Correctness Proofs for Systolic Algorithms: Palindromes and Sorting

L. Kossen

W.P. Weijland

Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

In designing VLSI-circuits it is very useful, if not necessary, to construct the specific circuit by placing simple components in regular configurations. Systolic systems are circuits built up from arrays of cells and therefore very suitable for formal analysis and induction methods. In two examples correctness proofs are given using bisimulation semantics with asynchronous cooperation. These examples also have been worked out by Hennessy in a setting of failure semantics with synchronous cooperation. Finally the notion of process creation is introduced and used to construct machines with unbounded capacity.

1. Introduction

In this article we will present simple descriptions of so-called *systolic systems*. Such systems can be looked at as a large integration of identical cells in such a way that the behaviour of the total system strongly resembles the behaviour of the individual cells. In fact the total system behaves like one of its individual cells 'on a larger scale'.

For example one can think of a machine sorting arrays of numbers with a certain maximum length. Suppose we need a machine handling arrays that are much longer. A typical 'systolic approach' to this problem would be to try to *interconnect* the smaller machines such that the total circuit sorts arrays of a greater length. As a matter of fact this specific example will be worked out in the following sections. In designing VLSI-circuits (short for *very large scale integrated* circuits) it is very useful, if not necessary, to construct the specific circuit by placing simple components in regular configurations ([9]). Otherwise one looses all intuition about the behaviour of the circuit that is eventually constructed. For this reason one may see systolic systems as a sort of regular subclass of VLSI-circuits which is very suitable for formal analysis. As we will see from two typical examples from Kung [8] these regular circuits can easily be analyzed as to their correct behaviour.

In designing a systolic system, finding a correct definition of the individual cells turns out to be the main problem. Apparently we already have in mind what we want the total network to do and hence we may assume there is some general *specification* of the desired behaviour. Indeed this specification may be

general in the sense that it only needs to describe the 'outside behaviour' of the machine without specifying in detail the internal actions.

On the other hand looking for a correct definition of the individual cells we are working with a much more detailed description, since all relevant actions need to be described. This means we are looking for a certain *implementation* that satisfies the general specification we had in mind.

In this article we will add an extra element to ACP_τ denoting chaos (see Brookes, Hoare and Roscoe [4] and Bergstra, Klop and Olderog [3]). One can look at this element, written as Ω, as a process which runs totally out of order without any restriction as to its behaviour. We assume that Ω does not (successfully) terminate.

There is a specific reason for introducing Ω in ACP_τ. In fact, in a specification Ω will stand for a process that is of no theoretical interest to us at the moment. Think for example of the behaviour of a computer just after memory-overflow occurs: in reasoning about the correct behaviour of the machine we do not *specify* what the machine should do after having announced its memory- overflow; the machine even may cause a deadlock instead of announcing its memory-overflow at all, since the announcement itself is already a diverging step from its specified behaviour.

So, not having specified part of its behaviour, we could say that the same specification can be implemented by many different machines. This notion '...is implemented by...' will be denoted by $\models$ in the sequel.

We will define a new relation $\models$ on processes in an algebraical setting as is shown below in Table 1. By *definition* we assume $\models$ to be reflexive, transitive and closed under contexts. Moreover we assume all general laws holding for atoms to hold for Ω as well.

In Koymans and Mulder [7] this notion has already been worked out in a semantical setting of process graphs. So far it has not been verified whether this leads to the same interpretation.

$\Omega \cdot x = \Omega$	CH1
$\Omega + x = \Omega$	CH2
$\Omega \models x$	IM1
$a \cdot \Omega \models \delta$	IM2

TABLE 1. The axioms of chaos and of implementation

Within the semantical setting of Process Algebra (see Bergstra and Klop [2]), in two specific examples we will be able to prove *correctness* of certain implementations of systolic systems with respect to these specifications. These proofs already were presented by Hennessy [6] using *synchronous* ('clocked') cooperation between cells. In the following, however, we will specify *asynchronous* versions of these examples. We therefore construct *delay-insensitive*

circuits (see Ebergen [5]), which says that the system can 'wait' for communication at its channels without starting to malfunction.

Other authors working with formal specifications to describe the behaviour of VLSI-circuits are for instance Milne [10] and Rem [11].

It turns out that ACP_τ provides us with a convenient proof system in which correctness proofs can be presented in a fairly standard way.

At this place we especially want to thank Jos Baeten who took the trouble to check this article several times and who gave so much of his support in developing its content.

2. A PALINDROME-RECOGNISER

In the following we will describe a machine which is able to recognise palindromes from strings of input symbols i.e. a machine that answers 'true' if and only if a given string of input symbols is equal to its reverse.

Suppose S is a finite set of symbols from which the input strings are built up.

The actions of sending and receiving a symbol d along a certain channel are written as $s(d)$ and $r(d)$ respectively. Moreover we have a predicate *ispal* with strings of symbols as its domain which is true if and only if its argument is a palindrome. Finally we write $|w|$ for the length of the string w.

Now we can easily write down the specification of the palindrome-recogniser PAL as is done in Table 2.

$$\mathrm{PAL}(\epsilon) = s(\mathrm{true}) \cdot \mathrm{PAL}(\epsilon) + \sum_{x \in S} r(x) \cdot s(\mathrm{true}) \cdot \mathrm{PAL}(x)$$

$$\mathrm{PAL}(w) = \sum_{x \in S} r(x) \cdot s(\mathrm{ispal}(x \cdot w)) \cdot \mathrm{PAL}(x \cdot w) \qquad (|w| > 0)$$

TABLE 2. A specification of the palindrome-recogniser PAL

The specification in Table 2 describes precisely our intuition about what a palindrome-recogniser should do.

Note that the machine PAL only receives input symbols. Since it is clear that a palindrome-recogniser should not throw away any of its received information the machine described in Table 2 needs to be able to contain arbitrarily long strings of symbols. In practice, however, machines are of a finite size. So from a more practical point of view we should first give a specification of a machine that only works on input strings with a limited length.

In Table 3 a machine PAL_k is specified working exactly like the previous palindrome-recogniser but now with a limit to the length of its input strings. For reasons to be explained later this limit is put $2k$ instead of k.

We assume our machine PAL_k has an in/output channel numbered $k+1$. So $s_{k+1}(d)$ and $r_{k+1}(d)$ will denote the actions of sending and receiving a symbol d.

$$\mathrm{PAL}_0(w)=s_1(\mathrm{true})\cdot\mathrm{PAL}_0(w)+\sum_{x\in S} r_1(x)\cdot\Omega \qquad (0\leqslant|w|)$$

$$\mathrm{PAL}_{k+1}(\epsilon)=s_{k+2}(\mathrm{true})\cdot\mathrm{PAL}_{k+1}(\epsilon)+\sum_{x\in S} r_{k+2}(x)\cdot s_{k+2}(\mathrm{true})\cdot\mathrm{PAL}_{k+1}(x)$$

$$\mathrm{PAL}_{k+1}(w)=\sum_{x\in S} r_{k+2}(x)\cdot s_{k+2}(\mathrm{ispal}(x\cdot w))\cdot\mathrm{PAL}_{k+1}(x\cdot w) \qquad (0<|w|<2(k+1))$$

$$\mathrm{PAL}_{k+1}(w)=\sum_{x\in S} r_{k+2}(x)\cdot\Omega \qquad (2(k+1)\leqslant|w|)$$

TABLE 3. A specification of PAL_k for arbitrary natural number k

The fourth equation tells us that if PAL_k has reached its maximum capacity it will turn into chaos, i.e. it will not be restricted any more as to its behaviour. Indeed if the machine has thrown away any of its input it can never react like a palindrome-recogniser with respect to this input.

We will now introduce an implementation of a palindrome-recogniser of some given size k. This means we will construct a machine implementing the specification given above.

As mentioned in the introduction this particular implementation has the look of a large integration of identical cells. As a matter of fact each cell itself is again a palindrome-recogniser of size 2. We will prove that a merge of k such cells gives us exactly a palindrome-recogniser of size $2k$.

Consider the cell pictured in Figure 1. The i-th cell C_i has two communication channels i and $i+1$. Internally C_i has three storage locations, one for boolean values and two for symbols.

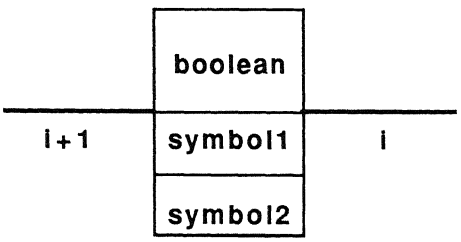

FIGURE 1. An individual cell, C_i, of the palindrome-recogniser

The cell C_i has three distinct states.

(0) In the initial state the cell carries no symbols, i.e.: carries the empty word, and since the empty word is a palindrome it can always output the boolean value true to the left. If a symbol is input from the left it is stored in the location **symbol2,** then the boolean value true is output to the left

since a word consisting of a single symbol always is a palindrome. The cell now is in state one.

(1) In state one a symbol is input from the left and a boolean from the right (in any order), and stored in the remaining locations **symbol1** and **boolean.** The cell is now in state two.

(2) In state two the cell contains two symbols **symbol1** and **symbol2** forming a word that is a palindrome iff **symbol1** = **symbol2.** Now a boolean value b is output to the left, which is calculated according to the formula

$$b = \textbf{boolean} \wedge (\textbf{symbol1} = \textbf{symbol2}).$$

Hence before deciding about its output the cell C_i *consults* messages received from the outside world. Together with this boolean output the symbol in location **symbol1** is output to the right (in any order interleaved) making room for new input symbols. The cell is now in state one once more.

In the language of ACP_τ the behaviour of the cell C_i described above can be expressed by the equations shown in Table 4. The fourth equation defines a machine called TC which stands for *terminal cell.* This terminal cell has a fairly destructive behaviour with respect to its input data since they are simply thrown away. Since TC never 'contains' any symbol (or always contains the empty string) it can always output a boolean value true and thus behaves like a palindrome-recogniser of size zero (note that the empty string is a palindrome). In the sequel we write B for the set of booleans {true, false}.

$$C_i = s_{i+1}(\text{true}) \cdot C_i + \sum_{x \in S} r_{i+1}(x) \cdot s_{i+1}(\text{true}) \cdot C'_i(x)$$

$$C'_i(x) = (\sum_{y \in S} r_{i+1}(y) \| \sum_{v \in B} r_i(v)) \cdot C''_i(x, y, v)$$

$$C''_i(x, y, v) = (s_{i+1}(|x = y| \wedge v) \| s_i(y)) \cdot C'_i(x)$$

$$\text{TC} = s_1(\text{true}) \cdot \text{TC} + \sum_{x \in S} r_1(x) \cdot \text{TC}$$

TABLE 4. Formal definition of the behaviour of an individual cell

Note that the second equation violates the scope rules of $\sum$ since y and v are bounded variables in the first term. We will nevertheless use this notation as a shorthand for the correct but much more complex term

$$\sum_{y \in S} r_{i+1}(y) \cdot (\sum_{v \in B} r_i(v) \cdot C''_i(x,y,v)) + \sum_{v \in B} r_i(v) \cdot (\sum_{y \in S} r_{i+1}(y) \cdot C''_i(x, y, v)).$$

We prefer not to introduce a formal notion here.

From the cells described above we now construct a stronger machine by putting the cells in a chain and defining communications between connected cells.

Consider the configuration as pictured in Figure 2 below. Since now the cells are connected by their channels it is easy to see how we should define an appropriate communication function. Through channel i the cells C_{i-1} and C_i communicate by the communication action $s_i(x)|r_i(x)$. Any separate action $s_i(x)$ or $r_i(x)$ means something like 'waiting to communicate' and since we do not want our machine to wait eternally for communication we have to encapsulate them. The only exceptions are $s_{k+1}(x)$ and $r_{k+1}(x)$ since there is no cell C_{k+1} to communicate with them. Hence these two actions can communicate with the outside world.

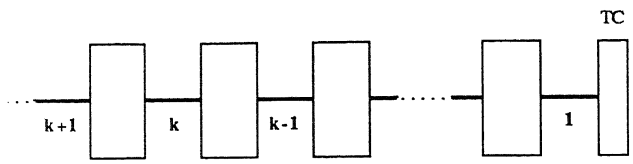

FIGURE 2. A chain configuration of k cells

From now we assume k to be fixed.

We have in general the following *communication function* defined on atomic actions:

$$s_i(x)|r_i(x) = c_i(x) \quad \text{for all } x \in S \cup B \text{ and } i < k+1$$
$$a|b = \delta \quad \text{for all other pairs of actions } a,b \in A.$$

The *encapsulation set* H_k of actions resulting in a deadlock is defined as

$$H_k = \{s_i(x), r_i(x): x \in S \cup B \text{ and } i < k+1\}.$$

The *abstraction set* I of invisible machine actions is defined as

$$I = \{c_i(x): x \in S \cup B \text{ and } i < \omega\}.$$

Note that by definition machine actions are *invisible* if and only if they do not occur in the specification of the particular machine. One can also look at them as *internal actions* that can not be influenced from the outside.

The machine pictured in Figure 2 can algebraically be described as a communication merge $M(k)$ of k individual cells i.e:

$$M(k) = \tau_I \partial_{H_k}(C_k \| \cdots \| C_1 \| \mathrm{TC}).$$

In the following we will formally prove that $M(k)$ indeed is an implementation of the palindrome-recogniser given in Table 3.

3. A FORMAL PROOF OF CORRECTNESS

Before turning to the formal proof itself let us first try an example to see how the machine works. Indeed this gives us some intuition about the practical behaviour of $M(k)$ which will be helpful later in this section. The specific example given below was found in [8].

In Figure 3 four connected cells are pictured and we can look at the machine until the string *baabaaba* is input. As we see, immediately after receiving a new input symbol the machine returns a boolean value at the left-most channel to state whether the string in the machine is a palindrome or not.

In Figure 4 we connect the *terminal cell* TC to our previous machine and assume *aababba* has already been input. When in addition *abb* is input we get as output, true, although *abbaababba* is not a palindrome. So we see that the behaviour of the machine depends on the length of the input.

If the input gets too long TC will destruct input symbols loosing all relevant information about them.

We will now get to the main fact in this paragraph which will be proved by means of the equations of ACP_τ together with RSP, the Recursive Specification Principle which says that if two processes satisfy the same guarded recursive specification then they are equal.

To do this we first need to give a more detailed specification of the machine we have constructed so far. As a matter of fact we will prove our machine to be equal to the process DP_k specified below in Table 5.

$$DP_0(w)=TC \qquad (0\leqslant|w|)$$

$$DP_{k+1}(\epsilon)=s_{k+2}(\text{true})\cdot DP_{k+1}(\epsilon)+\sum_{x\in S} r_{k+2}(x)\cdot s_{k+2}(\text{true})\cdot DP_{k+1}(x)$$

$$DP_{k+1}(w)=\sum_{x\in S} r_{k+2}(x)\cdot s_{k+2}(\text{ispal}(x\cdot w))\cdot DP_{k+1}(x\cdot w) \qquad (0<|w|<2(k+1))$$

$$DP_{k+1}(w)=\sum_{x\in S} r_{k+2}(x)\cdot s_{k+2}(\text{ispal}(x\cdot f(k+1,w)))\cdot DP_{k+1}(x\cdot w) \qquad (2(k+1)\leqslant|w|)$$

$$TC=s_1(\text{true})\cdot TC+\sum_{x\in S} r_1(x)\cdot TC$$

where a function $f(k,w)$ is defined as

$$f(k,w)=\begin{cases} w & \text{if } |w|<2k \\ \text{first}(k-1,w)\cdot\text{last}(k,w) & \text{otherwise}\end{cases}$$

with the obvious extra functions

$$\text{first}(k,x_1\cdots x_n)=(x_1\cdots x_k)$$

$$\text{last}(k,x_1\cdots x_n)=(x_{n-k+1}\cdots x_n).$$

TABLE 5. A specification of DP_k for arbitrary natural number k

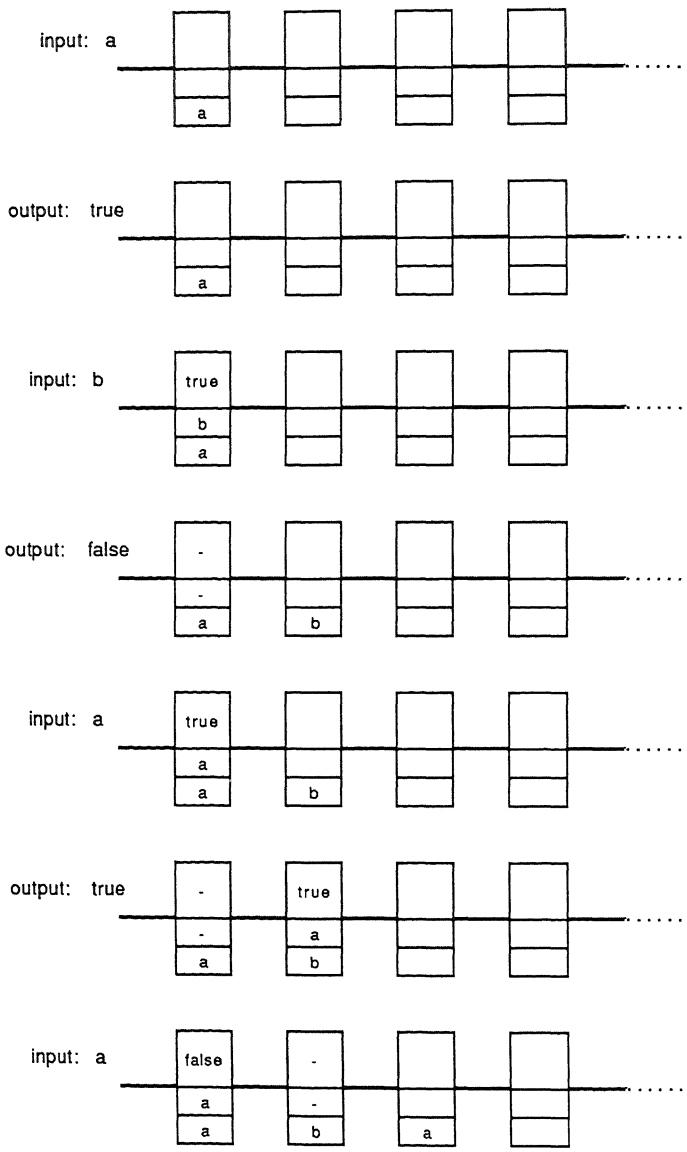

FIGURE 3. An example to give an idea of how the machine works (to be continued)

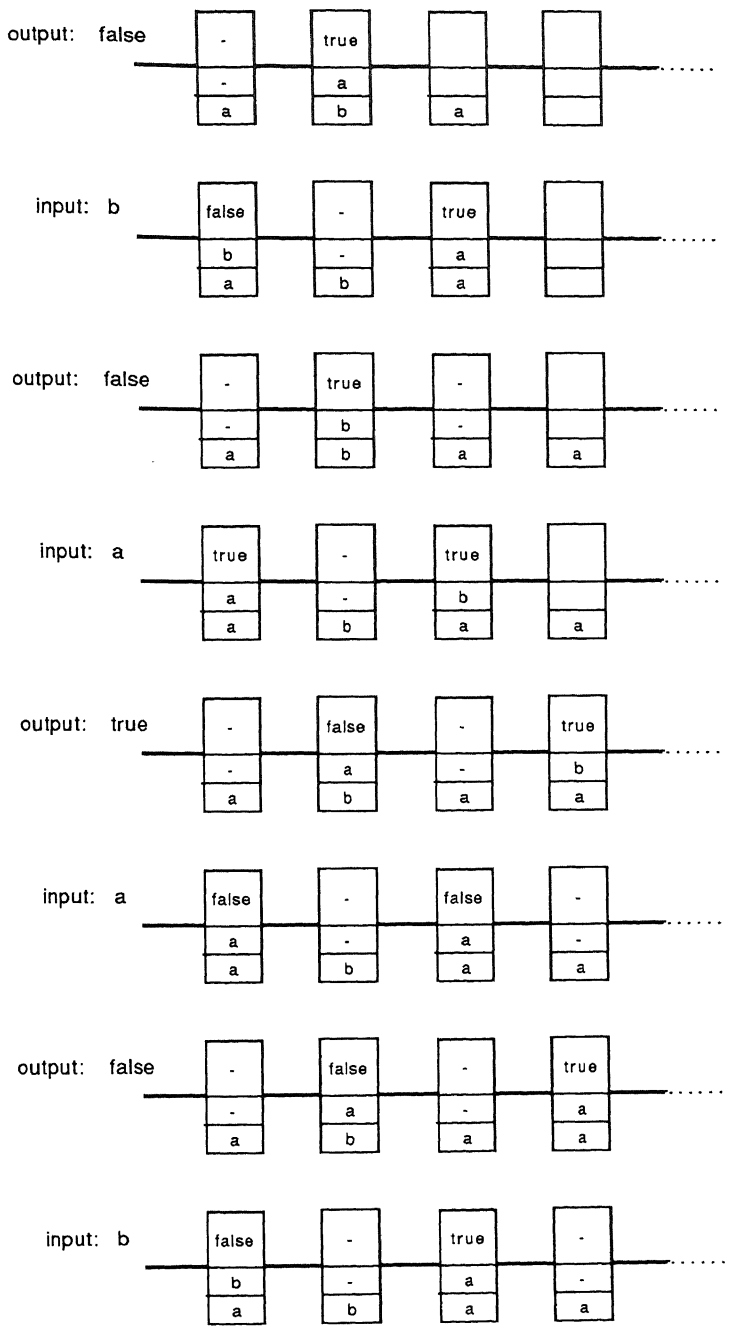

FIGURE 3. (continued)

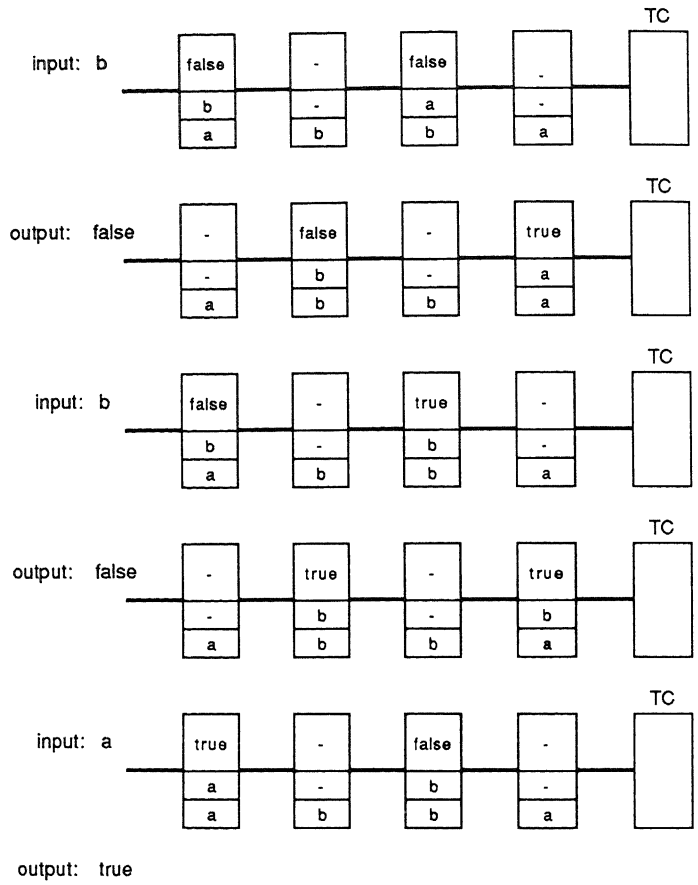

FIGURE 4. The machine now in connection with the terminal cell

Comparing the specifications of DP_k and PAL_k (see Table 5 and Table 3) one can see immediately that DP_k is a more detailed version of PAL_k. From the axioms in Table 1 it follows easily that $PAL_k(\epsilon) \vdash DP_k(\epsilon)$.

FACT. $M(k) = \tau_I \partial_{H_k}(C_k \| \cdots \| C_1 \| TC) = DP_k(\epsilon)$.

PROOF. By induction on k.

$k=0$: $\tau_I \partial_{H_0}(M(0)) = \tau_I \partial_{H_0}(TC) = TC = DP_0(\epsilon)$.

$k+1$: we first prove

$$\tau_I \partial_{H_{k+1}}(C_{k+1} \| DP_k(\epsilon)) = DP_{k+1}(\epsilon).$$

Then the result can easily be proved by use of the conditional axioms. It is easily checked that the following two equations hold:

$$\tau_I \partial_{H_{k+1}}(C_{k+1} \| DP_k(\epsilon)) = s_{k+2}(\text{true}) \cdot \tau_I \partial_{H_{k+1}}(C_{k+1} \| DP_k(\epsilon)) + \\ + \sum_{x \in S} r_{k+2}(x) \cdot s_{k+2}(\text{true}) \cdot \tau_I \partial_{H_{k+1}}(C'_{k+1}(x) \| DP_k(\epsilon)). \quad (1)$$

$$\tau_I \partial_{H_{k+1}}(C'_{k+1}(x) \| DP_k(\epsilon)) = \\ = \tau \cdot \sum_{y \in S} r_{k+2}(y) \cdot \tau_I \partial_{H_{k+1}}(C''_{k+1}(x,y,\text{true}) \| DP_k(\epsilon)). \quad (2)$$

To continue we need a definition.

DEFINITION. $g(k,v) = \begin{cases} v & \text{if } |v| \leqslant 2k \\ \text{first}(k,v) \cdot \text{last}(k,v) & \text{otherwise} \end{cases}$

Now we can formulate what is in fact the crucial induction hypothesis:

LEMMA. *For all symbols $x,y \in S$ and strings $v \in S^*$ we have*

(i) $\tau_I \partial_{H_{k+1}}(C'_{k+1}(x) \| (s_{k+1}(\text{ispal}(g(k,v))) \cdot DP_k(v))) = \tau \cdot DP_{k+1}(v \cdot x)$

(ii) $\tau_I \partial_{H_{k+1}}(C''_{k+1}(x, y, \text{ispal}(g(k,v))) \| DP_k(v))$
$= \tau \cdot s_{k+1}(\text{ispal}(y \cdot f(k+1, v \cdot x))) \cdot DP_{k+1}(y \cdot v \cdot x).$

PROOF. Let

$$Q(x, v) = \tau_I \partial_{H_{k+1}}(C'_{k+1}(x) \| (s_{k+1}(\text{ispal}(g(k, v))) \cdot DP_k(v))).$$

Now we prove that we have

$$Q(x, v) = \tau \cdot \sum_{y \in S} r_{k+2}(y) \cdot s_{k+2}(\text{ispal}(y \cdot f(k+1, v \cdot x))) \cdot Q(x, y \cdot v)$$

which gives us precisely $\tau \cdot DP_{k+1}(v \cdot x)$ given in Table 5, and hence Lemma (i) by RSP. We have

$$Q(x,v) =$$

$$= \tau_I \partial_{H_{k+1}}(((\sum_{y \in S} r_{k+2}(y) \| \sum_{b \in B} r_{k+1}(b)) \cdot C''_{k+1}(x, y, b)) \| \\ \| (s_{k+1}(\text{ispal}(g(k,v))) \cdot DP_k(v)))$$

$$= \sum_{y \in S} r_{k+2}(y) \cdot \tau_I \partial_{H_{k+1}}((\sum_{b \in B} r_{k+1}(b) \cdot C''_{k+1}(x, y, b)) \| (s_{k+1}(\text{ispal}(g(k,v))) \cdot DP_k(v))) + \\ + \tau \cdot \tau_I \partial_{H_{k+1}}((\sum_{y \in S} r_{k+2}(y) \cdot C''_{k+1}(x, y, \text{ispal}(g(k,v)))) \| DP_k(v))$$

$$= \tau \cdot \sum_{y \in S} r_{k+2}(y) \cdot \tau_I \partial_{H_{k+1}}(C''_{k+1}(x, y, \text{ispal}(g(k,v))) \| DP_k(v)) \quad \text{(using axiom T2).}$$

Furthermore we have

$$\tau_I \partial_{H_{k+1}}(C''_{k+1}(x, y, \text{ispal}(g(k,v)))\|\text{DP}_k(v)) =$$

$$= s_{k+2}(|x=y| \wedge \text{ispal}(g(k,v))) \cdot \tau_I \partial_{H_{k+1}}((s_{k+1}(y) \cdot C'_{k+1}(x))\|$$
$$\|(\sum_{z \in S} r_{k+1}(z) \cdot s_{k+1}(\text{ispal}(z \cdot f(k, v))) \cdot \text{DP}_k(z \cdot v))) +$$
$$+ \tau \cdot \tau_I \partial_{H_{k+1}}((s_{k+1}(|x=y| \wedge \text{ispal}(g(k,v))) \cdot C'_{k+1}(x))\|$$
$$\|(s_{k+1}(\text{ispal}(y \cdot f(k, v))) \cdot \text{DP}_k(y \cdot v)))$$

$$= \tau \cdot s_{k+2}(|x=y| \wedge \text{ispal}(g(k,v))) \cdot$$
$$\cdot \tau_I \partial_{H_{k+1}}(C'_{k+1}(x)\|(s_{k+1}(\text{ispal}(y \cdot f(k, v))) \cdot \text{DP}_k(y \cdot v)))$$

and since

$$|x=y| \wedge \text{ispal}(g(k,v)) = \text{ispal}(y \cdot f(k, v \cdot x))$$
$$y \cdot f(k,v) = g(k, y \cdot v)$$

we have

$$= \tau \cdot s_{k+2}(\text{ispal}(y \cdot f(k+1, v \cdot x))) \cdot$$
$$\cdot \tau_I \partial_{H_{k+1}}(C'_{k+1}(x)\|(s_{k+1}(\text{ispal}(g(k, y \cdot v))) \cdot \text{DP}_k(y \cdot v)))$$
$$= \tau \cdot s_{k+2}(\text{ispal}(y \cdot f(k+1, v \cdot x))) \cdot Q(x, y \cdot v).$$

After substitution we find

$$Q(x, v) = \tau \cdot \sum_{y \in S} r_{k+2}(y) \cdot s_{k+2}(\text{ispal}(y \cdot f(k+1, v \cdot x))) \cdot Q(x, y \cdot v)$$

which is precisely what we wanted.
By RSP we have Lemma (i). Note that we implicitly proved (ii). □ (Lemma)

The rest of the proof is straightforward:
With Lemma (ii) and (2) we have

$$\tau_I \partial_{H_{k+1}}(C'_{k+1}(x)\|\text{DP}_k(\epsilon)) =$$
$$= \tau \cdot \sum_{y \in S} r_{k+2}(y) \cdot s_{k+2}(\text{ispal}(y \cdot f(k+1, x))) \cdot \text{DP}_{k+1}(y \cdot x)$$
$$= \tau \cdot \text{DP}_{k+1}(x).$$

Finally with (1) we have

$$\tau_I \partial_{H_{k+1}}(C_{k+1} \| DP_k(\epsilon)) =$$

$$= s_{k+2}(\text{true}) \cdot \tau_I \partial_{H_{k+1}}(C_{k+1} \| DP_k(\epsilon)) + \sum_{x \in S} r_{k+2}(x) \cdot s_{k+2}(\text{true}) \cdot \tau \cdot DP_{k+1}(x)$$

$$= DP_{k+1}(\epsilon)$$

using RSP again.

Note that we have proved

$$\tau_I \partial_{H_{k+1}}(C_{k+1} \| \tau_I \partial_{H_k}(C_k \| \cdots \tau_I \partial_{H_1}(C_1 \| TC) \cdots)) = DP_{k+1}(\epsilon).$$

It is easy to prove by induction, however, that

$$\alpha(C_{k+1}) | (\alpha(C_k \| M(k-1)) \cap H_k) \subseteq H_k$$

for all k (in fact $\alpha(C_{k+1}) | (\alpha(C_k \| M(k-1)) \cap H_k) = \varnothing$, and

$$\alpha(C_{k+1}) | (\alpha(C_k \| M(k-1)) \cap I) = \varnothing).$$

So because $H_{k+1} \supseteq H_k$ we find with the use of the conditional axioms CA1, CA2 and CA5:

$$\tau_I \partial_{H_{k+1}}(C_{k+1} \| \tau_I \partial_{H_k}(C_k \| \cdots \tau_I \partial_{H_1}(C_1 \| TC) \cdots)) =$$

$$= \tau_I \partial_{H_{k+1}}(\tau_I \partial_{H_k}(\cdots \tau_I \partial_{H_1}(C_{k+1} \| C_k \| \cdots \| C_1 \| TC) \cdots)).$$

Since

$$H \cap I = \varnothing$$

we have

$$\tau_I \partial_{H_{k+1}}(\tau_I \partial_{H_k}(\cdots \tau_I \partial_{H_1}(C_{k+1} \| C_k \| \cdots \| C_1 \| TC) \cdots)) =$$

$$= \tau_I \cdots \tau_I \partial_{H_{k+1}} \cdots \partial_{H_1}(C_{k+1} \| C_k \| \cdots \| C_1 \| TC)$$

by axiom CA7 and finally with axioms CA5 and CA6 we find

$$\tau_I \cdots \tau_I \partial_{H_{k+1}} \cdots \partial_{H_1}(C_{k+1} \| C_k \| \cdots \| C_1 \| TC) =$$

$$= \tau_I \partial_{H_{k+1}}(C_{k+1} \| C_k \| \cdots \| C_1 \| TC)$$

which is exactly $M(k+1)$.

Therefore, we have $M(k) = DP_k(\epsilon)$, for all k. This finishes the induction. □

Finally we find

$$PAL_k(\epsilon) \vdash DP_k(\epsilon) = M(k) = \tau_I \partial_{H_k}(C_k \| \cdots \| C_1 \| TC)$$

so we have

$$PAL_k(\epsilon) \vdash \tau_I \partial_{H_k}(C_k \| \cdots \| C_1 \| TC)$$

which is the formal way to express that $\tau_I \partial_{H_k}(C_k \| \cdots \| C_1 \| TC)$ indeed is a palindrome-recogniser.

4. A SORTING MACHINE

A second example of a machine implemented by a 'systolic system' is a sorting machine. A sorting machine can input a sequence of numbers and output them in increasing order. First we will discuss a restricted sorting machine which is a sorting machine with a restricted capacity. For a good performance of such a restricted sorting machine with capacity n it is necessary that the machine does not contain more than n numbers. If the absolute value of the difference of the number of input and output actions is greater than n, the behaviour of the restricted sorting machine is undefined. Later on, a sorting machine which can contain an arbitrary amount of numbers will be discussed.

Before we discuss in what way the restricted sorting machine is constructed we first state its expected external behaviour. This is done in Table 6.

$$\mathrm{SORT}_k(\varnothing) = s_k(\mathrm{empty}) \cdot \mathrm{SORT}_k(\varnothing) + \sum_{d \in D} r_k(d) \cdot \mathrm{SORT}_k(\{d\})$$

$$\mathrm{SORT}_k(B) = s_k(\mu B) \cdot \mathrm{SORT}_k(B - \{\mu B\}) + \sum_{d \in D} r_k(d) \cdot \mathrm{SORT}_k(B \cup \{d\}) \quad 0 < |B| < k$$

$$\mathrm{SORT}_k(B) = s_k(\mu B) \cdot \mathrm{SORT}_k(B - \{\mu B\}) + \sum_{d \in D} r_k(d) \cdot \Omega \quad |B| = k$$

TABLE 6. Specification of a restricted sorting machine with capacity $k\,(k > 0)$

Some explanation is useful here. B is a bag or multiset with $|B|$ elements. $\varnothing$ is the empty bag. If bag B is not empty the minimal element of B is denoted by μB. On bags the operations $\cup$ and $-$ are defined in the standard way. $\mathrm{SORT}_k(B)$ is the restricted sorting machine of capacity k with contents B. SORT_k has a communication channel k. Through this channel the restricted sorting machine can output (s_k) and input (r_k) data to and from the outside. A datum can be a number or a special symbol called 'empty'. The relevance of sending an empty signal is made clear in the implementation part later on. There it turns out to be an inevitable action as a result of that implementation. The Ω stands for the process chaos discussed in Section 2. Ω is encountered when the content of the restricted sorting machine gets greater than its capacity. The behaviour of the machine then becomes irrelevant.

Now we will describe the implementation of a restricted sorting machine of a certain capacity by connecting a number of identical cells. It shall be proved that k connected cells plus one special cell is an implementation of the restricted sorting machine SORT_{2k}. The notion of implementation, denoted by $\vDash$ is described in Section 2. Before we discuss a chain of cells we first turn to an individual cell.

An individual cell has two storage locations called MIN and MAX and two communication channels. The channels of cell C_i $(i > 0)$ are called i and $i-1$.

Elements of a number set D can be stored in MIN and MAX. Elements of D and 'empty' can be transmitted through the communication channels. An individual cell C_i is pictured in Figure 5.

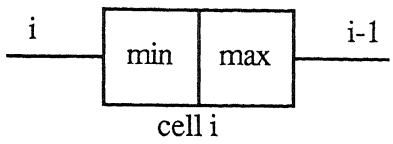

FIGURE 5. An individual cell C_i

Each cell can be in three states.

(0) In this state both storage locations MIN and MAX are empty. The cell C_i can receive a number from the left. This number is stored in MIN and the cell enters state (1). Another possible action is sending an 'empty' to the left. In this case the cell remains in state (0). State (0) is also the initial state for each cell C_i $(i>0)$.

(1) In state (1), MIN is filled and MAX is empty (really empty). A number from the left can be received. The minimum of the content of MIN and the received number is stored in MIN. The other number is stored in MAX. State (2) is entered. The second possibility is sending the content of MIN to the left and entering state (0) again.

(2) Now MIN and MAX are both filled and the content of MIN is less than or equal to the content of MAX. The cell C_i can receive a number from the left and send the content of MAX to the right. MIN becomes the minimum of the content of MIN and the received number. The other number is stored in MAX. The cell remains in state (2). The other action the cell can perform is sending the content of MIN to the left. Now two possibilities arise: the cell receives an empty signal from the right, MIN gets the content of MAX and the cell enters state (1). The second possibility is receiving a number from the right. MIN becomes the minimum of MAX and the received number and MAX becomes the maximum of the two. The cell C_i remains in state (2).

Because we are building a restricted sorting machine an extra cell is needed. This cell is called C_0. It is pictured in Figure 6. This cell has one communication channel called 0 and contains no storage locations. C_0 remains always in the same state. In this state C_0 is able to send an 'empty' to the left or receive a number from the left. The number received disappears completely. C_0 can be considered as a cell crushing the incoming numbers.

The specification of the cell C_i $(i\geqslant 0)$ is given in Table 7.

All parts needed for building a restricted sorting machine have been discussed. A restricted sorting machine with capacity $2k$ can be built by interconnecting $k+1$ cells C_i $(0\leqslant i\leqslant k)$. C_i and $C_{i-1}(1\leqslant i\leqslant k)$ communicate through channel $i-1$. Channel k is the external input/output channel for the machine.

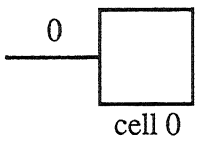

FIGURE 6. The terminal cell

$$
\begin{aligned}
i=0: \quad & C_0 = \sum_{d\in D} r_0(d)\cdot C_0 + s_0(\text{empty})\cdot C_0 \\
i>0: \quad & C_i = \sum_{d\in D} r_i(d)\cdot C'_i(d) + s_i(\text{empty})\cdot C_i \\
& C'_i(d) = \sum_{e\in D} r_i(e)\cdot C''_i(sw(d,e)) + s_i(d)\cdot C_i \\
& C''_i(d,e) = \sum_{f\in D} r_i(f)\cdot s_{i-1}(e)\cdot C_i''(sw(d,f)) \\
& \quad + s_i(d)\cdot(r_{i-1}(\text{empty})\cdot C'_i(e) + \sum_{f\in D} r_{i-1}(f)\cdot C''_i(sw(e,f))) \\
& \text{Here } sw \text{ stands for swap: } sw(d,e)=(\min(d,e),\max(d,e))
\end{aligned}
$$

TABLE 7. Specification of an individual cell C_i for $i \geqslant 0$

When an internal cell i (that is a cell which is not the first cell in the chain) performs an action $s_i(d)$, $r_i(d)$, $s_{i-1}(d)$ or $r_{i-1}(d)$, $d \in D \cup \{\text{empty}\}$, this action must be matched by a complementary action of a neighbouring cell. For cell C_k only actions $s_{k-1}(d)$ and $r_{k-1}(d)$ must be answered by complementary actions of cell $k-1$. This is achieved in Process Algebra by defining communications $c_i(d)$ as the result of $s_i(d)$ and $r_i(d)$ and encapsulating the individual actions $s_i(d)$ and $r_i(d)$. Of course the actions $r_k(d)$ and $s_k(d)$ are not encapsulated because these actions are the communications with the outside world. To illustrate that this chain of k cells plus one special cell really gives a restricted sorting machine of capacity $2k$ an example is worked out in Figure 7. In this case $k=3$.

A formal description of the machine discussed before and pictured in Figure 7 is given in Table 8. We call the empty restricted sorting machine built from k normal cells plus the terminal cell $\text{SORT}^*_{2k}(\epsilon)$. H_k is the encapsulation set and contains the actions that should not be performed without a partner. To describe the external behaviour of the restricted sorting machine we abstract from the internal actions that still can be performed after encapsulation. Symbols to be abstracted from are in I. The resulting sorting machine is called $\text{SORT}^*_{2k}(\epsilon)$. Now it will be proved that this restricted sorting machine is an implementation of the restricted sorting machine defined by the specification in Table 6.

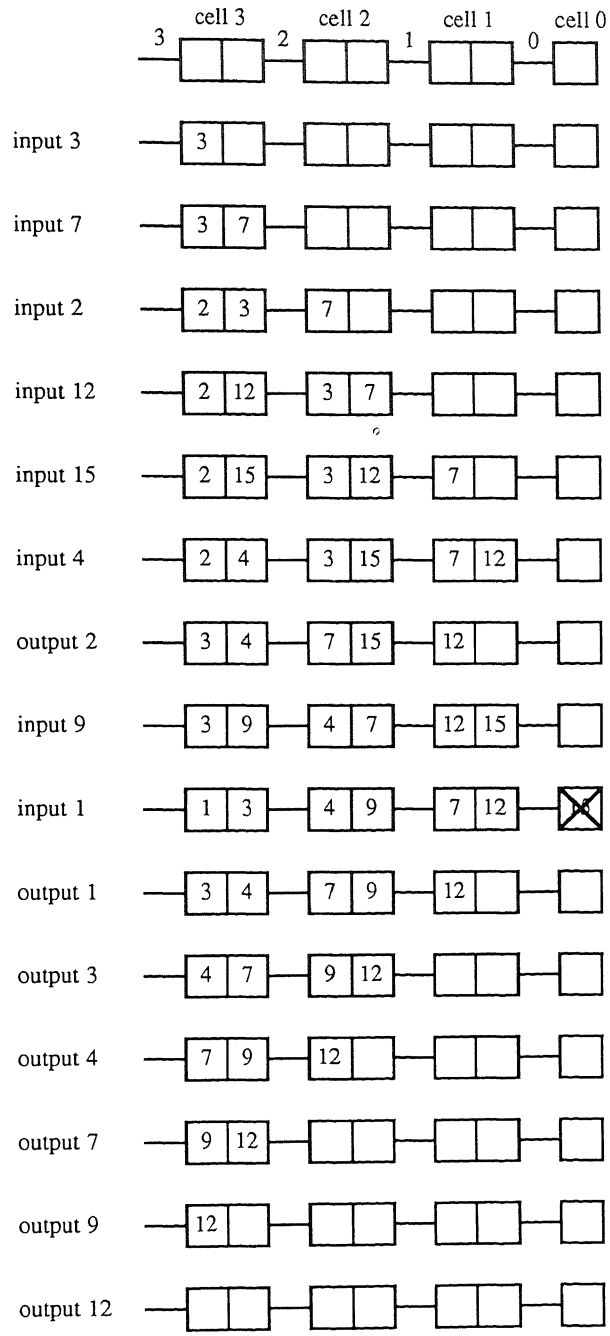

FIGURE 7. Example of restricted sorting machine

$\mathrm{SORT}^*_{2k}(\epsilon) = \tau_I \circ \partial_{H_k}(C_k \| \cdots \| C_0)$

Communication actions $c_i(d)$ for each pair $r_i(d)$, $s_i(d)$, $d \in D \cup \{\text{empty}\}$, $i < k$: $c_i(d) = r_i(d) | s_i(d)$

$H_k = \{s_i(d),\ r_i(d)\colon d \in D \cup \{\text{empty}\},\ 0 \leqslant i < k\}$

$I = \{c_i(d)\colon d \in D \cup \{\text{empty}\},\ i \geqslant 0\}$

TABLE 8. A restricted sorting machine with capacity k for $k > 0$

FACT. *For all* $k \geqslant 1$: $\mathrm{SORT}_{2k}(\varnothing) \vdash \mathrm{SORT}^*_{2k}(\epsilon)$.

Before we turn to the proof some definitions are given.

DEFINITION.

(i) A sequence $\langle d_1, \ldots, d_n \rangle$ is called correctly ordered (c.o.) if and only if $d_i \leqslant d_{i+2}$ and $d_i \leqslant d_{i+1}$ for all odd i. Note that every sequence contained in the restricted sorting machine at any time will be c.o., as illustrated in Figure 7.

(ii) On sequences $\langle d_1, \ldots, d_n \rangle$ a function sw is defined inductively as follows:
$sw(\epsilon) = \epsilon$
$sw(\langle d_1 \rangle) = \langle d_1 \rangle$
$sw(w) = \langle \min(d_1, d_2), \max(d_1, d_2) \rangle \star sw(w')$ if $|w| \geqslant 2 w = \langle d_1, d_2 \rangle \star w'$

FACT. (i) *if* w *is c.o. then* $sw(\langle d \rangle \star w)$ *is c.o.*
(ii) *if* $\langle d \rangle \star w$ *is c.o. then* $sw(w)$ *is c.o.*
(iii) *if* $w \star \langle d \rangle$ *is c.o. then* $sw(w)$ *is c.o.*

PROOF. The proof consists of two parts. First we prove that $\mathrm{SORT}^*_{2k}(\epsilon)$ is a solution of the specification, formulated in Table 9. Next we prove that any solution of that specification also is an implementation of the restricted sorting machine specified in Table 6. First we define $\mathrm{SORT}'_{2k}(w)$ for $k \geqslant 0$, $w \in S^*$, $0 \leqslant |w| \leqslant 2k$ and w c.o. $\mathrm{SORT}'_{2k}(w)$ is defined below:

DEFINITION.

(i) $\mathrm{SORT}'_0(\epsilon) = C_0$
(ii) $\mathrm{SORT}'_{2k+2}(\epsilon) = \tau_I \circ \partial_{H_{k+1}}(C_{k+1} \| \mathrm{SORT}'_{2k}(\epsilon))$
(iii) $\mathrm{SORT}'_{2k+2}(\langle d_1 \rangle) = \tau_I \circ \partial_{H_{k+1}}(C'_{k+1}(d_1) \| \mathrm{SORT}'_{2k}(\epsilon))$
(iv) $\mathrm{SORT}'_{2k+2}(\langle d_1, d_2 \rangle \star w) = \tau_I \circ \partial_{H_{k+1}}(C''_{k+1}(d_1, d_2) \| \mathrm{SORT}'_{2k}(w))$

Now $\mathrm{SORT}''_{2k}(\epsilon) = \mathrm{SORT}^*_{2k}(\epsilon)$ is proved in two steps:

$$\mathrm{SORT}''_{2k}(\epsilon)=s_k(\mathrm{empty})\cdot\mathrm{SORT}''_{2k}(\epsilon)+\sum_{d\in D} r_k(d)\cdot\mathrm{SORT}''_{2k}(\langle d\rangle)$$

for all w with $|\langle d_1\rangle\star w|<2k$ and $\langle d_1\rangle\star w$ c.o.:

$$\mathrm{SORT}''_{2k}(\langle d_1\rangle\star w)=\sum_{d\in D} r_k(d)\cdot\mathrm{SORT}''_{2k}(sw(\langle d\rangle\star\langle d_1\rangle\star w)) + s_k(d_1)\cdot\mathrm{SORT}''_{2k}(sw(w))$$

for all w with $|\langle d_1\rangle\star w\star\langle d_{2k}\rangle|=2k$ and $\langle d_1\rangle\star w\star\langle d_{2k}\rangle$ c.o.:

$$\mathrm{SORT}''_{2k}(\langle d_1\rangle\star w\star\langle d_{2k}\rangle)=\sum_{d\in D} r_k(d)\cdot\mathrm{SORT}''_{2k}(sw(\langle d\rangle\star\langle d_1\rangle\star w)) + s_k(d_1)\cdot\mathrm{SORT}''_{2k}(sw(w\star\langle d_{2k}\rangle))$$

TABLE 9. Intermediate specification of a restricted sorting machine with capacity $2k$

first step: for all $k\geqslant 1$, w c.o. and $0\leqslant w\leqslant 2k$:

$$\mathrm{SORT}''_{2k}(w)=\mathrm{SORT}'_{2k}(w) \qquad (\star)$$

second step: for all $k\geqslant 1$: $\mathrm{SORT}'_{2k}(\epsilon)=\mathrm{SORT}^*_{2k}(\epsilon)$ $(\star\star)$

PROOF $(\star)$. The first step will be proved by induction on k and the length of the content. $k=1$: three subcases have to be considered:

$w=\epsilon$ (i)

$$\mathrm{SORT}'_2(\epsilon) = \tau_I\circ\partial_{H_1}(C_1\|C_0) =$$
$$=\tau_I\circ\partial_{H_1}\Big(\sum_{d\in D} r_1(d)\cdot C'_1(d)+s_1(\mathrm{empty})\cdot C_1\|C_0\Big)$$
$$=\sum_{d\in D} r_1(d)\cdot\tau_I\circ\partial_{H_1}(C'_1(d)\|C_0)+s_1(\mathrm{empty})\cdot\tau_I\circ\partial_{H_1}(C_1\|C_0)$$

Using the definition of $\mathrm{SORT}'_{2k}(w)$ the intended result is obtained:

$$\mathrm{SORT}'_2(\epsilon)=\sum_{d\in D} r_1(d)\cdot\mathrm{SORT}'_2(\langle d\rangle)+s_1(\mathrm{empty})\cdot\mathrm{SORT}'_2(\epsilon)$$

$w=\langle d_1\rangle$ (ii)

$$\mathrm{SORT}'_2(\langle d_1\rangle)=\tau_I\circ\partial_{H_1}(C'_1(d_1)\|C_0)$$
$$=\tau_I\circ\partial_{H_1}\Big(\sum_{d\in D} r_1(d)\cdot C''_1(sw(d,d_1))+s_1(d_1)\cdot C_1\|C_0\Big)$$

$$= \sum_{d \in D} r_1(d) \cdot \tau_I \circ \partial_{H_1}(C''_1(sw(d,d_1)) \| \mathrm{SORT}'_0(\epsilon)) +$$

$$+ s_1(d_1) \cdot \tau_I \circ \partial_{H_1}(C_1 \| \mathrm{SORT}'_0(\epsilon))$$

Using again the definition of $\mathrm{SORT}'_{2k}(w)$:

$$\mathrm{SORT}'_2(\langle d_1 \rangle) = \sum_{d \in D} r_1(d) \cdot \mathrm{SORT}'_2(sw(\langle d, d_1 \rangle)) + s_1(d) \cdot \mathrm{SORT}_2(\epsilon)$$

$\boxed{w = \langle d_1, d_2 \rangle, w \text{ is c.o.}}$ (iii)

$$\mathrm{SORT}'_2(\langle d_1, d_2 \rangle) = \tau_I \circ \partial_{H_1}(C''_1(d_1, d_2) \| C_0)$$

$$= \sum_{d \in D} r_1(d) \cdot \tau_I \circ \partial_{H_1}((s_0(d_2) \cdot C''_1(sw(d, d_1))) \| C_0) +$$

$$+ s_1(d_1) \cdot \tau_I \circ \partial_{H_1}((r_0(\mathrm{empty}) \cdot C'_1(d_2) + \sum_{e \in D} r_0(e) \cdot C''_1(sw(d_2, e))) \| C_0)$$

$$= \sum_{d \in D} r_1(d) \cdot \tau \cdot \tau_I \circ \partial_{H_1}(C''_1(sw(d, d_1)) \| C_0) + s_1(d_1) \cdot \tau \cdot \tau_I \circ \partial_{H_1}(C'_1(d_2) \| C_0)$$

so we have

$$\mathrm{SORT}'_2(\langle d_1, d_2 \rangle) = \sum_{d \in D} r_1(d) \cdot \mathrm{SORT}'_2(sw(\langle d, d_1 \rangle)) + s_1(d_1) \cdot \mathrm{SORT}'_2(\langle d_2 \rangle)$$

$k = n + 1$ where $n > 0$. Five cases have to be considered:

$\boxed{w = \epsilon}$ (i)

$$\mathrm{SORT}'_{2(n+1)}(\epsilon) = \tau_I \circ \partial_{H_{n+1}}(C_{n+1} \| \mathrm{SORT}'_{2n}(\epsilon)) =$$

$$= \tau_I \circ \partial_{H_{n+1}}(\sum_{d \in D} r_{n+1}(d) \cdot C'_{n+1}(d) + s_{n+1}(\mathrm{empty}) \cdot C_{n+1} \| \mathrm{SORT}'_{2n}(\epsilon))$$

$$= \sum_{d \in D} r_{n+1}(d) \cdot \tau_I \circ \partial_{H_{n+1}}(C'_{n+1}(d) \| \mathrm{SORT}'_{2n}(\epsilon)) +$$

$$+ s_{n+1}(\mathrm{empty}) \cdot \tau_I \circ \partial_{H_{n+1}}(C_{n+1} \| \mathrm{SORT}'_{2n}(\epsilon))$$

$$= \sum_{d \in D} r_{n+1}(d) \cdot \mathrm{SORT}'_{2(n+1)}(\langle d \rangle) + s_{n+1}(\mathrm{empty}) \cdot \mathrm{SORT}'_{2(n+1)}(\epsilon)$$

This expression is in the form of the specification of Table 9.

$\boxed{w=\langle d_1\rangle}$ (ii)

$$\mathrm{SORT}'_{2(n+1)}(\langle d_1\rangle)=\tau_I\circ\partial_{H_{n+1}}(C'_{n+1}(d_1)\|\mathrm{SORT}'_{2n}(\epsilon))$$

$$=\tau_I\circ\partial_{H_{n+1}}(\sum_{d\in D} r_{n+1}(d)\cdot C''_{n+1}(sw(d,d_1))+s_{n+1}(d_1)\cdot C_{n+1}\|\mathrm{SORT}'_{2n}(\epsilon))$$

$$=\sum_{d\in D} r_{n+1}(d)\cdot\tau_I\circ\partial_{H_{n+1}}(C''_{n+1}(sw(d,d_1))\|\mathrm{SORT}'_{2n}(\epsilon))+$$

$$+s_{n+1}(d_1)\cdot\tau_I\circ\partial_{H_{n+1}}(C_{n+1}\|\mathrm{SORT}'_{2n}(\epsilon))=$$

$$=\sum_{d\in D} r_{n+1}(d)\cdot\mathrm{SORT}'_{2(n+1)}(sw(\langle d,d_1\rangle))+s_{n+1}(d_1)\cdot\mathrm{SORT}'_{2(n+1)}(\epsilon)$$

Again in the form of the specification of Table 9.

$\boxed{|w|=2,\ w=\langle d_1,d_2\rangle,\ w \text{ c.o.}}$ (iii)

$$\mathrm{SORT}'_{2(n+1)}(\langle d_1,d_2\rangle)=\tau_I\circ\partial_{H_{n+1}}(C''_{n+1}(d_1,d_2)\|\mathrm{SORT}'_{2n}(\epsilon))$$

$$=\sum_{d\in D} r_{n+1}(d)\cdot\tau_I\circ\partial_{H_{n+1}}(s_n(d_2)\cdot C''_{n+1}(sw(d,d_1))\|\mathrm{SORT}'_{2n}(\epsilon))+$$

$$+s_{n+1}(d)\cdot\tau_I\circ\partial_{H_{n+1}}((r_n(\mathrm{empty})\cdot C'_{n+1}(d_2)+$$

$$+\sum_{e\in D} r_n(e)\cdot C_{n+1}(sw(d_2,e)))\|\mathrm{SORT}'_{2n}(\epsilon))$$

(induction hypothesis)

$$=\sum_{d\in D} r_{n+1}(d)\cdot\tau\cdot\tau_I\circ\partial_{H_{n+1}}(C''_{n+1}(sw(d,d_1))\|\mathrm{SORT}'_{2n}(\langle d_2\rangle))+$$

$$+s_{n+1}(d_1)\cdot\tau\cdot\tau_I\circ\partial_{H_{n+1}}(C'_{n+1}(d_2)\|\mathrm{SORT}'_{2n}(\epsilon))$$

$$=\sum_{d\in D} r_{n+1}(d)\cdot\mathrm{SORT}'_{2(n+1)}(sw(\langle d,d_1,d_2\rangle))+s_{n+1}(d)\cdot\mathrm{SORT}'_{2(n+1)}(\langle d_2\rangle)$$

which is the intended form.

$\boxed{3\leqslant|w|<2k,\ w=\langle d_1,d_2\rangle\star w',\ w'=\langle d_3\rangle\star w'',\ w \text{ c.o.}}$ (iv)

$$\mathrm{SORT}'_{2(n+1)}(w)=\tau_I\circ\partial_{H_{n+1}}(C''_{n+1}(d_1,d_2)\|\mathrm{SORT}'_{2n}(w'))=$$

$$= \sum_{d \in D} r_{n+1}(d) \cdot \tau_I \circ \partial_{H_{n+1}}(s_n(d_2) \cdot C''_{n+1}(sw(d, d_1)) \| \mathrm{SORT}'_{2n}(w')) +$$

$$+ s_{n+1}(d_1) \cdot \tau_I \circ \partial_{H_{n+1}}((r_n(\mathrm{empty}) \cdot C'_{n+1}(d_2) +$$

$$+ \sum_{e \in D} r_n(e) \cdot C''_{n+1}(sw(d_2, e))) \| \mathrm{SORT}'_{2n}(w'))$$

(induction hypothesis)

$$= \sum_{d \in D} r_{n+1}(d) \cdot \tau \cdot \tau_I \circ \partial_{H_{n+1}}(s_n(d_2) \cdot C''_{n+1}(sw(d, d_1)) \|$$

$$\| \mathrm{SORT}'_{2n}(sw(\langle d_2 \rangle \star \langle d_3 \rangle \star w'))) +$$

$$+ s_{n+1}(d_1) \cdot \tau \cdot \tau_I \circ \partial_{H_{n+1}}(C''_{n+1}(sw(d_2, d_3)) \| \mathrm{SORT}'_{2n}(sw(w'')))$$

$$= \sum_{d \in D} r_{n+1}(d) \cdot \mathrm{SORT}'_{2(n+1)}(sw(\langle d \rangle \star w)) + s_{n+1}(d_1) \cdot \mathrm{SORT}'_{2(n+1)}(sw(\langle d_2 \rangle \star w'))$$

again the intended form.

$$\boxed{|w| = 2k,\ w = \langle d_1, d_2 \rangle \star w',\ w' = \langle d_3 \rangle \star w',\ w'' = w''' \star \langle d_k \rangle} \qquad \text{(v)}$$

$$\mathrm{SORT}'_{2(n+1)}(w) = \tau_I \circ \partial_{H_{n+1}}(C''_{n+1}(d_1, d_2) \| \mathrm{SORT}'_{2n}(w'))$$

$$= \sum_{d \in D} r_{n+1}(d) \cdot \tau_I \circ \partial_{H_{n+1}}(s_n(d_2) \cdot C''_{n+1}(sw(d, d_1)) \| \mathrm{SORT}'_{2n}(w')) +$$

$$+ s_{n+1}(d_1) \cdot \tau_I \circ \partial_{H_{n+1}}\{(r_n(\mathrm{empty}) \cdot C'_{n+1}(d_2) +$$

$$+ \sum_{f \in D} r_n(f) \cdot C''_{n+1}(sw(d_2, f))) \| \mathrm{SORT}'_{2n}(w')\}$$

(induction hypothesis)

$$= \sum_{d \in D} r_{n+1}(d) \cdot \tau \cdot \tau_I \circ \partial_{H_{n+1}}(C''_{n+1}(sw(d, d_1)) \| \mathrm{SORT}_{2n}(sw(\langle d_2 \rangle \star w'''))) +$$

$$+ s_{n+1}(d_1) \cdot \tau \cdot \tau_I \circ \partial_{H_{n+1}}(C''_{n+1}(sw(d_2, d_3)) \| \mathrm{SORT}'_{2n}(sw(w'')))$$

$$= \sum_{d \in D} r_{n+1}(d) \cdot \mathrm{SORT}'_{2(n+1)}(sw(\langle d \rangle \star w)) +$$

$$+ s_{n+1}(d_1) \cdot \mathrm{SORT}'_{2(n+1)}(sw(\langle d_2 \rangle \star w'))$$

which is the intended form.

From this we can conclude $\mathrm{SORT}'(w)$ satisfies the specification in Table 9. Using RSP we get for all $k \geqslant 1$, w c.o. and $0 \leqslant |w| \leqslant 2k$:

$$\mathrm{SORT}'_{2k}(w) = \mathrm{SORT}''_{2k}(w).$$

This ends the proof of $(\star)$. □

PROOF (**). $\mathrm{SORT}'_{2k}(\epsilon)=\mathrm{SORT}^*_{2k}(\epsilon)$, is proved by induction on k.

$k=1$ $\mathrm{SORT}'_2(\epsilon)=\tau_I\circ\partial_{H_1}(C_1\|C_0)=\mathrm{SORT}^*_2(\epsilon)$

$k>1$ $\mathrm{SORT}'_{2k}(\epsilon)=\tau\circ\partial_{H_k}(C_k\|\mathrm{SORT}'_{2k-2}(\epsilon))$

$=\tau_I\circ\partial_{H_k}(C_k\|\tau_I\circ\partial_{H_{k-1}}(C_{k-1}\|\cdots\|C_0))$

Because $H_k\cap I=\varnothing$ one can rewrite this to

$=\tau_I\circ\partial_{H_k}\circ\tau_I(C_k\|\partial_{H_{k-1}}(C_{k-1}\|\cdots\|C_0))$

Because $\alpha(C_k)|\alpha(\partial_{H_{k-1}}(C_{k-1}\|\cdots\|C_0))\cap I=\varnothing$ axiom CA2 can be applied

$=\tau_I\circ\partial_{H_k}(C_k\|\partial_{H_{k-1}}(C_{k-1}\|\cdots\|C_0))$

Because $\alpha(C_k)|\alpha(C_{k-1}\|\cdots\|C_0)\cap H_{k-1}\subseteq H_{k-1}$ axiom CA1 can be applied

$=\tau_I\circ\partial_{H_k}\circ\partial_{H_{k-1}}(C_k\|C_{k-1}\|\cdots\|C_0)$

Using axiom CA5 the induction step is completed

$=\tau_I\circ\partial_{H_k}(C_k\|C_{k-1}\|\cdots\|C_0)=\mathrm{SORT}^*_{2k}(\epsilon)$

This ends the proof of (**). □

Comparing the specification of Table 9 to the one in Table 10 we directly conclude that $\mathrm{SORT}'''_{2k}(\epsilon)\vDash\mathrm{SORT}''_{2k}(\epsilon)$ follows from the definition of $\vDash$.

Because of the transitivity of $\vDash$ and since $x=y\Rightarrow x\vDash y$ we only need to prove the equation $\mathrm{SORT}'''_{2k}(\epsilon)=\mathrm{SORT}_{2k}(\varnothing)$ to prove $\mathrm{SORT}_{2k}(\varnothing)\vDash\mathrm{SORT}^*_{2k}(\epsilon)$. Consider the specification in Table 10 then it follows that $\mathrm{SORT}_{2k}(B_w)$ (B_w denotes the bag containing the elements of w, w c.o., $0\leqslant|w|\leqslant 2k$) satisfies this specification, substituting it for $\mathrm{SORT}'''_{2k}(w)$. It is crucial here that of any correctly ordered sequence the first element also is the minimal element of that sequence. Using RSP we can conclude $\mathrm{SORT}'''_{2k}(\epsilon)=\mathrm{SORT}_{2k}(\varnothing)$. So, $\mathrm{SORT}_{2k}(\varnothing)\vDash\mathrm{SORT}^*_{2k}(\epsilon)$. □

$$\mathrm{SORT}'''_{2k}(\epsilon)=s_k(\mathrm{empty})\cdot\mathrm{SORT}'''_{2k}(\epsilon)+\sum_{d\in D}r_k(d)\cdot\mathrm{SORT}'''_{2k}(\langle d\rangle)$$

$$\mathrm{SORT}'''_{2k}(\langle d_1\rangle\star w)=\sum_{d\in D}r_k(d)\cdot\mathrm{SORT}'''_{2k}(sw(\langle d\rangle\star\langle d_1\rangle\star w))+$$
$$+s_k(d_1)\cdot\mathrm{SORT}'''_{2k}(sw(w'))\quad\text{for } |\langle d_1\rangle\star w|<2k-1,\ \langle d_1\rangle\star w\ \text{ c.o.}$$

$$\mathrm{SORT}'''_{2k}(\langle d_1\rangle\star w\star\langle d_{2k}\rangle)=\sum_{d\in D}r_k(d)\cdot\Omega+s_k(d_1)\cdot\mathrm{SORT}'''_{2k}(sw(\langle d_1\rangle\star w\star\langle d_{2k}\rangle))$$
$$\text{for } |\langle d_1\rangle\star w\star\langle d_{2k}\rangle|=2k,\ w\ \text{c.o.}$$

TABLE 10. Intermediate specification of restricted sorting machine using Ω

5. A PALINDROME-RECOGNISER WITH UNRESTRICTED CAPACITY

In this section we will remove the restriction on the length of the input string of Section 3. Thus no terminal cell is present. The specification of this machine is given in Table 11 (compare with Table 2). k is the name of the input/output channel. Note that the subscript k in $\mathrm{PAL}_k(w)$ has nothing to do with its capacity. It just indicates the name of the input/output channel.

$$\mathrm{PAL}_k(\epsilon)=\sum_{x\in S} r_k(x)\cdot s_k(\mathrm{true})\cdot\mathrm{PAL}_k(x)+s_k(\mathrm{true})\cdot PAL_k(\epsilon)$$

$$\mathrm{PAL}_k(w)=\sum_{x\in S} r_k(x)\cdot s_k(\mathrm{ispal}(xw))\cdot\mathrm{PAL}_k(xw) \quad (|w|>0)$$

TABLE 11. Specification of palindrome recogniser with unbounded capacity

When more capacity is needed, a new cell is created. A cell can be in two major states: it is a cell left from the last cell or the last cell in the chain. The last cell is always empty. When the last cell is filled it creates a new cell on the right.

As an extension of ACP the mechanism of process creation is described in [3]. With this mechanism it is possible to create a new process concurrent with the present one. To make process creation possible a creation atom and a special operator E_ϕ are introduced. We assume that a creation atom is neither a result of a communication nor communicates with another atom. For all $d\in D$, where D is a set of data, creation atoms $cr(d)$ are introduced. This in combination with the special operator E_ϕ gives a mechanism to create a process $\phi(d)$. When E_ϕ is applied to a process all the atoms which are not creation atoms will be executed without any problem. Whenever a creation atom is detected a new process will be started. The axioms for process creation are formulated in Table 12.

$\mathrm{E}_\phi(a)=a$	
$\mathrm{E}_\phi(\tau)=\tau$	
$\mathrm{E}_\phi(cr(d))=\overline{cr}(d)\cdot\mathrm{E}_\phi(\phi(d))$	
$\mathrm{E}_\phi(\tau x)=\tau\cdot\mathrm{E}_\phi(x)$	
$\mathrm{E}_\phi(ax)=a\cdot\mathrm{E}_\phi(x)$	$a\notin cr(D)$, $a\in \mathrm{A}\cup\{\delta\}$
$\mathrm{E}_\phi(cr(d)\cdot x)=\overline{cr}(d)\cdot\mathrm{E}_\phi(\phi(d)\|\|x)$	$d\in D$
$\mathrm{E}_\phi(x+y)=\mathrm{E}_\phi(x)+\mathrm{E}_\phi(y)$	

TABLE 12. Axioms for process creation

The atom $\overline{cr}(d)$ indicates that the process $\phi(d)$ has been created.

Since a creation atom neither communicates nor is the result of a communication, the following propositions hold.

PROPOSITION 1. *For all closed terms* x: $E_\phi \circ E_\phi(x) = E_\phi(x)$.

PROPOSITION 2. *For all closed terms* x, y: $E_\phi(x \| y) = E_\phi(x) \| E_\phi(y)$.

We assume these propositions to hold for all recursively defined processes. An example of process creation is given below. This example can be found in [3].

EXAMPLE. $D = \{d\}$, $\phi(d) = a \cdot cr(d) \| b \cdot cr(d)$, $a|b = \delta$. When $P = E_\phi(cr(d))$ then using proposition 2 we have $P = \overline{cr}(d) \cdot (aP \| bP)$.

Now let's return to our palindrome-recogniser and see how, in this specific example, process creation works. We will first discuss an individual cell C_i which is pictured in Figure 8.

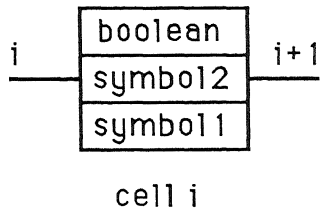

FIGURE 8. An individual cell C_i

Note that the names of the channels are reversed in comparison with Section 2.

A cell C_i can be in three states.

(0) The cell is the last cell and it is empty. When a symbol is received from the left a new cell is created. The symbol is stored in symbol1 and the cell enters state (1). The second possibility is that a true signal can be sent to the left. In this case, the cell remains in the same state.

(1) The cell contains one symbol. It can receive a symbol from the left and a boolean value from the right in either order. These are stored in locations symbol2 and boolean respectively. The cell enters state (2).

(2) The cell contains two symbols. We need the boolean value b to be calculated in the following way:

$$b = \mathbf{boolean} \wedge (\mathbf{symbol1} = \mathbf{symbol2})$$

The cell sends value b to the left and symbol2 to the right. The cell enters state (1) again.

A formal description of an individual cell is given in Table 13. C_i, C'_i, and

C''_i correspond to the states (0), (1) and (2) respectively.

$$C_i = \sum_{x \in S} r_i(x) \cdot cr(i+1) \cdot s_i(\text{true}) \cdot C'_i(x) + s_i(\text{true}) \cdot C_i$$

$$C'_i(x) = (\{\sum_{y \in S} r_i(y)\} \| \{\sum_{v \in B} r_{i+1}(v)\}) \cdot C''_i(x, y, v)$$

$$C''_i(x, y, v) = (s_i(|x = y| \text{ and } v) \| s_{i+1}(y)) \cdot C_i(x)$$

$$\phi(i+1) = C_{i+1}$$

TABLE 13. Specification of an individual cell

An example (the same example as pictured in Section 4) is written out in Figure 9 on the next page. A formal definition of the palindrome-recogniser with input/output channel k is given in Table 14.

$$\text{PAL}^*_k(\epsilon) = \tau_I \circ \partial_{H_k} \circ \text{E}_\phi(C_k)$$

Communications for $i \geqslant 1$: $c_i(d) = s_i(d) | r_i(d)$

Process creation for $i \geqslant 1$: $\phi(i) = C_i$

$$H_k = \{s_i(d), r_i(d) : d \in S \cup B,\ i > k\}$$

$$I = \{c_i(d), \overline{cr}(i) : d \in S \cup B,\ i \geqslant 1\}$$

TABLE 14. Formal definition of implementation of palindrome-recogniser

This definition is extended in the following Table 15.

$$\text{IPAL}_k(\epsilon) = \tau_I \circ \partial_{H_k} \circ \text{E}_\phi(C_k)$$

$$\text{IPAL}_k(x) = \tau_I \circ \partial_{H_k} \circ \text{E}_\phi(C'_k(x) \| \text{IPAL}_{k+1}(\epsilon))$$

$$\text{IPAL}_k(yx) = \tau_I \circ \partial_{H_k} \circ \text{E}_\phi(C'_k(x) \| \text{IPAL}_{k+1}(y))$$

$$\text{IPALH}_k(ywx) = \tau_I \circ \partial_{H_k} \circ \text{E}_\phi(C''_k(x, y, \text{ispal}(w)) \| \text{IPAL}_{k+1}(w))$$

$$\text{IPAL}_k(wx) = \tau_I \circ \partial_{H_k} \circ \text{E}_\phi(C'_k(x) \| \text{IPALH}_{k+1}(w)) \quad |w| \geqslant 2$$

H_k, I, communications, ϕ: see Table 14.

TABLE 15. Alternative implementation of the palindrome-recogniser $k \geqslant 1$

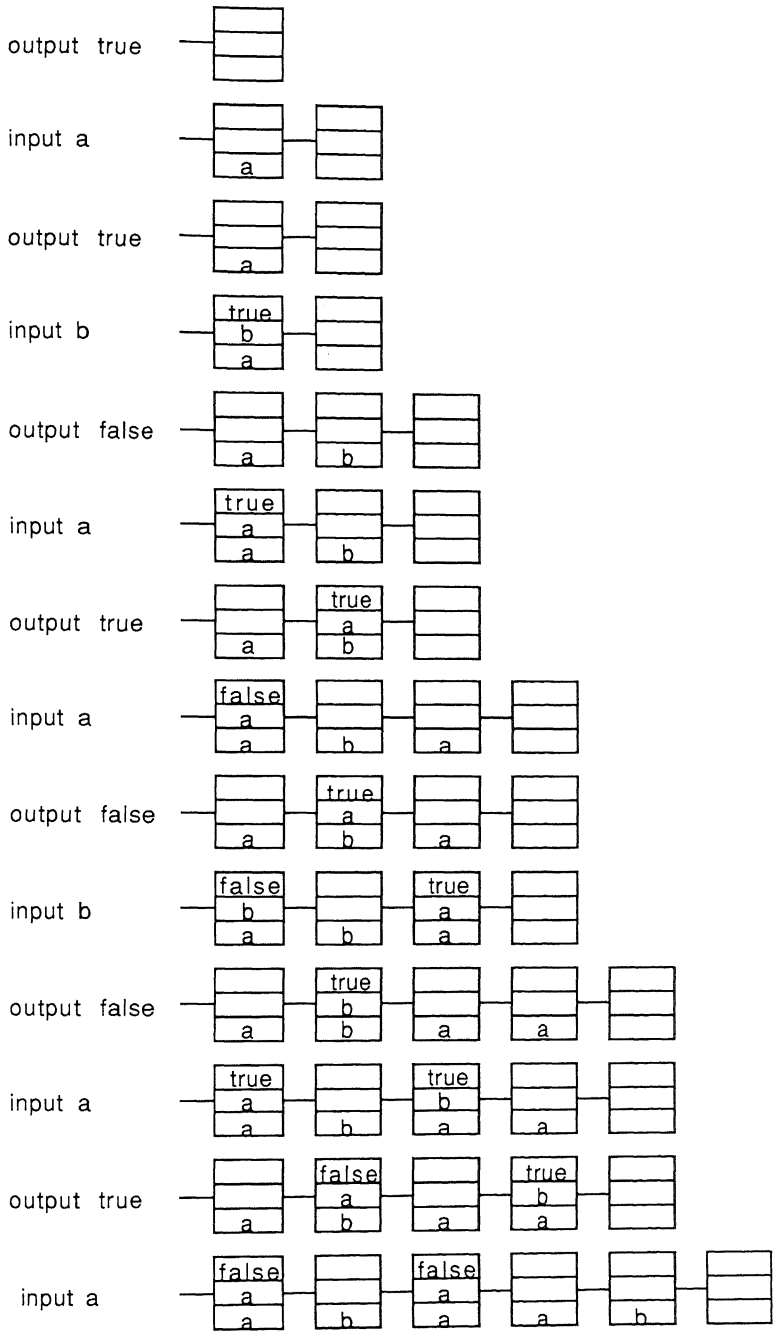

FIGURE 9. Example of unrestricted palindrome-recogniser

FACT. *For all* $k \geqslant 1$: $\mathrm{PAL}_k(\epsilon) = \mathrm{PAL}^*_k(\epsilon)$.

PROOF. Consider the definitions in Tables 14 and 15. It is immediate that for all k $\mathrm{PAL}^*_k(\epsilon) = \mathrm{IPAL}_k(\epsilon)$. We will prove that the processes given in Table 15 are specified by the specification formulated in Table 16. From Table 11 and Table 16 it is not difficult to see that $\mathrm{PAL}_k(\epsilon) = \mathrm{PAL}'_k(\epsilon)$. Substitute $\mathrm{PAL}_k(\epsilon)$ for $\mathrm{PAL}'_k(\epsilon)$ and $\tau \cdot \mathrm{PAL}_k(xw)$ for $\mathrm{PAL}'_k(xw)$. This is a solution of the specification in Table 16 and by RSP it follows that $\mathrm{PAL}_k(\epsilon) = \mathrm{PAL}'_k(\epsilon)$. So what we need to prove is that the process defined in Table 15 is specified by the specification in Table 16.

$$\mathrm{PAL}'_k(\epsilon) = \sum_{x \in S} r_k(x) \cdot s_k(\mathrm{true}) \cdot \mathrm{PAL}'_k(x) + s_k(\mathrm{true}) \cdot \mathrm{PAL}'_k(\epsilon)$$

$$\mathrm{PAL}'_k(w) = \tau \cdot \sum_{x \in S} r_k(x) \cdot \mathrm{PALH}'_k(xw) \quad |w| \geqslant 1$$

$$\mathrm{PALH}'_k(w) = \tau \cdot s_k(\mathrm{ispal}(w)) \cdot \mathrm{PAL}'_k(w) \quad |w| \geqslant 2$$

TABLE 16. Alternative specification of the palindrome-recogniser for $k \geqslant 1$

PROPOSITION. $\mathrm{IPAL}_k(\epsilon)$ *satisfies the specification in Table 16.*

PROOF. This is proved for all k simultaneously with induction on the length of the content of the palindrome-recogniser. The proof considers five cases where in each case the previous cases are assumed to hold for all k.

$w = \epsilon$ (i)

$$\mathrm{IPAL}_k(\epsilon) = \tau_I \circ \partial_{H_k} \circ \mathrm{E}_\phi(C_k)$$

$$= \sum_{x \in S} r_k(x) \cdot \tau_I \circ \partial_{H_k} \circ \mathrm{E}_\phi(s_k(\mathrm{true}) \cdot C'_k(x) \| C_{k+1}) + s_k(\mathrm{true}) \cdot \tau_I \circ \partial_{H_k} \mathrm{E}_\phi(C_k)$$

Using $\mathrm{E}_\phi(x \| y) = \mathrm{E}_\phi(x) \| \mathrm{E}_\phi(y)$, $\mathrm{E}_\phi(x) = \mathrm{E}_\phi \circ \mathrm{E}_\phi(x)$ and the axioms CA1, CA2, CA5 and CA7, and the fact that when I and H don't contain creation atoms E_ϕ can be pushed through the τ_I and ∂_H operators, we find

$$= \sum_{x \in S} r_k(x) \cdot s_k(\mathrm{true}) \cdot \tau_I \circ \partial_{H_k} \circ \mathrm{E}_\phi(C'_k(x) \| \tau_I \circ \partial_{H_{k+1}} \circ \mathrm{E}_\phi(C_{k+1}))$$

$$+ s_k(\mathrm{true}) \cdot \tau_I \circ \partial_{H_k} \circ \mathrm{E}_\phi(C_k)$$

$$= \sum_{x \in S} r_k(x) \cdot s_k(\text{true}) \cdot \tau_I \circ \partial_{H_k} \circ E_\phi(C'_k(x) \| \text{IPAL}_{k+1}(\epsilon)) +$$

$$+ s_k(\text{true}) \cdot \tau_I \circ \partial_{H_k} \circ E_\phi(C_k)$$

So for all $k \geqslant 1$:

$$\text{IPAL}_k(\epsilon) = \sum_{x \in S} r_k(x) \cdot s_k(\text{true}) \cdot \text{IPAL}_k(x) + s_k(\text{true}) \cdot \text{IPAL}_k(\epsilon).$$

$\boxed{w = x}$ (ii)

$$\text{IPAL}_k(x) = \tau_I \circ \partial_{H_k} \circ E_\phi(C'_k(x) \| \text{IPAL}_{k+1}(\epsilon))$$

using step (i) we get

$$= \tau_I((\sum_{y \in S} r_k(y) \cdot c_{k+1}(\text{true}) +$$

$$+ c_{k+1}(\text{true}) \cdot \sum_{y \in S} r_k(y)) \cdot \partial_{H_k} \circ E_\phi(C''_k(x, y, \text{true}) \| \text{IPAL}_{k+1}(\epsilon)))$$

$$= \tau \cdot \sum_{y \in S} r_k(y) \cdot \tau_I \circ \partial_{H_k} \circ E_\phi(C''_k(x, y, \text{true}) \| \text{IPAL}_{k+1}(\epsilon))$$

(using step (i) and T2).
So for all $k \geqslant 1$:

$$\text{IPAL}_k(x) = \tau \cdot \sum_{y \in S} r_k(y) \cdot \text{IPAL}_k(yx).$$

$\boxed{w = yx}$ (iii)

$$\text{IPAL}_k(yx) = \tau_I \circ \partial_{H_k} \circ E_\phi(C'_k(x) \| \text{IPAL}_{k+1}(y))$$

Using (ii) and T2,

$$= \tau \cdot \sum_{z \in S} r_k(z) \cdot \tau_I \circ \partial_{H_k} \circ E_\phi(C''_k(x, z, \text{true}) \| \text{IPAL}_{k+1}(y)))$$

So for all $k \geqslant 1$:

$$\text{IPAL}_k(yx) = \tau \cdot \sum_{z \in S} r_k(z) \cdot \text{IPALH}_k(zyx).$$

$\boxed{w = yvx,\ |v| \geqslant 0}$ (iv)

$$\text{IPALH}_k(yvx) = \tau_I \circ \partial_{H_k} \circ E_\phi(C''_k(x, y, \text{ispal}(v)) \| IPAL_{k+1}(v))$$

Using the induction hypothesis we obtain:

$$= \tau_I(s_k(|x=y| \wedge \text{ispal}(v)) \cdot c_{k+1}(y) +$$
$$+ c_{k+1}(y) \cdot s_k(|x=y| \wedge \text{ispal}(v)) \cdot \partial_{H_k} \circ E_\phi(C'_k(x) \| \text{IPALH}_{k+1}(yv)))$$

$$= \tau \cdot s_k(|x=y| \wedge \text{ispal}(v)) \cdot \tau_I \circ \partial_{H_k} \circ E_\phi(C'_k(x) \| \text{IPALH}_{k+1}(yv)) \qquad \text{(using T2)}$$

So for all $k \geqslant 1$:

$$\text{IPALH}_k(yvx) = \tau \cdot s_k(\text{ispal}(yvx)) \cdot \text{IPAL}_k(yvx).$$

$w = vx, |v| \geqslant 2$ (v)

$$\text{IPAL}_k(vx) = \tau_I \circ \partial_{H_k} \circ E_\phi(C'_k(x) \| \text{IPALH}_{k+1}(v))$$

Using the induction hypothesis and T2 we obtain:

$$= \tau \cdot \sum_{y \in S} r_k(y) \cdot \tau_I \circ \partial_{H_k} \circ E_\phi(C''_k(x,y,\text{ispal}(v)) \| \text{IPAL}_{k+1}(v)))$$

So for all $k \geqslant 1$:

$$\text{IPAL}_k(vx) = \tau \cdot \sum_{y \in S} r_k(y) \cdot \text{IPALH}_k(yvx).$$

This ends the proof of the proposition. Then, using RSP as described above we obtain the desired equality $\text{PAL}^*_k(\epsilon) = \text{PAL}_k(\epsilon)$. □

6. The sorting machine with unrestricted capacity

After handling the restricted sorting machine in Section 5 we now come to the sorting machine with unrestricted capacity. The specification of a sorting machine with infinite capacity, which we call sorting machine from now on, is given in Table 17. Note that the subscript in $\text{SORT}_k(B)$ indicates the name of the input/output channel.

The implementation of the sorting machine is different from the implementation of the restricted sorting machine. The number of cells of the restricted sorting machine was fixed but the sorting machine is built by using a variable number of cells. The last cell is always empty. When this last cell receives a number from the left it creates a new cell. When a stop signal is received the cell stops working and disappears.

$$\text{SORT}_k(\varnothing) = \sum_{d \in D} r_k(d) \cdot \text{SORT}_k(\{d\}) + s_k(\text{empty}) \cdot \text{SORT}_k(\varnothing)$$
$$\text{SORT}_k(B) = \sum_{d \in D} r_k(d) \cdot \text{SORT}_k(B \cup \{d\}) + s_k(\mu B) \cdot \text{SORT}_k(B - \{\mu B\}) \qquad |B| > 0$$

Table 17. Specification of a sorting machine with infinite capacity for $k \geqslant 1$

Just like the cells of the restricted sorting machine these cells can contain

two numbers in MIN and MAX. The content of MIN is less than or equal to the content of MAX. The last cell can create a new cell when needed. A cell C_i is pictured in Figure 10. Note that the names of the channels are reversed in comparison with Section 5.

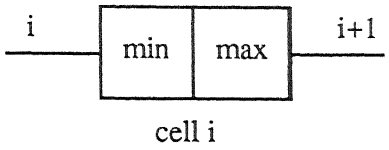

FIGURE 10. An individual cell

A cell can be in three states.

(0) The cell is empty. From the left the cell can receive a number or the stop signal. When the cell receives a number from the left a new cell on the right is created. The number is stored in MIN. The cell enters state (1). When the cell receives the stop signal the cell stops working and disappears.

(1) The cell contains one number, stored in MIN. (a) The cell can receive a number from the left. The minimum of the content of MIN and the received number is stored in MIN. The larger of the two numbers is stored in MAX. State (2) is entered. (b) The cell can send the content of MIN to the left. Because the cell has become empty a stop signal is send to the right. The cell enters state (0).

(2) The cell contains two numbers, stored in MIN and MAX. (a) When a number is received from the left, the content of MAX is send to the right. The minimum of the content of MIN and the received number is stored in MIN. The other number is stored in MAX. The cell remains in the same state. (b) When a number is sent to the left two possibilities arise. If an empty signal is received from the right, then the content of MAX is stored in MIN, MAX becomes empty and the cell changes to state (1). Receiving a number from the right doesn't change the state of the cell. The minimum of the content of MAX and the received number is stored in MIN. The other number is stored in MAX.

A formal description of an individual cell is given in Table 18.

$$C_i = \sum_{d \in D} r_i(d) \cdot cr(i+1) \cdot C'_i(d) + r_i(\text{stop}) + s_i(\text{empty}) \cdot C_i$$

$$C'_i(d) = \sum_{e \in D} r_i(e) \cdot C''_i(sw(d, e)) + s_i(d) \cdot s_{i+1}(\text{stop}) \cdot C_i$$

$$C''_i(d,e) = \sum_{f \in D} r_i(f) \cdot s_{i+1}(e) \cdot C''_i(sw(d, f)) +$$

$$+ s_i(d) \cdot (\sum_{f \in D} r_{i+1}(f) \cdot C''_i(sw(f, e)) + r_{i+1}(\text{empty}) C'_i(e))$$

$$sw(d,e) = (\min(d, e), \max(d,e))$$

TABLE 18. Specification of cell i, $i \geqslant 0$

In Figure 11 below a chain configuration of cells is pictured to illustrate how the unbounded sorting machine works. Note how cells are created and killed. These cells are connected in the same way as is done in the restricted sorting machine implementation. The behaviour of the sorting machine with input/output channel k is described in Table 19. Note that the subscript of $\mathrm{SORT}^*_k(\epsilon)$ indicates the name of the input/output channel, and has nothing to do with its capacity.

$\mathrm{SORT}^*_k(\epsilon) = \tau_I \circ \partial_{H_k}(\mathrm{E}_\phi(C_k)), \quad k \geqslant 1$

Communication: $c_i(d) = r_i(d) | s_i(d), \ d \in D \cup \{\text{empty, stop}\}), \ i \geqslant 1$

Process creation: $\phi(i) = C_i, \quad i \geqslant 2$

$H_k = \{r_i(d), s_i(d): d \in D \cup \{\text{empty, stop}\}, i \geqslant k+1\} \cup \{r_k(\text{stop})\}$

$I = \{c_i(d), \overline{cr}(i+1): d \in D \cup \{\text{empty, stop}\}, i \geqslant 1\}$

TABLE 19. Formal description of a sorting machine with input/output channel k

FACT. *For all* $k \geqslant 1$ $\mathrm{SORT}^*_k(\epsilon) = \mathrm{SORT}_k(\varnothing)$.

PROOF. The definitions of c.o. and *sw* (see Section 5) will be used in this proof. Similarly to the restricted sorting machine section an intermediate specification is given in Table 20. This specification includes the possibility to stop the sorting machine. An extended definition for the chain of cells is given in Table 21.

$$\mathrm{SORT}'_k(\epsilon) = \sum_{d \in D} r_k(d) \cdot \mathrm{SORT}''_k(\langle d \rangle) + r_k(\text{stop}) + s_k(\text{empty}) \cdot \mathrm{SORT}''_k(\epsilon)$$

$$\mathrm{SORT}''_k(\langle d_1 \rangle * w) = \sum_{d \in D} r_k(d) \cdot \mathrm{SORT}''_k(sw(\langle d \rangle * \langle d_1 \rangle * w)) +$$

$$+ s_k(d_1) \cdot \mathrm{SORT}'_k(sw(w)) \quad \langle d_1 \rangle * w \text{ c.o.}$$

TABLE 20. Intermediate specification of unrestricted sorting machine

By putting $r_k(\text{stop})$ in H_k the machine described above is obtained. This step is necessary to make a proof by induction possible. However, the desired equation is obtained after abstraction from $r_k(\text{stop})$ in $\mathrm{SORT}'_k(\epsilon)$.

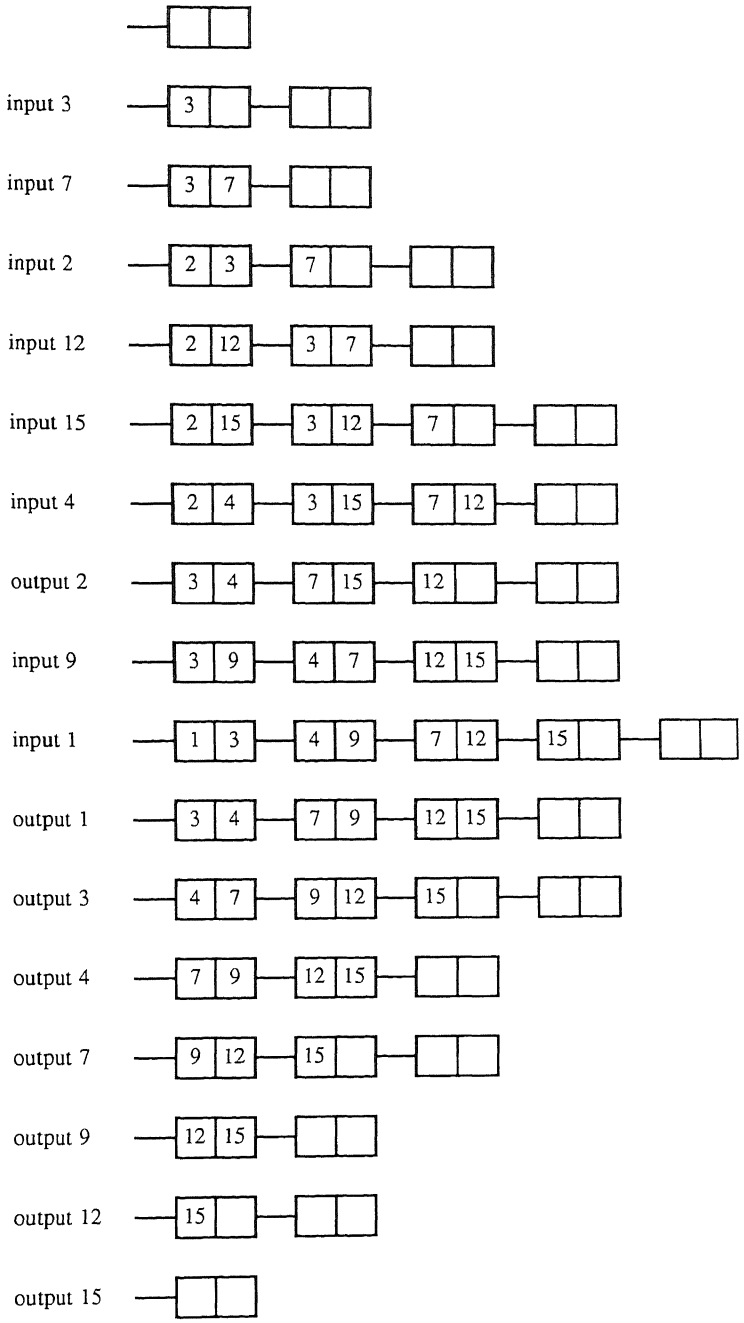

FIGURE 11. Example of unrestricted sorting machine

$\mathrm{SORT}'_k(\epsilon)=\tau_I\circ\partial_{H'_k}\mathrm{E}_\phi(C_k)$
$\mathrm{SORT}'_k(\langle d_1\rangle)=\tau_I\circ\partial_{H'_k}\mathrm{E}_\phi(C'_k(d_1)\|C_{k+1})$
$\mathrm{SORT}'k(\langle d_1,d_2\rangle)=\tau_I\circ\partial_{H'_k}\circ\mathrm{E}_\phi(C''_k(d_1,d_2)\|C_{k+1})$ $\quad\langle d_1,d_2\rangle$ c.o.
$\mathrm{SORT}'_k(\langle d_1,d_2\rangle\star w)=\tau_I\circ\partial_{H'_k}\circ\mathrm{E}_\phi(C''_k(d_1,d_2)\|\mathrm{SORT}'_k(w))$ $\quad\langle d_1,d_2\rangle\star w$ c.o., $\|w\|>0$
Communication: $c_i(d)=r_i(d)\|s_i(d)$ $\quad d\in D\cup\{\text{empty, stop}\},\ i\geqslant 1$
Process creation: $\phi(i)=C_i$ $\quad i\geqslant 2$
$H'_k=\{r_i(d), s_i(d)\colon d\in D\cup\{\text{empty, stop}\},\ i\geqslant k+1\}$
$I=\{c_i(d), \overline{cr}(i+1)\colon d\in D\cup\{\text{empty, stop}\},\ i\geqslant 1\}$

TABLE 21. Alternative definition for the implementation

PROPOSITION.
(I) *for all* $k\geqslant 1, w$ c.o.: $\mathrm{SORT}''_k(w)=\mathrm{SORT}'_k(w)$
(II) *for all* $k\geqslant 1, \mathrm{SORT}_k(\varnothing)=\partial_{H^*_k}(\mathrm{SORT}'_k(\epsilon))$, *where* $H^*_k=\{r_k(\text{stop})\}$
(III) *for all* $k\geqslant 1, \mathrm{SORT}^*_k(\epsilon)=\partial_{H^*_k}(\mathrm{SORT}'_k(\epsilon))$, *where* $H^*_k=\{r_k(\text{stop})\}$

PROOF. (I) This will be done by induction on the length of the content simultaneously for all k. The cases (i), (ii) and (iii) are the basic steps. Case (iv) is the induction step.

(i) for all k

$$\mathrm{SORT}'_k(\epsilon)=\sum_{d\in D} r_k(d)\cdot\mathrm{SORT}'_k(\langle d\rangle)+r_k(\text{stop})+s_k(\text{empty})\cdot\mathrm{SORT}'_k(\epsilon)$$

$$\mathrm{SORT}_k(\epsilon)=\tau_I\circ\partial_{H'_k}\circ\mathrm{E}_\phi(C_k)$$

$$=\tau_I\circ\partial_{H'_k}\circ\mathrm{E}_\phi\Big(\sum_{d\in D} r_k(d)\cdot cr(k+1)\cdot C'_k(d)+r_k(\text{stop})+s_k(\text{empty})\cdot C_k\Big)$$

$$=\sum_{d\in D} r_k(d)\cdot\tau_I\circ\partial_{H'_k}\circ\mathrm{E}_\phi(C'_k(d)\|C_{k+1})+r_k(\text{stop})+s_k(\text{empty})\cdot\tau_I\circ\partial_{H'_k}\mathrm{E}_\phi(C_k)$$

$$=\sum_{d\in D} r_k(d)\cdot\mathrm{SORT}'_k(\langle d\rangle)+r_k(\text{stop})+s_k(\text{empty})\cdot\mathrm{SORT}'_k(\epsilon)$$

(ii) $\mathrm{SORT}'_k(\langle d_1\rangle)=\sum_{d\in D} r_k(d)\cdot\mathrm{SORT}'_k(sw(\langle d,d_1\rangle))+s_k(d_1)\cdot\mathrm{SORT}'_k(\epsilon)$

$$\mathrm{SORT}'_k(\langle d_1\rangle)=\tau_I\circ\partial_{H'_k}\circ\mathrm{E}_\phi(C'_k(d_1)\|C_{k+1})=$$

$$=\sum_{d\in D} r_k(d)\cdot\tau_I\circ\partial_{H'_k}\circ\mathrm{E}_\phi(C''_k(sw(d,d_1))\|C_{k+1})+s_k(d_1)\cdot\tau_I\circ\partial_{H'_k}\circ\mathrm{E}_\phi(C_k)$$

$$= \sum_{d \in D} r_k(d) \cdot \mathrm{SORT}'_k(sw(\langle d, d_1 \rangle)) + s_k(d_1) \cdot \mathrm{SORT}'_k(\epsilon)$$

(iii) $\mathrm{SORT}'_k(\langle d_1, d_2 \rangle) =$

$$= \sum_{d \in D} r_k(d) \cdot \mathrm{SORT}'_k(sw(\langle d, d_1, d_2 \rangle)) + s_k(d_1) \cdot \mathrm{SORT}'_k(\langle d_2 \rangle) \quad \langle d_1, d_2 \rangle \text{ c.o.}$$

$$\mathrm{SORT}'_k(\langle d_1, d_2 \rangle) = \sum_{d \in D} r_k(d) \cdot \tau_I \circ \partial_{H'_k} \circ \mathrm{E}_\phi(C''_k(d_1, d_2) \| C_{k+1})$$

$$= \sum_{d \in D} r_k(d) \cdot \tau_I \circ \partial_{H'_k} \circ \mathrm{E}_\phi(C''_k(sw(d, d_1)) \| C'_{k+1}(d_2) \| C_{k+2}) +$$

$$+ s_k(d_1) \cdot \tau_I \circ \partial_{H'_k} \circ \mathrm{E}_\phi(C'_k(d_2) \| C_{k+1}).$$

Using the standard concurrency axioms and $\mathrm{E}_\phi(x \| y) = \mathrm{E}_\phi(x) \| \mathrm{E}_\phi(y)$ we obtain:

$$\tau_I \circ \partial_{H'_k} \circ \mathrm{E}_\phi(C''_k(sw(d, d_1)) \| C'_{k+1}(d_2) \| C_{k+2}) =$$

$$= \tau_I \circ \partial_{H'_k}(\mathrm{E}_\phi(C''_k(sw(d, d_1))) \| \mathrm{E}_\phi(C'_{k+1}(d_2) \| C_{k+2})).$$

Applying conditional axioms CA, $\mathrm{E}_\phi \circ \mathrm{E}_\phi(x) = \mathrm{E}_\phi(x)$ and using the fact that when I and H don't contain creation atoms E_ϕ can be pushed through the τ_I and ∂_H operators this last expression becomes

$$= \tau_I \circ \partial_{H'_k} \circ \mathrm{E}_\phi(C''_k(sw(d, d_1)) \| \tau_I \circ \partial_{H'_{k+1}} \circ \mathrm{E}_\phi(C'_{k+1}(d_2) \| C_{k+2})).$$

Using the definitions in Table 21:

$$= \tau_I \circ \partial_{H'_k} \circ \mathrm{E}_\phi(C''_k(sw(d, d_1)) \| \mathrm{SORT}'_{k+1}(\langle d_2 \rangle)) = \mathrm{SORT}'_k(sw(\langle d, d_1, d_2 \rangle)).$$

Making this observation the desired result is obtained:

$$\mathrm{SORT}'_k(\langle d_1, d_2 \rangle) =$$

$$= \sum_{d \in D} r_k(d) \cdot \mathrm{SORT}'_k(sw(\langle d, d_1, d_2 \rangle)) + s_k(d_1) \cdot \mathrm{SORT}'_k(\langle d_2 \rangle).$$

(iv) This case is the induction step. The proposition will be proved for all k and w, $|w| \geqslant 3$, assuming it has already been proved for all $k \geqslant 1$ and w', $|w'| < |w|$. In this proof $w = \langle d_1, d_2 \rangle \star v$ is c.o., $|v| \geqslant 1$ and $v = \langle d_3 \rangle \star v'$.

$$\mathrm{SORT}'_k(w) = \sum_{d \in D} r_k(d) \cdot \mathrm{SORT}'_k(sw(\langle d \rangle \star w)) + s_k(d_1) \cdot \mathrm{SORT}'_k(sw(\langle d_2 \rangle \star v))$$

$$\mathrm{SORT}'_k(\langle d_1, d_2 \rangle \star v) = \tau_I \circ \partial_{H'_k} \circ \mathrm{E}_\phi(C''_k(d_1, d_2) \| \mathrm{SORT}'_{k+1}(v))$$

Using the induction hypothesis on $\mathrm{SORT}'_{k+1}(v)$, $|v| \geqslant 1$, we obtain:

$$= \sum_{d \in D} r_k(d) \cdot \tau_I \circ \partial_{H'_k} \circ \mathrm{E}_\phi(C''_k(sw(d, d_1)) \| \mathrm{SORT}''_{k+1}(sw(\langle d_2 \rangle \star v)))$$

$$+ s_k(d_1) \cdot \tau_I \circ \partial_{H'_k} \mathrm{E}_\phi(C''_k(sw(d_2, d_3)) \| \mathrm{SORT}'_{k+1}(v')).$$

Considering the definition of sw we obtain:

$$= \sum_{d \in D} r_k(d) \cdot \mathrm{SORT}'_k(sw(\langle d \rangle * w)) + s_k(d_1) \cdot \mathrm{SORT}'_k(sw(\langle d_2 \rangle * v))$$

Using RSP we have the desired result. This ends the proof of proposition I.

PROOF. (II) For w c.o. $\mathrm{SORT}'_k(w)$ satisfies the specification in Table 20. Then it is easy to deduce that $\partial_{H^*_k}(\mathrm{SORT}'_k(\epsilon))$ is specified by the specification in the following Table 22.

$$\partial_{H^*_k}(\mathrm{SORT}'_k(\epsilon)) = \sum_{d \in D} r_k(d) \cdot \partial_{H^*_k}(\mathrm{SORT}'_k(\langle d \rangle)) + s_k(\mathrm{empty}) \cdot \partial_{H^*_k}(\mathrm{SORT}'_k(\epsilon))$$

$$\partial_{H^*_k}(\mathrm{SORT}_k(\langle d_1 \rangle * w)) = \sum_{d \in D} r_k(d) \cdot \partial_{H^*_k}(\mathrm{SORT}'_k(sw(\langle d, d_1 \rangle * w))) + s_k(d_1) \cdot \partial_{H^*_k}(\mathrm{SORT}'_k(sw(w))) \qquad \langle d_1 \rangle * w \text{ c.o.}$$

TABLE 22. Specification of $\partial_{H^*_k}(\mathrm{SORT}'_k(\epsilon))$

To prove proposition II substitute for all c.o. w $\mathrm{SORT}_k(B_w)$ for $\partial_{H^*_k}(\mathrm{SORT}'_k(w))$ where B_w is the bag containing the elements in the sequence w. Because the first element of a c.o. w is the minimal element of the sequence it is easy to see that $\mathrm{SORT}_k(\varnothing)$ satisfies the specification in Table 22. Then, with RSP the equation in proposition II is proved.

PROOF. (III) This is proved using the conditional axioms CA5 and CA7.
We find:
for all $k \geqslant 1$:

$$\partial_{H^*_k}(\mathrm{SORT}'_k(\epsilon)) = \partial_{H^*_k} \circ \tau_I \circ \partial_{H'_k} \circ \mathrm{E}_\phi(C_k) = \tau_I \circ \partial_{H^*_k \cup H'_k} \circ \mathrm{E}_\phi(C_k) = \mathrm{SORT}^*_k(\epsilon)$$

so we can conclude for all k $\mathrm{SORT}^*_k(\epsilon) = \mathrm{SORT}_k(\varnothing)$. □

REFERENCES

1. J.A. BERGSTRA (1989). *A Process Creation Mechanism in Process Algebra.* This volume.
2. J.A. BERGSTRA, J.W. KLOP (1986). Algebra of communicating processes. J.W. DE BAKKER, M. HAZEWINKEL, J.K. LENSTRA (eds.). *Mathematics and Computer Science,* CWI Monograph 1, North-Holland, Amsterdam, 89-138.
3. J.A. BERGSTRA, J.W. KLOP, E.-R. OLDEROG (1987). Failures without chaos: a new process semantics for fair abstraction. M. WIRSING (ed.). *Proc. IFIP Conf. on Formal Description of Programming Concepts - III,* Ebberup 1986, North-Holland, 77-103.

4. S.D. BROOKES, C.A.R. HOARE, A.W. ROSCOE (1984). A theory of communicating sequential processes. *J. ACM 31*, 560-599.
5. J.C. EBERGEN (1986). *A Technique to Design Delay-Insensitive VLSI Circuits,* CWI Report CS-R8622, Centre for Mathematics and Computer Science, Amsterdam.
6. M. HENNESSY (1986). Proving systolic systems correct. *TOPLAS 8(3),* 344-387.
7. C.P.J. KOYMANS, J.C. MULDER (1989). *A Modular approach to Protocol Verification using Process Algebra.* This volume.
8. K.T. KUNG (1979). Let's design algorithms for VLSI systems. *Proceedings of the Conference on VLSI: Architecture, Design, Fabrication,* California Institute of Technology.
9. C.A. MEAD, L.A CONWAY (1980). *Introduction to VLSI-systems,* Addison-Wesley Publ. Comp., Reading, Massachusetts.
10. G.J. MILNE (1983). CIRCAL: a calculus for circuit description. *Integration 1,* 121-160.
11. M. REM (1983). Partially ordered computations with applications to VLSI-design. J. DE BAKKER, J. VAN LEEUWEN (eds.). *Proc. Found. of Comp. Sci. IV.2,* MC Tract 159, Centre for Mathematics and Computer Science, Amsterdam, 1-44.