Rudolf Kaehr
(1942-2016)

Title

Memristive Recursivity

Towards stack-free computation

Archive-Number / Categories

3_33 / K09, K08, K11

Publication Date

2013

Keywords / Topics

RECURSIVITY AND MEMORY : Two different kinds of memory and recursion, Trito-Trans-Successor TTS, Diamond recursion schemes, Recurrence and retro-gradeness in formal systems
MEMRISTIVITY AND MEMORY, RECURSIVITY AND MEMRISTIVE MEMORY

Disciplines

Computer Science Logic and Foundations of Mathematics, Cybernetics, Theory of Science, Memristive Systems

Abstract

Towards stack-free computation.
There are two fundamentally different aspects to consider for memristive iteration and recursion and computation. One is modeling existing mathematical concepts in a more economic way, and is not in anyway re-modeling its mathematical concepts. The other aspect of memristive computation takes the fact into account that memristive computing is structurally fundamentally different from the established mathematical and physical concept of computation.
The first is based on recursivity, the latter on retrograde recursivity.

Citation Information / How to cite

Rudolf Kaehr: "Memristive Recursivity", www.vordenker.de (Sommer Edition, 2017) J. Paul (Ed.), http://www.vordenker.de/rk/rk_Memristive-Recursivity_2013.pdf

Memristive Recursivity

Towards stack-free computation

Rudolf Kaehr Dr.phil

Copyright ThinkArt Lab ISSN 2041-4358

Abstract

Towards stack-free computation.
There are two fundamentally different aspects to consider for memristive iteration and recursion and computation. One is modeling existing mathematical concepts in a more economic way, and is not in anyway re-modeling its mathematical concepts. The other aspect of memristive computation takes the fact into account that memristive computing is structurally fundamentally different from the established mathematical and physical concept of computation.
The first is based on recursivity, the latter on retrograde recursivity.
(work in progress, 0.3, July/Nov. 2013)

1. Recursivity and memory

Recursivity and memory

Recursivity is memory intense
With each new loop the results of the previous loop has to be handled.
These values of the last computation are stored as objects to be reused for the new calculation.

> *"Q: Does the recursive version usually use less memory?*
> *A: No -- it usually uses more memory (for the stack)."*

http://pages.cs.wisc.edu/~vernon/cs367/notes/6.RECURSION.html

The stack

> ***"****To remember 'where it got up to', the program has a stack. A stack is a special area of memory set aside for remembering 'where to go back to' every time the program makes a method call.*

Implications for recursion

> ***"****So what implications does this all have for recursion? Well, each time we make a recursive call, we 'eat up' a bit more space on the stack. So the maximum depth of recursion is limited by:*
> *our thread's stack size (the amount of memory allocated to the stack);*
> *the number of parameters and local variables used on each call to our method."*

http://www.javamex.com/tutorials/techniques/recursion_how.shtml

> *"During recursion, function calls continue to require more and more stack memory which does not get released until the recursive chain terminates. Stack overflow results when memory allocations are beyond what the stack is able to provide. So if a function has too many levels of recursive calls, one can run out of memory."*

http://joel.inpointform.net/software-development/explanation-of-stack-heap-and-recursion-causing-stack-overflow/

Again,

> *"In the recursion example, notice how the result of each call must be remembered, to do this each recursive call requires an entry on the stack until all recursive calls have been made. This makes the recursive call more expensive in terms of memory. While in the tail recursive example, there are no intermediate values that need to be stored on the stack, the intermediate value is always passed back as a parameter."*

http://myadventuresincoding.wordpress.com/tag/recursion/

Thus, the quesion is: How are the calls remembered? Are there any paradigmatical differences to observe?

A first answer is given by the distinction of *external* and *intrinsic* memory functions.

The first is conceptually realized by the methods of mathematical recursion and technically by the application of CMOS devices for storage.

The second is conceptually realized by the methods of morphogrammatic retrogradeness and technically by the applictions of memristive systems based on memristors.

1.1. Two different kinds of memory and recursion

There are, therefore, two fundamentally different situations to distinguish where the interaction of recursivity and memory are appearing.

One is the classical case, memory is a technical device to realize recursive functions. In this case, that corresponds to the classical mathematical situation of iteration and recursion, memory is not a genuine concept of the definition of recursion.

Memory occurs in the context of the implementation for programming and computation.

In fact, memory occurs in recursive mathematical calculations as a mental representation by the mathematician.

It is a fundamentally different situation if memory is involved in the very definition of the formal concept of recursion as it is constitutive for morphogrammatic calculations. Or memory appears as an 'external' device of calculation.

In this case, memory is implemented, or as it is also called "in-sourced" into the very definition of iterability as it is constitutive for any morphogrammatic iteration, recursion and reflection.

The morphogrammatic concept of retro-grade recursion shall be used to model the behavior of memristive systems in respect of the iterability of its operations.

Afroze Ahmed, Memristor Seminar Report, Feb 22, 2012

"Memristance is a property of an electronic component to retain its resistancelevel even after power had been shut down or lets it remember (or recall) the lastresistance it had before being shut off."

http://www.scribd.com/doc/82430989/Memristor-Seminar-Report

"Memristive architectures are ideally suited for computation within a memory, and thus memristors should not be regarded only as memory, but also as nanoscale computing units." (Lehtonen)

As far as I can see, the paradigmatical difference between external and intrinsic memory functions is not yet in the focus of the known research and development of memristors and the behaviour of memristive systems.

Due to the pressure of producing success in this field, research is concentrated on the obvious prossibilities offered by memristive systems: smaller, faster, cheaper, and the same.

1.2. Recurrence and retro-gradeness in formal systems

1.2.1. Conceptual analysis

Morphogrammatics of retro-grade recursion, or reflection, is a second-order concept, that implies a double recurrence for its recursivity.

As an example I follow the "accumulator" concept of recursion in Scala.

"While in the tail recursive example, there are no intermediate values that need to be stored on the stack, the intermediate value is always passed back as a parameter."

http://myadventuresincoding.wordpress.com/2010/06/20/tail-recursion-in-scala-a-simple-example/#comments

A deconstructive interpretation of the classical recursion concept might use the more implicite version of reursion with the construct of an "accumulator".

By a speculative turn, the "accumulator" written in Scala and realized in CMOS gets a transformation towards an intrinsic implementation of the 'accumulation' into the very concept of 'number' itself.

'Number' in this scenario is a retrograde construct, realized by memristive computation.

Hence, there is not anymore an iterative and recursive understanding of iterability applied.

Iteration and retrogradeness

Graph scheme for mathematical recursion

$$\begin{tikzcd}
 & (E) \arrow[d] & \\
 & (\Gamma) \arrow[dl] \arrow[dr] & \\
(A) & & (\delta) \arrow[ul, bend right]
\end{tikzcd}$$

$E(n,\ a) = (o,\ n,\ a),$

$\Gamma(i,\ n,\ \omega) \equiv n = i$

$A(i,\ n,\ \omega')$

$\delta(i, n, \omega) = (i',\ n,\ \omega')$

$m' = m + 1\ ,\quad m \in \mathfrak{m}$

R. Peters, Dialectica 47/48, *p*. 375, 1958

Successor : **succ** $_i\left(\left[\mathbf{MG}\right]\right)$:

$\forall\, i \in \left(m : g\left(\mathrm{MG}\right)\right),\ 1 \le i \le \mathrm{mg}\left(\mathrm{MG}\right) + 1$

$$\begin{tikzcd}
\text{selection} \arrow[d] & \text{choice} \arrow[l, "\text{retrogression}"'] & \\
\left(\left[\mathrm{mg}_1 \ldots \mathrm{mg}_i \ldots \mathrm{mg}_n\right] \longrightarrow \right. \arrow[d] & \left.\left[\mathrm{mg}_1 \ldots \mathrm{mg}_i \ldots \mathrm{mg}_n\right]\right) \arrow[u] \arrow[r, "\text{result}"] & \left[\mathrm{mg}_1 \ldots \mathrm{mg}_i \ldots \mathrm{mg}_n\ \mathrm{mg}_{n+1}\right] \\
\textit{choice} \arrow[rr, "\text{progression}"'] & & \textit{selection} \arrow[u]
\end{tikzcd}$$

Trito-Trans-Successor TTS

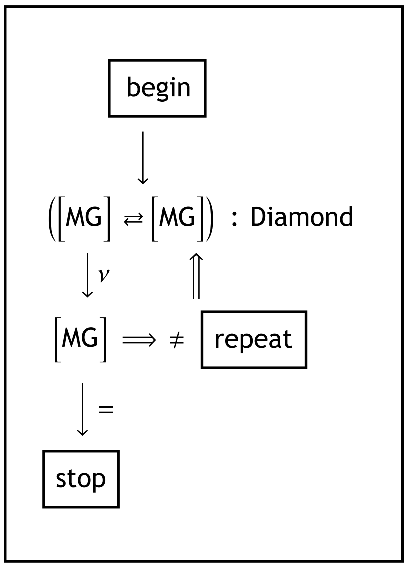
```
fun TTS ts =
    map (fn i => ts@[i])
        (fromto 1 ((AG ts) + 1));
```

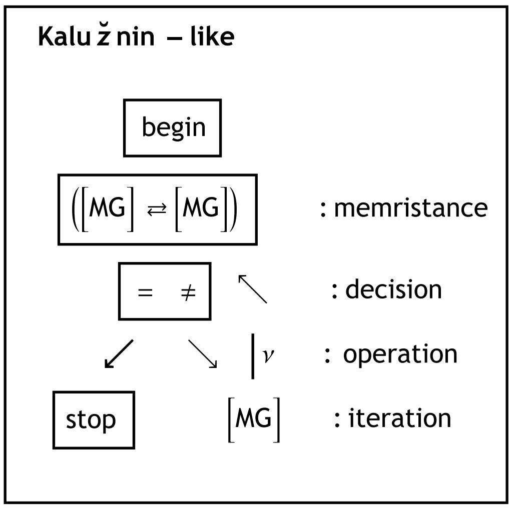
With:
ts : trito-sequence,
AG: aggregation,
fun AG ks = length (rd ks);

```
- TTS [1,2];
val it = [[1,2,1],[1,2,2],[1,2,3]] : int list list
```

direct successors: [1,2,1],[1,2,2],
accretive successor: [1,2,3]

Diamond recursion schemes





The crucial element of a morphogrammatic recursion scheme is obviously its chiastic concept of retro-grade iteration defined by the self-application unit: "d*iamond*".
The Diamond structure represents the
mechanism of memristance of the morphogram MG. MG is

Diamond

$$([MG] \rightleftarrows [MG]) = \begin{tikzcd} ([MG] \arrow[d, shift right] & {[MG])} \arrow[l, "retro"'] \arrow[d, shift right] \\ ([MG] \arrow[u, shift right] \arrow[r, "progr"'] \arrow[ur, phantom, "X"] & {[MG])} \arrow[u, shift right] \end{tikzcd} :: (\text{fromto } 1 ((AG\ ts) + 1)$$

The iteration *operation* (prolongation, succeson) ν : [diamond] $\xrightarrow{\nu}$ [MG] and the *decision* unit: $\boxed{= \quad \neq}$ with its two modes: "=" for "stop", and "≠" for repetition "repeat".

All together defines the morphogrammatic recursion scheme as a composition of the units "diamond", "decision", "operation, calculation" and "repetition, iteration".

This unit "Diamond" is checking the possibilities of the prolongation (successor) operation that is needed because the succession is not abstractly, i.e. independently of the history of the operations, defined, and is not depending on an abstract, pre-given alphabet.

This retrograde feature is implemented by the procedure: (fromto 1 ((AG ts) + 1). That is, the function AG (ts) delivers the values of the 'historical' constellation (state) of the trito-sequence ts, and for accretion, the value 1 is added.

$$\mathbf{succ}_i\left(\mathrm{MG}\right):$$

$$\begin{tikzcd}
\cdot \arrow[d] & \cdot \arrow[l, "\text{retrograde}"'] & \cdot \arrow[l, Leftarrow, "\text{repeat}"'] & \\
\text{begin}: \left(\left[\mathrm{MG}\right] \right. \arrow[r] \arrow[d] & \left. \left[\mathrm{MG}\right]\right) \arrow[u] \arrow[r] & \left[\mathrm{MG}\right] \arrow[u, Rightarrow, dash, "\text{no}"'] \arrow[r, Rightarrow, "\text{yes}"] & \left[\mathrm{MG}\right]: \textit{stop} \\
\cdot \arrow[rr, "\text{prograde } v"] & & \cdot \arrow[u] &
\end{tikzcd}$$

Example

For the start, the aggregation AG is of the example is AG[1,2] = 2, hence the accretion is 3.
Therefore, out of the historical event '[1,2]', just two prolongation in the mode of iteration are defined: [1,2,1] and [1,2,2]. This situation is complemented by accretion to the third pattern [1,2,3]. There are no more possibilities for a prolongation of [1,2] available.

Accretion

$$[1,2] \Longrightarrow [1,2,1],\ [1,2,2],\ [1,2,3]:$$

$$\begin{tikzcd}
 & {} \arrow[d] & {} \arrow[l] & {} \arrow[l, Leftarrow, "{[1,2,3]}"'] \\
\boxed{\text{start:}} & ({[1,2]} \arrow[r] \arrow[d] & {[1,2]}) \arrow[r] \arrow[u] & {[1,2,1]} \arrow[u, Rightarrow, "\neq"'] \\
 & {} \arrow[rr] & & \nu = 1 \arrow[u]
\end{tikzcd}$$

$$\Downarrow(1)$$

$$\begin{tikzcd}
{} \arrow[d] & {} \arrow[l] & {} \arrow[l, Leftarrow, "{[1,2,3]}"'] \\
({[1,2]} \arrow[r] \arrow[d] & {[1,2]}) \arrow[r] \arrow[u] & {[1,2,2]} \arrow[u, Rightarrow, "\neq"'] \\
{} \arrow[rr] & & \nu = 2 \arrow[u]
\end{tikzcd}$$

$$\Downarrow(2)$$

$$\begin{tikzcd}
{} \arrow[d] & {} \arrow[l] & & \\
({[1,2]} \arrow[r] \arrow[d] & {[1,2]}) \arrow[r] \arrow[u] & {[1,2,3]}: & \boxed{= \text{stop}} \\
{} \arrow[rr] & & \nu = 3 \arrow[u] &
\end{tikzcd}$$

$$\Downarrow(3)$$

$$(\text{collect}): \{[1,2,1], [1,2,2], [1,2,3]\}$$

1.2.2. Bifunctoriality of diamond recursion

In contrast to classical approaches, morphogrammatic calculations are irreducibly polysemic, delivering different results depending on the complexity of the calculated objects. But this kind of polysemy is conceived as simultaneous and its different aspects are seen as mediated.

On the other hand, it seems reasonable to interpret the results as changing position in the contextural grid only if accretive operations are involved. Hence, a so called "*transkontexturale Überschreitung*" (Gunther) is connected with a succession of an *iterative* and an *accretive* prolongation. After Gunther, a trans-contextural transition happens only if a succession is defined by a connection of iterative and accretive successors.

Therefore, accretion shall be an indicator for a change of position in the contextural grid.

Mathematically this shall be modeled in the framework of polycontextural functoriality.

A technical metaphor might be the change of position in a *poly-layered memristive crossbar system*.

Polyfunctorial modeling

Following strictly the wording of the '*trans-contextural transition*', a modeling is proposed that takes both parts, the iterative and the accretive, as separated, and distributed over the polycontextural grid. The mediation of the parts is guaranteed by the polycontexturality of the grid as such. But there is not yet a direct mediation of the parts intended nor is their any interaction possible.

This Guntherian strategy might still be regulated by a restricted concept of mediation that is not yet taking the *tabularity* of polycontexturality into account but follows a hierarchy of reflexional levels (Reflexionsstufen, Gunther).

This situation is mapped onto the polycontextural concept of distributed levels of bifunctoriality.

The "*hat*" in the examples sketches the general framework of polyfunctoriality. There are 3 dis-contextural (strictly disjunct) universes (Grothendieck) involved, symbolized as $\mathcal{U}^{(3)}$. All 3 universa are mediated by "$\amalg$" , and are incorporating functors *f* and g.

The "*head*" has the design of the distributed functors f_i, g_i, i=1,2,3. While the "*body*" formulates the mediated bifunctoriality of the functors. The indices are indicating the loci of the functors in the polycontextural grid,

Interchangeability of a 3 – contextural category with composition $(\circ)$ and mediation $(\amalg)$

Hat:

$$\mathcal{U}^{(3)} = \left(\mathcal{U}_1 \amalg_{1.2} \mathcal{U}_2\right) \amalg_{1.2\;\;.3} \mathcal{U}_3$$

$$\left(\mathcal{U}_1 \cap_{1.2} \mathcal{U}_2\right) \cap_{1.2\;\;.3} \mathcal{U}_3 = \emptyset :$$

$$\mathcal{U}_i = \left\{f_i, g_i\right\},\ i = 1, 2, 3$$

Head:

$$\left[\begin{matrix} g_1 & - & g_3 \\ f_1 & g_2 & - \\ - & f_2 & f_3 \end{matrix}\right]:$$

Body:

$$\left(\begin{matrix}\left(\begin{matrix}\left(f_1 \circ_{1.0\;\;.0} g_1\right)\\ \amalg_{1.2\;\;.0}\\ \left(f_2 \circ_{0.2\;\;.0} g_2\right)\end{matrix}\right)\\ \amalg_{1.2\;\;.3}\\ \left(f_3 \circ_{0.0\;\;.3} g_3\right)\end{matrix}\right) = \left(\begin{matrix}\left(\begin{matrix} f_1\\ \amalg_{1.2\;\;.0}\\ f_2\end{matrix}\right)\\ \amalg_{1.2\;\;.3}\\ f_3\end{matrix}\right) \circ_1 \circ_2 \circ_3 \left(\begin{matrix}\left(\begin{matrix} g_1\\ \amalg_{1.2\;\;.0}\\ g_2\end{matrix}\right)\\ \amalg_{1.2\;\;.3}\\ g_3\end{matrix}\right)$$

A further modeling that is insisting on the *simultaneity,* and not just the parallelism, of the iterative and accretive results is achieved with a '*transpositional*' distribution of the accretive results together with its iterative functorial parts over a tabular concept of polycontexturality.

Interchangeability of a 3 – contextural category with composition, mediation $(\amalg)$ and transposition $(\diamond)$

$$\begin{pmatrix} f_1 \\ \amalg_{1.2} \\ f_2 \diamond_{2.1} f_1 \\ \amalg_{2.3} \\ f_3 \diamond_{3.1} f_1 \end{pmatrix} \begin{bmatrix} \circ_{1.1} & - & - \\ \circ_{2.1} & \circ_{2.2} & - \\ \circ_{3.1} & - & \circ_{3.3} \end{bmatrix} \begin{pmatrix} g_1 \\ \amalg_{1.2} \\ g_2 \diamond_{2.1} g_1 \\ \amalg_{2.3} \\ g_3 \diamond_{3.1} g_1 \end{pmatrix} = \begin{pmatrix} (f_1 \circ_{1.1} g_1) \\ \amalg_{1.2} \\ (f_2 \circ_{2.2} g_2) \diamond_{2.1} (f_1 \circ_{2.1} g_1) \\ \amalg_{2.3} \\ (f_3 \circ_{3.3} g_3) \diamond_{3.1} (f_1 \circ_{3.1} g_1) \end{pmatrix}$$

There also might be a good reason to understand accretion, i. e. interaction, not just as a transposition but as a *replication* in the sense of a *reflectional* use of the accretive part of the calculation.

Interchangeability and replication $(\square)$

$$\begin{pmatrix} f_1 \square_{1.2} f_1 \\ \amalg_{1.2} \\ f_2 \\ \amalg_{2.3} \\ f_3 \end{pmatrix} \begin{bmatrix} [\circ_{1.1} & \circ_{1.2}] & - - \\ - & \circ_{2.2} & - \\ - - & & \circ_{3.3} \end{bmatrix} \begin{pmatrix} g_1 \square_{1.2} g_1 \\ \amalg_{1.2} \\ g_2 \square_{2.1} g_1 \\ \amalg_{2.3} \\ g_3 \end{pmatrix} =$$

$$\begin{pmatrix} ((f_1 \circ_{1.1} g_1) \square_{1.2} (f_1 \circ_{1.2} g_1)) \\ \amalg_{1.2} \\ (f_2 \circ_{2.2} g_2) \square_{2.1} (f_1 \circ_{2.1} g_1)) \\ \amalg_{2.3} \\ (f_3 \circ_{3.3} g_3) \end{pmatrix}$$

And obviously, it could be necessary to distribute the accretive part of the operation over the *replicative* and the *transpositional* dimension of the grid too.

Interpretation

operator: *succ*,
operand: [1,2],
operation:

$$\text{succ}\left[1,2\right] = \left\{\left[1,2,1\right],\left[1,2,2\right],\left[1,2,3\right]\right\} :$$

$$\text{succ}_{\text{iter}}\left[1,2\right] = \left\{\left[1,2,1\right],\left[1,2,2\right]\right\} : \left(f_{1} \circ_{1.0\quad .0}\, g_{1}\right)$$

$$\text{succ}_{\text{accr}}\left[1,2\right] = \left\{\left[1,2,3\right]\right\} \qquad\qquad : \left(f_{2} \circ_{2.0\quad .0}\, g_{2}\right)$$

Null

Polycontextural mediation of succ$\left[1,2\right]$

$$\left(\begin{array}{c} \left(\begin{array}{c} \left(\text{succ}_{\text{iter}} \circ_{1.0\quad .0}\left[1,2\right]\right) \\ \coprod_{1.2\quad .0} \\ \left(\text{succ}_{\text{accr}} \circ_{0.2\quad .0}\left[1,2\right]\right) \end{array}\right) \\ \coprod_{1.2\quad .3} \\ \text{comp}\left(\text{succ}_{\text{iter}}, \text{succ}_{\text{accr}}\right) \end{array}\right) =$$

$$\left(\begin{array}{c} \left(\begin{array}{c} \text{succ}_{\text{iter}} \\ \coprod_{1.2\quad .0} \\ \text{succ}_{\text{accr}} \end{array}\right) \\ \coprod_{1.2\quad .3} \\ \text{comp}_{1.2\quad .3} \end{array}\right) \circ_{1} \circ_{2} \circ_{3} \left(\begin{array}{c} \left(\begin{array}{c} \left[1,2\right] \\ \coprod_{1.2\quad .0} \\ \left[1,2\right] \end{array}\right) \\ \coprod_{1.2\quad .3} \\ \left(\text{succ}_{\text{iter}}, \text{succ}_{\text{accr}}\right) \end{array}\right)$$

Short notation of succ[1,2] mediation

$$\left(\begin{array}{c} \left(\begin{array}{c} \text{succ}_{\text{iter}}\left[1,2\right] = \left\{\left[1,2,1\right],\left[1,2,2\right]\right\} \\ \coprod_{1.2\quad .0} \\ \text{succ}_{\text{accr}}\left[1,2\right] = \left\{\left[1,2,3\right]\right\} \end{array}\right) \\ \coprod_{1.2\quad .3} \\ \text{comp}\left(\text{succ}_{\text{iter}}, \text{succ}_{\text{accr}}\right) \end{array}\right)$$

This kind of distribution and mediation seems to be quite strait forward and is not posing any interpretational problems at all.

Matrix notation for mediation of succ[1,2]

$$\boxed{\begin{array}{cccc} PM & \mathrm{O}_1 & \mathrm{O}_2 & \mathrm{O}_3 \\ \mathrm{M}_1 & S_{1.1} & - & - \\ \mathrm{M}_2 & - & S_{2.2} & - \\ \mathrm{M}_3 & - & - & S_{3.3} \end{array}} =$$

$$\boxed{\begin{array}{cccc} PM & \mathrm{O}_1 & \mathrm{O}_2 & \mathrm{O}_3 \\ \mathrm{M}_1 & \{[1, 2, 1], [1, 2, 2]\} & - & - \\ \mathrm{M}_2 & - & \{[1, 2, 3]\} & - \\ \mathrm{M}_3 & - & - & \mathrm{comp}(\mathrm{S1}, \mathrm{S2}) \end{array}}$$

Transposition (transjunction)

Polycontextural modeling is not limited to a ' diagonal' interpretation of of the concept of mediation.

Polycontextural mediation and transposition

$$\left(\begin{array}{c} \left(\mathrm{succ}_{\mathrm{iter}} \circ_{1.1} [1, 2]\right) \\ \amalg_{1.2} \\ \left(\mathrm{succ}_{\mathrm{accr}} \circ_{2.2} [1, 2]\right) \diamond_{2.1} \left(\mathrm{succ}_{\mathrm{iter}} \circ_{2.1} [1, 2]\right) \diamond_{1.1} \left(\mathrm{succ}_{\mathrm{iter}} \circ_{3.1} [1, 2]\right) \\ \amalg_{2.3} \\ \left(f_3 \circ_{3.3} g_3\right) \diamond_{3.1} \left(f_1 \circ_{3.1} g_1\right) \end{array}\right)$$

$$=$$

$$\left(\begin{array}{c} \mathrm{succ}_{\mathrm{iter}} \\ \amalg_{1.2} \\ \mathrm{succ}_{\mathrm{accr}} \diamond_{2.1} \mathrm{succ}_{\mathrm{iter}} \\ \amalg_{2.3} \\ \left(f_3 \circ_{3.3} g_3\right) \diamond_{3.1} \left(f_1 \circ_{3.1} g_1\right) \end{array}\right) \left[\begin{array}{ccc} \circ_{1.1} & - & - \\ \circ_{2.1} & \circ_{2.2} & - \\ \circ_{3.1} & - & \circ_{3.3} \end{array}\right] \left(\begin{array}{c} [1, 2] \\ \amalg_{1.2} \\ [1, 2] \diamond_{2.1} [1, 2] \\ \amalg_{2.3} \\ \mathrm{comp}\left(g_3 \diamond_{3.1} g_1\right) \end{array}\right)$$

Short notation for mediation and transposition of succ[1,2]

$$\left(\left(\begin{array}{c} \text{succ}_{\text{iter}}\left[1,2\right]=\left\{\left[1,2,1\right],\left[1,2,2\right]\right\}_{1.1} \\ \amalg_{1.2\quad .0} \\ \text{succ}_{\text{accr}}\left[1,2\right]=\left\{\left[1,2,3\right]\right\}_{2.2}\diamond_{2.1}\left(\text{succ}_{\text{iter}}\circ_{\ 2.1}\left[1,2\right]\right)=\left\{\left[1,2,1\right],\left[1,2,2\right]\right\}_{2.1}\diamond_{3.1}(\text{s} \\ \amalg_{1.2\quad .3} \\ \text{comp}\left(\text{succ}_{\text{iter}},\text{succ}_{\text{accr}}\right)_{3.3} \end{array}\right.\right.$$

$$\left(\left(\begin{array}{c} \text{succ}_{\text{iter}}\left[1,2\right]=\left\{\left[1,2,1\right],\left[1,2,2\right]\right\} \\ \amalg_{1.2\quad .0} \\ \left(\text{succ}_{\text{accr}}\left[1,2\right]\right)\diamond_{2.1}\left(\text{succ}_{\text{iter}}\circ_{2.1}\left[1,2\right]\right)\diamond_{3.1}\left(\text{succ}_{\text{iter}}\circ_{3.1}\left[1,2\right]\right) \\ \amalg_{1.2\quad .3} \\ \text{comp}\left(\text{succ}_{\text{iter}},\text{succ}_{\text{accr}}\right) \end{array}\right)\right)$$

This kind of transpositional mediation is probably not self-evident at a first glance. Because the succession function is interpreted not just as accretive but also as 'transjunctive', i.e. transpositional, and its direct result set at the locus $S_{1.1}$ gets an additional *interpretation* at the loci $S_{2.1}$and $S_{3.1}$.

Operations and their results in polycontextural systems are always *interpreted* results and not just prima facie objects. Interpretation is a mode of iterability, therefore the objects of interpretation are repeated, modeled, thematized as something that differs from the 'given' object of thematization. Iterability involves memory. Also the 'same is different', it its *memory* function guarantees that it is the same and not something totally different.

Therefore, different representations of the interpretations have to be registered as differently localized. Each has a different meaning even if they look the same, syntactically, or are having abstractly the same mathematical definition.

Matrix notation for mediation and transposition of succ[1,2]

PM	O_1	O_2	O_3
M_1	$S_{1.1}$	$S_{2.1}$	$S_{3.1}$
M_2	–	$S_{2.2}$	–
M_3	–	–	$S_{3.3}$

=

PM	O_1	O_2	O_3
M_1	$\{[1,2,1],[1,2,2]\}$	$\{[1,2,1],[1,2,2]\}$	$\{[1,2,1],[1,2,2]\}$
M_2	–	$\{[1,2,3]\}$	–
M_3	–	–	$\text{comp}\left(\text{succ}_{\text{iter}}, \text{succ}_{\text{accr}}\right)$

Interpretation for replication

The mechanism of interpretation of *replication* works along the steps demonstrated for the transpositional case.

The same holds for *mixed* situation of replication and transposition. Additionally, a new kind of iterative intervention is introduced with the *iteration* of an object at the same contextural locus.

Matrix for replication

PM	O_1	O_2	O_3
M_1	$S_{1.1}$	x	x
M_2	$S_{1.2}$	$S_{2.2}$	x
M_3	$S_{1.3}$	x	$S_{3.3}$

Matrix for transposition and replication

PM	O_1	O_2	O_3
M_1	$S_{1.1}$	$S_{2.1}$	$S_{3.1}$
M_2	$S_{1.2}$	$S_{2.2}$	–
M_3	$S_{1.3}$	–	$S_{3.3}$

Matrix for 'fractalization' (iteration) and replication

PM	O_1	O_2	O_3
M_1	$S_{1.1}$	$S_{2.1}$	–
M_2	$S_{1.2}$ $_{.2}$ $_{.2}$	$S_{2.2}$	$S_{3.2}$
M_3	–	$S_{2.3}$	$S_{3.3}$

Accretion and mediation

With each accretion of a calculation the range of polycontexturality might be augmented. For the *balanced* matrix, m and n are equal.

$$\text{succ}_{\text{accr}}\left(\text{PM}^{(m,n)}(\text{O, M})\right) \Longrightarrow \left(\text{PM}^{(m+1,\,n+1)}(\text{O, M})\right)$$

$$\text{succ}_{\text{iter}}\left(\text{PM}^{(m,n)}(\text{O, M})\right) \Longrightarrow \left(\text{PM}^{(m,n)}(\text{O, M})\right)$$

A more differentiated appraoch is considering, as far as possible, the concept of *monomorphies* of morphograms.

Example for monomorphies

```
- kconcat [1,2,1,3,2,2][1];
val it = [[1,2,1,3,2,2,1],[1,2,1,3,2,2,2],[1,2,1,3,2,2,3],[1,2,1,3,2,2,4]] :
int list list
```

MG0 =[1,2,1,3,2,2]

MG4 =[1,2,1,3,2,2,4]

MG	loc_1	loc_2	loc_3	loc_4	
Dec	mg_1	mg_2	mg_1	mg_3	mg_2
MG^1	1	–	1	–	–
MG^2	–	2	–	–	–
MG^3	–	–	–	3	–
MG^2	–	–	–	–	22

$\Longrightarrow$

MG	loc_1	loc_2	loc_3	loc_4	loc_5	
Dec	mg_1	mg_2	mg_1	mg_3	mg_2	mg_4
MG^1	1	–	1	–	–	–
MG^2	–	2	–	–	–	–
MG^3	–	–	–	3	–	–
MG^2	–	–	–	–	22	–
MG^4	–	–	–	–	–	4

MG3 =[1,2,1,3,2,2,3]

MG1 =[1,2,1,3,2,2,1]

MG2 = [1,2,1,3,2,2,2]

MG	loc_1	loc_2	loc_3	loc_4	loc_5	
Dec	mg_1	mg_2	mg_1	mg_3	mg_2	mg_3
MG^1	1	–	1	–	–	–
MG^2	–	2	–	–	–	–
MG^3	–	–	–	3	–	–
MG^1	–	–	–	–	22	–
MG^4	–	–	–	–	–	3

MG	loc_1	loc_2	loc_3	loc_4	loc_5	
Dec	mg_1	mg_2	mg_1	mg_3	mg_2	mg_1
MG^1	1	–	1	–	–	–
MG^2	–	2	–	–	–	–
MG^3	–	–	–	3	–	–
MG^2	–	–	–	–	22	–
MG^4	–	–	–	–	–	1

MG	loc_1	loc_2	loc_3	loc_4	
Dec	mg_1	mg_2	mg_1	mg_3	mg_2
MG^1	1	–	1	–	–
MG^2	–	2	–	–	–
MG^3	–	–	–	3	–
MG^1	–	–	–	–	222
MG^4	–	–	–	–	–

1.2.3. Programming aspects

Recursion

```
scala> def factorial(number:Int) : Int = {
     |     if (number == 1)
     |        return 1
     |     number * factorial (number - 1)
     | }
factorial: (number: Int)Int
scala> println(factorial(5))    120
```

"The new accumulator parameter stores the intermediate value, so we are no longer doing a calculation against the value returned from the function like we were before."

```
scala> def factorial(accumulator: Int, number: Int) : Int = {
     |   if (number == 1)
     |     return accumulator
     |   factorial (number * accumulator, number - 1)
     | }
factorial: (accumulator: Int, number: Int)Int
scala> println(factorial(1,5))     120
```

More for Java at:
http://stackoverflow.com/questions/8183426/factorial-using-recursion-in-java

Recursion on the base of the retro-gradeness of morphograms

```
scala> def mg-factorial(morphogram:Morph List Int) : Morph = {
          |  if (morphogram == [ ])
          |     return [ ]
          |  kmul [morphogram (n)][morphogram (factorial(n-1))]
          | }
mg-factorial: (morphogram: Morph) Morph List
```

```
fun **allTcontextureFac** n =
    allkmul (Tcontexture (n)) (Tcontexture (fac (n - 1)));
val allTcontextureFac = fn : int -> int list list list

fun **TcontextureFacNum** n = Tcontexture (fac n);
val TcontextureFacNum = fn : int -> int list list
```

Example

- **allTcontextureFac** 3;

```
val it =
  [[[1,1,1,1,1,1]],[[1,1,1,2,2,2]],[[1,1,2,1,1,2]],

[[1,1,2,2,2,1],[1,1,2,3,3,1],[1,1,2,2,2,3],[1,1,2,3,3,4]],[[1,2,1,1,2,1]],

[[1,2,1,2,1,2],[1,2,1,3,1,3],[1,2,1,2,3,2],[1,2,1,3,4,3]],[[1,2,2,1,2,2]],

[[1,2,2,2,1,1],[1,2,2,3,1,1],[1,2,2,2,3,3],[1,2,2,3,4,4]],[[1,2,3,1,2,3]],
  [[1,2,3,2,3,1],[1,2,3,3,1,2],[1,2,3,2,1,4],[1,2,3,2,4,1],[1,2,3,4,1,2],
   [1,2,3,3,1,4],[1,2,3,3,4,1],[1,2,3,4,3,1],[1,2,3,4,1,5],[1,2,3,4,5,1],
   [1,2,3,2,3,4],[1,2,3,3,4,2],[1,2,3,4,3,2],[1,2,3,2,4,5],[1,2,3,4,5,2],
   [1,2,3,3,4,5],[1,2,3,4,3,5],[1,2,3,4,5,6]]] : int list list list
```

Reflectional analysis of allTcontextureFac 3

EWhat we get with this approach of a reflectianal analysis of morphic factorials, allTcontextureFac 3, is a 6 layered system of mediated contextures

S1: [1,1,1,1,1,1],
S2: [1,1,1,2,2,2],[1,1,2,1,1,2], [1,2,1,1,2,1],
[1,1,2,2,2,1], [1,2,1,2,1,2], [1,2,2,1,2,2],[1,2,2,2,1,1]

S3: [1,1,2,3,3,1],[1,1,2,2,2,3],[1,2,1,3,1,3],[1,2,1,2,3,2],
[1,2,2,3,1,1],[1,2,2,2,3,3],
[1,2,3,1,2,3],[1,2,3,2,3,1],[1,2,3,3,1,2],[[1,2,3,1,2,3],

S4:
[1,2,1,3,4,3],[1,1,2,3,3,4],[1,2,2,3,4,4],[1,2,3,2,1,4],[1,2,3,2,4,1],[1,2,3,4,1,2],
[1,2,3,3,1,4],[1,2,3,3,4,1],[1,2,3,4,3,1],,[1,2,3,2,3,4],
[1,2,3,3,4,2],[1,2,3,4,3,2],

S5:[1,2,3,4,1,5],[1,2,3,4,5,1],[1,2,3,4,1,5],[1,2,3,4,5,1],[1,2,3,2,4,5],[1,2,3,4,5,2],
[1,2,3,3,4,5],[1,2,3,4,3,5],

S6:[1,2,3,4,5,6].

This classification corresponds to fac n: fac 3 = 6.

A **reduction** of allTcontextureFac 3 to the **deutero**-structure is given by the following procedure.

setof(map dnf(flat(allTcontextureFac 3)));

```
[[1,1,1,1,1,1],
[1,1,1,1,2,2],[1,1,2,2,2,2], [1,1,1,2,2,2]
[1,1,1,2,3,3],[1,1,2,2,2,3],[1,1,1,2,2,3],[1,2,2,2,3,3],[1,1,2,2,3,3],
[1,2,2,3,4,4],[1,1,2,2,3,4],[1,1,2,3,3,4],[1,2,2,3,3,4],
[1,1,2,3,4,5],[1,2,2,3,4,5],[1,2,3,3,4,5],
[1,2,3,4,5,6]].
```

A further genuin reduction to its **proto**-structure is achieved with the following procedure:

- setof(map pnf(flat(allTcontextureFac 3)));

```
[[1,1,1,1,1,1],
[1,1,1,1,1,2],
[1,1,1,1,2,3],
[1,1,1,2,3,4],
[1,1,2,3,4,5],
[1,2,3,4,5,6]] .
```

2. Memristivity and memory

"Well, each time we make a recursive call, we 'eat up' a bit more space on the stack. "

http://www.researchgate.net/publication/221504050_Fractional-order_Memristive_Systems

This fact gets replaced by the memristive fact: Each time we make a recursive call, we continue the call at the place it stopped before. At this place the value of the *history* is stored (memorized) by the memristive memory inside the previous call. The structure of this memorized 'state' dictates how and to what range further operations might be applied.

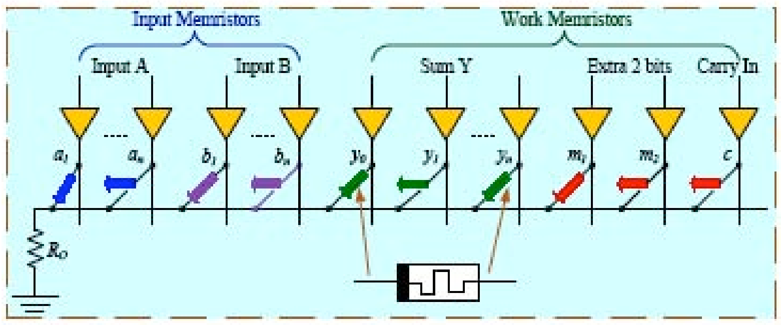


Figure 10: A memristor-based n-bit adder.

Saraju P. Mohanty, Memristor: From Basics to Deployment

> *"Digital design using memristor will need lots of research effort as Boolean logic can't be directly implemented."*

Memristance is a property of an electronic component to retain its resistance level even after power had been shut down or lets it remember (or recall) the last resistance it had before being shut off.

Eero Lehtonen, Memristive Computing, 2012

> *"The main conclusion of this thesis is that memristive computing will be advantageous in large-scale, highly parallel mixed-mode processing architectures. This can be justified by the following two arguments.*
> *First, since processing can be performed directly within memristive memory architectures, the required circuitry, processing time, and possibly also power consumption can be reduced compared to a conventional CMOS implementation.*
> *Second, intrachip communication can be naturally implemented by a memristive crossbar structure."*

http://www.doria.fi/bitstream/handle/10024/79925/AnnalesAI446LehtonenDISS.pdf

I have argued in previous papers that memristivity is making a crucial difference on a *paradigmatic* level and is not depending on such complex properties of systems like "*large-scale, highly parallel mixed-mode processing architectures*".

Even the simplest step of iteration, especially for recursion, has two different forms of conceptualization and realization: the repetitive "*stack-oriented*" and the memristive "*history-oriented"* approaches.

> *"Memristive architectures are ideally suited for computation within a memory, and thus memristors should not be regarded only as memory, but also as nanoscale computing units."* (Lehtonen)

The 'history'-oriented approach gets its development with the application of patterns, i.e. morphograms, that have to be calculated. On the level of classical atomistic elements, say numbers or symbols, the merits of memristivity as a possibility to go beyond classical computing are not yet accessible.

In other words, memristivity leads into a new paradigm of computation if applied on patterns of constellations. Hence memristics is surpassing the possibilities of isolated memristors if it comes as patterns of activity instead of a system of single memristors.

Early papers
http://www.thinkartlab.com/pkl/lola/Memristics/Memristics:Memristors, again.pdf
http://www.thinkartlab.com/pkl/lola/Memristics/Part-II/Memristics-crossbar.pdf
http://memristors.memristics.com/Why-Not/Why-Not.html

3. Recursivity and memristive memory

Memristive recursivity continues where it ended without the require of a separated memory devise, a stack or pile, for the continuation of the recursion.

Memristive recursivity depends on the memristance of the memristive system and not on a separated memory.

Memristance is an intrinsic property of memristive systems. Thus it has not to be added from the outside. There is no glue included.

Because of the chiastic interplay of distributed and mediated computation and memory the separation of both is arbitrary for memristive systems.

Victor Erokhin, 2012

> *"Given the property of memristance, it seems more interesting to design new systems using these new functional properties e.g. the capability to memorize the history of inputs to the device. In this respect, the output will be not a simple binary decision, but some intermediate value, that depends on the duration of the inputs. Such processing is to some degree similar to a brain-like logical decision making process: an answer (YES / NO) depends not only on the configuration of external stimuli, but also on past experience."*

http://arxiv.org/pdf/1212.3425v1.pdf

4. Memristive electronic devices

J. Joshua Yang, Dmitri B. Strukov and Duncan R. Stewart, Memristive devices for computing

> *"Memristive devices are electrical resistance switches that can retain a state of internal resistance based on the history of applied voltage and current. These devices can store and process information, and offer several key performance characteristics that exceed conventional integrated circuit technology.*

> *"Here, we focus on the chemical and physical mechanisms of memristive devices, and try to identify the key issues that impede the commercialization of memristors as computer memory and logic."*

https://www.ece.ucsb.edu/~strukov/papers/2013/NatNano2013.pdf

http://www.cse.unt.edu/~smohanty/Publications_Journals/2013/Mohanty_IEEE-Potentials_2013May_Memristor.pdf