# — vordenker-archive —

## Rudolf Kaehr

(1942-2016)

### Title

Playing Chiasms, Proemiality and Bifunctoriality

From Conceptual Graphs to Formulas and Procedures

### Archive-Number / Categories

3_21 / K05, K07, K08

### Publication Date

2012

### Keywords / Topics

- BIFUNCTORIALITY IN PROGRAMMING : Programming Proemiality in SML, Implementing transjunctions, Bifunctoriality with Combinatory Logic
- DISTRIBUTED PROCESSORS

### Disciplines

Cybernetics, Computer Sciences, Logic and Mathematics, Theory of Science

### Abstract

The recent paper about the *Tabularity of Polycontextural Logics* gets a complementary reflection of the mechanism of dissemination, i.e. distribution and mediation, by focusing on the conditions of mediation of distributed formal systems. In focus are some programming strategies, like bifunctoriality. Historical background information of implementations of polycontextural logics and attempts to formalize Gunther's proemial relationship are sketched. – See also: 3_19

### Citation Information / How to cite

**Rudolf Kaehr**: "Playing Chiasms, Proemiality and Bifunctoriality", www.vordenker.de (Sommer Edition, 2017)
J. Paul (Ed.), http://www.vordenker.de/rk/rk_Playing-Chiasms-and-Bifunctoriality_2012.pdf

### Categories of the RK-Archive

K01 Gotthard Günther Studies
K02 Scientific Essays
K03 Polycontexturality – Second-Order-Cybernetics
K04 Diamond Theory
K05 Interactivity
K06 Diamond Strategies
K07 Contextural Programming Paradigm
K08 Formal Systems in Polycontextural Constellations
K09 Morphogrammatics
K10 The Chinese Challenge or A Challenge for China
K11 Memristics Memristors Computation
K12 Cellular Automata
K13 RK and friends

# Playing Chiasms, Proemiality and Bifunctoriality

*From Conceptual Graphs to Formulas and Procedures*

**Rudolf Kaehr Dr.phil**[@]

 ISSN 2041-4358


## Abstract

The recent paper about the *Tabularity of Polycontextural Logics* gets a complementary reflection of the mechanism of dissemination, i.e. distribution and mediation, by focusing on the conditions of *mediation* of distributed formal systems. In focus are some programming strategies, like bifunctoriality. Historical background information of implementations of polycontextural logics and attempts to formalize Gunther' s proemial relationship are sketched.
(Work in progress, v. 0.2, 25.July 2012)


# 1. Bifunctoriality in Programming

## 1.1. Dortmunder Trials in the early 1990s

### 1.1.1. Programming Proemiality in SML

```
PROEMIAL
fun copy (ref(atom(x,ref Emark,ref wq))) =
        ref(atom(x,ref Emark,ref wq))
   |copy (ref(comb(x,ref Emark,ref wq))) =
        ref(comb(x,ref Emark,ref wq))
   |copy (ref(app((rator,rand),ref Emark,ref wq))) =
        ref(app((copy rator,copy rand),ref Emark,ref wq));

fun equal (ref(atom(x,_,_))) (ref(atom(y,_,_))) = (x=y)
   |equal (ref(comb(x,_,_))) (ref(comb(y,_,_))) = (x=y)
   |equal (ref(app((xrator,xrand),_,_)))
          (ref(app((yrator,yrand),_,_))) =
        (equal xrator yrator) andalso (equal xrand yrand)
   |equal__=false;

Discussion
PROEM:    |equal
          (ref(app((xrator,xrand),_,_)))
          (ref(app((yrator,yrand),_,_))) =

          (equal xrator yrator) andalso
          (equal xrand yrand)
```

**Diagram :**

$$\begin{array}{ccc} \text{xrator} & \longrightarrow & \text{xrand} \\ \updownarrow & & \updownarrow \\ \text{yrator} & \longrightarrow & \text{yrand} \end{array} : \left(\text{equal xrator yrator}\right) \text{ andalso } \left(\text{equal xrand yrand}\right).$$

In combinatory logic terms this construction of Mahler corresponds to the transition from the cominatory logical Sxyz = xz(yz) to the 'bifunctorial' construction of proemiality: PR S xy fg = fx(gy). Bifunctoriality was clearly realized in the description and the implementation of the model. But its mathmatical construct of bifunctoriality that would probabily have been of help for the academics to accept the ingeniosity of the model.

*"The type of proemial relationship which applies to the parallel evaluation of two combinator expressions* (f x) *and* (g y) *cannot be determined from the term structure; instead it can only be found out by looking at*

*the structure of pointer equality* $\equiv_z$ *and pointer difference* $(!\equiv_z)$ *of* f, x, g *and* y.*"* (Mahler, pLISP, 1992)

In ML terms: P (app1 rator1) (app2 rator2) --> (quote parEval(app1 rator1) parEval(app2 rator2)).

```
Translation
(a :- b): app (xrator,xrand)
(c :- d): app (yrator,yrand)
                              =
p a c   :  (equal xrator yrator)
p b d   :  (equal xrand yrand)
```

The proposed strategy shall be to start a programming language with bifunctoriality (interchangeability, metamorphosis, chiasm and proemiality) at the very beginning and to show that the common patterns of ordinary programming are just a case of a specific reduction of proemiality.

**(P (f x) (g y)) => (P parEval(f x) parEval(g y))**
*"The P-Combinator is not a classical Termtransformation. On the term level it does not change anything. But it produces two asynchronous Tasks (f x) and (g y) that are left on their own.*
*This Combinator is used to produce explicit parallelity. If both processes work with equal variables, the processes are linked like in (P (f x)(g x)).*
*With the P-Combinator it is possible to create interesting topologies like (P (f x) (x y)). In the first process x functions as the operand, in the second it is the operator. And this categorical exchange is performed simultaneously!*
*Such computational topologies are found in Polycontextural Logics (where they are formalized as "Proemial Relations"), meta-level architectures, computational reflection with causal connection and in simulations of self-referential, paradoxical and autopoietic systems."* (Mahler)

Thomas Mahler, pLISP: Parallel Functional Programming
http://www.thinkartlab.com/pkl/tm/plisp/ENGLISH.EPS
http://www.thinkartlab.com/pkl/tm/plisp/pr-java/index.htm

**General Scheme:** `A : B, C : D = A, C : B, D`

As a simple model I take the 2-domain **bifunctoriality** as it is implemented in Haskell for a bifunctor.

HASKELL: bimapS :: (a :- b) -> (c :- d) -> p a c :- p b d

Trivially, we get the fundamental transitivity of *uni*-versal programming (and math) as a reduction of bifunctoriality:

TRANS: (a :- b) -> (b :- c) -> (a :- c)

### 1.1.2. Implementing transjunctions

In a first step, the tableaux rules for transjunctions had been established and the distribution over different subsystems had been recognized and formalized. Nevertheless, the tableaux didn't offer much heuristic guidance to proceed properly a proof of a formula with transjunctions and negations included.

The developed tableaux proof system LOLA, by Bashford and Joemann, applied a rule that properly separates junctional from transjunctional parts of a proof. This was the crucial step for a reasonable heuristics for polycontextural tableaux proofs, supporting both, the machine and the human victim.

I wasn't aware that the term rule R1 *"(α simul δ) et (β simul γ) = (α et β) simul (γ et δ)*" is build on the base of the principle of bifunctoriality of the functions "simul" and "par" (Bob Coecke)

The similarity of the logical term rule to the programming strategy for an implementation of the proemial relationship by Mahler wasn't obvious either.

Nevertheless, the theorem prover LOLA implemented in SML/NJ is properly working albeit the matrix approach to the architecture of polycontextural logics, especially the difference of transpositional (for transjunctions) and replicational 'dimensions' (for implications), wasn't conceived in full. Unfortunately, the so called system-changes forced by the sub-system permutations of the negation operations which had been formalized properly years before (early 1970s) didn't got a direct and full implementation in LOLA-SML/NJ.

Today it has become more clear that a singular theorem prover that is simulating the different contextures generated by transjunctions and replications has to be replaced by a polycontextural theorem prover that is running simultaneously at each contexture autonomically and that are connected by the super-operators.

A *first* step to a *conservative* implementation of such a simultaneity might be achieved with the application of parallelism in programming, like the "*spawn*" concept for ML or HASKELL (GHC).
(cf. *A Note on the Tabularity Polycontextural Logics*
http://memristors.memristics.com/Notes%20on%20Polycontextural%20Logics

/Notes%20on%20Polycontextural%20Logics.html)

### 1.1.3. Interchange of sorts and universes

Years before the SML adventures, I sketched a proemial relationship between logical universes and sorts as an attempt to extend the conceptual framework of many-sorted predicate logics, a logical system of fundamental relevance for the theoretical study of programming (Joseph Goguen).

*"The modeling strategy for chiastic types in polycontextural situations is similar to the modeling strategy for chiasms in many-sorted logics. There, the chiasm is between uni-verse(s) and sorts of disseminated logics. Sorts in one logic can change to become uni-verses in other mediated logics. And in reverse, universes can change to sorts. Thus, chiasms are equally operating on many-sorted algebras as on typed calculi."* (Kaehr, 2006)
http://www.thinkartlab.com/pkl/lola/poly-Lambda_Calculus.pdf

Looking back, it turned out that the mathematics of the complex structures of bifunctors in 2-categories had been at hand with the "*Doppelkategorien*" of Hasse/Michler at least from 1966 on.

I never got a hint emphasizing this direction of thought from any of the academics who had been sceptical to the project of a formalization and implementation of polycontextural logics. The East-German mathematician Horst Reichel offered me some help but I didn't see my project mature enough to enter a collaboration with his highly complex category-theoretic formalizations. Jochen Pfalzgraf's successful formalizations of some aspects of polycontextural logics in the framework of category theory didn't consider the topics of interchangeability and bifunctoriality.

Quite obviously, the concept of bifunctoriality also wasn't implemented explicitly by any (functional) programming languages at that time (1990). This has changed, outside of academic interests, only recently.

**Chiasms of terms and types**

*"Thus, a type has two functionalities at once, a type as a type and a type as a term. Therefore, this double meaning has to be distributed over different localization of the complex constellation. Otherwise it simply would produce unnecessary conflictive overlapping. The* matrix *shows clearly the kind of distribution, the* diagram *is visualizing the process of the chiasm."* (2006)

Today I shall continue:
"and the *formula* 'finally' formalizes the bifunctorial interchangeability of types and terms in the framework of a polycontextural diamond category theory."

At the 'end' of the research program, suddenly stopped by the administration, it turned out that the main advances had been structured by the formula:

**Proemiator**

$$PR\,S\,x\,y\,f\,g = f\,x\,(g\,y)$$

$$PR\,S\,x\,y\,f\,g = \begin{pmatrix} f \longrightarrow x \\ \amalg \\ g \longrightarrow y \end{pmatrix} \text{ and more explictly by considering its super – additivity :}$$

$$PR\,S\,x\,y\,f\,g = \begin{pmatrix} \begin{pmatrix} S1 : f \longrightarrow x \\ \amalg_{1.2} \\ S2 : g \longrightarrow y \end{pmatrix} \\ \amalg_{1.2.3} \\ S3 : (f, g) \longrightarrow (x, y) \end{pmatrix}$$

Hence, from Gunther's relational formula of the proemial relaionship to my own diagrammatic formulations, to a proemial LISP and more, the journey got as a next step some academic disguise with the polycontextural subversion of the category-theoretic concept of bifunctoriality.

*Dissemination: Introducing the proemial relationship*
http://www.thinkartlab.com/pkl/media/DERRIDA/Proemial%20Relationship.html

**Early applications of the proemial relationship**

Bernhard J. Mitterauer, The proemial synapse. Consciousness generating glial-neuronal units
*"This type of relation may be an inevitable prerequisite for any theory of consciousness. Its formal description is as follows: Glia (G) dominate the neuronal components (N) by modifying them. Therefore, G play the role of a relator and N is the relatum. If this relationship changes inversely, N becomes the relator and G the relatum. Since the proemial relationship is cyclically organized, glial-neuronal synapses are capable of changing their relational positions in the sense of an iterative self-reflection mechanism. Hence, it seems to be legitimate to speak of proemial synapses."*
www.volitronics-institute.at/files/scanned31.pdf

## 1.2. Functional Programming and Bifunctoriality

### 1.2.1. A collection of attempts

Planet Haskell

```
#if 0
class BifunctorS (p :: Constraint -> Constraint -> Constraint) where
  bimapS :: (a :- b) -> (c :- d) -> p a c :- p b d
#endif
```

In an even more ideal world, it would be enriched using something like

```
#ifdef POLYMORPHIC_KINDS
class Category (k :: x -> x -> *) where
  id :: k a a
  (.) :: k b c -> k a b -> k a c
instance Category (:-) where
  id = refl
  (.) = trans
#endif
```

where x is a kind variable, then we could obtain a more baroque and admittedly far less thought-out bifunctor class like:

```
#if 0
class Bifunctor (p :: x -> y -> z) where
  type Left p :: x -> x -> *
  type Left p = (->)
  type Right p :: y -> y -> *
  type Right p = (->)
  type Cod p :: z -> z -> *
  type Cod p = (->)
  bimap :: Left p a b -> Right p c d -> Cod p (p a c) (p b d)
#end
```

**Definition 2 Bifunctor.**

$$
\begin{aligned}
&Bifunctor : Type \quad ::= \\
&\quad \Sigma FF_{obj} : \ Type \to Type \to Type, \\
&\quad\quad FF_{arr} : \ \Pi A, B, C, D \mid Type.\ (A \to B) \to (C \to D) \to FF_{obj}\, A\ C \to FF_{obj}\, B\ D \\
&\quad\quad\quad \Pi A, B : Type.\ FF_{arr}\, I_A\, I_B \doteq I_{FF_{obj}\, A\ B} \\
&\quad\quad\quad \wedge\ \Pi A, B, C, D, E, F : Type,\ h : A \to B,\ f : B \to C,\ k : D \to E,\ g : E \to F \\
&\quad\quad\quad\quad FF_{arr}(f \circ h)(g \circ k) \doteq (FF_{arr} f\ g) \circ (FF_{arr}\, h\ k)
\end{aligned}
$$

The shorthand $bifunctorial(F_{obj})(F_{arr})$ is used to denote the preservation properties of

*Example 3 Polymorphic Lists.*

$$FF^{list}_{obj}(A, X : Type) : \ Type \quad ::= \quad \mathbf{1} + (A \times X)$$

$$FF^{list}_{arr}(A, B, C, D \mid Type)(f : A \to B)(g : C \to D) : \ (FF^{list}_{obj}\ A\ C) \to (FF^{list}_{obj}\ B\ D) \quad ::= \quad I_1$$

## 1.3. Bifunctoriality with Combinatory Logic

### 1.3.1. Transitivity

Quite obviously, there is no bifunctoriality neither in Combinatory logic nor in Category theory on a basic and first-order level of the primary definitions of the formal languages (systems, calculi, scriptures).

Bifunctoriality enters category theory as a secondary construct on the base of a product. The same happens with combinatory logic and its application in functional programming. Bifunctoriality in functional programming is based on types and their multiplicity on a secure singular ground.

Transjunctions in the sense of polycontextural logic are breaking the linearity of transitivity.

Transjunctions are the primary “functions” or operations of polycontextural logics, both in respect of their conceptual and their numerical relevance.

**S functor**

```
fun S f g x = f x (g x)
(* val S = ('a -> ('b -> 'c)) -> (('a -> 'b) -> ('a -> 'c)) *)
```

**BIF: S1, S2, S3 :**

```
S1: S f g x = f x (g x)
              ||
S2: S f g x = f x (g x)
```

$$\text{BIF}\left(S1,\ S2\right) = \left(\begin{array}{c} S1 : f\ g\ x = f\ x\ \left(g\ x\right) \\ \amalg_{1.2} \\ S2 : f\ g\ x = f\ x\ \left(g\ x\right) \\ \amalg_{1.2.3} \\ S3 : f\ g\ x = f\ x\ \left(g\ x\right) \end{array}\right)$$

$$\left(f_1 \circ g_1\ x\right) \amalg_{1.2} \left(f_2 \circ g_2\ x\right) = \left(\left(f_1\ x\right) \amalg_{1.2} \left(f_2\ x\right)\right) \circ \left(\left(g_1\ x\right) \amalg_{1.2} \left(g_2\ x\right)\right)$$

**Interchangeability for $S^{(3,1)}$ : Sxyz = xz (yz)**

$$\left(\begin{array}{c} \left(f_1 \circ g_1\ x\right) \\ \amalg_{1.2} \\ \left(f_2 \circ g_2\ x\right) \\ \amalg_{1.2.3} \\ \left(f_3 \circ g_3\ x\right) \end{array}\right) = \left(\begin{array}{c} \left(f_1\ x\right) \\ \amalg_{1.2} \\ \left(f_2\ x\right) \\ \amalg_{1.2.3} \\ \left(f_3\ x\right) \end{array}\right) \circ \left(\begin{array}{c} \left(g_1\ x\right) \\ \amalg_{1.2} \\ \left(g_2\ x\right) \\ \amalg_{1.2.3} \\ \left(g_3\ x\right) \end{array}\right)$$

**Interchangeability for $S^{(3,1)}$ : Sxyz = xz (yz)**

$$\left(\begin{array}{c} \left(S_1 \circ f_1\ g_1\ x\right) \\ \amalg_{1.2} \\ \left(S_2 \circ f_2\ g_2\ x\right) \\ \amalg_{1.2.3} \\ \left(S_3 \circ f_3\ g_3\ x\right) \end{array}\right) = \left(\begin{array}{c} \left(f_1\ x\right) \\ \amalg_{1.2} \\ \left(f_2\ x\right) \\ \amalg_{1.2.3} \\ \left(f_3\ x\right) \end{array}\right) \circ \left(\begin{array}{c} \left(g_1\ x\right) \\ \amalg_{1.2} \\ \left(g_2\ x\right) \\ \amalg_{1.2.3} \\ \left(g_3\ x\right) \end{array}\right)$$

```
(* val S = ('a -> ('b -> 'c)) -> (('a -> 'b) -> ('a -> 'c)) *)
```

$$\text{val}^{(3)}\ S^{(3)} = \text{fn}^{(3)}\ :$$

$$\left(\begin{array}{c} \left('a \to \left('b \to 'c\right)\right)_1 \\ \amalg_{1.2} \\ \left('a \to \left('b \to 'c\right)\right)_2 \\ \amalg_{1.2.3} \\ \left('a \to \left('b \to 'c\right)\right)_3 \end{array}\right) = \left(\begin{array}{c} \left('a \to 'b\right)_1 \\ \amalg_{1.2} \\ \left('a \to 'b\right)_2 \\ \amalg_{1.2.3} \\ \left('a \to 'b\right)_3 \end{array}\right) \circ \left(\begin{array}{c} \left('a \to 'c\right)_1 \\ \amalg_{1.2} \\ \left('a \to 'c\right)_2 \\ \amalg_{1.2.3} \\ \left('a \to 'c\right)_3 \end{array}\right)$$

**Interchangeability for $K^{(3,1)}$ : Kxy = x**

$$\left(\begin{array}{c} \left(K_1 \circ xy_1\right) \\ \amalg_{1.2} \\ \left(K_2 \circ xy_2\right) \end{array}\right) = \left(\begin{array}{c} \left(K_1\right) \\ \amalg_{1.2} \\ \left(K_2\right) \end{array}\right) \circ \left(\begin{array}{c} \left(xy_1\right) \\ \amalg_{1.2} \\ \left(xy_2\right) \end{array}\right) = \left(\begin{array}{c} \left(x_1\right) \\ \amalg_{1.2} \\ \left(x_2\right) \end{array}\right)$$

### 1.3.2. Operator/operand chiasm

```
bi - fun II :
fun equal (fst 1 (a, _) = a 1) (fst 3 (a, _) = a 3) = (a 1 = a 3) : equal xrator, yrato
  | equal (snd 2 (_, b) = b 2) (snd 3 (_, b) = b 3) = (b 2 = b 3) : equal xrand , yrand
  | exch (fst 1 (a, _) = a 1) (snd 2 (_, b) = b 2) = (a 1 ≍ b 2) : exch xrator, yrand
  | exch (snd 1 (_, b) = b 1) (fst 2 (a, _) = a 2) = (b 1 ≍ a 2) : exch xrand, yrator
```

```
fst 1 ⟶ snd 1
  |       ↕    : (equal xrator yrator) andalso (equal xrand yrand)
  |     fst 2 ⟶ snd 2
  |               |
fst 3     ⟶     snd 3
```

```
fst 3 (a, _) = fst 1 (a, _) ⟶ snd 1 (_, b)
  ↓              ↕            X   ↕
snd 3 (_, b) = snd 2 (_, b) ⟵ fst 2 (a, _)
```

```
fun equal (ref (atom (x, _, _))) (ref (atom (y, _, _))) = (x = y)
  | equal (ref (comb (x, _, _))) (ref (comb (y, _, _))) = (x = y)
  | equal (ref (app ((xrator, xrand), _, _)))
          (ref (app ((yrator, yrand), _, _))) =
          (equal xrator yrator) andalso (equal xrand yrand)
  | equal__ = false;
```

**Diagram :**

```
xrator⟶xrand
  ↕        ↕  : (equal xrator yrator) andalso (equal xrand yrand)
yrator⟶yrand
```

### 1.3.3. Distributed buildins

```
(*  ***** @ is builtin *****                    *)
(* fun @  (nil,ys)  = ys                        *)
(*    | @ (x::xs,ys) = x :: @(xs,ys)            *)
(* val @ = fn : 'a list -> 'a list -> 'a list *)
(* infix 5 @                                   *)
```

**Multiprocessor system**

```
(*  ***** @ is builtin *****                    *)
(* fun @  (nil,ys)  = ys                        *)
(*    | @ (x::xs,ys) = x :: @(xs,ys)            *)
(* val @ = fn : 'a list -> 'a list -> 'a list *)
(* infix 5 @                                  *)
```

$$\text{val}^{(3,1)}\ @^{(3,1)} = \text{fn}^{(3,1)} :$$
$$'a\ \text{list} \to 'a\ \text{list} \to 'a\ \text{list}$$
$$'a\ \text{list} \to 'a\ \text{list} \to 'a\ \text{list}$$
$$'a\ \text{list} \to 'a\ \text{list} \to 'a\ \text{list}$$

$$\text{val}^{(3,1)}\ @^{(3,1)} = \text{fn}^{(3,1)} :$$

$$\begin{pmatrix} \begin{pmatrix} 'a\ \text{list} \to \left('a\ \text{list} \to 'a\ \text{list}\right)_1 \\ \amalg_{1.2} \\ 'a\ \text{list} \to \left('a\ \text{list} \to 'a\ \text{list}\right)_2 \end{pmatrix} \\ \amalg_{1.2.3} \\ \left('a\ \text{list} \to \left('a\ \text{list} \to 'a\ \text{list}\right)_3\right) \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} ('a\ \text{list})_1 \\ \amalg_{1.2} \\ ('a\ \text{list})_2 \end{pmatrix} \\ \amalg_{1.2.3} \\ ('a\ \text{list})_3 \end{pmatrix} \circ \begin{pmatrix} \begin{pmatrix} 'a\ \text{list} \to ' \\ \amalg_{1.2} \\ 'a\ \text{list} \to ' \end{pmatrix} \\ \amalg_{1.2} \\ 'a\ \text{list} \to ' \end{pmatrix}$$

| $\text{val}^{(3,1)}\ @^{(3,1)} = \text{fn}^{(3,1)}$ | locus1 | locus2 | locus3 |
|---|---|---|---|
| val1 | fun @ (nil, ys) = ys<br>\| @ (x::xs, ys) =<br>x :: @(xs, ys) | - | - |
| val2 | - | fun @ (nil, ys) = ys<br>\| @ (x::xs, ys) =<br>x :: @(xs, ys) | - |
| val3 | - | - | fun @ (nil, ys) = ys<br>\| @ (x::xs, ys) =<br>x :: @(xs, ys) |

**Dissemination of the builtin** $@^{(3,1)}$

```
fun @1
  (nil, ys) = ys
  | @(x::xs, ys) = x :: @(xs, ys)
```

```
fun @2
  (nil, ys) = ys
  | @(x::xs, ys) = x :: @(xs, ys)
```

```
fun @3
        (nil, ys) = ys
          | @ (x::xs, ys) = x :: @(xs, ys)
```

**Chiasm of @**

$$\begin{array}{ccc} @^3 = @^1 & \longrightarrow & @^1 \\ \downarrow \quad \updownarrow & X & \updownarrow \\ @^3 = @^2 & \longleftarrow & @^2 \end{array}$$

**Diamond of @**

$$\begin{array}{ccccc} @^3 = @^1 & \longrightarrow & @^1 = @^4 \\ \downarrow \quad \updownarrow & X & \updownarrow \quad \uparrow \\ @^3 = @^2 & \longleftarrow & @^2 = @^4 \end{array}$$

**Interchangeability for Diss - CL** $^{(3,3)}$

$$\left(\begin{array}{c} \left(\begin{array}{c} (@_1 \circ @_1) \\ \amalg_{1.2} \\ (@_2 \circ @_2) \end{array}\right) \\ \amalg_{1.2.3} \\ (@_3 \circ @_3) \end{array}\right) = \left(\begin{array}{c} \left(\begin{array}{c} (@_1) \\ \amalg_{1.2} \\ (@_2) \end{array}\right) \\ \amalg_{1.2.3} \\ (@_3) \end{array}\right) \circ \left(\begin{array}{c} \left(\begin{array}{c} (@_1) \\ \amalg_{1.2} \\ (@_2) \end{array}\right) \\ \amalg_{1.2.3} \\ (@_3) \end{array}\right)$$

## 2. Distributed processors

$@_{i.j}$ : Processor active at $(i, j)$

$\varphi_{i.j}$ : Processor inactive at $(i, j)$

$\text{OPER}^{(3,1)}_{\text{junct}(1,2,3)}$ :

| PM | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|
| $M_1$ | $\varphi_{1.1}$ | $\varphi_{2.1}$ | $\varphi_{3.1}$ |
| $M_2$ | $\varphi_{1.2}$ | $\varphi_{2.2}$ | $\varphi_{3.2}$ |
| $M_3$ | $\varphi_{1.3}$ | $\varphi_{2.3}$ | $\varphi_{3.3}$ |

$\longrightarrow$

| PM | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|
| $M_1$ | $@_{1.1}$ | $\varphi_{2.1}$ | $\varphi_{3.1}$ |
| $M_2$ | $\varphi_{1.2}$ | $@_{2.2}$ | $\varphi_{3.2}$ |
| $M_3$ | $\varphi_{1.3}$ | $\varphi_{2.3}$ | $@_{3.3}$ |

$\text{OPER}^{(3,1)}_{\substack{\text{trans}(1)\\ \text{junct}(2,3)}}$ :

| PM | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|
| $M_1$ | $\varphi_{1.1}$ | $\varphi_{2.1}$ | $\varphi_{3.1}$ |
| $M_2$ | $\varphi_{1.2}$ | $\varphi_{2.2}$ | $\varphi_{3.2}$ |
| $M_3$ | $\varphi_{1.3}$ | $\varphi_{2.3}$ | $\varphi_{3.3}$ |

$\longrightarrow$

| PM | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|
| $M_1$ | $@_{1.1}$ | $@_{2.1}$ | $@_{3.1}$ |
| $M_2$ | $\varphi_{1.2}$ | $@_{2.2}$ | $\varphi_{3.2}$ |
| $M_3$ | $\varphi_{1.3}$ | $\varphi_{2.3}$ | $@_{3.3}$ |

$\text{OPER}^{(3,1)}_{[<>\supset\vee]}$ :

| PM | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|
| $M_1$ | $\varphi_{1.1}$ | $\varphi_{2.1}$ | $\varphi_{3.1}$ |
| $M_2$ | $\varphi_{1.2}$ | $\varphi_{2.2}$ | $\varphi_{3.2}$ |
| $M_3$ | $\varphi_{1.3}$ | $\varphi_{2.3}$ | $\varphi_{3.3}$ |

$\longrightarrow$

| PM | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|
| $M_1$ | $@_{1.1}$ | $@_{2.1}$ | $@_{3.1}$ |
| $M_2$ | $\varphi_{1.2}$ | $@_{2.2}$ | $\varphi_{3.2}$ |
| $M_3$ | $@_{1.3}$ | $\varphi_{2.3}$ | $\varphi_{3.3}$ |

$\text{OPER}_{[<>\supset\vee]} = \left[ <> : 1.1,\ 2.1,\ 3.1;\ \supset : 2.2;\ \vee : 1.3 \right]$

The super-operators SOPS are the programming strategies, the distributed processor on the kenomic matrix are the programmed machines to be programmed firstly, contexturally, i.e. depending on the loci/places of the processors and secondly, by the types of operations involved. The involved operations then are the localized junctional, transpositional, replicational and reflectional logico-arithmetic operations.

The super-operators are activating or deactivating the disseminated processors according to their operational structure.

Because of the exchange mechanism of operator and operand on the level of the hardware processors, a feature that is not realizable within the possibilities of classical processors and architectures, it is proposed that by taking into account the new possibilities of memristive approaches to realize such mechanisms of interchangeability with a successive application of devices based on memristors and memristive systems, such limits of traditional computation might be, in principle, overcome.

It is understood that the main novelty of *memristors* is not in the domain of quantities, like speed and storage, but in the functionality of the exchangeability of “processor” and “memory” functions of the “same” computing device at the “same” place.

Hence, the dissemination, defined by distrubution and mediation, of the activity, i.e. inter- and trans-activity of the processors of the grid, is managed by the interchangeability of the main features of computability, computation and memorization, and realized by the application of memristors and their distribution in crossbar systems.

Logical and symbolic processes are distributed over the kenomic matrix. But this distribution is not a static architectonic fact but is involved in the process of interactions between different processors. In this sense, the relization of a transpositional distribution is seen as an interaction between different processors. The ‘main’ processor of a transjunctional operation is ‘sending’ an activation messige to the transpositioned processor to realize the transjunction addressed by the main processor. The main processors in the design are the ‘diagonal’ processors of the grid. This is not a restriction to a mxm-matrix. Other configurations are easily produced, and each processor might play the role of a ‘main’ processor.

| **oto** | locus1 | locus2 | locus3 |
|---|---|---|---|
| va1 | [t1 .1 : (t1 : X \|\| t1 : Y)]<br>[f1 .1 : (f1 : X && f1 : Y )] | [t2 .1 : (f2 : X && t2 : Y) \|\|<br>(t2 : X && f2 : Y)]<br>[f2 .1 : (t2 : X && t2 : Y)] | [t3 .1 : (f2 : X && t2 : Y)<br>(t2 : X && f2 : Y)<br>[f3 .1 : (f2 : X && f2 : Y) |
| val2 | - | [t2 .2 : (t2 : X && t2 : Y)]<br>[f2 .2 : (f2 : X && f2 : Y)] | - |
| val3 | - | - | [t3 .3 : (t3 : X \|\| t3 : Y<br>[f3 .3 : (f3 : X && f3 : Y |

$$\text{OPER}^{(3,1)}_{[\text{oto}]} = \begin{pmatrix} \varphi_{1.1} \longrightarrow @_{1.1} \\ \varphi_{2.2} \longrightarrow @_{2.2} \mid @_{2.1} \mid @_{3.1} \\ \varphi_{3.3} \longrightarrow @_{3.3} \end{pmatrix}$$

| **oto : true1, 3 ?** | locus1 | locus2 | locus3 |
|---|---|---|---|
| va1 | t1 .1 :<br>(t1 : X \|\| t1 : Y) | t2 .1 :<br>(f2 : X && t2 : Y) \|\|<br>(t2 : X && f2 : Y) | t3 .1 :<br>(f2 : X && t2 : Y) \|\|<br>(t2 : X && f2 : Y) |
| val2 | - | $\nwarrow \nearrow$<br>$\varphi_{2.2}$ | - |
| val3 | - | - | t3 .3 :<br>(t3 : X \|\| t3 : Y) |

**Question :** true1 .3 (oto) ?

t1 .1 | t2 .1 | t3 .1

t3 .3

⇓

**dec1 :** t1 .1 (frml) ? | t2 .1 (frml) ? | t3 .1 (frml) ?

tree tree tree

**dec2 :** t3 .3 (frml) ?

tree

⇓

**repeat** val(frml(tree)) ⟹ **answer !**

| PM | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|
| $M_1$ | $\varphi_{1.1}$ | $\varphi_{2.1}$ | $\varphi_{3.1}$ |
| $M_2$ | $\varphi_{1.2}$ | $\varphi_{2.2}$ | $\varphi_{3.2}$ |
| $M_3$ | $\varphi_{1.3}$ | $\varphi_{2.3}$ | $\varphi_{3.3}$ |

⟶

| PM | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|
| $M_1$ | $@_{1.1}$ | $@_{2.1}$ | $@_{3.1}$ |
| $M_2$ | $\varphi_{1.2}$ | $@_{2.2}$ | $\varphi_{3.2}$ |
| $M_3$ | $@_{1.3}$ | $\varphi_{2.3}$ | $\varphi_{3.3}$ |

$$\text{OPER}^{(3,1)}_{[<>\supset\vee]} = \begin{pmatrix} \varphi_{1.1} \longrightarrow @_{1.1} \mid @_{2.1} \mid @_{3.1} \\ \varphi_{2.2} \longrightarrow @_{2.2} \\ \varphi_{3.3} \longrightarrow @_{1.3} \end{pmatrix}$$

**Multi – pr ocessor model for f1H5 :**

$$\mathbf{H5} = (X \text{ laa } Y) \text{ iij } (X \text{ laa } Y)$$

Multi-Processor-System for matrix-distribution of tableaux_forests = (intra-process:{append, remove, leave}, inter-process: {send, receive}).

| ***input : f1H5*** <br> ***question :*** <br> ***f1H5 = taut?*** | locus1 | | locus2 | locus3 |
|---|---|---|---|---|
| process1 | @1.1 : <br> ***input*** : tree(1.1) <br> ***process***(tree(1.1)) : <br> ID(1.1) = {append, remove} <br> stop(1.1) = **output**(1.1) | | @1.2 : <br> ***receive*** BIF from @2.1 : <br> ***input*** : tree(1.2) : <br> ***process***(tree(1.2)) : <br> ID(1.2) = {append, remove} <br> stop(1.2) = **output**(1.2) | [inactive] |
| process2 | @2.1 : <br> ***input*** : tree(2.1) <br> ***process***(tree(2.1)) : <br> BIF(2.1) = [ID(2.1) <br> ***process***(ID(2.1)) : <br> {append, remove} <br> stop(2.1) = **output**(2.1) | – <br> – <br> BIF(1.2)] <br> ***send*** BIF to @1.2 : <br> leave(2.1) : <br> ***process***(tree(1.2)) | [inactive] | [inactive] |
| process3 | [inactive] | | [inactive] | [inactive] |
| ***output for*** **f1H5 : taut for Sys1 .1 , Sys2 .1 , Sys1 .2 = taut f1H5** | **taut** for Sys1 .1 and Sys2 .1 | | **taut** for Sys1 .2 | – |

**The matrix development as a concurrent procedure with spawn**

| *spawn* expr0 = **f1H5** | | |
|---|---|---|
| expr 1.1 : *spawn* expr1 .1 | expr2 .1 : *spawn* expr2 .1 | |
| expr1 .1 | expr2 .1 | expr1 .2 |
| stop1 .1 | stop2 .1 | stop1 .2 |

XXX