Parallel programs for the recognition of P-invariant segments

Parallel Programs for the Recognition of P-invariant Segments

by

J.P. Katoen and L.A.M. Schoenmakers

91/03

April, 1991

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author or the editor.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

Parallel Programs for the Recognition of P-invariant Segments

J.P. Katoen
Philips Research Laboratories
P.O. Box 80.000, 5600 JA Eindhoven, The Netherlands

L.A.M. Schoenmakers
Eindhoven University of Technology
Dept. of Mathematics and Computing Science
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

March 12, 1991

Abstract

Let P be an arbitrary, but fixed permutation on $[0..N)$, with $N \geq 0$. A so-called recognition program determines for each segment of length N of its input sequence whether it is invariant under P. In this paper we design several parallel recognition programs. All these programs consist of a linear arrangement of cells and have constant response time. The major difference between these programs is in the size of the cells. A distinctive feature of the space efficient solutions is that they contain —in addition to the links between neighbour cells— links between non-neighbour cells. For some well-known instances of the general problem, such as palindrome recognition and square recognition, these space efficient solutions are systolic and comparable to the conventional ones; this is illustrated for the square recognition problem. Depending on P, however, the space efficient solutions may in general be non-systolic.

0 Introduction

In a series of papers design techniques for fine-grained parallel programs (in particular, linear systolic arrays) have been demonstrated by solving instances of the following general recognition problem: given a permutation P on $[0..N)$, with $N \geq 0$, the problem is to design a parallel program that determines for each segment of N successive elements of its input sequence whether it is invariant under P ("P-invariant" for short). That is, we have to design a parallel program with input channel a of arbitrary type T, output channel b of type Bool, satisfying the following i/o-relation:

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

for $i \geq 0$, where $a(i)$ and $b(i)$ denote the $(i+1)$-st elements of sequences a and b, respectively.

Simple instances of this problem are the recognition of

(i) palindromes [5, 6, 7]: $P(j) = N - 1 - j$,

and,

(ii) squares (or carrés) [5, 6]: $P(j) = (j + K) \bmod N,\ N = 2K$.
Square recognition may be generalized into

(iii) K-rotations [2]: $P(j) = (j + K) \bmod N,\ 0 \le K < N$.
Finally, we mention

(iv) perfect-shuffles [3]: $P(j) = 2(j \bmod K) + j \operatorname{div} K,\ N = 2K$,
and its generalization, which is mentioned but not solved in [3]:

(v) KL-shuffles: $P(j) = K(j \bmod L) + j \operatorname{div} L,\ N = KL$.

In Section 2 we solve the general problem as specified by (0) in a rather unusual, but systematic way. In order to resume the conventional design technique advocated in [1, 4] we first solve problem (ii) in Section 1. (Readers familiar with this technique may skip this section.) In Section 3 this conventional solution is compared to the solution obtained by instantiating the program derived in Section 2. It turns out that the latter program can be transformed into the conventional one. The same applies to problems (i) and (iii). However, the programs obtained for (iv) and (v) are not systolic because the number of output channels of some cells is proportional to N. This is shown in Section 4. Finally, in Section 5 some distinctive features of our approach are summarized.

1 Recognition of squares

In this section we briefly sketch the derivation of a parallel program (or component) C_K, $K \geq 0$, that recognizes squares. We apply the design technique explained in [1, 4]. The i/o-relation of C_K reads (cf. (ii) and (0)):

$$b(i) \equiv (\forall j : 0 \le j < 2K : a(i + j) = a(i + (j + K) \bmod 2K)) \quad ,$$

for $i \geq 0$. So, this component determines for each segment $a[i..i+2K)$ whether it is a square. This fact is more simply expressed by the following equivalent i/o-relation (cf. [7, Section 5.2]):

$$(1) \quad b(i) \equiv (\forall j : 0 \le j < K : a(i + j) = a(i + j + K)) \quad .$$

It follows from this i/o-relation that $b(i)$ depends on all elements of $a[i..i+2K)$, and, consequently, that (1) requires a communication behaviour like $a^{2K}; (b; a)^*$.

The obvious way to start the derivation is to generalize (1) in some way, thereby obtaining specifications of components C_k, $0 \leq k \leq K$, with the intention that C_k has C_{k-1} as subcomponent ($k \neq 0$). From experience, however, we know that such components must have a communication behaviour that depends on k (e.g., $a^{2k}; (b; a)^*$), and, consequently, that such components do not have identical commands. Moreover, a communication behaviour like $a^{2k}; (b; a)^*$ requires component C_k to detect that the $2k$-th communication along a has occurred. This dependence on k makes the components unnecessary complicated.

To obtain a simpler communication behaviour we therefore design a slightly different component C'_K with i/o-relation

$$(2) \quad b(i) \equiv (\forall j : 0 \le j < K : a(i + j - 2K) = a(i + j - K)) \quad ,$$

for $i \geq 2K$. This component, then, determines for each segment $a[i-2K..i)$ with $i \geq 2K$ whether it is a square. Hence, we may take $(b; a)^*$ as communication behaviour and C'_K may be used to build component C_K as follows:

com C_K (**in** a:T, **out** b:Bool) :
  sub $p{:}C'_K$
  |[**var** x:T; w:Bool;
    $(a?x, p{\cdot}b?w;\ p{\cdot}a!x)^{2K}$
    $;(a?x, p{\cdot}b?w;\ p{\cdot}a!x, b!w)^*$
  ]|
moc .

So, the first $2K$ outputs of subcomponent p are neglected by C_K. The external communication behaviour of C_K is $a^{2K};(a;b)^*$.[0]

The program notation used in this paper resembles the notation used in [7]. For the above program the following explanation is in order. After the name, C_K, of the component the names and types of (external) input and output channels are listed. p is a subcomponent of C_K of type C'_K. The block, delineated by |[and]|, consists of a declaration part and a command. Commands are denoted in a CSP-like notation [0]. This entails that for channel a directed from component p to q and expression E the simultaneous execution of $a!E$ in p and $a?x$ in q establishes the assignment $x{:=}E$ in q. The comma indicates arbitrary interleaving of the communications connected by it; it takes precedence over the semicolon.

Now we are left with the problem of designing C'_K. As an appropriate generalization of (2) we design components C'_k, $0{\le}k{\le}K$, satisfying

$$(3) \quad b(i) \equiv (\forall j : 0 \le j < k : a(i+j-K-k) = a(i+j-k)) \quad ,$$

for $i{\ge}K{+}k$. Then, however, values in $b[0..K{+}k)$ are not specified. In order that (3) defines $b(i)$ for all natural i, sequence a will be extended by defining $a(j)$ for $j{<}0$; a suitable extension of a is chosen such that relatively simple relations result.

The derivation proceeds by partitioning i/o-relation (3) into simpler ones until we end up with recurrence relations for the individual communications along the channels. As communication behaviour we take $(b;a)^*$. For C'_0 we have $b(i) \equiv$ true for all natural i, so we proceed with the case $1{\le}k{\le}K$. From (3) it immediately follows that $b(0) \equiv$ true provided that we extend sequence a (mentally) such that $a(j){=}\surd$ for $j{<}0$, where $\surd$ is an arbitrary value of type T. For $i{\ge}0$ we derive

$$\begin{aligned}
& b(i+1) \\
\equiv\ & \{\ (3)\ \} \\
& (\forall j : 0 \le j < k : a(i+1+j-K-k) = a(i+1+j-k)) \\
\equiv\ & \{\ \text{split off } j = k{-}1\ \} \\
& a(i-K) = a(i)\ \wedge\ (\forall j : 0 \le j < k-1 : a(i+j-K-(k-1)) = a(i+j-(k-1))) \\
\equiv\ & \{\ \text{introduce subcomponent } p \text{ of type } C'_{k-1} \text{ with } p{\cdot}a(i) = a(i), \text{ for } i{\ge}0;\ (3)\ \} \\
& a(i) = a(i-K)\ \wedge\ p{\cdot}b(i) \quad .
\end{aligned}$$

The desired communication behaviour of C'_k is $(b;a)^*$. Hence, the following internal communication behaviour for C'_k is possible: $b\,;(a, p{\cdot}b;\ p{\cdot}a, b)^*$. Using this communication

[0] By using a direct connection between channels a and $p{\cdot}a$ this can be improved slightly to $a^{2K};(b;a)^*$. Such a connection is sometimes represented by an equality $a = p{\cdot}a$ (see [7, Section 1.4]).

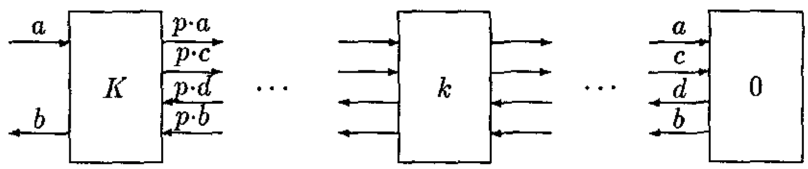

Figure 0: Conventional network for square recognition ($0<k<K$).

behaviour, C'_k has $a(i)$ and $p{\cdot}b(i)$ at its disposal for the computation of $b(i+1)$. To provide $a(i-K)$ we have several options. A simple solution is to buffer the last K values received along a in each component, but this solution is rejected because it makes the components too bulky. To avoid this buffering, the conventional solution in this case is to equip each component C'_k, except for C'_K, with an extra input channel c, say, satisfying

(4) $\quad c(i) = a(i-K) \quad ,$

for $i\geq 0$. The appropriate communication behaviour now is $(b;a,c)^*$. Component C'_k $(1\leq k<K)$ must supply $p{\cdot}c(i) = p{\cdot}a(i-K)$ to its subcomponent p; since $p{\cdot}a(i-K) = a(i-K)$, this boils down to $p{\cdot}c(i) = c(i)$. To let C'_K supply $a(i-K)$ to its subcomponent, components C'_k, $0\leq k<K$, get an extra output channel d satisfying

(5) $\quad d(i) = a(i-(k+1)) \quad ,$

for $i\geq 0$. We then have for C'_K:

$$
\begin{array}{ll}
 & p{\cdot}c(i) \\
= & \{\ (4)\ \} \\
 & p{\cdot}a(i-K) \\
= & \{\ p{\cdot}a(i)=a(i)\ \} \\
 & a(i-((K-1)+1)) \\
= & \{\ p \text{ is of type } C'_{K-1};\ (5)\ \} \\
 & p{\cdot}d(i) \quad ,
\end{array}
$$

so C'_K supplies $a(i-K)$ to its subcomponent by simply returning the values received along $p{\cdot}d$. Components C'_k $(0<k<K)$ determine output d as follows: $d(0) = \surd$ and $d(i+1) = p{\cdot}d(i)$. C'_0 does it differently: $d(0) = \surd$ and $d(i+1) = a(i)$. The structure of the network is now as depicted in Figure 0. The programs become:

com C'_0 (**in** a, c:T, **out** b:Bool, d:T) :
  |[**var** x, y:T;
    $b!\text{true}, d!\surd$
    $;(a?x, c?y; b!\text{true}, d!x)^*$
  ]|
moc ,

and for $0<k<K$:

```
com C'_k (in a,c:T, out b:Bool,d:T) :
   sub p:C'_{k-1}
   |[var x,y,z:T; w:Bool;
       b!true, d!√
       ;(a?x, p·b?w, c?y, p·d?z
        ;p·a!x, b!(x=y ∧ w), p·c!y, d!z
        )*
   ]|
moc ,
```

and finally:

```
com C'_K (in a:T, out b:Bool) :
   sub p:C'_{K-1}
   |[var x,y:T; w:Bool;
       b!true
       ;(a?x, p·b?w, p·d?y
        ;p·a!x, b!(x=y ∧ w), p·c!y
        )*
   ]|
moc .
```

This completes a quite conventional derivation.

The speed of a computation is analyzed by means of sequence functions (see [7, Section 2.5]) which exhibit a possible execution order by assigning all communications to time slots. For sequence function σ_k, natural $\sigma_k(a,i)$ denotes the time slot to which the $(i+1)$-st communication along channel a of component C'_k is assigned. For channels a and b we have for instance the following sequence functions:

$$\begin{aligned}\sigma_k(a,i) &= 2i+1+K-k\\ \sigma_k(b,i) &= 2i+K-k \quad ,\end{aligned}$$

for $0 \le k \le K$. Since $\sigma_k(b,i)$ is a linear function of i, we say that C_K has constant response time. Furthermore, the latency is the period of time which elapses between the production of an output value and the receipt of the last input value on which it depends. In our program $b(i)$ depends on $a[i-2K..i)$, and therefore it follows from the above sequence functions that it has constant latency.

2 Recognition of P-invariant segments

We now derive a parallel program P_N, say, satisfying (0). As in the conventional approach our first step is to introduce a component P'_N with i/o-relation

(6) $\quad b(i) \equiv (\forall j : 0 \le j < N : a(i+j-N) = a(i+P(j)-N)) \quad ,$

for $i \ge N$. So, P'_N determines for each segment $a[i-N..i)$ with $i \ge N$ whether it is P-invariant. We take $(b;a)^*$ as communication behaviour for P'_N, and by neglecting the first N outputs of P'_N we obtain a program for P_N.

To obtain a program for P'_N we try to design components P'_n, $0 \le n \le N$, satisfying (cf. generalization (3) of (2)):

$$(7)\quad b(i) \equiv (\forall j : 0 \leq j < n : a(i+j-n) = a(i+P(j)-n)) \quad ,$$

for $i \geq n$, and with $(b;a)^*$ as communication behaviour. By extending sequence a with negatively indexed elements, $b(i)$ will be defined for all natural i.

From (7) it immediately follows that $b(i) \equiv$ true for P'_0. For $n>0$ we have $b(0) \equiv$ true provided that we extend a such that $a(j) = \surd$ for $j<0$ —as in the previous section. For $i \geq 0$ we derive:

$$\begin{array}{ll}
 & b(i+1) \\
\equiv & \{ (7) \} \\
 & (\forall j : 0 \leq j < n : a(i+1+j-n) = a(i+1+P(j)-n)) \\
\equiv & \{ \text{split off } j = n{-}1 \} \\
 & a(i) = a(i+P(n{-}1)-(n{-}1)) \\
 & \quad \wedge\ (\forall j : 0 \leq j < n{-}1 : a(i+j-(n{-}1)) = a(i+P(j)-(n{-}1))) \\
\equiv & \{ \text{introduce subcomponent } p : P'_{n-1} \text{ with } p{\cdot}a(i) = a(i), \text{ for } i \geq 0;\ (7) \} \\
 & a(i) = a(i+P(n{-}1)-(n{-}1)) \ \wedge\ p{\cdot}b(i) \quad .
\end{array}$$

Now recall that the intended communication behaviour is $(b;a)^*$, hence $a(i)$ will be available for the computation of $b(i{+}1)$ but the whereabouts of $a(i+P(n{-}1)-(n{-}1))$ are unclear. It is even possible that this value has not yet been received by P'_n, namely in case $P(n{-}1) > n{-}1$.

Fortunately, the following observation helps us out. The right-hand side of (0) can be rewritten as follows:

$$\begin{array}{ll}
 & (\forall j : 0 \leq j < N : a(i+j) = a(i+P(j))) \\
\equiv & \{ \text{domain split} \} \\
 & (\forall j : 0 \leq j < N \ \wedge\ P(j) > j : a(i+j) = a(i+P(j))) \\
 & \quad \wedge\ (\forall j : 0 \leq j < N \ \wedge\ P(j) = j : a(i+j) = a(i+P(j))) \\
 & \quad \wedge\ (\forall j : 0 \leq j < N \ \wedge\ P(j) < j : a(i+j) = a(i+P(j))) \\
\equiv & \{ \text{dummy change } j := P^{-1}(j) \text{ in first conjunct; } P(P^{-1}(j)) = j \} \\
 & (\forall j : 0 \leq P^{-1}(j) < N \ \wedge\ j > P^{-1}(j) : a(i+P^{-1}(j)) = a(i+j)) \\
 & \quad \wedge\ (\forall j : 0 \leq j < N \ \wedge\ P(j) < j : a(i+j) = a(i+P(j))) \\
\equiv & \{ P^{-1} \text{ is a permutation on } [0..N) \} \\
 & (\forall j : 0 \leq j < N \ \wedge\ P^{-1}(j) < j : a(i+j) = a(i+P^{-1}(j))) \\
 & \quad \wedge\ (\forall j : 0 \leq j < N \ \wedge\ P(j) < j : a(i+j) = a(i+P(j))) \quad .
\end{array}$$

So, the original problem may be solved by solving two identical —but simpler— problems: because P^{-1} is as arbitrary as P, it suffices to design components P''_n, $0 \leq n \leq N$, satisfying:

$$(8)\quad b(i) \equiv (\forall j : 0 \leq j < n \ \wedge\ P(j) < j : a(i+j-n) = a(i+P(j)-n)) \quad ,$$

for $i \geq 0$, and with $(b;a)^*$ as communication behaviour.

Proceeding as above, we then obtain the following relations for P''_n $(n \neq 0)$:

$$\begin{array}{lll}
p{\cdot}a(i) & = & a(i) \\
b(0) & \equiv & \text{true} \\
b(i+1) & \equiv & (F_n \Rightarrow a(i) = a(i+P(n{-}1)-(n{-}1))) \ \wedge\ p{\cdot}b(i) \quad ,
\end{array}$$

where F_n abbreviates $P(n-1) < n-1$. Note that $a(i + P(n-1) - (n-1))$ is required for the computation of $b(i+1)$ only if F_n holds, which ensures that this value has already been received and has been passed on to subcomponent p via $p{\cdot}a$.

2.0 Conventional solution

From the relations above it follows that component P''_n ("cell n" for short) needs two a-values to compute $b(i+1)$ when F_n holds. With $b;(a, p{\cdot}b; p{\cdot}a, b)^*$ as communication behaviour, $a(i)$ and $p{\cdot}b(i)$ are available. In order to retrieve $a(i+P(n-1)-(n-1))$ the conventional approach is to introduce auxiliary channels between neighbouring cells. Since $P(n-1)-(n-1) < 0$, a first guess is to equip components P''_n with an extra output channel c such that $c(i) = a(i+P(n)-n)$ in case $P(n) < n$. We would then have

$$b(i + 1) \equiv (F_n \Rightarrow a(i) = p{\cdot}c(i)) \ \wedge \ p{\cdot}b(i) \quad .$$

Unfortunately, it is impossible to compute $c(i)$ from $p{\cdot}c(i)$ in this way, since we do not have a relation between $P(n)$ and $P(n-1)$. The fact that we are dealing with an arbitrary permutation P forces us to introduce an *array* of output channels C. An appropriate i/o-relation for this array of channels is given by:

$$C[m](i) = \begin{cases} a(i + P(m) - n) & , P(m) < n \\ \text{“don't care”} & , P(m) \geq n \end{cases} \quad ,$$

for $0 \leq m < N$. Then we may take $C[m](i) = \surd$ for P''_0, and for $0 < n \leq N$ we take $C[m](0) = \surd$ and

$$C[m](i+1) = \begin{cases} a(i) & , P(m) = n-1 \\ p{\cdot}C[m](i) & , P(m) \neq n-1 \end{cases} \quad ,$$

for $i \geq 0$, or, equivalently:

$$C[m](i+1) = \begin{cases} a(i) & , m = P^{-1}(n-1) \\ p{\cdot}C[m](i) & , m \neq P^{-1}(n-1) \end{cases} \quad .$$

It is interesting to note that P''_N's output channel C satisfies $C[m](i) = a(i+P(m)-N)$ for $0 \leq m < N$, hence $C(i)$ is a permutation of $a[i-N..i)$. Component P''_n can now compute $b(i+1)$ as follows:

$$b(i + 1) \equiv (F_n \Rightarrow a(i) = p{\cdot}C[n - 1](i)) \ \wedge \ p{\cdot}b(i) \quad .$$

The computation of $C(i+1)$ within a cell takes $\mathcal{O}(N)$ time when done sequentially. It is however trivial to do this in parallel to achieve $\mathcal{O}(1)$ time. The problem with this "conventional" solution is that it is very expensive, even more when one realizes that we have to do all of the above for P^{-1} as well. To summarize: we have obtained a program with constant response time and constant latency at the expense of an area quadratic in N (N cells consisting of N cells each).

2.1 Alternative solution

A key step in the conventional approach is that we try to retrieve value $a(i + P(n-1) - (n-1))$ from cell $n-1$ (component P''_{n-1}), a value which has been passed on to subcomponents in the mean time. Depending on $n-1 - P(n-1)$, this value has reached some cell k, say, with $k<n$, in the time slot when it is needed by cell n. Our idea now is to link cell n ($n>0$) with the appropriate cell k, such that value $a(i + P(n-1) - (n-1))$ can be supplied to n by k at the right moment, thereby avoiding the need for buffers in both cells. More precisely, we add an auxiliary channel directed from cell k to cell n, called c in cell k and $q{\cdot}c$ in cell n —accordingly, q will be used as local name for cell k in cell n— and we determine k such that channel $q{\cdot}c$ satisfies

(9) $\quad q{\cdot}c(i) = a(i + P(n-1) - (n-1)) \quad ,$

for $i \geq 0$. For cell n ($n>0$) we then have

$$\begin{array}{lcl} p{\cdot}a(i) & = & a(i) \\ c(i) & = & a(i) \\ b(0) & \equiv & \text{true} \\ b(i+1) & \equiv & (F_n \Rightarrow a(i) = q{\cdot}c(i)) \ \wedge \ p{\cdot}b(i) \quad . \end{array}$$

A possible communication behaviour that is consistent with the partial order that these relations impose is

(10) $\quad b\,; (a, p{\cdot}b, q{\cdot}c\,; p{\cdot}a, b, c)^* \quad .$

Unfortunately, this behaviour causes deadlock (cf. [7]): since cells n and k may be arbitrarily far apart, cell k will initially not be ready to participate in a communication along c. As a solution to this problem we alter the communication behaviour of odd numbered cells so as to activate all cells "right from the start":

(11) $\quad p{\cdot}b\,; (p{\cdot}a, b, c\,; a, p{\cdot}b, q{\cdot}c)^* \quad .$

(In Section 2.2 we give another solution to this problem.) Obviously, communication behaviours of neighbouring cells match and communication behaviours w.r.t. channel c match if and only if $n-k$ is odd.

Since odd and even numbered cells are distinguished we obtain two kinds of cells which satisfy slightly different relations. For even n ($n \neq 0$) we take the relations as found before. For odd n, we take, in accordance with (11), $p{\cdot}a(i) = a(i-1)$ for $i \geq 0$, and, consequently, since $p{\cdot}a(i) = \surd$ and $a(i-1) = \surd$ for $i<0$, we have $p{\cdot}a(i) = a(i-1)$ for all integer i. Now $b(0) \equiv$ true and for $i \geq 0$ we derive:

$$\begin{array}{cl} & b(i+1) \\ \equiv & \{ \text{ see previous derivation (page 5) } \} \\ & F_n \Rightarrow a(i) = a(i + P(n-1) - (n-1)) \\ & \quad \wedge \ (\forall j : 0 \leq j < n-1 \wedge P(j) < j : a(i + j - (n-1)) = a(i + P(j) - (n-1))) \\ \equiv & \{ \ p{\cdot}a(i) = a(i-1) \text{ for all integer } i \ \} \\ & F_n \Rightarrow a(i) = a(i + P(n-1) - (n-1)) \ \wedge \\ & (\forall j : 0 \leq j < n-1 \ \wedge \ P(j) < j : p{\cdot}a(i+1+j - (n-1)) = p{\cdot}a(i+1+P(j) - (n-1))) \end{array}$$

$\equiv$ { (9); p is of type P''_{n-1}, (8) }
$(F_n \Rightarrow a(i) = q{\cdot}c(i)) \ \wedge \ p{\cdot}b(i+1)$.

Thus we take the following relations for odd numbered cells:

$$\begin{array}{lcl} p{\cdot}a(i) & = & a(i-1) \\ c(i) & = & a(i-1) \\ b(0) & \equiv & \text{true} \\ b(i+1) & \equiv & (F_n \Rightarrow a(i) = q{\cdot}c(i)) \ \wedge \ p{\cdot}b(i+1) \ . \end{array}$$

Given the above relations, we can now compute k such that (9) holds and $n-k$ is odd. As cell $n-1$ is, viewed from cell n, equivalent to subcomponent p, we may write q as p^{n-k}. Since the relations for even and odd numbered cells are different, we distinguish the cases k is even (and n is odd) and k is odd (and n is even).

If k is even, we have $q{\cdot}a(i){=}q{\cdot}c(i)$, and, in order to avoid buffering in both cell n and cell k, we want k to satisfy $p^{n-k}{\cdot}a(i) = a(i + P(n-1) - (n-1))$. From the relations above it can be verified that $p^{n-k}{\cdot}a(i) = a(i - (n-k+1) \operatorname{div} 2)$, using that n is odd. This gives rise to the following equation:

(12) $k: \ \ P(n-1) - (n-1) = -(n-k+1) \operatorname{div} 2$.

For odd k, we have $q{\cdot}a(i-1){=}q{\cdot}c(i)$, so we want k to satisfy: $p^{n-k}{\cdot}a(i-1) = a(i + P(n-1) - (n-1))$. Now n is even and therefore $p^{n-k}{\cdot}a(i-1) = a(i-1-(n-k) \operatorname{div} 2)$. As equation for k we thus obtain $P(n-1) - (n-1) = -((n-k) \operatorname{div} 2 + 1)$, but, since $n-k$ is odd, this equation is equivalent to (12).

Using that $n-k+1$ is even we obtain as solution to (12):

(13) $k_n = 2P(n-1) - n + 3$.

Channel c is thus directed from cell k_n to cell n, $0<n\leq N$. Depending on P, however, k_n may be negative, and the array of cells is therefore extended with negatively numbered cells whose only purpose is to buffer a-values that are to be returned along c-channels. These cells are programmed as follows ($n<0$). For even n:

com P''_n (**in** a:T, **out** c:T) :
  sub $p{:}P''_{n-1}$
  |[**var** x:T;
    $(a?x; p{\cdot}a!x, c!x)^*$
  ]|
moc ,

and for odd n:

com P''_n (**in** a:T, **out** c:T) :
  sub $p{:}P''_{n-1}$
  |[**var** x:T;
    $p{\cdot}a!\surd, c!\surd$
    $; (a?x; p{\cdot}a!x, c!x)^*$
  ]|
moc .

Of course, there should be a last cell to end this array. As stated on page 5 the original problem (0) is solved by solving two identical problems (for P and its inverse). The index of the last cell in the array is therefore given by

(14) $(\mathrm{Min}\, n : 0<n\leq N \wedge F_n : k_n) \text{ min } 0 \text{ min } (\mathrm{Min}\, n : 0<n\leq N \wedge G_n : l_n)$,

where $G_n \equiv P^{-1}(n-1) < n-1$ and $l_n = 2P^{-1}(n-1)-n+3$. The program for this cell is omitted.

For positive n we obtain the following programs. For even n:

```
com P''_n (in a:T, out b:Bool, c:T) :
   sub p:P''_{n-1}, q:P''_{k_n}, r:P''_{l_n}
   |[var x, y, z:T; w:Bool;
       b!true
       ;(a?x, p·b?w, q·c?y, r·c?z
        ;p·a!x, b!((F_n ⇒ x=y) ∧ (G_n ⇒ x=z) ∧ w), c!x
        )*
   ]|
moc ,
```

and, for odd n:

```
com P''_n (in a:T, out b:Bool, c:T) :
   sub p:P''_{n-1}, q:P''_{k_n}, r:P''_{l_n}
   |[var x, y, z:T; w:Bool;
       p·b?w; p·a!√, b!true, c!√
       ;(a?x, p·b?w, q·c?y, r·c?z
        ;p·a!x, b!((F_n ⇒ x=y) ∧ (G_n ⇒ x=z) ∧ w), c!x
        )*
   ]|
moc .
```

Finally, for $n=0$ we find (assuming that cell 0 is not the last cell of the array):

```
com P''_0 (in a:T, out b:Bool, c:T) :
   sub p:P''_{-1}
   |[var x:T;
       b!true
       ;(a?x; p·a!x, b!true, c!x)*
   ]|
moc .
```

The resulting programs can be simplified significantly by removing redundant channels and/or components. For example, input channel $q{\cdot}c$ may be removed from cell n when $\neg F_n$ holds. Such simplifications will be applied and further explained in Section 3.

Like the "conventional" solution from Section 2.0, this solution has constant response time and constant latency, but the attractive thing about this solution is that its size is linear in N.[1] A serious problem is however that it may be non-systolic; this will be illustrated in Section 4.

2.2 Yet another solution

In the previous section we have distinguished odd and even cells in order to avoid deadlock. Deadlock could occur because cell k could initially be unable to engage in a communication with cell n along channel c. Another way to avoid such a deadlock is therefore to avoid these initial communications along c in cell n. To this end we take a communication behaviour of the following form:

(15) $b\,;\,(a, p{\cdot}b\,;\,p{\cdot}a, b, c)^t\,;\,(a, p{\cdot}b, q{\cdot}c\,;\,p{\cdot}a, b, c)^*$,

where t is determined such that cell k is able to communicate along $q{\cdot}c$. Note that $b(1)$ through $b(t)$ have to be computed without the use of channel $q{\cdot}c$. Since it turns out that t is smaller than n (see below), this is no problem: it is sufficient that relation (8) holds for $i{\geq}n$, and therefore we may take arbitrary values for $b(1)$ through $b(t)$.

For the above communication behaviour we first determine an expression for k_n, the cell to which cell n is to be connected. We do this by means of sequence functions. The relevant sequence functions for cell n are given by:

$$\begin{aligned}\sigma_n(a,i) &= 2i+1+N-n\\ \sigma_n(c,i) &= 2i+2+N-n\\ \sigma_n(q{\cdot}c,i) &= 2t+2i+1+N-n \quad .\end{aligned}$$

Since we want to have $a(i)$ and $a(i+P(n{-}1)-(n{-}1))$ available in cell n in the same time slot, we have the following equation for k_n, using that $c(i) = a(i)$:

$$\sigma_n(a,i) = \sigma_{k_n}(c, i+P(n{-}1)-(n{-}1)) \quad ,$$

which has the same solution as equation (12):

$$k_n = 2P(n{-}1) - n + 3 \quad .$$

Given this expression for k_n we can now compute t. We determine t such that the communication behaviours of cells n and k_n match. As equation for t we obtain:

$$t: \quad \sigma_n(q{\cdot}c,i) = \sigma_{k_n}(c,i) \quad ,$$

for $i{\geq}0$. Using the above sequence functions we find:

$$\begin{aligned} & 2t+2i+1+N-n = 2i+2+N-k_n\\ \equiv\ & \{\text{ above relation for } k_n \}\\ & 2t-n = 1-(2P(n{-}1)-n+3)\\ \equiv\ & \{\ \}\\ & t = (n{-}1) - P(n{-}1) \quad .\end{aligned}$$

[1] The program texts for the components may suggest a non-linear network. For each integer n, however, there is at most one instance of component P''_n, which may occur more than once as subcomponent of components with larger numbers.

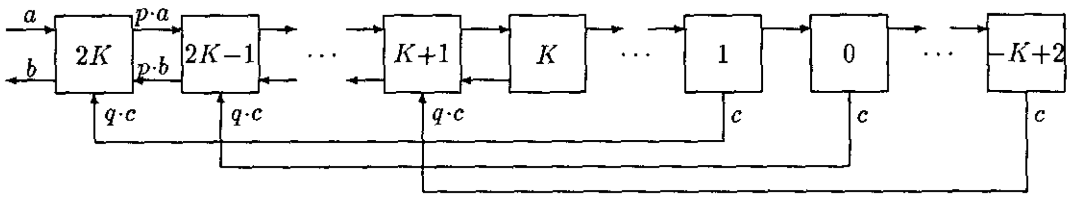

Figure 1: General program for square recognition.

Since channel $q{\cdot}c$ is only used in cells for which F_n holds, we have that $P(n-1) < n-1$ and hence that $0<t$. Furthermore we have that $t < n$ because $P(n-1)\geq 0$. Hence $0<t<n$.

For P^{-1} we obtain a similar communication behaviour, which can be "merged" with the communication behaviour for P.

The disadvantage of this solution is that the cells are not identical because the length of the initialisation in cell n equals $(n-1) - P(n-1)$.

3 Comparison

In this section we generate a program for the square recognition problem by instantiating the program for arbitrary P given in Section 2.1. Subsequently, the thus obtained solution is compared with the one presented in Section 1. For the sake of convenience we assume K to be even and sufficiently large (e.g., $K\geq 4$).

As a first step, we observe that the permutation for the square problem, given by $P(j) = (j+K) \bmod 2K$ for $0\leq j<2K$, is equal to its inverse. Consequently, $G_n \equiv F_n$ and $k_n = l_n$, and therefore we can simplify the general program significantly by removing subcomponents r.

A further reduction is possible by observing that F_n is equivalent to $(n-1+K) \bmod 2K < n-1$ which may be simplified to $K<n\leq 2K$. This enables us to remove input $q{\cdot}c$ from cells n, $1\leq n\leq K$. For $K<n\leq 2K$ we have $P(n-1)=n-1-K$ so we obtain (cf. (13)): $k_n = n-2K+1$. Since $F_n \equiv K<n\leq 2K$, it follows from (14) that the last cell has number $-K+2$, and moreover that $-K+2\leq k_n<2$, as a consequence of which output c may be removed from cells n with $2\leq n\leq 2K$.

Since $\neg F_n$ holds for $0<n\leq K$ and $b(i) \equiv$ true for cell 0, we have $b(i) \equiv$ true for all these cells, and therefore we can remove the b-channels from cells 0 through $K-1$ and let cell K generate sequence b. Figure 1 gives an impression of the network thus obtained; it consists of $3K-1$ cells. By folding the array of cells over a 180 degrees between cells $K+1$ and K, and between cells 2 and 1, the length of each c-channel can be reduced to a constant (independent of K), which implies that the program is systolic.

To obtain a program comparable with the program from Section 1 we apply two more transformations.

Inspection of the computation performed by the network in Figure 1 learns us that cells $-K+2$ through 0 can be removed at the expense of $K-1$ extra channels between cells $K+1$ through $2K$. This improvement follows from the observation that cell 1 sends, in the same time slot, the same a-value to both cell 0 (via $p{\cdot}a$) and cell $2K$ (via c). In the next time slot cell 0 passes this a-value to cell $2K-1$ via c, which however could equally well be retrieved from cell $2K$. For this purpose we equip cell $2K-1$ with an additional input channel e, say, called $p{\cdot}e$ in cell $2K$. A similar reasoning applies to cells $-K+3$ through 0.

Finally, we integrate cells 1 through K with cells $2K$ through $K+1$, respectively. That is, mentally we fold the array between cells $K+1$ and K over a 180 degrees, and then we combine the cells opposite to each other. For this purpose we rename the a-channels of cells 1 through K to f-channels.

This results in the following programs (assuming that K is even).

```
com P''_2K (in a:T, out b:Bool) :
  sub p:P''_2K-1
  |[var x, z:T; w:Bool;
      b!true
      ;(a?x, p·b?w, p·f?z
       ;p·a!x, b!(x=z ∧ w), p·e!z
       )*
  ]|
moc .
```

For $K<n<2K$ we have for even n:

```
com P''_n (in a, e:T, out b:Bool, f:T) :
  sub p:P''_n-1
  |[var x, y, z:T; w:Bool;
      b!true
      ;(a?x, p·b?w, e?y, p·f?z
       ;p·a!x, b!(x=y ∧ w), p·e!y, f!z
       )*
  ]|
moc ,
```

and, for odd n:

```
com P''_n (in a, e:T, out b:Bool, f:T) :
  sub p:P''_n-1
  |[var x, y, z:T; w:Bool;
      p·b?w; p·a!√, b!true, p·e!√, f!√
      ;(a?x, p·b?w, e?y, p·f?z
       ;p·a!x, b!(x=y ∧ w), p·e!y, f!z
       )*
  ]|
moc .
```

Finally, cell K generates true's along channel b (assuming K even):

```
com P''_K (in a, e:T, out b:Bool, f:T) :
  |[var x, y:T;
      b!true
      ;(a?x, e?y; b!true, f!x)*
  ]|
moc .
```

Apart from the fact that all cells are active "right from the start" —performing dummy actions initially— the obtained network is equivalent to the conventional one.

4 A non-systolic program

As mentioned before, instantiation of the program for arbitrary P may result in a non-systolic solution. Take, for example, the perfect-shuffle permutation. Its inverse is given by $P^{-1}(j) = K(j \bmod 2) + j \operatorname{div} 2$, for $0 \leq j < 2K$. For odd n we have $P^{-1}(n-1) = (n-1)/2$, so it immediately follows that G_n, i.e. $(n-1)/2 < n-1$, holds for $n > 1$ (n odd). We then obtain $l_n = 2$ for all cells n with n odd and larger than one, which means that all these cells are connected to cell 2. In other words, cell 2 "broadcasts" the same a-value to all these cells. Evidently, the resulting program is therefore not systolic. (In [3] a systolic program for perfect-shuffle recognition is derived. In that program the computation is organized such that only a small number of cells need the same a-value in the same time slot. This program is outside the scope of the approach presented in this paper.)

In order to guarantee that instantiation of the program from Section 2 results in a systolic program, that is, a program in which the fan-out of each cell is bounded, P should satisfy the following restriction:

$$(\# m, n : 0 \leq m < N \wedge F_m \wedge 0 \leq n < N \wedge F_n : k_n = k_m) \leq M \quad ,$$

where # denotes 'number of' and M is a positive constant (independent of N). Using (13) the above formula reduces to

$$(16) \quad (\# m, n : 0 \leq m < N \wedge F_m \wedge 0 \leq n < N \wedge F_n : 2(P(m-1) - P(n-1)) = m - n) \leq M \quad .$$

Of course, P^{-1} has to meet a similar requirement. Using (16) it can easily be verified whether our approach yields a systolic solution for a particular P. For instance, for palindrome recognition we have $P(n-1) = N - n$ which obviously satisfies (16) with $M = 1$.

5 Summary of results

Typical for the "linear array" solutions to several instances of the general recognition problem [2, 3, 5, 6, 7] is that at some stage in the design auxiliary channels are introduced between neighbour cells to carry input values (to the program) indirectly via a chain of neighbouring cells to the right cell at the right moment. It is shown that for the general problem this approach forces us to introduce an array of auxiliary channels, resulting in a program of a size quadratic in N. To obtain a program of linear size, a quite different approach is taken, in which an (input) value is directly retrieved from the cell that received that value just before. In this way, cells that are arbitrarily far apart may be connected and the need for buffering in linked cells is avoided. Depending on P, the array of cells is extended with a number of extra cells whose sole purpose is to buffer input values that are to be returned via the direct feedback connections.

To ensure the feasibility of the above approach, we have transformed the problem of recognizing P-invariant segments into two simpler problems involving P and P^{-1}. Another problem that had to be solved was the design of a deadlock-free communication behaviour. We have chosen to let the communication behaviours of odd and even numbered cells alternate so as to activate all cells "right from the start" —performing dummy actions initially. We have also shown that it is possible to avoid these initial communications altogether, the drawback of this solution being that the cells of the resulting program have a more complicated initialisation.

Using our general solution, it is rather straightforward to construct a parallel program for an instance of P. For some concrete cases the resulting program can be transformed into the more "conventional" programs. This is illustrated by means of the square recognition problem. Depending on the permutation at hand, the solution may be non-systolic because the number of output channels of some cells may be proportional to N. We have characterized the permutations for which it is guaranteed that a systolic solution results.

In conclusion, the direct retrieval of input values from the cell that received this value just before is the major design decision made. Furthermore, cells are started simultaneously by distinguishing odd and even cells, and extra cells are introduced that only buffer input values that are to be returned via auxiliary channels. The resulting program has a size linear in N and has constant response time and constant latency. The traditional approach leads to program with a size quadratic in N. Therefore the applied technique is considered to be a fruitful extension of the conventional design technique [1, 4].

Acknowledgements

We would like to thank Prof. F.E.J. Kruseman Aretz for some helpful comments on the presentation of this paper. The members of the Eindhoven Algorithm Club are also gratefully acknowledged for a critical reading of an earlier version of this paper.

References

[0] Hoare C.A.R.
Communicating Sequential Processes.
Communications of the ACM **21** (1978) 666–677.

[1] Kaldewaij A., Rem M.
The Derivation of Systolic Computations.
Science of Computer Programming **14** (1990) 229–242.

[2] Katoen J.P., Rem M.
Recognizing K-rotated Segments.
In: proceedings workshop on Massive Parallelism: Hardware, Programming and Applications, Amalfi, Italy (1989). (to appear)

[3] Katoen J.P., Schoenmakers L.A.M.
Recognizing Perfect-Shuffles.
In: J.P. Katoen, Case Studies in Calculational Program Design, Eindhoven University of Technology, The Netherlands (1989) 49–61.

[4] Rem M.
Trace Theory and Systolic Computations.
In: J.W. de Bakker et al. (eds), PARLE'87: Parallel Architectures and Languages Europe, LNCS **258**, Springer-Verlag (1987) 14–33.

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

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

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

In this series appeared :

No.	Author(s)	Title
85/01	R.H. Mak	The formal specification and derivation of CMOS-circuits.
85/02	W.M.C.J. van Overveld	On arithmetic operations with M-out-of-N-codes.
85/03	W.J.M. Lemmens	Use of a computer for evaluation of flow films.
85/04	T. Verhoeff H.M.L.J.Schols	Delay insensitive directed trace structures satisfy the foam the foam rubber wrapper postulate.
86/01	R. Koymans	Specifying message passing and real-time systems.
86/02	G.A. Bussing K.M. van Hee M. Voorhoeve	ELISA, A language for formal specification of information systems.
86/03	Rob Hoogerwoord	Some reflections on the implementation of trace structures.
86/04	G.J. Houben J. Paredaens K.M. van Hee	The partition of an information system in several systems.
86/05	J.L.G. Dietz K.M. van Hee	A framework for the conceptual modeling of discrete dynamic systems.
86/06	Tom Verhoeff	Nondeterminism and divergence created by concealment in CSP.
86/07	R. Gerth L. Shira	On proving communication closedness of distributed layers.
86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arun Kumar	Compositional semantics for real-time distributed computing (Inf.&Control 1987).
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language.
86/10	J. Hooman	A compositional proof theory for real-time distributed message passing.
86/11	W.P. de Roever	Questions to Robin Milner - A responder's commentary (IFIP86).
86/12	A. Boucher R. Gerth	A timed failures model for extended communicating processes.
86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4).

86/14	R. Koymans	Specifying passing systems requires extending temporal logic.
87/01	R. Gerth	On the existence of sound and complete axiomati zations of the monitor concept.
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases.
87/03	G.J. Houben J.Paredaens	A formal approach to distributed information systems.
87/04	T.Verhoeff	Delay-insensitive codes - An overview.
87/05	R.Kuiper	Enforcing non-determinism via linear time temporal logic specification.
87/06	R.Koymans	Temporele logica specificatie van message passing en real-time systemen (in Dutch).
87/07	R.Koymans	Specifying message passing and real-time systems with real-time temporal logic.
87/08	H.M.J.L. Schols	The maximum number of states after projection.
87/09	J. Kalisvaart L.R.A. Kessener W.J.M. Lemmens M.L.P. van Lierop F.J. Peters H.M.M. van de Wetering	Language extensions to study structures for raster graphics.
87/10	T.Verhoeff	Three families of maximally nondeterministic automata.
87/11	P.Lemmens	Eldorado ins and outs. Specifications of a data base management toolkit according to the functional model.
87/12	K.M. van Hee and A.Lapinski	OR and AI approaches to decision support systems.
87/13	J.C.S.P. van der Woude	Playing with patterns - searching for strings.
87/14	J. Hooman	A compositional proof system for an occam-like real-time language.
87/15	C. Huizing R. Gerth W.P. de Roever	A compositional semantics for statecharts.
87/16	H.M.M. ten Eikelder J.C.F. Wilmont	Normal forms for a class of formulas.
87/17	K.M. van Hee G.-J.Houben J.L.G. Dietz	Modelling of discrete dynamic systems framework and examples.

87/18	C.W.A.M. van Overveld	An integer algorithm for rendering curved surfaces.
87/19	A.J.Seebregts	Optimalisering van file allocatie in gedistribueerde database systemen.
87/20	G.J. Houben J. Paredaens	The R^2 -Algebra: An extension of an algebra for nested relations.
87/21	R. Gerth M. Codish Y. Lichtenstein E. Shapiro	Fully abstract denotational semantics for concurrent PROLOG.
88/01	T. Verhoeff	A Parallel Program That Generates the Möbius Sequence.
88/02	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specification for Information Systems.
88/03	T. Verhoeff	Settling a Question about Pythagorean Triples.
88/04	G.J. Houben J.Paredaens D.Tahon	The Nested Relational Algebra: A Tool to Handle Structured Information.
88/05	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specifications for Information Systems.
88/06	H.M.J.L. Schols	Notes on Delay-Insensitive Communication.
88/07	C. Huizing R. Gerth W.P. de Roever	Modelling Statecharts behaviour in a fully abstract way.
88/08	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	A Formal model for System Specification.
88/09	A.T.M. Aerts K.M. van Hee	A Tutorial for Data Modelling.
88/10	J.C. Ebergen	A Formal Approach to Designing Delay Insensitive Circuits.
88/11	G.J. Houben J.Paredaens	A graphical interface formalism: specifying nested relational databases.
88/12	A.E. Eiben	Abstract theory of planning.
88/13	A. Bijlsma	A unified approach to sequences, bags, and trees.
88/14	H.M.M. ten Eikelder R.H. Mak	Language theory of a lambda-calculus with recursive types.

88/15	R. Bos C. Hemerik	An introduction to the category theoretic solution of recursive domain equations.
88/16	C.Hemerik J.P.Katoen	Bottom-up tree acceptors.
88/17	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable specifications for discrete event systems.
88/18	K.M. van Hee P.M.P. Rambags	Discrete event systems: concepts and basic results.
88/19	D.K. Hammer K.M. van Hee	Fasering en documentatie in software engineering.
88/20	K.M. van Hee L. Somers M.Voorhoeve	EXSPECT, the functional part.
89/1	E.Zs.Lepoeter-Molnar	Reconstruction of a 3-D surface from its normal vectors.
89/2	R.H. Mak P.Struik	A systolic design for dynamic programming.
89/3	H.M.M. Ten Eikelder C. Hemerik	Some category theoretical properties related to a model for a polymorphic lambda-calculus.
89/4	J.Zwiers W.P. de Roever	Compositionality and modularity in process specification and design: A trace-state based approach.
89/5	Wei Chen T.Verhoeff J.T.Udding	Networks of Communicating Processes and their (De-)Composition.
89/6	T.Verhoeff	Characterizations of Delay-Insensitive Communication Protocols.
89/7	P.Struik	A systematic design of a paralell program for Dirichlet convolution.
89/8	E.H.L.Aarts A.E.Eiben K.M. van Hee	A general theory of genetic algorithms.
89/9	K.M. van Hee P.M.P. Rambags	Discrete event systems: Dynamic versus static topology.
89/10	S.Ramesh	A new efficient implementation of CSP with output guards.
89/11	S.Ramesh	Algebraic specification and implementation of infinite processes.
89/12	A.T.M.Aerts K.M. van Hee	A concise formal framework for data modeling.

89/13	A.T.M.Aerts K.M. van Hee M.W.H. Hesen	A program generator for simulated annealing problems.
89/14	H.C.Haesen	ELDA, data manipulatie taal.
89/15	J.S.C.P. van der Woude	Optimal segmentations.
89/16	A.T.M.Aerts K.M. van Hee	Towards a framework for comparing data models.
89/17	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra.
90/1	W.P.de Roever-H.Barringer C.Courcoubetis-D.Gabbay R.Gerth-B.Jonsson-A.Pnueli M.Reed-J.Sifakis-J.Vytopil P.Wolper	Formal methods and tools for the development of distributed and real time systems, pp. 17.
90/2	K.M. van Hee P.M.P. Rambags	Dynamic process creation in high-level Petri nets, pp. 19.
90/3	R. Gerth	Foundations of Compositional Program Refinement - safety properties - , p. 38.
90/4	A. Peeters	Decomposition of delay-insensitive circuits, p. 25.
90/5	J.A. Brzozowski J.C. Ebergen	On the delay-sensitivity of gate networks, p. 23.
90/6	A.J.J.M. Marcelis	Typed inference systems : a reference document, p. 17.
90/7	A.J.J.M. Marcelis	A logic for one-pass, one-attributed grammars, p. 14.
90/8	M.B. Josephs	Receptive Process Theory, p. 16.
90/9	A.T.M. Aerts P.M.E. De Bra K.M. van Hee	Combining the functional and the relational model, p. 15.
90/10	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra, p. 30. (Revised version of CSNotes 89/17).
90/11	P. America F.S. de Boer	A proof system for process creation, p. 84.
90/12	P.America F.S. de Boer	A proof theory for a sequential version of POOL, p. 110.
90/13	K.R. Apt F.S. de Boer E.R. Olderog	Proving termination of Parallel Programs, p. 7.
90/14	F.S. de Boer	A proof system for the language POOL, p. 70.
90/15	F.S. de Boer	Compositionality in the temporal logic of concurrent systems, p. 17.

90/16	F.S. de Boer C. Palamidessi	A fully abstract model for concurrent logic languages, p. 23.
90/17	F.S. de Boer C. Palamidessi	On the asynchronous nature of communication in concurrent logic languages: a fully abstract model based on sequences, p. 29.
90/18	J.Coenen E.v.d.Sluis E.v.d.Velden	Design and implementation aspects of remote procedure calls, p. 15.
90/19	M.M. de Brouwer P.A.C. Verkoulen	Two Case Studies in ExSpect, p. 24.
90/20	M.Rem	The Nature of Delay-Insensitive Computing, p.18.
90/21	K.M. van Hee P.A.C. Verkoulen	Data, Process and Behaviour Modelling in an integrated specification framework, p. 37.
91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses of "if..., then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.