CSE 132A

A few practice problems on concurrency control

1. Consider the following schedule:

$$w_3(E)r_1(D)w_2(C)w_3(A)r_1(E)w_1(B)r_1(B)w_2(E)r_4(A)w_4(C)$$

(i) Draw the precedence graph for this schedule. (ii) Is the schedule conflict serializable? If yes, give an equivalent serial schedule.

Solution The precedence graph has the following edges:

$$< T1, T2 >< T3, T1 >, < T3, T2 >, < T2, T4 >< T3, T4 > .$$

The graph is acyclic so the schedule is conflict serializable. An equivalent serial schedule is $T_3; T_1; T_2; T_4$.

2. Give an example of two transactions T1 and T2 that lock the same data entities, such that T1 satisfies two-phase locking, T2 violates two-phase locking, and there is a schedule for T1 and T2 that is not conflict serializable.

Solution An example is the following:

$$T1 = l(A); w(A); l(B); w(B); u(A); u(B)$$

$$T2 : l(A); w(A); u(A); l(B); w(B); u(B)$$

A schedule for $T1$ and $T2$ that is not conflict serializable is:

$$\begin{array}{ll} T2: & l(A) \\ T2: & w(A) \\ T2: & u(A) \\ T1: & l(A) \\ T1: & w(A) \\ T1: & l(B) \\ T1: & w(B) \\ T1: & u(A) \\ T1: & u(B) \\ T2: & l(B) \\ T2: & w(B) \\ T2: & u(B) \end{array}$$

The precedence graph for this schedule is $< T2, T1 >, < T1, T2 >$, which is cyclic.

3. Consider a concurrency control protocol requiring that every transaction request data entities in a fixed linear order (i.e., the data entities are $A_1 ... A_n$ and no transaction can request A_i after A_j if $i < j$).

(i) Does the above protocol ensure serializability of all resulting schedules ?

(ii) Does the protocol prevent deadlocks ? (Hint: consider a "waits-for" graph whose nodes are the transactions, and at a given time there is an edge from transaction t to transaction t' iff at that time t is waiting for a data entity held by t'. A deadlock occurs iff at some time the "waits-for" graph has a cycle.)

Solution

(i) The protocol does not ensure serializability. For example, take the transactions T1, T2 in problem 2, where A is replaced by $A1$ and B by $A2$. Both transactions satisfy the protocol but the schedule shown in problem 2 is not serializable.

(ii) The protocol does prevent deadlock. Consider a "waits-for" graph for n transactions $T_1, \ldots, T_n$. The nodes are $T_1, \ldots, T_n$ and there is an edge from T to T' if T waits for T', i.e. T' holds the lock on some data entity and T has requested the lock on the same and is waiting for T' to release it. Deadlock occurs if there is a cycle in this graph. To see that this cannot happen, suppose there is a cycle

$$T_1 \rightarrow ... \rightarrow T_n \rightarrow T_1$$

For $j < n$, let A_{i_j} be the data entity such that T_{j+1} holds the lock on A_{i_j} and T_j is waiting for the lock on the same. For $j = n$ let A_{i_n} be the entity on which T_1 holds the lock and for which T_n is waiting. Notice that for $j < n$, T_{j+1} holds the lock on A_{i_j} and is waiting for the lock on $A_{i_{j+1}}$ so by the protocol, $i_j < i_{j+1}$. Similarly, T_1 holds the lock on A_{i_n} and is waiting for the lock on A_{i_1}, so $i_n < i_1$. In summary we have $i_1 < i_2 < \ldots i_n < i_1$ which is a contradiction. This shows that there cannot be a cycle in the "waits-for" graph so the protocol prevents deadlock.