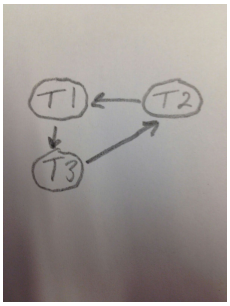
Ryan Ballenger
CSCIE66 Problem Set 3
Part 1

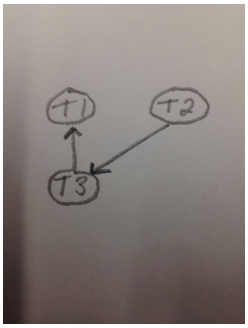
1.

a.) 3 -> 2
2 -> 1
1 -> 3



No, it is not conflict serializable. It cannot be serialized because there is a loop in the precedence graph.

b.) 3 -> 1
2 -> 3



Yes, it is conflict serializable. The only equivalent serial schedule is {T2, T3, T1}.

2.

a.)

T1	**T2**
...	
	r(A)
	r(C)
	u(c)
	w(A)
	u(A)
sl(A)	
r(A)	
xl(C)	
w(C)	
u(B)	
u(C)	
u(A)	
	commit
commit	

b.) No, T2 releases its lock on B too soon. It releases its lock on B, the rest of the T2 transaction finishes, and then it commits. For rigorous locking, the locks need to be held until the commit.

c.) Yes, T1 reads T2's write and T2 commits before T1. For recoverability, a transaction cannot commit if it has read an uncommitted write. This is satisfied by T2 committing first.

d.) The transaction starts as outlined in the handout, signified by "...".

T1	**T2**
...	
	r(A)
	r(C)
	u(c)
	w(A)
	u(A)
sl(A)	
r(A)	
xl(C)	
w(C)	
commit	

u(B)
u(C)
u(A)
commit

In this case, T1 commits as soon as possible after writing C. It has read the uncommitted version of A making this schedule not recoverable. After T1 finishes unlocking, T2 commit finally commits.

e.) No it is not cascadeless because exclusive locks aren't held through the commit step. T2 releases its exclusive lock on A aftering writing it, and T1 reads A before T2 commits.

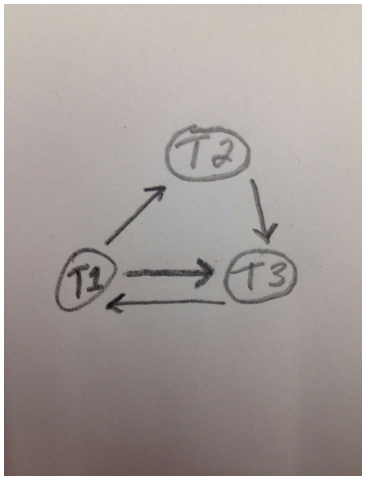
3.

a. Denied - T2 already has an exclusive lock on A and ul1(A) cannot proceed.
b. Granted - Multiple shared locks are allowed on B.
c. Granted - There are only shared locks on B meaning an update lock is allowed.
d. Denied - Shared locks are not allowed when an update lock (ul3(C)) exist on an item.
e. Granted - Update locks can upgrade if there are no other locks present.
f. Denied - T3 has an update lock (ul3(C)) and only one update lock is allowed per entry.

4.

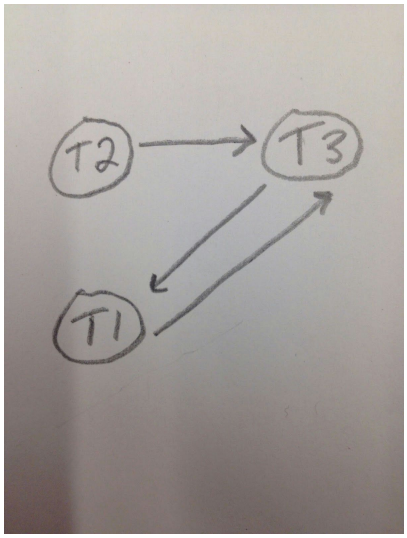
a.)

T1	**T2**	**T3**	Red: WAIT
	s2(C)		
	r2(C)		
s1(D)			
r1(D)			
xl1(C)			
		s3(C)	
		r3(C)	
	xl2(C)		
		xl3(D)	



b.)

T1	**T2**	**T3**	Red: WAIT
		xl3(A)	
		w3(A)	
	s2(A)		
s1(B)			
r1(B)			
		s3(B)	
		r3(B)	
		xl3(B)	
xl1(B)			



5.

a.)

S1 starts at 1, S2 starts at 2, ...

r2(A):

A has not been written so the read is allowed.
The transaction proceeds by doing the read.
A is given a RTS of 2.

w4(A):

T4 writes A. The RTS for A is at 2 and TS(T4) is at 4.
The transaction proceeds and writes A.
A is given a WTS at 4.

r3(A):

Denied because TS(T3) = 3 is less than the WTS(A) at 4.
T3 is aborted because the value A has been overwritten.
No changes are made to the state.

w1(A):

Denied because there is a RTS(A) at 2 and TS(T1) is 1.
T1 is aborted because it should have written A before the RTS(A) at 2.
No changes are made to the state.

w2(A):

Ignored because TS(T2) = 2 is less than a WTS(A) at 4.
The transaction continues and is not aborted.
No changes are made to the state.

r5(A):

Allowed because TS(T5)=5 is greater than the WTS(A) at 4.
The transaction T5 reads A and continues.
No changes are made to the state.

b.)

S1 starts at 1, S2 starts at 2, ...

r2(A):

A.commit == true and the read proceeds.
The transaction reads A and continues.
A is given a RTS at 2.

w4(A):

T4 writes A. The RTS for A is at 2 and TS(T4) is at 4.
The transaction proceeds and writes A.
A is given a WTS at 4 and the A.commit is set to FALSE

r3(A):

Denied because TS(T3) = 3 is less than the WTS(A) at 4.
T3 is aborts and restarts at time 3.
T3 tries to read A but **waits** because A.commit == FALSE

w1(A):

Denied because there is a RTS(A) at 2 and TS(T1) is 1.
T1 is aborted and restarted at a time of 7.
T1 writes A and does not have to wait. WTS(A) is now at 7 and A.commit == FALSE.

w2(A):

Ignored because the WTS(A) = 7 and this would make a WTS(A) = 2.
The transaction continues and is not aborted.
T2 must wait because A.commit == FALSE.

r5(A):

Not allowed because TS(T5)=5 is less than WTS(A) = 7.
The transaction T5 restarts at 8 and tries to read A.
T5 must **wait** because the commit.A == FALSE

C1:

T1 commits and A.commit == TRUE.
Now the waiting transactions can proceed.

w2(A):

Proceeds, as the smallest numbered transaction, and is ignored.
w2(A) is ignored because the WTS(T2) at 2 is less than the WTS(A) at 7.
Also RTS(A) = 2 does not affect this write at time of 2.

r3(A):

Proceeds but the transaction must be aborted and restarted.
The TS(T3)=6 is less than WTS(A) at 7. T3 aborts and restarts again at time 9.
T3 successfully reads A because A.commit == TRUE.

R5(A)

Proceeds and can read A. The WTS(A) at 7 is less than TS(T5) at 8.
Also the A.commit == TRUE meaning the read happens;

The remaining transactions commit(C2, C3, C4, and C5), and since A.commit == TRUE the state of the system does not change.

c.)

a.)
S1 starts at 1, S2 starts at 2, ...

r2(A):

Read of version A(0) occurs.
T2 proceeds,no changes are made to A, and A(0) gets an RTS=2.

w4(A):

Writes a new version of A(4).
T4 proceeds and the only change to A is the new version A(4).

r3(A):

Reads the version A(0) because its timestamp is at 3,
which is smaller than A(4). T3 continues and A(0) gets an RTS at 3.

w1(A):

Aborts and restarts because it should have written before A(0)'s RTS=2.
T1 restarts at time 6 and a new version A(6) is written.

w2(A):

T2 aborts and restarts at time 7 because it was too late for A(0)'s RTS at 3.
T2 starts with r2(A) which reads A(6). A(6) gets a RTS at 7.
w2(A) writes a new version of A named A(7).

r5(A):

The read occurs and reads version A(4).
A(4) gets a RTS at 5.

Finally all five transactions commit which does not change the state of the system, considering it has no commit bits.