After two issues with parallel and functional programming exercises, we return to the design of 'imperative' programs. Both problems address the computation of a maximum value associated with an integer sequence \( X(i: 0 \leq i < N) \). In Exercise 47 this maximum value is the largest segment-wise product of \( X \). This may be determined, without introducing auxiliary arrays, in \( O(N) \) time. I owe this problem to A. Kaldewaij.

Exercise 48, which is due to W.H.J. Feijen, is much harder. A *surpasser* of an array element is a greater value to the right: \( X(j) \) is a surpasser of \( X(i) \) if \( i < j \) and \( X(i) < X(j) \). We have to compute the maximum number of surpassers of the elements of \( X \). The introduction of two auxiliary arrays suffices to arrive at an \( O(N \log N) \) solution.

**Exercise 47: Maximum product**

Find a statement list \( S \) such that

\[
\begin{align*}
[[N: int; \{N \geq 0\}] \\
X(i: 0 \leq i < N): \text{array of int}; \]
\end{align*}
\]

\[
[[r: int; S \]
\]

\[
\begin{align*}
\{r = (\text{MAX } p, q: 0 \leq p \leq q \leq N: \text{ prod}(p, q))\}
\end{align*}
\]

\[]

where, for \( 0 \leq p \leq q \leq N \),

\[
\text{prod}(p, q) = \begin{cases} 
1 & \text{if } p = q \\
\text{prod}(p, q - 1) \ast X(q - 1) & \text{if } p < q.
\end{cases}
\]

**Exercise 48: Number of surpassers**

Solve \( S \) in
\[ [\{N: \text{int}; \{N \geq 1\}\} \]
\[ X(i: 0 \leq i < N): \text{array of int}; \]
\[ [r: \text{int}; \]
\[ S \]
\[ \{r = (\operatorname{MAX} i: 0 \leq i < N: f(i))\} \]
\[ ]\]

where, for \(0 \leq i < N\),
\[ f(i) = (\{Nj: i < j < N: X(i) < X(j)\}). \]

**Solution of Exercise 45 (square recognition, parallel)**

We have to design a parallel program \(P\) with communication behaviour
\[ \langle b!\text{bool}; a?\text{int}; a?\text{int}\rangle^* \]

and input/output relation
\[ b(i) = (\{Aj: 0 \leq j < i: a(j) = a(i+j)\}), \quad i \geq 0. \quad (1) \]

Rather than comparing segments of sequence \(a(j: j \geq 0)\), as is required by (1), we first solve the simpler problem of comparing segments of different sequences: we design a parallel program \(Q\) with communication behaviour
\[ \langle b!\text{bool}; a?\text{int}; c?\text{int}; a?\text{int}\rangle^* \]

and i/o relation
\[ b(i) = (\{Aj: 0 \leq j < i: c(j) = a(i+j)\}), \quad i \geq 0. \quad (3) \]

Using (3), we derive for \(Q\)
\[ b(0) = \text{true} \quad (4) \]

and, for \(i \geq 0\),
\[ b(i+1) = (\{Aj: 0 \leq j < i+1: c(j) = a(i+1+j)\}) = (\{Aj: 0 \leq j < i: c(j) = a(i+1+j)\} \land (c(i) = a(2*i+1))). \quad (5) \]

Comparing the universal quantifications in (5) and (3) calls for the introduction of a subprocess \(p\) of the same type \(Q\) that operates on sequences \(c(j: j \geq 0)\) and \(a(j: j \geq 1)\):
\[ p.c(i) = c(i) \quad \text{and} \quad p.a(i) = a(i+1) \quad (6) \]

for \(i \geq 0\). Since \(p\) is of type \(Q\), it has i/o relation (3):
\[ p.b(i) = (\{Aj: 0 \leq j < i: p.c(j) = p.a(i+j)\}) \]
for $i \geq 0$, or, by (6),
\[ p.b(i) = (A j: 0 \leq j < i: c(j) = a(i + j + 1)). \]

Hence, (5) simplifies to
\[ b(i+1) = p.b(i) \land (c(i) = a(2*i + 1)). \] (7)

As for the outputs, we have external output $b$ and internal outputs (towards subprocess $p$) $p.c$ and $p.a$. Relations (4), (6), and (7) show how the output values may be computed from the input values. We extend the external communication behaviour of $Q$, which is $(b; a; c; a)^*$, in the following way with internal communications:
\[ b; a; c \]
\[ ; (a; p.b; b; p.a; a; p.c; c; p.a)^*. \] (8)

In order to check whether all input values arrive at the appropriate moments, we add in (8) to each communication the number of the communication at that port:
\[ b(0); a(0); c(0) \]
\[ ; (a(2*i + 1); p.b(i); b(i+1); p.a(2*i) \]
\[ ; a(2*i + 2); p.c(i); c(i+1); p.a(2*i + 1) \]
\[ )^*_{i=0}. \]

Comparing the above with relations (4), (6), and (7), we observe that each output value may be computed from the last input values received. Hence, three variables (one for each input port) suffice. Maintaining $z = c(i)$ as an invariant of the repetition, we obtain the following solution:
\[ Q: \quad [\begin{array}{l}
\text{[p: sub Q;}
\x; z: \text{int; y: bool;}
\text{b! true; a? x; c? z}
\text{; (a? x; p.b? y; b! (y \land (z = x)); p.a! x}
\text{; a? x; p.c! z; c? z; p.a! x}
\text{)*}
\]}
\]

The program has constant response time.

We now return to our original problem. We can obtain a solution for $P$ by adding to $Q$ a process $R$ that, given sequence $a$, generates sequence $c$ such that
\[ c(i) = a(i), \quad i \geq 0. \]

This is, of course, a simple process, except that the communication behaviour should concur with (2), i.e. it should be $(a; c; a)^*$. For $R$, $a$ is input and $c$ output. In other words, $R$ is a FIFO-buffer that receives two inputs between any two successive outputs. Its design is not difficult:
\[ c(0) = a(0) \quad \text{and} \quad c(i+1) = a(i+1) = p.c(i) \]
provided subprocess \( p \) (of \( R \)) is also of type \( R \) and \( p.a(i) = a(i+1) \). A possible code for \( R \) is

\[
R: \quad \llbracket p: \text{sub } R; \\
x, z: \text{int}; \\
a?x; c!x \\
; (a?x; p.a!x; a?x; p.c?z; c!z; p.a!x) \rrbracket
\]

The solution for \( P \) is then the following composition of \( R \) and \( Q \):

\[
P: \quad \llbracket p: \text{sub } R; q: \text{sub } Q; \\
x: \text{int}; y: \text{bool}; \\
(q.b?y; b!y \\
; a?x; p.a!x; q.a!x \\
; p.c?x; q.c!x \\
; a?x; p.a!x; q.a!x \\
) \rrbracket
\]

Notice that in the second and fourth line of the repetition all \( a \)-values received are duplicated: they are sent both to \( p \) (for later use as \( c \)-values) and to \( q \). There is also a solution that does not duplicate values. We obtain this solution by adding to \( Q \) an integer output \( d \) with i/o relation

\[
d(i) = a(i), \quad i \geq 0
\]

and communication behaviour (ignoring types and directions)

\[
(b; a; d; c; a)^*.
\]

Then, since the subprocess is of the same type,

\[
p.d(i) = p.a(i)
\]

or, by (6),

\[
p.d(i) = a(i+1)
\]

Consequently,

\[
d(0) = a(0) \quad (9)
\]

and, for \( i \geq 0 \),

\[
d(i+1) = p.d(i). \quad (10)
\]
As (9) and (10) indicate, the addition of output \(d\) (and internal input \(p.d\)) entails just a slight change of \(Q\):

\[
Q': \begin{cases} 
   w, x, z: \text{int}; y: \text{bool}; \\
   b! \text{true}; a?x; d!x; c?z \\
   ; (a?x; p.b?y; b!(y \land (z = x)); p.a!x \\
   ; a?x; p.d?w; d!w; p.c!z; c?z; p.a!x \\
   
\end{cases}
\]

We now have an output \(d\) at which the \(a\)-values are produced at the same rate as they are needed by input \(c\). We can, therefore, obtain a solution for \(P\) by connecting, in cell 0 so to speak, output \(d\) to input \(c\), by which both ports disappear. More precisely, we replace

\[
a?x; d!x; c?z
\]

in the initialization by \(a?z\), and

\[
d!w; p.c!z; c?z
\]

in the repetition by

\[
p.c!z; z := w.
\]

This results in the following solution:

\[
P: \begin{cases} 
   w, x, z: \text{int}; y: \text{bool}; \\
   b! \text{true}; a?z \\
   ; (a?x; p.b?y; b!(y \land (z = x)); p.a!x \\
   ; a?x; p.d?w; p.c!z; z := w; p.a!x \\
   
\end{cases}
\]

As in the preceding solution, we have constant response time.

**Solution of Exercise 46 (square recognition, functional)**

The functional version of the square recognition problem is specified by

\[
[[p :: \text{[int]} \rightarrow \text{[bool]}]; \\
\{ p.a.i = (\forall j: 0 \leq j < i: a.j = a.(i + j)), i \geq 0 \}
\]

As in the preceding exercise, we first design a function that operates on two sequences:

\[
q :: \text{[int]} \rightarrow \text{[int]} \rightarrow \text{[bool]}
\]
given by
\[ q.c.d.i = (A_j: 0 \leq j < i: c_j = d.(i+j)), \quad i \geq 0. \]

Then
\[ p.a = q.a.a \quad (12) \]

We find, similar to (4) and (5),
\[ q.c.(v:d).0 = true \quad (13) \]

and, for \( i \geq 0, \)
\[ q.c.(v:d).(i+1) = q.c.d.i \land (c.i = d.(2*i)) = q.c.d.i A (c.i = d.(2*i)) \quad (14) \]

We introduce a function to compute (14):
\[ r : [bool] \rightarrow [int] \rightarrow [int] \rightarrow [bool] \quad (15) \]

given by
\[ r.e.f.g.i = e.i \land (f.i = g.(2*i)), \quad i \geq 0. \]

By (14) we then have, for \( i \geq 0, \)
\[ q.c.(v:d).(i+1) = r.(q.c.d).c.d.i \quad (16) \]

Formulae (13) and (16) show that function \( q \) may be programmed as follows:
\[ q.c.(v:d) = true : r.(q.c.d).c.d. \quad (17) \]

The program for \( r \) is rather straightforward. For \( i \geq 0 \)
\[ r.(y:e).(z:f).(x:w:g).(i+1) = (y:e).(i+1) \land ((z:f).(i+1) = (x:w:g).(2*i+2)) = e.i \land (f.i = g.(2*i)) = r.e.f.g.i \]

and
\[ r.(y:e).(z:f).(x:w:g).0 = y \land (z = x) \]

Consequently,
\[ r.(y:e).(z:f).(x:w:g) = y \land (z = x) : r.e.f.g \quad (18) \]
Combining—in that order—(12), (11), (17), (15), and (18) yields the following program for $P$:

$P$: 

\[ p.a = q.a.a \]

where

\[
[q :: [int] \rightarrow [int] \rightarrow [bool];
q.c.(v:d) = true : r.(q.c.d).c.d
\]

where

\[
[r :: [bool] \rightarrow [int] \rightarrow [int] \rightarrow [bool];
 r.(y:e).(z:f).(x:w:g) = y \land (z = x) : r.e.f.g
\]

In the computation of $q.c.d.(i+1)$ we have, by following the strategy of Exercise 45, split the domain $0 \leq j < i + 1$ of quantification into $0 \leq j < i$ and $j = i$. Splitting into $j = 0$ and $1 \leq j < i + 1$ would have given a slightly different program. It may easily be checked that

\[
q.(z:c).(v:w:d).(i+1) = (z = (w:d).i) \land q.c.d.i
\]

By introducing a function $s$ that, given $z$, satisfies

\[
s.e.f.i = (z = e.i) \land f.i
\]

we have

\[
q.(z:c).(v:w:d).(i+1) = s.(w:d).(q.c.d).i
\]

This would lead to the following program:

\[
p.a = q.a.a
\]

where

\[
[q :: [int] \rightarrow [int] \rightarrow [bool];
 q.(z:c).(v:w:d) = true : s.(w:d).(q.c.d)
\]

where

\[
[s :: [int] \rightarrow [bool] \rightarrow [bool];
s.(x:e).(y:f) = (z = x) \land y : s.e.f
\]

The same function $s$ occurred, under the name $q$, in the palindrome solution presented in *Small Programming Exercises* 18. I owe the functional solutions of the square problem to Rob Hoogerwoord.