— vordenker-archive —

Rudolf Kaehr
(1942-2016)

Title

Memristive Stack Machines
Based on retrograde recursivity and distinctive enaction

Archive-Number / Categories

3_03 / K11

Publication Date

2011

Keywords

TOPICS: Memristivity and recursivity – Different approaches to formalize memristivity – CI and stacks – Computational Stack – Simple Enactional STACK – Tabular memristive stacks / Polycontextural matrix / Quadralectic memristive stacks

Disciplines

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

Abstract

The idea of a memristive STACK machine based on the retrograde recursivity of its data structure and on the construct of enaction for its operators shall be sketched in a descriptive and semi-formal manner.

Citation Information / How to cite

Rudolf Kaehr: "Memristive Stack Machines", www.vordenker.de (Sommer Edition, 2017) J. Paul (Ed.),
URL: http://www.vordenker.de/rk/rk_Memristive-Stack-Machines_2011.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

Memristive Stack Machines

Based on retrograde recursivity and distinctive enaction

Rudolf Kaehr Dr.phil
Copyright ThinkArt Lab ISSN 2041-4358

Abstract

The idea of a memrsitive STACK machine based on the retrograde recursivity of its data structure and on the construct of enaction for its operators shall be sketched in a descriptive and semi-formal manner.

1. Memristivity and recursivity

1.1. Different approaches to formalize memristivity

Up to now, it seems, that there are five different approaches available to deal formally with aspects memristivity in a formal and operative manner.

The focus of this exercise shall be oriented on enactional approach with some kenogrammatic features only. *Enaction*, additionally to the *retrogradeness* of kenomic recursivity, is an interesting new property of memristive formalisms to be studied.

Because the discovery of memristivity in nano-electronics by Leon Chua 1971 and its realization by the team of Stanley Williams at HP in 2008 is very recent and not yet studied formally from a non-electronic and system-theoretic point of view, this study remains still in a very experimental and temporary status of reflection and elaboration.

The graphematic possibilities of studying memristivity in formal systems at hand for now are:

1. the mode of *semiotic* identity with recursivity,
2. the mode of *contextural* comlexity with proemiality,
3. the mode of *kenogrammatic* similarity with retrogradeness,
4. the *indicational* mode of “topology-free constellations of signs” with

enaction, and
5. the mode of *monomorphic* bisimilarity of morphogrammatics with bisimulation.

Other modes are possible as further realizations of graphematic styles of inscriptions.

Every symbolization system entails its own paradigm of programming languages.

Properties

Semiotics a=a, a≠b, a(bc) = (ab)c
The semiotic or *symbolic* mode of thematization is ideal for atomistic binary physical systems as they occur as digital computers.

Polycontexturality (ab) = ((ab)c), (ac)(bd) = (ab)(cd)
The *contextural* or interactional mode of thematization is ideal for ambigous complex physical systems as they occurs in distributed and interacting digital computer systems.

Kenogrammatics a=b, (aa) ≠ (ab), (ab) = (ab)|(ac)
The *kenogrammatical* mode of thematization is ideal for pre-semiotic complex behavioural systems as it occurs in memristive physical systems.

Calculus of Indication a=a, a≠b, ab = ba
The *indicational* mode of thematization is ideal for singular decision systems as they occur in simple action systems.

Morphogrammatics a=b, (aa) ≠ (aaa), (aba) = (abba)
The *monomorphic* mode of thematization is ideal for metamorphic systems as they occur in complex memristive actional systems.

Some properties might be collected temporarily in the table:

styles	***semiotic***	***contextural***	***kenogrammatic***	***indicational***	***m***
recursive	+	+	+	+	
retrograde	−	+	+	−	
enaction	−	−	−	+	
metamorph	−	+	−	−	
super − add	−	+	+	−	

Recursive functions are memory intensive. It might be possible to re-design the mechanisms of recursivity in comutational systems with the help of a memristive thematization of the very basic properties of recursivity.

Further characterizations

In analogy to the operation LIST on objects:
LIST: a, b, c ⟶ (abc)
LIST: (ab), a, b, c ⟶ ((ab)abc))
we shall define a general thematization function or operation THEMATH which is interpreting a proposed ”set” or “agglomeration” of objects as *semiotical*, *contextural*, *kenogrammatical*, *indicational* or *monomorphical*:
THEMATH : a, a, b,

$$c \longrightarrow \begin{cases} \text{LIST}(a, a, b, c) \longrightarrow (a\,a\,b\,c) \\ \text{CONTEXTURE}(a, a, b, c) \longrightarrow (((a \square a) \amalg d)(c \amalg e) \amalg f) \\ \text{KENOS}(a, a, b, c) \longrightarrow [a\,b\,c] \\ \text{INDIC}(a, a, b, c) \longrightarrow \langle a\,b \rangle \\ \text{MORPHIC}(a, a, b, c) \longrightarrow [[a\,a][b][c]] \end{cases}$$

$$\text{CONTEXTURE}(a, b, c, d) \longrightarrow (((a \square b) \amalg d)(c \amalg e) \amalg f) :$$

$$\begin{pmatrix} a & \amalg & b & \amalg & d \\ - & c & \amalg & e & - \\ - & - & f & - & - \end{pmatrix} = ((a\,b\,d)(c\,e)f)$$

$$\text{CONTEXTURE}(a, a, b, c) \longrightarrow (((a \square a) \amalg d)(c \amalg e) \amalg f) :$$

$$\begin{pmatrix} a & \square & - & \amalg & d \\ a & c & \amalg & e & - \\ - & - & f & - & - \end{pmatrix} = (((a\,a\,d)(c\,e)f)$$

1.2. CI and stacks

Following Wolfram's statement, according to M. Schreiber:
"A kind of form is all you need to compute. A system can emulate rule 110 if it can distinguish: *More than one is one but one inside one is none*.
Simple distinctions can be configured into forms which are able to perform universal computations."

Applied to one of the simplest models of computing, the STACK, we get a distinctional stack model. This observation corresponds properly with Wolfram/Schreiber's statement.

What is still missing are the *memristive* properties. Memristivity enters the game with an *enactional* interpretation of the operation "Pop". But this makes sense only in the framework of a disseminated, i.e. polycontextural stack model
Applied to the simplest model of computing, the STACK, we get a distinctional stack model.

What is still missing are the memristive properties.
Memristivity enters the game with an enactional interpretation of the operation "*pop*".
But this makes sense only in the framework of a disseminated, i.e. polycontextural stack model.

'Keller' machines which are remembering their 'kellert' (cancelled) states. Or: Register machines which are registering their cancelled states.

Connection with the Calculus of Indication (CI):
Interpretation

$$"\textit{More than one is one}" : \{\}\{\} \rightleftharpoons \{\}$$

$$"\textit{one inside one is none}" : \{\{\}\} \rightleftharpoons \text{ø},$$

$$"\text{is}" : \rightleftharpoons$$

Stack operations

$$J1 : \{\} \longrightarrow \{\}\{\} : \Longrightarrow \text{push}, \quad (\text{dup})$$

$$J2 : \{\{\}\} \longrightarrow \varnothing : \Longrightarrow \text{pop}, \quad (\text{drop})$$

Basic operations for STACK
push (a – - aa)
pop (a - –)

Distinction model of STACK
push $\{\} \rightarrow \{\}\{\}$
pop $\{\{\}\} \rightarrow \emptyset$

Memristic STACK
push $\{\}_i \rightarrow \{\}_i\{\}_i$
pop $\{\{\}_{i.i}\}_{i.i} \longrightarrow \begin{pmatrix} \emptyset_{i.i} \\ \{\}_{i.i+1} \end{pmatrix}$

Morphic STACK
push $\{\}_i \rightarrow \{\}_i\{\}_i \mid \{\}_i\{\}_{i+1}$
pop $\{\{\}_{i.i}\}_{i.i} \longrightarrow \begin{pmatrix} \emptyset_{i.i} \\ \{\}_{i.i+1} \end{pmatrix}$

With *pop* a memristic function shall be implemented with the application of the enaction operator:

$$\text{pop}\ (\ a - \emptyset\ ) \Rightarrow \text{pop}(\ a_{1.1} \longrightarrow \begin{pmatrix} \emptyset_{1.1} \\ a_{1.2} \end{pmatrix}).$$

This *pop* operation is emptying the stack "$q_{1.1}$" from its symbol "$a_{1.1}$" and is pushing, at the same time, the symbol $a_{1.1}$ onto another reflectional stack "$q_{1.2}$" as the symbol "$a_{1.2}$".
Hence, enaction is a composition of an *elimination* step (popping, emptying, reading) and a transitional step of *pushing* (writing) the data onto another neighboring stack system.

With "$\urcorner$" for "{ }" we get:

$$\text{pop}\ (\ \urcorner - \emptyset) = (\ \overline{\urcorner_{1.1}}\rceil_{1.1} \longrightarrow \emptyset),$$

$$(\ \overline{\urcorner_{1.1}}\rceil_{1.1} - \emptyset) \longrightarrow (\ \begin{matrix} \emptyset_{1.1} \\ \urcorner_{1.2} \end{matrix}\ ).$$

Retrogradeness of “push"
The case of a Morphic STACK with a retrograde definition of the push operation is not considered at this place. Retrogradeness is involved with additional operations, say ADD, but not yet for “push”. A system with enactional “pop” and retrograde “push” is defined for the mix of indicational and kenomic formalisms. The operation “push” belongs to the repeatability of events and is therefore involved with retrograde recursivity. Hence the concatenational “push” with “push (a – aa)" becomes “push: X = (a) ⟶ Xa | Xb".

Why stack machines?
What happens with a tabular organization of a stack? The tabular matrix is supporting the distribution of contextural and morphogrammatic-based distributions of stack machines.

"In computer science, a stack machine is a model of computation in which the computer's memory takes the form of one or more stacks.” (WiKi)

A similar exercise with LISP will be published soon.

1.3. Computational Stack

1.3.1. Concept of a STACK

STACK as a category:
Following Axel Poigné (LNCS 240, 1985, p. 107):

"Let T be the category of terms T_{STACK} generated by the signature
sig STACK is
sorts *nat*, *stack*
ops o ⟶ *nat*, suc : *nat* ⟶ *nat*
empty : ⟶ *stack*
push : *stack nat* ⟶ *stack*
pop : *stack* ⟶ *stack*
top : *stack* ⟶ *nat*

There are two atomic predicates
eq_{nat} : *nat* x *nat*, eq_{stack} : *stack* x *stack*.

The axioms are specified in the usual logical notation :

$$\mathrm{tt} \vdash \mathrm{eq}_{\mathrm{nat}}\left(0, 0\right),\ \mathrm{tt} \vdash \mathrm{eq}_{\mathrm{stack}}\left(\mathit{empty}, \mathit{empty}\right)$$

$$\mathrm{eq}_{\mathrm{nat}}\left(\mathrm{m}, \mathrm{n}\right) \vdash \mathrm{eq}_{\mathrm{nat}}\left(\mathrm{suc}\left(\mathrm{m}\right), \mathrm{suc}\left(\mathrm{n}\right)\right)$$

$$\mathrm{eq}_{\mathrm{stack}}\left(\mathrm{x}, \mathrm{y}\right) \wedge \mathrm{eq}_{\mathrm{nat}}\left(\mathrm{m}, \mathrm{n}\right) \vdash \mathrm{eq}_{\mathrm{stack}}\left(\mathrm{push}\left(\mathrm{x}, \mathrm{m}\right), \mathrm{push}\left(\mathrm{y}, \mathrm{n}\right)\right)$$

$$\mathrm{eq}_{\mathrm{stack}}\left(\mathrm{x}, \mathrm{x}\right) \wedge \mathrm{eq}_{\mathrm{nat}}\left(\mathrm{m}, \mathrm{m}\right) \vdash \mathrm{eq}_{\mathrm{stack}}\left(\mathrm{pop}\left(\mathrm{push}\left(\mathrm{x}, \mathrm{m}\right)\right), \mathrm{x}\right)$$

$$\mathrm{eq}_{\mathrm{stack}}\left(\mathrm{x}, \mathrm{x}\right) \wedge \mathrm{eq}_{\mathrm{nat}}\left(\mathrm{m}, \mathrm{m}\right) \vdash \mathrm{eq}_{\mathrm{nat}}\left(\mathrm{top}\left(\mathrm{push}\left(\mathrm{x}, \mathrm{m}\right)\right), \mathrm{m}\right) \bullet$$

For **example**, the basic Forth stack operators are described as:

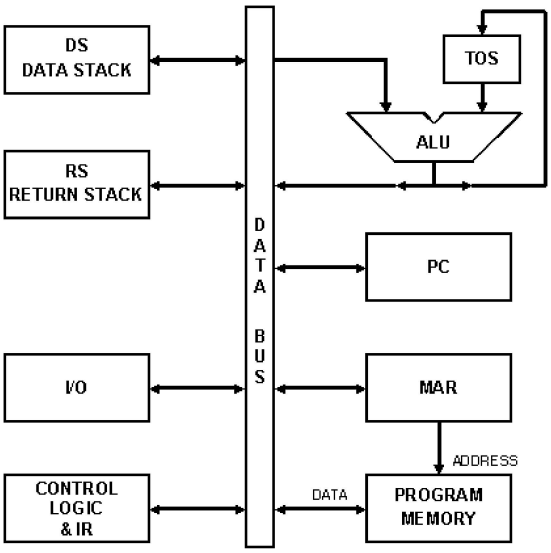
```
      ( before -- after )
dup  ( a -- a a )
drop ( a -- )
swap ( a b -- b a )
over ( a b -- a b a )
rot  ( a b c -- b c a )
```

The main operations of a stack machine are PUSH and POP, also called DUP and DROP for FORTH.

"The two operations applicable to all stacks are:

- a *push* operation, in which a data item is placed at the location pointed to by the stack pointer, and the address in the stack pointer is adjusted by the size of the data item;
- a *pop* or pull operation: a data item at the current location pointed to by the stack pointer is removed, and the stack pointer is adjusted by the size of the data item."

1.3.2. Stack machine



http://www.ece.cmu.edu/~koopman/stack_computers/sec3_2.html
http://www.dcs.gla.ac.uk/~marks/thesis.pdf

Morphogrammatic interpretation
DATA/RETURN STACK:
The data of a memristive stack machines are in fact morphograms. A distinction-theoretic option with an application of the Calculus of Indication is preferable for reaons of introduction.

ALU: Morhogrammatics
The arithmetical and logical operations for a memristive stack are defined accoring the struture of memristive objects. Hence, retrograde recursivenes and enaction of morphograms has to become the guiding paradigm for a memristive stack machine.

CONTROL LOGIC: Polycontexturality
The control logic for polycontextural memristive stack machines is ruled, certainly, by a polycontextural logic which is surpassing the limits of non-distributed classical logics. Hence, any contextural place of a memrsitive stack gets its own logic, and arithmetics too.

Hence, because of its polycontextural definition, a memrsitive stack machine is not simply a kind of a multi-stack machine but a system of mediation of stack machines.

Instructions

Some typical stack *instructions* for the classical case of a stack machine (and Forth).

Instruction	Data Stack input -> output	Function
!	N1 ADDR ->	*Store* N1 at location ADDR in program memory
+	N1 N2 -> N3	*Add* N1 and N2, giving sum N3
-	N1 N2 -> N3	*Subtract* N2 from N1, giving difference N3
>R	N1 ->	*Push* N1 onto the return stack
@	ADDR -> N1	*Fetch* the value at location ADDR in program memory, returning N1
AND	N1 N2 -> N3	*Perform* a bitwise AND on N1 and N2, giving result N3
DROP	N1 ->	*Drop* N1 from the stack
DUP	N1 -> N1 N1	*Duplicate* N1, returning a second copy of it on the stack

Example

Input	Operation	Stack
-	stack	⊥
1	Push operand	1
2	Push operand	2, 1
4	Push operand	4, 2, 1
*	Multiply	8, 1
+	Add	9
3	Push operand	3, 9
3	Pop operand	9
9	Pop operand	⊥ .

1.4. Simple Enactional STACK

1.4.1. Reflectional enaction

Preconditions
Retrograde recursivity of kenogrammatics and its laws of sameness of morphograms.
Enactional meristivity of reflectional and interactional operations in polycontextural configurations.

Strategy for the design of a memristive stack concept:
POP: enactional memristics, i.e. the operation POP is at once destructive and conservative.
PUSH: concatenation memristics (retrograde recursivity), i.e. the concatenational aspect of PUSH is reflecting its morphogrammatic design which is not atomistic but holistic.
Enactional POP

Reflectional enaction

$$\overline{\ulcorner}_{i.j}\,\big|_{i.j} \rightleftharpoons \begin{pmatrix} \phi_{i.j} \\ \ulcorner_{i.j+1} \end{pmatrix}$$

Iteration

$$\text{pop}\left(\ulcorner_{\text{i.j}} \longrightarrow \begin{pmatrix} \phi_{\text{i.j}} \\ \ulcorner_{\text{i.j+1}} \end{pmatrix}\right):$$

$$\text{pop}\left(\ulcorner_{\text{i.j}} \longrightarrow \overline{\ulcorner}_{i.j}\,\big|_{i.j}\right) \longrightarrow \begin{pmatrix} \phi_{\text{i.j}} \\ \ulcorner_{\text{i.j+1}} \end{pmatrix}$$

$$\text{pop}_{1.2}\left(\text{pop}_{1.1}\left(\overline{\ulcorner}_{1}\,\big|_{1}\right)\right) \longrightarrow \text{pop}_{1.2}\begin{pmatrix} \phi_{1.1} \\ \ulcorner_{1.2} \end{pmatrix} \longrightarrow \begin{pmatrix} \phi_{1.1} \\ \phi_{1.2} \\ \ulcorner_{1.3} \end{pmatrix}$$

Parallelism

$$\mathrm{pop}^{(2)}: \begin{pmatrix} \mathrm{pop}_{1.1}: \ulcorner\urcorner_{1.1} \longrightarrow \begin{pmatrix} \phi_{1.1} \\ \ulcorner\urcorner_{1.2} \end{pmatrix} \\ \mathrm{pop}_{2.2}: \ulcorner\urcorner_{2.2} \longrightarrow \begin{pmatrix} \phi_{2.2} \\ \ulcorner\urcorner_{2.3} \end{pmatrix} \end{pmatrix}$$

Matrix model

$$\mathrm{pop}^{(2)}: \begin{array}{|c|ccc|} \hline & O^1 & O^2 & O^3 \\ M^1 & \overline{\ulcorner\urcorner_{1.1}}_{1.1} & - & - \\ M^2 & - & \overline{\ulcorner\urcorner_{2.2}}_{2.2} & - \\ M^3 & - & - & - \\ \hline \end{array} \Longrightarrow \begin{array}{|c|ccc|} \hline & O^1 & O^2 & O^3 \\ M^1 & \varnothing_{1.1} & - & - \\ M^2 & \ulcorner\urcorner_{1.2} & \varnothing_{2.2} & - \\ M^3 & - & \ulcorner\urcorner_{2.3} & - \\ \hline \end{array}$$

Mathematical definition

Enactional STACK$^{(3,3)}$

sig STACK$^{(3,3)}$ is

sorts$^{(3,3)}$ $nat^{(3,3)}$, $stack^{(3,3)}$

ops$^{(3,3)}$ $o^{(3,3)} \longrightarrow nat^{(3,3)}$, $\mathrm{suc}^{(3,3)}: nat^{(3,3)} \longrightarrow nat^{(3,3)}$

$\mathrm{empty}^{(3,3)}: \longrightarrow stack^{(3,3)}$

$\mathrm{push}^{(3,3)}: stack^{(3,3)}\, nat^{(3,3)} \longrightarrow stack^{(3,3)}$

$\mathrm{pop}^{(3,3)}: stack^{(3,3)} \longrightarrow stack^{(3,3)}$

$\mathrm{top}^{(3,3)}: stack^{(3,3)} \longrightarrow nat^{(3,3)}$

Semantics

$\mathrm{tt}^{(3,3)} \vdash \mathrm{eq}_{\mathrm{nat}}\left(0, 0\right)^{(3,3)}$, $\mathrm{tt}^{(3,3)} \vdash \mathrm{eq}_{\mathrm{stack}}\left(empty, empty\right)^{(3,3)}$

$\mathrm{eq}_{\mathrm{nat}}\left(m, n\right) \vdash \mathrm{eq}_{\mathrm{nat}}\left(\mathrm{suc}\left(m\right), \mathrm{suc}\left(n\right)\right)$

$\mathrm{eq}_{\mathrm{stack}}\left(x, y\right) \wedge \mathrm{eq}_{\mathrm{nat}}\left(m, n\right) \vdash \mathrm{eq}_{\mathrm{stack}}\left(\mathrm{push}\left(x, m\right), \mathrm{push}\left(y, n\right)\right)$

$\mathrm{eq}_{\mathrm{stack}}\left(x, x\right) \wedge \mathrm{eq}_{\mathrm{nat}}\left(m, m\right) \vdash \mathrm{eq}_{\mathrm{stack}}\left(\mathrm{pop}\left(\mathrm{push}\left(x, m\right)\right), x\right)$

$\mathrm{eq}_{\mathrm{stack}}\left(x, x\right) \wedge \mathrm{eq}_{\mathrm{nat}}\left(m, m\right) \vdash \mathrm{eq}_{\mathrm{nat}}\left(\mathrm{top}\left(\mathrm{push}\left(x, m\right)\right), m\right)$ •

Enactional "pop":

$$\mathrm{pop}: \begin{pmatrix} \mathrm{stack}_i \longrightarrow \left(\mathrm{stack}_i,\ \mathrm{stack}_{i+1}\right) \\ \mathrm{nat}_i \longrightarrow \left(\mathrm{empty}_i,\ \mathrm{nat}_{i+1}\right) \end{pmatrix}:$$

$$\mathrm{pop}_{i.j}^{(3,3)}\ :\ \left(\left(stack_{j.i}\right) \longrightarrow \left(\mathrm{nil}_{j.i}\right)\right) \longrightarrow \left[\left(nil_{j.i}\right); \left(stack_{j.i+1}\right)\right]$$

$$\mathrm{pop}_{i.1}^{(3,3)}:\ \left(\left(stack_{i.1}\right) \longrightarrow \left(\mathrm{nil}_{i.1}\right)\right) \longrightarrow \left[\left(nil_{i.1}\right); \left(stack_{i.2}\right)\right],\ i = 1,\ 2,\ 3$$

$$\mathrm{pop}_{i.2}^{(3,3)}:\ \left(\left(stack_{i.2}\right) \longrightarrow \left(\mathrm{nil}_{i.2}\right)\right) \longrightarrow \left[\left(nil_{i.2}\right); \left(stack_{i.3}\right)\right]$$

$$\mathrm{pop}_{i.3}^{(3,3)}:\ \left(\left(stack_{i.3}\right) \longrightarrow \left(\mathrm{nil}_{i.3}\right)\right) \longrightarrow \left[\left(nil_{i.3}\right); \left(stack_{i.4}\right)\right]\ \bullet$$

$$\mathrm{tt}^{(3,3)}: \left(\mathrm{tt}_1,\ \mathrm{tt}_2,\ \mathrm{tt}_3\right) = \left(t_1 \longrightarrow f_1 \equiv t_2 \longrightarrow f_2 \Longrightarrow t_3 \longrightarrow f_3\right)$$

$$\mathrm{eq}_{\mathrm{stack}}\left(x, x\right)_{i.j}^{(3,3)} \wedge \mathrm{eq}_{\mathrm{nat}}\left(m, m\right)_{i.j}^{(3,3)} \vdash$$

$$\mathrm{eq}_{\mathrm{stack}}\left(\mathrm{pop}_{i.j}^{(3,3)}\left(\mathrm{push}_{i.j}^{(3,3)}\left(x, m\right)\right), x\right)$$

Model

$\mathrm{stack}_{1.1}$	$\mathrm{stack}_{2.1}$	$\mathrm{stack}_{3.1}$
$\mathrm{stack}_{1.2}$	$\mathrm{stack}_{2.2}$	$\mathrm{stack}_{3.2}$
$\mathrm{stack}_{1.3}$	$\mathrm{stack}_{2.3}$	$\mathrm{stack}_{3.3}$

$\mathrm{nat}_{1.1}$	$\mathrm{nat}_{2.1}$	$\mathrm{nat}_{3.1}$
$\mathrm{nat}_{1.2}$	$\mathrm{nat}_{2.2}$	$\mathrm{nat}_{3.2}$
$\mathrm{nat}_{1.3}$	$\mathrm{nat}_{2.3}$	$\mathrm{nat}_{3.3}$

Examples

$$\left(\mathbf{1}\right)\ \mathrm{pop}_{i.1}{}^{(3,3)}:\ \left(\left(stack_{i.1}\right)\longrightarrow\left(\mathrm{nil}_{i.1}\right)\right)\longrightarrow\left[\left(nil_{i.1}\right);\left(stack_{i.2}\right)\right]:$$

$$\mathrm{pop}_{1.1}^{(3,3)}\left[\left(\ulcorner\,\urcorner_{1.1}\right);\left(\mathrm{nil}_{1.2}\right);\left(\mathrm{nil}_{1.3}\right)\right]\longrightarrow\left[\left(\mathrm{nil}_{1.1}\right);\left(\ulcorner\,\urcorner_{1.2}\right);\left(\mathrm{nil}_{1.3}\right)\right].$$

$$\mathrm{pop}_{1.1}^{(3,3)}:\ \begin{array}{|ccc|}\hline \left(\ulcorner\,\urcorner\right)_{1.1} & \mathrm{stack}_{2.1} & \mathrm{stack}_{3.1}\\ \left(\mathrm{nil}_{1.2}\right) & \mathrm{stack}_{2.2} & \mathrm{stack}_{3.2}\\ \left(\mathrm{nil}_{1.3}\right) & \mathrm{stack}_{2.3} & \mathrm{stack}_{3.3}\\ \hline\end{array}\ \Longrightarrow$$

$$\begin{array}{|ccc|}\hline \left(\mathrm{nil}_{1.1}\right) & \mathrm{stack}_{2.1} & \mathrm{stack}_{3.1}\\ \left(\ulcorner\,\urcorner\right)_{1.2} & \mathrm{stack}_{2.2} & \mathrm{stack}_{3.2}\\ \left(\mathrm{nil}_{1.3}\right) & \mathrm{stack}_{2.3} & \mathrm{stack}_{3.3}\\ \hline\end{array}$$

$$\left(\mathbf{2}\right)\ \mathrm{eq}_{\mathrm{stack}}\left(x,x\right)\wedge\mathrm{eq}_{\mathrm{nat}}\left(m,m\right)\vdash\mathrm{eq}_{\mathrm{stack}}\left(\mathrm{pop}\left(\mathrm{push}\left(x,m\right)\right),x\right)$$

$$\left(\mathrm{pop}_{i.j}^{(3,3)}\left(\mathrm{push}_{i.j}^{(3,3)}\left(x,m\right)\right),x\right):$$

with

$$x=\left[\left(\mathrm{nil}_{1.1}\right);\left(\mathrm{nil}_{1.2}\right);\left(\mathrm{nil}_{1.3}\right)\right],\quad m=\left(\ulcorner\,\urcorner_{1.1}\right)$$

and

$$\mathrm{push}_{i.j}^{(3,3)}\left(\left[\left(\mathrm{nil}_{1.1}\right);\left(\mathrm{nil}_{1.2}\right);\left(\mathrm{nil}_{1.3}\right)\right],\left(\ulcorner\,\urcorner_{1.1}\right)\right)=$$

$$\mathrm{push}_{1.1}^{(3,3)}\left(\left[\left(\ulcorner\,\urcorner_{1.1}\right);\left(\mathrm{nil}_{1.2}\right);\left(\mathrm{nil}_{1.3}\right)\right]\right),$$

and

$$\mathrm{pop}_{1.1}^{(3,3)}\left(\left[\left(\ulcorner\,\urcorner_{1.1}\right);\left(\mathrm{nil}_{1.2}\right);\left(\mathrm{nil}_{1.3}\right)\right]\right)=$$

$$\left[\left(\mathrm{nil}_{1.1}\right);\left(\ulcorner\,\urcorner_{1.2}\right);\left(\mathrm{nil}_{1.3}\right)\right]$$

therefore

$$\left[\left(\text{nil}_{1.1}\right);\left(\text{nil}_{1.2}\right);\right.$$
$$\left.\left(\text{nil}_{1.3}\right)\right] \neq \left[\left(\text{nil}_{1.1}\right);\left(\ulcorner_{1.2}\urcorner_{1.2}\right);\left(\text{nil}_{1.3}\right)\right].$$

$$\text{eq}_{\text{stack}}\left(x, x\right) \wedge \text{eq}_{\text{nat}}\left(m, m\right) \vdash \text{eq}_{\text{stack}}\left(\text{pop}_{i.j}^{(3,3)}\left(\text{push}_{i.j}^{(3,3)}\left(x, m\right)\right), x\right) =$$

false.

Symmetry/asymmetry of pop and push

Classical case : symmetry of pop and push

$$\vdash \text{eq}_{\text{stack}}\left(\text{pop}\left(\text{push}\left(x, m\right)\right), x\right) = \text{true}$$

$$\text{eq}_{\text{stack}}\left(\text{pop}\left(\text{push}\left(x, m\right)\right), x\right) \longrightarrow \text{eq}$$
$$_{\text{stack}}\left(\text{pop}\left(\text{push}\left(x, m\right) = \left(m, x\right)\right) = x, x\right) = \text{true}$$

Enactional case : asymmetry of pop and push

$$\text{eq}_{\text{stack}}\left(\text{pop}\left(\text{push}\left(x, m\right)\right), x\right) \longrightarrow \text{eq}$$
$$_{\text{stack}}\left(\text{pop}_{i.j}\left(\text{push}\left(x, m\right)\right), x\right) = \text{false}$$

$$\text{eq}_{\text{stack}}\left(\text{pop}_{1.1}\left(\text{push}_{1.1}\left(x, m\right) = \left(m_{1.1}, x_{1.1}\right)\right) = \left(\perp_{1.1}; m_{1.2}\right), x\right) = \text{false}$$

1.4.2. Interactional enaction

Complementary to the reflectional enaction the interactional enation is introduced.

$$\boxed{\ulcorner_{i.j}\urcorner_{i.j} \Longleftrightarrow \begin{pmatrix} \phi_{i.j} \\ \ulcorner_{i+1.j} \end{pmatrix}}$$

$$\boxed{\begin{matrix} \ulcorner_1 & - & - \\ - & \ulcorner_2 & - \\ - & - & - \end{matrix}} \longrightarrow \boxed{\begin{matrix} \phi_{1.1} & \ulcorner_{2.1} & - \\ - & \phi_{2.2} & \ulcorner_{3.2} \\ - & - & - \end{matrix}}$$

1.4.3. Memristive STACK

In contrast to the destructive definition of POP for the classical STACK, we add a memristive definition for POP, which is cancelling the addressed state at his address but is simultaneously storing the value of the state at its enactional domain.
Hence, the memristive stack concept is *destructive* in its monocontextural function and at once *memristive* in its polycontextural behavior.
storing :: "to place or leave in a location (as a warehouse, library, or computer memory) for preservation or later use or disposal." (Webster)

If a parcel drops out from a staple, its vanishing gets registered by the memory of that annihilation. Annihilation gets registered.

"After execution, the parameters have been erased and replaced with any return values."

Memristive PUSH
The classical definition of PUSH is atomistic, linear and abstract. In contrast, the memristive PUSH has to reflect the retrogradeness of any iterability, here, the character of the iteration of the morphogrammatic PUSH operation of the STACK.

Input	**Operation**	**Stack**	**STACK**
$a_{1.1}$	Push operand	$a_{1,1}$	PUSH($a_{1.1}$) $\longrightarrow$ $a_{1.1}$
$a_{1.1}$	Pop operand	$[\perp_{1.1}, a_{1.2}]$	POP( ) $\longrightarrow$ $[\perp_{1.1}, a_{1.2}]$

A classical STACK maschine is neutral to its data, i.e. any data accepted might be dublicated, i.e. droped. This is expressed with 2 sorts of terms for the category of a STACK: *nat* and *stack*.
How are memristive STACK machines defined in respect to PUSH?

Also the data type (nat, indicational,kenomic, morphogrammatic) are not crucial to demonstrate the mechanism of the enactional stack, it might be interesting as a next step towards a enactional stack machine to know how the operation ADD is working in the different settings.

Sign repertoire
sign = { ";" , "|", ", ", "(", ")"},
operators ={ Pop, Add, Push},
terms ={ t, $\perp$}

Enactional case:

Input	Operation	Stack
$a_{1.1}$	Push	$a_{1.1}$
$a_{1.1}$	Pop_{Refl}	$[\perp_{1.1}, a_{1.2}]$
$a_{1.1}$	$\text{Pop}_{\text{Inter}}$	$[\perp_{1.1}, a_{2.1}]$

(1) Natural numbers stack machine example with Push, Pop and Add

Input	Operation	Stack	Rule
1	Push operand	$1_{1.1}$	
2	Push operand	$2_{1.1}, 1_{1.1}$	
4	Push operand	$4_{1.1}, 2_{1.1}, 1_{1.1}$	
$*$	$\text{Multiply}_{1.1}$	$8_{1.1}, 1_{1.1}$	
$+$	$\text{Add}_{1.1}$	$9_{1.1}$	
3	Push operand	$3_{1.1}, 9_{1.1}$	
3	Pop operand	$\perp_{1.1}, 9_{1.1}; 3_{1.2}$	EN
9	Pop operand	$\perp_{1.1}; 9_{1.2}, 3_{1.2}$	EN
5	Push operand	$5_{1.1}; 9_{1.2}, 3_{1.2}$	
$*$	$\text{Multiply}_{1.2}$	$5_{1.1}; 27_{1.2}$	EN
4	$\text{Push}_{1.1;\ 1.2}$	$4_{1.1}, 5_{1.1}; 4_{1.2,}\ 27_{1.2}$	PAR, EN
$+$	$\text{Add}_{1.1}$	$9_{1.1}; 4_{1.2,}\ 27_{1.2}$	
$+$	$\text{Add}_{1.2}$	$9_{1.1}; 31_{1.2}$	EN
31	$\text{Pop}_{1.2}$	$9_{1.1}; \perp_{1.2}; 31_{1.3}$	EN
9	$\text{Pop}_{1.1}$	$\perp_{1.1}; 9_{1.2}; 31_{1.3}$	EN

Register shifts

How to move the content of one register to neighbor register?

Say, $(1_{1.1},\ 1_{1.1},\ 1_{1.1})$ to $(1_{1.2},\ 1_{1.2},\ 1_{1.2})$?

Input	Operation	Stack	Rule
1	Push operand	$1_{1.1}; \perp_{1.2}$	
1	Push operand	$1_{1.1}, 1_{1.1}; \perp_{1.2}$	
1	Push operand	$1_{1.1}, 1_{1.1}, 1_{1.1}; \perp_{1.2}$	
1	Pop operand	$1_{1.1}, 1_{1.1}; 1_{1.2}$	
1	Pop operand	$1_{1.1}; 1_{1.2}, 1_{1.2}$	
1	Pop operand	$\perp_{1.1}; 1_{1.2}, 1_{1.2}, 1_{1.2}$ •	

(2) Indicational stack machine example with Push, Pop and Add

Input	Operation	Stack
$\ulcorner_{1.1}$	Push	$\ulcorner_{1.1}$
$\ulcorner_{1.1}$	Push	$\ulcorner_{1.1}$, $\ulcorner_{1.1}$
$+\,J1_{1.1}$	$Add_{1.1}$	$(\ulcorner_{1.1}\ \ulcorner_{1.1})$
$(\ulcorner_{1.1}\ \ulcorner_{1.1})$	$Pop_{1.1}$	$\perp_{1.1}$; $(\ulcorner_{1.2}\ \ulcorner_{1.2})$
$\ulcorner_{1.2}$	$Push_{1.2}$	$\perp_{1.1}$; $\ulcorner_{1.2}$, $(\ulcorner_{1.2}\ \ulcorner_{1.2})$
$\ulcorner_{1.1}$	$Push_{1.1}$	$\ulcorner_{1.1}$; $\ulcorner_{1.2}$, $(\ulcorner_{1.2}\ \ulcorner_{1.2})$
$\ulcorner_{1.2}$	$Push_{1.2}$	$\ulcorner_{1.1}$; $\ulcorner_{1.2}$, $\ulcorner_{1.2}$, $(\ulcorner_{1.2}\ \ulcorner_{1.2})$
$\ulcorner_{1.1}$	$Push_{1.1}$	$\ulcorner_{1.1}$, $\ulcorner_{1.1}$; $\ulcorner_{1.2}$, $\ulcorner_{1.2}$, $(\ulcorner_{1.2}\ \ulcorner_{1.2})$
$+\,J1_{1.1}$	$Add_{1.1}$	$(\ulcorner_{1.1}\ \ulcorner_{1.1})$; $\ulcorner_{1.2}$, $\ulcorner_{1.2}$, $(\ulcorner_{1.2}\ \ulcorner_{1.2})$
$+\,J1_{1.2}$	$Add_{1.2}$	$(\ulcorner_{1.1}\ \ulcorner_{1.1})$; $(\ulcorner_{1.2}\ \ulcorner_{1.2})$, $(\ulcorner_{1.2}\ \ulcorner_{1.2})$
$(\ulcorner_{1.1}\ \ulcorner_{1.1})$	$Pop_{1.1}$	$\perp_{1.1}$; $(\ulcorner_{1.2}\ \ulcorner_{1.2})$, $(\ulcorner_{1.2}\ \ulcorner_{1.2})$, $(\ulcorner_{1.2}\ \ulcorner_{1.2})$
$(\ulcorner_{1.2}\ \ulcorner_{1.2})$	$Pop_{1.2}$	$\perp_{1.1}$; $\perp_{2.2}$, $(\ulcorner_{1.2}\ \ulcorner_{1.2})$, $(\ulcorner_{1.2}\ \ulcorner_{1.2})$; $(\ulcorner_{1.3}\ \ulcorner_{1.3})$
$+J1_{1.2}$	$Add_{1.2}$	$\perp_{1.1}$; $(\ulcorner_{1.2}\ \ulcorner_{1.2}\ \ulcorner_{1.2}\ \ulcorner_{1.2})$; $(\ulcorner_{1.3}\ \ulcorner_{1.3})$
$\ulcorner_{1.1}$, $\ulcorner_{2.2}$	$Push_{1.1,\,2.2}$	$\ulcorner_{1.1}$; $(\ulcorner_{1.2}\ \ulcorner_{1.2}\ \ulcorner_{1.2}\ \ulcorner_{1.2})$; $(\ulcorner_{1.3}\ \ulcorner_{1.3})\ \vert\ \ulcorner_{2.2}\bullet$

Additions

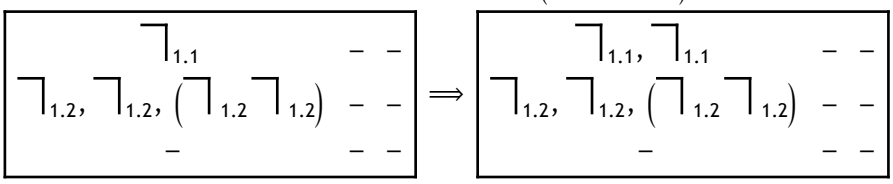
$ADD_{i.i}(\ulcorner_{i.i}, \ulcorner_{i.i}) = (\ulcorner_{i.i}\ \ulcorner_{i.i}) = (\ulcorner\ulcorner)_{i.i}$: iteratiive addition

$ADD_{i.j}(\ulcorner_{i.i}, \ulcorner_{i.j+1}) = (\ulcorner_{i.i}\ \ulcorner_{i.j+1})$: reflectiive addition (omitted)

Modell : iteration

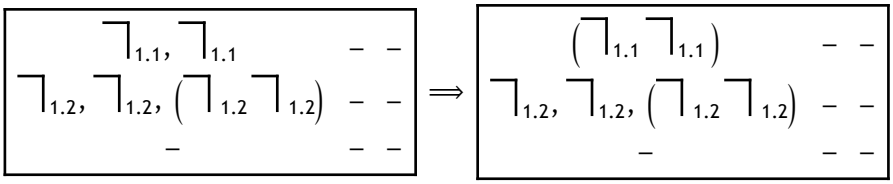
$\urcorner_{1.1}$ Push $_{1.1}$ $\urcorner_{1.1}, \urcorner_{1.1}; \urcorner_{1.2}, \urcorner_{1.2}, (\urcorner_{1.2}\urcorner_{1.2})$

$$\boxed{\begin{array}{ccc} \urcorner_{1.1} & - & - \\ \urcorner_{1.2}, \urcorner_{1.2}, (\urcorner_{1.2}\urcorner_{1.2}) & - & - \\ - & - & - \end{array}} \Longrightarrow \boxed{\begin{array}{ccc} \urcorner_{1.1}, \urcorner_{1.1} & - & - \\ \urcorner_{1.2}, \urcorner_{1.2}, (\urcorner_{1.2}\urcorner_{1.2}) & - & - \\ - & - & - \end{array}}$$

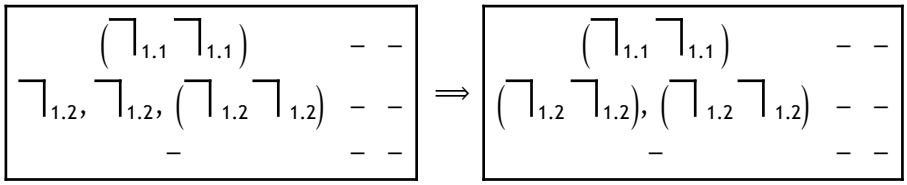


$+$ J1 $_{1.1}$ Add $_{1.1}$ $(\urcorner_{1.1}\urcorner_{1.1}); \urcorner_{1.2}, \urcorner_{1.2}, (\urcorner_{1.2}\urcorner_{1.2})$

$$\boxed{\begin{array}{ccc} \urcorner_{1.1}, \urcorner_{1.1} & - & - \\ \urcorner_{1.2}, \urcorner_{1.2}, (\urcorner_{1.2}\urcorner_{1.2}) & - & - \\ - & - & - \end{array}} \Longrightarrow \boxed{\begin{array}{ccc} (\urcorner_{1.1}\urcorner_{1.1}) & - & - \\ \urcorner_{1.2}, \urcorner_{1.2}, (\urcorner_{1.2}\urcorner_{1.2}) & - & - \\ - & - & - \end{array}}$$

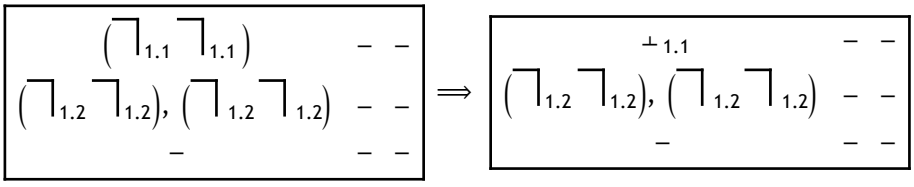


$+$ J1 $_{1.2}$ Add $_{1.2}$ $(\urcorner_{1.1}\urcorner_{1.1}); (\urcorner_{1.2}\urcorner_{1.2}), (\urcorner_{1.2}\urcorner_{1.2})$



$$\boxed{\begin{array}{ccc} (\urcorner_{1.1}\urcorner_{1.1}) & - & - \\ \urcorner_{1.2}, \urcorner_{1.2}, (\urcorner_{1.2}\urcorner_{1.2}) & - & - \\ - & - & - \end{array}} \Longrightarrow \boxed{\begin{array}{ccc} (\urcorner_{1.1}\urcorner_{1.1}) & - & - \\ (\urcorner_{1.2}\urcorner_{1.2}), (\urcorner_{1.2}\urcorner_{1.2}) & - & - \\ - & - & - \end{array}}$$

$(\urcorner_{1.1}\urcorner_{1.1})$ Pop $_{1.1}$ $\perp_{1.1}; (\urcorner_{1.2}\urcorner_{1.2}), (\urcorner_{1.2}\urcorner_{1.2}), (\urcorner_{1.2}\urcorner_{1.2})$

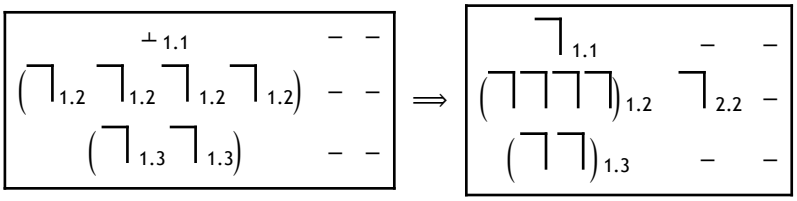


$$\boxed{\begin{array}{ccc} (\urcorner_{1.1}\urcorner_{1.1}) & - & - \\ (\urcorner_{1.2}\urcorner_{1.2}), (\urcorner_{1.2}\urcorner_{1.2}) & - & - \\ - & - & - \end{array}} \Longrightarrow \boxed{\begin{array}{ccc} \perp_{1.1} & - & - \\ (\urcorner_{1.2}\urcorner_{1.2}), (\urcorner_{1.2}\urcorner_{1.2}) & - & - \\ - & - & - \end{array}}$$

$[\ldots]$

$\urcorner_{1.1}, \urcorner_{2.2}$ Push $_{1.1,2.2}$ $\perp_{1.1}; (\urcorner_{1.2}\urcorner_{1.2}\urcorner_{1.2}\urcorner_{1.2}); (\urcorner_{1.3}\urcorner_{1.3}) \mid \urcorner_{2.2}$

$$\boxed{\begin{array}{ccc} \perp_{1.1} & - & - \\ (\urcorner_{1.2}\urcorner_{1.2}\urcorner_{1.2}\urcorner_{1.2}) & - & - \\ (\urcorner_{1.3}\urcorner_{1.3}) & - & - \end{array}} \Longrightarrow \boxed{\begin{array}{ccc} \urcorner_{1.1} & - & - \\ (\urcorner\urcorner\urcorner\urcorner)_{1.2} & \urcorner_{2.2} & - \\ (\urcorner\urcorner)_{1.3} & - & - \end{array}}$$



(3) Kenomic stack machine example with Push, Pop and Add

a $_{1.1}$ Push $(a)_{1.1}$

$a_{1.1}$ Push $(a)_{1.1}, (a)_{1.1}$

+MG1 Add $_{1.1}$ $(a\,a)_{1.1\times.1} \mid (ab)_{1.1\times.2}$: Branching

$a_{1.1\times.1}$ Pop $(a)_{1.1\times.1} \mid (ab)_{1.1\times.2}$

$a_{1.1\times.1}$ Pop $\perp_{1.1\times.1} \mid (ab)_{1.1\times.2}; (a)_{1.2\times.1}$

$a_{1.1\times.2}$ Push $(ab)_{1.1\times.2}; (aa)_{1.1\times.2}, (ab)_{1.2\times.1}$

$a_{1.1\times.1}$ Push $_{1.1}$ $(a)_{1.1\times.1,} (ab)_{1.1\times.1}; (aa)_{1.2}, (ab)_{1.2}$

$a_{1.2}$ Push $_{1.2}$ $(aba)_{1.1\times.1}; (aaa)_{1.2}, (ab)_{1.2}$

$a_{1.2}$ Pop $_{1.2}$ $(aba)_{1.1}; (aa)_{1.2}, (ab)_{1.2}$

$a_{1.1}$ Pop $_{1.1}$ $(ab)_{1.1}; (aa)_{1.2}, (ab)_{1.2}$

$a_{1.1}$ Pop $_{1.1}$ $(a)_{1.1}; (aa)_{1.2}, (ab)_{1.2}$

$a_{1.1}$ Pop $_{1.1}$ $\perp_{1.1}; (a)_{1.2}, (aa)_{1.2}, (ab)_{1.2}$

+ ADD $_{1.2}$ $\perp_{1.1}; (aaa)_{1.2\times.1} \mid (aab)_{1.2\times.2}, (ab)_{1.2}$

Branching

$a_{1.1}$ Push $(a)_{1.1}$

$a_{1.1}$ Push $(a)_{1.1}, (a)_{1.1}$

+ Add $_{1.1}$ $(a\,a)_{1.1\times.1} \mid (ab)_{1.1\times.2}$: Branching,

$a_{1.1\,1}$ Pop $_{1.1\times.1}$ $(\perp)_{1.1\times.1} \mid (ab)_{1.1\times.2}; (a)_{1.2\times.1}$

$a_{1.2\times.1}$ Pop $_{1.2\times.1}$ $(\perp)_{1.1\times.1} \mid (ab)_{1.1\times.2}; (\perp)_{1.2\times.1}; (a)_{1.3\times.2}$

$a_{1.3\times.1}$ Pop $_{1.3\times.1}$ $(\perp)_{1.1\times.1} \mid (ab)_{1.1\times.2};$

$(\perp)_{1.2\times.1}; (\perp)_{1.3\times.2}; (a)_{1.4\times.1}$

Null

(4) Monomorphic stack machine example with Push, Pop and Add

start : ... $(aa)_{1.1}; (ab)_{1.2}$

$(aa)_{1.1}$ Pop $_{1.1}$ $\perp_{1.1}; (a\,a)_{1.2}, (ab)_{1.2}$

$(aa)_{1.2}$ Pop $_{1.2}$ $\perp_{1.1}; (ab)_{1.2}; (aa)_{1.3}$

$(ab)_{1.2}$ Pop $_{1.2}$ $\perp_{1.1}; \perp_{1.2}, \perp_{1.2}; (ab)_{1.3}, (aa)_{1.3}$

$(ab)_{1.2}$ $Pop_{1.2}$ $\perp_{1.1}; \perp_{1.2}; (ab)_{1.3}, (aa)_{1.3}$ NORM

$(a)_{1.3}$ $Pop_{1.3}$ $\perp_{1.1}; \perp_{1.2}; (a)_{1.3}, (aa)_{1.3}; (a)_{1.4}$

$(a)_{1.3}$ $Pop_{1.3}$ $\perp_{1.1}; \perp_{1.2}; (aa)_{1.3}; (a)_{1.4}, (a)_{1.4}$

$(aa)_{1.3}$ $Pop_{1.3}$ $\perp_{1.1}; \perp_{1.2}; \perp_{1.3}; (aa)_{1.4}, (a)_{1.4}, (a)_{1.4}$

$(aa)_{1.4}$ $Pop_{1.4}$ $\perp_{1.1}; \perp_{1.2}; \perp_{1.3}; (a)_{1.4}; (aa)_{1.5}$

$(a)_{1.4}$ $Pop_{1.4}$ $\perp_{1.1}; \perp_{1.2}; \perp_{1.3}; \perp_{1.4}; (a)_{1.5}, (aa)_{1.5}$

1.4.4. Enactional stacks in the framework of polycontexturality

Superoperator had been introduced to deal with polycontextural formalisms for logics and programming languages.
In this light, enaction is connected with the superoperation of *replication* and *bifurcation*. Reflectional enaction is replication with cancelling and interactional enaction is bifurcation with cancelling.

Superoperators

sops: { ID, PERM, RED, REPL, BIF}

STACK Identity
$ID(a_{1.1}) \longrightarrow (a_{1.1})$

STACK Replication
$Rep(a_{1.1}; \perp_{1.2}) \longrightarrow (a_{1.1}; a_{1.2})$

Reflectional enaction
$Rep_{EN}(a_{1.1}) \longrightarrow (\perp_{1.1}; a_{1.2})$

STACK Permutation
$Perm(a_{1.1}; a_{2.2}) \longrightarrow (a_{2.2}; a_{1.1})$

STACK Reduction
$Red_{1.1}(a_{1.1}; a_{2.2}) \longrightarrow (a_{1.1}; a_{1.1})$

STACK Bifurcation
$Bif(a_{1.1}; \perp_{2.1}; \perp_{3.1}) \longrightarrow (a_{1.1}; a_{2.1}; a_{3.1})$

Interactional enaction

$Bif_{EN}(a_{1.1}; \perp_{2.1}; \perp_{3.1}) \longrightarrow (\perp_{1.1}; a_{2.1}; a_{3.1})$

1.4.5. Memristive operators

Operators with memristive properties are: enaction and retrograde recursivity. This paper is focused on the memristivity of enaction. Other aspects are studied elsewhere.

> *"Memristance is a property of an electronic component. If charge flows in one direction through a circuit, the resistance of that component of the circuit will increase and if charge flows in the opposite direction in the circuit, the resistance will decrease. If the flow of charge is stopped by turning off the applied voltage, the component will 'remember' the last resistance that it had, and when the flow of charge starts again the resistance of the circut will be what it was last active."*
> *"In other words, a memristor is 'a device which* bookkeeps *the charge passing its own port'"* (Stanley Williams)

A new operator, inverse to DROP shall be GET. This operator is defined to restore the cancelled state out of the cancellation by a reverse reconstruction of the result of enaction.

DROP or Push, in its memristive definition, is representing the action of 'stopping' the flow and keeping the state in a different mode, while GET is "remembering' the last resistance" by getting the restored state from POP back into its domain. This is not any kind of *creatio ex nihilo,* or a crude double bookkeeping, but an interplay on different levels of realization of states; states as produced and states as remembered.

Both together, DROP and GET, are managing the "bookkeeping" of the memristive device.
Cancelling, while keeping (with POP), and keeping, while cancelling (with GET). There is obviously a nice chiasm in the game.

Interplay of erasing and restoring

Memristive enaction

$$\mathrm{POP}\left(t_{i.j}\right): \left[t_{i.j}\right] \longrightarrow \begin{bmatrix} \left(\perp\right)_{i.j} \\ t_{i.j+1} \end{bmatrix}$$

$$\mathrm{GET}\left(\perp_{i.j}\right): \begin{bmatrix} t_{i.j+1} \\ \left(\perp\right)_{i.j} \end{bmatrix} \longrightarrow \left[t_{i.j}\right]$$

Example

NR. Input	Operation	Stack	Rule
$(0)\ (aa)_{1.2}$	$\mathrm{POP}_{1.2}$	$\perp_{1.1};\ \perp_{1.2}, (ab)_{1.2}; (aa)_{1.3}$	$(0,\ \mathrm{POP})$
$(1)\ (\perp)_{1.2}$	$\mathrm{GET}_{1.2}$	$(ab)_{1.1}; \perp_{1.2}; (aa)_{1.3}$	$(1,\ \mathrm{GET})$
$(2)\ (ab)_{1.1}$	$\mathrm{POP}_{1.1}$	$\perp_{1.1};\ (ab)_{1.2}; (aa)_{1.3}$	$(2,\ \mathrm{POP})$

Properties of memristive enaction

One amazing property of memristive enaction is its 'stability' or 'pemanence', i.e. its *persistence* .

Persistence:
To all enactional cancellation of a complexion there always remains at least one last enacted state. There might be an interpretation of the abstract *persistence* and the *endurance* of a state in a physical memristive system.

1.5. Tabular memristive stacks

1.5.1. Polycontextural matrix

To understand the new concept of a memristive stack we have to consider its polycontexturality and, at least, is properties of reflectional and interactional operations.

These two features are enabling the stack-theoretic distinction of reflectional and interactional enaction for stack operations. Especially the POP-operation has 3 stack-specific modi of action:
1) classical *cancelling* (annihilation) of a state,
2) *reflectional* and
3) *interactional* enaction.
Enaction is the double operation of cancelling (eliminating) and storing the eliminated state *as* a memorized state, or as a recorded/archived state in another domain.
These enactional operations are enabling the stack to behave as a memristive system.

Enactional stacks are based on a special poly-stack construction, i.e. data type.

Hence, memristive stacks are defining memristive data structures or even data paradigms, which are demanding for corresponding programming paradigms, logics and arithmetics.

A tabular stack is the data structure for a memristive enactional stack machine.

Hence, the core model for memristive stacks is the well known "*kenomic matrix*" of polycontextural logic.

The basic structure of a enactional matrix is its bifunctorial interchangeability of its properties and features.

1.5.2. Quadralectic memristive stacks

From a distinction-theoretical point of view, the single-distinction approach is not even half the option. Thematization, as an explication of "understanding and acting" gets a first formalization and operationalization with the paradigm of a quadralectic stack.

$$\text{quadralectic STACK} = \begin{array}{c|c} \left(\underline{\text{STACK}}\right\rfloor\right)_j^5 & \left(\left\lfloor\underline{\text{STACK}}\right)_k^4 \\ \hline \left(\overline{\text{STACK}}\right\rceil\right)_l^4 & \left(\left\lceil\overline{\text{STACK}}\right)_m^3 \end{array}.$$

$$\text{STACK}^{(m,\,n)} = \left(\text{sorts},\ \text{ops},\ \text{sops}\right)$$

$$\text{sorts} = \left(\text{stack},\ \text{sign}\right\}$$

$$\text{ops} = \left\{\text{pop},\ \text{push},\ \text{top},\ \text{ADD}\right).$$

$$\text{sops} = \left\{\text{id},\ \text{perm},\ \text{repl},\ \text{red},\ \text{bif}\right\}$$

Null

form for quadralectics

$$q_j^n = \begin{array}{c|c} \left(\lrcorner\right)_j^{n-1} & \left(\llcorner\right)_k^{n-2} \\ \hline \left(\urcorner\right)_l^{n-2} & \left(\ulcorner\right)_m^{n-3} \end{array}$$

enactional forms for quadralectics

$$q_j^n = \begin{array}{c|c} \left(\begin{array}{c} \lrcorner^{\,i.j} \\ \phi_{\,i.j+1} \end{array}\right)_j^{n-1} & \left(\begin{array}{c} {}^{i.j}\llcorner \\ \phi_{\,i+1.\,j} \end{array}\right)_k^{n-2} \\ \hline \left(\begin{array}{c} \phi_{\,i.j} \\ \urcorner_{\,i+1.\,j} \end{array}\right)_l^{n-2} & \left(\begin{array}{c} \phi_{\,i.j} \\ \ulcorner_{\,i.j+1} \end{array}\right)_m^{n-3} \end{array}$$

Hence, the " data typ " of *a* quadralectic stack is a $4-$ tupel of discontextural distinctions.

Interchangeability of quadralectic enaction

$$\left(\begin{array}{c} \left(\lrcorner^{\,i.j}_{\phi_{\,i.j+1}} \right)^{n-1}_{j} \\ \amalg_{1.2} \\ \left(\urcorner_{i+1.\,j} \phi_{\,i.j} \right)^{n-2}_{l} \end{array} \right) \left(\circ_{1.2} \right) \left(\begin{array}{c} \left(\llcorner_{\phi_{\,i+1.\,j}}^{\,i.j} \right)^{n-2}_{k} \\ \amalg_{1.2} \\ \left(\ulcorner_{i.j+1} \phi_{\,i.j} \right)^{n-3}_{m} \end{array} \right) \rightleftharpoons$$

$$\left(\begin{array}{c} \left(\left(\lrcorner^{\,i.j}_{\phi_{\,i.j+1}} \right)^{n-1}_{j} \circ^{1.0} \left(\llcorner_{\phi_{\,i+1.\,j}}^{\,i.j} \right)^{n-2}_{k} \right) \\ \amalg_{1.2} \\ \left(\left(\urcorner_{i+1.\,j} \phi_{\,i.j} \right)^{n-2}_{l} \circ^{0.2} \left(\ulcorner_{i.j+1} \phi_{\,i.j} \right)^{n-3}_{m} \right) \end{array} \right)$$