4. CCS: Calculus of Communicating Systems (parallel)

José Proença
(slides mainly from Nelma Moreira)

Concurrent programming (CC3040) 2025/2026

CISTER – U.Porto, Porto, Portugal

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

Operadores do CCS

". " Prefixo a execução de $\alpha.P$ começa com a execução da ação $\alpha \in Act$ e depois comporta-se como P

"+ " Escolha O processo $P + Q$ comporta-se como o processo P ou o processo Q. É a escolha não determinística

"| " Composição Paralela O processo $P|Q$ representa a execução concorrente de P e Q (que progridem independentemente no tempo).

Exemplo

Exemplos de $1 - Buffer$ *(de uma posição)*

Considera as equações de Γ *seguintes.*

1. *Calcula* $[\![Buffer]\!]_\Gamma$.

$$Buffer \quad := \quad put?.get?.Buffer$$

2. *Calcula* $[\![put?.BufferM]\!]_\Gamma$.

$$BufferM \quad := \quad put?.get?.BufferM + get?.put?.BufferM$$

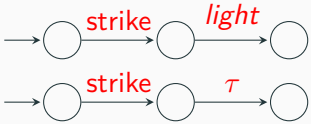
3. *Calcula* $[\![Buffer0]\!]_\Gamma$.

$$\begin{aligned} Buffer0 &:= put?.Buffer1 \\ Buffer1 &:= get?.Buffer2 + get?.Buffer0 \\ Buffer2 &:= get?.Buffer1 \end{aligned}$$

Composição paralela

Vimos que a composição paralela (intercalada) de dois sistemas de transição correspondia ao seu produto.

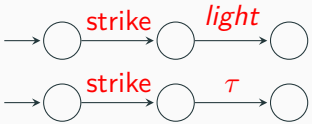
Por exemplo, tendo dois fósforos: Um acende, outro não



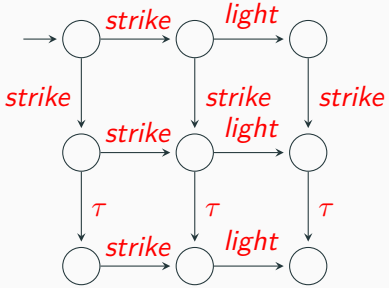
Composição paralela

Vimos que a composição paralela (intercalada) de dois sistemas de transição correspondia ao seu produto.

Por exemplo, tendo dois fósforos: Um acende, outro não



O produto seria então:



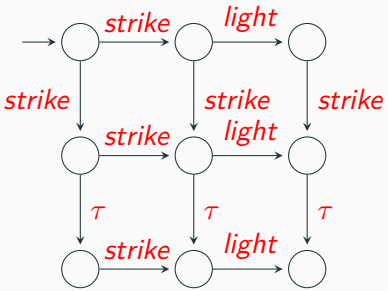
Composição paralela

Vimos que a composição paralela (intercalada) de dois sistemas de transição correspondia ao seu produto.

Podiamos então ter o seguinte termo do CCS_0:

$$strike.light.strike.\tau.0 + strike.strike.light.\tau.0 + strike.\tau.strike.light.0 + strike.strike.\tau.light.0$$

O produto seria então:



Composição paralela

Vimos que a composição paralela (intercalada) de dois sistemas de transição correspondia ao seu produto.

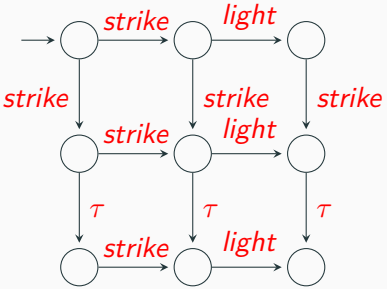
Podiamos então ter o seguinte termo do CCS_0:

$$strike.light.strike.\tau.0 + strike.strike.light.\tau.0 + strike.\tau.strike.light.0 + strike.strike.\tau.light.0$$

Mas vamos considerar um operador próprio que permite mais flexibilidade e termos mais concisos. No caso anterior seria:

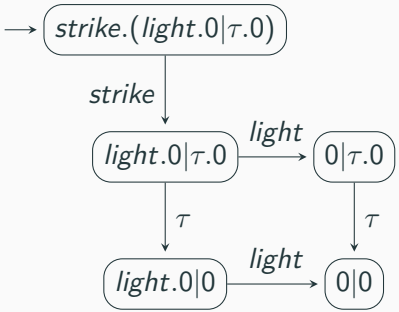
$$strike.light.0 | strike.\tau.0$$

O produto seria então:

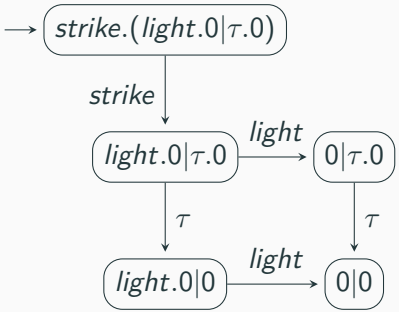


Mais dois fósforos: $\mathit{strike}.(\mathit{light}.0|\tau.0)$

Mais dois fósforos: $strike.(light.0|\tau.0)$



Mais dois fósforos: $strike.(light.0|\tau.0)$



Supomos sempre que todas as ações são instântaneas.

Regras de inferência *Par*

$$\text{ParE}\ \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}$$

$$\text{ParD}\ \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$$

Para que dois sistemas possam comunicar temos de implementar algum tipo de sincronia, neste caso de *handshaking*: dois processos comunicam com um par de ações que partilham o mesmo "nome".

Para que dois sistemas possam comunicar temos de implementar algum tipo de sincronia, neste caso de *handshaking*: dois processos comunicam com um par de ações que partilham o mesmo "nome". Seja

- O conjunto de ações observáveis *Com* é dividido em dois
- $Com = A^! \cup A^?$
- $A^!$ conjunto de ações de saída (*output*) (envio)
- $A^?$ conjunto de ações de entrada (*input*) (recebidas)
- Ações com o mesmo nome formam um par e são complementares:
- um processo envia $a^!$ e e outro recebe $a^?$
- O complemento de $a \in A^! \cup A^?$ designa-se por $\overline{a}$: se $a \in A^!$ então $\overline{a} \in A^?$ e vice-versa.
- Para a ação interna τ, temos $\tau = \overline{\tau}$.
- $\forall \alpha \in Act, \overline{\overline{\alpha}} = \alpha$.

Regra de inferência de Sincronia

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P|Q \xrightarrow{\cdots} P'|Q'}$$

Regra de inferência de Sincronia

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P|Q \xrightarrow{\cdots} P'|Q'}$$

A sincronização não é observável por processos externos.

Regra de inferência de Sincronia

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P|Q \xrightarrow{\cdots} P'|Q'}$$

A sincronização não é observável por processos externos.

A ação interna τ representa a sincronização para o exterior.

Regra de inferência de Sincronia

$$\text{Sync}\ \frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

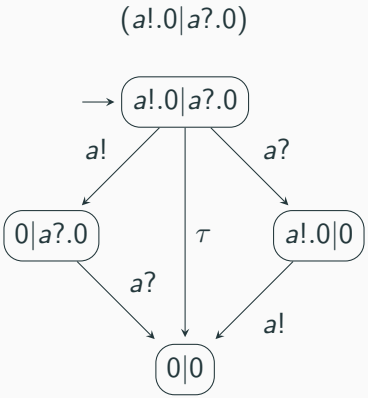
$$\text{ParE}\ \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}$$

$$\text{ParD}\ \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$$

Exemplo

$$(a!.0 | a?.0)$$

Exemplo



A regra Sync não exclui a utilização de ParD e ParE.

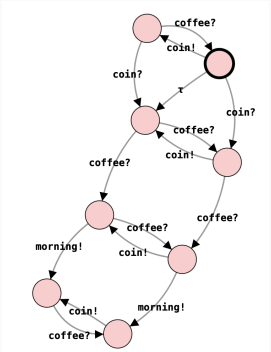
Exemplo

Seja o seguinte modelo de uma máquina de café e um utilizador:

$$User := coin!.coffee?.morning!.0$$

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

Desenha o LTS de $[\![(User|Machine)]\!]_\Gamma$ usando as regras de inferência.



Operador de Restrição

- Proibe que pares de ações observáveis (a e $\overline{a}$) sejam usadas individualmente.
- Força a sincronia (a aplicação da regra Sync)
- $P\backslash H$ onde P é um processo e H um conjunto de ações de comunicação que serão proibidas (por vezes em vez de $a!, a?$ apenas se escreve a em H).
- As ações internas τ não podem estar em H

$$\text{Res}\ \frac{P \xrightarrow{\alpha} P' \qquad \alpha \notin H}{P\backslash H \xrightarrow{\alpha} P'\backslash H}$$

Exemplo de Restrição

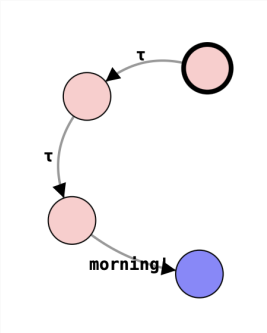
$$((a!.0|a!.0)|a?.0)\backslash\{a!, a?\} \xrightarrow{\tau} ((a!.0|0)|0)\backslash\{a!, a?\}$$

$$((a!.0|a!.0)|a?.0)\backslash\{a!, a?\} \xrightarrow{\tau} ((0|a!.0)|0)\backslash\{a!, a?\}$$

$$\text{ParE}\ \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad \text{ParD}\ \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'} \qquad \frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\bar{a}} Q'}{P|Q \xrightarrow{\cdots} P'|Q'} \qquad \text{Res}\ \frac{P \xrightarrow{\alpha} P' \qquad \alpha \notin H}{P\backslash H \xrightarrow{\alpha} P'\backslash H}$$

Exemplo

No exemplo anterior calcula o LTS $[\![(User|Machine)\backslash\{coin, coffeee\}]\!]_\Gamma$ e testa no Pseuco.com ou no ccs-caos.



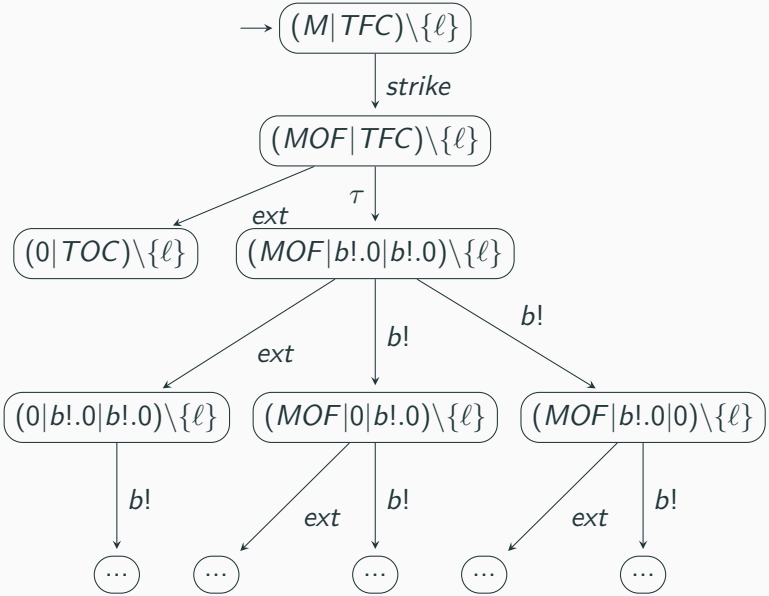
Indica o termo que etiqueta cada estado.

Considera as seguintes equações

$$\begin{aligned} Match &:= strike.MatchOnFire \\ MatchOnFire &:= light!.MatchOnFire + extinguish!.0 \\ TwoFireCracker &:= light?.(bang!.0 | bang!.0) \end{aligned}$$

Constrói o LTS a expressão abaixo e testa no Pseuco.com ou no ccs-caos.

$$(Match | TwoFireCracker) \backslash \{light\}$$


$(M|TFC)\backslash\{\ell\}$
strike
$(MOF|TFC)\backslash\{\ell\}$
τ
ext
$(0|TOC)\backslash\{\ell\}$
$(MOF|b!.0|b!.0)\backslash\{\ell\}$
ext
$b!$
$b!$
$(0|b!.0|b!.0)\backslash\{\ell\}$
$(MOF|0|b!.0)\backslash\{\ell\}$
$(MOF|b!.0|0)\backslash\{\ell\}$
$b!$
ext
$b!$
ext
$b!$
…
…
…
…
…

CCS (quase) completo

Seja $Com = A^! \cup A^?$ conjunto de ações de comunicação, $Act = Com \cup \{\tau\}$ um conjunto de ações, e Var um conjunto de nomes (variáveis). As expressões do CCS são

$$P \quad ::= \quad 0 \mid X \mid P + P \mid \alpha.P \mid P|P \mid P\backslash H$$

onde $\alpha \in Act$, $X \in Var$ e $H \subseteq Com$. Supomos um conjunto Γ de equações $X := P$.

Semântica do CCS

A semântica das expressões do *CCS* é então

$$[\![\_]\!] : (Var \to CCS) \to CCS \to LTS_{CCS}$$

tal que

$$[\![P]\!]_\Gamma = (CCS, \longrightarrow_\Gamma, P)$$

com

$$LTS_{CCS} = \{(CCS, T, s) \mid T \subseteq CCS \times Act \times CCS, \wedge s \in CCS\}$$

onde $\longrightarrow_\Gamma$ a mais pequena relação que satisfaz as regras de inferência.

Pref $\dfrac{}{\alpha.P \xrightarrow{\alpha} P}$

EscD $\dfrac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$

Sync $\dfrac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\bar{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$

Res $\dfrac{P \xrightarrow{\alpha} P' \qquad \alpha \notin H}{P\backslash H \xrightarrow{\alpha} P'\backslash H}$

ParE $\dfrac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}$

EscE $\dfrac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$

ParD $\dfrac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$

Rec $\dfrac{P \xrightarrow{\alpha} P' \qquad \Gamma(X) = P}{X \xrightarrow{\alpha} P'}$

Mais regras de prioridade

- $P|Q|R$ é $(P|Q)|R$

Mais regras de prioridade

- $P|Q|R$ é $(P|Q)|R$
- $\alpha.P|Q$ é $(\alpha.P)|Q$

Mais regras de prioridade

- $P|Q|R$ é $(P|Q)|R$
- $\alpha.P|Q$ é $(\alpha.P)|Q$
- $P + R|Q$ é $(P + R)|Q$

Expressividade do CCS

- CCS_0 só pode produzir sistemas de transição finitos acíclicos

Expressividade do CCS

- CCS_0 só pode produzir sistemas de transição finitos acíclicos
- CCS_0^ω com Γ finito só pode produzir sistemas de transição estado-finitos (a menos de isomorfismo)

Expressividade do CCS

- CCS_0 só pode produzir sistemas de transição finitos acíclicos
- CCS_0^{ω} com Γ finito só pode produzir sistemas de transição estado-finitos (a menos de isomorfismo)
- CCS pode produzir sistemas e transição infinitos e com ramificação infinita

Expressividade do CCS

- CCS_0 só pode produzir sistemas de transição finitos acíclicos
- CCS_0^{ω} com Γ finito só pode produzir sistemas de transição estado-finitos (a menos de isomorfismo)
- CCS pode produzir sistemas e transição infinitos e com ramificação infinita
- Ex: $X := a.0|X$ tem ramificação infinita (Verifica!)

Expressividade do CCS

- CCS_0 só pode produzir sistemas de transição finitos acíclicos
- CCS_0^{ω} com Γ finito só pode produzir sistemas de transição estado-finitos (a menos de isomorfismo)
- CCS pode produzir sistemas e transição infinitos e com ramificação infinita
- Ex: $X := a.0|X$ tem ramificação infinita (Verifica!)
- com mais uma operação - renomeação $P[f]$ tem a potência duma Máquina de Turing.

Operador de renomeação

- Já vimos uma máquina de café em CCS

Operador de renomeação

- Já vimos uma máquina de café em CCS
- $CM := coin?.coffee!.CM$

Operador de renomeação

- Já vimos uma máquina de café em CCS
- $CM := coin?.coffee!.CM$
- Mas agora se quisermos um chocolate?

Operador de renomeação

- Já vimos uma máquina de café em CCS
- $CM := coin?.coffee!.CM$
- Mas agora se quisermos um chocolate?
- Seria igual apenas mudando a accão de envio.

Operador de renomeação

- Já vimos uma máquina de café em CCS
- $CM := coin?.coffee!.CM$
- Mas agora se quisermos um chocolate?
- Seria igual apenas mudando a accão de envio.
- $ChM := coin?.choco!.ChM$

Operador de renomeação

- Já vimos uma máquina de café em CCS
- $CM := coin?.coffee!.CM$
- Mas agora se quisermos um chocolate?
- Seria igual apenas mudando a accão de envio.
- $ChM := coin?.choco!.ChM$
- e o mesmo para outros produtos.

Operador de renomeação

- Já vimos uma máquina de café em CCS
- $CM := coin?.coffee!.CM$
- Mas agora se quisermos um chocolate?
- Seria igual apenas mudando a accão de envio.
- $ChM := coin?.choco!.ChM$
- e o mesmo para outros produtos.
- Então podemos ter

Operador de renomeação

- Já vimos uma máquina de café em CCS
- $CM := coin?.coffee!.CM$
- Mas agora se quisermos um chocolate?
- Seria igual apenas mudando a accão de envio.
- $ChM := coin?.choco!.ChM$
- e o mesmo para outros produtos.
- Então podemos ter
- $VM := coin?.item!.VM$

Operador de renomeação

- Já vimos uma máquina de café em CCS
- $CM := coin?.coffee!.CM$
- Mas agora se quisermos um chocolate?
- Seria igual apenas mudando a accão de envio.
- $ChM := coin?.choco!.ChM$
- e o mesmo para outros produtos.
- Então podemos ter
- $VM := coin?.item!.VM$
- e definir

$$\begin{aligned} CM &:= VM[coffee/item] \\ ChM &:= VM[choc/item] \\ &\dots \end{aligned}$$

Operador de renomeação

Seja $f : Act \rightarrow Act$ uma função de renomeação tal que

$$\begin{aligned} f(\tau) &:= \tau, \\ f(\overline{a}) &:= \overline{f(a)} \ \forall a \in Com. \end{aligned}$$

A função f pode representar-se da forma $[b_1/a_1, \ldots b_n/a_n]$ se $f(a_i) = b_i$.

Então, sendo P um processo $P[f]$ é também um processo em que cada ação a é substituida por $f(a)$ (assim como os complementos).

$$\begin{aligned} (a.B + b.B)[a/b, b/a] &= (b.B + a.B) \\ (a.0 + \overline{a}.A)[a/b] &= (a.0 + \overline{a}.A) \end{aligned}$$

Regra de inferência para $P[f]$

$$\text{Rel}\ \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

Regra de inferência para $P[f]$

$$\text{Rel}\ \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

Exemplo

Seja $A := a.b.B$, *calcular*

$$[\![(A|b.a.B) + (b.A)[a/b]]\!]_\Gamma$$

Operadores Estáticos e Dinâmicos

- Operadores dinâmicos: $\cdot$ e $+$

Operadores Estáticos e Dinâmicos

- Operadores dinâmicos: $\cdot$ e $+$
- Desaparecem após se efectuar uma transição

Operadores Estáticos e Dinâmicos

- Operadores dinâmicos: $\cdot$ e $+$
- Desaparecem após se efectuar uma transição

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

$$\text{EscD}\ \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

- Operadores estáticos: $|$ e $\setminus$ (também $[f]$)

Operadores Estáticos e Dinâmicos

- Operadores dinâmicos: $\cdot$ e $+$
- Desaparecem após se efectuar uma transição
- Operadores estáticos: $|$ e $\setminus$ (também $[f]$)
- Não desaparecem após ser efectuada uma transição

Operadores Estáticos e Dinâmicos

- Operadores dinâmicos: $\cdot$ e $+$
- Desaparecem após se efectuar uma transição
- Operadores estáticos: $|$ e $\setminus$ (também $[f]$)
- Não desaparecem após ser efectuada uma transição

$$\text{Sync}\ \frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

$$\text{ParD}\ \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$$

$$\text{Res}\ \frac{P \xrightarrow{\alpha} P' \qquad \alpha \notin H}{P\setminus H \xrightarrow{\alpha} P'\setminus H}$$

Não é permitida recursão sobre operadores estáticos.

$$
\begin{aligned}
P &::= 0 \mid X \mid P + P \mid \alpha.P \mid R \\
R &::= 0 \mid R + R \mid \alpha.R \mid R|R \mid R\backslash H
\end{aligned}
$$

Não é permitida recursão sobre operadores estáticos.

$$
\begin{array}{rcl}
P & ::= & 0 \mid X \mid P + P \mid \alpha.P \mid R \\
R & ::= & 0 \mid R + R \mid \alpha.R \mid R|R \mid R\backslash H
\end{array}
$$

Proposição

Se Γ é uma função parcial cujo contradomínio só tem expressões regulares então o Reach($[\![P]\!]_\Gamma$) é finito por estados para todo $P \in CCS$ (i.e. o LTS atingível é finito por estados).

Não é permitida recursão sobre operadores estáticos.

$$
\begin{array}{rcl}
P & ::= & 0 \mid X \mid P + P \mid \alpha.P \mid R \\
R & ::= & 0 \mid R + R \mid \alpha.R \mid R|R \mid R\backslash H
\end{array}
$$

Proposição

Se Γ é uma função parcial cujo contradomínio só tem expressões regulares então o Reach($[\![P]\!]_\Gamma$) é finito por estados para todo $P \in CCS$ (i.e. o LTS atingível é finito por estados).

Este é o cálculo que se usa mais na prática.

Buffer Paralelos

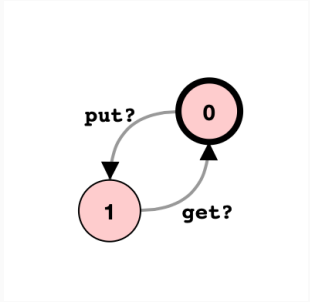
Calcular $[\![Buffer|Buffer]\!]_\Gamma$. para

$$Buffer \quad := \quad put?.get?.Buffer$$

Buffer Paralelos

Calcular $[\![Buffer|Buffer]\!]_\Gamma$. para

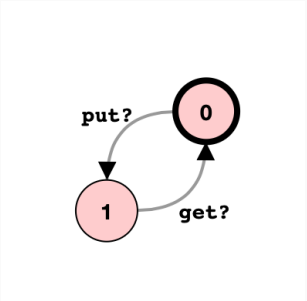
$$Buffer \quad := \quad put?.get?.Buffer$$



Buffer Paralelos

Calcular $[\![Buffer|Buffer]\!]_\Gamma$. para

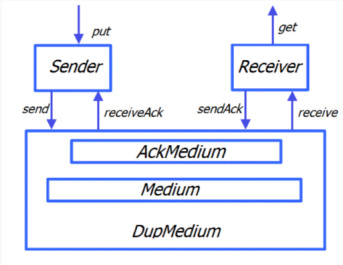
$$Buffer := put?.get?.Buffer$$



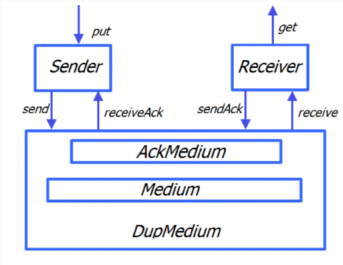
$$BufferL := put?.pass!.BufferL$$
$$BufferR := pass?.get?.BufferR$$

Calcular
$[\![(BufferL|BufferR)\backslash\{pass!, pass?\}]\!]$

Exemplo de um Protocolo com Ack (Pseuco)

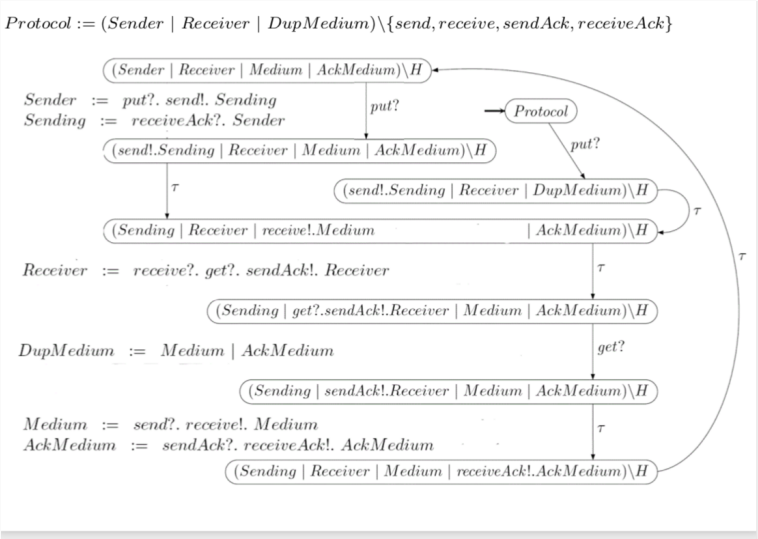


Exemplo de um Protocolo com Ack (Pseuco)



$$
\begin{aligned}
Sender &:= put?.send!.Sending \\
Sending &:= receiveAck?.Sender \\
Receiver &:= receive?.get?.sendAck!.Receiver \\
Medium &:= send?.receive!.Medium \\
AckMedium &:= sendAck?.receiveAck!.AckMedium \\
DupMedium &:= Medium|AckMedium \\
ProtocolG &:= (Sender \mid Receiver \mid DupMedium)\backslash \\
&\quad \{send, receive, sendAck, receiveAck\}
\end{aligned}
$$

Exemplo de um Protocolo com Ack (Pseuco, Holger H.)

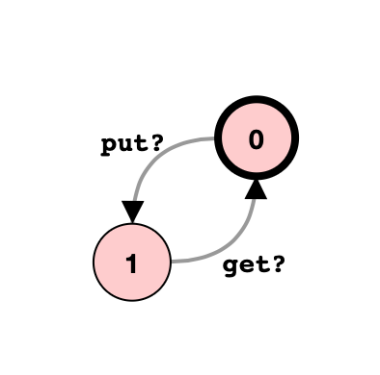


Exemplo de um Protocolo com erro no meio – Pseuco

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

Comportamento Externo

Para um observador externo os processos *Buffer*, *Protocol* e *ProtocolG* têm o mesmo comportamento (mas LTSs diferentes, claro!).



Exclusão Mútua

- Secção Crítica: porção de código que só pode ser executado por um processo num dado instante
- Supõe-se que a execução da secção crítica por um só processo termina.
- MUTEX o problema consiste em ter
  1. um algoritmo de entrada *acquire_mutex*()
  2. um algoritmo de saída *release_mutex*()
- Enquadrando a secção crítica garantem

  Exclusão mútua: que o código da zona crítica é executado no máximo por um processo em cada instante.

  ***Starvation-freedom*:** cada processo que invoca *acquire_mutex*() termina, permitindo assim que os processos que querem entrar na zona crítica o possam fazer.

Exclusão mútua em CCS

Se se modelar a entrada e a saída da zona crítica fica (para dois processos).

$$MutexSpec \quad := \quad enter1.exit1.MutexSpec + enter2.exit2.MutexSpec$$

Algoritmo de Peterson para a Exclusão Mútua

- P_1, P_2 processos
- variáveis partilhadas b_1, b_2 e k, sendo
- se $k = i$ então P_i pode entrar
- P_1 faz $k = 2$ (dá o privilégio a P_2)
- e simétricamente para P_2
- $b_i = \mathit{true}$ quando P_i espera
- $b_i = \mathit{false}$ quando P_i sai da zona crítica.

Processo P_i

while true do
  noncricital actions
  $b_i \leftarrow$ **true;**
  $k \leftarrow j$**;**
  while $b_j \wedge k = j$ **do**
    skip;
  critical actions
  $b_i \leftarrow$ **false**;

Algoritmo de Peterson em CCS

O CCS não tem variáveis, mas estas são processos que comunicam a outros processos que necessitam de ler ou escrever os seus valores. Isto é

- As variáveis são processos cujos estados são os seus possíveis valores
- Para b_1 temos estado B_{1t} se o seu valor for **true** e o estado B_{1f} se o seu valor for **false**.
- Outros processos podem ler ou escrever o valor de variáveis comunicando com o respectivo processo o valor pretendido
- Para um processo ler **true** em b_1 sincroniza com esse processo por uma ação (canal) $b1rt$
- Para um processo escrever **false** em b_1 sincroniza com esse processo por uma ação (canal) $b1wf$
- Análogamente se define o comportamento de k que pode tomar os valores 1 e 2.

Processos para as variáveis

$$
\begin{aligned}
B_{1f} &:= b1rf!.B_{1f} + b1wf?.B_{1f} + b1wt?.B_{1t} \\
B_{1t} &:= b1rt!.B_{1t} + b1wf?.B_{1f} + b1wt?.B_{1t}
\end{aligned}
$$

Processos para as variáveis

$$
\begin{aligned}
B_{1f} &:= b1rf!.B_{1f} + b1wf?.B_{1f} + b1wt?.B_{1t} \\
B_{1t} &:= b1rt!.B_{1t} + b1wf?.B_{1f} + b1wt?.B_{1t}
\end{aligned}
$$

$$
\begin{aligned}
B_{2f} &:= b2rf!.B_{2f} + b2wf?.B_{2f} + b2wt?.B_{2t} \\
B_{2t} &:= b2rt!.B_{2t} + b2wf?.B_{2f} + b2wt?.B_{2t}
\end{aligned}
$$

$$
\begin{aligned}
B_{1f} &:= b1rf!.B_{1f} + b1wf?.B_{1f} + b1wt?.B_{1t} \\
B_{1t} &:= b1rt!.B_{1t} + b1wf?.B_{1f} + b1wt?.B_{1t}
\end{aligned}
$$

$$
\begin{aligned}
B_{2f} &:= b2rf!.B_{2f} + b2wf?.B_{2f} + b2wt?.B_{2t} \\
B_{2t} &:= b2rt!.B_{2t} + b2wf?.B_{2f} + b2wt?.B_{2t}
\end{aligned}
$$

$$
\begin{aligned}
K_1 &:= kr1!.K_1 + kw1?.K_1 + kw2?.K_2 \\
K_2 &:= kr2!.K_2 + kw1?.K_1 + kw2?.K_2
\end{aligned}
$$

Processos para P_1 e P_2

- apenas modelar a entrada e saída da zona crítica
- fazer a iniciallização das variáveis b_i e k
- supomos que não podem terminar na zona crítica ou ficar lá para sempre
- para representar o ciclo **while** para P_1 temos um estado P_{11} tal que
  - ler os valores de b_2 e k
  - esperar se $b_2 = \textbf{true} \wedge k = 2$
  - mudar de estado, P_{12}
  - em P_{12} entrar e sair da zona critica
  - mas como avaliar $b_j \wedge k = j$?
  - avaliamos da esquerda para a direita: se b_j é **true** então avaliamos se $k = j$; a segunda não é avaliada se a primeira for falsa.

Processos para P_1 e P_2 em CCS

$$
\begin{aligned}
P_1 &:= b1wt!.kw2!.P_{11} \\
P_{11} &:= b2rf?.P_{12} + b2rt?.(kr2?.P_{11} + kr1?.P_{12}) \\
P_{12} &:= enter1.exit1.b1wf!.P_1
\end{aligned}
$$

Processos para P_1 e P_2 em CCS

$$
\begin{aligned}
P_1 &:= b1wt!.kw2!.P_{11} \\
P_{11} &:= b2rf?.P_{12} + b2rt?.(kr2?.P_{11} + kr1?.P_{12}) \\
P_{12} &:= enter1.exit1.b1wf!.P_1
\end{aligned}
$$

$$
\begin{aligned}
P_2 &:= b2wt!.kw1!.P_{21} \\
P_{21} &:= b1rf?.P_{22} + b1rt?.(kr1?.P_{21} + kr2?.P_{22}) \\
P_{22} &:= enter2.exit2.b2wf!.P_2
\end{aligned}
$$

Se $k = 1$ no inicio

$$Peterson \quad := \quad (P_1|P_2|B_{1f}|B_{2f}|K_1)\backslash L$$

onde L são todas as ações excepto as de entrada e saída na zona critica.

Ficheiro no Pseuco.com:
`https://pseuco.com/#/edit/remote/tkxz8fpu1t8ke56bj9uo`

Ficheiro no ccs-caos: `https://lmf.di.uminho.pt/ccs-caos/?Peterson`

Será que *MutexSpec* e *Peterson* são equivalentes? Mas para que equivalência? Interessa-nos o comportamento observável...

Teste da exclusão mútua

Contudo podemos mostrar que no algoritmo de Peterson um processo não entra na zona critica se o outro lá está. Para tal usamos um monitor, i.e., outro processo que executando em paralelo irá detectar se há um erro:

$$MutualTest := enter1!.MutualTest1 + enter2!.MutualTest2$$
$$MutualTest1 := exit1!.MutalTest + enter2!.bad!.0$$
$$MutualTest2 := exit2!.MutalTest + enter2!.bad!.0$$

Executando

$$(Peterson|MutualTest)\backslash\{enter1, enter2, exit1, exit2\}$$

ver se a ação *bad*! pode ser executada. Pode-se mostrar que não.