Coursework 1: Answers

Mark Handley

Question 1:

cost Max = 4
ACCOUNT(I=0) = ACCOUNT[I],
ACCOUNT[i:0..Max] = ( read[i] -> ACCOUNT[i]
| write[n:0..Max] -> ACCOUNT[n]).

TRANSACTION = (from.read[i:0..Max] -> transfer[j:0..i]
| from.write[i-j] ->
| to.read[k:0..Max] -> to.write[k+j] ->
| TRANSACTION).

||MOVEMONEY = (your:ACCOUNT(2) || my:ACCOUNT(0) ||
| a:TRANSACTION || b:TRANSACTION)
/{{a.from.read,b.to.read})/your.read,
| {b.from.read,a.to.read}/my.read,
| {a.to.write,b.from.write}/my.write,
| {b.to.write,a.from.write}/your.write}. 
Q1: Check Progress

Progress violation for actions:
\{a, b\}.\{from.\{read, write\}[1..4], to.\{read[1..4], write[1..8]\}, transfer[1..4]\}

Trace to terminal set of states:
- a.from.read.2
- b.from.read.0
- a.transfer.2
- a.from.write.0
- a.to.read.0
- a.to.write.2
- a.from.read.0
- a.transfer.0
- a.from.write.0
- b.transfer.0
- b.from.write.0

Actions in terminal set:
\{a, b\}.\{\{from, to\}.\{read, write\}[0], transfer[0]\}

Q1: Interference: losing money

\begin{tabular}{ll}
\textbf{a. TRANSACTION} & \textbf{b. TRANSACTION} \\
\texttt{a.from.read.2} & \texttt{(YOUR==2)} \\
\texttt{a.transfer.2} & \texttt{b.from.read.0} \texttt{(MY == 0)} \\
\texttt{a.from.write.0} & \texttt{(0 \textarrow{->} YOUR)} \\
\texttt{a.to.read.0} & \texttt{(MY == 0)} \\
\texttt{a.to.write.2} & \texttt{(2 \textarrow{-> MY)} \\
\end{tabular}
Q1: Interference: gaining money

<table>
<thead>
<tr>
<th>a TRANSACTION</th>
<th>b TRANSACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.from.read.2 (YOUR == 2)</td>
<td>b.from.read.0 (MY == 0)</td>
</tr>
<tr>
<td>b.transfer.0</td>
<td></td>
</tr>
<tr>
<td>b.from.write.0 (MY == 0)</td>
<td></td>
</tr>
<tr>
<td>b.to.read.2 (YOUR == 2)</td>
<td></td>
</tr>
<tr>
<td>a.transfer.2</td>
<td></td>
</tr>
<tr>
<td>a.from.write.0 (0 -&gt; YOUR)</td>
<td></td>
</tr>
<tr>
<td>a.to.read.0 (MY == 0)</td>
<td></td>
</tr>
<tr>
<td>a.to.write.2 (2 -&gt; MY)</td>
<td></td>
</tr>
<tr>
<td>b.to.write.2 (2 -&gt; YOUR)</td>
<td></td>
</tr>
</tbody>
</table>

Q2: Adding Locks to Accounts

LOCK = (lock -> unlock -> LOCK).

TRANSACTION = (from.lock -> to.lock ->
from.read[i:0..Max] -> transfer[j:0..i] ->
from.write[i-j] ->
to.read[k:0..Max] -> to.write[k+j] ->
from.unlock -> to.unlock -> TRANSACTION).

||MOVEMONEY = (your:ACCOUNT(2) || my:ACCOUNT(0) ||
your:LOCK || my:LOCK ||
a:TRANSACTION || b:TRANSACTION)
/\{a.from.read,b.to.read}/your.read,
{b.from.read,a.to.read}/my.read,
{a.to.write,b.from.write}/my.write,
{b.to.write,a.from.write}/your.write,
{a.from.lock,b.to.lock}/your.lock,
{a.from.unlock,b.to.unlock}/your.unlock,
{a.to.lock,b.from.lock}/my.lock,
{a.to.unlock,b.from.unlock}/my.unlock}. 
Q2: Deadlock

a.from.lock
- a acquires the lock on your.ACCOUNT

b.from.lock
- b acquires the lock on my.ACCOUNT

Now:

a needs to acquire the lock on my.ACCOUNT
b needs to acquire the lock on your.ACCOUNT

DEADLOCK.

Q3: Solutions: A Single Lock

LOCK = (lock -> unlock -> LOCK).

TRANSACTION = (lock ->
  from.read[i:0..Max] -> transfer[j:0..i] ->
  from.write[i-j] ->
  to.read[k:0..Max] -> to.write[k+j] ->
  unlock -> TRANSACTION).

||MOVEMONEY = (your:ACCOUNT(2) || my:ACCOUNT(0) ||
  {a,b}::LOCK ||
  a:TRANSACTION || b:TRANSACTION)

Problem: removes all concurrency from the system. Not a problem here, but would not work for a large back with many accounts and many transactions.
Q3 Solutions: Asymmetry

\[ \text{TRANSACTION1} = (\text{from.lock} \rightarrow \text{to.lock} \rightarrow \]
\[ \text{from.read}[i:0..\text{Max}] \rightarrow \text{transfer}[j:0..i] \rightarrow \]
\[ \text{from.write}[i-j] \rightarrow \]
\[ \text{to.read}[k:0..\text{Max}] \rightarrow \text{to.write}[k+j] \rightarrow \]
\[ \text{from.unlock} \rightarrow \text{to.unlock} \rightarrow \text{TRANSACTION}). \]
\[ \text{TRANSACTION2} = (\text{to.lock} \rightarrow \text{from.lock} \rightarrow \]
\[ \text{from.read}[i:0..\text{Max}] \rightarrow \text{transfer}[j:0..i] \rightarrow \]
\[ \text{from.write}[i-j] \rightarrow \]
\[ \text{to.read}[k:0..\text{Max}] \rightarrow \text{to.write}[k+j] \rightarrow \]
\[ \text{from.unlock} \rightarrow \text{to.unlock} \rightarrow \text{TRANSACTION}). \]
\[ || \text{MOVEMONEY} = (\text{your:ACCOUNT}(2) || \text{my:ACCOUNT}(0) || \]
\[ \text{your:LOCK} || \text{my:LOCK} || \]
\[ a: \text{TRANSACTION1} || b: \text{TRANSACTION2}) \]

Problem: \text{my:LOCK} is not needed as \text{your:LOCK} covers it. This reduces to a complex single-lock system.

Q3 Solutions: Backoff.

\[ \text{TRANSACTION} = \]
\[ (\text{from.lock} \rightarrow \]
\[ (\text{to.lock} \rightarrow \]
\[ \text{from.read}[i:0..\text{Max}] \rightarrow \text{transfer}[j:0..i] \rightarrow \]
\[ \text{from.write}[i-j] \rightarrow \]
\[ \text{to.read}[k:0..\text{Max}] \rightarrow \text{to.write}[k+j] \rightarrow \]
\[ \text{to.unlock} \rightarrow \text{from.unlock} \rightarrow \text{TRANSACTION} \rightarrow \]
\[ \text{from.unlock} \rightarrow \text{TRANSACTION}). \]

In an implementation, the backoff would have to be for a random interval to avoid persistent collisions. This adds delay and complication.
Q3: Solutions

\[
\text{TRANSACTION} = (\text{from.lock} \rightarrow \text{from.read}[i:0..\text{Max}] \rightarrow \\
\text{transfer}[j:0..i] \rightarrow \text{from.write}[i-j] \rightarrow \\
\text{from.unlock} \rightarrow \\
\text{to.lock} \rightarrow \\
\text{to.read}[k:0..\text{Max}] \rightarrow \text{to.write}[k+j] \rightarrow \\
\text{to.unlock} \rightarrow \\
\text{TRANSACTION}).
\]

- There are really two critical sections, one for each read..write pair.
- Each needs to be protected by a lock, but there’s no need to lock both accounts at the same time.
- This prevents the deadlock while still permitting concurrency.