Notes on semi-Thue Systems in a Context of Morphogrammatics

Further explanations of the formal notions behind morphic cellular automata

Rudolf Kaehr Dr.phil
 ISSN 2041-4358

Abstract

The main differences between symbolic and morphic formal systems: Instead of the Kleene star for the symbolic universes, morphic universes are generated by the Stirling cross. Symbolic substitution and concatenation is preserving production concatenation. Morphic substitution/concatenation is opening up a system of interactive complexions of derivations. Comparison of substitution based production systems (Thue, Post, Markov) with Hausser's systems of "possible continuations" of Left-Associative languages is sketched.

1. Semi-Thue Systems

1.1. Production systems

1.1.1. Deconstruction remarks

"It is a gross simplification to view languages as sets of strings. The idea that they can be defined by means of formal processes did not become apparent until the 1930s. The idea of formalizing rules for transforming strings was first formulated by Axel Thue *(1914). The observation that languages (in his case formal languages) could be seen as generated from semi Thue systems, is due to* Emil Post. *Also, he has invented independently what is now known as the Turing machine and has shown that this machine does nothing but string transformations. [...] The idea was picked up by* Noam Chomsky *and he defined the hierarchy which is now named after him (see for example (Chomsky, 1959), but the ideas have been circulating earlier)."* Marcus Kracht 2003, The Mathematics of Language, Rewriting Systems, p. 53

"In formal language theory, languages are sets of strings over some alphabet. We assume throughout that an alphabet is a finite,

> *nonempty set, usually called A. It has no further structure, it only defines the material of primitive letters.*" (ibd, p. 16)
> http://www.lix.polytechnique.fr/~bournez/MPRI/formal.pdf

A deconstruction of "*sign*", "*string*" and "*set*" is necessary to understand morphogrammatics and morphogrammatic semi-Thue systems, morphic finite state machines and morphic cellular automata as introduced in recent papers. A further deconstruction has to go into the topics of finiteness and infiniteness of alphabets and strings. It also has to be seen that the term "*sign*" is understood as a purely syntactical mark, letter or character and is not involved in any serious semiotical distinctions.

The kernel of formal language considerations is the monoid, $\mathcal{M}$ = (M, ◦, 1) and the Kleene (star) production A*.

A monoid is a triple $\mathcal{M}$ = (M, ◦, 1) where : "◦" is a binary operation on M and 1 an element such that for all x, y, z ∈ M
the following holds.
x ◦ 1 = x (Idempotence)
1 ◦ x = x (Idempotence)
(x ◦ y) ◦ z = x ◦ (y ◦ z) (Associativity).

Hence, a deconstruction of a monoid $\mathcal{M}$ has firstly to deconstruct the *binary operation* (composition) "◦" and then, secondly, more or less as a consequence of it, a deconstruction of the *elements* of $\mathcal{M}$.

A deconstruction of the concept of set-theoretical elements has led to the introduction of a new 'data-type', the kenogrammatic and morphogrammatic data patterns used in kenomic and morphic cellular automata constructions.

Criticism: Just an abstraction more?
At a first glance it seems that such a deconstruction which leads from the Kleene product to a Stirling distribution might simply be an *abstraction* over the set of values producing an equivalence class as it is well known. Hence, Stirling $K^* = \Sigma^*/_{eq}$. There are some academic publications insisting on such profound insight. Furthermore it is trivial to conclude that the same abstraction holds for the introduction of kenomic cellular automata: kenoCA = $ECA/_{eq}$. In such a view the kenomic rules are just an abstraction of the CA rules. If we consider the situation for $ECA^{(3,2)}$ with the *complete* rule set 2^3= 8 and a complete rule range of 2^{2^3} = 256 and the correspond-

ing kenoCA$^{(3,2)}$ with an incomplete rule set of StirlingSn2(3, 2) = 4 and an incomplete rule range of StirlingSn2(2^3, 2) = 128, then results look quite trivially as an abstraction from 2^3 to $2^3/2 = 4$ and 2^{2^3} to $2^{2^3}/2 = 128$. Unfortunately, the complete rule set for the elementary kenoCA$^{(3,4)}$ is StirlingSn2(4, 4) = 15 and not 8. As a consequence of this asymmetry between complete rule sets, different kinds of rules, methods and features are surpassing the classical definitions of CAs without the Stirling approach those new constellations wouldn't be accessible. •

There is not much chance to achieve such a transformation of the concept and functioning of an elementary operation like the *composition* "◦" in a monoid.

Nevertheless, there are some still recent but well elaborated and tested approaches to recognize. The diamondization of composition has been demonstrated in my papers to a *Diamond Category Theory*.

With " x ◦ 1 = x" and "1 ◦ x = x" it follows that the equation "x ◦ 1 = x = 1 ◦ x" holds. This is not surprising and has its rock solid foundations in first-order logic and category theory and their epistemologies.

Does it hold for morphogrammatics? Obviously not! The equation might be interpreted as the equality of right- and left-oriented self-identity of the object "x" of a morphism.

$$(X \circ 1) \,|\, x = X \,|\, x \qquad (\text{Diamond idempotence})$$
$$(1 \circ X) \,|\, x = x \,|\, X \qquad (\text{Diamond idempotence})$$

$$f_X \circ f_{id} \Rightarrow \begin{pmatrix} X \in \text{Iter} \\ X \in \text{Accr} \end{pmatrix}$$

http://www.thinkartlab.com/pkl/lola/Semiotics-in-Diamonds/Semiotics-in-Diamonds.html

Hence, even the simplest presumbtion, namly that X = X has to be deconstructed.
As a consequence, the obvious symmetry of A = B iff B = A is not obvious anymore.

A deconstruction of associativity of composition follows, at first, quite automatically:
The context-independent associativity "(x ∘ y) ∘ z = x ∘ (y ∘ z)" becomes the contextualized associativity
"(X ∘ Y)|(x; y) ∘ Z | z = X | x ∘ (Y ∘ Z)|(y; z)".

Diamondization of associativity of composition

$$\forall x, y, z: \left(x \circ y\right) \circ z =_{SEM} x \circ \left(y \circ z\right)$$

$$\forall X, Y, Z; \forall x, y, z, u: \left(\left(X \circ Y\right) \middle| \left(x; y\right) \circ Z \middle| z\right) \middle| u =_{DIAM} \left(X \middle| x \circ \left(Y \circ Z\right) \middle| \left(y\right.\right.$$

This has consequences for any introductory rule like R_0: ⟶ X.
There is no simple beginning in a diamond world. Setting a beginning is always multiple, at least double: a beginning as an iterative or an accretive beginning. The act of beginning happens in a context of a beginning and has its own notion in a calculus of beginnings.

Hence, R_0: ⟶ X becomes diamond R_0: ⟶ X | x.

Therefore, the statement of a beginning kenogram [kg] of kenogrammatic sequences in a trito-universe TU as in TU = ([1] Tsucc) of a recursive formula is just a beginning of the process of deconstruction of the notions and terms of keno- and morphogrammatics and not an end at all.

Nevertheless, diamond-theoretic thematizations had been, more or less, omitted in the proposals on kenomic and morphic cellular automata, finite-state machines and semi-Thue systems.

And just the Stirling effect is in focus that is affecting the rules of the morphogrammatic game of semi-Thue systems and cellular automata deconstruction.

Hence, the universe of trito-structural kenogram sequences, kgs, TU, remains defined without its diamond environment as
TU = [[1] Tsucc], with x + 0 = 0 + x = x.

Further deconstructions of the concept of ‘beginnings’ in formal systems at:

"Quadralectic Diamonds: Four-foldness of beginnings",
http://www.thinkartlab.com/pkl/lola/Quadralectic%20Diamonds/Quadralectic%20Diamonds.pdf

1.1.2. Langton's rules for simple linear growth

A classical example of a production system is introduced by Langton's L-system.

> *"Here is an example of the simplest kind of L-system. The rules are* context free*, meaning that the context in which a particular part is situated is* not *considered when altering it. There must be only one rule per part if the system is to be* deterministic.

The rules: (the "recursive description" of a GTYPE)

1) A ⟶ CB
2) B ⟶ A
3) C ⟶ DA
4) D ⟶ C

When applied to the initial seed structure "A," the following structural history develops (each successive line is a successive time step):

time	structure	rules applied (L to R)
1.	A	: start
	↓	
2.	CB	: rule1 on 1.
	↙↘	
3.	DA A	: rule3 on 2. C, rule2 on 2. B
	↙ ↓ ↘	
4.	C CB CB	: rule4 on 3. D, rule1 on 3. A, rule1 on 3. A

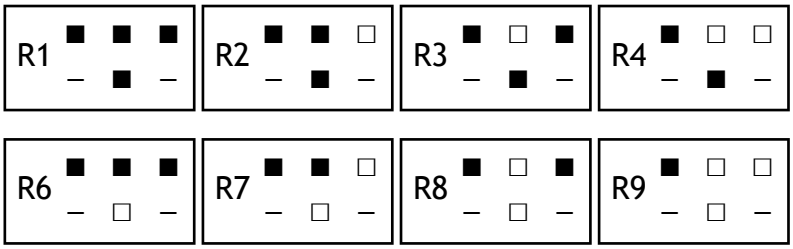
Christopher Langton, Artificial Life, 1989, p. 26

.l	1	2	3	4	**5**	6	7	8	9	rule = rule1 .2 .3 .4
0	□	□	□	□	***A***	□	□	□	□	0) initial " seed "
1	□	□	□	*C*	–	*B*	□	□	□	1) rule 1 replaces *A* with CB
2	□	□	*D*	–	*A*	□	*A*	□	□	2) rule 3 : *C* with DA, rule 2 : *B* with *A*
3	□	*C*	□	*C*	–	*B*	*C*	–	*B*	3) rule 4 : *D* with *C*; rule 1 : AA with CB's
4	□	□	□	□	□	□	□	□	□	stop

Atomic elements are substituted by unary and binary elements. Binary elements are seen as a concatenation of 2 identical unary elements. Because of this atomism or elementarism a kenomic abstraction is empty,

i.e. all unary elements are kenomically equivalent.
Because rewritting systems are *substitutional* systems the point of substitution in this case is atomistic.

kenoCA$^{(2)}$ rule set



Example for kenoCA rules of the form: [axb] ⟶ y
rule1: [AAA] ⟶ A
rule7: [AAB] ⟶ B
rule8: [ABA] ⟶ B
rule4: [ABB] ⟶ A

Nr.l	1	2	3	4	**5**	6	7	8	9	rule = rule1 .7 .8 .4
0	□	□	□	*x*	*A*	*x*	□	□	□	0) *A* initial "seed" (0; 5)
1	□	□	*x*	*B*	*B*	*A*	*x*	□	□	R7(0; 3, 4, 5) : BBA | *B*, R8(0; 4, 5, 6) : ABA | *B*, R4(0; 5,
2	□	*x*	*A*	*A*	*B*	*B*	*A*	*x*	□	R1(1; 2, 3, 4), r1(1; 3, 4, 5) : BBB | *A*, R7(1; 4, 5, 6)
										R8(1; 5, 6, 7) : ABA | *B*, R4(1; 6, 7, 8) : ABA | *B*

kenoCA

Nr.l	1	2	3	4	**5**	6	7	8	9	rule = rule1 .7 .8 .4
1	□	□	□	□	■	□	□	□	□	R7(1; 3, 4, 5), R8(1; 4, 5, 6), R4 (1; 5, 6, 7)
2	□	□	□	*x*	*x*	■	□	□	□	1, 1, 7, 8, 4
3	□	□	■	■	*x*	*x*	■	□	□	7, 4, 7, 4, 7, 8, 4
4	□	*x*	■	*x*	■	*x*	*x*	■	□	1, 7, 8, 8, 8, 4, 7, 8, 4
5	■	*x*	*x*	*x*	*x*	■	*x*	*x*	■	stop

1.1.3. Introducing kenogrammatic rules

Technical alphabet, standard normal form of kenograms: {A, B, C}.
Rules: rule1, rule2, rule3.

One trito-equivalence of the calculus, not applicable to the technical, meta-linguistic alphabet:
A = $_{KG}$C:

P_{KG} = [mode=KG; {A, B, C}, rule1, rule2, rule3]
rule1: ⟶ A

rule2: A ⟶ AB
rule3: AB ⟶ C

A, AB, C, AB, C, AB, ...: chiastic interchange between A(2) and C(3).

The term A(1) as operator is set, C(3) as operand of the operator AB (3) becomes the operator A(2). Hence, A(2) is involved in the chiasm of operator and operand, playing both roles at once. Considering the roles of C, the same holds for a B in place of C.

Différance and memristivity

Strictly speaking, we encouter with this tiny P_{KG} – system a situation where both, the *halt* and the *continuation* of the production, happens at once. The chiastic interplay of the situation "A" and the situation "C" is playing the *différance* of the difference of "A" and "C" and its defer of change from "C" to "A". Jaques Derrida's différance, which is neither a word nor a term, and is phoneticlly indistinguishable from "différence", plays on the fact that the French word *différer* means both "*to defer*" and "*to differ*." *"Différance as* temporization*, différance as* spacing. *How are they to be joined?"* (J. Derrida)
Jacques Derrida, Différance, http://www.stanford.edu/class/history34q/readings/Derrida/Differance.html

This mechanism of a *chiastic* interchange between A and C invites to interpret it as a *memristive* mechanism and probably as the smallest model of non-destructive self-referentiality in/of a formal system. The *self-referentiality* of the production scheme seems to be obvious. What isn't obvious at a first glance is its *memristivity*. Memristivity is involved with the chiasm between 'operand' C and 'operator' A. The property of re-entry and sameness has to be remembered during the substitution. In a classical setting, nothing of this kind has to be reached because it is presumed and installed from the 'outside' by an external designer/user of the rules that the re-entry 'port' is not missed and that the object has not changed in the process of substitution from one identity (A/C) to another identity (C/A).

For the kenomic calculus, the *technical* alphabet is build by distinctive letters, characters, elements, but inside the kenomic game and its rules, all occurrences of monadic elements are kenomically equal.

The substitution of C from rule3 to rule2 as A has the *choice* to decide for a kenomic or for a symbolic interpretation of the substitution. With a *symbolic* interpretation the calculus stops here because the application is

refused. For a *kenomic* interpretation rule2 holds, and the game goes on. Hence, with A $\neq_{SEM}$ C, the semiotic rule system is terminating with C, and with A $=_{KG}$ C the production goes on with C $=_{KG}$ A.

Hence, the *general decision problem* gets confronted with an as yet unknown situation of a rewriting system having properly a state and at once not having that state in the calculus.

Consequences for the concepts and constructions of replication, cloning and self-production/production of a self (autopoiesis) have at first to deconstruct the underlying concepts of iterability in their concepts of recursion.

Decidability and non-decidability, therefore, is not focussed on the identification or non-identification of an object, i.e. a state, with decidable or non-decidable properties but on the interaction between applications inside and between formal systems.

Further examples

P_{ID} = [mode=ID; {A, B, C}, rule1, rule2, rule3]
rule1: $\longrightarrow$ A
rule2: A $\longrightarrow$ AB
rule3: AB $\longrightarrow$ C

A, AB, C.

Production systems are based on substitution. Kenomic and morphic substitutions are context-dependent.

Chiastic rule applications are not to be confused with the *identity* systems with A(1) = A(2) in a (self-referential) *circular* rule system:

P_{ID} = [mode=ID; {A, B}, rule1, rule2]
Alphabet = {A, B}
Rules (Id):
rule1: A $\longrightarrow$ AB
rule2: AB $\longrightarrow$ A.

A, AB, A, AB, ... : non-terminating

Logic, Copies and DNA Replication

"In logic there is a level beyond the simple copying of symbols that contains a non-trivial description of self-replication. *The schema is as*

follows: There is a universal building machine B *that can accept a text or description x (the program) and build what the text describes. We let lowercase* x *denote the description and uppercase* X *denote that which is described. Thus* B *with* x *will build* X*. In fact, for bookkeeping purposes we also produce an extra copy of the text x. This is appended to the production* X *as X,* x*. Thus* B*, when supplied with a description* x*, produces that which* x *describes, with a copy of its description attached. Schematically we have the process shown below.*

B, x ⟶ B, x; X, x

Self-replication is an immediate consequence of this concept of a universal building machine. Let b *denote the text or program for the universal building machine. Apply* B *to its own description.*

B, b ⟶ B, b; B, b

The universal building machine reproduces itself. Each copy is a universal building machine with its own description appended. Each copy will proceed to reproduce itself in an unending tree of duplications. In practice this duplication will continue until all available resources are used up, or until someone removes the programs or energy sources from the proliferating machines."

Louis H. Kauffman, Biologic
http://arxiv.org/pdf/quant-ph/0204007

Chiastic self-reference with 2 trito-equivalences: $A =_{KG} C$ and $AB =_{KG} BA$

P_{KG} = [mode=KG, {A, B, C}, rule1, rule2, rule3]
Alphabet = {A, B, C}
rule1: ⟶ A
rule2: A ⟶ AB
rule3 : BA ⟶ C

(1): A, AB, A, AB, A, ...

(2): A, AB, C, AB, C, ...

Also they differ as resulting productions semiotically, both production chains are kenogrammatically equivalent: (1) $=_{KG}$ (2).

Polycontextural productions

Further interesting results are obtained by polycontextural production systems. In this context of *polycontexturality* it is obvious to state *"The same is different"*.

$P_{PCL}^{(3)} = (P^1 \amalg P^2) \amalg P^3 =$ [mode=Poly$K^{(3)}$, $\{A, B, C\}^{(3)}$, (rule1, rule2, rule3)$^{(3)}$], $\amalg$: mediation

Alphabet = $\{A, B, C\}^{(3)}$:

Alph = $\{A, B, C\}^1$	Alph = $\{A, B, C\}^2$	Alph = $\{A, B, C\}^3$
rule1.1 : $\longrightarrow$ A	rule1.2 : $\longrightarrow$ A	rule1.3 : $\longrightarrow$ A
rule2.1 : A $\longrightarrow$ AB	rule2.2 : A $\longrightarrow$ AB	rule2.3 : A $\longrightarrow$ AB
rule3.1 : AB $\longrightarrow$ A	rule3.2 : AB $\longrightarrow$ A	rule3.3 : AB $\longrightarrow$ A

A^1, AB^1, $A^1 = A^2$, $(AB)^2$, $A^2 = A^1$, $(AB)^1$, ...

The *substitution* process distributed between $P^1$and P^2 might be reflected from the third position of P^3 from which it is reasonable to state that there is no classical circular production with ($A^1 = A^2/A^2 = A^1$) but a chiastic self-referentiality between the two mediated contexturally different production systems $P^1$and P^2 albeit both are using the "same" alphabets and the "same" rules.

The questions of termination of programs, calculations, productions and the Halting problem are one side of the classical constellations. The other side is that a non-terminating program has different meanings in the new constellation. Computation as an interactive media is not problem solving and is therefore 'beyond' the classical questions of termination and non-termination. Media of computation don't have a start or an initial configuration nor do they have a terminal goal. Non-termination in polycontextural and morphic systems is not the same as empty repetition, infinite loop or endless iteration in the classical framework.

This hint or metaphoric construction is not excluding the conservation of the classical situations and their results 'inside' the different contextures.

Hence, *non-termination* is not anymore a bad property of 'algorithmic' systems but the intrinsic character of inter-medial activity.

Some further entertainment from "Nick Haflinger":
For a more philosophical intervention, go to "*Nancy: Destruktion als Erinnerung der Struktion oder Techné"* at:
http://player.vimeo.com/video/2846627?title=0&byline=0&portrait=0
Or you might prefer: *"Slickaphonics - Procrastination (Wow Bag - 1983)"* at:
http://www.youtube.com/watch?v=k2F7eWtYwkc

The real thing? Peter Wegner, *Interactive Computation*
http://www.cse.uconn.edu/~dqg/inter_book.html

1.2. Semi-Thue Systems

1.2.1. Definitions for semi-Thue systems

Following *PlanetMath* we get a helpful definition of a semi-Thue system.
http://planetmath.org/encyclopedia/GenerableByASemiThueSystem.html

"A semi-Thue system $\mathfrak{S}$ is a pair (Σ, P) where Σ is an **alphabet** and P is a non-empty **finite binary relation** on Σ^* , the Kleene star of Σ .

Elements of P are variously called *defining relations*, **productions**, or *rewrite rules*, and $\mathfrak{S}$ itself is also known as a *rewriting system*. If $(x, y) \in P$, we call x the *antecedent*, and y the *consequent*.
Instead of writing $(x, y) \in P$ or xPy , we usually write

$$x \longrightarrow y.$$

Let $\mathfrak{S} = (\Sigma, P)$ be a semi-Thue system.
Given a *word* u over Σ, we say that a word v over Σ is *immediately derivable* from u if there is a defining relation
$x \longrightarrow y$ such that

$$u = rxs \text{ and } v = rys,$$

for some words r, s (which may be empty) over Σ.

If v is *immediately* derivable from u, we write

$$u \Rightarrow v.$$

Let P' be the set of all pairs $(u, v) \in \Sigma^{\bullet} \times \Sigma^*$ such that $u \Rightarrow v$.
Then $P \subseteq P'$, and

If $u \Rightarrow v$, then $wu \Rightarrow wv$ and $uw \Rightarrow vw$ for any word w."

> If $\mathbf{u} \Rightarrow \mathbf{v}$, then $\mathbf{uw} \Rightarrow \mathbf{vw}$ and $\mathbf{wu} \Rightarrow \mathbf{wv}$ for any word $\mathbf{w} \in \Sigma^*$.

Example
"Let $\mathfrak{S}$ be a semi-Thue system over the alphabet $\Sigma = \{a,b,c\}$, with the set of defining relations given by
$P = \{ab \longrightarrow bc, bc \longrightarrow cb\}$. Then words ac^3b , a^2c^2b and as bc^4 are all derivable from a^2bc^2 :

$a^2bc^2 \Rightarrow a(bc)c^2 \Rightarrow ac(bc)c \Rightarrow ac^2(cb) = ac^3b$,
$a^2bc^2 \Rightarrow a^2(cb)c \Rightarrow a^2c(cb) = a^2c^2b$, and
$a^2bc^2 \Rightarrow a(bc)c^2 \Rightarrow (bc)cc^2 = bc^4$." (PlanetMath)

"Under $\mathfrak{S}$, we see that if v is derivable from u , then they have the same *length*: $|u| = |v|$. Furthermore, if we denote $|a|_u$ the *number* of *occurrences* of letter a in a word u , then $|a|_v \leq |a|_u$, $|c|_v \leq |c|_u$, and $|b|_v = |b|_u$. Also, in *order* for a word u to have a non-trivial word v (non-trivial in the sense that $u \neq v$) derivable from it, u must have either ab or bc as a *subword*. Therefore, words like a^3 or $c^3b^4a^2$ have no non-trivial derived words from them." (PlanetMath)

1.2.2. Discussion of the presuppositions

Each repetition of the rules rule1= ab ⟶ bc and rule2 = bc ⟶ cb is realizing an *identification* of the result (operand)of the substitution with the initial word of the applied rule in the mode of identity.

1. a**ab**c2 ⇒ a(**bc**)c2 : by rule1 = ab ⟶ bc
2. a(**bc**)c2 ⇒ a**c(b**c)c : by rule2 = bc ⟶ cb

Rules: ab ⟶ bc/bc ⟶ cb

Rule1 recognizes in the mode of identity the left-most substring "ab" of the word and substitutes it at the place of its occurrence in that word with "bc".
The second step recognizes "bc", now as an antecedent for the rule2 and replaces it at the place of its occurrence with the succedent of the rule2 "cb".

This procedure is highly obvious. We are used to it. And such an explicit description I tried to give is quite superfluous, except we want to explain the procedure to a robot or to an alien. At least I need it to understand my own aversion against the frozenness of the whole paradigm of mathematical formalization.

But this game is presuming several "intuitions" which are not obvious at all.

It is not necessarily obvious that the whole procedure is supported by the principle of identity. A reuse of a word or a string, like "bc" in the antecedent of the rule1 and as a precedent in the rule2, has to match the matching conditions of iterability in the mode of identity. If the two occur-

rences of “bc” differ in the process of application, the substitution fails. Hence, the letters of the word are taken in a strictly atomistic and essentialistic sense to guarantee identity independently of their use and their role in the game. The use of the signs is not changing the signs.

Matching conditions
Matching conditions for the composition of the rules r1 and r2:

r1= ab ⟶ bc, r2 = bc ⟶ cb,
cod(r1) ≡ dom(r2) ⇒ r1 ◦ r2

As shown in earlier papers, a reflection of the matching conditions is enabling the possibility of an antidromic formalization by saltatories. Saltatories are complementary constructions to categories.

Morphogrammatic turn
For the application of the rules there is no need to know the internal structure of the words u , v and w.

With the morphogrammatic turn things are getting slightly more dynamic. A new kind of interplay between *identification* and *application* opens up first chances to avoid the frozenness of operative formalisms.

The presumption of identity in the substitution process gets some interesting deliberation and generalization.
If the substitution rule holds for “any word” under identity, a first attempt to liberalize its application happens with the understanding of a word (w) as a kenomic word [w].
Hence, for w = (ab), any keno-word of the form (ab) is applicable: [w] = {ab} = [bc] = ... = [&, #]. Both, (w) and [a] are of the same morphogrammatic structure and are of the same ‘length’.

A more radical generalization is achieved with the abstraction of *bisimilarity*: Two words (morphograms) $[w_1]$, $[w_2]$ are equal iff the have the same behavior. Hence, the length of $[w_1]$ and $[w_2]$ is not anymore defining sameness of morphograms.

Additional to identity and equality, some more kinds of thematizations enter the game of symbolic or ’mathematical’ writing: equivalence, simularity, bisimularity and different types of metamorphosis.

Definitions, theorems, methods, applications to recall the state of the art

approach to formal language theory, look at:
John M. Abela , ETS Learning of Kernel Languages, 2002
http://www.cs.unb.ca/~goldfarb/Theses/John's_Thesis.pdf

1.3. Kenomic semi-Thue systems

1.3.1. Semiotics

Σ^* denotes the set of all finite strings of symbols from Σ. This statement of semiotics becomes in morphogrammatics:

Sn2(Σ, $*$) denotes the universe of all finite tritograms of monomorphies from Sn2(Σ).

Sn2(Σ, $*$) is the trito-universe of keno-sequences “ks”. (Morphogrammatik, p. 77)

Sets in the trito-universe of keno-sequences are not sets in a definitorial sense, they might be called collections.
The objects (elements) of a collection are tritograms (ks-sequences) and are not defined by the set-theoretical rules of elements and ensemble (sets) which are based on identical concepts of first-order logic.

Σ = {a, b, c}
Σ^* = {a, b, c, aa, bb, cc, ab, ac, bc, aaa, bbb, ccc, ...}

StirlingSn2(Σ^*) = {a, aa, ab, aaa, aab, aba, abb, abc, aaaa, ...}

nfirstq(n, seq):

- nfirstq (3, TU) = {a, aa, ab, aaa, aab, aba, abb, abc}

Kenosequence: length (states) and technical signs (a, b, c)

kseq (3):
{[aaa], [aab], [aba], abb], [abc]}.

$$\begin{pmatrix} 1 \times 2 \times 3 \\ a\,a\,a \end{pmatrix}$$

Sn2(3, 3) = 1+2+1= 5

An element of the alphabet is a sequence of length 1: [a] = $\binom{1}{a}$.

[a] = $\binom{1}{a}$, [b] = $\binom{1}{b}$: [a] $=_{KG}$ [b]

Semiotically there are n unary elements in an alphabet:

(a) = $\binom{1}{a}$, (b) = $\binom{1}{b}$.

(a), (b) ∈ Alph: (a) $\neq_{SEM}$ (b) ⇒ $\begin{pmatrix} 1 \\ \swarrow\searrow \\ a \quad b \end{pmatrix}$

sum(Sn2(1,1)) = 1

Like semiotic sign sequences are defined by their *length* and their *alphabet*, kenogrammatic sequences *kseq* are defined by their positions and the realization of the positions, i.e. by their interaction of place and inscription.

Semiotic sequences are therefore defined by their *Cartesian* product $|sign|^{|length|}$ and keno-sequences are defined by their *Stirling* distribution(-partition) of places and kenograms: Sn2(place, kenos).

A morphogrammatic turn which is focused on monomorphies instead of kenograms is changing the presupposition of equal length for the equivalence (sameness) of morphograms, too. Two morphograms are the same iff their behavior is not distinguishable. That is, two morphograms are bisimilar if they have equal behavior. The abstraction of bisimilarity takes the fact into account that there are different fundamental morphogrammatic operations and therefore an abstraction over the operators instead of the morphograms as 'objects' is applied.

Hence, for semiotics the star or Kleene closure is

$$\Sigma^* = \dot{\bigcup_{i=0}} \Sigma_i = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3 \cup \ldots \cup \Sigma_n ,$$

the kenogrammatic universe TU instead is defined by

TU = ([1], Tsucc).

Sn2(Σ, n)

nfirstq(n, seq)

nfirstq(3, seq):
[aaa], [aab], [aba], [abb], [abc].

monomorphic TU = ([1], monomorphy(Tsucc)).

morph(nfirstq(3, seq)):
{[aaa], [aa][b], [a][b][a], [a][bb], [a][b][c]}.

Contextual repetition of monomorphies.

$A =_{MG} B$ iff EN(A) = EN(B)

Two words are *kenogrammatically* equivalent iff they have the identical EN-structure.

1.3.2. Tectonics

A calculus is defined by 2 alphabets and a set of rules (Paul Lorenzen, Haskell Curry):
1. the alphabet of signs
2. the alphabet of variables.

Two *semiotic* words are semiotically equal iff they are of the same length and all the occurrences in the words are equal (equiform) at the same places (positions) of the words (string).
Concatenation of words and star product of words, the empty word.

An introduction of a kenogrammatic calculus is applying the EN-abstraction on the objects of the calculus, i.e. on the words of the sign-alphabet and not yet on the meta-objects, i.e. the alphabet of the variables.

The rules of the calculus are applied to the signs of the alphabet with the help of variables which are not elements of the set of signs.

Second-order rules
Rules for morphogrammatics are not fully defined by the concept of *constant*, i.e. elements from a pre-given alphabet, and *variables* over the alphabet.

Additionally to the classic requisites of constant and variable, the *rules* have to be calculated, i.e. produced by rules on a different may be meta-

level. Hence, the notions of *constant*, *variable* and *rules* together are determining the rules of a morphogrammatic calculus. Prolongation, continuation, concatenation etc. are ruled by rules of rules, therefore the rules of morphogrammatic operations, like iteration and accretion, are second-order rules.

iteration: MG $\longrightarrow$ MGx, x$\in$ AG(MG)
accretion: MG $\longrightarrow$ MGx, x$\in$ AG(MG)+1

AG(MG) is the operation (rule) to calculate the constants of the continuation operation (rule) applied to the encountered morphogram.

In fact, there are no first-order constants, like elements from an alphabet, in the game. Morphogrammatic 'constants' are calculated, i.e. produced, hence variables or second-order constants. In this sense, they are not constants but variables determined by the preceding kenograms of the morphogram. The first-order constants are the elements of a semiotic alphabet which is involved technically by supporting the notational systems of morphogrammatics.

Nor are there any variable as stable containers of previous productions in the game of kenomic calculi and algorithms.
An application of a rule depends 1. on its definition and 2. on the structure of the previous productions represented by the variables. Therefore, as it is explained before, a concatenation is always depending on the 'conctenat' too. That is the crucial difference between atomistic and kenomic production rules.

Calculi
Stroke calculus (Lorenzen, 1950/60s)
Atom : {|}
Variable : {n}
Rules : {R_0, R_1}
R_0: $\longrightarrow$ |
R_1: n $\longrightarrow$ n|

(R_2: $R_1 \in$ iteration)

Production: |, ||, |||, ||||, ...

Kenomic calculus (Kaehr 1970s)

Atom : {[kg]} :(= kenomic constant)
Variable : {[mg]} :(= kenomic variable)
Rules : {R_0, R_1}
R_0: $\longrightarrow$ [kg]
R_1: [mg] $\longrightarrow$ [mg][kg] :(= $conc_{keno}$([mg], [kg]))

(R_2: $R_1 \in$ iteration, accretion)

Example
Atom : {[a]}
Variable : {[mg]}
Rules : {R_0, R_1}
R_0: $\longrightarrow$ [a]
$R_{1.1}$: mg = [a] $\longrightarrow$ $conc_{keno}$(mg = [a], kg = [a]) = {[aa], [ab]} $\in$ iteration, accretion
$R_{1.2}$: mg = [aa] $\longrightarrow$ $conc_{keno}$(mg = [aa], kg = [a]) = {[aaa, [aab]}
mg = [ab] $\longrightarrow$ $conc_{keno}$(mg = [ab], kg = [a]) = {[aba, [abb], [abc]}.

short:
Atom : {[a]}
Variable : {[mg]}
Rules : {R_0, R_1}
R_0 : $\longrightarrow$ [a]
R_1: [mg] $\longrightarrow$ [mg]^[a]:

$R_{1.1}$: [a] $\longrightarrow$ [a][a]) = {[a]^[a], [a]^[b]} = {[aa], [ab]}
$R_{1.2}$: [aa] $\longrightarrow$ [aa][a]) = {[aa]^[a], [aa]^[b]} = {[aaa], [aab]}
[ab] $\longrightarrow$ [ab][a]) = {[ab]^[a], [ab]^[b], [ab]^[c]}.

Production (out of [a]): [a], [aa], [ab], [aaa], [aab], [aba], [abb], [abc]...

This example, again, shows clearly the dependence of the alphabet from the applications of the rules and surely, the dependence of the rules from the generated alphabet. The classical definition is constructed *over* an alphabet Σ by a binary relation (x, y) $\in \Sigma^*$ x Σ^*, while the kenomic case is constructed *out* of a 'beginning' [a] generating 'words' by binary rules of the Stirling universe
(x, y) $\in$ (StirlingSn2(Σ, *) x StirlingSn2(Σ, *)) = (x, y) $\in K^\bullet$ x $K^\bullet$.

For notational reasons we have to add to the start alphabet of the Stirling universe of a calculus an alphabet, i.e. a *technical* sign repertoire, of technical letters, characters like brackets, dots etc.
Because the definition of binary relations depends on a Cartesian product

and the kenomic 'binary relation' on a Stirling distribution, kenomic relations are in fact technically not binary relations at all.

Quite obviously, Lorenzen calculi are alphabet-*stable*, they are defined *over* a pre-given alphabet. Therefore, their tectonics is *hierarchical*. Kenomic calculi are alphabet-*variable*. The alphabet is part of the production, and the production depends solely on the precedent productions and not on an abstract application of rules *over* an alphabet. Hence, their tectonics is not hierarchical but *heterarchical* and is defining a retro-grade recursivity. This little difference is in fundamental conflict with the main statement of computation (Gurevich):*"The vocabulary does not change during that evolution."*

Stable base set vs. self-modifying media
"The choice of the vocabulary is dictated by the chosen abstraction level. In a proper formalization, the vocabulary reflects only truly invariant features of the algorithm rather than details of a particular state. In particular, the vocabulary does not change during the computation. *One may think about a computation as an evolution of the initial state. The vocabulary does not change during that evolution.*
Is it reasonable to insist that the vocabulary does not change during that evolution?

"There are also so-called self-modifying *or "non-von-Neumann" algorithms which change their programs during the computation. For such an algorithm, the so-called program is just a part of the data. The real program changes that part of the data, and the* real program does not change.

"*While the base set can change from one initial state to another,* it does not change during the computation. *All states of a given run have the same base set. Is this plausible? There are, for example, graph algorithms which require new vertices to be added to the current graph. But where do the new vertices come from? We can formalize a piece of the outside world and stipulate that the initial state contains an infinite naked set, the reserve. The new vertices come from the reserve, and thus the base set does not change during the evolution. Who does the job of getting elements from the reserve? The* environment.

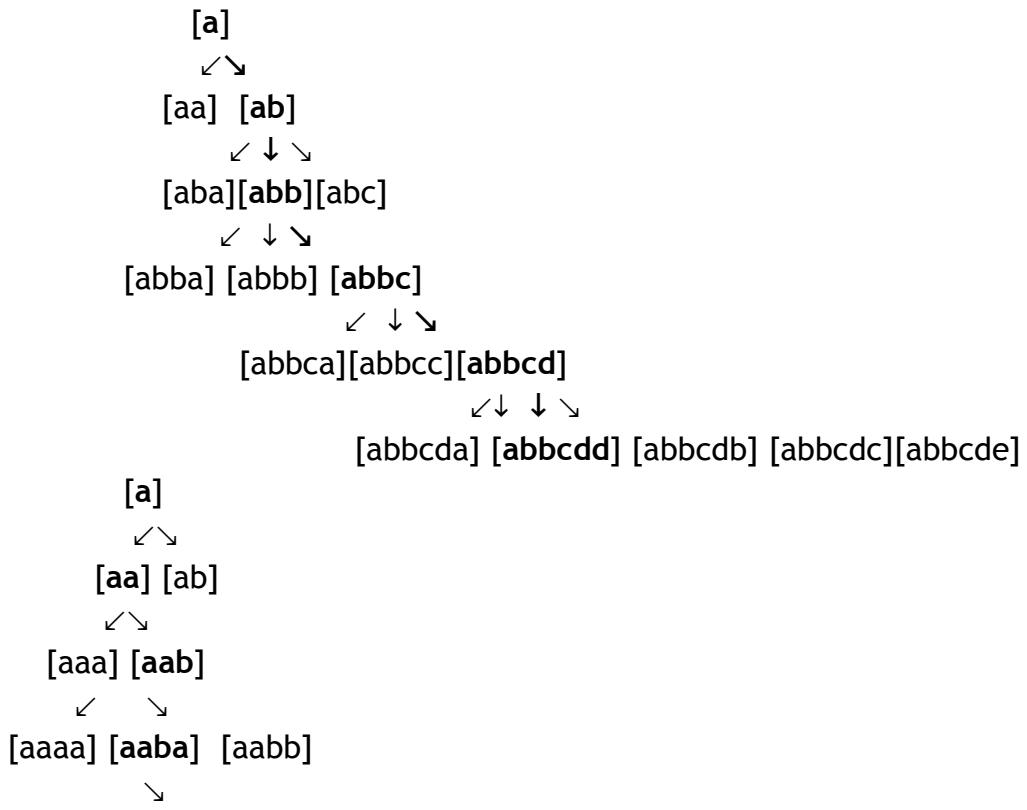
"*Formalizing this, we can use a special external function to fish out an*

element from the reserve. It is external *in the sense that it is controlled by the environment."* (Gurevich)
Yuri Gurevich, Sequential Abstract State Machines Capture Sequential Algorithms
ACM Transactions on Computational Logic, vol. 1, no. 1 (July 2000), pages 77–111.
http://research.microsoft.com/en-us/um/people/gurevich/Opera/141.pdf

Comparison between "Calculuses and Formal Systems" by Haskell B. Curry, Dialectica 47/48, pp. 249-271, 1958

A new challenge for polycontextural designs of formal languages, grammars, rewriting systems and calculi occurs with the chiastification of the object- and meta-system, i.e. the chiasm of objects (alphabets, signs, keno- and morphograms) and variables (schemes, frames,)

Same length morphograms
Encountered morphogram MG_2 = [abbcdd]. How can it be produced by which rules from "axiom" MG_1 =[aabaac]?
Both morphograms are correctly produced by the rules of the morphogrammatic system. Both are of the same length, therefore they cannot be equivalent. Hence, there is no derivation in the morphogrammatic calculus from $MG_1$to MG_2.



[aabaa]
↘
[aabaac].

Both produced morphograms are of the same 'length', hence there is no evolving production rule which is generating a 'word' of the same 'length' with a different pattern and the same path.
Therefore, the rules of *differentiation*, called emanation, shall be introduced to transform morphograms of the same complexity into each other of the same length.

1.3.3. Semi-Thue systems with morphograms

A keno-string rewriting system or **keno-semi-Thue system** is a tuple (Σ, R) where tnf(Σ) is an alphabet, usually assumed finite. The elements of the set Sn2(Σ *) (* is the Kleene star here, Sn2(Σ *) is the Stirling distribution) are finite (possibly empty) keno-strings on tnf(Σ), sometimes called keno-sequences or morphograms in formal writing systems; we will simply call them keno-strings here.

Two keno-strings A and B are **equivalent** iff EN(A) = EN(B),

A behavioral or actional approach is contemplating on the behavior of kenograms and not on the semio-ontological question of what *is* a kenogram.
Therefore a mix of different definitions of sign-use is possible: EN, TNF, SEMiotic, MONomorphy, etc.

Example
Semiotic alphabet: Σ_{SEM} = {a, b}
Kenomic words over the semiotic alphabet Σ_{SEM} of length 3:
Sn2(Σ*, 3) = {a, aa, ab, aaa, aab, aba, abb}

Sn2(Σ*)≡ K*
K* is the trito-universe TU.

Monomorphies of K*
Monomorphies in kenomic systems are a kind of an analogy to a "*sub-string*" in a word or string production system.

For Σ_{SEM} = {a, b}, length(K*) = 3:
Monomorphies of K*(3) are μ(K*, 3) = {[a], [aa], [aaa],[aa][b], [a][b][a], [a][bb]}.

Hence, $\mu(K^*)$ = {[a], [aa], [aaa]}.

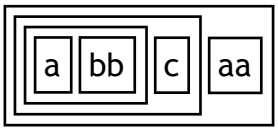
1.3.4. Monomorphies

Monomorphic notation

Monomorphies in morphograms are playing a similar role as atomic signs in sign sequences.

The monomorphies of the morphogram MG = [abbcaa] are writen in a table with the distinctions *locus*, *monomorphy* and *kenogram* as follows.

Monomorphies are produced by the monomorphic decomposition Dec of the morphogram MG: Dec(MG) = (mg_1, mg_2, mg_3, mg_1) wit the kenograms {a, b, c} for mg_1= [a], mg_2 = [bb], mg_3 = [c].



=

MG	loc_1	loc_2	loc_3	loc_4
Dec	mg_1	mg_2	mg_3	mg_1
Ken	*a*	*b*	*c*	*a*
	ø	*b*	ø	*a*

$MG^{1.2\ .3\ .1}$	loc_1	loc_2	loc_3	loc_4
$MG^{1.0\ .3\ .0}$	mg_1	–	mg_3	–
$MG^{0.2\ .0\ .0}$	–	mg_2	–	–
$MG^{0.0\ .0\ .1}$	–	–	–	mg_1

$$\text{morphogram} = \left|\text{kenoms}\right|_{\text{locus}}^{[\text{mg}]}$$

$$\left[\text{abbcaa}\right] = \left(1_1^1, 2_2^2, 1_3^3, 2_4^1\right)$$

Systematics of morphograms

Positionality of morphograms : $<$ Position, Locality, Place $>$.

Position of the morphogram in *a*

morphogrammatic system defined by emanation and evolution.

Locality of the monomorphies in *a* morphogram;

loci are offering place for different monomorphies.

Monomorphies might be reduced to homogeneous

patterns or they might keep some structuration.

Place of *a* kenom in *a* monomorphy depending on the length of the monomorphy.

Position $MG^{(m,n)}$

$MG^{(m)}$	locus
$Dec\left(MG^{(m)}\right)$	monomorphy
$Ken\left(MG^{(m)}\right)$	kenom

morphogrammatics
morphogram
monomorphy
[[locus [[place [[kenom]]]]]]

Decomposition of morphograms into monomorphies (Gunther)
Given a morphogram [aabc] how to decompose it into its monomorphies?

1. **[a]**
2. **[aa]** **[ab]**
3. [aaa] **[aab]** [aba] [abb] [abc]
4. [aaaa] [aaab] .... **[aabc]** ...[abcd]

Dec([aabc]) = ([aab]; [aa], [ab]; [a])

[aabc] ⟶ [aab]|[a] ⟶ [aa]|[ab] ⟶ [a].
Production of [aabc]: [a] $\longrightarrow_{iter}$ [aa] $\longrightarrow_{accr}$ [aab] $\longrightarrow_{accr}$ [aabc].

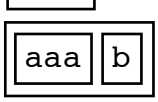
In contrary to the semiotic case, composition and decomposition of morphograms are not symmetric.

The decomposition Dec([aabc]) is ”over-complete”, according to Gunther’s classification of complexity into *incomplete* (I), *complete* (C)and *over-complete*(O) because it decomposes into two monomorphies of the same length, [aa] and [ab]. Nevertheless, the morphogram [ab] decomposes finally into the monomorphy [a]. Hence, the remaining basic monomorphies of [aabc] are [a] and [aa]. The monomorphy [aa] is not decomposable into smaller monomorphies, say [a], because it is a morphogram without differentiation which would be necessary for a decomposition of the morphogram.

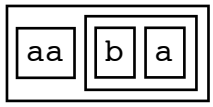
Decomposition for $MG^{(4)}$

Dec([aaaa]) = [aaaa]

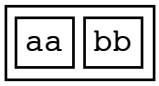
Dec([aaab]) = [aaa], [a]



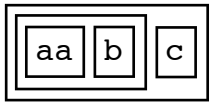
Dec([aaba]) = [aa] |[ab], [a]



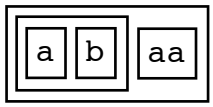
Dec([aabb]) = [aa]



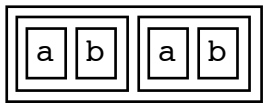
Dec([aabc]) = [aab], [aa]|[ab], [a]



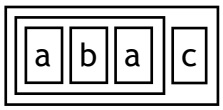
Dec([abaa]) = [ab] |[aa], [a]



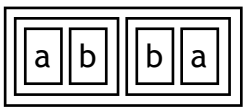
Dec([abab]) = [aba], [ab], [a]



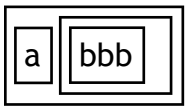
Dec([abac]) = [aba], [ab], [a]



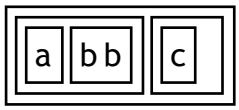
Dec([abba]) = [abb], [aa], [ab], [a]



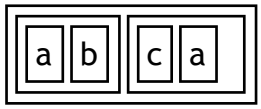
Dec([abbb]) = [a], [aaa]



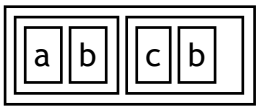
Dec([abbc]) = [a], [aa], [ab], [a]



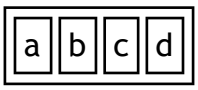
Dec([abca]) = [ab, [aa]|[ab], [a]



Dec([abcb]) = [aab, [aa]|[ab], [a]



Dec([abcd]) = [abc], [ab], [a].



Decomposition of [abbcaa]

Dec([abbcaa]) = ([abbc], [aab]|[abb], [aa]|[ab], [a]).

StirlingSn2(m, n), m = n = 1 to 6

1. [**a**] : 1
2. [**aa**] [**ab**] : 2
3. [aaa] [**aab**] [aba] [**abb**] [abc] : 5
4. [aaaa] [aaab] ....[**abbc**] ... [abcd] :15
5. [aaaaa] [aaaab] .....[abcde] : 52
6. [aaaaaa] ... [**a bb c aa**] ... [abcdef] : 203

1.3.5. Morphogrammatic rewriting rules

$P \in Sn2(K^*, K^*)$
$(u, v) \in Sn2(K^*, K^*)$ such that $u \Rightarrow v$.
Then $P \subseteq P'$, and
If $u \Rightarrow v$, then $uw \Rightarrow vw$ and $wu \Rightarrow wv$ for any monomorphy w.

$u \Rightarrow v$ iff there exists a context C and $(u, v) \in P$ such that $u = C(x)$ and $v = C(y)$.

"Definition 1.34 A **context** is a pair $C = \langle y, z \rangle$ of strings.
The **substitution** of x into C, in symbols C(x), is defined to be the string y^x^z.
We say that x occurs in v in the context C if $v = C(x)$.
Every occurrence of x in a string v is uniquely defined by its context.
We call C a **substring** occurrence of x in v." (Kracht)

Contexts and contextures
$C_{id}(x)$, $C_{equi}(x)$, $C_{sim}(x)$, $C_{bisim}(x)$, $C_{morph}(x)$.

Example
$(aa) \Rightarrow (aaa)$, then $(aa)w_1 \Rightarrow (aaa)w_2$, $w_3(aa) \Rightarrow w_4(aaa)$; $C_{id}(w_i, w_j)$ and $w_i = w_j$, i, j = 1,2,3,4

1.3.6. The Fibonacci word exercise

> *"Fibonacci words are easily defined by iterating a morphism. In fact, the Fibonacci morphism is among the absolute simplest (more precisely shortest) conceivable morphism: discard the one letter alphabet, and try to define a non trivial short morphism on two letters. It suffices, for this, that the image of one letter has length two, and you already get Fibonacci's morphism."* (Jean Berstel, Fibonacci Words - A survey, in: G. Rozenberg, A. Salomaa, The Book of L, 1985, pp. 13 - 27)

Production of Fibonacci words
A = {a, b}
e: $A^* \longrightarrow A^*$, A^* is the Kleene product of signs, *e* is a morphism from A^* to A^*.
e(a) = ab
e(b) = a

Iteration of this morphism defines the *Fibonacci words*.
f_0= a, f_1= ab

$f_{n+2} = f_{n+1} f_n$

$f_0 = a$
$f_1 = ab$
$f_2 = aba$ 1.0
$f_3 = abaab$ 2.1
$f_4 = abaababa$ 3.2
$f_4 = abaababaabaaba$ 4.3

Kenogrammatic analogon to Fibonacci words

$A = \{a, b\}$
$\{a, b\} \subseteq A: (a) \neq_{SEM} (b)$
$e_{KG}: K^* \longrightarrow K^*$, K^* is the Stirling distribution of kenogram sequences.
$\{a, b\} \subseteq K^* : [a] =_{KG} [b]$
$e_{KG}([a]) =_{KG} [ab]$
$e_{KG}([b]) =_{KG} [a]$

$f_0 = [a], f_1 = [ab]$
$f_{n+2} = f_{n+1} (f_n)_{KENO}$

Kenogrammatic composition with iteration and accretion.

$$f_{n+2} = f_{n+1} \begin{pmatrix} f_n^{iter} \\ \amalg \\ f_n^{acc} \end{pmatrix}$$

$A = \{a, b\}$ is the alphabet of the kenomic Fibonacci example. Also atomic " signs " are kenomically equal, i.e. $(a) =_{KG} (b)$, the signs of the alphabet are used here as *technical* signs in a kenomic standard notation form (tnf). Therefore, at least two different games have to be distinguished in the definition of the kenomic Fibonacci word system :
1. The ' semiotics' of the head of the formal language, i.e. the sign repertoire (alphabet) and
2. the definition of the behavior of the standard signs in the calculus, i.e. as kenograms of kenomically defined operations (recursion) f_n.

Fibonacci derivations $FIB^{(m)}_{KG}(a, ab)$:
$f_0 = a$
$f_1 = ab$
$f_2 = aba; abb; abc$ 1.0

f_3 =
aba'ab, aba'ba, aba'ac, aba'bc, aba'ca, aba'cb, aba'cd; 2.1
abb'ab, abb'ba, abb'ac, abb'bc, abb'ca, abb'cb, abb'cd;
abc'ab, abc'ba, abc'ac, abc'bc, abc'ca, abc'cb, abc'cd.

1, 1, 3, 21, 85, ...

$f_2 = f_1(f_0) = \text{LI}$(aba; abb; abc), i.e. the mediated parallelism $\begin{pmatrix} \text{aba} \\ \text{II} \\ \text{abb} \\ \text{II} \\ \text{abc} \end{pmatrix}$.

Fibonacci derivations $\text{FIB}^{(2)}_{KG}$(a, ab):
f_0 = a
f_1 = ab
f_2 = aba; abb 1.0
f_3 = aba'ab, aba'ba, 2.1
abb'ab, abb'ba,

1, 1, 2, 4, ...

$f_2 = f_1(f_0) = \text{LI}$(aba; abb), i.e. the mediated parallelism $\begin{pmatrix} \text{aba} \\ \text{II} \\ \text{abb} \end{pmatrix}$.

1.3.7. Inversion and equivalence

In a general morphogrammatic word algebra the equality (equivalence, similarity, bisimilarity) of words has to be defined formally. A simple operator, the *inversion* of the order of a word, called *reflector*(refl) offers some distinctions between words.

Inverse words produced by the Fibonacci word system might be compared.

For f_1 = (ab) we get refl(f_1) = (ba). The word (ba) is not a word of the semiotic Fibonacci word system.
A kenomic consideration shows that both words (ab) and (ba) are equivalent: (ab) $=_{KG}$ refl(ab).

This holds generally for all symmetrical kenomic words:

$H_{1,2} \in$ SYM: $H_1 =_{KG} H_2$.

This nice property of kenomic word systems is helping to reduce the work into half, like *duality* in category theory is offering "*two for one*" (Herrlich).

2. Finite State Automata

2.1. Classical FSA

2.1.1. Automaton and language

Finite State Machine

"We consider non-deterministic finite state machines with no accepting states, defined as follows.
A *finite state machine* (FSM) is a quadruple $M = (\Sigma, Q, q_0, \delta)$, where Σ is the alphabet of input symbols, Q is the set of states, q_0 is the *initial* state, and δ is the *transition* function, which maps $Q \times \Sigma$ to subsets of Q. If every $\delta(q, a)$ contains exactly one state, then M is *deterministic*.
In this case we may write $\delta(q, a) = q'$ instead of $\delta(q, a) = \{q'\}$."
http://www.cs.yale.edu/homes/aspnes/papers/lata2011-proceedings.pdf

Binary relation

A binary relation, denoted by $\rightarrow$, is any subset of the Cartesian product $P \times P$.
For any binary relation $\rightarrow \subset P \times P$:
$domain(\rightarrow) =_{def} \{ a \mid \exists b, (a, b) \in \rightarrow \}$, and
$range(\rightarrow) =_{def} \{ b \mid \exists a, (a, b) \in \rightarrow \}$.

Automaton

$M = (Q, \Sigma, \delta, q_0, F)$

Q : States

Σ : Alphabet

δ : state – transition function

q_0 : initial state

$F \subseteq Q$: set of final states

FSM transition function

δ: $\mathfrak{Q} \times \Sigma \longrightarrow \mathfrak{Q}$,
rewriting rules: $q_i a_k \longrightarrow q_j$, with q_i, q_j are states, $a_k$is input symbol

The *transition* function $\delta : Q \times \Sigma \rightarrow Q$ of a DFA can be extended to $Q \times \Sigma^*$ as follows:
$\delta(q, \varepsilon) = q$
$\delta(q, wa) = \delta(\delta(q, w), a)$.

Language
$L = \{ \omega \in \Sigma^* \mid q_0\omega \Rightarrow^* q_f\varepsilon$, with $q_f \in F\}$

2.2. Kenomic FSA

2.2.1. Explanations and motivations

This exercise is focusing on the transition rule (function). The consequences for the concepts of the *alphabet*, the states and the initial state will be reflected later and will be conceived then as the pre-conditions of the new understanding of the transition function and the concept of the kenogrammatic finite state machines (kenoFSM) as such.

Elementary *cellular automata* are collections of simple finite state machines.
In a similar sense, morphogrammatic cellular automata are interacting collections of elementary kenomic 'finite state automata'. Each term, 'finite', 'state', 'automata', deserves a proper deconstruction.

In earlier approaches, the strategic order was inverse. The focus was on the intriguing situation of the 'non'-alphabet character of kenogrammatics and its paradoxical consequences. The new approach plays with the fact of the *Stirling* character of the kenogram sequences and morphograms and with a standard representation of the 'non'-representable alphabet and kenogrammatic sequences, i.e. the trito-normal form (tnf).

In other words, only the kind of usage of marks defines their role as semiotic, kenogrammatic or morphogrammatic in the graphematic game. Hence, marks in an alphabet are playing in the context of the alphabet their semiotic role. In the use of a kenomic context, the mark of the alphabet are playing the roles of kenograms. Then as a collection of kenograms, all elements of an alphabet are kenomically the same.

One of the most elicit analysis of an abstract theory of computation is given by Gurevich's *Abstract State Machines* (ASM). This way of thinking was reflected in my "Skizze-0.9.5" from 2003. Like with Konrad Zuse, computation is defined by Gurevitch as a step-wise *transition* in *time*, guided by *rules*, from an *initial* to a *terminal* object, the result of the computation.
Obviously, the limits of this paradigm are clear: no *interactivity*. Computation is conceived as problem-solving and not as a media of interacting

processes, without beginning nor end.

2.2.2. Keno-Languages and -Automata

A deconstruction of FSM and rewriting systems has to start wit a deconstruction of the underlying basics concepts. One important basic concept is the *binary relation*.

A first deconstructive step would have to *contextualize* the concept of relationality of the concept of binary relation which is based on relational logic and the Wiener-Kuratovsky definition of an ordered pair of elements. A second step has to deconstruct the concept of *binarity*, P × P, of the binary relation.

P × P $\mapsto$ P x P; Q : contextualization (context-logical decomposition)
P × P $\mapsto$ StirlingSn2(P, 2): From sets to distributions.

Binary relation
A kenomic binary relation, denoted by →, is any subset of the Stirling distribution StirlingSn(P, 2) . For any kenomic binary relation → ⊂ StirlingSn(P, 2) :
domain(→) $=_{def}$ { a | ∃b, (a, b) ∈ →}, and
range(→) $=_{def}$ { b | ∃a, (a, b) ∈ →}.

A n-ary kenomic relation, denoted $\rightarrow^n$, is any subset of the Stirling distribution StirlingSn(P, n).

kenoFSM-Automaton
$M = ([Q], \Sigma, \delta_{keno}, [q_0], [F])$
$[Q]$: States
Σ : Alphabet, technical
δ_{KENO} : keno – state – transition function
$[q_0]$: kenomic initial state
$[F] \subseteq [Q]$: set of final kenomic states

kenoFSM transition function
δ_{keno} : Sum(Stirling Sn2(Q, Σ)) $\xrightarrow[EQ]{}$ $[Q]$

Retro-grade recursion
$\delta_{keno}([q,] \varepsilon) = [q]$
$\delta_{keno}([q], [w]\hat{}[a]) = \delta_{keno}(\delta_{keno}([q], [w]), [a])$.

rewriting rules: $[q_i][a_k] \xrightarrow[EQ]{} [q_j]$, with $[q_i]$, $[q_j]$ as states, $[a_k]$ is input

symbol.

Language

L = $\{\omega \in K^* \mid [q_0]\,[\omega] \underset{EQ}{\Longrightarrow}^* [q_f]\varepsilon$, with $[q_f] \in F\}$, K^* = Sum(StirlingSn2(Σ, *))

2.2.3. Formal aspects of kenomic cellular automata

Alphabet, language and classical CA

"An alphabet Σ is a finite nonempty set of symbols. Σ^* denotes the set of all finite strings of symbols from Σ. The empty string is denoted λ. A language is any subset of Σ^*. Σ^k denotes those elements of Σ^* of length k. The symbols in a string *s* of length *n* are indexed from 1 to n and *s[i]* denotes the i^{th} symbol of *s*.

"Kari [6] notes that cellular automata have several fundamental properties of the *physical* world: they are massively *parallel*, *homogeneous*, and *reversible*, have only *local* interactions, and facilitate formulation of conservation laws based on local update rules. We consider one-dimensional asynchronous reversible cellular automata with *insertions* and *deletions* because they support universal computation.

"A *cellular automaton* C = (Σ, δ) is composed of an *alphabet* of symbols Σ and a set δ *transition* rules of the form axb $\leftrightarrow$ ayb for *substitutions* or ab $\leftrightarrow$ axb for *insertions* and *deletions*, where a, b, x, y $\in \Sigma$.
The idea is that the value of a given cell of the automaton may change only when both its neighbors have specific values.

"For s_1, $s_2 \in \Sigma^*$, s_1 can reach s_2 in one step of C , denoted $s_1 \rightarrow_C s_2$, if applying one transition rule to s_1 yields s_2. And s_1 can reach s_2 in C if $s_1 \rightarrow^*_C s_2$. Given an input string s $\in \Sigma^*$, a snapshot of C on input s is any string s' such that s can reach s' in C."

2.2.4. Operations on kenoCAs

Union of machines
Examples

A. Classical CAs
1. Q = states
$Q = \{(r_1, r_2) \mid r_1 \in Q_1$ and $r_2 \in Q_2\}$

The set is the *Cartesian* product of sets $Q_1$and Q_2 and is written Q_1 x Q_2.
union (R9, R4) = |head(R9)| x |head(R4)| = 3 x 3 = 9

2. Alphabet: $\Sigma_1 = \Sigma_2 = \{■, □\}$

3. rules

$(r_1, r_2) \in Q, a \in \Sigma$:

$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$

4. initial

$q_0 = (q_1, q_2)$

B. Kenogrammatic CAs

kenogrammatics of union

1. States

$Q = \{(r_1; r_2) | r_1 \in Q_1; r_2 \in Q_2\}$

The set is the *Stirling* union of sets Q_1 and Q_2 and is written StirlingSn2(Q_1, Q_2).

StirlingSn2(Q_1) x StirlingSn2(Q_2) ≠ StirlingSn2(Q_1 x Q_2)

add(R9, R4) = StirlingSn2(add(head(R9), head(R4))) = 4

2. Alphabet:

$\Sigma = \Sigma_1 = \Sigma_2$:

add(Σ_1, Σ_2) > Σ

$add_{iter}(\Sigma_1, \Sigma_2) = \Sigma$

$add_{accr}(\Sigma_1, \Sigma_2)$ = add(Σ_1, Tsucc(Σ_2)) > Σ

add(Σ_1={■, □}, Σ_2 = {■, □}) =

Σ_1={■, □},

Σ_2 = Tsucc({■, □}) = {■, □, ■}.

3. rules

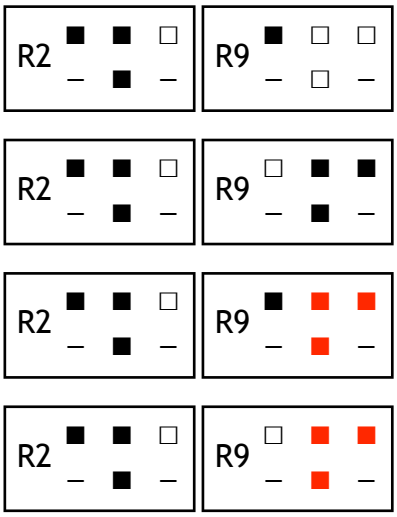
$(r_1, r_2) \in Q, a \in \Sigma$:

$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(\text{Tsucc}(r_2, a)))$

r_1= [■ ■ □],

r_2 = [■ □ □]

add(R2, R9) =



4. initial

$q_0 = (q_1, q_2) \Rightarrow q_0 = \text{add}(q_1, q_2)$

Example

$q_0 = \text{add}(q_1, q_2) = (q_1, q_2, q_3, q_4)$

$q_0 = (q_1 = \{[\blacksquare \quad \square]\}, q_2 = \{[\square, \blacksquare]\}, q_3 = \{[\blacksquare \quad \blacksquare]\}, q_4 = \{[\square, \blacksquare]\}$

Concatenation of kenomic languages

$A =_{SEM} \{a, b\}$, $B =_{SEM} \{b, c\}\}$

$AB =_{SEM} \{ab, ac, bb, bc\}$

$A =_{KENO} \{1, 2\}$, $B =_{KENO} \{1, 2\}$

kconcat ([1, 2], [1, 2]):

$[AB] =_{KENO} \{[1212],[1221],[1213],[1231],[1223],[1232],[1234]\}$.

2.2.5. Symmetric cellular automata and reduction

"Exploiting the symmetry with respect to renaming of q states of cellular automata allows us to reduce the number of rules to consider. Namely, it suffices to consider only orbits (equivalence classes) of the rules under q! permutations forming the group S_q*."* Vladimir V. Kornyak, Cellular Automata with Symmetric Local Rules, 2006

"Definition 3.1. A cellular automaton A = (S, N, δ) is said to be *symmetric* if

$\delta(s_1, s_2, \ldots, s_{|N|}) = \delta(s_{\sigma(1)}, s_{\sigma(2)}, \ldots, s_{\sigma(|N|)})$,

for every $s_1, s_2, \ldots, s_{|N|} \in S$ and $\sigma \in S_{|N|}$ (the permutation group of $|N|$ degree)."

http://www.mtns2004.be/database/papersubmission/upload/341.pdf

Permutations are not in conflict with the concept of identity of their elements. Identity is of the elements is a precondition for their permutation.

Permutational equivalence is not kenogrammatic sameness. On a kenogrammatic level, permutational equivalence leads from the trito- to the deutero-level of structuration. Therefore, the permutational reduction of CAs is different from a kenomic reduction of CAs.

3. Conditions for concatenation and substitution

3.1. Types of compositions

MG |	*H*	*m* |	*H*	*m* |	*H*	*m* |	*H*	*m* |	*H*	*m*
$=_{MG}$	+	+	+	+	+	+	−	−	+−	+−
$=_{sem}$	+	+	+	−	−	−	−	−	−	−
⌢	+	+	+	+	+	+	−	−	□	□
type	id	□	eq	□	sim	□	bisim	□	metamorph	□
CA	CCA	□	kenoCA	□	morphCA	□	bisimCA	□	metamCA	□

3.1.1. Equaity: Concatenation

$\mathbf{u \Rightarrow_{id} v}$**:**
$u \Rightarrow_{MG} v$, $w_1 u \Rightarrow_{SEM} w_2 v$ and $uw_3 \Rightarrow_{SEM} vw_4$ and
$w_1 =_{sem} w_2 =_{sem} w_3 =_{sem} w_4$. **: [++++]**

(Trivially equal : $\mathbf{u \Rightarrow_{id} v}$**:** $u \Rightarrow_{SEM} v$ and $w_1 =_{sem} w_2 =_{sem} w_3 =_{sem} w_4$.)

$\mathbf{u \Rightarrow_{id} v}$**:** $u \Rightarrow_{MG} v$, $u \Rightarrow_{SEM} v$ and $w_{1,2,3,4} \in C_{ID}$.

$$\frac{u \Rightarrow_{id} v}{wu \Rightarrow_{SEM} wv \qquad uw \Rightarrow_{SEM} vw \qquad w_{1,2,3,4} \in C_{ID}}$$

Equality
$w \in C_{ID}$

$$\frac{u \underset{ID}{\Rightarrow} v}{w_1 u \underset{ID}{\Rightarrow} w_2 v, \; uw_3 \underset{ID}{\Rightarrow} vw_4}$$

Table

$$\begin{bmatrix} \textbf{Id} & wx & xw & w_{12} & w_{34} \\ MG & + & + & + & + \\ SEM & + & + & + & + \end{bmatrix} = \begin{bmatrix} \textbf{Id} & wx & xw & w_{12} & w_{34} \\ SEM & + & + & + & + \end{bmatrix}$$

Example

$u = v = (aab),$

$w_1 =_{sem} w_2 =_{sem} w_3 =_{sem} w_4 = (cc)$

$$\frac{(aab) \underset{ID}{\Rightarrow} (aab)}{(aab)(cc) \underset{ID}{\Rightarrow} (aab)(cc),\quad (cc)(aab) \underset{ID}{\Rightarrow} (cc)(aab)}$$

Lambda –Example I

$$(\lambda w.ww)((\lambda v.vv)u) \qquad : t_0 \quad (\text{subst} : (\lambda\, v\, t)\, s) \to t[v/s]_{ID})$$

$$\swarrow \qquad \searrow$$

$$(\lambda w.ww)(uu) \quad (\lambda v.vv)\, u\, ((\lambda v.vv)u) \qquad : t_1, t_2,\ u =_{SEM} u$$

$$| \qquad\qquad |$$

$$| \qquad u\, u\, ((\lambda v.vv)u)$$

$$\searrow \qquad \swarrow$$

$$uu(uu) \qquad : t_1 =_{ID} t_2.$$

Cellular automata scheme

CA transition rule

$$\mathbf{CA} = \left[\mathrm{CA},\ \phi\right]$$

$$\left[c_{i}\text{-}1\left(t\right),\ c_{i}\left(t\right),\ c_{i+1}\left(t\right)\right]$$

$$\downarrow \phi = \text{transition}$$

$$c_{i}\left(t+1\right)$$

Null

3.1.2. Equivalence: Juxtaposition

$u \Rightarrow_{eq} v$:

$w_1u \Rightarrow_{MG} w_2v$, $w_3u \Rightarrow_{SEM} w_4v$,
$uw_1 \Rightarrow_{MG} vw_2$, $uw_3 \Rightarrow_{SEM} vw_4$,

$w_i \in C_{EQ}$, $i = 1,\ldots,4$:

$w_1 \neq_{sem} w_2$, $w_1 =_{MG} w_2$,
$w_3 \neq_{sem} w_4$, $w_3 =_{MG} w_4$,
$w_1 =_{sem} w_3$, $w_1 =_{MG} w_3$,
$w_2 =_{sem} w_4$, $w_2 =_{MG} w_4$.

Equivalence

$w \in C_{EQ}$

$$u \underset{EQ}{\Longrightarrow} v$$

$$\overline{w_1 u \underset{EQ}{\Longrightarrow} w_2 v,\ uw_3 \underset{EQ}{\Longrightarrow} vw_4}$$

Table

Equ	wx	xw	w_{12}	w_{34}	w_{13}	w_{24}
MG	+	+	+	+	+	+
SEM	−	−	−	−	+	+

Example

$u = \left[\mathrm{aab}\right]$, $v = \left[\mathrm{bba}\right]$

$w_{1,2} = \left[cc\right],\ w_{3,4} = \left[dd\right]:$

$$\frac{\left[aab\right] \underset{EQ}{\Rightarrow} \left[bba\right]}{\left[aab\right]\left[cc\right] \underset{EG}{\Rightarrow} \left[bba\right]\left[dd\right] \quad \left[cc\right]\left[aab\right] \underset{EQ}{\Rightarrow} \left[dd\right]\left[bba\right]}$$

Lambda –Example –II

$$\text{Terms} = \left\{u,\ v,\ w,\ x,\ y,\ z\right\}$$

$$\text{subst} : \left(\lambda\, v\, t\right) s \right) \rightarrow t\left[v/s\right]_{keno}$$

$$\left(\lambda w.ww\right)\left(\left(\lambda v.vv\right)u\right) \qquad : t_0$$

$$\swarrow \qquad \searrow$$

$$\left(\lambda v.vv\right)x\left(\left(\lambda v.vv\right)x\right)\left(\lambda w.ww\right)\left(uu\right) \ : t_1, t_2,\ u =_{KENO} x$$

$$\downarrow \qquad\qquad \vert$$

$$xx\left(\left(\lambda v.vv\right)x\right) \qquad \vert$$

$$\searrow \qquad \downarrow$$

$$xx\left(xx\right) =_{MG} uu\left(uu\right) \qquad : t_1 \neq_{SEM} t_2,\ t_1 =_{KENO} t_2$$

Lambda –Example –III

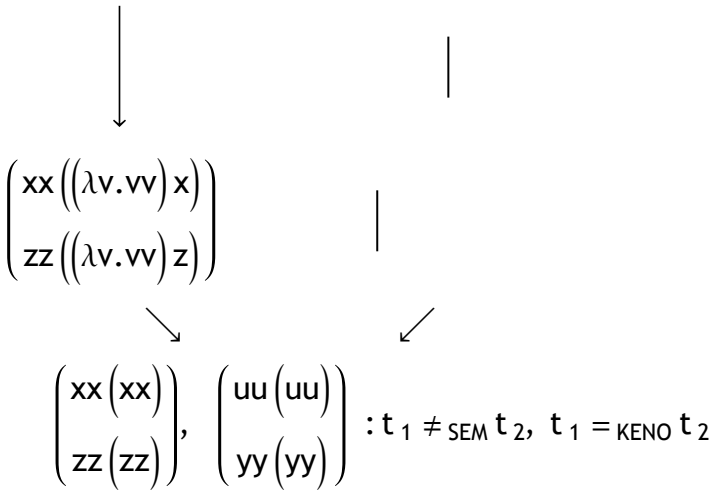
$$\text{Terms} = \left\{u,\ v,\ w,\ x,\ y,\ z\right\}$$

$$\text{subst} : \left(\lambda\, v\, t\right) s \right) \rightarrow t\left[v/s\right]_{keno}$$

$$\left(\lambda w.ww\right)\left(\left(\lambda v.vv\right)u\right) \qquad : t_0$$

$$\swarrow \qquad \searrow$$

$$\left(\begin{matrix}\left(\lambda v.vv\right)x\left(\left(\lambda v.vv\right)x\right)\\ \left(\lambda v.vv\right)z\left(\left(\lambda v.vv\right)z\right)\end{matrix}\right), \left(\begin{matrix}\left(\lambda w.ww\right)\left(uu\right)\\ \left(\lambda w.ww\right)\left(yy\right)\end{matrix}\right)$$



kenoCA transition scheme

$$\textbf{kenoCA} = \left[\mathrm{CA},\ \mu,\ \phi\right]$$

$$\mathrm{Const}\left(t\right):\ \left[c_{i}\text{-}1\left(t\right),\ c_{i}\left(t\right),\ c_{i+1}\left(t\right)\right]$$

$$\downarrow \mu = \mathrm{ENtoKS}\left(\mathrm{ENS}\left(c_{i-1}\, c_{i}\, c_{i+1}\right)\right)$$

$$\in \left\{\mathrm{Head}\left(\mathrm{Rule}\right)\right\}$$

$$\downarrow \phi\left(\mu\right) = \mathrm{rule}$$

$$\mathrm{Const}\left(t+1\right):\quad c_{i}\left(t+1\right):\mathrm{Rule}-\mathrm{Result}$$

3.1.3. Similarity: Cooperation

$$\frac{\left(u \Rightarrow_{\mathrm{SIM}} v\right) \Rightarrow \begin{pmatrix} w_{1}\, u \Rightarrow_{\mathrm{SIM}} w_{2}\, v \\ u w_{3} \Rightarrow_{\mathrm{SIM}} v w_{4} \end{pmatrix}}{}$$

$$wu\neg \Rightarrow_{\mathrm{SEM}} wv,\quad uw\neg \Rightarrow_{\mathrm{SEM}} vw$$

$$w \in C_{\mathrm{SIM}}:$$

$$w_{1} \neq_{\mathrm{sem}} w_{2},\ w_{1} =_{\mathrm{MG}} w_{2}$$

$$w_{3} \neq_{\mathrm{sem}} w_{4},\ w_{3} =_{\mathrm{MG}} w_{4}$$

$$w_{1} \neq_{\mathrm{sem}} w_{3},\ w_{1} =_{\mathrm{MG}} w_{3}$$

$$w_{2} \neq_{\mathrm{sem}} w_{4},\ w_{2} =_{\mathrm{MG}} w_{4}$$

Similarity

$w \in C_{SIM}$

$$\frac{u \underset{SIM}{\Longrightarrow} v}{w_1 u \underset{SIM}{\Rightarrow} w_2 v,\ u w_3 \underset{SIM}{\Rightarrow} v w_4}$$

Table $u \underset{SIM}{\Rightarrow} v$

$u \underset{SIM}{\Rightarrow} v$	wx	xw	w_{12}	w_{34}	w_{13}	w_{24}
MG	+	+	+	+	+	+
SEM	−	−	−	−	−	−
$\frown$	+	+	.	.	.	.

Example: $u \underset{SIM}{\Rightarrow} v$

u = [aab], v = [bba]

$u \Rightarrow_{MG} v$, $u \neg\Rightarrow_{SEM} v$

w_1 = [cc], w_2 = [dd] : $w_1 \neq_{sem} w_2$, $w_1 =_{MG} w_2$

w_3 = [ee], w_4 = [ff] : $w_3 \neq_{sem} w_4$, $w_3 =_{MG} w_4$

$w_i \in$SIM, i=1,2.3.4

length(w_1) = length(w_2)

$w_1 \neq_{sem} w_2$

sem(w_i) $\frown$ sem(u) = ø, i = 1,2

length(w_3) = length(w_4)

$w_3 \neq_{sem} w_4$

sem(w_i) $\frown$ sem(v) = ø, i = 3,4

length(w_1) = length(w_3)

$w_1 \neq_{sem} w_3$

sem(w_i) $\frown$ sem(v) = ø, i = 1, 3

length(w_2) = length(w_4)

$w_2 \neq_{sem} w_4$

sem(w_i) $\frown$ sem(v) = ø, i = 2, 4

$$\frac{\left[\text{aab}\right] \underset{\text{SIM}}{\Rightarrow} \left[\text{bba}\right]}{\left[\text{aab}\right]\left[\text{cc}\right] \underset{\text{SIM}}{\Rightarrow} \left[\text{bba}\right]\left[\text{dd}\right],\quad \left[\text{ee}\right]\left[\text{aab}\right] \underset{\text{SIM}}{\Rightarrow} \left[\text{ff}\right]\left[\text{bba}\right]}$$

Example I – SIM

$$\text{Terms} = \left\{u,\ v,\ w,\ x,\ y,\ z\right\}$$

$$\text{subst} : \left(\lambda\, v\, t\right) s \to t\left[v/s\right]_{\text{SIM}}$$

$$\left(\lambda w.ww\right)\left(\left(\lambda v.vv\right)u\right) \qquad : t_0$$

$$\swarrow \qquad \searrow$$

$$\left(\lambda v.vv\right)x\left(\left(\lambda v.vv\right)x\right) \quad \left(\lambda w.ww\right)\left(uu\right) : \left(\text{br}_{u-x}\right), \left(\text{a1}\right); \left(\text{br}_{u-u}\right) : t_1, t_2$$

$$\downarrow \qquad \vert$$

$$xx\left(\left(\lambda v.vv\right)x\right) \qquad \vert \qquad : \left(\text{br}_{x-x}\right), \left(\text{a2}\right) \ : t_1$$

$$\searrow \qquad \downarrow$$

$$xx\left(yy\right) =_{\text{SIM}} uu\left(uu\right) \qquad : \left(\text{br}_{x-y}\right);\ \left(\text{a1}\right), \left(\text{br}_{u-u}\right)$$

$$: t_1 \neq_{\text{SEM}} t_2,\ t_1 \neq_{\text{keno}} t_2,$$

$$: t_1 =_{\text{SIM}} t_2,\ .$$

3.1.4. Bisimilarity: Fusion

Bisimilarity

$w_1 \underset{\text{BIS}}{\Rightarrow} w_2$ iff

$\exists\ w_1,\ w_2 : \left(w_1\, w_2\right) \in \left(\text{Dec, Vk, Vs, EVk, EVs}\right)$

$\text{length}\left(w_1\right) \neq \text{length}\left(w_2\right)$:

$\exists\ \left(x_1, x_2\right) : \text{EVk}\left(\left[w_1\right]\right) = \left(x_1, x_2\right)$

$\exists\ \left(y_1, y_2\right) : \text{EVs}\left(\left[w_2\right]\right) = \left(y_1, y_2\right)$

$$\text{IF}\begin{pmatrix}\text{Vs}\left(x_1, x_2\right) = \left[w_2\right] \\ \text{Vk}\left(y_1, y_2\right) = \left[w_1\right]\end{pmatrix}\text{THEN}\ \left(w_1 \underset{\text{BIS}}{\Rightarrow} w_2\right).$$

Bisimilarity

$$Vs\left(EVk\left(w_1\right)\right) = Vk\left(EVs\left(w_2\right)\right)$$

Table: $u \underset{BIS}{\Longrightarrow} v$

$u \underset{BIS}{\Longrightarrow} v$	wx	xw	w_{12}	w_{34}	w_{13}	w_{24}
MG	−	−	−	−	+	+
SEM	−	−	−	−	−	−
$\frown$	+	+	.	.	.	.
length	−	−	−	−	+	+

Conditions : $u \underset{BIS}{\Rightarrow} v$

$wu \neg \Rightarrow_{SEM} wv$, $uw \neg \Rightarrow_{SEM} vw$

$w \in C_{BIS}$:

$w_1 \neq_{sem} w_2$, $w_1 \neq_{MG} w_2$

$w_3 \neq_{sem} w_4$, $w_3 \neq_{MG} w_4$

$w_1 \neq_{sem} w_3$, $w_1 =_{MG} w_3$

$w_2 \neq_{sem} w_4$, $w_2 =_{MG} w_4$

Bisimilarity

$w \in C_{BIS}$:

$$\frac{u \underset{BIS}{\Longrightarrow} v}{w_1 u \underset{BIS}{\Longrightarrow} w_2 v,\ uw_3 \underset{BIS}{\Longrightarrow} vw_4}$$

Example

$u \underset{BIS}{\Rightarrow} v,\ w_1,\ w_2$

For

$\text{length}\left(w_1\right) \neq \text{length}\left(w_2\right)$

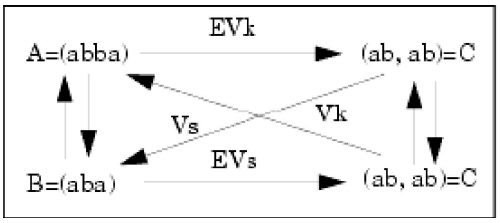
and

$\text{EVk}\left(\left[w_1\right]\right) = \left(\left[ab\right], \left[ab\right]\right)$

$\text{EVs}\left(\left[w_2\right]\right) = \left(\left[ab\right], \left[ab\right]\right)$

and

$$\frac{\begin{pmatrix} \text{Vs}\left(\left[ab\right], \left[ab\right]\right) = \left[w_2\right] \\ \text{Vk}\left(\left[ab\right], \left[ab\right]\right) = \left[w_1\right] \end{pmatrix} \quad \textbf{then} \quad \text{Vs}\left(\text{EVk}\left(w_1\right)\right) = \text{Vk}\left(\text{EVs}\left(w_2\right)\right)}{\left(w_1\, u \underset{BIS}{\Rightarrow} w_2\, v\right)}$$



Example – BIS

$\text{Terms} = \left\{u,\ v,\ w,\ x,\ y,\ z\right\}$

$\text{subst} : \left(\lambda\, v\, t\right) s \rightarrow t\left[v/s\right]_{BIS}$

$$\left(\lambda w.ww\right)\left(\left(\lambda v.vv\right)u\right) \qquad : t_0$$

$$\swarrow \qquad \searrow$$

$$\left(\lambda v.vvv\right) y\left(\left(\lambda v.vvv\right)x\right) \quad \left(\lambda w.ww\right)\left(uu\right) \quad : \left(br, xx =_{BIS} vvv\right), \left(a1\right); \left(br_{v-u}\right)$$

$$\downarrow \qquad\qquad \mid$$

$$yyy\left(\left(\lambda v.vvv\right)x\right) \qquad \mid \qquad : \left(br\right), \left(a2\right) \ : t_1$$

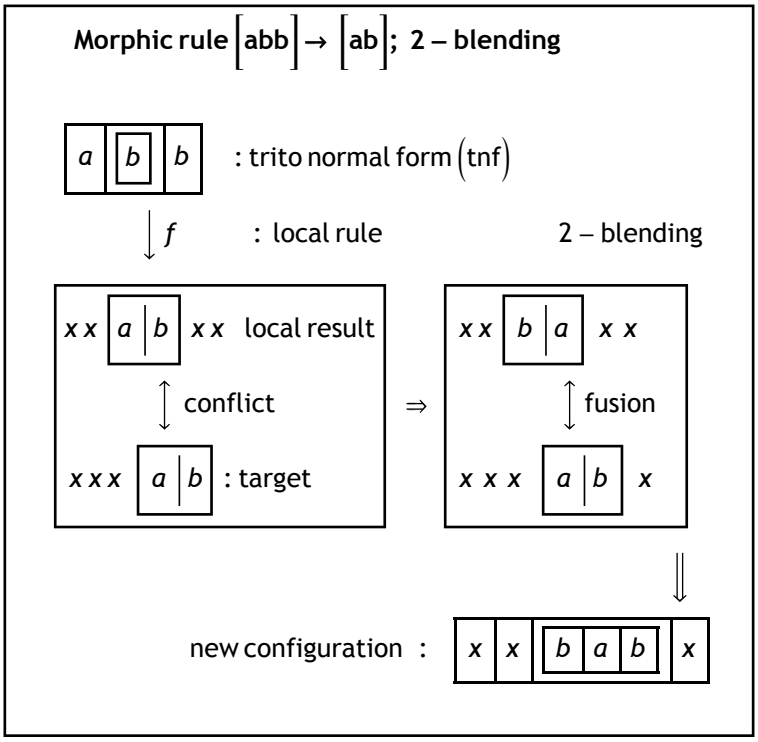
$$\searrow \qquad \downarrow$$

$$yyy\left(xxx\right) =_{BIS} uu\left(uu\right) \qquad : \left(br_{v-x}\right);\ \left(a1\right), \left(br_{w-u}\right)$$

$$: t_1 =_{SIM} t_2, \ .$$

http://memristors.memristics.com/Church-Rosser%20Morphogrammatics/Church-Rosser%20in%20Morphogrammatics.pdf

Morphic transition rule scheme



Transition scheme for morphCA

$$\textbf{morphCA} = \left[CA,\ \mu,\ \varphi,\ \phi\right],$$

$$c_i(t):\quad \left[c_i\text{-}1(t),\ c_i(t),\ c_{i+1}(t)\right]$$

$$\downarrow \phi = \varphi(\mu)$$

$$c_i(t+1):\qquad \left[c_i\ c'_i\right](t+1)$$

3.1.5. Metamorphosis of rewriting

Recall, "Elements of P are variously called *defining relations*, **productions**, or *rewrite rules*, and $\mathfrak{S}$ itself is also known as a *rewriting system*. If (x, y)$\in$ P, we call x the *antecedent*, and y the *consequent*.
Instead of writing (x, y)$\in$ P or xPy , we usually write

$$x \longrightarrow y."$$

Up to now, the transformation rules of rewriting systems had been defined in a still quite straight forward sense of succession of *antecedent* and *conse-*

quent by substitution.

Funny candidates joined this game which was opened up by Thoralf Skolem with his identity conserving productions. Kenogrammatic rewriting systems introduced an abstraction on the 'data' transforming sign systems to kenogrammatic systems. With the idea of overlapping and fusion a mechanism to deal with overdetermined rewriting systems had been opened up as morphic systems.

Nevertheless, the brave succession and hierarchical order of *antecendents* and *precedents* had been untouched and accepted by this change of the modi of interaction, and was leading the introduction of the semiotic, kenomic and morphic concepts of rewriting systems.

A much more intriguing situation is possible with the idea of *metamorphic* transformations.

A full fledged involvement of the concept of diamond categorical *interchangeability* of distributed functors allows to introduce the paradox of a simultaneity of *sameness* and *differentness* in the game of interchanging roles.
Therefore, morphisms are not just changing objects in the mode of *equality*, *equivalence*, *similarity* and *bisimilarity* but in the mode of *metamorphosis* too. Metamorphosis is understood in the strict sense of an interplay of change and pertinence.
Hence, an antecedent is not just producing its precendent by a transition rule but is at once also keeping itself in the game of change as the antecendet of a precendent.

3.1.6. Types of change

1. transition $_{\text{succession}}$:

$$\text{Antecedent} \longrightarrow \text{Precedent},\ \text{succession} - \text{modi} = \left\{\text{id},\ \text{eq},\ \text{sim},\ \text{bisim}\right\}$$

2. transition $_{\text{chiasm}}$:

$$\begin{pmatrix} \text{Antecedent}_1 \longrightarrow \text{Precedent}_1 \\ \updownarrow \quad \text{X} \quad \updownarrow \\ \text{Precedent}_2 \longleftarrow \text{Antecedent}_2 \end{pmatrix}$$

The wording here is

Antecedents becomes Precedents and Precedents becomes Antecedents.

3. transition $_{\text{polycontextural}}$ **:**

$$\begin{pmatrix} \text{Antecedent}_{1.3} \longrightarrow \text{Precedent}_1 \\ \downarrow\uparrow \quad \text{X} \quad \updownarrow \\ \text{Precedent}_{2.3} \longleftarrow \text{Antecedent}_2 \end{pmatrix}$$

The wording here is, *"Antecedents becomes Precedents and Precedents becomes Antecedents. The result is reflected in system3".*

4. transition $_{\text{diamond}}$ **:**

$$\begin{pmatrix} \text{Antecedent}_{1.3} \longrightarrow \text{Precedent}_1 \\ \downarrow\uparrow \quad \text{X} \quad \updownarrow \\ \text{Precedent}_{2.3} \longleftarrow \text{Antecedent}_2 \end{pmatrix} \Big| \left(\text{system}_4\right)$$

$$\text{system}_4 : \text{Precedent}_4 \longleftarrow \text{Antecedent}_4$$

"The matching conditions of the chaistic and polycontextural construction are reflected in the 'antidromic' system4."

5.transition$_{\text{metamorphosis}}$**:**

$$\left(\text{system}_{1.1}\right) \amalg \left(\text{system}_{2.2}\right)$$

$$\left[\begin{matrix} \left(\text{system}_{1.1 \diamond 2.1}\right) \amalg \left(\text{system}_{2.2 \diamond 1.2}\right) \\ \left(\text{system}_{1.2}\right) \amalg \left(\text{system}_{2.1}\right) \end{matrix}\right]$$

$$\left(\text{system}_{1.1 \diamond 2.1}\right) \amalg \left(\text{system}_{2.2 \diamond 1.2}\right)$$

$$\left(\text{metaphor}: \textit{Gregor Samsa}\left(\textit{Franz Kafka}\right)\right.$$
$$\left.\textit{as Gregor in the process of metamorphosis}\right)$$

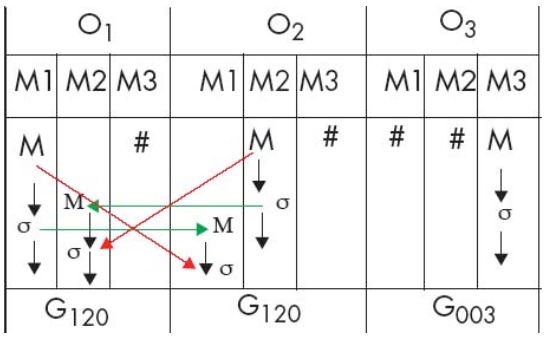
Null

6. transition $_{\text{diamond–metamorphosis}}$**:**

$$\left(\text{system}_{1.1}\right) \amalg \left(\text{system}_{2.2}\right)$$
$$\left[\begin{array}{c}\left(\text{system}_{1.1 \diamond 2.1}\right) \amalg \left(\text{system}_{2.2 \diamond 1.2}\right) \\ \left(\text{system}_{1.2}\right) \amalg \left(\text{system}_{2.1}\right) \\ \left(\text{system}_{1.1 \diamond 2.1}\right) \amalg \left(\text{system}_{2.2 \diamond 1.2}\right)\end{array}\right] \Big| \left(\begin{array}{l}\text{system}_{4:(1.1-2.2)} \\ \text{system}_{4:(1.1-2.1;\,2.2-1.2)} \\ \text{system}_{4:(1.2-2.1)} \\ \text{system}_{4:(1.1-2.1;\,22-12)}\end{array}\right)$$

$$\left(\text{metaphor}: \textit{Gregor Samsa in metamorphosis,}\right.$$
$$\left.\textit{reflecting the metamorphosis of his environment}\right)$$

3.1.7. Type/term model of metamorphosis



"The wording here is not only *"types becomes terms and terms becomes types"* but *"a type as a term becomes a term"* and, at the same time, *"a type as type remains a type"*. Thus, *"a type as a term becomes a term and as a type it remains a type"*. And the same round for terms.

Full wording for a chiasm between terms and types over two loci

Explicitly, first the green round,

" A type $\sigma_{1.1}$ as a term $M_{2.1}$ becomes a term $M_{2.1}$
and as a type $\sigma_{1.1}$ it remains a type $\sigma_{1.1}$ for a term $M_{1.1}$ ".

And,

" A type $\sigma_{2.2}$ as a term $M_{1.2}$ becomes a term $M_{1.2}$
and as a type $\sigma_{2.2}$ it remains a type $\sigma_{2.2}$ for a term $M_{2.2}$ ".

And simultaneously, mediated,
the second round in red, the same for *terms*:

" A term $M_{1.1}$ as a type $\sigma_{2.1}$ becomes a type $\sigma_{2.1}$
and as a term $M_{1.1}$ it remains a term $M_{1.1}$ for a type $_{1.1}$ ".
And,
" A term $M_{2.2}$ as a type $\sigma_{1.2}$ becomes a type $\sigma_{1.2}$
and as a term $M_{2.2}$ it remains a term $M_{2.2}$ for a type $_{2.2}$ ".

And finally, between terms $M_{1.1}$ and $M_{2.2}$ and types $\sigma_{1.1}$ and $\sigma_{2.2}$,
a categorial *coincidence* is realized.
While between terms and types a *morphism* (order relation) exists.

http://memristors.memristics.com/Polyverses/Polyverses.html
http://memristors.memristics.com/Dominos/Domino%20Approach%20to%20Morphogrammatics.html
http://www.thinkartlab.com/pkl/lola/From%20Ruby%20to%20Rudy.pdf

3.1.8. Interchangeability of metamorphosis

Metamorphic interactivity

$$\left[\left(M, \sigma\right),\ \approx,\ \diamond,\ \circ,\ \amalg\right]$$

$$\begin{pmatrix} \left(\left(\sigma_1 \approx M'_2\right) & \circ & \left(M'_1 \approx \sigma_2\right)\right) \\ \diamond & \amalg & \diamond \\ \left(\left(M_1 \approx \sigma'_2\right) & \circ & \left(\sigma'_1 \approx M_2\right)\right) \end{pmatrix} :$$

$$\begin{bmatrix} \left(M_2 \approx M'_2\right) \\ \amalg \quad \diamond \\ \left(M_1 \approx M'_1\right) \end{bmatrix} \circ \begin{bmatrix} \left(\sigma_2 \approx \sigma'_2\right) \\ \amalg \quad \diamond \\ \left(\sigma_1 \approx \sigma'_1\right) \end{bmatrix} =$$

$$\begin{bmatrix} \left(M_2 \circ \sigma_2\right) \\ \amalg \\ \left(M_1 \circ \sigma_1\right) \end{bmatrix} \approx \begin{bmatrix} \left(M'_2 \circ \sigma'_2\right) \\ \diamond \\ \left(M'_1 \circ \sigma'_1\right) \end{bmatrix}$$

$\circ$: composition, $\amalg$: mediation
$\diamond$: interchange, $\approx$: similarity

3.1.9. Some summary

Interdependence of operators $\left(\circ,\ \amalg,\ \diamond,\ \approx\right)$: Metamorphism

$$\begin{pmatrix} \left(M_1 \circ \sigma_1\right) \amalg \left(M_2 \circ \sigma_2\right) \\ \left(\sigma'_2 \diamond M'_1\right) \\ \left(\sigma'_1 \diamond M'_2\right) \end{pmatrix} \Longleftrightarrow \begin{pmatrix} M_1 \approx \sigma'_2 \\ \sigma'_1 \approx M_2 \\ \sigma_1 \approx M'_2 \\ M'_1 \approx \sigma_2 \end{pmatrix}$$

Interdependence of operators $\left(\circ,\ \bigotimes,\ \equiv\right)$: Equality

$$\left[\begin{pmatrix}(M_1 \circ \sigma_1)\\ \otimes \\ (M_2 \circ \sigma_2)\end{pmatrix} = \begin{pmatrix}M_1\\ \otimes \\ M_2\end{pmatrix} \circ \begin{pmatrix}\sigma_1\\ \otimes \\ \sigma_2\end{pmatrix}\right] \Longleftrightarrow \begin{pmatrix}M_1 \equiv M_1\\ \sigma_1 \equiv \sigma_1\\ M_2 \equiv M_2\\ \sigma_2 \equiv \sigma_2\end{pmatrix}$$

Interdependence of the operators $(\circ, \amalg, \simeq)$: **Similarity**

$$\left[\begin{pmatrix}(M_1 \circ \sigma_1)\\ \amalg \\ (M_2 \circ \sigma_2)\end{pmatrix} = \begin{pmatrix}M_1\\ \amalg \\ M_2\end{pmatrix} \circ \begin{pmatrix}\sigma_1\\ \amalg \\ \sigma_2\end{pmatrix}\right] \Longleftrightarrow \begin{pmatrix}M_1 \simeq M_2\\ \sigma_1 \simeq \sigma_2\end{pmatrix}$$

3.2. Transitive closures

3.2.1. Linearity

"Next, take the **reflexive transitive closure** P" of P' . Write a ⇒ b for (ab)∈ P". So a ⇒* b means that either a = b , or there is a finite chain a = a_1, ..., a_n = b such that a_i ⇒ a_{i+1} for i =1, ..., n-1. When a ⇒* b, we say that b is *derivable* from a."

"Concatenation preserves derivability:
a ⇒ * b and c ⇒ * d imply ac ⇒ * bd ." (PlanetMath)

Morphogramatics of concatenation

a ⇒ * b and c ⇒ *d imply a^c ⇒ * b^d .

Depending on the definition of the concatenation operation "^", different realizations have to be distinguished.

Candidates are:

Identity (equality)
Equivalence
Similarity
Bisimilarity
Metamorphosis.

Identity
a $\Rightarrow_{id}$ * b and c $\Rightarrow_{id}$ *d imply a^c $\Rightarrow_{id}$ * b^d

Equivalence

$$\left(a \Rightarrow_{eq} * b\right) \text{and } \left(c \Rightarrow_{eq} * d\right) \text{imply} \begin{pmatrix} a \hat{}^{i} c \Rightarrow_{eq} * b \hat{}^{i} d \\ \amalg \\ a \hat{}^{j} c \Rightarrow_{eq} * b \hat{}^{j} d \end{pmatrix}$$

$$\begin{pmatrix} \left(a \Rightarrow_{eq} * b\right) \\ \amalg \\ \left(c \Rightarrow_{eq} * d\right) \end{pmatrix} \Rightarrow \left(\begin{pmatrix} (a) \\ \amalg \\ (c) \end{pmatrix} \Rightarrow_{eq} \begin{pmatrix} (b) \\ \amalg \\ (d) \end{pmatrix} \right)$$

3.2.2. Bifunctoriality

For ambiguous semi-Thue systems, like the morphogrammatic semi-Thue systems, the interplay of bifunctorial interchangeability gets some relevance in the definition of the rewriting system as such.

Up to isomorphism and down to kenomic sameness
Two kenomic semi-Thue systems are equal iff they are equivalent, i.e. isomorphic.

3.3. Category theory and rewriting systems

3.3.1. Graph transformation

"We have introduced a new notion of abstract rewriting system based on categories. These systems are designed for dealing with abstract rewriting frameworks where rewrite steps are defined by means of matches. We have defined the properties of (horizontal) composition as well as functoriality of rewriting in our abstract setting and we have illustrated these properties throughout several algebraic graph rewriting systems." (F. Prost et al)
http://arxiv.org/pdf/1101.3417v3

Also *horizontal* and *vertical* aspects of categorical compositions are considered, the main point still is to develop a well glued approach in the sense of the Berlin school of Graph transformation (Ehrig, König). The ultimate glue is offered by the category-theoretic *span* concept. Pushouts and spans are *"used for describing graph transformation systems as categorical rewriting systems"*.

*"In a categorical rewriting system, the matches introduce a "*vertical *dimension", in addition to the "*horizontal *dimension" provided by the rules. This composition gives rise to the bicategory of categorical rewriting systems (as for spans, we get a bicategory rather than a category, because*

the unicity of pushouts is only up to isomorphism)." (ibd)
http://content.imamu.edu.sa/Scholars/it/net/petritalk.pdf

Bifunctoriality
In contrast, a more or less glue-free construction is introduced by the concept of categorical *bifunctoriality* and its generalization to a diamond category-theoretic interchangeability of morphisms and contextures. In this approach "*horizontal*" and "*vertical*" structures of graph transformation systems are not *glued* together but are interacting in the framework of *interchangeability*.

3.3.2. Pushouts and diamonds

"After having developed some insights and experiences with the diamond approach and its complementary structures, a design of diamond category theory might be introduced which is not as close to the introductory analogy to classic category theory."

Excerpts from: Kaehr, Category of Glue III, (2009), unpublished.
http://www.thinkartlab.com/pkl/lola/Category%20Glue%20II/Category%20Glue%20II.html

Hetero-morphisms are reflecting the matching conditions of the *composition* of morphisms in a category.
There is an analogy between the *concatenation* of production rules in rewriting systems and the composition of morphisms. Graph transformation systems and graph grammars are surpassing the limitations of *linear* concatenation of sign sequences. Graph transformation is formalized by Schneider, Ehrig et al. by categorical *pushouts*.

"Therefore, graph transformations become attractive as a modeling and programming paradigm for complex-structured software and graphical interfaces. In particular, graph rewriting is promising as a comprehensive framework in which the transformation of all these very different structures can be modeled and studied in a uniform way." (Ehrig, Padberg, p. 3)

Hetero-morphisms of pushouts are reflecting the complexity of graph composition.
Because of its complexity a more complex *interplay* between hetero-morphisms and graph composition is opened up.
Focused on graph derivations, the saltatorical hetero-morphisms are complementarily defined. But the inverse complementary situation holds too. During a graph derivation, the saltatorical system might be changed and

therefore re-defining the structural conditions of the categorical graph derivation.

This kind of mutual interplay had been defined for categories and saltatories concerning the matching conditions of the composition of morphisms. Therefore, the interplay in diamondized graph systems is a generalization of the compositional approach.

3.3.3. Concatenation and pushouts

Production systems are based on concatenation. They have an initial and a terminal object.
A generalization of concatenation production systems is introduced by a transition from strings to graphs. Strings consists of atomic signs. Graphs are composed by elementary graphs, consisting of nodes and edges. Hence, graph grammars are a generalization of sign production systems. Sign production systems are mapped as trees, graph transformations as graphs.

Graph transformation and graph grammars based on pushout constructions are well embedded in category theory. Pushouts and their dual pullbacks are save categorical constructions based on the composition rules for morphisms in categories.
Categories in general are well complemented by saltatories.

Because pushouts are defined in categories, a diamondization of pushouts follows naturally.
Hence, pushouts as models for graph grammars gets diamondized pushouts for diamond graph grammars.

As a consequence of the dependence of graph grammars from category theory, it seems obvious that graph transformations are not surpassing the limitations of computability of sign production systems.
Graph transformation sequences are computational equivalent to sign production sequences.
This correspondence between the computability of sign production systems and graph derivation might be disturbed by diamondized graph grammars.