A correction on EWD651.

The day after I had mailed the copies of EWD651 to its various recipients I discovered that it was miserably wrong: the transfer from the L-group to the R-group did not work properly. In the new version the boolean \( L \) is replaced by the four-valued integer \( k \).

A notational difference is the introduction of the integers \( p_L \) and \( p_R \), counting the numbers of blocked processes in the L-group and in the R-group respectively. The former variables \( w_L \) and \( w_R \) have disappeared, their values being \( p_L - n_L \) and \( p_R - n_R \) respectively.

The integer \( k \) controls whether a process with a false guard will arrive in the L-group or in the R-group. In contrast to EWD651, in which the value of \( L \) was left undefined when both groups were empty, we have now decided that the first process to be blocked will come in the R-group, thus being faithful to the intention of maintaining \( m = 0 \) or \( p_L = 0 \) or \( p_R > 0 \). Initially we have \( k = 1 \). We shall now describe the meaning of the variable \( k \).

\( k = 0 \).

The process finding its guard false either just entered the critical activity via \( P(m) \) or is retesting its guard; in the latter case it came from the L-group. In either case it is directed towards the L-group. During the test of a guard with \( k = 0 \), we have \( p_R = n_R > 0 \), and all the processes in the R-group have a false guard.

\( k = 1 \).

If the process finding its guard false just entered the critical activity via \( P(m) \), we had \( p_L = p_R = 0 \), and the process is entered into the R-group. If the process finding its guard false is retesting its guard, it came from the R-group and returns to it, and the values of the guards of the processes in the L-group --if any-- are unknown.

\( k = 2 \).

This state, which is one of the transfer states, cannot occur with \( m = 1 \),
, hence a process finding its guard false has not just entered the critical activity. The process that is retesting its guard came from the L-group and will be directed into the R-group. The state \( k \geq 2 \) remains until the L-group is empty, so as to ensure that all L-processes escape or become an R-process before a new process is admitted via \( P(m) \). This is done in order to exclude infinite overtaking of a process in the L-group. During \( k = 2 \) we have \( p_R = n_R \), and all processes in the R-group --if any-- have a false guard.

\( k = 3 \).

This second transfer state can also not occur with \( m = 1 \). It is only entered when in the "middle" of the transfer of processes from the L-group to the R-group --i.e. when \( k = 2 \) -- one of the processes escapes via \( S \). As soon as that has happened, we are no longer sure that all processes in the R-group have a false guard. Therefore all the processes in the R-group have to retest their guard before the transfer from the L-group to the R-group can be resumed. When with \( k = 3 \) a process finds its guard false, it came from the R-group and will be returned to the R-group, just as in state \( k = 1 \).

The values of the guards of the processes in the L-group --if any-- are unknown, when it has been established that the R-group only contains processes with a false guard and the L-group is not empty, the transfer will be resumed with \( k = 2 \).

When, with \( p_R > 0 \), it has been established that all processes in the R-group have a false guard -- \( p_R = n_R \) -- the primary case distinction is whether the L-group is empty or not. In the first case, the critical activity is terminated via \( V(m) \) with \( k = 0 \), because a new process that blocks itself, should do so in the L-group. In the second case --because when processes from the R-group are tested, the guards of those in the L-group are never known-- those in the L-group have to retest their guard. The last process (re)entering the R-group did so with \( k = 1, 2, \) or \( 3 \); the L-testing has to be resumed with \( k = 0, 2, 2 \) respectively, hence the

\[ \text{do odd}(k) - k := k - 1 \text{ od} \]

Upon completion of an \( S \), when there are no blocked processes, the critical activity is terminated via \( V(m) \) with \( k = 1 \), because the first new
\[ P(m); \]

\[ \text{do non } \text{Bi} \rightarrow \]

\[ \text{if } k = 0 \rightarrow \]

\[ pL, nL := pL + 1, nL + 1; \]

\[ \text{if } pL > nL \rightarrow V(tL) \] \[ pL = nL \rightarrow V(m) \] \[ \text{fi} \]

\[ P(sL); nL := nL - 1; \]

\[ \text{if } nL > 0 \rightarrow V(sL) \] \[ nL = 0 \rightarrow V(tL) \] \[ \text{fi} \]

\[ P(tL); pL := pL - 1 \]

\[ \text{if } k > 0 \rightarrow \]

\[ pR, nR := pR + 1, nR + 1; \]

\[ \text{if } pR > nR \rightarrow V(tR) \]

\[ pR = nR \rightarrow \]

\[ \text{if } pL = 0 \rightarrow k := 0; V(m) \]

\[ \text{if } pL > 0 \rightarrow \text{do odd}(k) \rightarrow k := k - 1 \text{ ad}; \]

\[ \text{if } nL > 0 \rightarrow V(sL) \] \[ nL = 0 \rightarrow V(tL) \] \[ \text{fi} \]

\[ \text{fi} \]

\[ P(sR); nR := nR - 1; \]

\[ \text{if } nR > 0 \rightarrow V(sR) \] \[ nR = 0 \rightarrow V(tR) \] \[ \text{fi} \]

\[ P(tR); pR := pR - 1 \]

\[ \text{ad}; \]

\[ \text{Si} \];

\[ \text{if } pR = 0 \rightarrow \]

\[ \text{if } pL = 0 \rightarrow k := 1; V(m) \]

\[ \text{if } pL > 0 \rightarrow k := 2; \text{if } nL > 0 \rightarrow V(sL) \] \[ nL = 0 \rightarrow V(tL) \] \[ \text{fi} \]

\[ \text{fi} \]

\[ \text{if } pR > 0 \rightarrow \]

\[ \text{do even}(k) \rightarrow k := k + 1 \text{ ad}; \text{if } nR > 0 \rightarrow V(sR) \] \[ nR = 0 \rightarrow V(tR) \] \[ \text{fi} \]

\[ \text{fi} \]

blocked process should be entered into the R-group. Otherwise testing is resumed with priority to the R-group. If the R-group is empty -- possible values of \( k \) are 1, 2, and 3 -- the transfer from the L-group to the R-group is started or continued with \( k = 2 \), because the R-group (being empty) contains.
no processes with a possibly true guard. If the R-group is not empty, the
testing of the R-group is started or continued. The S has been executed
with \( k = 0, 1, 2, \) or \( 3 \); testing will be resumed with \( k = 1, 3, 3 \), hence the
\[
\text{do even}(k) \rightarrow k := k + 1 \text{ od}
\]
independent of the question whether the L-group is empty or not.

**Note.** The integer \( k \) was introduced when I had discovered the need for the
state \( k = 2 \), but not yet the need for the state \( k = 3 \). Had I foreseen
that fourth state, I would have used a second boolean, \( \text{tf} \) say ("transfer"),
and would have coded
\[
\begin{align*}
k = 0 & \quad \text{as} \quad \text{L and non tf} \\
k = 1 & \quad \text{as} \quad \text{non L and non tf} \\
k = 2 & \quad \text{as} \quad \text{L and tf} \\
k = 3 & \quad \text{as} \quad \text{non L and tf}
\end{align*}
\]
and the statements:
\[
\begin{align*}
\text{do odd}(k) \rightarrow k := k - 1 \text{ od} \quad \text{and} \quad \text{do even}(k) \rightarrow k := k + 1 \text{ od}
\end{align*}
\]
simply as:
\[
\begin{align*}
\text{L := true} & \quad \text{and} \quad \text{L := false}
\end{align*}
\]
respectively. (End of note.)

I can only describe the blunder of EWD651 as "most instructive", because
I know exactly how it occurred: we did not stick to our own rules, fell back
into our old bad habits and rushed into coding! Besides that the whole ex-
perience provides a (totally unintended but welcome) confirmation of my often
stated conjecture that pictures give a false sense of security. Although
somewhat humiliated I am actually glad that I blundered so clearly!

I wish everybody a happy 1978!

---

Plataanstraat 5
5671 AL Nuenen
The Netherlands

prof.dr.Edsger W. Dijkstra
Burroughs Research Fellow