EXERCISE 3

Andrea Crotti (#299466), Can Liu (#286105), Sebastian Roidl (#281941)

11 Novembre 2009

Exercise 3

`DEADLINE:` *2009-11-11 Mer*

1) Recoverability

1. Prove the following

a) Every schedule belonging to avoid cascading abort (ACA) is also recoverable (RC)

Supposing that s fulfills the requirements for **ACA** and given that:

- a commit action must be always the last of the transaction
- reads x from t$_j$ in s $\rightarrow c_j < r_i(x)$

We directly have that also $c_j < c_i$, which is the requirement for RC. So $s \in ACA \rightarrow s \in RC$

b) Each schedule that is strict is also in ACA

Let s be a schedule element of ST. Therefore it holds by definition:
$w_j(x) <_s p_i(x), j \neq i \rightarrow a_j <_s p_i(x) c_j <_s p_i(x)$ where p$_i$(x) is either a read or a write operation.
Now, let p$_i$(x) be a read operation, we got:
$w_j(x) <_s r_i(x)$ and $j \neq i \rightarrow a_j <_s r_i(x) or c_j <_s r_i(x)$, which would be the definition of ACA.

Therefore s is also element of ACA. q.e.d.
In other words, ACA provides that a transaction doesn't read before the

other one is aborted or committed; ST provides that a transaction neither reads nor writes an element before the other one is aborted.
Therefore if a schedule is strict, it has to be ACA too, as strictness only add another condition to the ACA condition.

2. Test if in RC, ACA, ST

a) s1 = r3(y) r1(x) w1(x) r2(x) c1 w3(y) w2(x) c3 c2

- **RC** Yes, t2 reads from t1 and $c1 <_s c2$
- **ACA** No, t2 reads from t1 but $c1 >_s r2(x)$. In case of abort of the transaction t1 we could still have problems here. So it's also not **ST**

b) s2 = r2(x) r3(y) w2(x) c2 r3(x) w3(x) c3 r1(x) w1(x) c1

- **RC** Yes, t3 reads from t2 and $c2 <_s c3$, t1 reads from t3 and $c3 <_s c1$
- **ACA** Yes, t3 reads from t2 and $c2 <_s r3(x)$, t1 reads from t3 and $c3 <_s r1(x)$
- **ST** Yes, we can check easily that after every write of w_i(x) we have a c_i, there are no writes or reads of x between last write and commit.

c) s3 = r1(x) w1(x) r3(x) w3(x) c3 r2(x) w2(x) c2 a1

- **RC** No, because t3 reads from t1 but there is not a commit c1. So as logical consequence s3 it's not also **ACA** or **ST**

2) 2PL / S2PL / Deadlock handling

1. Prove that schedules produced by 2PL are conflict serializable.

The complete proof of this is given at page 70. Rephrasing it we get that:

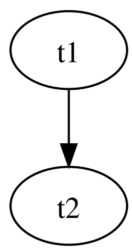
- take the rules for the application of locking
- see that if two transactions are in conflict we must negate one of the constraint of 2PL
- see that the graph generated by 2PL is acyclic

2. Given

- s1 = w1(x) r2(y) r1(x) c1 r2(x) w2(y) c2
- s2 = w1(x) r2(y) r1(x) c1 r2(x) w2(y) a2
- s3 = r1(x) r2(x) w3(x) w4(x) w1(x) c1 w2(x) c2 c3 c4

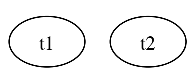
a) Compute the conflict graphs of the schedules above.

- s1

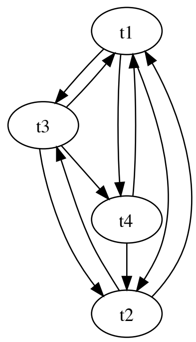
  Which is an acyclic graph, so $s1 \in CSR$

- s2

  Would be the same as s1, but there's an abort action which deletes the conflict between t1 and t2, so $s2 \in CSR$

- s3



This is a cyclic graph, so $s3 \notin CSR$

b) For each input schedule write down the corresponding schedule indicating the necessary locking (rl/wl) and unlocking (ru/wu) operations.

Write down the resulting output schedules for 2PL (transactions must unlock resources as soon as possible) and S2PL. In case of a deadlock the transaction with the lowest index is aborted. Once aborted, transactions are restarted as new at the end of the original schedule (abort-restart)

- s1
  - **2PL:**
    wl1(x) w1(x) r1(x) wu1(x) c1 rl2(y) wl2(y) r2(y) ru2(y) r2(x) wu2(y) c2
  - **S2PL:**
    We just need to move the unlockings to the end of the transaction
    wl1(x) w1(x) r1(x) wu1(x) c1 rl2(y) wl2(y) r2(y) r2(x) ru2(y) wu2(y) c2
- s2
  - **2PL:**
    wl1(x) w1(x) r1(x) wu1(x) c1

- **S2PL:**
  wl1(x) w1(x) r1(x) wu1(x) c1

- s3

  In this case we have to abort transactions and move them at then end of the schedule given the policy *abort-restart*

  - **2PL**
    rl2(x) r2(x) wl2(x) w2(x) wu2(x) ru2(x) c2 wl3(x) w3(x) wu3(x) c3 wl4(x) w4(x) wu4(x) c4 rl1(x) r1(x) wl1(x) w1(x) wu1(x) ru1(x)
  - **S2PL:**
    rl2(x) r2(x) wl2(x) w2(x) wu2(x) ru2(x) c2 wl3(x) w3(x) wu3(x) c3 wl4(x) w4(x) wu4(x) c4 rl1(x) r1(x) wl1(x) w1(x) wu1(x) ru1(x)