Recognizing Perfect-Shuffles

Joost-Pieter Katoen Berry Schoenmakers
Dept. of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

November 29, 1989

1 Introduction

In the series "Recognizing ..." we present two fine-grained parallel programs dealing with the recognition of perfect-shuffles. A segment $a[m..m{+}2n)$, $0{\le}m \wedge 0{\le}n$, of an infinite sequence a is called *perfect-shuffle invariant* (or, a *perfect-shuffle*, for short) if

$$(\forall j : 0 \le j < 2n : a(m+j) = a(m + 2(j \bmod n) + j \operatorname{div} n)).$$

Here, $a(j)$ denotes the $(j{+}1)$-st element of sequence a $(j{\ge}0)$.

In the next section we derive a parallel program which determines for each segment of N $(N{\ge}1)$ successive inputs whether it is a perfect-shuffle. In section 3 we design a program for determining whether the whole segment thus far received is a perfect-shuffle. Throughout the derivations, performance considerations, such as space and speed requirements, are the driving force for the numerous design decisions made. For both problems, we obtain linear systolic arrays which have constant response time and constant latency.

For an introduction to the calculational design technique for parallel programs used in this note we refer to [1] and [4]. The notation used has been adopted from [1].

2 Fixed-length perfect-shuffles

For fixed N, $N{\ge}1$, we are to design a component PS_N with input a, output b, and i/o-relation

(1) $\quad b(i) \equiv (\forall j : 0 \le j < 2N : a(i+j) = a(i + 2(j \bmod N) + j \operatorname{div} N)),$

for $i{\ge}0$. That is, we have to design a component that determines for each segment $a[i..i{+}2N)$ whether it is a perfect-shuffle. It follows that $b(i)$ refers to all elements of $a[i..i{+}2N)$, and, consequently, that (1) requires a communication behavior like $a^{2N}; (b;a)^*$.

The obvious way to start the derivation is to generalise (1) in some way, thereby obtaining specifications of components PS_n, for $1{\le}n{\le}N$, with the intention that PS_n has PS_{n-1} as subcomponent $(n{\ne}1)$. From experience, however, we know that such components must have a communication behavior that depends on n (e.g., $a^{2n}; (b;a)^*$), and, consequently, that such components do not have identical commands. Moreover, a communication behavior like $a^{2n}; (b;a)^*$ requires component PS_n to count the number of initial communications along a (or other means to detect that the $(2n)$-th communication along a has occurred). This dependence on n makes the components unnecessary complicated.

In order to obtain a simpler communication behavior, like $(a;b)^*$, we consider a slightly different component PS'_N with i/o-relation

(2) $\quad b(i) \equiv (\forall j : 0 \le j < 2N : a(i+j-2N+1) = a(i + 2(j \bmod N) + j \operatorname{div} N - 2N + 1)),$

for $i{\geq}2N{-}1$. Component PS'_N, then, determines for each segment $a(i{-}2N..i]$ with $i{\geq}2N{-}1$ whether it is a perfect-shuffle. Hence, this component may be used to build PS_N as follows:

> **com** PS_N (**in** a:int, **out** b:bool) :
> **sub** $p{:}PS'_N$ **bus**
> $[a = p{\cdot}a]$
> $|[$**var** y:bool
> $(p{\cdot}b?y)^{2N-1}$; $(p{\cdot}b?y; b!y)^*$
> $]|$
> **moc** .

As before, a first step in the derivation of a program for PS'_N could be the introduction of components PS'_n $(1{\leq}n{\leq}N)$ whose i/o-relations are obtained by generalising (2). However, (2) is only defined for $i{\geq}2N{-}1$ and, again from experience, we know that this results in i/o-relations for PS'_n which are partially defined, e.g. only for $i{\geq}2n{-}1$, or, in any case, on a domain dependent on n. Calculations with such i/o-relations tend to be rather difficult. To avoid such complications we extend relations like (2), mentally, to all natural i by defining $a(j)$ for $j{<}0$. In the sequel, we frequently encounter i/o-relations which refer to negatively indexed elements of input sequences. Suitable extensions of these sequences will be chosen "on the fly" during the derivations.

Note that for PS'_N, $b(i)$ depends on $a((i{-}2N)\max(-1)..i]$, and therefore a simple communication behavior independent of N, like $(a;b)^*$, is possible for PS'_N.

Now we are faced with the problem of designing PS'_N. As a first step, we rewrite the right-hand side of (2) so as to eliminate div and mod:

$$\begin{array}{ll}
 & (\forall j : 0{\leq}j{<}2N : a(i+j-2N+1) = a(i+2(j \bmod N)+j \operatorname{div} N-2N+1)) \\
= & \{ \text{ domain split; properties of div and mod } \} \\
 & \quad (\forall j : 0 \leq j < N : a(i+j-2N+1) = a(i+2j-2N+1)) \\
 & \wedge\ (\forall j : N \leq j < 2N : a(i+j-2N+1) = a(i+2j-4N+2)) \\
= & \{ \text{ dummy substitution } j{:=}j{+}N \text{ in second conjunct; merging conjuncts } \} \\
 & (\forall j : 0 \leq j < N : \quad a(i+j-2N+1) = a(i+2j-2N+1) \\
 & \qquad\qquad\qquad\quad \wedge\, a(i+j-N+1) = a(i+2j-2N+2)) \\
= & \{ \text{ dummy substitution } j{:=}N{-}1{-}j \ \} \\
 & (\forall j : 0 \leq j < N : a(i-j-N) = a(i-2j-1) \ \wedge\ a(i-j) = a(i-2j)).
\end{array}$$

Hence, the i/o-relation for PS'_N may be written as

(3) $\quad b(i) \equiv (\forall j : 0 \leq j < N : a(i-j-N) = a(i-2j-1) \ \wedge\ a(i-j) = a(i-2j)).$

As a modest generalisation of (3) we propose to design components PS'_n, for $1{\leq}n{\leq}N$, satisfying

(4) $\quad b(i) \equiv (\forall j : 0 \leq j < n : a(i-j-N) = a(i-2j-1) \ \wedge\ a(i-j) = a(i-2j)),$

for $i{\geq}0$. In case $n{\neq}1 \wedge i{\geq}0$, however, we get stuck:

$$\begin{array}{ll}
 & b(i+1) \\
= & \{ \ (4) \ \} \\
 & (\forall j : 0{\leq}j{<}n : a(i+1-j-N){=}a(i+1-2j-1) \wedge a(i+1-j){=}a(i+1-2j)) \\
= & \{ \text{ split off } j = 0\text{; dummy substitution } j{:=}j{+}1 \ \} \\
 & \quad a(i+1-N) = a(i) \ \wedge\ a(i+1) = a(i+1) \\
 & \wedge\ (\forall j : 0{\leq}j{<}n{-}1 : a(i-j-N) = a(i-2j-2) \ \wedge\ a(i-j) = a(i-2j-1))
\end{array}$$

The latter conjunct differs from the output of component PS'_{n-1} (cf.(4)). We observe two ways to resolve this problem. One way is to take the following generalisation of (3) as new i/o-relation for PS'_n:

$$\begin{array}{rl}
b(i) \ \equiv \ (\forall j : 0 \leq j < n : & a(i-j-N) = a(i-2j-1-N+n) \\
 & \wedge\ a(i-j) = a(i-2j-N+n)).
\end{array}$$

The other way is to introduce components Q_n $(1 \leq n \leq N)$ satisfying

(5) $b(i) \equiv (\forall j : 0 \leq j < n : a(i-j-N) = c(i-2j-1) \ \wedge \ a(i-j) = c(i-2j))$,

which have c as additional input channel. It turns out that both approaches lead to the same program. Here, we follow the second approach.

By construction, PS'_N can then be build with a subcomponent p of type Q_N, taking $pa(i)=a(i)$, $pc(i)=a(i)$, and $b(i)=pb(i)$. The only task left is now to design components Q_n. We aim at $(a, c; b)^*$ as communication behavior for Q_n. Below both sequences a and c are extended "to the left" for components Q_n such that (5) is defined for all natural i.

First we consider $1<n \leq N$. For $i=0$ we derive:

$$\begin{array}{ll} & b(0) \\ = & \{ \ (5) \ \} \\ & (\forall j : 0 \leq j < n : a(-j-N) = c(-2j-1) \ \wedge \ a(-j) = c(-2j)) \\ = & \{ \text{ extend } a \text{ and } c \text{ as given by (6) } \} \\ & a(0) = c(0), \end{array}$$

with

(6) $(\forall i : i < 0 : a(i) = a(0) \ \wedge \ c(i) = c(0))$.

We justify this extension at the end of this section. For $i \geq 0$ we derive:

$$\begin{array}{ll} & b(i+1) \\ = & \{ \ (5) \ \} \\ & (\forall j : 0 \leq j < n : a(i+1-j-N) = c(i+1-2j-1) \wedge a(i+1-j) = c(i+1-2j)) \\ = & \{ \text{ split off } j=0\text{; dummy substitution } j:=j+1 \ \} \\ & \quad a(i+1-N) = c(i) \ \wedge \ a(i+1) = c(i+1) \\ & \wedge \ (\forall j : 0 \leq j < n-1 : a(i-j+N) = c(i-2j-2) \ \wedge \ a(i-j) = c(i-2j-1)) \\ = & \{ \text{ introduce subcomponent } q \text{ of type } Q_{n-1} \text{ with } q \cdot a(i) = a(i) \\ & \qquad \text{and } q \cdot c(i) = c(i-1)\text{, for all integer } i\text{; (5) } \} \\ & a(i+1-N) = c(i) \ \wedge \ a(i+1) = c(i+1) \ \wedge \ q \cdot b(i). \end{array}$$

Recall that the intended communication behavior for Q_n is $(a, c; b)^*$. Therefore, $c(i)$, $a(i+1)$, $c(i+1)$, and $q \cdot b(i)$ will be available for the computation of $b(i+1)$. In order to supply $a(i+1-N)$ to component Q_n we introduce an auxiliary input channel d, satisfying, for $i \geq 0$:

(7) $d(i) = a(i+1-N)$.

Then $q \cdot d(i) = a(i+1-N)$ as well, hence we have that $q \cdot d(i) = d(i)$. Component PS'_N has subcomponent p of type Q_N. On account of (7), PS'_N has to supply $a(i+1-N)$ to p along channel $p \cdot d$. For this purpose we equip Q_n with an additional output channel e and i/o-relation:

(8) $e(i) = a(i+1-n)$.

PS'_N can now feed $a(i+1-N)$ to subcomponent p by simply returning the values received along $p \cdot e$:

com PS'_N (**in** a:int, **out** b:bool) :
  sub p:Q_N **bus**
  $[a = p \cdot a, a = p \cdot c, p \cdot b = b]$
  |[**var** v:int
    $(p \cdot e?v \ ; \ p \cdot d!v)^*$
  ]|
moc .

For the newly introduced output e we have ($1{<}n{\leq}N$):

$$\begin{array}{cl} & e(0) \\ = & \{\ (8)\ \} \\ & a(1-n) \\ = & \{\ (6)\ \} \\ & a(0) \end{array}$$

For $i{\geq}0$ we have:

$$e(i+1) = q{\cdot}e(i).$$

The relations found for Q_n are summarized below, where we have singled out the case $i{=}0$ in the relation for $q{\cdot}c$ in order to use (6):

$$\begin{array}{lcl} q{\cdot}a(i) & = & a(i) \\ q{\cdot}c(0) & = & c(0) \\ q{\cdot}c(i+1) & = & c(i) \\ q{\cdot}d(i) & = & d(i) \\ b(0) & \equiv & a(0) = c(0) \\ b(i+1) & \equiv & d(i) = c(i)\ \wedge\ a(i+1) = c(i+1)\ \wedge\ q{\cdot}b(i) \\ e(0) & = & a(0) \\ e(i+1) & = & q{\cdot}e(i). \end{array}$$

For the outputs of Q_1 we have:

$$\begin{array}{cl} & b(i) \\ = & \{\ (5)\ \} \\ & (\forall j : 0 \leq j < 1 : a(i-j-N) = c(i-2j-1)\ \wedge\ a(i-j) = c(i-2j)) \\ = & \{\ \ \} \\ & a(i-N) = c(i-1)\ \wedge\ a(i) = c(i)\ , \end{array}$$

and, trivially, $e(i) = a(i)$. Distinguishing $i{=}0$ and $i{>}0$ in the relation for b, we thus find for Q_1:

$$\begin{array}{lcl} b(0) & \equiv & a(0) = c(0) \\ b(i+1) & \equiv & d(i) = c(i)\ \wedge\ a(i+1) = c(i+1) \\ e(i) & = & a(i). \end{array}$$

We next try to find a communication behavior for Q_n, conform to the above relations, that requires minimal buffering. Observe that the relation for $b(i{+}1)$ refers to both $c(i)$ and $c(i{+}1)$. This requires an extra variable to hold $c(i)$. Omitting details (see section 3 for an example of a construction of a suitable communication behavior), we obtain the following communication behavior for Q_n ($n \neq 1$):

(9) $\quad a, c\ ;\ q{\cdot}a, b, q{\cdot}c, e\ ;\ (a, q{\cdot}b, c, d, q{\cdot}e\ ;\ q{\cdot}a, b, q{\cdot}c, q{\cdot}d, e)^*,$

which requires no additional buffering. Notice the alternation of input and output actions.

The program for Q_n, $1{<}n{\leq}N$, now reads:

$$
\begin{array}{l}
\textbf{com}\ Q_n\ (\textbf{in}\ a,c,d\text{:int},\ \textbf{out}\ b\text{:bool};\ e\text{:int})\ :\\
\quad \textbf{sub}\ q{:}Q_{n-1}\ \textbf{bus}\\
\quad |[\textbf{var}\ z,x,xx,w,v\text{:int};\ y\text{:bool}\\
\qquad a?z, c?x\ ;\ q{\cdot}a!z, b!(z=x), q{\cdot}c!x, e!z\\
\qquad ;(xx := x\\
\qquad\ ;a?z, q{\cdot}b?y, c?x, d?w, q{\cdot}e?v\\
\qquad\ ;q{\cdot}a!z, b!(z=x \wedge w=xx \wedge y), q{\cdot}c!xx, q{\cdot}d!w, e!v\\
\qquad\ )^*\\
\quad]|\\
\textbf{moc}\ .
\end{array}
$$

For Q_1 we obtain:

$$
\begin{array}{l}
\textbf{com}\ Q_1\ (\textbf{in}\ a,c,d\text{:int},\ \textbf{out}\ b\text{:bool};\ e\text{:int})\ :\\
\quad |[\textbf{var}\ z,x,xx,w\text{:int};\\
\qquad a?z, c?x\ ;\ b!(z=x), e!z\\
\qquad ;(xx := x\\
\qquad\ ;a?z, c?x, d?w\\
\qquad\ ;b!(z=x \wedge w=xx), e!z\\
\qquad\ )^*\\
\quad]|\\
\textbf{moc}\ .
\end{array}
$$

For analyzing the speed of the computation we use sequence functions [6]. A possible sequence function for component Q_n, $1<n\leq N$, inferred from (9), is:

$$
\begin{array}{lcl}
\sigma_n(a,i) & = & \sigma_n(c,i)\ =\ 2i+N-n\\
\sigma_n(b,i) & = & \sigma_n(e,i)\ =\ \sigma_n(q{\cdot}a,i)\ =\ \sigma_n(q{\cdot}c,i)\ =\ 2i+N-n+1\\
\sigma_n(d,i) & = & \sigma_n(q{\cdot}b,i)\ =\ \sigma_n(q{\cdot}e,i)\ =\ 2i+N-n+2\\
\sigma_n(q{\cdot}d,i) & = & 2i+N-n+3.
\end{array}
$$

It follows that Q_n, PS'_N as well as PS_N all have constant response time and constant latency.

We thus obtained a simple and efficient parallel program for the recognition of perfect-shuffles of length N. An important design decision was the introduction of PS'_N. Since PS'_N is only used in PS_N, the first $2N-1$ outputs of PS'_N may be chosen arbitrarily. In our design PS'_N has a subcomponent of type Q_N whose first $2N-1$ outputs are fixed by (6), which, in effect, means that $b(i)$ equivales "$a(0)^{2N-1-i}a[0..i]$ is a perfect-shuffle", for $0\leq i<2N$. There are, of course, many alternatives for (6); for instance,

$$(\forall i : i<0 : a(i)=\surd\ \wedge\ c(i)=\surd),$$

where $\surd$ is an arbitrary integer value. We do not discuss these alternatives because the resulting programs differ only negligible.

3 Initial perfect-shuffles

In this section we design a component PS with i/o-relation

$$b(i) \equiv (\forall j : 0\leq j<2i : a(j)=a(2(j \bmod i)+j \operatorname{div} i)),$$

for $i\geq 0$. That is, we design a component that computes for each initial segment $a[0..2i)$ whether it is a perfect-shuffle. Since $b(i)$ depends on all elements of $a[0..2i)$, we aim at a communication behavior that agrees with $(a^2,b)^*$, for instance, $(a;b;a)^*$.

As in the previous section, our first step is to eliminate div and mod from the i/o-relation. To that end, we derive:

$$
\begin{aligned}
& (\forall j : 0 \le j < 2i : a(j) = a(2(j \bmod i) + j \operatorname{div} i)) \\
= \quad & \{ \text{ domain split; properties of div and mod } \} \\
& (\forall j : 0 \le j < i : a(j) = a(2j)) \ \wedge \ (\forall j : i \le j < 2i : a(j) = a(2(j-i)+1)) \\
= \quad & \{ \text{ dummy substitution } j{:=}j{+}i \text{ in second conjunct } \} \\
& (\forall j : 0 \le j < i : a(j) = a(2j)) \ \wedge \ (\forall j : 0 \le j < i : a(i+j) = a(2j+1)).
\end{aligned}
$$

Hence, the i/o-relation for PS may be written as

$$b(i) \equiv c(i) \ \wedge \ d(i),$$

where

$$
\begin{aligned}
c(i) \quad &\equiv \quad (\forall j : 0 \le j < i : a(j) = a(2j)), \text{ and} \\
d(i) \quad &\equiv \quad (\forall j : 0 \le j < i : a(i+j) = a(2j+1)).
\end{aligned}
$$

A straightforward calculation yields the following simple recurrence relation for sequence c:

$$
\begin{aligned}
& c(0) && \equiv \quad \text{true}, \\
(10) \quad & c(i+1) && \equiv \quad a(i) = a(2i) \ \wedge \ c(i).
\end{aligned}
$$

Recall that the intended communication behavior for PS is $(a^2, b)^*$. Therefore, PS will have $a(2i)$ at its disposal for the computation of $c(i{+}1)$, but $a(i)$ should be retrieved from a subcomponent. For this purpose we will use the subcomponent introduced for the computation of sequence d below.

For sequence d we do not have a simple recurrence relation. As in our solution for fixed-length perfect-shuffles, we have to introduce a subcomponent, p of type Q say, to cater for the computation of d. The occurrence of i in $a(i{+}j){=}a(2j{+}1)$ forces the introduction of an extra input channel, e say (cf. introduction of channel c on page 3). We thus obtain the following i/o-relation for Q as a generalisation of the definition of d:

$$f(i) \equiv (\forall j : 0 \le j < i : e(i+j) = a(2j+1)),$$

for $i{\ge}0$. It should be noted, however, that this i/o-relation refers only to odd-indexed elements of sequence a. Therefore, we rather design a component Q with inputs $a1$ and e, output f, and i/o-relation

$$(11) \quad f(i) \equiv (\forall j : 0 \le j < i : e(i+j) = a1(j)).$$

Then $d(i) \equiv p{\cdot}f(i)$, provided that $p{\cdot}e(i) = a(i)$ and $p{\cdot}a1(i) = a(2i{+}1)$. Since PS feeds sequence a into p along channel $p{\cdot}e$, we decide to equip component Q with an additional output g, such that

$$(12) \quad g(i) = e(i),$$

in order to supply $a(i)$ to PS for the computation of $c(i{+}1)$.

For Q we aim at a communication behavior that agrees with $(a1, e^2, f, g)^*$, since this matches the intended communication behavior $(a^2, b)^*$ for PS and the relations between PS and its subcomponent p. This concludes the specification of Q.

So, let us now try to design Q. Trivially:

$$f(0) \equiv \text{true},$$

and, for $i{\ge}0$:

$$
\begin{aligned}
& f(i+1) \\
= \quad & \{ \ (11) \ \} \\
& (\forall j : 0 \le j < i+1 : e(i+1+j) = a1(j))
\end{aligned}
$$

$$
\begin{array}{ll}
= & \{ \text{ split off } j = i \ \} \\
& e(2i+1) = a1(i) \ \wedge \ (\forall j : 0 \le j < i : e(i+1+j) = a1(j)) \\
= & \{ \text{ introduce subcomponent } q \text{ of type } Q \text{ with } q{\cdot}a1(i){=}a1(i) \text{ and } q{\cdot}e(i){=}e(i+1);\ (11) \ \} \\
& e(2i+1) = a1(i) \ \wedge \ q{\cdot}f(i).
\end{array}
$$

Hence, we are heading for a recursive component Q. For the other output of Q we have

$$g(0) = e(0),$$

and

$$
\begin{array}{ll}
& g(i+1) \\
= & \{ \ (12) \ \} \\
& e(i+1) \\
= & \{ \text{ see derivation above } \} \\
& q{\cdot}e(i) \\
= & \{ \ q \text{ is of type } Q;\ (12) \ \} \\
& q{\cdot}g(i).
\end{array}
$$

Gathering the above results, we obtain the following summary of relations for component Q:

$$
\begin{array}{lcl}
q{\cdot}a1(i) & = & a1(i) \\
q{\cdot}e(i) & = & e(i+1) \\
f(0) & \equiv & \text{true} \\
f(i+1) & \equiv & e(2i+1) = a1(i) \ \wedge \ q{\cdot}f(i) \\
g(0) & = & e(0) \\
g(i+1) & = & q{\cdot}g(i),
\end{array}
$$

where q is a subcomponent of type Q.

We now derive a suitable communication behavior for Q. We do so by ordering the communications of Q by means of a sequence function, σ say. Since component Q is recursive we have the following requirements for σ:

$$
\begin{array}{lcl}
\sigma(q^k{\cdot}a1, i) & = & Ai + B + kI \\
\sigma(q^k{\cdot}e, i) & = & Ci + D + kI \\
\sigma(q^k{\cdot}f, i) & = & Ei + F + kI \\
\sigma(q^k{\cdot}g, i) & = & Gi + H + kI,
\end{array}
$$

for all $i{\ge}0$ and $k{\ge}0$. That is, we aim at a linear sequence function for Q with A, C, E, G, and I positive naturals and B, D, F, and H naturals. To avoid the need for buffering we have the following additional requirements for σ which follow from the above relations for Q:

$$
\begin{array}{lclcl}
\sigma(q{\cdot}a1, i) & < & \sigma(a1, i+1) & & \\
\sigma(e, i+1) & < & \sigma(q{\cdot}e, i) & < & \sigma(e, i+2) \\
\sigma(e, 2i+1) & < & \sigma(f, i+1) & < & \sigma(e, 2i+2) \\
\sigma(a1, i) & < & \sigma(f, i+1) & < & \sigma(a1, i+1) \\
\sigma(q{\cdot}f, i) & < & \sigma(f, i+1) & & \\
\sigma(e, 0) & < & \sigma(g, 0) & < & \sigma(e, 1) \\
\sigma(q{\cdot}g, i) & < & \sigma(g, i+1) & , &
\end{array}
$$

for all $i{\ge}0$. By a straightforward calculation it follows that

$$\begin{array}{lllll} A & = & E & = & 2C \\ C & < & I & < & 2C \\ F & < & D & < & C+F \\ F & < & B & < & 2C+F \\ D & < & H & < & C+D \\ I & < & G & . & \end{array}$$

We choose A, C, E, and G as small as possible, because this minimizes the response time. We find A=4, C=2, E=4, and G=4, and further I=3. From the remaining constants F should be the smallest and is chosen to be 0 (hence, the first communication of Q occurs in time slot 0). Then it follows that B<4, D=1, and H=2. Since PS has to supply the odd-indexed elements of its input sequence a to its subcomponent along channel $p{\cdot}a1$, we decide to take B=3. The following communication behavior corresponds to the sequence function thus found:

$$CB_Q = f \ ; \ e \ ; \ g \ ; \ (a1, e, q{\cdot}f \ ; \ q{\cdot}e, f \ ; \ e, q{\cdot}g \ ; \ q{\cdot}a1, q{\cdot}e, g)^*.$$

We learned the above method for constructing a suitable sequence function from Lex Bijlsma.

Next we derive a communication behavior for PS, CB_{PS} say, that conforms to the relations found for PS:

$$\begin{array}{lll} p{\cdot}a1(i) & = & a(2i+1) \\ p{\cdot}e(i) & = & a(i) \\ c(0) & \equiv & \text{true} \\ c(i+1) & \equiv & p{\cdot}g(i) = a(2i) \ \wedge \ c(i) \\ b(i) & = & c(i) \ \wedge \ p{\cdot}f(i), \end{array}$$

with p a subcomponent of type Q. Since Q has external communication behavior $(f; e; g; a1, e)^*$, we have

$$CB_{PS}{\restriction}p{\cdot}\{a1, e, f, g\} = (p{\cdot}f \ ; \ p{\cdot}e \ ; \ p{\cdot}g \ ; \ p{\cdot}a1, p{\cdot}e)^*.$$

We propose as overall communication behavior

$$CB_{PS} = c \ ; \ (a, p{\cdot}f \ ; \ b, p{\cdot}e \ ; \ a, p{\cdot}g \ ; \ p{\cdot}a1, c, p{\cdot}e)^*,$$

in which c stands for the computation of elements of sequence c. Notice that inputs and outputs are grouped together, respectively. Integration of CB_{PS} and the relations found for PS yields the following program:

```
com PS (in a:int, out b:bool) :
  sub p : Q bus
  |[var z0, z1, u, t:int; x:bool;
      x := true
    ; ( a?z0, p·f?u
      ; b!(x ∧ u), p·e!z0
      ; a?z1, p·g?t
      ; p·a1!z1, x := (t=z0 ∧ x), p·e!z1
      )*
  ]|
moc .
```

The external communication behaviour is $(a; b; a)^*$. The sequence function found for Q is extended as follows to incorporate the events in the command of PS:

$$
\begin{aligned}
\sigma(a,i) &= 2i \\
\sigma(b,i) &= 4i+1 \\
\sigma(p{\cdot}a1,i) &= 4i+3 \\
\sigma(p{\cdot}e,i) &= 2i+1 \\
\sigma(p{\cdot}f,i) &= 4i \\
\sigma(p{\cdot}g,i) &= 4i+2.
\end{aligned}
$$

Since $\sigma(a, 2i{+}2) - \sigma(a, 2i) = 4$, our program has constant response time. From the i/o-relation of PS it follows that $b(i{+}1)$ depends on $a[0..2i{+}2)$. Hence, our program has also constant latency, for $\sigma(b, i{+}1) - \sigma(a, 2i{+}1) = 3$.

Finally, we obtain for Q:

```
com Q (in a1, e:int, out f:bool; g:int) :
  sub q : Q bus
  |[var z, v, t:int; u:bool;
       f!true  ;  e?v ; g!v
     ; ( a1?z, e?v, q·f?u
       ; q·e!v, f!(v=z ∧ u)
       ; e?v, q·g?t
       ; q·a1!z, q·e!v, g!t
       )*
  ]|
moc .
```

We remark that the decision to let PS feed only odd-indexed elements of a to Q is important because this enabled us to design Q without the need for buffering.

4 Concluding remarks

For a number of years the problems of palindrome recognition (e.g., [1],[2]) and square recognition (e.g., [3],[6]) have been around to serve as examples to explain techniques for designing (fine-grained) parallel programs (in particular, linear systolic arrays). Typical for the solutions to these problems is that at some stage in the design auxiliary channels are introduced which carry the values supplied to the program so as to take them to the right place at the right moment (e.g., channels c, d, and e in our solution for fixed-length perfect shuffles). Problems of this type are such instructive exercises because all attention may be focused on the question what auxiliary channels to introduce, while the computations to be performed by the cells play only a minor role. Therefore, they provide good stepping-stones towards more advanced problems whose solutions also require the computation of some auxiliary results (see, for example, rank order filtering [4]).

The fixed-length versions of the recognition problems can be viewed as instances of the general problem of recognizing *P-invariant segments*, where P is a given permutation on $[0..M)$, for fixed M ($M{\geq}0$). The general i/o-relation reads:

$$b(i) \equiv (\forall j : 0 \leq j < M : a(i+j) = a(i+P(j))),$$

for $i{\geq}0$. E.g., for palindrome recognition permutation P is given by $P(j){=}M{-}1{-}j$, for $0{\leq}j{<}M$.

In order to find out whether we could also deal with more difficult permutations, we concocted the problem of recognizing K-rotation invariant segments (see [5]), as a natural generalisation of the square recognition problem. It is shown in [5] that our design method leads relatively easy to a nice solution. The solution for fixed-length K-rotation invariant segments from [5] is however not very efficient, because there exists a solution with fewer channels and simpler cells. Anne

Kaldewaij pointed out to us that the fixed-length square problem can be solved more efficiently, if one starts with the following i/o-relation:

$$b(i) = (\# j : 0 \le j < N : a(i+j) = a(i+N+j)) = N,$$

or,

$$b(i) = (\# j : 0 \le j < N : a(i+j) \neq a(i+N+j)) = 0,$$

for $i \geq 0$. The same approach is applicable to the K-rotations problem. "Unfortunately", the problem is now not that difficult anymore. For this reason we conceived the problem solved in this note. Rewriting i/o-relation (1) in the above form doesn't make much sense.

In order to put an end to this train of examples, we should try to solve the general problem. We have already investigated some ideas w.r.t. a possible general approach and we hope to report on this in a future note. Here, we only remark that being P-invariant is the same as being P^{-1}-invariant, for

$$\begin{array}{ll} & (\forall j : 0 \le j < M : a(i+j) = a(i+P(j)) \\ = & \{ \text{ dummy substitution } j{:=}P^{-1}(j) \ \} \\ & (\forall j : 0 \le P^{-1}(j) < M : a(i+P^{-1}(j)) = a(i+P(P^{-1}(j))) \\ = & \{ \ \} \\ & (\forall j : 0 \le j < M : a(i+j) = a(i+P^{-1}(j)). \end{array}$$

For example, the inverse of the perfect-shuffle permutation ps is given by

$$ps^{-1}(j) = N(j \bmod 2) + j \text{ div } 2,$$

for $0{\le}j{<}2N$. Hence, we could also have used ps^{-1} instead of ps in (1). We leave it to the reader to verify that nevertheless i/o-relation (4) emerges from this alternative i/o-relation when one proceeds along the same lines as in the beginning of section 2. For the other examples the inverse permutations are not essentially different from the original permutations.

We end this note with an interesting observation. Comparison of ps and ps^{-1} reveals that ps changes into ps^{-1}, when 2 and N are swapped in its defining expression, and vice versa for ps^{-1}. This inspires us to consider the connection between permutations $P_{K,L}$ and their inverses on $[0..KL)$, where $P_{K,L}$ is defined, for $K{>}0 \wedge L{>}0$, by

$$P_{K,L}(j) = K(j \bmod L) + j \text{ div } L,$$

for j in $[0..KL)$. We find the following nice property:

$$P_{K,L}^{-1} = P_{L,K}.$$

Recognizing $P_{K,L}$-invariant segments (or, $P_{L,K}$-invariant segments, if you like) seems to be a worthy subject of another note in the series "Recognizing ...".

References

1. Rem, M.
   Trace Theory and Systolic Computations
   In: J.W. de Bakker et al. (eds.), *PARLE'87: Parallel Architectures and Languages Europe*, LNCS **258** (1987) 14–33.
2. Rem, M.
   Designing Parallel Processes
   Lecture notes, Eindhoven University of Technology, The Netherlands (1988).
3. Robert, Y., Tchuente, M.
   Réseaux Systoliques pour des Problèmes de Mots
   R.A.I.R.O Informatique Théorique/Theoretical Informatics **19**(2) (1985) 107–123.
4. Kaldewaij, A., Rem, M.
   A Derivation of a Systolic Rank Order Filter with Constant Response Time
   In: J.L.A. van de Snepscheut (ed.), *Mathematics of Program Construction*, LNCS **375** (1989) 281–296.
5. Katoen, J.P., Rem, M.
   Two Parallel Programs for Recognizing K-rotated Sequences
   In: *Massive Parallelism: Hardware, Programming and Applications*, Amalfi (Italy, 1989) (to appear).
6. Zwaan, G.
   Parallel Computations
   Ph.D. thesis, Eindhoven University of Technology, The Netherlands (1989).