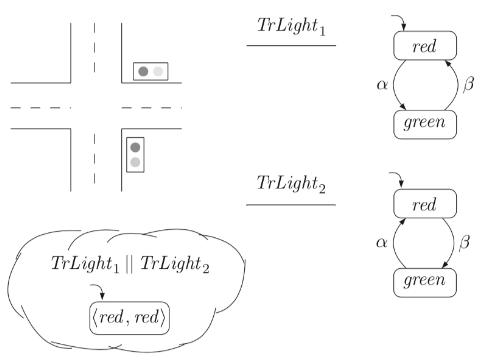
Propriedades temporais lineares dos sistemas de transição

Aula 7

Deadlock (ponto sem saída)

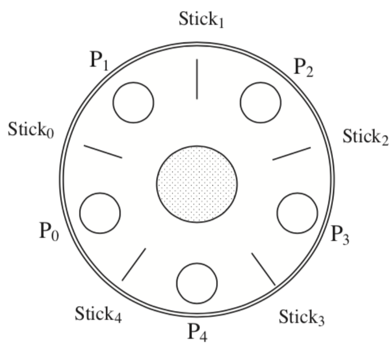
- Um estado terminal é um estado sem saída
- Ocorre naturalmente em programas sequências
- Mas em sistemas paralelos não são desejáveis e correspondem a um *erro*.
- Que em geral denotam um *Deadlock*
  - o sistema está num estado terminal
  - mas pelo menos uma componente está num estado (local) não terminal
- Ex: duas componentes estão mutuamente à espera uma da outra.

Deadlock para semáforos de trânsito



A composição paralela $T_1||T_2$ não tem transições do estado inicial (red, red). Só há as transições $(red, green) \xrightarrow{\alpha} (green, red)$ e $(green, rd) \xrightarrow{\beta} (red, green)$.

Jantar dos filósofos

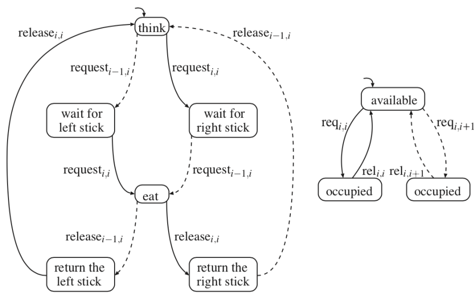


- 5 Filósofos, P_i para $i = 0, \dots 4$
- 5 pauzinhos, S_i para $i = 0, \dots 4$
- 1 taça de arroz central
- comem ou pensam
- para comer necessitam de dois pauzinhos:
- para P_i , S_i (à esquerda) e S_{i-1} (à direita)
- para P_{i+1} , S_{i+1} (à esquerda) e S_i (à dieita)

Jantar dos filósofos

- Se todos os filósofos tentarem comer ao mesmo tempo cada um só tem pauzinho: *Deadlock*.
- *Objectivo:* desenhar um protocolo que permita que todos os filósofos comam...embora tenham de esperar.
- Pretende-se que
  - (*deadlock freedom*) pelo menos um P coma e pense um número infinito de vezes (n.i.d.v)
  - (*individual freedom*) uma solução justa (*fair*) deve garantir que cada P_i coma e pense um número infinito de vezes

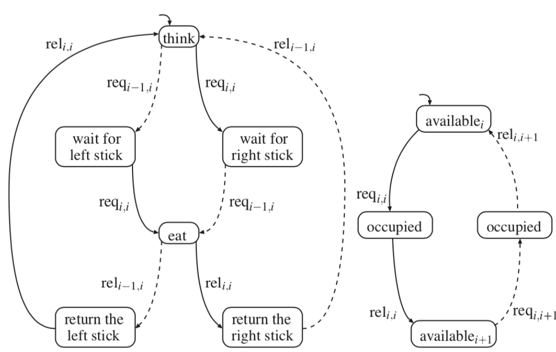
Primeira tentativa



$$P_4||S_3||P_3||S_2||P_2||S_1||P_1||S_0||P_0||S_4$$

- *Deadlock* se todos os P_i requisitarem S_i (à esquerda)
- de $\langle t4, a_3, t_3, a_2, t_2, a_1, t_1, a_0, t_0, a_4\rangle$
- pela sequência de ações $req_{4,4}, req_{3,3}, req_{2,2}, req_{1,1}, req_{0,0}$ chega-se a *Estado terminal:* $\langle w_{3,4}, o_{3,3}, w_{2,3}, o_{2,2}, w_{1,2}, o_{1,1}, w_{0,1}, o_{0,0}\rangle$

Segunda tentativa



Solução: não permitir que P_i e P_{i+1} requisitem um mesmo pauzinho. No estado inicial os ímpares começam com $available_{i,i}$ e os pares $available_{i,i+1}$.

Sistema de Transições

Seja $T = (S, Act, \longrightarrow, I, AP, L)$. Para $s \in S$ (ou $C \subseteq S$) e $\alpha \in Act$, *Post* é o conjunto dos sucessores directos de α e Pre o dos predecessores:

$$\begin{aligned}
Post(s,\alpha) &= \{s' \in S \mid s \xrightarrow{\alpha} s'\} \\
Pre(s,\alpha) &= \{s' \in S \mid s' \xrightarrow{\alpha} s\} \\
Post(C,\alpha) &= \bigcup_{s\in C} Post(s,\alpha) \\
Pre(C,\alpha) &= \bigcup_{s\in C} Pre(s,\alpha) \\
Post(s) &= \bigcup_{\alpha\in Act} Post(s,\alpha) \\
Pre(s) &= \bigcup_{\alpha\in Act} Pre(s,\alpha) \\
Post(C) &= \bigcup_{\alpha\in Act} Post(C,\alpha) \\
Pre(C) &= \bigcup_{\alpha\in Act} Pre(C,\alpha)
\end{aligned}$$

Análise baseada em estados

- $T = (S, Act, \longrightarrow, I, AP, L)$ sistema de transições
- Act, modela interação/comunicação
- AP, especificação de propriedades
- Abstraindo as ações Act temos um grafo de estados

$$G_T = (S, \{\ (s,s') \in S \times S \mid s' \in Post(s)\ \})$$

- os nós são os estados
- as arestas transições sem etiquetas
- $Post^*(s) = \{s' \in S \mid s \longrightarrow s'\}$
- $Pre^*(s) = \{s \in S \mid s \longrightarrow s'\}$

Fragmentos de caminhos

- dado um fragmento de execução finito ou infinito

$$\begin{aligned}
\rho &= s_0\alpha_1 s_1\alpha_2 s_2 \ldots \alpha_n s_n \\
\rho_1 &= s_0\alpha_1 s_1\alpha_2 s_2 \ldots
\end{aligned}$$

- um *fragmento de um caminho* é a projeção nos estados

$$\begin{aligned}\hat{\pi} &= s_0 s_1 s_2 \ldots s_n \\ \pi &= s_0 s_1 s_2 \ldots\end{aligned}$$

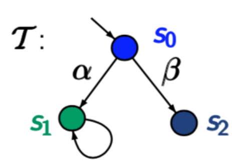
- *inicial* se $s_0 \in I$
- *maximal* se infinito ou termina num estado terminal
- *caminho* se inicial e maximal
- *caminho* de s: maximal e começa em s
- $Paths(T)$, conjunto de caminhos e $Paths(s)$ caminhos de s
- $Paths_{fin}(s)$ fragmentos finitos de caminho que começam em s

Notação

$$\begin{aligned}\hat{\pi} &= s_0 s_1 s_2 \ldots s_n \\ \pi &= s_0 s_1 s_2 \ldots\end{aligned}$$

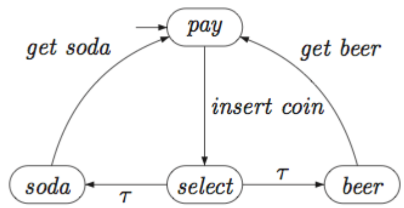
- $\pi[j] = s_j$ ou $\hat{\pi}[j] = s_j$
- $\pi[\ldots j] = s_0 \cdots s_j$ ou $\hat{\pi}[\ldots j] = s_0 \cdots s_j$
- $\pi[j \ldots] = s_j s_{j+1} \cdots$ ou $\hat{\pi}[j \ldots] = s_j s_{j+1} \cdots s_n$
- $first(\pi) = s_0$ e $first(\hat{\pi}) = s_0$
- $last(\pi) = \perp$ e $last(\hat{\pi}) = s_n$
- $len(\pi) = \infty$ e $len(\hat{\pi}) = n$

Caminhos



- Quantos caminhos tem? 2
- $Paths(T) = \{s_0 s_1 s_1 \ldots, s_0 s_2\} = \{s_0^\omega, s_0 s_2\}$
- $Paths(s_1) = \{s_1 s_1 s_1 \ldots\} = \{s_1^\omega\}$
- $Paths_{fin}(s_1) = \{s_1^n \mid n \geq 1\}$

Máquina de Vendas



$$\begin{array}{rcl} \pi_1 & = & payselectsodapayselectsoda\cdots \\ \pi_2 & = & selectsodapayselectsoda\cdots \\ \pi_3 & = & payselectsodapayselectsoda \end{array}$$

- π_1 é um caminho
- π_2 é maximal mas não inicial
- π_3 não termina nume estado terminal

Traços

- $T = (S, Act, \longrightarrow, I, AP, L)$ sistema de transições
- Em vez de sequências de estados considerar a sequência dos conjuntos de proposições atómicas válidas nos estados

$$\begin{array}{rcl} \rho & = & s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \cdots \\ \pi & = & s_0 s_1 s_2 \ldots \\ trace(\pi) & = & L(s_0)L(s_1)L(s_2)\ldots \\ \rho & = & s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \cdots s_n \\ \hat{\pi} & = & s_0 s_1 s_2 \ldots s_n \\ trace(\hat{\pi}) & = & L(s_0)L(s_1)L(s_2)\cdots L(s_n) \end{array}$$

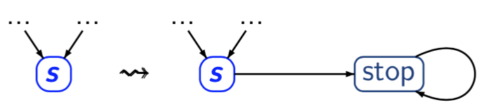
- Um traço é uma sequência de conjuntos de proposições atómicas
- isto é uma palavra de 2^{AP} que pode ser finita de $(2^{AP})^+$ ou infinita de $(2^{AP})^\omega$

Traços de sistemas de transições

- Π conjunto de caminhos, $trace(\Pi) = \{trace(\pi) \mid \pi \in \Pi\}$.
- $trace(s) = \{trace(\pi) \mid s = first(\pi)\}$
- $Traces(s) = trace(Paths(s))$
- $Traces(T) = \bigcup_{s \in I} Traces(s)$

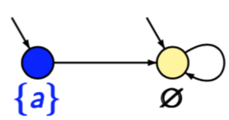
Sistemas de transições sem estados terminais

- *Normalmente vamos evitar estados terminais*
- i.e, não vamos ter sequências finitas (sejam elas, execuções, caminhos ou traços)
- então um traço é uma palavra de $(2^{AP})^\omega$, i.e
  $Traces(T) \subseteq (2^{AP})^\omega$
- Para *eliminar* estados terminais:
  - pré-calculámos esses estados
  - se forem aceitáveis completamos o sistema com um estado "morto"



  - se for um *deadlock* (ou outro erro), temos de corrigir o sistema.

Sistemas de transições sem estados terminais



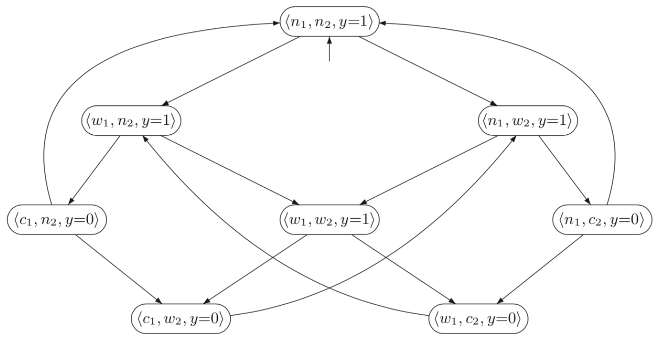
- $Traces(T) = \{\{a\}\emptyset^\omega, \emptyset^\omega\}$
- $Traces_{fin}(T) = \{\{a\}\emptyset^n \mid n \geq 0\} \cup \{\emptyset^m \mid m \geq 1\}$

Em geral

$$\begin{aligned} Traces(T) &= \{trace(\pi) \mid \pi \in Paths(T)\} \subseteq (2^{AP})^\omega \\ Traces_{fin}(T) &= \{trace(\hat{\pi}) \mid \hat{\pi} \in Paths_{fin}(T)\} \subseteq (2^{AP})^\star \end{aligned}$$

onde π caminho e $\hat{\pi}$ fragmento finito de caminho.

Exclusão mútua com semáforo



- Suponhamos $AP = \{c_1, c_2\}$

$$\begin{array}{rcl} \pi & = & \langle n_1, n_2, y1\rangle\langle w_1, n_2, y1\rangle\langle c_1, n_2, y0\rangle\langle n_1, n_2, y1\rangle \\ & & \langle n_1, w_2, y1\rangle\langle n_1, c_2, y0\rangle\langle n_1, n_2, y1\rangle \dots \\ trace(\pi) & = & \emptyset\emptyset\emptyset\{c_1\}\emptyset\emptyset\{c_2\}\emptyset \dots \end{array}$$

Propriedades Temporais Lineares (LT)

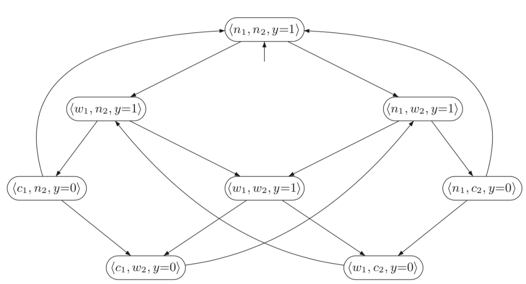
- $T = (S, Act, \longrightarrow, I, AP, L)$ sistema de transições sem estados terminais sobre AP
- caracterizam os traços desejáveis em T
- uma propriedade LT é uma especificação dos conjuntos de palavras infinitas sobre 2^{AP} que são admissiveis. Então
- uma propriedade LT é uma linguagem $P \subseteq (2^{AP})^\omega$
- Relação de satisfação (T *satisfaz* P)

$$T \models P \text{ sse } Traces(T) \subseteq P$$

- $s \in S$ satistaz P

$$s \models P \text{ sse } Traces(s) \subseteq P$$

Exclusão Mútua



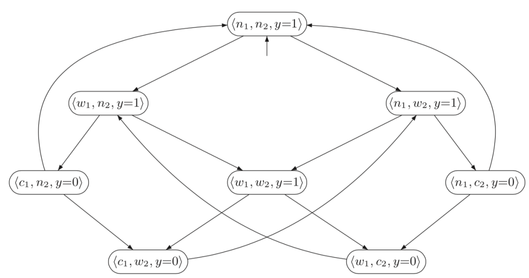
$AP = \{w_1, w_2, c_1, c_2\}$

- Propriedade da exclusão mútua (*segurança*)

$$P_{MUTEX} \quad = \quad \{A_0A_1A_2\ldots \mid \{c_1, c_2\} \not\subseteq A_i,\ i \geq 0\}$$

- $T_{Sem} \models P_{MUTEX}$

***Starvation freedom* (*Vivacidade*)**



$$P_{notstarve} \quad = \quad \{A_0A_1A_2\ldots \mid (\overset{\infty}{\exists}\ j.\ w_i \in A_j) \rightarrow (\overset{\infty}{\exists}\ j.\ c_i \in A_j)\}$$

- onde $\overset{\infty}{\exists}\ j$ significa *existe um número infinito de j's* ,i.e, $\forall k \geq 0 \exists j \geq k$
- mas $\emptyset(\{w_2\}\{w_1, w_2\}\{c_1, w_2\})^\omega \in Traces(T_{Sem})$
- Logo, $T_{Sem} \not\models P_{notstrave}$
- Mas $T_{Pet} \models P_{notstrave}$ (verifica)

Propriedades de Segurança (***Safety***)

- *Nada de mau deve acontecer*
- Ex: exclusão mútua, deadlocks
- podem ser propriedades
  - que se têm de verificar em cada estado (invariantes)
  - que se têm de verificar em fragmentos finitos de caminhos

Invariantes

- Invariante é uma propriedade Φ que se tem de verificar em todos os estados atinguíveis

$$P_{inv} \quad = \quad \{A_0A_1A_2\ldots \in (2^{AP})^\omega \mid \forall j \geq 0.\ A_j \models \Phi\}$$

- supomos Φ uma fórmula da lógica proposicional sobre AP
- $T \models P_{inv}$ sse $L(s) \models \Phi$ para todo $s \in Reach(T)$
- i.e. $L(s_0) \models \Phi$ e se $L(s) \models \Phi$ e $s \xrightarrow{\alpha} s'$ então $L(s') \models \Phi$
- P_{MUTEX} é um invariante com $\Phi = \neg c_1 \vee \neg c_2$
- Para verificar um invariante usa-se um DFS em T e caso falhe pode-se obter um fragmento de caminho $\pi = s_0 \ldots s_n$ tal que $s_i \models \Phi$ mas $s_n \not\models \Phi$.
- complexidade é $O(N(1+|\Phi|)+N)$ onde $N = |Reach(T)|$ e $M = \sum_{s \in S} |Post(s)|$ (numero de transições). Nota que $s \models \Phi$ corresponde a saber se uma fórmula proposicional se verifica para uma dada valorização.

Propriedades de Segurança (*Safety*)

- Exemplo de uma máquina ATM:

  > O dinheiro só deve ser disponibilizado depois do PIN ter sido verificado correctamente

- não é uma propriedade de estado
- uma execução infinita que viola este requisito tem um prefixo que é *mau*:
- neste caso disponibilizar o dinheiro sem verificar o PIN
- Dado $\pi = s_0 s_1 s_2 s_3 \ldots \in S^\omega$ o seu conjunto de prefixos finitos é $pref(\pi) = \{\pi[\ldots j] \mid j \geq 0\}$
- Uma propriedade LT P_{safe} é de *segurança* se

$$\forall \sigma \in (2^{AP})^\omega \setminus P_{safe} \exists \hat{\sigma} \in pref(\sigma),$$
$$P_{safe} \cap \{\sigma' \in (2^{AP})^\omega \mid \hat{\sigma} \in pref(\sigma')\} = \emptyset$$

- $\hat{\sigma}$ é o *prefixo mau* para P_{safe}

Prefixos maus de uma P_{safe}

- $BadPref(P_{safe})$ é o conjunto de prefixos maus de P_{safe}
- Um prefixo mau é *minimal* se nenhum menor for mau
- $MinBadPref(P_{safe})$ conjunto de prefixos minimais de P_{safe}
- Um invariante P_{inv} é uma propriedade de segurança
  - Seja Φ a condição de P_{inv}, todas as palavras finitas $A_0 \ldots A_n \in (2^{AP})^+$ tal que $A_i \models \Phi$ para $0 \leq i \leq n-1$ e $A_n \not\models \Phi$ são prefixos maus minimais para P_{inv}

Semáforos de trânsito

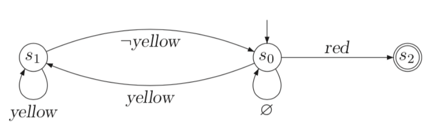
- A propriedade P_1
- *cada vermelho é precedido imediatamente por um amarelo*
- é P_{safe} mas não P_{inv}
- Seja $AP = \{r, g, y\}$
- $P_2 = \{A_0 A_1 \ldots \in (2^{AP})^\omega \mid A_j \neq \emptyset\}$
- i.e *pelo menos uma luz está acessa em cada momento*
- prefixos maus de P_1 são prefixos que contêm $\emptyset$
- prefixos mínimos maus de P_1 são prefixos que terminam em $\emptyset$
- $P_3 = \{A_0 A_1 \ldots \in (2^{AP})^\omega \mid |A_j| \leq 1\}$
- i.e *nunca estão duas luzes acessas ao mesmo tempo*
- prefixos maus são palavras finitas que têm $\{r, g\}$, $\{r, y\}$ ou $\{g, y\}$
- e as minimais terminam nesses conjuntos.

Semáforos de trânsito

- Seja $AP' = \{r, y\}$, então $P_1 = \{A_0 A_1 \ldots \mid A_i \subseteq AP' \wedge i \geq 0 \wedge (r \in A_i \rightarrow y \in A_{i-1})\}$
- os prefixos maus são as palavras finitas que violam esta condição.
- p.e $\emptyset\emptyset\{r\}$ ou $\{y\}\{y\}\{r\}\emptyset\{r\}$

$BadPref(P_{safe})$ **podem ser linguagens regulares**

- $BadPref(P_{safe})$ são linguagens de palavras finitas.
- Logo se forem regulares podem exprimir-se por um autómato finito ou uma expressão regular.
- P_1 é regular e uma autómato sobre $\{r, y\}$ para os prefixos minimais é



- onde $\neg y$ corresponde a $\{r\}$ ou $\emptyset$.
- $MinBadPref(P_2) = (2^{AP})^* \emptyset$
- $MinBadPref(P_3) = (2^{AP})^*(\{r, g\} + \{r, y\} + \{g, y\})(2^{AP})^*$

Máquina de vendas

- *o número de moedas é sempre pelo menos igual ao número de bebidas fornecidas*
- Seja $AP \supseteq \{pay, drink\}$

$$\begin{aligned} P_4 \quad = \quad & \{A_0 A_1 A_2 \ldots \mid \forall i \geq 0, \\ & |\{0 \leq j \leq i \mid pay \in A_j\}| \geq |\{0 \leq j \leq i \mid drink \in A_j\}|\} \end{aligned}$$

- isto é, qualquer prefixo de um caminho tem de verificar a propriedade
- $\emptyset\{\, pay \,\}\{\, drink \,\}\{\, pay \,\}\{\, drink \,\}\{\, drink \,\}$ é mau
- e P_4 não é regular

Propriedades de Segurança Regulares

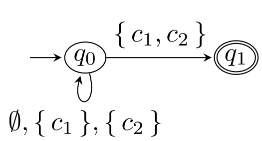
- P_{safe} é regular se $BadPref(P_{safe})$ é regular
- o que é equivalente a $MinBadPref(P_{safe})$ ser regular
- um invariante P_{inv} com condição Φ é sempre regular:
  - Seja $Sat(\Phi) = \{\ A \subseteq AP \mid A \models \Phi\ \}$ e

$$BadPref(P_{inv}) \quad = \quad Sat(\Phi)^\star(\neg Sat(\Phi))(2^{AP})^\star$$

- para $AP = \{\, a, b \,\}$ e $\Phi = a \vee \neg b$
- $Sat(\Phi) = \{\{a\}, \{a, b\}, \emptyset\}$
- $\neg Sat(\Phi) = \{\, \{\, b \,\} \,\}$
- $2^{AP} = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$

Exclusão Mútua

- $\Phi_{MUTEX} = \neg(c_1 \wedge c_2)$
- $\neg Sat(\Phi_{MUTEX}) = \{\, \{\, c_1, c_2 \,\} \,\}$
- $BadPref(P_{MUTEX}) = \{\, A_0 A_1 \ldots A_n \mid \exists 0 \leq i \leq n,\ \{\, c_1, c_2 \,\} \in A_i \,\}$
- $MinBadPref(P_{MUTEX}) = \{\, A_0 A_1 \ldots A_n \mid \{\, c_1, c_2 \,\} \in A_n \wedge \exists 0 \leq j < n,\ \{\, c_1, c_2 \,\} \notin A_j \,\}$
- que tem o seguinte NFA sobre alfabeto $2^{\{\, c_1, c_2 \,\}}$:



Relação de satisfação para P_{safe}

- $T \models P_{safe}$ sse $Trace_{fin}(T) \cap BadPref(P_{safe}) = \emptyset$
- $\Leftarrow$
- Seja $Trace_{fin}(T) \cap BadPref(P_{safe}) = \emptyset$ e suponhamos que $T \not\models P_{safe}$.
- Então existe $\pi \in Paths(T)$ com $trace(\pi) \notin P_{safe}$
- Então, π começa com um prefixo mau $\hat{\sigma}$ de P_{safe}.
- Mas então $\hat{\sigma} \in Trace_{fin}(T) \cap BadPref(P_safe)$
- o que é absurdo

Relação de satisfação para P_{safe}

- $T \models P_{safe}$ sse $Trace_{fin}(T) \cap BadPref(P_{safe}) = \emptyset$
- $\Rightarrow$
- Suponhamos $T \models P_{safe}$ e $Trace_{fin}(T) \cap BadPref(P_{safe}) \neq \emptyset$
- Existe $\hat{\sigma} = A_1 \ldots A_n \in Trace_{fin}(T) \cap BadPref(P_{safe})$
- que pode ser estendido a $\sigma = A_1 \ldots A_n A_{n+1} \ldots \in Trace(T)$.
- Então $\sigma \notin P_{safe}$
- e portanto $T \not\models P_{safe}$
- o que é absurdo

Fecho de P

- Seja $\sigma \in (2^{AP})^\omega$
- $pref(P) = \bigcup_{\sigma \in P} pref(\sigma)$
- O fecho de uma propriedade LT P é

$$closure(P) \quad = \quad \{\, \sigma \in (2^{AP})^\omega \mid pref(\sigma) \subseteq pref(P) \,\}$$

- Temos que
- P é uma propriedade de segurança se e só se $closure(P) = P$

Propriedades de Vivacidade/Progresso (*Liveness*)

- *Algo de bom irá acontecer*
- as propriedades de segurança são violadas em execuções finitas
- as propriedades de progresso só são violadas em execuções infinitas
- uma propriedade LT P_{live} é de vivacidade/progresso se

$$pref(P_{live}) \quad = \quad (2^{AP})^{\star}$$

- i.e., cada palavra finita pode ser estendida a (é um prefixo de) a uma palavra infinita que satisfaz P.
- i.e. para todos $x \in (2^{AP})^{\star}$ existe $\sigma \in (2^{AP})^{\omega}$ tal que $x\sigma \in P$.

Exclusão mútua

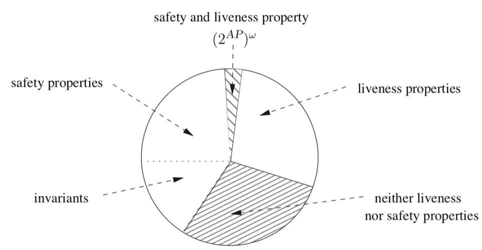
1. *cada processo irá entrar na região crítica*
2. *cada processo irá entrar na região crítica um número infinito de vezes*
3. *cada processo em espera irá entrar na região crítica*

$$\begin{array}{rcl}
AP & = & \{\, w_1, c_1, w_2, c_2 \,\} \\
P_1 & = & \{\, A_0A_1 \ldots \mid A_j \subseteq AP \wedge (\exists j \geq 0 c_1 \in A_j) \wedge (\exists j \geq 0 c_2 \in A_j) \,\} \\
P_2 & = & \{\, A_0A_1 \ldots \mid A_j \subseteq AP \wedge (\overset{\infty}{\exists}\, j \geq 0.\, c_1 \in A_j) \wedge (\overset{\infty}{\exists}\, j \geq 0.\, c_2 \in A_j) \,\} \\
P_3 & = & \{\, A_0A_1 \ldots \mid A_j \subseteq AP \wedge \forall j \geq 1 (w_1 \in A_j \rightarrow \exists k > j.\, c_1 \in A_j) \wedge \\
 & & \forall j \geq 1 (w_2 \in A_j \rightarrow \exists k > j.\, c_2 \in A_j) \,\}
\end{array}$$

P_1, P_2, P_3 são propriedades de progresso/vivacidade (verifica!)

Segurança e Vivacidade (*safety/liveness*)

- As propriedades de segurança e vivacidade são disjuntas?*Sim...*
- Todas as propriedades LT são de uma dessas classes? *Não*
- E qualquer propriedade LT , P é equivalente a uma propriedade P' que é a interseção duma propriedade de segurança com uma de vivacidade.



$P_{safe} \cap P_{live} = (2^{AP})^\omega$

- Suponhamos P uma propriedade de vivacidade sobre AP.
- Por definição, $pref(P) = (2^{AP})^\star$
- Então $closure(P) = \{ \sigma \in (2^{AP})^\omega \mid pref(\sigma) \subseteq pref(P) \} = (2^{AP})^\omega$
- Se P é também uma propriedade de segurança então $closure(P) = P$
- e portanto $P = (2^{AP})^\omega$.

Teorema da Decomposição

- Para toda a propriedade LT P sobre AP, existe P_{safe} e existe P_{live} tal que
- $P = P_{safe} \cap P_{live}$
- para tal prova-se que
- $P_{safe} = closure(P)$ (verifica!)
- e $P_{live} = P \cup (2^{AP})^\omega \setminus closure(P)$ (verifica!)
- e como por definição $P \subseteq closure(P)$ então
- $P = closure(P) \cap P$ e é evidente que $P = P_{safe} \cap P_{live}$

Justeza (*Fairness*)

- as estratégias para lidar com o não determinismo devem ser realistas, i.e. *justas*
- para tal impõe-se condições nos comportamentos infinitos de modo a permitir a verificação de propriedades de vivacidade

- Ex: Dado um servidor e N clientes (p_1,..., p_N) que requerem o um serviço
- O servidor pode começar por p_1, depois p_2, etc. e servir o primeiro cliente que requeira o serviço
- Quando terminar se servir esse cliente, volta ao procedimento de seleção verificando de novo tela ordem p_1, p_2, etc.
- Se p_1 estiver sempre a requerer o serviço, mais nenhum cliente é servido
- esta é uma estratégia *injusta*

Restrições de Justeza

Não-condicional ex: *todos os processos têm a sua vez na região crítica um n.i.d.v*

Justeza forte *cada processo que requer entrar na região crítica pode fazê-fo um n.i.d.v*

Justeza Fraca *um processo que esteja continuamente pronto a partir de certa altura tem a sua vez um n.i.d.v*

onde *n.i.d.v*=número infinito de vezes

Justeza baseada em ações

- Dado $T = (S, Act, \longrightarrow, AP, I, L)$
- $Act(s) = \{\, \alpha \in Act \mid \exists s' \in S.\, s \xrightarrow{\alpha} s' \,\}$, ações executáveis em s
- Seja $A \subseteq Act$
- $\rho = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots,$
- ρ é *incondicionalmente* A-justo se

$$\overset{\infty}{\exists}\, j.\, \alpha_j \in A.$$

- ρ é *fortemente* A-justo se

$$(\overset{\infty}{\exists}\, j.\, Act(s_j) \cap A \neq \emptyset) \rightarrow \overset{\infty}{\exists}\, j.\, \alpha_j \in A.$$

- ρ é *fracamente* A-justo se

$$(\overset{\infty}{\forall}\, j.\, Act(s_j) \cap A \neq \emptyset) \rightarrow \overset{\infty}{\exists}\, j.\, \alpha_j \in A.$$

- onde $\overset{\infty}{\forall}\, j$ significa todos j excepto um número finito.

Relação entre Tipos de Justeza

- Se ρ é *incondicionalmente* A-justo alguma ação de A é executada um n.i.d.v
- ρ é *fortemente* A-justo se as ações de A não são continuamente ignoradas caso possam ser executadas um n.i.d.v
- ρ é *fracamente* A-justo se uma ação de A pode ser executada quase então alguma ação de A é executada um n.i.d.v
- incondicionalmente A-justo $\Rightarrow$ fortemente $\Rightarrow$ fracamente

Assunção de Justeza

- uma condição de justeza impõe um requisito a todas as condições dum conjunto A
- para se poder impor condições diferentes tem de se considerar conjuntos de conjuntos de ações para cada tipo de justeza
- $F = (F_{uncond}, F_{strong}, F_{weak})$
- onde $F_{uncond}, F_{strong}, F_{weak} \subseteq 2^{Act}$.
- Diz que uma execução é F-justa se se verificam todas as condições.
- esta noção estende-se a caminhos e traços (sendo designados por F-caminhos ou F-traços)

Relação de satisfazibilidade justa para propriedades LT

- $T = (S, Act, \longrightarrow, I, AP, L)$ sistema de transições
- $FairPaths_F(s)$ conjunto de F-caminhos de s
- Seja $FairTraces_F(s) = trace(FairPaths_F(s))$
- $FairTraces_F(T) = \bigcup_{s \in S} FairTraces_F(s)$
- Seja P uma propriedade LT e F uma assunção de justeza sobre Act, T satisfaz justamente P, i.e $T \models_F P$ se e só se $FairTraces_F(T) \subseteq P$.
- Nota: em geral as condições de justeza não afectam as propriedades de segurança

Referências