Suggested Solutions for
Written Exam, December 7, 2016

PROBLEM 1

Question 1.1

The direct translation of the semaphore operations may subsequently be reduced to:

\[ B \ast C \]

[The place \((\ast)\) is a reminiscent of \(P_B\)'s control loop preventing \(B\) from firing concurrently with itself.]

Question 1.2

[It is seen that after \(A\) followed by the first \(B\), \(C\) may execute. Consuming the \(SC\) signal generated by the second \(B\), \(C\) will then have to wait for \(A\) again. From the net, it is readily observed that \(A\) must wait for \(C\) except for the first time.]

\[ I \triangleq c \leq a \leq c + 1 \]

Question 1.3

Moving the \(p(SA)\) operation to the end of \(P_A\), the semaphore signalling may be directly translated to CSP communications:

```
  process P1;
  repeat
    A;
    P2!();
    P2!();
    P3?();
  forever

  process P2;
  repeat
    P1?();
    B;
    P3!();
  forever

  process P3;
  repeat
    P2?();
    C;
    P2?();
    P2?();
  forever
```

Alternatively, \(P_2\) may do \(B\) twice in each round synchronizing with \(P_3\) inbetween.

```
  process P1;
  repeat
    A;
    P2!();
    P3?();
  forever

  process P2;
  repeat
    P1?();
    B;
    P3!();
  forever

  process P3;
  repeat
    P2?();
    C;
    P2?();
    P1!();
  forever
```
PROBLEM 2

Question 2.1

(a) The statement pairs are checked for critical references with respect to each other:

<table>
<thead>
<tr>
<th>Pair</th>
<th>Mutually atomic</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a, b$</td>
<td>NO</td>
<td>Two critical references in $b$.</td>
</tr>
<tr>
<td>$a, c$</td>
<td>NO</td>
<td>In $c$, writing $y$ and reading $x$ are both critical.</td>
</tr>
<tr>
<td>$a, d$</td>
<td>YES</td>
<td>In $d$ the only critical reference is the writing of $x$.</td>
</tr>
<tr>
<td>$b, c$</td>
<td>YES</td>
<td>Only one critical reference in each statement.</td>
</tr>
<tr>
<td>$b, d$</td>
<td>NO</td>
<td>In $b$, both reading and writing of $x$ are critical.</td>
</tr>
<tr>
<td>$c, d$</td>
<td>NO</td>
<td>Two critical references in both $c$ and $d$.</td>
</tr>
</tbody>
</table>

[Being executed indivisibly, readings and writings in statement $a$ count as a single critical reference.]

(b) Transition diagrams:

\[
\begin{align*}
  c: & \\
  k_0 & \xrightarrow{c_1} k_1 & t_1 & := y + x \\
  k_1 & \xrightarrow{c_2} k_2 & t_2 & := y + x + 3 \\
  k_2 & & d_1 & := t_1 \\
  k_1 & & d_2 & := t_2 - 1
\end{align*}
\]

[Location and action labels not required.]

(c) Going through the six possible interleavings of the atomic actions, the possible final values of $(x, y)$ are found to be:

$(-1, 2), (-1, 3), (2, 3)$

Question 2.2

(a) $P$ is preserved by $c$ only.

$Q$ is preserved by all three actions $a$, $b$, and $c$. [By $a$ since $Q$ implies $x \geq 1$. By $b$ since it cannot be executed when $Q$ holds.]

$R$ is preserved by $a$ and $b$.

Question 2.3

(a) The states $(0, 0), (2, 1)$ repeated forever will satisfy all parts of $F$.

(b) Assuming $F$, we may conclude the invariant $0 \leq y \leq x$. Hence either $x > 0$ or $x = 0$ in which case also $y = 0$. Therefore the guard of $a_3$ is constantly satisfied and $a_3$ is guaranteed to be executed under the assumption of weak fairness. No other guards are guaranteed to be constantly true.
(c) It follows from (b), that \( a_3 \) is guaranteed to be executed under strong fairness as well. Assuming \( F \), we may directly induce that \( \square \diamond y = 1 \) and hence \( a_2 \) will be executed under strong fairness. It also follows that \( \square \diamond x > 0 \), but not that \( x \) has to be 1. Also the conditions \( x > 0 \) and \( y = 0 \) need not occur at the same time. Thus, only \( a_2 \) and \( a_3 \) are guaranteed to be executed under the assumption of strong fairness.

**PROBLEM 3**

**Question 3.1**

(a) By initializing \( S \) to the full range, any early call of \texttt{get} will block until the first message is sent.

(b) A call of \texttt{send} should be waiting only if less than the required number of receivers have got the message:

\[
I \triangleq \text{waiting}(\text{Done}) > 0 \Rightarrow |S| < \text{lim}
\]

Initially, \( I \) trivially holds because \( \text{Done} \) is empty. When entering \( \text{Done} \), \( S \) is empty and \( \text{lim} \geq 1 \) and hence \( I \) holds. Since \( S \) is grown by one element at a time, when \( |S| < \text{lim} \) ceases to hold, \( S = \text{lim} \) in which case \( \text{Done} \) is signalled. Since only one sender is assumed, this will render \( \text{Done} \) empty, satifying \( I \).

(c) Since no check is made after the two calls of \texttt{wait}, in case of spurious wake-ups, the monitor would just be left. In \texttt{send}, then the required receivers might not all have got the message and in \texttt{get}, a message may be received again.

To deal with spurious wake-ups, proper conditions will have to be checked after the calls of \texttt{wait}.

In \texttt{get}, the given wait condition should just be rechecked:

\[
\text{function get(id : Range) returns Message} \{
\quad \text{while id} \in S \text{ do wait(NewMsg)};
\}
\]

In \texttt{send}, the condition must reflect \( I \):

\[
\text{procedure send(msg : Message, k : Range) } \{
\quad \text{while } |S| < \text{lim} \text{ do wait(Done)}
\}
\]

**Question 3.2**

In general, if a message is sent with \( k = n \), all receivers must have obtained the message before the sender can send a new message. If sent with \( k < n \), the message may be overwritten before every receiver has got it. From this, it follows that the receiver with \( id = 1 \) will receive messages \( a \) and \( d \), but maybe not messages \( b \), and \( c \). Hence the possible received message sequences are:

\[
a, b, c, d \\
a, c, d \\
a, b, d \\
a, d
\]
Question 3.3

[One of the processes, say $P_m$, is appointed barrier coordinator. This may await the arrival of the other processes by sending a dummy message to all of them. Afterwards the other processes should wait for another dummy message from the coordinator in order to leave the barrier. By waiting for all others having got the second message, the coordinator becomes ready for the next round:]

\[
\text{process } P_i[i : 1..m - 1] \quad \text{process } P_m
\]

\[
\begin{align*}
\text{repeat} & \quad \text{repeat} \\
\quad & \quad \\
msg_i & := \text{Multicast.get}(i); \quad \text{Multicast.send(VoidMsg, } m - 1); \\
msg_j & := \text{Multicast.get}(i); \quad \text{Multicast.send(VoidMsg, } m - 1); \\
\quad & \quad \\
\text{forever} & \quad \text{forever}
\end{align*}
\]

[Here the $msg_i$ variables are dummy message variables and VoidMsg is a dummy message.]

Question 3.4

Once the server process accepts a call of $\text{send}(msg, k)$ it should pass the message to $k$ distinct receivers before ending the rendezvous. After this, further receivers may pick up the message at the outermost level until the next message is sent:

\[
\text{module Multicast}
\]

\[
\begin{align*}
\text{op } & \text{send}(msg : \text{Message}, k : \text{Range}); \\
\text{op } & \text{get}(id : \text{Range}) \text{ returns } \text{Message};
\end{align*}
\]

\[
\begin{align*}
\text{body}
\end{align*}
\]

\[
\text{process } Control;
\]

\[
\begin{align*}
\text{var } & \text{cur : Message;} \\
S : & \text{ set of Range } := \text{Range}; \\
\text{repeat} & \quad \text{repeat} \\
\text{in } & \text{send(msg, k)} \quad \text{in Multicast.send(VoidMsg, } m - 1); \\
\rightarrow & \text{cur := msg;} \quad \text{Multicast.send(VoidMsg, } m - 1); \\
S & := \{ \}; \quad \text{Multicast.send(VoidMsg, } m - 1); \\
\text{for } & i \text{ in } 1..k \text{ do} \\
\quad & \text{in get(id) and id } \notin S \rightarrow S := S \cup \{id\}; \\
\text{ni} & \quad \text{ni} \\
\quad & \text{for i in 1..k do} \\
\quad & \text{in get(id) and id } \notin S \rightarrow S := S \cup \{id\}; \\
\text{ni} & \quad \text{ni} \\
\text{return cur} & \quad \text{return cur} \\
\text{ni} & \quad \text{ni} \\
\text{forever;} & \quad \text{forever;}
\end{align*}
\]

\[
\text{end Multicast;}
\]