# Transaktionsorientierte Informationssysteme SS 2000

## Übung 3 Beispiellösung

### Aufgabe 1:

Show that the O2PL protocol with lock table LT8 is not deadlock-free.

Consider the following schedule:

$$\begin{array}{lllll} r_1(x) & & & w_1(y) & \text{ul} \\ & r_2(y) & w_2(x) & & \text{ul} \end{array}$$

This is a deadlock: both transactions are on hold and wait for the other to start releasing locks.

### Aufgabe 2:

For the following three histories, check whether they are monoversion histories, or members of MVSR or MCSR:

$m_1 = w_0(x_0)w_0(y_0)w_0(z_0)c_0r_3(x_0)w_3(x_3)c_3w_1(x_1)c_1r_2(x_1)w_2(y_2)w_2(z_2)c_2$
$m_2 = w_0(x_0)w_0(y_0)c_0w_1(x_1)c_1r_3(x_1)w_3(x_3)r_2(x_1)c_3w_2(y_2)c_2$
$m_3 = w_0(x_0)w_0(y_0)c_0w_1(x_1)c_1r_2(x_1)w_2(y_2)c_2r_3(y_0)w_3(x_3)c_3$

a) Schedule $m_1$ is already serial and has a standard version function, so it is trivially in MCSR, and therefore in MVSR.

b) In schedule $m_2$, we can push operations $r_3(x_1)w_3(x_3)$ behind $w_2(x_2)$, because there is no rw conflict on this way. We get the equivalent serial monoversion schedule $t_0t_1t_2t_3$, so $m_2$ is in MCSR, and therefore also in MVSR.

c) This schedule is in MCSR, because its associated mv-conflict-graph is acyclic, consisting only of an edge from $t_2$ to $t_3$. We get an equivalent serial schedule by commuting $t_0$'s operations to the right, just before $t_3$, resulting in the schedule $t_1t_2t_0t_3$. While it appears to be counter-intuitive to push the initializing transaction to the middle of the schedule, this is a perfectly legal operation by the definition of MCSR.

   Based on this result, it is clear that this schedule is in MVSR, too. For an alternative proof, we choose the version orders $x_0 << x_3 << x_2$ and $y_0 << y_2$ and build the corresponding MVSG shown in Figure 1.

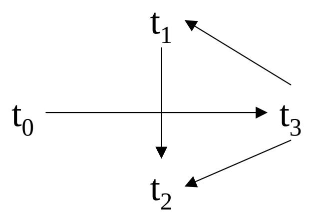


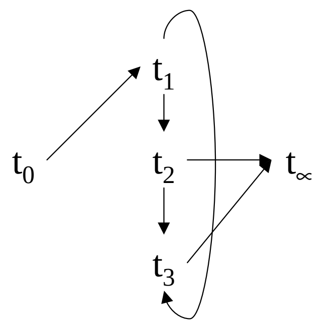

Abbildung 1: MVSG for schedule $m_3$

## Aufgabe 3:

For the multiversion schedule

$m = w_0(x_0)w_0(y_0)c_0r_1(x_0)w_1(x_1)r_2(x_1)w_2(y_2)w_1(y_1)w_3(x_3)$ test whether there exists a version order $<<$ such that $MVSG(m, <<)$ is acyclic.

If we chose $x_0 << x_1 << x_3$ and $y_0 << y_1 << y_2$, we get the following acyclic graph:



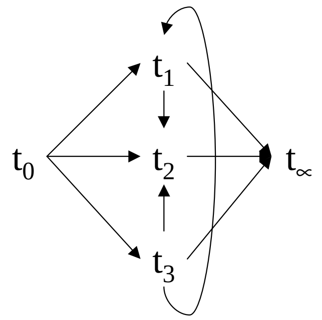

## Aufgabe 4:

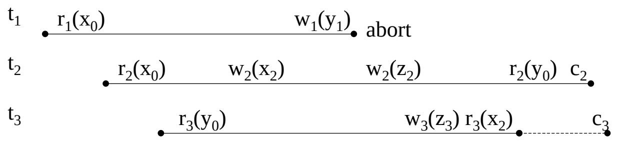
Consider the following schedule, given in the multiversion form:

$r_1(x)r_2(x)r_3(y)w_2(x)w_1(y)c_1w_2(z)w_3(z)r_3(x)c_3r_2(y)c_2$

Show that this schedule is multiversion serializable, i.e., could be allowed by a multiversion concurrency control. Give a feasible version function, and also a feasible version order. How do the resulting executions (i.e., output schedules) under the MVTO and 2V2PL protocols look like?

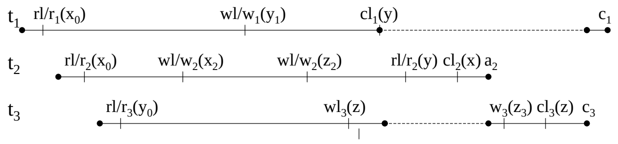
To show that this schedule is in MVSR, we choose the version order $x_0 << x_2$, $y_0 << y_1$, and $z_0 < z_3 < z_2$, and get the following acyclic MVSG:





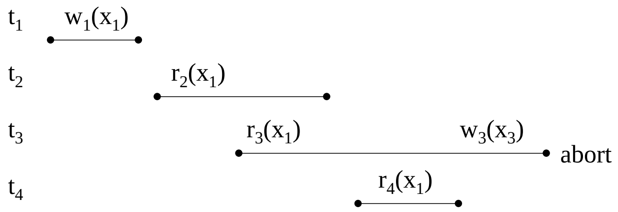

This is the resulting output under the MVTO protocol:

This is the resulting output under the 2V2PL protocol:




## Aufgabe 5:

Consider the input schedule $w_1(x)c_1r_2(x)r_3(x)c_2r_4(x)c_4c_3$ Give the resulting output schedule under the MVTO protocol.




## Aufgabe 6:

Reconsider the ROMV protocol that has been specifically geared for read-only transactions. What happens if the protocol is relaxed in such a way that update transactions use the timestamp-based version selection for their read steps? That is, update transactions would still use conventional exclusive locks for writes, but would exploit versioning for reads by selecting the most recent version that was committed at the time for the update transaction's begin. Is this protocol still correct in that it guarantees MVSR schedules?

This protocol is not correct anymore, as it allows the following schedule:

$$\begin{array}{llll} r_1(x)r_1(y) & & w_1(x)c_1 & \\ & r_2(x)r_2(y) & & w_2(y)c_2 \end{array}$$

This schedule is not in MVSR. It may violate a constraint like $x + y > 10$.