

# TOWARDS A COMPLETE DESCRIPTION OF PROCESSING SYSTEMS

Hermann VON ISSENDORFF

Hauptstr. 40, D-21745 Hemmoor

Deutschland

email: hviss@compuserve.com

phone: 00 47 4771-5769

Received: September 30, 1997

## Abstract

The paper presents a unified term algebra for the complete description of every network of cooperating processing units no matter whether they are clocked or self-timed. The description is called complete because it covers the topology, the functions and the time behaviour of the network. Even storage can explicitly be described. A processing unit may be as small as a hardware gate or as large as any computer. The main purpose of the paper is a concise introduction of the basic elements, the operators, the sorts and the signature of the term algebra. Its expressive power is then demonstrated by the complete specification of a D-flipflop.

*Keywords:* processing network description, digital hardware description, network programming language, layout language.

## 1. Introduction

Nature has found a way of coding all the features of an organism into a small set of chromosomes. The features include the building plan of the organism, the building schedule of when, where and what kind of cells are to be generated, the spatial distribution and the operations of the organs and limbs, and the immense variety of internal and external asynchronous interactions. Starting from a single cell, the organism is generated by an increasing number of interacting cells.

Regarding the set of chromosomes as a set of linear programs and the set of cells as a set of processing units executing the programs models the organism as a huge concurrent network of asynchronously cooperating processing units. If this is compared to our ability of describing, building and programming computer systems all our seemingly great achievements shrink to humble first steps. On the other hand, this tremendous difference gives confidence that great improvements can still be expected in computer science.

The formalism presented in this paper may be regarded as a step towards this goal. This formalism makes it possible to describe every network

of cooperating processing units, no matter if they are clocked or self-timed. Even storage can be represented. The description covers the properties which can be paraphrased by the questions ‘what’, ‘when’ and ‘where’. The formalism is therefore considered as complete. In this paper we only give a short introduction of the basic features.

The language of the formalism is a term algebra, called acton algebra. The acton algebra specifies the topology of the actons in a biplanar space. An acton models a processing unit by specifying the functions, the operational behaviour and the physical properties. A least processing system is a hardware gate. Every nonelementary acton is an abstraction of a pair of adjacent smaller actons.

The functions and the time behaviour of the actons are defined by a novel temporal Boolean algebra derived from the physical model of a network of causally related binary events (VON ISSENDORFF, 1995, 1997). Here we will only give a brief introduction:

A binary event, or a *bent* for short, is the least possible event. Its occurrence generates a bit and thereby marks a point of time. Thus a causal relation between a pair of bents establishes a temporal as well as a functional dependency: A bent occurs only after its causally preceding bents have occurred. The bit is a function of the bits of the causally preceding bents.

Considering time as a consequence of causality implies an elementary gap of time between every pair of causally immediately related bents. Counting the maximum number of elementary time gaps between any two causally related bents gives rise to a time measure. Allowing forward and backward counting this method can be extended to finite causally coherent posets of bents. This way it becomes possible to establish a unique time metric, that means a linear ordering of time. Making use of the time metric all bents can then be transformed onto the time of a selected bent, i.e. into a set of simultaneous bents. This set of simultaneous bents represents the proper base for defining a three-valued Boolean algebra, called bent algebra. The bent algebra reduces to the classical Boolean algebra after all bents have occurred and thus have assumed their final values.

This paper is based on the following notions and notations:

- General equality is expressed by the symbol “=”.
- Equality between predicates or between binary events is called equivalence but designated by special symbols.
- The definitions are expressed by first order predicate logic. Negation, conjunction, implication and equivalence are designated by the symbols  $\sim$ ,  $\&$ ,  $\rightarrow$  and  $\leftrightarrow$ , universal and existential quantifiers by  $\forall$  and  $\exists$ , and truth values by  $T$  and  $F$ . The universal quantifier is usually omitted to increase readability.
- In bent algebra negation, conjunction, disjunction, implication and equivalence are designated by the symbols  $\neg$ ,  $\wedge$ ,  $\vee$ ,  $\Rightarrow$  and  $\Leftrightarrow$ , and truth values by  $1$ ,  $0$ , and  $\#$ .

- The letters  $u, v, x, y, z$  will be used to denote acton terms, and the letters  $a, b, c$  to denote single actons.

## 2. Acton Definition

Let  $E$  be a coherent finite poset of causally related bents. We will identify  $E$  as a process, that is as any finite system which changes with time. It may be the flight of a bird, the evolution of the weather, the production of a car, the evaluation of a computer program, or the operations of the computer while the program is evaluated.

An acton is a physical processing unit which is able to perform one or several activities. Each activity simulates a convex cutting from  $E$ . The attribute convex means that the cutting must not both precede and succeed another cutting. This way two activities either exclusively precede or succeed each other or are independent. With this definition a network of activities is an order preserving subset of the poset  $E$ .

A bent generates a bit of information thereby marking a point of time. It is represented by a three-valued variable which up to that point of time has the value '#', and either '1' or '0' thereafter. A bent constant is a bent which has already occurred and is therefore restricted to the final values  $\{1, 0\}$ . A signal is a bent with a prefixed final value. Since there are two final values there is a signal  $s \in \{\#, 1\}$  and there is a complementary signal  $\bar{s} \in \{\#, 0\}$ .

In two-state systems a bent needs two variables in order to be represented. Two-state systems can either be clocked or self-timed. The different codes are shown in *Table 1* and *Table 2*. In clocked systems a bent is defined by the conjunction of a bent constant and a signal. A bent constant is represented by a bit and a signal by the leading or trailing edge of a clock pulse. The actual states of a bit and a clock pulse are designated by  $\{H, L\}$ . In *Table 1* the leading edge has been chosen. In self-timing systems a bent is represented by a pair of bits only one of which assumes the state  $H$  during the finite time of observation.

*Table 1.* Bent representation in clocked systems

bent	clock pulse	bit
1	H	H
0	H	L
#	L	H,L

An acton defines the operational relations between the outgoing lines and the ingoing lines of the cutting, that means their functional and tempo-

Table 2. Bent representation in self-timed systems

bent	bit 1	bit 2
1	H	L
0	L	H
#	L	L

ral relations. The outgoing lines are represented by an ordered set of output bents and the ingoing lines by an ordered set of input bents. A least acton relates either one or two input bents to a single output bent.

An acton is represented by  $a[p]$ , where  $a$  is the acton name and  $p$  is a bent term representing a control condition. The output bents of  $a[p]$  assume a final value only if and after the control term  $p$  has assumed a final value. The output bents assume the value 0 if  $p$  becomes 0. They assume an individual final value only if  $p$  becomes 1. If  $p$  is constantly 1 the control condition is omitted, that is instead of  $a[1]$  just  $a$  is written.

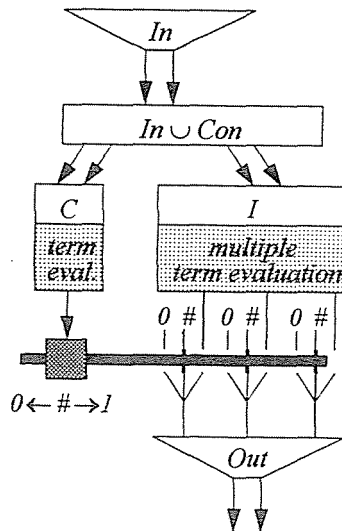


Fig. 1. Schematic view of an acton

Fig. 1 depicts the schematic view of an acton. Each output bent is an individual Boolean function over input bents. The set of input bents is the union of a set of actual input bents  $In$  and a set of internal bent constants  $Con$ . The internal bent constants may either be built-in and thus permanent, or a parameter, which is set prior to the occurrence of the input bents. The union  $In \cup Con$  splits up into the disjoint sets  $I$  and  $C$ , where  $I$  serves to individually define each output bent, and  $C$  serves to define the

control condition.  $In$  contains at least a signal, and  $Con$  contains at least the value '1'. The set of bents which may occur when the acton is active is called  $Gen$ . The output set  $Out$  is a subset of  $Gen$ , i.e.,  $Out \subseteq Gen$ .

An important feature is that an acton need not be clocked. In general the output bents are not synchronized but may occur at any time. Some output bents of an acton may actually never occur.

According to (VON ISSENDORFF, 1995, 1997) a coherent partial ordering of bents  $E$  can be mapped into a sequence of discrete time slices each of which is represented by a set of simultaneous bent mappings. The set of simultaneity sets forms a class which represents the linear time ordering. In general there are several linear time orderings which can be derived from  $E$ . Enumerating them by  $i$  the  $i$ th linear time ordering of  $E$  is designated by  $E/\sigma_i$ . Linear time orderings of subsets of  $E$  are accordingly designated, e.g.  $C/\sigma_i$ ,  $CCE$ . We are now ready to precisely define an acton by expressing each output bent as a function of terms over the information set  $I/\sigma_i$  and the control set  $C/\sigma_i$ :

$$\begin{aligned} \exists p \in F(C/\sigma_i). \forall r \in Out/\sigma_i. \exists q \in F(I/\sigma_i). \\ \text{if } p = \# \text{ then } r \Leftrightarrow \# \text{ else } r \Leftrightarrow p \wedge q. \end{aligned} \quad (1)$$

An acton is called neutral if the input information  $I \subseteq In$  is directly transferred to the output, i.e., if there exists a one-to-one mapping between  $I$  and  $Out$ . A neutral acton will be designated by a  $\mathcal{S}$ -symbol. The cardinality of its input and output depends on the preceding or succeeding actons and need not explicitly be specified for this reason. Several neutral actons will be distinguished by indexing. A neutral acton  $\mathcal{S}[p]$  is formally defined by

$$\begin{aligned} \exists p \in F(C/\sigma_i). \forall r \in Out/\sigma_i. \exists q \in F(I/\sigma_i). \\ \text{if } p = \# \text{ then } r \Leftrightarrow \# \text{ else } r \Leftrightarrow p \wedge q. \end{aligned} \quad (2)$$

If a neutral acton is unconditional, i.e., if  $p=1$ , each output bent is a 1-function of an individual input bent. Both sets have the same order and the same cardinality. Unconditional neutral actons represent the wiring between actons which are not directly connected. They can also be regarded as a bypass modelling the flow of data.

In synchronized systems as for instance in clocked systems one needs to know when an acton terminates, that means when all output bents have assumed their final states. This can be achieved by generating an extra termination signal  $so$  from the conjunction of the signals of the individual output bents, i.e., by

$$so = \forall r \in Out. \pi(r \vee \neg r). \quad (3)$$

Note that the generation of  $so$  takes time, meaning that  $so$  is only available one time step after the occurrence of the last output bent. In the same

way we may introduce an activation signal  $si$  as the signal immediately succeeding the occurrence of all input bents, i.e.,

$$si = \forall q \in In. \pi(q \vee \neg q). \quad (4)$$

The evaluation time of an acton can be determined by means of a time function  $\mu$  which was introduced by definition (11) in (VON ISSENDORFF, 1995)

$$n_a \times t_0 = \mu(so_a) - \mu(si_a). \quad (5)$$

The evaluation time of an acton  $a$  is defined as the number  $n_a$  of an elementary time step  $t_0$ . Note, that  $n_a$  may only be available after termination.

An elementary acton represents a hardware gate. For each Boolean function denoted by *NOT*, *AND*, *OR*, *NAND*, *NOR* there is a corresponding elementary acton denoted by  $N$ ,  $A$ ,  $O$ ,  $\bar{A}$ ,  $\bar{O}$ . Actually, since every computable function can be expressed by either *NAND*- or *NOR*-functions it would suffice to admit only actons of type  $\bar{A}$  or of type  $\bar{O}$ .

### 3. Acton Term Operators

An acton network is a two-dimensional representation of a coherent poset of actons. An acton network with a single input and a single output is called an acton term. A single acton is an elementary acton term.

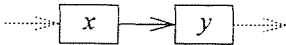
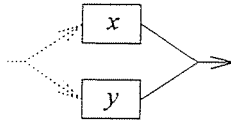
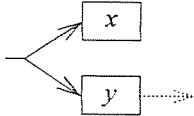
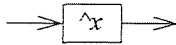
In order to formally describe an acton network three binary connectives, a unary operator and two sorts of delimiters need to be introduced. That will be the topic of this section.

The connectives together with their topological as well as functional properties are shown in *Table 3*.

Two different acton terms  $x$ ,  $y$  where  $x$  exclusively precedes  $y$  and where  $y$  exclusively succeeds  $x$  form a sequential term which has the same input as  $x$  and the same output as  $y$ . This relation is expressed by a binary connective designated by a 'greater' symbol. A term  $(x > y)$  is functionally defined by  $In(x > y) = In_x$ ,  $Out(x > y) = Out_y$ , and  $Out_x = In_y$ . In addition,  $(x > y)$  defines a direction of propagation from term  $x$  to term  $y$ . The activities, i.e., the occurrences of bents, propagate from the acton network  $x$  to the acton network  $y$ . More precisely, each elementary acton of term  $y$  has at least one elementary acton of term  $x$  as a predecessor. Since the acton networks are assumed to be different each is necessarily located at a different place. Thus the propagation of activities means a change in time and space.

Two different acton terms  $x$ ,  $y$  which commonly precede or succeed another term represent a concurrent term. The first case is called a join term and designated by a double slash. The second case is called a fork term and designated by a single slash. A join term  $(x // y)$  is functionally defined by  $In(x // y) = In_x \cup In_y$ ,  $Out(x // y) = Out_x \cup Out_y$ , and  $Out_x \neq \phi$ ,  $Out_y \neq \phi$ . The terms  $x$  and  $y$  need not have a common predecessor. Therefore, the input

Table 3. Properties of the operators

Sequence: $(x>y)$		$In_{(x>y)} = In_x$ $Out_{(x>y)} = Out_y$ $Out_x = In_y$
Join: $(x//y)$		$In_{(x//y)} = In_x \cup In_y$ $Out_{(x//y)} = Out_x \cup Out_y$ $Out_x \neq \phi, Out_y \neq \phi$
Fork: $(x/y)$		$In_{(x/y)} = In_x \cup In_y$ $In_x \neq \phi, In_y \neq \phi$ $Out_{(x/y)} = Out_x \cup Out_y$ $Out_x = \phi$
Reflection: $(^x)$		$In_{^x} = ^In_x$ $Out_{^x} = ^Out_x$

of either one of them or both may as well be empty. The fork term  $(x/y)$  is functionally defined by  $In(x/y) = In_x \cup In_y$ ,  $Out(x/y) = Out_x \cup Out_y$ ,  $In_x \neq \phi$ ,  $In_y \neq \phi$ . At most one of the terms  $x, y$  may have a successor and therefore may have a nonempty output.

Since the terms  $x$  and  $y$  in  $(x//y)$  as well as in  $(x/y)$  are functionally independent they are temporally independent, too. There is nothing by which a temporal ordering could be defined. Nevertheless they do have a spatial ordering orthogonal to the direction of propagation. In a two-dimensional space this ordering can be defined by a left/right relation. Here we stipulate that the first term of a join or a fork is at the left side of the other.

Finally we introduce a unary operator which reflects an acton term at the direction of propagation. This reflection operator is designated by the prefix symbol  $\wedge$ . The reflection reverses the order of the input and output of a term. For convenience the reversed order of a set is also expressed by the  $\wedge$ -symbol, e.g.,  $\wedge In_x = In_{\wedge x}$ .

As a result we get a description of acton networks in a two-dimensional space, where one dimension is distinguished by the direction of propagation. This topology can be turned into metric if the real width and length of the actons are taken into account. It is thus possible to calculate the physical area of any planar processing system. A particular important area of application is the layout of digital circuits.

The inclusion of time into the metric is not quite that simple. Every elementary acton has a time delay that is given by technology. However,

the time delay of an acton network can only be determined if the number of loop cycles is known. Thus in general a time metric can only be introduced if there are no loops at all like in combinational digital circuits or if the number of loop cycles is predetermined.

Table 4. Properties of the general delimiters

$*_i[p]:$	$p=1$ $In_{*_i}=\phi$ $Out_{*_i}=read(*_i)$	$*'_j[q]:$	$q=1$ $In_{*_j}=write(*'_j)$ $Out_{*_j}=\phi$
$i=j \rightarrow Out_{*_i}=In_{*_j}$			

A finite partial ordering has at least one minimal and at least one maximal element. In acton networks these elements are represented by delimiters. The delimiters are special actons which serve as links to the environment. A delimiter for the representation of minimal elements is designated by a star, and a delimiter for the representation of maximal elements is designated by a primed star. Both delimiters are assumed to be unconditional. In an abstract semantical interpretation, a  $\star$ -acton comprises the complete causal pre-world of its output bents and a  $\star'$ -acton the complete causal post-world of its input bents. In a more concrete semantical interpretation, a  $\star$ -acton serves as an entry which activates the subsequent acton network and supplies it with input parameters by importing a set of bents from the outside world. The least import of a  $\star$ -acton is an activation signal. Likewise, a  $\star'$ -acton serves as an exit which exports a set of bent to the outside world. The least export of a  $\star'$ -acton is a termination signal. A  $\star$ -acton can therefore be regarded as a reading or message receiving device and a  $\star'$ -acton as a writing or message sending device. The formal definitions of the delimiters are listed in Table 4. A  $\star$ -acton is characterized by an empty input and a nonempty output, and a  $\star'$ -acton by a nonempty input and an empty output. For comparison, the input and the output of internal actons are never empty. Both types of actons are always indexed. Identical indices denote that the output of the  $\star$ -acton and the input of the  $\star'$ -acton are identical. Such a pair is called matching. Note that a  $\star$ -acton and a  $\star'$ -acton which occur at different points of time can never match, although the output of the first and the input of the second may of course assume the same final values.

#### 4. A Language for Acyclic Acton Networks

The acton networks which can be represented by the operators introduced so far are acyclic, i.e., each acton can only perform an individual activity.



The network of actons is therefore isomorphic to the network of activities. A common acyclic network is a combinational hardware circuit. In the next section the language will be extended to represent every processing structure.

The acton and the two delimiters serve as terminals of three sorts of acton terms which are called *head*, *body* and *tail* and are symbolized by the bold abbreviations **h**, **b**, and **t**. A head term has an empty input and a nonempty output. It always begins with a  $\star$ -acton. A tail term has a nonempty input and an empty output. It always ends with a  $\star'$ -acton. A body term has a nonempty input and a nonempty output. Finally, there is the sort of complete acton networks designated by **as** which is characterized by a nonempty input and a nonempty output. Terms with a nonempty output, i.e., of sort **h** or **b**, are always succeeded by a term with a nonempty input, i.e., of sort **b** or **t**. Terms of sort **h** or **b** can therefore be designated as predecessors, and those of sort **b** or **t** as successors. The join-connective only applies to predecessors, the fork-connective only applies to successors, and the sequential connective only combines a predecessor and a successor.

Round brackets are used in order to delimit the terms. As usual, their number is diminished by introducing a precedence  $\wedge, //, /, >$ , where  $\wedge$  binds most.

Table 5 shows the signature of the basic network language.

Table 5. Signature of the basic language

$>$	$\mathbf{b \times b \rightarrow b}$	$//$	$\mathbf{b \times b \rightarrow b}$	$/$	$\mathbf{t \times t \rightarrow t}$	$\wedge$	$\mathbf{b \rightarrow b}$	$a[p]$	$\rightarrow \mathbf{b}$
	$\mathbf{h \times b \rightarrow h}$		$\mathbf{h \times b \rightarrow b}$		$\mathbf{t \times b \rightarrow b}$		$\mathbf{h \rightarrow h}$	$*$	$\rightarrow \mathbf{h}$
	$\mathbf{b \times t \rightarrow t}$		$\mathbf{b \times h \rightarrow b}$		$\mathbf{b \times t \rightarrow b}$		$\mathbf{t \rightarrow t}$	$\star'$	$\rightarrow \mathbf{t}$
	$\mathbf{h \times t \rightarrow as}$		$\mathbf{h \times h \rightarrow h}$		$\mathbf{as \times as \rightarrow as}$				

The language definition does not include the usual laws of commutativity, associativity, distributivity or idempotency, by which different but functionally identical terms are equated. This is because each acton expression is a unique description of a processing system. For instance substituting the join-term  $x//y$  by the join-term  $y//x$  generates a topologically different system. Equating both terms means a loss of topological information, and hence an abstraction. Thus instead of providing a set of term equations here we can only offer a set of term replacement rules which preserve the functions but change the topology. They are listed in Table 6. The rules are described by a term above and a term below a horizontal line and by vertical arrows indicating the direction of the replacement. If there are particular restrictions apart from those given by the connectives this is expressed by conditions which are put in square brackets beside the arrow. For instance,

the rules a and b are only defined for **b**-terms, the rule h for **h**-terms, and the rule i for **t**-terms.

Table 6. Topological term replacement rules

Identity:	a. $\frac{x \uparrow}{(x>S)\downarrow} [x \in b]$	b. $\frac{x \uparrow}{(S>x)\downarrow} [x \in b]$	
Commutativity:	c. $\frac{(x//y)\uparrow}{(y//x)\downarrow}$	d. $\frac{(x/y)\uparrow}{(y/x)\downarrow}$	
Associativity:	e. $\frac{((x>y)>z)\uparrow}{(x>(y>z))\downarrow}$	f. $\frac{((x//y)//z)\uparrow}{(x//(y//z))\downarrow}$	g. $\frac{((x/y)/z)\uparrow}{(x/(y/z))\downarrow}$
	h. $\frac{((x>y)//z)\uparrow}{(x>(y//z))\downarrow} [z \in h]$	i. $\frac{(z/(x//y))\uparrow}{((z/x)//y)\downarrow} [z \in t]$	
Reflexivity:	j. $\frac{x \uparrow}{\wedge(\wedge x)\downarrow}$		
Distributivity:	k. $\frac{\wedge(x>y)\uparrow}{(\wedge x>\wedge y)\downarrow}$	l. $\frac{\wedge(x//y)\uparrow}{(\wedge y//\wedge x)\downarrow}$	m. $\frac{\wedge(x/y)\uparrow}{(\wedge y/\wedge x)\downarrow}$

The language defined by the operators is recursively reducible. This means that every valid acton expression can be reduced to a single term by repeatedly substituting binary terms by single terms. At first glance, this property seems to severely restrict the types of network structures which can be described by the language. And indeed just a simple network like that at the left side of Fig. 2 cannot be reduced recursively and therefore does not have a language representation.

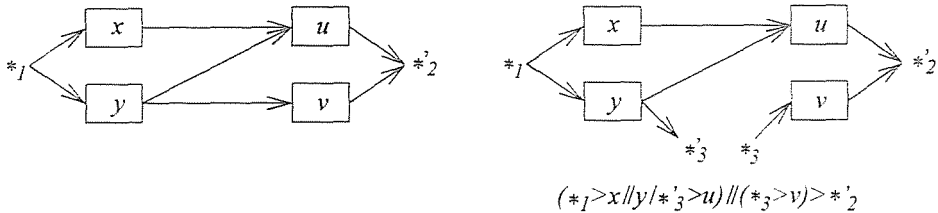


Fig. 2. Nonreducible and reducible networks

Fortunately this problem can be solved by introducing an implicit representation of the unconditional neutral acton, i.e., by replacing  $S_i$  by a pair of matching delimiters  $(*_i, *_i)$ , where  $i \in \mathbb{N}$  and  $In_{*_i} = In_{S_i}$  and  $Out_{*_i} = In_{S_i}$ . This makes it possible to transform every nonreducible ex-

pression into a reducible expression by turning some of the sequential connections of a nonreducible acton network into a pair of delimiters.

The join-connection does not only serve for the description of what is generally conceived as concurrency but also for the description of alternatives. In ordinary programming languages the alternatives are represented as *if* or *if-then-else*-statements. Actually, every alternative consists of two independent branches, which could concurrently be evaluated. In an acton network an alternative is just a join-connection of two acton terms with complementary conditions followed by a special acton  $O^+$  acting as a multiple OR. The output of  $O^+$  is defined as a pairwise disjunction over the output bents of the upper and the lower preceding terms, where  $card(Out_a) = card(Out_b)$ . The pairwise evaluation can be formally captured by a scalar product over the outputs.  $O^+$  can thus completely be defined by  $\varphi(Out_{O^+}) = \vee(Out_a \cdot Out_b)$ , where  $\varphi$  describes the delay.

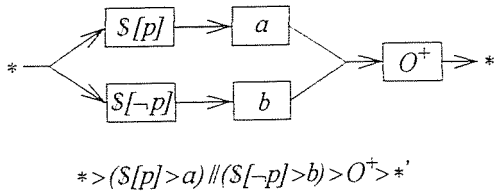


Fig. 3. If-then-else network

Fig. 3 depicts the familiar *if-then-else*-statement of a conventional programming language. A simple *if*-statement is described if  $b$  is a neutral acton.

### 5. A General Language for Acton Networks

In this section we are introducing another pair of delimiters by which two sequentially connected acton terms can be wound up into a topological loop. The delimiters are called loop head and loop tail and are designated by a circle and a primed circle symbol. Both are presumed to not depend on external conditions. The  $o'$ -acton returns its input to the  $o$ -acton, and the  $o$ -acton joins the input of the  $o'$ -acton with its own input. The joining of both inputs is done by concatenation. Since there are two choices how the concatenation can be done we specify that the returned input is located at the right side of the  $o$ -input. A left hand location of the returned input is achieved by reflecting the  $o$ -acton.

The return of data from the  $o'$ -acton to the  $o$ -acton cannot explicitly be expressed by the language. Physically, it can be realized by means of a second plane. The formal definitions of the  $o$ -acton and the  $o'$ -acton are listed in Table 7.

Table 7. Properties of the loop delimiters

$o[p]: \quad p=1$ $In_o \neq \phi$ $I_o = (In_o, In_o')$ $Out_o \neq \phi$	$o'[p]: \quad p=1$ $In_o' \neq \phi$ $Out_o' = \phi$
--	--

The introduction of the loop delimiters gives rise to three more sorts of acton terms. First there are the two term sorts lh and lt, called loop head and loop tail, which are created by the delimiters. In addition there is a term sort le, called loop exit. The domain and range of the loop induced sorts and their relation to the basic sorts are shown in Table 8.

Table 8. Signature extensions for the general network language

$>: lh \times b \rightarrow lh$	$//: lh \times h \rightarrow lh$	$/: lt \times b \rightarrow le$	$\wedge: lh \rightarrow lh$	$o: \rightarrow lh$
$b \times lt \rightarrow lt$	$h \times lh \rightarrow lh$	$b \times lt \rightarrow le$	$lt \rightarrow lt$	$o': \rightarrow lt$
$lh \times le \rightarrow b$	$lh \times b \rightarrow lh$	$t \times lt \rightarrow lt$	$le \rightarrow le$	
	$b \times lh \rightarrow lh$	$lt \times t \rightarrow lt$		
	$lh \times lh \rightarrow lh$	$lh \times t \rightarrow lh$		
	$lt \times lt \rightarrow lt$	$t \times lh \rightarrow lh$		

There are two options how the loop construct can be used. The first option is using it for the topological arrangement of acton networks. For instance, a sequential acton network described by  $* > y > z > *'$  can be turned into a functional equivalent acton network  $* > o > (y > o')/z > *'$ , where the acton terms  $y$  and  $z$  are placed side-by-side. Fig. 4 shows the expressions and the structure of both networks. The dotted line in the loop structure indicates the hidden return of data.

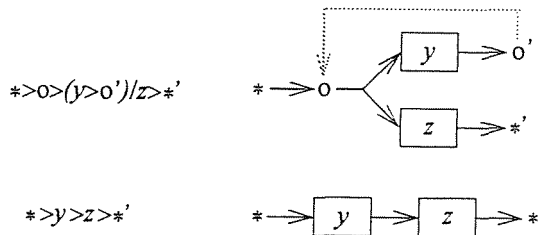


Fig. 4. Topological loop and functional equivalent sequence

More important, the loop construct can also be used to describe a feedback. A feedback means the joint processing of the input data and the returned data. It can be realized by an extra term immediately following the o-acton. The feedback network is depicted in Fig. 5. The extra term is designated by  $x$ .

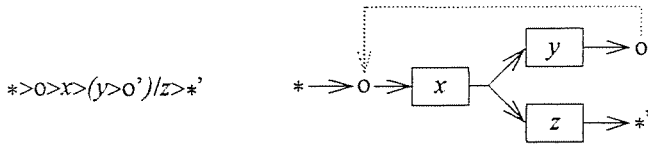


Fig. 5. Feedback loop

The feedback operation is a basic requirement for storage and repetition, which are the corner stones of computing. A feedback combines causally dependent data. Roughly, if the returned data are changed this leads to a repetition and if not this leads to a steady state, i.e., to storage. As an example we demonstrate the representation of a D-flipflop as shown in Fig. 6. The D-flipflop gets a bit  $D$  and a clock pulse  $CK$  for input and generates the complementary bits  $R$  and  $\bar{R}$  for output. In bent algebra a bit specifies a final bent value  $\{0,1\}$  and the leading edge of a clock pulse a signal  $\{\#,1\}$ .

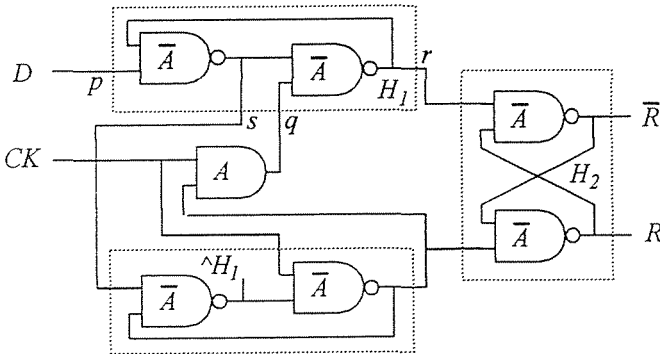


Fig. 6. D-flipflop

Table 9 summarizes the states of the flipflop. The flipflop is in a stable state as long as there is no clock pulse, i.e., as long as  $CK=\#$ , and it only changes its state if  $C=1$  and if  $D$  differs from  $R$ . The D-flipflop in Fig. 6 is realized by three functionally identical hold-circuits. They are encircled by dotted lines and designated by  $H_1$ ,  $\hat{H}_1$  and  $H_2$ . Each of them is built up by two NAND-gates. They are interconnected by an AND-gate. While  $H_2$  has a clearly different structure  $H_1$  and  $\hat{H}_1$  are just reflections of each other.

The NAND-gates of the hold-circuits form a noninverted feedback.

Table 9. States of the D-flipflop

D	CK	R	$\bar{R}$
0	#	R	$\bar{R}$
1	#	R	$\bar{R}$
0	1	0	1
1	1	1	0

The state of each hold-circuit therefore remains unchanged until a change is enforced by the input.

The hold-circuits can be transformed step-by-step into an acton term. Fig. 7 shows the graphical and formal representation of  $H_1$ . Two bents  $p$  and  $q$  are entered by  $\star_{pq}$ . Bent  $q$  is immediately sent to the second  $\bar{A}$  along the bridge  $\star'_q, \star_q$ . The two output bents  $r$  and  $s$  are returned by  $\star'_{rs}$ . It should be emphasized that each of the body actons has a unique topological position. The position can therefore be used as an implicit identifier. The first  $\bar{A}$  acts as the extra term which distinguishes a feedback loop from a topological loop.

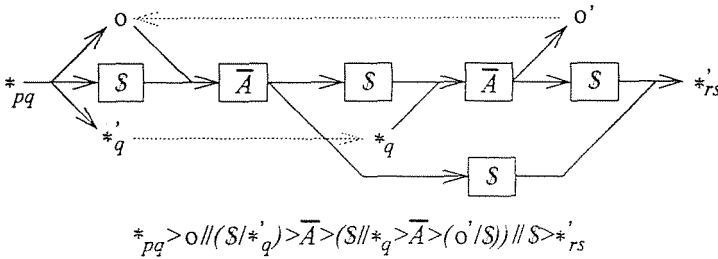


Fig. 7. Graphical and algebraical representation of  $H_1$

The acton terms of  $H_1$  and  $H_2$  can be set up in the same way. We leave this to the reader. The three abstractions are then used for the acton representation of the complete D-flipflop. Fig. 8 shows the result.

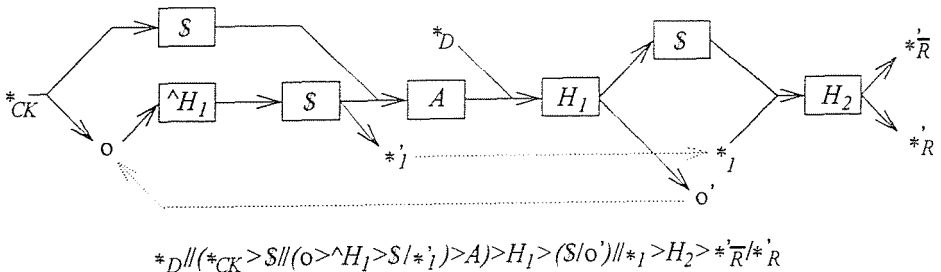


Fig. 8. Graphical and algebraical representation of the D-flipflop

## 6. Conclusion

This paper gives an introduction to the acton algebra and demonstrates its applicability to digital hardware circuits. A unique feature of the acton language is the explicit representation of storage. This makes it possible for the first time to completely define every digital circuit by a single formula. The formal representation can also be applied to every computing system. The topology can easily be turned into a metric if the actons are supplied with their planar dimensions. The space of the wiring is expressed by neutral actons. The acton algebra can therefore also be used for specifying the layout of digital circuits. A most important feature is that every layout can be realized on at most two planes. This is because the second plane is only used for point-to-point wiring. Every wire can thus be circumvented so that crossings are totally avoidable. The acton algebra can be abstracted to specific features of hardware or software systems. For instance, abstracting from the acton operations turns it into a layout language. Vice versa, abstracting from the topology turns it into a programming language, which can be related to conventional programming languages by means of conversion rules. Moreover there are term replacement rules which change the degree of parallelism of an acton network. Total parallelization or sequentialization can even be achieved automatically.

Meanwhile a parser has been built up for applications in C, C++, Delphi and Basic. It is freely available for scientific investigations.

## References

- [1] VON ISSENDORFF, H. (1995): Time and Logic: A Calculus of Binary Events, *Computing*, Vol. 54, No. 3, pp. 227-240.
- [2] VON ISSENDORFF, H. (1997): Some Generalizations of the Calculus of Binary Events, *Computing*, Vol. 58, No. 1, pp. 91-94.