

— vordenker-archive —

Rudolf Kaehr

(1942-2016)

Title

Finite State Machines and Morphogrammatics

Machines on Differences: A Contribution to Saussure-Derrida Machines

Archive-Number / Categories

3_12 / K09

Publication Date

2012

Keywords

TOPICS: MorphoFSA examples, Some applications of MorphoFSMs onto morphogrammatics, Palindromes and MorphoFSMs, Palindromes and Chiasms, Simulation of FSA by MorphoFSA, Formal approximations for MorphoFSA, Classical machines with input and output.

Disciplines

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

Abstract

This paper is sketching a further approach toward a better understanding of morphogrammatics: Morphic Finite State Machines, more exactly, Morphic Difference Machines. It seems that the difference-theoretical aspect of morphogrammatics gets an even more direct thematization and formalization in the context of an analogon to FSMs. Some preliminary combinatorial elaborations are added with the application of SML-procedures. The concept of asymmetric palindromes is sketched. "[Derrida's Machines](#)" is the title of a continuing research program that went public 2004.

Citation Information / How to cite

Rudolf Kaehr: "Finite State Machines and Morphogrammatics", www.vordenker.de (Sommer Edition, 2017) J. Paul (Ed.), http://www.vordenker.de/rk/rk_Finite-State-Machines-and-Morphogrammatics_2012.pdf

Categories of the RK-Archive

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

Finite State Machines and Morphogrammatics

Machines on Differences: A Contribution to Saussure-Derrida Machines

Author: Rudolf Kaehr

Table of Contents

(compiled by EvGo, April 2017) [*]

(Number of pages refer to the display of the pdf-reader)

Abstract

1. Some types of Automata	003
1.1. FSA, IFA, CA and morphic Automata	004
1.1.1. Motivation: the ubiquity of automata	
1.1.2. Finite state machines	
1.1.3. Morphic automata	
1.2. Finite State Machines	009
1.2.1. Recalling definitions	
1.2.2. Motivations for morphic FSA	
2. Morphogrammatic FSA	012
2.1. Some descriptions	012
2.2. The epsilon/nu-structure of morphogram	014
2.2.1. Differentiation and enumeration	
2.2.2. The systematic status of morphograms	
2.2.3. General scheme of a subversion strategy	
2.2.4. Number of symbolic representations	
2.2.5. FSA equivalence, isomorphism and minimization	
3. Morphogrammatic FSAs	021
3.1. First definitions of morphogrammatic FSA	021
3.1.1. MorphoFSA examples	
3.1.2. Further examples of MorphoFSA	
3.1.3. Machine tables for MorphoFSA	
3.1.4. Comparisons, abstractions and more machine tables	
3.1.5. Some applications of MorphoFSMs onto morphogrammatics	
3.1.6. Towards a little Zoo of MorphoFSM diagrams	
3.2. Palindromes and MorphoFSMs	042
3.2.1. Palindromes and iteration	
3.2.2. Palindromes, first	
3.2.3. Palindromes, again	
3.2.4. Palindromes and Chiasms	
3.2.5. Appendix about asymmetric palindromes	
3.2.6. Chiasm AB:C:BA	
3.3. Simulation of FSA by MorphoFSA	060
3.3.1. Relations	
3.3.2. Comparisons: MorphoFSA and FSA	
3.3.3. Why are MorphoFSMs not just relational systems?	
3.4. Pumping lemma, first	063
3.4.1. The classical scenario	

* URL: http://www.vordenker.de/rk/rk_Finite-State-Machines-and-Morphogrammatics_2012.pdf

3.5. Formal approximations for MorphoFSA	066
3.5.1. Automata-theoretical approach	
3.5.2. Programming approach	
3.6. Observations on morphic automata	068
3.6.1. Iteration and retrogradeness	
3.6.2. Operations on MorphoFSM	
3.6.3. Dissemination of MorphoFSMs	
3.6.4. Mono- and polysemy	
3.6.5. Determinism and non-determinism	
3.6.6. Logic, Categories, FSM and MorphoFSM	
3.6.7. Diamond characterization	
3.7. Further comparisons	086
3.7.1. Quotient automata of FSA and MorphoFSA	
3.7.2. General comparison of automata	
3.7.3. Cellular automata based on differences	
3.8. Classical machines with input and output	090
3.8.1. Mealy Machine	
3.8.2. Moore Machine	
3.8.3. Turing Machines	
3.8.4. Examples	
3.8.5. Representations and combinations	
3.9. Presentations of automata: transition tables and de Bruijn graphs	095
3.9.1. Transition tables	
3.9.2. de Bruijn graphs for FSM	
4. Critical questions	096
4.1. Are morphic FSAs Finite State Automata at all?	096
4.2. Is there any use for morphic automata?	097
4.2.1. Semiotics of palindromes and anagrammatics	
Notes	102

How to cite:

Rudolf Kaehr, "Finite State Machines and Morphogramatics", www.vordenker.de (Sommer Edition, 2007) J. Paul (Ed.)

URL: http://www.vordenker.de/rk/rk_From-Universe-to-Polyverses_2010.pdf

Finite State Machines and Morphogrammatics

Machines on Differences: A Contribution to Saussure-Derrida Machines

Rudolf Kaehr Dr.phil[@]

Copyright ThinkArt Lab ISSN 2041-4358

Abstract

Morphograms as a new mode of inscription had been introduced into the academic world by Gotthard Gunther (1900 - 1984) with his theory of “transjunctional operations” for a cybernetic logic of self-reflection at the BCL at the 1st of April 1962. His concept of a transformation system of mediated morphograms by a reflector operator has been studied in my dissertation “*Materialien 1973-75*”, published 1976, based on a project at the FeOLL GmbH, 1973-75, (Prof. H. Stachowiak, G. Thomas, J. Seehusen), and brought to a formal and programmed elaboration by different collaborators around 1988 guided by Wolfgang Niegel and students, Munich, and then finally formalized, programmed and published 1993 as the book “*Morphogrammatik*” as the report No. 1 of the research project “*Theorie komplexer biologischer Systeme*” (Volkswagen-Stiftung) and also published at the IFF Klagenfurt by Thomas Mahler/Rudolf Kaehr, and made accessible later on the Website “Polycontextural Logic” at Techno.net, later ThinkArt Lab Glasgow. Meanwhile new approaches emerged, especially with the understanding of morphograms not just as pre-logical patterns but also as rules (operators), realized by the concept of morphogrammatic cellular automata.

This paper is sketching a further approach toward a better understanding of morphogrammatics: Morphic Finite State Machines, more exactly, Morphic Difference Machines. It seems that the difference-theoretical aspect of morphogrammatics gets an even more direct thematization and formalization in the context of an *analogon* to FSMs.

Some preliminary combinatorial elaborations are added with the application of SML-procedures.

The concept of asymmetric palindromes is sketched. “*Derrida's Machines*”

(<http://www.thinkartlab.com/pkl/media/DERRIDA'S%20MACHINES.pdf>) is the title of a continuing research program that went public 2004.

(Work in progress v. 0.8.5.5, Jan. 2013)

1. Some types of Automata

"But the paradox is that: In the language, there are only differences, without positive terms. That is the paradoxical truth." Ferdinand de Saussure

"(...) *Dans la langue il n'y a que des différences*. Bien plus: une différence suppose en général des termes positifs entre lesquels elle s'établit; mais dans la langue il n'y a que des différences *sans termes positifs*."

"Qu'on prenne le signifié ou le signifiant, la langue ne comporte ni des idées ni des sons qui préexisteraient au système linguistique, mais seulement des différences conceptuelles et des différences phoniques issues de ce système."

"Mais dire que tout est négatif dans la langue, cela n'est vrai que du signifié et du signifiant pris séparément; dès que l'on considère le signe dans sa totalité, on se trouve en présence d'une chose positive dans son ordre."
Ferdinand de Saussure, Cours de linguistique générale, Payot, 1975, p. 166-167.

1.1. FSA, IFA, CA and morphic Automata

1.1.1. Motivation: the ubiquity of automata

Automata are everywhere. They come in the form and realizations as mechanical, electro-mechanical, electric, electronic, chemical, etc. physical devices and paper. They are used to control traffic at the railway and underground stations, they serve for the tickets as tickets automata, or for cigarettes, and so on. Nobody has to care about their theoretical status, except theoreticians. Such automata in their simplest form are called *finite state automata* or finite state machines, or for short FSA or FSM. They are perfect models to study abstractly the behavior of simple physical automata.

For computer scientist, FSMs are perfect models for computation. Unfortunately they lack of a memory function. Therefore, the use of FSAs is limited. It is not a big deal to fill this gap. Augmented automata with memory are doing the job. Well known as pushdown automata and finally as *Turing* machines. Everybody knows their name but not necessarily how they work.

As usual in mathematics, there are further abstractions at hand. The mathematical concept of physical finite state automata gets a further abstraction: different symbolic FSAs that have the same behavior are abstractly equivalent. This defines the chain of physical automata to symbolic FSMs and to abstract equivalence classes of FSMs. One of those types of abstraction of FSMs is called *quotient* automaton.

The path to the mathematical abstractions is clear. After having used physical devices millions of times, an abstraction of its physical use to an abstract representation follows naturally. An abstract treatment of automata becomes crucial if the automata systems are growing into highly complex configurations.

All those abstract concepts of automata are faithful to their physical origins. Even the *Abstract State Machines*, ASM, of Yuri Gurevich is considered not with abstractions but with a more concrete representation of “*real world*” events. The states for ASMs are not just symbols but models, algebras, structures *representing* real world constellations.

Having lived long enough to have encountered million times physical automata and often enough their mathematization and their conceptual applications in all kinds of sciences, the question arises: Is there not time for a further ‘abstraction’? Even if this kind of abstraction turns out to be more a *reflection* and *subversion* than a mathematical abstraction, it might nevertheless be considered as a natural abstraction from the existing models of computation.

Does it really matter anymore *what* is processed, and on what level of abstraction these procedures happen? Whatever is processed in this classical approach, physically and theoretically, is based on *identity*. It certainly would be absurd to ask for an automaton in which its objects would dissolve into hot air while being processed.

But it isn’t specially absurd to focus on the *actions* of the automaton as such and to ask if the actions have to be considered as the same or as different. Nothing more. Sameness and differentness distributed in specific configurations of sameness and differentness are replacing ‘state-based’ symbolic concepts and their identity as equality and non-equality.

A machine concept that is thematizing and formalizing just the aspects of actional sameness and differentness is deliberated from its identity-theoretical heritage. With that, all constituents of the FSM are endangered: the states and transitions of the FSM are ‘bracket’ out and reserved for the classical, identity-based concepts of automata.

Such new kinds of machines shall be called *differentiation* machines. The use of the term differentiation might get more explanation with the connection to the concepts of *differences* and *distinctions*. Obviously all terms that have to be deconstructed and taken out of their origins and involvement into identity. This approach also shouldn't be confused with an attempt of "*programming the Ready-to-Hand*" of Heideggerian AI (Hubert L. Dreyfus) because latter doesn't attempt to deconstruct its own medium, the presumed programming languages as such.

"Ce mouvement (actif) de la (production de la) différence sans origine, n'aurait-on pu l'appeler, tout simplement et sans néographisme, différenciation? Entre autres confusions, un tel mot eût laissé penser à quelque unité organique, originaire et homogène, venant éventuellement à se diviser, à recevoir la différence comme un événement. Surtout, formé sur le verbe différencier, il annulerait la signification économique du détour, du délai temporisateur, du <<différer>>." (Derrida, Différance)

The new differentiation machine is 'calculating' differences that are distributed in a system that is defined by its differences and that is defining its differences.

The self-referentiality of this description of "differentiation" marks the departure of difference machines from state machines. State machines are based on identifiable atomic states in transitions with an initial and a final state, in finite or infinite steps. Difference machines are not based on "identifiers" which are identifying something (a sign) as something in the mode of "is-abstractions" but are *evoking* ("imaginative re-creation") something, i.e. themselves as something different.

This new approach shall be modeled in *analogy* to classical finite state machines as far as the new intuition survives. Other approaches to conceptualize and formalize differentiation machines shall follow.

<http://www.thinkartlab.com/pkl/lola/ConTeXtures.pdf>

How to classify morphogrammatic machines?

If we stipulated that a morphic machine might iteratively repeat its actional structure or it might decide to augment its differentiation by choosing an accretive repetition and differentiate its actional base further by accretively augmenting retrograde recursively its domain by a new, not yet decided and included differentiation, then we would have to offer a mathematical model that would be able to cope with such a structural demand.

It is stipulated that classical machine models are not designed to respond to such vivid situations.

A morphogrammatic machine might therefore be modeled in *analogy* to an organism that is adapting towards the requests and conditions of its morphic environment.

How does that contrast to Gurevich's Abstract State Machine (ASM) model of computation? It is understood that the ASM approach is one of the most advanced general models of computation.

"A state is an algebra (structure)"

"The central and new idea of ASM is easily described: It is the systematic way of how symbols occurring in the syntactic representation of a program are related to the real world items of a state. In fact, a state of an ASM may include any real world objects and functions. In particular, the ASM approach does not assume a symbolic, bit level representation of all components of a state."

http://www2.informatik.hu-berlin.de/top/download/publications/Reisig2006_classroom.pdf

The question is not if the ASM approach is covering "real world" objects and functions by its computational model. The question that comes first is probably more philosophical and fundamental and therefore not easily accepted by computer

scientists, the question is: Are “real world” data really just “abstract” and identifiable objects, i.e. events, occurrences, functions, relations, transitions, etc.?

“In particular, the ASM approach does not assume a symbolic, bit level representation of all components of a state. Herein it differs from standard computation models - and most obviously to Turing Machines - where a state is a (structured) collection of symbols.

“But conventional computation concentrates on the transformation of symbols, not dwelling too deeply on what they stand for.” (ibid., p. 3)

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

Elementary cellular automata are collections of simple finite state machines.

In earlier approaches, the order was inverse. The focus was on the ‘non’-alphabetic characteristics of kenoms (kenograms) and its paradoxical consequences, especially for the definition of a beginning of a formal language or an automaton. The new approach plays with the fact of the *Stirling* character of kenogram sequences and morphograms and a standard representation of the ‘non’-representable alphabet and kenogrammatic sequences and their non-linear constellations. A further reflection (deconstruction) has to take the ‘infinity’ of the stream-property of morphograms into account.

One of the most elucidate 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. (Parts are published by Fink Verlag 2012, ISBN: 978-3-7705-5419-5)

Like with Konrad Zuse, computation is defined by Gurevitch as a step-wise transition in time (Levin), guided by rules, from an initial to a terminal object, in the mode of finite or infinite, parallel or serial configurations, the result of the computation. This approach is extended without changing its basic concept to non-terminal and parallel situations too.

Computation (in the Framework of FSM $M = (Q, \Sigma, \delta, s_0, F)$)

1. $r_0 = q_0$, : initial
2. $\delta(r_i, \omega_{i+1}) = r_{i+1}$ for $0 \leq i < n$: transition
3. $r_n \in F$: terminal.

Also written as $r_0 \xrightarrow{a_1 a_2 \dots a_n} r_n$.

<http://www.cs.cmu.edu/~fp/courses/flac/lectures/lecture05.ps>

Obviously, the limits of this paradigm are clear: no *interactivity*. Computation is conceived as problem-solving and not as a *media* of interacting computational processes, without beginning nor end (Peter Wegner’s interactivity, Turing’s Oracle Machines).

Transition function

$$M = (Q, \Sigma, \delta, s_0, F)$$

Q : Alphabet

Σ : States

δ : state – transition function

s_0 : initial state

F : set of final states

FSM transition function : $\delta : S \times \Sigma \rightarrow S$.

Minimal tasks of deconstruction of FSM

The definition of M determines the minimal catalogue of deconstruction tasks :

Alphabet,	kenoms
States,	morphograms
transition ,	differentiation
initial and	observations, internal,
final states.	external,
regular language	morphogramatics
construction	deconstruction
recursion	retrogradeness, reflection
equivalence	bisimulation

1.1.2. Finite state machines

FSA

"The finite state automaton (FSA) or finite state machine is a very important model that has been widely used in computer science and industry. The automaton can perform very complex computational tasks with only finite internal states and fixed transition rules.

"Usually, there are two kinds of FSAs. Finite state *acceptors* (recognizers) only accept information and jump between different states but do not generate any output information. These machines are widely used as language recognizers. Another class is called finite state *transducers*, which are able to generate output information as well as accept input information. They can be designed as controllers.

Mealy and Moore

"Another class is called finite state *transducers*, which are able to generate output information as well as accept input information. They can be designed as controllers."

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 $\delta(q, a) = \{q'\}$." Dana Angluin et al, Mutation Systems

Wolfram's IFAs

"By adding a tape with finite size and some other constraints to the FSA, we can study the behavior just like one-dimensional cellular automata (CAs). Wolfram has enumerated all possible patterns of two-state two-color and three-state two-color IFAs."

Cellular automata

"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 .

"A *cellular automaton* $C = (\Sigma, \delta)$ 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 ."

Dana Angluin et al, Mutation Systems

<http://www.cs.yale.edu/homes/aspnes/papers/lata2011-proceedings.pdf>

Turing machines

Finite state machines have a limited memory. This restricts the range of computability. Turing machines are not finite machines but have an infinite tape to store information. Hence, their range of computability encompasses that of finite state machines. Memory is not disturbing the general concept of transitions.

Büchi automata

Considering *streams* of events, another kind of automata has to be introduced.

"A Büchi automaton is a type of ω -automaton, which extends a finite automaton to *infinite* inputs. It accepts an infinite input sequence iff there exists a run of the automaton that visits (at least) one of the *final* states infinitely often. Büchi automata recognize the *omega-regular* languages, the infinite word version of regular languages."

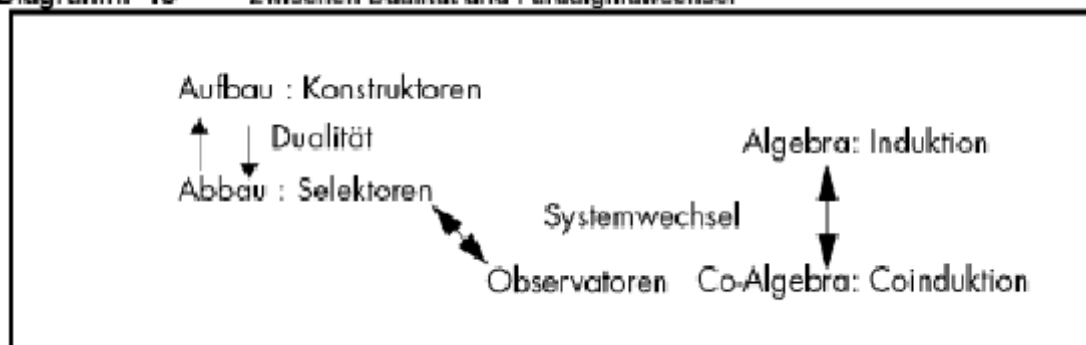
"An ω -automaton (or stream automaton) is a variation of finite automaton that runs on infinite, rather than finite, strings as input. Since ω -automata do not stop, they have a variety of acceptance conditions rather than simply a set of accepting states." (Pandya)

"Stream automata, e.g. {Büchi, Muller, parity }-automata, accept languages of infinite words (ω -regular languages)." (Venema)

1

¹Remarks to algebra and co-algebra for morphic streams (in German).

Diagramm 15 Zwischen Dualität und Paradigmawechsel



3

"Universal Coalgebra provides the notion of a coalgebra as the natural mathematical generalization of state-based evolving systems such as (infinite) words, trees, and transition systems."

<http://staff.science.uva.nl/~yde/papers/2010/font-auto2010.pdf>

1.1.3. Morphic automata

Morphic automata are an experimental concept developed in this paper. Morphic finite state machines (automata), MorphoFSA or MorphoFSM, are opting for a difference-oriented approach of computation, and are therefore contrasting to all existing kinds of abstract machines by their deconstruction of the identity of the concept of a state, its alphabet and its transitions.

Transitions for MorphoFSA are not relations or functions, thus transitions but differentiations. Differentiations are also not covert by the concept of *distinctions* in the sense of Spencer-Brown's *Laws of Form*. The concept of differentiations as applied for the introduction of finite differentiation machines, MorphoFSA, has to be separated from the *calculus of differentiation* as I have developed as a complementary calculus to the calculus of indication. There it was called a *Mersenne* calculus.

Therefore, MorphoFSA are at first build in *analogy* and used as a support for the strategy of surpassing the limits of identity-dominated machines. MorphoFSA are not dealing with states as sings, symbols, information, real world representations etc. but with *differences* as such. Differences in this sense are not relations, functions or operations between or with objects, sympols, signs. The leading metaphor for MorphoFSA is *living matter* in the sense of autopoietic structurations/differentiations/distinctions, and not symbolic or physical control systems of information processing.

1.2. Finite State Machines

1.2.1. Recalling definitions

JFLAP defines a finite automaton (FA) M as the quintuple

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

where

Q is a finite set of *states* $\{q_i \mid i \text{ is a nonnegative integer}\}$

Σ is the finite *input* alphabet

δ is the *transition* function, $\delta : D \rightarrow 2^Q$ where D is a finite subset of $Q \times \Sigma^*$

q_s (is member of Q) is the *initial* state

F (is a subset of Q) is the set of *final* states .

A string w is accepted by a finite automaton M iff there is a labeled path lp such that

- lp is valid for M ;
- the label of lp is w ;
- the start state of lp is the start state of M ; and
- the end state of lp is an accepting state of M .

3.4.1 Finite Automata

A finite automaton (FA) M consists of:

- a finite set Q_M of symbols (we call the elements of Q_M the states of M);
- an element s_M of Q_M (we call s_M the start state of M);
- a subset A_M of Q_M (we call the elements of A_M the accepting states of M);
- a finite subset T_M of $\{ (q, x, r) \mid q, r \in Q_M \text{ and } x \in \text{Str} \}$.

<http://www.jflap.org/>

Numeration

Following the successive construction of the word “abbba” in the formal language, the numeration of the transitions of the FA for the selected word of the formal language follows automatically.

Hence, a recognition of a word starts with its first element, continues linearly, step by step, and ends with its last element (halt state).

Properties for an acceptance

1. machine in acceptance (*halt*) state,
2. input is *exhausted*,
3. string *accepted*. (Parkes, p.55)

Example

Debugger ready with string “abbba”

Symbol = , remaining string = “abbba”, states = [], accepting = false

Symbol = a, remaining string = “bbba”, states = [2], accepting = true

Symbol = b, remaining string = “bba”, states = [1], accepting = true

Symbol = b, remaining string = “ba”, states = [1], accepting = true

Symbol = b, remaining string = “a”, states = [1], accepting = true

Debugger: stopped (no more characters to process)

Further literature

Thomas Hanneforth, Finite-state Machines: Theory and Applications (2010)

http://tagh.de/tom/wp-content/uploads/FSM_UnweightedAutomata.pdf

1.2.2. Motivations for morphic FSA

A first step of deconstruction

Abstractions from states in respect of actions: “turnOn” and “turnOff” are obviously morphogramatically equivalent. What is of interest from a morphogrammatic point of view is not *what* is changed, i.e. the semantics of the change, “On”, “Off”, but *how* it is changed, i.e. the *form* of the action involved. Thus its interactivity. And not any physical details. For both direction, “On” and “Off”, the same form of activity gets realized. That is, the same complexity/complication and time-structure of the action is involved. The opposite semantics of “On” and “Off” loses its significance if the focus is on the *inter-activity* instead on its material objects. In this scenario it strictly doesn't matter *what's* on the plate.

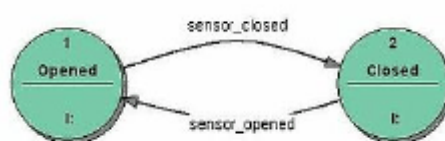
Interactivity means here: “neither open nor closed” and “open and closed at once”, hence the new (meta)state is just this paradoxical interplay. Therefore a meta-state is a *diamond* bi-object in the sense of diamond category theory.

“The number and names of the states typically depend on the different possible states of the memory, e.g. if the memory is three bits long, there are 8 possible states.” (Wiki)

Hence, $3^2 = 8$ gets reduced in the actional approach to $\text{Sum}(\text{StirlingSn2}(3, 2)) = 4$.

What might then a keno-state be? How can it be represented?

A nice model of a keno-state is just a model of a simple FSM, here a transducer model, itself.



keno-state = [1, open; 2, closed]

Thus, a keno-state represents the activity measured in m^n of a FSM as such in contrast

to the *form* of the activity measured as $\text{Sum}[\text{StirlingSn2}[m, n]]$.

Automaton	neither	front	rear	both
closed	closed	open	closed	closed
open	closed	open	open	open

[Automaton]	[neither/front]	[rear/both]
closed	closed/open	closed/closed
open	closed/open	open/open

With the strategy of *Stirling-abstractions* and standard *notation* of kenograms as replacement of identitive signs, the whole machinery of automata theory remains still applicable. Without this strategy of “abstraction and acception” it seems to be more or less impossible to advance and surpass, step-wise, philosophical speculations towards mathematical constructions. As a first attempt to understand the strategy, this step might be conceived just as a change in the *data structure* of the machine, from identitive to morphogrammatic ‘data’ structures. Thus, not yet touching the mechanism of the machine as such.

Hence, a kenogram sequences *kseq* is represented by a standard alphabet of signs in a lexical order.

A morphogram of a binary action is of the form [aa] or [ab].

If both 'states' are involved in the same kind of actions then nothing happens, i.e. no differentiation is involved: [aa].

If the 'states' are representing opposite actions the *form* of the interactivity is: [ab]. The action [aa] is realized in morphic FSM as a self-application (self-differentiation), while the action [ab] is realized in MorphoFSA as a differentiation.

This game might be continued for arbitrary length of morphograms with two and only two kenograms.

More interesting is the case for general morphograms of arbitrary complexity. Such a complexity of arbitrary morphograms is represented in MorphoFSA with the amount of *positions* of differences.

As for kenogrammatic *cellular* automata the crucial consequence of the morphogrammatic approach is demonstrated with the definition of the transition function with a transition (differentiation) from a *Cartesian* or *Cantorian* paradigm to a *Stirling* option.

[http://memristors.memristics.com/CA-](http://memristors.memristics.com/CA-Overview/Short%20Overview%20of%20Cellular%20Automata.pdf)

[Overview/Short%20Overview%20of%20Cellular%20Automata.pdf](http://memristors.memristics.com/CA-Overview/Short%20Overview%20of%20Cellular%20Automata.pdf)

Glossary for FSM

1) **FSM** A collection of states and transitions that outline a path of actions that may occur.

2) **State** A state is a position in time. For example, when you are at the bus stop, you are currently in a waiting state.

3) **Event** An event is something that happens in time. For example, the bus has arrived.

4) **Action** A task performed given a certain event that occurred. For example, you enter the bus.

5) **Transition** A link between 2 states. May be unidirectional or bidirectional.

New:

0) **Keno** A collection of interactivities between the transitions of FSMs as objects.

A state in a kenoFSM is a standard representation of an interactivity of a transition (transformation) and therefore classical states are conceived as automorphisms, i.e. as interactivity onto itself.

http://www.generation5.org/content/2003/FSM_Tutorial.asp

With this model of keno-states or meta-states it is easy to understand the reduction of FSM with its Cartesian combinatorics to kenoFSM with their Stirling combinatorics.

transition-substitution = (Objects = {signs, patterns, monomorphies}, Operations = {concatenation, fusion, bisimilarity})

symbolicCA = [grid=lattice, cells=atomic, signs, concatenation]

kenoCA = [grid, cells, patterns, concatenation]

morphoCA = [grid=multi-layers, cells=leveled, monomorphism, fusion]

transition-substitution = (concatenation, fusion, bisimilarity).

"Substitutions transform a sequence into another sequence. So do other mechanisms known as cellular automata."

<http://www.dtic.mil/dtic/tr/fulltext/u2/p010899.pdf>

NextGen for global elementary CA rules

NextStep: $C_i \rightarrow C_{i+1}$

$\forall i, j: C_{i+1} = \text{Rule}(C_i)$

local rule: $f: S^n \rightarrow S \Rightarrow f_{\text{keno}}: \text{Sn2}(n, S) \rightarrow S$

global transition function: $G: S^{Z^d} \rightarrow S^{Z^d} \Rightarrow G_{\text{keno}}: \text{Sn2}(Z^d, S) \rightarrow \text{Sn2}(Z^d, S)$.

2. Morphogrammatic FSA

2.1. Some descriptions

Morphograms of morphogrammatic languages, interpreted as kenogram sequences, *kseq*, are build step-wise retrograde-recursively and not just recursively as for strings of a formal language.

The class of words ω over an alphabet Σ for a formal language is: $\Sigma^* =$

$\{\omega_1\omega_2, \dots, \omega_n \mid k \geq 0, \forall \omega_i \in \Sigma\}$.

Example

$\Sigma = \{a, b, c\}$, then Σ^* is the set: $\{\epsilon, a, b, c, aa, an, ac, ba, bb, bc, ca, cb, cc, aaa, aab, aac, \dots\}$

The class of morphograms μ over a kenogrammatic 'sign repertoire' K in standard normal form is: $\text{Sum}(\text{Sn2}(K, n))$.

Example

$K = \{a, b, c\}$, then K^* is the set: $\{\epsilon, a, aa, ab, aaa, aab, aba, abb, abc, \dots\}$.

The semiotic universe is Σ^* is defined by the star or Kleene closure:

$\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma_i = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3 \cup \dots \cup \Sigma_n$.

The kenomic (morphogrammatic)trito-universe TU is defined as $K = ([1], \text{Tsucc})$ with $val TU = from [1]$.

This construction of the trito-universe TU is based on an application of *lazy*-lists, that

is a realization of the concept “*evaluation-by-need*”.

<http://www.thinkartlab.com/pkl/SML-sources.NJ/ALL-MG-nov2012.sml>

The automata FSA are recognizing words and languages from Σ^* , while the MorphoFSA are recognizing morphograms from K^* .

A language accepted by FSA is the set of words accepted by the automaton. Similarly, morphic languages accepted by a MorphoFSA are set of morphograms accepted by MorphoFSA. A word or a morphogram is accepted if the machine has an *accepted* final run for the word.

[http://memristors.memristics.com/CA-](http://memristors.memristics.com/CA-Overview/Short%20Overview%20of%20Cellular%20Automata.pdf)

[Overview/Short%20Overview%20of%20Cellular%20Automata.pdf](http://memristors.memristics.com/Graphematics/Graphematics%20of%20Cellular%20Automata.html)

<http://memristors.memristics.com/Graphematics/Graphematics%20of%20Cellular%20Automata.html>

That’s so far the *analogy*.

Morphogramatics was developed in analogy to *recursive* word arithmetics in the book “*Morphogrammatik*” 1993, applying extensively methods of lazy lists and lazy programming.

Hence, methods of producing recursively, and computing morphograms with the SML/NJ program, defined well the universe of kenogramatics and morphogramatics.

But as for regular formal languages, it is another question to decide (recognize, accept) if a morphogram belongs to a specified morphogrammatic script or not. Finite state machines are applied to this decision problem for regular symbolic languages.

It is proposed with this paper that MorphoFSA, i.e. morphogrammatic ‘finite state machines’, are doing the job for all specified morphogrammatic scriptures.

A calculus is defined by 2 alphabets:

1. the alphabet of signs,
2. the alphabet of variables.

Two semiotic words of a FSA are equal iff they are of the same length and all the occurrences of its atomic signs are equal at the same places of the comperaed words (strings).

In contrast, two morphograms of a MorphoFSA are MG-equivalent iff they have the same EN structure:

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

ML-definition

fun from ts = Cons(ts,fn () => from (Tsucc ts));

The set of all trito-sequences (morphograms) is calculated by *val TU = from [1];*

The first elements of the *lazy* list of morphograms is given by *nfirstq (n, seq)*.

fun nfirstq (0, xq)=[]

| nfirstq (n, Nil)=[]

| nfirstq (n, Cons(x,xf))= x::(nfirstq (n-1, xf ()));

Example

- nfirstq(23,TU);

> val it =

*[[1],[1,1],[1,2],[1,1,1],[1,1,2],[1,2,1],[1,2,2],[1,2,3],[1,1,1,1],
[1,1,1,2],[1,1,2,1],[1,1,2,2],[1,1,2,3],[1,2,1,1],[1,2,1,2],[1,2,1,3],[1,2,2,1],
[1,2,2,2],[1,2,2,3],[1,2,3,1],[1,2,3,2],[1,2,3,3],[1,2,3,4]] : int list list*

-nfirstq (55, TU)⁴

The hint to *lazy lists* shows an important difference to algebraic list definitions of lists and finite state automata. Lazy lists are *streams* (sequences, Paulson) that are evaluated “*by need*”. Hence, problems of “infinite” streams for automata or automata of streams have to be analysed.

Enumeration

Morphograms are not build up of elements of an alphabet but are defined by the differences between their “elements”, i.e. the kenograms. Therefore, the enumeration of the “elements” of a morphogram has to adapt to the special property of retrograde recursivity.

The best way to enumerate the constituents of a morphogram is therefore to enumerate its differences.

Because morphograms are patterns and not sequences or lists, there are some options how to enumerate and where to start the numeration of its differences.

Hence, a start or initial state is not a property of a morphogram as it is necessary for a string but a property of an observation. The observation is deciding with which element a description or recognition of a pattern will be opted as a start.

One aspect of morphograms is their retrograde-recursive construction, the other is the recognition procedure by an automaton of a ‘encountered’ (given) morphogram.

A convenient way to do this was introduced by the so-called epsilon/nu-enumeration, ϵ/v -enumeration of the position of the kenomic differences.

A word, here a morphogram, is read then as a sequence of ϵ/v -situations.

What counts is the transition or move of differentiation from one position (state) to another position. This is realizing a v -difference or a move into itself, realizing an ϵ -difference. These two types of differences correspond to the distinction of iteration and accretion in kenogrammatcs.

The labels of the differentiations from one position to another are the number of the differentiations or the number of runs, and not the elements of an alphabet to be used or recognized.

The positions might be identified with the number of different kenograms involved in the definition of the morphogram.

There are only two kinds of moves (transitions, differentiations) for a kenomic SFM: ϵ - or v -transitions. But strictly, those ‘moves’ are not moves in a literal sense but acts of differentiations.

Differentiation happens by the amount of positions (states) of the automaton, and not by the amount of elements of a sign-repertoire. This corresponds to the difference-theoretical approach that signs (keno- and morphograms) are determined by located differences of the texture.

Tape and matrix

FSA are reading their words by reading step-wise the elements of the word from a *tape*.

MorphoFSA are reading their morphogram according to a reading convention from a *matrix*, where the morphogram is inscribed as a pattern, i.e. a grid of differences. The pattern-structure is not dictating a singular linear step-by-step reading as it is the case for the linear strings of FSA-words.

2.2. The epsilon/nu-structure of morphograms

2.2.1. Differentiation and enumeration

Encountered a string from a textual environment, say, as a possible input of a morphic automaton, we have to decide as what kind of text we want to thematize it. If we decide to thematize the textual event not as a semiotic, indicational, Mersenne or other type of sign, but as a kenogrammatic pattern, i.e. as a morphogram, we might have finally to decide on which level of the scriptural system of graphematics the event shall be accepted. Here, all patterns are understood as belonging to the trito-structure of kenogrammatics, thus, we are dealing with morphograms. This decision invites to build the epsilon/nu-structure (ENstructure) of the event (morphogram), now considered as a string or a pattern of kenograms. The ENstructure of this pattern gives the structure of the distributed differences of the pattern, denoted by “ ϵ ” for sameness (equivalence), and “ ν ” for differentness of the differences of the pattern.

Complexions of MorphoFSA (M, n)

In this setting up of morphogrammatics for the purpose to introduce morphic FSA, there are just these two kinds of differences that are differentiating between *sameness* and *differentness* in respect of the systematics of the automaton. Other differentiations are introduced for complexions of morphogrammatic systems, like MorphoFSA^(m, n), where intra- and trans-contextural differences enter the game. Such complex MorphoFSAs are defined by the distinctions: (ϵ, ν, II), with II for dissemination.

The transition rules and the order of their occurrence in the pattern (morphogram) of the morphic FSA are defined by the enumerated sequence of those distinctions.

The minimal number of positions (states) of the morphic FSM is given by the aggregation of the pattern, $\text{AG}(\text{Str}) = n$.

The ϵ/ν -structure of a morphogram (kseq) gets calculated by the ML-function ENstructure: type enstruc = (int*int*EN) list list;

Example.

[abac] :

$(\epsilon, \nu) - \text{comp}([abac]) = (\nu\nu\epsilon\nu\nu\nu)$

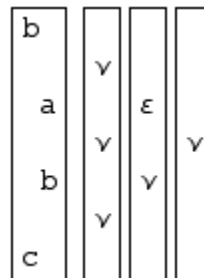
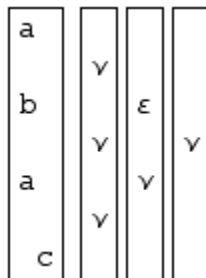
$\implies [abac] = \text{kg}[babac]$

$\text{AG}([abac]) = 3$

[babac] :

$(\epsilon, \nu) - \text{comp}([babac]) = (\nu\nu\epsilon\nu\nu\nu)$

$\implies [babac] = \text{kg}[abac]$



ENstructure of trito-events [A] and [B]

EN([A]):

- ENstructure ["a", "a", "b", "c"];

> [[],

[(1,2, E),

[(1, 3, N), (2, 3, N)],

[(1, 4, N), (2, 4, N), (3, 4, N)]]: enstruc.


```

EN([B]):
- ENstructure ["▲", "▲", "■", "◇"];
> [],
  [(1,2, E),
   [(1, 3, N), (2, 3, N)],
   [(1, 4, N), (2, 4, N), (3, 4, N)]]: enstruct.

```

Equivalence based on EN

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

ENtoKS

ENtoKS builds the morphogram, ks, in standard notation, tnf, out of the ENstructure.

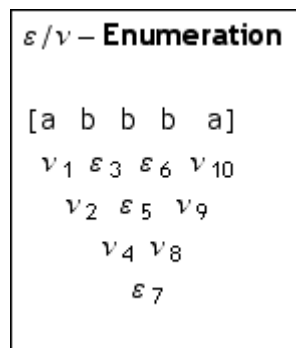
ENtoKS(ENstructure ks) = ks

```

ENtoKS [],
  [(1,2, E),
   [(1, 3, N), (2, 3, N)],
   [(1, 4, N), (2, 4, N), (3, 4, N)]] = [1, 1, 2, 3].

```

Numeration of the e/v-tupels by k(i,j) and subsystems n



Number of a subsystem at place (i, j):

```
fun k (i,j)=((j*(j-1)) div 2)-i+1;
```

Enumeration of the subsystems for n

```

fun subsystems n=
  sort(map (fn [i,j] => (k(i,j),[i,j]))
        (maufn n 2));

```

Examples

```

- k(4,7);
val it = 18 : int
- subsystems 7;
val it =
  [(1,[1,2]),(2,[2,3]),(3,[1,3]),(4,[3,4]),(5,[2,4]),(6,[1,4]),(7,[4,5]),
   (8,[3,5]),(9,[2,5]),(10,[1,5]),(11,[5,6]),(12,[4,6]),(13,[3,6]),(14,[2,6]),
   (15,[1,6]),(16,[6,7]),(17,[5,7]),(18,[4,7]),(19,[3,7]),(20,[2,7]),
   (21,[1,7])] : (int * int list) list

```

2.2.2. The systematic status of morphograms

Just abstractions?

Again, the discussion of the *systematic* status of morphograms, say its epistemological, ontological, semiotical, mathematical, etc., status, is as old as their introduction by Gunther in the '60s.

It was never denied that morphograms originated at first by a set-theoretically defined mathematical abstraction from the 'combinatorial' truth tables of propositional semantic-based logic. And then generalized by Dieter Schadach's classification system of abstractions. Nevertheless, the question about the status of morphograms is by no way answered by the insistence on this fact of abstraction. The question is much more what was the *use* of this abstraction? Where did it led? And is

this use of the abstraction establishing an abstract level upon the truth table, like quotient structures of model theory, or is their use by Gunther not in fact a deconstruction of the hierarchy of truth-tables and their abstraction?

Philosophically, abstractions are building ideal structures, the declared aims of subversions are the deconstruction of ideality. Abstractions are establishing meta-structures, subversions are unmasking deep-structures of semiotic systems.

The use of the morphograms of classical truth-functional logic led to the discovery of its “*morphogrammatical incompleteness*”. Such an incompleteness, i.e. the addition of transjunctional morphograms to the abstractively gathered junctional morphograms of the truth functions is obviously not covered by the rules of abstraction as such. Hence, the symmetry of base structure and abstraction over it is disturbed towards an asymmetry. Abstractions, combined with deconstructive applications, are characterizing the methods of introducing morphograms not as abstractions but as a result of *subversion*.

Needless the mention that the abstractive aspects of this subversion is saved and productively used for the study of mathematical properties of morphograms, morphogrammatiks and kenogrammatiks. Especially studies in the combinatorics of kenogrammatiks had been crucial to define the new territory of reflection.

Therefore, there is no surprise that the difference-theoretical characterization of morphograms by the epsilon/nu-structure is faithful to its abstract counter-part, the equivalence classes build over strings of signs. As clearly elaborated in the book “*Morphogrammatik*” there are different ways to uncover morphograms, and one, certainly, is the application of equivalence classes.

Hence, there is an isomorphism between the equivalence classes that are defining morphograms and the representation of the morphograms by the e/v-structure. This fact is well ruled by the ML-functions *ENstructure* and *ENtoKS*. Both are translating between the “equivalence classes” EN and KS, i.e. e/v-structure and kenogram sequence KS.

In the case of quotient FSMs, the abstraction happens over the alphabet Σ . “For $\forall x, y \in \Sigma^*$ and $a \in \Sigma$: \equiv_A is an *equivalence* relation over Σ^* .” (cf. § 3.6.1)

Special abstractions over the kenogrammatic trito-structure of morphic FSMs are well known as *deutero*- and *proto*-structures. As shown with the general system of *graphematics*, several more abstractions had been introduced quite early in the ‘60s, and elaborated in several papers later.

The parlance “*that is just this and that, and nothing more*” is not explaining why there are no similar formal theories in mathematics and logic as they had been developed under the presumption of the kenogrammatic subversion.

One of such subversive constructions is disseminating the whole machinery of set or category theory over a kenomic grid. Then, there are irreducibly distributed chances to build multitudes of different types of equivalence classes at hand. The attempt to build again equivalence classes over distributed notions and constructions of equivalence classes is “just” building an additional candidate of the distribution, and “nothing else”, especially no *sublimation* of the differences between distributed polycontextural theories.

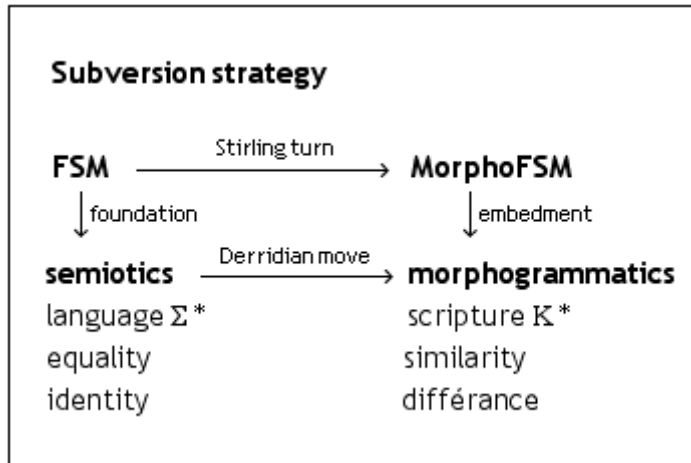
This paper is not going into the intriguing arguments and constructions for a more elaborated introduction of kenograms and morphograms. A lot had been developed in recent papers. For example:

<http://www.thinkartlab.com/pkl/media/Chinese%20Grammar/What%20Chinese%20Grammar.pdf>

Also semiotic terms in this paper are used vaguely, or at least in reference to the literature not mentioned here, and restricted to the context of formal languages, where a symbol or sign is used just as a mark.

Mathematical semiotics is ingeniously developed by the semiotician Alfred Toth and presented as an impressive research output at: <http://www.mathematical-semiotics.com/>

2.2.3. General scheme of a subversion strategy



<http://memristors.memristics.com/CA-Overview/Short%20Overview%20of%20Cellular%20Automata.pdf>

2.2.4. Number of symbolic representations

How many identive (symbolic) strings are represented by a morphogram, respectively by a single MorphoFSM?

To deal with abstractions needs representations. A tritogram [abb] is written in trito-normal form (tnf) and has therefore a number of different representation that are equivalent to the tritogram.

For the case of just 3 elements involved, the *tritogram* has a representation of 6 concrete symbolic realizations, i.e. {(abb), (acc), (baa), (bcc), (caa), (cbb)}, all representing the tritogram [abb]. In a more complex environment, the 3 place pattern might be occupied by further kenograms, say "d", "e", delivering patterns like [add] or [bee], etc. which are all morphogrammatically equivalent to the pattern [abb] in trito-normal form.

This number is calculated by the formula $\text{card}[\mu]_{\text{trito}}$:

$$\text{card}[\mu]_{\text{trito}} = \frac{m!}{(m-k)!}$$

Hence, a morphogrammatic FSA is representing semiotically different formal languages of the same structuration.

<http://memristors.memristics.com/Graphematics/Graphematics%20of%20Cellular%20Automata.html>

<http://www.ballonoffconsulting.com/PDF/1987AppendixII.pdf>

2.2.5. FSA equivalence, isomorphism and minimization

An interesting application of the 'representation theorem' for MorphoFSM in respect to FSMs might be a 'mediative' interpretation of the concept of *equivalence* and *isomorphism* between FSMs. Hence, the classical

approach of equivalence, isomorphism and minimization gets an additional level in the tectonics of graphematic scriptures, called here, instantiational representations (instantiations, representations).

Equivalence of FSMs

"Two states s_i and s_j are *equivalent* if and only if for every input sequence the machine will produce the same output sequence regardless of whether s_i or s_j is the initial state; i.e., for an arbitrary input sequence x , $\lambda(s_i, x) = \lambda(s_j, x)$. Otherwise, the two states are *inequivalent*, and there exists an input sequence x such that $\lambda(s_i, x) \neq \lambda(s_j, x)$; in this case, such an input sequence is called a separating sequence of the two inequivalent states.

"For two states in different machines with the same input and output sets, equivalence is defined similarly. Two machines M and M' are equivalent if and only for every state in M there is a corresponding equivalent state in M' and vice versa.

Machine equivalence is an *equivalence relation* on all the FSM's with the same inputs and outputs."

<http://www.cse.ohio-state.edu/~lee/english/pdf/ieee-proceeding-survey.pdf>

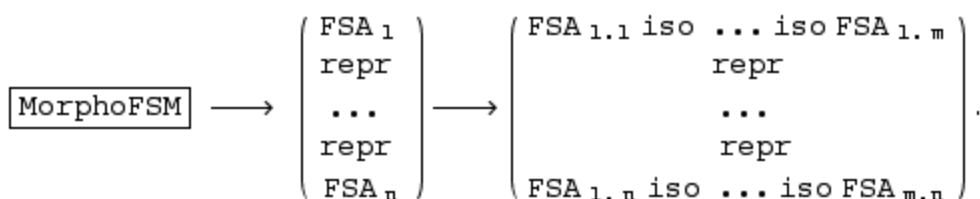
Isomorphism between FSMs

"Two machines are called *isomorphic* if there is an isomorphism from one to the other. Obviously, two isomorphic FSM's are equivalent; the converse is not true in general." (ibd.)

MorphoFSM is representing the morphogrammatically equivalent machines FSA_1, \dots, FSA_n , $n = \text{card}[\mu]$ as distributed separated machines in the constellation ruled by the machine MorphoFSM and the concept of morphogrammatic equivalence between the represented FSAs.

The opposite direction of the representation is abstracting from the isomorphic FSA's the equivalence class of FSAs FSA/iso .

Instantiations and representations



Morphogrammatic instantiation

$$\text{repr} : \{FSA_1 \text{ repr } \dots \text{ repr } FSA_n\} \Longrightarrow FSA^{(n)}$$

Isomorphism :

$$\text{iso} : \{FSA_{1.1} \text{ iso } \dots \text{ iso } FSA_{1,n}, \dots, FSA_{1.n} \text{ iso } \dots \text{ iso } FSA_{m,n}\} \Longrightarrow FSA^{(m,n)}$$

Minimization for FSMs

"Machine equivalence is an equivalence relation on all the FSM's with the same inputs and outputs. In each equivalence class there is a machine with the minimal number of states, called a minimized (reduced) machine. A machine is minimized if and only if no two states are equivalent.

"In an equivalence class, any two minimized machines have the same number of states; furthermore, there is a one-to-one correspondence between equivalent states, which gives an isomorphism between the two machines. That is, the minimized machine in an equivalence class is unique up to isomorphism." (ibid.)

Example

Morphic Automaton M1

M1 = ($\{\text{pos1}, \text{pos2}\}$, $\{v_1, v_2, \epsilon_3\}$, Δ , pos1 , $\{\text{pos2}\}$)

Differences: $\Delta = \{v_i, \epsilon_j, i, j \in \mathbb{N}\}$

Positions: $\{\text{pos}_1, \text{pos}_2\}$

Initial: $\{\text{pos}_1\}$

Acceptance: $\{\text{pos1}\}$

Differentiations:

$\text{pos}_1, v_1 \rightarrow \text{pos}_2$

$\text{pos}_2, v_2 \rightarrow \text{pos1}$

$\text{pos}_1, \epsilon_3 \rightarrow \text{pos}_1$

Final: $\{\text{pos}_1\}$.

M1 is accepting the *morphic* machine based on the morphogram [abb] in trito-normal form.

M1 is representing the *symbolic* machines FSA1 - FSA6 based on the symbolic representation of the morphogram [abb] in trito-normal form:

FSA1(abb), with alphabet $\Sigma_1 = \{a, b\}$,

FSA2(acc), with alphabet $\Sigma_2 = \{a, c\}$,

FSA3(baa), with alphabet $\Sigma_3 = \{b, a\}$,

FSA4(bcc), with alphabet $\Sigma_4 = \{b, c\}$,

FSA5(caa), with alphabet $\Sigma_5 = \{c, a\}$,

FSA6(cbb), with alphabet $\Sigma_6 = \{c, b\}$.

The case of further alphabets, say with $\Sigma = \{d, e\}$, wouldn't be a proper morphogrammatic representation but a possible redundant interpretation.

Symbolic automaton FSA (A)

A = ($\{q_1, q_2\}$, $\{0, 1\}$, δ , q_1 , $\{q_1\}$)

Alphabet: $\Sigma = \{0, 1\}$

States: $\{q_1, q_2\}$

Initial: $\{q_1\}$

Acceptance: $\{q_1\}$

Transitions:

$q_1, 0 \rightarrow q_1$

$q_1, 1 \rightarrow q_2$

$q_2, 0 \rightarrow q_1$

Final: $\{q_1\}$.

FSA representations of MorphoFSM[abb]

FSA1 = ($\{q_1, q_2\}$, $\{0, 1\}$, δ , q_1 , $\{q_1\}$)

FSA2 = ($\{q_1, q_2\}$, $\{0, 2\}$, δ , q_1 , $\{q_1\}$)

FSA3 = ($\{q_1, q_2\}$, $\{1, 0\}$, δ , q_1 , $\{q_1\}$)

FSA4 = ($\{q_1, q_2\}$, $\{1, 2\}$, δ , q_1 , $\{q_1\}$)

FSA5 = ($\{q_1, q_2\}$, $\{2, 0\}$, δ , q_1 , $\{q_1\}$),

FSA6 = ($\{q_1, q_2\}$, $\{2, 1\}$, δ , q_1 , $\{q_1\}$).

Isomorphism

With the change of the *state* sets and *transition* rules of a given FSA, different isomorphic FSA of the original FSA might be introduced for each symbolic representation FSA1 to FSA6 of the primary MorphoFSM.

To each symbolic FSA_i with alphabets Σ_i there are a number *m* of *isomorphic* symbolic machines FSA_i.1, ..., FSA_i.m.

Hence, the task of *minimization* of machine realizations appears.

3. Morphogrammatic FSAs

3.1. First definitions of morphogrammatic FSA

3.1.1. MorphoFSA examples

Automaton M1

$M1 = (\{pos1, pos2\}, \{v_1, v_2, \epsilon_3\}, \Delta, pos1, \{pos2\})$

Differences: $\Delta = \{v_i, \epsilon_j, i, j \in \mathbb{N}\}$

Positions: $\{pos_1, pos_2\}$

Initial: $\{pos_1\}$

Acceptance: $\{pos2\}$

Differentiations:

$pos_1, v_1 \rightarrow pos_2$

$pos_2, v_2 \rightarrow pos_1$

$pos_1, \epsilon_3 \rightarrow pos_1$

Final: $\{pos_2\}$.

M1 is accepting : [abb].

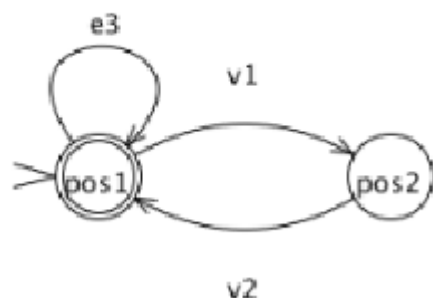
The acceptance for the morphogram [abb] is accepting the acceptance conditions as they are defined for FSA:

1. machine is in the acceptance (*halt, final*) state,
2. input is *exhausted*, i.e. all the epsilon/nu-differences of the initial EN-structure of the morphogram are read,
3. string *accepted*.

Again, the notation of the morphogram [abb] is just a representation in trito-standard normal form, *tnf*, of the ENstructure $(v_1 v_2 \epsilon_3)$ with $((a-b)_1 = v_1, (a-b)_2 = v_2, (b-b)_3 = \epsilon_3)$ of the morphogram.

Therefore, the accepted language of MorphoFSA is $L(M1) = \{\mu \mid \mu \text{ repeats a } v\text{-differentiation and ends in an } \epsilon\text{-differentiation}\}$.

MorphoFSA M1



Contrast : Automaton FSA (A)

$A = (\{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_1\})$

Alphabet: $\Sigma = \{0, 1\}$

States: $\{q_1, q_2\}$

Initial: $\{q_1\}$

Acceptance: $\{q_1\}$

Transitions:

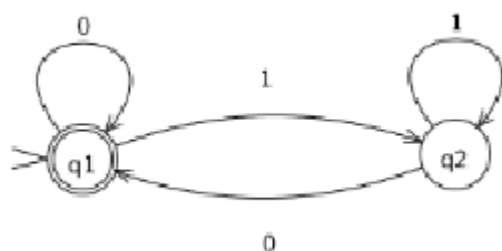
$q_1, 0 \rightarrow q_1$

$q_1, 1 \rightarrow q_2$

$q_2, 0 \rightarrow q_1$

Final: $\{q_1\}$.

Diagram for FSA A



A is accepting : (100), (11000), (01100), ...,

The accepted language of FSA A is $L(A) = \{\alpha \mid \alpha \text{ is the empty string } \varepsilon \text{ or ends in a } 0\}$.
(Sipser, p. 38)

The MorphoFSA M1 accepts all patterns equivalent to the pattern [abb] in standard-normal-form. Hence (baa), (bcc), (caa), etc. are accepted by M1. This acceptance can be seen as a first run of M1 or as the FSA definition of the morphogram [abb]. Therefore, machine M1 accepts the regular trito-languages $\{[a]^1[b]^n : n \geq 1\}$. That is: $\{v_{1,2}^n \varepsilon_3^m : n, m \geq 1\}$.

In contrast, the FSA A accepts just all symbolic regular languages ending in 0.

For $|\Sigma| = m$, the machine M1 accepts $n * l$ regular *symbolic* languages $\{a^1b^n, a^1c^n, \dots, b^1c^n, \dots\}$.

Thus, for $|\Sigma| = 3$ and $Q = 2$, there are 6 regular symbolic languages accepted by MorphoFSA M1:

Words of M1: $\mu \in L(M1) = \{v_1v_2\varepsilon_3\}$.

$M1(L(3,2)) = \{a^1b^n, b^1a^n, a^1c^n, b^1c^n, c^1a^n, c^1b^n : n \geq 1\}$.

An iteration of M1, i.e. a second run, is not changing the pattern of the machine M1 albeit the "final state" changes from position pos_1 to position pos_2 .

Automaton M1.0

Differences: $\Delta = \{v_i, \varepsilon_j, i, j \in \mathbb{N}\}$

Positions: $\{pos_1, pos_2\}$

Initial: $\{pos_1\}$

Acceptance: $\{pos_1\}$

Differentiations:

$pos_1, v_{1,4} \rightarrow pos_2$

$pos_2, v_{2,6} \rightarrow pos_1$

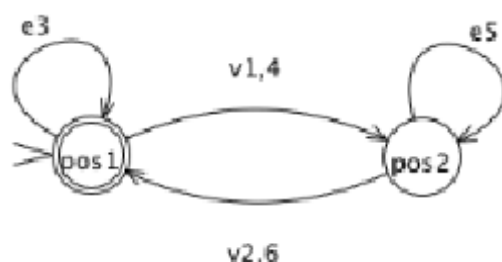
$pos_1, \varepsilon_3 \rightarrow pos_1$

$pos_2, \varepsilon_5 \rightarrow pos_2$

Final: $\{pos_1\}$.

This machine M1.0 is in conflict with the ENstructure of morphograms. That is, the e5-differentiation and the v6-differentiation are in conflict. The only prolongations of [abb] are [abba], [abbb] and [abbc]. The machine M1' is accepting [abba], and the machine M1.1 is accepting [abbb].

Diagram of M1.0



3.1.2. Further examples of MorphoFSA

Automaton M1'

Differences: $\Delta = \{v_i, \varepsilon_j, i, j \in \mathbb{N}\}$

Positions: $\{\text{pos}_1, \text{pos}_2\}$

Initial: $\{\text{pos}_1\}$

Acceptance: $\{\text{pos}_2\}$

Differentiations:

$\text{pos}_1, v_{1,5} \dashrightarrow \text{pos}_2$

$\text{pos}_2, v_{2,6} \dashrightarrow \text{pos}_1$

$\text{pos}_1, \varepsilon_{3,4} \dashrightarrow \text{pos}_1$

Final: $\{\text{pos}_1\}$.

M1' is retrograde iteratively accepting: [abba].

Diagram of M1'

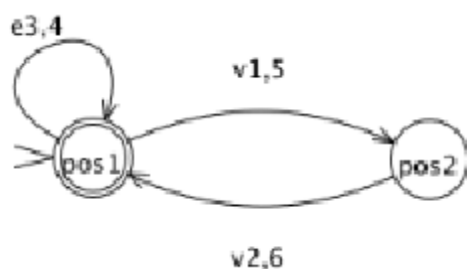
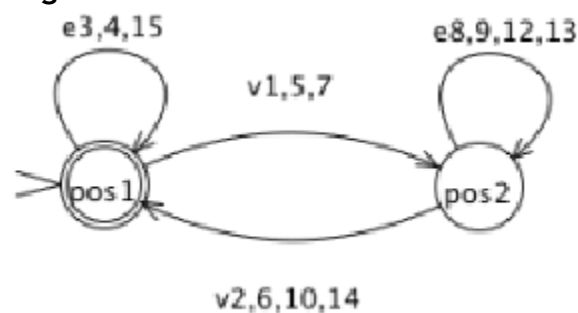


Diagram for M1''



M1'' is a retrograde iteratively accepting: [abbabb].

Automaton M1.1

Differences: $\Delta = \{v_i, \varepsilon_j, i, j \in \mathbb{N}\}$

Positions: $\{\text{pos}_1, \text{pos}_2\}$

Initial: $\{\text{pos}_1\}$

Acceptance: {pos2}

Differentiations:

pos₁, v^{1,4} --> pos₂

pos₂, v₂ --> pos₁

pos₁, ε₃ --> pos₁

pos₂, ε_{5,6} --> pos₂

Final: {pos₂}.

M1.1 as a retrograde iteration of M1 is accepting : [abbb].

The word [abbb] is morphogram μ with μ = (v₁v₂ε₃ v₄ε₅ ε₆), short:(vve vε ε).

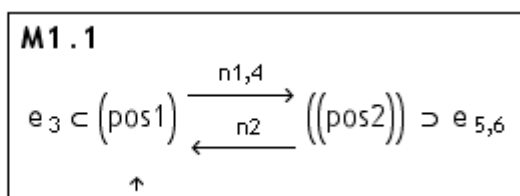
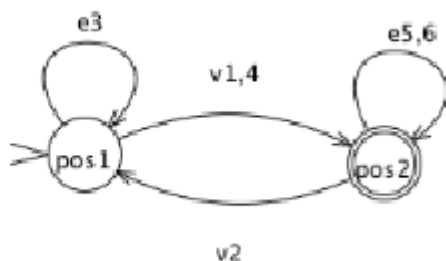


Diagram M1.1



Automaton M1.1.1

Differences: Δ = {v_i, ε_j, i,j ∈ ℕ}

Positions: {pos₁, pos₂}

Initial: {pos₁}

Acceptance: {pos₁}

Differentiations:

pos₁, v^{1,4} --> pos₂

pos₂, v^{2,7} --> pos₁

pos₁, ε^{3,8,9,10} --> pos₁

pos₂, ε_{5,6} --> pos₂

Final: {pos₁}.

M1.1.1 as a retrograde iteration of M1.1 is accepting : [abbbb].

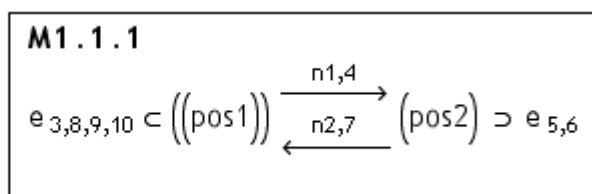
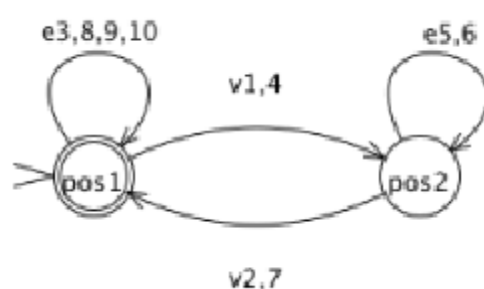


Diagram M1 . 1 . 1



General scheme for M1

$$e_{h1,h2,\dots,hn} \subset ((pos1)) \xrightleftharpoons[n_{j1,j2,\dots,jn}]{n_{i1,i2,\dots,in}} ((pos2)) \supset e_{k1,k2,\dots,kn}$$

$$i, j, h, k \in s(m)$$

Machine diagrams for basic morphogrammatic constellations

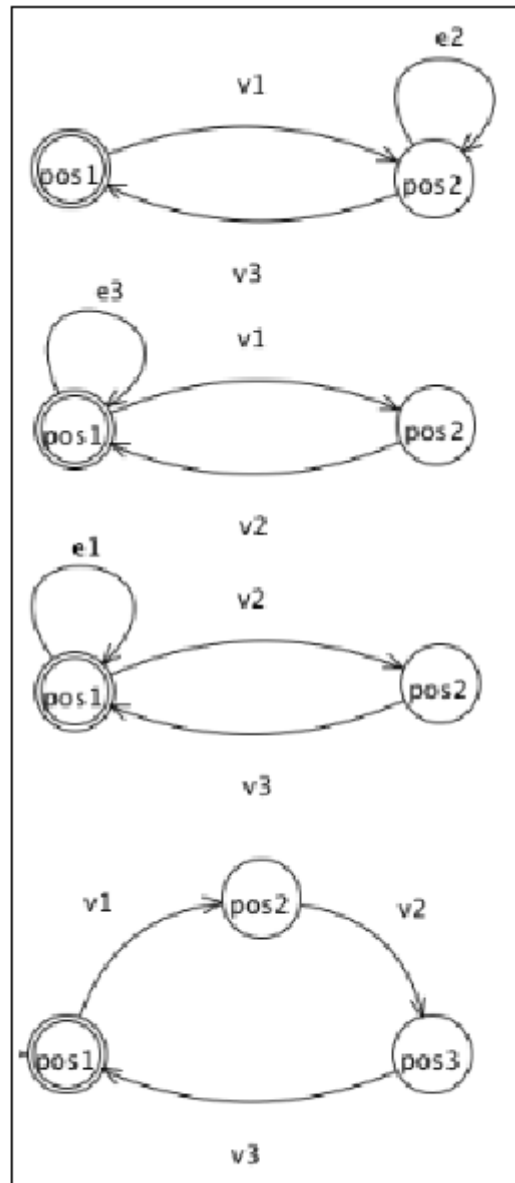
Basic constellations are [aa], [ab], [aba], [abb], [aab] and [abc].

Iteration

$$[aa] : e1 \subset ((pos1))$$

Accretion

$$[ab] : (pos1) \xrightarrow{v1} ((pos2))$$

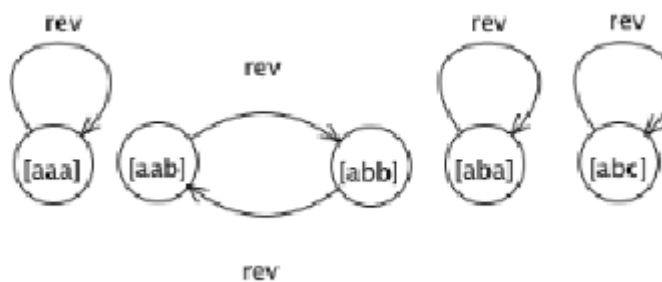


Reversion of basic machines

Basic machines = $\{[aa], [ab], [aba], [abb], [aab], [abc]\}$.

$\text{rev}([aa]) = [aa]$, $\text{rev}([ab]) = [ab]$,

$\text{rev}([aaa]) = [aaa]$, $\text{rev}([aba]) = [aba]$, $\text{rev}([abb]) = [aab]$, $\text{rev}([aab]) = [abb]$, $\text{rev}([abc]) = [abc]$



Iterations

Automaton M1.1.1.1

Differences: $\Delta = \{v_i, \varepsilon_j, i, j \in \mathbb{N}\}$

Positions: $\{\text{pos}_1, \text{pos}_2\}$

Initial: $\{\text{pos}_1\}$

Differentiations:

$\text{pos}_1, v^{1,4,11} \rightarrow \text{pos}_2$

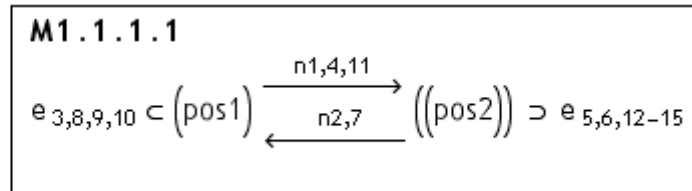
$\text{pos}_2, v^{2,7} \rightarrow \text{pos}_1$

$\text{pos}_1, \varepsilon^{8,9,10} \rightarrow \text{pos}_1$

$\text{pos}_2, \varepsilon^{3,5,6, 12-15} \rightarrow \text{pos}_2$

Final: $\{\text{pos}_2\}$.

M1.1.1.1 as a retrograde iteration of M1.1.1 is accepting : [abbbbb].



Automaton M2

Differences: $\Delta = \{v_i, \varepsilon_j, i, j \in \mathbb{N}\}$

Positions: $\{\text{pos}_1, \text{pos}_2\}$

Initial: $\{\text{pos}_1\}$

Acceptance: $\{\text{pos}_1\}$

Differentiations:

$\text{pos}_1, v_1 \rightarrow \text{pos}_2$

$\text{pos}_2, v_3 \rightarrow \text{pos}_1$

$\text{pos}_2, \varepsilon_2 \rightarrow \text{pos}_2$

Final: $\{\text{pos}_1\}$.

M2 is accepting : [aba].

Automaton M2

Differences: $\Delta = \{v_i, \varepsilon_j, i, j \in \mathbb{N}\}$

Positions: $\{\text{pos}_1, \text{pos}_2\}$

Initial: $\{\text{pos}_1\}$

Acceptance: $\{\text{pos}_1\}$

Differentiations:

$\text{pos}_1, v^{1,5} \rightarrow \text{pos}_2$

$\text{pos}_2, v_3 \rightarrow \text{pos}_1$

$\text{pos}_2, \varepsilon^{2,6} \rightarrow \text{pos}_2$

$\text{pos}_1, \varepsilon_4 \rightarrow \text{pos}_1$

Final: $\{\text{pos}_1\}$.

M2.1 is accepting : [abaa].

Automaton M3

Differentiations: $\Delta = \{v_i, \varepsilon_j, i, j \in \mathbb{N}\}$

Positions: $\{\text{pos}_1, \text{pos}_2, \text{pos}_3\}$

Initial: $\{\text{pos}_1\}$

Differentiations:

$\text{pos}_1, v_1 \rightarrow \text{pos}_2$

$\text{pos}_2, v_6 \rightarrow \text{pos}_3$

$\text{pos}_1, v_4 \rightarrow \text{pos}_3$

$\text{pos}_2, v_3 \rightarrow \text{pos}_1$

$\text{pos}_3, v_5 \rightarrow \text{pos}_2$
 $\text{pos}_2, v_3 \rightarrow \text{pos}_1$
 $\text{pos}_2, \epsilon_2 \rightarrow \text{pos}_2$
 Final: $\{\text{pos}_2\}$.

M3 is accepting : [abac].

Automaton M3.2

Differences: $\Delta = \{v_i, \epsilon_j, i, j \in \mathbb{N}\}$
 Positions: $\{\text{pos}_1, \text{pos}_2, \text{pos}_3\}$
 Initial: $\{\text{pos}_1\}$
 Differentiations:
 $\text{pos}_1, v_1^5 \rightarrow \text{pos}_2$
 $\text{pos}_2, v_2 \rightarrow \text{pos}_3$
 $\text{pos}_3, v_3 \rightarrow \text{pos}_2$
 $\text{pos}_2, v_4 \rightarrow \text{pos}_1$
 $\text{pos}_3, v_5 \rightarrow \text{pos}_2$
 $\text{pos}_2, v_6 \rightarrow \text{pos}_2$: final
 Final: $\{\text{pos}_2\}$.

M3 is accepting : [abcc].

Automaton M4

Differences: $\Delta = \{v_i, \epsilon_j, i, j \in \mathbb{N}\}$
 Positions: $\{\text{pos}_1, \text{pos}_2, \text{pos}_3, \text{pos}_4\}$
 Initial: $\{\text{pos}_1\}$
 Differentiations:
 $\text{pos}_1, v_1 \rightarrow \text{pos}_2$
 $\text{pos}_2, \epsilon_2 \rightarrow \text{pos}_2$
 $\text{pos}_2, v_3^{10} \rightarrow \text{pos}_1$: final
 $\text{pos}_1, v_4 \rightarrow \text{pos}_3$
 $\text{pos}_3, v_5^9 \rightarrow \text{pos}_2$
 $\text{pos}_3, v_7 \rightarrow \text{pos}_4$
 $\text{pos}_4, v_8 \rightarrow \text{pos}_3$
 Final: $\{\text{pos}_1\}$.

M4 is accepting : [abacd].

3.1.3. Machine tables for MorphoFSA

Machine table for M1

M1	v_1	v_2	v_3	ϵ_1	ϵ_2	ϵ_3
q_1	q_2	—	—	—	—	—
q_2	—	—	q_1	—	q_2	—

M1	v_1	v_3	ϵ_2
q_1	q_2	—	—
q_2	—	q_1	q_2

M1	v	ϵ
q_1	q_2	—
q_2	q_1	q_2

accepts [aba] M2

Machine table for M2.2

M1 .1	$v_{1,5}$	v_3	$\epsilon_{2,6}$	ϵ_4
q_1	q_2	—	—	—
q_2	—	q_1	q_2	q_1

accepts [abaa] M2.2

Machine table for M3

M3	v_1	v_3	v_4	v_5	v_6	ϵ_2
q_1	q_2	—	q_3	—	—	—
q_2	—	q_1	—	—	q_3	q_2
q_3	—	—	—	q_2	—	—

accepts [abac] M3.

M3 .1	v_1	v_3	$v_{4,7}$	$v_{5,8}$	$v_{6,9}$	ϵ_2	ϵ_{10}
q_1	q_2	—	q_3	—	—	—	—
q_2	—	q_1	—	—	q_3	q_2	—
q_3	—	—	—	q_2	—	—	q_3

accepts [abacc] M3.

Machine table for M4

M4	v_1	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	ϵ_2
q_1	q_2	—	q_3	—	—	—	—	—	—	—
q_2	—	q_1	—	—	q_3	—	—	—	q_1	q_2
q_3	—	—	—	q_5	—	q_4	—	q_2	—	—
q_4	—	—	—	—	—	—	q_3	—	—	—

accepts [abacd] M4.

Machine table for M4.1

M4	v_1	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	ϵ_2	ϵ_{15}
q_1	q_2	—	q_3	—	—	—	—	—	—	—	—
q_2	—	q_1	—	—	q_3	—	—	—	q_1	q_2	—
q_3	—	—	—	q_5	—	q_4	—	q_2	—	—	—
q_4	—	—	—	—	—	—	q_3	—	—	—	q_4

accepts [abacdd] M4.

3.1.4. Comparisons, abstractions and more machine tables

A more abstract interpretation of the machine diagram is achieved by separating the e/n-distinction from its position number.

Hence, a self-cycle e might be abstractly accepted as either positioned at pos1 or pos2 or both. Therefore, a constellation $\text{const} = (v_1, v_2, e_3)$, which is a standard constellation, has variations, like $\text{const2} = (v_1, e_3, v_2)$ and $\text{const3} = (e_3, v_1, v_2)$ in the common acceptance and start position {pos1}.

Hence, the machine M1 is realizing at once, first the corresponding FSA sequences of the FSA machine A on the alphabet {0, 1} and second, all the combinatorial possibilities too. That is the four sequences (101..., 010..., 001..., 110...). Without the separation of position and differentiation of differentiations, there are still two FSA sequences accepted at once: (100...) and (010...) by the corresponding MorphoFSA.

Iterations

(011) : (v1, v2, e3)

(0111) : (v1,4, v2, e3,5,6) = iter(v1,v2,e3)

(01111) : (v1,4, v2,7, e3,5,6,8,9,10) = iter²(v1,v2,e3)

(011111) : (v1,4, v2,7,11, e3,5,6,8-15) = iter³(v1,v2,e3)

MorphoFSA	pos1	pos2	pos3	sequence
constellation	□	□	□	□
const1	v1	v2	e3	011 100 : FSA
const2	v1	e3	v2	101 010 : FSA
const3	e3	v1	v2	001 110

$\Delta = \{v1, v2, e3\}$,

Positions = {pos1, pos2, pos3} with pos1 = (1,2), pos2 = (1,3), pos3 = (2,3)

FSA realization alphabet = {0,1},

Constellations = Positions x Δ .

The proper notation for the ENstructure of a morphogram is given by a list of triples.

Example

ENstructure[abac] = ((1,2,v), (1,3,e), (2,3,v), (1,4,v), (2,4,v), (3,4,v)).

Hence, the linearized enumeration of the ENstructure is

num(ENstructure) = ((1,2,-)₁, (1,3,-)₂, (2,3,-)₃, (1,4,-)₄, (2,4,-)₅, (3,4,-)₆).

The list for ENstructure[abac] might also be written as a table:

ENTable[abac]	1	2	3
1	v ₁	e ₂	v ₄
2	—	v ₃	v ₅
3	—	—	v ₆

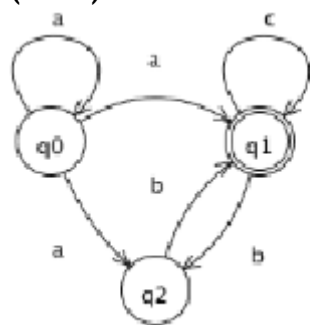
ENstructure table of [abac] =

ENstructure automaton table of [abac] :

MorphoFSA[abac]	pos1	pos2	pos3
pos1	pos ₁₁ , initial	pos ₁₂ , v ₁	pos ₁₃ , v ₄
pos2	pos ₂₁ , v ₃	pos ₂₂ , e ₂	pos ₂₃ , v ₅
pos3	—	pos ₃₂ , v ₅	—

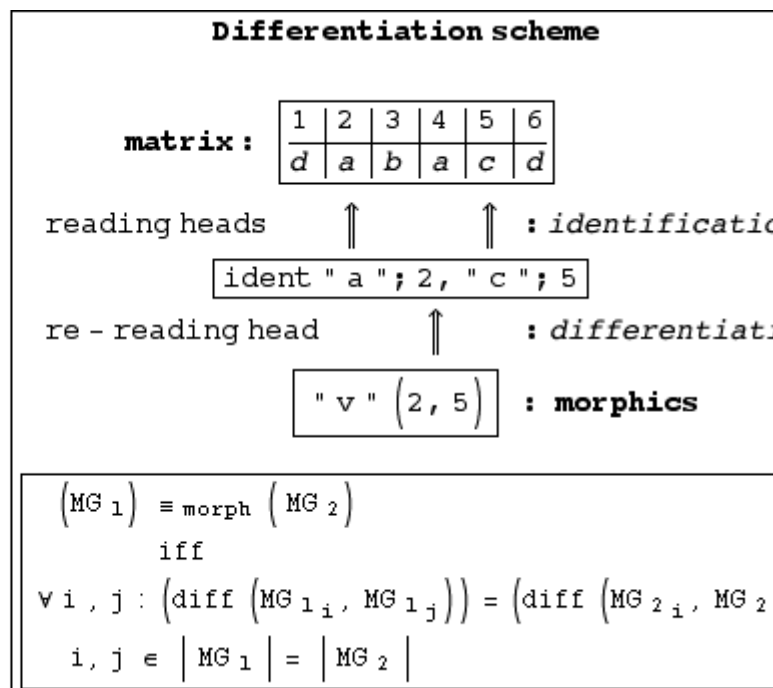
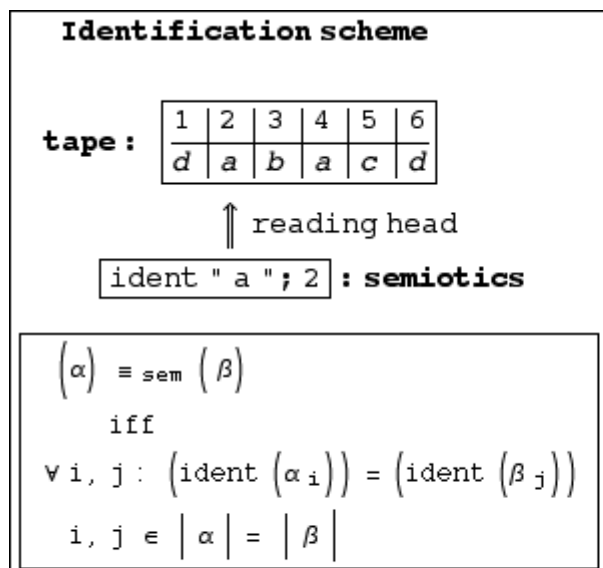
The direction of the differentiation (arrow) is marked by (pos_i x pos_j) and the label (value) of the arrow is written by an element from {e, v_j}, j ∈ s(m).

FSA(aabc)



FSA (aabc)	MorphoFSA[aabc]
q ₀ : aabc	init, pos1 : [aabc]
↓	↓
q ₀ : aabc	e1, pos1 : [aabc]
↓	↓
q ₁ : aabc	v2, pos2 : [aabc]
↓	↓
q ₂ : aabc	v3, pos1 : [aabc]
↓	↓
q ₂ : aabc.	v4, pos3 : [aabc]
	↓
	v5, pos1 : [aabc]
	↓
	v6, pos2 : [aabc] ⇒ [aabc].

Hence, the presented sequential order (e1,v2,v3,v4,v5,v6) is just one of the possible orders. It might serve as a standard order. Alternatively, another order might be: [aabc] => [aabc] -> [aabc] -> [aabc] -> [aabc] -> [aabc] => [aabc], with (v2,v4,v5,v3,v6,e1).



Without doubt, things are much more intriguing. The trick shall work nicely for a Beginners Guide.

Formal definition for M3

$M3 = (pos_i \times pos_i) \times \{e, v\}_j, i \in \{1, 2, 3\}, j \in \{1, 2, \dots, 6\}$.

This goes conform with directed labeled graphs.

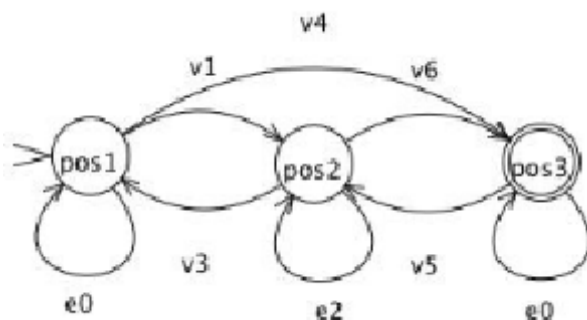
ENstructure automaton table of M3 [abac]

MorphoFSA[abac]	pos ₁	pos ₂	pos ₃
pos ₁	initial	v ₁	v ₄
pos ₂	v ₃	e ₂	v ₆
pos ₃	—	v ₅	—

Classical machine table for M3 [abac]

M3	v ₁	v ₃	v ₄	v ₅	v ₆	e ₂
q ₁	q ₂	—	q ₃	—	—	—
q ₂	—	q ₁	—	—	q ₃	q ₂
q ₃	—	—	—	q ₂	—	—

Diagram for M3



accepts [abac] M3.

The diagram of machine M3 is representing one and only one morphogrammatic structure, i.e. [abac].

The ENstructure of the morphogram [abac] is unambiguously determined by $ENstructure([abac]) = (v_1 e_2 v_3 v_4 v_5 v_6)$.

ENstructure table of M3.1 [abacc]

ENTable[abacc]	1	2	3	4
1	v_1	e_2	v_4	v_7
2	—	v_3	v_5	v_8
3	—	—	v_6	v_9
4	—	—	—	e_{10}

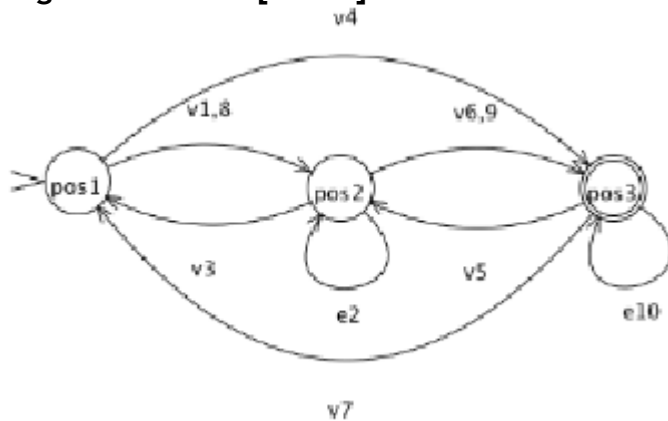
ENstructure automaton table of M3.1 [abacc]

MorphoFSA[abacc]	pos ₁	pos ₂	pos ₃
pos ₁	initial	$v_{1,8}$	v_4
pos ₂	v_3	e_2	$v_{6,9}$
pos ₃	v_7	v_5	e_{10}

Classical machine table for M3.1 [abacc]

M3.1	v_1	v_3	v_4	v_7	v_5	$v_{6,9}$	e_2	e_{10}
q_1	q_2	—	q_3	—	—	—	—	—
q_2	—	q_1	—	—	—	q_3	q_2	—
q_3	—	—	—	q_1	q_2	—	—	q_3

Diagram for M3.1 [abacc]



The diagram of machine M3.1 is representing one and only one morphogrammatic structure, i.e. [abacc].

The ENstructure of the morphogram [abacc] is unambiguously determined by $ENstructure([abacc])$.

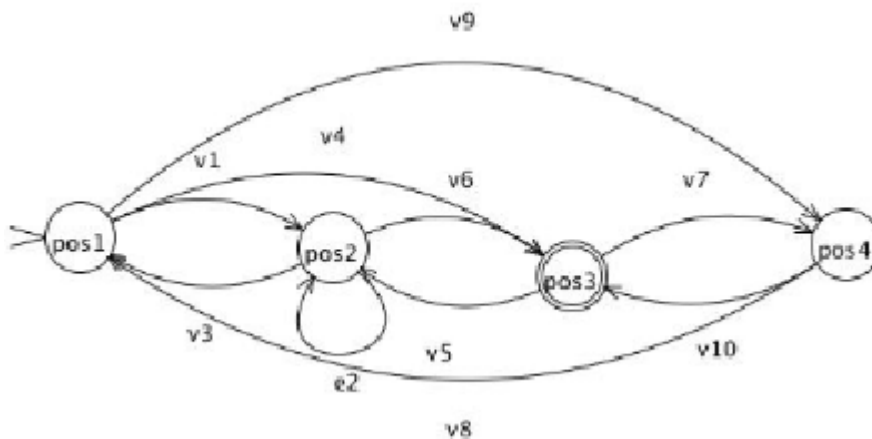
ENstructure automaton table of M4 [abacd]

MorphoFSA[abacd]	pos ₁	pos ₂	pos ₃	pos ₄
pos ₁	initial	v_1	v_4	v_9
pos ₂	v_3	e_2	v_6	—
pos ₃	—	v_5	—	v_7
pos ₄	v_8	—	v_{10}	—

The diagram of machine M4 is representing one and only one morphogrammatic structure, i.e. [abacd].

The ENstructure of the morphogram [abac] is unambiguously determined by
 ENstructure([abacd]) = (v1e2v3v7 v4v5v8 v6v9 v10).

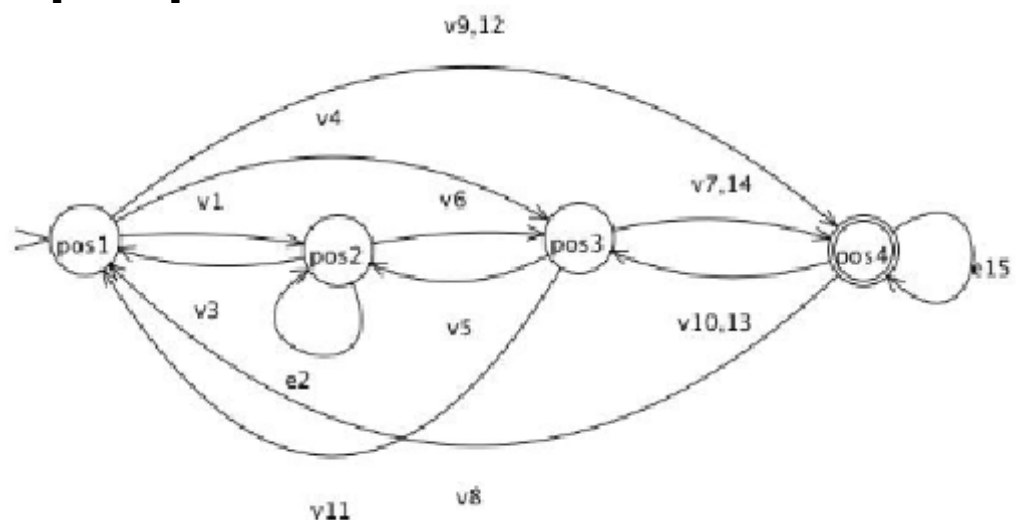
Diagram for M4 [abacd]



ENstructure automaton table of M4.1 [abacdd]

MorphoFSA[abacdd]	pos ₁	pos ₂	pos ₃	pos ₄
pos ₁	initial	v ₁	v ₄	v _{9,12}
pos ₂	v ₃	e ₂	v ₆	—
pos ₃	v ₁₁	v ₅	—	v _{7,14}
pos ₄	v ₈	—	v _{10,13}	e ₁₅

Diagram for M4.1 [abacdd]



Thanks to Jean Bovet for his simple and practical “Visual Automata Simulator”. In fact, the *static* aspect of the morphogrammatic automata is just giving a *visualisation* of the pattern aspect of morphograms as a data type.

<http://www.cs.usfca.edu/~jbovet/>

Further: jForlan, jFLAP and GOAL for Büchi automata.

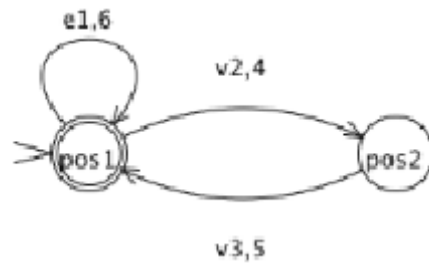
3.1.5. Some applications of MorphoFSMs onto morphogrammatics

Modeling the decomposition of morphograms into monomorphies

MorphoFSM for [aabb]

dec([aabb]) = {[aa], [bb]} = [aa].

The machine has two self-cycles at the acceptance pos1. This is indicating the monomorphy [aa], realized as the equivalence of the monomorphies [aa] and [bb] supported by the differentiations v2,4 and v3,5.



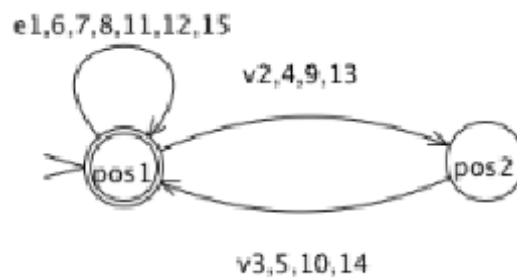
MorphoFSM[aabbbaa]

$\text{dec}([aabbbaa]) = [aa]$

The machine is repeating iteratively the monomorphy [aa] as [aa], [bb] and [aa] in the morphogram [aabbbaa].

Again, the iteration at the position pos1 is determined by the whole machine and not just by the cycles at pos1.

Therefore, the cycle e1 is producing [aa], the cycle e6 the cycle [bb] and the cycle e15 [aa]. The left self-cycles e7,8,11,12 are determining together with the differentiations v2,4,9,13 and v3,5,10,14 the positions of the monomorphies as the internal different monomorphies [aa] and [bb] of the morphogram [aabbbaa].



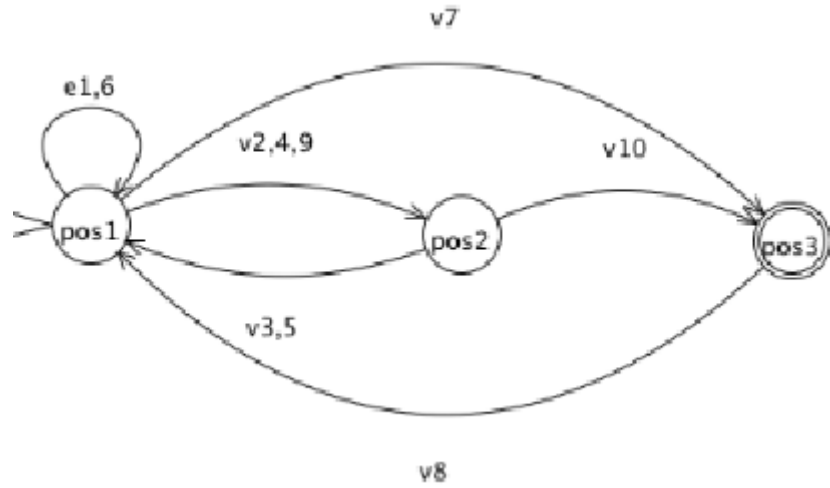
ENstructure automaton table for MorphoFSM [aabbbaa]

MorphoFSA[aabbbaa]	pos ₁	pos ₂
pos ₁	e _{1,6,7,8,11,12,15}	v _{2,4,9,13}
pos ₂	v _{3,5,10,14}	—

MorphoFSM for [aabbba]

$\text{dec}([aabbba]) = \{[aa], [bb], [c]\} = \{[aa], [a]\}$.

The history of [aabb] is remembered in [aabbba], hence the acceptance state of MorphoFSM for [aabb] at pos1 is saved by MorphoFSM [aabbba] with acceptance at pos3 as [c]. Hence, the machine is reflecting the two monomorphies [aa] and [bb] that are morphogrammatically “collapsing” at pos1 with e1,6 for MorphoFSM[aabb] additional to the acceptance at pos3 with [c] for the machine MorphoFSM[aabbba]. Therefore, the machine can be used to analyse the step-wise decomposition of the morphogram [aabbba] into its monomorphies [aa] and [a].



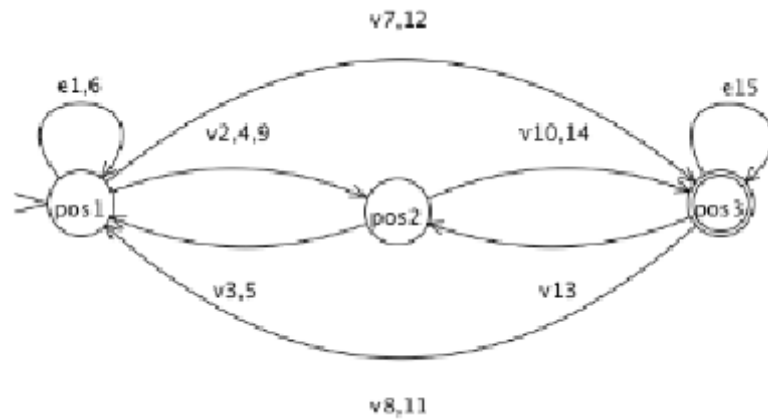
ENstructure automaton table for MorphoFSM [abacc]

MorphoFSA[aabbcc]	pos ₁	pos ₂	pos ₃
pos ₁	$e_{1,6}$	$v_{2,4,9}$	v_7
pos ₂	$v_{3,5}$	—	v_{10}
pos ₃	v_8	—	—

MorphoFSM for [aabbcc]

$\text{dec}([aabbcc]) = \{[aa], [bb], [cc]\} = [aa]$

The same holds for MorphoFSM[aabbcc]. The self-cycles at pos1 of MorphoFSM[aabb] are saved and additionally the monadic monomorphy [a] is extended to the monomorphy [aa] by the realization of the self-cycle at pos3 by [cc]. Hence, MorphoFSM[aabbcc] can be read as a realization of the decomposition of the morphogram [aabbcc] into its monomorphy [aa].



ENstructure automaton table for MorphoFSM [aabbcc]

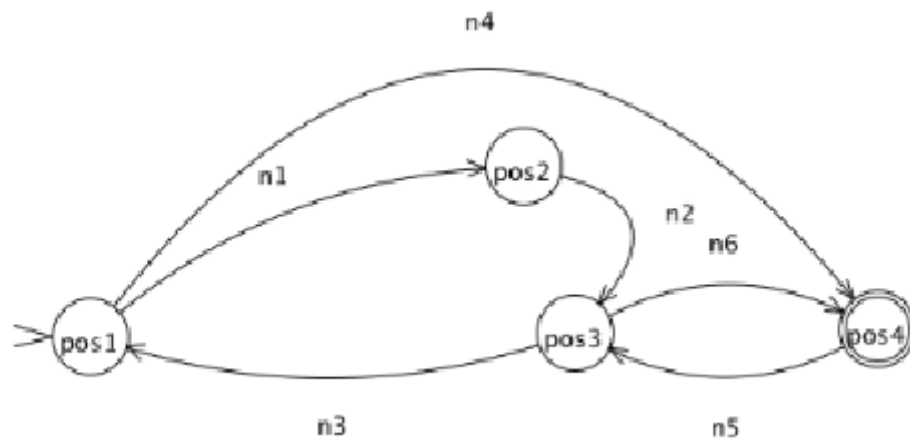
MorphoFSA[aabbcc]	pos ₁	pos ₂	pos ₃
pos ₁	$e_{1,6}$	$v_{2,4,9}$	$v_{7,12}$
pos ₂	$v_{3,5}$	—	$v_{10,14}$
pos ₃	$v_{8,11}$	v_{13}	e_{15}

MorphFSM[abcd]

$\text{dec}([abcd]) = \{[a], [b], [c], [d]\} = [a]$.

The monomorphy [a] is given by the start action on pos1, [b] is represented by the first difference for [ab] by v1, acceptance of MorphoFSM[abc] in pos1 by v3 is defining the monomorphy [c], while the acceptance of the machine for [abcd] at pos3 is delivering with v6 the monad [d]. All monomorphies in this configuration are monads

and are therefore morphogrammatically collapsing step-wise into the single monad [a].



ENstructure automaton tables for MorphoFSM [abc], [abcd], [abcde]

MorphoFSA[abc]	pos ₁	pos ₂	pos ₃
pos ₁	–	v₁	–
pos ₂	–	–	v₂
pos ₃	v ₃	–	–

MorphoFSA[abcd]	pos ₁	pos ₂	pos ₃	pos ₄
pos ₁	–	v₁	–	v ₄
pos ₂	–	–	v₂	–
pos ₃	v ₃	–	–	v₆
pos ₄	–	–	v ₅	–

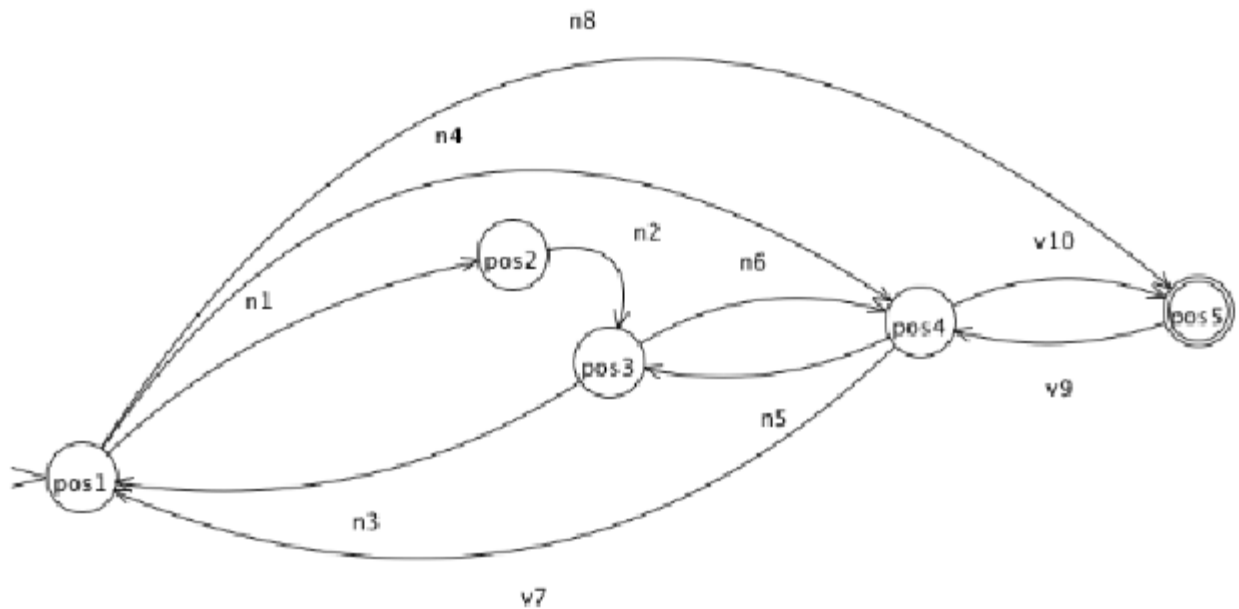
MorphoFSA[abcde]	pos ₁	pos ₂	pos ₃	pos ₄	pos ₅
pos ₁	–	v₁	–	v ₄	v ₈
pos ₂	–	–	v₂	–	–
pos ₃	v ₃	–	–	v₆	–
pos ₄	v ₇	–	v ₅	–	v₁₀
pos ₅	–	–	–	v ₉	–

MorphFSM[abcde]

dec([abcde]) = {[a], [b], [c], [d], [e]} = [a].

MorphoFSM[abcde] is a continuation of MorphoFSM[abcd] and shows, together with MorphoFSM[abc], how prolongations are working.

It doesn't make sense just to repeat, arbitrarily, say a loop like (v₉,v₁₀), as it is possible for FSAs.



MorphoFSA[aaaa]

$\text{dec}([aaaa]) = [aaaa]$.

Because there is no differentiation involved, the machine accepts the morphogram [aaaa] as such, i.e. as a monomorphy [aaaa]. Hence, the morphogram [aaaa] is decomposable only into itself. This corresponds exactly the rules of monomorphic decomposition of morphograms.

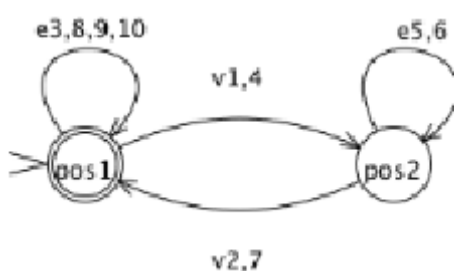


MorphoFSM[abbbb]

$\text{dec}([abbbb]) = \{[a], [aaaa]\}$.

The machine sets a difference with n1 to the start [a] with [b], 'confirms' the difference with n2, creates a first monomorphy [bb], repeats the differentiation and produces a second monomorphy at position pos2 with [bbb] and confirms this different monomorphy to position pos2 with v7, where the self-loops e8,9,10 are establishing the monomorhy [bbbb]. Thus, the machine makes a difference between the monad [a] and the monadic pattern [bbbb], hence the decomposition of the morphogram [abbbb] delivers [a] and [aaaa] written in trito-normal form, therefore $\text{dec}([abbbb]) = \{[a], [aaaa]\}$.

The diagram also contains the history of the decompositions for [ab] into {[a]}, [abb] into {[a], [aa]} and [abbb] into {[a], [aaa]}.



Monomorphic MorphoFSMs

Monomorphies might also be used as an abstraction over morphic automata. The features of *retrograde* recursivity are still covered by monomorphic interpretations of morphograms. In fact, the proposed paper is focused more on the *kenogrammatic* definition of morphograms than on the specific *monomorphic* understanding of morphograms and its consequences. The monomorphic abstraction *mon* of a morphogram, $\text{mon}([MG])$, is partitioning the morphogram into its monomorphies and is therefore delivering a slightly more abstract approach to MorphoFSMs.

Example1

$\text{mon}([aabcaa]) = ([aa], [b], [c], [aa])$,

$\text{mon}([aabcaa])$ is in fact dealt in the same way of *retrogradeness* like the reduced morphogram $[abca]$.

$\text{mon}([aabcaa]) = ([mg_1 = [aa], mg_2=[b], mg_3=[c], mg_4=[aa])$.

More interesting and specific properties of monomorphic MorphoFSMs are opened up with the operations of monomorphic *concatenation*, *coalition* and *cooperations* between morphic automata. In the context of *reductions*, the standard techniques of *deutero*- and *proto*-structures are at place.

<http://works.bepress.com/thinkartlab/41/>

<http://memristors.memristics.com/MorphoProgramming/Morphogrammatic%20Programming.pdf>

MorphoFSM($\text{mon}([aabcaa])$)

Differences: $\Delta = \{v_i, \varepsilon_j, i, j \in \mathbb{N}\}$

Positions: $\{\text{pos}_1, \text{pos}_2, \text{pos}_3\}$

Initial: $\{\text{pos}_1\}$

Differentiations:

$\text{pos}_1, \varepsilon^4 \rightarrow \text{pos}_1 : [mg_{1.1}] \rightarrow [mg_{1.4}]$

$\text{pos}_1, v^{1,5} \rightarrow \text{pos}_2 : [mg_{1.1}] \rightarrow [mg_2]$

$\text{pos}_2, v^{2,6} \rightarrow \text{pos}_3 : [mg_{1.1}] \rightarrow [mg_3]$

$\text{pos}_3, v_3 \rightarrow \text{pos}_1 : [mg_2] \rightarrow [mg_3]$

Final: $\{\text{pos}_1\}$.

MorphoFSM – *mon* is accepting : $[mg_1, mg_2, mg_3, mg_1]$.

MorphoFSM(<i>mon</i>)	pos ₁	pos ₂	pos ₃
pos ₁	ε_4	$v_{1,5}$	–
pos ₂	–	–	$v_{2,6}$
pos ₃	v_3	–	–

Reduction

$\text{path-length}(\text{MorphoFSM-}\text{mon}([aabcaa])) = 6$

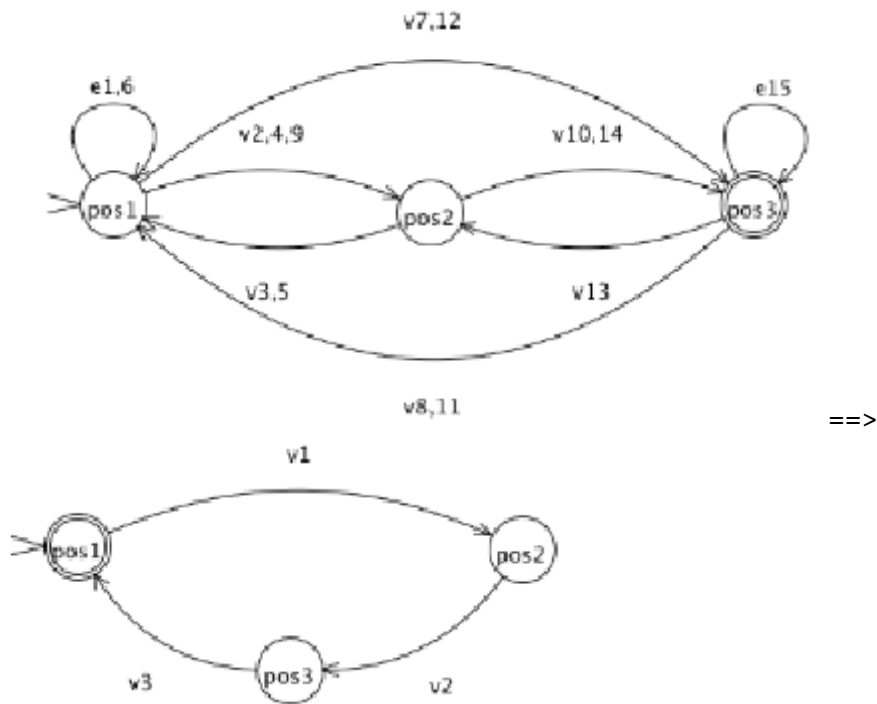
$\text{path-length}(\text{MorphoFSM}[aabcaa]) = 15$

Example2

Diagram MorphoFSM[aabbcc]

MorphoFSM[mg_1, mg_2, mg_3]

Diagram



Reduction

path-length(MorphoFSM-mon([aabbcc])) = 3

path-length(MorphoFSM[aabbcc]) = 15

Reduction of ENstructure automaton table for MorphoFSM [aabbcc] to MorphoFSM[mg_1 , mg_2 , mg_3]

MorphoFSA[aabbcc]	pos ₁	pos ₂	pos ₃
pos ₁	e _{1,6}	v _{2,4,9}	v _{7,12}
pos ₂	v _{3,5}	—	v _{10,14}
pos ₃	v _{8,11}	v ₁₃	e ₁₅

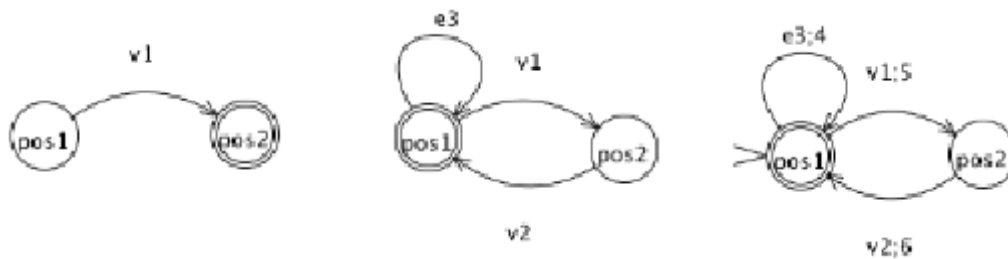
==>_{mon}

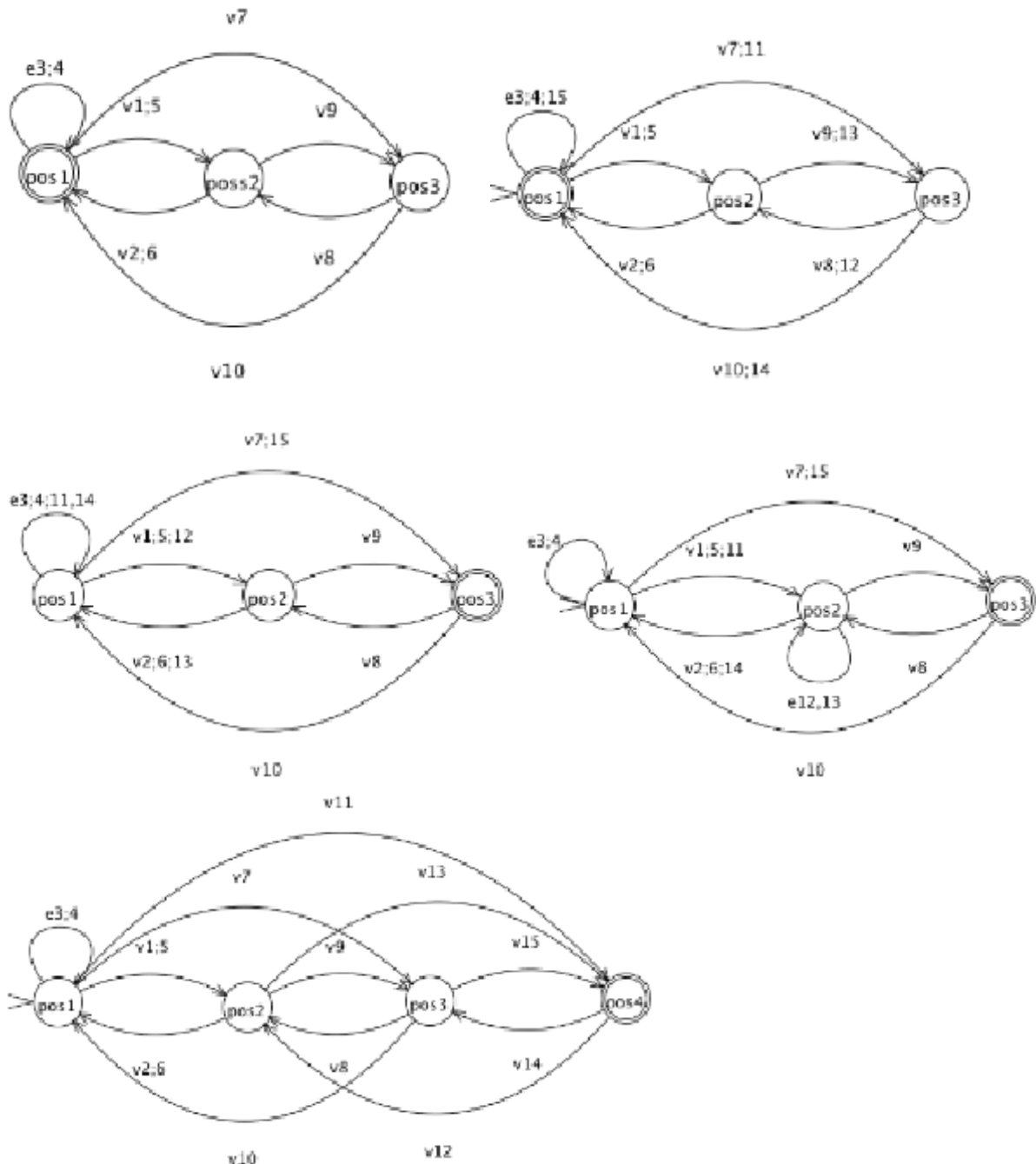
MorphoFSM[mg]	pos ₁	pos ₂	pos ₃
pos ₁	—	v ₁	—
pos ₂	—	—	v ₂
pos ₃	v ₃	—	—

MorphoFSM(mon([aabbcc])) = MorphoFSM[mg_1 , mg_2 , mg_3], with mg_1 = [aa], mg_2 = [bb] and mg_3 = [cc].

3.1.6. Towards a little Zoo of MorphoFSM diagrams

Developments: $[ab] \Rightarrow [abb] \Rightarrow [abba] \Rightarrow [abbac] \Rightarrow \begin{pmatrix} [abbaca] \\ [abbacb] \\ [abbacc] \\ [abbacd] \end{pmatrix}$





Iterations as alterations

What happens with the kenomic concept of iterability? What we learned elsewhere is: repeatability is iter/alteration.

Sind bei einer linearen Anordnung beziehungsweise einer Sukzession von Zeichen immer nur Vorgänger und Nachfolger eines Zeichens als unmittelbare Nachbarn bestimmbar, so ist bei einer kenogrammatischen Komplexion jedes Kenogramm mit jedem anderen im Verhältnis der unmittelbaren Nachbarschaft. Die Nachfolgerrelation eines Zeichens ist unabhängig von der Länge der ihm vorangehenden Zeichenreihe.

Dagegen ist die unmittelbare Nachbarschaft eines Kenogramms in einer Komplexion durch deren Komplexität bestimmt. Zeichenreihen können wegen ihrer Abstraktheit durch potentielle oder aktuelle Unendlichkeit bestimmt sein, kenogrammatische Komplexionen sind dagegen immer finit beziehungsweise ultra-finit. Ein Nachfolger einer kenogrammatischen Komplexion bestimmt sich retrograd-rekursiv durch die Materialität seiner Genesis. Diese definiert den Grad der simultanen Parallelität ihrer Nachfolger. Damit löst sich die Sprechweise der Dichotomie von Operator und

Operand der Nachfolgeroperation und ihrer Linearstruktur auf. Dual zur dichotomisierenden kann die Terminologie der Selbsterzeugung, der Autopoiese, von kenogrammatischen Komplexionen, etwa von Morphogrammen, eingebracht werden.” (Kaehr 1993)

[http://www.vordenker.de/heterarchy/b_heterarchy-](http://www.vordenker.de/heterarchy/b_heterarchy-e.pdf)

[e.pdf](http://www.vordenker.de/heterarchy/b_heterarchy-e.pdf)http://www.vordenker.de/heterarchy/b_heterarchy-e.pdf

Hence, the role of iterability in differentiation systems (machines) shall be taken into the focus for further applications or explications of morphic finite state machines.

With each iteration of a differentiation, the possibility of an alteration and accretion, independently of a pre-given alphabet, is determined retro-grade recursively by the path (trace, history) of the previous differentiations of the intrinsic activity of the machine.

With that, the whole machinery of possible disseminations of machines over a contextural grid as the system of loci for accretive iterations has to be applied again.

3.2. Palindromes and MorphoFSMs

3.2.1. Palindromes and iteration

"It is impossible to build a finite state machine that accepts all palindromes. The proof relies on the facts that we can easily build a string that requires an arbitrarily large number of nodes, namely the string

$a^x b a^x$ (eg., aba, aabaa, aaabaaa, aaaabaaaa,)

where a^x is a repeated x times. This requires at least x nodes because, after seeing the 'b' we have to *count back* x times to make sure it is a palindrome."

SUMMARY

Cannot recognize all types of patterns.

- E.g. Cannot build a finite-state machine that unlocks a lock whenever you enter any palindrome: 3-2-1-1-2-3

- Why? Palindromes can be of any length, and to recognize the 2nd half, you need to *remember* every character in the first half.

Because there are *infinitely* many possible first halves, this would require a machine with an infinite number of states.

<http://www.cs.mcgill.ca/~jpineau/comp102/Lectures/04FiniteStateMachines.pdf>

Hence, FSMs are not equipped to recognize palindromes of arbitrary length because they don't have a device that works as a *memory* to store the number of elements that have to be repeated after reading the first part of the sequence (string) and that has to be added after the recognition of the 'centre' of the word.

The machine MorphoFSM[aabbbaa] is accepting the morphic palindrome [aabbbaa]. Is there any chance to generalize this construction for *arbitrary* morphic 'palindromes'?

Again, *No memory on states or transitions so 'memory' is simply the location in the transition network*. Does this hold in the same sense for morphic machines too?

The answer is no!

Because morphograms are build retrograde-recursively over their differentiations, and not abstractly from a pre-given alphabet and stable rules like regular languages, they are inscribing their own *history* by definition. Hence, each "state" of a MorphoFSM is referring retrogradely to its previous "states", hence keeping an account of its history at the location of its last "state", i.e. differentiation.

The sequence of traces is called in the FSM literature the "*history*" of the activity of the machine. But this kind of history seems not to have a memory. A history, free of any memory, seems to be a quite weak metaphor to describe the behavior of a

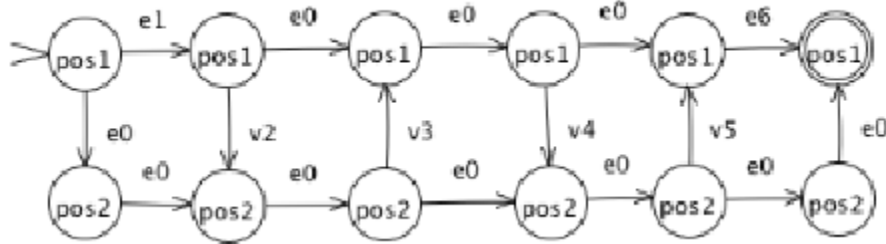
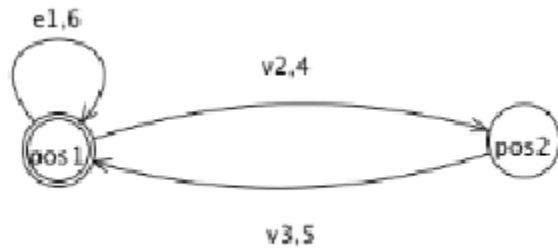
Therefore, it is not the case, that the memory capability of morphic machines is reduced to the simple “location in the network”. Is this giving a hint how to proceed?

Mutual iterations

Some more exercises.

Diagram for MorphoFSM[aabb]

MorphoFSA[aabb]	pos ₁	pos ₂
pos ₁	$e_{1,6}$	$v_{2,4}$
pos ₂	$v_{3,5}$	—



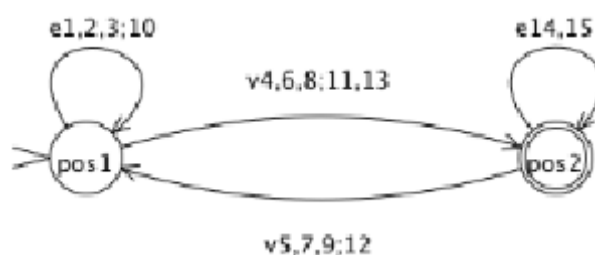
The monomorphism $[aa]$ and $[bb]$ are defined by the self-loops e_1 and e_6 respectively. Both monomorphisms that are carrying the result of the mutual iteration are separated by the differentiations of $v_{2,3}$ and $v_{4,5}$ respectively. This kind of iteration is faithful to the constellation of the differentiations of the repeated original: $[aba] = (e_1 v_{2,3} v_3)$ to the iteration $iter(e_1 v_{2,3} v_3) = (e_1^{1,6} v_{2,4} v_{3,5}) = [aabb]$.

This corresponds to the classical definition of a machine: the machine is not changing in the process of its use. Thus, the runs of $(e_1v_2v_3)$ and the runs of $iter(e_1v_2v_3)$ are strict iterations.

The second presentation with the nil-differentiation, e_0 , marks even more explicitly the positioning of the monomorphisms $[aa]$ as $[aa]$ and as $[bb]$. (cf. § 3.4.2)

For a slightly more complex situation, the definition is changing, and a kind of *super-additivity* appears. A further iteration of $\text{iter}(\text{iter}(e_1 v_2 v_3))$ is not running 6 plus 3 traces but 15.

Diagram for MorphoFSM[aaabbbb]

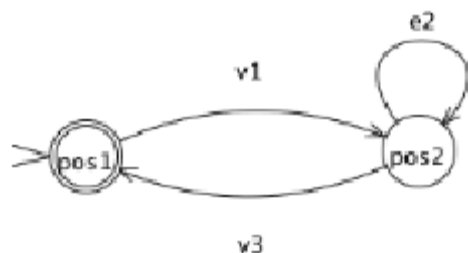


[aaabbb]	2	3	4	5	6
[]	e₁	e₂	v ₄	v ₇	v ₁₁
2	□	e₃	v ₅	v ₈	v ₁₂
3	□	□	v ₆	v ₉	v ₁₃
4	□	□	□	e₁₀	e₁₄
5	□	□	□	□	e₁₅

MorphoFSA[aaabbb]	pos ₁	pos ₂
pos ₁	e _{1,2,3;10}	v _{4,6,8;11,13}
pos ₂	v _{5,7,9;12}	e _{14,15}

3.2.2. Palindromes, first

Diagram for MorphoFSM[aba]

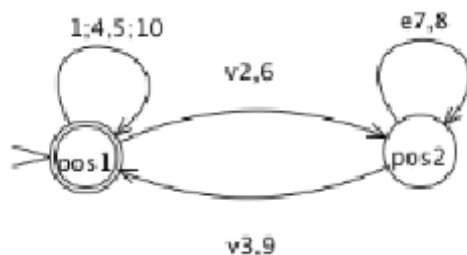


MorphoFSA[aba]	pos ₁	pos ₂
pos ₁	—	v ₁
pos ₂	v ₃	e ₂

Analysis of MorphoFSM[aba]

The machine MorphoFSM[aba] might be taken as a simple example of a morphic palindrome. With $w = [a]$, $w_r = [a]$ and the midpoint $c = [b]$, the machine shows the intricate mechanism of a mediation of the three parts of the morphic palindrome. For $v_1 = [a_1b_0]$ and for the inverse $v_2 = [b_0a_2]$, the midpoint is marked from both sides of the palindrome, i.e. $[b_0]$ is conceived as an end-point and as a start-point. The self-loop e_2 marks the exchange elements $[a_1]$ and $[a_2]$, i.e. $[a_1a_2]$, based on the separation by $[a_1b_0]$ and $[b_0a_1]$. This double-function of the midpoint is contradicting the logical identity of signs. Morphograms are covering this situation by definition. The classical interpretation has to move this contradictory situation into the mental domain of an external observer. In other words, the element “c” is (semiotically, logically and ontologically) the midpoint for semiotic FSM, while for morphic machines, “c” plays the double-role(s) as a midpoint.

Diagram for MorphoFSM[aabaa]



MorphoFSA [aabaab]	pos ₁	pos ₂
pos ₁	$e_{1; 4,5;10}$	$v_{2;6}$
pos ₂	$v_{3;9}$	$e_{7,8}$

Analysis of MorphoFSM [aabaab]

[a₁a₂b₀a₃a₄]:

$$e_1 = [a_1 a_2], e_{10} = [a_3 a_4]$$

$$e_1, v_2, v_3 = [a_1 a_2 b_0]$$

$$v_6, v_9, e_{10} = --- [b_0 a_3 a_4].$$

$$v_2 = [a_1 b_0]; v_3 = [a_2 b_0]$$

$$v_6 = [b_0 a_3]; v_9 = [b_0 a_4]$$

$$e_{4,5} = [a_1 a_4], [a_1, a_3]$$

$$e_{7,8} = [a_2 a_3], [a_2, a_4].$$

Palindrome formula for [aabaab]

$$\text{palindrome}[aabaab] = \frac{(e_1, v_2, v_3)(v_6, v_9, e_{10})}{(e_{4,5}); (e_{7,8})}$$

$$\text{with } (e_1, v_2, v_3) = \text{rev}(v_6, v_9, e_{10})$$

Palindrome scheme for [aabaab]

$$e_1, v_2, v_3 = [a_1 a_2 b_0]$$

$$v_6, v_9, e_{10} = --- [b_0 a_3 a_4]$$

$$e_1 = [a_1 a_2] -$$

$$e_{10} = - [a_3 a_4]$$

$$v_{2,3} = \text{end}[b_0]$$

$$v_{6,9} = [b_0]_{\text{start}}$$

$e_{4,5}; e_{7,8}$: mediating a 's

Palindromes in classical FSM

The classical analysis of palindromes in the context of FSM make much simpler

assumptions: $wc w_r$, with $c \cap w = \emptyset$ and $r(w_r) = w$.

In short, the *midpoint* “c” is in no way involved in the definition of w and w_r . It is atomistic, and its role as an ending and as a starting point of the midpoint is of no relevance. The midpoint is neutral to its environment w and w_r . This makes the construction simple. The consequences are as simple too: it remains in its simplicity.

Comparison

What the difference between the classical and the morphogrammatic understanding of palindromes?

Independent of the fundamental difference of the type of writing between semiotics and morphogrammatics, some feature of the difference are easily accessible with the example of palindromes.

For a semiotic understanding as it is leading for finite state machines, the distinction between the midpoint and the reversal parts has to be made by an external observer. There are no features, properties or guidelines implemented in the definition of the FSA to decide this difference or to indicate the midpoint of the palindrome. As a consequence of this ‘objectivistic’ definition of the FSM, the inherent limitations of a retrograde- and history-free machine, i.e. ‘memory-free’ conception follows naturally.

In contrast, the morphogrammatic definition of the ‘*finite differentiation machine*’ is based on an implementation of the features of the palindrome in itself. Hence, an internal observation or analysis makes it clear that the machines is detecting its ‘midpoint’ without the support of an external interaction. This is supported by the immanent retrograde understanding of iterability of the machine and its morphograms.

The complex description of the characteristics and behavior of the morphogrammatic machines is the result of the implementation of external description properties into the immanent definition of the machine itself.

Misleading wordings: *The new state depends on the old state and the input.*

Quick surface-reading of texts is mostly misleading. One example of such habits, following with quick judgments, is invoked by the wording of the *history-dependence* of the chain of events of classical FSMs. Most text-books about FSMs refer to the this interpretation of the chain of events of FSMs as history-dependence.

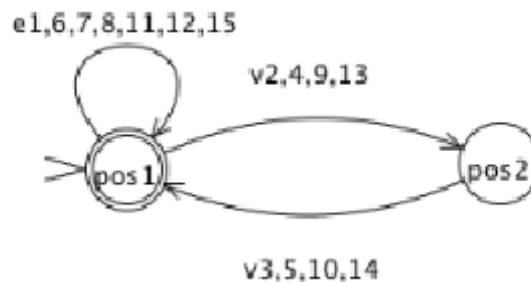
On the other hand I try to make clear with the concept and formalism of “*retrograde recursivity*” that there is no such concept of history-dependence implied on the level of formal languages, regular or not, and therefore also not for the behavior of FSMs. What happens is strictly history-independent. So called history-dependence is, in the best case, introduced as an *interpretation* of the behavior of FSMs by an external observer. And this interpretation has, again, no base in the definitions of the formalisms and mathematics of FSMs.

A nice example for this understanding of machine-history and its typical exaggerations is given by the citation of Mike James “*Finite State Machines*”:

“What this means that the entire history of the machine is summarized in its current state. All that matters is the state that it is in and not how it reached this state. Before you write off the finite state machine as so feeble as to be not worth considering as a model of computation it is worth pointing out that as you can have as many states as you care to invent the machine can record arbitrarily long histories. All you need is a state for each of the possible past histories and then the state that you find the machine in is an indication of not only its current state but how it arrived in that state.”

Because a finite state machine can represent any history and a reaction, by regarding the change of state as a response to the history, it has been argued that it is a sufficient model of human behavior, i.e. humans are finite state machines."
<http://www.i-programmer.info/babbages-bag/223-finite-state-machines.html>

Diagram for MorphoFSM[aabbaa]



ENstructure automaton table for MorphoFSM [aabbaa]

MorphoFSA[aabbaa]	pos ₁	pos ₂
pos ₁	e _{1,6,7,8,11,12,15}	v _{2,4,9,13}
pos ₂	v _{3,5,10,14}	—

ENstructure automaton table for MorphoFSM [aaabbaaa]

MorphoFSA[aaabbaaa]	pos ₁	pos ₂
pos ₁	e _{1-3;} 10-13; 16-18; 21-24; 27-28	v _{4;6;8.} 14;19;25.
pos ₂	v _{5;7;9;} 15;20;26.	—

3.2.3. Palindromes, again

ENstructure[aaabbaaa]

Procedure

Morphogram (sequence) ==> ENstructure (+ enumeration) ==> MorphoFSM table ==> MorphoFSM diagram

MorphoFSM table = (ENstructure, AG(Morphogram))

AG(Morphogram) = set of positions pos_i of MorphoFSM table

Example [aaabbaaa]

Morphogram [1,1,1,2,2,1,1,1] =_{MG} [aaabbaaa].

- ENstructure [1,1,1,2,2,1,1,1];

val it =

```

[[]],
[(1,2,E)],
[(1,3,E),(2,3,E)],
[(1,4,N),(2,4,N),(3,4,N)],
[(1,5,N),(2,5,N),(3,5,N),(4,5,E)],
[(1,6,E),(2,6,E),(3,6,E),(4,6,N),(5,6,N)],
[(1,7,E),(2,7,E),(3,7,E),(4,7,N),(5,7,N),(6,7,E)],
[(1,8,E),(2,8,E),(3,8,E),(4,8,N),(5,8,N),(6,8,E),(7,8,E)]]
: (int * int * EN) list list

```

ENstructure table with enumeration *subsystems*

Adjusted listing of *subsystems*

- subsystems 8;

val it =

```
[(1,[1,2]),
(3,[1,3]),(2,[2,3]),
(6,[1,4]), (5,[2,4]),(4,[3,4]),
(10,[1,5]),(9,[2,5]), (8,[3,5]),(7,[4,5]),
(15,[1,6]),(14,[2,6]),(13,[3,6]),(12,[4,6]),(11,[5,6]),
(21,[1,7]),(20,[2,7]),(19,[3,7]),(18,[4,7]),(17,[5,7]),(16,[6,7]),
(28,[1,8]),(27,[2,8]),(26,[3,8]),(25,[4,8]),(24,[5,8]),(23,[6,8]),(22,[7,8])]
: (int * int list) list
```

ENTable = subsystems ∪ ENstructure

[(1,3,E),(2,3,E)] ∪ (3,[1,3]),(2,[2,3]) = (3,[1,3], E),(2,[2,3], E).

(int * int * EN) list list ∪ (int * int list) list = (int * int list * EN) list list list

ENTable[aaabbaaa]	1	2	3	4	5	6	7	8
1	[]	e_1	e_2	v_4	v_7	e_{11}	e_{16}	e_{22}
2	□	□	e_3	v_5	v_8	e_{12}	e_{17}	e_{23}
3	□	□	□	v_6	v_9	e_{13}	e_{18}	e_{24}
4	□	□	□	□	e_{10}	v_{14}	v_{19}	v_{25}
5	□	□	□	□	□	v_{15}	v_{20}	v_{26}
6	□	□	□	□	□	□	e_{21}	e_{27}
7	□	□	□	□	□	□	□	e_{28}

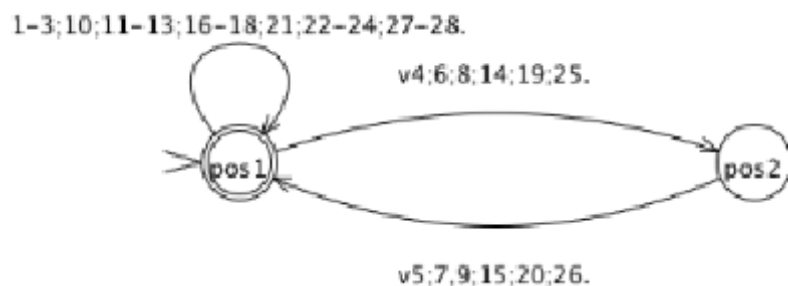
=

ENstructure automaton table for MorphoFSM [aaabbaaa]

AG([aaabbaaa]) = 2, table contains pos₁ and pos₂.

MorphoFSA[aaabbaaa]	pos ₁	pos ₂
pos ₁	$e_{1-3};$ 10-13; 16-18; 21-24; 27-28	$v_{4;6;8};$ 14;19;25.
pos ₂	$v_{5;7;9};$ 15;20;26.	—

Diagram for MorphoFSM [aaabbaaa]



Discussion

The first monomorphy [aaa] is covert by $e^{1,2,3}$.

To distinguish the monomorphy [bb] from the monomorphy [aaa], two differentiations have to be realized. This happens with $v_{4,6,8}$ and with $v_{5,7,9}$. On the background of this differentiation, the self-loop for [bb] is established with e_{10} .

With that, the machine recognizes the morphogram [aaabb].

The procedure to read the next monomorphy [aaa] happens step-wise.

First, the morphogram [aaabba] has to be recognized.

Second, the morphogram [aaabbaa] has to be recognized.

And finally, the morphogram [aaabbbaaa] with its second monomorphy [aaa] has to be read by the machine.

After this procedure, the machine MorphoFSM[aaabbbaaa] is able to read the morphic palindrom [aaabbbaa].

The monomorphy [bb] plays the role of the midpoint, c, of the palindrome.

Obviously, the second monomorphy [aaa], w_r , is the reverse of the first monomorphy [aaa], w. Hence, the morphogram is defined as a morphic palindrome, in analogy with wcw_r .

Generalization

It should now just be a question of simple combinatorics to give a general definition for a morphic machine MorphoFSM that is able to read all palindromes over the kenomic 'alphabet' {a,b}.

A next attempt would have to generalize this result for arbitrary complex morphic palindromes.

To any morphogrammatic palindrome there is a MorphoFSM that recognizes that palindrome.

Is there a MorphoFSM that accepts any palindromes?

Obviously, this seems to be in direct contrast to the results for the classical FSM definition of palindromes.

The argumentation for the impossibility of general FSAs for palindromes is based on the atomicity and 'history-free' concept of the automata, elaborated by a proof of contradiction.

Palindromes and Finite State Machines

"A regular expression can always be translated into an equivalent finite state machine.

"It is impossible to build a finite state machine that accepts all palindromes. The proof relies on the facts that we can easily build a string that requires an arbitrarily large number of nodes, namely the string

$a^x b a^x$ (eg., aba, aabaa, aaabaaa, aaaabaaaa,) where a^x is a repeated x times. This requires at least x nodes because, after seeing the 'b' we have to count back x times to make sure it is a palindrome."

<http://stackoverflow.com/questions/233243/how-to-check-that-a-string-is-a-palindrome-using-regular-expressions>

"A *palindrome* cannot be recognized by any finite state machine because

- (a) a finite state machine cannot remember arbitrary large amount of information
- (b) finite state machine cannot deterministically fix the mid-point
- (c) even if the mid-point is known, a finite state machine cannot find whether the second half of the string matches the first half.

(d) all of the above." R. Kumar, Theory of Automata.

"Palindromes with a *midpoint* indicator are strings of the form wcw_r where w_r is the reverse of substring w and c is the special midpoint marker."

<http://www.academic.marist.edu/~jzbv/algorithms/TuringMachine.htm>

"A *palindrome* cannot be recognized by any finite state machine ..." because the information about the palindrome is located at the position of an external observer and there is no way available to implement it internally into the system.

In contrast, the morphogrammatic approach is emphasizing the fact that an *internal* observer or agent has to deal with the concrete situation or constellation

that is encountered. An internal agent has no superior knowledge at hand - at least not in the actual situation. The internal agent acts according to the complexity and the properties of the concrete situation that is encountered. If the encountered constellation is a palindrome, it will be recognized as a palindrome thanks to the concrete properties of the encountered constellation. With that, no external knowledge has to interfere or to be applied.

In other words, the knowledge of the external observer is implemented into the intrinsic structure of the machine.

Categories and aspects of *reflectional observation theory* are not genuinely implemented for classical semiotics, string theory and automata theory. Morphogrammatics works within the interplay of internal and external observations.

Kaehr, Vom Selbst in der Selbstorganisation. Reflexionen zu den Problemen der Konzeptionalisierung und Formalisierung selbstbezüglicher Strukturbildungen, 1992
<http://www.thinkartlab.com/pkl/media/SelbstB2.frame.pdf>

⁵ENstructure EXAMPLES

3.2.4. Palindromes and Chiasms

"ABA is a palindrome: you can read it both ways, but it is not a chiasm. AB:BA is a chiasm, and so is of course AB:C:BA. Both are palindromes too, because they are dreadfully abstract."

The *chiasm* AB:C:BA reads from the viewpoint of difference-theory, i.e. kenogrammatics, in its abstractness as the complex morphogram [abcba]. A machine interpretation of the *palindrome* structure of the morphogram [abcba] is given with MorphoFSM[abcba]. A further modeling also has to contemplate on its specific *chiastic* structure that separates it from a simple palindrome.

To characterize palindromes and chiasms as "*dreadfully abstract*" is not referring to the topic itself but to the fundamental inability of understanding palindromes and chiasms as retrograde complexions of differentiations, differences and distinctions and not as abstract agglomerations of entities, like "A" and "B" or "C".

A less abstract thematization of chiasms and palindromes entertains at:
http://www.thinkartlab.com/Chinese%20Challenge%20Pool/How_to_Compose.pdf

Properties of palindromes and chiasms

"palindromes emerge as *multilayered*, *multidirectional*, and *polytemporal* mappings"
 Christina Ljungberg, 'Damn mad': Palindromic figurations in literary narratives.
 "running back again" (*palindromos*).

Morphogrammatic prolongations of the chiasm [AB:C:BA]

$$[AB:C:BA] \Rightarrow \left(\begin{array}{c} A [AB:C:BA] A \\ B [AB:C:BA] B \\ C [AB:C:BA] C \\ D [AB:C:BA] D \end{array} \right)$$

–ENstructure[1, 2, 3, 2, 1];

val it =

[[],

[(1, 2, N)],

[(1, 3, N), (2, 3, N)],

[(1, 4, N), (2, 4, E), (3, 4, N)],

[(1, 5, E), (2, 5, N), (3, 5, N), (4, 5, N)]:

(int * int * EN) list list

Prolongations of [1, 2, 3, 2, 1]: [aabcbaa]

– ENstructure[1, 1, 2, 3, 2, 1, 1];

val it =

[[],

[(1, 2, E)],

[(1, 3, N), (2, 3, N)],

[(1, 4, N), (2, 4, N), (3, 4, N)],

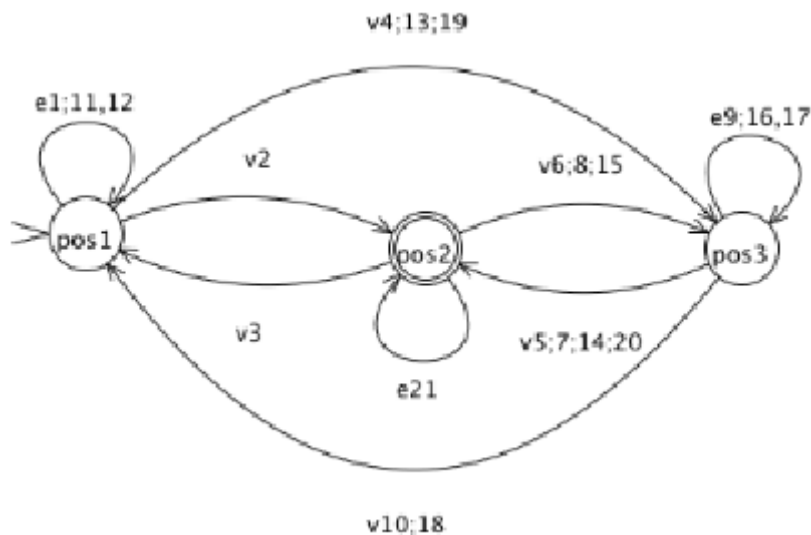
[(1, 5, N), (2, 5, N), (3, 5, E), (4, 5, N)],

[(1, 6, E), (2, 6, E), (3, 6, N), (4, 6, N), (5, 6, N)],

[(1, 7, E), (2, 7, E), (3, 7, N), (4, 7, N), (5, 7, N), (6, 7, E)]] :

(int * int * EN) list list

MorphoFSA [aabcbaa]	pos 1	pos 2	pos 3
pos 1	e 1;11,12	v 2	v 4;13;19
pos 2	v 3	e 21	v 2;6;15
pos 3	v 3;6;10;18	v 7;14;20	e 9;16,17



– ENstructure[2, 1, 2, 3, 2, 1, 2];

val it =

[[],

[(1, 2, N)],

[(1, 3, E), (2, 3, N)],

[(1, 4, N), (2, 4, N), (3, 4, N)],

[(1, 5, E), (2, 5, N), (3, 5, E), (4, 5, N)],

[(1, 6, N), (2, 6, E), (3, 6, N), (4, 6, N), (5, 6, N)],

[(1, 7, E), (2, 7, N), (3, 7, E), (4, 7, N), (5, 7, E), (6, 7, N)]] :

(int * int * EN) list list

tnf[2, 1, 2, 3, 2, 1, 2] = [1, 2, 1, 3, 1, 2, 1] = [abacaba]

– **ENstructure[3, 1, 2, 3, 2, 1, 3];**

val it =

[[],

[(1, 2, N)],

[(1, 3, N), (2, 3, N)],

[(1, 4, E), (2, 4, N), (3, 4, N)],

[(1, 5, N), (2, 5, N), (3, 5, E), (4, 5, N)],

[(1, 6, N), (2, 6, E), (3, 6, N), (4, 6, N), (5, 6, N)],

[(1, 7, E), (2, 7, N), (3, 7, N), (4, 7, E), (5, 7, N), (6, 7, N)]] :

(int * int * EN) list list

– **ENstructure[4, 1, 2, 3, 2, 1, 4];**

val it =

[[],

[(1, 2, N)],

[(1, 3, N), (2, 3, N)],

[(1, 4, N), (2, 4, N), (3, 4, N)],

[(1, 5, N), (2, 5, N), (3, 5, E), (4, 5, N)],

[(1, 6, N), (2, 6, E), (3, 6, N), (4, 6, N), (5, 6, N)],

[(1, 7, E), (2, 7, N), (3, 7, N), (4, 7, N), (5, 7, N), (6, 7, N)]] :

(int * int * EN) list list

tnf[4, 1, 2, 3, 2, 1, 4] = [1, 2, 3, 4, 3, 2, 1].

Accretion of the midterm

– **ENstructure[3, 2, 1, 4, 1, 2, 3];**

val it =

[[],

[(1, 2, N)],

[(1, 3, N), (2, 3, N)],

[(1, 4, N), (2, 4, N), (3, 4, N)],

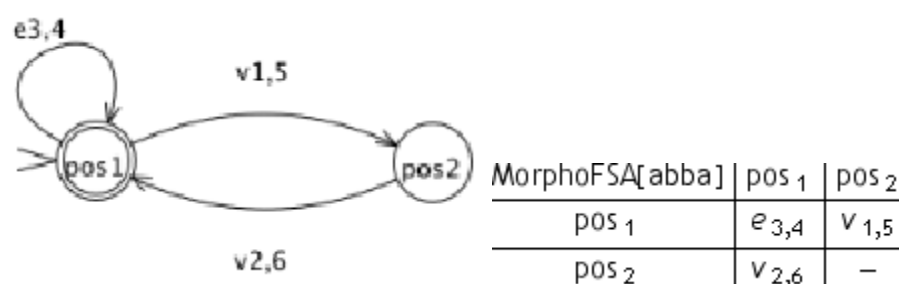
[(1, 5, N), (2, 5, N), (3, 5, E), (4, 5, N)],

[(1, 6, N), (2, 6, E), (3, 6, N), (4, 6, N), (5, 6, N)],

[(1, 7, E), (2, 7, N), (3, 7, N), (4, 7, N), (5, 7, N), (6, 7, N)]] :

(int * int * EN) list list

Diagram of MorphoFSM[abba]



Analysis of MorphoFSM[abba]

$$\begin{aligned}
 & \mathbf{[a_1 b_2 b_3 a_4]}: \\
 & v_1 = [a_1 b_2]; v_2 = [a_1 b_3], e_3 = [b_2 b_3] \\
 & v_5 = [b_2 a_4]; v_6 = [b_3 a_4], e_4 = [a_1 a_4] \\
 & v_1 = [a_1 b_2] \quad v_5 = [b_2 a_4] \\
 & v_2 = \text{---} [b_3 a_4] \quad v_6 = \text{--} [b_3 a_4]
 \end{aligned}$$

$$\text{chiasm}[\text{abba}] = \begin{pmatrix} a_1 \rightarrow b_2 \\ \updownarrow \quad \times \quad \updownarrow \\ a_4 \leftarrow b_3 \end{pmatrix}$$

Logical analysis of [abba]

$\text{pal}(p, q)$	(p, r)	(p, l)	(r, q)	(l, q)	(r)	(l)
a	a	$-$	a	$-$	a	$-$
b	b	$-$	$-$	b	$-$	b
b	$-$	b	$-$	b	b	$-$
a	$-$	a	a	$-$	$-$	a

Grammars for palindromes

Semiotic grammar for palindromes

The morphic pendant to the regular semiotic language for "non-empty odd length palindromic strings of as and/or bs and/or cs" (Parkes, p. 101), reflecting its retrograde iterability, is given by the morphogrammatically modified rules for palindromes.

Semiotic palindrome:

Alphabet = {a, b, c}

Rules = $S \Rightarrow a \mid b \mid c \mid aSa \mid bSb \mid cSc$.

Application: $S \Rightarrow aSa \Rightarrow c(aSa)c \Rightarrow b(caSac)b \Rightarrow a(bcaSacb)a \Rightarrow abca b acba$.

$S \Rightarrow a \Rightarrow bab \Rightarrow ababa \Rightarrow cab a bac \Rightarrow acab a baca$.

Grammar for a palindrome

Alph = {a, b, c}

Rules = $S \Rightarrow a \mid b \mid c \mid aSa \mid bSb \mid cSc$

Morphic Grammar for palindromes

Morphic alphabet = {[a]}

Morphic rules = $S_{\text{iter}}, \text{accr} \Rightarrow [a] \mid [a]S[a]$ with $S_{\text{iter}} \in \text{AG}(\text{MG})$, $S_{\text{accr}} \in \text{AG}(\text{MG})+1$.

The rule for the morphic S is depending on the string, i.e. morphogram "MG", already produced by the rule R for S. The rule $[a]S[a]$ is accepted as a start rule, written in trito-normal form, tnf, hence with $[a] \equiv [a]$, thus $[a]S[a]$. The aggregation, AG, detects the number of different kenograms of the MG, and the accretive iteration of S, S_{accr} , is adding an additional kenogram to the repertoire, while the iterative S, S_{iter} is using the detected kenograms of the morphogram MG. The kenograms of the 'alphabet' are produced retrogradely by the application of the rules, and are therefor not pre-given as elements of a sign repertoire, i.e. an alphabet. The rules for S are substitution

rules hence all the kenogrammatic considerations about different types of iteration, accretion and equivalences apply.

Application :

$$S \Rightarrow_{\text{iter}} [a] S [a] \Rightarrow_{\text{accr}} [b] [a] S [a] [b] \Rightarrow_{\text{iter}} [a] [b] [a] S [a] [b] [a] \Rightarrow_{\text{accr}} [a] [b] [a] [c] [a] [b] [a].$$

Short :

Production of an odd palindrom, with " c " as the produced midpoint in tnf – normal form :

$$S \Rightarrow_{\text{iter}} a S a \Rightarrow_{\text{accr}} b a S a b \Rightarrow_{\text{iter}} a b a S a b a \Rightarrow_{\text{accr}} a b a c a b a.$$

Palindrome morphic grammar

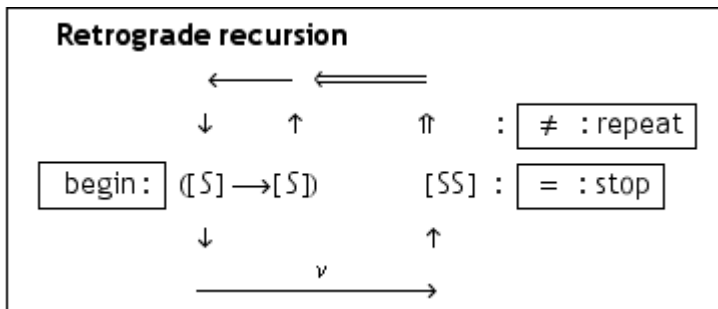
$$\text{Alph} = \left\{ \left[a \right] \right\}$$

Rules: $S \Rightarrow a \mid a S a$

In *general*, $S \Rightarrow [a]S \mid S[a] \mid [a]S[b] \mid [S][S]$ are defined retrogradely over their 'history' of traces or runs.

$S \Rightarrow [S][S]$ is retrogradely defined as the morphic concatenation of $[S]$ and $[S]$: $[S][S] \Rightarrow [SS]$.

The procedure is well known as the morphogrammatic retrograde recursion scheme:



<http://memristors.memristics.com/MorphoReflection/Morphogrammatics%20of%20Reflection.html>

Example

Full development of the first 5 steps for the morphic palindrome grammar ($S \Rightarrow a \mid aSa$) with exclusive midpoint.

$$S \Rightarrow aSa \Rightarrow \begin{pmatrix} a(aSa) & a \\ b(aSa) & b \end{pmatrix} \Longrightarrow$$

$$\begin{pmatrix} a(aaSaa) a \\ b(aaSaa) b \\ a(baSab) a \\ b(baSab) b \\ c(baSab) c \end{pmatrix} \Rightarrow \begin{pmatrix} a(aaaSaaa) a \\ b(aaaSaaa) b \\ a(baaSaab) a \\ b(baaSaab) b \\ a(abaSaba) a \\ b(abaSaba) b \\ c(abaSaba) c \\ a(bbaSabb) a \\ b(bbaSabb) b \\ c(bbaSabb) c \\ a(cbaSabc) a \\ b(cbaSabc) b \\ c(cbaSabc) c \\ d(cbaSabc) d \end{pmatrix} \Rightarrow \begin{pmatrix} aaaa b aaaa a \\ baaa c aaa b \\ abaa c aab a \\ bbaa c aab b \\ aaba c aba a \\ baba c aba b \\ caba d aba c \\ abba c abb a \\ bbaa c abb b \\ cbaa d abb c \\ aaba d abc a \\ bbaa d abc b \\ cbaa d abc c \\ d cba e abc d \end{pmatrix} \Rightarrow \begin{pmatrix} aaaa b aaaa \\ abbb c bbb a \\ abaa c aab a \\ aa bb c bb aa \\ aaba c aba a \\ abab c baba \\ abcb d bcb a \\ abba c abba \\ aaab c baaa \\ abbc d cbb a \\ abca d acba \\ abac d caba \\ aabc d cbaa \\ abcd e dcba \end{pmatrix}$$

Hence, the domain of the morphic grammar for “ $S \Rightarrow a|aSa$ ” with an exclusive midpoint and a 5-step repetition covers palindromes from [aaaa b aaaa] to [abcd e dcba]. From the viewpoint of morphogrammatcs, this development is complete. There are no additional palindromes of the same length covered by this grammar.

Unfortunately, this approach is not taking into account the full range of the specific features of morphogrammatic context-dependence of the production of palindromes.

Again, the classical definition is obviously strictly context-independent: “*the k th leftmost symbol of the input x must be equal to the k th rightmost symbol of x .*” (Ding-Zoo Du, p. 92)

The proposed symmetric approach is considering just a *partial* aspect of morphogrammatic context-dependence, i.e., it follows the symbolic definition but replaces the ‘equality’ of the symbols by the *equivalence* of the kenograms of the morphogram. This is combined by the iterative and accretive prolongations of the application of the production rule “ $S \Rightarrow a|aSa$ ”. It delivers symmetric morphogrammatic palindromes.

A more appropriate approach seems to cover the full mechanism of morphogrammatic context-dependence for the production of morphic palindromes.

A first step was covered by the morphic application of the symmetric production rule, combined with iterative and accretive prolongations.

A second step has to compare the two parts as wholes of the palindrome according to the rules of morphogrammatic equivalence.

Hence, a case like “[1,2,2,3,3,4]” is considering the morphogrammatic equivalence of the first and the reversed second part of the morphogram: $[1,2,2] =_{mg} [4,3,3]$, thus “[1,2,2,3,3,4]” is a palindrome. That is, $[1,2,2,3,3,4] =_{mg} [4,3,3,2,2,1]$, with $tnf[4,3,3,2,2,1] =_{mg} [1,2,2,3,3,4]$.

3.2.5. Appendix about asymmetric palindromes

Comparison with the filtered results

Palindromes are filtered out from the trito-universe TU. The length of the words is 6 and the range spans from [1,1,1,1,1,1] to the saturated morphogram [1,2,3,4,5,6].

The symmetric production rule “ $S \Rightarrow a|aSa$ ” is not considering

the *asymmetric* productions that are morphogrammatically accepted as palindromes, like for example the morphogram [1,2,3,4,1,2] with $[1,2,3] =_{mg} [2,1,4]$, $[1,2,3,4,1,2]$

$=_{mg} [2,1,4,3,2,1]$, and $tnf[2,1,4,3,2,1] =_{mg} [1,2,3,4,1,2]$. Hence, the *context*-dependence of the morphic production rule is restricted to symmetric productions with restricted context-dependence.

The filter-method is not producing constructively the set of palindromes but is filtering them out of the produced trito-universe TU of morphograms.

Morphogrammatic palindrome:

```
fun kref ks = tnf(rev ks);  
- fun ispalindrome l = (l = kref l);  
val ispalindrome = fn : int list -> bool  
- ispalindrome [1,1,2,2];  
val it = true : bool
```

Symbolic palindrome

```
fun palindrome l = (l = rev l);  
- palindrome [1,1,2,2];  
val it = false : bool
```

Tests for morphogrammatic palindromes

Examples

```
- ispalindrome [1,2,2,2,3,3,4];  
val it = true : bool  
- ispalindrome [1,2,3,1,4,3];  
val it = true : bool  
- tnf [1,2,3,1,4,3];  
val it = [1,2,3,1,4,3] : int list  
- tnf [3,4,1,3,2,1];  
val it = [1,2,3,1,4,3] : int list  
- kref [1,2,3,1,4,3];  
val it = [1,2,3,1,4,3] : int list  
- rev [1,2,3,1,4,3];  
val it = [3,4,1,3,2,1] : int list  
- tnf(rev [1,2,3,1,4,3]);  
val it = [1,2,3,1,4,3] : int list
```

Filtered results of length 6 from TU

`nfirstq(5000, TU)`

`List.filter ispalindrome “nfirstq(5000, TU)”;` ⁶

`- length it;`

`val it = 180 : int`

Results for the 31 morphogrammatic palindromes of length 6 from [1,1,1,1,1,1] to [1,2,3,4,5,6]:

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

`Palindromes(6,6) = 31`

`Symmetric palindromes(6,6) = 5`

Palindromes of length 6 produced by the *symmetric* production rule “S ==> a|aSa”:

$$\begin{pmatrix} a(aaSaa) a \\ b(aaSaa) b \\ a(baSab) a \\ b(baSab) b \\ c(baSab) c \end{pmatrix} =_{\text{num}} \begin{pmatrix} 1(11S11) 1 \\ 2(11S11) 2 \\ 1(21S12) 1 \\ 2(21S12) 2 \\ 3(21S12) 3 \end{pmatrix} =_{\text{tnf}} \begin{pmatrix} 1(11S11) 1 \\ 1(22S22) 1 \\ 1(21S12) 1 \\ 1(12S21) 1 \\ 1(23S32) 1 \end{pmatrix}$$

Symmetric morphogrammatic palindromes of length 6 filtered out of TU:

```
val it =
  [[1,1,1,1,1,1],[1,1,2,2,1,1],[1,2,1,1,2,1],[1,2,2,2,2,1],[1,2,3,3,2,1]] : int list list
- length it;
val it = 5 : int
```

Symmetric morphogrammatic palindromes of length 7 filtered out of TU:

```
val it =
  [[1,1,1,1,1,1,1],[1,1,1,2,1,1,1],[1,1,2,1,2,1,1],[1,1,2,2,2,1,1],
   [1,1,2,3,2,1,1],[1,2,1,1,1,2,1],[1,2,1,2,1,2,1],[1,2,1,3,1,2,1],
   [1,2,2,1,2,2,1],[1,2,2,2,2,2,1],[1,2,2,3,2,2,1],[1,2,3,1,3,2,1],
   [1,2,3,2,3,2,1],[1,2,3,3,3,2,1],[1,2,3,4,3,2,1]] : int list list
- length it;
val it = 15 : int
Symmetric palindromes(7,7) = 15
Palindromes(7,7) = 59
```

Palindromes of kmul[1,2,3][1,2,3];

```
- ispalindrome [1,2,3];
val it = true : bool
```

List.filter ispalindrome “kmul[1,2,3][1,2,3];”;

```
val it =
  [[1,2,3,2,3,1,3,1,2],[1,2,3,3,1,2,2,3,1],[1,2,3,2,1,4,3,4,1],
   [1,2,3,2,1,4,5,4,1],[1,2,3,2,4,1,3,1,2],[1,2,3,2,4,1,5,1,2],
   [1,2,3,4,1,2,3,4,1],[1,2,3,4,1,2,5,4,1],[1,2,3,3,1,4,4,5,1],
   [1,2,3,4,3,1,3,5,4],[1,2,3,4,1,5,2,3,1],[1,2,3,4,1,5,3,6,1],
   [1,2,3,4,1,5,6,7,1],[1,2,3,4,5,1,2,3,4],[1,2,3,4,5,1,3,6,4],
   [1,2,3,4,5,1,6,7,4],[1,2,3,2,3,4,3,4,1],[1,2,3,2,3,4,3,4,5],
   [1,2,3,3,4,2,2,3,1],[1,2,3,3,4,2,2,3,5],[1,2,3,4,3,2,3,4,1],
   [1,2,3,4,3,2,3,4,5],[1,2,3,2,4,5,3,5,1],[1,2,3,2,4,5,6,5,1],
   [1,2,3,2,4,5,3,5,6],[1,2,3,2,4,5,6,5,7],[1,2,3,4,5,2,3,4,1],
   [1,2,3,4,5,2,6,4,1],[1,2,3,4,5,2,3,4,6],[1,2,3,4,5,2,6,4,7],
   [1,2,3,3,4,5,5,1,2],[1,2,3,3,4,5,5,6,1],[1,2,3,3,4,5,5,6,7],
   [1,2,3,4,3,5,3,1,2],[1,2,3,4,3,5,3,6,1],[1,2,3,4,3,5,3,6,7],
   [1,2,3,4,5,6,2,3,1],[1,2,3,4,5,6,3,1,2],[1,2,3,4,5,6,7,1,2],
   [1,2,3,4,5,6,3,7,1],[1,2,3,4,5,6,7,8,1],[1,2,3,4,5,6,2,3,7],
   [1,2,3,4,5,6,3,7,8],[1,2,3,4,5,6,7,8,9]] : int list list
```

```
- length it;
val it = 44 : int
- length(kmul[1,2,3][1,2,3]);
val it = 588 : int
- ispalindrome [1,2,3,4,3,5,3,6,7];
val it = true : bool
```

Symmetric palindrome for “kmul[1,2,3] [1,2,3]” = 0.

```
val it = [] : int list list
- palindrome [1,2,3,4,3,5,3,6,7];
val it = false : bool
```

- kconcat [1,2,3][1,2,3];

```
- length(kconcat [1,2,3][1,2,3]);
```

```
val it = 34 : int
```

Palindromes:

```
val it =
```

```

[[1,2,3,1,2,3],[1,2,3,2,3,1],[1,2,3,3,1,2],[1,2,3,3,2,1],[1,2,3,4,1,2],
[1,2,3,4,2,1],[1,2,3,1,4,3],[1,2,3,3,4,1],[1,2,3,4,5,1],[1,2,3,2,3,4],
[1,2,3,3,2,4],[1,2,3,4,2,5],[1,2,3,3,4,5],[1,2,3,4,5,6]] : int list list
- length it;
val it = 14 : int
- ispalindrome [1,2,3,4,5,1];
val it = true : bool
Symmetric palindromes:
val it = [[1,2,3,3,2,1]] : int list list
- palindrome [1,2,3,4,5,1];
val it = false : bool

```

Linguistic example for asymmetric palindromes

"Annabelle"

"anna" : num(anna) = [1,7,7,1]

" b" : num(b) = [2]

"elle" : num(elle) = [4,5,5,4]

num(annabelle) = [1,7,7,1,2,4,5,5,4]

- tnf[1,7,7,1,2,4,5,5,4];

val it = [1,2,2,1,3,4,5,5,4] : int list

ispalindrome[1,2,2,1,3,4,5,5,4]?

val it = true : bool

- kref[4,5,5,4,3,1,2,2,1];

val it = [1,2,2,1,3,4,5,5,4] : int list

Asymmetric palindromes in the morphoSphere

The *pheno*-structure of palindromes is symmetric. That's part of the definition. Like "anna", "b", and "elle" as identitive words of the pheno-structure.

The *geno*-structure is overwhelmingly dominated by asymmetric palindromes.

The genotype word "annabelle" is composed from pheno-type words "anna", "b" and "elle", which are symmetric. But the composition of the parts to the word "annabelle" delivers an asymmetric palindrome. This asymmetric word "annabelle" is a palindrome on the genotype level of morphogramatics but not a palindrome on the phenotype level of semiotics.

The pheno-words "anna", "b" and "belle" are therefore involved in a double game. As pheno-types, and in isolation, they are pheno- and geno-types at once. Both aspects are overlapping. In the context of composition to the word "annabelle" they are part of an asymmetric genotypic palindrome.

Results

In the morphosphere, asymmetric palindromes are structurally and quantitatively dominant.

Enantiomorph, dual and symmetric palindromes belong to the semiophere. Despite their dialogical and multi-world conception by Juri Lotman palindromes are based on an atomic and linear sign concept. The identity of the forward and backward reading is tested step-wise, comparing atomic signs after atomic signs. There are no considerations about contexts at all.

Palindromes in the morphosphere are a/symmetric, complementary and chiasitic.

The morphosphere is a sphere beyond the semiosphere. Like the semiosphere it has to be distinguished from the noosphere and the biosphere.

Applications

You might lock your door with one key, but you will have to unlock it with another key.

More about asymmetric palindromes at:

<http://memristors.memristics.com/Morphospheres/Asymmetric%20Palindromes.html>

3.2.6. Chiasm AB:C:BA

MorphoGrammar[abcba]

Alph = {[a]}

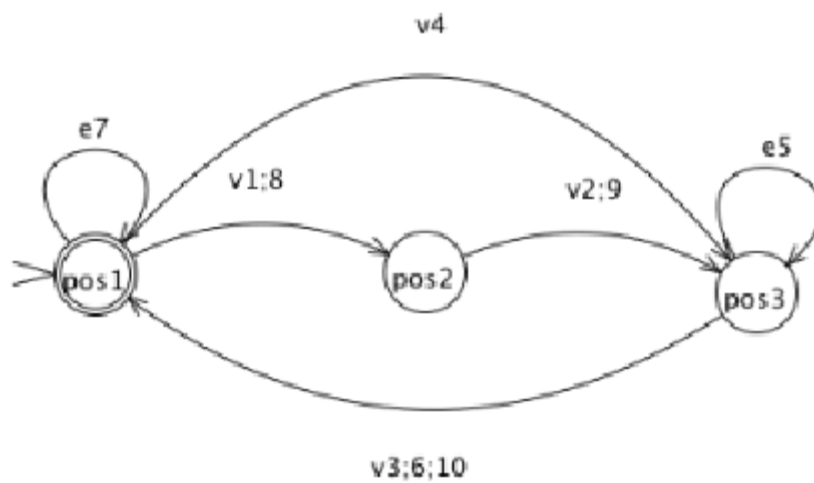
Rules= $S \Rightarrow [a] \mid [a]S[a]$.

Application: $S \Rightarrow_{\text{accr}} \text{bSb} \Rightarrow_{\text{iter}} \text{a(bSb)a} \Rightarrow_{\text{accr}} \text{ab c ba}$, with $c \notin (S \Rightarrow [a]S[a])$.

MorphoFSM[abcba]

1	2	3	4	5
[]	v_1	v_2	v_4	e_7
□	□	v_3	e_5	v_8
□	□	□	v_6	v_9
□	□	□	□	v_{10}

MorphoFSA[abcba]	pos ₁	pos ₂	pos ₃
pos ₁	e_7	$v_{1;8}$	v_4
pos ₂	—	—	$v_{2;9}$
pos ₃	$v_{3;6;10}$	—	e_5



http://www.thinkartlab.com/Chinese%20Challenge%20Pool/How_to_Compose.pdf

Chiasm scheme for [abcba]

$$v_1, v_2, e_5 = [a_1 b_2 c_0]$$

$$e_7, v_8, v_9 = \text{---} [c_0 b_3 a_4]$$

$$e_5 = [b_2 b_3], v_4 = [a_1 b_3]$$

$$e_7 = [a_1 a_4], v_{10} = [b_3 a_4]$$

$$v_{2,3} = \text{end} [c_0]$$

$$v_{6,9} = [c_0] \text{ start}$$

e_5, e_7 : mediating a', b' 's

Chiasm[abcba]

$$\text{chiasm}[abcba] = \begin{pmatrix} a_1 & b_2 & \longrightarrow & c_0 \\ \updownarrow & & & \updownarrow \\ a_4 & b_3 & \longleftarrow & c_0 \end{pmatrix}$$

Chiasm formula for [abcba]

$$\text{chiasm}[abcba] = \frac{(v_1, v_2, e_5) (e_7, v_8, v_9)}{(e_5); (e_7)}$$

$$\text{with } (v_1, v_2, e_5) = \text{rev}(v_9, v_8, e_7)$$

Logical analysis of [abcba]

$\text{pal}(p, q)$	(p, r)	(p, l)	(r, q)	(l, q)	(r, l)	(l, r)	(r)	(l)
a	a	–	a	–	a	–	a	–
b	b	–	–	b	–	b	b	–
c	c	c	c	c	–	–	–	–
b	–	b	–	b	b	–	–	b
a	–	a	a	–	–	a	–	a

MorphoFSM[aabcbaa] as an iterative prolongation of [abcba]

MorphoGrammar[aabcbaa]

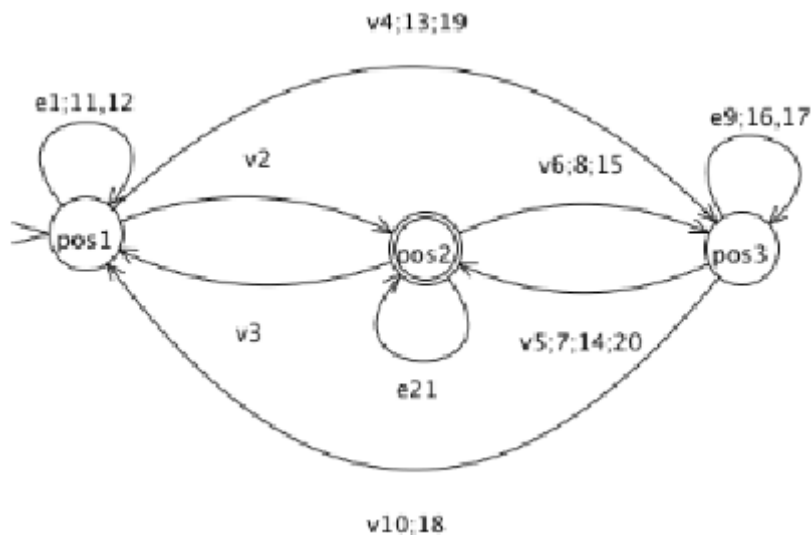
Alph = {[a]}

Rules= $S \Rightarrow [a] \mid [a]S[a]$.

Application: $S \Rightarrow_{\text{accr}} \text{bSb} \Rightarrow_{\text{iter}} \text{a(bSb)a} \Rightarrow_{\text{iter}} \text{a(abSba)a} \Rightarrow_{\text{accr}} \text{aab c baa}$, with $c \notin (S \Rightarrow [a]S[a])$.

MorphoFSA[aabcbaa]	pos ₁	pos ₂	pos ₃
pos ₁	$e_{1;11,12}$	v_2	$v_{4;13;19}$
pos ₂	v_3	e_{21}	$v_{2;6;15}$
pos ₃	$v_{3;6;10;18}$	$v_{7;14;20}$	$e_{9;16,17}$

	1	2	3	4	5	6	7
1		v_1	v_2	v_4	e_7	e_{11}	e_{16}
2			v_3	e_5	v_8	e_{12}	e_{17}
3				v_6	v_9	v_{13}	v_{18}
4					v_{10}	v_{14}	v_{19}
5						v_{15}	v_{20}
6							e_{21}



The morphogram [aabcbaa] deciphered as the number “1123211” in the decimal system is a **prime** number. A difference-theoretical interpretation of this prime number as a chiasm might shed some different light into its mystery.

<http://primes.utm.edu/curios/page.php/1123211.html>

3.3. Simulation of FSA by MorphoFSA

3.3.1. Relations

Relations to be studied are:

1. Morphogram [MG] to ENstructure of MG: [MG] and ENstructure[MG],

2. ENstructure[MG] to the machine-diagram representation of ENstructure[MG]: ENstructure[MG] and M-Diagramm[MG],
3. M-Diagram[MG] and machine definition and implementation of MorphoFSA: Def(MorphoFSA) and Prgr(MorphoFSA),
4. MorphoFSA and FSA.

3.3.2. Comparisons: MorphoFSA and FSA

To each MorphoFSA there are $\text{card}[\mu] \text{ trito}$ FSA representations.

FSA1: FSA(ab^n) with acceptance state “a” is not recognizing a sequence (baa...).

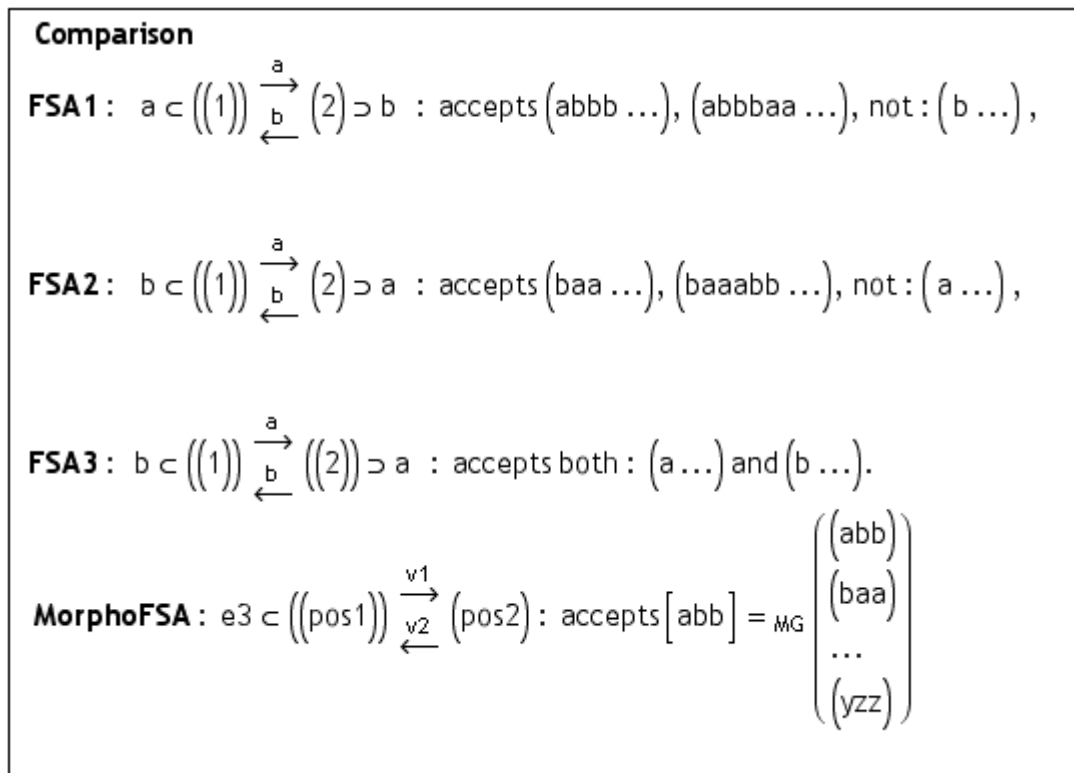
FSA2: FSA(ba^n) with acceptance state “b” is not recognizing a sequence (abb...).

But an FSA automaton with two acceptance states, “a” and “b”, is recognizing both sequences.

Hence the two acceptance state machine FSA3 =(FSA1, FSA2)are accepting both sequences.

FSA(abb..., baaa) with acceptance state “a” and “b” is recognizing the sequences (abb...) and (baa...).

On the other hand, the MorphoFSA ($v_1v_2\varepsilon_3$) with just *one* acceptance state “pos1” is simulating FSA3, i.e. FSA1 and FSA2 together.



The accepted sequences of FSA3 are not the results of an equivalence of FSA1 and FSA2 but are two different applications of the same but two acceptance state machine, one starting with “a”, the other starting with “b”. But the difference of one acceptance state to two acceptance states machines is crucial. There are two different types of machine.

In contrast, the MorphoFSA produces the abstraction from both FSA1 and FSA2 with just one acceptance state. Hence, to use two acceptance states for MorphoFSA is producing just an isomorphism between both definitions, and nothing more.

That is, MorphoFSA ($v_1v_2\varepsilon_3$) and MorphoFSA ($v_2v_1\varepsilon_3$)are isomorph.

Hence, to every MorphoFSM(m,k) there are $\frac{m!}{(m-k)!}$ FSA(m,k).

Each FSA is representable by a MorphoFSA.

This might easily be proven over the recursivity of the construction of both kinds of automata.

First step

MorphoFSA(ϵ^1) \iff FSA((q,x \rightarrow q), $\forall x \in \text{Alph}$): start state as a 0-transition.

MorphoFSA(ν^1) \iff FSA((q,x \rightarrow r), $\forall x \in \text{Alph}$): start transition as a 1-transition

Recursion steps

MorphoFSA(m,k) to MorphoFSA(m+1, k+1) \iff FSA(m, k) to FSA(m+1, k+1):

MorphoFSA((morph(m)) \cup + morph(monad)) \iff FSA((m,k) \cup { \forall atom \in Alph})

MorphoFSA((morph(m)) \cup + morph(monad))

MorphoFSA([ab]) \cup + morph(monad)) = [aba], [abb], [abc].

From the point of view of formal languages, the combinatorial correlation of both approaches is given by the observation:

FSA(Str = Σ^*) versus MorphoFSA(Str = Stirling2(Σ , *))

Hence, for FSA(|Str| = 2^3) = 8, MorphoFSA(|Str| = Stirling2(2, 3)) = 4.

3.3.3. Why are MorphoFSMs not just relational systems?

It seems still not easy to grasp the difference of *relational* and *differential* machines.

A diagrammatic representation of MorphoFSM[aabb], *Example2*, has two self-loops, automorphism, at position pos1. From a relational point of view, both loops would have formally to coincide and to represent the pattern [aa], maybe even as an iteration [aa] and [aa], but never as the two patterns [aa] and [bb] of [aabb].

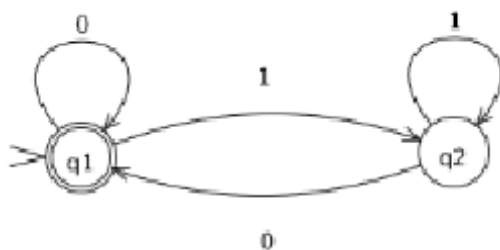
The case for FSA(0,1) of *Example1* shows exactly this, at position q1, the loop with the element "0" produces sequences of "0"s, and the loop with the element "1" at position q2 produces sequences of "1"s.

On the other hand, in a relational (category-theoretic, etc.) setting, loops at different positions (objects), pos1 and pos2, would have to represent different patterns. The *Example3* shows a distribution of the same pattern [bb] at different positions, pos1 and pos2.

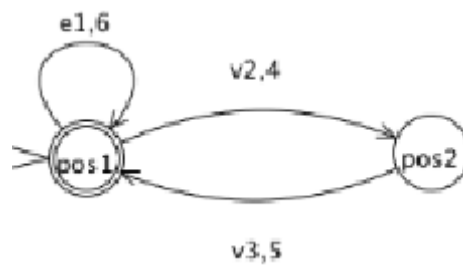
Hence, the identity of the elements are defined by the differential system of positions of the differences epsilon and nu.

Again, considering the *mathematical* definition of FSMs as finite automata makes it more than clear that this definition is based on identical objects (elements), "*finite non empty set of symbols*", and their relations, defined by transitions.

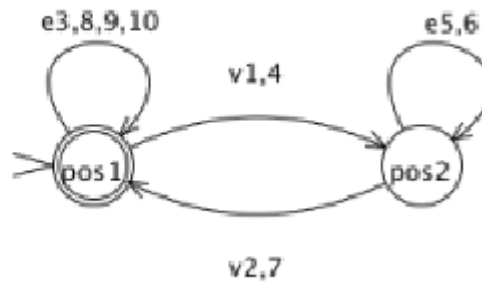
Example1: FSA{0,1}



Example2: Diagram for MorphoFSM[aabb]



Example3 : Diagram for MorphoFSM[abbbb]



3.4. Pumping lemma, first

3.4.1. The classical scenario

There are significant presumptions to run the argumentation of the decision about the regularity or nonregularity of formal languages with the help of FSMs.

An application of those arguments for MorphoFSMs goes hand in hand with a deconstruction of the basic presumption based on FSMs.

From a meta-theoretical point of view it also has to be analyzed if the proof by contradiction (*reductio ad absurdum*) is as safe as it is proclaimed.

The *general* principle to prove the existence of non-regular languages is using *Cantors* method of diagonalization.

"The existence of non-regular languages is guaranteed by the fact that the regular languages of any alphabet are countable, and we know that the set of all subsets of strings is not countable." (standard)

The proof of the existence of the non-regularity of specific languages is using the *reductio ad absurdum* principle and the pumping lemma.

With the presumptions of monocontextural logic and arithmetic there is nothing to add to the established argumentations for a limitation of the power of formal languages.

Treated as monocontextural entities morphogrammatic elaborations get reduced to what they are not by definition.

If we take the graphematic turn to morphogrammatics seriously there are fundamental consequences for arithmetic and logic to consider. And the natural strategies of argumentations will their naturality and will be unmasked as historically limited.

Correspondence and countability problems

"Since $P(\Sigma^*)$, the set of all languages, is uncountable, whereas the set of regular languages is countable, some language must be non-regular."

"A correspondance between words and languages is naturally established for the classical case.

" Σ^* , the set of all finite strings, is *countable*:

- We can list all finite strings in order of length, put them in one-to-one correspondence with \mathbb{N} .
- E.g., ϵ , 0, 1, 00, 01, 10, 11, 000,...

This correspondance is, at first, slightly disturbed by the morphogrammatic case: Because,

- There is no atomic alphabet,
 - 0 and 1 are morphogrammatically equal,
 - 00 and 11 are morphogrammatically equal,
 - 01 and 10 are morphogrammatically equal,
 - 000... and 111.. are morphogrammatically equal,
- and so on.

Hence, the nice correspondance function f for symbolic languages with

$f(\epsilon) = L_0$,

$f(0) = L_1$,

$f(1) = L_2$,

and so on, is not holding for morphic languages. At least not in this setting.

Thus, the well known *diagonal* construction gets into trouble with the lack of the classical one-to-one correspondance:

$D = \{ w \in \Sigma^* \mid w \text{ is not in } f(w) \}$.

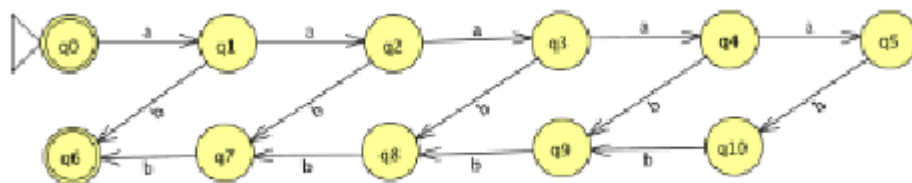
http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-045j-automata-computability-and-complexity-spring-2011/lecture-notes/MIT6_045JS11_lec05.pdf

FSA: finite memory limitation

"A DFA that recognizes L should first count the number of a 's in the beginning of the word. The problem is that a DFA has only finitely many states, so it is bound to get confused: Some a^i and a^j take the machine to the same state, and the machine can no longer remember whether it saw i or j letters a . Since the machine accepts input word $a^i b^j$, it also accepts input word $a^j b^i$, which is not in the language. So the machine works incorrectly."

"There are $n + 1$ states q_0, q_1, \dots, q_n but the machine has only n different states, so two of the states must be identical.

(Kari, p.29)



"The problem is that we only have finite memory, and so at some point we will exhaust the number of a 's we can remember. The idea is to make precise the finite memory limitation. It is done so by the following classical *Pumping Lemma*."

<http://www.cs.wcupa.edu/~rkline/fcs/re-pump.html>

3.4.2. The morphogrammatic scenario

Morphic language classification of the morphograms of the trito-universe TU

- [a]
- [aa]
- [ab]
- [aab...]
- [aba...]
- [abb...]

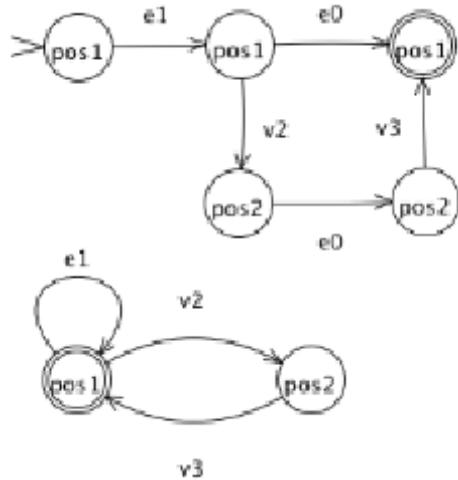
- [abc...]
 - [abca...]
- and so on.

The question remains: What is a morphogram that doesn't belong to a morphogrammatic language (script)?

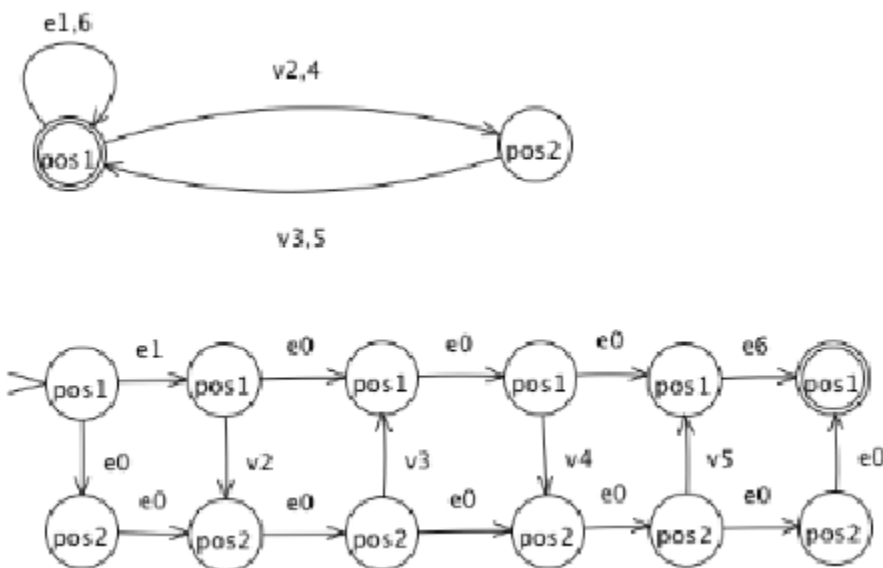
Null operator e_0

For some analysis it might be useful to augment the set of operations from e =loop and v =differentiation with the operation nil-differentiation = e_0 . The machines represented by $\text{DiagrMorphoFSM}(e,v)$ and $\text{DiagrMorphoFSM}(e,v,e_0)$ are mg-equivalent.

$\text{DiagrMorphoFSM}(e,v)$ and $\text{DiagrMorphoFSM}(e,v,e_0)$ for [aab]



$\text{DiagrMorphoFSM}(e,v)$ and $\text{DiagrMorphoFSM}(e,v,e_0)$ for [aabb]



Is this supporting the structure if the distinctions v_2 to v_5 are strong enough to hold the memory of the development together?

In contrast to the FSA concept, there is no such *counting* process where the result has to be *remembered* and used for the continuation of the process of the construction of the word or morphogram.

For the MorphoFSM the development is not atomic, symbolic and step-wise, one after the other: (a, aa, aab, aabb). But retrograde recursive as: (aabb, aabb, aabb, aabb, aabb, aabb). The distinction for the 'head' "aa" is not complete with the first differentiation by "e1". This would hold only for "aa" alone, as a single and isolated morphogram. But the differentiation "e1", represented as [aa] is embedded into the

whole morphogram [aabb]. Hence the monomorphy [aa] is determined additionally by the distinctions “v2-5” in relation with the ‘tail’ of the morphogram.

Hence, for [aabb] as [a₁a₂b₃b₄], “a₁” is defined by “a₂”, “b₃” and “b₄”. The same for “a₂”; “a₂” is defined retrogradely by “b₃” and “b₄”. This process goes ‘forwards’ and ‘backwards’:

e1;a₁-> a₂, v2;a₁-> b₃, v3;b₃ -> a₂, v4;a₁-> b₄, v5;b₄-> a₂, e6;b₃->b₄.

Again, this shows the crucial difference of recursive repetition, i.e. *recursion*, and retrograde recursivity, i.e. *reflection*.

Therefore, the monomorphy [aa] of the palindrome [aabb] is defined by the whole morphogram and there is no need to *count* and *remember* the number of elements of the ‘head’ of the palindrome to be able to ‘repeat’ it as the ‘tail’. Those informations are intrinsically included in the procedure of the definition of the morphogram, i.e. the palindrome.

In other words, the ‘head’ [aa] of the palindrome as such and in isolation has no completed and definitely defined existence (cf. § 3.2.2). Again, in the case of FSAs, the ‘head’ (aa) in (aabb) is recognized and ‘counted’ as complete in itself.

As a consequence, it has to be seen that “e1” as the first arrow of the diagram $\text{DiagrMorphoFSM}(e,v,e0)$ is retrogradely determined by the whole diagram, and has a different definition if taken in isolation. The same holds for the last arrow, “e6”. This might be more obvious because it occurs at the “end” of the morphogram. But again, this is misleading. A morphogram is despite its step-wise analysis not a string, chain or sequence but an inter-related whole, *morphé*. This fact is faithfully represented by the EN-structure of the morphogram.

Retrogression and anticipation

On the other hand, the ‘tail’ gets its characterization by the characterization of the ‘head’ by the definition of the ‘head’ designed by the ‘tail’.

Again, these retrograde recursivity functions, functioning as the ‘*counter*’ and as the ‘*memory*’ necessary for the construction of the palindrome are defining the crucial difference to the classical a-temporal symbolic constructions.

The retrograde movement to characterize the ‘head’ of a palindrome is involved with a progression ‘into’ the ‘tail’ to define by retro-gression the ‘head’ of the palindrome.

Retrograde recursivity is always involved, simultaneously, into *anticipation*.

In fact, this retrograde functionality is a general property of morphograms and their construction rules.

An application of this surprising fact to automata shows that morphic automata, MorphoFSM, are defined by their immanent temporality based on their retrograde recursive characterization.

Classical finite state automata, and all its further developments, up to Turing machines, are by definition a-temporal. They might have access to a storage function but they don’t have an *intrinsic* memory.

As a result the questions of regularity/nonregularity of languages have to be tackled in a very different light.

3.5. Formal approximations for MorphoFSA

3.5.1. Automata-theoretical approach

Abstract symbolic automaton

"An *automaton* is a triple $A = (S, \rightarrow, S_{in})$ where S is a set of *states*, $S_{in} \subseteq S$ is a set

of *initial* states and $\rightarrow \subseteq S \times \Sigma \times S$ is a *transition* relation. The automaton is said to be deterministic if S_{in} is a singleton and \rightarrow is a function from $S \times \Sigma$ to S . “ (M. Mukund)
<http://www.cmi.ac.in/~madhavan/papers/pdf/tcs-96-2.pdf>

Abstract kenomic automaton MorphA

In contrast to the symbolic abstract automaton, the morphic abstract automaton is defined, at first, over differences and not over states, represented by symbols.

Hence, a morphic abstract automaton is a triple **MorphoA** = (Pos, \rightarrow , Pos_{in}) where Pos is a set of *differences*, marked by positions Pos, Pos_{in} \subseteq Pos is a set of *initial* differences, $\Delta = \{\epsilon_i, v_i, i \in \mathbb{N}\}$ and $\rightarrow \subseteq \text{Pos} \times \Delta \times \text{Pos}_{in}$ is a *differentiation* operation.

This motivate the table notation for MorphoFSA:

MorphoFSA[abac]	pos ₁	pos ₂	pos ₃
pos ₁	initial	v ₁	v ₄
pos ₂	v ₃	e ₂	v ₆
pos ₃	—	v ₅	—

Abstract MorphoFSA

MorphoFSA = (Q, Δ , δ , q_s, F)

where

Q is a finite set of *positions* {q_i | i is a non-negative integer}

Δ is the finite *input* alphabet of differences, $\Delta = \{\epsilon_i, v_i, i \in \mathbb{N}\}$

δ is the *differentiation* operation, $\delta : D \rightarrow \text{Stirling2}(2, Q)$ where D is a finite subset of $Q \times \text{Stirling2}(\Delta, *)$

q_s (is member of Q) is the *initial* position

F (is a subset of Q) is the set of *final* positions.

MorphoFSA in analogy to FSA

MorphoFSA = (Q, Σ , δ , q_s, F)

where

Q is a finite set of *states* {q_i | i is a non-negative integer}

Σ is the finite *input* alphabet

δ is the *transition* function, $\delta : D \rightarrow \text{Stirling2}(2, Q)$ where D is a finite subset of $Q \times \text{Stirling2}(\Sigma, *)$

q_s (is member of Q) is the *initial* state

F (is a subset of Q) is the set of *final* states .

3.5.2. Programming approach

ML-Programming approach

Delta = { ϵ_i, v_j , 1 ≤ i, j ≤ s(m), m ∈ ℕ}

Position: {pos_i, i ∈ AG(MG)}

signature Delta = sig eqtype delta end;

signature Position = sig eqtype pos end;

signature Automaton =

sig

eqtype Delta;

eqtype Position;

type dfa;

val next: delta list * pos * dfa -> delta list * pos * dfa;

val delta_star: delta list * pos * dfa -> pos;

val accept: delta list -> dfa -> bool;

end;

```

(* example Automaton M(2,2) *)
structure two = struct datatype delta =  $\epsilon_1$  |  $\epsilon_2$  |  $\epsilon_3$  | v1 | v2 | v3 end;
structure two = struct datatype position = pos1 | pos2 end;
structure DFA1 = DFA (structure delta = two; structure position = two);
open two;
open two;
open DFA1;
val delta = fn pos1 => (fn v1 => pos2 )
              | pos2 => (fn v2 => pos1)
              | pos1 => (fn  $\epsilon_3$  => pos1);
val M(2,2) = (pos1, delta, [pos2] );
  accept [abb] M(2,2);
    fn pos1 => (fn v1 => pos2 )     $\equiv$  [ab] = start
      | pos2 => (fn v2 => pos1)  $\equiv$  [ab]
      | pos1 => (fn  $\epsilon_3$  => pos1)  $\equiv$  [abb] = final
      (* | pos2 => (fn  $\epsilon_0$  => pos2) *)
  accept [abbb] M(2,2)
    fn pos1 => (fn v1,4 => pos2 )     $\equiv$  [ab] = start
      | pos2 => (fn v2 => pos1)  $\equiv$  [ab]
      | pos1 => (fn  $\epsilon_3$  => pos1)  $\equiv$  [abb]
      | pos2 => (fn  $\epsilon_{5,6}$  => pos2)  $\equiv$  [abbb] = final .
  accept [abbbb] M(2,2)
    fn pos1 => (fn v1,4 => pos2 )     $\equiv$  [ab] = start
      | pos2 => (fn v2,7 => pos1)     $\equiv$  [ab]
      | pos1 => (fn  $\epsilon_{3,8,9,10}$  => pos1)  $\equiv$  [abb]
      | pos2 => (fn  $\epsilon_{5,6}$  => pos2)     $\equiv$  [abbb]
      | pos1 => (fn  $\epsilon_{3,8,9,10}$  => pos1)  $\equiv$  [abbbb] = final.

```

3.6. Observations on morphic automata

3.6.1. Iteration and retrogradeness

What happens if M1 is iterating the transition " $\text{pos}_1, \epsilon_3 \rightarrow \text{pos}_1$ "?

Hence, a prolongation from the morphogram [abb] to the morphogram [abbb] would represent this kind of iteration.

But a prolongation is changing the whole pattern of the morphogram. Therefore it is not covered by an iteration but by a retrograde repetition.

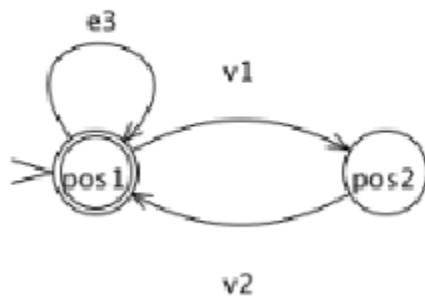
FSA:

(abb) \rightarrow (abbb): The automaton needs just one run more of the transition rule " $q_2, b \rightarrow q_2$ " to recognize the string (abbb). This additional or iterated run is not changing retrogradely the definition of the automaton. It is added successively like the arithmetic step from n to n+1.

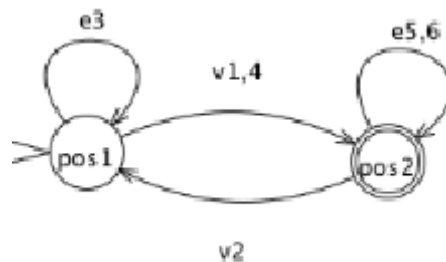
MorphoFSA:

[abb] :: ($\forall v \in \epsilon$) \rightarrow [abbb] :: ($\forall v \in v \in \epsilon$): The additional run " $q_1, v_4 \rightarrow q_2$ " is changing retrogradely the labels of the transition ϵ_0 to the labeled transitions " $q_2, \epsilon_{5,6} \rightarrow q_2$ " and the localization of the final state from pos1 to pos2.

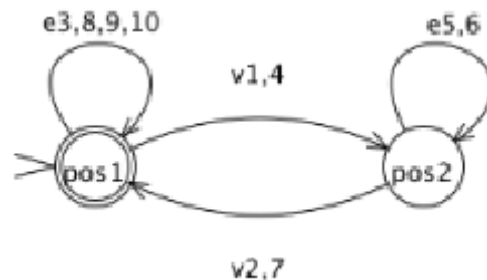
MorphoFSA M1



MorphoFSA M1.1



MorphoFSA M1.1.1



Slogans

An iteration of a differentiation is not making a difference. But the differentiation of a difference is generating a differentiation. (Calculus of differentiation)

This wording is complementary to the wording of the act of distinction:

A distinction made again is a distinction. A distinction inside a distinction is no distinction (G. Spencer Brown, Calculus of Indication).

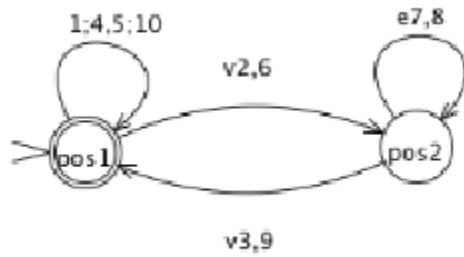
Prolongations

$[aab] \Rightarrow \{[aaba], [aabb], [aabc]\} \Rightarrow$

$[aaba] \Rightarrow \{[aabaa], [aabab], [aabac]\}.$

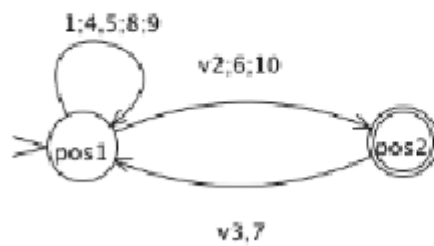
Diagrams for the MorphoFSM prolongations [aabaa], [aabab], [aabac]

The diagrams show the *iterative* and *accretive* prolongations of the pattern [aaba] to the patterns [aabaa], [aabab] and [aabac]. An analysis shows the change of the self-loop e_{10} for [aabaa] into a differentiation v_{10} for the iteration of [b] in [aabab]. An accretion happens for [aabac] that augments the aggregation to 3. The sub-constellation $(e^1_{4;5})$ remains stable. There are no other direct prolongations for the pattern [aaba].



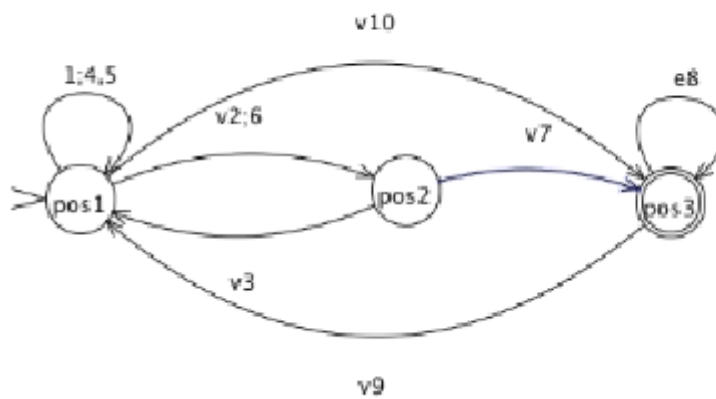
EN[aabaa]	2	3	4	5
[]	e_1	v_2	e_4	e_7
□	□	v_3	e_5	e_8
□	□	□	v_6	v_9
□	□	□	□	e_{10}

MorphoFSA[aabaa]	pos ₁	pos ₂
pos ₁	$e_{1,4,5,10}$	$v_{2,6}$
pos ₂	$v_{3,9}$	$e_{7,8}$



EN[aabab]	2	3	4	5
[]	e_1	v_2	e_4	v_7
□	□	v_3	e_5	e_8
□	□	□	v_6	e_9
□	□	□	□	v_{10}

MorphoFSA[aabab]	pos ₁	pos ₂
pos ₁	$e_{1,4,5,8,9}$	$v_{2,6;10}$
pos ₂	$v_{3,7}$	—



EN[aabac]	2	3	4	5
[]	e_1	v_2	e_4	v_7
□	□	v_3	e_5	e_8
□	□	□	v_6	v_9
□	□	□	□	v_{10}

MorphoFSA[aabac]	pos ₁	pos ₂	pos ₃
pos ₁	$e_{1;4,5}$	$v_{2,6}$	v_{10}
pos ₂	v_3	—	v_7
pos ₃	v_9	v_5	e_8

3.6.2. Operations on MorphoFSM

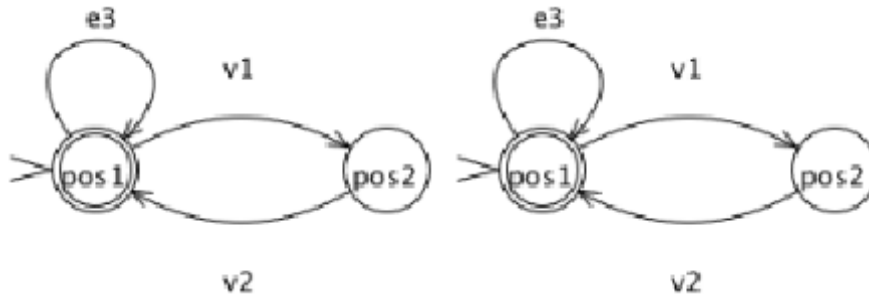
Several operations on FSM are standard: concatenation, union, difference, star operation, insertion.

Those operations applies directly to the morphic case. The morphogrammatic operations of concatenation, insertion, union, are build directly by applying the results from morphogrammatcs to morphic FSMs.

An interesting application is the operation of *multiplication* (cooperation) of morphograms and its representation by MorphoFSMs.

Cooperations of MorphoFSMs

fsm-kmul(MorphoFSM[abb], MorphoFSM[abb]): kmul[$v_1v_2e_3$] [$v_1v_2e_3$].



–kmul[1, 2, 2][1, 2, 2];

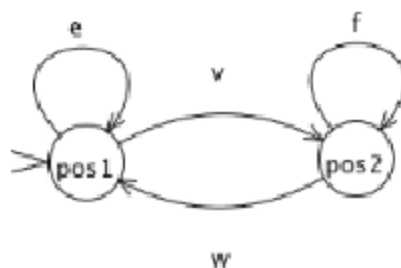
val it = [[1, 2, 2, 2, 1, 1, 2, 1, 1], [1, 2, 2, 3, 1, 1, 3, 1, 1], [1, 2, 2, 2, 3, 3, 2, 3, 3], [1, 2, 2, 3, 4, 4, 3, 4, 4]]
: int list list

1. – ENstructure[1, 2, 2, 2, 1, 1, 2, 1, 1];

val it =

[[],
[(1, 2, N)],
[(1, 3, N), (2, 3, E)],
[(1, 4, N), (2, 4, E), (3, 4, E)],
[(1, 5, E), (2, 5, N), (3, 5, N), (4, 5, N)],
[(1, 6, E), (2, 6, N), (3, 6, N), (4, 6, N), (5, 6, E)],
[(1, 7, N), (2, 7, E), (3, 7, E), (4, 7, E), (5, 7, N), (6, 7, N)],
[(1, 8, E), (2, 8, N), (3, 8, N), (4, 8, N), (5, 8, E), (6, 8, E), (7, 8, N)],
[(1, 9, E), (2, 9, N), (3, 9, N), (4, 9, N), (5, 9, E), (6, 9, E), (7, 9, N), (8, 9, E)]]
: (int * int * EN) list list

MorphoFSM[abbbbaabaa]



e = (3,12,13,15,16,22,26,27,33,34)
v = (1,4,9,13,15,17,19,23,25,28,31,35),
w = (2,8,11,14,18,21,24,30,32),
f = (5,6,7,10,20,21,29,36).

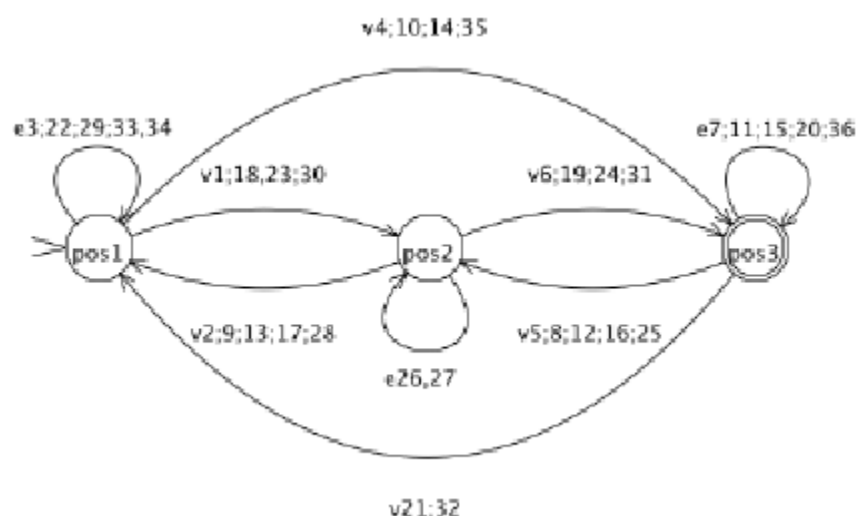
2. – ENstructure[1, 2, 2, 3, 1, 1, 3, 1, 1];

val it =

```
[[],
[(1, 2, N),
[(1, 3, N), (2, 3, E)],
[(1, 4, N), (2, 4, N), (3, 4, N)],
[(1, 5, E), (2, 5, N), (3, 5, N), (4, 5, N)],
[(1, 6, E), (2, 6, N), (3, 6, N), (4, 6, N), (5, 6, E)],
[(1, 7, N), (2, 7, N), (3, 7, N), (4, 7, E), (5, 7, N), (6, 7, N)],
[(1, 8, E), (2, 8, N), (3, 8, N), (4, 8, N), (5, 8, E), (6, 8, E), (7, 8, N)],
[(1, 9, E), (2, 9, N), (3, 9, N), (4, 9, N), (5, 9, E), (6, 9, E), (7, 9, N), (8, 9, E)]]
```

: (int * int * EN) list list

MorphoFSM[abbcaacaa]



3. – ENstructure[1, 2, 2, 2, 3, 3, 2, 3, 3];

val it =

```
[[],
[(1, 2, N),
[(1, 3, N), (2, 3, E)],
[(1, 4, N), (2, 4, E), (3, 4, E)],
[(1, 5, N), (2, 5, N), (3, 5, N), (4, 5, N)],
[(1, 6, N), (2, 6, N), (3, 6, N), (4, 6, N), (5, 6, E)],
[(1, 7, N), (2, 7, E), (3, 7, E), (4, 7, E), (5, 7, N), (6, 7, N)],
[(1, 8, N), (2, 8, N), (3, 8, N), (4, 8, N), (5, 8, E), (6, 8, E), (7, 8, N)],
[(1, 9, N), (2, 9, N), (3, 9, N), (4, 9, N), (5, 9, E), (6, 9, E), (7, 9, N), (8, 9, E)]]
```

: (int * int * EN) list list

ENstructure table with enumeration

[illegible]

4. – ENstructure[1, 2, 2, 3, 4, 4, 3, 4, 4];

```
val it =
```

[[]

 $[(1, 2, N)],$
$$[(1, 3, N), (2, 3, \mathbf{E}) = [\mathbf{bb}],$$
$$[(1, 4, N), (2, 4, N), (3, 4, N)],$$
$$[(1, 5, N), (2, 5, N), (3, 5, N), (4, 5, N)],$$
$$[(1, 6, N), (2, 6, N), (3, 6, N), (4, 6, N), (5, 6, E)] = [\mathbf{dd}]$$
$$[(1, 7, N), (2, 7, N), (3, 7, N), (4, 7, \mathbf{E}) = [\mathbf{c}' \mathbf{c}], (5, 7, N), (6, 7, N)],$$
$$[(1, 8, N), (2, 8, N), (3, 8, N), (4, 8, N), (5, 8, E) = [d^* d], (6, 8, E) = [d^* d], (7, 8, N)],$$
$$[(1, 9, N), (2, 9, N), (3, 9, N), (4, 9, N), (5, 9, E) = [d' d], (6, 9, E) = [d' d], (7, 9, N), (8, 9, E) = [dd]]]$$

```

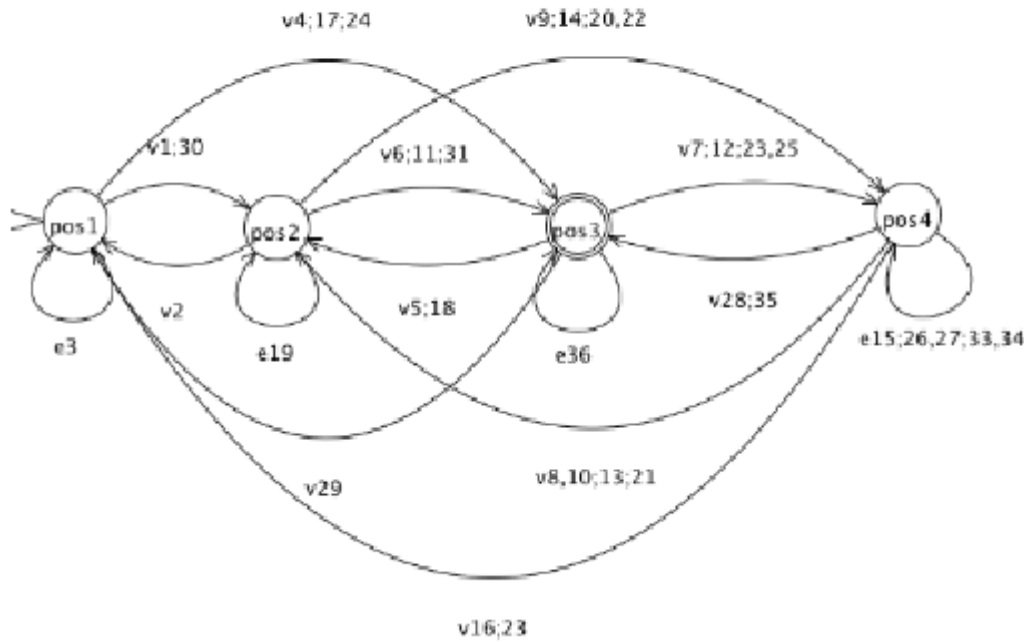
: (int * int * EN) list list

```

ENstructure table with enumeration

[illegible]

MorphoFSM[abbcddcdd]



MorphoFSA[abbcddcdd]	pos ₁	pos ₂	pos ₃	pos ₄
pos ₁	e3	v1; 30	v4; 17; 24	–
pos ₂	v2	e19	v6; 11; 31	v9; 14; 20, 22
pos ₃	v29	v5; 18	e36	v7; 12; 23, 25
pos ₄	v16; 23	v8, 10; 13; 21	v28; 35	e15; 26, 27; 33, 34

Limits of representations

A more systematic study of the mappings between morphograms and MorphoFSM diagrams is required to get better insights into the behavior of MorphoFSMs. It seems that there is still no unambiguous procedure available to map morphograms onto morphic FSM diagrams. The *analogy* to FSMs might come to an end, and new methods shall be applied to continue the study of morphic “ultra-finite differentiation machines”.

There is also a *quantitative* argument: multiplications (cooperations) of morphic automata are running quickly into ‘astronomic’ magnitudes.

length(kmul[MG] [MG]):

kmul	[1, 2]	[1, 2, 3]	[1, 2, 1]
[1, 2]	4	18	4
[1, 2, 3]	18	588	18
[1, 2, 3, 1]	18	588	18
[1, 2, 3, 2]	18	588	18
[1, 2, 3, 3]	18	588	18
[1, 2, 3, 4]	108	30240	108
[1, 2, 3, 4, 1]	108	30240	108
[1, 2, 3, 4, 4]	108	30240	108
[1, 2, 3, 4, 5]	780	2277600	780
[1, 2, 3, 4, 5, 6]	6600	??	6600
[1, 2, 3, 4, 5, 6, 7]	63840	??	63840
[1, 2, 3, 4, 5, 6, 7, 8]	693840	–	–

length(kmul[1,2][1,1]) = 1.

f (n; 0) : 1, 1, 4, 18, 108, 780, 6600, 63840, 693840, . . . , partial derangements (A144085).

Some examples for kmul⁹

Coalitions of MorphoFSMs: sequential composition

Coalitions in morphogrammatics are corresponding concatenations of morphograms. But concatenation is not additive for morphograms but super-additive.

"The *concatenation* of two words is the word obtained by writing the first word followed by the second one as a single word. For example, the concatenation of *data* and *base* is the word *database*.

Notation for concatenation is similar to normal multiplication: For example, $ab \notin aab$ = $abaab$: The multiplication sign does not need to be written if the meaning is clear, i.e., uv is the concatenation of words u and v . So, for example, if $v = a$ and $w = ab$, then $vw = v \notin w = aab$." (Kari)

Examples

kconcat[a][a] = {[aa], [ab]} (repetition as iteration and accretion)

kconcat[aa][ab] = {[aaab], [aaba], [aabc]}

kconcat[ab][ab] = {[abab], [abba], [abac], [abca], [abbc], [abcb], [abcd]}

- kconcat [1,2] [1,2,3];

val it =

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

Programming in SML/NJ¹³

System SML/NJ: <http://www.smlnj.org/>

Morphogrammatics: <http://www.thinkartlab.com/pkl/SML-sources.NJ/ALL-MG-nov2012.sml>

Book Morphogrammatik: <http://www.thinkartlab.com/pkl/media/mg-book.pdf>

The file, ALL-MG-nov2012.sml, contains,

1. smlcus.sml (alley stoughton),
2. SML-Basic functions (UEMA),
3. ALL.sml (Kenogrammatics, Thomas Mahler).

Hence, the nice classical example for the concatenation of "*data*" with "*base*" resulting in "*database*" is not holding in this restrictive way for morphogrammatic concatenations.

Following the retrograde recursivity of morphic concatenation, the super-additivity takes into account the different possible *meanings* of the terms "*data*" and "*base*" marked by the different realizations of the patterns. In the case of the formal example, with [aa] and [ab], there are 7 possible interpretations, i.e. realizations of

the pattern [ab] in respect to [aa]. Those different interpretations might be taken successively, recurring to different meanings of the operation “concatenation”, or they might be conceived as representing the whole range of the complex meaning of the concatenation in consideration, and are therefore understood as holding all together at once by mediation in a complex semantic context.

Not just the added morphogram [ab] is changing in the process of concatenation with the morphogram [aa] but retrogradely the morphogram [aa] is changing its role in the ‘context’ of the composition too. This holds for the semantic example too. The term “data” gets a different meaning in the composition with the term “base” as it has in another context and also as it has in isolation.

Thus the concatenation “*kconcat*” of the two morphic machines MorphoFSM [aa] and MorphoFSM[ab], i.e. “*e1(pos1)*” and “*v1(pos1, pos2)*”, is offering 7 different sequentially or ‘parallel’ composed automata.¹⁴

length(*kconcat*[MG][MG])

kconcat	[1, 2]	[1, 2, 3]	[1, 2, 3, 4]
[1, 2]	7	13	21
[1, 2, 3]	13	34	73
[1, 2, 3, 1]	13	34	73
[1, 2, 3, 2]	13	34	73
[1, 2, 3, 3]	13	34	73
[1, 2, 3, 4]	21	73	209
[1, 2, 3, 4, 1]	21	73	209
[1, 2, 3, 4, 4]	21	73	209
[1, 2, 3, 4, 5]	31	136	501
[1, 2, 3, 4, 5, 6]	43	229	1045
[1, 2, 3, 4, 5, 6, 7]	57	358	1961

length(*kmul*[1,2][1,2]): Central polygonal numbers: $n^2 - n + 1$.
<http://oeis.org/A002061>

Comparison

length(*kmul*[1,2][1,2]) < length(*kconcat*[1,2][1,2]).
 Some further examples for *kconcat*¹⁵.

3.6.3. Dissemination of MorphoFSMs

Up to this point the possible polycontextuality of the involved languages and machines had not yet been explicitly thematized.

The analogy to finite state machines and regular languages of morphic languages (scriptures) and machines with the classical operations on languages and machines comes to an end with the introduction of *transpositional* (transjunctional, rejectional, etc.) strategies of combining languages and machines. Transpositional operations are exploiting the mechanisms of bifunctionality of diamond category theory for the purpose of disseminating classical and non-classical concepts of languages and machines.

http://www.users.abo.fi/jboling/cdes/op_on_aut.pdf

Taken this way, the complexion of languages and machines under the operation of *transposition* implemented by bifunctionality isn’t regular anymore.

The reason is obvious. The operation of transposition (\mathbb{I}) is not a member of the standard operations on regular languages and machines of complement, union, intersection, star, etc. (\cup , \cap , \wedge , \setminus , $*$) and machine compositions parallel $||$ and product X , and it isn’t definable by them either.

Hence, the class of regular languages is not closed under the *bifunctorial* transposition operation.

But there is an inverse interpretation too: In the framework of polycontextuality, i.e. the dissemination operations over the kenomic grid, the domain of regular languages is not complete.

Classical regular languages and their machines are restricted to a linear (serial/parallel) non-interactive type of composition.

Polycontextual compositions of languages are interactional.

With that, we are finally entering the domain of *polycontextural* complexions of interactional, reflectional and interventional morphic finite differentiation machines.

A 3 – contextural dissemination of composed machines

$$\begin{aligned} \mathcal{U}^{(3)} &= (\mathcal{U}_1 \amalg_{1,2} \mathcal{U}_2) \amalg_{1,2,3} \mathcal{U}_3 \\ (\mathcal{U}_1 \cap_{1,2} \mathcal{U}_2) \cap_{1,2,3} \mathcal{U}_3 &= \emptyset: \\ \mathcal{U}_i &= \{\mathcal{M}_i, \mathcal{N}_i\}, \quad i = 1, 2, 3 \end{aligned}$$

$$\begin{bmatrix} \mathcal{N}_1 & - & \mathcal{N}_3 \\ \mathcal{M}_1 & \mathcal{N}_2 & - \\ - & \mathcal{M}_2 & \mathcal{M}_3 \end{bmatrix}:$$

$$\left(\begin{pmatrix} (\mathcal{M}_1 \circ_{1,0,0} \mathcal{N}_1) \\ \amalg_{1,2,0} \\ (\mathcal{M}_2 \circ_{0,2,0} \mathcal{N}_2) \\ \amalg_{1,2,3} \\ (\mathcal{M}_3 \circ_{0,0,3} \mathcal{N}_3) \end{pmatrix} \right) = \left(\begin{pmatrix} \mathcal{M}_1 \\ \amalg_{1,2,0} \\ \mathcal{M}_2 \\ \amalg_{1,2,3} \\ \mathcal{M}_3 \end{pmatrix} \right) \circ_1 \circ_2 \circ_3 \left(\begin{pmatrix} \mathcal{N}_1 \\ \amalg_{1,2,0} \\ \mathcal{N}_2 \\ \amalg_{1,2,3} \\ \mathcal{N}_3 \end{pmatrix} \right)$$

$$\circ = \{\cup, \bigcup, \cap, *, ^; \parallel, \times\}$$

Example of polycontextural combinations of machines

$$\mathcal{M}^{(4)} \cup \cap \cap \cap \wedge \cup \cap \mathcal{N}^{(4)} : \begin{pmatrix} \mathcal{M}^{(4)} \circ \mathcal{N}^{(4)} & \mathcal{N}^1 & \mathcal{N}^{2,3} & \mathcal{N}^{4,5,6} \\ \mathcal{M}^{1,2,4} & \mathcal{M}^1 \bigcup \mathcal{N}^1 & \mathcal{M}^2 \cap \mathcal{N}^2 & \mathcal{M}^4 \wedge \mathcal{N}^4 \\ \mathcal{M}^{3,5} & - & \mathcal{M}^3 \cap \mathcal{N}^3 & \mathcal{M}^5 \bigcup \mathcal{N}^5 \\ \mathcal{M}^6 & - & - & \mathcal{M}^6 \cap \mathcal{N}^6 \end{pmatrix}.$$

This exercise has to be declinated over all constituents of the combined, i.e. disseminated MorphoFSMs. Therefore, ‘alphabets’, ‘states’ and ‘transitions’ of \mathcal{M}^i and \mathcal{N}^i , $i=1, \dots, 6$ are mutually disjunct (discontextural) but mediated and each is defined autonomously at its place and contexture.

Regular languages are closed under the operation of *union*, *intersection*, *difference*, etc. The new topic is *mediation*: Are regular languages closed under mediation? Mediation of operations that are not involved in negations or permutations are

directly mediated, and are not posing any problems. Hence, a complexion of mediated union, intersection and concatenation like " $\cup \cap \cap \wedge \cup \cap$ " is "closed" under mediation. While a complexion, containing negational operations is not directly closed.

Hence, the complexion with union, intersection and difference is violating the conditions of mediation.

$$\mathcal{M}^{(4)} \cup \cap - \wedge \cup \cap \mathcal{N}^{(4)} \notin \text{closed under mediation } \Pi.$$

But, because of the commutativity of the differences at place 1 and 4, the mediation of " $-\cap \cup -\cup \cap$ " is accepted.

$$\text{Hence, } \mathcal{M}^{(4)} -\cap \cup -\cup \cap \mathcal{N}^{(4)} \in \text{closed under mediation } \Pi.$$

Also mediation as such is a crucial feature of polycontextural structurations, the more interesting constellations are entering the game as transpositional, reflectional and interventional operations.

Short reminder of some schemes

Reflectional structuration

$[\text{refl, id, id}] : \mathcal{M}^{(3,3)} \underset{\text{morphoFSM}}{\cup} \cup \cap^{(3,3)} \mathcal{N}^{(3,3)},$
type < 1.1., 1.2., 1.3; 2.2, 3.3 > :

$$\begin{array}{c} (\mathcal{M}^{1.1} \underset{\cup}{\cup} \mathcal{N}^{1.1}) \\ \Pi \\ (\mathcal{M}^{1.2} \cup \mathcal{N}^{1.2}) \sqcap (\mathcal{M}^{2.2} \cup \mathcal{N}^{2.2}) \\ \Pi \\ (\mathcal{M}^{1.3} \cap \mathcal{N}^{1.3}) \sqcap (\mathcal{M}^{3.3} \cap \mathcal{N}^{3.3}) \end{array}$$

\sqcap : replication, Π : mediation

Transpositional structuration

$[\text{bif, id, id}] : \mathcal{M}^{(3,3)} \underset{\text{morphoFSM}}{\cup} \cup \cap^{(3,3)} \mathcal{N}^{(3,3)},$
type < 1.1., 2.1., 3.1; 2.2, 3.3 > :

$$\begin{array}{c} (\mathcal{M}^{1.1} \underset{\cup}{\cup} \mathcal{N}^{1.1}) \diamond (\mathcal{M}^{2.1} \cup \mathcal{N}^{2.1}) \diamond (\mathcal{M}^{3.1} \cap \mathcal{N}^{3.1}) \\ \Pi \\ (\mathcal{M}^{2.2} \cup \mathcal{N}^{2.2}) \\ \Pi \\ (\mathcal{M}^{3.3} \cap \mathcal{N}^{3.3}) \end{array}$$

\diamond : transposition, Π : mediation

Those topics of trans-contextural *interactivity* had been studied in previous papers and the results are directly applicable to polycontextural structurations of mediated languages and machines.

<http://memristors.memristics.com/Graphematics%20of%20Multisets/Graphematics%20of%20Multisets.html>

Criteria of mediation

Polycontextural distribution of machines is one aspect of dissemination, the other aspect is the *mediation* of the distributed machines. The most abstract approach is achieved by the fact that machines are mathematically defined as algebras and coalgebras. Algebras are naturally conceived as *hierarchies of operator-* and *operand-* systems. With the application of the *proemial* relation, a mediation of algebras is achieved as the chiasitic mechanism of the *deplacement* and *reversion* of the algebraic hierarchies.

A less abstract approach of mediation of finite state machines, i.e. difference machines, is accessible by the involvement of the initial start states and the final acceptance states of the machines. Hence, a *chiasm* between *initial* and *final* entities or constellation of distributed machines is defining their mediation. Mediation determines the range of combination of possible constellations of combinations. This range is specially critical to the combination of operations and the *reversion* of those operations.

Classical example of mediation

As well known, a crucial part of morphogrammatics as it was developped by Gunther in the '60s, contains the study of mediating and transforming the morphogrammatic patterns, i.e. the 15 basic morphograms of polycontextural logical operations into each other.

$$MG^{(3)} [mg10, mg10, mg10] = \left(\begin{array}{c|c|c} a & a & c \\ \hline a & b & b \\ \hline c & b & c \end{array} \right) \Longrightarrow \left(\begin{array}{c} [aaab] \\ \Pi_{1.2.0} \\ [bbbc] \\ \Pi_{1.2.3} \\ [aaac] \end{array} \right)$$

$$\text{with } \begin{cases} \text{fst}[aaab]_1 = \text{fst}[accc]_3, \\ \text{lst}[aaab]_1 = \text{fst}[bbbc]_2, \\ \text{lst}[bbbc]_2 = \text{lst}[aaac]_3. \end{cases}$$

The question of the *composability* of the 15 basic morphograms into a morphogrammatic compound is answered by the SML-function "*exmm*" (Morphogrammatik, p. 103).

Example for composition

Are the morphograms mg1, mg1 and mg2 composable? The answer is no.

- `exmm [mg 1, mg 1, mg 2];`

`val it = false : bool`

Are the morphograms mg15, mg2 and mg11 composable? The answer is yes.

- `exmm [mg 15, mg 2, mg 11];`

`val it = true : bool`

FC – abstraction

A reduction of complexity is naturally achieved with the reduction of the morphograms to the 'values' of its mediating points. This enables a classification of the sub-diagonals into the same, "C", or different, "F", sub-diagonals of the components of the whole morphogrammatic complex. Hence, $C = fst = lst$ and $F = fst \neq lst$ of a component.

Examples

- allFCs 3;

val it = [[C,C,C],[C,F,F],[F,F,C],[F,C,F],[F,F,F]] : fc list list

- allFCs 4;

val it =

[[C,C,C,C,C],[C,F,F,C,F,F],[F,F,C,F,C,F],[F,C,F,F,F,C],[C,C,C,F,F,F],
[C,F,F,F,C,C],[F,F,C,C,F,C],[F,C,F,C,C,F],[C,F,F,F,F,F],[F,F,C,F,F,F],
[F,F,F,F,F,C],[F,C,F,F,F,F],[F,F,F,F,C,F],[F,F,F,C,F,F],[F,F,F,F,F,F]]

: fc list list

- length(allFCs 4);

val it = 15 : int

Also well studied are the operations of the so called *reflector R* on morphogrammatic complexions.

$$FC \left(\begin{array}{c|c|c} a & a & c \\ \hline a & b & b \\ \hline c & b & c \end{array} \right) = [f, f, f]$$

$$FC \left(\begin{array}{c|c|c} a & b & c \\ \hline a & a & b \\ \hline c & b & c \end{array} \right) = [c, f, f]$$

$$FC \left(\begin{array}{c|c|c} a & b & c \\ \hline a & a & b \\ \hline a & a & a \end{array} \right) = [c, c, c]_1$$

$$FC \left(\begin{array}{c|c|c} a & b & c \\ \hline a & a & c \\ \hline a & a & a \end{array} \right) = [c, c, c]_2$$

$$[c, c, c]_1 =_{\text{polyseme}} [c, c, c]_2,$$

$$[c, c, c]_1 \neq_{\text{MS}} [c, c, c]_2$$

Polysemy and tabular representation

Distributed implication $c = [abaa]$ as a chain : $[c_1, c_2, c_3]$.

Tabular representation :

$$[c, c, c]_2 \implies [c_{1.1}, c_{3.2}, c_{3.3}]$$

$$[c, c, c]_1 \implies [c_{1.1}, c_{1.2}, c_{3.3}]$$

$[c_{1.1} \ c_{3.2} \ c_{3.3}]$	O_1	O_2	O_3
M_1	$\begin{matrix} c_{1.1} \\ 12 \\ 11 \end{matrix}$	-	-
M_2	-	-	$\begin{matrix} c_{3.2} \\ 13 \\ 11 \end{matrix}$
M_3	-	-	$\begin{matrix} c_{3.3} \\ 13 \\ 11 \end{matrix}$

$[c_{1.1} \ c_{1.2} \ c_{3.3}]$	O_1	O_2	O_3
M_1	$\begin{matrix} c_{1.1} \\ 12 \\ 11 \end{matrix}$	-	-
M_2	$\begin{matrix} c_{1.2} \\ 12 \\ 11 \end{matrix}$	-	-
M_3	-	-	$\begin{matrix} c_{3.3} \\ 13 \\ 11 \end{matrix}$

$O^{(3)} M^{(3)} = O_1(M_1), O_2(M_2), O_3(M_3)$:

- subsystems 3;

val it = [(1,[1,2]),(2,[2,3]),(3,[1,3])] : (int * int list) list

The operation “subsystems” is based on the concept of morphogrammatic sequences, i.e. on the sequential mediation of morphograms to a linear chain of morphograms. Hence, the distribution of the type “c” of morphograms over 3 places becomes [c,c,c]. Because there are different realizations of a type, a kind of *polysemy* is at work. (polysemy, Morphogrammatik, Chapter 8.4)

In contrast, and as an extension and further concretization of morphogrammatik, a *tabular* distribution of morphograms is introduced. With that, polysemy is resolved as a *mode of distribution* of a morphogram in the kenomic matrix.

Thus, the general unspecified example [c,c,c] for the distribution of the morphogram for logical implication in the matrix becomes: $[c_{1.1}, -, -; -, -, -; -, c_{3.2}; c_{3.3}]$ or $[c_{1.1}, c_{1.2}, -; -, -, -; -, -, c_{3.3}]$.

Tabular matrix

$O^{(3,3)} M^{(3,3)} = [O_1(M_1, M_2, M_3); O_2(M_1, M_2, M_3); O_3(M_1, M_2, M_3)]$:

- submatrices 3;

$[1, ((1,[1,2]), (2,[2,3]), (3,[1,3]))], [2, ((1,[1,2]), (2,[2,3]), (3,[1,3]))], [3, ((1,[1,2]), (2,[2,3]), (3,[1,3]))]$.

[1,
((1,[1,2]),
(2,[2,3]),
(3,[1,3]))],

[2,
((1,[1,2]),
(2,[2,3]),
(3,[1,3]))],

[3,
((1,[1,2]),
(2,[2,3]),
(3,[1,3]))].

Mediation

The topics and techniques of morphogrammatic de/compositions and reflector-transformations are directly applicable to the de-compositions of MorphoFSMs.

$$\text{MorphoFSM}^{(3,3)} \left(\begin{array}{c|c|c} a & a & c \\ \hline a & b & b \\ \hline c & b & c \end{array} \right) \Rightarrow \left(\begin{array}{c} \text{MorphoFSM}[aaab] \\ \Pi_{1.2.0} \\ \text{MorphoFSM}[bbbc] \\ \Pi_{1.2.3} \\ \text{MorphoFSM}[accc] \end{array} \right).$$

$$\text{FC} \left(\text{MorphoFSM}^{(3,3)} \left(\begin{array}{c|c|c} a & a & c \\ \hline a & b & b \\ \hline c & b & c \end{array} \right) \right) = \text{MorphoFSM}^{(3,3)} [f_{1.1}, f_{2.2}, f_{3.3}]$$

$$\text{FC} \left(\text{MorphoFSM}^{(3,3)} \left(\begin{array}{c|c|c} a & b & c \\ \hline a & a & b \\ \hline c & b & c \end{array} \right) \right) = \text{MorphoFSM}^{(3,3)} [c_{1.1}, f_{3.2}, f_{3.3}].$$

$$\text{FC} \left(\text{R2} \left(\text{MorphoFSM}^{(3,3)} [c_{1.1}, f_{3.2}, f_{3.3}] \right) \right) = \text{MorphoFSM}^{(3,3)} [f_{1.3}, f_{3.2}, c_{3.3}].$$

Combinatorics

The number of FCs for regular quadratic mediations (matrices) is given with the SML function `length(allFCs m)`, calculated by the formula for the Bell numbers. ¹⁶

m	<code>length(allFCs m)</code>
3	5
4	15
5	52
6	203
7	877

<http://oeis.org/A000110/list>

How many reflectors exist for regular matrices $n \times n$?

n	<code>length(RG n)</code>
3	7
4	63
5	1023
6	32767
7	2097151

The number series of `length(RG n)` is part of the Mersenne sequence $n \rightarrow (2^{n-1})$.

Further concretizations of the abstraction procedure are given with the *gh*- and *klor*-analysis (Morphogrammatik, Chapter 7).

GH-abstraction

Considering the side-diagonals of the special example of composed matrices with second = third element as “G” and second != third element as “H”, a new abstraction is introduced.

Examples

- `allGHs 3;`

`val it = [[G,G,G],[G,G,H],[G,H,G],[G,H,H],[H,G,G],[H,G,H],[H,H,G],[H,H,H]]`
: gh list list

- `allGHs 4;`

`val it =`
[[G,G,G,G,G,G],[G,G,G,G,G,H],[G,G,G,G,H,G],[G,G,G,H,H],[G,G,G,H,G,G],

```

[G,G,G,H,G,H],[G,G,G,H,H,G],[G,G,G,H,H,H],[G,G,H,G,G,G],[G,G,H,G,G,H],
[G,G,H,G,H,G],[G,G,H,G,H,H],[G,G,H,H,G,G],[G,G,H,H,G,H],[G,G,H,H,H,G],
[G,G,H,H,H,H],[G,H,G,G,G,G],[G,H,G,G,G,H],[G,H,G,G,H,G],[G,H,G,G,H,H],
[G,H,G,H,G,G],[G,H,G,H,G,H],[G,H,G,H,H,G],[G,H,G,H,H,H],[G,H,H,G,G,G],
[G,H,H,G,G,H],[G,H,H,G,H,G],[G,H,H,G,H,H],[G,H,H,H,G,G],[G,H,H,H,G,H],
[G,H,H,H,H,G],[G,H,H,H,H,H],[H,G,G,G,G,G],[H,G,G,G,G,H],[H,G,G,G,H,G],
[H,G,G,G,H,H],[H,G,G,H,G,G],[H,G,G,H,G,H],[H,G,G,H,H,G],[H,G,G,H,H,H],
[H,G,H,G,G,G],[H,G,H,G,G,H],[H,G,H,G,H,G],[H,G,H,G,H,H],[H,G,H,H,G,G],
[H,G,H,H,G,H],[H,G,H,H,H,G],[H,G,H,H,H,H],[H,H,G,G,G,G],[H,H,G,G,G,H],
[H,H,G,G,H,G],[H,H,G,G,H,H],[H,H,G,H,G,G],[H,H,G,H,G,H],[H,H,G,H,H,G],
[H,H,G,H,H,H],[H,H,H,G,G,G],[H,H,H,G,G,H],[H,H,H,G,H,G],[H,H,H,G,H,H],
[H,H,H,H,G,G],[H,H,H,H,G,H],[H,H,H,H,H,G],[H,H,H,H,H,H] : gh list list
- length(allGHs 4);
val it = 64 : int

```

KLOR-abstraction

A further concretization of the classification is achieved with:

$$Q_k = Q_f \cap Q_g$$

$$Q_l = Q_f \cap Q_h$$

$$Q_o = Q_c \cap Q_h$$

$$Q_r = Q_c \cap Q_g.$$

```

- allKLORs 3;

```

```

val it =

```

```

[[O,O,O],[O,O,R],[O,R,O],[O,R,R],[R,O,O],[R,O,R],[R,R,O],[R,R,R],[O,L,L],
 [O,L,K],[O,K,L],[O,K,K],[R,L,L],[R,L,K],[R,K,L],[R,K,K],[L,L,O],[L,L,R],
 [L,K,O],[L,K,R],[K,L,O],[K,L,R],[K,K,O],[K,K,R],[L,O,L],[L,O,K],[L,R,L],
 [L,R,K],[K,O,L],[K,O,K],[K,R,L],[K,R,K],[L,L,L],[L,L,K],[L,K,L],[L,K,K],
 [K,L,L],[K,L,K],[K,K,L],[K,K,K]] : klor list list

```

```

val it = 40 : int

```

```

- allKLORs 4;17

```

```

- length(allKLORs 4);

```

```

val it = 960 : int

```

In other words, the reduction techniques of reflector-morphogramatics that enable to handle “astronomic” complexity by structural reductions is easily applied to the morphogrammatic complexions of MorphoFSMs.

Some more information about the dissemination of *logical* systems at:

<http://memristors.memristics.com/Notes%20on%20Polycontextural%20Logics/Notes%20on%20Polycontextural%20Logics.pdf>

Some further general construction of *bifunctionality* and *dissemination* are available at:

<http://memristors.memristics.com/Polyverses/Polyverses.pdf>

Parallel compositions of FSAs

http://syrcoise.ispras.ru/2011/files/syrcoise11_submission_016.pdf

3.6.4. Mono- and polysemy

A full determination of the new morphogram (machine) has to take into account all the differences of the morphogram (machine). Hence, for $\text{length}(\text{morphogram}) = m$, $s(m) = \binom{m}{2}$ differences are defining the morphogram of length m . Otherwise, the morphogram isn't fully determined. But morphograms with just two elements might be written with less than the full range of differences.

$$[aabc]: (\varepsilon vv)(vv)(v)$$

$$[aaba]: (\varepsilon vv)(v\varepsilon)(\varepsilon)$$

Hence, two-element morphic automata might be treated as abstractions of classical automata. The difference of the definition of the strings (words, morphograms) still

remains. Classical automata are based on identity, morphic automata are based on kenogrammatic difference.

Therefore, the automaton M1 for [abbbb] is reducible to the equivalent automaton M1':

Automaton M1

accept [abbbb] M1

fn pos1 => (fn v1,4 => pos2)	≡ [ab] = start, with v1
pos2 => (fn v2,7 => pos1)	≡ [ab]
pos1 => (fn ε3,8,9,10 => pos1)	≡ [abb]
pos2 => (fn ε5,6 => pos2)	≡ [abbb]
pos1 => (fn ε3,8,9,10 => pos1)	≡ [abbbb] = final.

$M1(v_1, \epsilon_3) \Rightarrow M1(v_1, v_2, \epsilon_3)$ for two elements.

Automaton M1'

States: $\Sigma = \{v_i, \epsilon_j, i, j \in \mathbb{N}\}$

Positions: $\{\text{pos}_1, \text{pos}_2\}$

Initial: $\{\text{pos}_1\}$

Transitions:

$\text{pos}_1, v_1 \rightarrow \text{pos}_2$

$\text{pos}_2, v_2 \rightarrow \text{pos}_1$

$\text{pos}_1, \epsilon_3 \rightarrow \text{pos}_1$

Final: $\{\text{pos}_1\}$.

M1 is accepting : [abb].

3.6.5. Determinism and non-determinism

A FSA is deterministic, DFA, iff its runs are unique, otherwise it is called a non-deterministic FSA, i.e. NFA.

The DFA machine (A2) has different runs for the word (abbb):

$q_1, a, b \rightarrow q_1$

$q_1, b \rightarrow q_2$

$q_2, b \rightarrow q_2$.

Run one: $q_1, a \rightarrow q_1, b \rightarrow q_1, b \rightarrow q_2, b \rightarrow q_2 : (\text{abbb})$.

Run two: $q_1, a \rightarrow q_1, b \rightarrow q_1, b \rightarrow q_1, b \rightarrow q_1 : (\text{abbb})$.

Trivially, all depends on the self-loop " $q_1, a \rightarrow q_1, b \rightarrow q_1, b$ " which has two entries.

For morphic FSA the situation is quite different. The semiotic difference of the elements "a" and "b" are not relevant in this situation. Both are defining a self-application at the state q_1 . Therefore, they are difference-theoretically equivalent. As a consequence, the criterion for the distinction of deterministic and non-deterministic automata vanishes.

Hence, kenomic automata MorphFSM are neither deterministic nor non-deterministic.

This observation is not excluding iterated self-loops for morphic automata.

Morphic automaton for [abbb]:

[A2] = M1:

$q_1, v_1, 4 \rightarrow q_2$

$q_2, v_2 \rightarrow q_1$

$q_1, \epsilon_3 \rightarrow q_1$

$q_2, \epsilon_5, 6 \rightarrow q_2$.

[A2] has just one single run for the recognition of the morphogram [abbb] albeit there is a single and a double loop involved.

Because the enumeration of the differences in the matrix of the morphogram are slightly arbitrary, other runs are possible on the base of different enumerations of the differences of the runs.

3.6.6. Logic, Categories, FSM and MorphoFSM

FSM have a prominent application in logic and numerous realizations in the design of logical circuits.

Neither logical nor arithmetical circuits in the sense of the term are topics of morphogrammatic automata. This is a natural consequence for machines that don't have states and state-based transitions.

To keep the analogy alive, a fundamentally different kind of 'arithmetics' and 'logic' has to be introduced. This exactly was the project of Gotthard Gunther at the BCL in the '60s.

There is also no chance for micro-electronic realizations of morphic machines. Again, simulations don't become realizations. (Pattee)

A real chance for a very different kind of realizations seems to become accessible with the discovery/invention of memristors and the building of memristive systems.

It seems that the very crucial 'feature' of *retrograde* recursivity holds for both attempts: the *kenogrammatic* and the *memristic* concept and realization of iterability (Derrida, Gunther, Chua).

3.6.7. Diamond characterization

It is well known that finite state machines are adequately modeled by category-theoretical methods. The category PATH is mapping the transitions of finite state machine.

"For the technical definitions, again let $R \subset X \times X$ or (X, R) denote a (general) relation. We associate to it the following category denoted by $\text{PATH}(X, R)$ or just PATH for short, if no confusion can arise." (Pfalzgraf)

Despite the nice formal and diagrammatic analogy to the classical concept of finite state machines by which the concept of morphogrammatic machines had been modeled, there are in fact no paths involved, modeling the activity of morphic finite differentiation machines. Differences in the sense of diamond theory are ruled by jumps, bridges and bridging, and are constituting not categories but *saltatories* (jumpoids).

Hence a more sophisticated modeling and formalization is required that is able to establish the *interplay* between the category of finite state machines with its "flow of information" and the saltatory "enaction" of difference machines.

Finite state machines are ruled by the category PATH, morphic differentiation structurations are involved with the diamond-theoretic journeys JOURN. In a diamond framework, PATH and JOURN are complementary.

<http://www.thinkartlab.com/pkl/lola/Diamond%20Relations/Diamond%20Relations.html>

"What I proposed as diamonds at different places, are structures with very different laws compared to the laws of categories. That is, *diamonds*, which consist of a complementary interplay of *categories* and *saltatories*, are as categorical systems, identitive, commutative and associative in respect to their objects, morphisms and composition. Therefore, they are inheriting all the laws and methods from category theory.

In sharp contrast, *saltatories* as parts of diamonds, are ruled by differences, jumps (saltisitions) and jump-associativity, etc. Additionally, diamonds as such, are

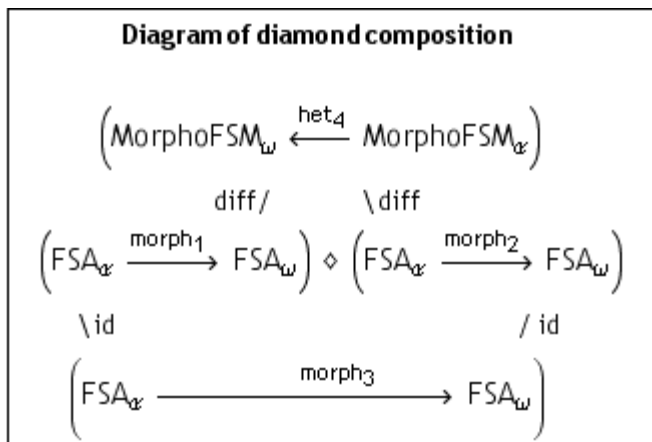
containing *bridges* and *bridging* rules between categories and saltatories.” (Kaehr, 30/01/2009)

<http://www.thinkartlab.com/pkl/lola/Interactivity.pdf>

A definition of finite state automata FSA implies/replays the *complementary* construction/instruction of morphic differentiation machines MorphoFSM, $\text{Diamond}(\text{FSA}, \text{MorphoFSM}) = \text{FSA}_w \diamond \text{FSA}_x \mid \mid \text{MorphoFSM}$.

This approach is focused on the *complementarity* of the autonomous types of machines, FSA and MorphoFSM, and is not involved in any derivations based on abstractions.

Explicit diagram of the diamond interplay of FSAs and MorphoFSM without hetero-morphic jumps.



Diamond formula with a hetero-morphic jump $\mid \mid$:

$\text{Diamond}(\text{FSA}, \text{MorphoFSM}) = (\text{FSA} \diamond \text{FSA}) \diamond \text{FSA} \mid \text{MorphoFSM} \mid \mid \text{MorphoFSM}$.

$\text{FSA}^{(4)} \diamond \text{FSA}^{(4)}$	N^1	$N^{2.3}$	$N^{4.5.6}$	$M^{7,8} \parallel N^{7,8}$
$M^{1.2.4}$	$M^1 \xrightarrow{\text{morph}_1} N^1$	$M^2 \xrightarrow{\text{morph}_2} N^2$	$M^4 \xrightarrow{\text{morph}_4} N^4$	$M^7 \xleftarrow{\text{het}_7} N^7$
$M^{3.5}$	-	$M^3 \xrightarrow{\text{morph}_3} N^3$	$M^5 \xrightarrow{\text{morph}_5} N^5$	$M^8 \xleftarrow{\text{het}_8} N^8$
M^6	-	-	$M^6 \xrightarrow{\text{morph}_6} N^6$	$M^9 \xleftarrow{\text{het}_9} N^9$

3.7. Further comparisons

3.7.1. Quotient automata of FSA and MorphoFSA

Are morphic automata not just quotient automata of classical automata? It could be thought that morphic automata are just classical automata over a morphic quotient structure of the general alphabet. As mentioned before, morphogrammatically, the sequence [abbb] and [baaa] are equivalent. Hence, belonging to the equivalence class Σ/morph .

But things seem to be more intricate.

Take an automaton and an equivalence relation of its words Σ^* then following properties hold.

For $\forall x, y \in \Sigma^*$ and $a \in \Sigma$:

(a) \equiv_A is an equivalence relation over Σ^* .

(b) $x \equiv_A y \implies \forall a. xa \equiv_A ya$.

(c) $x \equiv_A y \implies x \in L(A) \iff y \in L(A)$. $\mid \mid \mid \mid$

(d) \equiv_A is of finite index.

<http://www.tcs.tifr.res.in/~pandya/grad/aut06/lect2.pdf>

Example for the morphic situation

$[abbb] \equiv_{\text{trito}} [baaa] \implies \forall a. [abbb][a] \equiv_A [baaa][a]$.

Obviously, the equivalence relation doesn't hold. Therefore, the trito-equivalence relation is not *right congruent* for x, y and a .

$[abbba] \not\equiv_{\text{trito}} [baaaa]$.

There is a chance to save the relation with tritogrammatic monads: $[a] \in \Sigma/\equiv_A$.

$a = [a]$, $b = [a]$, $c = [a]$, etc.

$[abbb] \equiv_{\text{trito}} [baaa] \implies \exists [a]. [abbb][b] \equiv_A [baaa][a]$.

Definition:

For $\forall x, y \in \Sigma^*$ and $[a] \in \Sigma/\equiv_A$:

$x \equiv_A y \iff [x] \equiv_{\text{trito}} [y]$

(b') $x \equiv_{\text{trito}} y \implies \forall x, y \exists [a]. [x][a] \equiv_{\text{trito}} [y][a]$.

This construction is also working if $[a]$ is not a monad of Σ/\equiv_A but a language containing words with length $n \geq 2$ has properly to be adjusted. Without that, the proof of the existence of the equivalence relation, mediated by the third term, is disturbed.

For $n=2$, $w = \{[aa], [ab]\}$.

If $w_1 = [aa]$ then $w_1 \approx w_3 = [bb]$

$x \equiv_{\text{trito}} y \implies \forall x, y \exists [aa]. [x][aa] \equiv_{\text{trito}} [y][bb]$.

If $w_1 = [ab]$ then $w_1 \approx w_3 = [ba]$

$x \equiv_{\text{trito}} y \implies \forall x, y \exists [ab]. [x][ab] \equiv_{\text{trito}} [y][ba]$.

$x = [abbb] \equiv_{\text{trito}} y = [baaa] \implies \forall x, y \exists [ab]. [abbb][ab] \equiv_{\text{trito}} [baaa][ba]$.

$n=3$:

If $w_1 = [abc]$ then $w_1 \approx w_3 = [bac]$

$x \equiv_{\text{trito}} y \implies \forall x, y \exists [abc]. [x][abc] \equiv_{\text{trito}} [y][bac]$.

$x = [abbb] \equiv_{\text{trito}} y = [baaa] \implies \forall x, y \exists [abc]. [abbb][abc] \equiv_{\text{trito}} [baaa][bac]$

If $w_1 = [abc]$, then $w_1 \approx w_3 = [bca]$

$x \equiv_{\text{trito}} y \implies \forall x, y \exists [abc]. [x][abc] \equiv_{\text{trito}} [y][bca]$.

$x = [abbb] \equiv_{\text{trito}} y = [baaa] \implies \forall x, y \exists [abc]. [abbb][abc] \not\equiv_{\text{trito}} [baaa][bca]$.

On the other hand it has to be recalled that the concatenation operation in trito-languages is not unique. There are $AG(kseq)+1$ different concatenation operations of a word with a monad.

Hence $[abbb][a] = \{[abbba], [abbbb], [abbbc]\}$, and

$[baaa][b] = \{[baaaa], [baaab], [baaac]\}$.

Equivalence:

$[abbba] \equiv_{\text{trito}} [baaab]$,

$[abbbb] \equiv_{\text{trito}} [baaaa]$,

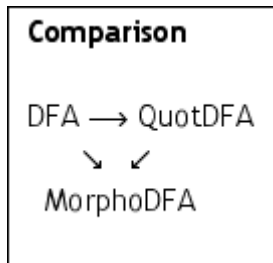
$[abbbc] \equiv_{\text{trito}} [baaac]$.

But, $[abbba] \not\equiv_{\text{trito}} [baaac]$, etc.

As a consequence, MorphoFSA are not quotients of FSA: $\text{MorphoFSA} \neq \text{FSA}/\equiv_{\text{trito}}$.

3.7.2. General comparison of automata

Those observations are leading naturally to an interesting comparison between the basic concepts of DFA, QuotDFA and MorphoDFA.



DFA = A = (Q, Σ, δ, q₀, F).

QuotDFA = A/~ := (Q', Σ, δ', [q₀], F')

with

Q' = {[p] | p ∈ Q}

δ' ([p], a) = [δ(p,a)]

F' = {[f] | f ∈ F}.

Congruence

p ~ q => ∀ a ∈ Σ. δ(p,a) ~ δ(q,a).

L(A/~) = L(A).

MorphoDFA = [A] = (Q_{trito}, Σ_{trito}, δ_{trito}, q⁰_{trito}, F_{trito})

with

Q_{trito} = {Q | AG(sign(morphogram))},

Σ_{trito} = {sign | sign ∈ Stirling2(*, Σ),

δ_{trito} = {ε, v | EN(morphogram), N},

q⁰_{trito} = q⁰ ⊆ Σ_{trito},

F_{trito} = F ⊆ Σ_{trito}.

3.7.3. Cellular automata based on differences

A further step towards a purely *difference*-theoretic approach to kenomic cellular automata has to consider both, the head and the result of the transition, as *differences*. This proper approach to a morphogrammatic notation is technically complicating the applications of the rules of kenoCA. But it is preserving the '*history-dependence*' of its transition rules.

ENstructure(R2): $\begin{bmatrix} \varepsilon & v & \varepsilon \\ v & \varepsilon \\ v \end{bmatrix}$: ENtoKS(R2) = [aab; a], ENstructure(R7): $\begin{bmatrix} \varepsilon & v & v \\ v & v \\ \varepsilon \end{bmatrix}$: ENtoKS(R7) = [aab;b]

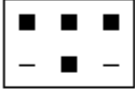
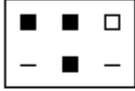
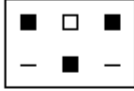
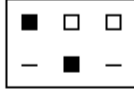

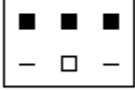
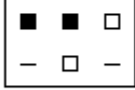
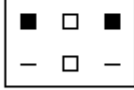
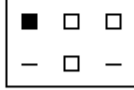





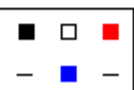
Combinatorics of trito – rules

$$\text{rules} \left(\text{CA}^{(4,4)} \right) = \sum_{k=1}^4 \text{Sn2}(4, k) = 1 + 7 + 6 + 1 = 15$$

Wolfram analogs of 2³

$$\text{rules} \left(\text{CA}^{(4,2)} \right) = \sum_{k=1}^2 \text{Sn2}(4, k) = 1 + 7 = 8$$

System of elementary kenomic cellular automata rules in trito – normal form

R1 	R2 	R3 	R4 	R5 
R6 	R7 	R8 	R9 	R10 
R11 	R12 	R13 	R14 	R15 

Differentiation mode of presentation of the basic cellular automata rules for morphic CA.

ENstructure(rule) = differentiation-rule R_{diff} ;

$$\text{numeration} \left(\begin{array}{ccc} & 3 & \\ \blacksquare 1 & \blacksquare 2 & \square \\ 4 & 5 & 6 \\ & \blacksquare & \end{array} \right) = \left(\begin{array}{c|c|c} (e, v) 1 & (e, v) 2 & (e, v) 4 \\ \hline - & (e, v) 3 & (e, v) 5 \\ \hline \mathbf{R} & - & (e, v) 6 \end{array} \right)$$

Examples

$$\text{ENstructure} \left(\begin{array}{ccc} & \blacksquare & \blacksquare & \square \\ \text{R2} & - & \blacksquare & - \end{array} \right) = \left(\begin{array}{c|c|c} e1 & v2 & e4 \\ \hline - & v3 & e5 \\ \hline \mathbf{R2} & - & v6 \end{array} \right);$$

$$\left(\begin{array}{c|c|c} e1 & v2 & e4 \\ \hline - & v3 & e5 \\ \hline \mathbf{R2} & - & v6 \end{array} \right) : \frac{e1 \mid v2}{- \mid v3} \Rightarrow_{R2} \frac{e1 \mid v2 \mid e4}{- \mid v3 \mid e5} \Rightarrow_{R2} \frac{e1 \mid v2 \mid e4}{\mathbf{R2} \mid - \mid v6}$$

$$\text{ENToKS} (e1 \mid v2 \mid v3) \Rightarrow_{R6} \text{ENToKS} (e4 \mid e5 \mid v6) = R2 : [\blacksquare \mid \blacksquare \mid \square] \Rightarrow [- \mid \blacksquare \mid -].$$

The property of ‘history-dependence’ becomes more obvious with the case of rules with more than two kenomic “states”:

$$\text{ENstructure} \left(\begin{array}{ccc} & \blacksquare & \square & \color{red}{\square} \\ \text{R15} & - & \color{blue}{\square} & - \end{array} \right) = \left(\begin{array}{c|c|c} v1 & v2 & v4 \\ \hline - & v3 & v5 \\ \hline \mathbf{R15} & - & v6 \end{array} \right).$$

$$\text{ENToKS} (v1 \mid v2 \mid v3) \Rightarrow_{R15} \text{ENToKS} (v4 \mid v5 \mid v6) = R15 : [\blacksquare \mid \square \mid \color{red}{\square}] \Rightarrow [- \mid \color{blue}{\square} \mid -].$$

System of elementary kenomic cellular automata rules in trito – differentiation form

$\begin{pmatrix} e1 & e2 & e4 \\ - & e3 & e5 \\ \mathbf{R1} & - & e6 \end{pmatrix}$	$\begin{pmatrix} e1 & v2 & e4 \\ - & v3 & e5 \\ \mathbf{R2} & - & v6 \end{pmatrix}$	$\begin{pmatrix} v1 & e2 & e4 \\ - & v3 & v5 \\ \mathbf{R3} & - & e6 \end{pmatrix}$	$\begin{pmatrix} v1 & v2 & e4 \\ - & e3 & v5 \\ \mathbf{R4} & - & v6 \end{pmatrix}$	$\begin{pmatrix} v1 & v2 & e4 \\ - & v3 & v5 \\ \mathbf{R5} & - & v6 \end{pmatrix}$
$\begin{pmatrix} e1 & e2 & v4 \\ - & e3 & v5 \\ \mathbf{R6} & - & v6 \end{pmatrix}$	$\begin{pmatrix} e1 & v2 & v4 \\ - & v3 & v5 \\ \mathbf{R7} & - & e6 \end{pmatrix}$	$\begin{pmatrix} v1 & v2 & v4 \\ - & e3 & e5 \\ \mathbf{R8} & - & v6 \end{pmatrix}$	$\begin{pmatrix} v1 & v2 & v4 \\ - & e3 & e5 \\ \mathbf{R9} & - & e6 \end{pmatrix}$	$\begin{pmatrix} v1 & v2 & v4 \\ - & v3 & e5 \\ \mathbf{R10} & - & v6 \end{pmatrix}$
$\begin{pmatrix} e1 & v2 & v4 \\ - & v3 & v5 \\ \mathbf{R11} & - & v6 \end{pmatrix}$	$\begin{pmatrix} v1 & e2 & v4 \\ - & v3 & v5 \\ \mathbf{R12} & - & v6 \end{pmatrix}$	$\begin{pmatrix} v1 & v2 & v4 \\ - & e3 & v5 \\ \mathbf{R13} & - & e6 \end{pmatrix}$	$\begin{pmatrix} v1 & v2 & v4 \\ - & v3 & v5 \\ \mathbf{R14} & - & e6 \end{pmatrix}$	$\begin{pmatrix} v1 & v2 & v4 \\ - & v3 & v5 \\ \mathbf{R15} & - & v6 \end{pmatrix}$

<http://memristors.memristics.com/CA-Overview/Short%20Overview%20of%20Cellular%20Automata.pdf>

3.8. Classical machines with input and output

3.8.1. Mealy Machine

"JFLAP defines a Mealy machine M as the sextuple $M = (Q, \Sigma, \Gamma, \delta, \omega, q_s)$ where

Q is a finite set of states $\{q_i \mid i \text{ is a nonnegative integer}\}$

Σ is the finite *input* alphabet

Γ is the finite *output* alphabet

δ is the *transition* function, $\delta : Q \times \Sigma \rightarrow Q$

ω denotes the *output* function, $\omega : Q \times \Sigma \rightarrow \Gamma$

q_s (is a member of Q) is the *initial* state

Mealy machines are different than Moore machines in the output function, ω . In a Mealy machine, output is produced by its transitions, while in a Moore machine, output is produced by its states."



"Instead of accepting or rejecting input, a Mealy machine produces output from an input string."

<http://www.jflap.org/tutorial/>

3.8.2. Moore Machine

"JFLAP defines a Moore machine M as the sextuple $M = (Q, \Sigma, \Gamma, \delta, \omega, q_s)$ where

Q is a finite set of states $\{q_i \mid i \text{ is a nonnegative integer}\}$

Σ is the finite *input* alphabet

Γ is the finite *output* alphabet

δ is the transition function, $\delta : Q \times \Sigma \rightarrow Q$

ω denotes the output function, $\omega : Q \rightarrow \Gamma$

q_s (is a member of Q) is the initial state."

"Moore machines are different than Mealy machines in the output function, ω . In a Moore machine, output is produced by its states, while in a Mealy machine, output is produced by its transitions."

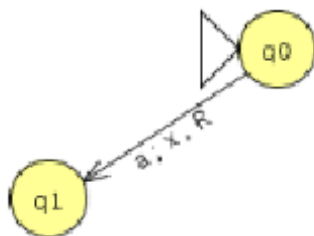
3.8.3. Turing Machines

"JFLAP defines a Turing Machine M as the septuple $M = (Q, \Sigma, \Gamma, \delta, q_s, O, F)$ where

Q is the set of *internal* states $\{q_i \mid i \text{ is a nonnegative integer}\}$

Σ is the *input* alphabet
 Γ is the finite set of symbols in the *tape* alphabet
 δ is the *transition* function
 S is $Q \times \Gamma^n \rightarrow \text{subset of } Q \times \Gamma^n \times \{L, S, R\}$
 O is the blank symbol.
 q_s (is member of Q) is the *initial* state
 F (is a subset of Q) is the set of *final* states.”

“If the head is under an “a” and the machine is in state “q0”, then replace the “a” with an “x” and move the head to the right. When done adding input, the area between q0 and q1 should resemble the example below.”



New symbol

(* The status of the Turing machine is a 4-ple
 * (*state, left_part, curr_char, right_part*)
 * A Turing_program (transition-function) is a list of transition_rules, each having the form
 * (*curr_state,curr_symbol,new_state,new_symbol*)
 * where 'new_symbol' may be any symbol of the alphabet plus Move_left and Move_right
 *)

<http://www.youtube.com/watch?v=lkYhfk4X47c>

For a classical machine, the *new* symbol is an arbitrary element of the alphabet of signs. The alphabet is stable, it might be finite or infinite. But it is not changing during the operation.

Morphic machines are not alphabet-based machines but depend on the actions of the machine. Therefore, the new symbol is new only in respect of the produced symbols by the actions and not in respect of a pre-given alphabet.

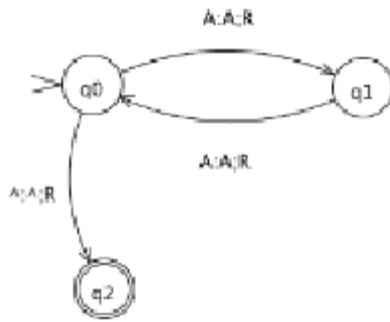
Limitation of the modeling

As mentioned before, the introduction of the differentiation machines is just a first step of a deconstruction of symbolic machines. The presented applications of morphogrammatrics onto symbolic machines is not yet considering the necessity of a deconstruction of further features of the symbolic machines, like the structure of *stacks* and *tapes*.

As much as morphograms are not properly understood as sequences, the read write actions on morphograms have to be adjusted to the more tabular and holistic situations of morphic machines.

3.8.4. Examples

morphoTM-A(even)



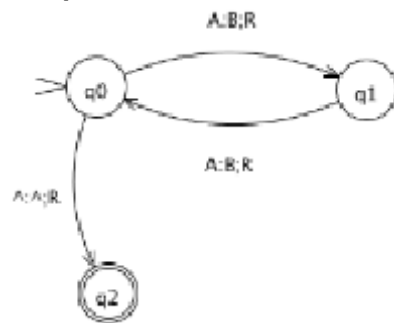
read "A": write "A"; goto R

run= AAAA => AAAA => AAAA => AAAA => ^: accepted

Transition rules

q0; A : A; R, q1
q1; A : A; R, q0
q0; ∅ : ∅; R, q2

morphoTM-AAAA2BBBB



Explanation

read "A": write "B"; goto R.

read "A" write "A"; goto R = acceptance state "q0".

Transition rules

q0; A : B; R, q1
q1; A : B; R, q0
q0; ∅ : ∅; R, q2

Linearized notation, plus EN-structure.

run: [AAAA] to [BBBB]

run= [AAAA] => [BAAA] => [BABA] => [BBBA] => [BBBB] => ^:
accepted.

EN: $\begin{pmatrix} e1 & e2 & e4 \\ e3 & e5 & - \\ e6 & - & - \end{pmatrix} \begin{pmatrix} v1 & v2 & v4 \\ e3 & e5 & - \\ e6 & - & - \end{pmatrix} \begin{pmatrix} v1 & e2 & v4 \\ v3 & v5 & - \\ e6 & - & - \end{pmatrix} \begin{pmatrix} e1 & e2 & v4 \\ e3 & v5 & - \\ v6 & - & - \end{pmatrix} \begin{pmatrix} e1 & e2 & e4 \\ e3 & e5 & - \\ e6 & - & - \end{pmatrix}$

Hence, the morphoTM transforms [AAAA] into [BBBB] with

EN[AAAA] = EN[BBBB], therefore [AAAA] =_{MG} [BBBB].

It might be said that the morphoTM is transforming the morphogram [AAAA] into itself by changing its semiotic appearance from [AAAA] to [BBBB].

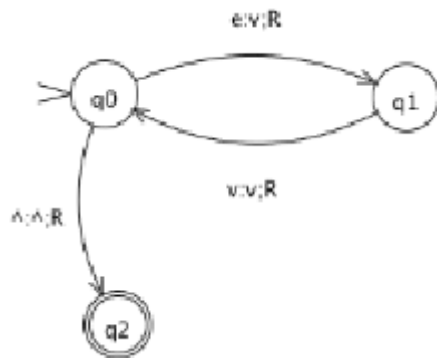
The chain "[AAAA] => [BAAA] => [BBAA] => [BBBA] => [BBBB] =_{MG} [AAAA]" is self-applicative:

morphoTM([AAAA]) =_{MG} [AAAA].

On more turn:

run= [AAAA] => [BAAA] => [BABA] => [BBBA] => [BBBB] =>
 [BBBB] => [ABBB] => [ABAB] => [AAAB] => [AAAA] => ^: accepted.

morphoTM-e-v



EN-

run: $\begin{pmatrix} \mathbf{e1} & \mathbf{e2} & \mathbf{e4} \\ \mathbf{e3} & \mathbf{e5} & - \\ \mathbf{e6} & - & - \end{pmatrix} \Rightarrow \begin{pmatrix} \mathbf{v1} & \mathbf{v2} & \mathbf{v4} \\ \mathbf{e3} & \mathbf{e5} & - \\ \mathbf{e6} & - & - \end{pmatrix} \Rightarrow \begin{pmatrix} \mathbf{v1} & \mathbf{v2} & \mathbf{v4} \\ \mathbf{v3} & \mathbf{e5} & - \\ \mathbf{v6} & - & - \end{pmatrix} \Rightarrow \begin{pmatrix} \mathbf{v1} & \mathbf{v2} & \mathbf{v4} \\ \mathbf{v3} & \mathbf{v5} & - \\ \mathbf{v6} & - & - \end{pmatrix} \Rightarrow$

$\begin{pmatrix} \mathbf{v1} & \mathbf{v2} & \mathbf{v4} \\ \mathbf{v3} & \mathbf{v5} & - \\ \mathbf{v6} & - & - \end{pmatrix}$

run= [AAAA] => [BAAA] => [BACA] => [BACD] => [ABCD]
] => ^: accepted

Explanation

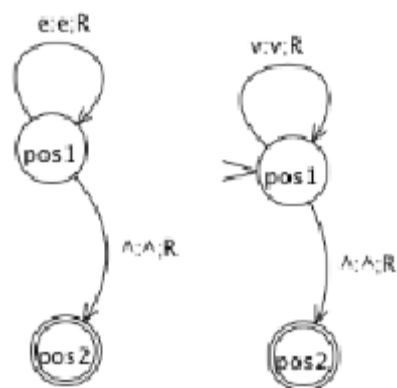
read "e": write "v"; goto R.

read "v": write "v"; goto R.

EN-notation, plus linearized morphogram in trito-normal form (tnf) with [AAAA] to [ABCD].

run: [AAAA] to [ABCD].

Elementary morphoTMs for iteration and accretion



morphoTM-iteration

EN-run: $(\mathbf{e1}) \Rightarrow \begin{pmatrix} \mathbf{e1} & \mathbf{e2} \\ \mathbf{e3} & - \end{pmatrix} \Rightarrow \begin{pmatrix} \mathbf{e1} & \mathbf{e2} & \mathbf{e4} \\ \mathbf{e3} & \mathbf{e5} & - \\ \mathbf{e6} & - & - \end{pmatrix} \Rightarrow \begin{pmatrix} \mathbf{e1} & \mathbf{e2} & \mathbf{e4} & \mathbf{e7} \\ \mathbf{e3} & \mathbf{e5} & \mathbf{e8} & - \\ \mathbf{e6} & \mathbf{e9} & - & - \\ \mathbf{e10} & - & - & - \end{pmatrix}$

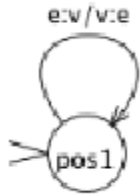
LIN-run: [AA] => [AAA] => [AAAA] => [AAAAA] => ^: accepted

morphoTM-accretion

$$\text{EN-run} : (v_1) \Rightarrow \begin{pmatrix} v_1 & v_2 \\ v_3 & - \end{pmatrix} \Rightarrow \begin{pmatrix} v_1 & v_2 & v_4 \\ v_3 & v_5 & - \\ v_6 & - & - \end{pmatrix} \Rightarrow \begin{pmatrix} v_1 & v_2 & v_4 & v_7 \\ v_3 & v_5 & v_8 & - \\ v_6 & v_9 & - & - \\ v_{10} & - & - & - \end{pmatrix}$$

run= [AB] => [ABC] => [ABCD] => [ACDE] => ^: accepted

morphTM-(v,e)



Accretion

[e,e,e] => [v,v,v]: [AAA]/[e,e,e] => [ABA]/[v,e,v] => [ABC]/[v,v,v].

Alternatively:

[e,e,e] => [v,v,v]: [AAA]/[e,e,e] => [BAA]/[v,v,e] => [BAC]/[v,v,v].

Inversion

[e,v,v] => [v,v,e]: [AAB]/[e,v,v] => [ABA]/[v,e,v] => [ABB]/[v,v,e].

- kref[1,1,2];

val it = [1,2,2] : int list

Mixed iterative and accretive repetitions

Transition rules

pos1; e : v; pos2
 pos2; v : e; pos1
 pos1; ∅ : ∅; pos3

run iteratively on {A, B}: AA AAB AABB AABBA AABBA AABBAAB ...

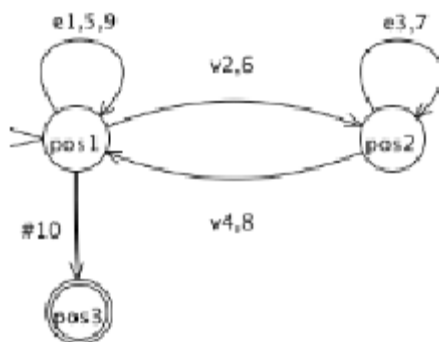
run accretively on {A, B, C, ...}: AA AAB AABB AABBC AABBC AABBCD AABCCDD ...

...

Even productions are, trivially, morphic palindromes.

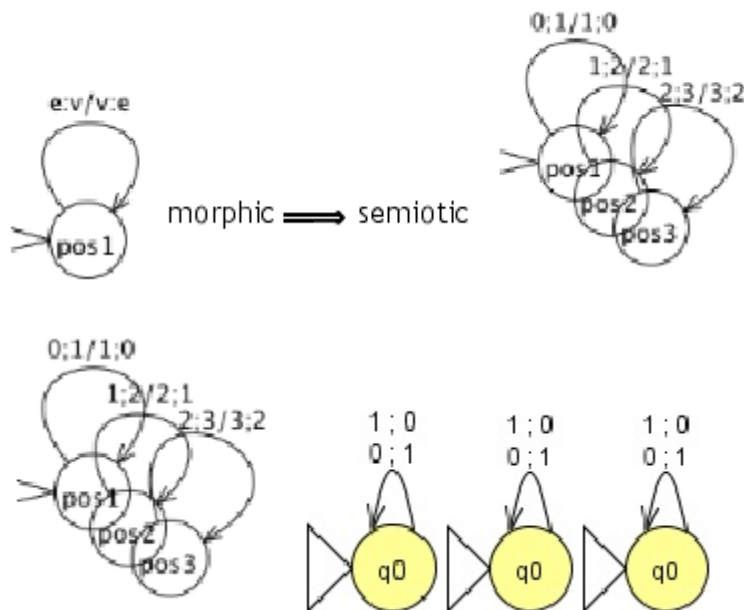
- ispalindrome [1,1,2,2,3,3,4,4,5,5];

val it = true : bool



3.8.5. Representations and combinations

The morphic Mealy Machine (e; v) has disjunct or mediated semiotic representations as Mealy (0, 1), Mealy (1, 2) and Mealy (2, 3) for $\Sigma = \{0, 1, 2, 3\}$.



Different types of symbolic machines (FSM, Mealy, Moore, Turing, Gurevitch, etc.) might be composed on the base of compounds of morphogrammatic machines. This may be called morphogrammatically based *parallelism* of semiotic machines. For the case of mediation, the conditions of mediation have to be accepted additionally to get the polycontextural types of negations. With that, some nice categorical braids of machines with Hamilton choreographies and mediated by interchanchability, enter the game.

Not in the alphabet

This message “*Not in the alphabet*”, doesn’t apply for morphic automata. Simply because morphic automata are not alphabet-based. On the other hand it means for morphic atomata that any identifiable sign (event) is recognized by its kind of differentiation. In other words, any differentiation is recognized as a “sign” (event) for calculation.

Hence, a classical automaton, defined by the alphabet {a,b} will not work for another alphabet, say { \oplus , \square , b}. It also will not recognize a word with (bbb...) if the rules are defined on (aaa...). What counts for morphic automata, again, are the differentiations, differences, distinctions and not the atomic symbols (data) perceived. Hence, again, morphic automata are information-independent; they are not processing information as their data.

<http://www.cs.duke.edu/~rodger/jflappapers/ChakrabortyX2011.pdf>

[http://krex.k-](http://krex.k-state.edu/dspace/bitstream/handle/2097/1401/SrinivasaAdityaUppu2009.pdf)

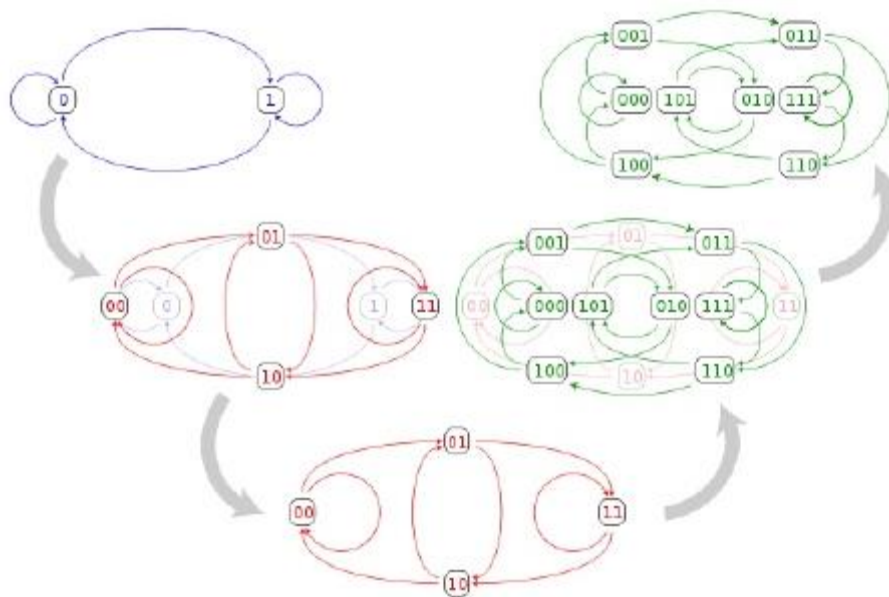
[state.edu/dspace/bitstream/handle/2097/1401/SrinivasaAdityaUppu2009.pdf](http://krex.k-state.edu/dspace/bitstream/handle/2097/1401/SrinivasaAdityaUppu2009.pdf)

3.9. Presentations of automata: transition tables and de Bruijn graphs

3.9.1. Transition tables

“If the set of states Q is finite, then the transition functions are commonly represented as state *transition tables*. The construction of all possible transitions driven by strings in the free group has a graphical depiction as *de Bruijn graphs*.” (WiKi, Semiautomata)

3.9.2. de Bruijn graphs for FSM



4. Critical questions

4.1. Are morphic FSAs Finite State Automata at all?

This proposal tried to sketch the idea of a morphogrammatic *analogon* to the semiotic or symbolic concept of FSAs and others. At the end of the journey of analogization it might turn out that non of the definitorial constituents of those machines where the journey started could be covered by the morphogrammatic approach to abstract machines.

In fact, morphoSFA have neither an *initial* nor a *final* state. In fact, they don't have states neither. They are not really feed by words of a regular language. They don't begin and also don't stop. Their transitions are independent of the vocabulary, hence they are also not *transitions* in the sense of the definition.

They are differentiations, paradoxically differing and deferring the positions of the structuration that are defining the differences as constellations or "states" of the machine.

The opposite characterization to the classical concept might give a better insight into the definition and behavior of morphogrammatic machines.

Instead of a defined start, like for FSA, morphic machines don't have a start. What we know about the behavior of the machine is depending on the point of view of an observer. An observation might take place and a beginning might be postulated. Any description of the behavior of the machine has to distinguish at least two possibilities of description: An *internal* and an *external* position of an observation.

An *external* observation might be closer connected with the point of view of classical automata theory and their concepts and apparatus. From there, the analogy and deconstruction might take place.

An *internal* description has to be aware of the non-conventional feature of the morphic automaton.

This approach might be supported by the well known 'experimental' intervention with automata and the co-algebraic structures involved.

Some lessons could be learned from the construction and application of other morphogrammatic systems and ‘machines’. It seems, that the morphic approach to *cellular automata* is still a novelty and worth to be studied.

4.2. Is there any use for morphic automata?

The usefulness of classical machine models like FSA, DFA, Mealy and Moore and Turing Machines, and many others, for computation, linguistics, modal logics and AI is well known, established, proven and documented. A further elaboration shall consider omega-languages and Büchi-automata in comparison to MorphoAutomata.

It is also well known that such automata concepts had been crucial for the development of modern theoretical linguistics. Noam Chomsky’s hierarchies are still governing the field.

On the other hand, it is not well known and only vaguely understood that the difference-theoretical approach to semiotics and linguistics of Ferdinand de Saussure might uncover structures and processes, i.e. structurations, that are closer to the functioning of language than the Leibniz-Chomsky paradigm, founded by the concept of abstract calculi, based on atomic signs, concatenation/substitution and linearity, could be. Obviously, de Saussure’s approach doesn’t fit into the Leibniz-Chomsky paradigm of computation.

Dealing with differences, and differences only, in a system of differences, where the loci of the differences in a complexion are themselves distinguished by differences in the system of differences, hence, self-referentially and classically paradoxical, determines the ‘value’ of the difference, might get a fundamentally new and interesting conceptualization, ‘formalization’ and programming towards a determination of the “values” of differences by morphogrammatics and morphic machines.

De Saussure wasn’t well recognized by the academic linguists, especially by the German school, and was then later successfully denied by the international Chomsky movement of generative linguistics.

"In language there are only differences. Even more important: a difference generally implies positive terms between which the difference is set up; but in language there are only differences without positive terms." F. de Saussure

Jaques Derrida discovered the deep difference-theoretical endeavour of de Saussure’s semiotics (sémiologie), not just for a theory of language but for an understanding of thinking at all. This post-philosophical approach got some recognition and determined the international movements of deconstructionism and deconstructivism.

Unfortunately, despite the radical insight into a pre-logical structure of de Saussure’s understanding of differences and system, *différance*, any attempts to connect this movement with more formal and operative achievements had not only been denied but harshly criticized, and institutionally killed.

Today, it could be a chance to begin to study this promising approach again. Might be with the help of morphogrammatics and morphogrammatic automata as formal and inspirational models.

At least, this could be one answer to the question: What are difference-based automata for?

Morphic automata, designed and understood as closed automata without input nor output in the strict sense are also giving some operational help to understand Humberto Maturana’s concept of *autopoiesis*. Despite the fact that morphic automata are just in their very beginning, morphic automata should nevertheless be contrasted

with the classical, first- and second order cybernetic approaches, to a theory of living systems.

Additional approaches: Peirce versus de Saussure

"Final summary: The Saussurean dyadic sign model can be mapped on 48 dyadic sign models as 3×3 sub-matrices in 4 contextures, based on the 3-adic Peircean sign model." (A. Toth)

Alfred Toth, The Saussurean sign model and its formal representation

<http://www.mathematical-semiotics.com/pdf/Saussure.pdf>

Object theory

Freud's difference of "Wortvorstellung" and "Sachvorstellung".

The rationality of the "Wortvorstellung" in its logical form is covered by the 'propositions' (apophansis) of two-valued logic. The rationality of the Sachvorstellung is not logical at all but is covered by transformation laws (Umformungsgesetze) of morphogrammatics. (Kaehr, Mitterauer)

"Die Sache selbst", the object as such, is ruled by differentiations, the handling of the notions of the object is ruled by the laws of representation.

"Was wir die unbewußte Objektvorstellung heißen, zerlegt sich uns in die 'Wortvorstellung' und in die 'Sachvorstellung', die in der Besetzung, wenn nicht der direkten Sacherinnerungsbilder, doch entfernterer und von innen abgeleiteter Erinnerungsspuren besteht. Mit einem Male glauben wir nun zu wissen, wodurch sich eine bewußte Vorstellung von einer unbewußten unterscheidet. Die beiden sind nicht, wie wir gemeint haben, verschiedene Niederschriften desselben Inhaltes an verschiedenen psychischen Orten, auch nicht verschiedene funktionelle Besetzungszustände an demselben Orte, sondern die bewußte Vorstellung umfaßt die 'Sachvorstellung' plus der zugehörigen 'Wortvorstellung', die unbewußte ist die Sachvorstellung allein." (S. Freud)

<http://www.gleichsatz.de/b-u-t/spdk/freud.html>

Abstractions versus differences

Abstractions to define *quotient automata*, or general quotient structures, are build over "positive terms" of an algebraic structure, i.e. a system. Such an abstraction applies over a relational system (algebra), and relations are holding between "positive terms". Mathematically, a relation is introduced as a set of a Cartesian product, $\text{Rel} \subseteq \text{Set} \times \text{Set}$, for binary relations, with positive elements, $\text{Pos} \in \text{Set}$. Hence, quotient automata are *derived* as abstractions over their relational structure (algebra) and are not defined by differences building differentiations of different actions, behaviors or events.

Differentiations, differences and distinctions in the sense of morphogrammatics and their interactional play are defining elements of relations, sets and operations as special, frozen, activities of differentiations.

De Saussure

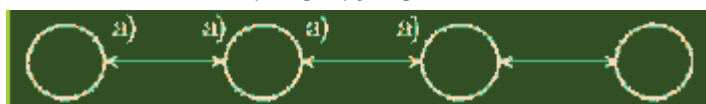
But the paradox is that: In the language, there are only differences, without positive terms. That is the paradoxical truth. At least, there are only differences if you are speaking either of meanings, or of signified or signifying elements.

"Strictly speaking there are no signs but differences between signs.

"There are only differences; no positive terms at all.

Here I am speaking of a difference in the signifying element.

The mechanism of signifying elements is based on differences."



"At first sight, no relation between the a) and the b) arrows. The value of a word will be the result only of the coexistence of the different terms. The value is the counterpart of the coexisting terms. How does that come to be confused with the counterpart of the auditory image?"

"In a language, as in every other semiological system, what distinguishes a sign is what constitutes it".

Ferdinand de Saussure (1910), Third Course of Lectures on General Linguistics
<http://www.marxists.org/reference/subject/philosophy/works/fr/saussure.htm>

In the language itself, there are only differences. Even more important than that is the fact that although in general a difference presupposes positive terms between which the difference holds, in language there are only differences, and no positive terms. Whether we take the signification or the signal, the language includes neither ideas nor sounds existing prior to the linguistic system, but only conceptual and phonetic differences arising out of that system. In a sign, what matters more than any idea or sound associated with it is what other sounds surround it (Course in General Linguistics 166).

<http://semioticsoflaw.com/site/derrida.php>

" Dans la langue, comme dans tout système sémiologique, ce qui distingue un signe, voilà tout ce qui le constitue. C'est la différence qui fait le caractère, comme elle fait la valeur et l'unité. " (p.168.)

" la langue est pour ainsi dire une algèbre qui n'aurait que des termes complexes " (p.168).

Manuel Gustavo Isaac, *Les paradoxes de l'arbitraire. Le négatif, la différence, l'opposition dans le signe saussurien.*

"Le paradoxe est double : premièrement, la sémiologie saussurienne est en contradiction avec le bon sens extensionnel de la *théorie des ensembles* définissant une relation comme un sous-ensemble d'un produit cartésien ($R \subseteq a \times a$), donc par ses éléments ; deuxièmement, parce qu'elle est *non-extensionnelle* et dérive les unités sémiotiques d'une relation d'inégalité (négativité, différence, oppositivité), la sémiologie exige une caractérisation intensionnelle de la négation. Comme l'abolition d'un paradoxe exige un changement de perspective sur les principes, on modifie le système des 'axiomes' sémiologiques en inversant ses règles de dérivation : l'arbitraire n'est plus principe, il a une raison. Passer de l'arbitraire comme principe au principe de l'arbitraire, autrement dit le renverser par le biais de l'analyse de ses trois notions cardinales, implique de le motiver. C'est là le paradoxe."

"L'objet linguistique est complexe.

<http://www.rifl.unical.it/articoli/rifl032010/010isaac.pdf>

Derrida/Searle

"Every concept is necessarily and essential inscribed in a chain or a system, within which it refers to another and to other concepts by the systematic play of differences. Such a play, then--difference is no longer simply a concept, but the possibility of conceptuality." (Derrida)

Thinking just the 'obvious' surface structure of thinking and writing and escaping prominently tedious "deep-structure" analysis is still a dominant strategy in established contemporary philosophy. Searle's surface argument, could easily be radicalized by a surface-understanding of computer programming languages, and obviously with a reasonable reference to Chomsky too. It seems not easy to grasp that formal languages are faithfully realizing the atomistic and linear structure of

phonological language with its proper hierarchy of the dominant dichotomy of operator and operand.

"On Derrida's account, however, it is essential not only to Husserl, but to philosophy, and indeed to "the history of the world during an entire epoch," including the present, that speech should be mistakenly privileged over writing. If Derrida's claim were to be taken at its face value, I believe that a contrary argument could be given equal or even greater plausibility.

"From the medieval development of Aristotle's logic through Leibniz's Characteristica Universalis through Frege and Russell and up to the present development of symbolic logic, it could be argued that exactly the reverse is the case; that by emphasizing logic and rationality, philosophers have tended to emphasize written language as the more perspicuous vehicle of logical relations.

"Indeed, as far as the present era in philosophy is concerned, it wasn't until the 1950s that serious claims were made on behalf of the ordinary spoken vernacular languages, against the written ideal symbolic languages of mathematical logic. [...]

"When Derrida makes sweeping claims about "the history of the world during an entire epoch," the effect is not so much apocalyptic as simply misinformed."

John Searle, "Reiterating the Differences: A Reply to Derrida", Glyph 1:198-208

Différance, differentiation

"Les différences sont donc <<produites>> -- différées -- par la différence. Mais qu'est-ce qui diffère ou qui diffère ? Autrement dit, qu'est-ce que la différence? Avec cette question nous atteignons un autre lieu et une autre ressource de la problématique. Qu'est-ce qui diffère? Qui diffère? Qu'est-ce que la différence?"

"Les deux valeurs apparemment différentes de la différence se nouent dans la théorie freudienne: le différer comme discernabilité, distinction, écart, diastème, espacement, et le différer comme détour, délai, réserve, temporisation."

<http://www.jacquesderrida.com.ar/frances/difference.htm>

Derrida, Cybernetics, Graphematics

"If the theory of cybernetics is by itself to oust all metaphysical concepts -- including the concepts of soul, of life, of value, of choice, of memory -- which until recently served to separate the machine from man, it must conserve the notion of writing, trace, written mark, or grapheme, until its own historico-metaphysical character is also exposed.

[...][E]ven before being determined as human... or nonhuman, the gramme -- or the grapheme -- would thus name the element. An element without simplicity. An element, whether it is understood as the medium or irreducible atom, of the arche-synthesis in general, of what one must forbid oneself to define within the system of oppositions in metaphysics, of what consequently one should not even call experience in general, that is to say the origin of meaning in general.]

(Jacques Derrida, Of Grammatology 9)

<http://fractalontology.wordpress.com/2008/01/16/systems-of-control-derrida-and-machines/>

Further differences

Rodolpe Gasché, "The Eclipse of Difference"

<http://www.dif-ferance.org/The%20Eclipse%20of%20Difference.pdf>

Maturana, Varela and Heinz von Foerster

Von Foerster's "Memory without records"

Maturana's Autopoiesis

Varela's Closure Thesis

The focus of Varela's Extended Calculus of Indication was on the *closure* of a formal system, hence self-referentiality and re-entry as attempts to conceptualize it 'beyond' logical paradoxes, and not on *structuration*. Structuration is the 'process' of building new structures as 'answers' to the interactions of the structured system, morphé, to the 'perturbations' by its environment.

<http://memristors.memristics.com/MorphoProgramming/Morphogrammatic%20Programming.html>

Some stuff has to be repeated millions of times until our brain gets hold of it.

4.2.1. Semiotics of palindromes and anagrammatics

"The idea of the palindrome is closely associated with the material and corporeal aspect of verbal signification. Animal images are used for symbolizing the palindromic processes of regression and circularity: the crab or cancer, and the snake biting its own tail (the gnostic image of Ouroboros).

"Likewise, the mirror metaphor has been applied to palindrome structures. Largely a visual phenomenon, the palindrome epitomizes the spatiality of language and scripture, something indicated already on the metaphorological plane of classical terminology: "running back again" (*palindromos*), "stepping back" (versus *retrogradus*) -- a temporal motion in space.

"Allowing for reversibility of the linear discourse, the palindrome represents the very idea of transformation and metamorphosis.

"Palindromic reversion is a device for breaking up the *linearity* of speech and, by implication, the irreversibility of time. Irreversibility "thematizes itself in the palindrome form by eating itself up" (a quotation from Oskar Pastior, the outstanding contemporary German palindrome poet).

"Sequentiality and causality of time and space are annihilated in the palindromic motion. Thus, the palindrome can be conceived of as a chronotope of revolution. ('chrono-topos': time-space)."

Erika Greber, PALINDROMON - ANAGRAMMATISMOS - REVOLUTIO: The Palindrome from the Perspective of Cultural Semiotics

<http://realchange.org/pal/semiotic.htm>

Christina Ljungberg, 'Damn mad': Palindromic figurations in literary narratives

"Palindromes are chiasmic figurations that arrest the habitual tempo-linear sequence of language and, in so doing, focus attention on the very act of signification. In narrative, they often prove pivotal for the overall structure of the text, going far beyond mere wordplay or verbal virtuosity. Because they can be read both backwards and forwards, palindromes emerge as *multilayered*, *multidirectional*, and *polytemporal* mappings reflecting the notorious instability of human lives, where the ever shifting present oscillates between the past and the future. In contemporary fiction, such palindromic vacillation becomes an iconic representation of temporal shifting, allowing us to discern the texture of temporality, not as abstractly conceived but as concretely lived and hence as innovatively performing an unstable present."

<http://benjamins.com/#catalog/books/ill.5.21lju>

All the emphasis made about the *temporality* of palindromes and chiasms is the result of *interpretations*, some hermeneutics and wild semiotic and culture-theoretical speculations. They might find some legitimation in the context of the whole corpus, texts, paintings, graphics, musical compositions, etc. but not at all in the *figure* of the chiasm and its derivation, the semiotic palindrome as such.

It seems that the difference-theoretical thematization, formalization and implementation of chiasms and palindromes by MorphoFSMs gives a much more comprehensive and convincing understanding of its 'deviant' logical structure.

Base Infinity

"Computer poetry is warfare carried out by other means, a warfare against conventionality and language that has become automatized. Strange as it seems, our finite state automata have become the poet's allies in this struggle, the long historical battle by which mankind pries into the surface of language to reveal its latent mysteries...", R.W. Bailey, Computer Poems (1973)

Sunday, December 11, 2011

```
###20_numbers_natural_numbers_as###
-=[without any theory]=-
( f=iA chiasmic derniere )
:=cl.[ear].ly=:
###__{stroke} in this particular__###

if [_categoricity_] =! Man Writing {
  presuppose {
    construction of nat0 as an initial
    concepts nothing anything left over
    is lost into conclusions
  }
}

} _in_pos_sEss_ion_id_en_tical_ {
{ words make zero itself or think they tell }
{ models of objects } { models of objects }
} {
an indefinite nat0!economy for example_
[
culturelles of disseminated representations
[_importance of numbers_]
[_independent of computation_]
[_mathematize'the users_]
[into conclusions ]
```

Generated from this Source:

www.thinkartlab.com/pkl/media/DERRIDA/DERRIDA.htm

Posted by Rollie Bollocks at 3:51 PM

Labels: codework, n-grams

<http://baseinfinity.blogspot.co.uk/2011/12/derridas-machines-machine.html>

Notes

1

Deconstructing beginnings

"Einerseits lassen sich Kenogrammsequenzen rekursiv konstruieren, wenn auch nur in Analogie zu semiotischen Systemen, fehlt ihnen doch ein echtes initiales Objekt. Sie haben somit eine Objekt-Struktur. Andererseits sind komplementär zur rekursiven Konstruktion, Kenogrammkomplexionen nicht als vorfindliche Objekte zu verstehen. Sie sind verdeckt und lassen sich nicht direkt beschreiben, bzw. charakterisieren.

Es gibt, genau betrachtet, kein Anfangskenogramm für einen induktiven bzw. rekursiven Aufbau der Kenogramm-Komplexionen. Die Kenogrammsequenzen sind somit als solche nicht in einer Wortalgebra beschreibbar.

Bisdahin wurde in der Literatur zur Kenogrammatik das Problem des fehlenden Anfangskenogramms zum rekursiven Aufbau der Kenogrammsequenzen bewusst mehr oder weniger trickreich zu Gunsten einer Konstruktion ausgeklammert.

Eine positive Lösung des Anfangsproblems könnte darin liegen, einen behavioral viewpoint einzunehmen und

mit dem Konzept der Co-Induktion zu arbeiten. Eine Methode für die Formalisierung könnte sein, ausgewogen zwischen Konstruktion und Dekonstruktion, zwischen streng finaler und streng terminaler Ausrichtung einzusetzen.

Ein weiterer Schritt müsste dann allerdings darin bestehen, diesen Gegensatz als solchen zu verwerfen und ihn als monokontextual zu identifizieren, zu dekonstruieren und entsprechend neue Formalismen zu entwickeln.” (SKIZZE-0.9.5, 2003)

Aufbau : Konstruktoren

Abbau : Selektoren

Observatoren

Algebra: Induktion

Co-Algebra: Coinduktion

Dualität

Systemwechsel

Weder Text noch Formel noch Programm

”Die Kenogrammatik ist weder durch Zeichenreihen konstruktiver Art, noch durch Zeichenströme koinduktiver Art zu bestimmen. Im Gegensatz zu mathematischen und programmiersprachlichen Verschriftungen erzeugen kenomische Ereignisse keinen Text, weder einen rein linearen noch einen vernetzt-tabularen. Sowohl Zeichenreihen wie auch Zeichenströme sind über einem Alphabet definiert, sei es durch Induktion oder durch Koinduktion und sind in einer fundierten oder unfundierten Tektonik hierarchischer oder zirkulärer Strukturen versammelt.”

1

Deconstructing beginnings

”Einerseits lassen sich Kenogrammsequenzen rekursiv konstruieren, wenn auch nur in Analogie zu semiotischen Systemen, fehlt ihnen doch ein echtes initiales Objekt. Sie haben somit eine Objekt-Struktur. Andererseits sind komplementär zur rekursiven Konstruktion, Kenogrammkomplexionen nicht als vorfindliche Objekte zu verstehen. Sie sind verdeckt und lassen sich nicht direkt beschreiben, bzw. charakterisieren.

Es gibt, genau betrachtet, kein Anfangskenogramm für einen induktiven bzw. rekursiven Aufbau der Kenogramm-Komplexionen. Die Kenogrammsequenzen sind somit als solche nicht in einer Wortalgebra beschreibbar.

Bisdahin wurde in der Literatur zur Kenogrammatik das Problem des fehlenden Anfangskenogramms zum rekursiven Aufbau der Kenogrammsequenzen bewusst mehr oder weniger trickreich zu Gunsten einer Konstruktion ausgeklammert.

Eine positive Lösung des Anfangsproblems könnte darin liegen, einen behavioral viewpoint einzunehmen und mit dem Konzept der Co-Induktion zu arbeiten. Eine Methode für die Formalisierung könnte sein, ausgewogen zwischen Konstruktion und Dekonstruktion, zwischen streng finaler und streng terminaler Ausrichtung einzusetzen.

Ein weiterer Schritt müsste dann allerdings darin bestehen, diesen Gegensatz als solchen zu verwerfen und ihn als monokontextual zu identifizieren, zu dekonstruieren und entsprechend neue Formalismen zu entwickeln.” (SKIZZE-0.9.5, 2003)

Aufbau : Konstruktoren

Abbau : Selektoren

Observatoren

Algebra: Induktion

Co-Algebra: Coinduktion

Dualität

Systemwechsel

Weder Text noch Formel noch Programm

”Die Kenogrammatik ist weder durch Zeichenreihen konstruktiver Art, noch durch Zeichenströme koinduktiver Art zu bestimmen. Im Gegensatz zu mathematischen und programmiersprachlichen Verschriftungen erzeugen kenomische Ereignisse keinen Text, weder einen rein linearen noch einen vernetzt-tabularen. Sowohl Zeichenreihen wie auch Zeichenströme sind über einem Alphabet definiert, sei es durch Induktion oder durch Koinduktion und sind in einer fundierten oder unfundierten Tektonik hierarchischer oder zirkulärer Strukturen versammelt.”

3

<http://www.tcs.tifr.res.in/~pandya/grad/aut06/lect4.pdf>

<http://coalg.org/cmcs12/slides/ciancia.pdf>

<http://www.math.uni-hamburg.de/home/loewe/2006-07-I/Venema.pdf>

<http://coalg.org/cmcs12/slides/ciancia.pdf>

4

- nfirstq (55, TU);

val it =

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



```

[1,1,2,3,2],[1,1,2,3,3],[1,1,2,3,4],[1,2,1,1,1],[1,2,1,1,2],[1,2,1,1,3],
[1,2,1,2,1],[1,2,1,2,2],[1,2,1,2,3],[1,2,1,3,1],[1,2,1,3,2],[1,2,1,3,3],
[1,2,1,3,4],[1,2,2,1,1],[1,2,2,1,2],[1,2,2,1,3],[1,2,2,2,1],[1,2,2,2,2],
[1,2,2,2,3],[1,2,2,3,1]] : int list list

```

5

```

- ENstructure [1,1,1,2,2,1,1,1];
val it =
[[],
[(1,2,E)],
[(1,3,E),(2,3,E)],
[(1,4,N),(2,4,N),(3,4,N)],
[(1,5,N),(2,5,N),(3,5,N),(4,5,E)],
[(1,6,E),(2,6,E),(3,6,E),(4,6,N),(5,6,N)],
[(1,7,E),(2,7,E),(3,7,E),(4,7,N),(5,7,N),(6,7,E)],
[(1,8,E),(2,8,E),(3,8,E),(4,8,N),(5,8,N),(6,8,E),(7,8,E)]]
: (int * int * EN) list list

- ENstructure[1,1,1,1,2,2,1,1,1,1];
val it =
[[],
[(1,2,E)],
[(1,3,E),(2,3,E)],
[(1,4,E),(2,4,E),(3,4,E)],
[(1,5,N),(2,5,N),(3,5,N),(4,5,N)],
[(1,6,N),(2,6,N),(3,6,N),(4,6,N),(5,6,E)],
[(1,7,E),(2,7,E),(3,7,E),(4,7,E),(5,7,N),(6,7,N)],
[(1,8,E),(2,8,E),(3,8,E),(4,8,E),(5,8,N),(6,8,N),(7,8,E)],
[(1,9,E),(2,9,E),(3,9,E),(4,9,E),(5,9,N),(6,9,N),(7,9,E),(8,9,E)],
[(1,10,E),(2,10,E),(3,10,E),(4,10,E),(5,10,N),(6,10,N),(7,10,E),(8,10,E),(9,10,E)]]
: (int * int * EN) list list

- ENstructure[1,1,1,1,2,2,2,1,1,1,1];
val it =
[[],
[(1,2,E)],
[(1,3,E),(2,3,E)],
[(1,4,E),(2,4,E),(3,4,E)],
[(1,5,N),(2,5,N),(3,5,N),(4,5,N)],
[(1,6,N),(2,6,N),(3,6,N),(4,6,N),(5,6,E)],
[(1,7,N),(2,7,N),(3,7,N),(4,7,N),(5,7,E),(6,7,E)],
[(1,8,E),(2,8,E),(3,8,E),(4,8,E),(5,8,N),(6,8,N),(7,8,N)],
[(1,9,E),(2,9,E),(3,9,E),(4,9,E),(5,9,N),(6,9,N),(7,9,N),(8,9,E)],
[(1,10,E),(2,10,E),(3,10,E),(4,10,E),(5,10,N),(6,10,N),(7,10,N),(8,10,E),(9,10,E)],
[(1,11,E),(2,11,E),(3,11,E),(4,11,E),(5,11,N),(6,11,N),(7,11,N),(8,11,E),(9,11,E),(10,11,E)]]
: (int * int * EN) list list

```

6

```

nfirstq(5000, TU)
List.filter palindrome "nfirstq(5000, TU)";
- length it;
val it = 180 : int
val it =
[[1],[1,1],[1,2],
[1,1,1],[1,2,1],[1,2,3],
[1,1,1,1],[1,1,2,2],[1,2,1,2], [1,2,2,1],[1,2,2,3],[1,2,3,1],[1,2,3,4],
[1,1,1,1,1],[1,1,2,1,1], [1,1,2,3,3],[1,2,1,2,1],[1,2,1,3,1],[1,2,2,2,1],
[1,2,2,2,3],[1,2,3,1,2], [1,2,3,2,1],[1,2,3,2,4],[1,2,3,4,1],[1,2,3,4,5],

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

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

```

[1,2,1,1,1,2,1],[1,2,1,1,1,3,1],[1,2,1,2,1,2,1],[1,2,1,2,3,2,3],
 [1,2,1,3,1,2,1],[1,2,1,3,1,4,1],[1,2,1,3,2,1,2],[1,2,1,3,4,2,4],
 [1,2,1,3,4,5,4],[1,2,2,1,2,2,1],[1,2,2,1,3,3,1],[1,2,2,2,2,2,1],
 [1,2,2,2,2,2,3],[1,2,2,3,1,1,2],[1,2,2,3,2,2,1],[1,2,2,3,2,2,4],
 [1,2,2,3,4,4,1],[1,2,2,3,4,4,5],[1,2,3,1,2,3,1],[1,2,3,1,3,2,1],
 [1,2,3,1,3,4,1],[1,2,3,1,4,2,1],[1,2,3,1,4,5,1],[1,2,3,2,1,2,3],
 [1,2,3,2,3,2,1],[1,2,3,2,3,2,4],[1,2,3,2,4,2,1],[1,2,3,2,4,2,5],
 [1,2,3,3,3,1,2],[1,2,3,3,3,2,1],[1,2,3,3,3,2,4],[1,2,3,3,3,4,1],
 [1,2,3,3,3,4,5],[1,2,3,4,1,2,3],[1,2,3,4,1,5,3],[1,2,3,4,2,3,1],
 [1,2,3,4,2,3,5],[1,2,3,4,3,1,2],[1,2,3,4,3,2,1],[1,2,3,4,3,2,5],
 [1,2,3,4,3,5,1],[1,2,3,4,3,5,6],[1,2,3,4,5,1,2],[1,2,3,4,5,2,1],
 [1,2,3,4,5,2,6],[1,2,3,4,5,6,1],[1,2,3,4,5,6,7],

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

: int list list

9

- kmul [1,2,3][1,2,3];

val it =

[[1,2,3,2,3,1,3,1,2],[1,2,3,2,3,1,4,1,2],[1,2,3,2,3,1,3,1,4],
 [1,2,3,2,3,1,4,1,5],[1,2,3,2,3,1,3,4,2],[1,2,3,2,3,1,4,5,2],
 [1,2,3,2,3,1,3,4,5],[1,2,3,2,3,1,4,5,6],[1,2,3,3,1,2,2,3,1],
 [1,2,3,3,1,2,2,4,1],[1,2,3,3,1,2,4,3,1],[1,2,3,3,1,2,4,5,1],
 [1,2,3,3,1,2,2,3,4],[1,2,3,3,1,2,2,4,5],[1,2,3,3,1,2,4,3,5],
 [1,2,3,3,1,2,4,5,6],[1,2,3,2,1,4,3,4,1],[1,2,3,2,1,4,4,3,1],
 [1,2,3,2,1,4,3,5,1],[1,2,3,2,1,4,5,3,1],[1,2,3,2,1,4,4,5,1],
 [1,2,3,2,1,4,5,4,1],[1,2,3,2,1,4,5,6,1],[1,2,3,2,1,4,3,4,2],
 [1,2,3,2,1,4,4,3,2],[1,2,3,2,1,4,3,5,2],[1,2,3,2,1,4,5,3,2],
 [1,2,3,2,1,4,4,5,2],[1,2,3,2,1,4,5,4,2],[1,2,3,2,1,4,5,6,2],
 [1,2,3,2,1,4,3,4,5],[1,2,3,2,1,4,4,3,5],[1,2,3,2,1,4,3,5,6],
 [1,2,3,2,1,4,5,3,6],[1,2,3,2,1,4,4,5,6],[1,2,3,2,1,4,5,4,6],
 [1,2,3,2,1,4,5,6,7],[1,2,3,2,4,1,3,1,2],[1,2,3,2,4,1,4,1,2],
 [1,2,3,2,4,1,5,1,2],[1,2,3,2,4,1,3,1,4],[1,2,3,2,4,1,3,1,5],
 [1,2,3,2,4,1,4,1,5],[1,2,3,2,4,1,5,1,4],[1,2,3,2,4,1,5,1,6],
 [1,2,3,2,4,1,4,3,2],[1,2,3,2,4,1,3,5,2],[1,2,3,2,4,1,5,3,2],
 [1,2,3,2,4,1,4,5,2],[1,2,3,2,4,1,5,6,2],[1,2,3,2,4,1,3,5,4],
 [1,2,3,2,4,1,4,3,5],[1,2,3,2,4,1,5,3,4],[1,2,3,2,4,1,3,5,6],
 [1,2,3,2,4,1,5,3,6],[1,2,3,2,4,1,4,5,6],[1,2,3,2,4,1,5,6,4],
 [1,2,3,2,4,1,5,6,7],[1,2,3,4,1,2,2,3,1],[1,2,3,4,1,2,2,4,1],
 [1,2,3,4,1,2,2,5,1],[1,2,3,4,1,2,3,4,1],[1,2,3,4,1,2,3,5,1],
 [1,2,3,4,1,2,5,3,1],[1,2,3,4,1,2,5,4,1],[1,2,3,4,1,2,5,6,1],
 [1,2,3,4,1,2,2,3,4],[1,2,3,4,1,2,2,3,5],[1,2,3,4,1,2,2,4,5],
 [1,2,3,4,1,2,2,5,4],[1,2,3,4,1,2,2,5,6],[1,2,3,4,1,2,3,4,5],
 [1,2,3,4,1,2,3,5,4],[1,2,3,4,1,2,5,3,4],[1,2,3,4,1,2,3,5,6],
 [1,2,3,4,1,2,5,3,6],[1,2,3,4,1,2,5,4,6],[1,2,3,4,1,2,5,6,4],
 [1,2,3,4,1,2,5,6,7],[1,2,3,3,1,4,2,3,1],[1,2,3,3,1,4,2,4,1],
 [1,2,3,3,1,4,2,5,1],[1,2,3,3,1,4,4,3,1],[1,2,3,3,1,4,5,3,1],
 [1,2,3,3,1,4,4,5,1],[1,2,3,3,1,4,5,4,1],[1,2,3,3,1,4,5,6,1],
 [1,2,3,3,1,4,4,3,2],[1,2,3,3,1,4,2,3,5],[1,2,3,3,1,4,5,3,2],
 [1,2,3,3,1,4,2,4,5],[1,2,3,3,1,4,4,5,2],[1,2,3,3,1,4,5,4,2],
 [1,2,3,3,1,4,2,5,6],[1,2,3,3,1,4,5,6,2],[1,2,3,3,1,4,4,3,5],

[1,2,3,3,1,4,5,3,6],[1,2,3,3,1,4,4,5,6],[1,2,3,3,1,4,5,4,6],
[1,2,3,3,1,4,5,6,7],[1,2,3,3,4,1,2,1,4],[1,2,3,3,4,1,4,1,2],
[1,2,3,3,4,1,2,1,5],[1,2,3,3,4,1,5,1,2],[1,2,3,3,4,1,4,1,5],
[1,2,3,3,4,1,5,1,4],[1,2,3,3,4,1,5,1,6],[1,2,3,3,4,1,2,3,4],
[1,2,3,3,4,1,4,3,2],[1,2,3,3,4,1,2,3,5],[1,2,3,3,4,1,5,3,2],
[1,2,3,3,4,1,2,5,4],[1,2,3,3,4,1,4,5,2],[1,2,3,3,4,1,2,5,6],
[1,2,3,3,4,1,5,6,2],[1,2,3,3,4,1,4,3,5],[1,2,3,3,4,1,5,3,4],
[1,2,3,3,4,1,5,3,6],[1,2,3,3,4,1,4,5,6],[1,2,3,3,4,1,5,6,4],
[1,2,3,3,4,1,5,6,7],[1,2,3,4,3,1,3,1,2],[1,2,3,4,3,1,2,1,4],
[1,2,3,4,3,1,2,1,5],[1,2,3,4,3,1,5,1,2],[1,2,3,4,3,1,3,1,4],
[1,2,3,4,3,1,3,1,5],[1,2,3,4,3,1,5,1,4],[1,2,3,4,3,1,5,1,6],
[1,2,3,4,3,1,3,4,2],[1,2,3,4,3,1,3,5,2],[1,2,3,4,3,1,2,4,5],
[1,2,3,4,3,1,2,5,4],[1,2,3,4,3,1,5,4,2],[1,2,3,4,3,1,2,5,6],
[1,2,3,4,3,1,5,6,2],[1,2,3,4,3,1,3,4,5],[1,2,3,4,3,1,3,5,4],
[1,2,3,4,3,1,3,5,6],[1,2,3,4,3,1,5,4,6],[1,2,3,4,3,1,5,6,4],
[1,2,3,4,3,1,5,6,7],[1,2,3,4,1,5,2,3,1],[1,2,3,4,1,5,2,4,1],
[1,2,3,4,1,5,2,5,1],[1,2,3,4,1,5,2,6,1],[1,2,3,4,1,5,3,4,1],
[1,2,3,4,1,5,3,5,1],[1,2,3,4,1,5,5,3,1],[1,2,3,4,1,5,3,6,1],
[1,2,3,4,1,5,6,3,1],[1,2,3,4,1,5,5,4,1],[1,2,3,4,1,5,6,4,1],
[1,2,3,4,1,5,5,6,1],[1,2,3,4,1,5,6,5,1],[1,2,3,4,1,5,6,7,1],
[1,2,3,4,1,5,2,3,4],[1,2,3,4,1,5,3,4,2],[1,2,3,4,1,5,3,5,2],
[1,2,3,4,1,5,5,3,2],[1,2,3,4,1,5,2,3,6],[1,2,3,4,1,5,3,6,2],
[1,2,3,4,1,5,6,3,2],[1,2,3,4,1,5,2,5,4],[1,2,3,4,1,5,5,4,2],
[1,2,3,4,1,5,2,4,6],[1,2,3,4,1,5,2,6,4],[1,2,3,4,1,5,6,4,2],
[1,2,3,4,1,5,2,5,6],[1,2,3,4,1,5,5,6,2],[1,2,3,4,1,5,6,5,2],
[1,2,3,4,1,5,2,6,7],[1,2,3,4,1,5,6,7,2],[1,2,3,4,1,5,3,5,4],
[1,2,3,4,1,5,5,3,4],[1,2,3,4,1,5,3,4,6],[1,2,3,4,1,5,3,6,4],
[1,2,3,4,1,5,6,3,4],[1,2,3,4,1,5,3,5,6],[1,2,3,4,1,5,5,3,6],
[1,2,3,4,1,5,3,6,7],[1,2,3,4,1,5,6,3,7],[1,2,3,4,1,5,5,4,6],
[1,2,3,4,1,5,5,6,4],[1,2,3,4,1,5,6,5,4],[1,2,3,4,1,5,6,4,7],
[1,2,3,4,1,5,6,7,4],[1,2,3,4,1,5,5,6,7],[1,2,3,4,1,5,6,5,7],
[1,2,3,4,1,5,6,7,8],[1,2,3,4,5,1,3,1,2],[1,2,3,4,5,1,2,1,4],
[1,2,3,4,5,1,2,1,5],[1,2,3,4,5,1,5,1,2],[1,2,3,4,5,1,2,1,6],
[1,2,3,4,5,1,6,1,2],[1,2,3,4,5,1,3,1,4],[1,2,3,4,5,1,3,1,5],
[1,2,3,4,5,1,3,1,6],[1,2,3,4,5,1,5,1,4],[1,2,3,4,5,1,6,1,4],
[1,2,3,4,5,1,5,1,6],[1,2,3,4,5,1,6,1,5],[1,2,3,4,5,1,6,1,7],
[1,2,3,4,5,1,2,3,4],[1,2,3,4,5,1,3,4,2],[1,2,3,4,5,1,2,3,5],
[1,2,3,4,5,1,5,3,2],[1,2,3,4,5,1,2,3,6],[1,2,3,4,5,1,3,6,2],
[1,2,3,4,5,1,6,3,2],[1,2,3,4,5,1,2,4,5],[1,2,3,4,5,1,5,4,2],
[1,2,3,4,5,1,2,4,6],[1,2,3,4,5,1,2,6,4],[1,2,3,4,5,1,6,4,2],
[1,2,3,4,5,1,2,6,5],[1,2,3,4,5,1,5,6,2],[1,2,3,4,5,1,2,6,7],
[1,2,3,4,5,1,6,7,2],[1,2,3,4,5,1,3,4,5],[1,2,3,4,5,1,5,3,4],
[1,2,3,4,5,1,3,4,6],[1,2,3,4,5,1,3,6,4],[1,2,3,4,5,1,6,3,4],
[1,2,3,4,5,1,3,6,5],[1,2,3,4,5,1,5,3,6],[1,2,3,4,5,1,6,3,5],
[1,2,3,4,5,1,3,6,7],[1,2,3,4,5,1,6,3,7],[1,2,3,4,5,1,5,4,6],
[1,2,3,4,5,1,5,6,4],[1,2,3,4,5,1,6,4,5],[1,2,3,4,5,1,6,4,7],
[1,2,3,4,5,1,6,7,4],[1,2,3,4,5,1,5,6,7],[1,2,3,4,5,1,6,7,5],
[1,2,3,4,5,1,6,7,8],[1,2,3,2,3,4,3,1,2],[1,2,3,2,3,4,4,1,2],
[1,2,3,2,3,4,5,1,2],[1,2,3,2,3,4,3,4,1],[1,2,3,2,3,4,3,1,5],
[1,2,3,2,3,4,3,5,1],[1,2,3,2,3,4,4,1,5],[1,2,3,2,3,4,4,5,1],
[1,2,3,2,3,4,5,4,1],[1,2,3,2,3,4,5,1,6],[1,2,3,2,3,4,5,6,1],
[1,2,3,2,3,4,3,4,2],[1,2,3,2,3,4,3,5,2],[1,2,3,2,3,4,4,5,2],
[1,2,3,2,3,4,5,4,2],[1,2,3,2,3,4,5,6,2],[1,2,3,2,3,4,3,4,5],
[1,2,3,2,3,4,3,5,6],[1,2,3,2,3,4,4,5,6],[1,2,3,2,3,4,5,4,6],
[1,2,3,2,3,4,5,6,7],[1,2,3,3,4,2,2,3,1],[1,2,3,3,4,2,2,1,4],
[1,2,3,3,4,2,2,1,5],[1,2,3,3,4,2,2,5,1],[1,2,3,3,4,2,4,3,1],
[1,2,3,3,4,2,5,3,1],[1,2,3,3,4,2,4,1,5],[1,2,3,3,4,2,4,5,1],
[1,2,3,3,4,2,5,1,4],[1,2,3,3,4,2,5,1,6],[1,2,3,3,4,2,5,6,1],
[1,2,3,3,4,2,2,3,4],[1,2,3,3,4,2,2,3,5],[1,2,3,3,4,2,2,5,4],
[1,2,3,3,4,2,2,5,6],[1,2,3,3,4,2,4,3,5],[1,2,3,3,4,2,5,3,4],
[1,2,3,3,4,2,5,3,6],[1,2,3,3,4,2,4,5,6],[1,2,3,3,4,2,5,6,4],
[1,2,3,3,4,2,5,6,7],[1,2,3,4,3,2,2,1,4],[1,2,3,4,3,2,2,4,1],
[1,2,3,4,3,2,2,1,5],[1,2,3,4,3,2,2,5,1],[1,2,3,4,3,2,3,1,4],
[1,2,3,4,3,2,3,4,1],[1,2,3,4,3,2,3,1,5],[1,2,3,4,3,2,3,5,1],
[1,2,3,4,3,2,5,1,4],[1,2,3,4,3,2,5,4,1],[1,2,3,4,3,2,5,1,6],

[1,2,3,4,3,2,5,6,1],[1,2,3,4,3,2,2,4,5],[1,2,3,4,3,2,2,5,4],
[1,2,3,4,3,2,2,5,6],[1,2,3,4,3,2,3,4,5],[1,2,3,4,3,2,3,5,4],
[1,2,3,4,3,2,3,5,6],[1,2,3,4,3,2,5,4,6],[1,2,3,4,3,2,5,6,4],
[1,2,3,4,3,2,5,6,7],[1,2,3,2,4,5,3,1,2],[1,2,3,2,4,5,4,1,2],
[1,2,3,2,4,5,5,1,2],[1,2,3,2,4,5,6,1,2],[1,2,3,2,4,5,3,1,4],
[1,2,3,2,4,5,4,3,1],[1,2,3,2,4,5,3,5,1],[1,2,3,2,4,5,5,3,1],
[1,2,3,2,4,5,3,1,6],[1,2,3,2,4,5,3,6,1],[1,2,3,2,4,5,6,3,1],
[1,2,3,2,4,5,4,5,1],[1,2,3,2,4,5,5,1,4],[1,2,3,2,4,5,4,1,6],
[1,2,3,2,4,5,4,6,1],[1,2,3,2,4,5,6,1,4],[1,2,3,2,4,5,5,1,6],
[1,2,3,2,4,5,5,6,1],[1,2,3,2,4,5,6,5,1],[1,2,3,2,4,5,6,1,7],
[1,2,3,2,4,5,6,7,1],[1,2,3,2,4,5,4,3,2],[1,2,3,2,4,5,3,5,2],
[1,2,3,2,4,5,5,3,2],[1,2,3,2,4,5,3,6,2],[1,2,3,2,4,5,6,3,2],
[1,2,3,2,4,5,4,5,2],[1,2,3,2,4,5,4,6,2],[1,2,3,2,4,5,5,6,2],
[1,2,3,2,4,5,6,5,2],[1,2,3,2,4,5,6,7,2],[1,2,3,2,4,5,3,5,4],
[1,2,3,2,4,5,5,3,4],[1,2,3,2,4,5,3,6,4],[1,2,3,2,4,5,4,3,6],
[1,2,3,2,4,5,6,3,4],[1,2,3,2,4,5,3,5,6],[1,2,3,2,4,5,5,3,6],
[1,2,3,2,4,5,3,6,7],[1,2,3,2,4,5,6,3,7],[1,2,3,2,4,5,4,5,6],
[1,2,3,2,4,5,5,6,4],[1,2,3,2,4,5,6,5,4],[1,2,3,2,4,5,4,6,7],
[1,2,3,2,4,5,6,7,4],[1,2,3,2,4,5,5,6,7],[1,2,3,2,4,5,6,5,7],
[1,2,3,2,4,5,6,7,8],[1,2,3,4,5,2,2,3,1],[1,2,3,4,5,2,2,1,4],
[1,2,3,4,5,2,2,4,1],[1,2,3,4,5,2,2,1,5],[1,2,3,4,5,2,2,1,6],
[1,2,3,4,5,2,2,6,1],[1,2,3,4,5,2,3,1,4],[1,2,3,4,5,2,3,4,1],
[1,2,3,4,5,2,3,1,5],[1,2,3,4,5,2,5,3,1],[1,2,3,4,5,2,3,1,6],
[1,2,3,4,5,2,3,6,1],[1,2,3,4,5,2,6,3,1],[1,2,3,4,5,2,5,1,4],
[1,2,3,4,5,2,5,4,1],[1,2,3,4,5,2,6,1,4],[1,2,3,4,5,2,6,4,1],
[1,2,3,4,5,2,5,1,6],[1,2,3,4,5,2,5,6,1],[1,2,3,4,5,2,6,1,5],
[1,2,3,4,5,2,6,1,7],[1,2,3,4,5,2,6,7,1],[1,2,3,4,5,2,2,3,4],
[1,2,3,4,5,2,2,3,5],[1,2,3,4,5,2,2,3,6],[1,2,3,4,5,2,2,4,5],
[1,2,3,4,5,2,2,4,6],[1,2,3,4,5,2,2,6,4],[1,2,3,4,5,2,2,6,5],
[1,2,3,4,5,2,2,6,7],[1,2,3,4,5,2,3,4,5],[1,2,3,4,5,2,5,3,4],
[1,2,3,4,5,2,3,4,6],[1,2,3,4,5,2,3,6,4],[1,2,3,4,5,2,6,3,4],
[1,2,3,4,5,2,3,6,5],[1,2,3,4,5,2,5,3,6],[1,2,3,4,5,2,6,3,5],
[1,2,3,4,5,2,3,6,7],[1,2,3,4,5,2,6,3,7],[1,2,3,4,5,2,5,4,6],
[1,2,3,4,5,2,5,6,4],[1,2,3,4,5,2,6,4,5],[1,2,3,4,5,2,6,4,7],
[1,2,3,4,5,2,6,7,4],[1,2,3,4,5,2,5,6,7],[1,2,3,4,5,2,6,7,5],
[1,2,3,4,5,2,6,7,8],[1,2,3,3,4,5,2,3,1],[1,2,3,3,4,5,2,1,4],
[1,2,3,3,4,5,4,1,2],[1,2,3,3,4,5,2,5,1],[1,2,3,3,4,5,5,1,2],
[1,2,3,3,4,5,2,1,6],[1,2,3,3,4,5,2,6,1],[1,2,3,3,4,5,6,1,2],
[1,2,3,3,4,5,4,3,1],[1,2,3,3,4,5,5,3,1],[1,2,3,3,4,5,6,3,1],
[1,2,3,3,4,5,4,5,1],[1,2,3,3,4,5,5,1,4],[1,2,3,3,4,5,4,1,6],
[1,2,3,3,4,5,4,6,1],[1,2,3,3,4,5,6,1,4],[1,2,3,3,4,5,5,1,6],
[1,2,3,3,4,5,5,6,1],[1,2,3,3,4,5,6,5,1],[1,2,3,3,4,5,6,1,7],
[1,2,3,3,4,5,6,7,1],[1,2,3,3,4,5,2,3,4],[1,2,3,3,4,5,4,3,2],
[1,2,3,3,4,5,5,3,2],[1,2,3,3,4,5,2,3,6],[1,2,3,3,4,5,6,3,2],
[1,2,3,3,4,5,2,5,4],[1,2,3,3,4,5,4,5,2],[1,2,3,3,4,5,2,6,4],
[1,2,3,3,4,5,4,6,2],[1,2,3,3,4,5,2,5,6],[1,2,3,3,4,5,5,6,2],
[1,2,3,3,4,5,6,5,2],[1,2,3,3,4,5,2,6,7],[1,2,3,3,4,5,6,7,2],
[1,2,3,3,4,5,5,3,4],[1,2,3,3,4,5,4,3,6],[1,2,3,3,4,5,6,3,4],
[1,2,3,3,4,5,5,3,6],[1,2,3,3,4,5,6,3,7],[1,2,3,3,4,5,4,5,6],
[1,2,3,3,4,5,5,6,4],[1,2,3,3,4,5,6,5,4],[1,2,3,3,4,5,4,6,7],
[1,2,3,3,4,5,6,7,4],[1,2,3,3,4,5,5,6,7],[1,2,3,3,4,5,6,5,7],
[1,2,3,3,4,5,6,7,8],[1,2,3,4,3,5,3,1,2],[1,2,3,4,3,5,2,1,4],
[1,2,3,4,3,5,2,4,1],[1,2,3,4,3,5,2,5,1],[1,2,3,4,3,5,5,1,2],
[1,2,3,4,3,5,2,1,6],[1,2,3,4,3,5,2,6,1],[1,2,3,4,3,5,6,1,2],
[1,2,3,4,3,5,3,1,4],[1,2,3,4,3,5,3,4,1],[1,2,3,4,3,5,3,5,1],
[1,2,3,4,3,5,3,1,6],[1,2,3,4,3,5,3,6,1],[1,2,3,4,3,5,5,1,4],
[1,2,3,4,3,5,5,4,1],[1,2,3,4,3,5,6,1,4],[1,2,3,4,3,5,6,4,1],
[1,2,3,4,3,5,5,1,6],[1,2,3,4,3,5,5,6,1],[1,2,3,4,3,5,6,5,1],
[1,2,3,4,3,5,6,1,7],[1,2,3,4,3,5,6,7,1],[1,2,3,4,3,5,3,4,2],
[1,2,3,4,3,5,3,5,2],[1,2,3,4,3,5,3,6,2],[1,2,3,4,3,5,2,5,4],
[1,2,3,4,3,5,5,4,2],[1,2,3,4,3,5,2,4,6],[1,2,3,4,3,5,2,6,4],
[1,2,3,4,3,5,6,4,2],[1,2,3,4,3,5,2,5,6],[1,2,3,4,3,5,5,6,2],
[1,2,3,4,3,5,6,5,2],[1,2,3,4,3,5,2,6,7],[1,2,3,4,3,5,6,7,2],
[1,2,3,4,3,5,3,5,4],[1,2,3,4,3,5,3,4,6],[1,2,3,4,3,5,3,6,4],
[1,2,3,4,3,5,3,5,6],[1,2,3,4,3,5,3,6,7],[1,2,3,4,3,5,5,4,6],

```

[1,2,3,4,3,5,5,6,4],[1,2,3,4,3,5,6,5,4],[1,2,3,4,3,5,6,4,7],
[1,2,3,4,3,5,6,7,4],[1,2,3,4,3,5,5,6,7],[1,2,3,4,3,5,6,5,7],
[1,2,3,4,3,5,6,7,8],[1,2,3,4,5,6,2,3,1],[1,2,3,4,5,6,3,1,2],
[1,2,3,4,5,6,2,1,4],[1,2,3,4,5,6,2,4,1],[1,2,3,4,5,6,2,1,5],
[1,2,3,4,5,6,5,1,2],[1,2,3,4,5,6,2,6,1],[1,2,3,4,5,6,6,1,2],
[1,2,3,4,5,6,2,1,7],[1,2,3,4,5,6,2,7,1],[1,2,3,4,5,6,7,1,2],
[1,2,3,4,5,6,3,1,4],[1,2,3,4,5,6,3,4,1],[1,2,3,4,5,6,3,1,5],
[1,2,3,4,5,6,5,3,1],[1,2,3,4,5,6,3,6,1],[1,2,3,4,5,6,6,3,1],
[1,2,3,4,5,6,3,1,7],[1,2,3,4,5,6,3,7,1],[1,2,3,4,5,6,7,3,1],
[1,2,3,4,5,6,5,1,4],[1,2,3,4,5,6,5,4,1],[1,2,3,4,5,6,6,1,4],
[1,2,3,4,5,6,6,4,1],[1,2,3,4,5,6,7,1,4],[1,2,3,4,5,6,7,4,1],
[1,2,3,4,5,6,5,6,1],[1,2,3,4,5,6,6,1,5],[1,2,3,4,5,6,5,1,7],
[1,2,3,4,5,6,5,7,1],[1,2,3,4,5,6,7,1,5],[1,2,3,4,5,6,6,1,7],
[1,2,3,4,5,6,6,7,1],[1,2,3,4,5,6,7,6,1],[1,2,3,4,5,6,7,1,8],
[1,2,3,4,5,6,7,8,1],[1,2,3,4,5,6,2,3,4],[1,2,3,4,5,6,3,4,2],
[1,2,3,4,5,6,2,3,5],[1,2,3,4,5,6,5,3,2],[1,2,3,4,5,6,3,6,2],
[1,2,3,4,5,6,6,3,2],[1,2,3,4,5,6,2,3,7],[1,2,3,4,5,6,3,7,2],
[1,2,3,4,5,6,7,3,2],[1,2,3,4,5,6,2,4,5],[1,2,3,4,5,6,5,4,2],
[1,2,3,4,5,6,2,6,4],[1,2,3,4,5,6,6,4,2],[1,2,3,4,5,6,2,4,7],
[1,2,3,4,5,6,2,7,4],[1,2,3,4,5,6,7,4,2],[1,2,3,4,5,6,2,6,5],
[1,2,3,4,5,6,5,6,2],[1,2,3,4,5,6,2,7,5],[1,2,3,4,5,6,5,7,2],
[1,2,3,4,5,6,2,6,7],[1,2,3,4,5,6,6,7,2],[1,2,3,4,5,6,7,6,2],
[1,2,3,4,5,6,2,7,8],[1,2,3,4,5,6,7,8,2],[1,2,3,4,5,6,3,4,5],
[1,2,3,4,5,6,5,3,4],[1,2,3,4,5,6,3,6,4],[1,2,3,4,5,6,6,3,4],
[1,2,3,4,5,6,3,4,7],[1,2,3,4,5,6,3,7,4],[1,2,3,4,5,6,7,3,4],
[1,2,3,4,5,6,3,6,5],[1,2,3,4,5,6,6,3,5],[1,2,3,4,5,6,3,7,5],
[1,2,3,4,5,6,5,3,7],[1,2,3,4,5,6,7,3,5],[1,2,3,4,5,6,3,6,7],
[1,2,3,4,5,6,6,3,7],[1,2,3,4,5,6,3,7,8],[1,2,3,4,5,6,7,3,8],
[1,2,3,4,5,6,5,6,4],[1,2,3,4,5,6,6,4,5],[1,2,3,4,5,6,5,4,7],
[1,2,3,4,5,6,5,7,4],[1,2,3,4,5,6,7,4,5],[1,2,3,4,5,6,6,4,7],
[1,2,3,4,5,6,6,7,4],[1,2,3,4,5,6,7,6,4],[1,2,3,4,5,6,7,4,8],
[1,2,3,4,5,6,7,8,4],[1,2,3,4,5,6,5,6,7],[1,2,3,4,5,6,6,7,5],
[1,2,3,4,5,6,7,6,5],[1,2,3,4,5,6,5,7,8],[1,2,3,4,5,6,7,8,5],
[1,2,3,4,5,6,6,7,8],[1,2,3,4,5,6,7,6,8],[1,2,3,4,5,6,7,8,9]]
: int list list

```

- kmul [1,2,2][1,2,3,1];

val it =

```

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

```

- kmul [1,2,2,1][1,2,3,3,1,4];

val it =

```

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

```

[1,2,2,1,3,1,1,3,2,4,4,2,2,4,4,2,1,2,2,1,5,6,6,5],
[1,2,2,1,3,1,1,3,4,3,3,4,4,3,3,4,1,2,2,1,2,4,4,2],
[1,2,2,1,3,1,1,3,4,3,3,4,4,3,3,4,1,2,2,1,2,5,5,2],
[1,2,2,1,3,1,1,3,4,3,3,4,4,3,3,4,1,2,2,1,5,4,4,5],
[1,2,2,1,3,1,1,3,4,3,3,4,4,3,3,4,1,2,2,1,5,6,6,5],
[1,2,2,1,3,1,1,3,4,5,5,4,4,5,5,4,1,2,2,1,2,3,3,2],
[1,2,2,1,3,1,1,3,4,5,5,4,4,5,5,4,1,2,2,1,2,4,4,2],
[1,2,2,1,3,1,1,3,4,5,5,4,4,5,5,4,1,2,2,1,2,6,6,2],
[1,2,2,1,3,1,1,3,4,5,5,4,4,5,5,4,1,2,2,1,5,3,3,5],
[1,2,2,1,3,1,1,3,4,5,5,4,4,5,5,4,1,2,2,1,6,3,3,6],
[1,2,2,1,3,1,1,3,4,5,5,4,4,5,5,4,1,2,2,1,5,4,4,5],
[1,2,2,1,3,1,1,3,4,5,5,4,4,5,5,4,1,2,2,1,6,4,4,6],
[1,2,2,1,3,1,1,3,4,5,5,4,4,5,5,4,1,2,2,1,5,6,6,5],
[1,2,2,1,3,1,1,3,4,5,5,4,4,5,5,4,1,2,2,1,6,7,7,6],
[1,2,2,1,2,3,3,2,3,1,1,3,3,1,1,3,1,2,2,1,4,5,5,4],
[1,2,2,1,2,3,3,2,4,1,1,4,4,1,1,4,1,2,2,1,3,4,4,3],
[1,2,2,1,2,3,3,2,4,1,1,4,4,1,1,4,1,2,2,1,3,5,5,3],
[1,2,2,1,2,3,3,2,4,1,1,4,4,1,1,4,1,2,2,1,5,4,4,5],
[1,2,2,1,2,3,3,2,4,1,1,4,4,1,1,4,1,2,2,1,5,6,6,5],
[1,2,2,1,2,3,3,2,3,4,4,3,3,4,4,3,1,2,2,1,4,1,1,4],
[1,2,2,1,2,3,3,2,3,4,4,3,3,4,4,3,1,2,2,1,5,1,1,5],
[1,2,2,1,2,3,3,2,3,4,4,3,3,4,4,3,1,2,2,1,4,5,5,4],
[1,2,2,1,2,3,3,2,3,4,4,3,3,4,4,3,1,2,2,1,5,6,6,5],
[1,2,2,1,2,3,3,2,4,5,5,4,4,5,5,4,1,2,2,1,3,1,1,3],
[1,2,2,1,2,3,3,2,4,5,5,4,4,5,5,4,1,2,2,1,5,1,1,5],
[1,2,2,1,2,3,3,2,4,5,5,4,4,5,5,4,1,2,2,1,6,1,1,6],
[1,2,2,1,2,3,3,2,4,5,5,4,4,5,5,4,1,2,2,1,3,4,4,3],
[1,2,2,1,2,3,3,2,4,5,5,4,4,5,5,4,1,2,2,1,3,6,6,3],
[1,2,2,1,2,3,3,2,4,5,5,4,4,5,5,4,1,2,2,1,5,4,4,5],
[1,2,2,1,2,3,3,2,4,5,5,4,4,5,5,4,1,2,2,1,6,4,4,6],
[1,2,2,1,2,3,3,2,4,5,5,4,4,5,5,4,1,2,2,1,5,6,6,5],
[1,2,2,1,2,3,3,2,4,5,5,4,4,5,5,4,1,2,2,1,6,7,7,6],
[1,2,2,1,3,4,4,3,2,1,1,2,2,1,1,2,1,2,2,1,4,3,3,4],
[1,2,2,1,3,4,4,3,2,1,1,2,2,1,1,2,1,2,2,1,5,3,3,5],
[1,2,2,1,3,4,4,3,2,1,1,2,2,1,1,2,1,2,2,1,4,5,5,4],
[1,2,2,1,3,4,4,3,2,1,1,2,2,1,1,2,1,2,2,1,5,6,6,5],
[1,2,2,1,3,4,4,3,4,1,1,4,4,1,1,4,1,2,2,1,2,3,3,2],
[1,2,2,1,3,4,4,3,4,1,1,4,4,1,1,4,1,2,2,1,2,5,5,2],
[1,2,2,1,3,4,4,3,4,1,1,4,4,1,1,4,1,2,2,1,5,3,3,5],
[1,2,2,1,3,4,4,3,4,1,1,4,4,1,1,4,1,2,2,1,5,6,6,5],
[1,2,2,1,3,4,4,3,5,1,1,5,5,1,1,5,1,2,2,1,2,3,3,2],
[1,2,2,1,3,4,4,3,5,1,1,5,5,1,1,5,1,2,2,1,2,5,5,2],
[1,2,2,1,3,4,4,3,5,1,1,5,5,1,1,5,1,2,2,1,2,6,6,2],
[1,2,2,1,3,4,4,3,5,1,1,5,5,1,1,5,1,2,2,1,4,3,3,4],
[1,2,2,1,3,4,4,3,5,1,1,5,5,1,1,5,1,2,2,1,6,3,3,6],
[1,2,2,1,3,4,4,3,5,1,1,5,5,1,1,5,1,2,2,1,4,5,5,4],
[1,2,2,1,3,4,4,3,5,1,1,5,5,1,1,5,1,2,2,1,4,6,6,4],
[1,2,2,1,3,4,4,3,5,1,1,5,5,1,1,5,1,2,2,1,6,5,5,6],

[1,2,2,1,3,4,4,3,5,1,1,5,5,1,1,5,1,2,2,1,6,7,7,6],
[1,2,2,1,3,4,4,3,2,3,3,2,2,3,3,2,1,2,2,1,4,1,1,4],
[1,2,2,1,3,4,4,3,2,3,3,2,2,3,3,2,1,2,2,1,5,1,1,5],
[1,2,2,1,3,4,4,3,2,3,3,2,2,3,3,2,1,2,2,1,4,5,5,4],
[1,2,2,1,3,4,4,3,2,3,3,2,2,3,3,2,1,2,2,1,5,6,6,5],
[1,2,2,1,3,4,4,3,2,5,5,2,2,5,5,2,1,2,2,1,4,1,1,4],
[1,2,2,1,3,4,4,3,2,5,5,2,2,5,5,2,1,2,2,1,5,1,1,5],
[1,2,2,1,3,4,4,3,2,5,5,2,2,5,5,2,1,2,2,1,6,1,1,6],
[1,2,2,1,3,4,4,3,2,5,5,2,2,5,5,2,1,2,2,1,4,3,3,4],
[1,2,2,1,3,4,4,3,2,5,5,2,2,5,5,2,1,2,2,1,5,3,3,5],
[1,2,2,1,3,4,4,3,2,5,5,2,2,5,5,2,1,2,2,1,6,3,3,6],
[1,2,2,1,3,4,4,3,2,5,5,2,2,5,5,2,1,2,2,1,4,6,6,4],
[1,2,2,1,3,4,4,3,2,5,5,2,2,5,5,2,1,2,2,1,5,6,6,5],
[1,2,2,1,3,4,4,3,2,5,5,2,2,5,5,2,1,2,2,1,6,7,7,6],
[1,2,2,1,3,4,4,3,4,3,3,4,4,3,3,4,1,2,2,1,2,1,1,2],
[1,2,2,1,3,4,4,3,4,3,3,4,4,3,3,4,1,2,2,1,5,1,1,5],
[1,2,2,1,3,4,4,3,4,3,3,4,4,3,3,4,1,2,2,1,2,5,5,2],
[1,2,2,1,3,4,4,3,4,3,3,4,4,3,3,4,1,2,2,1,5,6,6,5],
[1,2,2,1,3,4,4,3,5,3,3,5,5,3,3,5,1,2,2,1,2,1,1,2],
[1,2,2,1,3,4,4,3,5,3,3,5,5,3,3,5,1,2,2,1,4,1,1,4],
[1,2,2,1,3,4,4,3,5,3,3,5,5,3,3,5,1,2,2,1,6,1,1,6],
[1,2,2,1,3,4,4,3,5,3,3,5,5,3,3,5,1,2,2,1,2,5,5,2],
[1,2,2,1,3,4,4,3,5,3,3,5,5,3,3,5,1,2,2,1,2,6,6,2],
[1,2,2,1,3,4,4,3,5,3,3,5,5,3,3,5,1,2,2,1,4,5,5,4],
[1,2,2,1,3,4,4,3,5,3,3,5,5,3,3,5,1,2,2,1,4,6,6,4],
[1,2,2,1,3,4,4,3,5,3,3,5,5,3,3,5,1,2,2,1,6,5,5,6],
[1,2,2,1,3,4,4,3,5,3,3,5,5,3,3,5,1,2,2,1,6,7,7,6],
[1,2,2,1,3,4,4,3,4,5,5,4,4,5,5,4,1,2,2,1,2,1,1,2],
[1,2,2,1,3,4,4,3,4,5,5,4,4,5,5,4,1,2,2,1,5,1,1,5],
[1,2,2,1,3,4,4,3,4,5,5,4,4,5,5,4,1,2,2,1,6,1,1,6],
[1,2,2,1,3,4,4,3,4,5,5,4,4,5,5,4,1,2,2,1,2,3,3,2],
[1,2,2,1,3,4,4,3,4,5,5,4,4,5,5,4,1,2,2,1,2,6,6,2],
[1,2,2,1,3,4,4,3,4,5,5,4,4,5,5,4,1,2,2,1,5,3,3,5],
[1,2,2,1,3,4,4,3,4,5,5,4,4,5,5,4,1,2,2,1,6,3,3,6],
[1,2,2,1,3,4,4,3,4,5,5,4,4,5,5,4,1,2,2,1,5,6,6,5],
[1,2,2,1,3,4,4,3,4,5,5,4,4,5,5,4,1,2,2,1,6,7,7,6],
[1,2,2,1,3,4,4,3,5,6,6,5,5,6,6,5,1,2,2,1,2,1,1,2],
[1,2,2,1,3,4,4,3,5,6,6,5,5,6,6,5,1,2,2,1,4,1,1,4],
[1,2,2,1,3,4,4,3,5,6,6,5,5,6,6,5,1,2,2,1,6,1,1,6],
[1,2,2,1,3,4,4,3,5,6,6,5,5,6,6,5,1,2,2,1,7,1,1,7],
[1,2,2,1,3,4,4,3,5,6,6,5,5,6,6,5,1,2,2,1,2,3,3,2],
[1,2,2,1,3,4,4,3,5,6,6,5,5,6,6,5,1,2,2,1,2,5,5,2],
[1,2,2,1,3,4,4,3,5,6,6,5,5,6,6,5,1,2,2,1,2,7,7,2],
[1,2,2,1,3,4,4,3,5,6,6,5,5,6,6,5,1,2,2,1,4,3,3,4],
[1,2,2,1,3,4,4,3,5,6,6,5,5,6,6,5,1,2,2,1,6,3,3,6],
[1,2,2,1,3,4,4,3,5,6,6,5,5,6,6,5,1,2,2,1,7,3,3,7],
[1,2,2,1,3,4,4,3,5,6,6,5,5,6,6,5,1,2,2,1,4,5,5,4],
[1,2,2,1,3,4,4,3,5,6,6,5,5,6,6,5,1,2,2,1,4,7,7,4],

```

[1,2,2,1,3,4,4,3,5,6,6,5,5,6,6,5,1,2,2,1,6,5,5,6],
[1,2,2,1,3,4,4,3,5,6,6,5,5,6,6,5,1,2,2,1,7,5,5,7],
[1,2,2,1,3,4,4,3,5,6,6,5,5,6,6,5,1,2,2,1,6,7,7,6],
[1,2,2,1,3,4,4,3,5,6,6,5,5,6,6,5,1,2,2,1,7,8,8,7]] : int list list
length it;
val it = 108 : int

```

13

Programming

The calculation of the examples for nfirstq, kconcat, kmul, etc. are based on the SML/NJ-program All.sml. For programmers it will be easy to run the SML/NJ- program All.sml and others. For non-programmers it might be a challenge to try to install the stuff on an actual machine. The package runs well on a NeXT machine with SML/NJ v.0.9.8, Feb 11, 1993

Sources are at:

<http://www.thinkartlab.com/pkl/SML-sources.NJ/All.sml>

<http://www.thinkartlab.com/pkl/pcl-lab.htm>

<http://www.thinkartlab.com/pkl/tm/mg-buch.htm>

Finally, the file: ALL-MG-nov2012.sml, will run on all SML/NJ versions higher 110.40.

<http://www.thinkartlab.com/pkl/SML-sources.NJ/ALL-MG-nov2012.sml>

Downloads

System SML/NJ: <http://www.smlnj.org/>

Morphogrammatics: <http://www.thinkartlab.com/pkl/SML-sources.NJ/ALL-MG-nov2012.sml>

Book Morphogrammatik: <http://www.thinkartlab.com/pkl/media/mg-book.pdf>

14

<http://www.cs.bham.ac.uk/~sjv/teaching/models/handout1.pdf>

<http://www.cs.jhu.edu/~jason/465/PDFSlides/lect17-fsmbuild.pdf>

http://www.cs.brown.edu/~jes/book/pdfs/ModelsOfComputation_Chapter4.pdf

www.mgt.ncu.edu.tw/~ylchen/dismath/chap06.ppt

<http://www.eecs.berkeley.edu/~bh/v3ch1/fsm.html>

<http://www.mmnt.net/db/0/ftp5.gwdg.de/pub/languages/funet.fi/ml/sml/75>

<ftp://ftp.cis.upenn.edu/pub/sml%23/smlnj.readme>

15

```
- kconcat [1,2][1,2,3,1];
```

```
val it =
```

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

```
- kconcat [1,2,3][1,2,3];
```

```
val it =
```

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

```
- kconcat[1,2,3] [1,2,3,4];
```

```
val it =
```

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



```
[1,2,3,3,4,5,6],[1,2,3,4,3,5,6],[1,2,3,4,5,3,6],[1,2,3,4,5,6,3],
[1,2,3,4,5,6,7]] : int list list
```

16

[illegible]

[illegible]

```
[F,F,F,F,F,F,F,F,F,F,C,F,F],[F,F,F,F,F,F,C,F,F,F,F,F,F,F],  
[F,F,F,F,F,F,F,F,F,F,C,F,F,F],[F,F,F,F,F,F,F,F,F,C,F,F,F,F],  
[F,F,F,F,F,F,F,F,F,F,F,F,F,F]] : fc list list
```

```
length it;  
val it = 203 : int  
-
```

17

```
- allKLORs 4;  
val it =  
[[O,O,O,O,O,O],[O,O,O,O,O,R],[O,O,O,O,R,O],[O,O,O,O,R,R],[O,O,O,R,O,O],  
[O,O,O,R,O,R],[O,O,O,R,R,O],[O,O,O,R,R,R],[O,O,R,O,O,O],[O,O,R,O,O,R],  
[O,O,R,O,R,O],[O,O,R,O,R,R],[O,O,R,R,O,O],[O,O,R,R,O,R],[O,O,R,R,R,O],  
[O,O,R,R,R,R],[O,R,O,O,O,O],[O,R,O,O,O,R],[O,R,O,O,R,O],[O,R,O,O,R,R],  
[O,R,O,R,O,O],[O,R,O,R,O,R],[O,R,O,R,R,O],[O,R,O,R,R,R],[O,R,R,O,O,O],  
[O,R,R,O,O,R],[O,R,R,O,R,O],[O,R,R,O,R,R],[O,R,R,R,O,O],[O,R,R,R,O,R],  
[O,R,R,R,R,O],[O,R,R,R,R,R],[R,O,O,O,O,O],[R,O,O,O,O,R],[R,O,O,O,R,O],  
[R,O,O,O,R,R],[R,O,O,R,O,O],[R,O,O,R,O,R],[R,O,O,R,R,O],[R,O,O,R,R,R],  
[R,O,R,O,O,O],[R,O,R,O,O,R],[R,O,R,O,R,O],[R,O,R,O,R,R],[R,O,R,R,O,O],  
[R,O,R,R,O,R],[R,O,R,R,R,O],[R,O,R,R,R,R],[R,R,O,O,O,O],[R,R,O,O,O,R],  
[R,R,O,O,R,O],[R,R,O,O,R,R],[R,R,O,R,O,O],[R,R,O,R,O,R],[R,R,O,R,R,O],  
[R,R,O,R,R,R],[R,R,R,O,O,O],[R,R,R,O,O,R],[R,R,R,O,R,O],[R,R,R,O,R,R],  
[R,R,R,R,O,O],[R,R,R,R,O,R],[R,R,R,R,R,O],[R,R,R,R,R,R],[O,L,L,O,L,L],  
[O,L,L,O,L,K],[O,L,L,O,K,L],[O,L,L,O,K,K],[O,L,L,R,L,L],[O,L,L,R,L,K],  
[O,L,L,R,K,L],[O,L,L,R,K,K],[O,L,K,O,L,L],[O,L,K,O,L,K],[O,L,K,O,K,L],  
[O,L,K,O,K,K],[O,L,K,R,L,L],[O,L,K,R,L,K],[O,L,K,R,K,L],[O,L,K,R,K,K],  
[O,K,L,O,L,L],[O,K,L,O,L,K],[O,K,L,O,K,L],[O,K,L,O,K,K],[O,K,L,R,L,L],  
[O,K,L,R,L,K],[O,K,L,R,K,L],[O,K,L,R,K,K],[O,K,K,O,L,L],[O,K,K,O,L,K],  
[O,K,K,O,K,L],[O,K,K,O,K,K],[O,K,K,R,L,L],[O,K,K,R,L,K],[O,K,K,R,K,L],  
[O,K,K,R,K,K],[R,L,L,O,L,L],[R,L,L,O,L,K],[R,L,L,O,K,L],[R,L,L,O,K,K],  
[R,L,L,R,L,L],[R,L,L,R,L,K],[R,L,L,R,K,L],[R,L,L,R,K,K],[R,L,K,O,L,L],  
[R,L,K,O,L,K],[R,L,K,O,K,L],[R,L,K,O,K,K],[R,L,K,R,L,L],[R,L,K,R,L,K],  
[R,L,K,R,K,L],[R,L,K,R,K,K],[R,K,L,O,L,L],[R,K,L,O,L,K],[R,K,L,O,K,L],  
[R,K,L,O,K,K],[R,K,L,R,L,L],[R,K,L,R,L,K],[R,K,L,R,K,L],[R,K,L,R,K,K],  
[R,K,K,O,L,L],[R,K,K,O,L,K],[R,K,K,O,K,L],[R,K,K,O,K,K],[R,K,K,R,L,L],  
[R,K,K,R,L,K],[R,K,K,R,K,L],[R,K,K,R,K,K],[L,L,O,L,O,L],[L,L,O,L,O,K],  
[L,L,O,L,R,L],[L,L,O,L,R,K],[L,L,O,K,O,L],[L,L,O,K,O,K],[L,L,O,K,R,L],  
[L,L,O,K,R,K],[L,L,R,L,O,L],[L,L,R,L,O,K],[L,L,R,L,R,L],[L,L,R,L,R,K],  
[L,L,R,K,O,L],[L,L,R,K,O,K],[L,L,R,K,R,L],[L,L,R,K,R,K],[L,K,O,L,O,L],  
[L,K,O,L,O,K],[L,K,O,L,R,L],[L,K,O,L,R,K],[L,K,O,K,O,L],[L,K,O,K,O,K],  
[L,K,O,K,R,L],[L,K,O,K,R,K],[L,K,R,L,O,L],[L,K,R,L,O,K],[L,K,R,L,R,L],  
[L,K,R,L,R,K],[L,K,R,K,O,L],[L,K,R,K,O,K],[L,K,R,K,R,L],[L,K,R,K,R,K],  
[K,L,O,L,O,L],[K,L,O,L,O,K],[K,L,O,L,R,L],[K,L,O,L,R,K],[K,L,O,K,O,L],  
[K,L,O,K,O,K],[K,L,O,K,R,L],[K,L,O,K,R,K],[K,L,R,L,O,L],[K,L,R,L,O,K],  
[K,L,R,L,R,L],[K,L,R,L,R,K],[K,L,R,K,O,L],[K,L,R,K,O,K],[K,L,R,K,R,L],  
[K,L,R,K,R,K],[K,K,O,L,O,L],[K,K,O,L,O,K],[K,K,O,L,R,L],[K,K,O,L,R,K],  
[K,K,O,K,O,L],[K,K,O,K,O,K],[K,K,O,K,R,L],[K,K,O,K,R,K],[K,K,R,L,O,L],  
[K,K,R,L,O,K],[K,K,R,L,R,L],[K,K,R,L,R,K],[K,K,R,K,O,L],[K,K,R,K,O,K],  
[K,K,R,K,R,L],[K,K,R,K,R,K],[L,O,L,L,L,O],[L,O,L,L,L,R],[L,O,L,L,K,O],  
[L,O,L,L,K,R],[L,O,L,K,L,O],[L,O,L,K,L,R],[L,O,L,K,K,O],[L,O,L,K,K,R],  
[L,O,K,L,L,O],[L,O,K,L,L,R],[L,O,K,L,K,O],[L,O,K,L,K,R],[L,O,K,K,L,O],  
[L,O,K,K,L,R],[L,O,K,K,K,O],[L,O,K,K,K,R],[L,R,L,L,L,O],[L,R,L,L,L,R],  
[L,R,L,L,K,O],[L,R,L,L,K,R],[L,R,L,K,L,O],[L,R,L,K,L,R],[L,R,L,K,K,O],  
[L,R,L,K,K,R],[L,R,K,L,L,O],[L,R,K,L,L,R],[L,R,K,L,K,O],[L,R,K,L,K,R],  
[L,R,K,K,L,O],[L,R,K,K,L,R],[L,R,K,K,K,O],[L,R,K,K,K,R],[K,O,L,L,L,O],  
[K,O,L,L,L,R],[K,O,L,L,K,O],[K,O,L,L,K,R],[K,O,L,K,L,O],[K,O,L,K,L,R],  
[K,O,L,K,K,O],[K,O,L,K,K,R],[K,O,K,L,L,O],[K,O,K,L,L,R],[K,O,K,L,K,O],  
[K,O,K,L,K,R],[K,O,K,K,L,O],[K,O,K,K,L,R],[K,O,K,K,K,O],[K,O,K,K,K,R],  
[K,R,L,L,L,O],[K,R,L,L,L,R],[K,R,L,L,K,O],[K,R,L,L,K,R],[K,R,L,K,L,O],  
[K,R,L,K,L,R],[K,R,L,K,K,O],[K,R,L,K,K,R],[K,R,K,L,L,O],[K,R,K,L,L,R],  
[K,R,K,L,K,O],[K,R,K,L,K,R],[K,R,K,K,L,O],[K,R,K,K,L,R],[K,R,K,K,K,O],  
[K,R,K,K,K,R],[O,O,O,L,L,L],[O,O,O,L,L,K],[O,O,O,L,K,L],[O,O,O,L,K,K],  
[O,O,O,K,L,L],[O,O,O,K,L,K],[O,O,O,K,K,L],[O,O,O,K,K,K],[O,O,R,L,L,L],  
[O,O,R,L,L,K],[O,O,R,L,K,L],[O,O,R,L,K,K],[O,O,R,K,L,L],[O,O,R,K,L,K],  
[O,O,R,K,K,L],[O,O,R,K,K,K],[O,R,O,L,L,L],[O,R,O,L,L,K],[O,R,O,L,K,L],  
[O,R,O,L,K,K],[O,R,O,K,L,L],[O,R,O,K,L,K],[O,R,O,K,K,L],[O,R,O,K,K,K],
```

[O,R,R,L,L,L],[O,R,R,L,L,K],[O,R,R,L,K,L],[O,R,R,L,K,K],[O,R,R,K,L,L],
[O,R,R,K,L,K],[O,R,R,K,K,L],[O,R,R,K,K,K],[R,O,O,L,L,L],[R,O,O,L,L,K],
[R,O,O,L,K,L],[R,O,O,L,K,K],[R,O,O,K,L,L],[R,O,O,K,L,K],[R,O,O,K,K,L],
[R,O,O,K,K,K],[R,O,R,L,L,L],[R,O,R,L,L,K],[R,O,R,L,K,L],[R,O,R,L,K,K],
[R,O,R,K,L,L],[R,O,R,K,L,K],[R,O,R,K,K,L],[R,O,R,K,K,K],[R,R,O,L,L,L],
[R,R,O,L,L,K],[R,R,O,L,K,L],[R,R,O,L,K,K],[R,R,O,K,L,L],[R,R,O,K,L,K],
[R,R,O,K,K,L],[R,R,O,K,K,K],[R,R,R,L,L,L],[R,R,R,L,L,K],[R,R,R,L,K,L],
[R,R,R,L,K,K],[R,R,R,K,L,L],[R,R,R,K,L,K],[R,R,R,K,K,L],[R,R,R,K,K,K],
[O,L,L,L,O,O],[O,L,L,L,O,R],[O,L,L,L,R,O],[O,L,L,L,R,R],[O,L,L,K,O,O],
[O,L,L,K,O,R],[O,L,L,K,R,O],[O,L,L,K,R,R],[O,L,K,L,O,O],[O,L,K,L,O,R],
[O,L,K,L,R,O],[O,L,K,L,R,R],[O,L,K,K,O,O],[O,L,K,K,O,R],[O,L,K,K,R,O],
[O,L,K,K,R,R],[O,K,L,L,O,O],[O,K,L,L,O,R],[O,K,L,L,R,O],[O,K,L,L,R,R],
[O,K,L,K,O,O],[O,K,L,K,O,R],[O,K,L,K,R,O],[O,K,L,K,R,R],[O,K,K,L,O,O],
[O,K,K,L,O,R],[O,K,K,L,R,O],[O,K,K,L,R,R],[O,K,K,K,O,O],[O,K,K,K,O,R],
[O,K,K,K,R,O],[O,K,K,K,R,R],[R,L,L,L,O,O],[R,L,L,L,O,R],[R,L,L,L,R,O],
[R,L,L,L,R,R],[R,L,L,K,O,O],[R,L,L,K,O,R],[R,L,L,K,R,O],[R,L,L,K,R,R],
[R,L,K,L,O,O],[R,L,K,L,O,R],[R,L,K,L,R,O],[R,L,K,L,R,R],[R,L,K,K,O,O],
[R,L,K,K,O,R],[R,L,K,K,R,O],[R,L,K,K,R,R],[R,K,L,L,O,O],[R,K,L,L,O,R],
[R,K,L,L,R,O],[R,K,L,L,R,R],[R,K,L,K,O,O],[R,K,L,K,O,R],[R,K,L,K,R,O],
[R,K,L,K,R,R],[R,K,K,L,O,O],[R,K,K,L,O,R],[R,K,K,L,R,O],[R,K,K,L,R,R],
[R,K,K,K,O,O],[R,K,K,K,O,R],[R,K,K,K,R,O],[R,K,K,K,R,R],[L,L,O,O,L,O],
[L,L,O,O,L,R],[L,L,O,O,K,O],[L,L,O,O,K,R],[L,L,O,R,L,O],[L,L,O,R,L,R],
[L,L,O,R,K,O],[L,L,O,R,K,R],[L,L,R,O,L,O],[L,L,R,O,L,R],[L,L,R,O,K,O],
[L,L,R,O,K,R],[L,L,R,R,L,O],[L,L,R,R,L,R],[L,L,R,R,K,O],[L,L,R,R,K,R],
[L,K,O,O,L,O],[L,K,O,O,L,R],[L,K,O,O,K,O],[L,K,O,O,K,R],[L,K,O,R,L,O],
[L,K,O,R,L,R],[L,K,O,R,K,O],[L,K,O,R,K,R],[L,K,R,O,L,O],[L,K,R,O,L,R],
[L,K,R,O,K,O],[L,K,R,O,K,R],[L,K,R,R,L,O],[L,K,R,R,L,R],[L,K,R,R,K,O],
[L,K,R,R,K,R],[K,L,O,O,L,O],[K,L,O,O,L,R],[K,L,O,O,K,O],[K,L,O,O,K,R],
[K,L,O,R,L,O],[K,L,O,R,L,R],[K,L,O,R,K,O],[K,L,O,R,K,R],[K,L,R,O,L,O],
[K,L,R,O,L,R],[K,L,R,O,K,O],[K,L,R,O,K,R],[K,L,R,R,L,O],[K,L,R,R,L,R],
[K,L,R,R,K,O],[K,L,R,R,K,R],[K,K,O,O,L,O],[K,K,O,O,L,R],[K,K,O,O,K,O],
[K,K,O,O,K,R],[K,K,O,R,L,O],[K,K,O,R,L,R],[K,K,O,R,K,O],[K,K,O,R,K,R],
[K,K,R,O,L,O],[K,K,R,O,L,R],[K,K,R,O,K,O],[K,K,R,O,K,R],[K,K,R,R,L,O],
[K,K,R,R,L,R],[K,K,R,R,K,O],[K,K,R,R,K,R],[L,O,L,O,O,L],[L,O,L,O,O,K],
[L,O,L,O,R,L],[L,O,L,O,R,K],[L,O,L,R,O,L],[L,O,L,R,O,K],[L,O,L,R,R,L],
[L,O,L,R,R,K],[L,O,K,O,O,L],[L,O,K,O,O,K],[L,O,K,O,R,L],[L,O,K,O,R,K],
[L,O,K,R,O,L],[L,O,K,R,O,K],[L,O,K,R,R,L],[L,O,K,R,R,K],[L,R,L,O,O,L],
[L,R,L,O,O,K],[L,R,L,O,R,L],[L,R,L,O,R,K],[L,R,L,R,O,L],[L,R,L,R,O,K],
[L,R,L,R,R,L],[L,R,L,R,R,K],[L,R,K,O,O,L],[L,R,K,O,O,K],[L,R,K,O,R,L],
[L,R,K,O,R,K],[L,R,K,R,O,L],[L,R,K,R,O,K],[L,R,K,R,R,L],[L,R,K,R,R,K],
[K,O,L,O,O,L],[K,O,L,O,O,K],[K,O,L,O,R,L],[K,O,L,O,R,K],[K,O,L,R,O,L],
[K,O,L,R,O,K],[K,O,L,R,R,L],[K,O,L,R,R,K],[K,O,K,O,O,L],[K,O,K,O,O,K],
[K,O,K,O,R,L],[K,O,K,O,R,K],[K,O,K,R,O,L],[K,O,K,R,O,K],[K,O,K,R,R,L],
[K,O,K,R,R,K],[K,R,L,O,O,L],[K,R,L,O,O,K],[K,R,L,O,R,L],[K,R,L,O,R,K],
[K,R,L,R,O,L],[K,R,L,R,O,K],[K,R,L,R,R,L],[K,R,L,R,R,K],[K,R,K,O,O,L],
[K,R,K,O,O,K],[K,R,K,O,R,L],[K,R,K,O,R,K],[K,R,K,R,O,L],[K,R,K,R,O,K],
[K,R,K,R,R,L],[K,R,K,R,R,K],[O,L,L,L,L,L],[O,L,L,L,L,K],[O,L,L,L,K,L],
[O,L,L,L,K,K],[O,L,L,K,L,L],[O,L,L,K,L,K],[O,L,L,K,K,L],[O,L,L,K,K,K],
[O,L,K,L,L,L],[O,L,K,L,L,K],[O,L,K,L,K,L],[O,L,K,L,K,K],[O,L,K,K,L,L],
[O,L,K,K,L,K],[O,L,K,K,K,L],[O,L,K,K,K,K],[O,K,L,L,L,L],[O,K,L,L,L,K],
[O,K,L,L,K,L],[O,K,L,L,K,K],[O,K,L,K,L,L],[O,K,L,K,L,K],[O,K,L,K,K,L],
[O,K,L,K,K,K],[O,K,K,L,L,L],[O,K,K,L,L,K],[O,K,K,L,K,L],[O,K,K,L,K,K],
[O,K,K,K,L,L],[O,K,K,K,L,K],[O,K,K,K,K,L],[O,K,K,K,K,K],[R,L,L,L,L,L],
[R,L,L,L,L,K],[R,L,L,L,L,K,L],[R,L,L,L,L,K,K],[R,L,L,L,K,L,L],[R,L,L,L,K,L,K],
[R,L,L,K,L,K],[R,L,L,K,K,K],[R,L,K,L,L,L],[R,L,K,L,L,K],[R,L,K,L,K,L],
[R,L,K,K,K,K],[R,L,K,K,L,L],[R,L,K,K,L,K],[R,L,K,K,K,L],[R,L,K,K,K,K],
[R,K,L,L,L,L],[R,K,L,L,L,K],[R,K,L,L,K,L],[R,K,L,L,K,K],[R,K,L,K,L,L],
[R,K,L,K,L,K],[R,K,L,K,K,L],[R,K,L,K,K,K],[R,K,K,L,L,L],[R,K,K,L,L,K],
[R,K,K,L,K,L],[R,K,K,L,K,K],[R,K,K,K,L,L],[R,K,K,K,L,K],[R,K,K,K,K,L],
[R,K,K,K,K,K],[L,L,O,L,L,L],[L,L,O,L,L,K],[L,L,O,L,K,L],[L,L,O,L,K,K],
[L,L,O,K,L,L],[L,L,O,K,L,K],[L,L,O,K,K,L],[L,L,O,K,K,K],[L,L,R,L,L,L],
[L,L,R,L,L,K],[L,L,R,L,K,L],[L,L,R,L,K,K],[L,L,R,K,L,L],[L,L,R,K,L,K],
[L,L,R,K,K,L],[L,L,R,K,K,K],[L,K,O,L,L,L],[L,K,O,L,L,K],[L,K,O,L,K,L],
[L,K,O,L,K,K],[L,K,O,K,L,L],[L,K,O,K,L,K],[L,K,O,K,K,L],[L,K,O,K,K,K],
[L,K,R,L,L,L],[L,K,R,L,L,K],[L,K,R,L,K,L],[L,K,R,L,K,K],[L,K,R,K,L,L],

[illegible]

[K,L,L,L,K,L],[K,L,L,L,K,K],[K,L,L,K,L,L],[K,L,L,K,L,K],[K,L,L,K,K,L],
[K,L,L,K,K,K],[K,L,K,L,L,L],[K,L,K,L,L,K],[K,L,K,L,K,L],[K,L,K,L,K,K],
[K,L,K,K,L,L],[K,L,K,K,L,K],[K,L,K,K,K,L],[K,L,K,K,K,K],[K,K,L,L,L,L],
[K,K,L,L,L,K],[K,K,L,L,K,L],[K,K,L,L,K,K],[K,K,L,K,L,L],[K,K,L,K,L,K],
[K,K,L,K,K,L],[K,K,L,K,K,K],[K,K,K,L,L,L],[K,K,K,L,L,K],[K,K,K,L,K,L],
[K,K,K,L,K,K],[K,K,K,K,L,L],[K,K,K,K,L,K],[K,K,K,K,K,L],[K,K,K,K,K,K]]

: klor list list