Modelli Concorrenti e Algoritmi Distribuiti

**a.a. 2019/2020**

**Prova scritta 16 giugno 2020**

# Esercizio 1

Lo schema di codice sotto riportato presenta un monitor (con semantica *signal and urgent wait*) che risolve il problema dei cinque filosofi a cena. In questa soluzione la condizione di accesso mutuamente esclusivo ad una forchetta è sostituito dalla condizione che due filosofi contigui non possano mangiare contemporaneamente (distanziamento sociale!).

**monitor** filosofi **{**

**bool** eating[0. .4] = [F F F F F]

**condition** attesa[0. .4]

**int** destra(i) **{**

**return** (i+1)%5

**}**

**int** sinistra(i) **{**

**return** (i-1)%5

**}**

**public void** iniziocena (i) **{**

**if** eating[destra(i)] .**OR**. eating[sinistra(i)] **wait**(attesa[i]);

eating[i] = T

**}**

**public void** finecena(i) **{**

eating[i] = F

**if** !empty**(**attesa[destra(i)] .**AND**. !eating[destra(destra(i))]

**signal** (attesa[destra(i)]);

**if** !empty**(**attesa[sinistra(i)] .**AND**. !eating[sinistra(sinistra(i))]

**signal** (attesa[sinistra(i)]);

**}**

**}**

**Filosofo\_i**

**repeat**

<pensa>

filosofi.iniziocena(i)

<mangia>

filosofi.finecena(i)

**forever**

E’ immediato provare che la formula seguente rappresenta un invariante di monitor (N.B. *Non ne è richiesta la dimostrazione*)

1. ∀i. [eating[i] ⇒ !eating[(destra(i)]

SiaEi una variabile intera che assume valore 1 se eating[i] vale T e valore 0 se eating[i] vale F. Utilizzando l’ invariante 1) dimostrare che anche la formula seguente è un invariante di monitor:

0 ≤ Σ4i=0 Ei ≤ 2

Modelli Concorrenti e Algoritmi Distribuiti

**a.a. 2019/2020**

**Prova scritta 7 luglio 2020**

# Esercizio 1

Lo schema di codice sotto riportato presenta un monitor (con semantica *signal and urgent wait*) per la gestione di un conto corrente condiviso.

**monitor** conto\_corrente**{**

**integer** saldo = valore\_iniziale;

**condition** attesa;

**public** **void** prelievo(**intege**r m)**{**

**while** (saldo < m) **wait**(attesa,m);

saldo = saldo - m;

**signal** (attesa);

**}**

**public** *void* deposito(**intege**r n)**{**

saldo = saldo + n;

**signal**(attesa);

**}**

}

Si supponga che ui indichi un generico processo bloccato sulla condition attesa, e req(ui) la cifra che il processo ui è in attesa di prelevare; si dimostri che la formula seguente è un invariante di monitor.

¬empty (attesa) ⇒ ∀ ui∈attesa req(ui) > saldo

Modelli Concorrenti e Algoritmi Distribuiti

**a.a. 2019/2020**

**Prova scritta 7 settembre 2020**

# Esercizio 1

Lo schema di codice sotto riportato si riferisce ad un sistema costituito da un ORSO e da un numero *n* (*n* > 1) di API. Le api depongono, una alla volta, una quantità fissa (quantum) di miele in un favo che ha la capacità di 4 quantum. L’orso accede al favo solo quando questo è tutto pieno e lo svuota completamente. Il favo è condiviso per cui le operazioni di deposito e di prelievo vanno eseguite in mutua esclusione.

Provare che:

.

1. la soluzione è corretta, i.e. detti Nd(t) e Ns(t) rispettivamente il numero di operazioni di deposito e di svuotamento eseguite fino all’istante t:

0 ≤ Nd(t) – 4\* Ns(t) ≤ 4

1. non è possibile che l’orso svuoti il favo mentre un’ape sta deponendo un quantum di miele;

**semaphore** full= 0;

**semaphore** empty = 4;

**semaphore** mutex = 1;

**process** APEi **{**

**while** (true) **{**

<vola>

**P**(empty);

**P**(mutex);

<deposita un quantum>;

**V**(mutex);

**V**(full);

**}**

**}**

**process** ORSO **{**

**while** (true) **{**

<dorme>

**P**(full);

**P**(full);

**P**(full);

**P**(full);

<svuota il favo>;

**V**(empty);

**V**(empty);

**V**(empty);

**V**(empty);

**}**

**}**

# Esercizio 1

Si consideri un sistema costituito da due processi concorrenti: il processo **PA** che in un ciclo infinito stampa il carattere “(“ e il processo **P**C che in un ciclo infinito stampa il carattere “)”.La stampante è unica e si vuole che l’operazione di stampa sia atomica e che ogni sequenza di parentesi stampate (di qualsiasi lunghezza) soddisfi i seguenti requisiti:

1. sia *ben parentesizzata*, cioè per ogni suo prefisso t il numero di parentesi aperte sia sempre maggiore o uguale al numero delle parentesi chiuse:

∀t. num ( “(“,t) - num (“)” ,t) ≤ 0

1. per ogni prefisso t, il *livello di annidamento*, cioè la differenza tra il numero di parentesi aperte e il numero di parentesi chiuse, sia sempre minore o uguale a 3:

∀t. num ( “(“,t) - num (“)” ,t) ≤ 3

(num (σ,t) indica numero di occorrenze del simbolo σ nel prefisso t).

Il monitor seguente, realizzato secondo la semantica *signal\_and\_urgent\_wait*, è una soluzione per questo sistema.

**monitor** parentesi **{**

**int** count = 0

**condition**  aperte

**condition** chiuse

**public void** stampa\_aperta() **{**

**if (**count==3) **wait**(aperte);

<print “(“>

count++

**signal**(chiuse) **}**

**public void** stampa\_chiusa () **{**

**if (**count==0) **wait**(chiuse);

<print “) “>

count--

**signal**(aperte) **}**

**}**

**cobegin** **{**

**process PA** **{**

**while** true

parentesi.stampa\_aperta

**}**

**process PC {**

**while** true

parentesi.stampa\_chiusa

**}**

**}**

Siano *nA* e *nC* le variabili di stato che contano rispettivamente il numero di parentesi aperte e chiuse stampate. Si dimostri che valgono i seguenti invarianti di monitor:

1. count = nA – nC
2. 0 < count < 3

Modelli Concorrenti e Algoritmi Distribuiti

**a.a. 2019/2020**

**Prova d’esame 18 febbraio 2020**

# Esercizio 1

Lo schema riportato definisce le primitive utilizzate da un sistema costituito da un processo produttore e da un processo consumatore per scambiarsi i dati; i dati devono essere consumati seguendo una politica LIFO (last in first out) per cui il buffer è realizzato utilizzando uno stack di dimensione n, n>0.

**T** stack[n]

**int** top = 0

**semaphore** full = 0

**semaphore** empty = n

**semaphore** mutex = 1

**public void** deposita (**T** d) **{ public T** preleva () **{**

**P**(empty) **P**(full)

**P**(mutex) **P**(mutex)

top++ **T** r = stack[top]

stack[top] = d top --

**V**(mutex) **V**(mutex)

**V**(full) **V**(empty)

**} return** r

**}**

1. Utilizzando gli invarianti dei semafori full e empty e gli invarianti topologici, dimostrare che Il valore della variabile top è limitato nell’intervallo [1. .n], cioè:

∀t. 0 ≤ val(top,t) ≤ n

1. Provare l’assenza di deadlock.