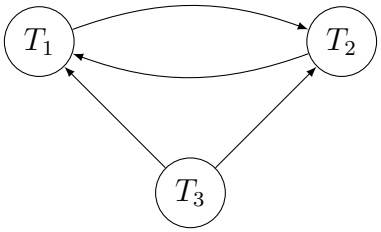
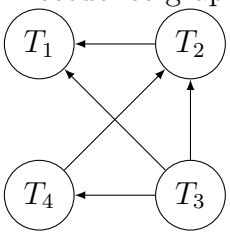
Problem 1

S1: Q1: Precedence graph:



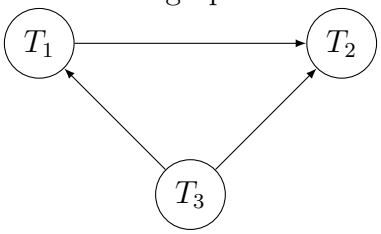
Q2: This schedule is not conflict-serializable because the precendence graph for the schedule is cyclic.

S2: Q1: Precedence graph:



Q2: This schedule is conflict-serializable because the precendence graph for the schedule is acyclic. An equivalent serial schedule would be T_3, T_4, T_2, T_1.

S3: Q1: Precedence graph:



Q2: This schedule is conflict-serializable because the precendence graph for the schedule is acyclic. An equivalent serial schedule would be T_3, T_1, T_2.

Problem 2

Q1: S_1 is not a valid schedule, because in the fifth action T_2 tries to lock object B while T_1 already has a lock on it.

S_2 is not a valid schedule, because in the fifth action T_1 tries to unlock object B, when it does not own a lock for object B.

Q2: T_1 in S_1 is well-formed, because all of its read/write actions are performed after locking the object and before unlocking the object.

T_1 in S_2 is not well-formed, bceause in the third action it tries to write to object B before it has a lock on that object.

Problem 3

This sequence follows 2PL, because neither transaction performs an unlock before it has all its locks.

Problem 4

(a) Undo <T, A, 10> by writing 10 into A on the disk, then write <Abort U> to the log, then write <Abort T> to the log.

(b) Assuming <Commit U> in the log means U has already been committed and written to disk. Write 30 to C, write 10 to A, then write <Abort T> to the log.

(c) Write 50 to E on the disk, write 30 to C, write 10 to A, and write <Abort T> to the log.

(d) Do nothing, since there are no un-committed transactions.