A systolic design for dynamic programming

by

Rudolf H. Mak and Pieter Struijk

89/2

February, 1989
This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science of 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.
A SYSTOLIC DESIGN FOR DYNAMIC PROGRAMMING

Rudolf H. Mak
Pieter Struik

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

1. Introduction

Systolic computations are concurrent computations that are particularly suited for implementation as VLSI-circuits. In [Zw] a systolic computation is characterized as a network of simple cells such that:

- the network consists of a regular arrangement of cells connected by unidirectional channels
- each cell consumes streams of input values and produces streams of output values
- each cell communicates with a fixed collection of neighbor cells only
- the communication behavior of a cell is independent of the values communicated
- synchronization of the cells is by message passing only

It is well known that dynamic programming can be formulated as a systolic computation in the above sense [ChaIbPa], [Che], [GuKuTh], [LoTch], [Mol]. With the exception of [Che] and [Mol] these papers, however, emphasize the VLSI-architecture and not the design process. In this paper we formally derive a systolic computation for dynamic programming. In this way we not only ensure correctness by design, but we also make explicit the trade-off between space and time. We shall, in particular, demonstrate that the introduction of two auxiliary variables per cell is sufficient to turn a quadratic solution into a linear one.
2. Program notation

We use a CSP-like notation [Ho] to denote the cells of a network. As an example of such program texts, also called components, consider:

\[
\text{com COPY1 (a?int , b!int) :} \\
| [\text{var } x : \text{int} \\
| (a?x ; b!x)^* \\
|] \\
\]

This component denotes a cell with input stream \(a?\) and output stream \(b!\). Throughout this paper we shall use the notational convention that the names of input (output) streams must end on the symbol "?" ("!"). The individual communication events of a stream, say \(s\), are denoted by \(s(k)\), for \(k \geq 0\). The body of the above program consists of the declaration of a local variable \(x\) and a command \((a?x ; b!x)^*\). This command is a repetition consisting of two statements. Statement \(a?x\) is an input statement. It is equivalent to \(x := a?(k)\), where \(k\) is the number of inputs previously received. Statement \(b!x\) is an output statement. It is equivalent to \(b!(k) := x\), where \(k\) is the number of outputs previously sent. We shall assume that the names used for streams and local variables are such that the input (output) statements are unambiguous, i.e. contain exactly one symbol "?" ("!"). As usual ";" denotes sequential composition and "*" denotes repetition.

A command determines both the order in which the input events and output events take place and the way in which the output values depend on the input values. The first item is called the communication behavior and the latter the i/o-relation. For instance, we say that component \(\text{COPY1}\) realizes the i/o-relation \(b!(k) = a?(k)\) and exhibits communication behavior \((a? ; b!)^*\). In general the communication behavior of a component defines a partial order on its communication events. For communication behavior \(S\) we denote this partial order by \(\rightarrow_S\). For instance, let \(S_1\) be the communication behavior of \(\text{COPY1}\). Then the
partial order $\mathcal{S}_1$ is given by (the reflexive transitive closure of) $a? (k) \overset{\mathcal{S}_1}{\rightarrow} a? (k+1)$, $b! (k) \overset{\mathcal{S}_1}{\rightarrow} b! (k+1)$, $a? (k) \overset{\mathcal{S}_1}{\rightarrow} b! (k)$ and $b! (k) \overset{\mathcal{S}_1}{\rightarrow} a? (k+1)$, for $k \geq 0$. Note that $a? (k) \overset{\mathcal{S}_1}{\rightarrow} b! (k)$ is enforced by the i/o-relation, whereas $b! (k) \overset{\mathcal{S}_1}{\rightarrow} a? (k+1)$ is a consequence of the fact that we have used only one local variable. Successive events of the same stream are always part of the partial order of a communication behavior. Therefore, they will be left unmentioned in the sequel.

Usually, components are specified by stating the required i/o-relation only. When the communication behavior is left unspecified, there are, in general, several components, each with a distinct communication behavior, that satisfy the specified i/o-relation. Consider for example the component COPY2 that establishes the same i/o-relation as COPY1, but has a more liberal communication behavior:

```
com COPY2 (a?int , b!int) :
| [var x , y : int
| a?x ; (a?y , b!x ; x := y)*
| ]
```

moc

The symbol "," in the command indicates that the communication events a? and b! can be performed in any order or even concurrently. The comma takes precedence over the semicolon. The meaning of the comma is perhaps best appreciated by considering the partial ordering of communication events it defines. Let $\mathcal{S}_2$ be the communication behavior of COPY2. Then $\overset{\mathcal{S}_2}{\rightarrow}$ is given by $a? (k) \overset{\mathcal{S}_2}{\rightarrow} b! (k)$ and $b! (k) \overset{\mathcal{S}_2}{\rightarrow} a? (k+2)$. Note that in the $k$-th step of the repetition both input event $a? (k+1)$ and output event $b! (k)$ must occur, but that neither $a? (k+1) \overset{\mathcal{S}_2}{\rightarrow} b! (k)$ nor $b! (k) \overset{\mathcal{S}_2}{\rightarrow} a? (k+1)$ is prescribed. The communication behavior of COPY1 is that of a one-place buffer, whereas COPY2 has the communication behavior of a two-place buffer. Note that the buffer capacity is reflected in the number of local variables. The assignment $x := y$ is called an internal event. If we ignore this internal event,
COPY2 turns out to be twice as fast as COPY1, since in each step of its repetition it can perform an input action and an output action concurrently. In the next section we introduce sequence functions for analyzing the time complexity of components. It is important to observe, however, that it is possible to trade space (i.e. variables) for time (i.e. a "better" sequence function). For a thorough treatment of the program notation and the partial orders they define we refer to [Zw].

3. Sequence functions

Let $r$ be a component with communication behavior $S$. A sequence function $\sigma$ for $r$ is a function that assigns to each event $e$ of $r$ a natural number $\sigma(e)$ such that $e \xrightarrow{S} e'$ implies $\sigma(e) < \sigma(e')$ for distinct events $e$ and $e'$ of $r$. We say that $\sigma$ respects $\xrightarrow{S}$. For instance, the function $\sigma$ defined by

\[
\begin{align*}
\sigma(a?,i) &= 2i, & i \geq 0 \\
\sigma(b!,i) &= 2i + 1, & i \geq 0
\end{align*}
\]

is a sequence function for COPY1. Note that we have written $\sigma(a?,i)$ instead of $\sigma(a?(i))$.

Sequence functions serve a number of purposes. In [Zw] it is shown that the existence of a set of "matching" sequence functions, one for each component of a network, guarantees the absence of deadlock. Moreover, each sequence function defines a scheduling of the events of a component. Therefore, they can be used to analyze the time complexity of components. The existence of a sequence function $\sigma$ such that $\sigma(e) = n$ implies that there exists a scheduling of the events of $r$ such that any sequence of events that entirely precedes event $e$ contains at most $n$ events. Likewise, the absence of a sequence function $\sigma$ such that $\sigma(e) < n$ implies that event $e$ has to be preceded by a sequence of at least $n$ events. Hence, sequence functions can be used to obtain both lower bounds and upper bounds for the time complexity of a component.
4. Dynamic programming

Dynamic programming is a technique to solve combinatorial optimization problems that involves the introduction of a set of recurrence relations. A fairly common recurrence scheme that is used, for instance, to determine the optimal order of matrix multiplication is the following:

Let \( N \geq 0 \). Moreover, let values \( W(i,j) \) be given for \( 0 \leq i \leq j \leq N \). Then \( C(i,j) \) is defined by

\[
C(i,i) = W(i,i) \quad 0 \leq i \leq N
\]
\[
C(i,j) = \left( \min m : i < m \leq j : W(i,j) + C(i,m-1) + C(m,j) \right) \quad 0 \leq i < j \leq N
\]

Given these equations one has to determine value \( C(0,N) \).

The remainder of this paper is devoted to the derivation of a systolic computation that computes value \( C(0,N) \). The first design decision involves the number of components in the design. Note that \( C(0,N) \) depends, among others values, on \( C(1,N) \) which in turn depends on \( C(2,N) \), etc. Hence any concurrent computation for \( C(0,N) \) has time complexity \( \Omega(N) \). Since the usual sequential solution has time complexity \( O(N^3) \) we should try to speed up this computation by a factor \( N^2 \). Assuming that the amount of concurrency in any one component is bounded by a constant, we see that \( O(N^2) \) components are needed to obtain this speed-up. Hence, in view of the set of recurrence relations, we decide to design a component \( DP_{i,j} \) for each value \( C(i,j) \). Note that the definition of \( C(i,j) \) is asymmetric. In the quantified terms

\[
W(i,j) + C(i,m-1) + C(m,j)
\]

the arguments depending on \( m \) are distinct. Our first step consists of rewriting the definition of \( C(i,j) \) into a more symmetric formula. Let \( 0 \leq i < j \leq N \) and \( h = j - i \div 2 \). Then

\[
C(i,j) = \begin{cases}
W(i,i) & 0 \leq i \leq N \\
\min m : i < m \leq j : W(i,j) + C(i,m-1) + C(m,j) & 0 \leq i < j \leq N
\end{cases}
\]
= \{ \text{introduction of an additional quantification} \}

\[
\begin{align*}
\min & \quad (\text{MIN } m : i \leq m < j : W(i,j) + C(i,m) + C(m+1,j)) \\
\min & \quad (\text{MIN } m : i < m \leq j : W(i,j) + C(i,m-1) + C(m,j))
\end{align*}
\]

= \{ \text{reducing overlap; } h = (j-i) \text{ div } 2 \text{ so } i+h \leq j-h \}

\[
\begin{align*}
\min & \quad (\text{MIN } m : i \leq m < j-h : W(i,j) + C(i,m) + C(m+1,j)) \\
\min & \quad (\text{MIN } m : i+h \leq m \leq j : W(i,j) + C(i,m-1) + C(m,j))
\end{align*}
\]

Since \(i < j\) both quantifications have a non-empty range, hence we can split off a term from both of them. In order to maintain the symmetry we can choose to split off either the pair \((m, m) = (i, j)\) or the pair \((m, m) = (j-h-1, i+h+1)\). Choosing \(m = i\) yields the term \(W(i,j) + C(i,i) + C(i+1,j)\). Choosing \(m = j-h-1\) yields the term \(W(i,j) + C(i,j-h-1) + C(j-h,j)\). We have already seen that the computation of \(C(0,N)\) has time complexity \(\Omega(N)\). In general the computation of \(C(i,j)\) has time complexity \(\Omega(j-i)\). Therefore, we prefer the pair \((m, m) = (j-h-1, i+h+1)\), since this pair offers better prospects with respect to time complexity. Splitting off \(k\) terms, \(1 \leq k \leq j-i-h\), yields the equality:

\[
C(i,j) = M(i,j,k) \min \left( \text{MIN } m : i \leq m < j-k : W(i,j) + C(i,m) + C(m+1,j) \right) \\
\min \left( \text{MIN } m : i+h+k \leq m : W(i,j) + C(i,m-1) + C(m,j) \right)
\]

where \(M(i,j,k)\) is the minimum of the terms that have been split off, and is given by

\[
M(i,j,k) = \min \left( \text{MIN } m : j-k \leq m < j : W(i,j) + C(i,m) + C(m+1,j) \right) \\
\min \left( \text{MIN } m : i+h \leq m : W(i,j) + C(i,m-1) + C(m,j) \right)
\]

Note that \(M(i,j,j-i-h) = C(i,j)\). Hence, let component \(DP_{i,j}\) compute \(C(i,j)\) using the recurrence relation for \(M\) given by

\[
M(i,j,1) = \min \left( W(i,j) + C(i,j-h-1) + C(j-h,j) \right) \\
W(i,j) + C(i,i+h) + C(i+h+1,j)
\]
\[ M(i,j,k+1) = \min_{i \leq k < j-i-h} \begin{cases} M(i,j,k) + W(i,j) + C(i,j-h-k-1) + C(j-h-k,j) \\ W(i,j) + C(i,i+h+k) + C(i+h+k+1,j) \end{cases} \quad 1 \leq k < j-i-h \]

In the derivation given below the following properties of \( h \) turn out to be useful. Recall that \( h \) is the name of the expression \((j-i) \div 2\).

\[(3.1) \quad \text{If } (j-i) \mod 2 = 0 \text{ then } i+h = (i+j)/2 = j-h \quad \text{and} \quad h_{i+1} = h_{j+1}. \]
\[(3.2) \quad \text{If } (j-i) \mod 2 = 1 \text{ then } i+h = (i+j-1)/2 = j-h-1 \quad \text{and} \quad h_{i+1} = h_{j+1} = h_{j+1}. \]

From the recurrence relation for \( M \) we observe that the computation of \( M(i,j,k+1) \) from \( M(i,j,k) \) requires four C-values. These values should be communicated to \( DP_{i,j} \) by its environment. Hence, component \( DP_{i,j} \) will be provided with four input streams, viz. \( a? \), \( b? \), \( d? \) and \( e? \). The \( k \)-th communication events, \( 0 \leq k < j-i-h \), of these streams should satisfy

\[
\begin{align*}
    a?(k) &= C(i,j-h-k-1) \\
    d?(k) &= C(i+h+k+1,j) \\
    b?(k) &= C(i,i+h+k) \\
    e?(k) &= C(j-h-k,j)
\end{align*}
\]

Note that although we designed component \( DP_{i,j} \) to produce just \( C(i,j) \), it is now able to produce the values \( C(i,m) \), \( i \leq m < j \), which it has received via \( a? \) and \( b? \), and the values \( C(m,j) \), \( i < m \leq j \), which it has received via \( d? \) and \( e? \), as well. We shall see in a moment that it is indeed required to do so. Returning our attention to the inputs of component \( DP_{i,j} \), we have to determine which of the other components \( DP_{m,n} \) should provide these inputs. On account of symmetry (i.e. interchanging the arguments and replacing \( i+x \) by \( j-x \)) we restrict our attention to the \( a? \)- and \( b? \)-values. Note that \( b?(j-i-h-1) = C(i,j-1) \). From all the components that are able to produce this value component \( DP_{i,j-1} \) will be the one that produces this value at first, since it is the component that actually computes \( C(i,j-1) \). Moreover, by the remark made above, it is able to produce the other inputs as well.
Consequently, component $DP_{i,j}$ has to produce two output streams $a!$ and $b!$ which it sends to component $DP_{i,j+1}$. The $k$-th communications, $0 \leq k < j+1-i-h_{j+1}$, of these streams should satisfy i/o-relations

$$a!(k) = C(i,j+1-h_{j+1} -k-1)$$
$$b!(k) = C(i,i+h_{j+1} +k)$$

Consider the case $(j-i) \mod 2 = 0$. Then by (3.1)

$$a!(k) = C(i,j-h-k)$$
$$b!(k) = C(i,i+h+k) \quad 0 \leq k \leq j-i-h$$

Therefore $a!(k)$ and $b!(k)$ satisfy the equalities

$$a!(0) = C(i,j-h)$$
$$a!(k+1) = C(i,j-h-k-1) = a?!(k) \quad 0 \leq k < j-i-h$$
$$b!(k) = b?!(k) \quad 0 \leq k < j-i-h$$
$$b!(j-i-h) = C(i,j)$$

Hence, there are two values that cannot be obtained directly from the corresponding inputs. Value $C(i,j)$ is no problem since this is exactly the value $M(i,j,j-i-h)$ that has to be computed by component $DP_{i,j}$. Value $C(i,j-h)$ is no problem either, since by (3.1) $C(i,j-h) = C(i,i+h)$ and therefore $a!(0) = b?!(0)$. In the case that $DP_{i,j}$ receives no inputs, i.e. $i = j$, we find that $a!(0) = b!(0) = W(i,j)$.

Next consider the case $(j-i) \mod 2 = 1$. Then by (3.2)

$$a!(k) = C(i,j-h-k-1)$$
$$b!(k) = C(i,i+h+k+1) \quad 0 \leq k < j-i-h$$

Therefore $a!(k)$ and $b!(k)$ satisfy the equalities

$$a!(k) = a?!(k) \quad 0 \leq k < j-i-h$$
$$b!(k) = b?!(k+1) \quad 0 \leq k < j-i-h-1$$
$$b!(j-i-h-1) = C(i,j)$$
Finally, we reformulate the recurrence equation for $M$ in terms of stream events:

\[
M(i,j,1) = (W(i,j) + a?(0) + e?(0)) \min (W(i,j) + b?(0) + d?(0))
\]

\[
M(i,j,k+1) = M(i,j,k) \min (W(i,j) + a?(k) + e?(k)) \min (W(i,j) + b?(k) + d?(k)) \quad 1 \leq k < j-i-h
\]

5. Communication behavior

In this section we construct a communication behavior for $DP_{i,j}$ that is consistent with the equalities derived in the previous section. It turns out that the most economical communication behavior, i.e. the one that requires only one variable per pair of corresponding streams, viz. $(a?, a!) \ldots, (e?, e!)$, yields a solution with quadratic time complexity. Introducing one additional variable for each of the pairs $(a?, a!)$ and $(e?, e!)$ turns out to be sufficient to obtain a linear solution.

Since the computation of $M(i,j,k)$ enforces some synchronization of communications we will model these computations by a sequence of internal actions $\tau(k)$ for $0 \leq k < j-i-h$, where $\tau(k)$ represents the computation of $M(i,j,k+1)$. In the actual program text for component $DP_{i,j}$ these internal actions $\tau(k)$ are replaced by the corresponding computations on a local variable $m$ (see section 6). From the recurrence relation for $M$ we obtain the following partial order for the internal actions:

\[
\begin{align*}
 &a?(0) \rightarrow \ldots \rightarrow a?(j-i-h-1) \\
 &\tau(0) \rightarrow \ldots \rightarrow \tau(j-i-h-1) \\
 &b?(0) \rightarrow \ldots \rightarrow b?(j-i-h-1)
\end{align*}
\]

Again we have ignored the $d?$- and $e?$-events on account of symmetry. A communication behavior that is consistent with (i.e. includes) this partial order is

\[
S_0 : (a?, b? ; \tau)^{j-i-h}
\]
Some further partial orders are obtained from the equalities for $a!$ and $b!$. First we consider the case $(j-i) \mod 2 = 0$, where $i < j$. From the equalities for $a!$ we obtain the partial order

$$
\begin{align*}
& a!(0) \rightarrow a!(1) \rightarrow \cdots \rightarrow a!(j-i-h) \\
& b?(0) \rightarrow a?(0) \rightarrow \cdots \rightarrow a?(j-i-h-1)
\end{align*}
$$

This partial order is included in the union of the partial orders of the following communication behaviors:

$$
S_1: \quad b?; b?j+i-h-1, a!j+i-h+1
$$

$$
S_2: \quad (a!; a?)j+i-h; a!
$$

The equalities for $b!$ yield the partial order

$$
\begin{align*}
& b!(0) \rightarrow \cdots \rightarrow b!(j-i-h-1) \rightarrow b!(j-i-h) \\
& b?(0) \rightarrow \cdots \rightarrow b?(j-i-h-1) \rightarrow \tau(j-i-h-1)
\end{align*}
$$

Or in terms of communication behaviors:

$$
S_3: \quad (b?; b!)j+i-h; b!
$$

$$
S_4: \quad b!j+i-h, \tau j+i-h; b!
$$

In the case that $(j-i) \mod 2 = 1$ we find the partial orders

$$
\begin{align*}
& a!(0) \rightarrow \cdots \rightarrow a!(j-i-h-1) \\
& a?(0) \rightarrow \cdots \rightarrow a?(j-i-h-1)
\end{align*}
$$

and

$$
\begin{align*}
& b!(0) \rightarrow \cdots \rightarrow b!(j-i-h-2) \rightarrow b!(j-i-h-1) \\
& b?(1) \rightarrow \cdots \rightarrow b?(j-i-h-1) \rightarrow \tau(j-i-h-1)
\end{align*}
$$

Or in terms of communication behaviors:
The communication behaviors above have been chosen in such a way that there is no need for buffering within the component. To be precise, we have included the pairs \( a!(k) \xrightarrow{S_2} a?(k) \), \( b!(k) \xrightarrow{S_3} b?(k+1) \), \( a!(k) \xrightarrow{S_5} a?(k+1) \), and \( b!(k) \xrightarrow{S_6} b?(k+1) \) in the communication behaviors. We will now show that this leads to a quadratic solution, i.e. every family of matching sequence functions \( \sigma_{i,j} \), where \( \sigma_{i,j} \) is a sequence function for \( DP_{i,j} \), has the property that \( \sigma_{i,j}(b!,(j-i) \text{ div } 2) \in \Omega((j-i)^2) \). By a matching family of sequence functions \( \sigma_{i,j} \) we mean that

\[
\sigma_{i,j}(a!,k) = \sigma_{i,j}(a?,k) \quad \sigma_{i,j}(d!,k) = \sigma_{i,j}(d?,k)
\]

\[
\sigma_{i,j}(b!,k) = \sigma_{i,j}(b?,k) \quad \sigma_{i,j}(e!,k) = \sigma_{i,j}(e?,k)
\]

Note that these equalities reflect that output stream \( a! \) of component \( DP_{i,j-1} \) is equal to input stream \( a? \) of component \( DP_{i,j} \), etc.. For \( i < j \) and \( j-i \text{ mod } 2 = 0 \) we derive

\[
\sigma_{i,j}(b!,k+1) - \sigma_{i,j}(a!,k) \geq \{ \sigma_{i,j} \text{ respects } S_2 \text{, hence } \sigma_{i,j}(a?,k) \geq \sigma_{i,j}(a!,k) + 1 \} \text{, and}
\]

\[
\sigma_{i,j}(b!,k+1) - \sigma_{i,j}(b?,k+1) \geq \{ \sigma_{i,j} \text{ respects } S_3 \text{, hence } \sigma_{i,j}(b!,k+1) \geq \sigma_{i,j}(b?,k+1) + 1 \}
\]

\[
\sigma_{i,j}(b!,k+1) - \sigma_{i,j}(a?,k) + 2 \geq \{ \sigma_{i,j} \text{ respects } S_5 \text{ and } S_6 \}
\]

\[
\sigma_{i,j-1}(b?,k+2) - \sigma_{i,j-1}(a?,k+1) + 4 = \{ \sigma_{i,j-1} \text{ and } \sigma_{i,j-2} \text{ match} \}
\]

\[
\sigma_{i,j-2}(b!,k+2) - \sigma_{i,j-2}(a!,k+1) + 4
\]

Thus it follows that for \( 0 \leq 2m \leq h - (k+1) \)
\[ \sigma_{i,j}(b!,k+1) - \sigma_{i,j}(a!,k) \]
\[ \geq \{ \text{applying the above inequality \( m \) times, } k+m+1 \leq j-2m-i-h_{j-2m} \} \]
\[ \sigma_{i,j}(b!,k+m+1) - \sigma_{i,j}(a!,k+m) + 4m \]
\[ \geq \{ \sigma_{i,j} \text{ respects } S_2, S_0 \text{ and } S_3 \} \]
\[ 4m \]

Choosing \( m \) as large as possible, i.e. \( 2m = h - (k+1) \), we find
\[ \sigma_{i,j}(b!,k+1) - \sigma_{i,j}(a!,k) \geq j - i - 2(k+1) \]
\[ 0 \leq k < (j-i) \div 2 \]

Hence
\[ \sigma_{i,j}(b!, (j-i) \div 2) \]
\[ \geq \sigma_{i,j}(b!, (j-i) \div 2) - \sigma_{i,j}(b!,0) \]
\[ \geq (S \ k : 0 \leq k < (j-i) \div 4 : \sigma_{i,j}(b!,2k + 2) - \sigma_{i,j}(b!,2k) ) \]
\[ \geq (S \ k : 0 \leq k < (j-i) \div 4 : \sigma_{i,j}(a!,2k + 2) - \sigma_{i,j}(a!,2k + 1) ) \]
\[ \geq (S \ k : 0 \leq k < (j-i) \div 4 : j - i - 4k ) \]

This proves that \( \sigma_{i,j}(b!, (j-i) \div 2) \in \Omega((j-i)^2) \).

A careful analysis of the derivation given above reveals that of the four communication behaviors \( S_2, S_3, S_5 \) and \( S_6 \) involved, \( S_2 \) and \( S_5 \) are responsible for this quadratic lower bound. For instance, the inequality \( \sigma_{i,j}(a?,k) - \sigma_{i,j}(a!,k) \geq 1 \) is a consequence of \( a!(k) \xrightarrow{S_2} a?(k) \), but this particular ordering of events is not required by the equalities for \( a! \). The inequality \( \sigma_{i,j}(b!,k) - \sigma_{i,j}(b?,k) \geq 1 \) on the other hand is a consequence of \( b?(k) \xrightarrow{S_1} b!(k) \), and this ordering of events is required by the equalities for \( b! \).

Interchanging the order of the events \( a?(k) \) and \( a!(k) \) in \( S_2 \) removes the constraint \( \sigma_{i,j}(a?,k) - \sigma_{i,j}(a!,k) \geq 1 \). This is an application of the general rule that inputs may be
advanced and outputs may be postponed, since this does not disturb causal dependencies between events. For reasons of symmetry we will interchange the order of these events in $S_5$ as well. Thus we obtain

$$S_2': \quad (a? ; a!)^{j-i-h} ; a!$$

$$S_5': \quad a? ; (a? ; a!)^{j-i-h-1} ; a!$$

It is not hard to show that it is not sufficient to change just $S_2$ or $S_5$, but we leave this as an exercise for the reader.

Combining $S_0$, $S_1$, $S_2'$, $S_3$, and $S_4$ when $(j-i) \mod 2 = 0$ yields communication behavior

$$S_{\text{even}}: \quad (a? , b? ; a! , b! , \tau)^{j-i-h} ; a! , b!$$

Similarly, combining $S_0$, $S_5'$, $S_6$, and $S_7$ when $(j-i) \mod 2 = 1$ yields communication behavior

$$S_{\text{odd}}: \quad a? , b? ; \tau ; (a? , b? ; a! , b! , \tau)^{j-i-h-1} ; a! , b!$$

6. Components

Extending the communication behaviors $S_{\text{even}}$ and $S_{\text{odd}}$ derived in the previous section in an appropriate way with variables yields the desired components $DP_{i,j}$. We distinguish three cases.

Case 1: $0 \leq i = j \leq N$.

```plaintext
com DP_{i,j} (a!\text{int}, b!\text{int}, c!\text{int}, d!\text{int}):
    | [\text{con } w = W(i,j)]
    | a!w, b!w, c!w, d!w
    |
moc
```
Case 2: $0 \leq i < j \leq N$ and $(j-i) \mod 2 = 0$.

```plaintext
com DP_{i,j} (a?int, b?int, d?int, e?int, a!int, b!int, d!int, e!int):

| [con h, w = \((j-i) \div 2\), W(i,j)]
| [var xa, xb, xd, xe, ya, yd, m : int]
| a?ya, b?xb, d?yd, e?xe
| a!xb, b!xb, d!xe, e!xe
| m := (w + ya + xe) \min (w + xb + yd)
| (a?xa, b?xb, d?xd, e?xe

  \{ xa = a?(k) \land xb = b?(k) \land xd = d?(k) \land xe = e?(k)
  \land ya = a?(k-1) \land yd = d?(k-1) \land m = M(i,j,k) \land 1 \leq k < j-i-h \}
| a!ya, b!xb, d!yd, e!xe
| m, ya, yd := m \min (w + xa + xe) \min (w + xb + xd), xa, xd
|)
| j\leftarrow h-1
| a!ya, b!m, d!yd, e!m

]}
```

The command of the component given above consists of three parts. In the initial part the first communication events of each stream are dealt with. It is this part that distinguishes the case $(j-i) \mod 2 = 0$ from the case $(j-i) \mod 2 = 1$ given below. The second part is a repetition which constitutes the main part of the command. Between curly brackets we have added an assertion whose validity may help the reader to see that this component realizes the specified i/o-relations. The final part consists of the last output events in which, via streams b! and e!, the computed value $C(i,j)$ is communicated. Note that although according to $S_{\text{even}}$ assignments to $m$ may occur concurrently with output events, component $DP_{i,j}$ performs the output events before the assignments to $m$. This does not deteriorate the performance,
since assignments to $m$ are now concurrent with assignments to $ya$ and $yd$ and these assignments must occur after the output events anyway. Furthermore, note that the component requires 7 variables. In particular there is an additional variable both for the $a$- and the $d$-communications.

Case 3: $0 \leq i < j \leq N$ and $(j-i) \mod 2 = 1$.

```plaintext
com DP_{i,j} (a?int, b?int, d?int, e?int, a!int, b!int, d!int, e!int):
    | [con h, w = (j-i) div 2, W(i,j)]
    | var xa, xb, xd, xe, ya, yd, m : int
    | a?ya, b?xb, d?yd, e?xe
    ; m := (w + ya + xe) min (w + xb + yd)
    ; ( a?xa, b?xb, d?xd, e?xe
        { xa = a?(k) \land xb = b?(k) \land xd = d?(k) \land xe = e?(k)
            \land ya = a?(k-1) \land yd = d?(k-1) \land m = M(i,j,k) \land 1 \leq k < j-i-h }
    ; a!ya, b!xb,d!yd, e!xe
    ; m, ya, yd := m min (w + xa + xe) min (w + xb + xd), xa, xd
    )_{j-i-h-1}
    ; a!ya, b!m, d!yd, e!m
```

**moc**

We end this section with the exhibition of a family of matching sequence functions $\sigma_{i,j}$, $0 \leq i \leq j \leq N$, for the components given above. These sequence functions are such that

$$
\sigma_{i,j}(a!,k) = \sigma_{i,j}(b!,k) = \sigma_{i,j}(d!,k) = \sigma_{i,j}(e!,k)
$$

$$
\sigma_{i,j}(a?,k) = \sigma_{i,j}(b?,k) = \sigma_{i,j}(d?,k) = \sigma_{i,j}(e?,k)
$$

for all relevant $k$. For $(j-i) \mod 2 = 0$ we find
\[
\sigma_{i,j}(a!,k) = 5((j-i) \text{ div } 2) + 3k - 1 \\
\sigma_{i,j}(a?,k) = 5((j-i) \text{ div } 2) + 3k \\
\sigma_{i,j}(\tau,k) = 5((j-i) \text{ div } 2) + 3k + 1
\]

Note that \(\sigma_{i,j}\) is not defined for \(a?\) and \(\tau\) when \(i = j\), as indeed should be the case. For 
\((j-i) \mod 2 = 1\) we find

\[
\sigma_{i,j}(a?,k) = 5((j-i) \text{ div } 2) + 3k \\
\sigma_{i,j}(\tau,k) = 5((j-i) \text{ div } 2) + 3k + 2 \\
\sigma_{i,j}(a!,k) = 5((j-i) \text{ div } 2) + 3k + 4
\]

One easily verifies that for all \(i, j, 0 \leq i < j \leq N\), and for all \(k, 0 \leq k \leq (j-i-1) \text{ div } 2\)

\[
\sigma_{i,j}(a?,k) = \sigma_{i,j-1}(a!,k) \\
\sigma_{i,j}(d?,k) = \sigma_{i+1,j}(d!,k)
\]

Therefore, the functions \(\sigma_{i,j}\) indeed constitute a family of matching sequence functions.

Hence, according to \([Zw]\) our systolic computation is free from deadlock. Since all components perform a finite computation there is no divergence either. Moreover, we have obtained an optimal solution since \(\sigma_{i,j}(a!, (j-i) \text{ div } 2) = 4(j-i) \in O(j-i)\). Recall that \(\sigma_{i,j}(a!, (j-i) \text{ div } 2) = C(i,j)\) and in section 4 we have shown that any systolic computation of \(C(i,j)\) requires linear time.

7. Combining components

In this section we show that four neighboring components can be combined into one component, thereby reducing the number of cells to \(N^2/8\), without adversely affecting the computational complexity. There are several benefits in combining components. The amount of communication channels can be reduced considerably and there is also a slight reduction in the number of variables. On a VLSI-chip communication channels consist of bundles of wires and variables consist of sets of flip-flops (registers). The number of wires in a bundle and the
number of flip-flops per register depend on the wordlength of the integers. Together, wires and flipflops are responsible for a considerable part of the chip area. Therefore, it is worthwhile to investigate this optimization.

Assume that \( N \mod 2 = 1 \) and let components \( p : DP_{2i+1,2j+1} \), \( q : DP_{2i,2j} \), \( r : DP_{2i+1,2j+1} \) be given for \( 0 \leq i \leq j \leq N \div 2 \). Moreover, let \( s : DP_{2j,2j} \) be given for \( 0 \leq i < j \leq N \div 2 \). We denote the conglomerate of these four components by a new component \( DDP_{i,j} \). We first show how the number of communication channels can be reduced. From the sequence functions of the previous section we observe that \( \sigma_{2i+1,2j+1}(r.a!,k) \mod 3 = (5(j-i) + 3k + 4) \mod 3 \neq (5(j-i) + 3k) \mod 3 = \sigma_{2i+1,2j+1}(p.a!,k) \mod 3 \). Hence, these communications can be interleaved in a single output stream \( a! \) of component \( DDP_{i,j} \). A similar argument can be given for the pairs \( (r.b!,p.b!), (r.d!,q.d!) \) and \( (r.e!,q.e!) \). Moreover, since \( r.a? = q.a! \), a matching pair of statements consisting of input statement \( r.a? \) of \( r \) and output statement \( q.a!E \) of \( q \), can be replaced by an internal event \( v := E \) of \( DDP_{i,j} \). All in all, this reduces the number of channels by a factor of 4. In addition to the reduction of channels, the number of variables can be reduced from 28 (= 4×7) to 24. In particular, each of the variable pairs \( (rxb,qxb) \), \( (rxe,pxe) \), \( (pxb,sxb) \) and \( (qxe,sxe) \), where \( rxb \) indicates variable \( xb \) of component \( r \) etc., can be replaced by a single variable. This requires, however, a rescheduling of some events. Consider the pair \( (rxb,qxb) \). To start with, we recall the scheduling of the events of \( q \) and \( r \) that involve these variables. For \( 0 \leq k < j-i-1 \) we have:

\[
\begin{align*}
\sigma_{2i+2j}(q.b?,k+1) &= 5(j-i-1) + 3k + 7 = \sigma_{2i+2j}(r.\tau,k) \\
\sigma_{2i+2j}(q.b!,k+1) &= 5(j-i-1) + 3k + 8 = \sigma_{2i+2j}(r.b?,k+1) \\
\sigma_{2i+2j}(q.\tau,k+1) &= 5(j-i-1) + 3k + 9 = \sigma_{2i+2j}(r.b!,k)
\end{align*}
\]

Collecting the corresponding statements from the commands of \( q \) and \( r \) into a single (piece of) command for \( DDP_{i,j} \) yields:
Here we have already incorporated the modification of channels mentioned above. It is not hard to see that all communication events can be scheduled one unit later without invalidating the computation of either qm or rm. Recall that rxb := qxb represents a pair of matching communication events and has to be rescheduled as well. As a result we obtain:

\[
\begin{align*}
    b?qxb, \quad rm := rm & \min (rw + rxa + rxe) \min (rw + rxb + rxd) \\
    ; \quad rxb := qxb \\
    ; \quad b!rxb, \quad qm := qm & \min (qw + qxa + qxe) \min (qw + qxb + qxd)
\end{align*}
\]

It is now easy to see that we may replace the variables qxb and rxb by a single variable, say rqxb. Thus we obtain:

\[
\begin{align*}
    b!rqxb, \quad rm := rm & \min (rw + rxa + rxe) \min (rw + rqxb + rxd) \\
    ; \quad b?qxb \\
    ; \quad rxb, \quad qm := qxb, qm & \min (qw + qxa + qxe) \min (qw + rqxb + qxd)
\end{align*}
\]

As a result of combining components in this fashion the distinction between the odd and the even cases of the previous section has disappeared. Instead, there are slightly more boundary cases. In the previous section there was only one boundary case, viz. the case \( i = j \), where there are no input streams. For a component of type DDP\(_{i,j}\) there are several boundary cases. This has to do with the three stages in the execution of a component of type DP\(_{i,j}\). For instance if \( i+1 = j \), then component \( s : DP_{2i+1,2j} \) has finished before component \( r : DP_{2i+2j+1} \) has started, and before components \( p : DP_{2i+2j+1} \), and \( q : DP_{2i+2j} \) have started their repetition. From \( i+3 = j \) onwards there will be a moment that all four components execute their repetition. Hence, for \( 3 \leq j - i \) all components DDP\(_{i,j}\) are similar. We end this section with
the program text of \( DDP_{i,j} \) for this general case and leave the boundary cases as an exercise for the reader.

Let \( 0 \leq i \leq j \leq N \div 2 \) and \( 3 \leq j-i \).

\[
\text{com } DDP_{i,j}(a?\text{int}, b?\text{int}, d?\text{int}, e?\text{int}, a?\text{lint}, b?\text{lint}, d?\text{lint}, e?\text{lint}) :
\]
\[
\begin{align*}
\text{[con } \& \text{ pw, qw, rw, sw } &= W(2i+1,2j+1), W(2i,2j), W(2i,2j+1), W(2i+1,2j) \\
\text{[var } \& \text{ pm, qm, rm, sm, rqxb, psxb, rpxe, qsxe, } & \\
\text{ pya, qya, rya, sya, pxa, qxa, rxa, sx, sya, } & \\
\text{ pyd, qyd, ryd, syd, px, qx, rx, sx, } & \\
\text{ pya := sya, qyd := syd } & \\
\text{ ; b?psxb, e?qsxe } & \\
\text{ ; sm := (sw + sya + qsxe) min (sw + psxb + syd) } & \\
\text{ ; a?sxa, d?sx } & \\
\text{ ; (a?qya, b?psxb, d?pyd, e?qsxe } & \\
\text{ ,pya := sya, qyd := syd } & \\
\text{ ) } & \\
\text{ ; (a!psxb, b!rqxb, d!qsxe, e!rpxe } & \\
\text{ ,sya := sxa, syd := sx } & \\
\text{ ,sm := sm min (sw + sxa + qsxe) min (sw + psxb + sx) } & \\
\text{ ) } & \\
\text{ ; (a?sxa, b!psxb, d?sx, e!qsxe } & \\
\text{ ,pm := (pw + pya + rpxe) min (pw + psxb + pyd) } & \\
\text{ ,qm := (qw + qya + qsxe) min (qw + rqxb + qyd) } & \\
\text{ ) } & \\
\text{ ; (a?qxa, b!rqxb, d?pxd, e!rpxe } & \\
\text{ ,pxa := sya, qxd := syd } & \\
\end{align*}
\]

19
,rm := (rw + rya + rpxe) min (rw + rqxb + ryd) 
)

; (a!pya, b?rqxb, d!qyd, e?rpxe
  ,rxa := qya, sya := sxa, rxd := pyd, syd := sxd
  ,sm := sm min (sw + sxa + qsxe) min (sw + psxb + sxd)
)

; (a!rya, a?sxa, b!rqxb, d!ryd, d?sxd, e!rpxe
  ,pm := pm min (pw + pxa + rpxe) min (pw + psxb + pxd)
  ,qm := qm min (qw + qxa + qsxe) min (qw + rqxb + qxd)
  ,pya := pxa, qya := qxa, pyd := pxd, qyd := qxd
)

; (a?qxa, b!rqxb, d?psxb, d?qxd, e!rpxe, e?qsxe
  ,pxa := sya, rya := rxa, qxd := syd, ryd := rxd
  ,rm := rm min (rw + rxa + rpxe) min (rw + rqxb + rxm)
)

; (a!pya, b?rqxb, d!qyd, e?rpxe
  ,rxa := qya, sya := sxa, rxd := pyd, syd := sxd
  ,sm := sm min (sw + sxa + qsxe) min (sw + psxb + sxd)
)

; (a!rya, b!rqxb, d!ryd, e!rpxe
  ,pm := pm min (pw + pxa + rpxe) min (pw + psxb + pxd)
  ,qm := qm min (qw + qxa + qsxe) min (qw + rqxb + qxd)
  ,pya := pxa, qya := qxa, pyd := pxd, qyd := qxd
)

; (a?qxa, b!rqxb, d?pxd, e!rpxe
  ,psxb := sm, qsxe := sm
\texttt{pxa := sya , rya := rxa , qx := syd , ryd := rxd} \\
\texttt{.rm := \texttt{rm \texttt{min (rw + rxa + rpxe) min (rw + rqxb + rxd)}}; (\texttt{a!pya , d!qyd})} \\
\texttt{,rxa := qya , rxd := pyd; (\texttt{a!rya , b!rqxb , d!ryd , e!rpxe})} \\
\texttt{.pm := pm \texttt{min (pw + pxa + rpxe) min (pw + psxb + px)}; (\texttt{a!pya , d!qyd})} \\
\texttt{,rqxb := qm , rxd := pyd , rpxe := pm; (\texttt{a!rya , b!pm , d!ryd , e!qm})} \\
\texttt{(b!rqxb , e!rpxe)} \\
\texttt{; (a!rya , d!ryd)} \\
\texttt{; b!rm , c!rm} \\
\texttt{]} \\
\texttt{moc}
8. Conclusions

We have shown that a systolic computation for dynamic programming can be formally derived. A similar result is obtained in [Che] but our derivation is much simpler. It is not clear if this is a consequence of the fact that we have tackled the problem as a single specific programming task, whereas [Che] aims at a more general synthesis method. Anyway, it seems to us that we have taken substantially fewer design decisions to obtain the same result.

Our formal approach has resulted in a far better understanding of existing systolic computations for dynamic programming. For instance, in [GuKuTh] communication between neighboring cells are realized via a single "wire". Each wire is split into a "fast belt" and a "slow belt" and it is described when the cells put values on these belts. In our approach the status of these belts becomes clear, the fast belts stand for the b- and e-streams, whereas the slow belts stand for the a- and d-streams. Moreover, in [GuKuTh] it is claimed that each cell has only five registers. The registers that cater for the values of the slow belts, however, have two stages. This phenomenon is also much better understood in terms of our local variables. As we have shown two variables per slow belt are not only sufficient but also necessary to obtain a linear time complexity. Similar observations are made in [PuSu], where an a posteriori proof of the Guibas-Kung-Thompson design is given.

In [LoTch] it is shown as a novelty that the number of processors required can be reduced to $N^2/8$. This requires, however, a very intricate scheduling. We have shown that the same reduction can be obtained if we combine four neighboring cells into one new cell in a rather straightforward manner.

In [Ry] it is argued on theoretical grounds that it should be possible to solve dynamic programming problems in $\log^2 N$ time using $N^6$ processors on a perfect shuffle computer or a cube-connected computer. This does not contradict the linear time lower bound we have found, since both computers consist of a network of cells in which each cell has a number of
neighbors that is logarithmic in the total number of cells. It remains an interesting task to formally derive a systolic computation for dynamic programming in those cases as well.

Finally, we like to stress that in our solution all synchronization is done by message passing. As a consequence it is not necessary, albeit not forbidden, that all cells of the network operate in lock-step. This is in contrast with the systolic array of [GuKuTh] where all array elements are required to operate synchronously. Hence, our systolic computation has the advantage that it can be realized as a delay-insensitive circuit [Eb].

Acknowledgements.

We would like to thank Anne Kaldewaij and Martin Rem for their comments on an earlier version of this paper.

References.


In this series appeared:

<table>
<thead>
<tr>
<th>No.</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>85/01</td>
<td>R.H. Mak</td>
<td>The formal specification and derivation of CMOS-circuits</td>
</tr>
<tr>
<td>85/02</td>
<td>W.M.C.J. van Overveld</td>
<td>On arithmetic operations with M-out-of-N-codes</td>
</tr>
<tr>
<td>85/03</td>
<td>W.J.M. Lemmens</td>
<td>Use of a computer for evaluation of flow films</td>
</tr>
<tr>
<td>85/04</td>
<td>T. Verhoeff, H.M.J.L. Schols</td>
<td>Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate</td>
</tr>
<tr>
<td>86/01</td>
<td>R. Koymans</td>
<td>Specifying message passing and real-time systems</td>
</tr>
<tr>
<td>86/02</td>
<td>G.A. Bussing, K.M. van Hee, M. Voorhoeve</td>
<td>ELISA, A language for formal specifications of information systems</td>
</tr>
<tr>
<td>86/03</td>
<td>Rob Hoogerwoord</td>
<td>Some reflections on the implementation of trace structures</td>
</tr>
<tr>
<td>86/04</td>
<td>G.J. Houben, J. Paredaens, K.M. van Hee</td>
<td>The partition of an information system in several parallel systems</td>
</tr>
<tr>
<td>86/05</td>
<td>Jan L.G. Dietz, Kees M. van Hee</td>
<td>A framework for the conceptual modeling of discrete dynamic systems</td>
</tr>
<tr>
<td>86/06</td>
<td>Tom Verhoeff</td>
<td>Nondeterminism and divergence created by concealment in CSP</td>
</tr>
<tr>
<td>86/07</td>
<td>R. Gerth, L. Shira</td>
<td>On proving communication closedness of distributed layers</td>
</tr>
<tr>
<td>86/09</td>
<td>C. Huizing, R. Gerth, W.P. de Roever</td>
<td>Full abstraction of a real-time denotational semantics for an OCCAM-like language</td>
</tr>
<tr>
<td>86/10</td>
<td>J. Hooman</td>
<td>A compositional proof theory for real-time distributed message passing</td>
</tr>
<tr>
<td>86/11</td>
<td>W.P. de Roever</td>
<td>Questions to Robin Milner - A responder's commentary (IFIP86)</td>
</tr>
<tr>
<td>86/12</td>
<td>A. Boucher, R. Gerth</td>
<td>A timed failures model for extended communicating processes</td>
</tr>
<tr>
<td>Year</td>
<td>Author(s)</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>86/14</td>
<td>R. Koymans</td>
<td>Specifying passing systems requires extending temporal logic</td>
</tr>
<tr>
<td>87/01</td>
<td>R. Gerth</td>
<td>On the existence of sound and complete axiomatizations of the monitor concept</td>
</tr>
<tr>
<td>87/02</td>
<td>Simon J. Klaver, Chris F.M. Verberne</td>
<td>Federatieve Databases</td>
</tr>
<tr>
<td>87/03</td>
<td>G.J. Houben, J.Paredaens</td>
<td>A formal approach to distributed information systems</td>
</tr>
<tr>
<td>87/04</td>
<td>T. Verhoeff</td>
<td>Delay-insensitive codes - An overview</td>
</tr>
<tr>
<td>87/05</td>
<td>R. Kuiper</td>
<td>Enforcing non-determinism via linear time temporal logic specification.</td>
</tr>
<tr>
<td>87/06</td>
<td>R. Koymans</td>
<td>Temporele logica specificatie van message passing en real-time systemen (in Dutch).</td>
</tr>
<tr>
<td>87/07</td>
<td>R. Koymans</td>
<td>Specifying message passing and real-time systems with real-time temporal logic.</td>
</tr>
<tr>
<td>87/08</td>
<td>H.M.J.L. Schols</td>
<td>The maximum number of states after projection.</td>
</tr>
<tr>
<td>87/10</td>
<td>T. Verhoeff</td>
<td>Three families of maximally nondeterministic automata.</td>
</tr>
<tr>
<td>87/11</td>
<td>P. Lemmens</td>
<td>Eldorado ins and outs. Specifications of a data base management toolkit according to the functional model.</td>
</tr>
<tr>
<td>87/12</td>
<td>K.M. van Hee and A. Lapinski</td>
<td>OR and AI approaches to decision support systems.</td>
</tr>
<tr>
<td>87/13</td>
<td>J.C.S.P. van der Woude</td>
<td>Playing with patterns, searching for strings.</td>
</tr>
<tr>
<td>87/14</td>
<td>J. Hooman</td>
<td>A compositional proof system for an occam-like real-time language</td>
</tr>
</tbody>
</table>
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. Struijk  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.