6. CCS com passagem de valores

José Proença
(slides mainly from Nelma Moreira)

Concurrent programming (CC3040) 2025/2026

CISTER – U.Porto, Porto, Portugal

https://fm-dcc.github.io/cp2526

U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

CISTER - Research Centre in
Real-Time & Embedded
Computing Systems

Máquina de Café

O modelo básico

$$Machine \quad := \quad coin?.coffee!.Machine$$

é muito simplista. Normalmente temos de usar várias moedas

Máquina de Café

$$\begin{aligned} \mathit{Machine} &:= \mathit{coin}?.\mathit{Machine} + \mathit{coin}?.\mathit{PaidMachine} \\ \mathit{PaidMachine} &:= \mathit{coffee}?.(\mathit{change}!.\mathit{Machine} + \tau.\mathit{Machine}) \end{aligned}$$

Máquina de Café

Mas, suponham uma máquina em que o café custa 5 euros e aceita moedas de 1, 2 e 5 euros. Como tomar café? Solução: considerar todas as possibilidades... e troco

Máquina de Café

Mas, suponham uma máquina em que o café custa 5 euros e aceita moedas de 1, 2 e 5 euros. Como tomar café? Solução: considerar todas as possibilidades... e troco

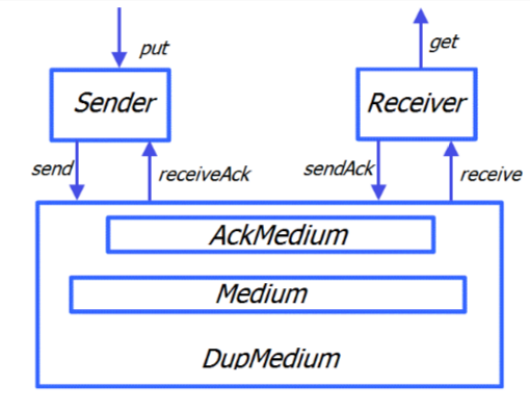
$$
\begin{aligned}
Machine0 &:= coin1?.Machine1 + coin2?.Machine2 + coin5?.Machine5 \\
Machine1 &:= coin1?.Machine2 + coin2?.Machine3 + coin5?.Machine6 \\
Machine2 &:= coin1?.Machine3 + coin2?.Machine4 + coin5?.Machine7 \\
Machine3 &:= coin1?.Machine4 + coin2?.Machine5 + coin5?.Machine8 \\
Machine4 &:= coin1?.Machine5 + coin2?.Machine6 + coin5?.Machine9 \\
Machine5 &:= coffee?.Machine0 \\
Machine6 &:= coffee?.change1!.Machine0 \\
Machine7 &:= coffee?.change1!.change1!.Machine0 \\
Machine8 &:= coffee?.change1!.change1!.change1!.Machine0 \\
Machine9 &:= coffee?.change1!.change1!.change1!.change1!.Machine0
\end{aligned}
$$

Máquina de Café

Mas, suponham uma máquina em que o café custa 5 euros e aceita moedas de 1, 2 e 5 euros. Como tomar café? Solução: considerar todas as possibilidades... e troco

$$
\begin{aligned}
Machine0 &:= coin1?.Machine1 + coin2?.Machine2 + coin5?.Machine5 \\
Machine1 &:= coin1?.Machine2 + coin2?.Machine3 + coin5?.Machine6 \\
Machine2 &:= coin1?.Machine3 + coin2?.Machine4 + coin5?.Machine7 \\
Machine3 &:= coin1?.Machine4 + coin2?.Machine5 + coin5?.Machine8 \\
Machine4 &:= coin1?.Machine5 + coin2?.Machine6 + coin5?.Machine9 \\
Machine5 &:= coffee?.Machine0 \\
Machine6 &:= coffee?.change1!.Machine0 \\
Machine7 &:= coffee?.change1!.change1!.Machine0 \\
Machine8 &:= coffee?.change1!.change1!.change1!.Machine0 \\
Machine9 &:= coffee?.change1!.change1!.change1!.change1!.Machine0
\end{aligned}
$$

Exemplo de um Protocolo com erro no meio –Pseuco



Exemplo de um Protocolo com erro no meio –Pseuco

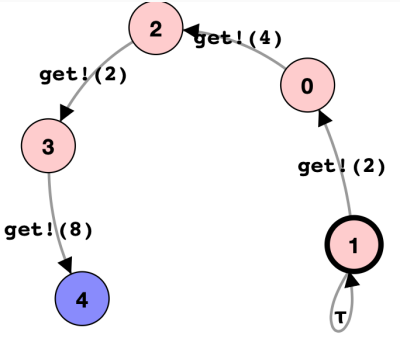
$$
\begin{aligned}
Sender &:= put?.send!.Sending \\
Sending &:= receiveAck?.Sender + receiveNAck?.send!.Sending \\
Receiver &:= receive?.get?.sendAck!.Receiver + \\
&\quad gargled?.sendNAck!.Receiver \\
Medium &:= send?.(receive!.Medium + i.garbled!.Medium) \\
AckMedium &:= sendAck?.receiveAck!.AckMedium + \\
&\quad sendNAck?.receivedNAck!.AckMedium \\
DupMedium &:= Medium|AckMedium \\
Protocol &:= (Sender \mid Receiver \mid DupMedium)\backslash \\
&\quad \{send, receive, sendAck, receiveAck, \\
&\quad receiveNAck, sendNAck, garbled\}
\end{aligned}
$$

Envio de uma mensagem

$$
\begin{aligned}
Sender &:= put?x.send!x.Sending[x] \\
Sending[x] &:= receiveAck?.Sender + receiveNAck?.send!x.Sending[x] \\
Receiver[x] &:= receive?x.get?x.sendAck!.Receiver[x] + \\
&\quad gargled?.sendNAck!.Receiver[x] \\
Medium &:= send?x.(receive!x.Medium + i.garbled!.Medium) \\
AckMedium &:= sendAck?.receiveAck!.AckMedium + \\
&\quad sendNAck?.receivedNAck!.AckMedium \\
DupMedium &:= Medium|AckMedium \\
Protocol &:= (Sender \mid Receiver \mid DupMedium)\backslash \\
&\quad \{send, receive, sendAck, receiveAck, \\
&\quad receiveNAck, sendNAck, garbled\}
\end{aligned}
$$

Envio de uma mensagem

$$Protocol|put!2.put!4.put!2.put!8.0\backslash\{put\}$$



CCS_{vp} com passagem de valor

- $a!v$: saída do valor v no canal a (enviar)

CCS_{vp} com passagem de valor

- $a!v$: saída do valor v no canal a (enviar)
- $a?v$: entrada do valor v pelo canal a (receber)

CCS_{vp} com passagem de valor

- $a!v$: saída do valor v no canal a (enviar)
- $a?v$: entrada do valor v pelo canal a (receber)
- ou usar variáveis

CCS_{vp} com passagem de valor

- $a!v$: saída do valor v no canal a (enviar)
- $a?v$: entrada do valor v pelo canal a (receber)
- ou usar variáveis
- $a!x$: saída do valor guardado em x no canal a (enviar)

CCS_{vp} com passagem de valor

- $a!v$: saída do valor v no canal a (enviar)
- $a?v$: entrada do valor v pelo canal a (receber)
- ou usar variáveis
- $a!x$: saída do valor guardado em x no canal a (enviar)
- $a?x$: entrada de um valor que se guarda em x pelo canal a (receber)

CCS_{vp} com passagem de valor

- $a!v$: saída do valor v no canal a (enviar)
- $a?v$: entrada do valor v pelo canal a (receber)
- ou usar variáveis
- $a!x$: saída do valor guardado em x no canal a (enviar)
- $a?x$: entrada de um valor que se guarda em x pelo canal a (receber)
- e os nomes dos processos podem ter variáveis com parâmetros permitindo assim enviar e receber valores ($A[x, y]$)

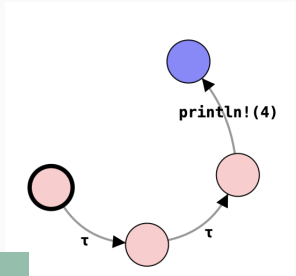
Uma célula de memória (1-*buffer*)

$$B := put?x.B1[x]$$
$$B1[x] := get!(x+1).B$$

Então temos para

$$(B|put!3.get?y.println!y)\{*, println\}$$

o LTS



CCS_{vp} com passagem de valor

Sendo $\mathbb{V}$ um conjunto de valores e $\mathbb{K}$ um conjunto de canais temos

Sendo $\mathbb{V}$ um conjunto de valores e $\mathbb{K}$ um conjunto de canais temos

$$\begin{aligned} A^{!} &= \{a!v \mid a \in \mathbb{K}, v \in \mathbb{V}\} \cup \{a! \mid a \in \mathbb{K}\}, \\ A^{?} &= \{a?v \mid a \in \mathbb{K}, v \in \mathbb{V}\} \cup \{a? \mid a \in \mathbb{K}\}, \\ Com &= A^{!} \cup A^{?} \\ Act &= Com \cup \{\tau\} \end{aligned}$$

Sendo $\mathbb{V}$ um conjunto de valores e $\mathbb{K}$ um conjunto de canais temos

$$\begin{aligned} A^! &= \{a!v \mid a \in \mathbb{K}, v \in \mathbb{V}\} \cup \{a! \mid a \in \mathbb{K}\}, \\ A^? &= \{a?v \mid a \in \mathbb{K}, v \in \mathbb{V}\} \cup \{a? \mid a \in \mathbb{K}\}, \\ Com &= A^! \cup A^? \\ Act &= Com \cup \{\tau\} \end{aligned}$$

$$\begin{aligned} P &::= 0 \mid X[r_1, \ldots, r_n] \mid P + P \mid \chi.P \mid P|P \mid P\backslash H \\ \chi &::= \tau \mid a! \mid a? \mid a!v \mid a?v \mid a!x \mid a?x \end{aligned}$$

onde $X \in Var$, $x \in D$, $r_i \in D \cup \mathbb{V} \cup \mathbb{K}$

Regras do CCS_{vp}

$$\text{Pref}\ \frac{\alpha \in Act}{\alpha.P \xrightarrow{\alpha} P}$$

$$\text{Pref}\ \frac{\alpha \in Act}{\alpha.P \xrightarrow{\alpha} P}$$

A avaliação de expressões $e \Downarrow z$: a expressão e avalia para z.

$$\text{Pref}\ \frac{\alpha \in Act}{\alpha.P \xrightarrow{\alpha} P}$$

A avaliação de expressões $e \Downarrow z$: a expressão e avalia para z.

$$\text{Output}\ \frac{e \Downarrow z}{a!e.P \xrightarrow{a!z} P}$$

$$\text{Valor}\ \frac{e \Downarrow z}{a?e.P \xrightarrow{a?z} P}$$

$$\text{Input}\ \frac{v \in \mathbb{V}}{a?x.P \xrightarrow{a?v} P\{v/x\}}$$

onde $P\{v/x\}$ é P onde x é substituito por v (e x não é uma ação)

$$\text{Pref}\ \frac{\alpha \in Act}{\alpha.P \xrightarrow{\alpha} P}$$

$$\text{Input}\ \frac{v \in \mathbb{V}}{a?x.P \xrightarrow{a?v} P\{v/x\}}$$

onde $P\{v/x\}$ é P onde x é substituito por v (e x não é uma ação)

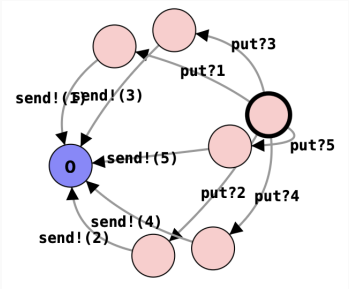
$$\begin{aligned}
(a!y.P)\{v/x\} &= a!y.P\{v/x\} \quad \text{se } y \neq x \\
(a!x.P)\{v/x\} &= a!v.P\{v/x\} \\
(a?y.P)\{v/x\} &= a?y.P\{v/x\} \quad \text{se } y \neq x \\
(a?x.P)\{v/x\} &= a?x.P \\
X[x]\{v/x\} &= X[v] \\
X[y]\{v/x\} &= X[y] \quad \text{se } y \neq x
\end{aligned}$$

Para as restantes expressões é passado para as subexpressões.

$$\begin{aligned} send!y.Sending[x]\{3/x, 5/y\} &= send!5.Sending[3] \\ send!y.Sending[x]\{3/x, 5/y, receive/send\} &= receive!5.Sending[3] \end{aligned}$$

$$\begin{aligned} send!(x+y)\{3/x, 5/y\} &= sent!(8) \\ 3+5 &\Downarrow 8 \end{aligned}$$

Calcular o LTS de $put?x : 0..5.send!x.0$



$$\text{Rec}\ \frac{P\{v_1/r_1,\ldots,v_n/r_n\} \xrightarrow{\alpha} P' \qquad \Gamma(X[r_1,\ldots,r_n]) = P}{X[v_1,\ldots,v_n] \xrightarrow{\alpha} P'}$$

when: Bloqueador Condicional

Supomos $\mathbb{V} = \mathbb{Z}$ ($CCS^{\mathbb{Z}}_{vp}$)

$$B[x] \quad := \quad when(x < 4)put?.B[x+1] + when(x > 0)get?.B[x-1]$$

$B[0]$ ou $B[5]$ o que fazem?

when: Bloqueador Condicional

Supomos $\mathbb{V} = \mathbb{Z}$ (CCS^{Z}_{vp})

A expressão $when(b)P$ se b é verdade comporta-se como P senão bloqueia.

Vamos só considerar expressões com inteiros.

$$\text{cond}\ \frac{P \xrightarrow{\alpha} P' \qquad b \Downarrow True}{when(b)P \xrightarrow{\alpha} P'}$$

Acrescentamos à gramática

$$
\begin{array}{rcl}
P & ::= & 0 \mid X[r_1, \ldots, r_n] \mid P + P \mid \chi.P \mid P|P \mid P\backslash H \mid \mathit{when}(b)P \\
\chi & ::= & \tau \mid a! \mid a? \mid a!v \mid a?v \mid a!x \mid a?x
\end{array}
$$

$$
\begin{aligned}
\mathit{Machine}[b] &:= \mathit{when}(b < 5)\mathit{coin}?c.\mathit{Machine}[b + c] \\
&\quad + \mathit{when}(b \geq 5)\mathit{coffee}!.\mathit{ReturnMachine}[b - 5] \\
\mathit{ReturnMachine}[b] &:= \mathit{when}(b > 0)\mathit{change}!.\mathit{ReturnMachine}[b - 1] \\
&\quad + \mathit{Machine}[0] \\
\mathit{User} &:= \mathit{coin}!2.\mathit{coin}!2.\mathit{coin}!2.\mathit{coffee}?.\mathit{change}?.0
\end{aligned}
$$

Calcula $[\![(\mathit{Machine}[0] | \mathit{User}) \backslash \{\mathit{coin}, \mathit{change}, \mathit{coffee}\}]\!]_{\Gamma}$.

$$
\begin{aligned}
IterMult[z, x, y] \quad &:= \quad when(x > 0)i.IterMult[z + y, x - 1, y] \\
&\quad\quad +when(x == 0)println!z.0 \\
IterMult[0, 3, 7]
\end{aligned}
$$

$$\text{Pref}\ \frac{\alpha \in Act}{\alpha.P \xrightarrow{\alpha} P} \qquad \text{Output}\ \frac{e \Downarrow z}{a!e.P \xrightarrow{a!.z} P}$$

$$\text{Input}\ \frac{v \in \mathbb{V}}{a?x.P \xrightarrow{a?v} P\{v/x\}} \qquad \text{Valor}\ \frac{e \Downarrow z}{a?e.P \xrightarrow{a?z} P}$$

$$\text{Rec}\ \frac{P\{v_1/r_1, \ldots, v_n/r_n\} \xrightarrow{\alpha} P' \qquad \Gamma(X[r_1, \ldots, r_n]) = P}{X[r_1, \ldots, r_n] \xrightarrow{\alpha} P'} \qquad \text{cond}\ \frac{P \xrightarrow{\alpha} P' \qquad b \Downarrow True}{when(b)P \xrightarrow{\alpha} P'}$$

Factorial

$$
\begin{aligned}
Fac[n, j] \quad := \quad & when(j > 0)i.Fac[n * j, j - 1] \\
& + when(j == 0)println!n.0
\end{aligned}
$$

Calcular $Fac[1, 5]$

Células de Memória (partilhada)

$$Cell_x[cur] \quad := \quad get_x!cur.Cell_x[cur] + set_x?new : 0..2.Cell_x[new]$$

Células de Memória (partilhada)

Mais geralmente:

$$Cell[rd, wr, x] \quad := \quad rd!x.Cell[rd, wr, x] + wr?y.Cell[rd, wr, y]$$

Células de Memória (partilhada)

Mais geralmente:

$$Cell[rd, wr, x] \quad := \quad rd!x.Cell[rd, wr, x] + wr?y.Cell[rd, wr, y]$$

- $Cell[rd, wr, 5]$
- $Cell[rdA, wrA, 0] | Cell[rdB, wrB, 0]$

Células de Memória (partilhada)

Mais geralmente:

$$Cell[rd, wr, x] \;:=\; rd!x.Cell[rd, wr, x] + wr?y.Cell[rd, wr, y]$$

- $Cell[rd, wr, 5]$
- $Cell[rdA, wrA, 0] | Cell[rdB, wrB, 0]$

$$\begin{aligned}
Cells &:= Cell[rdA, wrA, 0] | Cell[rdB, wrB, 0] \\
Serve &:= mult?.rdA?x : R.rdB?y : R.IterMult[0, x, y] \\
IterMult[z, x, y] &:= when(x > 0)i.IterMult[z + y, x - 1, y] \\
&\quad + when(x == 0)println!z.Serve \\
Use &:= wrA!7.wrB!5.mult!.0
\end{aligned}$$

$(Cells|Serve|Use)\setminus\{rdA, wrA, rdB, wrB, mult\}$

Factorial - Versão iterativa

$$
\begin{aligned}
Fac[n, j] \quad := \quad & when(j > 0)i.Fac[n * j, j - 1] \\
& + when(j == 0)println!n.0
\end{aligned}
$$

Ficheiro no Pseuco.com:

`https://pseuco.com/#/edit/remote/5bglrm4937vze10dvlqi`

Factorial - Versão iterativa

$$\begin{aligned} Fac[n,j] &:= when(j>0)i.Fac[n*j, j-1] \\ &\quad + when(j==0)println!n.0 \end{aligned}$$

$$\begin{aligned} Fak &:= rdJ?j : R.(when(j>0)rdN?n.wrN!(n*j).wrJ!(j-1).Fak \\ &\quad + when(j==0)rdN?n.print!n.0) \\ Cell[v,rd,wr] &:= rd!v.Cell[v,rd,wr] + wr?x : R.Cell[x,rd,wr] \\ Cells &:= Cell[0,rdN,wrN] | Cell[0,rdJ,wrJ] \end{aligned}$$

$$(wrN!1.wrJ!5.Fak|Cells)\backslash\{rdN, wrN, rdJ, wrJ\}$$

O CCS_{vp} pode ser embebido no CCS

..logo é só "syntatic sugar"...

CCS_{vp}	CCS
$a!v.P$	$a_v!.P$
$a?x.P$	$\sum_{v\in\mathbb{V}} a_v?.P\{v/x\}$
$X[u_1,\ldots,u_n]$	$X_{u_1,\ldots,u_n}$

Isto é basta usar ações e nomes indexados, podendo ser considerados conjuntos infinitos de índices ($\mathbb{V}$ ou D)

O seguinte código

```
while (a > 0) {
 println("loop");
 a = a-1;
}
println("a is zero");
```

pode ser escrito em CCS

```
P[a] := when( a<=0)i.Q[a]
        + when(a>0) println!"loop". P[a-1]
Q[a] :=println!"a is zero"
```

$P[3]$ teria o seguinte LTS