Homework 5 solutions

1. Transactions go through several states in execution, until it commits or aborts eventually. What are all possible sequences of states? And elaborate on why these states may occur.

**Answer:** The possible sequences of states are:-

a. **active**→**partially committed**→**committed**. This is the normal sequence a successful transaction will follow. After executing all its statements it enters the partially committed state. After enough recovery information has been written to disk, the transaction finally enters the committed state.

b. **active**→**partially committed**→**aborted**. After executing the last statement of the transaction, it enters the partially committed state. But before enough recovery information is written to disk, a hardware failure may occur destroying the memory contents. In this case the changes which it made to the database are undone, and the transaction enters the aborted state.

c. **active**→**failed**→**aborted**. After the transaction starts, if it is discovered at some point that normal execution cannot continue (either due to internal program errors or external errors), it enters the failed state. It is then rolled back, after which it enters the aborted state.

2. Consider the following two transactions:

T1: read(B);
read(A);
if B = 0 then A := A + 1;
write(A).

T2: read(A);
read(B);
if A = 0 then B := B + 1;
write(B).

Let the consistency requirement be $A = 0 \lor B = 0$, with $A = B = 0$ the initial values.

a. Show that every serial execution involving these two transactions preserves the consistency of the database.

b. Show a concurrent execution of T1 and T2 that produces a nonserializable schedule.

c. Is there a concurrent execution of T1 and T2 that produces a serializable schedule?

**Answer:**

a. There are two possible executions: $T1 T2$ and $T2 T1$.

Case 2: $A B$

initially 0 0
after $T1 \ 1 \ 0$
after $T2 \ 1 \ 0$
Consistency met: $A = 0 \lor B = 0 \equiv F \lor T = T$
Case 1: $A \ B$
initially $0 \ 0$
after $T2 \ 0 \ 1$
after $T1 \ 0 \ 1$
Consistency met: $A = 0 \lor B = 0 \equiv T \lor F = T$

b. Any interleaving of $T1$ and $T2$ results in a non-serializable schedule.

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>read(A)</strong></td>
<td><strong>read(B)</strong></td>
</tr>
<tr>
<td><strong>read(B)</strong></td>
<td><strong>read(A)</strong></td>
</tr>
</tbody>
</table>
| if $A = 0$ \ then $B = B + 1$ | if $B = 0$ \ then $A = A + 1$
| **write(B)** | **write(A)** |

c. There is no parallel execution resulting in a serializable schedule. From part a. we know that a serializable schedule results in $A = 0 \lor B = 0$. Suppose we start with $T2$ **read(A)**. Then when the schedule ends, no matter when we run the steps of $T1$, $B = 1$. Now suppose we start executing $T1$ prior to completion of $T2$. Then $T1$ **read(B)** will give $B$ a value of $0$. So when $T1$ completes, $A = 1$. Thus $B = 1 \land A = 1 \rightarrow (A = 0 \lor B = 0)$. Similarly for starting with $T1$ **read(B)**.

3. Is the schedule for the following precedence graph conflict serializable? Why?

![Precedence Graph]

**Answer:** There is a serializable schedule corresponding to the precedence
graph below, since the graph is acyclic. A possible schedule is obtained by doing a topological sort, that is, $T_1, T_2, T_3, T_5, T_4$. 