THE UNIVERSITY OF MELBOURNE
SCHOOL OF COMPUTING AND INFORMATION SYSTEMS
COMP90020 DISTRIBUTED ALGORITHMS

Tutorial Week 9: Transactions and Concurrency Control

Solutions

42.

(a) $x_T = read(j, T); y_T = read(i, T); x_U = read(k, U); write(j, 44, T); write(i, 33, T); commit(T);$
cont- $write(i, 55, U); y_U = read(j, U); write(k, 66, U); commit(U);$

(b) $x_T = read(j, T); y_T = read(i, T); x_U = read(k, U); write(j, 44, T); write(i, 33, T);$ cont- $write(i, 55, U); commit(T); y_U = read(j, U); write(k, 66, U); commit(U);$

(c) $x_T = read(j, T); x_U = read(k, U); write(i, 55, U); y_T = read(i, T); y_U = read(j, U);$
cont- $write(k, 66, U); commit(U); write(j, 44, T); write(i, 33, T); commit(T);$

43.

(a) There are 24 possilbe interleavings of the nested transactions T_1, T_2, U_1, U_2 as every execution is serial (each transaction has a single op). There are 6 possible serially equivalent interleavings for the non-nested case.

(b) Nested transactions allow more serially equivalent interleavings because: i) there is a larger number of serial executions ii) there is more potential for overlap between transactions iii) the scope of the effect of conflicting can be narrowed

44.

(a) $a.withdraw(4, T_1); commit(T_1); c.withdraw(3, U_1); commit(U_1);$
$b.deposit(4, T_2); commit(T_2); b.deposit(3, U_2); commit(U_2);$

(b) $a.withdraw(4, T_1); commit(T_1); c.withdraw(3, U_1); commit(U_1);$
$b.deposit(4, T_2); b.deposit(3, U_2); commit(T_2); commit(U_2);$

(c) Does not reduce concurrency between siblings T1,T2 or U1,U2 however it does affect families of transactions that have contending writes on the same object.

45. (a) and (b) are serially equivalent. (c) and (d) are serially equivalent. Only (b) and (c) could occur under 2PL as (a) and (d) violate the release phase.

46. Only the last transaction will commit. See excalidraw for more.