A Parallel Program for the Recognition of $P$-invariant Segments

Joost Katoen
Philips Research Laboratories
Dept. of Information and Software Technology
P.O. Box 80.000, 5600 JA Eindhoven, The Netherlands
e-mail: katoen@prl.philips.nl
tel.: +31-40-743137, fax: +31-40-744004.

Berry Schoenmakers
Eindhoven University of Technology
Dept. of Mathematics and Computing Science
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
e-mail: wsinbs@win.tue.nl
tel.: +31-40-474318, fax: +31-40-436685.
Let \( P \) be an arbitrary, but fixed permutation on \([0..N]\), with \( N \geq 0 \). In this paper we design a parallel program that determines for each segment of \( N \) successive elements of its input sequence whether it is invariant under \( P \). The program consists of a linear arrangement of cells and contains—in addition to links between neighbour cells—connections between cells that are arbitrarily far apart. For some well-known instances of the general problem, such as palindrome recognition and square recognition, the solutions are systolic and comparable to the “traditional” ones. Depending on \( P \), however, the solutions may be non-systolic. Figure 1 gives an impression of the obtained parallel palindrome recognizer.

![Diagram](image)

**Figure 1**: General program for palindrome recognition.

1. **INTRODUCTION**

Problems like palindrome recognition and square recognition have been used in several papers [5, 6, 7, 8] to serve as examples to explain design techniques for fine-grained parallel programs (in particular, linear systolic arrays). Problems of this type are in vogue, because at solving them all attention may be focused on arranging the computation such that inputs to the program are transferred to the right cell at the right moment, while the computations to be performed by the individual cells play only a minor role.

The recognition problems can be considered as instances of the general problem of recognizing \( P \)-invariant “segments”, where \( P \) is an arbitrary, but fixed permutation on \([0..N]\), for fixed \( N \) \((N \geq 0)\). For instance, for palindrome recognition permutation \( P \) is given by \( P_j = N-1-j \). In this paper we show how we design a parallel program that determines for each segment of length \( N \) of its input sequence whether it is \( P \)-invariant. The design is drawn from [4] which also contains some alternative solutions to the general problem.

We adopt the design technique as explained in [2, 5]. According to this technique parallel programs are specified by \( i/o \)-relations—relations between input and output values of the program—and \emph{communication behaviours} that state the order in which the communications take place. Programs are denoted in a CSP-like notation [1].
2. DERIVATION

The problem is to design a parallel program with input channel $a$ of arbitrary type $T$ and output channel $b$ of type Bool that determines for each segment $a[i..i+N]$ ($i \geq 0$) whether it is $P$-invariant. More precisely, the program has to satisfy the following i/o-relation

$$b(i) \equiv (\forall j : 0 \leq j < N : a(i + j) = a(i + P_j)) \tag{1}$$

where $a(i)$ and $b(i)$ denote the $(i+1)$-st elements of sequences $a$ and $b$, respectively. It follows from (1) that $b(i)$ depends on all elements of $a[i..i+N]$, and, consequently, that (1) requires a communication behaviour like $a^N; (b;a)^*$. The obvious way to start the derivation is to generalize (1) in some way, thereby obtaining specifications of $N+1$ cells. From experience (see e.g. [8]), however, we know that such cells must have a communication behaviour that depends on $n$ (e.g., $a^n; (b;a)^*$), and, consequently, that such cells do not have identical commands. To obtain a simpler communication behaviour, such as $(b;a)^*$, we let $b(i)$ depend on segment $a[i-N..i)$ and change for $i \geq N$ the i/o-relation into

$$b(i) \equiv (\forall j : 0 \leq j < N : a(i + j - N) = a(i + P_j - N)) \tag{2}$$

Note that nothing is specified about values $b(0)$ through $b(N-1)$. By neglecting the first $N$ outputs of the program we obtain a solution to (1).

As a first step in the design we generalize (2) by replacing $N$ by variable $n$ ($0 \leq n \leq N$). So the problem is divided into the design of $N+1$ cells, where cell $n$ establishes

$$b_n(i) \equiv (\forall j : 0 \leq j < n : a(i + j - n) = a(i + P_j - n)) \tag{3}$$

for $i \geq n$, and with $(b; a)^*$ as (external) communication behaviour. Sequence $a$ is fed to cell $n$ via input $a_n$: $a_n(i) = a(i)$. The output values of cell $N$ now solve (2).

The derivation proceeds by deriving relations from (3) that express how cell $n$ computes its output values from other values. It is immediate that $b_0(i) \equiv$ true for $i \geq 0$, and for $n \geq 1$ and $i \geq n-1$ we derive

$$b_n(i + 1)$$

$$\equiv \{ \text{ (3) } \}$$

$$\{ \forall j : 0 \leq j < n : a(i + 1 + j - n) = a(i + 1 + P_j - n) \}$$

$$\equiv \{ \text{ split off } j = n-1; \text{ let } D_n = n-1 - P_{n-1} \}$$

$$a(i) = a(i - D_n) \land (\forall j : 0 \leq j < n-1 : a(i + j - (n-1)) = a(i + P_j - (n-1)))$$

$$\equiv \{ \text{ (3) } \}$$

$$a(i) = a(i - D_n) \land b_{n-1}(i)$$

In case $D_n < 0$, $a(i - D_n)$ has not yet been received by cell $n$, and, consequently, $(b; a)^*$ is not a possible communication behaviour for all cells. Fortunately, the following observation helps us out. The right-hand side of (1) can be transformed as follows:

$$\{ \text{ domain split } \}$$

$$\{ \forall j : 0 \leq j < N : a(i + j) = a(i + P_j) \}$$

$$\land (\forall j : 0 \leq j < N \land P_j > j : a(i + j) = a(i + P_j))$$

$$\land (\forall j : 0 \leq j < N \land P_j = j : a(i + j) = a(i + P_j))$$
\[ \land (\forall j : 0 \leq j < N \land P_j < j : a(i + j) = a(i + P_j)) \]
\[ \equiv \{ \text{ dummy change } j := P_j^{-1} \text{ in first conjunct } \} \]
\[ (\forall j : 0 \leq P_j^{-1} < N \land j > P_j^{-1} : a(i + P_j^{-1}) = a(i + j)) \]
\[ \land (\forall j : 0 \leq j < N \land P_j < j : a(i + j) = a(i + P_j)) \]
\[ \equiv \{ P^{-1} \text{ is a permutation on } [0..N] \} \]
\[ (\forall j : 0 \leq j < N \land P_j^{-1} < j : a(i + j) = a(i + P_j^{-1})) \]
\[ \land (\forall j : 0 \leq j < N \land P_j < j : a(i + j) = a(i + P_j)) \; . \]

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 cells establishing \((i \geq n)\)
\[ b_n(i) \equiv (\forall j : 0 \leq j < n \land P_j < j : a(i + j - n) = a(i + P_j - n)) \; , \]
which enables \((b : a)^*\) as communication behaviour for all cells.

Proceeding as above we obtain the following relation for \( n > 0 \)
\[ b_n(i + 1) \equiv (D_n > 0 \Rightarrow a(i) = a(i - D_n)) \land b_{n-1}(i) \; , \]
for \( i \geq n - 1 \). Note that \( a(i - D_n) \) is required for the computation of \( b_n(i+1) \) only if \( D_n > 0 \), which ensures that this value has already been received by cell \( n \) and has been passed on to cell \( n-1 \) in the mean time.

The simplest way to make \( a(i - D_n) \) available to cell \( n \) is to buffer the last \( N \) values received along \( a \) in each cell, but this makes the cells too bulky. In the solutions to several instances of (1) the “old” \( a \)-value is retrieved (indirectly) from cell \( n-1 \) by introducing auxiliary channels between neighbouring cells. However, the fact that we are dealing with an arbitrary permutation \( P \) forces us to equip each cell with an array of auxiliary channels (see [4]), resulting in a program of a size quadratic in \( N \).

In order to obtain a program of linear size we take a quite different approach. Observe that \( a(i - D_n) \) has reached some cell \( k \), \( k < n \), at the time it is needed by cell \( n \). Our idea now is to retrieve \( a(i - D_n) \) directly from cell \( k \) thereby avoiding the need for buffers in both cells. More precisely, we add an auxiliary channel \( c \) directed from cell \( k \) to cell \( n \) and we determine \( k \) such that
\[ c_k(i) = a(i - D_n) \; , \]
for \( i \geq 0 \). For cell \( n \) \((n > 0)\) we then have
\[ a_{n-1}(i) = a_n(i) \]
\[ c_n(i) = a_n(i) \]
\[ b_n(i + 1) \equiv (D_n > 0 \Rightarrow a_n(i) = c_k(i)) \land b_{n-1}(i) \; . \]

Now, a possible overall communication behaviour is
\[ b_n ; (a_n, b_{n-1}, c_k; a_{n-1}, b_n, c_n)^* \; , \]
where the comma denotes arbitrary interleaving of the communications it connects. Unfortunately, this behaviour causes deadlock (cf. [8]): cells are activated one by one in a “pass it on, neighbour!” fashion starting at cell \( N \), but 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”:

\[ b_{n-1} ; (a_{n-1}, b_n, c_n ; a_n, b_{n-1}, c_k)^* . \]  

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 (5), \( a_{n-1}(i) = a_n(i-1) \), and we thus obtain

\[
\begin{align*}
  a_{n-1}(i) &= a_n(i-1) \\
  c_n(i) &= a_n(i-1) \\
  b_n(i+1) &= (D_n > 0 \Rightarrow a_n(i) = c_k(i)) \land b_{n-1}(i+1).
\end{align*}
\]

Given the relations for odd and even \( n \), we can now compute \( k \) such that (4) holds and \( n-k \) is odd. This gives rise to the following equation (see [4])

\[ k : P_{n-1} - (n-1) = -(n-k+1) \text{ div } 2 , \]

with solution

\[ k_n = 2P_{n-1} - n + 3. \]

Channel \( c \) is thus directed from cell \( k_n \) to cell \( n, 1 \leq n \leq N \). Using that \( D_n > 0 \), it follows from (6) that \( -N+3 \leq k_n < n \). Since \( k_n \) may be negative the array of cells is extended with cells—like cells \( -N+3 \) through \( N/2-1 \) in Figure 1—which sole purpose is to buffer input values that are to be returned via the \( c \)-connections. These cells are programmed as follows \( (n<0) \). For even \( n \) and odd \( n \), respectively:

\[
[[\text{var } x:T; \\
  (a_n?x; a_{n-1}!x, c_n!x)^*]]  \\
[[\text{var } x:T; \\
  (a_{n-1}!x, c_n!x; a_n?x)^*]]
\]

Of course, there should be a last cell to end the array. As stated before, (1) 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

\[(\text{Min } n : 1 \leq n \leq N \land D_n > 0 : k_n) \text{ min } 0 \text{ min } (\text{Min } n : 1 \leq n \leq N \land E_n > 0 : l_n)\]

where \( E_n = (n-1) - P_{n-1}^{-1} \) and \( l_n = 2P_{n-1}^{-1} - n + 3 \). For lack of space, the program for this cell is omitted.

For positive \( n \) we obtain the following programs. For even \( n \):

\[
[[\text{var } x, y, z:T; w: \text{Bool} ; \\
  b_n!w \\
  ; (a_n?x, b_{n-1}?w, c_n?y, c_n?z \\
  ; a_{n-1}!x, b_n!(D_n > 0 \Rightarrow x=y) \land (E_n > 0 \Rightarrow x=z) \land w), c_n!x)^*]]
\]
and, for odd \( n \):

\[
\left[ \text{var } x, y, z: T; \, w: \text{Bool}; \,
    b_n = ? w \\
    ; (a_{n-1} ! x, b_n ! ((D_n > 0 \Rightarrow x = y) \land (E_n > 0 \Rightarrow x = z) \land w), c_n ! x \\
    ; a_n ! x, b_{n-1} ? w, c_n ? y, c_n ? z \right)^a
\]

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

\[
\left[ \text{var } x: T;\,
    b_0 ! \text{true}; (a_0 ? x; a_{-1} ! x, b_0 ! \text{true}, c_0 ! x)^a \right]
\]

The resulting programs can be simplified significantly by removing redundant channels and/or cells. For example, input channel \( c_{k_n} \) may be removed from cell \( n \) when \( D_n < 0 \).

3. 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 \( P_{j}^{-1} = K(j \mod 2) + j \div 2 \), for \( 0 \leq j < 2K \). For odd \( n \) we have \( P_{n-1}^{-1} = (n-1)/2 \), so it immediately follows that \( E_n < 0 \), 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 general program 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 for all \( k \):

\[
(# n : 1 \leq n \leq N \land D_n > 0 : k_n = k) \leq M ,
\]

where \# denotes ‘number of’ and \( M \) is a positive constant (independent of \( N \)). Of course, \( P^{-1} \) has to meet a similar requirement.

As pointed out to us by Wim Kloosterhuis, the fact that the program for perfect-shuffle recognition is not systolic is a direct consequence of distinguishing odd and even numbered cells so as to avoid deadlock. By distinguishing cells modulo 3, say, a systolic solution to the perfect-shuffle problem can be obtained. In this way systolic solutions can be obtained for many more permutations.

4. CONCLUSION

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 a program with a size quadratic in $N$. Therefore the applied technique is considered to be a fruitful extension to the design technique advocated in [2, 5].

REFERENCES


