Esercizi Race Condition

[2023gennaio24.pdf](https://drive.google.com/open?id=16s2kKPwCmiKc5tl1WbbEPvaiO2_InXdE)

Assumiamo un array di int A inizializzato con [10,20].

* Un thread esegue l’operazione A[1]=A[2]+5.
* L’altro thread A[2]=A[1]+55.

Si argomenti in modo formale se possono verificarsi race condition su A.

Funzione che formalizza l’operazione del primo thread (cioè A[1] = A[2] + 5):

f1([m,n]) = [n+5,n].

Nota: in particolare, vale che

f1([10,20]) = [20+5,20] = [25,20] e

f1([10,65]) = [65+5,65] = [70,65].

Funzione che formalizza l’operazione del secondo thread (cioè A[2] = A[1] + 55):

f2([m,n]) = [m,m+55].

Nota: in particolare, vale che

f2([10,20]) = [10,10+55] = [10,65] e

f2([25,20]) = [25,25+55] = [25,80].

Valori ammissibili su A, cioè valore che otterremmo in caso di esecuzione sequenziale delle due operazioni:

f2(f1([x,y])) = f2([y+5,y]) = [y+5,y+5+55] = [y+5,y+60]

e

f1(f2([x,y])) = f1([x,x+55]) = [x+55+5,x+55] = [x+60,x+55].

Nel nostro esempio:

f2(f1([10,20])) = [20+5,20+60] = [25,80]

f1(f2([10,20])) = [10+60,10+55] = [70,65]

Possiamo avere r.c. perchè l’array può assumere, per esempio, il valore [25,20] oppure [10,65] che è diverso dai due ammissibili.

[2023febbraio09.pdf](https://drive.google.com/open?id=1qKwePRZUu-EswYP2mWXwvFwyimSGnHi6)

Assumiamo un array di int A inizializzato con [30,10,20] e condiviso da due thread.

* Un thread esegue l’istruzione A[0] = A[0] + A[1].
* L’altro thread esegue l’istruzione A[0] = A[0] \* A[2].

Si argomenti in modo formale se possono verificarsi race condition su A.

Funzione che formalizza l’operazione del primo thread (cioè A[0] = A[0] + A[1]):

f1([a, b, c]) = [a+b, b, c].

Nota: in particolare, vale che

f1([30, 10, 20]) = [30+10, 10, 20] = [40, 10, 20] e

f1([600, 10, 20]) = [600+10, 10, 20] = [610, 10, 20].

Funzione che formalizza l’operazione del secondo thread (cioè A[0] = A[0] \* A[2]):

f2([a, b, c]) = [a\*c, b, c].

Nota: in particolare, vale che

f2([30, 10, 20]) = [30\*20, 10, 20] = [600, 10, 20] e

f2([40, 10, 20]) = [40\*20, 10, 20] = [800, 10, 20].

Valori ammissibili su A, con A inizializzato con [x, y, z], cioè valori che otterremmo in

caso di esecuzione sequenziale delle due operazioni:

f2(f1([x, y, z])) = f2([x+y, y, z]) = [(x+y) \* z, y, z]

e

f1(f2([x, y, z])) = f1([x\*z, y, z]) = [(x\*z) + y, y, z]

Nel nostro esempio:

f2(f1([30,10,20])) = [(30 + 10) \* 20, 10, 20] = [40 \* 20, 10, 20] = [800, 10, 20]

f1(f2([30,10,20])) = [(30 \* 20) + 10, 10, 20] = [600 + 10, 10, 20] = [610, 10, 20]

Possiamo avere r.c. perchè l’array può assumere, per esempio, il valore [40, 10, 20] che è diverso dai due ammissibili.

[2024gennaio08](https://docs.google.com/document/u/0/d/1seupWGND9z7mrYsXRdbt3JW7EH73Kuvohz8g_Pb8y4Q/edit)

Assumiamo che due thread condividano una variabile double X, con valore iniziale X=100.

* Il primo thread esegue l’istruzione X=50;
* Il secondo thread esegue l’istruzione if (X>70) { X=200; } else { X=0; }

Discutere, formalmente, se è possibile che si verifichino race condition sulla variabile X.

L’operazione eseguita dal primo thread può essere formalizzata con la funzione f1: R—> R definita come segue (x è un valore reale, non va confuso con la variabile X):

f1(x) = 50.

L’operazione eseguita dal secondo thread può essere formalizzata con la funzione f2: R—> R definita, per parti, come segue:

f2(x) = 200, se x > 70

f2(x) = 0, se x <= 70

Pertanto, componendo le funzioni come segue, spieghiamo il comportamento dei thread quando la loro attività viene sequenzializzata in ordine arbitrario:

f1(f2(x)) = f1(200) = 50, se x > 70

f1(f2(x)) = f1(0) = 50, se x <= 70

e

f2(f1(x)) = f2(50) = 0

In particolare, tenendo conto che il valore iniziale di X è 100, vale che:

f1(f2(100)) = f1(200) = 50

e

f2(f1(x)) = f2(50) = 0

Possiamo pertanto dire che avremo race condition se e solo se X può acquisire valori diversi da

50 e 0.

Una possibile esecuzione è la seguente: il secondo thread testa la guardia (X>70), che risulta vera, perde il processore, il secondo thread esegue l’assegnamento, il primo thread riacquisisce il processore e assegna il valore 200 alla variabile X, che non verrà modificato in seguito. Essendo 200 diverso da 50 e da 0, concludiamo che questa esecuzione da luogo a race condition.

[2024gennaio24](https://docs.google.com/document/u/0/d/1bnvhexZ-PYxeHPgDPIt_D0CLwQyujbNjR2h6XQ9I54Q/edit)

Si assuma la variabile X condivisa da due thread che invocano il seguente metodo M:

* public void M( ) { if ( X >= 0 ){ X = - 5; } else { X = 12; }

Discutere formalmente se si possono verificare race condition su X.

L’effetto sulla variabile X dell’esecuzione di ognuno dei due thread può essere formalizzato con la funzione f: R—> R definita, per parti, come segue:

f(y) = 12, se y < 0.

f(y) = -5, se y >= 0.

Pertanto, il comportamento dei due thread che eseguono sequenzialmente può essere formalizzato applicando due volte f:

f(f(y)) = f(12) = -5, se y < 0.

f(f(y)) = f(-5) = 12, se y >= 0.

Pertanto, se il valore iniziale di X è non negativo, per non avere race condition la variabile X deve assumere valore finale 12.

Altrimenti, se il valore iniziale di X è negativo, per non avere race condition la variabile X deve assumere valore finale -5.

Assumiamo che inizialmente X valga 10. Un thread potrebbe testare la guardia e perdere il processore prima di eseguire il ramo “then”.

L’altro thread potrebbe eseguire tutto il metodo. Il primo thread riprenderebbe in seguito l’esecuzione ed eseguirebbe il ramo “then”.

Alla fine X varrebbe -5, ma l’unico valore ammissibile partendo da X=10 è 12, pertanto in questo caso avremmo race condition.

Si assuma una classe con una variabile di istanza int X. Supponiamo che il valore iniziale (impostato dal costruttore) sia 100.

Si assuma che la classe abbia il metodo:

* public void M(int Y) { if(X<15) {X=X+2;} else {Y=X+30;} }

Assumiamo due thread che eseguono in concorrenza. Assumiamo che entrambe invochino il metodo M sul medesimo oggetto. Discutere in modo formale se sono possibili race condition sulla variabile di istanza X di tale oggetto.

…

Non c’e race condition

X=100

M (int Y) {

if(X<15)

X=X+2;

else

Y=X+30;

}

Y=130