COMP207 Lab Exercises
Tutorial 1 - Week 3

Tutorial 1a – Transaction processing

Tutorial 1a explains transaction processing concepts, including database and file organisation, transaction control examples and buffer management. Make sure you understand these concepts.

Tutorial 1b – Problems that can occur in concurrent schedules

Tutorial 1b contains examples of problems that can occur when transactions can be interleaved arbitrarily. Make sure you understand these examples.

Tutorial 1c – Precedence graphs and conflict-serialisability

As shown in Tutorial 1c, conflict-serialisability can be tested using precedence graphs. Below you can find an additional example of how the test works, followed by two exercises to test your skills in applying the test.

Example 1. We apply the test to the following schedule:

$$S_1 : r_1(X); r_2(Z); r_1(Z); r_3(X); r_3(Y); w_1(X); w_3(Y); r_2(Y); w_2(Z); w_2(Y)$$

We first construct the precedence graph for S_1. We start by determining the conflicts in S_1 (recall the definition of a conflict from Tutorial 1c):

$$r_3(X)\text{-}w_1(X),\ w_3(Y)\text{-}r_2(Y),\ w_3(Y)\text{-}w_2(Y),\ r_1(Z)\text{-}w_2(Z),\ r_3(Y)\text{-}w_2(Y)$$

Here, the pair $r_3(X)$-$w_1(X)$ represents a conflict between the two operations $r_3(X)$ and $w_1(X)$ in S_1, where $r_3(X)$ comes first in S_1, and similarly for the other p airs. The conflicts lead to the following precedence graph of S_1:

$$\begin{tikzcd}
\boxed{T_1} \arrow[r] & \boxed{T_2} & \boxed{T_3} \arrow[l] \arrow[ll, bend left]
\end{tikzcd}$$

This graph is *acyclic*: in the graph there is no path that has the same start and end node. Consequently, S_1 is conflict-serialisable.

A conflict-equivalent serial schedule can now be obtained by ordering the nodes in the precedence graph such that each edge goes from left to right (this is always possible if the graph is acyclic, and is called a *topological ordering* or *topological sort*):

$$\begin{tikzcd} \boxed{T_3} \arrow[r] \arrow[rr, bend left] & \boxed{T_1} \arrow[r] & \boxed{T_2} \end{tikzcd}$$

The conflict-equivalent serial schedule is then the schedule that executes all the transactions in this order. In our case, the conflict-equivalent serial schedule for S_1 is the schedule that first executes T_3, then T_1, and finally T_2.

Example 2. We next apply the test to the following schedule:

$$S_2 : r_1(X); r_2(Z); r_3(X); r_1(Z); r_2(Y); r_3(Y); w_1(X); w_2(Z); w_3(Y); w_2(Y)$$

Again, we first construct the precedence graph for S_2. The conflicts in S_2 are:

$$r_3(X)\text{-}w_1(X),\ r_1(Z)\text{-}w_2(Z),\ r_2(Y)\text{-}w_3(Y),\ r_3(Y)\text{-}w_2(Y),\ w_3(Y)\text{-}w_2(Y)$$

These conflicts lead to the following precedence graph of S_2:

$$\begin{tikzcd} \boxed{T_1} \arrow[r] & \boxed{T_2} \arrow[r, bend left] & \boxed{T_3} \arrow[l, bend left] \arrow[ll, bend left] \end{tikzcd}$$

This graph is *not acyclic*: it contains directed cycle $T_2 \rightarrow T_3 \rightarrow T_2$. It follows that S_2 is *not* conflict-serialisable.

Exercise 1:

i) Use precedence graphs to test whether the following schedules are conflict-serialisable.

ii) If the schedules are conflict-serialisable, construct the equivalent serial schedules.

- S_3 : $r_3(X)$; $r_2(X)$; $w_3(X)$; $r_1(X)$; $w_1(X)$
- S_4 : $r_3(X)$; $r_2(X)$; $r_1(X)$; $w_3(X)$; $w_1(X)$
- S_5 : $r_1(X)$; $r_2(X)$; $r_3(Y)$; $w_1(X)$; $r_2(Z)$; $r_2(Y)$; $w_2(Y)$; $w_1(Z)$