Suggested Solutions for
Written Test, December 12, 2017

PROBLEM 1

Question 1.1

(a) Transition diagrams:

\[ \begin{align*}
\text{\(P_1:\)} & \quad a_1: x := 3 \\
& \quad b_1 \\
& \quad a_2: y := y + 1 \\
& \quad b_2 \\
\end{align*} \]

\[ \begin{align*}
\text{\(P_2:\)} & \quad b_0: t := x \\
& \quad b_1: \ t := t + 2 \\
& \quad b_2: \ y := \ t + 2 \\
& \quad b_3: x := 1 \\
\end{align*} \]

[Location and action labels not required.]

(b) It is observed that the final value of \(x\) can become 3 only if \(P_2\) is fully executed before \(P_1\). In that case, \(y\) will become 3. Otherwise, \(P_2\) will read \(x\) as either 0 or 3 before setting \(y\) and finally setting \(x\) to 1. Furthermore, \(P_1\) may or may not have incremented \(y\) before it is set by \(P_2\). This leaves the final value of \(y\) to be one of \(\{2, 3, 5, 6\}\) when \(x\) ends at 1."

It is found that the possible final states \((x, y)\) are:

\((1, 2), (1, 3), (1, 5), (1, 6), (3, 3)\)

Question 1.2

(a) \(P\) is preserved by \(a\) and \(c\). [By \(a\) since \(x + (x + 1)\) cannot become 0 for any integer \(x\).]

\(Q\) is preserved by \(a\), \(b\), and \(c\). [By \(c\) since the guard is not satisfied.]

\(R\) is preserved by \(c\). [As the guard ensures that both \(x\) and \(y\) are positive before the action and hence \(x > 0\) after too.]

Question 1.3

(a) The state sequence \((0, 0), (0, 1)\) repeated forever will satisfy all parts of \(F\).

(b) \(G\) follows from \(F\). Infinitely often \(y = 0\) and then either \(x = 0\) or \(x > 0\) and hence \(x > y\) leading to \(x = 0\).

\(H\) does not follow from \(F\). For instance \(F\) is satisfied by the sequence \((1, 0), (0, 1)\) repeated forever, but \(H\) is not.

\(I\) does not follow from \(F\) as it allows \(x\) to remain 0 as in the sequence in (a).

\(J\) follows from \(F\). If at any time \(x\) remains 0 in all future, \(\Box \Diamond x \neq y\) will lead to \(y\) becoming non-zero and hence \(y > 0\).
PROBLEM 2

Question 2.1

The \( P \) operation may be implemented by getting a positive number of tokens and returning the surplus ones:

\[
P : \{ \text{var } c : \text{nat} := \text{TokBuf.get}(1); \text{ if } c > 1 \text{ then } \text{TokBuf.put}(c - 1) \} \\
V : \text{TokBuf.put}(1)
\]

As \( \text{get}(1) \) will return at least one token and as \( \text{put} \) may be safely called with zero tokens, the \( P \) operation may be simplified to:

\[
P : \{ \text{var } c : \text{nat} := \text{TokBuf.get}(1); \text{TokBuf.put}(c - 1) \}
\]

or even:

\[
P : \text{TokBuf.put}(\text{TokBuf.get}(1) - 1)
\]

Question 2.2

(a) \textbf{monitor} \textit{TokBuf}

\[
\text{var } \text{count} : \text{nat} := C_0; \\
\text{Check : } \text{condition;}
\]

\textbf{procedure} \textit{put}(n : \text{nat}) \{ \\
\text{count} := \text{count} + n; \\
\text{signal}_\text{all}(\text{Check})
\}

\textbf{function} \textit{get}(t : \text{nat}) \text{returns} \text{nat} \{
\text{var } r; \\
\text{while } \text{count} < t \text{ do } \text{wait}(\text{Check}); \\
r := \text{count}; \\
\text{count} := 0; \\
\text{return} r
\}

\]

(b) The following should be an invariant of the monitor:

For any call \( \text{get}(t) \) waiting in \( \text{Check} \), \( t > \text{count} \)

(c) [To avoid unnecessary signalling, a variable \( \text{min} \) is used to hold the smallest threshold value of any \( \text{get}-\text{call} \) waiting in \( \text{Check} \). As long as \( \text{count} \) has not reached this value, no signalling is necessary.]

In the declaration part of the monitor, the following variable is added:

\[
\text{var } \text{min} : \text{nat}^* := \infty;
\]

where \( \text{nat}^* = \text{nat} \cup \{\infty\} \).
In put, \textit{signal\_all}(Check) is replaced by:

\begin{verbatim}
if count \geq min then \{ min := \infty; signal\_all(Check) \}
\end{verbatim}

Finally, in get, the \textbf{while}-loop is replaced by:

\begin{verbatim}
while count < t do \{ if t < min then min := t;
wait(Check) \}
\end{verbatim}

(d) The solution to (c) still works in the presence of spurious wakeups as the invariant, that \textit{min} is less or equal to the threshold \textit{t} of any call \textit{get}(t) waiting in \textit{Check}, is unaffected by calls spuriously leaving the condition queue and because the waiting condition is rechecked in the \textbf{while} test.

\textbf{Question 2.3}

[Reader/writer synchronization may be achieved by using a token for each of the \textit{N} readers. A reader must obtain one of these for reading, while a writer must take them all.]

The module \textit{TokBuf} should be initialized with \textit{C_0} = \textit{N}.

Readers: \begin{verbatim}
TokBuf.put(TokBuf.get(1) - 1);
reading
TokBuf.put(1);
\end{verbatim}

Writers: \begin{verbatim}
TokBuf.get(N);
writing
TokBuf.put(N);
\end{verbatim}

[As in Question 2.1, a single token may be obtained with a combined \textit{get}/\textit{put} call. The solution is neither fair towards readers nor writers.]

\textbf{PROBLEM 3}

\textbf{Question 3.1}
**Question 3.2**

In order to control the ending barrier one of the processes, \( P_n \), is appointed coordinator for the barrier synchronization. Otherwise, the solution reflects the synchronization pattern shown in the Petri Net.

\[
\begin{align*}
\text{var } & SA[1..n] : \text{semaphore}; & \quad \text{— previous } A\text{'s done} \\
& SB[1..n - 1] : \text{semaphore}; & \quad \text{— } B_i \text{ done} \\
& SC[1..n - 1] : \text{semaphore}; & \quad \text{— OK to restart } P_i
\end{align*}
\]

All semaphores are initialized to 0

\[
\begin{align*}
\text{process } & P[i : 1..n - 1]; \\
\text{repeat} & A_i; \\
& \text{for } j \text{ in } 1..i - 1 \text{ do } P(SA[i]); \\
& \text{for } j \text{ in } i + 1..n \text{ do } V(SA[j]); \\
& B_i; \\
& V(SB[i]); \\
& P(SC[i]) \\
\text{forever}
\end{align*}
\]

[As \( SA[1] \) is not effectively used, it could be eliminated. It is possible to use common semaphore \( SB \) instead of \( SB[1..n] \). The wait for previous \( A \)'s may also be done by sending a ripple signal down the line.]

**Question 3.3**

By accepting the \( doneA(i) \) calls in order, it is ensured that all previous \( A \)'s have been executed. The final barrier may be implemented in the standard way.

\[
\begin{align*}
\text{module } & \text{Synch} \\
& \text{op } doneA(i : \text{integer}); \\
& \text{op } doneB(); \\
\text{body} \\
& \text{process } Control; \\
& \text{repeat} \\
& \quad \text{for } j \text{ in } 1..n \\
& \quad \text{in } doneA(i) \text{ and } i = j \rightarrow \text{skip } ni \\
& \quad \text{in } doneB() \text{ and } ?doneB = n \rightarrow \\
& \quad \text{for } j \text{ in } 1..n - 1 \text{ do} \\
& \quad \quad \text{in } doneB() \rightarrow \text{skip } ni \\
& \quad ni \\
& \text{forever} \\
& \text{end Synch;}
\end{align*}
\]