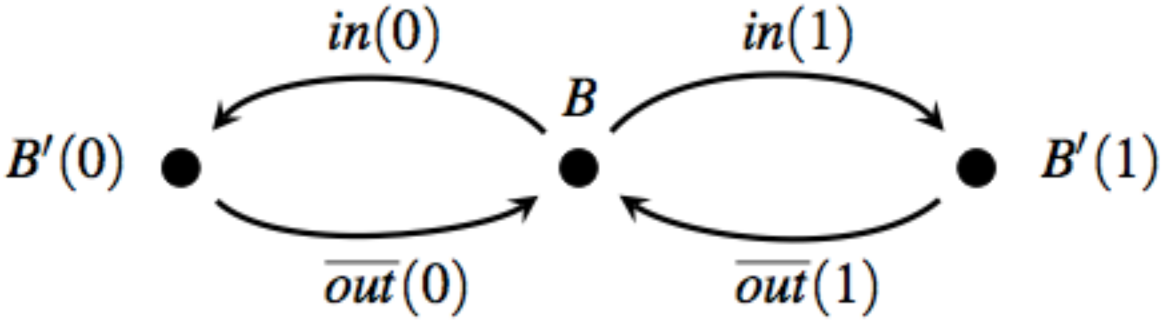
Value-passing

- CCS “puro” ammette solo sincronizzazione, ma non reale comunicazione con scambio di valori.
- Assumiamo di avere un insieme D di dati (di solito l’insieme N dei numeri naturali)
- Azioni di input parametrizzate: a(x)
- Azioni di output con valori: ‘a(v)
- Costanti parametrizzate: $A(x_1, .., x_n) = p$, intendendo che tutte le x_1 sono distinte e che in p le variabili possono essere solo i parametri di A.

Esempio

- B = in(x).B'(x)   B'(y) = 'out(y mod 2).B

- La x di in(x) lega il seguito (B'(x)). La y di B'(y) lega il seguito ('out(y mod 2)). (N.B.: l'espressione y mod 2 viene calcolata.)
- Se assumiamo che i valori trasmittibili siano solo 0 e 1, allora il suo lts deve essere:



Sintassi

- Si assume l’esistenza di due categorie sintattiche (non specificate):
  - Espressioni aritmetiche e, e’, … e.g., 3 + x
  - Espressioni booleane b, b’, … e.g., x > 2 & 2 < y

  P ::= 0 | **a(x).Q | ‘a(e).Q** | tau.Q | P + P **| if b then P else P**

  Q ::= P | Q|Q | (va)Q | **C(x_1, .., x_n)**
- Un termine Q è un processo se, oltre ad essere fully defined e guardato, ogni occorrenza di una variabile è legata o da un input o da una definizione di costante.
- Esempio: in A(x) = ‘out(x+y).B y non è legata

Esempio: if-then-else

- One-position buffer che fa l'output n! del fattoriale del suo input n:

$$Fact \stackrel{def}{=} in(x).F(x,1)$$

$$F(x,y) \stackrel{def}{=} \textbf{if } x = 0 \textbf{ then } \overline{out}(y).Fact \textbf{ else } \tau.F(x-1, x \times y)$$

dove il numero di passi tau è pari al valore dell'input.

Semantica SOS

$$\text{(In)}\quad \frac{}{a(x).p \xrightarrow{a(v)} p[v/x]}\ \text{for any } v \in D \qquad \text{(Tau)}\quad \frac{}{\tau.p \xrightarrow{\tau} p}$$

$$\text{(Out)}\quad \frac{}{\bar{a}(e).p \xrightarrow{\bar{a}(v)} p}\quad \text{if } e \text{ has value } v$$

$$\text{(Cons)}\ \frac{p[v_1/x_1,\ldots,v_n/x_n] \xrightarrow{\mu} p'}{C(e_1,\ldots,e_n) \xrightarrow{\mu} p'}\quad \text{if } C(x_1,\ldots,x_n) \stackrel{def}{=} p \text{ and each } e_i \text{ has value } v_i$$

$$\text{(Then)}\ \frac{p \xrightarrow{\mu} p'}{\textbf{if } b \textbf{ then } p \textbf{ else } q \xrightarrow{\mu} p'}\quad \text{if } b \text{ has value } true$$

$$\text{(Else)}\ \frac{q \xrightarrow{\mu} q'}{\textbf{if } b \textbf{ then } p \textbf{ else } q \xrightarrow{\mu} q'}\quad \text{if } b \text{ has value } false$$

Encoding to VP-CCS su CCS (1)

- L'estensione del value-passing è inessenziale, in quanto è possibile, dato p in VP-CCS, ottenere un p' in CCS, tale che i due generino lts isomorfi (a meno di leggero renaming delle etichette).
- Idea dell'encoding: un input a(x) genera un azione a_v, per ogni valore v nell'insieme D dei dati. Lo stesso per le costanti.
- Esempio: confronta l'lts generato con le regole VP per

$$B = in(x).B'(x) \quad B'(y) = \text{'}out(y \bmod 2).B$$

con l'lts generato con le regole CCS per

$$PB = in_0.B_0 + in_1.B_1 \qquad B_0 = \text{'}out_0.PB \qquad B_1 = \text{'}out_1.PB$$

Encoding to VP-CCS su CCS (2)

$$[\![\mathbf{0}]\!] = \mathbf{0} \qquad\qquad [\![\tau.p]\!] = \tau.[\![p]\!]$$

$$[\![a(x).p]\!] = \Sigma_{v\in D}\, a_v.[\![p[v/x]]\!] \qquad\qquad [\![\bar{a}(e).p]\!] = \overline{a_v}.[\![p]\!] \quad \text{if } e \text{ has value } v$$

$$[\![p_1 + p_2]\!] = [\![p_1]\!] + [\![p_2]\!] \qquad\qquad [\![p_1 \,|\, p_2]\!] = [\![p_1]\!] \,|\, [\![p_2]\!]$$

$$[\![(\nu a)p]\!] = (\nu\{a_v \,|\, v \in D\})[\![p]\!] \qquad [\![\textbf{if } b \textbf{ then } p \textbf{ else } q]\!] = \begin{cases} [\![p]\!] \text{ if } b \text{ has value } true \\ [\![q]\!] \text{ if } b \text{ has value } false \end{cases}$$

Moreover, we have $[\![A(e_1,\ldots,e_n)]\!] = A_{v_1,\ldots,v_n}$ if each e_i has value v_i. Furthermore, for any single constant definition $A(x_1,\ldots,x_n) \stackrel{def}{=} p$, we have a family of constant definitions

$$A_{v_1,\ldots,v_n} \stackrel{def}{=} [\![p[v_1/x_1,\ldots,v_n/x_n]]\!]$$

one for each vector $(v_1,\ldots,v_n) \in D^n$.

Encoding to VP-CCS su CCS (3)

- Osserva che il termine ottenuto è davvero un termine CCS solo se l'insieme dei dati D è finito. Altrimenti avremmo:
  - Somma infinita in corrispondenza di input
  - Infinite costanti per implementarne una di VP-CCS
  - Restrizione su insiemi infiniti

Encoding to VP-CCS su CCS (4)

- Correttezza dell'encoding (con D finito):

Proposition 3.8. *Given a finite set D of values, the lts for a value-passing CCS process p, generated with the rules of Table 3.2, is isomorphic (modulo renaming of $a(v)$ with a_v) to the lts for the pure CCS process $[\![p]\!]$, generated with the rules of Table 3.1.*

Proof. One has to prove that $p \xrightarrow{a(v)} p'$ implies $[\![p]\!] \xrightarrow{a_v} [\![p']\!]$ and, conversely, that $[\![p]\!] \xrightarrow{a_v} q$ implies there exists p' such that $q = [\![p']\!]$ and $p \xrightarrow{a(v)} p'$. (Analogously, for the other labels $\overline{a}(v)/\overline{a}_v$ and τ.) This can be proved by induction on the proof of the transitions and is left as an exercise to the reader. Then, the bijection f maps a value-passing CCS process p' reachable from p to its encoding $[\![p']\!]$. $\square$

CCS-VP e π-calcolo

- In CCS-VP si assume che l'insieme dei dati D e l'insieme delle azioni Act siano disgiunti, così che un valore ricevuto non possa poi essere usato come canale di comunicazione:
- a(x).'x(b).C   qui il valore v ricevuto da a viene poi usato come nome di canale per trasmettere il valore b.
- Se i nomi dei canali diventano valori trasmittibili, allora il linguaggio permette mobilità dei canali e struttura a flow-graph riconfigurabili dinamicamente. Questi aspetti sono trattati nel π-calcolo.

Esempio (1): Buffers

Specifica sequenziale:

Buffer FIFO:

$$
\begin{aligned}
B_{fifo} &\stackrel{def}{=} in(x).B_1(x) \\
B_1(x) &\stackrel{def}{=} in(y).B_2(y,x) + \overline{out}(x).B_{fifo} \\
B_2(x,y) &\stackrel{def}{=} \overline{out}(y).B_1(x)
\end{aligned}
$$

Buffer BAG:

$$
\begin{aligned}
B_{bag} &\stackrel{def}{=} in(x).B'_1(x) \\
B'_1(x) &\stackrel{def}{=} in(y).B'_2(y,x) + \overline{out}(x).B_{bag} \\
B'_2(x,y) &\stackrel{def}{=} \overline{out}(x).B'_1(y) + \overline{out}(y).B'_1(x)
\end{aligned}
$$

- N.B: i due non sono equivalenti! A meno che $|D| = 1$
- N.B: se D ha k elementi, mi servono $k^2 + k + 1$ costanti in CCS puro.

Esempio (1): Buffers (2)

- One position buffer: $B \stackrel{def}{=} in(x).B'(x),\ B'(x) \stackrel{def}{=} \overline{out}(x).B$

- Implementazione parallela del bag-buffer: B|B
- B|B è strongly bisimile a B_{bag} perché

$$R = \{(B_{bag}, B \mid B)\} \cup \{(B'_1(v), B'(v) \mid B) \mid v \in D\} \cup \{(B'_2(u,v), B'(v) \mid B'(u)) \mid u, v \in D\}$$

R è una strong bisimulation up to ~.

Esempio (1): Buffers (3)

- Pipeline Buffer: Implementazione parallela del FIFO Buffer

$$
\begin{array}{llll}
Buf \stackrel{def}{=} (\nu d)(Buf_1 \,|\, Buf_2) & & & \\
Buf_1 \stackrel{def}{=} in(x).Buf_1'(x) & & Buf_1'(x) \stackrel{def}{=} \overline{d}(x).Buf_1 & \\
Buf_2 \stackrel{def}{=} d(x).Buf_2'(x) & & Buf_2'(x) \stackrel{def}{=} \overline{out}(x).Buf_2 &
\end{array}
$$

- Buf è weakly bisimile a B_{fifo} perché

$$
\begin{aligned}
S = {} & \{(B_{fifo}, Buf)\} \cup \{(B_{fifo}, (\nu d)(Buf_1 \,|\, Buf_2))\} \cup \\
& \{(B_1(v), (\nu d)(Buf_1'(v) \,|\, Buf_2)) \mid v \in D\} \cup \\
& \{(B_1(v), (\nu d)(Buf_1 \,|\, Buf_2'(v))) \mid v \in D\} \cup \\
& \{(B_2(u,v), (\nu d)(Buf_1'(u) \,|\, Buf_2'(v))) \mid u, v \in D\}
\end{aligned}
$$

è una weak bis.

Esempio (2): Unbounded bag buffer

- Specifica sequenziale:

$$UB(\varepsilon) \stackrel{def}{=} in(x).UB(x)$$

$$UB(x_1,\ldots x_n) \stackrel{def}{=} \Sigma_{i=1}^{n}\overline{out}(x_i).UB(x_1,\ldots,x_{i-1},x_{i+1},\ldots x_n) + in(y).UB(y,x_1,\ldots x_n)$$

- La traduzione in CCS puro produrrebbe infinite costanti (anche se D è finito) perché le tuple possono crescere indefinitamente.
- Implementazione: $U \stackrel{def}{=} in(x).(U \mid \overline{out}(x).\mathbf{0})$

- Esercizio: dimostra che U e UB(ε) sono strong bisimili!

Esempio (3): Stack

- Specifica sequenziale di uno stack:

$$
\begin{aligned}
Stack(\varepsilon) &\doteq empty.Stack(\varepsilon) + push(x).Stack(x) \\
Stack(x\sigma) &\doteq \overline{pop}(x).Stack(\sigma) + push(y).Stack(yx\sigma)
\end{aligned}
$$

- Anche se D è finito, mi servono infinite costanti in CCS puro, perché il parametro σ può crescere indefinitamente.

Esempio (3): Stack (2)

- Simile al counter:

$$S \stackrel{def}{=} empty.S + push(x).((\nu a)(S_1(x) \,|\, a.S))$$

$$S_1(x) \stackrel{def}{=} \overline{pop}(x).\bar{a}.\mathbf{0} + push(y).((\nu b)(S_2(y) \,|\, b.S_1(x)))$$

$$S_2(y) \stackrel{def}{=} \overline{pop}(y).\bar{b}.\mathbf{0} + push(x).((\nu a)(S_1(x) \,|\, a.S_2(y)))$$

- Usa solo 2k +1 costanti in CCS puro, se D ha k elementi.
- Si può dimostrare che Stack(ε) è weakly bisimile a S, nello stesso modo in cui abbiamo dimostrato l'analogo per il counter.

Esempio (4): Coda ovvero Unbounded Fifo buffer

- Specifica sequenziale:

$$Queue(\varepsilon) \stackrel{def}{=} empty.Queue(\varepsilon) + in(x).Queue(x)$$

$$Queue(x\sigma) \stackrel{def}{=} \overline{out}(x).Queue(\sigma) + in(y).Queue(x\sigma y)$$

- Implementazione:

$$Q \stackrel{def}{=} empty.Q + in(x).((\nu l)(L(x) \mid H))$$

$$L(x) \stackrel{def}{=} \overline{out}(x).\bar{l} \qquad H \stackrel{def}{=} l.Q + in(x).((\nu l')(M'(x) \mid H')) \qquad M'(x) \stackrel{def}{=} l.L'(x)$$

$$L'(x) \stackrel{def}{=} \overline{out}(x).\bar{l}' \qquad H' \stackrel{def}{=} l'.Q + in(x).((\nu l)(M(x) \mid H)) \qquad M(x) \stackrel{def}{=} l'.L(x)$$

Esempio (4): Coda ovvero Unbounded Fifo buffer (2)

- Implementazione: lista di processi che cresce verso destra ed ha la forma

$$\ldots L(v_1) \mid \ldots \mid M(v_2) \mid \ldots \mid M(v_3) \mid \ldots \mid H \ldots$$

dove L rappresenta la coda (left), i vari M gli elementi intermedi, e H la testa (head) dove inserire nuovi elementi in coda.

- Se facciamo l'out di v1 e un input di v4, la nuova configurazione diventa

$$\ldots \mathbf{0} \mid \ldots \mid L(v_2) \mid \ldots \mid M(v_3) \mid \ldots \mid M(v_4) \mid \ldots \mid H \ldots$$

dove L(v1) è diventato 0, ed ha trasformato M(v2) in L(v2).

Esempio (4): Coda ovvero Unbounded Fifo buffer (3)

$$
\begin{array}{llr}
Q & \xrightarrow{in(v_1)} (\nu l)(L(v_1) \mid H) & \xrightarrow{in(v_2)} \\
& (\nu l)(L(v_1) \mid (\nu l')(M'(v_2) \mid H')) & \xrightarrow{\overline{out}(v_1)} \xrightarrow{\tau} \\
& (\nu l)(\mathbf{0} \mid (\nu l')(L'(v_2) \mid H')) & \xrightarrow{in(v_3)} \\
& (\nu l)(\mathbf{0} \mid (\nu l')(L'(v_2) \mid (\nu l)(M(v_3) \mid H))) & \xrightarrow{\overline{out}(v_2)} \xrightarrow{\tau} \\
& (\nu l)(\mathbf{0} \mid (\nu l')(\mathbf{0} \mid (\nu l)(L(v_3) \mid H))) & \xrightarrow{\overline{out}(v_3)} \xrightarrow{\tau} \\
& (\nu l)(\mathbf{0} \mid (\nu l')(\mathbf{0} \mid (\nu l)(\mathbf{0} \mid Q))) &
\end{array}
$$

- Dimostrare che Q e Queue(ε) sono weak bisimili è difficile (vedi sezione 3.6 del libro).