CS1555/CS2055 Recitation 11 Solution

Objective: Practice Concurrency Control, Recovery

Part 1: Concurrency Control

1. Consider the following two transactions:

T1:	r1(A) ;
	r1(B);
	If A=0 then B:= B+1;
	w1(B);
		T2:	r2(C);
			r2(B);
			r2(A);
			if B>C then {A:= A+1; C:=C+1;}
			w2(C)
			w2(A)

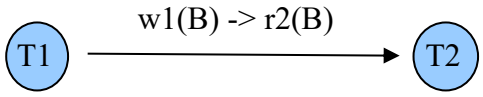
- For each of the following histories/schedules:
  a) Is it a valid history?
  b) Use *serializability graphs* to check whether it is serializable or not, and if it is, what is the equivalent serial history/schedule

H1: r1(A) r1(B) r2(C) w1(B) r2(B) r2(A)w2(C) w2(A)
H2: r1(A) r1(B) r2(C) r2(B) w1(B) r2(A)w2(C) w2(A)

Answer:

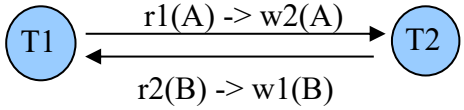
a) *Yes, they are valid histories because the order of statements in each transaction is preserved.*

b) Serializability graph for H1:



The graph is acyclic, so the history is serializable. The equivalent serial history is <T1, T2>

Serializability graph for H2:



The graph contains a cycle, so it is not serializable.

2. Consider the following two transactions:

T1:	r1(A) ;	T2:	r2(A);
	A:=A+100;		A:=A*2;
	w1(A);		w2(A);
	r1(B);		r2(B);
	B:=B+100;		B:=B*2;
	w1(B)		w2(B)

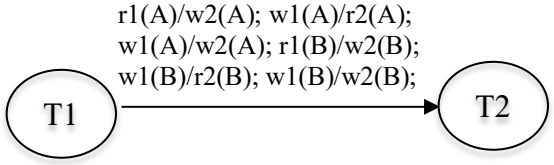
- For each of the following histories/schedules check:
  a) Is it a serializable history?
  b) What histories are conflict equivalent?

H1: r1(A), w1(A), r1(B), w1(B), r2(A), w2(A), r2(B), w2(B)
H2: r1(A), w1(A), r2(A), w2(A), r1(B), w1(B), r2(B), w2(B)
H3: r1(A), w1(A), r2(A), w2(A), r2(B), w2(B), r1(B), w1(B)
H4: r2(A), w2(A), r1(A), r2(B), w1(A), w2(B), r1(B), w1(B)
H5: r1(A), w1(A), r1(B), w1(B), r2(A), w2(A), w2(B), r2(B)

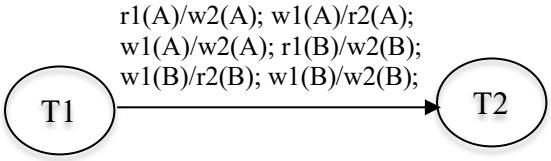
Answer:

a)

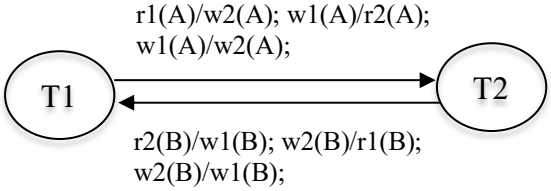
H1: is serializable and serial since all operations of T1 are executed before all operations of T2. The serializability graph is:



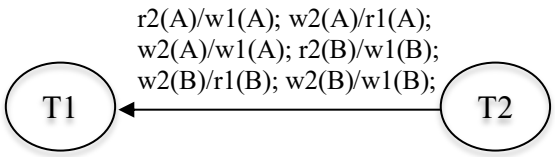
H2: is serializable. The serializability graph is:



H3: is not serializable. The serializability graph is:



H4: is serializable. The serializability graph is:



H5: is not a valid history.

b) H1 and H2 are conflict equivalent because the schedules have the same type of conflicts.
H3 and H4 are NOT conflict equivalent.
H5 is not a valid history, so it is not conflict equivalent to other histories.

3. Consider the following history, with lock and unlock statements added for each transaction:

a) Does the history follow 2PL protocol?

Yes. All locks are requested before the data is used and they are released only before the commit step.

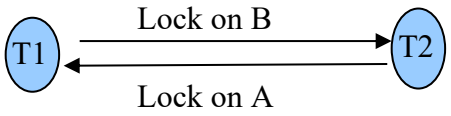
b) Did deadlock happen?

We use the wait-for graph to check this, as follows:

T1	T2	Wait-free
rl1(A) r1(A)		yes
rl1(B) r1(B)		yes
	rl2(C) r2(C)	yes
	rl2(B) r2(B)	yes, because is a read lock after another read lock from T1
wl1(B) w1(B)		no, T1 waits for T2 (add T1 → T2 to the WFG)
	rl2(A) r2(A)	yes

	wl2(C) w2(C)	yes
	wl2(A) w2(A)	no, T2 waits for T1 (add T2 → T1 to the WFG) (**deadlock happens!**)
unlock1(A), unlock1(B) commit	unlock2(C), unlock2(A) unlock2(B) commit	

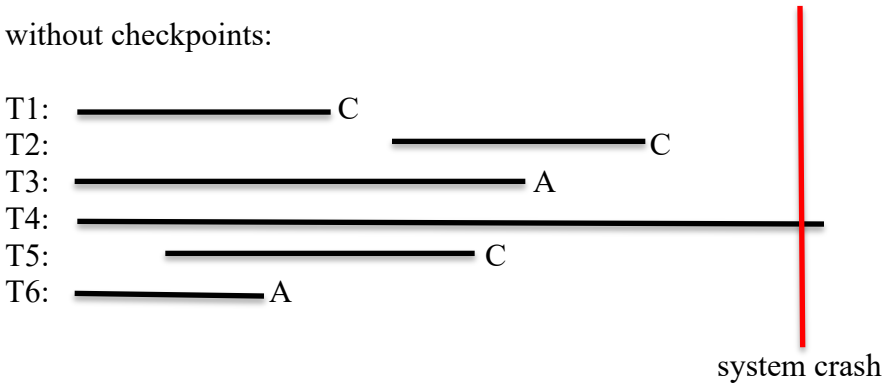
The corresponding wait-for graph contains a cycle, which means deadlock has happened.



Part 2. Recovery

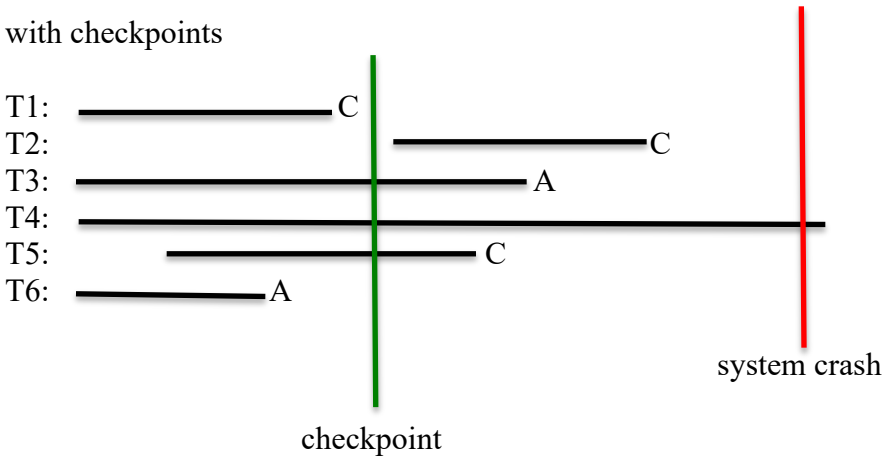
For the following transaction executions, state what the system should do when it restarts after a crash:

a. without checkpoints:



REDO (to preserve DURABILITY)	T1, T2, T5
UNDO (to preserve ATOMICITY)	T3, T4, T6

b. with checkpoints



REDO (to preserve DURABILITY)	T2, T5
UNDO (to preserve ATOMICITY)	T3, T4

T1 is not included anymore because it committed before the checkpoint;
T6 is not included anymore because it aborted before the checkpoint and the system already rolled back everything at the checkpoint