



VPC 19-20

Esercizio di model-checking con GreatSPN  
e NUSMV e esercizio di confronto con  
algebra dei processi

Prof.ssa Susanna Donatelli

Universita' di Torino

[www.di.unito.it](http://www.di.unito.it)

susi@di.unito.it



# Visione di insieme

---

Valuteremo la correttezza di diverse soluzioni per la mutua esclusione proposte dal libro di testo di M. Ben-Ari "Principles of Concurrent and Distributed Programming", cap. 3

Tutti gli algoritmi dovranno essere implementati in NuSMV e in GreatSPN, le tre proprietà del prossimo lucido devono essere definite in LTL e CTL

I primi due algoritmi devono essere sviluppati anche in algebra dei processi e si chiede di confrontare la soluzione a rete di Petri con quella di algebra dei processi usando le nozioni di equivalenza viste a lezione

Particolare attenzione a: progresso, fairness, consistenza dei risultati ottenuti per lo stesso algoritmo nei vari formalismi



# La mutua esclusione

---

Definizione del problema:

1. Ognuno degli N processi esegue un loop infinito di istruzioni divise in due gruppi: la sezione critica e la sezione non critica
2. La correttezza di un algoritmo di mutua esclusione è definita dalla congiunzione delle seguenti condizioni:
  - **1. Mutua esclusione:** le istruzioni delle sezioni critiche di due o più processi non possono essere eseguite in modo interfogliato
  - **2. Assenza di deadlock:** Se qualche processo cerca di accedere alla regione critica eventualmente un processo potrà farlo
  - **3. Assenza di starvation individuale:** Se un processo cerca di accedere alla regione critica eventualmente quel processo potrà farlo
3. Assumiamo che le variabili usate dal protocollo di accesso siano usate solo dal protocollo di accesso
4. C'è progresso nella regione critica (se un processo inizia l'esecuzione in regione critica alla fine terminerà tale esecuzione)
5. Non si richiede progresso da parte dei processi nelle istruzioni che non appartengono alla regione critica



# La mutua esclusione (3.2)

Prima soluzione (testo del Ben-Ari): una singola variabile turn, quando turn vale 1 entra il processo 1, quando turn vale due entra il processo 2. Può essere più semplice assumere che la variabile turn possa valere p o q anzichè 1 o 2

<b>Algorithm 3.2: First attempt</b>	
integer turn $\leftarrow$ 1	
<b>p</b>	<b>q</b>
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: await turn = 1	q2: await turn = 2
p3: critical section	q3: critical section
p4: turn $\leftarrow$ 2	q4: turn $\leftarrow$ 1

# Algoritmo 3.2 in CSP

- Composizione di tre processi: P, Q, variabile Turn
  - P e Q hanno un comportamento sequenziale
  - Ogni program counter diventa un nome di processo, e descriviamo con quale azione si passa da un valore di program counter ad un altro
  - Processi senza scelta
  - Await turn = x significa che il processo rimane bloccato sino a quando non si verifica turn = x

<b>Algorithm 3.2: First attempt</b>	
integer turn $\leftarrow$ 1	
<b>p</b>	<b>q</b>
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: await turn = 1	q2: await turn = 2
p3: critical section	q3: critical section
p4: turn $\leftarrow$ 2	q4: turn $\leftarrow$ 1

3.2 in CSP

$$(P \parallel Q) \parallel_S T_P$$

$$S = \{isp, isq, setp, setq\}$$

$$T_P ::= \underline{isp}. T_P + \underline{setp}. T_P + \underline{setq}. T_Q$$

$$\underline{T_Q} ::= isq. T_Q + \underline{setp}. T_P + \underline{setq}. T_Q$$

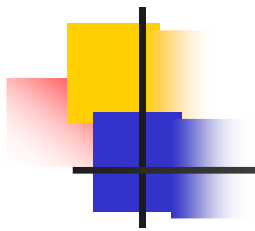
$$P ::= \underline{ncsp}. P_2 + \underline{ncsp}. \underline{P}$$

$$P_2 ::= \underline{isp}. P_3$$

$$P_3 = \underline{csp}. P_4$$

$$P_4 = \underline{setq}. P$$

La scelta non deterministica su ncsp – aggiunta del termine rosso - garantisce la condizione 5 (*Non si richiede progresso da parte dei processi nelle istruzioni che non appartengono alla regione critica*)



$$(P \parallel Q) \parallel_{S} T_P$$

$\swarrow$  MCS P

$\searrow$  MCS Q

$$P \parallel Q_2 \parallel T_P$$

$$\rightarrow (P_2 \parallel Q) \parallel T_P$$

$\swarrow$  IS P

$\searrow$  MCS Q

$$(P_3 \parallel Q) \parallel T_P$$

$$(P_2 \parallel Q_2) \parallel T_P$$

$\swarrow$  IS P

$$(P_3 \parallel Q_2) \parallel T_P$$

$\swarrow$  CS

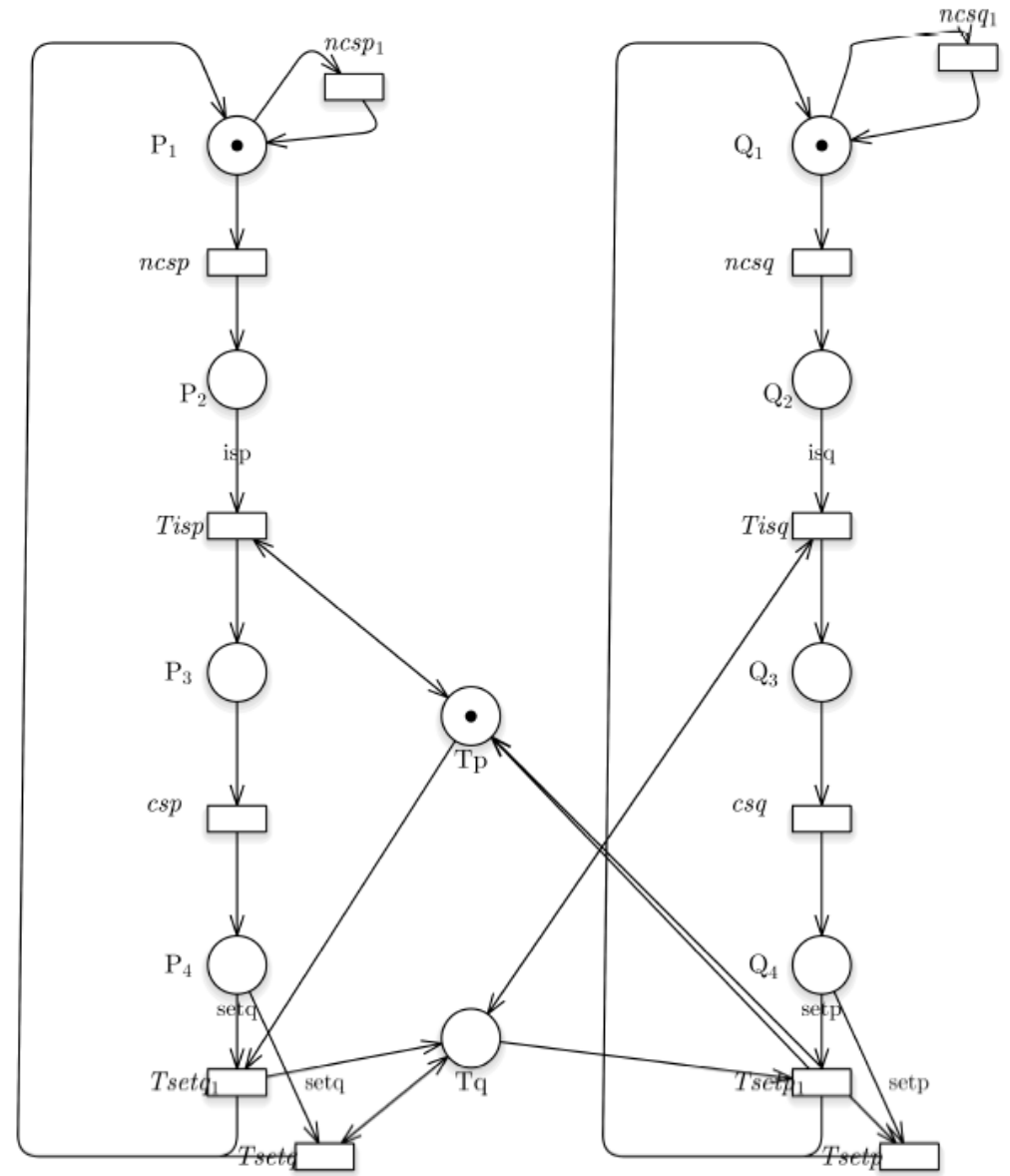
$\searrow$  MCS Q

$$(P_4 \parallel Q) \parallel T_P$$



Rete dell'alg. 3.2 ottenuta con i seguenti passi:

1. composizione parallela delle reti P (reti con posti di nome  $P_i$ ) e Q (reti con posti di nome  $Q_i$ ), con insieme di sincronizzazione vuoto
2. composizione della rete al punto 1 con la rete della variabile Turn (posti  $T_p$  e  $T_q$ )







# Analisi generale

---

**P-invarianti:** 3 p-semiflussi di peso 1, identificano i tre processi P, Q, Turn. Si deduce che tutti i posti sono 1-bounded, che la variabile Turn vale sempre o P oppure Q, e sempre esattamente uno dei due valori, e che ogni processo è sempre in esattamente un solo valore di program counter.

**T-invarianti:** 2 T-semiflussi, identificano due ccli, uno tutto di P e uno tutto di Q che, se effettivamente eseguibili dalla marcatura iniziale, riportano la rete nella marcatura iniziale stessa. Sono eseguibili?



# Analisi di proprietà su RG e model-checking

**Mutua esclusione:** le istruzioni delle sezioni critiche di due o più processi non possono essere eseguite in modo interfogliato

- Proposizioni atomiche:            ``P in regione critica" diventa #P4 ==1  
    ``Q in regione critica" diventa #Q4 ==1
- CTL: AG !(#P4 ==1 && #Q4 ==1)
- LTL: G !(#P4 ==1 && #Q4 ==1)

**Assenza di deadlock:** Se qualche processo cerca di accedere alla regione critica eventualmente un processo potrà farlo

- Proposizioni atomiche: "qualche processo cerca di accedere alla regione critica" diventa  
    #P2 ==1 || #Q2 == 1  
"qualche processo cerca di accedere alla regione critica" diventa  
    #P4 ==1 || #Q4 == 1
- CTL: ????
- LTL: ????

**3. Assenza di starvation individuale:** Se un processo cerca di accedere alla regione critica eventualmente quel processo potrà farlo (due proprietà, una per ogni processo)

- CTL: ???
- LTL: ???



# La mutua esclusione (3.5)

Il Ben-Ari propone anche una versione semplificata, minimale.

<b>Algorithm 3.5: First attempt (abbreviated)</b>	
integer turn $\leftarrow$ 1	
<b>p</b>	<b>q</b>
loop forever p1: await turn = 1 p2: turn $\leftarrow$ 2	loop forever q1: await turn = 2 q2: turn $\leftarrow$ 1



# La mutua esclusione (3.6)

Il Ben-Ari propone questa ulteriore soluzione, basata su due variabili.

<b>Algorithm 3.6: Second attempt</b>	
boolean wantp $\leftarrow$ false, wantq $\leftarrow$ false	
<b>p</b>	<b>q</b>
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: await wantq = false	q2: await wantp = false
p3: wantp $\leftarrow$ true	q3: wantq $\leftarrow$ true
p4: critical section	q4: critical section
p5: wantp $\leftarrow$ false	q5: wantq $\leftarrow$ false



# La mutua esclusione (3.8)

Il Ben-Ari propone anche questa terza soluzione, sempre basata su due variabili. Quest soluzione inverte le istruzioni di setting di wantp e di attesa su wantq (e viceversa per l'altro processo).

<b>Algorithm 3.8: Third attempt</b>	
boolean wantp $\leftarrow$ false, wantq $\leftarrow$ false	
<b>p</b>	<b>q</b>
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: wantp $\leftarrow$ true	q2: wantq $\leftarrow$ true
p3: await wantq = false	q3: await wantp = false
p4: critical section	q4: critical section
p5: wantp $\leftarrow$ false	q5: wantq $\leftarrow$ false

# La mutua esclusione (3.9)

Il Ben-Ari propone anche questa quarta soluzione, sempre basata su due variabili. Questa soluzione evita l' "intestardimento" dei processi nel voler entrare in regione critica, settando e resettando la propria variabile "want" per permettere all'altro processo di passare.

<b>Algorithm 3.9: Fourth attempt</b>	
boolean wantp $\leftarrow$ false, wantq $\leftarrow$ false	
<b>p</b>	<b>q</b>
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: wantp $\leftarrow$ true	q2: wantq $\leftarrow$ true
p3: while wantq	q3: while wantp
p4:     wantp $\leftarrow$ false	q4:     wantq $\leftarrow$ false
p5:     wantp $\leftarrow$ true	q5:     wantq $\leftarrow$ true
p6: critical section	q6: critical section
p7: wantp $\leftarrow$ false	q7: wantq $\leftarrow$ false

# La mutua esclusione (3.10)

La combinazione del primo e del quarto tentativo portano all'algorithmo di mutua esclusione noto come "algorithmo di Dekker".

<b>Algorithm 3.10: Dekker's algorithm</b>	
boolean wantp $\leftarrow$ false, wantq $\leftarrow$ false integer turn $\leftarrow$ 1	
<b>p</b>	<b>q</b>
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: wantp $\leftarrow$ true	q2: wantq $\leftarrow$ true
p3: while wantq	q3: while wantp
p4:     if turn = 2	q4:     if turn = 1
p5:         wantp $\leftarrow$ false	q5:         wantq $\leftarrow$ false
p6:         await turn = 1	q6:         await turn = 2
p7:         wantp $\leftarrow$ true	q7:         wantq $\leftarrow$ true
p8: critical section	q8: critical section
p9: turn $\leftarrow$ 2	q9: turn $\leftarrow$ 1
p10: wantp $\leftarrow$ false	q10: wantq $\leftarrow$ false





# Cosa dovete fare

---

Costruzione modello nuSMV e analisi delle proprietà 1, 2 e 3 per tutti gli algoritmi dati. Potete definire anche ulteriori proprietà per assicurarvi che il modello rispetti effettivamente gli algoritmi dati – usare LTL e CTL. Confrontare e giustificare i risultati sulla base della verifica delle proprietà e degli eventuali contro-esempi e witnesses.

Costruzione modello rete di Petri e analisi delle proprietà 1, 2 e 3 per tutti gli algoritmi dati. Potete definire anche ulteriori proprietà per assicurarvi che il modello rispetti effettivamente gli algoritmi dati. Usare il solver RGMEDD4 di GreatSPN. Confrontare e giustificare i risultati sulla base della verifica delle proprietà e degli eventuali contro-esempi e witnesses. Confrontare 3.6 con 3.8 usando tecniche di riduzione strutturale e calcolo di equivalenze assumendo come azioni visibili la richiesta della regine critica e la concessione della stessa.



# Cosa dovete fare

---

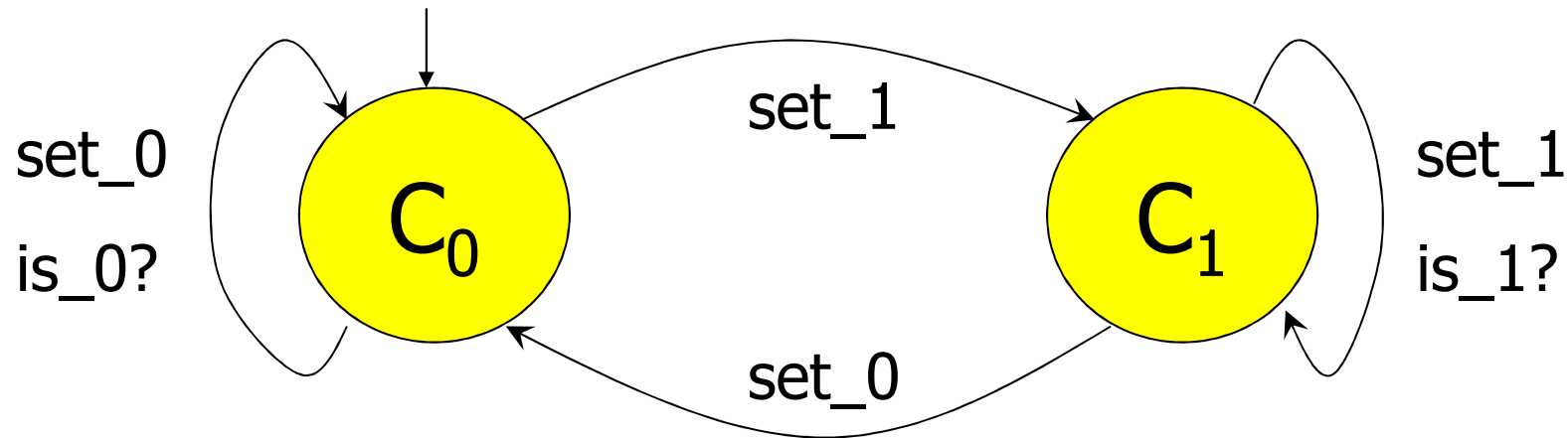
Costruzione modello ad algebra dei processi (CCS o CSP, a scelta) per i primi due algoritmi (3.2 e 3.6) e confronto con modello a rete di Petri calcolando le varie equivalenze fra il derivation graph e il reachability graph del 3.2 e fra il derivation graph e il reachability graph del 3.6. Confronto fra 3.2 e 3.6 in algebra dei processi usando le equivalenze: assumere che le azioni che non sono presenti in uno dei due algoritmi siano modellati da azioni non osservabili tau e, se serve, usare la versione di bisimulazione estesa a considerare la azioni tau (weak bisimulation).

Attenzione a fairness e alla corretta resa, nel modello, delle condizioni di progresso: c'è sicuramente progresso solo in regione critica e nel protocollo di accesso. Verificare, su almeno un modello, quali sono invece le conseguenze di richiedere il progresso in regione critica ma non nel protocollo di accesso.

Creare una tabella di confronto dei risultati ottenuti con NuSMV e con GreatSPN e commentare/giustificare eventuali discrepanze nei risultati o nei contro-esempi/witnesses

# Modeling binary variable

Suggerimento per la modellazione delle variabili in algebra  
dei processi



$$C_0 = \text{is\_0?} \cdot C_0 + \text{set\_1} \cdot C_1 + \text{set\_0} \cdot C_0$$

$$C_1 = \text{is\_1?} \cdot C_1 + \text{set\_0} \cdot C_0 + \text{set\_1} \cdot C_1$$