1 Properties of Schedules

1. Consider schedule $S_1$:

   $S_1 = r_3(A), r_2(B), w_2(B), r_1(B), w_3(C), c_3, r_2(D), c_2, r_1(D), c_1$

   (a) Is $S_1$ cascadeless? Explain briefly.

   No, $S_1$ is not cascadeless. The only write operation of one transaction that is followed by a read operation of another transaction on the same data item is $w_2(B)$ followed by $r_1(B)$. Since $T_2$ does not commit before $T_1$ executes $r_1(B)$, $S_1$ is not cascadeless.

   (b) Is $S_1$ recoverable? Explain briefly.

   Yes, $S_1$ is recoverable. The only write operation of one transaction that is followed by a read operation of another transaction on the same data item is $w_2(B)$ followed by $r_1(B)$. Since $T_2$ commits before $T_1$ commits, $S_1$ is recoverable.

   (c) Draw the precedence graph of schedule $S_1$.

   

   (d) Is $S_1$ conflict-serializable? Explain briefly.

   Schedule $S_1$ is conflict-serializable, because its precedence graph is acyclic.

   (e) If $S_1$ is conflict-serializable, write down all equivalent serial schedules.

   • $T_3, T_2, T_1$
   • $T_2, T_3, T_1$
   • $T_2, T_1, T_3$

   (f) If $S_1$ is not conflict-serializable, make the schedule conflict-serializable. You may perform exactly one exchange, i.e. you can swap exactly two operations. The order of operations within a transaction must be preserved. Show that your modified schedule is conflict-serializable.

   $S_1$ is already conflict-serializable.

2. Consider schedule $S_2$:

   $S_2 = w_3(D), r_1(A), w_2(D), w_2(A), w_3(A), c_3, w_1(C), r_2(D), c_2, c_1$

1
(a) Is $S_2$ cascadeless? Explain briefly.

Yes, $S_2$ is cascadeless. The only write operation of one transaction that is followed by a read operation of another transaction on the same data item is $w_3(D)$ followed by $r_2(D)$. Since $T_3$ commits before $T_2$ executes $r_2(D)$, $S_2$ is cascadeless.

(b) Is $S_2$ recoverable? Explain briefly.

Yes, $S_2$ is recoverable because it is also cascadeless.

(c) Draw the precedence graph of schedule $S_2$.

(d) Is $S_2$ conflict-serializable? Explain briefly.

$S_2$ is not conflict-serializable, because its precedence graph is cyclic.

(e) If $S_2$ is conflict-serializable, write down all equivalent serial schedules.

$S_2$ is not conflict-serializable

(f) If $S_2$ is not conflict-serializable, make the schedule conflict-serializable.

You may perform exactly one exchange, i.e. you can swap exactly two operations. The order of operations within a transaction must be preserved. Show that your modified schedule is conflict-serializable.

By having swapped the two operations highlighted in red we obtained a new schedule $S'_2$ that is conflict-serializable.

$S'_2 = w_3(D), r_1(A), w_2(A), w_3(A), w_2(A), c_3, w_1(C), r_2(D), c_2, c_1$

As the precedence graph of $S'_2$ is acyclic (see below), schedule $S'_2$ is conflict-serializable.

2 Two-Phase Locking

1. Consider schedule $S$:

$S = r_1(A), r_2(D), r_1(C), w_1(C), w_2(A), c_2, r_1(B), c_1$
(a) Extend schedule $S$ with lock and unlock statements such that it adheres to the S2PL protocol and is deadlock-free. If this is impossible, explain why.

$$S' = sl_1(A), r_1(A), sl_2(D), r_2(D), xl_1(C), r_1(C), sl_1(B),
        u_1(A), xl_2(A), w_2(A), u_2(A), u_2(D), c_2, r_1(B), u_1(B), u_1(C), c_1$$

(b) Extend schedule $S$ with lock and unlock statements such that it adheres to the SS2PL protocol and is deadlock-free. If this is impossible, explain why.

It is impossible to extend $S$ with lock and unlock statements such that it adheres to the SS2PL protocol. The reason is that in SS2PL $T_1$ must hold its shared lock $sl_1(A)$ until it commits, but according to $S$ transaction $T_2$ needs to acquire its exclusive lock $xl_2(A)$ before $T_1$ commits. Since this two locks are incompatible, schedule $S$ cannot be produced by a SS2PL scheduler.

2. Prove or disprove the following statement: The Two-Phase Locking (2PL) protocol permits all possible conflict-serializable schedules.

We disprove this statement with a counterexample, showing a schedule that is conflict-serializable, but could not have been produced by a 2PL scheduler. Schedule $S = r_1(A), w_1(A), r_1(A)$ is conflict-serializable as it is conflict-equivalent to the serial schedule $r_1(A), w_1(A), r_1(A), r_2(A)$. However, this schedule cannot be produced by a 2PL scheduler as shown next. We distinguish two cases:

- $T_1$ acquires a lock first. $T_1$ write-locks data item $A$ before executing $r_1(A), w_1(A)$ and can release this lock only after its read $r_1(A)$. In between this two operations, $T_2$ cannot acquire any lock on $A$ and therefore cannot perform $r_2(A)$.
- $T_2$ acquires a lock first. $T_2$ read-locks data item $A$, but then $T_1$ cannot acquire the write-lock on $A$ required to execute the first two operations in schedule $S$: $r_1(A), w_1(A)$. 
