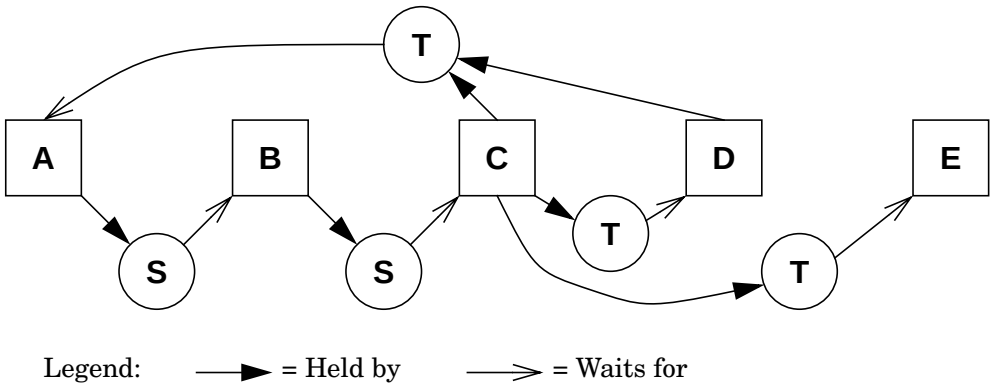
Exercise session 4 – Deadlock: Solutions

Deadlock analysis

Among the conditions for a deadlock to occur, it is *circular hold-wait* which we hope is not fulfilled. To check this, we draw the resource allocation graph which looks like:



Legend: ⟶ = Held by ⟶ = Waits for

a) The graph contains one cycle. There is therefore one way in which the system may deadlock (but we do not know it actually will since that depends on timing, sheduling, logics of the application, etc.).

b) In the graph we see that circular hold-wait requires two S threads and one T thread (at least). In other words, deadlock **cannot occur** if there is only one thread S. In the code we see that (assume T is interrupted in useACD) S can either be at useAB which means that T is blocked but S can run because it does not need C, or S is at useBC where it gets blocked by T which can run since A is not used by S.

c) If there are at least two threads of type S, one may be executing useAB and the other may be executing useBC. If that happens when T is executing useACD, the system will deadlock!

d) The simplest change to avoid deadlock is to let T allocate A first of all. Then both threads allocate resources in the same order (A,B,..E) which prevents deadlock. However, a possible problem is then that useAB in S and useCD in T will never run concurrently which may cause problems in the application. A better alternative probably is to use the allocation order BCDEA which means that B is reserved before A in the beginning of S. This solution does not impose restrictions on the concurrency of the system.