HÁSKÓLINN Í REYKJAVÍK
REYKJAVIK UNIVERSITY

Benjamín Aage B. Birgisson

Exercise 5.1.1:

(a)

- P1 is holding 1 unit of R1, and requesting 1 unit of R2.
- P2 is holding 1 unit of R1, and requesting 1 unit of R2.
- P3 if requesting 1 unit of R2, 1 unit of R3, and is holding 1 unit of R4.
- P4 is holding 1 unit of R3, and requesting 1 unit of R4.

P1 should be fine, as it is requesting a resource (R2) at the same time as P2 and P3. As P1 has highest priority of them all, it should get the resource.

P2 is the same as P1, as it is requesting a resource (R2) at the same time as P1 and P3. As P2 has higher priority than P3, and the resource has allocated P1 one of its resources, currently holding 1 resource free, the free resource should be allocated to P2.

P3 is blocked, as it is requesting a resource (R3) which has all its units already allocated, as well as resource R2.

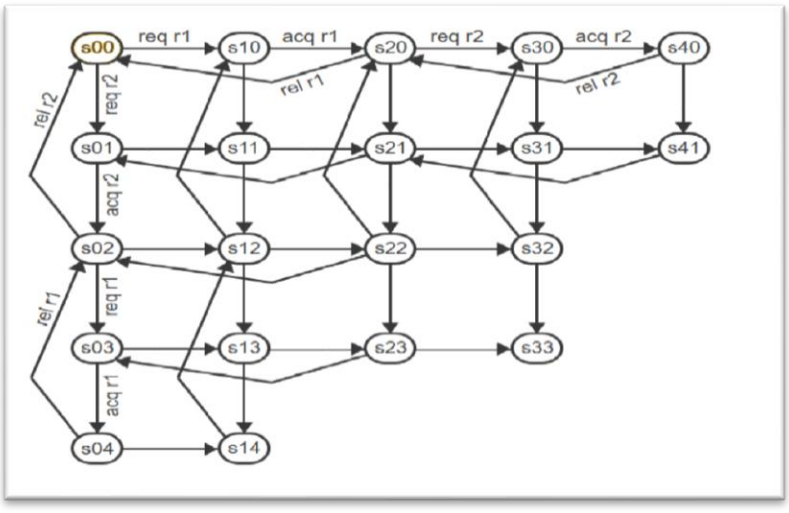
P4 is blocked, as it is requesting a resource (P4) which has all its units already allocated.

(b)

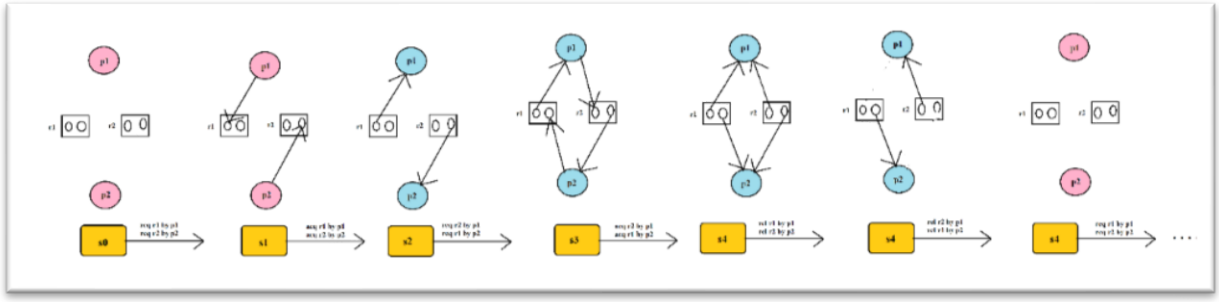
- Process P3 and process P4 are deadlocked, as they are both requesting the same two resources, where one is holding 1 resource and being blocked on the other, and vice versa for the other resource.

Exercise 5.1.3:

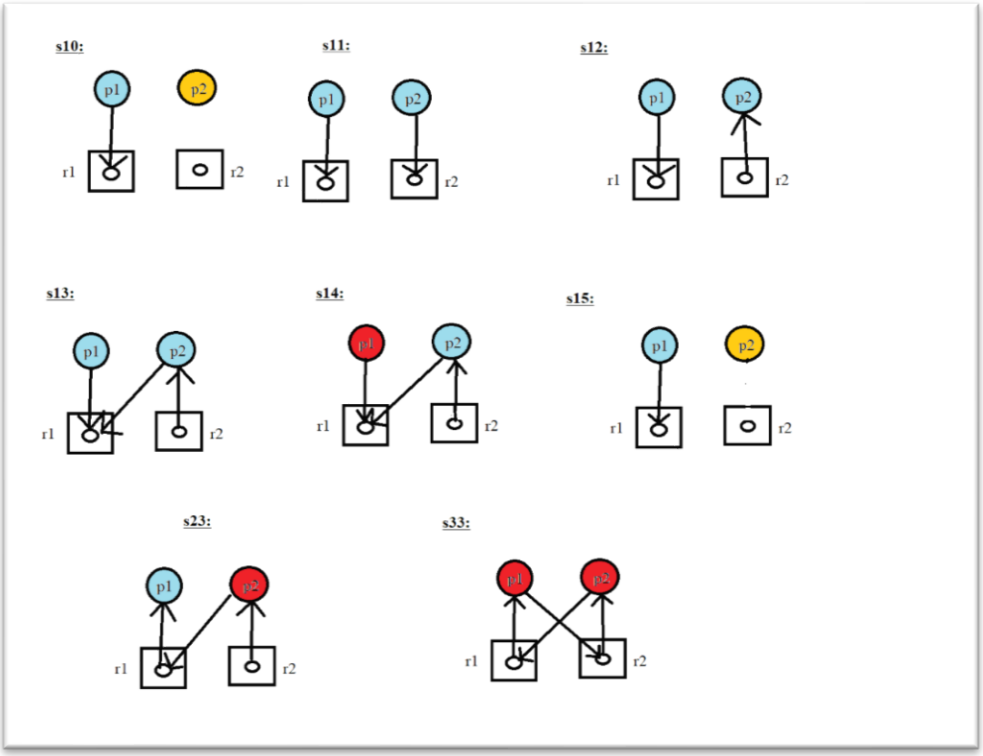
(a)



…. Or(?):

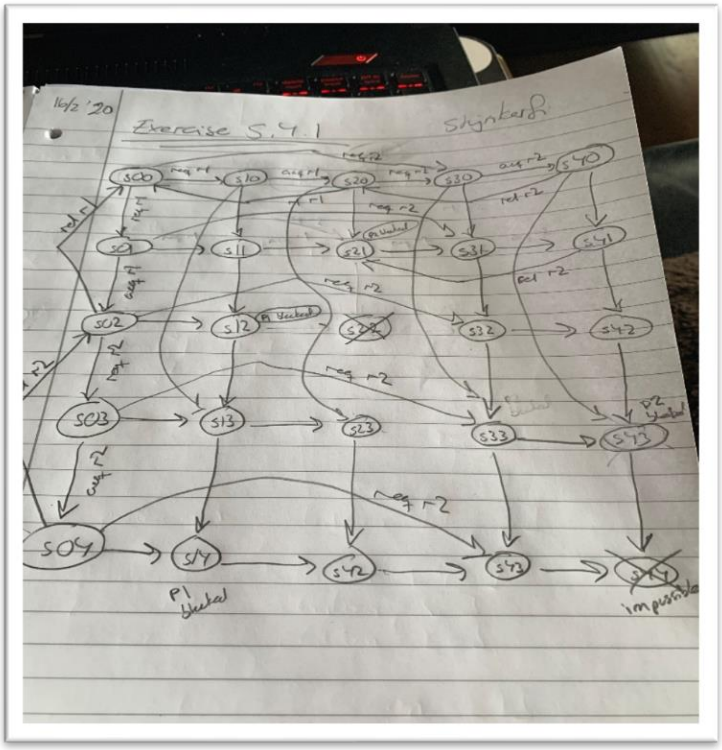


(b)



Exercise 5.1.4:

(a)



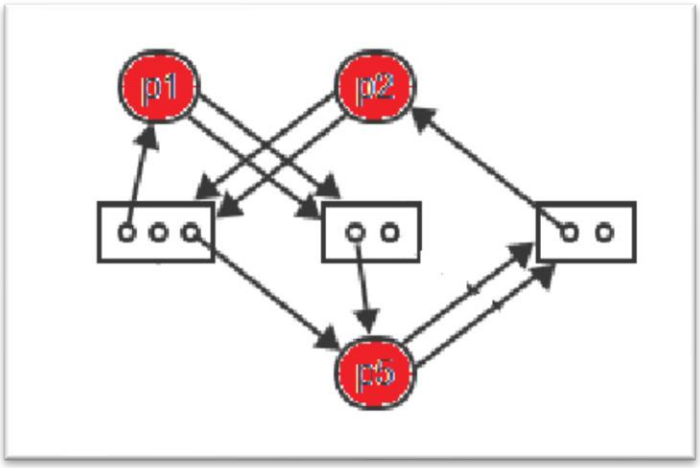
(b)

- P1 is blocked in states: s12, s14
- P2 is blocked in states: s21, s43
- Deadlock in state: s42

Exercise 5.2.1:

(a)

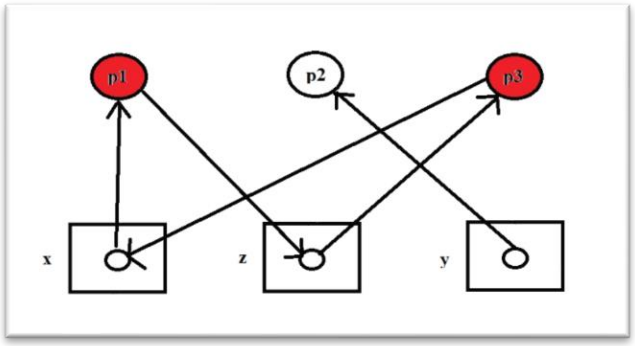
- Remove p6, unblocking p3
- Remove p3, unblocking none
- Remove p4, unblocking none
- … cannot be reduced more, resulting in a deadlock within the given graph:



Exercise 5.2.2:

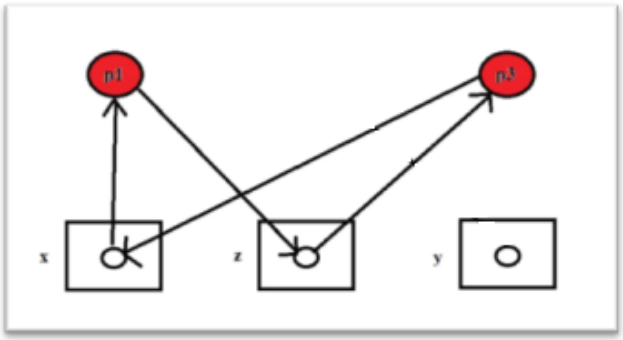
(a)

- This process would lead to a deadlock state, as there is a cycle between process p1 and process p3



(b)

- Removing p2 from the graph above, which is the only possible move, shows that the another process cannot be removed from the graph as there is a cycle between allocating and requesting a resource between process p1 and process p3 (as stated above).



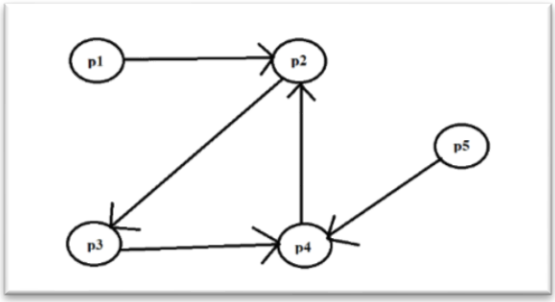
(c)

- Yes, by initializing semaphore z to a greater value then 1 would avoid a deadlock. Then process p1 could be removed from the graph (currently running), and as a result freeing up space for process p3 to run.

Exercise 5.2.3:

(a)

- Yes, as there is a cycle between processes p2, p3 and p4.



Exercise 5.3.1:

(a)

- request r2,2 by p2 (0 unit of r2 left)
- request r1,1 by p1 (2 unit of r1 left)
- request r1,3 by p2 (requested 3 units, only two units left in r1)
  → p2 becomes blocked
- request r2,1 by p1 (requested 1 unit, none is left in r2)
  → p1 become blocked
  → Deadlock state

(b)

- Say, that in state s0 process p1 has acquired 1 unit of resource in resource r1, and process p2 has acquired 1 unit of resource in resource r1 as well.
- Then, in state s1 both processes p1 and p2 make another request for 1 unit of resource r1, which only has 1 unit left free.

- By the banker's algorithm, if the request of process p1 is treated as a request edge, process 2 then becomes blocked. Then, by removing p1 unblocks p2. Since the graph is completely reducible hereby, the new state s1 is accepted as a valid state.

Exercise 5.3.2:

(a)

- As there are only 100 units available in R, and there has already been allocated 65 of these 100 units, there is no possibility to prevent a deadlock within the use of the banker's algorithm. For example, if the claim of process p2 is treated as allocated edge, there is still 15 units left unallocated that make the processor blocked (and processor p1 blocked as well in the process). And same, if the claim of process p1 is treated as allocated edge, there is still 25 units unallocated, which makes both process p1 and p2 blocked.

(b)

- - Yes, both requests of the two new processes can be granted, as there is 65 units allocated out of 100 units in total, making 35 units still free.
- The process p3 has a request of 15 units, which then can be granted and then removed, making the process p4 available for a request of 25 units, which is granted and can thereafter be removed as well.

Exercise 5.4.2:

(a)

- With the ordered resources policy, all resources are ordered:

$$seq(r1) < seq(r2) < seq(r3) < seq(r4).$$

- Therefore, the following access sequences would violate an ordered resource policy and could lead to a deadlock:
  - Process 1: 2, 3, 5, 6
  - Process 2: 2, 3, 4, 5