Models and Peterson's Algorithm

This is Peterson's algorithm:

```
boolean[] enter = {false, false};   int yield = 0 || 1;
```

thread t_0

```
1 while (true) {
2  // entry protocol
3  enter[0] = true;
4  yield = 0;
5  await (!enter[1]
         || yield != 0);
6  critical section { ... }
7  // exit protocol
8  enter[0] = false;
9 }
```

thread t_1

```
while (true) {                  10
 // entry protocol              11
 enter[1] = true;               12
 yield = 1;                     13
 await (!enter[0]               14
        || yield != 1);
 critical section { ... }       15
 // exit protocol               16
 enter[1] = false;              17
}                               18
```

The successors of $\langle yield = 0, \triangleright 6, enter[0] = T, \triangleright 14, enter[1] = T \rangle$ are:

1. The t_0 successor is $\langle yield = 0, \triangleright 3, enter[0] = F, \triangleright 14, enter[1] = T \rangle$.
2. The t_0 successor is $\langle yield = 0, \triangleright 8, enter[0] = T, \triangleright 14, enter[1] = T \rangle$.
3. The t_1 successor is $\langle yield = 0, \triangleright 6, enter[0] = T, \triangleright 15, enter[1] = T \rangle$.
4. There is no t_1 successor.

This is Peterson's algorithm:

```
boolean[] enter = {false, false};   int yield = 0 || 1;
```

thread t_0

```
1  while (true) {
2   // entry protocol
3   enter[0] = true;
4   yield = 0;
5   await (!enter[1]
          || yield != 0);
6   critical section { ... }
7   // exit protocol
8   enter[0] = false;
9  }
```

thread t_1

```
10 while (true) {
11  // entry protocol
12  enter[1] = true;
13  yield = 1;
14  await (!enter[0]
          || yield != 1);
15  critical section { ... }
16  // exit protocol
17  enter[1] = false;
18 }
```

The successors of $\langle yield = 0, \triangleright 6, enter[0] = T, \triangleright 14, enter[1] = T \rangle$ are:

1. The t_0 successor is $\langle yield = 0, \triangleright 3, enter[0] = F, \triangleright 14, enter[1] = T \rangle$.
2. The t_0 successor is $\langle yield = 0, \triangleright 8, enter[0] = T, \triangleright 14, enter[1] = T \rangle$.
3. The t_1 successor is $\langle yield = 0, \triangleright 6, enter[0] = T, \triangleright 15, enter[1] = T \rangle$.
4. There is no t_1 successor.

How can the t_1 successor of
$\langle \mathit{yield} = 0, \triangleright 6, \mathit{enter}[0] = T, \triangleright 14, \mathit{enter}[1] = T \rangle$ be
$\langle \mathit{yield} = 0, \triangleright 6, \mathit{enter}[0] = T, \triangleright 15, \mathit{enter}[1] = T \rangle$? Both threads are in their critical sections!!!

- State $\langle \mathit{yield} = 0, \triangleright 6, \mathit{enter}[0] = T, \triangleright 14, \mathit{enter}[1] = T \rangle$ does not exist.
- There is a bug in Peterson's algorithm.
- The previous slide was wrong.
- State $\langle \mathit{yield} = 0, \triangleright 6, \mathit{enter}[0] = T, \triangleright 14, \mathit{enter}[1] = T \rangle$ is never entered.

How can the t_1 successor of
$\langle \mathit{yield} = 0, \triangleright 6, \mathit{enter}[0] = T, \triangleright 14, \mathit{enter}[1] = T \rangle$ be
$\langle \mathit{yield} = 0, \triangleright 6, \mathit{enter}[0] = T, \triangleright 15, \mathit{enter}[1] = T \rangle$? Both threads are in their critical sections!!!

- State $\langle \mathit{yield} = 0, \triangleright 6, \mathit{enter}[0] = T, \triangleright 14, \mathit{enter}[1] = T \rangle$ does not exist.
- There is a bug in Peterson's algorithm.
- The previous slide was wrong.
- State $\langle \mathit{yield} = 0, \triangleright 6, \mathit{enter}[0] = T, \triangleright 14, \mathit{enter}[1] = T \rangle$ is never entered.

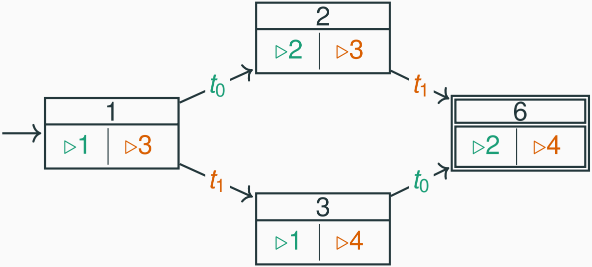
What does Peterson's algorithm achieve?

1. Mutual exclusion using only atomic reads and writes
2. Mutual exclusion and first-come-first-served fairness
3. Mutual exclusion using busy waiting
4. Mutul exclusion using test-and-set operations

What does Peterson’s algorithm achieve?

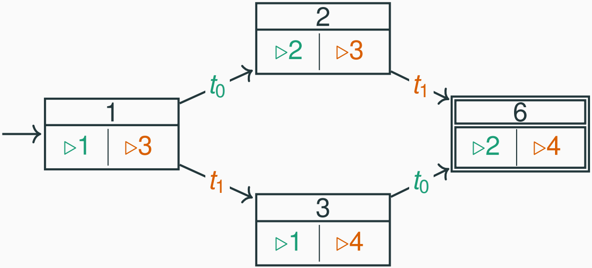
1. Mutual exclusion using only atomic reads and writes
2. Mutual exclusion and first-come-first-served fairness
3. Mutual exclusion using busy waiting
4. Mutul exclusion using test-and-set operations

What properties does the following state/transition diagram show?



1. No deadlocks can occur
2. There are no race conditions
3. No starvation can occur, but deadlocks may occur
4. Neither deadlocks nor race conditions may occur

What properties does the following state/transition diagram show?



1. No deadlocks can occur
2. There are no race conditions
3. No starvation can occur, but deadlocks may occur
4. Neither deadlocks nor race conditions may occur

Which of the following are strategies to avoid deadlocks?

1. Using locks
2. Requiring that all threads acquire locks in the same order
3. Limiting the amount of concurrency
4. Using counting semaphores instead of binary semaphores

Which of the following are strategies to avoid deadlocks?

1. Using locks
2. Requiring that all threads acquire locks in the same order
3. Limiting the amount of concurrency
4. Using counting semaphores instead of binary semaphores