Programação Concorrente

O Problema da Seção Crítica

Aula-03

Introdução

- O problema da **Seção Crítica** (Critical Section)
  - Suponha um conjunto de N processos executando em um laço infinito, um conjunto de instruções que podem ser divididas em dois subconjuntos:
    - Seção Crítica SC (CS)
    - Seção Não-Crítica SNC (NCS)

Introdução

- A **CORRETUDE** de uma solução requer:
  - **Exclusão mútua**: instruções pertencentes a uma seção crítica de dois ou mais processos não podem ser entrelaçadas
  - **Livre de deadlock**: se vários processos querem entrar na seção crítica, então eventualmente um deles irá conseguir.
  - **Livre de starvation**: se algum processo quer entrar em sua seção crítica, então eventualmente ele irá conseguir.

Introdução

- Mecanismo de sincronização
  - Uma forma de sincronização deve ser garantida para que os requisitos da corretude sejam satisfeitos.
  - Esse mecanismo consiste de instruções adicionais que são colocadas **antes** e **depois** da seção crítica.
  - As instruções colocadas antes, são chamadas de **pré-protocolo**.
  - As instruções colocadas depois, **pós-protocolo**.

Introdução

- Mecanismo de sincronização
  - Solução para dois processos:

Algorithm 3.1. Critical section problem

global variables

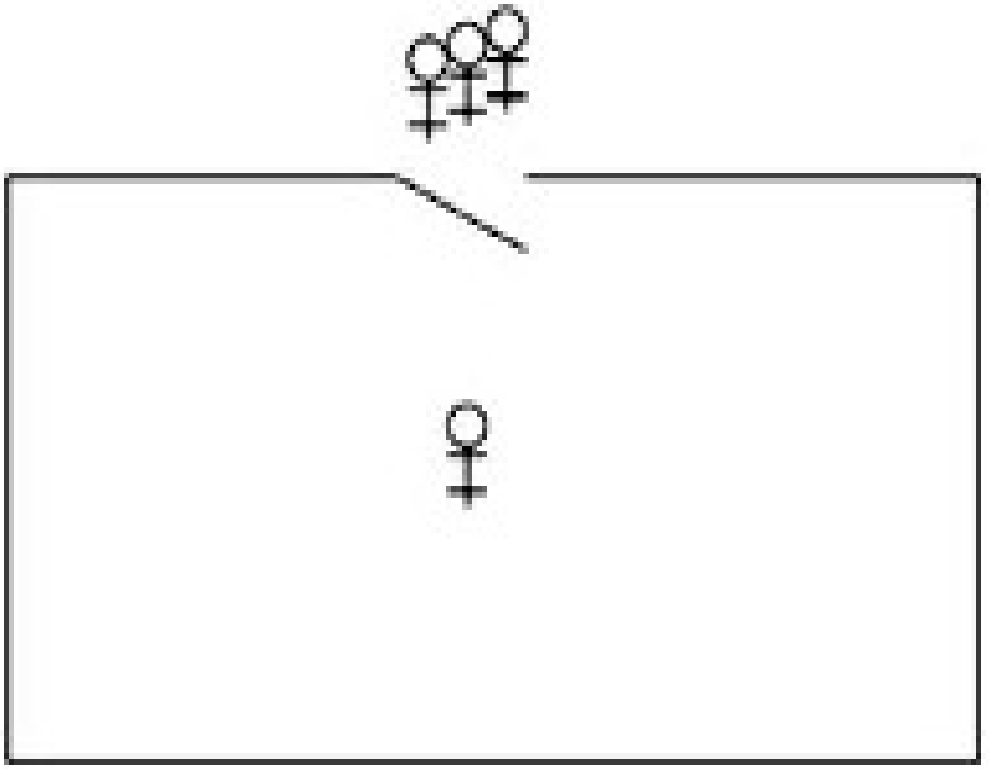
p	q
local variables loop forever non-critical section preprotocol critical section postprotocol	local variables loop forever non-critical section preprotocol critical section postprotocol

Introdução

- Os protocolos podem requerer variáveis locais e globais. No entanto variáveis usadas na seção crítica não são usadas na seção não-crítica e vice-versa.
- A seção crítica, deve **prosseguir,** ou seja, uma vez que um processo entra nela, ele deve sair.
- A seção não-crítica, **não precisa** obrigatoriamente prosseguir. Por quê?

Introdução

- Analogia



Introdução

- *The critical section problem is intended to model a system that performs complex computation, but occasionally needs to access data or hardware that is shared by several processes.*
- *It is unreasonable to expect that individual processes will never terminate, or that nothing bad will ever occur in the programs of the system.*
- *For example, a check-in kiosk at an airport waits for a passenger to swipe his credit card or touch the screen. When that occurs, the program in the kiosk will access a central database of passengers, and it is important that only one of these programs update the database at a time.*
- *The critical section is intended to model this operation only, while the non-critical section models all the computation of the program* ***except*** *for the actual update of the database.*

Introdução

- *It would not make sense to require that the program actually participate in the update of the database by programs in other kiosks, so we allow it to wait indefinitely for input, and we take into account that the program could malfunction.*
- *In other words, we do not require that the non-critical section progress. On the other hand, we do require that the critical section progress so that it eventually terminates.*
- *The reason is that the process executing a critical section typically holds a "lock" or "permission resource," and the lock or resource must eventually be released so that other processes can enter their critical sections.*
- *The requirement for progress is reasonable, because critical sections are usually very short and their progress can be formally verified.*

Seção Crítica

- Primeira solução

Algorithm 3.2. First attempt

```
integer turn ← 1
```

p	q
loop forever p1: non-critical section p2: await turn = 1 p3: critical section p4: turn ← 2	loop forever q1: non-critical section q2: await turn = 2 q3: critical section q4: turn ← 1

- Esta solução é correta?
  - Exclusão mútua (apenas um processo por vez na SC)?
  - Livre de Deadlock?
  - Livre de Starvation (considerando Fairness)?

Corretude

- Exemplos

Algorithm 3.3. History in a sequential algorithm

```
integer a ← 1, b ← 2
p1: Millions of statements
p2: a ← (a+b)*5
p3: . . .
```

Algorithm 3.4. History in a concurrent algorithm

```
integer a ← 1, b ← 2
```

p	q
p1: Millions of statements	q1: Millions of statements
p2: a ← (a+b)*5	q2: b ← (a+b)*5
p3: . . .	q3: . . .

Um cenário sequencial tem **fácil previsão**. Um cenário paralelo torna-se **impossível** prever.

Corretude

- Na computação sequencial, é possível prever com 100% de certeza o resultado final das variáveis envolvidas.
- Na computação concorrente, é impossíve predizer o resultado final devido ao entrelaçamento das instruções envolvidas.

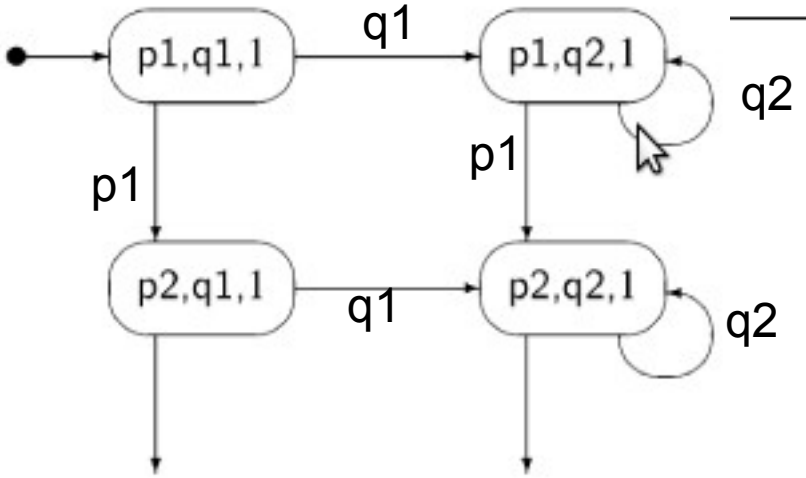
Corretude

- Usando diagramas

Algorithm 3.2. First attempt

```
integer turn ← 1
```

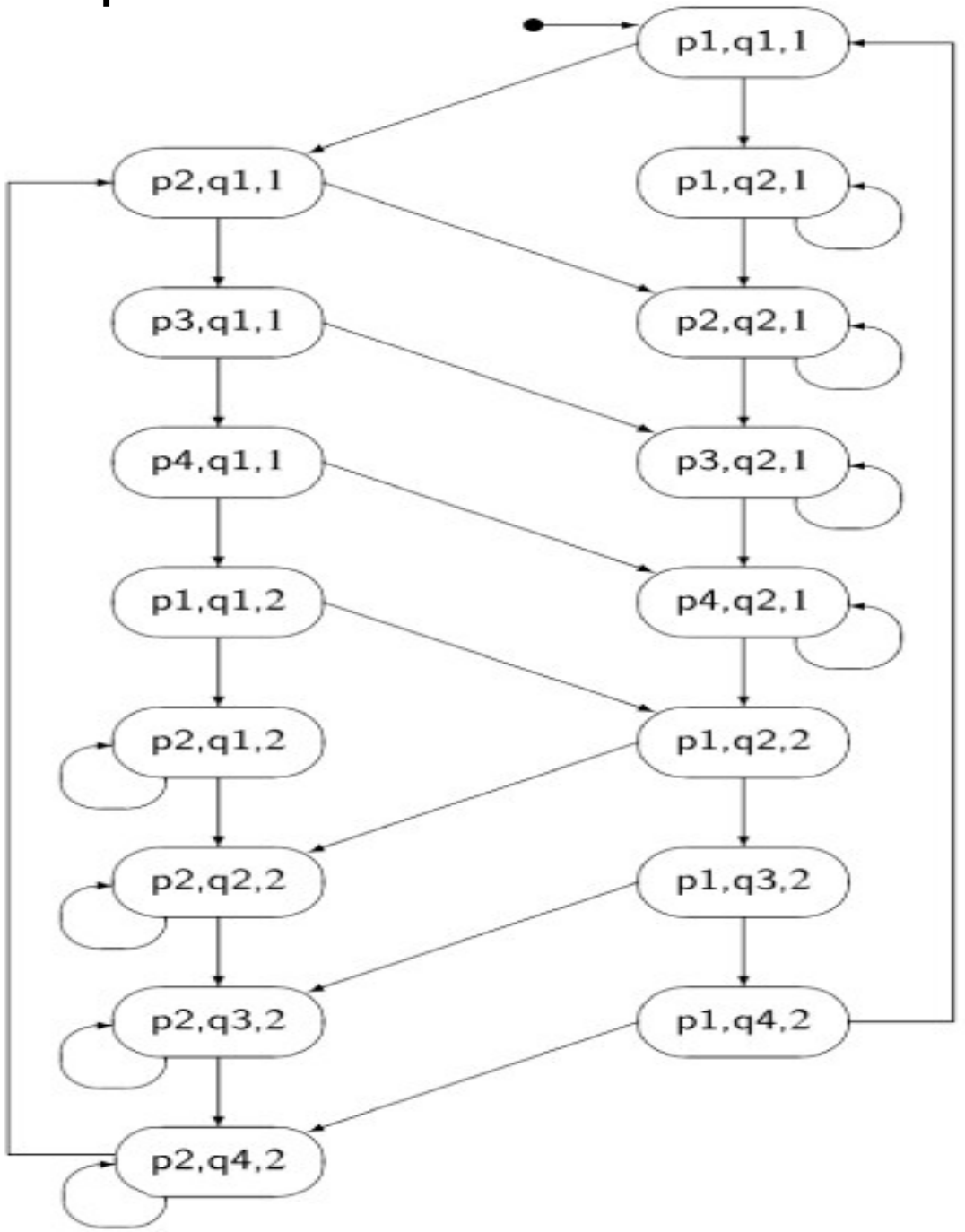
p	q
loop forever p1: non-critical section p2: await turn = 1 p3: critical section p4: turn ← 2	loop forever q1: non-critical section q2: await turn = 2 q3: critical section q4: turn ← 1



Corretude

- Quantos estados podem ter um diagrama de estados?
  - *Suppose that the algorithm has N processes with ni statements in process i, and M variables where variable j has mj possible values.*
  - *so the total number of states is (n1 x . . . x nN) x (m1 . . . x mM).*
  - *(n1 x . . . x nN)* = número de instruções do processo 1 vezes o número de instruções do próximo processo até N.
  - *(m1 . . . x mM)* = possíveis valores da variável m1 vezes possíveis valores da variável m2 até a variável M.
  - Qual a quantidade de estados para o algorítimo da primeira tentativa.

Diagrama expandido da primeira tentativa.



16 estados.

*A quick check shows that neither of the states **(p3, q3, 1)** nor **(p3, q3, 2**) occurs; thus we have proved that the mutual exclusion property holds for the first attempt.*

Corretude

- Problemas
  - Processos com poucos passos (*statements*) geram muitos estados.
  - O tamanho do diagrama pode ficar **impraticável**.
  - Tente **abreviar** seu diagrama (// nas seções).

Algorithm 3.5. First attempt (abbreviated)

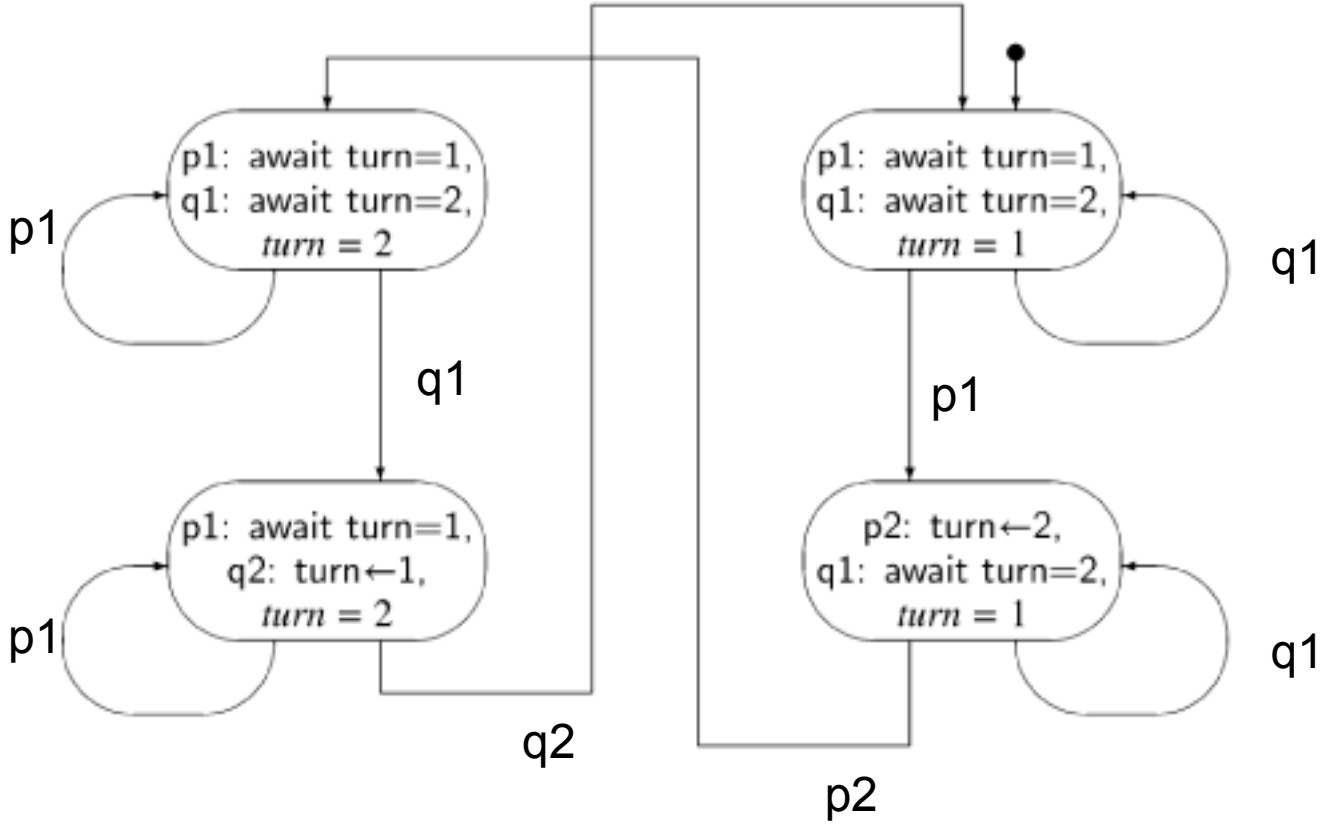
```
integer turn ← 1
```

p	q
loop forever p1: await turn = 1 p2: turn ← 2	loop forever q1: await turn = 2 q2: turn ← 1

Primeira tentativa abreviada.

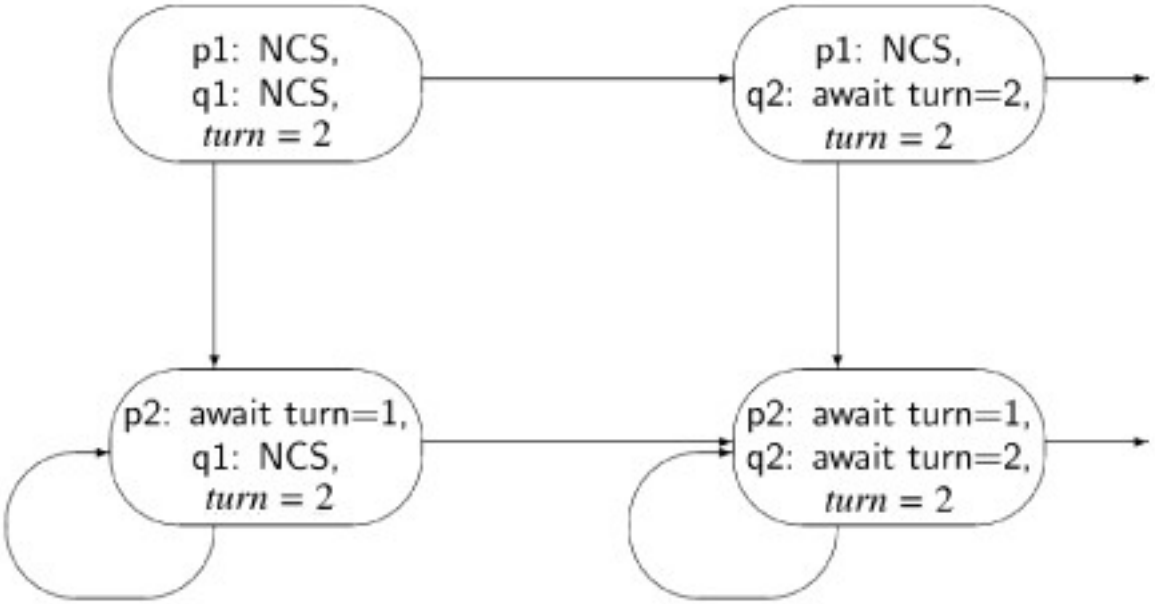
Corretude

- Diagrama reduzido



Corretude

- Problema da primeira tentativa?



- Exclusão mútua: garantido pelo diagrama
- Livre de *Deadlock*: basta ver se algum processo entra na SC.
- *Starvation*: problema quando algum processo entra na região não crítica.

O problema da primeira tentativa é o compartilhamento global de uma única variável que permite entrar na SC. Se um processo "morre", o outro fica bloqueado!

Corretude

- *Free from DeadLock?*
  - *Consider the upper left state (await turn=1, await turn=2, turn = 2). Both processes are trying to execute their critical sections; if process q tries to execute await turn=2, it will succeed and enter its critical section.*
  - *Consider now the lower left state (await turn=1, turn 1, turn = 2). Process p may try to execute await turn=1, but since turn = 2, it does not change the state. By the assumption of progress on the critical section and the assignment statement, process q will eventually execute turn 1, leading to the upper right state. Now, process p can enter its critical section. Therefore, the property of freedom from deadlock is satisfied.*

Corretude

- *Free from Starvation?*
  - *Finally, we have to check that the algorithm is free from starvation, meaning that if any process is about to execute its preprotocol (thereby indicating its intention to enter its critical section), then eventually it will succeed in entering its critical section.*
  - *The problem will be easier to understand if we consider the state diagram for the unabbreviated algorithm; where NCS denotes the non-critical section. Consider the state at the lower left. Process p is trying to enter its critical section by executing p2: await turn=1, but since turn = 2, the process will loop indefinitely in its await statement until process q executes q4: turn1.*
  - *But process q is at q1: NCS and there is no assumption of progress for non-critical sections. Therefore, starvation has occurred: process p is continually checking the value of turn trying to enter its critical section, but process q need never leave its non-critical section, which is necessary if p is to enter its critical section*

Corretude

- Segunda tentativa

Algorithm 3.6. Second attempt

```
boolean wantp ← false, wantq ← false
```

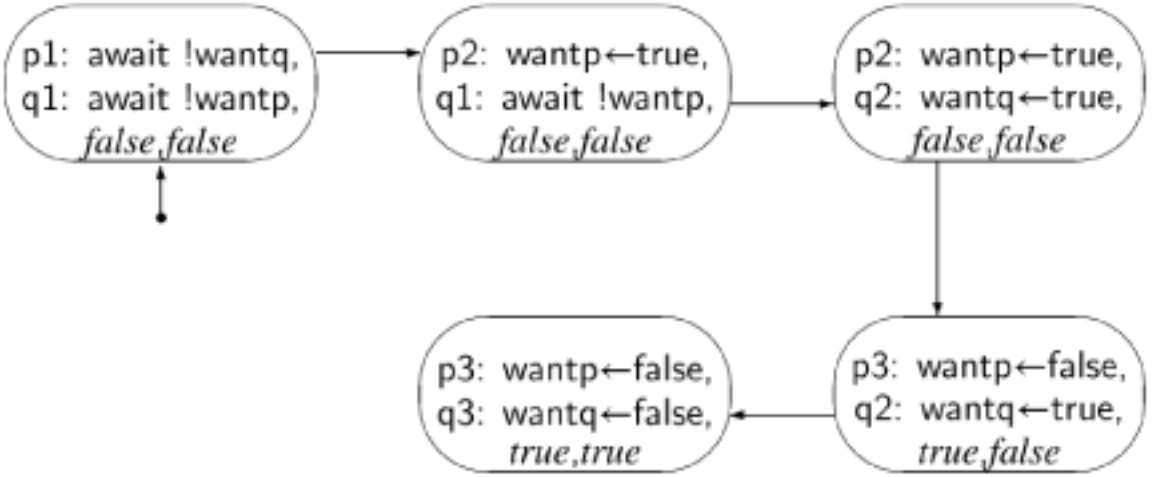
p	q
loop forever p1: non-critical section p2: await wantq = false p3: wantp ← true p4: critical section p5: wantp ← false	loop forever q1: non-critical section q2: await wantp = false q3: wantq ← true q4: critical section q5: wantq ← false

Let us construct a state diagram for the abbreviated algorithm:

Algorithm 3.7. Second attempt (abbreviated)

```
boolean wantp ← false, wantq ← false
```

p	q
loop forever p1: await wantq = false p2: wantp ← true p3: wantp ← false	loop forever q1: await wantp = false q2: wantq ← true q3: wantq ← false



Paths in the state diagram correspond to scenarios; this portion

Process p	Process q	wantp	wantq
p1: await wantq=false	q1: await wantp=false	*false*	*false*
p2: wantp←true	**q1: await wantp=false**	*false*	*false*
p2: wantp←true	q2: wantq←true	*false*	*false*
p3: wantp←false	**q2: wantq←true**	*true*	*false*
p3: wantp←false	q3: wantq←false	*true*	*true*

Problema: dois processos entram na seção crítica.
(perfeito interleaving)

Corretude

- Terceira tentativa

Algorithm 3.8. Third attempt

```
boolean wantp ⟵ false, wantq ⟵ false
```

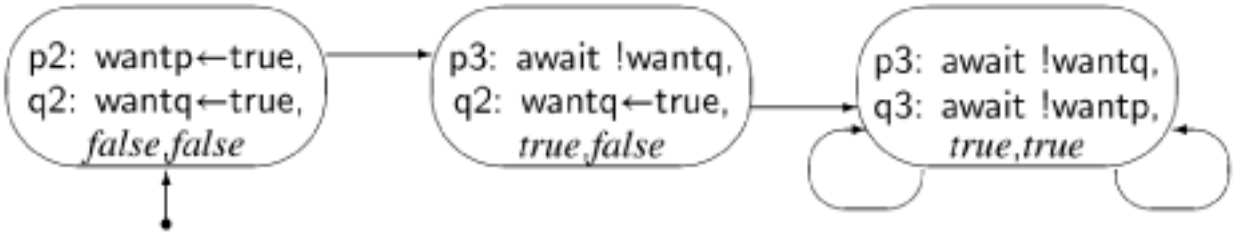
p	q
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: wantp ⟵ true	q2: wantq ⟵ true
p3: await wantq = false	q3: await wantp = false
p4: critical section	q4: critical section
p5: wantp ⟵ false	q5: wantq ⟵ false

Na terceira tentativa, o comando “await” passa a fazer parte da SC.

Corretude

Process p	Process q	wantp	wantq
p1: non-critical section	q1: non-critical section	*false*	*false*
p2: wantp←true	**q1: non-critical section**	*false*	*false*
p2: wantp←true	q2: wantq←true	*false*	*false*
p3: await wantq=false	**q2: wantq←true**	*true*	*false*
p3: await wantq=false	q3: await wantp=false	*true*	*true*

This can also be seen in the following fragment of the state diagram:



DEADLOCK!

Corretude

- Quarta tentativa

Algorithm 3.9. Fourth attempt

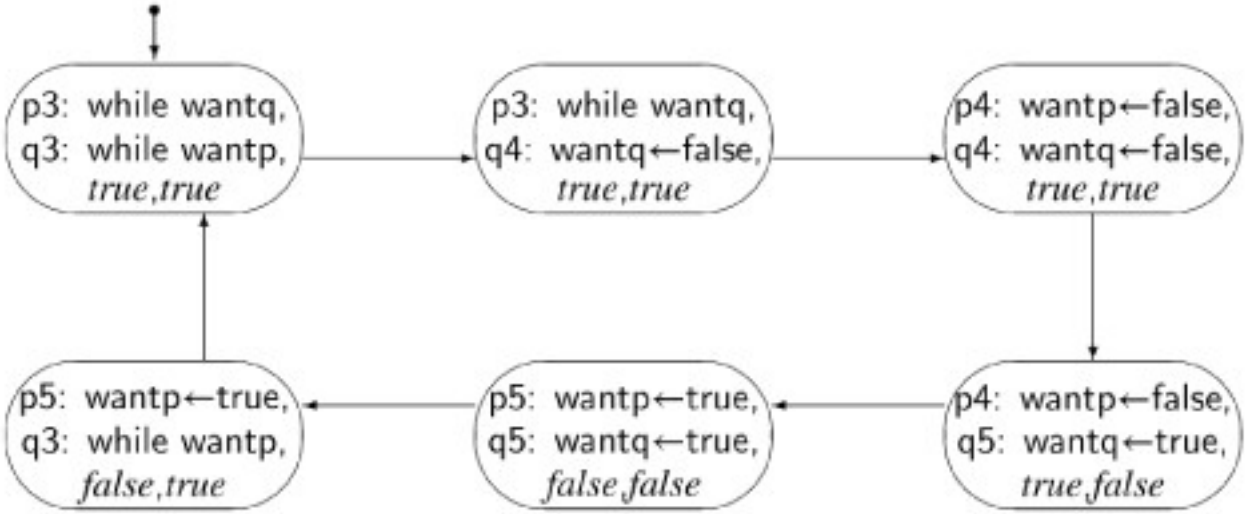
```
boolean wantp ← false, wantq ← false
```

p	q
loop forever p1: non-critical section p2: wantp ← true p3: while wantq p4: wantp ← false p5: wantp ← true p6: critical section p7: wantp ← false	loop forever q1: non-critical section q2: wantq ← true q3: while wantp q4: wantq ← false q5: wantq ← true q6: critical section q7: wantq ← false

Quando um processo entente que outro também quer entrar na SC, ele desiste.

Corretude

- Problema de Starvation (situação não realista)!



Dekker

Algorithm 3.10. Dekker's algorithm

```
boolean wantp ← false, wantq ← false
integer turn ← 1
```

p	q
`loop forever`	`loop forever`
`p1: non-critical section`	`q1: non-critical section`
`p2: wantp ← true`	`q2: wantq ← true`
`p3: while wantq`	`q3: while wantp`
`p4: if turn = 2`	`q4: if turn = 1`
`p5: wantp ← false`	`q5: wantq ← false`
`p6: await turn = 1`	`q6: await turn = 2`
`p7: wantp ← true`	`q7: wantq ← true`
`p8: critical section`	`q8: critical section`
`p9: turn ← 2`	`q9: turn ← 1`
`p10: wantp ← false`	`q10: wantq ← false`

Combinação da primeira e quarta tentativa

Corretude

Exercícios:

- 1,2,3,4,6
- Entregar na próxima aula.