Parallel Usage Checking – An Observation

Barry M Cook

4Links Limited, UK

SpaceWire Designers, Consultants, Manufacturers
Parallel Usage Checking ...

X: -- a “shared” variable
PAR
A - a process that ‘uses’ X
B - a process that ‘uses’ X

\[ \text{X:} \]

\[
\begin{array}{c|c|c}
\text{A} & \text{B} & \text{X:} \\
\end{array}
\]

… detects a possible problem if A or B or both write to X

Require: **CREW – Concurrent Read, Exclusive Write**

20110620, CPA2011, *Parallel Usage Checking – An Observation*, B M Cook, 4Links Limited
An acceptable program:

Give each parallel process its own variable and send data between processes through a communication channel

C: -- a communication channel

X: A     Y: B

X is local to A and Y is local to B and there is no conflict
A typical use:

Process A (say) places data in its variable, X, and – at some point – passes these values to process B where they are stored in its local variable, Y, for use.

\[
\begin{align*}
\text{C: -- a communication channel} & \\
\text{X:} & \quad \text{SEQ} \quad \text{…} \quad \text{-- fill X} \quad \text{…} \quad \text{C ! X} \quad \text{…} \\
\text{Y:} & \quad \text{SEQ} \quad \text{…} \quad \text{-- use Y} \quad \text{…}
\end{align*}
\]

Note: the relative time of action is NOT accurately represented by the relative positions of words in the above picture …

In fact, the communication \textit{synchronises} the processes
Re-drawing the picture:

C: -- a communication channel

<table>
<thead>
<tr>
<th></th>
<th>Y:</th>
</tr>
</thead>
<tbody>
<tr>
<td>X:</td>
<td>SEQ</td>
</tr>
<tr>
<td>SEQ</td>
<td>SEQ</td>
</tr>
<tr>
<td>A₁</td>
<td>B₁</td>
</tr>
<tr>
<td>C ! X</td>
<td>C ? Y</td>
</tr>
<tr>
<td>A₂</td>
<td>B₂</td>
</tr>
</tbody>
</table>

The synchronising communication divides each process into temporally distinct parts – and we can see that it is perfectly safe for A₁ and B₂ (or B₁ and A₂) to use a shared variable:

C: -- a synchronising channel (no data)
X: -- a shared variable

<table>
<thead>
<tr>
<th></th>
<th>SEQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>A₁</td>
<td>B₁ -- uses X</td>
</tr>
<tr>
<td>C !</td>
<td>C ?</td>
</tr>
<tr>
<td>A₂</td>
<td>B₂ -- uses X</td>
</tr>
</tbody>
</table>
Efficiency

Using separate variables and communicating (possibly a large amount of) data can be slow.

Using a shared variable requires no data transfer and can be much more efficient.
Formalising the efficient version

We can re-write the shared variable version:

C: -- a synchronising channel (no data)
X: -- a shared variable

\begin{center}
\begin{tabular}{ccc}
  SEQ & SEQ \\
  $A_1$ & $B_1$ \\
  $C!$ & $C?$ \\
  $A_2$ & $B_2$
\end{tabular}
\end{center}

As:

C: -- a synchronising channel (no data)
X: -- a shared variable

SEQ

\begin{center}
\begin{tabular}{ccc}
  $A_1$ & $B_1$
\end{tabular}
\end{center}

-- communicate

\begin{center}
\begin{tabular}{ccc}
  $A_2$ & $B_2$
\end{tabular}
\end{center}

And parallel-usage check it in the usual way.
Or

<table>
<thead>
<tr>
<th>X: -- variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>C: -- channel</td>
</tr>
<tr>
<td>PAR</td>
</tr>
<tr>
<td>SEQ</td>
</tr>
<tr>
<td>A₁</td>
</tr>
<tr>
<td>C !</td>
</tr>
<tr>
<td>A₂</td>
</tr>
<tr>
<td>SEQ</td>
</tr>
<tr>
<td>B₁</td>
</tr>
<tr>
<td>C ?</td>
</tr>
<tr>
<td>B₂</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X: -- variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>C: -- channel</td>
</tr>
<tr>
<td>SEQ</td>
</tr>
<tr>
<td>PAR</td>
</tr>
<tr>
<td>A₁</td>
</tr>
<tr>
<td>B₁</td>
</tr>
<tr>
<td>-- communicate</td>
</tr>
<tr>
<td>PAR</td>
</tr>
<tr>
<td>A₂</td>
</tr>
<tr>
<td>B₂</td>
</tr>
</tbody>
</table>

is equivalent to

Represents the required solution

Proves that it is safe - but is probably not a good implementation
But …

We do need to make sure the behaviour is adequately controlled and the transformation is valid.

e.g. a loop in the above example …

\[
\text{WHILE TRUE}
\begin{align*}
\text{SEQ} \\
A_1 \\
C! \\
A_2
\end{align*}
\]

… means that \( A_1 \) is both before and after \( A_2 \)

(a solution is to use another communication to synchronise after \( A_2/B_2 \))
Why not just write the transformed version?

1. It may be less efficient (see above)

2. "There are two ways of constructing a software design: one way is to make it so simple that there are obviously no deficiencies and the other is to make it so complicated that there are no obvious deficiencies."
   
   Professor Sir C.A.R "Tony" Hoare

It is better to write programs in a way that reflects the problem solution – and is easily seen to be correct.
Conclusions

Parallel usage checking is required.

Program transformation before checking can allow a larger range of acceptable programs
… that may be more efficiently implemented

We often think of program transformations as steps towards implementation

I suggest that we might also use (possibly different) transformations purely / additionally as steps towards correctness checking