Recognizing K-rotated Segments

J.P. Katoen M. Rem
Dept. of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

September 1989
revised February 1990

Abstract

This paper explains a calculational design technique for fine-grained parallel programs by means of an example. The example deals with the recognition of K-rotation invariant (or K-rotated) segments which is a generalisation of the well-known square recognition problem. Two parallel programs are derived. The first one determines for each segment of N successive inputs whether it is K-rotated, the latter determines whether the whole sequence thus far received is K-rotated.

1 Introduction

For sequential program design a calculational way of programming has been developed so as to construct a program and its correctness proof hand-in-hand [1]. According to this technique programs are derived from their formal specifications by manipulating formulae. Since at each step of the derivation the correctness is preserved the program is correct by design, and, consequently, a correctness proof afterwards becomes superfluous. By means of an example we illustrate that a rather similar approach is also very effective for the design of (fine-grained) parallel programs. The parallel programs (also called components) we design are regular networks of cells that communicate synchronously with each other by exchanging messages along directed channels. Communication can take place between neighbor cells only. One of the cells communicates with the environment of the network: this is the only cell involved in the external input and output of the program.

Briefly, the design technique proceeds as follows. First, a specification of the problem at hand is constructed which constitutes a relation between sequences of input and output (so-called i/o-relations). The role of i/o-relations is similar to that of invariants in sequential program design. The derivation proceeds by partitioning or manipulating the i/o-relation into simpler ones, for instance by splitting off one or more terms. A term may give rise to the introduction of a subcomponent, for example if it is an instance of the original i/o-relation. Eventually we end up with relations for the individual communications along the channels. These relations impose requirements upon the order in which

communications take place (e.g., a value may only be involved in the computation once it is received). This order is independent of the values communicated. The next objective is to construct a deadlock-free communication behavior, satisfying the prescribed order, in which each output value depends only on the valus received last, thus causing minimal buffering.

Although the design technique is calculational it is certainly not automatic: in the numerous design decisions we let performance considerations, such as space and speed requirements, be our compass. A cell receives values in variables. The total number of variables is a good measure for the space requirements of the network. In order to assess the speed of the computation we use sequence functions, which makes concepts as response time and latency explicit.

The example we consider deals with the recognition of K-rotation invariant segments. A segment $a[m..m+n)$, $0<m$ and $0<n$, of an infinite sequence a is called K-rotation invariant (or, K-rotated, for short), for $K\geq 0$, if

$$(1) \quad (\mathbf{A}j : 0 \leq j < n : a(m+j) = a(m + ((j+K) \ \mathbf{mod} \ n)) \ ) \quad ,$$

where $a(j)$ denotes the $(j+1)$-th element of sequence a. Observe that being K-rotated is equivalent to being $(K \ \mathbf{mod} \ n)$-rotated. Since each segment is also 0-rotated, we confine ourselves to $0<K<n$. First a program is designed that determines for each segment of N $(N\geq 1)$ successive inputs whether it is K-rotated. The resulting network consists of $N+1$ cells with 4 variables per cell. It has constant response time and constant latency. Subsequently, a parallel program is designed that determines whether the whole sequence thus far received is K-rotated. The solution is a linear network of $K+1$ cells with 4 variables per cell, having constant response time and constant latency.

The notation used has been adopted from [3].

2 Fixed-length K-rotated segments

2.1 Formal specification

We are to design a component, *KR* say, with (integer) input a and (boolean) output b that determines for each segment $a[i..i+N)$ whether it is K-rotated $(0<K<N)$. The i/o-relation reads

$$(2) \quad b(i) \equiv (\mathbf{A}j : 0 \leq j < N : a(i+j) = a(i + ((j+K) \ \mathbf{mod} \ N)) \ ) \quad ,$$

for $i\geq 0$. From (2) we infer that $b(i)$ depends on all elements of $a[i..i+N)$. This gives rise to a communication behavior in which $a(i+N-1)$ is received before $b(i)$ is produced, e.g. $a^N;(b;a)^*$. In order to obtain a communication behavior independent of N, we let $b(i)$ depend on $a(i-N..i]$ and change, for $i\geq N-1$, (2) into

$$(3) \quad b(i) \equiv (\mathbf{A}j : 0 \leq j < N : a(i+j-N+1) = a(i + ((j+K) \ \mathbf{mod} \ N) - N + 1) \ ) \quad .$$

The obvious way to start the derivation is to generalise (3) in some way, thus obtaining specifications of components KR_n, for $1{\leq}n{\leq}N$, with the intention that KR_n has KR_{n-1} as subcomponent ($n{\neq}1$). However, (3) is only defined for $i{\geq}N{-}1$ and from experience we know that this results in i/o-relations for KR_n which are defined on a domain dependent on n. To avoid this we extend relations like (3), mentally to all natural i by defining $a(j)$ for $j{<}0$. During the derivation a suitable extension is chosen, such that relatively simple relations result.

For the right-hand side of (3) we derive

$$
\begin{array}{ll}
 & (\mathbf{A}j : 0 \leq j < N : a(i+j-N+1) = a(i+((j+K)\ \mathbf{mod}\ N)-N+1)\ ) \\
= & \quad \{\,\text{dummy change } j := N{-}j{-}1\,\} \\
 & (\mathbf{A}j : 0 \leq j < N : a(i-j) = a(i+((N-1-j+K)\ \mathbf{mod}\ N)-N+1)\ ) \\
= & \quad \{\,\text{domain split } (0{<}K{<}N)\ ;\ \mathbf{mod}\ \text{calculus}\,\} \\
 & \quad (\mathbf{A}j : 0 \leq j < K : a(i-j) = a(i-j-N+K)) \\
 & \wedge\ (\mathbf{A}j : K \leq j < N : a(i-j) = a(i-j+K)) \\
= & \quad \{\,\text{dummy change } j := j{+}K \text{ in second conjunct ; let } L = N{-}K\,\} \\
 & \quad (\mathbf{A}j : 0 \leq j < K : a(i-j) = a(i-j-L)) \\
 & \wedge\ (\mathbf{A}j : 0 \leq j < L : a(i-j-K) = a(i-j))\quad .
\end{array}
$$

Due to the resemblance of the latter conjucts we suggest to introduce components, Q_M say, satisfying

$$c(i) \equiv (\mathbf{A}j : 0 \leq j < M : a(i-j) = a(i-j-N+M))\quad ,$$

for $i{\geq}0$. Component KR can then be build with a subcomponent p of type Q_K and a subcomponent q of type Q_L taking $p{\cdot}a(i) = a(i)$, $q{\cdot}a(i) = a(i)$, and $b(i) = p{\cdot}c(i) \wedge q{\cdot}c(i)$. The only task left now is to design component Q_M. As a natural generalisation of the relation for c we propose to design components Q_m, $1{\leq}m{\leq}M$, satisfying

$$(4)\quad c(i) \equiv (\mathbf{A}j : 0 \leq j < m : a(i-j) = a(i-j-\hat{N}))\quad ,$$

where $\hat{N} = N{+}M$. We aim at $(a;c)^*$ as (external) communication behavior. First we consider $1{<}m$. For $i{=}0$ we derive

$$
\begin{array}{ll}
 & c(0) \\
= & \quad \{\,(4)\,\} \\
 & (\mathbf{A}j : 0 \leq j < m : a(-j) = a(-j-\hat{N})) \\
= & \quad \{\,\text{extend } a \text{ according to (5), see below}\,\} \\
 & \text{true}\quad ,
\end{array}
$$

where

$$(5)\quad (\mathbf{A}i : i < 0 : a(i) = a(0))\quad .$$

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

$$
\begin{array}{ll}
 & c(i+1) \\
= & \quad \{\,(4)\,\} \\
 & (\mathbf{A}j : 0 \le j < m : a(i+1-j) = a(i+1-j-\hat{N})) \\
= & \quad \{\,\text{split off } j = 0\,\} \\
 & a(i+1) = a(i+1-\hat{N}) \wedge (\mathbf{A}j : 1 \le j < m : a(i+1-j) = a(i+1-j-\hat{N})) \\
= & \quad \{\,\text{dummy change } j := j+1\,\} \\
 & a(i+1) = a(i+1-\hat{N}) \wedge (\mathbf{A}j : 0 \le j < m-1 : a(i-j) = a(i-j-\hat{N})) \\
= & \quad \{\,\text{introduce subcomponent } r \text{ of type } Q_{m-1} \text{ with } r{\cdot}a(i){=}a(i)\ ;\ (4)\,\} \\
 & a(i+1) = a(i+1-\hat{N}) \wedge r{\cdot}c(i) \quad .
\end{array}
$$

Recall that the intended communication behavior is $(a;c)^*$. Hence, $a(i{+}1)$ and $r{\cdot}c(i)$ are available for the computation of $c(i{+}1)$. In order to make $a(i{+}1{-}\hat{N})$ available to Q_m variables could be introduced to record all elements of $a(i{-}\hat{N}..i{+}1]$. This would, however, make the components rather bulky. A more attractive solution is to supply the 'old' a-values via an auxiliary channel. To this end we introduce an input channel d satisfying

(6) $\quad d(i) = a(i+1-\hat{N}) \quad ,$

for $i{\ge}0$. Then $r{\cdot}d(i) = a(i{+}1{-}\hat{N})$ as well, hence we have $r{\cdot}d(i) = d(i)$. It should be noticed that we could also introduce a (boolean) input channel d satisfying $d(i) \equiv a(i{+}1){=}a(i{+}1{-}\hat{N})$, since $a(i{+}1{-}\hat{N})$ does not depend on m. This slight optimisation is not elaborated here, since the resulting programs differ only negligible.

Recall that subcomponents p and q are of type Q_K and Q_L respectively. Due to the introduction of the auxiliary input channel d component KR has to supply $a(i{+}1{-}L)$ and $a(i{+}1{-}K)$ to p resp. q. A convenient way to make for instance $a(i{+}1{-}L)$ available to p is to build a separate component of L cells. Notice that q consists of exactly L cells. This suggests to add an auxiliary output channel to q, or more generally, to components Q_m, $1{\le}m{\le}M$, satisfying

(7) $\quad e(i) = a(i+1-m) \quad ,$

for $i{\ge}0$. KR can now feed $a(i{+}1{-}L)$ to p by simply returning the values received along $q{\cdot}e$. Similarly it can supply $a(i{+}1{-}K)$ to q by returning the values received along $p{\cdot}e$. For the newly introduced output channel e we have $(1{<}m)$

$$
\begin{array}{ll}
 & e(0) \\
= & \quad \{\,(7)\,\} \\
 & a(1-m) \\
= & \quad \{\,(5)\,\} \\
 & a(0) \quad ,
\end{array}
$$

and for $i{\ge}0 : e(i{+}1) = r{\cdot}e(i)$.

2.2 The programs

Gathering the results, we have the following relations for Q_m

$$\begin{aligned} r{\cdot}a(i) &= a(i) \\ r{\cdot}d(i) &= d(i) \\ c(0) &\equiv \text{true} \\ c(i+1) &\equiv a(i+1) = d(i) \wedge r{\cdot}c(i) \\ e(0) &= a(0) \\ e(i+1) &= r{\cdot}e(i) \quad . \end{aligned}$$

We next try to find a communication behavior, conforming to the above relations, in which each output value depends only on the values received last, thus causing minimal buffering. In the sequel we always aim at minimal buffering when considering communication behaviors. To denote communication behaviors a notation like regular expressions is used. Here, ';' denotes sequential composition, '*' denotes repetition, and ',' denotes that two communications may take place in any order or even concurrently. The star binds strongest and the comma takes precedence over the semicolon. The relations above give rise to the following partial orders on the communications

$$\begin{aligned} CB(r{\cdot}a) &= (a \ ; \ r{\cdot}a)^* \\ CB(r{\cdot}d) &= (d \ ; \ r{\cdot}d)^* \\ CB(c) &= a, c \ ; \ (a, d, r{\cdot}c \ ; \ c)^* \\ CB(e) &= a \ ; \ (e \ ; \ r{\cdot}e)^*, a^* \quad , \end{aligned}$$

which can be combined into the overall communication behavior

$$(8) \quad CB = a \ ; \ c, e, r{\cdot}a \ ; \ (a, d, r{\cdot}c, r{\cdot}e \ ; \ c, e, r{\cdot}a, r{\cdot}d)^* \quad .$$

Notice the alternation of input and output actions. Since the external communication behavior of Q_m

$$CB\lceil\{a, c, d, e\} = a \ ; \ (c, e \ ; \ a, d)^* \quad ,$$

and the internal communication behavior of Q_{m+1}

$$CB\lceil\{r{\cdot}a, r{\cdot}c, r{\cdot}d, r{\cdot}e\} = r{\cdot}a \ ; \ (r{\cdot}c, r{\cdot}e \ ; \ r{\cdot}a, r{\cdot}d)^*$$

match, the program suffers from deadlock (according to [9]).

The program texts are obtained by integrating the communication behavior and the relations found. We use a CSP-like notation [2] for our program notation. This entails that communications are distributed assignments, i.e. for channel a from Q_m to Q_{m+1} and expression E the simultaneous execution of $a!E$ in Q_m and $a?x$ in Q_{m+1} establishes the assignment $x{:=}E$. The program for Q_m now reads ($1{<}m{\leq}M$)

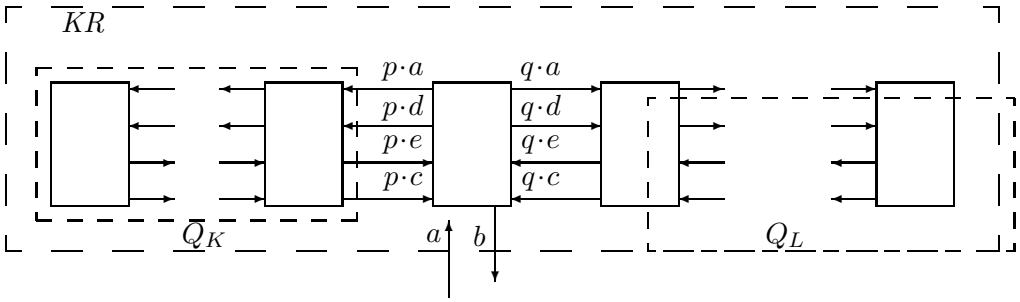

Figure 1: The program for fixed-length K-rotated segments

$$
\begin{array}{l}
[\![\ \textbf{var}\ va, vd, ve : \text{int}; vc : \text{bool};\\
\quad a?va\ ;\ c!\text{true}, e!va, r\cdot a!va\\
\quad ;(a?va, d?vd, r\cdot c?vc, r\cdot e?ve\\
\qquad ; c!(va{=}vd \wedge vc), e!ve, r\cdot a!va, r\cdot d!vd\\
\qquad)^*\\
]\!]\quad .
\end{array}
$$

Now we consider the design of Q_1. For the outputs of Q_1 we have

$$
\begin{array}{ll}
 & c(i)\\
= & \quad \{\ (4)\ \}\\
 & (\mathbf{A}j : 0 \le j < 1 : a(i-j) = a(i-j-\hat{N}))\\
= & \quad \{\ \}\\
 & a(i) = a(i-\hat{N})\quad ,
\end{array}
$$

and, trivially, $e(i) = a(i)$. Distinguishing $i{=}0$ and $i{>}0$ in the relation for c, and using the specification of d, we summarise

$$
\begin{array}{rcl}
c(0) & \equiv & \text{true}\\
c(i+1) & \equiv & a(i+1) = d(i)\\
e(i) & = & a(i)\quad .
\end{array}
$$

An appropriate communication behavior for Q_1 is $a;(c,e;a,d)^*$. The program thus obtained is

$$
\begin{array}{l}
[\![\ \textbf{var}\ va, vd : \text{int};\\
\quad a?va\ ;\ c!\text{true}, e!va\\
\quad ;(a?va, d?vd\ ;\ c!(va = vd), e!va)^*\\
]\!]\quad .
\end{array}
$$

The parallel program designed is illustrated in figure 1. For component KR we have the following relations

$$
p\cdot a(i) \quad = \quad a(i)
$$

$$
\begin{array}{rcl}
q{\cdot}a(i) & = & a(i) \\
b(i) & \equiv & p{\cdot}c(i) \wedge q{\cdot}c(i) \\
p{\cdot}d(i) & = & q{\cdot}e(i) \\
q{\cdot}d(i) & = & p{\cdot}e(i) \quad .
\end{array}
$$

Omitting details, we obtain the following communication behavior for KR

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

The external communication behavior of the program is $a;(a;b)^*$. Eventually the program reads

$$
\begin{array}{l}
|[\ \textbf{var} \ va, ve, vee : \mathrm{int}; vc, vcc : \mathrm{bool} \\
\quad a?va \ ; \ p{\cdot}a!va, q{\cdot}a!va \\
\quad ; (a?va, p{\cdot}c?vc, q{\cdot}c?vcc, p{\cdot}e?ve, q{\cdot}e?vee \\
\qquad ; b!(vc \wedge vcc), p{\cdot}a!va, q{\cdot}a!va, p{\cdot}d!vee, q{\cdot}d!ve \\
\qquad)^* \\
]| \quad .
\end{array}
$$

The recognition of K-rotated segments is a natural generalisation of the recognition of squares. A segment is called a square if it is the concatenation of two identical segments. Systolic arrays for the recognition of squares are presented in [5, 7]. For $N{=}2K$, the K-rotation recognition problem reduces to the square recognition problem. In this case $K{=}L$, and, consequently, p and q are identical. Hence, p or q may be eliminated from the program. As a nice result, the program thus obtained is equivalent to the program one would have obtained starting with the specification of the square recognition problem at once.

2.3 A short complexity analysis

The speed of the computation is analysed by means of sequence functions [9]. A sequence function exhibits a possible execution order by assigning all communications to time slots. For sequence function σ the natural number $\sigma(a,i)$ denotes the time slot to which the $(i{+}1)$-th communication along channel a can be assigned. Omitting communications along channels $q{\cdot}s$, where s is a dummy, since these can be scheduled simultaneously with those along $p{\cdot}s$, a possible sequence function for KR is inferred from (9)

$$
\begin{array}{rcl}
\sigma(a,i) & = & 2i \\
\sigma(p{\cdot}a,i) & = & 2i+1 \\
\sigma(p{\cdot}c,i) & = & \sigma(p{\cdot}e,i) \ = \ 2i+2 \\
\sigma(b,i) & = & \sigma(p{\cdot}d,i) \ = \ 2i+3 \quad .
\end{array}
$$

For the sake of brevity the sequence functions for components Q_m are omitted. Response time is a measure for the time elapsing between successive external communications. From

$$\sigma(b,i) - \sigma(a,i+1) = 1$$

we conclude that our program has constant response time. Whenever an output value is produced some time will have elapsed since the last input value on which it depends was received. This time period is called the latency. From (3) we infer that $b(i)$ depends on $a(i{-}N..i]$. The latency is, consequently,

$$\sigma(b,i) - \sigma(a,i) = 3 \quad .$$

3 Initial K-rotated segments

3.1 Specification and derivation

In this section we design a component, *Rot* say, with (integer) input channel a and (boolean) output channel b, satisfying

$$(10)\quad b(i) \equiv (\mathbf{A}j : 0 \leq j < i : a(j) = a((j+K) \textbf{ mod } i)) \quad ,$$

for $i{\geq}0$. That is, *Rot* determines for each segment $a[0..i)$ whether it is K-rotated. We aim at a communication behavior $(b;a)^*$ for *Rot*. For reasons of simplicity we confine ourselves to $i{\geq}K$. From (10) it immediately follows

$$b(K) \equiv \text{true} \quad ,$$

and for $i{\geq}K$ we derive

$$\begin{array}{ll}
 & b(i+1) \\
= & \quad \{\,(10)\,\} \\
 & (\mathbf{A}j : 0 \leq j < i+1 : a(j) = a((j+K) \textbf{ mod } (i+1)) \;) \\
= & \quad \{\,\text{domain split, using } (j{+}K) \textbf{ mod } (i{+}1){=}j{+}K{-}i{-}1, \\
 & \quad\;\; \text{for } j{+}K{\geq}i{+}1,\ K{<}i{+}1, \text{ and } j{<}i{+}1\,\} \\
 & \quad (\mathbf{A}j : 0 \leq j < i+1-K : a(j) = a(j+K)) \\
 & \wedge\ (\mathbf{A}j : i+1-K \leq j < i+1 : a(j) = a(j+K-i-1)) \\
= & \quad \{\,\text{dummy change } j{:=}j{+}i{+}1{-}K \text{ in second conjunct ; (11), see below}\,\} \\
 & c(i+1) \wedge (\mathbf{A}j : 0 \leq j < K : a(i+j+1-K) = a(j)) \\
= & \quad \{\,\text{split off } j = K{-}1 \text{ ; (12), see below}\,\} \\
 & c(i+1) \wedge a(i) = a(K-1) \wedge d(i) \quad ,
\end{array}$$

where, for $i{\geq}K$

$$(11)\quad c(i) \equiv (\mathbf{A}j : 0 \leq j < i-K : a(j) = a(j+K)) \quad \text{, and}$$

$$(12)\quad d(i) \equiv (\mathbf{A}j : 0 \leq j < K-1 : a(i+j+1-K) = a(j)) \quad .$$

A straightforward calculation yields the following simple recurrence relations for c:

$$\begin{array}{rcl}
c(K) & \equiv & \text{true} \\
c(i+1) & \equiv & a(i-K) = a(i) \wedge c(i) \quad , i \geq K
\end{array}$$

The intended communication behavior for *Rot* is $(b;a)^*$. Therefore, $a(i)$ and $c(i)$ will be available for the computation of $c(i{+}1)$, but $a(i{-}K)$ should be retrieved from a subcomponent. For this purpose we will use the subcomponent introduced for the computation of sequence d below. Since we do not have a simple recurrence relation for d, a subcomponent, p of type Q_{K-1} say, is introduced. As a modest generalisation of (12), the i/o-relation for components Q_k, $0{\leq}k{<}K$ is

(13) $d(i) \equiv (\mathbf{A}j : 0 \leq j < k : a(j) = a(i+j-k))$,

for $i{>}k$. We focus our attention on the design of p. For $k{=}0$ we simply have: $d(i) \equiv$ true. Consider $0{<}k{<}K$ and $i{\geq}k$. We derive

$$
\begin{array}{ll}
 & d(i+1) \\
= & \quad \{\, (13) \,\} \\
 & (\mathbf{A}j : 0 \leq j < k : a(j) = a(i+1+j-k)) \\
= & \quad \{\, \text{split off } j = k{-}1 \,\} \\
 & a(k-1) = a(i) \wedge (\mathbf{A}j : 0 \leq j < k-1 : a(j) = a(i+j-(k-1)) \ ) \\
= & \quad \{\, \text{introduce subcomponent } q \text{ of type } Q_{k-1} \text{ with } q{\cdot}a(i) = a(i) \ ;\ (13) \,\} \\
 & a(k-1) = a(i) \wedge q{\cdot}d(i) \quad .
\end{array}
$$

In order to supply $a(i-K)$ to component *Rot* we equip Q_k with an auxiliary output channel e, satisfying

(14) $e(i) = a(i-1-k)$,

for $i{>}k$. *Rot* then receives $a(i{-}K)$ to along channel $p{\cdot}e$. For output e we have for $k = 0$: $e(i+1) = a(i)$, and for $0{<}k{\leq}i$ we derive

$$
\begin{array}{ll}
 & e(i+1) \\
= & \quad \{\, (14) \,\} \\
 & a(i-k) \\
= & \quad \{\, (14),\ q \text{ is of type } Q_{k-1} \,\} \\
 & q{\cdot}e(i) \quad .
\end{array}
$$

3.2 The program

Summarising, we have the following relations for *Rot*

$$
\begin{array}{rcl}
b(i) & = & \text{`arbitrary'} \quad , i < K \\
b(K) & \equiv & \text{true} \\
b(i+1) & \equiv & c(i+1) \wedge a(i) = a(K-1) \wedge p{\cdot}d(i) \quad , i \geq K \\
c(K) & \equiv & \text{true} \\
c(i+1) & \equiv & a(i) = p{\cdot}e(i) \wedge c(i) \quad , i \geq K \\
p{\cdot}a(i) & = & a(i) \quad .
\end{array}
$$

A possible communication behavior that conforms to these relations is

(15) $b\ ;\ (a, p{\cdot}d, p{\cdot}e\ ;\ b, p{\cdot}a)^*$.

A sequence function σ for *Rot* is now given by

$$\begin{array}{rcl} \sigma(b,i) & = & 2i \\ \sigma(a,i) & = & \sigma(p{\cdot}d,i) \ = \ \sigma(p{\cdot}e,i) \ = \ 2i+1 \\ \sigma(p{\cdot}a,i) & = & 2i+2 \quad , \end{array}$$

and we conclude that our program has constant response time and constant latency. For Q_k $(0{<}k{<}K)$ the relations found are

$$\begin{array}{rcl} q{\cdot}a(i) & = & a(i) \\ d(i) & = & \text{'arbitrary'} \quad , i \leq k \\ d(i+1) & \equiv & a(k-1) = a(i) \wedge q{\cdot}d(i) \quad , i \geq k \\ e(i) & = & \text{'arbitrary'} \quad , i \leq k \\ e(i+1) & = & q{\cdot}e(i) \quad , i \geq k \quad . \end{array}$$

Omitting details an overall communication behavior is $d, e\ ;\ (a, q{\cdot}d, q{\cdot}e\ ;\ d, e, q{\cdot}a)^*$. It can easily be checked that we have absence of deadlock. A possible sequence function, σ_k, for Q_k is:

$$\begin{array}{rcl} \sigma_k(d,i) & = & \sigma_k(e,i) \ = \ 2i+K-k \\ \sigma_k(a,i) & = & \sigma_k(q{\cdot}d,i) \ = \ \sigma_k(q{\cdot}e,i) \ = \ 2i+1+K-k \\ \sigma_k(q{\cdot}a,i) & = & 2i+2+K-k \quad . \end{array}$$

Observe that the relation for $d(i{+}1)$ refers to $a(k{-}1)$, which requires an extra variable to hold $a(k{-}1)$. A more serious problem, however, is that $a(k{-}1)$ has to be selected from sequence a. This causes an initialisation dependent on k which makes each component Q_k specific. From the definition of σ_k it follows $\sigma_k(a,i) = \sigma_k(q{\cdot}d,i)$, in particular we have $\sigma_k(a,k{-}1) = \sigma_k(q{\cdot}d,k{-}1)$. Recall that output d is not specified for $i{\leq}k$. By choosing, for instance,

$$\begin{array}{rcl} d(k) & \equiv & \text{true} \\ d(i) & \equiv & \text{false} \quad , i < k \end{array}$$

the initialisation no longer depends on k: when input $q{\cdot}d$ is true for the first time, $a(k{-}1)$ is received in the same time slot and can be selected. Hence, a simple check of the data received along $q{\cdot}d$ suffices (see the programs below).

For the sake of completeness we present the programs of *Rot* and Q_k $(0{<}k{<}K)$. For *Rot* we obtain, choosing $b(i) \equiv \text{true}$ for $i{<}K$,

$$
\begin{array}{l}
[\![\ \textbf{var}\ x, va, ve : \mathrm{int}; c, vd : \mathrm{bool};\\
\quad b!\mathrm{true}\ ;\ vd, c := \mathrm{false}, \mathrm{true}\\
\quad ;\textbf{do}\ \neg vd \rightarrow (a?x, p{\cdot}d?vd, p{\cdot}e?ve\ ;\ b!\mathrm{true}, p{\cdot}a!x)\ \textbf{od}\\
\quad ;(a?va, p{\cdot}d?vd, p{\cdot}e?ve\\
\quad\ ;c := (va = ve) \wedge c\\
\quad\ ;b!(c \wedge va = x \wedge vd), p{\cdot}a!va\\
\quad\ )^*\\
]\!]\quad .
\end{array}
$$

For Q_k $(0{<}k{<}K)$ the program reads as follows

$$
\begin{array}{l}
[\![\ \textbf{var}\ x, va, ve : \mathrm{int}; vd : \mathrm{bool};\\
\quad d!\mathrm{false}, e!\text{`arbitrary'}\ ;\ vd := \mathrm{false}\\
\quad ;\textbf{do}\ \neg vd \rightarrow (a?x, q{\cdot}d?vd, q{\cdot}e?ve\\
\qquad\qquad\qquad ;d!vd, e!ve, q{\cdot}a!x)\\
\quad \textbf{od}\\
\quad ;(a?va, q{\cdot}d?vd, q{\cdot}e?ve\\
\quad\ ;q{\cdot}a!va, d!(va = x \wedge vd), e!ve\\
\quad\ )^*\\
]\!]
\end{array}
$$

The relations found for Q_0, using $d(0) \equiv \mathrm{true}$, are

$$
\begin{array}{rcl}
d(i) & \equiv & \mathrm{true}\\
e(0) & = & \text{`arbitrary'}\\
e(i+1) & = & a(i)\quad ,\ i \geq 0\quad .
\end{array}
$$

Its communication behavior is $(d, e\ ;\ a)^*$. The program is

$$
\begin{array}{l}
[\![\ \textbf{var}\ va : integer;\\
\quad (d!\mathrm{true}, e!va\ ;\ a?va)^*\\
]\!]\quad .
\end{array}
$$

4 Conclusions

In this note we have derived two parallel programs for recognizing K-rotated segments in a rather calculational way. We would not have been able to design these programs by operational reasoning. The derivations are easy to understand and form an instructive example of the design method of parallel programs as presented in [4, 6]. More advanced problems are treated in, for instance [6, 8].

The choice to define $a(j){=}a(0)$ for $j{<}0$ was not made a priori, but throughout the derivation. This choice greatly simplified the derivations in section 2.

An important observation, due to Wim Kloosterhuis, is to exploit unspecified communications so as to eliminate a component specific initialisation. Similar observations have

been made in [8], where unspecified communications were cleverly used in order to derive the program presented in that paper.

Acknowledgements
Thanks are due to the Eindhoven VLSI club for their suggestions on the presentation. We would like to thank Wim Kloosterhuis for commenting on a draft version of this note. Thanks are also due to Anne Kaldewaij for suggesting the problem and to Berry Schoenmakers for explanatory discussions and for his helpful comments on a draft version.

References

[1] E.W. Dijkstra and W.H.J. Feijen, *A Method of Programming* (Addison-Wesley, Reading, Massachusetts, 1988).

[2] C.A.R. Hoare, *Communicating Sequential Processes*, Communications of the ACM **21** (1978), pp.666-677.

[3] M. Rem, *Trace Theory and Systolic Computations*, in proceedings PARLE'87 : Parallel Architectures and Languages Europe, (J.W. de Bakker et al., eds.), Lecture Notes in Computer Science 258, Springer-Verlag, Berlin, 1987, pp.14-33.

[4] M. Rem, *Designing Parallel Processes*, Lecture notes, Eindhoven University of Technology, The Netherlands (1988).

[5] Y. Robert and M. Tchuente, *Réseaux Systoliques pour des Problèmes de Mots*, R.A.I.R.O Informatique Théorique/Theoretical Informatics **19**(2) (1985), pp.107-123.

[6] A. Kaldewaij and M. Rem, *A Derivation of a Systolic Rank Order Filter with Constant Response Time*, in proceedings : Mathematics of Program Construction, (J.L.A. van de Snepscheut ed.), Lecture Notes in Computer Science 375, Springer-Verlag, Berlin, 1989, pp.281-296.

[7] J.L.A. van de Snepscheut and J.B. Swenker, *On the Design of Some Systolic Algorithms*, Journal of the ACM **36**(4) (1989), pp.826-840.

[8] P. Struik, *A Systematic Design of a Parallel Program for Dirichlet Convolution*, Computing Science Note 89/7, Eindhoven University of Technology, The Netherlands (1989).

[9] G. Zwaan, *Parallel Computations*, Ph.D.-thesis, Eindhoven University of Technology, The Netherlands (1989).