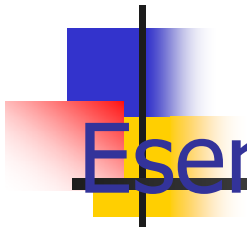
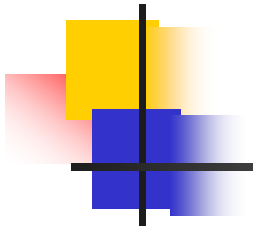


VPC 22-23
Esercizio di model-checking con GreatSPN e NUSMV e esercizio di confronto con algebra dei processi

Prof.ssa Susanna Donatelli
Universita' di Torino
www.di.unito.it
susi@di.unito.it

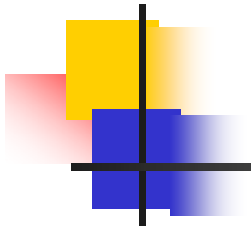


Visione di insieme

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

Gli algoritmi dovranno essere implementi in NuSMV e in GreatSPNAlcuni algoritmi devono essere sviluppati anche in algebra dei processi

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 potra' 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

Algorithm 3.2: First attempt

integer turn ← 1

p	q
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 ← 2	q4: turn ← 1

La mutua esclusione (3.6)

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

Algorithm 3.6: Second attempt

boolean wantp ← false, wantq ← false

p		q	
	loop forever		loop forever
p1:	non-critical section	q1:	non-critical section
p2:	await wantq = false	q2:	await wantp = false
p3:	wantp ← true	q3:	wantq ← true
p4:	critical section	q4:	critical section
p5:	wantp ← false	q5:	wantq ← 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).

Algorithm 3.8: Third attempt	
boolean wantp ← false, wantq ← false	
p	**q**
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: wantp ← true	q2: wantq ← true
p3: await wantq = false	q3: await wantp = false
p4: critical section	q4: critical section
p5: wantp ← false	q5: wantq ← false

La mutua esclusione (3.10)

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

Algorithm 3.10: Dekker's algorithm

```
boolean wantp ← false, wantq ← false
integer turn ← 1
```

p		q	
	loop forever		loop forever
p1:	non-critical section	q1:	non-critical section
p2:	wantp ← true	q2:	wantq ← true
p3:	while wantq	q3:	while wantp
p4:	if turn = 2	q4:	if turn = 1
p5:	wantp ← false	q5:	wantq ← false
p6:	await turn = 1	q6:	await turn = 2
p7:	wantp ← true	q7:	wantq ← true
p8:	critical section	q8:	critical section
p9:	turn ← 2	q9:	turn ← 1
p10:	wantp ← false	q10:	wantq ← false

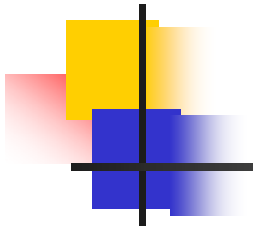
Suggerimenti per modello in CSP/CCS

- 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

Algorithm 3.2: First attempt

integer turn ← 1

p		q	
	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 ← 2	q4:	turn ← 1



Cosa dovete fare - nuSMV

Costruzione modello nuSMV e costruzione degli stati raggiungibili.

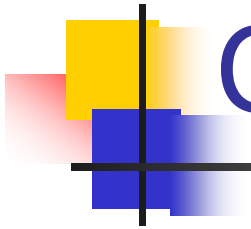
Analisi delle proprietà 1, 2 e. Potete definire anche ulteriori proprietà per assicurarvi che il modello rispetti effettivamente gli algoritmi dati – usare LTL, CTL e, nel caso, CTL*. Confrontare e giustificare i risultati sulla base della verifica delle proprietà e degli eventuali contro-esempi e witnesses.

9 cfu – laboratorio corto: applicazione agli algoritmi 3.2 e 3.10

6 cfu – laboratorio corto: applicazione agli algoritmi 3.2 e 3.10

9 cfu – laboratorio lungo : applicazione agli algoritmi 3.2, 3.6, 3.8 e 3.10

6 cfu – laboratorio lungo : applicazione agli algoritmi 3.2, 3.6, 3.8 e 3.10



Cosa dovete fare – reti di Petri

Costruzione modello a reti di Petri con GreatSPN in modalità composizionale.
Costruzione degli stati raggiungibili.

Calcolo e interpretazione di P-semiflussi (e relativi invarianti) e T-semiflussi.

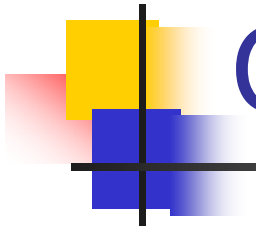
Analisi delle proprietà 1, 2 e. Potete definire anche ulteriori proprietà per assicurarvi che il modello rispetti effettivamente gli algoritmi dati – usare LTL, CTL e, nel caso, CTL*. Usare il solver StarMC Model Checker di GreatSPN. Confrontare e giustificare i risultati sulla base della verifica delle proprietà e degli eventuali contro-esempi e witnesses.

9 cfu – laboratorio corto: applicazione agli algoritmi 3.2 e 3.10

6 cfu – laboratorio corto: applicazione agli algoritmi 3.2 e 3.10

9 cfu – laboratorio lungo : applicazione agli algoritmi 3.2, 3.6, 3.8 e 3.10

6 cfu – laboratorio lungo : applicazione agli algoritmi 3.2, 3.6, 3.8 e 3.10

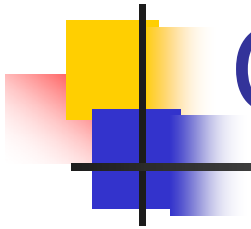


Cosa dovete fare – CCS/CSP

Costruzione modello ad algebra dei processi (CCS o CSP, a scelta) per gli algoritmi 3.6 e 3.8

Confronto fra i due usando le equivalenze. Scegliere quali azioni debbano essere mappate su azioni non osservabili tau e, se serve, usare la versione di bisimulazione estesa a considerare la azioni tau (weak bisimulation).

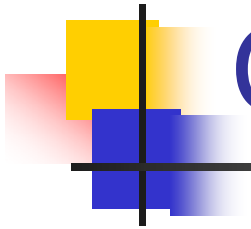
Solo per 9 cfu laboratorio lungo.



Cosa dovete fare – RdP

Applicare le regole di riduzione a 3.6 e 3.8. Verificare se le reti di Petri ottenute hanno un numero minore di stati (rispetto alle reti di partenza) e se, fra loro, hanno uguale struttura o uguale numero di stati. Calcolare liveness e assenza di deadlock e discutere se sia possibile verificare le proprietà 1, 2 e 3 anche nelle reti ridotte

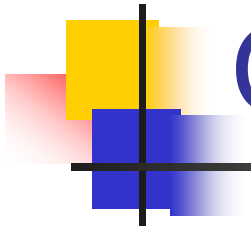
Solo per 9 cfu con laboratorio lungo.



Cosa dovete fare – RdP

Applicare le regole di riduzione a 3.6 e verificare se la rete di Petri ottenuta un numero minore di stati. Calcolare liveness e assenza di deadlock e discutere se sia possibile verificare le proprietà 1, 2 e 3 anche nella rete ridotta

Solo per 6 cfu con laboratorio lungo.



Cosa dovete fare – confronto

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

Inserire una tabella con (almeno) una riga per ogni algoritmo e pari colonne per nuSMV e GreatSPN.

Colonne:

- numero di stati raggiungibili (|RS|, output di print_reachable_states in NuSMV)
- presenza/assenza deadlock
- valore vero/falso delle tre proprietà

Attenzione alla 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.