Checking for Conflict Serializibility

Dr. Seema Gupta Bhol

Checking Whether a Schedule is Conflict Serializable Or Not

- A non-serial schedule gets checked for conflict serializability using the following steps:
- Figure out all the conflicting operations and enlist them.
- Create a precedence graph. For every transaction in the schedule, draw a node in the precedence graph.
- If Xi(A) and Yj(A) represent a conflict pair, then draw an edge from Ti to Tj for each conflict pair. The precedence graph ensures that Ti gets executed before Tj.
- The next step involves checking for any cycle formed in the graph.
- **A schedule is a conflict serializable if there is no cycle present**.

Precedence Graph or Serialization Graph

Step 1: Draw a node for each transaction in the schedule.

Step 2: For each pair of conflicting operations (i.e., operations on the same data item by different transactions), draw an edge from the transaction that performed the first operation to the transaction that performed the second operation. The edge represents a dependency between the two transactions.

Step 3: If there are multiple conflicting operations between two transactions, draw multiple edges between the corresponding nodes.

Step 4: If there are no conflicting operations between two transactions, do not draw an edge between them.

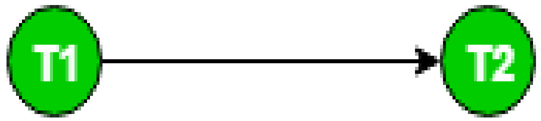
Step 5: Once all the edges have been added to the graph, check if the graph contains any cycles. If the graph contains cycles, then the schedule is not conflict serializable. Otherwise, the schedule is conflict serializable.

Consider the schedule S:r1(x) r1(y) w2(x) w1(x) r2(y)

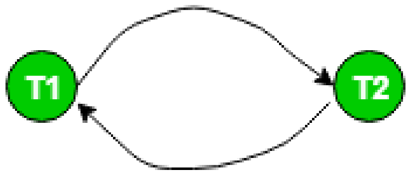
Creating Precedence Graph

Step 1: Make two nodes corresponding to Transaction T_1 and T_2.

Step 2: For the conflicting pair r1(x) w2(x), where r1(x) happens before w2(x), draw an edge from T_1 to T_2.

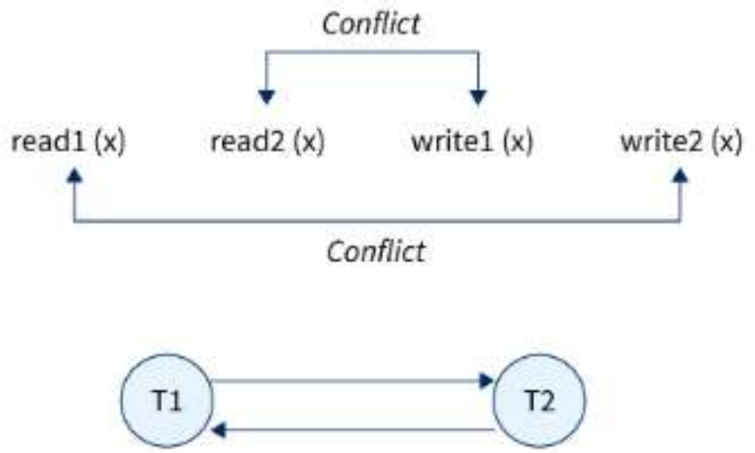


Step 3: For the conflicting pair w2(x) w1(x), where w2(x) happens before w1(x), draw an edge from T_2 to T_1.



Since the graph is cyclic, we can conclude that it is **not conflict serializable** to any schedule serial schedule.

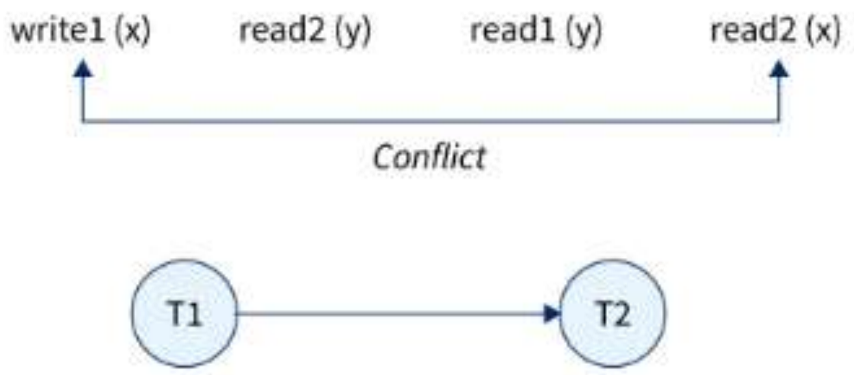
Example1



T1	T2
Read (x)	
	Read (x)
Write(x)	
	Write(x)

The above example has a pair of conflicting operations. The transaction drawn on a precedence graph makes a cycle. Thus the schedule is not conflict serializable.

Example 2



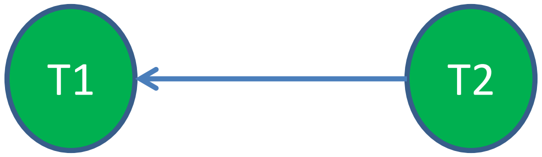
T1	T2
write(x)	
	Read (y)
Read(y)	
	Read(x)

The above example has only one conflict pair. Drawing the transaction on a graph does not produce a cycle. Thus it is conflict serializable.

Example 3

read2 (x) read2 (y) write2 (x) read2 (x) read1 (y)

T1	T2
	Read(x)
	Read (y)
	write(x)
	Read(x)
Read(y)	

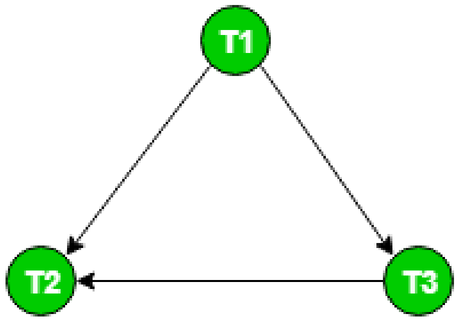


The above example has a pair of conflicting operations, the transaction drawn on a precedence graph does not make a cycle. Thus the schedule is conflict serializable.

Example 4

r1(x) r3(y) w1(x) w2(y) r3(x) w2(x)

T1	**T2**	**T3**
Read(x)		
		Read (y)
write(x)		
	write(y)	
		Read(x)
	write(x)	


$$\begin{tikzcd} & T1 \arrow[dl] \arrow[dr] & \\ T2 & & T3 \arrow[ll] \end{tikzcd}$$

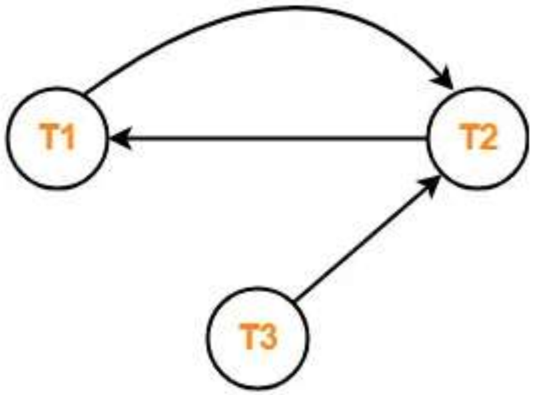
Since the graph is acyclic, the schedule is conflict serializable.

Example 5

r1(x) r2(x) r1(y) r2(y) r3(y) w1(x)w2(y)

T1	**T2**	**T3**
Read(x)		
	Read (x)	
Read(y)		
	Read(y)	
		Read(y)
write(x)		
	Write(y)	

- $R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
- $R_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
- $R_3(B)$, $W_2(B)$ $(T_3 \rightarrow T_2)$

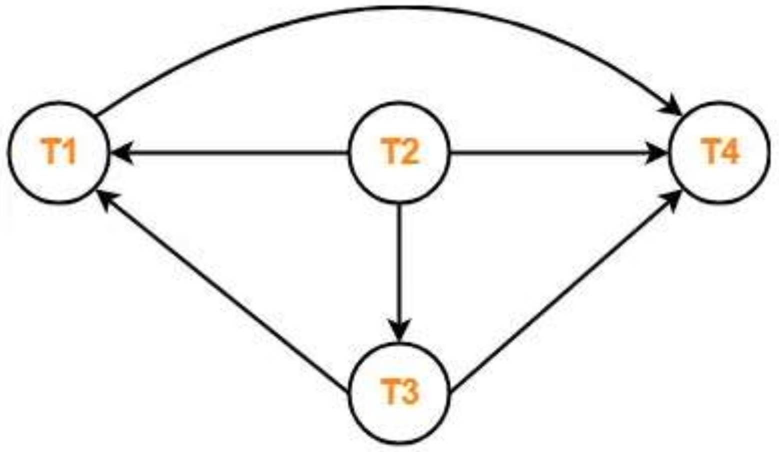


Clearly, there exists a cycle in the precedence graph.
Therefore, the given schedule S is not conflict serializable.

Example 6

T1	T2	T3	T4
	Read(x)		
		write(x)	
		commit	
write(x)			
commit	Write(y)		
	Read(z)		
	commit		
			Read(x)
			Read(y)
			commit

$R_2(X)$, $W_3(X)$	$(T_2 \rightarrow T_3)$
$R_2(X)$, $W_1(X)$	$(T_2 \rightarrow T_1)$
$W_3(X)$, $W_1(X)$	$(T_3 \rightarrow T_1)$
$W_3(X)$, $R_4(X)$	$(T_3 \rightarrow T_4)$
$W_1(X)$, $R_4(X)$	$(T_1 \rightarrow T_4)$
$W_2(Y)$, $R_4(Y)$	$(T_2 \rightarrow T_4)$



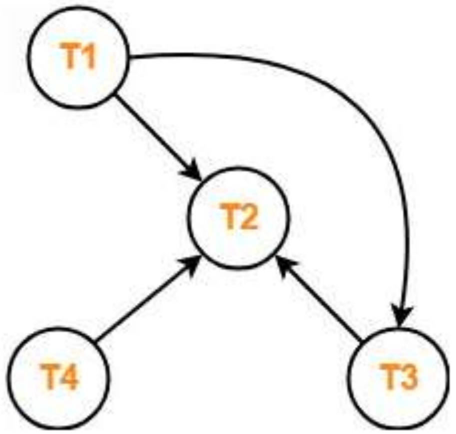
Clearly, there exists no cycle in the precedence graph.
Therefore, the given schedule S is conflict serializable.

Example 7

T1	T2	T3	T4
			Read(x)
	Read(x)		
		Read(x)	
write(y)			
	Write(x)		
		Read(y)	
	Write(y)		

List all the conflicting operations and determine the dependency between the transactions-

$R_4(A)$, $W_2(A)$	$(T_4 \rightarrow T_2)$
$R_3(A)$, $W_2(A)$	$(T_3 \rightarrow T_2)$
$W_1(B)$, $R_3(B)$	$(T_1 \rightarrow T_3)$
$W_1(B)$, $W_2(B)$	$(T_1 \rightarrow T_2)$
$R_3(B)$, $W_2(B)$	$(T_3 \rightarrow T_2)$



Clearly, there exists no cycle in the precedence graph.
Therefore, the given schedule S is conflict serializable.

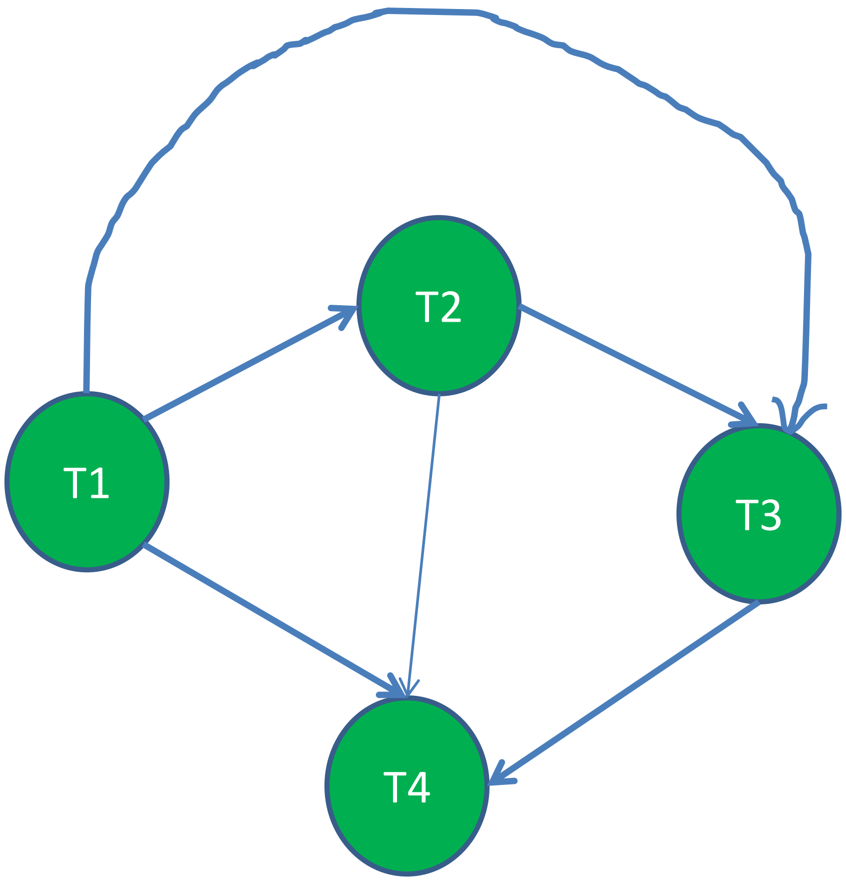
Ex-1

T1	T2	T3	T4
Read(x)			
	Read(x)		
		Read(x)	
			Read(x)
write(y)			
	Write(y)		
		Write(y)	
			Write(y)

Answer the following questions:

(a) What is the precedence graph for the schedule?

(b) Is the schedule serialisable?



Clearly, there exists no cycle in the precedence graph.
Therefore, the given schedule S is conflict serializable.

Ex-2

Consider the following two schedules:

S1: w3(A); r1(A); w1(B); r2(B); w2(C); r3(C)

S2: r1(A); r2(A); r3(B); w1(A); r2(C); r2(B); w2(B); w1(C)

Answer the following questions:

(a) What is the precedence graph for the schedule S1,S2?

(b) Is the schedule serialisable?

Solution:

(a) S1 gives a cyclic graph
$$\begin{tikzcd} & T_1 \arrow[dl] & \\ T_2 \arrow[rr] & & T_3 \arrow[ul] \end{tikzcd}$$
. This means that the schedule is not serialisable. Is it equivalent to one of these plans $T_1; T_2; T_3 \mid T_1; T_3; T2 \mid T_2; T_3; T1$?

If T_1 calculates a new value in B depending on what it reads from A, then it is of importance if T_3 has written a new value in A before or after T_1 reads it. The same applies for T_2 and T_3. Thus. This plan is not equivalent with any serial plan.

(b) S2 This graph is without cycles
$$\begin{tikzcd} & T_1 & \\ T_2 \arrow[ur, bend left] \arrow[ur] & & T_3 \arrow[ll] \end{tikzcd}$$
. This means the schedule is conflict serialisable. It is equivalent with $T_3; T_2; T_1$. Since T_3 only reads values, the serial schedule $T_2; T_1; T_3$ will have the same effect on the database. It is, however, not conflict equivalent with the plan above. The application that initiated T_3 will get a different return value than the original plan gave.

Consider the following two transactions:

```
T1                                  T2
 read(A)                             read(B)
 read(B)                            read(A)
 if A = 0 then B := B + 1           if B = 0 then A := A + 1;
 write(B)                           write(A)
```

Add lock and unlock instructions to transactions T1 and T2, so that they observe the two-phase locking protocol. Can the execution of these transactions result in a deadlock?

Lock and unlock instructions:

```
T1                          T2
 lock-S(A)                   lock-S(B)
 read(A)                     read(B)
lock-X(B)                    lock-X(A)
read(B)                     read(A)
if A = 0 then B := B + 1    if B = 0 then A := A + 1
write(B)                    write(A)
 unlock(A)                   unlock(B)
 unlock(B)                   unlock(A)
```

Execution of these transactions can result in deadlock. For example, consider the following partial schedule:

```
T1              T2
 lock-S(A)       ---
 ---            lock-S(B)
-----            read(B)
read(A)          ---
lock-X(B)       ---
read(B)         ---
                lock-X(A)
```

The transactions are now deadlocked.