Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

L. Kossen, W.P. Weijland

Verification of a systolic algorithm for string comparison

Computer Science/Department of Software Technology Report CS-R8734 July

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

Verification of a systolic algorithm for string comparison

L. Kossen

W.P. Weijland

Dept. of Computer Science, University of Amsterdam,

P.O. Box 41882, 1009 DB Amsterdam, The Netherlands

Abstract: A self-timed systolic system computing the edit distance between two strings is proved correct by means of an algebraical concurrency theory ACP (Algebra of Communicating Processes, [BK1]). A systolic system is a system consisting of a great number of concurrently operating and cooperating elements. In the system described here (also discussed in [LL]), the flow of control is regulated by the elements themselves: the system is self-timed. A formal approach can be helpful to construct complex systems such as VLSI-circuits. Other verifications of systolic algorithms can be found in [KW] and [WE].

Key words and phrases: concurrency, process algebra, synchronous communication, asynchronous cooperation, self-timed system, systolic system, VLSI, correctness proof.
1985 Mathematics subject classification:
68Q35, 68Q60, 68Q10, 68Q55.
1982 CR categories: B.7.1, D.2.4, F.1.2, F.3.2

Note: Partial support received from the European Communities under ESPRIT contract no. 432, An Integrated Formal Aproach to Industrial Software Development (Meteor).
This report will be submitted for publication elsewhere.

1. INTRODUCTION

In the current research on (hardware) verification one of the main goals is to find strong proof systems and tools to verify the designs of algorithms and architectures. For instance, in the development of integrated circuits ('chips') the important stage of testing a prototype (to save the high costs of producing defective processors) can be dealt with much more efficiently, when a

Report CS-R8734
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

strong verification tool is available. Therefore, developping a verification theory has very high priority and is subject of study at many universities and scientific institutions.
However, working on detailed verification theories is not the only approach to this problem. Once having a basic theory, the development of case studies is of utmost importance to provide us with new ideas. Furthermore, one can focus on special design techniques, which turn out to fit conveniently in the theory. For example, because the regular configuration of these circuits, *systolic arrays* are very suitable for formal analysis and induction methods (see HENNESSY [HEN], KOSSEN and WEIJLAND [KW], MULDER and WEIJLAND [MW], REM [RE] and WEIJLAND [WE]). Indeed, systolic arrays have grown very popular in the last few years.
In this paper we will present a theory called *Algebra of Communicating Processes* (short: ACP, see BERGSTRA and KLOP [BK1]), which is an algebraical theory providing us with a formal description of concurrent processes. Some of the main theoretical results are presented, to make it possible for the reader to understand how to work in ACP. Next, a simple description of a systolic algorithm for string comparison will be presented, which is a modified version of the one described by LIPTON and LOPRESTI [LL]. We will present a *correctness proof* for the string analyzer within the setting of ACP.
Reading other texts about systolic systems, it is possible to list a few characteristics of them. A systolic system consists of a large number of (almost) identical elements, placed in a regular configuration. So far, practically all systolic algorithms that have been developped, have a simple linear configuration. A systolic system is characterized by a large data flow between the elements, and a highly parallel cooperation of the elements.
As in [KW], [MW] and [WE], the considered system is a so-called self-timed system. In clocked systems the flow of control is regulated by a clock, but in self-timed systems this is done by the relative cooperation of the elements. Here, an element is initiated by receiving inputs from other elements. It is possible, however, that new input is offered to an element during its internal computation. The sending element has to wait until the receiver is ready to receive. More on this subject can be found in MEAD and CONWAY [MC].
The notions mentioned above are relevant to the design of VLSI-circuits. Designing electrical circuits one has to deal with functional and timing aspects. A circuit is functionally correct when it computes the 'correct' answer, given a certain input. A circuit is correctly timed when the elements are connected in such a way that the relative speed of elements and wires do not give rise to malfunctioning. Building up a circuit from self-timed elements makes it possible to omit the timing aspects. So, to prove a circuit correct one only has to prove functional correctness of the circuit. A more comprehensive discussion can be found in [MC].
The systolic array presented in this report is an asynchronous version of the systolic array

described in [LL]. The string-to-string analyzer has really been implemented, and is used to compare DNA strings. In this paper we will show that correctness of an asynchronous version with synchronous communication of this machine can be verified within ACP.

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

2. THE ALGEBRA OF COMMUNICATING PROCESSES

The axiomatic framework in which we present this document is ACP_τ, the Algebra of Communicating Processes with silent steps, as described in [BK2]. In this section, we give a brief review of ACP_τ.

Process algebra starts from a collection A of given objects, called atomic actions, atoms or steps. These actions are taken to be indivisible, usually have no duration and form the basic building blocks of our systems. The first two compositional operators we consider are $\cdot$, denoting sequential composition, and + for alternative composition. If x and y are two processes, then $x \cdot y$ is the process that starts the execution of y after the completion of x, and x+y is the process that chooses either x or y and executes the chosen process. Each time a choice is made, we choose from a set of alternatives. We do not specify whether the choice is made by the process itself, or by the environment. Axioms A1-5 in table 1 below give the laws that + and $\cdot$ obey. We leave out $\cdot$ and brackets as in regular algebra, so $xy + z$ means $(x \cdot y) + z$.

On intuitive grounds $x(y + z)$ and $xy + xz$ present different mechanisms (the moment of choice is different), and therefore, an axiom $x(y + z) = xy + xz$ is not included.

We have a special constant δ denoting deadlock, the acknowledgement of a process that it cannot do anything anymore, the absence of an alternative. Axioms A6,7 give the laws for δ.

In process algebra parallelity is modeled by the parallel composition operator $\|$, called merge. The merge of processes x and y will interleave the actions of x and y, except for the communication actions. In $x\|y$, we can either do a step from x, or a step from y, or x and y both synchronously perform an action, which together make up a new action, the communication action. This trichotomy is expressed in axiom CM1. Here, we use two auxiliary operators $\lfloor\!\lfloor$ (left-merge) and $|$ (communication merge). Thus, $x\lfloor\!\lfloor y$ is $x\|y$, but with the restriction that the first step comes from x, and $x|y$ is $x\|y$ with a communication step as the first step. Axioms CM2-9 give the laws for $\lfloor\!\lfloor$ and $|$. On atomic actions, we assume a communication function given, obeying laws C1-3.

Finally, we have on the left-hand side of table 1 the laws for the encapsulation operator ∂_H. Here H is a set of atoms, and ∂_H blocks actions from H, renames them into δ. The operator ∂_H can be used to encapsulate a process, i.e. to block communications with the environment.

$x+y = y+x$	A1	$x\tau = x$	T1
$x+(y+z) = (x+y)+z$	A2	$\tau x + x = \tau x$	T2
$x+x = x$	A3	$a(\tau x+y) = a(\tau x+y)+ax$	T3
$(x+y)z = xz+yz$	A4		
$(xy)z = x(yz)$	A5		
$x+\delta = x$	A6		
$\delta x = \delta$	A7		
$a\|b = b\|a$	C1		
$(a\|b)\|c = a\|(b\|c)$	C2		
$\delta\|a = \delta$	C3		
$x\|\|y = x\lfloor\lfloor y + y\lfloor\lfloor x + x\|y$	CM1		
$a\lfloor\lfloor x = ax$	CM2	$\tau\lfloor\lfloor x = \tau x$	TM1
$ax\lfloor\lfloor y = a(x\|\|y)$	CM3	$\tau x\lfloor\lfloor y = \tau(x\|\|y)$	TM2
$(x+y)\lfloor\lfloor z = x\lfloor\lfloor z + y\lfloor\lfloor z$	CM4	$\tau\|x = \delta$	TC1
$ax\|b = (a\|b)x$	CM5	$x\|\tau = \delta$	TC2
$a\|bx = (a\|b)x$	CM6	$\tau x\|y = x\|y$	TC3
$ax\|by = (a\|b)(x\|\|y)$	CM7	$x\|\tau y = x\|y$	TC4
$(x+y)\|z = x\|z+y\|z$	CM8		
$x\|(y+z) = x\|y+x\|z$	CM9	$\partial_H(\tau) = \tau$	DT
		$\tau_I(\tau) = \tau$	TI1
$\partial_H(a) = a$ if $a\notin H$	D1	$\tau_I(a) = a$ if $a\notin I$	TI2
$\partial_H(a) = \delta$ if $a\in H$	D2	$\tau_I(a) = \tau$ if $a\in I$	TI3
$\partial_H(x+y) = \partial_H(x)+\partial_H(y)$	D3	$\tau_I(x+y) = \tau_I(x)+\tau_I(y)$	TI4
$\partial_H(xy) = \partial_H(x)\cdot\partial_H(y)$	D4	$\tau_I(xy) = \tau_I(x)\cdot\tau_I(y)$	TI5

table 1. ACP_τ.

The right-hand side of table 1 is devoted to laws for Milner's silent step τ (see MILNER [MI]). Laws T1-3 are Milner's τ-laws, and TM1,2 and TC1-4 describe the interaction of τ and merge. Finally, τ_I is the abstraction operator, that renames atoms from I into τ.
In table 1 we have $a,b,c \in A_\delta$ (i.e. $A \cup \{\delta\}$), x,y,z are arbitrary processes, and $H,I \subseteq A$.

definition i) Let t be a term over ACP_τ, and x a variable in t. Suppose that the abstraction operator τ_I does not occur in t. Then we say that an occurrence of x in t is *guarded* if t has a subterm of the form a·s, with $a \in A_\delta$ (so $a \neq \tau$!) and this x occurs in s. (I.e. each variable is 'preceded' by an atom.)

ii) A *recursive specification* over ACP_τ is a set of equations $\{x = t_x : x \in X\}$, with X a set of variables, and t_x a term over ACP_τ and variables X (for each $x \in X$). No other variables may occur in t_x.

iii) A recursive specification $\{x = t_x : x \in X\}$ is *guarded* if no t_x contains an abstraction operator τ_I, and each occurrence of a variable in each t_x is guarded.

notes: i) The constant τ cannot be a guard, since the presence of a τ does not lead to unique solutions: to give an example, the equation $x = \tau x$ has each process starting with a τ as a solution.

ii) A definition of guardedness involving τ_I is very complicated, and therefore, we do not give such a definition here. The definition above suffices for our purposes.

The *Recursive Definition Principle (RDP)* is the assumption that each guarded recursive specification has at least one solution, and the *Recursive Specification Principle (RSP)* is the assumption that each guarded recursive specification has at most one solution. In this paper, we assume RDP and RSP to be valid. Abusing language, we also use the variables in a guarded recursive specification for the process that is its unique solution.

In BAETEN, BERGSTRA and KLOP [BBK1], a model is presented for ACP_τ, consisting of rooted, directed multigraphs, with edges labeled by elements of $A \cup \{\delta,\tau\}$, modulo a congruence relation called rooted $\tau\delta$-bisimulation (comparable to Milner's observational congruence, see [MI]). In this model all axioms presented in this paper hold, and also principles RDP and RSP hold.

$(x \between y) \between z = x \between (y \| z)$
$(x \mid ay) \between z = x \mid (ay \between z)$
$x \mid y = y \mid x$
$x \| y = y \| x$
$x \mid (y \mid z) = (x \mid y) \mid z$
$x \| (y \| z) = (x \| y) \| z$

---

table 2. Standard concurrency.

The axioms of *Standard Concurrency* (displayed in table 2) will also be used in the sequel. A proof that they hold for all closed terms can be found in BERGSTRA and KLOP [BK2].
As one can easily see, encapsulation and abstraction cannot in general be distributed over $\|$ since in a merge processes may do a communication step and thus it is of great importance which comes first, the encapsulation (or abstraction) operator or the merge. Next, *conditional axioms* will be presented to state conditions for distributing τ_I and ∂_H over $\|$.

Definition: The *alphabet* of a process is the set of atomic actions that it can perform. So an alphabet is a subset of A. In order to define the alphabet function α on processes, we have the axioms in table 3 (for $a \in A$, x,y are arbitrary processes; see [BBK2]).

$\alpha(\delta) = \emptyset$	AB1
$\alpha(\tau) = \emptyset$	AB2
$\alpha(ax) = \{a\} \cup \alpha(x)$	AB3
$\alpha(\tau x) = \alpha(x)$	AB4
$\alpha(x + y) = \alpha(x) \cup \alpha(y)$	AB5
$\alpha(x) = \cup_{n \geq 1} \alpha(\pi_n(x))$	AB6
$\alpha(\tau_I(x)) = \alpha(x) - I$	AB7

table 3. Alphabet.

Note that $\alpha(\delta) = \alpha(\tau) = \emptyset$ is necessary by axioms A6 and T1. The axioms AB6 and AB7 can be proved from AB1-5 for closed terms, but are needed here to define the alphabet on general processes. Now we can formulate the conditional axioms as is done in table 4.

$\alpha(x) \mid (\alpha(y) \cap H) \subseteq H$	$\Rightarrow$	$\partial_H(x \| y) = \partial_H(x \| \partial_H(y))$	CA1
$\alpha(x) \mid (\alpha(y) \cap I) = \emptyset$	$\Rightarrow$	$\tau_I(x \| y) = \tau_I(x \| \tau_I(y))$	CA2
$\alpha(x) \cap H = \emptyset$	$\Rightarrow$	$\partial_H(x) = x$	CA3
$\alpha(x) \cap I = \emptyset$	$\Rightarrow$	$\tau_I(x) = x$	CA4
$H = J \cup K$	$\Rightarrow$	$\partial_H(x) = \partial_J \circ \partial_K(x)$	CA5
$I = J \cup K$	$\Rightarrow$	$\tau_I(x) = \tau_J \circ \tau_K(x)$	CA6
$H \cap I = \emptyset$	$\Rightarrow$	$\tau_I \circ \partial_H(x) = \partial_H \circ \tau_I(x)$	CA7

table 4. Conditional axioms.

In [BBK2] the axioms CA1-7 have been proved to hold for all closed ACP_τ-terms. We will assume that they hold for all processes.

3. STRING COMPARISON

The systolic array considered in this report compares two strings, and computes the edit distance between them. Before we turn to this machine it is necessary to say something about strings. A more comprehensive discussion on this subject can be found in WAGNER and FISCHER [WF].
We assume strings to be built up of characters from a character set C. The set of finite strings is denoted by C^*. This set also contains the empty string ε.

A string can be transformed into another string by application of a few basic operations. The operations we consider are:

1. changing a character into another character
2. deleting a character
3. inserting a character

A notation for a string operation is given by $a\to b$ ($a,b\in C\cup\{\varepsilon\}$, and not both $a=\varepsilon$ and $b=\varepsilon$). When $a,b\in C$, $a\to b$ denotes a change operation, and if $a=\varepsilon$ or $b=\varepsilon$, we have the delete and insert operation, repectively. A string vaw can be transformed into the string vbw, notation $vaw \to vbw$, using the operation $a\to b$ ($a,b\in C\cup\{\varepsilon\}$, $v,w\in C^*$). When a string v is transformed into a string w using a sequence S of basic operations this is denoted by $v\twoheadrightarrow^S w$. An example of a string transformation is given below.

example A possible transformation sequence for ACDA $\twoheadrightarrow$ AACF is:

$$ACDA \to ADA \to AA \to AAC \to AACF$$

From this example we can conclude that the transformation sequence need not be unique.
To each basic operation $a\to b$ a nonnegative cost value $cost(a\to b)$ is assigned. This cost function has two constraints: $cost(a\to a) = 0$ and $cost(a\to b) + cost(b\to c) \geq cost(a\to c)$. This function can be extended to all string transformations. When $v\twoheadrightarrow^S w$ and S is a sequence $s_1,\ldots,s_n$ of basic operations, the cost of this transformation is defined by $cost(v\twoheadrightarrow^S w) = cost(s_1) + \ldots + cost(s_n)$. Now it is possible to define the edit distance of two strings. The edit distance is defined by $ed(v,w) = \min\{cost(v\twoheadrightarrow^S w): v\twoheadrightarrow^S w\}$. In [WF] a useful fact is proved:

fact. Given a cost function *cost*, and the values *ed*(v,w), *ed*(va,w) and *ed*(v,wa), the value *ed*(va,wb) can be computed in the following way:

$$ed(va,wb) = \min\{ed(v,w)+cost(a\rightarrow b), ed(va,w)+cost(\varepsilon\rightarrow b), ed(v,wb)+cost(a\rightarrow\varepsilon)\}$$

Thus, the edit distance of two strings v,w can be computed using the edit distance of strings v' and w' of smaller length. In this way the computation of the edit distance is reduced to the level of a basic transformation. Given two strings v,w with lengths n and m respectively, this computation can be modeled by an $n+1 \times m+1$ matrix. The i-th character of the string v is indicated by v_i. In table 5 the algorithm to compute the elements of the matrix is presented.

$\alpha_{00} = 0$

$\alpha_{i0} = \sum_{1\le\ell\le i} cost(v_\ell\rightarrow\varepsilon) \quad 1\le i\le m$

$\alpha_{0j} = \sum_{1\le\ell\le j} cost(\varepsilon\rightarrow w_\ell) \quad 1\le j\le n$

$\alpha_{ij} = \min\{\alpha_{i-1j-1} + cost(v_i\rightarrow w_j), \alpha_{i-1j} + cost(v_i\rightarrow\varepsilon), \alpha_{ij-1} + cost(\varepsilon\rightarrow w_j)\}$

---

table 5. Definition of the elements of the edit distance matrix

Using the fact above, we conclude that the edit distance of strings v,w equals α_{nm}. In this paper we will give some examples using the cost function defined in table 6. We assume that the character set C includes a special symbol '□' to indicate the empty string. In table 6 we will use □ instead of ε. This symbol is included to handle the empty string as an object (a memory content).

$cost(a\rightarrow b) = 2$ if $a,b\in C/\{\square\}$ and $a\neq b$

$cost(a\rightarrow b) = 1$ if $a\in C/\{\square\}$ and $b=\square$ or $a=\square$ and $b\in C/\{\square\}$

$cost(a\rightarrow b) = 0$ if $a,b\in C$ and $a=b$

---

table 6. Example of a cost function.

An example of an edit matrix, using the cost function of table 6, is pictured in figure 1. There, we compute that *ed*(ADCAB,ACBAE) = 4. Observing the way this matrix is computed we can

conclude that some elements can be computed independently of each other. For example, after α_{11}, α_{12} and α_{21} have been computed, the elements α_{13} and α_{31} can be computed independently of each other. So, a process that computes this matrix can be divided into several concurrent processes. This is what has been done in the implementation discussed in this report.

		A	C	B	A	E
	0	1	2	3	4	5
A	1	0	1	2	3	4
D	2	1	2	3	4	5
C	3	2	1	2	3	4
A	4	3	2	3	2	3
B	5	4	3	2	3	4

figure 1. Example of edit distance matrix.

4. THE STRING TO STRING ANALYZER: AN INFORMAL DESCRIPTION

Before we turn to a formal description, we discuss the string-to-string analyzer informally. In [LL], a clocked systolic array is presented which computes the edit distance of two strings. However, the systolic array we will discuss is not clocked but self-timed. Therefore, the number of internal states of this machine is larger than the number of internal states of the clocked version from [LL].

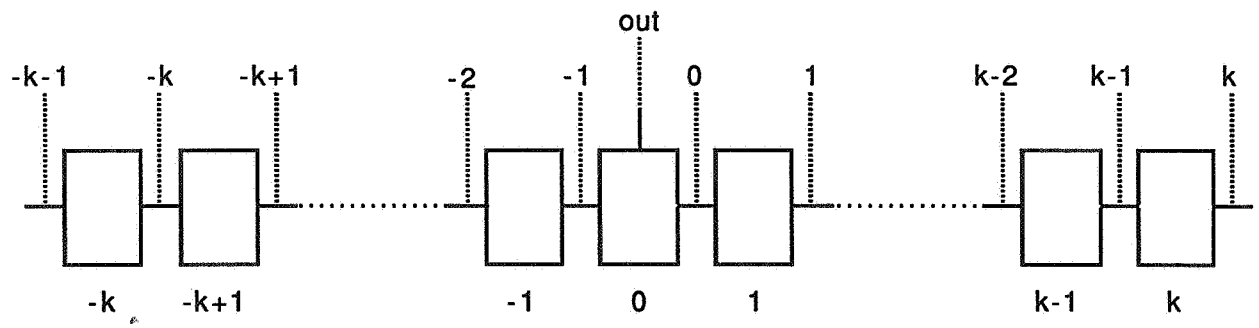

figure 2. The machine configuration.

In this paper we will prove that a systolic array consisting of 2k+1 particular elements, given a correct input, computes the edit distance of two strings of equal length up to a maximum length of k+1. The configuration of 2k+1 cells is pictured in figure 2. The cells are numbered -k,...,0,...,k and each cell i has two communication channels i-1 and i. Channel i is on the right of cell i and channel i-1 is on the left. Cell 0 has one extra channel called **out**. Cell 0 and a cell i for i≠0 are pictured in figure 3. A machine with 2(k+1)+1 cells is obtained by adding two cells, named -k-1 and k+1, to the machine consisting of 2k+1 cells.

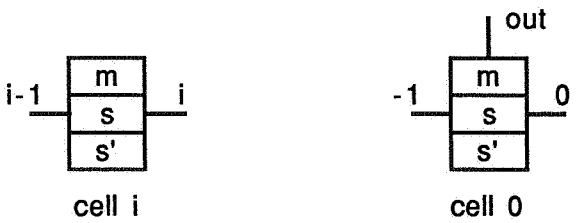

figure 3. Individual cells.

An individual cell i can be in two major states:

(i) *receive state*

In this state the cell contains a number m. The cell can receive a pair consisting of a number n and a symbol s, (s,n), from the left and a pair consisting of a number n' and a symbol s', (s',n'), from the right. These receive actions can be performed in either order. After these symbols and numbers have been received the cell computes

$$f(n,m,n',s,s') = \min(n+cost(\square\rightarrow s'), m+cost(s\rightarrow s'), n'+cost(s\rightarrow\square))$$

and m is replaced by this new number. The cell enters the send state.

(ii) *send state*

The cell still contains the two received symbols and a number m (computed as indicated above). The pair (s',m) can be sent to the left and the pair (s,m) can be sent to the right. These sending actions can be done in either order. After the cell has sent these two pairs to the neighbours/environment the cell enters the receive state. Moreover, cell 0 can send the number m to the outside, along channel **out**.

After the individual cells have been discussed we will consider the behaviour of 2k+1 cells, connected together. The two strings of which we want to compute the edit distance, are input at channels -k-1 and k, respectively. We will prove that this machine, starting in a certain initial state, is able to compute the edit distance of two strings of equal length with length smaller than or equal to k+1. A restriction is that each character input is accompanied by a certain number. When we

want to compute *edit*(v,w), each v_i is accompanied by *edit*($v_{1,i}$,ε) and each w_i is accompanied by *edit*($w_{1,i}$,ε). Here, the string consisting of the first j characters of v is indicated by $v_{1,j}$. Because of the restrictions to the cost function, this edit distance is easy to compute. When we want to compute *edit*(v,ε) we just have to compute the summation of all the individual delete actions. For example, using the cost function of table 6 this means that the j-th character input at either side is paired with the number j. We will explain the behaviour of the machine by means of the edit distance matrix and an example. Because the clocked version is easier to understand we first give an example of a clocked machine. In each state the cells have performed the same number of steps. The initial state of the machine is the following one:

(i) cells -k-1 and k contain the number 0 and they are in the receive state

(ii) the other cells are alternately in the receive state (and will then contain the number 0) and the send state (and will in that case contain the number 0 and two blanks).

		B	A	A	D
	0	1	2	3	4
A	1	2	1	2	3
C	2	3	2	3	4
D	3	4	3	4	3
B	4	3	4	5	4

figure 4. edit matrix of the example

The edit matrix of the example we worked out is in figure 4. The input/output session for this example using the clocked machine is given in figure 5. After the fourth state every new 'tick' the next diagonal of the edit matrix is computed. Notice, that the pairs of numbers and symbols input after the pairs of the strings do not influence the final result. However, to make this clocked machine work, some data have to be supplied after the relevant data have been input.

As said before,we discuss a version of the machine which is unclocked and self-timed. The outermost channels are controlled by the environment. The environment supplies the inputs and accepts the outputs. The communications inside the machine are controled by the relative cooperation of the cells. A machine state is called *stable* when no internal moves can be made. A machine state is unstable if the machine can make internal moves without communicating with the outside world. In figure 5 internal communications and communications with the environment are

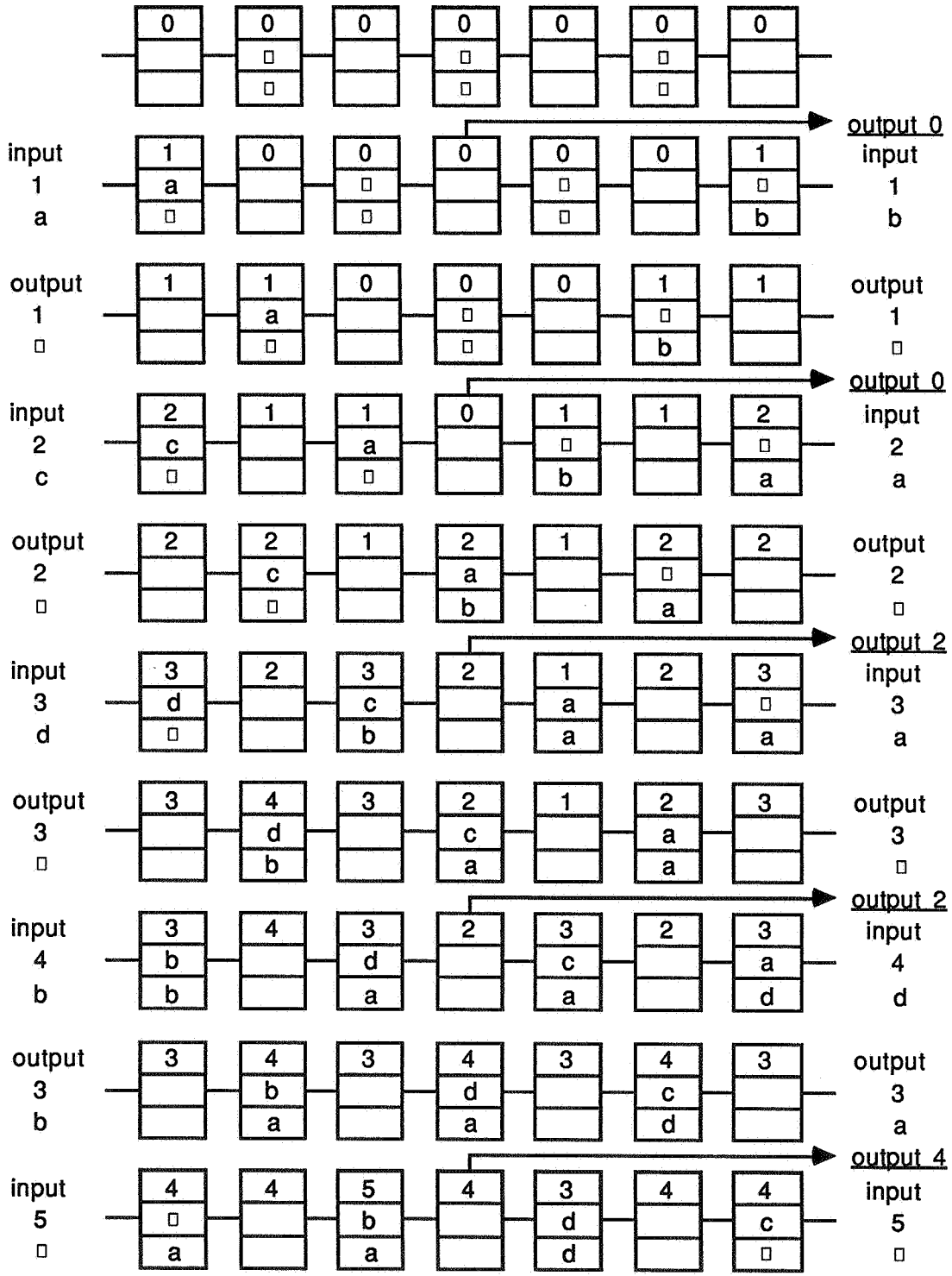

figure 5. Worked out example for clocked machine (to be continued).

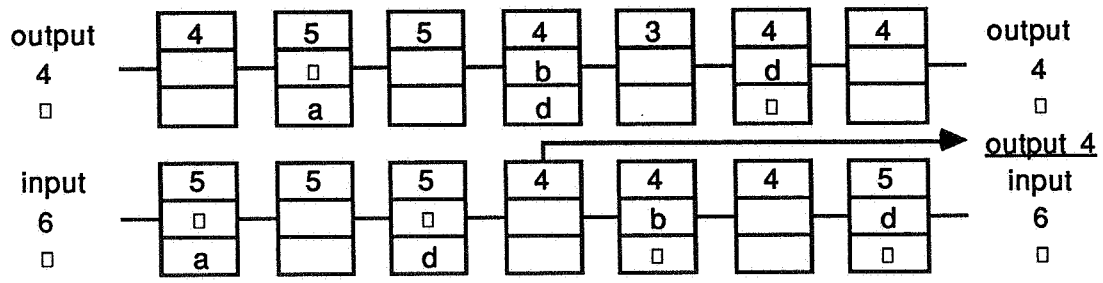

figure 5. continued.

performed at the same time. The initial state is a stable one, which is described as follows

(i) cell -k up to cell -1 contain 0 and □ in the lower symbol storage location (see figure 2). The cells are ready to receive a pair from the left.

(ii) cell 0 contains 0 and is ready to receive two pairs, one from either side.

(iii) cell 1 up to cell k contain 0 and □ in the upper symbol storage location. The cell is ready to receive a pair consisting of a number and a symbol from the right.

In figure 6 the initial state is pictured. The arrows pointing towards a cell indicate that the cell is ready to perform a receive action on that side. The starting state pictured in figure 6 can be considered as the starting state pictured in figure 5 after all the possible internal actions have been performed.

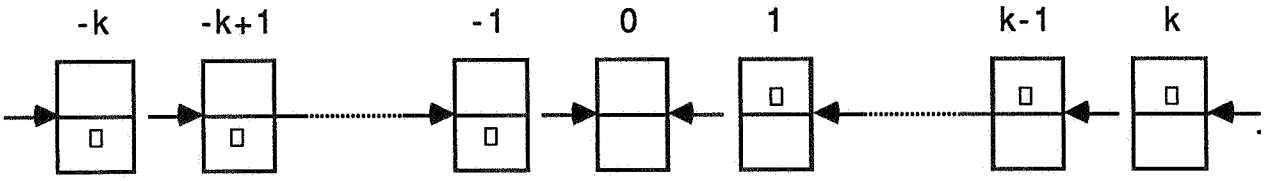

figure 6. The initial state of the machine.

We will assume that all the numbers output along channel **out** are accepted. So, we just block the channels -k-1 and k. Now the same example pictured in figure 5 is worked out in figure 7, but here we just consider stable states. Note, that the intermediate states the machine encounters need not necessarily be the pictured ones. We have chosen for one particular order. In the formal part we will make clear that to the outside there is no real difference between states that differ only in internal steps. Again, observe the correspondence between the matrix of figure 4 and the stable states of the machine.

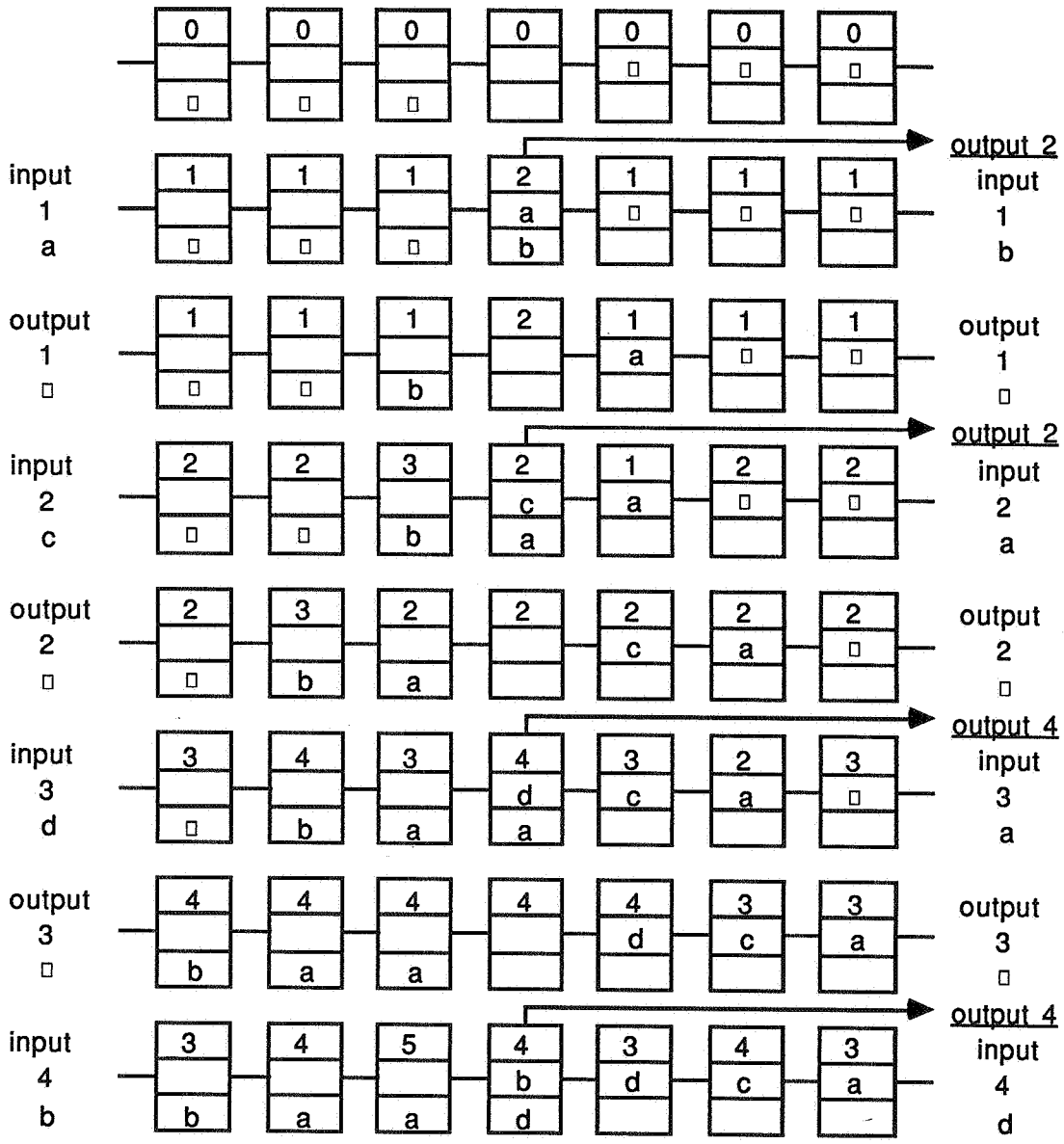

figure 7. example of the self-timed machine, considering only stable states

5. THE STRING TO STRING ANALYZER: FORMAL SPECIFICATION AND PROOF

In this section we will formalize what has been presented in section 4. We will present a correctness proof for the self-timed machine within the setting of ACP_τ. First we will give a specification for the individual cells, as is done in table 7. What has been explained in words in section 4 is formalized here. Although written out in table 7, from now on we will abbreviate $\Sigma_{s\in C,n\in N} r_i(s,n)$ by $\Sigma r_i(s,n)$. Taking $n=\alpha_{ij-1}$, $m=\alpha_{i-1j-1}$ and $n'=\alpha_{i-1j}$ (entities of the edit matrix of

specification of an individual cell i≠0:

$$CR_i(m) = [\Sigma_{s\in C,n\in N}\, r_{i-1}(s,n) \| \Sigma_{s'\in C,n'\in N}\, r_i(s',n')]\cdot CS_i(f(n,m,n',s,s'),s,s')$$

$$CS_i(m,s,s') = [s_{i-1}(s',m) \| s_i(s,m)]\cdot CR_i(m)$$

specification of cell 0:

$$CR_0(m) = [\Sigma_{s\in C,n\in N}\, r_{-1}(s,n) \| \Sigma_{s'\in C,n'\in N}\, r_0(s',n')]\cdot CS^+_0(f(n,m,n',s,s'),s,s')$$

$$CS^+_0(m,s,s') = [s_{-1}(s',m) \| s_0(s,m) \| s_{out}(m)]\cdot CR_0(m)$$

$$f(n,m,n',s,s') = \min(n+\text{cost}(\varepsilon\rightarrow s'),\ m+\textit{cost}(s\rightarrow s'),\ n'+\text{cost}(s\rightarrow\varepsilon))$$

table 7. Specification of an individual cell.

strings v and w) and $s'=w_j$, and $s=v_j$, the function f is precisely $ed(v_{1,i},w_{1,j})$. The i-th character of string v is indicated by v_i. The string consisting of the first i characters of v is indicated by $v_{1,i}$.

i≠0:

$$CR^L_i(m,s',n') = \Sigma_{s\in C,n\in N} r_{i-1}(s,n)\cdot CS_i(f(n,m,n',s,s'),s,s')$$

$$CR^R_i(m,s,n) = \Sigma_{s'\in C,n'\in N}\, r_i(s',n')\cdot CS_i(f(n,m,n',s,s'),s,s')$$

$$CS^L_i(m,s') = s_{i-1}(s',m)\cdot CR_i(m) \qquad CS^R_i(m,s) = s_i(s,m)\cdot CR_i(m)$$

i=0:

$$CR^L_0(m,s',n') = \Sigma_{s\in C,n\in N} r_{-1}(s,n)\cdot CS^+_0(f(n,m,n',s,s'),s,s')$$

$$CR^R_0(m,s,n) = \Sigma_{s'\in C,n'\in N} r_0(s',n')\cdot CS^+_0(f(n,m,n',s,s'),s,s')$$

$$CS_0(m,s,s') = [s_{-1}(s',m) \| s_0(s,m)]\cdot CR_0(m)$$

$$CS^L_0(m,s') = s_{-1}(s',m)\cdot CS^+_0(m) \qquad CS^R_0(m,s) = s_0(s,m)\cdot CS^+_0(m)$$

$$CS^{++}_0(m) = s_{out}(m)\cdot CR_0(m)$$

$$CS^{L+}_0(m,s') = [s_{-1}(s',m) \| s_{out}(m)]\cdot CR_0(m) \qquad CS^{R+}_0(m,s) = [s_0(s,m) \| s_{out}(m)]\cdot CR_0(m)$$

table 8. shorthands for states of the cells used in this paper

Next, we will formalize the initial state of the machine. In ACP_τ, a systolic array of 2k+1 connected cells is modeled by a communication merge of 2k+1 cells. In case of an internal send or receive action (i.e. not at the external channels -k-1, k or **out**), the machine has to communicate with a complementary action from a neighbouring cell. This is modeled within ACP_τ by encapsulating the individual send and receive actions. Abstraction from internal communications is achieved by renaming these communications into the silent step τ of Milner [MI]. This, together with the description of the initial state given in section 4, results in the formalization given in table 9. I_k and H_k are the abstraction set and the encapsulation set, respectively.

$INIT_k = \tau_{I_k} \circ \partial_{H_k}\{ \|_{-k \le \ell < 0} CR^L_\ell(0,\square,0) \| C_0(0) \|_{0 < \ell \le k} CR^R_\ell(0,\square,0) \}$

communications: $r_\ell(s,n) | s_\ell(s,n) = c_\ell(s,n)$ $s \in C$, $n \in N$, $-k-1 < \ell < k$

$H_k = \{r_\ell(s,n), s_\ell(s,n)$: $s \in C$, $n \in N$, $-k-1 < \ell < k\}$

$I_k = \{c_\ell(s,n)$: $s \in C, n \in N, -k-1 < \ell < k\}$

---

table 9. Initial state of the string-to-string analyzer of capacity k+1.

After specifying the initial state of the machine we will specify the environment. As already has been indicated a correct computation of the edit distance of two string depends on the correctness of the input. A formal specification of the environment for a machine of capacity k+1 is given in table 10.

$$ENV_{k+1}(v,w) = (\Pi_{1 \le \ell < k+1} [(s_{-k-1}(v_\ell, \alpha_{\ell 0}) \| s_k(w_\ell, \alpha_{0\ell})) \cdot (\Sigma\, r_{-k-1}(s,n) \| \Sigma\, r_k(s',n'))]) \cdot (s_{-k-1}(v_{k+1}, \alpha_{k+10}) \| s_k(w_{k+1}, \alpha_{0k+1}))$$

---

table 10. Specification of the environment.

What we want to prove now is stated in the theorem below. The sets H'_k and I'_k are used to obtain the synchronous communication between machine and environment. H'_k contains the actions that are encapsulated. Further, the communications we want to abstract from, are in I'_k.

The *encapsulation set* H'_k is defined by
$H'_k = \{r_{-k-1}(s,n), r_k(s,n), s_{-k-1}(s,n), s_k(s,n) : s \in C, n \in N\}$,
and the *communication set* I'_k is defined by:
$I'_k = \{c_{-k-1}(s,n), c_k(s,n) : s \in C, n \in N\}$
having the following communications: $s_\ell(s,n) \mid r_\ell(s,n) = c_\ell(s,n)$, ℓ=-k-1 or ℓ=k

Theorem (correctness) Let $\alpha_{\ell\ell}$ be the values on the diagonal of the edit matrix of v and w. Then:

$$\tau_{I'_k} \circ \partial_{H'_k}\{ENV_{k+1}(v,w) \| INIT_k\} = \tau \cdot \Pi_{1 \le \ell \le k+1} s_{out}(\alpha_{\ell\ell}) \cdot \delta$$

proof The criterion of correctness used here is similar to the one in [BK3]. The rest of this paper is devoted to the proof of this theorem. The theorem will be proved in two steps. We prove that the two terms $\tau_{I'_k} \circ \partial_{H'_k}\{ENV_{k+1}(v,w) \| INIT_k\}$ and $\tau \cdot \Pi_{1 \le \ell \le k+1} s_{out}(\alpha_{\ell\ell}) \cdot \delta$ satisfy the same guarded specification, which is given in table 11. Then, by RSP, both processes are equal.

$EI^R_{i,j} = \tau \cdot EI^{RL}_{i,j} + \tau \cdot EI^{RR}_{i,j} + s_{out}(\alpha_{ii}) \cdot EI^R_{i+1,j}$	$1 \le i,j \le k,\ i \le j$
$EI^{RR}_{i,j} = \tau \cdot EI^S_{i,j+1} + s_{out}(\alpha_{ii}) \cdot EI^{RR}_{i+1,j}$	"
$EI^{RL}_{i,j} = \tau \cdot EI^S_{i,j+1} + s_{out}(\alpha_{ii}) \cdot EI^{RL}_{i+1,j}$	"
$EI^R_{i,j} = \tau \cdot EI^{RL}_{i,j} + \tau \cdot EI^{RR}_{i,j}$	$1 \le i \le k+1,\ 0 \le j \le k,\ i=j+1$
$EI^{RR}_{i,j} = \tau \cdot EI^S_{i,j+1}$ $\quad EI^{RL}_{i,j} = \tau \cdot EI^S_{i,j+1}$	"
$EI^S_{i,j} = \tau \cdot EI^{SL}_{i,j} + \tau \cdot EI^{SR}_{i,j} + s_{out}(\alpha_{ii}) \cdot EI^S_{i+1,j}$	$1 \le i,j \le k,\ i \le j$
$EI^{SR}_{i,j} = \tau \cdot EI^R_{i,j} + s_{out}(\alpha_{ii}) \cdot EI^{SR}_{i+1,j}$	"
$EI^{SL}_{i,j} = \tau \cdot EI^R_{i,j} + s_{out}(\alpha_{ii}) \cdot EI^{SL}_{i+1,j}$	"
$EI^S_{i,j} = \tau \cdot EI^{SL}_{i,j} + \tau \cdot EI^{SR}_{i,j}$	$1 \le i,j \le k,\ i=j+1$
$EI^{SR}_{i,j} = \tau \cdot EI^R_{i,j}$ $\quad EI^{SL}_{i,j} = \tau \cdot EI^R_{i,j}$	"
$EI^S_{i,k+1} = s_{out}(\alpha_{ii}) \cdot EI^S_{i+1,k+1}$	$1 \le i \le k,\ j=k+1$
$EI^S_{k+1,k+1} = s_{out}(\alpha_{k+1k+1}) \cdot \delta$	$i=k+1,\ j=k+1$

table 11. Specification of the interaction between machine and environment.

The indices should be interpreted in the following way: i corresponds to the values α_{ii} the machine wants to send away through channel **out**. The j indicates how many symbols of the strings v and w have been input. Abstracting from all the internal actions we can discern 6 different states the environment/machine interaction can be in:

- the machine is able to receive two symbol/number pairs of v and w respectively from the environment in either order. This is indicated by a superscript R. However it is possible that the receive action is already completed on one side. When the receive action on the left has been completed the only possible action is a receive action on the right. This is indicated by the superscript RR. When the receive action on the right already has been performed, this is indicated by the superscript RL.
- the machine wants to send two symbol/number pairs to the environment by way of the two external channels -k-1 and k. This is indicated by the superscript S. Analogoulsly we have the superscripts SR and SL to indicate the not fully completed send actions.

Furthermore, notice the analogy between the number of send and receive states in the specification of table 11 and the number of receive and send actions of the environment. The environment sends k+1 symbols with the corresponding edit distance to the machine and from table 11 we can see that the machine passes k+1 receive states. The same can be verified for the k receive actions the environment performs. $EI^{R}_{1,0}$ corresponds to the starting state of the environment/machine combination. The superscript R indicates that the machine is in a receive state and the environment in the send state. The subscripts 1 and 0 indicate that 0 symbols have been input and that the first number to be output is α_{11}.

Now, we will prove the theorem by proving the following equations:

(i) $EI^{R}_{1,0} = \tau \cdot \Pi_{1 \le \ell \le k+1} s_{out}(\alpha_{\ell\ell}) \cdot \delta$

(ii) $EI^{R}_{1,0} = \tau_{I'k} \circ \partial_{H'k}\{ENV_{k+1}(v,w) \| INIT_k\}$

proof of (i): To prove that equation (i) holds we verify that $\tau \cdot \Pi_{1 \le \ell \le k+1} s_{out}(\alpha_{\ell\ell}) \cdot \delta$ is a solution of the specification mentioned in table 11. That this expression is a solution of the specification can easily be verified using the substitution mentioned in table 12. If the specification in table 11 is guarded we can use RSP and the desired equation is obtained. Thus, we just need to verify that this specification is indeed guarded. It is obvious that from state $EI^{S}_{i,k+1}$ it is not possible to perform any τ steps but just atomic actions. On the other hand from the other states $EI^{S*}_{i,j}$, $EI^{R*}_{i,j}$ ($* = R,L$ or blank) we can do no more than four τ steps without entering a state with a higher j. This and the fact that in state $EI^{S}_{i,k+1}$ only atomic actions can be performed, limits the number of τ

steps that can be performed. Thus, the specification is guarded and equation (i) has been proved. What is left to prove, is the second equation. As already mentioned in the previous section we want to formally describe the notion of a stable state. A stable state can be considered as a state from which only visible steps (outside of the abstraction set I) can be done. In any stable state only two or three visible atomic actions are possible. However, the environment may not be ready to accept a certain atomic action: such an action cannot be answered by a complementary action.

$$EI^{S}_{i,j} = EI^{SL}_{i,j} = EI^{SR}_{i,j} = \tau\cdot\prod_{i\le\ell\le k+1} s_{out}(\alpha_{\ell\ell})\cdot\delta \quad 1\le i,j\le k,\ i\le j$$

$$EI^{S}_{i,k+1} = \prod_{i\le\ell\le k+1} s_{out}(\alpha_{\ell\ell})\cdot\delta \quad 1\le i\le k+1,\ j=k+1$$

$$EI^{R}_{i,j} = EI^{RL}_{i,j} = EI^{RR}_{i,j} = \tau\cdot\prod_{i\le\ell\le k+1} s_{out}(\alpha_{\ell\ell})\cdot\delta \quad 1\le i\le k+1,\ 0\le j\le k,\ i\le j+1$$

table 12. Correspondence between table 11 and $\tau\cdot\prod_{1\le\ell\le k+1} s_{out}(a_{\ell\ell})\cdot\delta$.

So, after performing such an atomic action the machine enters a state which is of no use to the computation. What follows is not of any interest to us and can be left out. In this way a great number of stable states are redundant. The term redundancy is originally introduced in VAANDRAGER [VA]. Another paper discussing the notion of redundancy is KOYMANS and MULDER [KM]. In table 13 we have listed the stable states that are relevant with respect to the environment of table 10 (compare table 8).

The following notations and assumptions will be used in the table:
- k is a fixed number
- $\mathbf{n} = \langle n_{-k},..,n_0,..,n_k\rangle$ indicates the numbers the cells contain. The number stored in cell ℓ is called n_ℓ.
- $t = t_0..t_k$ or $t_1..t_k$ (dependign on |t|) indicates the sequence of symbols contained in the upper storage locations of the cells. These symbols are input from the left. The same holds for $u = u_0..u_k$ or $u_1..u_k$, only these symbols are received at the right and stored in the lower storage locations.
- p indicates the number of output actions which can be performed along channel **out**. This number may increase when pairs have been input at the channels -k-1 and k without performing the corresponding send actions along channel **out**.

To make this mess of formulas more clear to the reader we have visualized a number of states in figure 8. An arrow pointing towards a cell indicates the cell wants to receive a pair from that side

but is not able to do so before an action at one of the external channels has been performed (the state is stable). An arrow pointing away from a cell indicates that that cell wants to perform a send action but is not able to do so. To indicate that the cell contains a symbol of t in the upper storage loaction, the upper half of the cell has been shaded. When a symbol of string u is stored in the lower storage location of a cell, the lower half of that cell is shaded.

(i) $|u|=|t|=k$, $p=0$ $\quad STRING^{R}(\mathbf{n},t,u,0) =$

$$= \tau_{I_k} \circ \partial_{H_k}\{\|_{-k\le \ell<0} CR^{L}_{\ell}(n_{\ell},u_{-\ell},n_{\ell+1})\|CR_0(n_0)\| \|_{0<\ell\le k} CR^{R}_{\ell}(n_{\ell},t_{\ell},n_{\ell-1})\}$$

(ii) $|u|=k$, $|t|=k+1$, $p=0$ $\quad STRING^{RR}(\mathbf{n},t,u,0) =$

$$= \tau_{I_k} \circ \partial_{H_k}\{\|_{-k\le \ell<0} CS^{L}_{\ell}(n_{\ell},u_{-\ell})\| \|_{0\le \ell\le k} CR^{R}_{\ell}(n_{\ell},t_{\ell},n_{\ell-1})\}$$

(iii) $|u|=k+1$, $|t|=k$, $p=0$ $\quad STRING^{RL}(\mathbf{n},t,u,0)$: omitted, complementary to state (ii)

(iv) $|u|=|t|=k$, $0<p\le k$ $\quad STRING^{R}(n,t,u,p) =$

$$\tau_{I_k} \circ \partial_{H_k}\{\|_{-k\le \ell\le -p} CR^{L}_{\ell}(n_{\ell},u_{-\ell},n_{\ell+1})\| \|_{-p<\ell<0} CS^{R}_{\ell}(n_{\ell},t_{p+\ell})\|CS^{++}{}_0(n_0)\| \|_{0<\ell<p} CS^{L}_{\ell}(n_{\ell},u_{p-\ell})\| \|_{p\le \ell\le k} CR^{R}_{\ell}(n_{\ell},t_{\ell},n_{\ell-1})\}$$

(v) $|u|=k$, $|t|=k+1$, $0<p\le k$ $\quad STRING^{RR}(n,t,u,p) =$

$$\tau_{I_k} \circ \partial_{H_k}\{\|_{-k\le \ell<-p} CS^{L}_{\ell}(n_{\ell},u_{-\ell})\|CS_{-p}(n_{-p},t_0,u_p)\| \|_{-p<\ell<0} CS^{R}_{\ell}(n_{\ell},t_{p+\ell})\|CS^{++}{}_0(n_0)\| \|_{0<\ell<p} CS^{L}_{\ell}(n_{\ell},u_{p-\ell})\| \|_{p\le \ell\le k} CR^{R}_{\ell}(n_{\ell},t_{\ell},n_{\ell-1})\}$$

(vi) $|u|=k+1$, $|t|=k$, $0<p\le k$ $\quad STRING^{RL}(n,t,u,p)$: omitted, complementary to state (v)

(vii) $|u|=|t|=k+1$, $p=0$ $\quad STRING^{S}(\mathbf{n},t,u,0) =$

$$= \tau_{I_k} \circ \partial_{H_k}\{\|_{-k\le \ell<0} CS^{L}_{\ell}(n_{\ell},u_{-\ell})\|CS_0(n_0,t_0,u_0)\| \|_{0<\ell\le k} CS^{R}_{\ell}(n_{\ell},t_{\ell})\}$$

(viii) $|u|=k$, $|t|=k+1$, $p=0$ $\quad STRING^{SR}(\mathbf{n},t,u,0) =$

$$= \tau_{I_k} \circ \partial_{H_k}\{\|_{-k\le \ell<0} CR^{L}_{\ell}(n_{\ell},u_{-\ell},n_{\ell+1})\| \|_{0\le \ell\le k} CS^{R}_{\ell}(n_{\ell},t_{\ell})\}$$

---

table 13. Stable states relevant to the environment defined in table 10 (to be continued).

(ix) $|u|=k+1$, $|t|=k$, $p=0$ $\quad STRING^{SL}(\mathbf{n},t,u,0)$: omitted, complementary to state (viii)

(x) $|u|=|t|=k+1$, $p=1$ $\quad STRING^{S}(\mathbf{n},t,u,1) =$

$$= \tau_{I_k} \circ \partial_{H_k} \{ \|_{-k \le \ell < 0} CS^{L}{}_{\ell}(n_\ell, u_{-\ell}) \| CS^{+}{}_{0}(n_0, t_0, u_0) \| \|_{0 < \ell \le k} CS^{R}{}_{\ell}(n_\ell, t_\ell) \}$$

(xi) $|u|=k$, $|t|=k+1$, $p=1$ $\quad STRING^{SR}(\mathbf{n},t,u,1) =$

$$= \tau_{I_k} \circ \partial_{H_k} \{ \|_{-k \le \ell < 0} CR^{L}{}_{\ell}(n_\ell, u_{-\ell}) \| CS^{R+}{}_{0}(n_0, t_0) \| \|_{0 < \ell \le k} CS^{R}{}_{\ell}(n_\ell, t_\ell) \}$$

(xii) $|u|=k+1$, $|t|=k$, $p=1$ $\quad STRING^{SL}(\mathbf{n},t,u,p)$: omitted, complementary to state (xi)

(xiii) $|u|=|t|=k+1$, $1<p\le k+1$ $\quad STRING^{S}(\mathbf{n},t,u,p) =$

$$\tau_{I_k} \circ \partial_{H_k} \{ \|_{-k \le \ell \le -p} CS^{L}{}_{\ell}(n_\ell, u_{-\ell}) \| CS_{-p+1}(n_{-p+1}, t_0, u_{p-1}) \|$$
$$\|_{-p+1 < \ell < 0} CS^{R}{}_{\ell}(n_\ell, t_{p-1+\ell}) \| CS^{++}{}_{0}(n_0) \|$$
$$\|_{0 < \ell < p-1} CS^{L}{}_{\ell}(n_\ell, u_{p-1-\ell}) \| CS_{p-1}(n_{p-1}, t_{p-1}, u_0) \| \|_{p \le \ell \le k} CS^{R}{}_{\ell}(n_\ell, t_\ell) \}$$

(xiv) $|u|=k$, $|t|=k+1$, $1<p\le k$ $STRING^{SR}(\mathbf{n},t,u,p) =$

$$= \tau_{I_k} \circ \partial_{H_k} \{ \|_{-k \le \ell \le -p} CR^{L}{}_{\ell}(n_\ell, u_{-\ell}) \| \|_{-p < \ell < 0} CS^{R}{}_{\ell}(n_\ell, t_{p+\ell})$$
$$\| CS^{++}{}_{0}(n_0) \| \|_{0 < \ell < p-1} CS^{L}{}_{\ell}(n_\ell, u_{p-\ell}) \|$$
$$\| CS_{p-1}(n_{p-1}, t_{p-1}, u_0) \| \|_{p \le \ell \le k} CS^{R}{}_{\ell}(n_\ell, t_\ell) \}$$

(xv) $|u|=k+1$, $|t|=k$, $1<p\le k$ $\quad STRING^{SL}(\mathbf{n},t,u,p)$ omitted, complementary to state (xiv)

table 13. continued.

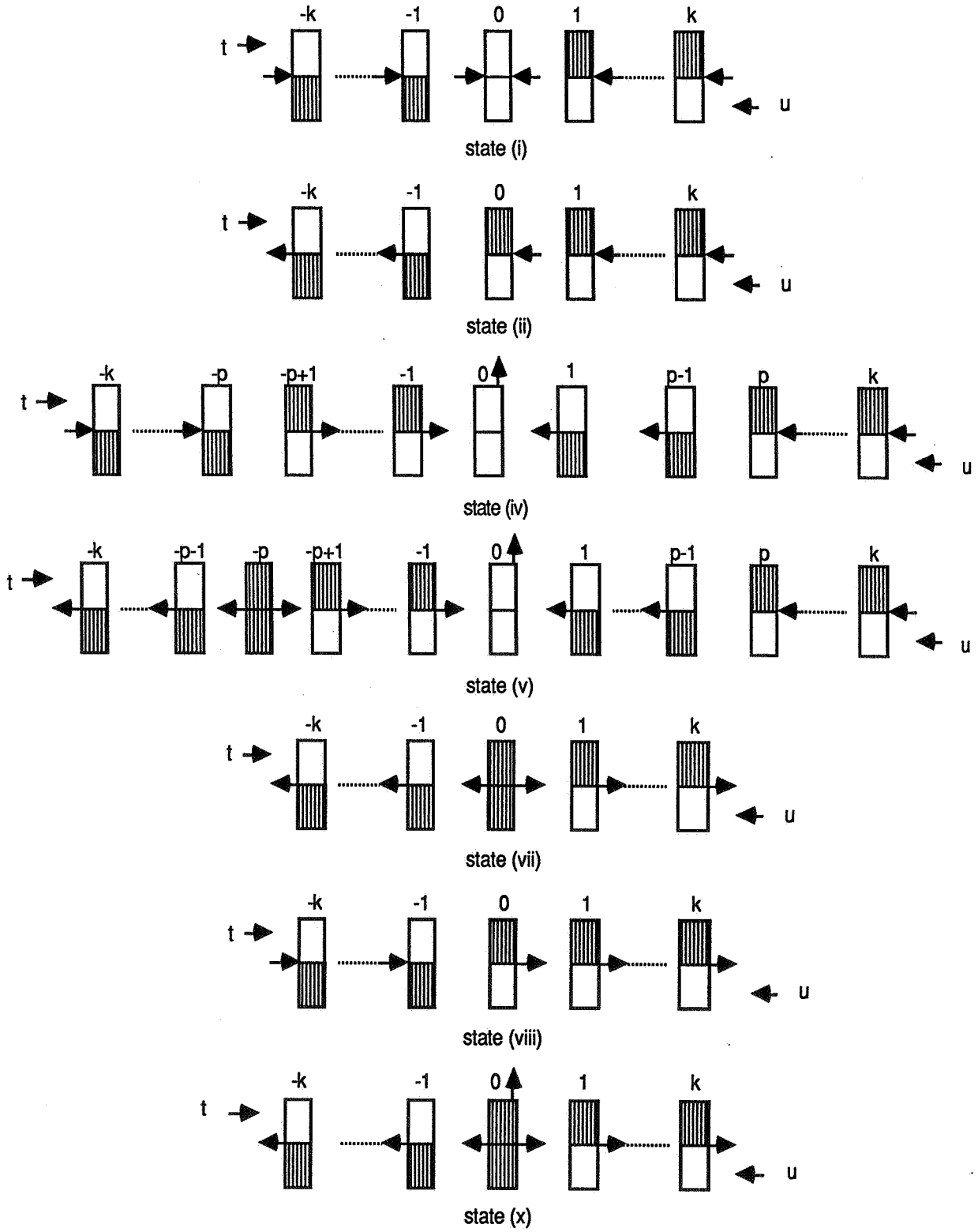

figure 8. Visualisation of the interesting stable states (to be continued).

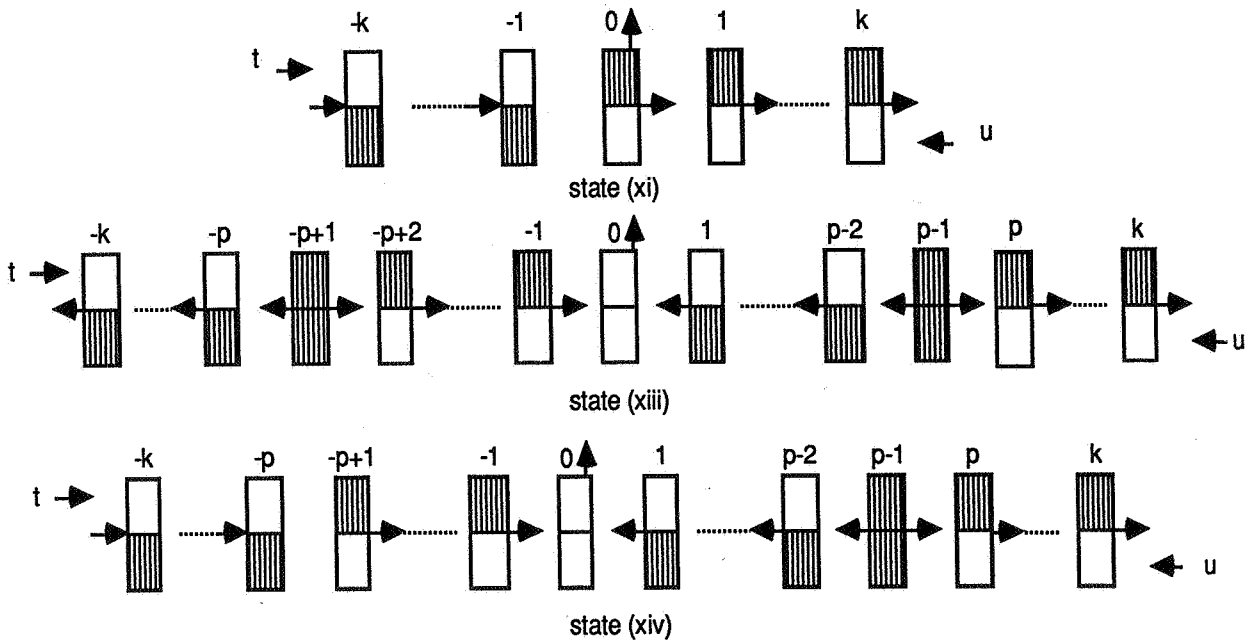

figure 8. continued.

After having given these stable states which are of interest with respect to the environment of table 10 we will give a relation between these states. However, it is possible that the machine is able to perform an action which cannot be accepted by the environment. This atomic action is redundant. In table 14 we have given a specification of the stable states. When the machine wants to perform a redundant action this action is succeeded by Ω. This to indicate 'it does not matter what happens after this action because this action is redundant'. In another context Ω is used in [KW]. Later on we will make it clear that the actions said to be redundant are indeed redundant. The function φ defines what happens to the numbers contained in the cells. The number of arguments of φ is larger than those which are listed but for sake of readibility we have omitted a few of them. To indicate a string t without its last symbol we use t-. When a symbol s is added to a string t, this is denoted by st. In the function φ_M we use indt and indu to indicate the index of the symbols of strings t and u contained in the cells -1 and 1 respectively.

$p=0$ $\quad \mathrm{STRING}^{R}(\mathbf{n},t,u,0) = \sum r_{-k-1}(s,n)\cdot \mathrm{STRING}^{RR}(\varphi_L(\mathbf{n},1),st,u,0) +$

$$+ \sum r_k(s',n')\cdot \mathrm{STRING}^{RL}(\varphi_R(\mathbf{n},1),t,s'u,0)$$

$p=0$ $\quad \mathrm{STRING}^{RR}(\mathbf{n},t,u,0) = s_{-k-1}(u_k,n_{-k})\cdot\Omega + \sum r_k(s',n')\cdot \mathrm{STRING}^{S}(\varphi_R(\mathbf{n},0),t,s'u,1)$

$p=0$ $\quad \mathrm{STRING}^{RL}(\mathbf{n},t,u,0) = \sum r_{-k-1}(s,n)\cdot \mathrm{STRING}^{S}(\varphi_L(\mathbf{n},0),st,u,1) + s_k(t_k,n_k)\cdot\Omega$

$1\le p\le k$ $\quad \mathrm{STRING}^{R}(\mathbf{n},t,u,p) = \sum r_{-k-1}(s,n)\cdot \mathrm{STRING}^{RR}(\varphi_L(\mathbf{n},p),st,u,p) +$

$$+ \sum r_k(s',n')\cdot \mathrm{STRING}^{RL}(\varphi_R(\mathbf{n},p),t,s'u,p) + s_{out}(n_0)\cdot \mathrm{STRING}^{R}(\varphi_M(\mathbf{n},p\text{-}1,p\text{-}1,p\text{-}1,p\text{-}1),t,u,p\text{-}1)$$

$1\le p\le k$ $\quad \mathrm{STRING}^{RR}(\mathbf{n},t,u,p) = s_{-k-1}(u_k,n_{-k})\cdot\Omega + \sum r_k(s',n')\cdot \mathrm{STRING}^{S}(\varphi_R(\mathbf{n},p),t,s'u,p+1)$

$$+ s_{out}(n_0)\cdot \mathrm{STRING}^{RR}(\varphi_M(\mathbf{n},p,p\text{-}1,p\text{-}1,p\text{-}1),t,u,p\text{-}1)$$

$1\le p\le k$ $\quad \mathrm{STRING}^{RL}(\mathbf{n},t,u,p) = \sum r_{-k-1}(s,n)\cdot \mathrm{STRING}^{S}(\varphi_L(\mathbf{n},p),st,u,p+1) + s_k(t_k,n_k)\cdot\Omega$

$$+ s_{out}(n_0)\cdot \mathrm{STRING}^{RL}(\varphi_M(\mathbf{n},p\text{-}1,p,p\text{-}1,p\text{-}1),t,u,p\text{-}1)$$

$p=0$ $\quad \mathrm{STRING}^{S}(\mathbf{n},t,u,0) = s_{-k-1}(u_k,n_{-k})\cdot \mathrm{STRING}^{SR}(\mathbf{n},t,u\text{-},0) + s_k(t_k,n_k)\cdot \mathrm{STRING}^{SL}(\mathbf{n},t\text{-},u,0)$

$p=0$ $\quad \mathrm{STRING}^{SR}(\mathbf{n},t,u,0) = \sum r_{-k-1}(s,n)\cdot\Omega + s_k(t_k,n_k)\cdot \mathrm{STRING}^{R}(\mathbf{n},t\text{-},u,0)$

$p=0$ $\quad \mathrm{STRING}^{SL}(\mathbf{n},t,u,0) = s_{-k-1}(u_k,n_{-k})\cdot \mathrm{STRING}^{R}(\mathbf{n},t,u\text{-},0) + \sum r_k(s',n')\cdot\Omega$

$1\le p\le k+1$ $\quad \mathrm{STRING}^{S}(\mathbf{n},t,u,p) = s_{-k-1}(u_k,n_{-k})\cdot \mathrm{STRING}^{SR}(\mathbf{n},t,u\text{-},p) +$

$$+ s_k(t_k,n_k)\cdot \mathrm{STRING}^{SL}(\mathbf{n},t\text{-},u,p) + s_{out}(n_0)\cdot \mathrm{STRING}^{S}(\varphi_M(\mathbf{n},p\text{-}1,p\text{-}1,p\text{-}2,p\text{-}2),t,u,p\text{-}1)$$

$1\le p\le k+1$ $\quad \mathrm{STRING}^{SR}(\mathbf{n},t,u,0) = \sum r_{-k-1}(s,n)\cdot\Omega + s_k(t_k,n_k)\cdot \mathrm{STRING}^{R}(\mathbf{n},t\text{-},u,p) +$

$$+ s_{out}(n_0)\cdot \mathrm{STRING}^{SR}(\varphi_M(\mathbf{n},p\text{-}1,p\text{-}1,p\text{-}2,p\text{-}1),t,u,p\text{-}1)$$

$1\le p\le k+1$ $\quad \mathrm{STRING}^{SL}(\mathbf{n},t,u,0) = s_{-k-1}(u_k,n_{-k})\cdot \mathrm{STRING}^{R}(\mathbf{n},t,u\text{-},p) + \sum r_k(s',n')\cdot\Omega +$

$$+ s_{out}(n_0)\cdot \mathrm{STRING}^{S}(\varphi_M(\mathbf{n},p\text{-}1,p\text{-}1,p\text{-}1,p\text{-}2),t,u,p\text{-}1)$$

$$\varphi_L(\mathbf{n},q) = \langle n'_{-k},..,n'_{-q},n_{-q+1},..,n_k\rangle \ , \ 0\le q\le k$$

$$n'_{-k} = f(n,n_{-k},n_{-k+1},s,u_k)$$

$$n'_{\ell} = f(n'_{\ell-1},n_{\ell},n_{\ell+1},s,u_{-\ell}) \quad -k<\ell\le -q$$

table 14. Relation between the stable states (to be continued).

$\varphi_R(n,q) = \langle n_{-k},..,n_{q-1},n'_q,..,n'_k\rangle$,	$0\le q\le k$
	$n'_{-k} = f(n_{k-1},n_k,n',t_k,s')$
	$n'_\ell = f(n_{\ell-1},n_\ell, n'_{\ell+1},t_\ell,s')$ $q\le\ell<k$
$\varphi_M(n,q,r,indt,indu) = \langle n_{-k},..,n_{-q},n'_{-q+1},..,n'_0,..,n'_{r-1},n_r,..,n_k\rangle$	$0\le q\le k$, $0\le r\le k$, $\lvert q-r\rvert\le 1$
$q=0$ or $r=0$	$n'_\ell = n_\ell$
otherwise	$n'_0 = f(n_{-1},n_0,n_1,t_{indt},u_{indu})$
	$n'_\ell = f(n_{\ell-1},n_\ell, n'_{\ell+1},t_{indt+\ell},u_{indu})$ $-q<\ell<0$
	$n'_\ell = f(n'_{\ell-1},n_\ell,n_{\ell+1},t_{indt},u_{indu-\ell})$ $0<\ell<r$

table 14. continued.

We will not present the proofs for all these equations but just for one of them. The other cases can be verified in the same way. In the proof we will use the following fact which can be proved by induction on the structures of X, Y, P and Q:

fact *: Let P, Q be closed terms and c, c_i ($i\in B$, B finite) atomic actions such that:

(i) $c\,|\,c_j=c^0$, for exactly one $j\in B$

(ii) c, c_i ($i\in B$) do not communicate with actions of P,Q

(iii) $c^0\in I$

(iv) $c\in H$, $H\supset\{c_i : i\in B\}$

then for all X,Y,P and Q: $\tau_I\circ\partial_H((P\|c)\cdot X\|(Q\|\Sigma_{i\in B}c_i)\cdot Y) = \tau\cdot\tau_I\circ\partial_H(P\cdot X\|Q\cdot Y)$.

We have for all $-k\le i<k$:

$H_{i,i+1} = \{r_i(s,n),s_i(s,n):s\in C,n\in N\}$ and

$I_{i,i+1} = \{c_i(s,n):s\in C,n\in N\}$.

proof of table 14: We will only consider the following equation from table 14:

$1\le p\le k$ $\quad STRING^R(n,t,u,p) = \Sigma\, r_{-k-1}(s,n)\cdot STRING^{RR}(\varphi_L(n,p),st,u,p) +$

$+\Sigma\, r_k(s',n')\cdot STRING^{RL}(\varphi_R(n,p),t,s'u,p) + s_{out}(n_0)\cdot STRING^R(\varphi_M(n,p-1,p-1,p-1,p-1),t,u,p-1)$

In table 11 we have

$$\mathrm{STRING}^{R}(\mathbf{n},t,u,p) =$$

$$\tau_{I_k} \circ \partial_{H_k}\{\|_{-k\le \ell \le -p} CR^{L}{}_{\ell}(n_{\ell},u_{-\ell},n_{\ell+1})\|$$

$$\|_{-p<\ell<0} CS^{R}{}_{\ell}(n_{\ell},t_{p+\ell})\|CS^{++}{}_{0}(n_0)\|$$

$$\|_{0<\ell<p} CS^{L}{}_{\ell}(n_{\ell},u_{p-\ell})\|\|_{p\le \ell \le k} CR^{R}{}_{\ell}(n_{\ell},t_{\ell},n_{\ell-1})\}$$

There are three possible atomic actions: $r_{-k-1}(s,n)$ (for some s,n), $r_k(s',n')$ (for some s',n') and $s_{out}(n_0)$. We shall work out the first and the third one. The second one is a complementary version of the first case and therefore omitted.

(i) After performing $r_{-k-1}(s,n)$ (for some s,n) we obtain:

$$\tau_{I_k} \circ \partial_{H_k}\{\|CS_{-k}(n'_{-k},s,u_k)\|CR^{L}{}_{-k+1}(n_{-k+1},u_{k-1},n_{-k+2})\|\dots\}$$

'...' is used to indicate the part of the expression which is not affected or changed. Using n'_{-k}, we mean that n'_{-k} as used in the definition of the φ functions (the same for n'_{ℓ}).

Using the conditional axioms CA5, CA6 and CA7 we can derive this equals

$$\tau_{I_k} \circ \partial_{H_k} \circ \tau_{I_{-k,-k+1}} \circ \partial_{H_{-k,-k+1}}\{CS_{-k}(n'_{-k},s,u_k)\|CR^{L}{}_{-k+1}(n_{-k+1},u_{k-1},n_{-k+2})\|\dots\}$$

Using standard concurrency and the conditional axioms CA1, CA7 and CA2 we obtain:

$$\tau_{I_k} \circ \partial_{H_k} \circ \tau_{I_{-k,-k+1}} \circ \partial_{H_{-k,-k+1}}\{\tau_{I_{-k,-k+1}} \circ \partial_{H_{-k,-k+1}}[CS_{-k}(n'_{-k},s,u_k)\|CR^{L}{}_{-k+1}(n_{-k+1},u_{k-1},n_{-k+2})]\|\dots\}$$

Using the fact mentioned above, the equation $\tau\cdot x\|y = \tau\cdot(x\|y)$ (proven to hold in [BBK2]) and the conditional axioms in the other direction, we have:

$$\tau\cdot\tau_{I_k} \circ \partial_{H_k}\{CS^{L}{}_{-k}(n'_{-k},u_k)\|CS_{-k+1}(n'_{-k+1},s,u_{k-1})\|\dots\}$$

This calculation holds for the cells -k+1 and -k+2, cells -k+2 and -k+3 etc. until cells -p-1 and -p. Finally we have:

$$\tau\cdot\tau_{I_k} \circ \partial_{H_k}\{\|_{-k\le \ell \le -p-1} CS^{L}{}_{\ell}(n'_{\ell},u_{-\ell})\|CS_{-p}(n'_{-p},s,u_p)\|$$

$$\|_{-p<\ell<0} CS^{R}{}_{\ell}(n_{\ell},t_{p+\ell})\|CS^{++}{}_{0}(n_0)\|$$

$$\|_{0<\ell<p} CS^{L}{}_{\ell}(n_{\ell},u_{p-\ell})\|\|_{p\le \ell \le k} CR^{R}{}_{\ell}(n_{\ell},t_{\ell},n_{\ell-1})\}$$

and this is exactly what we wanted because this equals $\tau\cdot STRING^{RR}(\varphi_L(n,p),ts,u,p)$. Then, with axiom T1 we have the desired result.

(ii) the case $r_k(s',n')$ (for certain s', n'), proceeds in the same way as case (i).

(iii) After performing $s_{out}(n_0)$ the expression becomes: $\tau_{I_k}\circ\partial_{H_k}\{\ldots\|CR_0(n_0)\|\ldots\}$

Cell 0 can receive the pairs (n_{-1},t_{p-1}) and (n_1,u_{p-1}). Using the conditional axioms and standard concurrency in the same way as in case (i) we obtain:

$$\tau_{I_k}\circ\partial_{H_k}\circ\tau_{I_{-k,-k+1}}\circ\partial_{H_{-k,-k+1}}\{\tau_{I_{-k,-k+1}}\circ\partial_{H_{-k,-k+1}}[CR_0(n_0)\,\|CS^L{}_1(n_1,u_{p-1})]\|\ldots\}$$

Using fact*, $\tau\cdot x\|y = \tau\cdot(x\|y)$, and the conditional axioms in a backward direction we have:

$$\tau\cdot\tau_{I_k}\circ\partial_{H_k}\{\ldots\|CR_0(n_0,u_{p-1},n_1)\|CR_1(n_1)\|\ldots\}$$

In this way we can compute all the internal actions in any order we want. Having executed all the internal actions we obtain the desired stable machine state. After cell 0 has sent n_0 through channel out cell 0 receives the pairs (n_{-1},t_{p-1}) and (n_1,u_{p-1}) from the cells -1 and 1 respectively. After these actions cell 1 can receive the pairs (n'_0,t_{p-1}) and (n_2,u_{p-2}) from cells 0 and 2 respectively. This can be done also for the cells -2 and 2 until cells -p+2 and p-2. After all these calculations we have as a final result:

$$\begin{aligned}\tau\cdot\tau_{I_k}\circ\partial_{H_k}\{&\|_{-k\le\ell\le-p}CR^L{}_\ell(n_\ell,u_{-\ell},n_{\ell+1})\|CR^L{}_{-p+1}(n_{-p+1},u_{p-1},n'_{-p+2})\|\\ &\|_{-p+1<\ell<0}CS^R{}_\ell(n'_\ell,t_{p-1+\ell})\|CS^{++}{}_0(n_0)\|\\ &\|_{0<\ell<p-1}CS^L{}_\ell(n_\ell,u_{p-1-\ell})\|CR^R{}_{p-1}(n_{p-1},t_0,n'_{p-2})\\ &\|_{p\le\ell\le k}CR^R{}_\ell(n_\ell,t_\ell,n_{\ell-1})\}\end{aligned}$$

and this precisely $\tau\cdot STRING^R(\varphi_M(n,p-1,p-1,p-1,p-1),t,u,p-1)$. So, by (i), (ii), and (iii) we have verified the equation we started with. All the other cases proceed in the same way using the technique of first working out the internal actions. *end of proof table 14.*

Using the specification of table 14 we will give an interpretation of the variables of table 11 in terms of machine states. We have listed this interpretation in table 15. We use some shorthands for the states of the environment. Although these shorthands seem rather self-evident, some explanation is given. The first superscript indicates what kind of actions the environment wants to perform: send (S) or receive (R) actions. The additional superscript (L, R or blank) indicates to what extent these action have been performed. R or L indicates that a send or receive action has to be performed at the right or left respectively. A blank indicates that both actions have to be performed. The subscript j indicates that the j-th pair of send or receive actions is performed. The notation $v_{j,1}$ is used to indicate string $v_{1,j}$ in reversed order. Notice that k+1 pairs have to be sent to the machine, while k pairs have to be received from the machine. When α_{ij} is mentioned twice in the number sequence n the same number is meant. This is not true for the numbers α_{i0} and α_{0j}: these may occur more than once but the total length is always 2k+1. When α_{ij} ($i\neq 0$ and $j\neq 0$) is mentioned twice the numbers suggested by the notation to be between these numbers can be omitted.

$\underline{1\leq i,j\leq k,\ i\leq j}$

$$EI^{S}_{i,j} = \tau_{I'k} \circ \partial_{H'k}\{ENV^{R}_{j} \|$$

$$\|STRING^{S}(<\alpha_{j0}..\alpha_{j0}\alpha_{j1}..\alpha_{ji-1}\alpha_{ji}\alpha_{j-1i}..\alpha_{i+1i}\alpha_{ii}\alpha_{ii+1}..\alpha_{ij-1}\alpha_{ij}\alpha_{i-1j}..\alpha_{1j}\alpha_{0j}..\alpha_{0j}>,$$
$$,v_{j,1}\square^{k+1-j}, w_{j,1}\square^{k+1-j}, j+1-i)\}$$

$$EI^{SR}_{i,j} = \tau_{I'k} \circ \partial_{H'k}\{ENV^{RR}_{j} \|$$

$$\|STRING^{SR}(<\alpha_{j0}..\alpha_{j0}\alpha_{j1}..\alpha_{ji-1}\alpha_{ji}\alpha_{j-1i}..\alpha_{i+1i}\alpha_{ii}\alpha_{ii+1}..\alpha_{ij-1}\alpha_{ij}\alpha_{i-1j}..\alpha_{1j}\alpha_{0j}..\alpha_{0j}>,$$
$$,v_{j,1}\square^{k+1-j}, w_{j,1}\square^{k-j}, j+1-i)\}$$

$$EIS^{SL}_{i,j} = \tau_{I'k} \circ \partial_{H'k}\{ENV^{RL}_{j} \|$$

$$\|STRING^{SL}(<\alpha_{j0}..\alpha_{j0}\alpha_{j1}..\alpha_{ji-1}\alpha_{ji}\alpha_{j-1i}..\alpha_{i+1i}\alpha_{ii}\alpha_{ii+1}..\alpha_{ij-1}\alpha_{ij}\alpha_{i-1j}..\alpha_{1j}\alpha_{0j}..\alpha_{0j}>,$$
$$,v_{j,1}\square^{k-i}, w_{j,1}\square^{k+1-j}, j+1-i)\}$$

---

table 15. Correspondence between $\tau_{I'k} \circ \partial_{H'k}\{ENV_k(v,w)\|INIT_k\}$ and table 11 (to be continued).

$1 \le i \le k+1,\ 1 \le j \le k,\ i = j+1$

$$EI^{S}_{i,j} = \tau_{l'k} \circ \partial_{H'k}\{ENV^{R}_{j} \,\|\, \|STRING^{S}(<\alpha_{j0}..\alpha_{j0}\alpha_{j1}\ldots\alpha_{jj-1}\alpha_{jj}\alpha_{j-1j}..\alpha_{1j}\alpha_{0j}\ldots\alpha_{0j}>, v_{j,1}\square^{k+1-j}, w_{j,1}\square^{k+1-j}, 0)\}$$

$$EI^{SR}_{i,j} = \tau_{l'k} \circ \partial_{H'k}\{ENV^{RR}_{j} \,\|\, \|STRING^{SR}(<\alpha_{j0}..\alpha_{j0}\alpha_{j1}\ldots\alpha_{jj-1}\alpha_{jj}\alpha_{j-1j}..\alpha_{1j}\alpha_{0j}\ldots\alpha_{0j}>, v_{j,1}\square^{k+1-j}, w_{j,1}\square^{k-j}, 0)\}$$

$$EIS^{SL}_{i,j} = \tau_{l'k} \circ \partial_{H'k}\{ENV^{RL}_{j} \,\|\, \|STRING^{SL}(<\alpha_{j0}..\alpha_{j0}\alpha_{j1}\ldots\alpha_{jj-1}\alpha_{jj}\alpha_{j-1j}..\alpha_{1j}\alpha_{0j}\ldots\alpha_{0j}>, v_{j,1}\square^{k-j}, w_{j,1}\square^{k+1-j}, 0)\}$$

$1 \le i \le k+1,\ j = k+1$

$$EI^{S}_{i,j} = \tau_{l'k} \circ \partial_{H'k}\{STRING^{S}(<\alpha_{k+11}..\alpha_{k+1i-1}\alpha_{k+1i}\alpha_{ki}..\alpha_{i+1i}\alpha_{ii}\alpha_{ii+1}..\alpha_{ik}\alpha_{ik+1}\alpha_{i-1k+1}..\alpha_{1k+1}>, ,v_{k+1,1}, w_{k+1,1}, k+2-i)\}$$

$1 \le i,j \le k,\ i \le j$

$$EI^{R}_{i,j} = \tau_{l'k} \circ \partial_{H'k}\{ENV^{S}_{j+1} \,\|STRING^{R}(<\alpha_{j0}..\alpha_{j0}\alpha_{j1}..\alpha_{ji-1}\alpha_{ji}\alpha_{j-1i}..\alpha_{i+1i}\alpha_{ii}\alpha_{ii+1}.. ..\alpha_{ij-1}\alpha_{ij}\alpha_{i-1j}..\alpha_{1j}\alpha_{0j}..\alpha_{0j}>,v_{j,1}\square^{k-j}, w_{j,1}\square^{k-j}, j+1-i)\}$$

$$EI^{RR}_{i,j} = \tau_{l'k} \circ \partial_{H'k}\{ENV^{SR}_{j+1} \|STRING^{RR}(<\alpha_{j+10}..\alpha_{j+10}\alpha_{j+11}..\alpha_{j+1i-1}\alpha_{j+1i}\alpha_{ji}..\alpha_{i+1i}\alpha_{ii}\alpha_{ii+1}.. ..\alpha_{ij-1}\alpha_{ij}\alpha_{i-1j}..\alpha_{1j}\alpha_{0j}..\alpha_{0j}>,,v_{j+1,1}\square^{k-j}, w_{j,1}\square^{k-j}, j+1-i)\}$$

$$EIS^{RL}_{i,j} = \tau_{l'k} \circ \partial_{H'k}\{ENV^{SL}_{j+1} \|STRING^{RL}(<\alpha_{j0}..\alpha_{j0}\alpha_{j1}..\alpha_{ji-1}\alpha_{ji}\alpha_{j-1i}..\alpha_{i+1i}\alpha_{ii}\alpha_{ii+1}.. ..\alpha_{ij}\alpha_{ij+1}\alpha_{i-1j+1}..\alpha_{1j+1}\alpha_{0j}..\alpha_{0j}>,v_{j,1}\square^{k-j},w_{j+1,1}\square^{k-j},j+1-i)\}$$

$1 \le i \le k+1,\ 0 \le j \le k,\ i = j+1$

$$EI^{R}_{i,j} = \tau_{l'k} \circ \partial_{H'k}\{ENV^{S}_{j+1} \|STRING^{R}(<\alpha_{j0}..\alpha_{j1}..\alpha_{jj-1}\alpha_{jj}\alpha_{j-1j}..\alpha_{1j}..\alpha_{0j}>,v_{j,1}\square^{k-j},w_{j,1}\square^{k-j},0)\}$$

$$EI^{RR}_{i,j} = \tau_{l'k} \circ \partial_{H'k}\{ENV^{SR}_{j+1} \,\| \|STRING^{RR}(<\alpha_{j+10}..\alpha_{j+10}\alpha_{j+11}..\alpha_{j+1j}\alpha_{jj}\alpha_{j-1j}..\alpha_{1j}\alpha_{0j}..\alpha_{0j}>,v_{j+1,1}\square^{k-j},w_{j,1}\square^{k-j},0)\}$$

$$EIS^{RL}_{i,j} = \tau_{l'k} \circ \partial_{H'k}\{ENV^{SL}_{j+1} \,\| \|STRING^{RL}(<\alpha_{j0}..\alpha_{j0}\alpha_{j1}..\alpha_{jj-1}\alpha_{jj}\alpha_{jj+1}..\alpha_{1j+1}\alpha_{0j+1}..\alpha_{0j+1}>,v_{j,1}\square^{k-j},w_{j+1,1}\square^{k-j},j+1-i \ \}$$

table 15. continued.

We will prove that this interpretation of the variables is a correct one. We will not write out all cases but just two of them:

(i) $1 \le i,j \le k,\ i \le j$

$$EI^S_{i,j} = \tau_{I'_k} \circ \partial_{H'_k}\{ENV^R_j \,\|$$

$$\|STRING^S(<\alpha_{j0}..\alpha_{j0}\alpha_{j1}..\alpha_{ji-1}\alpha_{ji}\alpha_{j-1i}..\alpha_{i+1i}\alpha_{ii}\alpha_{ii+1}..\alpha_{ij-1}\alpha_{ij}\alpha_{i-1j}..\alpha_{1j}\alpha_{0j}..\alpha_{0j}>,$$

$$,v_{j,1}\Box^{k+1-j}, w_{j,1}\Box^{k+1-j}, j+1-i)\}$$

In table 10 we have:

$$EI^S_{i,j} = \tau \cdot EI^{SL}_{i,j} + \tau \cdot EI^{SR}_{i,j} + s_{out}(\alpha_{ii}) \cdot EI^S_{i+1,j} \qquad 1 \le i,j \le k,\ i \le j$$

The environment can perform two actions: $r_{-k-1}(s',n')$ and $r_k(s,n)$ for certain (s',n') and (s,n). The machine state $STRING^S(\ldots)$ corresponds to state (xiii) of table 13, where $p=j+1-i$ and $t = v_{j,1}\Box^{k+1-j}$ and $u = w_{j,1}\Box^{k+1-j}$. In table 14 we have specified the states which can be reached after performing the possible actions:

$$STRING^S(\mathbf{n},t,u,p) = s_{-k-1}(u_k,n_{-k}) \cdot STRING^{SR}(\mathbf{n},t,u\text{-},p) +$$

$$+ s_k(t_k,n_k) \cdot STRING^{SL}(\mathbf{n},t\text{-},u,p) + s_{out}(n_0) \cdot STRING^S(\varphi_M(\mathbf{n},p\text{-}1,p\text{-}1,p\text{-}2,p\text{-}2),t,u,p\text{-}1)$$

The two communications, $c_{-k-1}(\Box,\alpha_{j0})$ and $c_k(\Box,\alpha_{0j})$ at the channels -k-1 and k give the following results after encapsulation:

$$\tau \cdot \tau_{I'_k} \circ \partial_{H'_k}\{ENV^{RR}_j\|$$

$$STRING^{SR}(<\alpha_{j0}..\alpha_{j0}\alpha_{j1}..\alpha_{ji-1}\alpha_{ji}\alpha_{j-1i}..\alpha_{i+1i}\alpha_{ii}\alpha_{ii+1}..\alpha_{ij-1}\alpha_{ij}\alpha_{i-1j}..\alpha_{1j}\alpha_{0j}..\alpha_{0j}>,$$

$$,v_{j,1}\Box^{k+1-j}, w_{j,1}\Box^{k-j}, j+1-i)\}$$

and

$$\tau \cdot \tau_{I'_k} \circ \partial_{H'_k}\{ENV^{RL}_j\|$$

$$STRING^{SL}(<\alpha_{j0}..\alpha_{j0}\alpha_{j1}..\alpha_{ji-1}\alpha_{ji}\alpha_{j-1i}..\alpha_{i+1i}\alpha_{ii}\alpha_{ii+1}..\alpha_{ij-1}\alpha_{ij}\alpha_{i-1j}..\alpha_{1j}\alpha_{0j}..\alpha_{0j}>,$$

$$,v_{j,1}\Box^{k-j}, w_{j,1}\Box^{k+1-j}, j+1-i)\}$$

Using the interpretation of table 15, these terms correspond to $\tau \cdot EI^{SR}_{i,j}$ and $\tau \cdot EI^{SL}_{i,j}$ respectively. What is left, is the third term as a result of the send action along channel out. After performing the action $s_{out}(\alpha_{ii})$ we can verify, using the function φ_M, that the term becomes:

i=j: $\tau\cdot\tau_{l'_k}\circ\partial_{H'_k}\{ENV^R_j \|$

$$STRING^S(<\alpha_{j0}..\alpha_{j0}\alpha_{j1}\ldots\alpha_{jj-1}\alpha_{jj}\alpha_{j-1j}..\alpha_{1j}\alpha_{0j}\ldots\alpha_{0j}>, v_{j,1}\square^{k+1-j}, w_{1,j}\square^{k+1-j}, 0)\}$$

or

i<j: $\tau\cdot\tau_{l'_k}\circ\partial_{H'_k}\{ENV^R_j \|$

$$STRING^S(<\alpha_{j0}.\alpha_{j0}\alpha_{j1}..\alpha_{ji}\alpha_{ji+1}\alpha_{j-1i+1}..\alpha_{i+2i+1}\alpha_{i+1i+1}\alpha_{i+1i+2}..\alpha_{i+1j-1}\alpha_{i+1j}\alpha_{ij}..\alpha_{1j}\alpha_{0j}..\alpha_{0j}>$$
$$, v_{j,1}{}^{k+1-j}, w_{j,1}\square^{k+1-j}, j-i)\}$$

and these two terms correspond exactly to $\tau\cdot EI^S_{i+1,j}$.

That this last term is really obtained using the function φ_M can be seen as follows:

$$n = <\alpha_{j0}..\alpha_{j0}\alpha_{j1}..\alpha_{ji-1}\alpha_{ji}\alpha_{j-1i}..\alpha_{i+1i}\alpha_{ii}\alpha_{ii+1}..\alpha_{ij-1}\alpha_{ij}\alpha_{i-1j}..\alpha_{1j}\alpha_{0j}..\alpha_{0j}>$$

$$t = v_{j,1}\square^{k+1-j}, \qquad u = w_{j,1}\square^{k+1-j}, \qquad p = j+1-i$$

The function φ_M becomes:

$\ell = 0$: $\quad n'_0 = f(\alpha_{i+1i},\alpha_{ii},\alpha_{ii+1},(v_{j,1}\square^{k+1-j})_{p-2},(w_{j,1}\square^{k+1-j})_{p-2})$

Using $(v_{j,1}\square^{k+1-j})_{p-2} = v_{i+1}$ and $(w_{j,1}\square^{k+1-j})_{p-2} = w_{i+1}$ and the definition of f we get: $n'_0 = \alpha_{i+1i+1}$

$\ell = -1$: $\quad n'_{-1} = f(\alpha_{i+2i},\alpha_{i+1i},\alpha_{i+1i+1},(v_{j,1}\square^{k+1-j})_{p-3},(w_{j,1}\square^{k+1-j})_{p-2})$

Using $(v_{j,1}\square^{k+1-j})_{p-3} = v_{i+2}$ and $(w_{j,1}\square^{k+1-j})_{p-2} = w_{i+1}$ and the definition of f we obtain:

$n'_{-1} = \alpha_{i+2i+1}$.

This can be done in the same way for all $-p+1<\ell<p-1$.

(ii) <u>$1\le i,j\le k$, $i\le j$, $j<k$</u>

$EI^{RR}_{i,j} = \tau_{l'_k}\circ\partial_{H'_k}\{ENV^{SR}_{j+1} \|$

$$STRING^{RR}(<\alpha_{j+10}..\alpha_{j+10}\alpha_{j+11}..\alpha_{j+1i-1}\alpha_{j+1i}\alpha_{ji}..\alpha_{i+1i}\alpha_{ii}\alpha_{ii+1}..\alpha_{ij-1}\alpha_{ij}\alpha_{i-1j}..\alpha_{1j}\alpha_{0j}..\alpha_{0j}>,$$
$$,v_{j+1,1}\square^{k-j},w_{j,1}\square^{k-j},j+1-i)\}$$

We have to prove that this term satisfies the following equation under the interpretation of table 15:

$$EI^{RR}_{i,j} = \tau\cdot EI^S_{i,j+1} + s_{out}(\alpha_{ii})\cdot EI^{RR}_{i+1,j}$$

The machine can perform three actions: $s_{-k-1}(\square,\alpha_{j+10})$, $r_k(s',n')$ (for some (s',n')) and $s_{out}(\alpha_{ii})$ (see

also table 13). The corresponding equation is:

$$STRING^{RR}(\mathbf{n},t,u,p) = s_{-k-1}(u_k,n_{-k})\cdot\Omega + \sum r_k(s',n')\cdot STRING^{S}(\varphi_R(\mathbf{n},p),t,s'u,p+1)$$
$$+ s_{out}(n_0)\cdot STRING^{RR}(\varphi_M(\mathbf{n},p,p-1,p-1,p-1),t,u,p-1)$$

It is not difficult to verify that the environment can not answer the first action: this action is redundant as already suggested. Performing the communication $c_k(w_{j+1},\alpha_{0j+1})$ we obtain:

$$\tau\cdot\tau_{I'k}\circ\partial_{H'k}\{ENV^{R}_{j+1} \parallel$$
$$STRING^{S}(<\alpha_{j+10}..\alpha_{j+10}\alpha_{j+11}..\alpha_{j+1i-1}\alpha_{j+1i}\alpha_{ji}..\alpha_{i+1i}\alpha_{ii}\alpha_{ii+1}..\alpha_{ij}\alpha_{ij+1}\alpha_{i-1j+1}\cdot\cdot$$
$$\alpha_{1j+1}\alpha_{0j+1}..\alpha_{0j+1}> ,v_{j+1,1}\square^{k-j}, w_{j+1,1}\square^{k-j} ,j+2-i)\}$$

The right side can be verified using the function φ_R, for example:

$\ell = k$: $\quad n'_k = f(\alpha_{0j},\alpha_{0j},\alpha_{0j+1},\square,s') = \alpha_{0j+1}$

$\ell = k+1-j$: $\quad n'_\ell = f(\alpha_{1j},\alpha_{0j},\alpha_{0j+1},v_1,s') = \alpha_{1j+1}$

The other cases are: $j+1-i \le \ell \le k$ and can be worked out in a similar way.

Performing the action $s_{out}(\alpha_{ii})$ gives:

$$\tau_{I'k}\circ\partial_{H'k}\{ENV^{SR}_{j+1} \parallel$$
$$STRING^{RR}(<\alpha_{j+10}..\alpha_{j+10}\alpha_{j+11}..\alpha_{j+1i}\alpha_{j+1i+1}\alpha_{ji+1}..\alpha_{i+2i+1}\alpha_{i+1i+1}\alpha_{i+1i+2}..\alpha_{i+1j-1}\alpha_{i+1j}\alpha_{ij}\cdot\cdot$$
$$\alpha_{1j}\alpha_{0j}..\alpha_{0j}>, v_{j+1,1}\square^{k-j}, w_{j,1}\square^{k-j}, j-i)\}$$

Using the function φ_M this is verified as follows:

$$\mathbf{n} = <\alpha_{j0}..\alpha_{j0}\alpha_{j1}..\alpha_{ji-1}\alpha_{ji}\alpha_{j-1i}..\alpha_{i+1i}\alpha_{ii}\alpha_{ii+1}..\alpha_{ij-1}\alpha_{ij}\alpha_{i-1j}..\alpha_{1j}\alpha_{0j}..\alpha_{0j}>$$
$$t = v_{j+1,1}\square^{k-j}, \qquad u = w_{j,1}\square^{k-j}, \qquad p = j+1-i .$$

The function φ_M becomes:

$\ell = 0$: $\quad n'_0 = f(\alpha_{i+1i},\alpha_{ii},\alpha_{ii+1},(v_{j,1}\square^{k+1-j})_{p-1},(w_{j,1}\square^{k-j})_{p-1})$.

Notice that the indices for string t start from 0 and for string u they start from 1.

Using $(v_{j,1}\square^{k+1-j})_{p-2} = v_{i+1}$ and $(w_{j,1}\square^{k+1-j})_{p-1} = w_{i+1}$ and the definition of f we get: $n'_0 = \alpha_{i+1i+1}$

$\ell = -1$: $\quad n'_{-1} = f(\alpha_{i+2i},\alpha_{i+1i},\alpha_{i+1i+1},(v_{j,1}\square^{k+1-j})_{p-2},(w_{j,1}\square^{k+1-j})_{p-1})$

Using $(v_{j,1}\square^{k+1-j})_{p-2} = v_{i+2}$ and $(w_{j,1}\square^{k+1-j})_{p-1} = w_{i+1}$ and the definition of f : $n'_{-1} = \alpha_{i+2i+1}$

This can be done in the same way for $-p+1 \leq \ell < p-1$. The other cases proceed in the same way. This ends the proof of the theorem. By proving that $\tau_{I'k} \circ \partial_{H'k}\{ENV_{k+1}(v,w) \| INIT_k\}$ and $\tau \cdot \Pi_{1 \leq \ell \leq k+1} s_{out}(\alpha_{\ell\ell}) \cdot \delta$ are both a solution of the specification of table 13, together with the fact that this specification is guarded and RSP we find that the specification of the string analyzer is correct. *end of proof.*

REFERENCES

[BBK1] J.C.M. Baeten, J.A Bergstra and J.W. Klop, *On the consistency of Koomen's Fair Abstraction Rule*, report CS-R8511, Centre of Mathematics and Computer Science, Amsterdam 1985, to appear in TCS.

[BBK2] J.C.M. Baeten, J.A Bergstra and J.W. Klop, *Conditional axioms and α/β–calculus in process algebra,* report FVI 86-17, Computer Science Department, University of Amsterdam 1986, to appear in: Proc. IFIP Conf. on Formal Description of Progr. Concepts, (M. Wirsing, ed.), Gl. Avernæs 1986, North Holland.

[BK1] J.A. Bergstra and J.W. Klop, *Algebra of communicating processes*, Proceedings of the CWI Symp. Math. and Comp. Sci., eds: J.W. de Bakker, M. Hazewinkel and J.K. Lenstra, North-Holland, pp. 89-138, Amsterdam 1986.

[BK2] J.A. Bergstra and J.W. Klop, *Algebra of communicating processes with abstraction*, Theor. Comp. Sci. 37, pp. 77-121, 1986.

[BK3] J.A. Bergstra and J.W. Klop, *Fair FIFO satisfy an algebraic criterion for protocol correctness*, report CS-R8405, Centre of Mathematics and Computer Science, Amsterdam 1984.

[HE] M. Hennessy, *Proving systolic systems correct*, TOPLAS 8 (3), pp. 344-387, 1986.

[KW] L. Kossen and W.P. Weijland, *Correctness proofs for systolic algorithms: palindromes and sorting*, report FVI 87-04, Computer Science Department, University of Amsterdam 1987.

[KM] C.P.J. Koymans and J.C. Mulder, *A modular approach to protocol verification using process algebra*, report LGPS 6, Department of Philosophy, University of Utrecht, 1985.

[LL] R.J. Lipton and D. Lopresti, *A systolic array for rapid string comparison*, Proc. of 1985 Chapel Hill Conference on Very Large Scale Integration (H. Fuchs ed.), Computer Science Press, pp 363-376, 1985.

[MC] C.A. Mead and L.A. Conway, *Introduction to VLSI-systems*, Addison-Wesley Publ. Comp., Reading, Massachusetts, 1980.

[MI] R. Milner, *A calculus of communicating systems*, Springer LNCS 92, 1980.

[MW] J.C. Mulder and W.P. Weijland, *Verification of an algorithm for log-time sorting by square comparison*, report CS-R8729, Centre of Mathematics and Computer Science, Amsterdam, 1987.

[RE] M. Rem, *Partially ordered computations with applications to VLSI-design*, Proc. Found. of Comp. Sc. IV.2 (J. de Bakker & J. van Leeuwen eds.), MC tract 159, pp. 1-44, MC, 1983.

[VA] F.W. Vaandrager, *Verification of two communication protocols by means of process algebra*, report CS-R8608, Centre of Mathemmatics and Computer Science, Amsterdam 1986.

[WF] R.A. Wagner and M.J. Fischer, *The string-to-string correction problem*, Journal of the Association for Computing Machinery, Vol. 21, No. 1, January 1974, pp. 168-173, 1974.

[WE] W.P. Weijland, *A systolic algorithm for matrix-vector multiplication*, report FVI 87-08, Computer Science Department, University of Amsterdam 1987.