



VPC 21-22

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

Tutti gli algoritmi dovranno essere implementi in NuSMV e in GreatSPN, le tre proprietà del prossimo lucido devono essere definite in LTL, CTL (e, nel caso, CTL fair o CTL*)

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

Attenzione quindi 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.



Cosa dovete fare - nuSMV

Costruzione modello nuSMV e costruzione degli stati raggiungibili per tutti gli algoritmi dati. Analisi della proprietà di assenza di deadlock e di mutual esclusione

Costruzione stati raggiungibili: comando `"print_reachable_states -v"`

Assenza di deadlock: check che la relazione successore sia totale (cioè per ogni stato esiste un successore: comando `"check_fsm"`)

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.



Cosa dovete fare – reti di Petri

Costruzione modello rete di Petri con GreatSPN 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. Usate il model checker di GreatSPN. Di default il model-checker non produce i controesempi. Witness e controesempi per il SOLO CTL, sono disponibili modificando il solver da "RGMEDD 5" a RGMEDD3" dal menu Edit/options/solvers.

Costruzione stati raggiungibili e check di assenza di deadlock;

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 – CCS/CSP

Costruzione modello ad algebra dei processi (CCS o CSP, a scelta) per i primi due algoritmi (3.2 e 3.6)

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).



Cosa dovete fare – confronto

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

Inserire una tabella con una riga per ogni algoritmo e tre colonne per nuSMV e tre colonne per GreatSPN.

Colonne:

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



Il problema e le proprietà

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 proprietà:
 - **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

Algorithm 3.2: First attempt

integer turn \leftarrow 1

p

loop forever

p1: non-critical section

p2: await turn = ~~1~~

p3: critical section

p4: turn \leftarrow ~~2~~

q

loop forever

q1: non-critical section

q2: await turn = ~~2~~

q3: critical section

q4: turn \leftarrow ~~1~~



La mutua esclusione (3.6)

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

Algorithm 3.6: Second attempt	
boolean wantp \leftarrow false, wantq \leftarrow 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 \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).

Algorithm 3.8: Third attempt	
boolean wantp \leftarrow false, wantq \leftarrow false	
p	q
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.

Algorithm 3.9: Fourth attempt	
boolean wantp \leftarrow false, wantq \leftarrow false	
p	q
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".

Algorithm 3.10: Dekker's algorithm	
boolean wantp \leftarrow false, wantq \leftarrow false integer turn \leftarrow 1	
p	q
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



Modeling binary variable

Nei lucidi su algebra dei processi trovate una discussione su come modellare le variabili in algebra dei processi e/o con le reti di Petri. Nelle pagine seguenti l'esempio odi modellizzazione che abbiamo svolto a lezione.

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

Algorithm 3.2: First attempt

integer turn \leftarrow 1

p

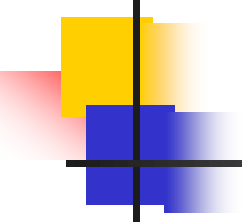
loop forever

p1: non-critical section
p2: await turn = 1
p3: critical section
p4: turn \leftarrow 2

q

loop forever

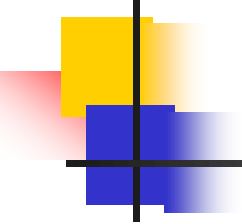
q1: non-critical section
q2: await turn = 2
q3: critical section
q4: turn \leftarrow 1



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*)



Derivation graph del processo CSP di 3.2



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

Il RG ha un'unica componente fortemente connessa, quindi la marcatura iniziale è un home state, la rete è reversibile, ogni transizione etichetta almeno un arco e quindi la rete, per la marcatura iniziale data, è viva, non ci sono deadlock

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: $AG \neg(\#P4 == 1 \ \&\& \ \#Q4 == 1)$
- LTL: $G \neg(\#P4 == 1 \ \&\& \ \#Q4 == 1)$

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: ???