— vordenker-archive —

Rudolf Kaehr
(1942-2016)

Title

Memristics LISPs

How to program memristive systems?

Archive-Number / Categories

3_04 / K11, K07

Publication Date

2011

Keywords

TOPICS: McCarthy's recursive LISP – F. L. Bauer's formal characteriszation of LISP – Recursive kenomic LISP – Special features of kenomic LISP – Examples of kenomic LISP – Bifunctoriality of CONS, CAR and CDR – Memristive self-referentiality – Ambiguity, double-meaning and morphograms –

Disciplines

Cybernetics, Computer Sciences, Artificial Intelligence and Robotics, Systems Architecture and Theory and Algorithms, Memristive Systems

Abstract

This draft is a first sketch of the basic definitions of a memristive polyLISP. The memristivity of the polyLISP refers to the fundamental retrogradness of its recursivity which is not just an iterative repetition of the "function itself" but a self-definition of the range and character of the operation involved. The hint to polycontexturality with the prefix "poly" hints to the fact that iterability for retrograde functions is "polycontextural", i.e. opening up different contextural domains instead of remaining immanently inside a domain closed by its closure conditions like it holds for classical recursivity.

Citation Information / How to cite

Rudolf Kaehr: "Memristics LISPs", www.vordenker.de (Sommer Edition, 2017) J. Paul (Ed.),
URL: http://www.vordenker.de/rk/rk_Memristics-LISPs_2011.pdf

Categories of the RK-Archive

Memristics' LISPs

How to program memristive systems?

Rudolf Kaehr Dr.phil
 ISSN 2041-4358

Abstract

A first sketch of the basic definitions of a memristive polyLISP. The memristivity of the polyLISP refers to the fundamental retrogradness of its recursivity which is not just an iterative repetition of the "function itself" but a self-definition of the range and character of the operation involved. The hint to polycontexturality with the prefix "poly" hints to the fact that iterability for retrograde functions is "polycontextural", i.e. opening up different contextural domains instead of remaining immanently inside a domain closed by its closure conditions like it holds for classical recursivity.

ULTRA DRAFT

1. Styles of thematizations

1.1. Four formal principle of memristics

Question: How to program memristive systems?

Four fundamental memristic strategies
The new approach of kenomic formal systems, calculi, machines, programming paradigms, is taking two very simple constructions, principles, strategies into account:

First: Retrograde recursivity
When ever something is repeated it has to be decided as what it shall be repeated. The scope or range of repetition of an element is fully depending on the history of former repetitions.

Second: Enaction
When ever something is annihilated, eliminated, erased, cancelled, its annihilation has to be registered, to be remembered, memorized at a location of registration. The registration of an annihilation might be involved as a proper object into further iter/alterations of repeatability and enaction.

The *first* principle is realized formally by the strategy of *retrograde recursivity* as it was introduced for keno- and morphogrammatics in the early 70s as a polycontextural answer to the topics of self-referentiality in Second-Order Cybernetics.

The *second* principle is realized by the strategy of *enaction*. Enacation seems to be a recent

invention/insight, well motivated by a polycontextural interpretation of the concepts of memristive systems (Chua, Williams) and a subversive move in the understanding of George Spencer Brown's *Law of Crossing* beyond semiotics and logic sketched at first around 2010.

Therefore, iterability in general is not only happening as retrograde iter/alteration but as an inscription of memristive enaction too.

Also the operation of enaction is new it nevertheless seems to have enough cultural background to be accepted as "natural".

There are other important new principles which are omitted at this place, like the principle of *localization* of systems and operations and the principle of *diamond* environments.

Third: Bifunctoriality

LISP operations are concatenative. This holds for all LISP dialects. Therefore, all combinations of Lisp operators are faithfull to the Lisp operational *closure*.

The combination (composition) of two Lisp operations is a Lisp operation.

$\forall\ op_i \in$ Lisp: op(op(...(op)) $\in$ Lisp

Hence, $op_1, op_2 \in$ Lisp $\Rightarrow op_2(op_1) \in$ Lisp.

Structural law of associativity

$$op_1 \circ \left(op_2 \circ op_3\right) = \left(op_1 \circ op_2\right) \circ op_3\right)$$

The new feature of kenoLisp is *bifunctoriality* between two operations and two different Lisp systems.

$$op_{1,2} \in \text{Lisp}_1,\ op_{3,4} \in \text{Lisp}_2 \Rightarrow \left(\left(op_1 \circ op_2\right) \text{II} \left(op_3 \circ op_4\right)\right) \in \text{Lisp}^{(2)}$$

law $\text{CONS}^{(4)}$

$$\text{CONS}\left(l_1, l_2, l_3, l_4\right)$$

$$\begin{bmatrix} op_1 & op_3 \\ op_2 & op_4 \end{bmatrix}:$$

$$\begin{pmatrix} op_1 \\ \text{II} \\ op_3 \end{pmatrix} * \begin{pmatrix} op_2 \\ \text{II} \\ op_4 \end{pmatrix} = \begin{pmatrix} \left(op_1 * op_2\right) \\ \text{II} \\ \left(op_3 * op_4\right) \end{pmatrix} .$$

S tructural law of bifunctoriality

$$\left(\mathrm{op}_1 * \ \mathrm{op}_2\right) \amalg \left(\mathrm{op}_3 * \ \mathrm{op}_4\right) = \left(\mathrm{op}_1 \amalg \ \mathrm{op}_3\right) * \left(\mathrm{op}_2 \amalg \ \mathrm{op}_4\right)$$

Fourth: Dissemination of Lisp over the kenomic matrix

Matrix model

[bif, id, id]	O_1	O_2	O_3
M_1	$\mathrm{LISP}_{1.1}$	$\mathrm{LISP}_{2.1}$	$\mathrm{LISP}_{3.1}$
M_2	–	$\mathrm{LISP}_{2.2}$	–
M_3	–	–	$\mathrm{LISP}_{3.3}$

Dissemination of bifunctoriality

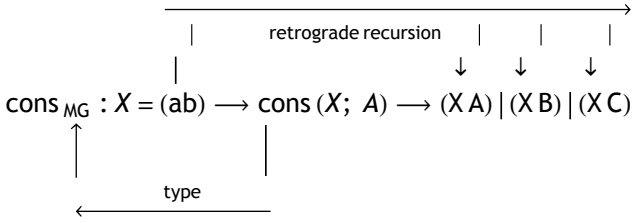
Transpositional composition

$$\begin{pmatrix} \mathrm{CAR}_1 \\ \amalg_{1.2} \\ \mathrm{CAR}_2 \diamond_{2.1} \mathrm{CAR}_1 \\ \amalg_{2.3} \\ \mathrm{CAR}_3 \diamond_{3.1} \mathrm{CAR}_1 \end{pmatrix} \begin{bmatrix} \circ_{1.1} \\ \\ \circ_{2.1} \circ_{2.2} \\ \\ \circ_{3.1} \circ_{3.3} \end{bmatrix} \begin{pmatrix} \mathrm{CDR}_1 \\ \amalg_{1.2} \\ \mathrm{CDR}_2 \diamond_{2.1} \mathrm{CDR}_1 \\ \amalg_{2.3} \\ \mathrm{CDR}_3 \diamond_{3.1} \mathrm{CDR}_1 \end{pmatrix} =$$

$$\begin{pmatrix} (\mathrm{CAR}_1 \circ_{1.1} \ \mathrm{CDR}_1) \\ \amalg_{1.2} \\ (\mathrm{CAR}_2 \circ_{2.2} \ \mathrm{CDR}_2) \diamond_{2.1} (\mathrm{CAR}_1 \circ_{2.1} \ \mathrm{CDR}_1) \\ \amalg_{2.3} \\ (\mathrm{CAR}_3 \circ_{3.3} \ \mathrm{CDR}_3) \diamond_{3.1} (\mathrm{CAR}_1 \circ_{3.1} \ \mathrm{CDR}_1) \end{pmatrix}$$

1.2. Styles of thematizations

Example for retrograde kenomic CONS



The kenomic operator “CONS” as an operator for construction is in general not predefined like in classical Lisp to act on atomic terms which are ruled by the principles of identity and equality. The new approach to a morpho- and kenogrammatic or thematic Lisp is enabling “CONS” to chose, in the process of application, the *mode* of the further steps of the application by *elect-*

ing the data paradigm to be involved. The graphematic possibilities for the data paradigm at hand for now are:
1. the mode of *semiotic* identity with recursivity,
2. the mode of *contextural* comlexity with proemial recursivity,
3. the mode of *kenogrammatic* similarity with retrograde recursivity,
4. the *indicational* mode of "topology-free constellations of signs" with recursive enaction, and
5. the mode of *monomorphic* bisimilarity of morphogrammatics with bisimulation and metamorphosis.

Other modes are possible as further realizations of graphematic styles of inscription. Known examples are the deutero- and proto-structure of kenogrammatics. On the other side, there are fuzzy-logical concepts for the semiotics mode of thematization; and others.

$$\text{THEMATIZATION} : a, a, b, c \longrightarrow$$
$$\begin{cases} \text{SET}(a, a, b, c) \longrightarrow \{a,\ b,\ c\} \\ \text{LIST}(a, a, b, c) \longrightarrow (a\,.\,a\,.\,b\,.\,c) \\ \text{CONTEXTURE}(a, a, b, c) \longrightarrow (((a \square a) \amalg d)(c \amalg e) \amalg f) \\ \text{KENOS}(a, a, b, c) \longrightarrow [xyz] \\ \text{INDIC}(a, a, b, c) \longrightarrow \langle a\,b\,c \rangle \\ \text{MORPHIC}(a, a, b, c) \longrightarrow [[xx][y][z]] \end{cases}$$

Domains of application
The semiotic or *symbolic* mode of thematization is ideal for atomistic binary physical systems as they occur in or as digital computers.

The *contextural* or interactional mode of thematization is ideal for ambigous complex physical systems as they occur in distributed and interacting digital computer and organic systems.

The *kenogrammatical* mode of thematization is ideal for pre-semiotic complex behavioural systems as they occur in memristive physical and cognitive/volitive systems.

The *indicational* mode of thematization is ideal for singular decision systems as they occur in simple actional systems where identity of the agents is relevant but not the order of their appearance.

The *monomorphic* mode of thematization is ideal for metamorphic systems as they occur in complex memristive actional systems.

1.2.1. Graphematic systems

Semiotics $a=a,\ a\neq b$, with $a(bc) = (ab)c$
(A)"If the two given tokens of strings have different lengths, then they are different. If they have equal lengths, then go to (B)."
(B) "For each position i from 1 to the common length, check whether the atom at the i-th position of x equals the atom at the i-th position of y. If this is true for all positions i, then the given tokens are equal, otherwise they are different."

IF length(X) = length(Y) and $\forall i\ x_i \in X,\ y_i \in Y: x_i \equiv y_i$ THEN X = Y.

Indicational a=a, a≠b, ab = ba, aa = a
(B') "Check whether each atom appears equally often in both string-tokens. If this is the case, then they are equal, otherwise they are different."

$\forall i,j: x_i \in X, y_j \in Y: \{x_i\} = \{y_j\}$ THEN X =Y

Kenogrammatics a=b, (aa) ≠ (ab), (aa) ≠ (aaa)
(B") "For each pair i,k, i<k, of positions, check whether within x there is equality between position i and k, and check whether wihin y there is equality between position i and k. If within both x and y there is equality, or if within both x and y there is inequality, then state equality for this pair of positions, otherwise state inequality for this pair of positions. If for each pair of positions there is equality, then x and y are equal. Otherwise they are not."

Deutero-Structure
(B"') "Take an atom a from x, find out the number k of atoms in x equal to a, and check whether in y there is an atom which occurs exactly k times. If not, then x and y are unequal. If yes, then remove the atoms just considered from x and y. If nothing is left, x and y are equal. Otherwise apply B"' to the remaining string-tokens."

Comparison
"The former are invariant w.r.t. permutations of the index set {1,...,n}, while the latter are invariant w.r.t. permutations of the alphabet A."
SEMIOTIC ABSTRACTIONS IN THE THEORIES OF GOTTHARD GÜNTHER AND GEORGE SPENCER BROWN By Rudolf Matzka, Munich, May 1993

Monomorphics a=b, (aa) ≠ (aaa)**, (**aba) = (abba)

The next feature of the operator "CONS" (also "append") is defined by the retrograde recursivity of iterability, i.e. the modes of 'concatenation' and 'succession'.

Depending on the hermeneutical process of thematization, the operation "CON" might chose its mode of realization. It might switch between different styles, or it might stay stable for a chosen possibility of thematization.

In a polycontextural situation it might be preferable to use simultaneously differend modes of thematizations.

Hence, before any decision for a certain programming paradigm and then for the main topics, a decision, i.e. an *election* of the mode of 'production' has to be installed.
An explication of the difference of *selection* (for Lambda Calculus) and *election* (for Contextural Programming) might be found at:
http://works.bepress.com/thinkartlab/20/

2. Recursive symbolic Lisp

2.1. McCarthy's recursive LISP

Following the clear exposition of the definition of recursive LISP as McCarthy has outlined in his inaugurating paper *"Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I, April 1960*" I will try to sketch a deconstructive approach to the idea and definitions of a recursive kenomic LISP along this historical guidlines.

McCarthy:

a. A Class of **Symbolic Expressions**. We shall now define the S-expressions (stands for symbolic). They are formed by using the special characters

.

)

(

and an infinite set of distinguishable atomic symbols.

c. The Elementary **S-functions** and **Predicates**. We introduce the following functions and predicates:

1. **atom**. atom[x] has the value of T or F according to whether x is an atomic symbol. Thus
atom [X] = T
atom [(X · A)] = F

2. **eq**. eq [x;y] is defined if and only if both x and y are atomic.
eq [x; y]= T if x and y are the same symbol, and eq [x; y] = F otherwise.

Thus eq [X; X] = T
eq [X; A] = F
eq [X; (X · A)] is undefined.

3. **car**. car[x] is defined if and only if x is not atomic. car [(e1 · e2)] = e1.
Thus car [X] is undefined.
car [(X · A)] = X
car [((X · A) · Y)] = (X · A)

4. **cdr**. cdr [x] is also defined when x is not atomic. We have cdr [(e1 · e2)] = e2. Thus cdr [X] is undefined.
cdr [(X · A)] = A
cdr [((X · A) · Y)] = Y

5. **cons**. cons [x; y] is defined for any x and y. We have
cons [e1; e2] = (e1 · e2). Thus
cons [X; A] = (X A)
cons [(X · A); Y] = ((X · A)Y)

car, cdr, and cons are easily seen to satisfy the relations
car [cons [x; y]] = x
cdr [cons [x; y]] = y
cons [car [x]; cdr [x]] = x, provided that x is not atomic.

2. **subst** [x; y; z]. This function gives the result of substituting the S-expression x for all occurrences of the atomic symbol y in the S-expression z. It is defined by

subst [x; y; z] = [atom [z] ⟶ [eq [z; y]⟶ x; T ⟶ z];
T ⟶ cons [subst [x; y; car [z]]; subst [x; y; cdr [z]]]]

As an example, we have

subst[(X · A);B; ((A · B) · C)] = ((A · (X · A)) · C)

3. **equal** [x; y]. This is a predicate that has the value T if x and y are the same S-expression, and has the value F otherwise. We have

equal [x; y] = [atom [x] ∧ atom [y] ∧ eq [x; y]]
∨[¬ atom [x] ∧¬ atom [y] ∧ equal [car [x]; car [y]] ∧ equal [cdr [x]; cdr [y]]].

The following functions are useful when S-expressions are regarded as lists.

1. **append** [x;y].
append [x; y] = [null[x] ⟶ y; T ⟶ cons [car [x]; append [cdr [x]; y]]]

An example is
append [(A, B); (C, D, E)] = (A, B, C, D, E)

2. **among** [x;y]. This predicate is true if the S-expression x occurs among the elements of the list y.
We have

among[x; y] = ¬ null[y] ∧ [equal[x; car[y]] ∨ among[x; cdr[y]]].

3. **pair** [x;y]. This function gives the list of pairs of corresponding elements of the lists x and y. We have

pair[x; y] = [null[x] ∧ null[y] ⟶ NIL; ¬atom[x] ∧¬atom[y] ⟶ cons[list[car[x]; car[y]]; pair[cdr[x]; cdr[y]]]].

An example is
pair[(A,B,C); (X, (Y, Z), U)] = ((A,X), (B, (Y, Z)), (C, U)).

John McCarthy, Recursive Functions of Symbolic Expressionsand Their Computation by Machine, Part I, MIT, April 1960
http://www-formal.stanford.edu/jmc/recursive.pdf

2.1.1. F. L. Bauer's formal characteriszation of LISP

$$
\begin{aligned}
&\textbf{type LISP} \equiv \\
&\left(\textbf{mode atom}\right) \textbf{lisp},\ \text{car},\ \ \text{cdr},\ \text{cons},\ \text{mklisp},\ \text{mkatom},\ \text{isatom}: \\
&\qquad \textbf{mode lisp}, \\
&\qquad \textbf{funct}\left(\left\{\textbf{lisp}\ \text{l}: \neg\ \text{isatom}\left(\text{l}\right)\right\}\right) \textbf{lisp}\ \text{car}, \\
&\qquad \textbf{funct}\left(\left\{\textbf{lisp}\ \text{l}: \neg\ \text{isatom}\left(\text{l}\right)\right\}\right) \textbf{lisp}\ \text{cdr}, \\
&\qquad \textbf{funct}\left(\textbf{lisp}, \textbf{lisp}\right) \textbf{lisp}\ \text{cons}, \\
&\qquad \textbf{funct}\left(\textbf{atom}\right) \textbf{lisp}\ \text{mklisp}, \\
&\qquad \textbf{funct}\left(\left\{\textbf{lisp}\ \text{l}: \text{isatom}\left(\text{l}\right)\right\}\right) \textbf{atom}\ \text{mkatom}, \\
&\qquad \textbf{funct}\left(\textbf{lisp}\right) \textbf{bool}\ \text{isatom}, \\
&\qquad \textbf{law}\ \text{CAR}: \quad \text{car}\left(\text{cons}\left(\text{k},\ \text{l}\right)\right) = \text{k}, \\
&\qquad \textbf{law}\ \text{CDR}: \quad \text{cdr}\left(\text{cons}\left(\text{k},\ \text{l}\right)\right) = \text{l}, \\
&\qquad \textbf{law}\ \text{CONS}: \quad \neg\ \text{isatom}\left(\text{l}\right) \rightarrow \text{cons}\left(\text{car}\left(\text{l}\right),\ \text{cdr}\left(\text{l}\right)\right) = \text{l}
\end{aligned}
$$

F. L. Bauer, H. Wössner, H. Wassner,
Algorithmische Sprache und Programmentwicklung , 1984

2.1.2. First steps to an algebraic characterization kenomic LISP

$$\textbf{mode keno} - \textbf{lisp}^{(m,n)}\ \chi \mid \textbf{matrix}(m,\ n) \equiv$$
$$\chi^{(m,n)} \mid$$
$$\left(\textbf{lisp}^{(m,n)}\ \chi\ \text{car},\ \textbf{lisp}^{(m,n)}\ \chi\ \text{cdr},\ \textbf{lisp}^{(m,n)}\ \chi\ \text{repl},\ \textbf{lisp}^{(m,n)}\ \chi\ \text{transp}\right)$$

$$\textbf{type kenoLISP}^{(m,n)} \equiv$$
$$\left(\textbf{mode keno},\ \textbf{matrix}(m,\ n)\right)\textbf{lisp}^{(m,n)},$$
$$\text{car, cdr, cons, repl, transp, mklisp, mkkenom, iskenom}:$$
$$\textbf{mode kenolisp},$$
$$\textbf{funct}\left(\left\{\textbf{lisp}\ l:\ \neg\ \text{iskenom}(l)\right\}\right)\textbf{lisp}\ \text{car},$$
$$\textbf{funct}\left(\left\{\textbf{lisp}\ l:\ \neg\ \text{iskenom}(l)\right\}\right)\textbf{lisp}\ \text{cdr},$$
$$\textbf{funct}\left(\left\{\textbf{lisp}\ l:\ \text{iskenom}(l)\right\}\right)\textbf{lisp}\ \text{repl},$$
$$\textbf{funct}\left(\left\{\textbf{lisp}\ l:\ \text{iskenom}(l)\right\}\right)\textbf{lisp}\ \text{transp},$$
$$\textbf{funct}(\textbf{lisp},\ \textbf{lisp})\ \textbf{lisp}\ \text{cons},$$
$$\textbf{funct}(\textbf{keno})\ \textbf{lisp}\ \text{mklisp},$$
$$\textbf{funct}\left(\left\{\textbf{lisp}\ l:\ \text{iskenom}(l)\right\}\right)\textbf{keno}\ \text{mkkenom},$$
$$\textbf{funct}(\textbf{lisp})\ \textbf{bool}_{i.j}\ \text{iskenom}_{i.j},$$
$$\textbf{bi} - \textbf{funct}(\textbf{lisp}_1,\ \textbf{lisp}_2)(\textbf{lisp}_3,\ \textbf{lisp}_4)\ \text{cons}^{(4)},$$

law CONS$^{(4)}$

$$\text{CONS}\left(l_1, l_2, l_3, l_4\right) = \begin{pmatrix} l_1 \\ \text{II} \\ l_3 \end{pmatrix} * \begin{pmatrix} l_2 \\ \text{II} \\ l_4 \end{pmatrix} = \begin{pmatrix} \left(l_1 * \ l_2\right) \\ \text{II} \\ \left(l_3 * \ l_4\right) \end{pmatrix},$$

law CAR : $\text{car}\left(\text{cons}\left(k, l\right) = k,\right.$

law CDR : $\text{cdr}\left(\text{cons}\left(k, l\right) = l,\right.$

law EN : $\text{repl}\left(\text{transp}\left(l\right)\right) = \text{transp}\left(\text{repl}\left(l\right)\right),$

law CONS : $\neg\ \text{iskenom}\left(l\right) \Rightarrow \text{cons}\left(\text{car}\left(l\right),\ \text{cdr}\left(l\right)\right) = l_{i.j}$

law ISKENOM1 : $\text{iskenom}\left(\text{mklisp}\left(a_{i.j}\right)\right),$

law ISKENOM2 $\neg\ \text{iskenom}\left(\text{cons}\left(k,\ l\right)\right),$

law MKKENOM : $\text{mkkenom}\left(\text{mklisp}\left(a_{i.j}\right)\right) = a_{i.j}$,

law MKKENOM$_{i.j+1}$: $\text{mkkenom}\left(\text{mklisp}\left(\text{repl}\left(\left(a_{i.j}\right)\right)\right)\right) = a$

law MKKENOM$_{i+1.\,j}$: $\text{mkkenom}\left(\text{mklisp}\left(\text{transp}\left(\left(a_{i.j}\right)\right)\right)\right) = a$

law MKLISP : $\text{iskenom} \Rightarrow \text{mklisp}\left(\text{mkkenom}\left(l_{i.j}\right)\right) = l_{i.j}$

endoftype

mode lisp$^{(m,\,n)}\ \chi \mid$ **matrix**$\left(m,\ n\right) \equiv$

$$\chi^{(m,\,n)} \mid \left(\textbf{lisp}^{(m,\,n)}\ \chi\ \text{car},\ \textbf{lisp}^{(m,\,n)}\ \chi\ \text{cdr},\ \textbf{lisp}^{(m,\,n)}\ \chi\ \text{repl},\ \textbf{lisp}^{(m,\,n)}\ \chi\ \text{transp}\right)$$

2.2. Recursive kenomic LISP

2.2.1. Basic terms: atom, eq, equal, subst

Based on McCarthy's presentation of LISP some first steps of its deconstruction shall follow.

Atoms are becoming *kenoms* and patterns of kenoms, i.e. *monomorphies* are building *morphograms*.

1. **kenom**. kenom[x] has the value of T_i or F_i according to whether x is a kenomic symbol.

Thus

kenom [X] = T_i

kenom [(X · A)] = F_i, i,j ∈ s(m)

Monomorphy

For eq [X, A] = T ⟶ monomorph[X . A] = T

2. **eq-kenom**. eq-kenom [x;y] is defined if and only if both x and y are kenomic.
eq-kenom [x; y] = T_i if x and y are of the same pattern, and eq-kenom [x; y] = F_i otherwise.

Thus eq-kenom [X; X] = T_i
X = pattern(A) $\longrightarrow$ eq-kenom [X; A] = T_i
eq-kenom [X; (X · A)] is undefined.

3. **equal** [x; y] = [atom [x] $\wedge$ atom [y] $\wedge$ eq [x; y]]
$\vee$[$\neg$ atom [x] $\wedge\neg$ atom [y] $\wedge$ equal [car [x]; car [y]] $\wedge$ equal [cdr [x]; cdr [y]]].

$$\text{equal} \Rightarrow \begin{pmatrix} \text{equal}-\text{symbol} : \textbf{equal}\left[x;\ y\right] \\ \text{equal}-\text{kenom} : \textbf{equal}\left[x;\ y\right] \\ \text{equal}-\text{monomorph} : \textbf{equal}\left[x;\ y\right] \end{pmatrix}$$

Bisimulation
eq-kenom [X; (X · A)] = U $\longrightarrow$ $\exists$ bisimul [X; (X · A)] = T

Examples
eq-kenom [a, a] = eq [a,b] = eq [b, z] = T, i.e eq[kenom_1, kenom_2] = T
eq-kenom [ab, ba] =T
non-eq-kenom [a, ab]
eq-kenom[aa, bb]=T $\longrightarrow$ monomorph[aa . aa] = T
eq-kenom [aba, abba] = U $\longrightarrow$ $\exists$ eq-bisimul [ab; abba] = T

Substitution
3. subst [x; y; z] = [atom [z] $\longrightarrow$ [eq [z; y]$\longrightarrow$ x; T $\longrightarrow$ z];
T $\longrightarrow$ cons [subst [x; y; car [z]]; subst [x; y; cdr [z]]]]

As an example, we have
subst[(X · A);B; ((A · B) · C)] = ((A · (X · A)) · C).
m h H $H_{h/m}$

Kenomic substitution
$H_1 =_{MG} H_2$, $H_1 \neq_{sem} H_2 \Rightarrow$
subst[m_1, h_1, H_1] = subst[m_2, h_1, H_2]

Context rules for substitution CRS
$\forall h,\ m_1 \in H_1, m_2 \in H_2,\ m_1 =_{MG} m_2,$
$m_1 \neq_{sem} m_2, h \neq_{sem} m_1,\ m_2,$
$\text{length}(m_1) = \text{length}(m_2),$
$\text{kenom}(m_1) \bigcap \text{kenom}(H_1) = \varnothing,$
$\text{kenom}(m_2) \bigcap \text{kenom}(H_2) = \varnothing :$

$$H_1 =_{MG} H_2 \Longrightarrow \textbf{Subst}_{h/m_1}(H_1) =_{MG} \textbf{Subst}_{h/m_2}(H_2)\ ;$$

modulo CRS

Example

$H_1 = \left[\text{aabbacc}\right],\ H_2 = \left[\text{aaccabb}\right],$

$H_1 =_{MG} H_2, \mathrm{H}_1 \neq_{sem} H_2$

$\mathrm{Dec}\left(H_1\right) = \left(\left[\text{aa}\right],\ \left[\text{bb}\right], \left[a\right], \left[\text{cc}\right]\right),$

$\mathrm{Dec}\left(H_2\right) = \left(\left[\text{aa}\right],\ \left[\text{cc}\right], \left[a\right], \left[\text{bb}\right]\right),$

$h = \left[\text{aa}\right],\ m_1 = \left[\text{ddd}\right],\ m_2 = \left[\text{eee}\right],$

$\mathrm{length}\left(m_1\right) = \mathrm{length}\left(m_2\right),$

$m_1 \neq_{sem} m_2,\ h \neq_{sem} m_1,\ m_2,$

$\mathrm{sem}\left(m_i\right) \bigcap \mathrm{sem}\left(\mathrm{H}_i\right) = \text{ø},\ i = 1,\ 2$

$\mathrm{Dec}\left(H_1\right) = \left(\left[\text{aa}\right],\ \left[\text{bb}\right], \left[a\right], \left[\text{cc}\right]\right)$

$\mathrm{Subst}\left(H_1\right)_{\left[\text{aa}\right]/_{\left[\text{ddd}\right]}}\left(\left[\text{aa}\right],\ \left[\text{bb}\right], \left[a\right], \left[\text{cc}\right]\right) = \left(\left[\text{ddd}\right],\ \left[\text{bb}\right], \left[a\right], \left[\text{cc}\right]\right)$

$\mathrm{Dec}\left(H_2\right) = \left(\left[\text{aa}\right],\ \left[\text{cc}\right], \left[a\right], \left[\text{bb}\right]\right)$

$\mathrm{Subst}\left(H_2\right)_{\left[\text{aa}\right]/_{\left[\text{eee}\right]}}\left(\left[\text{aa}\right],\ \left[\text{cc}\right], \left[a\right], \left[\text{bb}\right]\right) = \left(\left[\text{eee}\right],\ \left[\text{cc}\right], \left[a\right], \left[\text{bb}\right]\right)$

$\mathrm{H}_1 =_{MG} \mathrm{H}_2 \Longrightarrow \mathrm{Subst}\left(H_1\right)_{\left[\text{aa}\right]/_{\left[\text{ddd}\right]}} =_{MG} \mathrm{Subst}\left(H_2\right)_{\left[\text{aa}\right]/_{\left[\text{eee}\right]}}$

$\left[\text{aabbacc}\right] =_{MG} \left[\text{aaccabb}\right] \Longrightarrow \left[\text{dddbbacc}\right] =_{MG} \left[\text{eeeccabb}\right].$

2.2.2. Kenomic CDR, CAR, CONS

3. **car**. car[x] is defined if and only if x is not atomic. car [(e1 · e2)] = e1. Thus car [X] is undefined.

car [(X · A)] = X

car [((X · A) · Y)] = (X · A)

⇒

$\mathrm{CAR}\ [(\mathrm{X}_{\mathrm{i.j}} . \mathrm{A}_{\mathrm{i.j}})] = \mathrm{X}_{\mathrm{i.j}}$

$\mathrm{CAR}\ [(\mathrm{X}_{\mathrm{i.j}} . \mathrm{A}_{\mathrm{i.j}}) \cdot \mathrm{Y}_{\mathrm{i.j}})] = (\mathrm{X}_{\mathrm{i.j}} . \mathrm{A}_{\mathrm{i.j}})$.

Enactional car

$\mathrm{CAR}_{\mathrm{EN}}\ [(\mathrm{X}_{\mathrm{i.j}} . \mathrm{A}_{\mathrm{i.j}})] = \mathrm{X}_{\mathrm{i.j}} . \left(\perp_{\mathrm{i.j}}\right) \mid \mathrm{A}_{\mathrm{i.j+1}}$

$\mathrm{CAR}_{\mathrm{EN}}\ [(\mathrm{X}_{\mathrm{i.j}} . \mathrm{A}_{\mathrm{i.j}}) \cdot \mathrm{Y}_{\mathrm{i.j}})] = ((\mathrm{X}_{\mathrm{i.j}} . \mathrm{A}_{\mathrm{i.j}}) . \left(\perp_{\mathrm{i.j}}\right)) \mid \mathrm{Y}_{\mathrm{i.j+1}}$

4. **cdr**. cdr [x] is also defined when x is not atomic. We have cdr[(e1 · e2)] = e2. Thus cdr [X] is undefined.

cdr [(X · A)] = A

cdr [((X · A) · Y)] = Y

⇒

$CDR\ [(X_{i.j}\ .\ A_{i.j})] = A_{i.j}$

$CDR\ [((X_{i.j}\ .\ A_{i.j})\ \cdot\ Y_{i.j}\ )] = Y_{i.j}$

Enactional CDR

$CDR_{EN}\ [(X_{i.j}\ .\ A_{i.j})] = ((\perp_{i.j}).\ A_{i.j})\ |\ X_{i.j+1}$

$CDR_{EN}\ [((X_{i.j}\ .\ A_{i.j})\ \cdot\ Y_{i.j}\ )] = ((\perp_{i.j}).\ Y_{i.j}\ )\ |\ (X_{i.j+1}\ \cdot\ A_{i.j+1})$

5. **cons**. cons [x; y] is defined for any x and y. We have
cons [e1; e2] = (e1 · e2). Thus
cons [X; A] = (X A)
cons [(X · A); Y] = ((X · A) Y)
⇒
Kenomic CONS
X = (A) ⟶ CONS [X; A] = (X A)|(X B)
X = (X . A) ⟶ CONS [(X · A); Y] = ((X · A) Y)|((X · A) Z)

CONS, CAR and CDR

Enactional CAR

$$CAR\left[\left(X_{i.j}.\ A_{i.j}\right)\right] = X_{i.j}.\left(\perp_{i.j}\right)\ \Big|\ A_{i.j+1}$$

$$CAR\ \left[\left(\left(X_{i.j}.\ A_{i.j}\right)\cdot Y_{i.j}\right)\right] = \left(\left(X_{i.j}.\ A_{i.j}\right).\left(\perp_{i.j}\right)\right)\ \Big|\ Y_{i.j+1}$$

Enactional CDR

$$CDR\left[\left(X_{i.j}.\ A_{i.j}\right)\right] = \left(\left(\perp_{i.j}\right).\ A_{i.j}\right)\ \Big|\ X_{i.j+1}$$

$$CDR\left[\left(\left(X_{i.j}.\ A_{i.j}\right)\cdot Y_{i.j}\right)\right] = \left(\left(\perp_{i.j}\right).\ Y_{i.j}\right)\ \Big|\ \left(X_{i.j+1}.\ A_{i.j+1}\right)$$

Kenomic CONS

$$X = \left(A\right) \longrightarrow CONS\left[X;\ A\right] = \left(X\,A\right)\Big|\left(X\,B\right)$$

$$X = \left(X\ .\ A\right) \longrightarrow cons\left[\left(X\cdot A\right);\ Y\right] = \left(\left(X\cdot A\right)Y\right)\Big|\left(\left(X\cdot A\right)Z\right)$$

2.2.3. Enactional CAR and CDR as composed operators

Enaction was previously defined as a combination of replication and elimination. This fits together with an understanding of enactional operations as composed of replication and elimination in the sense of CDR and CAR in LISP. Replication is like transposition an operator belonging to the so called super-operators, ID, PERM, REPL, RED and BIF of polycontextural logic. Super-operators are applicable to all internal LISP terms and operators, hence not only to CAR and CDR but to CONS, APPEND, etc. too.
In this context, the LISP operators CAR and CDR are modeled by the polycontextural operation of reduction (RED), while the new LISP operation of enaction is modeled by replication (REPL). Both together are defining the enactional CDR and CAR albeit based now not on lists but on morphograms.

Classical operations like CAR, CDR and CONS are defined by the identity mapping ID. Hence ID(CAR) = CAR, ID(CDR) = CDR and ID(CONS) = CONS. Thus, ID : X $\longrightarrow$ X with X= {CAR, CDR, CONS}.

Both CAR_{EN} and CDR_{EN} are defined by CAR and CDR and the replicative operation of reflectional enaction.

Replicational enaction for CAR and CDR

$$CAR_{EN} \equiv CAR(REPL) :$$

$$CAR(REPL(X_{i.j}.A_{i.j})) = CAR(X_{i.j}.A_{i.j}) \mid A_{i.j+1} = X_{i.j}.(\perp_{i.j}) \mid A_{i.j+1}$$

$$CAR[REPL((X_{i.j}.A_{i.j}) \cdot Y_{i.j}))] =$$

$$CAR((X_{i.j}.A_{i.j}).Y_{i.j}) \mid Y_{i.j+1} = ((X_{i.j}.A_{i.j}).(\perp_{i.j})) \mid Y_{i.j+1}$$

$$CDR_{EN} \equiv CDR(REPL) :$$

$$CDR[REPL(X_{i.j}.A_{i.j})] = CDR((X_{i.j}).A_{i.j}) \mid X_{i.j+1} = ((\perp_{i.j}).A_{i.j}) \mid X_{i.j+1}$$

$$CDR[REPL((X_{i.j}.A_{i.j}) \cdot Y_{i.j})] =$$

$$CDR$$

$$((X_{i.j}).A_{i.j}).Y_{i.j} \mid (X_{i.j+1}.A_{i.j+1}) = ((\perp_{i.j}).Y_{i.j}) \mid (X_{i.j+1}.A_{i.j+1})$$

Replicational enaction matrix

CAR_{REPL} :

	O^1	O^2
M^1	$(X.A)_{1.1}$	–
M^2	–	–

$\Rightarrow$

	O^1	O^2
M^1	$(X.A)_{1.1}$	–
M^2	$A_{1.2}$	–

$\Rightarrow$

	O^1	O^2
M^1	$(X.\perp)_{1.1}$	–
M^2	$A_{1.2}$	–

CDR_{REPL} :

	O^1	O^2
M^1	$(X.A)_{1.1}$	–
M^2	–	–

$\Rightarrow$

	O^1	O^2
M^1	$(X.A)_{1.1}$	–
M^2	$X_{1.2}$	–

$\Rightarrow$

	O^1	O^2
M^1	$(\perp.A)_{1.1}$	–
M^2	$X_{1.2}$	–

Transpositional enaction for CAR and CDR

$$CAR(TRANSP) :$$

$$CAR_{TRANSP}(X_{i.j}.A_{i.j}) = X_{i.j}.(\perp_{i.j}) \mid A_{i+1.j}$$

$$CAR_{TRANSP}((X_{i.j}.A_{i.j}) \cdot Y_{i.j}) = ((X_{i.j}.A_{i.j}).(\perp_{i.j})) \mid Y_{i+1}$$

$$CDR(TRANSP) :$$

$$CDR_{TRANSP}[(X_{i.j}.A_{i.j})] = ((\perp_{i.j}).A_{i.j}) \mid X_{i+1.j}$$

$$CDR_{TRANSP}[(X_{i.j}.A_{i.j}) \cdot Y_{i.j}] = ((\perp_{i.j}).Y_{i.j}) \mid (X_{i+1.j}.A_{i+1.j})$$

Transpositional enaction matrix

CAR_{TRANSP} :

$$\begin{array}{|ccc|} \hline & O^1 & O^2 \\ M^1 & (X.A)_{1.1} & - \\ M^2 & - & - \\ \hline \end{array} \Rightarrow \begin{array}{|ccc|} \hline & O^1 & O^2 \\ M^1 & (X.A)_{1.1} & A_{2.1} \\ M^2 & - & - \\ \hline \end{array} \Rightarrow \begin{array}{|ccc|} \hline & O^1 & O^2 \\ M^1 & (X.\perp)_{1.1} & A_{2.1} \\ M^2 & - & - \\ \hline \end{array}$$

CDR_{TRANSP} :

$$\begin{array}{|ccc|} \hline & O^1 & O^2 \\ M^1 & (X.A)_{1.1} & - \\ M^2 & - & - \\ \hline \end{array} \Rightarrow \begin{array}{|ccc|} \hline & O^1 & O^2 \\ M^1 & (X.A)_{1.1} & X_{2.1} \\ M^2 & - & - \\ \hline \end{array} \Rightarrow \begin{array}{|ccc|} \hline & O^1 & O^2 \\ M^1 & (\perp.A)_{1.1} & X_{2.1} \\ M^2 & - & - \\ \hline \end{array}$$

Both together :

$F_{repl}\left(F_{transp}\right) = F_{trnsp}\left(F_{repl}\right) : F_{i.j} \longrightarrow F_{i+1.\,j+1}$

Enaction defined with DEFUN :

$CAR_{TRANSP:}\ \left(\text{DEFUN CAR}_{TRANSP}\left(X\right)\ \left(\text{CAR}\left(\text{TRANSP}\left(X\right)\right)\right)\right.$

$CDR_{TRANSP:}\ \left(\text{DEFUN CDR}_{TRANSP}\left(X\right)\ \left(\text{CDR}\left(\text{TRANSP}\left(X\right)\right)\right)\right.$

$CAR_{REPL:}\ \left(\text{DEFUN CAR}_{REPL}\left(X\right)\ \left(\text{CAR}\left(\text{REPL}\left(X\right)\right)\right)\right.$

$CDR_{REPL:}\ \left(\text{DEFUN CDR}_{REPL}\left(X\right)\ \left(\text{CDR}\left(\text{REPL}\left(X\right)\right)\right)\right.$.

2.2.4. Rules for CAR, CDR and CONS

car, cdr, and cons are easily seen to satisfy the relations

$1.\ \text{car}\left[\ \text{cons}\left[\ x;\ y\right]\right] = x$

$\Rightarrow$

$\text{CAR}\left[\text{CONS}\left[X;\ A\right]\right] = \text{CAR}\left[\left(X\,A\right)\middle|\left(X\,B\right)\right] = \text{CAR}\left(X\,A\right)\ \middle|\ \text{CAR}\left(X\,B\right) =$
$X_{i.j}\ \middle|\ A_{i.j+1}\ \middle|\ X_{i.j}\ \middle|\ B_{i.j+1} = X_{i.j}\ \middle|\ X_{i.j}\ \middle|\ A_{i.j+1}\ \middle|\ B_{i.j+1}.$

$\text{CAR}\left[\text{CONS}\left[X;\ A\right]\right] = X_{i.j};\ X_{i.j}\ \middle|\ A_{i.j+1}\ ;\ B_{i.j+1}.$

$\text{CONS}_{i.j+1}\left(\text{CAR}\left[\text{CONS}\left[X;\ A\right]\right]\right) = X_{i.j}\ \middle|\ \text{CONS}_{i.j+1}\left(A_{i.j+1}\ ;\ B_{i.j+1}\right) =$
$\left(X.X\right)_{i.j}\ \middle|\ \left(A_{i.j+1}\ B_{i.j+1}\right).$

$\left[X\ ;\ A\right]$ $\qquad\qquad$ $\text{CONS}\left[X\ ;\ A\right]$

$$\begin{array}{|cc|}\hline & O^1 \\ M^1 & (X;\ A)_{1.1} \\ M^2 & - \\ \hline\end{array} \Rightarrow \begin{array}{|cc|}\hline & O^1 \\ M^1 & (X.A)_{1.1}\ .1 \mid (X.B)_{1.1}\ .2 \\ M^2 & - \\ \hline\end{array} \Rightarrow$$

$\mathrm{CAR}_{EN}\left[(X\,A)\,|\,(X\,B)\right]$ $\qquad$ $\mathrm{CONS}_{1.2}\left(\mathrm{CAR}\left[(X\,A)\,|\,(X\,B)\right]\right)$

$$\begin{array}{|cc|}\hline & O^1 \\ M^1 & (X.\perp)_{1.1}\ .1 \mid (X.\perp)_{1.1}\ .2 \\ M^2 & A_{1.2} \mid B_{1.2} \\ \hline\end{array} \Rightarrow \begin{array}{|ccc|}\hline & O^1 & O^2 \\ M^1 & (X.X)_{1.1} & - \\ M^2 & (A\ B)_{1.2} & - \\ \hline\end{array}.$$

2. $\mathrm{cdr}\left[\mathrm{cons}\left[x;\ y\right]\right] = y$

$\Rightarrow$

$\mathrm{CDR}\left[\mathrm{CONS}\left[X;\ A\right]\right] = \mathrm{CDR}\left[(X\,A)\,|\,(X\,B)\right] = \mathrm{CDR}(X\,A)\ |\ \mathrm{CDR}(X\,B) =$

$A_{i.j}\ |\ X_{i.j+1}\ |\ B_{i.j}\ |\ X_{i.j+1} = A_{i.j};\ B_{i.j}\ |\ X_{i.j+1}\ |\ X_{i.j+1}.$

$\mathrm{CDR}\left[\mathrm{CONS}\left[X;\ A\right]\right] = A_{i.j};\ B_{i.j}\ |\ X_{i.j+1};\ X_{i.j+1}.$

$\mathrm{CONS}\left(\mathrm{CDR}\left[\mathrm{CONS}\left[X;\ A\right]\right]\right) = \mathrm{CONS}\left(A_{i.j};\ B_{i.j}\right)\ |\ X_{i.j+1} = \left(A_{i.j}\ B_{i.j}\right)\ |\ (X\ X)_{i.j+1}.$

$\left[X;\ A\right]$ $\qquad$ $\mathrm{CONS}\left[X;\ A\right]$

$$\begin{array}{|cc|}\hline & O^1 \\ M^1 & (X;\ A)_{1.1} \\ M^2 & - \\ \hline\end{array} \Rightarrow \begin{array}{|cc|}\hline & O^1 \\ M^1 & (X.A)_{1.1}\ .1 \mid (X.B)_{1.1}\ .2 \\ M^2 & - \\ \hline\end{array} \Rightarrow$$

$\mathrm{CDR}_{EN}\left[(X\,A)\,|\,(X\,B)\right]$

$$\begin{array}{|ccc|}\hline & O^1 & O^2 \\ M^1 & (X.\perp)_{1.1}\ .1 \mid (X.\perp)_{1.1}\ .2 & (A_{2.1}\ B_{2.1}) \\ M^2 & - & - \\ \hline\end{array} \Rightarrow$$

$\mathrm{CONS}_{1.2}\left(\mathrm{CDR}\left[(X\,A)\,|\,(X\,B)\right]\right)$ $\qquad$ $\mathrm{CONS}_{1.2}\left(\mathrm{CDR}\left[(X\,A)\,|\,(X\,B)\right]\right)$

$$\begin{array}{|ccc|}\hline & O^1 & O^2 \\ M^1 & (X.X)_{1.1} \mid (\perp.\perp)_{1.1} & (A\ B)_{2.1} \\ M^2 & - & - \\ \hline\end{array} \Rightarrow \begin{array}{|ccc|}\hline & O^1 & O^2 \\ M^1 & (X.X)_{1.1} & (A\ B)_{2.1} \\ M^2 & - & - \\ \hline\end{array}.$$

$\text{cons}\left[\text{car}\left[x\right]; \text{cdr}\left[x\right]\right] = x$, provided that x is not atomic.

$\Rightarrow$

$$\text{CONS}\left[\text{CAR}\left[X\right]; \text{CDR}\left[X\right]\right] = \text{CONS}\left[\left(X_{1.1}.\ \perp\right); \left(\perp . X_{1.2}\right)\right] =$$

$$\text{CONS}\left[\left(X_{1.1}.\ \perp\right); \left(\perp . X_{1.2}\right)\right] = \text{CONS}\left[X_{1.1}\ ;\ X_{1.2}\right] = \left(X_{1.1} \,\middle|\, X_{1.2}\right) = \begin{pmatrix} X_{1.1} \\ \square \\ X_{1.2} \end{pmatrix}.$$

$$\text{CONS}\left[\text{CAR}\left[X;\ A\right]; \text{CDR}\left[X;\ B\right]\right] = \text{CONS}\left[\left(X_{1.1}.A_{1.2}\right); \left(X_{1.2}\,.\,B_{1.1}\right)\right] =$$

$$\text{CONS}\left(\begin{pmatrix} X_{1.1} \\ \square \\ A_{1.2} \end{pmatrix}; \begin{pmatrix} B_{1.1} \\ \square \\ X_{1.2} \end{pmatrix}\right) = \begin{pmatrix} \left(X_{1.1}\ \ B_{1.1}\right) \\ \square \\ \left(X_{1.2}\ \ A_{1.2}\right) \end{pmatrix}$$

2.2.5. Formal approach

The idea of a kenomic LISP is based on two decisions. One is for a transition from lists to *morphograms*, the second is a dissemination of symbolic LISP over the *kenomic matrix* which is enabling new 'trans-contextural' operators, like mediation, replication and transposition.

Nevertheless, the new morphogram-based programming paradigm, polyLISP, kenoLISP or morphLISP, gets its first introduction as a mimickry of the methods of the list-based LISP.

$\text{LISP}^{(m,\,n)} = [\text{LISP; ops; sops; n, m} \in \mathcal{N}]$

Operators
ops:
CONS, CDR, CAR.
Super – Operators
sops :
ID : identity,
PERM : Permutation,
RED : Reduction,
$\diamond$: transposition, BIF,
$\square$: replication, Repl,
$\amalg$: mediation.

Object
$\text{Lisp}_{i.j}$ with operators CAR, CDR and CONS as basic ' intra – contextural ' operators.

$$KM^{(3,2)} = \begin{pmatrix} \left(X_{1.1} \diamond X_{2.1}\right) \diamond X_{3.1} \\ \square \amalg \square \amalg \square \\ \left(X_{1.2} \diamond X_{2.2}\right) \diamond X_{3.2} \\ \square \amalg \square \amalg \square \\ \left(X_{1.3} \diamond X_{2.3}\right) \diamond X_{3.3} \end{pmatrix}$$

Mediation : $\left(X_{1.1} \amalg X_{2.2}\right) \amalg X_{3.3}$

Replication : $\left(X_{i.1} \amalg X_{i.2}\right) \amalg X_{i.3}$

Transposition : $\left(X_{1.i} \amalg X_{2.i}\right) \amalg X_{3.i}$

2.2.6. Types of action

Typs of action on a kenomic object is either iterative, accretive, transposive or enactive and metamorphic.

Iteration

$$Op_{iter}\left(Ob_{i.j}\right) = \left(Ob_{i.j+1}\right)$$

$$OP_{iter}\left[\left(Ob_{i.j}.\, Ob_{i.j}\right).\, Ob_{i.j}\right] = \left[\left(Ob_{i.j}.\, Ob_{i.j}\right)_{i.j+1}.\, Ob_{i.j+1}\right]$$

Accretion

$$Op_{accr}\left(Ob_{i.j}\right) = \left(Ob_{i+1.\,j}\right)$$

$$OP_{accr}\left[\left(Ob_{i.j}.\, Ob_{i.j}\right) \cdot Ob_{i.j}\right] = \left[\left(Ob_{i.j}.\, Ob_{i.j}\right)_{i+1.\,j}.\, Ob_{i+1.\,j}\right]$$

Enaction

$$OP_{EN}\left[\left(Ob_{i.j}.\, OB_{i.j}\right)\right] = \begin{pmatrix} Ob_{i.j} \\ \square_{1.2} \\ Ob_{i.j+1} \end{pmatrix}$$

$$OP_{EN}\left[\left(Ob_{i.j}.\, Ob_{i.j}\right) \cdot Ob_{i.j}\right] = \begin{pmatrix} \left(Ob_{i.j}.\, Ob_{i.j}\right) \\ \square_{1.2} \\ Ob_{i.j+1} \end{pmatrix}$$

Transposition

$$OP_{REPL1}\left[\left(Ob_{i.j}.\, Ob_{i.j}\right)\right] = \begin{pmatrix} Ob_{1.1} \\ \amalg_{1.2} \\ Ob_{2.2} \diamond_{2.1} Ob_{2.1} \\ \amalg_{2.3} \\ Ob_{3.3} \diamond_{3.1} Ob_{3.1} \end{pmatrix}$$

Null

Replication

$$OP_{REPL1}\left[\left(Ob_{i.j}.\,Ob_{i.j}\right)\right] = \begin{pmatrix} Ob_{1.1}\,\square_{1.2}\,Ob_{1.2}\,\square_{1.3}\,Ob_{1.3} \\ \amalg_{1.2} \\ Ob_{2.2} \\ \amalg_{2.3} \\ Ob_{3.3} \end{pmatrix}$$

2.2.7. Logical topics

$$\text{atom}\left[X\right] = T$$

$$\text{atom}\left[\left(X \cdot A\right)\right] = F \in \text{LISP}.$$

$$\begin{pmatrix} \text{atom}\left[X^i\right] \\ \amalg \\ \text{atom}\left[X^j\right] \end{pmatrix} \Rightarrow \begin{pmatrix} \text{atom}\left[X^i\right] = T_i \\ \amalg \qquad \amalg \\ \text{atom}\left[X^j\right] = T_j \end{pmatrix} \in \text{kenoLISP}$$

$$\begin{pmatrix} \text{atom}\left[X^i\right] \\ \amalg \\ \text{atom}\left[\left(X^j.\,A^j\right)\right] \end{pmatrix} \Rightarrow \begin{pmatrix} \text{atom}\left[X^i\right] = T_i \\ \amalg \qquad \amalg \\ \text{atom}\left[\left(X^j.\,A^j\right)\right] = F_j \end{pmatrix} \in \text{kenoLISP}$$

2.3. Special features of kenomic LISP

2.3.1. Retrograde recursivity

Despite the name “kenomic Lisp” the proposed kenomic Lisp is not dealing with lists but with kenomic patterns, i.e. morphograms. This has consequences for the concept of recursivity, crucial to Lisp, and applies to different aspects of the newly discovered features of retrograde recursivity.

2.3.2. Parallelism

Parallelism, concurrence and simultaneity of processes, actions and interactions are primordial in kenoLISP and morphoLISP. The feature of parallelism is obvious for polycontextural systems. Each contexture in a polycontextural compound has structural space for its own formalism and therefore programming languages.
For kenomic and morphogrammatic systems the case is slightly less obvious. It becomes ‘natural’ with the understanding of kenomic operations. Kenomic operations are from the very beginning ‘dis-contextural’, delivering simultaneously different results.

2.3.3. Self-referentiality

A kind of self-referentiality is crucial for classical LISP especially for the definition of recursivity.
But this kind of self-referentiality is not based on a chiastic interplay of terms and operations but on the concept of a “self"-application of functions on its previous values.

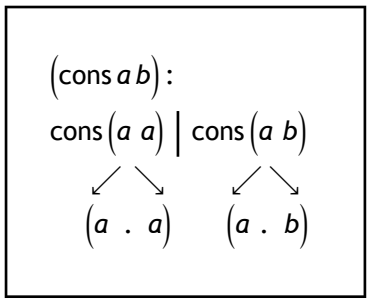
2.3.4. Enaction

The operation of enaction was introduced as a positive interpretation of the elimination of terms like with the double crossing in Spencer Brown’s calculus of indication: {{ }} = . Formal enaction accepts the elimination of the term but recalls it on another level of the formalism.

Formal enaction is therefore understood as a memristive cancellation of terms.

2.4. Examples of kenomic LISP

2.4.1. Kenomic CONS



$$X = (ab) \longrightarrow \text{cons}\,(\text{cons}\,(ab)\,a) \longrightarrow \text{cons}\,((a\,.\,b)\,a) \longrightarrow$$

$$\begin{pmatrix} \text{cons}_a\,((a\,.\,b)\,a) \longrightarrow (a.\,b.\,a) \\ \text{cons}_b\,((a\,.\,b)\,b) \longrightarrow (a.\,b.\,b) \\ \text{cons}_c\,((a\,.\,b)\,c) \longrightarrow (a.\,b.\,c) \end{pmatrix}.$$

$$X = (ab) \longrightarrow \text{cons}\,(\text{cons}\,(ab)\,a) \longrightarrow$$

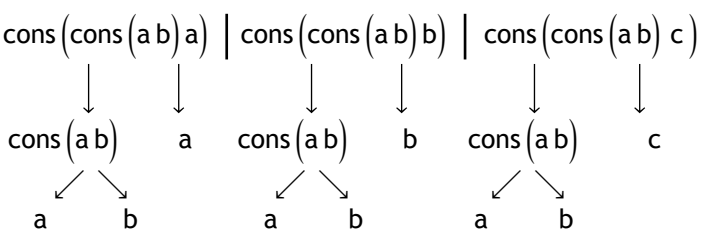
$$\begin{pmatrix} \text{cons}\,((a\,.\,a)\,a) \\ \text{cons}\,((a\,.\,b)\,a) \end{pmatrix} \longrightarrow$$

$$\begin{pmatrix} \text{cons}_a\,((a\,.\,a)\,a) \longrightarrow (a.\,a.\,a) \\ \text{cons}_b\,((a\,.\,a)\,b) \longrightarrow (a.\,a.\,b) \end{pmatrix}$$

$$\begin{pmatrix} \text{cons}_a\,((a\,.\,b)\,a) \longrightarrow (a.\,b.\,a) \\ \text{cons}_b\,((a\,.\,b)\,b) \longrightarrow (a.\,b.\,b) \\ \text{cons}_c\,((a\,.\,b)\,c) \longrightarrow (a.\,b.\,c) \end{pmatrix}.$$

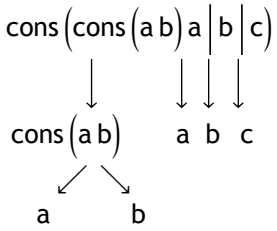
kenomic CONS example

$\text{cons}\left(\text{cons}\left(\text{a b}\right)\text{a}\right)$:



kenomic cons

$\text{cons}\left(\text{cons}\left(\text{a b}\right)\text{a}\right)$:



Kenomic recursion of "$\text{cons}\left(\text{ab, ab}\right)$":

$\text{cons}\left(\text{ab, ab}\right) = \text{cons}\left(\text{cons}\left(\text{ab, a}\right)\text{, a}\right)$:

$\text{cons}\left(\text{ab, a}\right) = \left(\text{aba}\right), \left(\text{abb}\right), \left(\text{abc}\right).$

$\text{cons}\left(\text{aba, a}\right) = \left(\text{abaa}\right)\#, \left(\text{abab}\right), \left(\text{abac}\right),$

$\text{cons}\left(\text{abb, a}\right) = \left(\text{abba}\right), \left(\text{abbb}\right)\#, \left(\text{abbc}\right),$

$\text{cons}\left(\text{abc, a}\right) = \left(\text{abca}\right), \left(\text{abcb}\right), \left(\text{abcc}\right)\#, \left(\text{abcd}\right).$

$\text{collect}\left(\text{cons}\left(\text{ab, ab}\right)\right) = \left\{\left(\text{abab}\right), \left(\text{abba}\right), \left(\text{abac}\right), \left(\text{abbc}\right), \left(\text{abca}\right), \left(\text{abcb}\right), \left(\text{abcd}\right)\right\}.$

Results with #,

like $\left(\text{abcc}\right)\#$ are in conflict with the structure of the pattern $\left[\text{ab}\right]$ of $\text{cons}\left(\text{ab, ab}\right)$.

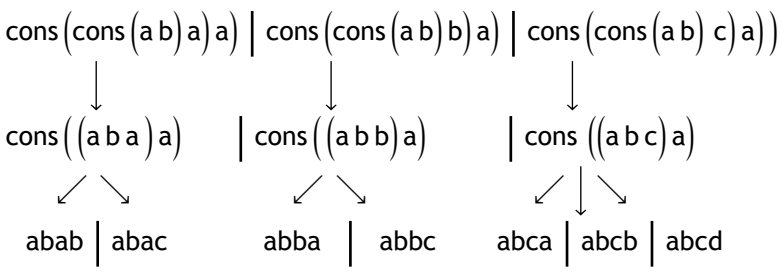
Therefore,

$\text{cons}\left(\text{ab, ab}\right) = \left\{\left(\text{abab}\right), \left(\text{abba}\right), \left(\text{abac}\right), \left(\text{abbc}\right), \left(\text{abca}\right), \left(\text{abcb}\right), \left(\text{abcd}\right)\right\}.$

Production scheme

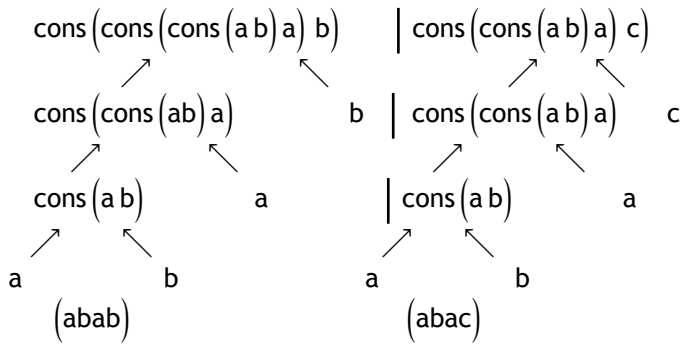
$$\mathbf{cons}\left(\mathbf{ab},\mathbf{ab}\right)=\mathbf{cons}\left(\mathbf{cons}\left(\mathbf{ab},\mathbf{a}\right),\mathbf{a}\right)$$

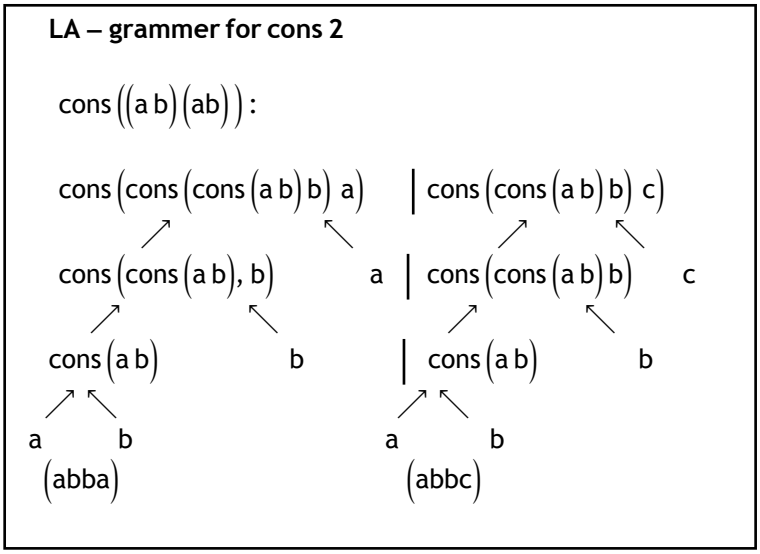
$\mathrm{cons}\left(\mathrm{cons}\left(\mathrm{a\ b}\right)\mathrm{a}\right)\mathrm{a}\right):$

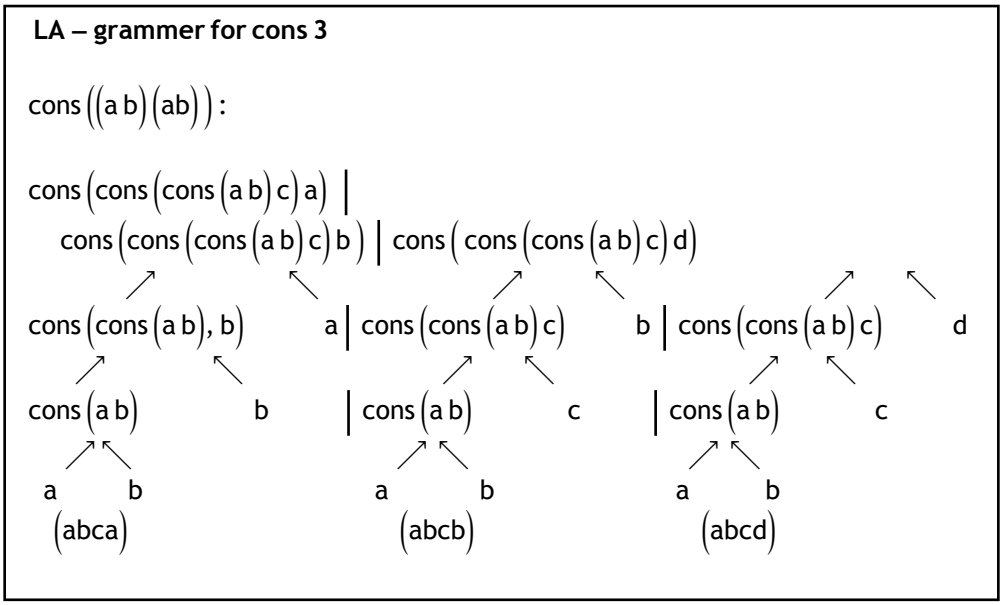


LA – grammer for cons 1

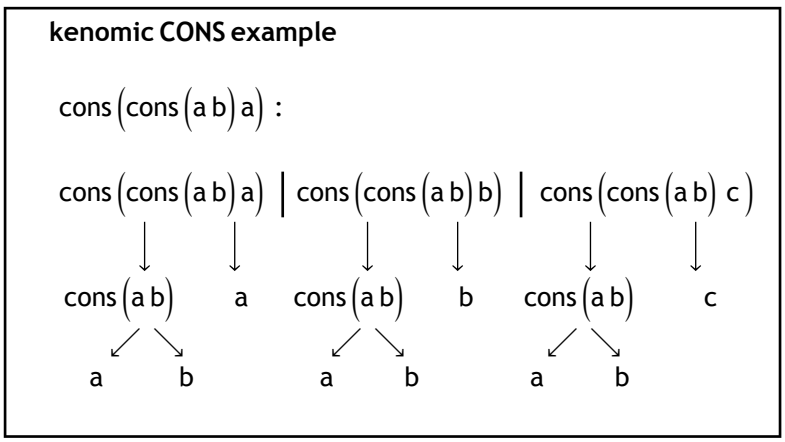
$\mathrm{cons}\left(\left(\mathrm{a\ b}\right)\left(\mathrm{ab}\right)\right):$







2.4.2. Kenomic CONS - and reduction



How are productions and their reductions related?
Obviously, there is an asymmetry between formula and solution involved.

"S-expressions are the fundamental data objects of LISP. They consist of(1) atoms and (2) CONS-cells.

A list is either (1) the atom NIL or (2) the result of CONSing an s-expression onto the beginning of an existing list."

"...the set of s-expression is closed under CONS,...list are s-expressions."

It follows that s-expressions are non-ambiguous.

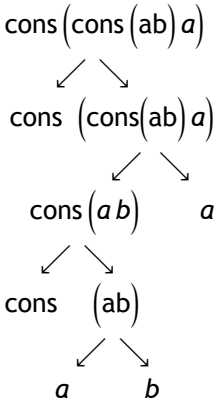
Kenomic expressions are not covered by a unique tree but by several "parallel" trees. Therefore, kenomic expressions have a set of trees, or a forest of trees, as their representation.

There is an asymmetry between operators (CONS, CAR, CDR) and data objects.

cons(cons(ab), a) ⟶ (aba), (abb), (abc).
On the other hand we have:
reverse CONS: (aba), (abb), (abc) ⟶ cons(cons(ab), a).

This offers a *reduction method* from objects to operators.

The syntactic structure of the example *"cons(cons(ab), a)"* is simply a singular tree while the kenomic structure is - in this case - a triple of trees.



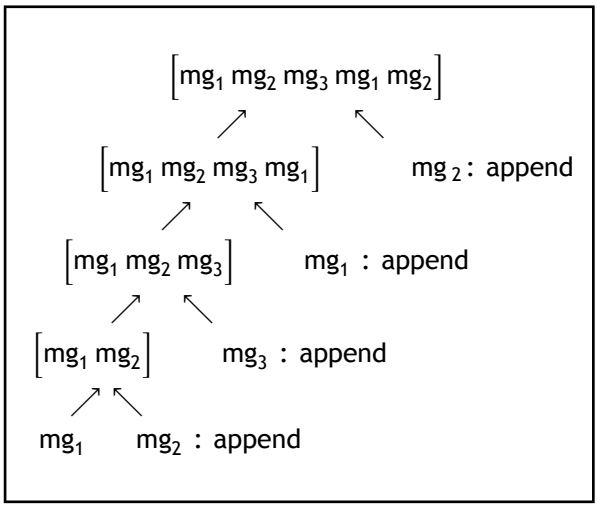
Hence, morphorams [aba], [abb], [abc] are operationally reducible to the form "cons(cons(ab), a)". That is, the 3 different morphograms have a common singular operational representation in "cons(cons(ab), a)". Or more generally, the operational CONS-representation is "CONS(CONS(X Y)X)".

$$\frac{\mathrm{CONS}\left(\mathrm{CONS}\left(\mathrm{X\,Y}\right)\mathrm{X}\right)}{\mathrm{CONS}\left(\mathrm{CONS}\left(\mathrm{X\,Y}\right)\mathrm{X}\right)\ \Big|\ \mathrm{CONS}\left(\mathrm{CONS}\left(\mathrm{X\,Y}\right)\mathrm{Y}\right)\ \Big|\ \mathrm{CONS}\left(\mathrm{CONS}\left(\mathrm{X\,Y}\right)\mathrm{Z}\right)}$$

2.4.3. Kenomic append as “possible continuations"

1. **append** [x;y].
append [x; y] = [null[x] ⟶ y; T ⟶ cons [car [x]; append [cdr [x]; y]]]

An example is
append[(A, B); (C, D, E)] = (A, B, C, D, E)



$$LA(MG) = \text{append}('mg_1\ 'mg_2\ 'mg_3\ 'mg_1\ 'mg_2) = (mg_1\ mg_2\ mg_3\ mg_1\ mg_2)$$

$$\text{keno} - \text{append}(ab,\ aab) = abaab,\ baaab,\ bcaab,\ = (abaab),\ (abbba),\ (abcca)$$

2.4.4. Kenomic CDR , CAR and CONS together

$$CDR\ [(X_{i.j}\ .\ A_{i.j})] = ((\perp_{i.j}).\ A_{i.j}) \mid X_{i.j+1}$$
$$CDR\ [((X_{i.j}\ .\ A_{i.j})\cdot Y_{i.j})] = ((\perp_{i.j}).Y_{i.j}) \mid (X_{i.j+1}\ .\ A_{i.j+1}).$$

$$cdr_{EN}[(aa_{i.j}\ .\ ba_{i.j})] = ((\perp_{i.j}).\ ba_{i.j}) \mid aa_{i.j+1},$$

$$CAR\ [(X_{i.j}\ .\ A_{i.j})] = X_{i.j}\ .\ (\perp_{i.j}) \mid A_{i.j+1},$$
$$CAR\ [(X_{i.j}\ .\ A_{i.j})\cdot Y_{i.j})] = ((X_{i.j}\ .\ A_{i.j})\ .\ (\perp_{i.j})) \mid Y_{i.j+1},$$

$$car_{i.j}[(aa_{i.j}\ .\ ba_{i.j})] = aa_{i.j}\ .\ (\perp_{i.j}) \mid ba_{i.j+1}.$$

$$X = (A) \longrightarrow cons[X; A] = (X\,A) \mid (X\,B)$$
$$X = (X.A) \longrightarrow cons[(X\cdot A); Y] = ((X\cdot A)\,Y) \mid ((X\cdot A)\,Z).$$

$$\mathrm{cons}_{i.j}\left(\mathrm{cdr}_{EN}\left(\mathrm{aa}_{i.j}\,.\,\mathrm{ba}_{i.j}\right),\mathrm{car}\left(\mathrm{aa}_{i.j}\,.\,\mathrm{ba}_{i.j}\right)\right)=$$

$$\mathrm{cons}_{i.j}\left(\left(\perp_{i.j}.\,\mathrm{ba}_{i.j}\right)\middle|\,\mathrm{aa}_{i.j+1}\right),\ \left(\mathrm{aa}_{i.j}.\perp_{i.j}\middle|\,\mathrm{ba}_{i.j+1}\right)=$$

$$\mathrm{cons}_{i.j}\left(\left(\mathrm{ba}_{i.j}\,\middle|\,\mathrm{aa}_{i.j+1}\right),\ \left(\mathrm{aa}_{i.j}.\,\middle|\,\mathrm{ba}_{i.j+1}\right)\right)=$$

$$\mathrm{cons}_{i.j}\left(\mathrm{ba}_{i.j},\ \mathrm{aa}_{i.j}\right)\,\middle|\,\mathrm{cons}_{i.j+1}\left(\mathrm{aa}_{i.j+1},\mathrm{ba}_{i.j+1}\right)=$$

$$\left(\mathrm{ba}_{i.j}\ \ \mathrm{aa}_{i.j}\,\middle|\,\mathrm{ba}_{i.j}\mathrm{bb}_{i.j}\right)\,\middle|\,\left(\mathrm{aa}_{i.j+1}\mathrm{ab}_{i.j+1}\,\middle|\,\mathrm{aa}_{i.j+1}\mathrm{ba}_{i.j+1}\right)=$$

$$\mathrm{cons}_{i.j}\left(\left(\mathrm{ba}_{i.j}\,\middle|\,\mathrm{aa}_{i.j+1}\right),\ \left(\mathrm{aa}_{i.j}.\,\middle|\,\mathrm{ba}_{i.j+1}\right)\right)=$$

$$\mathrm{cons}\left(\begin{pmatrix}\mathrm{ba}_{i.j}\\ \mathrm{aa}_{i.j+1}\end{pmatrix},\begin{pmatrix}\mathrm{aa}_{i.j}\\ \mathrm{ba}_{i.j+1}\end{pmatrix}\right)=\mathrm{cons}\begin{pmatrix}\left(\mathrm{ba}_{i.j},\mathrm{aa}_{i.j}\right),\\ \left(\mathrm{aa}_{i.j+1},\mathrm{ba}_{i.j+1}\right)\end{pmatrix}=\quad,\ \left[\mathrm{bifunct}\right]$$

$$\begin{pmatrix}\mathrm{cons}_{i.j}\left(\mathrm{ba}_{i.j},\mathrm{aa}_{i.j}\right),\\ \mathrm{cons}_{i.j+1}\left(\mathrm{aa}_{i.j+1},\mathrm{ba}_{i.j+1}\right)\end{pmatrix}=$$

$$\begin{pmatrix}\left(\mathrm{ba}_{i.j}\ \ \mathrm{aa}_{i.j}\,\middle|\,\mathrm{ba}_{i.j}\mathrm{bb}_{i.j}\right)\\ \left(\mathrm{aa}_{i.j+1}\mathrm{ab}_{i.j+1}\,\middle|\,\mathrm{aa}_{i.j+1}\mathrm{ba}_{i.j+1}\right)\end{pmatrix}=\begin{pmatrix}\left(\mathrm{abbb}\right)_{i.j}\,\middle|\,\left(\mathrm{abaa}\right)_{i.j}\\ \left(\mathrm{aaab}\right)_{i.j+1}\,\middle|\,\left(\mathrm{aaba}\right)_{i.j+1}\end{pmatrix}.$$

$$\mathrm{LISP}:\ \mathrm{cons}\left(\mathrm{cdr}\left(\mathrm{aa}\,.\,\mathrm{ba}\right),\mathrm{car}\left(\mathrm{aa}\,.\,\mathrm{ba}\right)\right)=\mathrm{cons}\left(\left(\mathrm{ba}\right),\left(\mathrm{aa}\right)\right)=\left(\mathrm{ba}.\mathrm{aa}\right)$$

2.5. Bifunctoriality of CONS, CAR and CDR

2.5.1. Bifunctoriality for CONS

Bifunctoriality of polyLISP is a new feature of memristive interchangeability, i.e. parallelism of compositions. Hence, also for kenomic systems interchangeability of its operations is crucial.

$$\begin{pmatrix}\mathrm{list1}\\ \amalg\\ \mathrm{list3}\end{pmatrix}*\begin{pmatrix}\mathrm{list2}\\ \amalg\\ \mathrm{list4}\end{pmatrix}=\begin{pmatrix}\left(\mathrm{list1}\ *\ \ \mathrm{list2}\right)\\ \amalg\\ \left(\mathrm{list3}\ *\ \ \mathrm{list4}\right)\end{pmatrix}$$

$\mathrm{CONS}^{(m)}$ is retrograde recursive , bifunctorial and super – additive,
CDR and CAR are memristive enactive.

5. **cons**. cons [x; y] is defined for any x and y. We have
cons [e1; e2] = (e1 · e2). Thus
cons [X; A] = (X A)
cons [(X · A); Y] = ((X · A) Y).

$X = (A) \longrightarrow \mathrm{cons}\,[X;\, A] = (X\, A) \mid (X\, B)$
$X = (X.A) \longrightarrow \mathrm{cons}\,[(X \cdot A);\, Y] = ((X \cdot A)\, Y) \mid ((X \cdot A)\, Z).$

$$(1)\ \mathrm{cons}\left[x;\, y\right] \Rightarrow$$

$$\mathrm{CONS}^{(2)}\left[x^1;\, y^1 \,\middle|\, x^2;\, y^2\right] \Rightarrow \begin{pmatrix} \mathrm{cons}^1\left[x^1;\, y^1\right] \\ \amalg \\ \mathrm{cons}^2\left[x^2;\, y^2\right] \end{pmatrix} :$$

$$\mathrm{cons}\left[e1;\, e2\right] = \left(e1 \cdot e2\right) \Rightarrow$$

$$\mathrm{CONS}\left(e1\,.\,e2 \,\middle|\, e3\,.\,e4\right) = \begin{pmatrix} e1 \\ \amalg \\ e3 \end{pmatrix} * \begin{pmatrix} e2 \\ \amalg \\ e4 \end{pmatrix} = \begin{pmatrix} \left(e1 * e2\right) \\ \amalg \\ \left(e3 * e4\right) \end{pmatrix}.$$

$$\mathrm{CONS}^{(3)}\left[x^1;\, y^1 \,\middle|\, x^2;\, y^2 \,\middle|\, x^3;\, y^3\right].$$

$$\mathrm{cons}\left[X;\, A\right] = \left(X\, A\right)$$
$$\Rightarrow$$
$$\mathrm{CONS}^{(3)}\left[X^1;\, A^1 \,\middle|\, X^2;\, A^2 \,\middle|\, X^3;\, A^3\right]:$$

$$\mathrm{CONS}^{(3)}\left[X^1;\, A^1 \,\middle|\, X^2;\, A^2 \,\middle|\, X^3;\, A^3\right] =$$
$$\begin{pmatrix} \mathrm{cons}^1\left[X^1;\, A^1\right] \\ \mathrm{cons}^2\left[X^2;\, A^2\right] \\ \mathrm{cons}^3\left[X^3;\, A^3\right] \end{pmatrix} = \left(\begin{pmatrix} \left(X^1\, A^1\right) \\ \left(X^2\, A^2\right) \end{pmatrix}\right) = \left(\left(\begin{pmatrix} X^1 \\ X^2 \end{pmatrix}\right)\left(\begin{pmatrix} A^1 \\ A^2 \end{pmatrix}\right)\right)$$

$$\left(\begin{pmatrix} \left(X_1 \circ^{1.0}_{.0} A_1\right) \\ \amalg_{1.2}^{.0} \\ \left(X_2 \circ^{0.2}_{.0} A_2\right) \\ \amalg_{1.2}^{.3} \\ \left(X_3 \circ^{0.0}_{.3} A_3\right) \end{pmatrix}\right) = \left(\begin{pmatrix} X_1 \\ \amalg_{1.2}^{.0} \\ X_2 \\ \amalg_{1.2}^{.3} \\ X_3 \end{pmatrix}\right) \circ_1 \circ_2 \circ_3 \left(\begin{pmatrix} A_1 \\ \amalg_{1.2}^{.0} \\ A_2 \\ \amalg_{1.2}^{.3} \\ A_3 \end{pmatrix}\right)$$

$$(2)\quad \mathrm{cons}\left[\left(X \cdot A\right);\ Y\right] = \left(\left(X \cdot A\right) Y\right)$$

$$\Rightarrow$$

$$\mathrm{CONS}^{(3)}\left[\left(X^1 . A^1\right);\ Y^1 \,\middle|\, \left(X^2;\ A^2\right) Y^2 \,\middle|\, \left(X^3;\ A^3\right) Y^3\right]:$$

$$\mathrm{CONS}^{(3)}\left[\left(X^1 . A^1\right);\ Y^1 \,\middle|\, \left(X^2;\ A^2\right) Y^2 \,\middle|\, \left(X^3;\ A^3\right) Y^3\right]:$$

$$\begin{pmatrix} \mathrm{cons}^1\left[\left(X^1 . A^1\right);\ Y^1\right] \\ \mathrm{cons}^2\left[\left(X^2 . A^2\right);\ Y^2\right] \\ \mathrm{cons}^3\left[\left(X^3 . A^3\right);\ Y^3\right] \end{pmatrix} =$$

$$\left(\begin{pmatrix} \left(\left(X^1 . A^1\right) Y^1\right) \\ \left(X^2 . A^2\right) Y^2 \\ \left(X^3 . A^3\right) Y^3 \end{pmatrix}\right) = \left(\left(\begin{pmatrix} \left(X^1 . A^1\right) \\ \left(X^2 . A^2\right) \\ \left(X^3 . A^3\right) \end{pmatrix}\right)\left(\begin{pmatrix} Y^1 \\ Y^2 \\ Y^3 \end{pmatrix}\right)\right).$$

$$\left(\begin{pmatrix} \left(\left(X^1 . A^1\right) \circ^{1.0}_{.0} Y^1\right) \\ \left(\left(X^2 . A^2\right) \circ^{0.2}_{.0} Y^2\right) \\ \left(\left(X^3 . A^3\right) \circ^{0.0}_{.3} Y^3\right) \end{pmatrix}\right) = \left(\begin{pmatrix} \left(X^1 . A^1\right) \\ \left(X^2 . A^2\right) \\ \left(X^3 . A^3\right) \end{pmatrix} \circ_1 \circ_2 \circ_3 \begin{pmatrix} \left(Y^1\right) \\ \left(Y^2\right) \\ Y^3 \end{pmatrix}\right)$$

2.5.2. Bifunctoriality for enactional CAR and CDR

Enactional CAR

$$\mathrm{CAR}_{\mathrm{EN}}\left[\left(X_{i.j} . A_{i.j}\right)\right] = X_{i.j} . \left(\perp_{i.j}\right) \,\middle|\, A_{i.j+1}$$

$$\mathrm{CAR}_{\mathrm{EN}}\left[\left(X_{i.j} . A_{i.j}\right) \cdot Y_{i.j}\right] = \left(\left(X_{i.j} . A_{i.j}\right) . \left(\perp_{i.j}\right)\right) \,\middle|\, Y_{i.j+1}).$$

$$\mathrm{CAR}_{\mathrm{EN}}\left[\left(X_{i.j} . A_{i.j}\right)\right] = \begin{pmatrix} X_{i.j} \\ \square_{1.2} \\ A_{i.j+1} \end{pmatrix}:$$

$$\begin{pmatrix} X_{i.j} . \left(\perp_{i.j}\right) \\ \square_{1.2} \\ A_{i.j+1} . \mathrm{NIL} \end{pmatrix} = \begin{pmatrix} X_{i.j} \\ \square_{1.2} \\ A_{i.j+1} \end{pmatrix} \circ_1 \circ_2 \begin{pmatrix} \left(\perp_{i.j}\right) \\ \square_{1.2} \\ \mathrm{NIL} \end{pmatrix}.$$

$$\mathrm{CAR}_{\mathrm{EN}}\left[\left(\mathrm{X}_{\mathrm{i.j}}.\mathrm{A}_{i.j}\right)\cdot \mathrm{Y}_{i.j}\right] = \begin{pmatrix} \left(\mathrm{X}_{\mathrm{i.j}}.\mathrm{A}_{i.j}\right) \\ \square_{1.2} \\ \mathrm{Y}_{i.j+1} \end{pmatrix}:$$

$$\begin{pmatrix} \left(\mathrm{X}_{\mathrm{i.j}}.\mathrm{A}_{i.j}\right).\left(\perp_{i.j}\right) \\ \square_{1.2} \\ \mathrm{Y}_{i.j+1}.\mathrm{NIL} \end{pmatrix} = \begin{pmatrix} \left(\mathrm{X}_{\mathrm{i.j}}.\mathrm{A}_{i.j}\right) \\ \square_{1.2} \\ \mathrm{Y}_{i.j+1} \end{pmatrix} {}^{\circ}1\,{}^{\circ}2 \begin{pmatrix} \left(\perp_{i.j}\right) \\ \square_{1.2} \\ \mathrm{NIL} \end{pmatrix}.$$

Enactional CAR

$$\mathrm{CAR}_{\mathrm{EN}}\left[\left(\mathrm{X}_{\mathrm{i.j}}.\mathrm{A}_{i.j}\right)\right] = \begin{pmatrix} \mathrm{X}_{i.j} \\ \square_{1.2} \\ \mathrm{A}_{i.j+1} \end{pmatrix}$$

$$\mathrm{CAR}_{\mathrm{EN}}\left[\left(\mathrm{X}_{\mathrm{i.j}}.\mathrm{A}_{i.j}\right)\cdot \mathrm{Y}_{i.j}\right] = \begin{pmatrix} \left(\mathrm{X}_{\mathrm{i.j}}.\mathrm{A}_{i.j}\right) \\ \square_{1.2} \\ \mathrm{Y}_{i.j+1} \end{pmatrix}$$

Enactional CDR

$$\mathrm{CDR}_{\mathrm{EN}}\left[\left(\mathrm{X}_{\mathrm{i.j}}.\mathrm{A}_{i.j}\right)\right] = \left(\perp_{i.j}\right).\mathrm{A}_{i.j}\Big|\mathrm{X}_{i.j+1}$$

$$\mathrm{CDR}_{\mathrm{EN}}\left[\left(\left(\mathrm{X}_{\mathrm{i.j}}.\mathrm{A}_{i.j}\right)\cdot \mathrm{Y}_{i.j}\right)\right] = \left(\perp_{i.j}\right).\mathrm{Y}_{i.j}\Big|\left(\mathrm{X}_{\mathrm{i.j+1}}.\mathrm{A}_{i.j+1}\right).$$

Enactional CDR

$$\mathrm{CDR}_{\mathrm{EN}}\left[\left(\mathrm{X}_{\mathrm{i.j}}.\mathrm{A}_{i.j}\right)\right] = \begin{pmatrix} \mathrm{A}_{i.j} \\ \square_{1.2} \\ \mathrm{X}_{i.j+1} \end{pmatrix}$$

$$\mathrm{CDR}_{\mathrm{EN}}\left[\left(\mathrm{X}_{\mathrm{i.j}}.\mathrm{A}_{i.j}\right)\cdot \mathrm{Y}_{i.j}\right] = \begin{pmatrix} \left(\mathrm{Y}_{\mathrm{i.j}}\right) \\ \square_{1.2} \\ \left(\mathrm{X}_{\mathrm{i.j+1}}.\mathrm{A}_{i.j+1}\right) \end{pmatrix}$$

With this connection to bifunctoriality established, the whole elaborted apparatus of polycontextural interchangeability of operations might be applied to develop a complex memristive polyLISP.

2.5.3. Typical cases of interchangeability of disseminated operations

Transpositional composition

$$\begin{pmatrix} CAR_1 \\ \amalg_{1.2} \\ CAR_2 \diamond_{2.1} CAR_1 \\ \amalg_{2.3} \\ CAR_3 \diamond_{3.1} CAR_1 \end{pmatrix} \begin{bmatrix} \circ_{1.1} \\ \\ \circ_{2.1} \circ_{2.2} \\ \\ \circ_{3.1} \circ_{3.3} \end{bmatrix} \begin{pmatrix} CDR_1 \\ \amalg_{1.2} \\ CDR_2 \diamond_{2.1} CDR_1 \\ \amalg_{2.3} \\ CDR_3 \diamond_{3.1} CDR_1 \end{pmatrix} =$$

$$\begin{pmatrix} (CAR_1 \circ_{1.1} CDR_1) \\ \amalg_{1.2} \\ (CAR_2 \circ_{2.2} CDR_2) \diamond_{2.1} (CAR_1 \circ_{2.1} CDR_1) \\ \amalg_{2.3} \\ (CAR_3 \circ_{3.3} CDR_3) \diamond_{3.1} (CAR_1 \circ_{3.1} CDR_1) \end{pmatrix}$$

Matrix model

[bif, id, id]	O_1	O_2	O_3
M_1	$LISP_{1.1}$	$LISP_{2.1}$	$LISP_{3.1}$
M_2	–	$LISP_{2.2}$	–
M_3	–	–	$LISP_{3.3}$

Reflectional interchangeability

$$\begin{pmatrix} CONS_1 \square_{1.2} CONS_1 \square_{1.3} CONS_1 \\ \amalg_{1.2} \\ CONS_2 \\ \amalg_{2.3} \\ CONS_3 \end{pmatrix} \begin{bmatrix} \circ_{1.i} \\ \circ_{2.2} \\ \\ \circ_{3.3} \end{bmatrix} \begin{pmatrix} APPEND_1 \square_{1.2} APPEND_1 \square_{1.3} APPEND_1 \\ \amalg_{1.2} \\ APPEND_2 \\ \amalg_{2.3} \\ APPEND_3 \end{pmatrix}$$

=

$$\begin{pmatrix} ((CONS_1 \circ_{1.1} APPEND_1) \square_{1.2} (CONS_1 \circ_{1.2} APPEND_1)) \square_{1.3} (CONS_1 \circ_{1.3} APPEND_1) \\ \amalg_{1.2} \\ (CONS_2 \circ_{2.2} APPEND_2) \\ \amalg_{2.3} \\ (CONS_3 \circ_{3.3} APPEND_3) \end{pmatrix}$$

Matrix model

[repl, id, id]	O_1	O_2	O_3
M_1	$LISP_{1.1}$	–	–
M_2	$LISP_{1.2}$	$LISP_{2.2}$	–
M_3	$LISP_{1.3}$	–	$LISP_{3.3}$

2.5.4. QUOTE and EVAL

"Quote is a a one-argument operation that stops the evaluation process before it reaches its argument." (Stark)

QUOTE data-expression ⟶ data-expression.

QUOTE, again, is involved into iterability, and therefore the possibility to distinguish between iterative and accretive quotation is accessible.

$$QUOTE_{ACCR}: data-expression_{i.j} \longrightarrow data-expression_{i.j+1}$$

$$QUOTE_{ITER}: data-expression_{i.j} \longrightarrow data-expression_{i.j}$$

Polycontextural QUOTE
"In a polycontextural setting we are free to choose a more flexible interplay between quotation and interpretation. To quote means to put the quoted sentence on a higher level of a reflectional order or to another heterachical actional level. We are not forced to limit ourselves to any kind of the well known intra-contextural meta-language hierarchies. Those local procedures are nevertheless not excluded at all but localized to their internal place.
To start the argumentation I simply map an index *i* to the sentences. A quotation is augmenting and an interpretation (evaluation) is reducing its value, say by 1."
http://www.thinkartlab.com/pkl/lola/Godel_Games-short.pdf
http://sds.podval.org/self-ref.html

2.6. Memristive self-referentiality

2.6.1. Self in LISP

One of the most striking properties of LISP is its ability for self-referential definition crucial for the definition of recursion and a whole trend of AI programming.

Under the title "*Self-processing*", Stark writes:

> "LISP's ability to process itself is a direct consequence of (1) the representation of the language in its data structure, (2) the simple algebraic syntax, and (3) the presence of functions such as QUOTE, DEFUN, FUNCTION, EVAL, and APPLY." (W. Richard Stark, LISP, Lore, and Logic, 1990, p. 92)

Those features are naturally implemented in the paradigm of kenomic LISP. The function "QUOTE" and "EVAL" might be leading to the accessibility of new self-referential constructions. (cf. Godel Games)

But with the development of kenomic definitions of the basic operations of LISP, CDR, CDR and CONS even more direct constructions of self-referentiality are in sight.

In fact, the kenomic retrograde recursivity of the basic operators is uncovered as a fundamentally self-referential notion and construction.

As a consequence, the neat reflectional construction of self-referentiality proposed in my small paper *"Gödel's Games"* gets a more direct realization on an even more profound level of the very understanding of iterability itself.

An important advantage of this kenomic concept of self-referentiality is the fact that it is stucturally determined in a way that avoids all those annoying misreadings and misunderstandings of the term "self" in all those self-referential configurations and speculations.

On the base of such fundamental properties of self-referentiality, constructions like *enaction* are supporting further aspects of operational self-applications.

2.6.2. Fundamental theorem for pure LISP

Fundamental theorem for pure LISP.
"Every algorithmically computable (in the informal sense) function can be computed by a program in pure LISP."

The new question is, can every memristive function of polyLISP be computed by a program in pure LISP?
In other words, is there a *reduction* mechanism which is able to reduce polyLISP to pure LISP?
What exactly would it mean if kenomic LISP wouldn't be reducible to pure LISP?

There is always a possibility of *simulating* a new formalism in terms of a traditional formalism. But again, simulations don't become realization.

List of translations	**Simulations**
Kenom to atomic symbol	Equivalence class of different symbols
Morphogram to list	Equivalence class of lists
Retrogradness to iterativity	Double recursion with conditions
Enaction to elimination	Elimination plus appending
Poly- to monocontexturality	Multi-sets of lists
Polyverse to universe	Multi-sorted domains
Super-additivity to composition	augmented composition
Polycontextural logic to logic	Multi-valued or modal logics

and so on.

3. Monomorphic Lisp

3.1. General

(aba) = (abba)
a=b, (aa) ≠ (aaa)

3.2. Concatenational

a≠b, (ab) = (ba), (aa) = (a)

comp: [(aa) (b) (cc)] ⟶ [aabcc]

decomp: [aabcc] ⟶ [(aa) (b) (cc)]

3.3. Metamorphical

(aba) = (abba)

4. Diamond LISP

According to the quadralectics of diamond strategies, the simple oriented approach to a kenomic LISP might be distributed and located into the qaudralectics of distinctions.

From the circularity of a list to a chiastic resolution of self-referentiality.

5. Applications

5.1. Ambiguity, double-meaning and morphograms

> "Ambiguity *is the wild child of language interpretation. Whether from the point of view of the philosopher, linguist, psychologist, lexicographer, or computer scientist, ambiguity problems have relentlessly resisted taming.*
> *Lexical ambiguity, or polysemy, arises when a word, or a phrase, is associated in the language system with more than one meaning.* Generativity *refers to the notion that words seem to be able to be used in new and creative ways, reflecting the generative power of language, but at the lexical level."* (Judith Klavans)
> http://www.aaai.org/Papers/Symposia/Spring/1995/SS-95-01/SS95-01-001.pdf
> http://193.6.132.75/honlap/whatispolysemy.pdf

A simple application "Fruit flies like a banana."
What's the meaning of an ambiguous sentence like *"Fruit flies like a banana."*?

As it is well known, the sentence has, at least, two meanings:
1. "the insects called fruit flies are positively disposed towards bananas."
2. "Something called fruit is capable of the same type of trajectory as a banana."
"*These two potential meanings are partly based on the (at least) two ways in which the phrase can be parsed.*"
(Alan P. Parkes, Introduction to Languages, Machines and Logic, 2002, p. 42)

Ambiguity in languages is reflected in the existence of more than one *parse tree* for one sentence of that language.

There are two strategies to deal with ambiguity:
1. Disambiguation and
2. Mediation and Bisimulation.

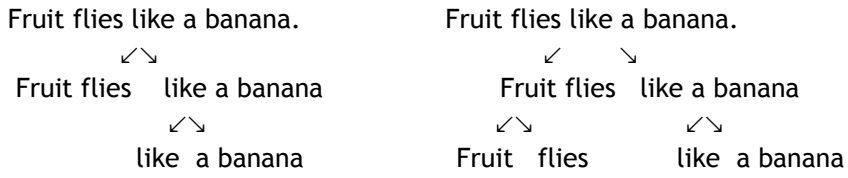
The decision necessary for the purpose of formal languages and computation is *disambiguation*. Disambiguation is eliminating polysemy and ambiguity of sentences.
Hence, both meanings of the sentence might be chosen and used but separately or just one meaning might be involved in further steps of reasoning, depending on a further context.

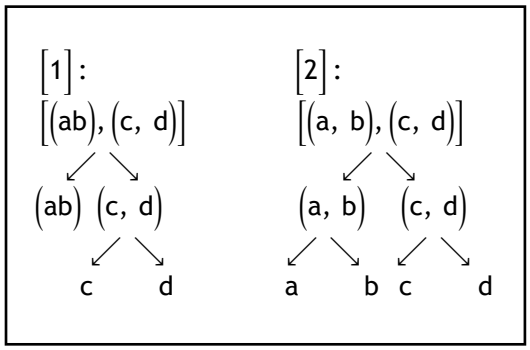
On the other hand, the strategy of *mediation* is supporting the polysemy of ambiguous sentences. As the example above is set, there is no context which could help to separate the meanings and to select one meaning only as prior to the other.

Hence, the sentence as such has both meanings at once. Therefore its logical status is demanding for three and not only for two logical places to realize its polysemic ambiguity. Two loci for the separated meanings are necessary and one more for the sentence as such having both meanings at once. This third position is not reducible to a syntactical choice to contrast the semantics of the sentence because the sentence also has syntactically two disjunct parse trees. But more important, the focus of the understanding of the ambiguos sentence is on polysemy, i.e. on the semantic ambiguity of its meaning. A further step will show that the purely semantic approach is not offering a possibility for a 're-solution' of the ambiguity problem of polysemic sentences. What is needed additionally is a pragmatic approach, here formalized with application of a morphogrammatic approach.

Thus, the third position, which places the double meaning of the sentence, shall be inscribed as the morphogram of the double sentence. A morphogram might be understood as an inscription of pre-logical 'meaning'. It therefore has to be able to deal consistently with semantic ambiguity without running into logical contradictions.

Binary tree analysis





$$[1]: \text{cons}((ab), \text{cons}(c, d)) = \text{cons}((ab), cd)) = ((ab)\,cd)$$

$$[2]: \text{cons}(\text{cons}((a, b), \text{cons}(c, d)) = \text{cons}((ab), (cd)) = (abcd).$$

$$\text{MED}\left(\left[1\right],\ \left[2\right]\right) = \begin{pmatrix} \text{cons}\left(\left(\text{ab}\right), \text{cons}\left(\text{c, d}\right)\right) \\ \text{II} \\ \text{cons}\left(\text{cons}\left(\left(\text{a, b}\right), \text{cons}\left(\text{c, d}\right)\right)\right) \end{pmatrix} = \begin{pmatrix} \left(\left(\text{ab}\right)\text{cd}\right) \\ \text{II} \\ \left(\text{abcd}\right) \end{pmatrix}$$

$$\left[3\right]: \left(\left\langle \text{ab}\right\rangle \text{cd}\right) \equiv_{\text{BISM}} \left(\text{abcd}\right) \Rightarrow$$

$$\begin{pmatrix} \left(\left\langle \text{ab}\right\rangle \text{cd}\right) & \longrightarrow & \left(\text{a, b, c, d}\right) \\ \updownarrow & \text{X} & \updownarrow \\ \left(\text{abcd}\right) & \longrightarrow & \left(\text{a, b, c, d}\right) \end{pmatrix}$$

Fusion

$\text{fus}\left(\text{a, b}\right) = \left\langle \text{ab}\right\rangle \Rightarrow \text{de} - \text{fus}\left(\left\langle \text{ab}\right\rangle\right) = \left(\text{ab}\right),$

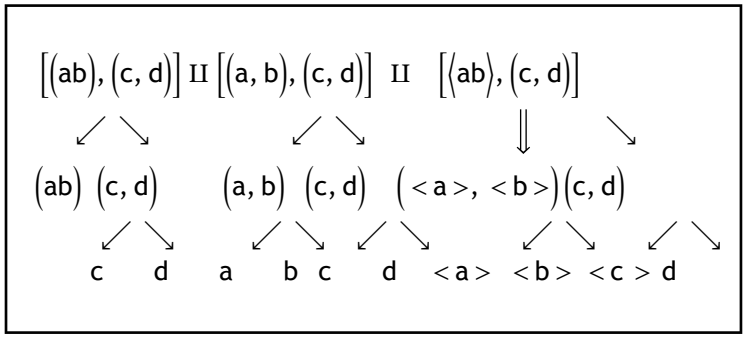
$\text{de} - \text{fus}\left(\left\langle \text{ab}\right\rangle\right) = \left(\text{car}\left\langle \text{ab}\right\rangle,\ \text{cdr}\left\langle \text{ab}\right\rangle\right) = \left(\text{ab}\right)$

$\text{cons}\left(\text{a, b}\right) = \left(\text{ab}\right)$

$\text{de} - \text{cons}\left(\text{ab}\right) = \left(\text{a, b}\right)$

$\text{de} - \text{cons}\left(\text{ab}\right) = \left(\text{car}\left(\text{ab}\right),\ \text{cdr}\left(\text{ab}\right)\right)$

Null



The operation of de/melting (or fusing together) is not anymore a formal operation in the sense of the lambda calculus or LISP. For both, the lambda calculus and LISP, the construct " 〈 ab〉 " with its 'composed' meaning "Fruit flies" is a singular term, i.e. a name for an ontological entity covered by the taxonomy of animals. Nevertheless, this singular name is synthesized, fused together, by two domains, the domain of fruits and the domain of flies. This enables a specific decomposition which is not part of the sentence as itself and its double meaning.

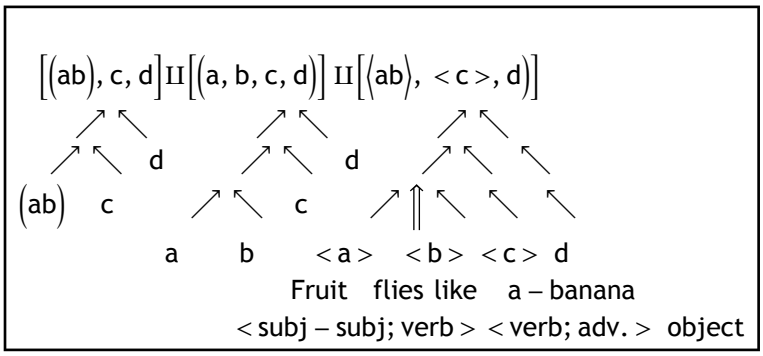
Hence, a decomposition of such a term needs a new abstraction, 〈ab〉, paired with a new operator "de/fus".

The result of the exercise shows a mediation of both meanings of the sentence and a kind of a morphogrammatic deep-structure of the double-meaning of the sentence as its 'meaning' as such. This deep-structure is 'unifying' the two observer depending readings of the sentence into an observer-independent inscription of its double-meaning, as its morphogram.

Left-associated tree analysis

Fruit flies like a banana.
↗↖ a banana
↗↖ like
Fruit-flies

Fruit flies like a banana.
↗↖ a-banana
↗↖ like
↗↖flies
Fruit



$$\text{cons}\left(\text{cons}\left(\langle ab\rangle,\ c\right),\ d\right) = \left[\langle ab\rangle\, cd\right]$$

$$\text{cons}\left(\text{cons}\left(\text{cons}\left(a, b\right),\ c,\ d\right)\right) = \left[(abcd)\right]$$

$$\text{cons}\left(\text{cons}\left(\text{cons}_{\text{planar}}\left(< a >,\ < b >\right),\ < c >,\ d\right)\right) =$$

$$\left[\begin{pmatrix}(ab)\\ \langle ab\rangle\end{pmatrix}\begin{pmatrix}(c)\\ \langle c\rangle\end{pmatrix}\begin{pmatrix}(d)\\ (d)\end{pmatrix}\right].$$

$$\text{cons}_{\text{planar}}\left(< a >,\ < b >\right) = \begin{pmatrix}(ab)\\ \langle ab\rangle\end{pmatrix},\ < c > = \begin{pmatrix}(c)\\ \langle c\rangle\end{pmatrix}.$$

Both sentences are analyzed by a time-linear principle of possible continuations (Hausser). In contrast, the third analysis is breaking in some respect the time-linearity and is introducing a **planar** extension of the ambiguous terms. Such planar constructions which are holding conflicting and antinomic terms together are covered by morphograms. Lists are not prepared to cover ambiguous terms because they are defined by atomic terms and linear constructions.