

Esercizio 1

1. Algoritmo A - Non corretto!

Per tutti gli algo supponiamo

$$f(1) = 0 \quad \text{et} \quad f(j) \neq 0 \quad \text{per tutti } j \neq 1$$

→ Process P mette found a true e si ferma poi

Process Q lo rimette a false con $j=0$ e
nonno dei due finisce

2. Algoritmo B. Non corretto!

→ Si può fare un'esecuzione con solo Process Q
e non finisce mai!

3. Algoritmo C. Non corretto!

→ Process P lo esegue, mette turn a 2

→ Process Q lo esegue, mette turn a 1

→ Process Q fa il ciclo e torna sul monitor

→ Process P mette found a true ed esce.

→ Process Q rimane bloccato

Algoritmo D Corretto

FIGURA 2. 1.

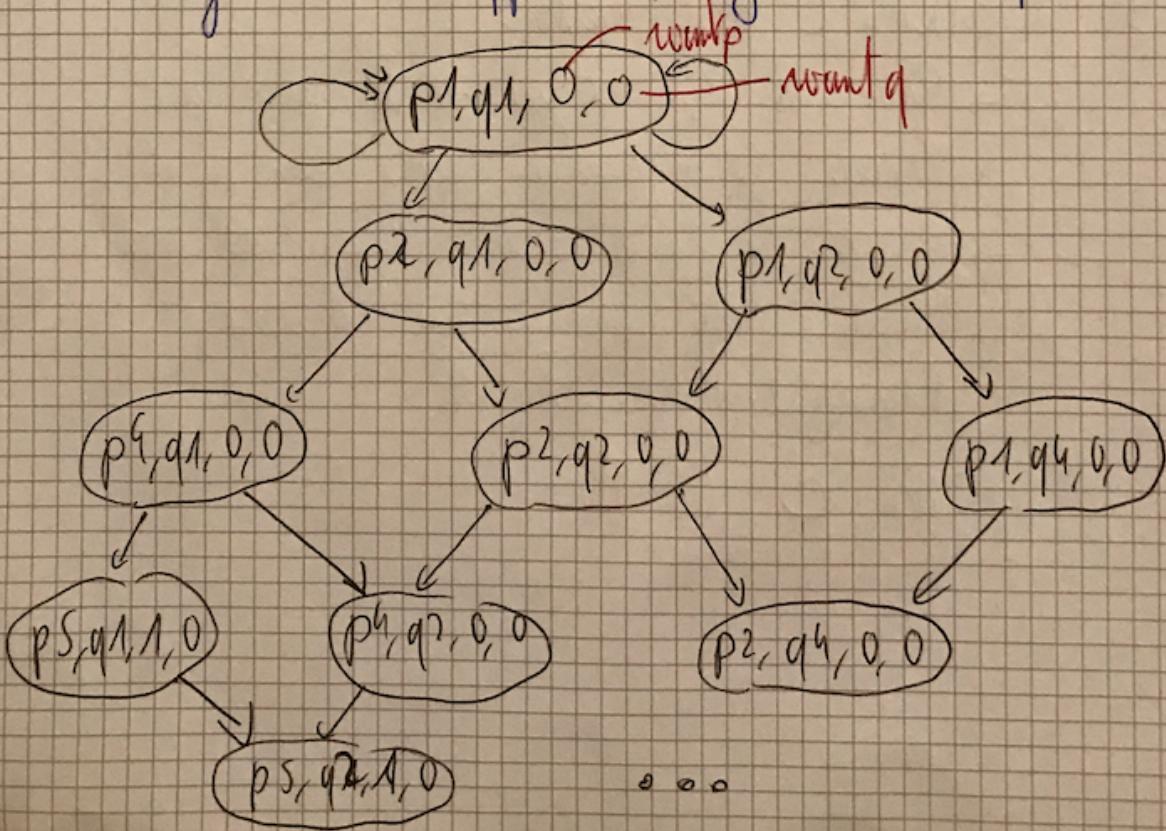
Process P

```
while (true) {
    p1 SNC
    p2 if (wantq == -1) {
        p3 wantp = -1;
    } else {
        p4 wantp = 1;
    }
    p5 while (wantp == wantq) {}
    p6 SC
    p7 wantp = 0
}
```

Process Q

```
while (true) {
    q1 SNC
    q2 if (wantp == -1) {
        q3 wantq = 1;
    } else {
        q4 wantq = -1;
    }
    q5 while (wantp == -wantq) {}
    q6 SC
    q7 wantq = 0
}
```

Il diagramma ha troppi stati, ne facciamo una parte:



2.2 L'algoritmo NON verifica la mutua esclusione.

P	Q	want p	want q
p1	q1	0	0
\downarrow p4	q1	0	0
p4	q4	0	0
\downarrow p4	q5	0	-1
\downarrow p4	q6	0	-1
\downarrow p5	q6	1	-1
\downarrow p6	q6	1	-1

L'algoritmo verifica l'assenza di deadlock!

Se $P \in Q$ sono bloccati, sono bloccanti

nel loro while e abbiamolo:

$want p == want q \& \& want p == -want q$.
questo è possibile solo se

$want p = want q = 0$ ma quando $P \in$ in
 $p4 \in Q$ è in q6 abbiamo $want p \in \{1, -1\}$ e

$want q \in \{-1, 1\}$. Dunque $want p \neq 0$ e $want q \neq 0$!

\Rightarrow NO DEADLOCK

L'algoritmo non garantisce l'assenza di starvation.

- Q si esegue e mette work q a -1
- P si esegue e ogni iterazione mette work p a -1 e passa lui in SC e se il while di Q è eseguito ogni volta che work p vale -1, Q rimane bloccato nel while!

3.1 Process P fa il primo flip e entra in SC

Process Q fa un altro flip e va in
while ($x \neq 0$)}} dove esce e fa un altro
flip e va in SC

⇒ Questo algoritmo non rispetta la mutua
esclusione!

3.2 Se cambiamo il flip con %3, l'algoritmo
rispetta la mutua esclusione

↳ Se un processo è in SC allora x vale 1, se
un altro arriva dopo per provare andrà nel
while ($x \neq 0$) e ci rimane finché x non si mette a 0

3. L'algoritmo rispetta l'assenza di deadlock

Per dimostrarlo, bisogna provare che i due processi non possono entrare tutti e due bloccati nel while($x \neq 0$) { }

↳ il motivo essendo che per arrivare a questo while si fa prima il while($flip() != 1$)

e se il risultato è 1 va in SC, se è 2 va nel while($x \neq 0$) ma è l'unica valore possibile. Quindi solo un processo può essere bloccato nel while($x \neq 0$)

Invece non rispetta l'assenza di starvation

- - Process P fa il flip va in SC
- Process Q fa il flip va in while($x \neq 0$) { }
- Process P mette x a 0
- Process Q va nel while($flip() != 1$) ma non fa il flip

Process P va sempre in SC, process Q no!

4.1 Mostriamo il problema con i due processi

P_{T1}} fa un giro completo aspettando
che S mette aut à 1, va in SC, mette end a true
req à 0, poi prosegue dritto e va in SC.

A questo punto P_{T2}} dovrà mettere req a 1
come end è a true, S può fare un giro e
mettere aut a 2 se P_{T2}} va in SC.

→ Il problema viene dal fatto che P_{T1}} non ha
aspettato che S mette aut à 0 per continuare.

L'algoritmo 8 risolve il problema

Con l'algo 8 c'è starvation, potrebbe essere che
S mette sempre 1 in aut!