Assignment 3

Problem 1

A transaction is strict two-phase locked if it is two-phase locked and holds all of its locks until after it commits.

A history is strict two-phase locked if it can be augmented with lock and unlock operations such that every transaction in the history is strict two-phase locked.

A history is normally-strict two-phase locked if it can be augmented with lock and unlock operations such that every lock operation immediately precedes the read or write operation that it synchronizes and every transaction in the history is strict two-phase locked.
A history is two-phase locked if it can be augmented with lock and unlock operations such that every transaction in the history is two-phase locked.

Notice that in a two-phase locked or strict two-phase locked history, a transaction can execute a lock operation long before the read or write operation that it synchronizes. This is not the case for normally-strict two-phase locked history.

Answer these questions for the histories below.

a) Is it normally-strict two-phase locked? If so, add lock operations to demonstrate it.
b) If not, is it strict two-phase locked? If so, add lock operations to demonstrate it.
c) If not, is it two-phase locked? If so, add lock operations to demonstrate it.
d) If not, draw the serialization graph to find out if it is serializable or not.

Histories (differences from H_1 are in bold):

H_1: $r_1[y]\ r_1[x]\ r_2[x]\ w_1[y]\ c_1\ w_2[y]\ c_2$

H_2: $r_1[y]\ r_1[x]\ r_2[x]\ \mathbf{w_2[x]}\ w_1[y]\ c_1\ w_2[y]\ c_2$

H_3: $r_1[y]\ r_1[x]\ r_2[x]\ w_1[y]\ w_2[y]\ c_2\ \mathbf{c_1}$

H_4: $r_1[y]\ r_1[x]\ r_2[x]\ \mathbf{w_2[x]}\ \mathbf{r_3[y]}\ w_1[y]\ c_1\ \mathbf{w_3[z]}\ \mathbf{c_3}\ w_2[y]\ c_2$

Extra credit ☺: Is it possible for a history to be strict two-phase locked but not normally-strict two phase locked? If so, give an example. If not, give a proof.

Problem 2

Consider a data manager that uses two-phase locking. Suppose all transactions are single-threaded sequential programs, so no transaction can have more than one outstanding read or write request that is blocked. Could a transaction be involved in more than one deadlock? Explain your answer.

Problem 3

Suppose each transaction is a sequential program with no internal concurrency (i.e., no multi-threading). Consider the following three transactions:

$T_1 = r_1[x,y]\ w_1[x]$

$T_2 = r_2[x]\ w_2[z]$

$T_3 = r_3[z]\ r_3[y]\ w_3[y]$

a) Suppose the data manager is using two-phase locking. Suppose it receives the transactions' reads and writes in the following sequence (i.e., the operations may execute in a different order than this):
H_1: $r_1[x,y]\ r_2[x]\ w_1[x]\ \ w_2[z]\ r_3[z]\ r_3[y]\ w_3[y]$
The transactions also submitted commit operations but the commits are not shown in the above sequence. Add each transaction's commit operation to the earliest spot in the sequence where the data manager could have received that commit.

b) Answer the same question for this sequence (the difference is highlighted in bold):
H_2: $r_1[x,y]\ r_2[x]\ w_1[x]$ $\mathbf{r_3[z]\ w_2[z]}$ $r_3[y]\ w_3[y]$