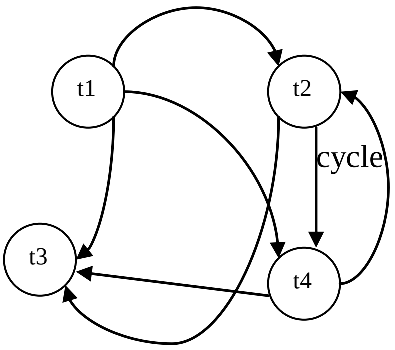
Exercise 1
Consider the following schedule. Show if it is conflict serializable (CSR) or view serializable (VSR). Explain why. If serializable, show equivalent serial schedules.

S: r1(x) r4(y) w1(z) r4(z) w2(y) r3(y) w1(x) w2(x) w3(z) w4(x) w3(x)

Solution

x: r1(x) w1(x) w2(x) w4(x) w3(x) Precedence constraints : t1<t2 and t1 < t4 and t1 < t3
y: r4(y) w2(y) r3(y) Precedence constraints : t4 < t2 and t2 < t3
z: w1(z) r4(z) w3(z) Precedence constraints : t1< t4 and t1 < t3 and t4 < t3



Precedence graph for schedule S

S is not CSR. Cycle in the precedence graph for S: t2<t4<t2

Note that : w1(x), w2(x) and w4(x) are blind write on data item X.
S is VSR. S is View equivalent to the serial achedule **S' = t1 t4 t2 t3**
Each read() instruction reads in both S and S' the value written by the same transaction.
Each data item has the same final write.

Exercise 2
Consider the following schedule, where each transaction is assumed to commit.

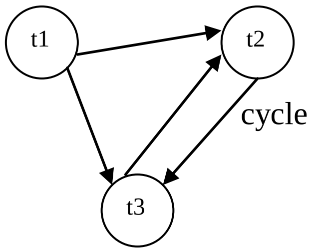
S: r1(y) r3(y) r1(x) w2(x) r2(y) w3(x) w2(y)

1) Show if it is conflict serializable (CSR) or view serializable (VSR). Explain why. If serializable, show equivalent serial schedules.
2) Apply the rigorous two-phase locking protocol to the schedule. Is the schedule accepted?
3) Apply the timestamp-ordering protocol to the schedule, assuming that aborted transactions are immediately restarted. Is the schedule accepted?

Solution

Point 1)

x: r1(x) w2(x) w3(x) Precedence constraints: t1 < t2, t1<t3, t2< t3
y: r1(y) r3(y) r2(y) w2(y) Precedence constraints : t1 < t2 and t3 < t2



Precedence graph for S

S is not CSR. Cycle in the precedence graph for S: t2<t3<t2

S is not VSR.
Consider the rule for final write() instructions : x : t2 < t3
Considere the rule for read() instructions y : t3 < t2

Point 2) rigorous 2PL

T1: lock_S(y) ok
T3: lock_S(y) ok
T1: lock_S(x) ok
T1: unlock()
T2: lock_X(x) ok
T2: lock_S(y) ok
T3: lock_X(x) T3 waits for T2
T2: lock_X(y) T2 waits for T3
Deadlock state.

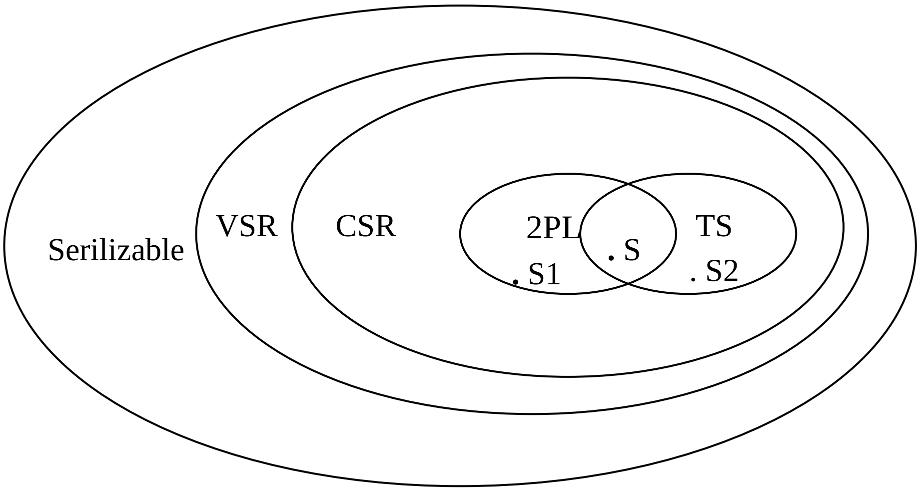
The schedule is not accepted because there exists a transaction that is made to wait.

Point 3) Timestamp-ordering protocol
x: RTS=0 WTS=0 y: RTS=0 WTS=0

r1(y) x: RTS=0 WTS=0 y: RTS=1 WTS=0
r3(y) x: RTS=0 WTS=0 y: RTS=3 WTS=0
r1(x) x: RTS=1 WTS=0 y: RTS=3 WTS=0
w2(x) x: RTS=1 WTS=2 y: RTS=3 WTS=0
r2(y) x: RTS=1 WTS=2 y: RTS=3 WTS=0
w3(x) x: RTS=1 WTS=3 y: RTS=3 WTS=0
w2(y) TS(T2) < RTS(y) T2 is aborted and restarted as T4
w4(x) x: RTS=1 WTS=4 y: RTS=3 WTS=0
r4(y) x: RTS=1 WTS=4 y: RTS=4 WTS=0
w4(y) x: RTS=1 WTS=4 y: RTS=4 WTS=4

The schedule is not accepted because a transaction is aborted.

Serializability, VSR, CSR, 2PL and TS concepts



S: r1(x) w1(x) r2(x) w2(x)

S1: r2(x) w2(x) r1(x) w1(x)

S2: <u>w2(x)</u> r3(x) **r1(y)** <u>w2(y)</u> w4(y) w5(y)

S2 is not 2PL
t2: lock_X(x) ok
t3: lock_S(x) t3 is made to wait
r3(x) could be executed if transaction t2 unlocked x before t3 reads x.
Since t2 also writes y, lock_X(y) must be executed before unlock(x) (2PL rule).
On the other hand, if t2 is holding lock_X(y), t1 can not read data item y before t2 writes y (t1 holds a shared lock on y when r1(y) is executed)