PROBLEMA DA SEÇÃO CRÍTICA

Definição:

- N processos/*threads* formados por laço infinito contendo dois blocos de instruções:
  - Seção Crítica (SC)
  - Seção Não Crítica (SNC)

- Queremos uma solução que controle as execuções de modo que satisfaçam as seguintes condições:
  - Exclusão mútua
  - Livre de *deadlock* e *starvation* individual

Exclusão mútua:

- Instruções da SC de mais de 1 processo não se intercalam
- Apenas um processo está na seção crítica por vez

Livre de *deadlock*:

- Se não há processo na seção crítica
- Se vários processos tentam entrar na seção crítica
  - Um deles deve conseguir entrar na SC

Livre de *starvation* (postergação indefinida) individual:

- Se um processo tenta entrar na SC
  - Em algum momento ele deve conseguir

Mecanismo de Sincronização:

- Instruções usadas para garantir as condições anteriores
- Divididas em:
  - Pré-protocolo:
    - Instruções que antecedem a seção crítica
  - Pós-protocolo:
    - Instruções que sucedem a seção crítica

Algoritmo para o Problema da seção crítica com 2 processos:

Variáveis globais

Processo 1	**Processo 2**
Variáveis locais	Variáveis locais
Laço infinito	Laço infinito
Seção não crítica	Seção não crítica
Pré-protocolo	Pré-protocolo
Seção Crítica	Seção Crítica
Pós-protocolo	Pós-protocolo
Fim_laço	Fim_laço

Obs.:

1. É assumido que os protocolos podem usar variáveis globais e/ou locais, mas não as usadas nas seções crítica e não crítica, e vice-versa.

2. As seções críticas devem terminar, ou seja, prosseguir para o pós-protocolo

3. As seções não críticas podem não terminar, no sentido de que podem encerrar o programa, entrar num laço infinito ou seguir para o pré-protocolo.

PRIMEIRA TENTATIVA DE SOLUÇÃO

int turn = 1

	Processo P		**Processo Q**
	Laço infinito		Laço infinito
p1	Seção não crítica	q1	Seção não crítica
p2	Espere turn = 1	q2	Espere turn = 2
p3	Seção Crítica	q3	Seção Crítica
p4	turn = 2	q4	turn = 1
	Fim_laço		Fim_laço

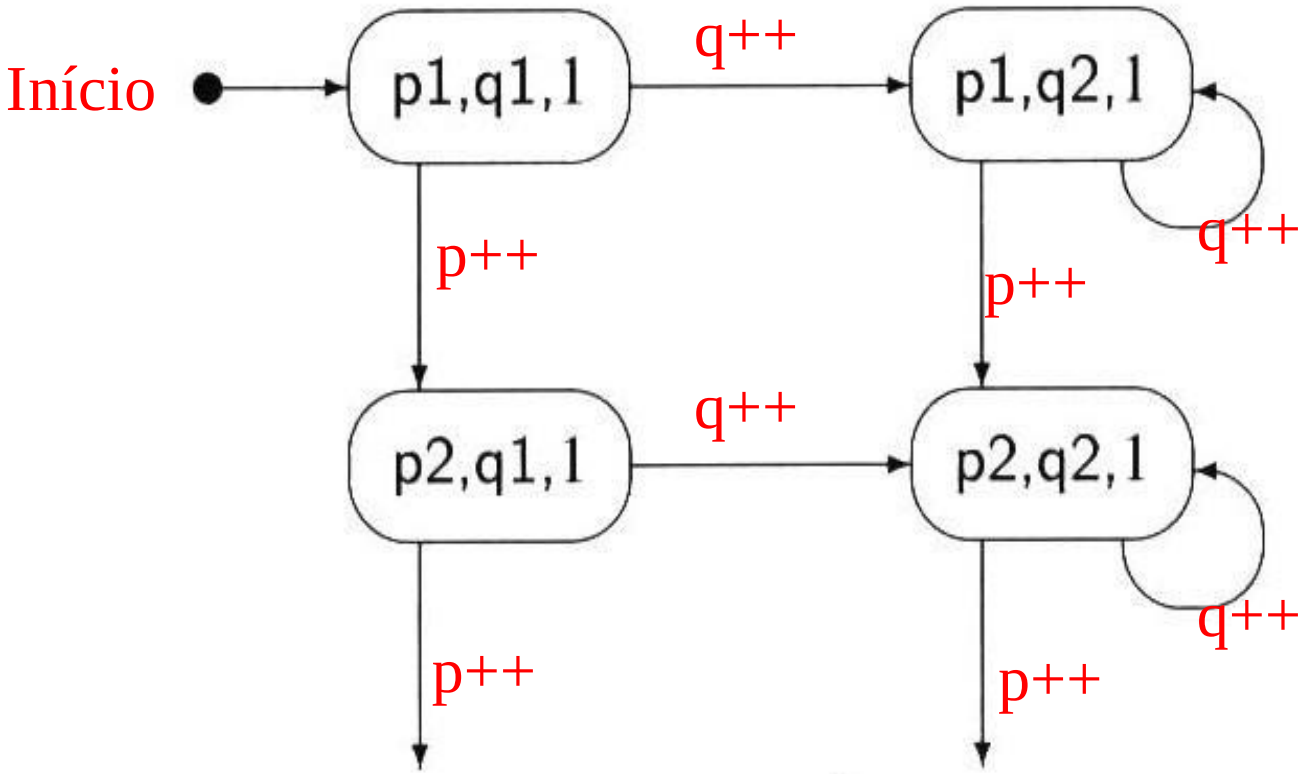
Correção:

- Para verificar a correção devemos construir o diagrama de estados
- Como o mecanismo de sincronização não usa as variáveis das seções críticas e não críticas:
  - Os estados devem conter apenas as variáveis do mecanismo de sincronização e os apontadores
  - O estado pode ser descrito como $\left(p_j, q_k, turn\right)$
- O algoritmo estará correto do ponto de vista da exclusão mútua se os estados:

$$\left(p_3, q_3, 1\right) \text{ e } \left(p_3, q_3, 2\right)$$

não aparecem no diagrama.

Fragmento inicial do diagrama



Notação: p++: Executa instrução de P

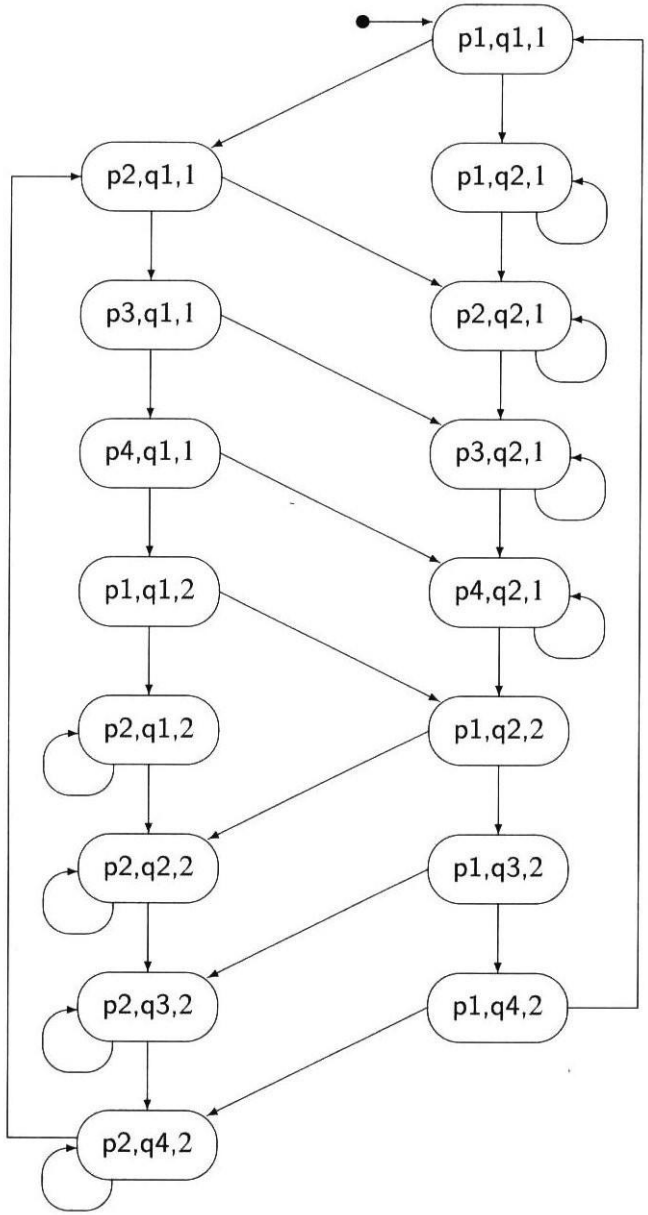
q++: Executa instrução de Q

Diagrama de estados completo:

Não aparecem os estados

$(p_3, q_3, 1)$ e $(p_3, q_3, 2)$

Logo, o algoritmo implementa a exclusão mútua.



Simplificando o diagrama de estados

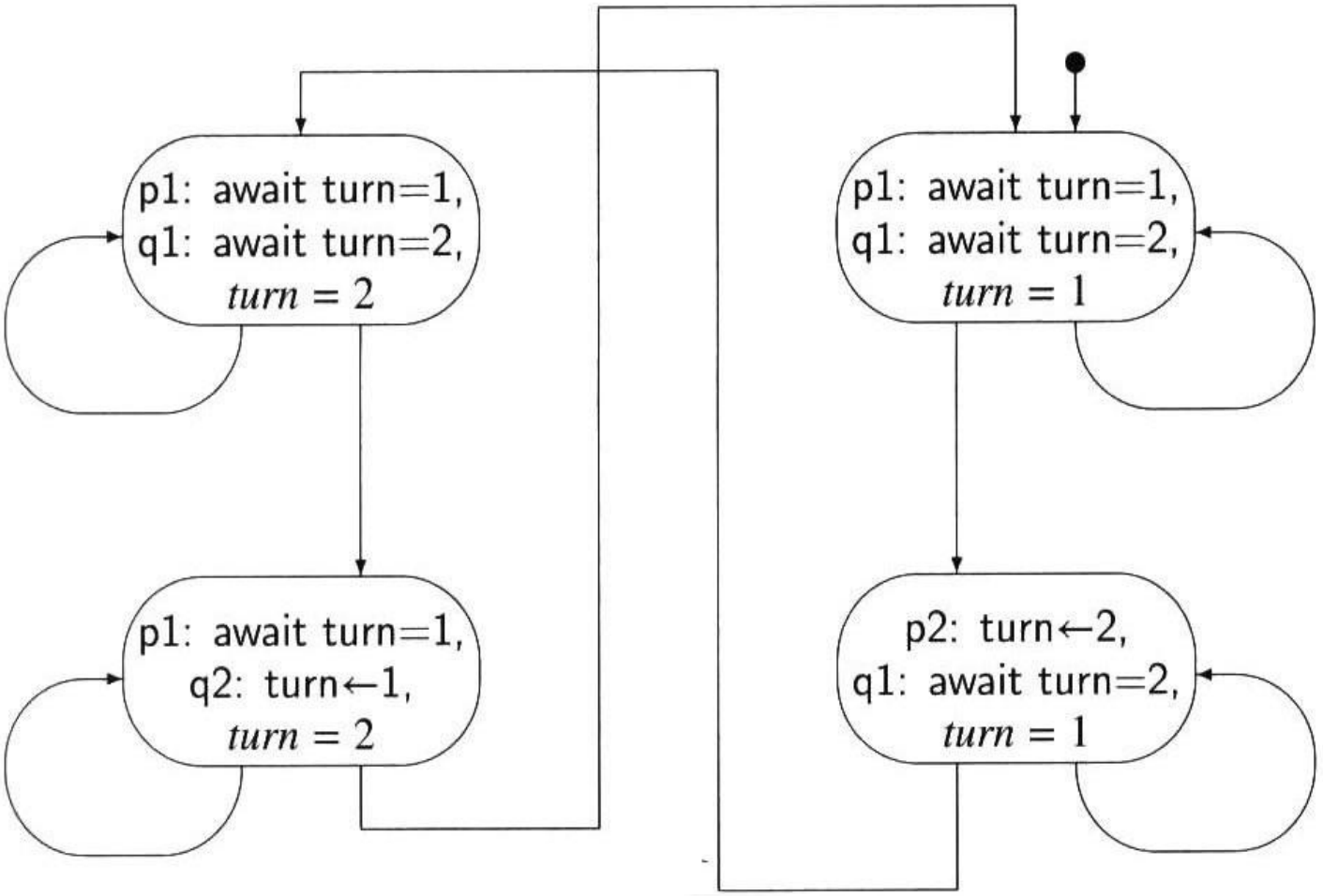
- Observando o código do algoritmo P, vemos que:
  - A seção não crítica é encerrada quando efetivamente se executa a instrução p_2
  - Semelhantemente, a seção crítica é encerrada quando se executa a instrução p_4
- Desta forma, podemos eliminar as instruções referentes às seções críticas e não críticas e verificar a existência de estados para as instruções seguintes

Código simplificado:

int turn = 1

	Processo P		**Processo Q**
	Laço infinito		Laço infinito
p1	Espere turn = 1	q1	Espere turn = 2
p2	turn = 2	q2	turn = 1
	Fim_laço		Fim_laço

Diagrama de estados do código simplificado:



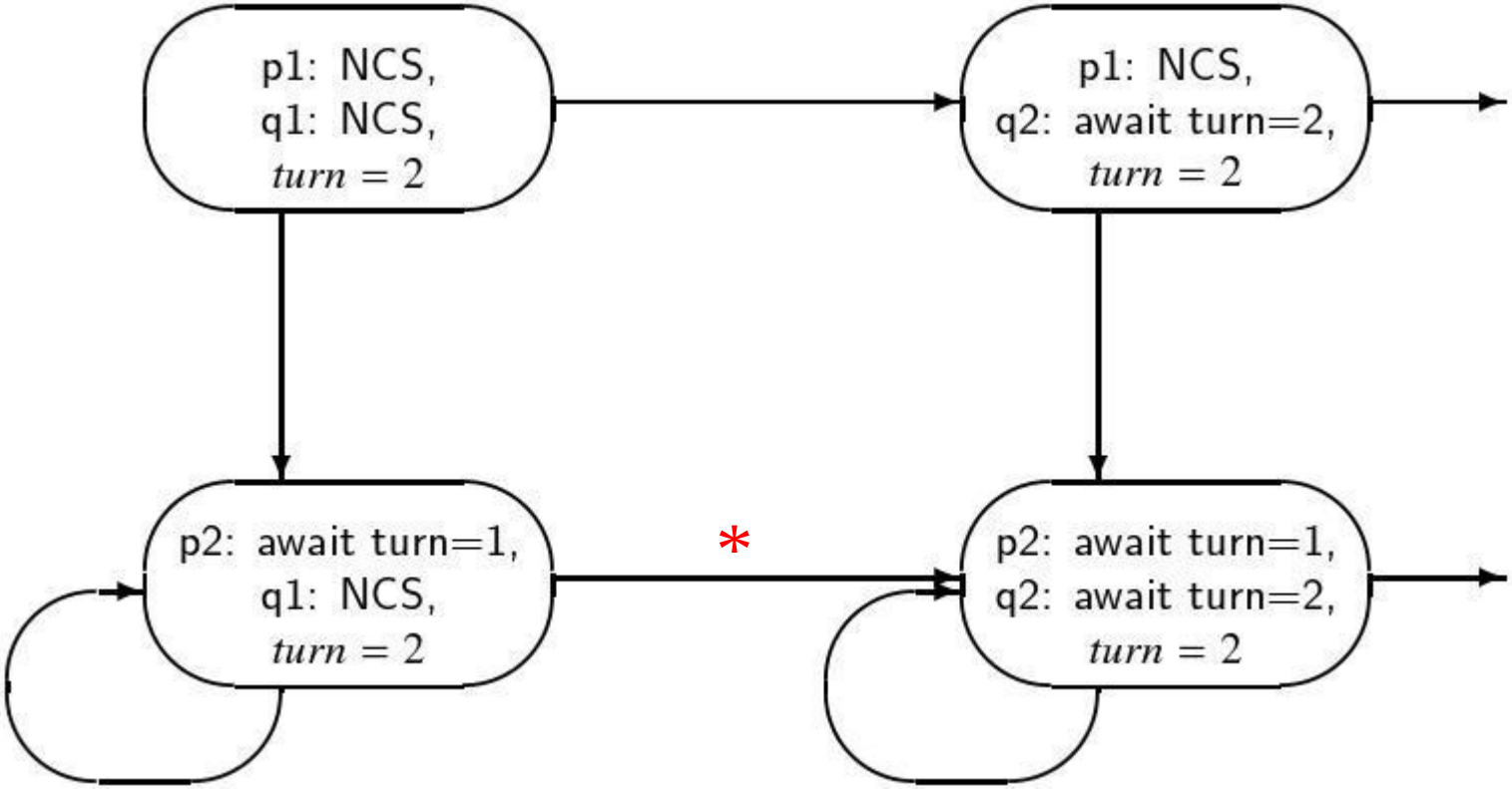
Não existem estados $(p_2, q_2, 1)$ e $(p_2, q_2, 2)$ => Exclusão mútua.

Análise de *deadlock:*

- É preciso analisar os estados onde os processos esperam executar o pré-protocolo
- No diagrama simplificado:
  - Estado $(p_1, q_1, 2)$: Q entra na seção crítica
  - Estado $(p_1, q_1, 1)$: P entra na seção crítica
  - Estado $(p_1, q_2, 2)$: A seção crítica de Q deve terminar. Logo, ocorre o pós-protocolo e temos o estado $(p_1, q_1, 1)$
  - Estado $(p_2, q_1, 1)$: A seção crítica de P deve terminar. Logo, ocorre o pósprotocolo e temos o estado $(p_1, q_1, 2)$
  - O algoritmo é livre de *deadlock*

Análise de *starvation*:

- Considerando o diagrama de estados não simplificado:



- A execução indicada por * só ocorre se a seção não crítica terminar

- Como assumimos que não há obrigatoriedade de término das seções não críticas

  - O algoritmo não está livre de *starvation.*

SEGUNDA TENTATIVA DE SOLUÇÃO

Considere o algoritmo:

boolean wantp = false, wantq = false

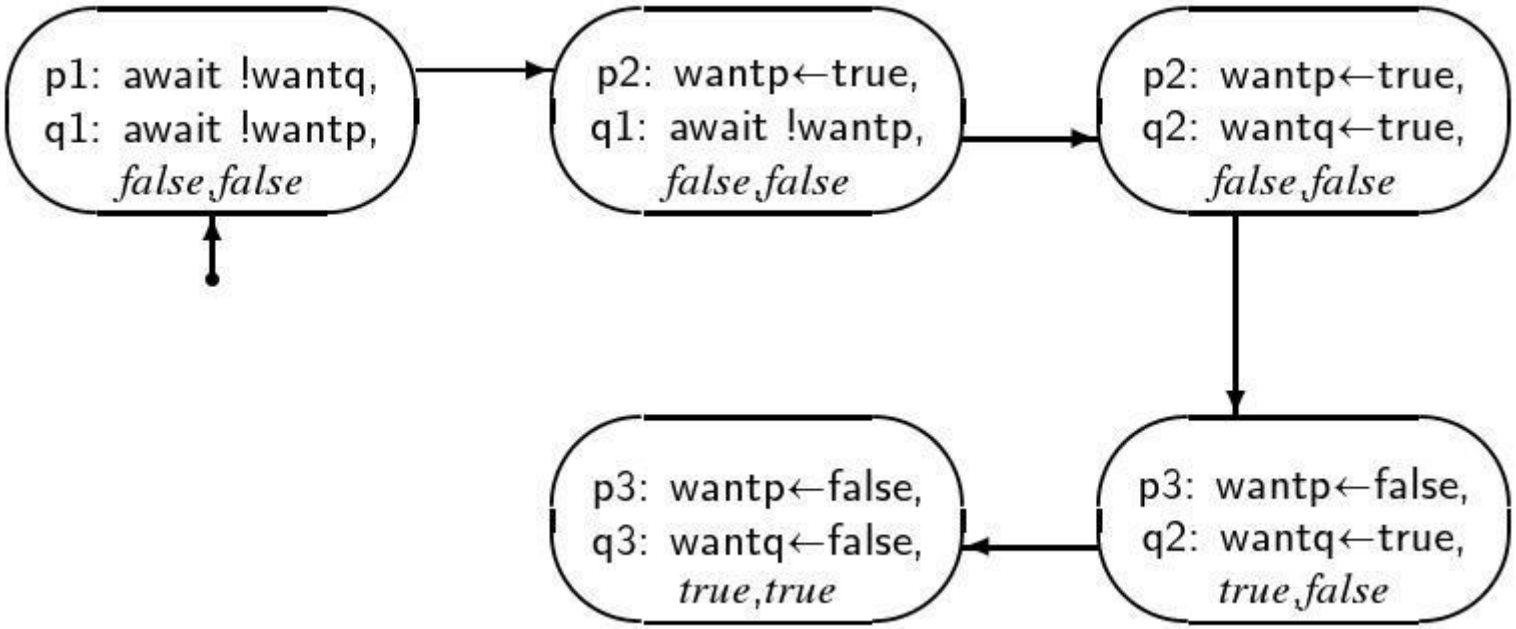
	Processo P		**Processo Q**
	Laço infinito		Laço infinito
p1	Seção não crítica	q1	Seção não crítica
p2	Espere wantq = false	q2	Espere wantp = false
p3	wantp = true	q3	wantq = true
p4	Seção Crítica	q4	Seção Crítica
p5	wantp = false	q5	wantq = false
	Fim_laço		Fim_laço

Algoritmo simplificado:

boolean wantp = false, wantq = false

	Processo P		**Processo Q**
	Laço infinito		Laço infinito
p1	Espere wantq = false	q1	Espere wantp = false
p2	wantp = true	q2	wantq = true
p3	wantp = false	q3	wantq = false
	Fim_laço		Fim_laço

Fragmento do diagrama de estados:



- Como aparece o estado $(p_3, q_3, true, true)$ não há garantia de exclusão mútua.

TERCEIRA TENTATIVA DE SOLUÇÃO

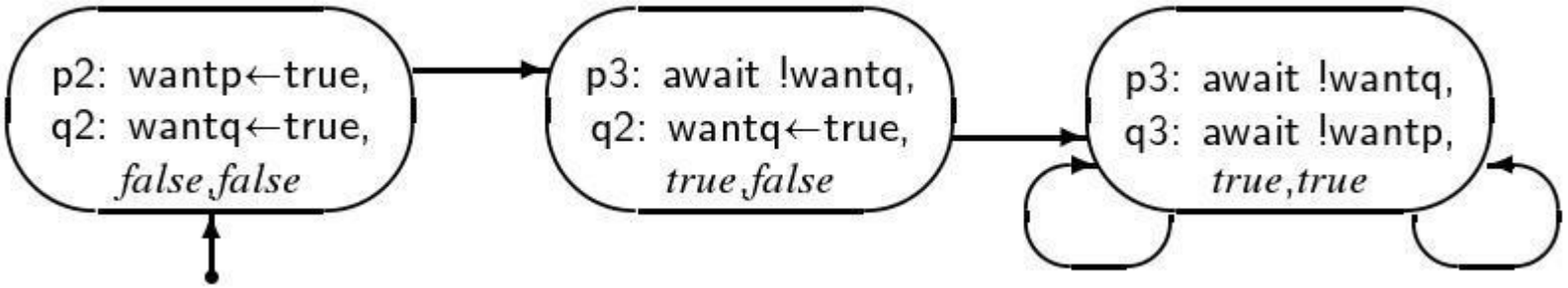
Considere o algoritmo:

boolean wantp = false, wantq = false

	Processo P		**Processo Q**
	Laço infinito		Laço infinito
p1	Seção não crítica	q1	Seção não crítica
p2	wantp = true	q2	wantq = true
p3	Espere wantq = false	q3	Espere wantp = false
p4	Seção Crítica	q4	Seção Crítica
p5	wantp = false	q5	wantq = false
	Fim_laço		Fim_laço

- Ideia:
  - A instrução de espera faz “parte” da seção crítica
  - Deve ser a instrução imediatamente anterior à SC

- Este algoritmo satisfaz a exclusão mútua (Exercício)
- **Não satisfaz** a condição de ser livre de *deadlock*, observe o seguinte fragmento de diagrama:



- Estado $\langle p_3, q_3, true, true \rangle$ não tem sequencia.

Exercício: Construa o diagrama de estados do algoritmo simplificado e mostre que o estado $(p_3, q_3, \cdot, \cdot)$ não ocorre.

boolean wantp = false, wantq = false

	Processo P		**Processo Q**
	Laço infinito		Laço infinito
p1	wantp = true	q1	wantq = true
p2	Espere wantq = false	q2	Espere wantp = false
p3	wantp = false	q3	wantq = false
	Fim_laço		Fim_laço

QUARTA TENTATIVA DE SOLUÇÃO

Considere o algoritmo:

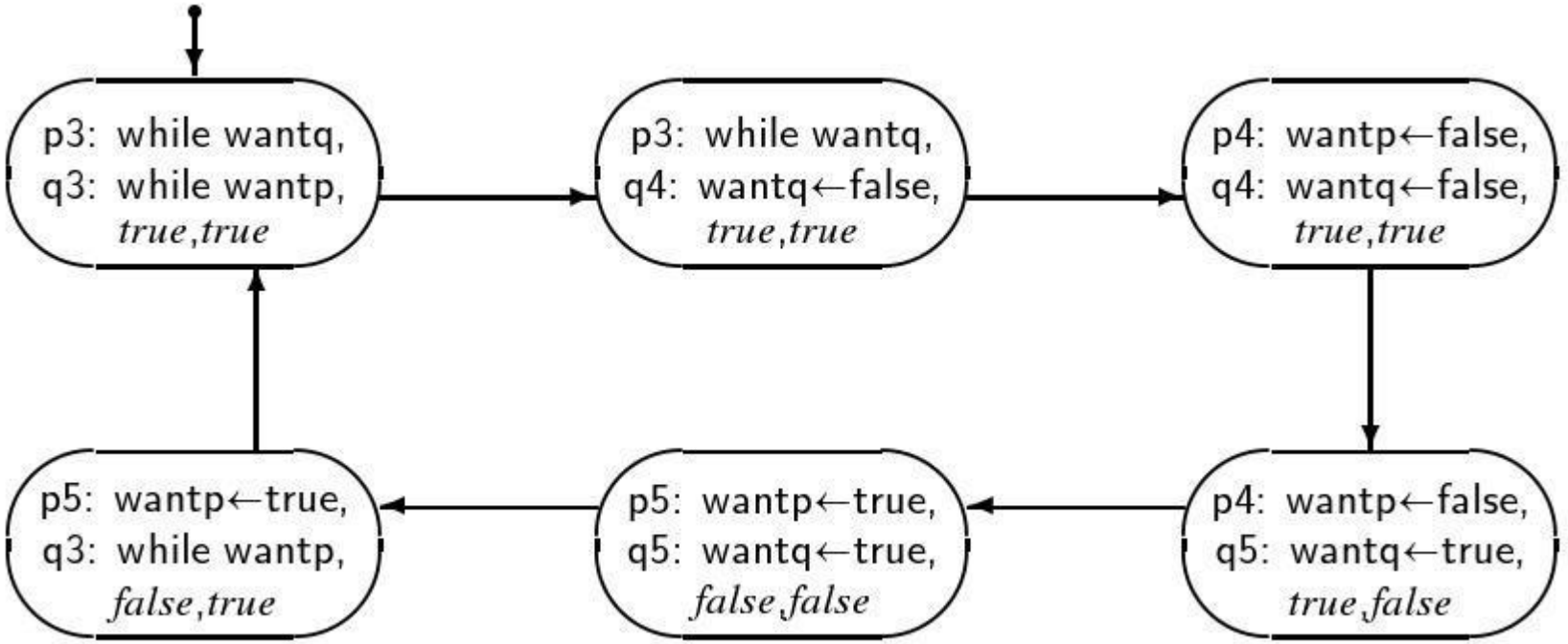
boolean wantp = false, wantq = false

	Processo P		**Processo Q**
	Laço infinito		Laço infinito
p1	Seção não crítica	q1	Seção não crítica
p2	wantp = true	q2	wantq = true
p3	while wantq	q3	while wantp
p4	wantp = false	q4	wantq = false
p5	wantp = true	q5	wantq = true
p6	Seção Crítica	q6	Seção Crítica
p7	wantp = false	q7	wantq = false
	Fim_laço		Fim_laço

- A ideia é que a alternância das travas *wantp* e *wantq*, com o intercalamento arbitrário, evite o *deadlock*.
- De fato:
  - Este algoritmo garante a exclusão mútua e evita o *deadlock*. (Exercício)

Análise de *starvation*:

- Não há garantia de estar livre de *starvation*
- Considere o caso em que há o perfeito intercalamento das respectivas instruções de P e Q, como mostra o fragmento de diagrama:



- O fragmento de diagrama mostra que, neste caso, nenhum dos dois processos chega a executar a SC
  - Este é uma situação rara, mas não pode ser descartada

ALGORITMO DE DEKKER

boolean wantp = false, wantq = false; int turn=1

	Processo P		**Processo Q**
	Laço infinito		Laço infinito
p1	Seção não crítica	q1	Seção não crítica
p2	wantp = true	q2	wantq = true
p3	while wantq	q3	while wantp
p4	Se (turn==2)	q4	Se (turn==1)
p5	wantp = false	q5	wantq = false
p6	Espere (turn==1)	q6	Espere (turn==2)
p7	wantp = true	q7	wantq = true
p8	Seção Crítica	q8	Seção Crítica
p9	turn=2	q9	turn=1
p10	wantp = false	q10	wantq = false
	Fim_laço		Fim_laço

- O algoritmo é uma junção da primeira e da quarta tentativa
- As variáveis *wantp* e *wantq* registram a intenção do respectivo processo em entrar na Seção Crítica
- Caso o outro processo tenha demonstrado a intenção de também entrar
  - O processo verifica a liberação do acesso dado pela variável *turn*

- O algoritmo de Dekker
  - Garante a exclusão mútua
  - É livre de *deadlock*
  - É livre de *starvation*

INSTRUÇÕES ATÔMICAS COMPLEXAS

- A dificuldade em resolver o problema da Seção Crítica é devido à separação das instruções de leitura e armazenamento
  - Esta separação permite o intercalamento
- Com instruções atômicas mais complexas, a solução torna-se mais simples

- Considere a instrução atômica Test_and_set, equivalente à:

Test_set(global, local)

local=global

global=1

- Neste caso, o algoritmo fica:

int trava = 0

	Processo P		**Processo Q**
	int local1;		int local2;
	Laço infinito		Laço infinito
p1	Seção não crítica	q1	Seção não crítica
	repeat		repeat
p2	Test_set(trava, local1)	q2	Test_set(trava, local2)
p3	until (local1==0)	q3	until (local2==0)
p4	Seção Crítica	q4	Seção Crítica
p5	trava = 0	q5	trava = 0
	Fim_laço		Fim_laço

- Outras instruções atômicas complexas que podem ser usadas são:
  - **Troca (a, b)**

    int temp;

    temp=a;

    a=b;

    b=temp;

  - **Busca_soma(global, local, x)** local=global

    global=global + x

- **Compare_Troque(global, antigo, novo)**

```
int temp;
temp = global;
Se (global == antigo)
        global = novo
retorne temp
```