# Aula 5

### Comunicação por canais

- Um canal pode ser um buffer FIFO
- são usados em protocolos de comunicação
- um *sistema de canais*
- tem $n$ processos $P_1$,..., $P_n$, onde cada um tem um grafo $PG_i$ com
- transições condicionais $g : \alpha$ ou *ações de comunicação*:
- $g : c!v$ transmitir a mensagem $v$ pelo canal $c$
- $g : c?x$ receber uma mensagem pelo canal $c$ e guardar na variável $x$.
- $\ell \overset{g:\alpha}{\hookrightarrow} \ell'$, $\ell \overset{g:c!v}{\hookrightarrow} \ell'$, ou $\ell \overset{g:c?x}{\hookrightarrow} \ell'$.
- podem ser síncronos ou assíncronos

### Canais

- Seja $c$ um buffer.
- $c!v$ coloca $v$ no fim do buffer $c$
- $c?x$ vai buscar o elemento no topo do buffer $c$ e guarda-o em $x$
- *capacidade do canal*,
$$cap(c) \in \mathbb{N} \cup \{\infty\},$$
indica o número máximo de mensagens que $c$ pode conter (pode ser finito ou infinito)
- *tipo do canal*, indica o tipo das mensagens que se pode transmitir sobre $c$, $dom(c)$.
- Seja $Chan$ um conjunto de canais, o conjunto das *ações de comunicação* é
$$\begin{aligned} comm \quad = \quad & \{c!v, c?x \mid c \in Chan \wedge v \in dom(c) \wedge \\ & x \in Var \wedge dom(x) \subseteq dom(c)\} \end{aligned}$$

**Síncronos e Assíncronos**

- Ex: um canal $c$ que transmite bits tem $dom(c) = \{0, 1\}$
- Se $cap(c) = 0$ o sistema corresponde a comunicação por *Handshaking*: transmissão e recepção simultânea, *síncrona*
- Se $cap(0) > 0$ há um atraso na transmissão e na recepção da mensagem: *passagem de mensagens assíncrona*

**Sistema de canais (CS)**

$CS = [PG_1|PG_2|\cdots|PG_n]$ sobre $(Var, Chan)$ onde $PG_i$ são grafos de programa sobre $(Var_i, Chan)$

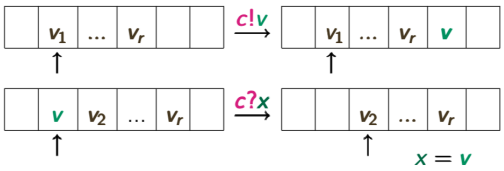
- $Var = \bigcup_{1\leq i\leq n} Var_i$ conjunto de variáveis tipificadas
- $Chan$ conjunto de canais tipificados com capacidades $cap(\cdot)$ e domínios $dom(\cdot)$
- $PG_i = (Loc_i, Act_i, Effect_i, \hookrightarrow_i, Loc_{0,i}, g_{0,i})$

$$\hookrightarrow_i \subseteq Loc_i \times (Cond(Var_i) \times (Act_i \cup Comm_i)) \times Loc_i$$

- $\ell \overset{g:\alpha}{\hookrightarrow}_i \ell'$, $g$ guarda
- $\ell \overset{g:c!v}{\hookrightarrow}_i \ell'$, envia o valor $v$ via o canal $c$
- $\ell \overset{g:c?x}{\hookrightarrow}_i \ell'$, recebe um valor que guarda na variável $x$ $v$ via o canal $c$
- Em geral, se $g = True$ omitimos $g$ : nas ações de comunicação.

**Comunicação se** $cap(c) > 0$

- $P_i$ pode executar a transição $\ell_i \overset{c!v}{\hookrightarrow}_i \ell_i'$ se e só se *o canal não está cheio* e $v$ é guardado no fim de $v$ $(add(c, v))$
- $P_j$ pode executar $\ell_j \overset{c?x}{\hookrightarrow}_j \ell_j'$ se *o canal c não está vazio* $(x := v; remove(c))$



**Comunicação se** $cap(c) = 0$ **(*rendezvous*)**

- o processo $P_i$ pode executar

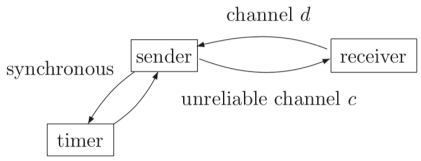
$$\ell_i \overset{c!v}{\hookrightarrow}_i \ell_i'$$

se simultaneamente existe um processo $P_j$ que oferece a ação complementar

$$\ell_j \overset{c?x}{\hookrightarrow}_j \ell_j'$$

- sendo o resultado equivalente a $x := v$ (sincronia).

***Alternating bit Protocol* (ABP)**




- o canal $c$ não é perfeito e pode perder mensagens enviadas
- o canal $d$ é perfeito e envia "acknowledgment"
- Pretende-se protocolo de comunicação que
- assegure que dados distintos de $S$ são entregues a $R$.
- Para tal $S$ tem de retransmitir mensagens
- e uma nova mensagem só é enviada quando houver garantia que a anterior foi recebida (*send and wait*))

***Alternating bit Protocol***

- $S$ envia a mensagem e um bit $y$ extra e ativa o *timer*
- se houver *timout* volta a enviar
- se $R$ enviou $y$ então inicia o timer e faz $y = \neg y$.

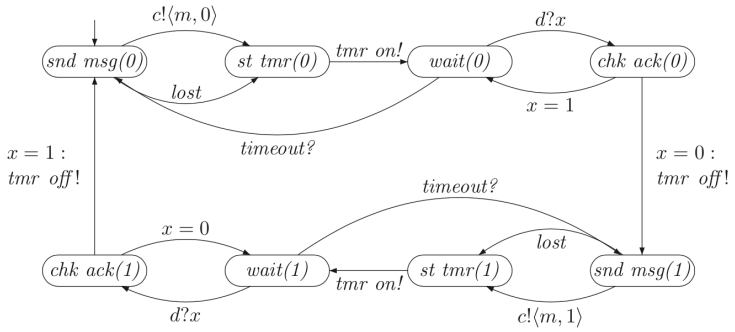
***Alternating bit Protocol***

- $S$ envia mensagens sobre $c$

  $$\langle m_0, b_0\rangle, \langle m_1, b_1\rangle, \ldots$$

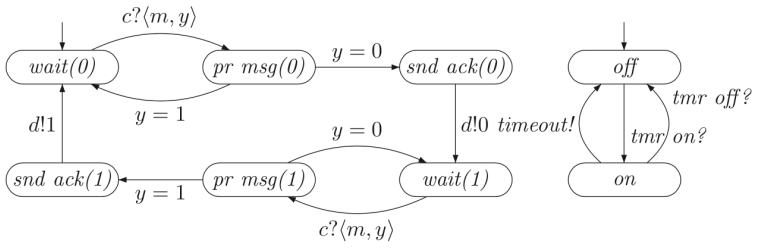
  e $b_0 = 0$, $b_1 = 1$, $b_2 = 0$, ...
- Quando $R$ recebe $\langle m, b\rangle$ envia o bit de controlo $b$ que recebeu pelo canal $d$
- Quando $S$ recebe $b$, $S$ transmite uma nova mensagem $m'$ com o bit $\neg b$.
- Mas se $S$ tiver de demorar muito a receber a mensagem de $R$, $S$ retransmite $\langle m, b\rangle$.

*PG* **para o *Sender***




$$\begin{aligned} Chan &= \{c, d, tmr\_on, tmr\_off, timeout\} \\ Var &= \{x, y, m_i\} \end{aligned}$$

*PG* **para *Receiver* and *Timer***



$$ABP \quad = \quad [S|Timer|R]$$

**Sistemas de transições para um $CS$**

Seja $CS = [PG_1|PG_2|\cdots|PG_n]$ sobre $(Var, Chan)$.

$$PG_i = (Loc_i, Act_i, Effect_i, \hookrightarrow_i, Loc_{0,i}, g_{0,i})$$

- estados $\langle \ell_1, \ldots, \ell_n, \eta, \zeta \rangle$
- $\ell_i$ localização em $PG_i$
- $\eta \in Eval(Var)$ atribuição de valores às variáveis
- $\zeta : Chan \to \bigcup_{c \in Chan} dom(c)^\star$ atribuição de valores dos canais
- para $c \in Chan$, $\zeta(c) \in dom(c)^\star$
- e $len(\zeta(c)) \leq cap(c)$
- $Eval(Chan)$ é o conjunto de todos os $\zeta$.

**Sistemas de transições para um $CS$**

Seja $CS = [PG_1|PG_2|\cdots|PG_n]$ sobre $(Var, Chan)$.

$$PG_i = (Loc_i, Act_i, Effect_i, \hookrightarrow_i, Loc_{0,i}, g_{0,i})$$

- estados iniciais: componentes $\ell_i \in Loc_{0,i}$
- inicialmente todos os canais estão vazios ($\zeta_0(c) = \varepsilon,\ c \in Chan$) e $len(\varepsilon) = 0$.
- $\zeta(c) = v_1 v_2 \cdots v_k$, com $v_1$ o *topo* do canal
- $len(\zeta) = k$
- $\zeta[c := w_1, w_2, \ldots, w_k]$ é a atribuição igual a $\zeta$ mas com $\zeta(c) = w_1 w_2 \cdots w_k$

$$\zeta[c := w_1, w_2, \ldots, w_k](c') = \begin{cases} \zeta(c') \text{ se } c' \neq c \\ w_1 w_2 \cdots w_k \text{se } c' = c \end{cases}$$

$T(CS)$

$$T(CS) = (S, Act, \longrightarrow, I, AP, L)$$

- $S = (Loc_1 \times \cdots \times Loc_n) \times Eval(Var) \times Eval(Chan)$
- $Act = \bigoplus_{0<1\leq n} Act_i \oplus \{\tau\}$, união disjunta
- $I = \{\ \langle \ell_1, \ldots, \ell_n, \eta, \zeta_0\rangle \mid \forall 0 < i \leq n(\ell_i \in Loc_{0,i} \wedge \eta \models g_{0,i})\ \}$
- $AP = \bigoplus_{0<1\leq n} Loc_i \oplus Cond(Var)$
- $L(\langle \ell_1, \ldots, \ell_n, \eta, \zeta\rangle) = \{\ell_1, \ldots, \ell_n\} \cup \{\ g \in Cond(Var) \mid \eta \models g\ \}$
- relação de transição $\longrightarrow$ tem regras para ações $\alpha \in Act_i$ e de passagem de mensagens.

**Concorrência/intercalagem para** $\alpha \in Act_i$

$$\frac{\ell_i \overset{g:\alpha}{\hookrightarrow}_i \ell_i' \wedge \eta \models g}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \zeta\rangle \overset{\alpha}{\longrightarrow} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \zeta\rangle}$$

com $\eta' = Effect(\alpha, \eta)$.

**Passagem de mensagens assíncrona para** $c \in Chan$ **e** $cap(c) > 0$

- receber um valor em $c$ e guardar em $x$

$$\frac{\ell_i \overset{g:c?x}{\hookrightarrow}_i \ell_i' \wedge \eta \models g \wedge \zeta(c) = v_1 \cdots v_k \wedge k > 0}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \zeta\rangle \overset{\tau}{\longrightarrow} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \zeta'\rangle}$$

  com $\eta' = \eta[x := v_1]$ e $\zeta' = \zeta[c := v_2 \cdots v_k]$.

- Transmitir um valor $v \in dom(c)$ sobre $c$

$$\frac{\ell_i \overset{g:c!v}{\hookrightarrow}_i \ell_i' \wedge \eta \models g \wedge \zeta(c) = v_1 \cdots v_k \wedge k < cap(c)}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \zeta\rangle \overset{\tau}{\longrightarrow} \langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta', \zeta'\rangle}$$

  com $\zeta' = \zeta[c := v_1 \cdots v_k v]$.

**Passagem de mensagens síncrona para** $c \in Chan$ **e** $cap(c) = 0$

$$\frac{\ell_i \overset{g_1:c?x}{\hookrightarrow}_i \ell_i' \wedge \eta \models g_1 \wedge \eta \models g_2 \wedge \ell_j \overset{g_2:c!v}{\hookrightarrow}_j \ell_j' \wedge i \neq j}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_j, \ldots \ell_n, \eta, \zeta\rangle \overset{\tau}{\longrightarrow} \langle \ell_1', \ldots, \ell_i', \ldots, \ell_j', \ldots \ell_n', \eta', \zeta\rangle}$$

com $\eta' = \eta[x := v]$.

**Quantos estados tem um sistema de transições...**

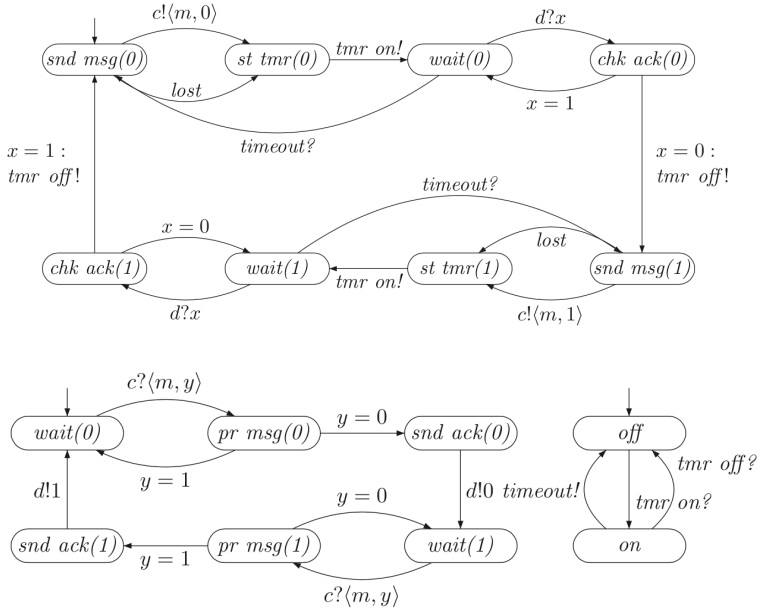
de um sistema de canais com:

- 2 processos com 2 localizações
- 2 variáveis Booleanas
- 2 canais de capacidade 10 de valores Booleanos
- ?

$$2 \times 2 \times 2 \times 2 \times (1 + 2 + 2^2 + \cdots + 2^{10}) = 2^4(2^{11} - 1)^2 > 2^{24}$$

se os canais não forem limitados, $cap(c) = \infty$, o número de estados também é $\infty$.

**ABP**




$T(ABP)$

- como timer pode fazer *timout* de modo arbitrário o número de mensagens em $c$ pode ser infinito,
- pelo que $T(ABP)$ pode ser infinito

- fragmento de execução onde a mensagem é perdida

| sender $S$ | timer | receiver $R$ | channel $c$ | channel $d$ | event |
|---|---|---|---|---|---|
| $snd\ msg(0)$ | *off* | $wait(0)$ | $\varnothing$ | $\varnothing$ | |
| $st\ tmr(0)$ | *off* | $wait(0)$ | $\varnothing$ | $\varnothing$ | loss of message |
| $wait(0)$ | *on* | $wait(0)$ | $\varnothing$ | $\varnothing$ | |
| $snd\ msg(0)$ | *off* | $wait(0)$ | $\varnothing$ | $\varnothing$ | timeout |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | |

**Ignorar retransmissões**

| sender $S$ | timer | receiver $R$ | channel $c$ | channel $d$ | event |
|---|---|---|---|---|---|
| $snd\ msg(0)$ | *off* | $wait(0)$ | $\varnothing$ | $\varnothing$ | |
| $st\ tmr(0)$ | *off* | $wait(0)$ | $\langle m, 0\rangle$ | $\varnothing$ | message with bit 0 transmitted |
| $wait(0)$ | *on* | $wait(0)$ | $\langle m, 0\rangle$ | $\varnothing$ | |
| $snd\ msg(0)$ | *off* | $wait(0)$ | $\langle m, 0\rangle$ | $\varnothing$ | timeout |
| $st\ tmr(0)$ | *off* | $wait(0)$ | $\langle m, 0\rangle\ \langle m, 0\rangle$ | $\varnothing$ | retransmission |
| $st\ tmr(0)$ | *off* | $pr\ msg(0)$ | $\langle m, 0\rangle$ | $\varnothing$ | receiver reads first message |
| $st\ tmr(0)$ | *off* | $snd\ ack(0)$ | $\langle m, 0\rangle$ | $\varnothing$ | |
| $st\ tmr(0)$ | *off* | $wait(1)$ | $\langle m, 0\rangle$ | 0 | receiver changes into mode-1 |
| $st\ tmr(0)$ | *off* | $pr\ msg(1)$ | $\varnothing$ | 0 | receiver reads retransmission |
| $st\ tmr(0)$ | *off* | $wait(1)$ | $\varnothing$ | 0 | and ignores it |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | |

**Paralelismo síncrono**

- na representação de sistema asíncromnos por sistemas de transição não há a noção do tempo que demora cada ação
- Mas, no caso de circuitos sequênciais que funcionam sincronamente com (pulso de ) um relógio
- é possivel simular esta sincronia obrigando todas as ações a serem executadas simultaneamente (com uma periodicidade fixa).

**Produto Síncrono**

$T_i = (S_i, Act_i, \longrightarrow_i, I_i, AP_i, L_i)$ para $i = 1, 2$

$$\begin{aligned} * : Act_1 \times Act_2 &\to Act \\ (\alpha, \beta) &\mapsto \alpha * \beta \end{aligned}$$

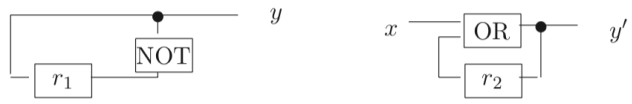
$$T_1 \otimes T_2 = (S_1 \times S_2, Act, \longrightarrow, I_1 \times I_2, AP_1 \cup AP_2, L)$$

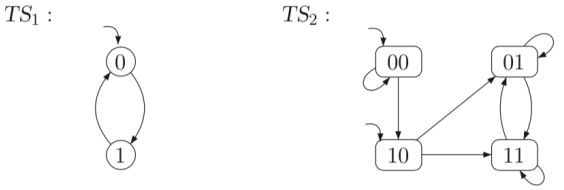
onde a relação de transição é definida pela regra

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \ \wedge \ s_2 \xrightarrow{\beta}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha * \beta} \langle s_1', s_2' \rangle}$$

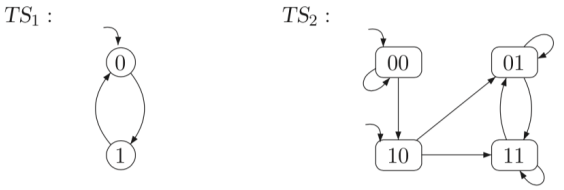
e $L(\langle s_1, s_2 \rangle) = L(s_1) \cup L(s_2)$.

**Produto Síncrono de Circuitos**

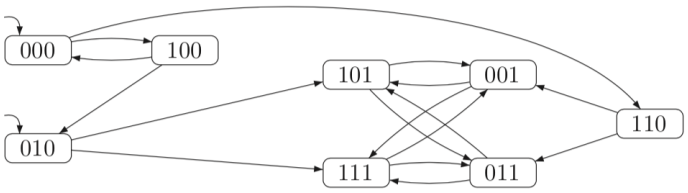


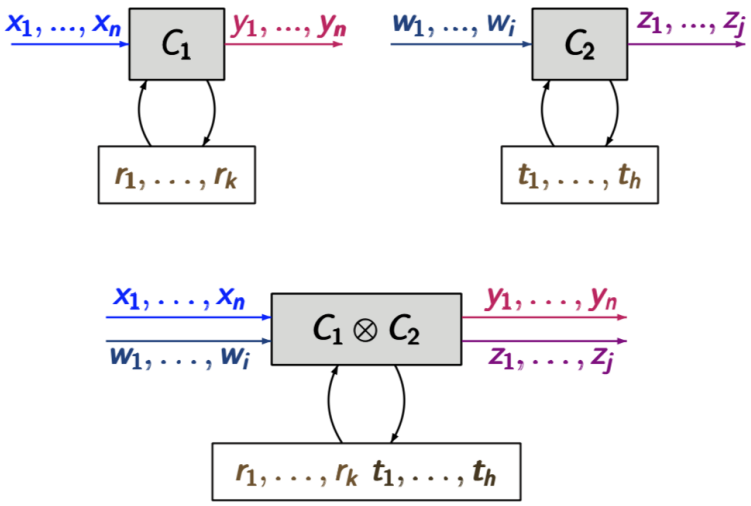

**Produto Síncrono de Circuitos**




$TS_1 \otimes TS_2$ :



**Produto Síncrono de Circuitos (geral)**




**Operadores paralelos (resumo)**

- concorrência pura $T_1|||T_2$
- passagem de mensagens síncrona $T_1||_{Syn}T_2$
- intercalagem de grafos de programa $P_1|||P_2$ (variáveis partilhadas)
- Sistemas de canais $[P_1|\ldots P_2]$ com
  - concorrentes, variáveis partilhadas
  - passagem de mensagens síncrona
  - passagem de mensagens assíncrona
- Paralelismo síncrono

**Problema da explosão do número de estados**

Um sistema de transições pode ser *muito grande*

- *infinito* se as variáveis tiverem dominios infinitos (p.e $\mathbb{N}$) ou forem estruras de dados infinitas (p.e. pilhas)
- *finitas* com um crescimento exponencial no número de componentes ou no número de variáveis e canais. P.e
- $|Loc_1|\cdots|Loc_2|\prod_{x\in Var}|dom(x)|.\prod_{c\in Chan}|dom(c)|^{cap(c)}$

- $L$ localizações por componente, $K$ canais de bits com capacidade $k$ e $M$ variáveis com $|dom(x)| \leq m$ o número de estados é

$$L \cdot m^M \cdot 2^{K \cdot k}$$

- Ex: $ABP$ se $cap(c) = cap(d) = 10$, $dom(c) = dom(m) = \{0, 1\}$ e $|Loc_T| = 2$, $|Loc_R| = 6$, $|Loc_S| = 8$ o número de estados é

$$2 \times 6 \times 8 \times 4^{10} \times (2^{11} - 1) > 3.2^{25}.$$

**Promela**

- "PROcess MEtaLAnguage"desenvolvida para
- SPIN Model checker `spinroot.com`
- em 2002 obteve o ACM System Software Award
- Um programa $\bar{P}$ é um conjunto de processos

$$P_1, \ldots P_n$$

  que executam concorrentemente com:

- partilha de variáveis
- canais FIFO ou síncronos
- para descrever cada processo usa-se uma *linguagem de comandos guardados*: instruções de linguagens imperativas, ações de comunicação e regiões atómicas
- comandos guardados: têm uma guarda e uma ação
- ver também [BA08]

**Subconjunto de instruções**

**proctype** *name*() {stmt}

$$\begin{array}{rcl} \mathsf{stmt} & ::= & \mathtt{skip} \mid x := \mathsf{expr} \mid c?x \mid c!\mathsf{expr} \mid \\ & & \mathsf{stmt}_1 \,;\, \mathsf{stmt}_2 \mid \mathtt{atomic}\{\mathsf{assignments}\} \mid \\ & & \textbf{if} \quad :: g_1 \Rightarrow \mathsf{stmt}_1 \quad \ldots \quad :: g_n \Rightarrow \mathsf{stmt}_n \quad \textbf{fi} \quad \mid \\ & & \textbf{do} \quad :: g_1 \Rightarrow \mathsf{stmt}_1 \quad \ldots \quad :: g_n \Rightarrow \mathsf{stmt}_n \quad \textbf{do} \end{array}$$

- as variáveis globais são declaradas fora dos processos

- as variáveis locais são declaradas nos processos
- $g_i$ são condições sobre as variáveis
- *skip* comando que termina num passo e não altera nada
- ; ou $\Rightarrow$ execução sequencial
- `atomic` passo que não pode ser intercalado com outros
- ver sem6antica operacional em [BKL08] (Cap. 2

**Seleção if-fi**

$$\textbf{if} \quad :: g_1 \Rightarrow \mathsf{stmt}_1 \quad \ldots \quad :: g_n \Rightarrow \mathsf{stmt}_n \quad \textbf{fi}$$

- escolha não determinística dos $stmt_i$ para os quais *a guarda $g_i$ se verifica* no estado actual.
- supõe-se que todo isso é executado num passo atómico.
- se nenhuma das guardas $g_1, \ldots, g_n$ se verificar no estado corrente o processo fica *bloqueado*.
- neste caso a execução de outros processos poderão desbloqueá-lo ou...
- notar que o **if-then-else** imperativo corresponde a

$$\textbf{if} \ :: g \Rightarrow \mathsf{stmt}_1 \ :: \neg g \Rightarrow \mathsf{stmt}_2 \ \textbf{fi},$$

- Mas aqui se existir **::else -¿** ele evita o bloqueio no caso de *todas* as guardas forem falsas.

**Repetição do-od**

$$\textbf{do} \quad :: g_1 \Rightarrow \mathsf{stmt}_1 \quad \ldots \quad :: g_n \Rightarrow \mathsf{stmt}_n \quad \textbf{do}$$

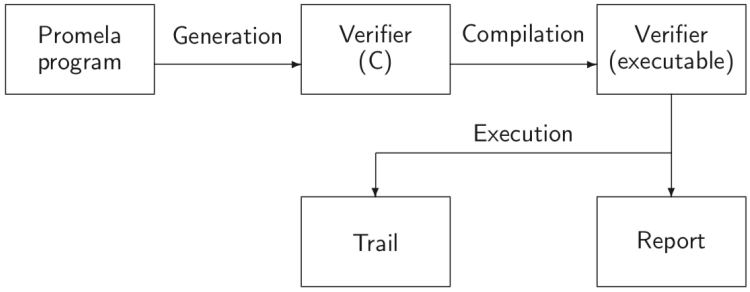
- Execução repetida da escolha não deterministica entre os comandos guardados cujas guardas sejam verdadeiras
- se todas as guardas forem falsas não bloqueia mas a repetição termina e a execução passa para o comando seguinte.
- também pode terminar com o comando **break**
- ou ter um comando **::else**
- existe também a *instrução de salto* **goto***label*, onde *label:* deve etiquetar a instrução que deverá ser executada em seguida.

**Verificação de programas sequênciais**

- Asserções que permitem verificar programas sequênciais
- **assert**(*cond*)
- um Model checker verifica todas as possíveis computações (nãodetermisticas)
- e se *cond* não se verificar para alguma delas é retornado um contra-exemplo.
- na verificação por dedução não será feito assim...

**Verificação com SPIN**

- Para ser mais eficiente o SPIN gera um "verificador"em linguagem C que depois de compilado permite executar a verificação.
- `spin -a max.pml; gcc -o pan pan.c; ./pan`
- Para analisar erros: `spin -t -p max.pml.trail`.




**Verificação de programas concorrentes**

Quando há mais que um processo.

- Em Promela:
  - Executar várias cópias:**active** [n] P()
  - **_pid** indica qual o processo a correr
  - **_nr_pr** indica o número de processos activos
  - em vez de **active**
  - primeiro processo: **init** { ... }
  - usar **run** para executar um processo especifico. Ex: **run**( P(1,5))

- no SPIN:
  - Execução aleatória:`spin pq.pml`
  - Execução interactiva: `spin -i pq.pml`
  - Construção de um verificador: `spin -a pq.pml`
  - ... ver outras opções

**Canais em Promela**

```
chan ch = [capacity] of { typename, ..., typename }
```

- que permite definir canais em que cada mensagem tem vários campos cada um com um dado tipo.
- em geral são declarados globalmente
- se forem locais "desaparecem"se o processo em que foram criados terminar
- podem ser passados com argumentos de processos
- nas ações de receber mensagens a variável pode ser "anónima"_ caso só interesse saber se algo foi recebido.
- pode haver *arrays* de canais: **chan** [2] = [3] of {**byte**, **bool**}
- **full**, **nfull**, **empty**, **nempty** funções Booleanas que testam o estado dos canais

# Referências


[BA08] Mordechai Ben-Ari. *Principles of the Spin Model Checker*. Springer, 2008.

[BKL08] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of Model Checking*. MIT Press, 2008.