Classe Quarta I2 : Esercitazione di Concorrenza

L’obiettivo dell’esercitazione è stabilire se la seguente affermazione è corretta:

**Per stabilire se due istruzioni P1 e P2 possono essere eseguite parallelamente in modo *safe* occorre accertare che:**

**W (P1) U W (P2) ≠ \varnothing**

**W (P1) ∩ R (P2) = \varnothing**

**W (P2) ∩ R (P1) = \varnothing**

Dove W (P) e R (P) trovano spiegazione nel documento allegato

**Risposta**

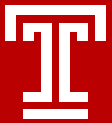
* + - **Vero**
    - **Falso**

Inoltre, dati:

P1 : a := x + y P2 : b := z + 1

P1 e P2 possono essere eseguiti concorrentemente senza race conditions?

**risposta:**

**CIS 307: Interleaving, Determinate Computations, Bernstein Conditions**

Operating Systems support Tasks (or Processes). But for now let's think of "Actitivities". Activities are more general than processes, since they may be expressed by software, or hardware, or even by what people do. They consist of sequential atomic actions (an atomic action executes in an all-or-nothing mode, with invisible intermediate states). Activities can be executed in parallel.  
Here are two activities, P and Q:

P: abc

Q: def

a, b, c, d, e, f, are atomic actions. Each action has always the same effect independent of what other actions may be executing at the same time.  
What is the effect of executing the two activities concurrently? We do not know for sure, but we know that it will be the same as obtained by executing sequentially an INTERLEAVING of the two activities. Here are the possible interleavings of these two activities:

abcdef

abdcef

abdecf

abdefc

adbcef

......

defabc

That is, the actions of the two activities are sequenced in all possible ways that preserve the order in which the actions appeared in the two activities.

The importance of the concept of interleaving is that it allows us to determine the meaning of concurrent programs.

Since the parallel execution of activities is equivalent to the sequential execution of an interleaving of these activities, it can complete with different results. We will say that a set of activities (i.e. a program) is DETERMINATE if any time it runs, for the same inputs, it gives the same outputs. Otherwise it is NON-DETERMINATE (it has a RACE CONDITION).

Here are two activities that are not determinate. The first activity just executes the assignment a := 1; The second activity just executes the assignment a := 2;

Supposed that you are given the following two activities P and Q:

**P:** x := 2;  **Q:** x := 3;

y := x – 1 ; y := x + 1;

What are the possible results of the parallel execution of these two activities? [Answer: x=3 and y=4, or x=2 and y=1, or x=2 and y=3, or x=3 and y=2. If you do not want to get some of these results, then you have to prevent the occurrence of the interleavings that caused those results. This is done with SYNCHRONIZATION MECHANISMS.]

The BERNSTEIN CONDITIONS are used to determine if a program is determinate.  
The READ SET and the WRITE SET of an action A are respectively the set of variables read by A and written by A. The Read Set of an activity P, R(P), is the union of the read sets of all the actions of P. Similarly, the Write Set of P, W(P), is the union of the write sets of all the actions of P.

For example, if P is:

x := u + v;

y := x \* w;

then R(P) = {u,v,x,w}, W(P) = {x, y}. Notice that x is both in R(P) and in W(P).  
Now the Bernstein Conditions:

Given activities P and Q, if

• The intersection of W(P) with W(Q) is empty, and

• The intersection of W(P) with R(Q) is empty, and

• The intersection of R(P) with W(Q) is empty then

The parallel execution of P and Q is determinate.

If these conditions are not satisfied, then it is possible that the parallel execution of P and Q is determinate [can you come up with an example?], but usually it is not.  
[Note: We have discussed these conditions considering only two activities, but as you have guessed they generalize to more activities.]

The Bernstein Conditions are informative but too demanding: in most situations we want to execute activities that read/write the same data, and yet we don't want to give up determinacy. Thus something else is needed, some way of limiting the interleavings that are possible when executing activities in parallel. That is what SYNCHRONIZATION MECHANISMS do for us.

*Sitografia: ingargiola.cis.temple.edu*

<http://www.cis.temple.edu/~ingargio/old/cis307s95/readings/interleave.html>

Temple University CIS Department  
Room 303, Computer Building(038-24)

1805 N. Broad St.  
Philadelphia, PA, 19122

EXERCISE: Suppose we are given the following sequential program:

a = x + y; /\* Statement s1 \*/

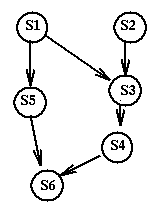
b = z + 1; /\* Statement s2 \*/

c = a - b; /\* Statement s3 \*/

w = c + 1; /\* Statement s4 \*/

d = a + e; /\* Statement s5 \*/

w = w \* d; /\* Statement s6 \*/



We can use the Bernstein conditions to derive a precedence graph from this sequential program.

The nodes in this graph are the statements s1, s2, s3, s4, s5, s6.

Does this graph respect Bernstein’s condition? Argument your answer.

This precedence graph could be executed using two processors. One processor could execute in order s1, s5, s6 and the other s2, s3, s4. Of course they should synchronize so that s3 executes after s1 and s6 after s4.This precedence graph is MAXIMALLY PARALLEL for the given program. That is, the only dependencies that appear in the graph are those required by the logic of the program.

A

A sequential program consists of the following five statements, S1 through S5. Considering each statement as a separate process, clearly identify *input set Ii* and *output set* Oi of each process. Restructure the program using Bernstein's conditions in order to achieve maximum parallelism between processes. If any pair of processes cannot be executed concurrently, specify which of the three conditions is not satisfied.

S1: A = B + C  
S2: C = B X D  
S3: S = 0  
S4: Do I = A, 100  
S = S + X(I)  
End Do  
S5: IF (S > 1000) C = C x 2