

VPC 19-20

Commenti su esercizio mutua esclusione in NuSMV

Prof.ssa Susanna Donatelli

Universita' di Torino

www.di.unito.it

susi@di.unito.it



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 e' definita dalla congiunzione delle seguenti condizioni:
 - **Mutua esclusione:** le istruzioni delle sezioni critiche di due o piu' processi non possono essere eseguite in modo interfolgiato
 - **Assenza di deadlock:** Se qualche processo cerca di accedere alla regione critica eventualmente un processo potra' farlo
 - **Assenza di starvation individuale:** Se un processo cerca di accedere alla regione critica eventualmente quel processo potra' farlo
3. Le variabili usate dal protocollo di accesso sono usate solo dal protocollo di accesso
4. C'e' progresso nella regione critica (se un processo inizia l'esecuzione in regione critica alla fine terminera' tale esecuzione)
5. Non si richiede progresso da parte dei processi nelle istruzioni che non appartengono alla regione critica

La mutua esclusione (1)

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.

Attenzione a non obbligare i processi al progresso per le istruzioni che non sono di regione critica.

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

```

MODULE user(turn, io, altro)
VAR
  state : {non-crit,enter, crit,exit};
ASSIGN
  init(state) := non-crit;
  next(state) :=
    case
      state = non-crit : {non-crit,enter};
      state = enter & turn = io : crit;
      state = crit : exit;
      state = exit : non-crit;
      TRUE : state;
    esac;
  next(turn) :=
    case
      next(state) = exit : altro;
      TRUE : turn;
    esac;
FAIRNESS
  running

```

Algorithm 3.2: First

integer turn \leftarrow 1

p

loop forever

p1: non-critical section

q1:

p2: await turn = 1

q2:

p3: critical section

q3:

p4: turn \leftarrow 2

q4:

Stati: la mutua esclusione è vera

```
----- State 1 -----  
turn = p  
procP.state = exit  
procQ.state = non-crit  
----- State 2 -----  
turn = p  
procP.state = crit  
procQ.state = non-crit  
----- State 3 -----  
turn = p  
procP.state = enter  
procQ.state = non-crit  
----- State 4 -----  
turn = p  
procP.state = non-crit  
procQ.state = non-crit  
----- State 5 -----  
turn = p  
procP.state = exit  
procQ.state = exit  
----- State 6 -----  
turn = p  
procP.state = crit  
procQ.state = exit  
----- State 7 -----  
turn = p  
procP.state = enter  
procQ.state = exit  
----- State 8 -----  
turn = p  
procP.state = non-crit  
procQ.state = exit
```

```
----- State 9 -----  
turn = p  
procP.state = exit  
procQ.state = enter  
----- State 10 -----  
turn = p  
procP.state = crit  
procQ.state = enter  
----- State 11 -----  
turn = p  
procP.state = enter  
procQ.state = enter  
----- State 12 -----  
turn = p  
procP.state = non-crit  
procQ.state = enter  
----- State 13 -----  
turn = q  
procP.state = non-crit  
procQ.state = exit  
----- State 14 -----  
turn = q  
procP.state = non-crit  
procQ.state = crit  
----- State 15 -----  
turn = q  
procP.state = non-crit  
procQ.state = enter
```

```
----- State 16 -----  
turn = q  
procP.state = non-crit  
procQ.state = non-crit  
----- State 17 -----  
turn = q  
procP.state = exit  
procQ.state = exit  
----- State 18 -----  
turn = q  
procP.state = enter  
procQ.state = exit  
----- State 19 -----  
turn = q  
procP.state = exit  
procQ.state = crit  
----- State 20 -----  
turn = q  
procP.state = enter  
procQ.state = crit  
----- State 21 -----  
turn = q  
procP.state = exit  
procQ.state = enter
```

```
----- State 22 -----  
turn = q  
procP.state = enter  
procQ.state = enter  
----- State 23 -----  
turn = q  
procP.state = exit  
procQ.state = non-crit  
----- State 24 -----  
turn = q  
procP.state = enter  
procQ.state = non-crit  
-----
```



Proprietà per la mutua esclusione (CTL)

-- specification **AG !(procP.state = crit & procQ.state = crit) is true**

-- specification **EF (procP.state = crit & procQ.state = crit) is false**

-- as demonstrated by the following execution sequence

Trace Description: CTL Counterexample

-> State: 1.1 <-

turn = p

procP.state = non-crit

procQ.state = non-crit



Proprietà per la mutua esclusione (LTL)

-- specification **F !(procP.state = crit & procQ.state = crit) is true**

-- specification **G !(procP.state = crit & procQ.state = crit) is true**

Assenza di deadlock (se un processo chiede, qualche processo potrà accedere) - CTL

-- AG (procQ.state = enter -> AF (procQ.state = crit | procP.state = crit)) is false

Trace Description: CTL Counterexample

```
-> State: 1.1 <-  
  turn = p  
  procP.state = non-crit  
  procQ.state = non-crit  
-> Input: 1.2 <-  
  _process_selector_ = procQ  
-- Loop starts here  
-> State: 1.2 <-  
  procQ.state = enter  
-> Input: 1.3 <-  
-- Loop starts here  
-> State: 1.3 <-  
-> Input: 1.4 <-  
  _process_selector_ = procP  
-- Loop starts here  
-> State: 1.4 <-  
-> Input: 1.5 <-  
  _process_selector_ = main  
-> State: 1.5 <-
```

Si noti che anche la versione rispetto al processo P è falsa

Assenza di deadlock (se un processo chiede, qualche processo potrà accedere) - CTL

Si noti che anche la versione rispetto al processo P è falsa, ed è falsa anche se chiediamo che un processo o l'altro chiedano di entrare

AG ((procQ.state = enter | procP.state = enter) -> AF (procQ.state = crit | procP.state = crit)) is false

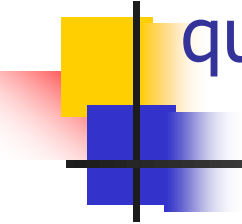
perché il problema è che se P decide di entrare quando il turno è di Q, se Q decide di non richiedere mai l'accesso, P non riuscirà mai ad accedere alla regione critica

invece

-- specification AG ((procQ.state = enter & procP.state = enter) -> AF (procQ.state = crit | procP.state = crit)) is true

cioè vi è assenza di deadlock solo se ho certezza che ambedue i processi vogliono entrare (il che viola l'ipotesi di permettere che i processi non siano forzati a progredire quando sono in regione non critica).

Assenza di deadlock (se un processo chiede, qualche processo potrà accedere) - LTL



Le proprietà nella versione LTL (cancellazione degli operatori di cammino) porta agli stessi risultati della verifica CTL



Presenza di starvation individuale (CTL)

-- **specification AG (procQ.state = enter -> AF procQ.state = crit) is false**

Trace Description: CTL Counterexample

```
-> State: 1.1 <-  
  turn = p  
  procP.state = non-crit  
  procQ.state = non-crit  
-> Input: 1.2 <-  
  _process_selector_ = procQ  
-- Loop starts here  
-> State: 1.2 <-  
  procQ.state = enter  
-> Input: 1.3 <-  
-- Loop starts here  
-> State: 1.3 <-  
-> Input: 1.4 <-  
  _process_selector_ = procP  
-- Loop starts here  
-> State: 1.4 <-  
-> Input: 1.5 <-  
  _process_selector_ = main  
-> State: 1.5 <-
```

Non sorprendente perchè se è falsa la proprietà di assenza di deadlock non ci possiamo aspettare che questa sia vera



Presenza di starvation individuale (CTL)

Si noti che invece

-- specification **AG (procQ.state = enter -> EF procQ.state = crit)** is true

cioè il processo che voglia entrare ha la possibilità di progredire, ma non in modo indipendente dalle scelte dell'altro processo

Presenza di starvation individuale (LTL)

-- specification G (procQ.state = enter -> F procQ.state = crit) is false

Trace Description: LTL Counterexample

-> State: 1.1 <-

turn = p

procP.state = non-crit

procQ.state = non-crit

-> Input: 1.2 <-

_process_selector_ = procP

-> State: 1.2 <-

-> Input: 1.3 <-

_process_selector_ = procQ

-> State: 1.3 <-

-> Input: 1.4 <-

-- Loop starts here

-> State: 1.4 <-

procQ.state = enter

-> Input: 1.5 <-

_process_selector_ = main

-- Loop starts here

-> State: 1.5 <-

-> Input: 1.6 <-

_process_selector_ = procP

-- Loop starts here

-> State: 1.6 <-

-> Input: 1.7 <-

_process_selector_ = procQ

-- Loop starts here

-> State: 1.7 <-

-> Input: 1.8 <-

_process_selector_ = main

-> State: 1.8 <-

Presenza di starvation individuale (LTL) – versione “infinitely often” – È utile?



--- specification **G (F (procQ.state = enter -> F procQ.state = crit))** is false

il controesempio è lo stesso