Exercises for Transaction Systems, summer term 2016
Linnea Passing (linnea.passing@tum.de)
http://www-db.in.tum.de/teaching/ss16/transactions/

Sheet No. 7 – Sample Solution

Exercise 1 (6 points) For the following histories test if they are MVSR or MCSR.

\[ s_1 = w_0(x_0) \ w_0(y_0) \ w_0(z_0) \ c_0 \ r_3(x_0) \ w_3(x_3) \ c_3 \ w_1(x_1) \ c_1 \ r_2(x_1) \ w_2(y_2) \ w_2(z_2) \ c_2 \]
\[ s_2 = w_0(x_0) \ w_0(y_0) \ c_0 \ w_1(x_1) \ c_1 \ r_3(x_1) \ w_3(x_3) \ r_2(x_1) \ c_3 \ w_2(y_2) \ c_2 \]
\[ s_3 = w_0(x_0) \ w_0(y_0) \ c_0 \ w_1(x_1) \ c_1 \ r_2(x_1) \ w_2(y_2) \ c_2 \ r_3(y_0) \ w_3(x_3) \ c_3 \]

Solution

- \( s_1 \) is mono-version and serial, thus in MCSR and MVSR
- \( s_2 \) is equivalent to \( t_0 t_1 t_2 t_3 \) and \( t_1 t_2 t_3 t_0 \), thus in MVSR. The multi-version conflict graph is empty, thus \( s_2 \) is also in MCSR.
- \( s_3 \) is equivalent to \( t_0 t_3 t_1 t_2 \) and \( t_1 t_2 t_0 t_3 \), thus in MVSR. The multi-version conflict graph contains only one edge (\( t_2 \) to \( t_3 \)), thus \( s_3 \) is also in MVSR.

Exercise 2 (5 points) For the schedule

\[ m = w_0(x_0) \ w_0(y_0) \ c_0 \ r_1(x_0) \ w_1(x_1) \ r_2(x_1) \ w_2(y_2) \ w_1(y_1) \ w_3(y_3) \]

Test whether there exists an order \( << \) such that \( MVSG(m,<<) \) is acyclic. If there is an acyclic graph, find an appropriate version function for a final transition \( t_\infty \) such that the graph remains acyclic.

Solution Because of the read operations \( r_1(x_0) \) and \( r_2(x_1) \), the edges \( t = 0 \rightarrow t_1 \) and \( t_1 \rightarrow t_2 \) are created. Next, assume that either \( x_0 << x_1 \) (which creates no new edge) or \( x_1 << x_0 \) (which creates the edge \( t_2 \rightarrow t_0 \)). We see that the latter assumption results in a circle, thus \( x_0 << x_1 \) is the version order we are looking for. A complete version order might look like this:

\[ x_0 << x_1 \]
\[ y_0 << y_1 << y_2 << y_3 \]

We add an infinity transaction that reads \( x_1 \) and \( y_3 \) (formal: \( h(r_\infty(x)) = w_1(x_1), h(r_{\infty f t y}(y)) = w_3(y_3) \)). This results in the new MVSG edges \( t_0 \rightarrow t_3, t_1 \rightarrow t_3, \) and \( t_2 \rightarrow t_3 \). As we can see, the graph remains acyclic.

Exercise 3 (5 points) For the schedule

\[ s = w_1(x) \ c_1 \ r_2(x) \ r_3(x) \ c_2 \ r_4(x) \ w_3(x) \ c_4 \ c_3 \]

give the resulting schedule under the MVTO protocol.
**Solution** \( w_1(x_0)\ c_1\ r_2(x_1)\ c_2\ r_3(x_1)\ r_4(x_0)\ a_3\ c_4 \)

Transaction \( t_3 \) has to abort because it creates a new version of \( x \), but the newer transaction \( t_4 \) has already read the old version of \( x \).

**Exercise 4 (5 points)** For the schedule

\[
s = r_1(x)\ w_1(x)\ r_2(x)\ w_2(y)\ r_1(y)\ w_2(x)\ c_2\ w_1(y)\ c_1
\]

give the resulting schedule under the 2V2PL protocol.

**Solution** \( rl_1(x)\ r_1(x_0)\ wl_1(x)\ w_1(x_1)\ rl_2(x)\ r_2(x_0)\ r_2(x_0)\ wl_2(y)\ w_2(y_2)\ rl_1(y)\ w_1(y_0) \)

The next step of both transactions, \( wl_2(x) \) and \( w_1(y) \) cause a deadlock because uncommitted writes to both \( x \) and \( y \) exist. Now, the 2V2PL scheduler has to detect this deadlock and abort one transaction.