

— vordenker-archive —

Rudolf Kaehr

(1942-2016)

Title

Exemplarischer Unterschied von Turing Maschine und morphogrammatischer
Maschinerie

Übergang von der Turing Berechnungsmaschine zur morphogrammatischen Umformungsmaschinerie

Archive-Number / Categories

3_26 / K09, K08, K02

Publication Date

2013

Keywords / Topics

- DEFINITION KLASSISCHER TURING-MASCHINEN : Definition einer Turingmaschine, Definition morphogrammatischer Turing-Maschinen, Spekulative Aspekte einer transTM, Iterative und akkretive Mehrdeutigkeit, Graphematics of the Stirling Turn: From equivalence classes to iterability
- WEITERE BEISPIELE : Beispiele für morphoTM, Beispiele für morphoFSM
- WHERE ARE THE DIAMONDS LEFT? : Intra-contextural interplay of morphic machines

Disciplines

Computer Science, Artificial Intelligence and Robotics, Cybernetics, Logic and Foundations of Mathematics, Epistemology, Systems Architecture and Theory and Algorithms, Theory of Science

Abstract

Der exemplarischer Unterschied einer Turing Maschine und ihrem morphogrammatischen Analogon, der morphoTM, wird in didaktischer Absicht eingeführt als eine Dekonstruktion der klassischen Begriffsbildungen der Definition einer Turingmaschine. Der Text ist in didaktischer Absicht verfasst, und lässt sich in vielen Teilen weitgehend ergänzen. Einige spekulative Motive werden entwickelt. Es wird auch auf die Publikation hingewiesen: Marc Jongen (Hrsg.) *Was wird Denken heißen? Kognition und Psyche im posthumanen Zeitalter*, Wilhelm Fink Verlag, 2013

Citation Information / How to cite

Rudolf Kaehr: "Exemplarischer Unterschied von Turing Maschine und morphogrammatischer Maschinerie", www.vordenker.de (Sommer Edition, 2017) J. Paul (Ed.),
http://www.vordenker.de/rk/rk_Unterschied-von-Turing-Maschine-und-morphogrammatischer-Maschinerie_2013.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 | |

Exemplarischer Unterschied von Turing Maschine und morphogrammatischer Maschinerie

Übergang von der Turing Berechnungsmaschine zur morphogrammatischen Umformungsmaschinerie

Rudolf Kaehr Dr. phil[@]

Copyright ThinkArt Lab ISSN 2041-4358

Abstract

Der exemplarischer Unterschied einer Turing Maschine und ihrem morphogrammatischen Analogon, der *morphoTM*, wird in didaktischer Absicht eingeführt als eine Dekonstruktion der klassischen Begriffsbildungen der Definition einer Turingmaschine.

Der Text ist in didaktischer Absicht verfasst, und lässt sich in vielen Teilen weitgehend ergänzen. Einige spekulative Motive werden entwickelt. Es wird auch auf die Publikation hingewiesen: *Marc Jongen (Hrsg.) Was wird Denken heißen? Kognition und Psyche im posthumanen Zeitalter, Wilhelm Fink Verlag, 2013*

(Work in progress, vers. 0.6, April 2013)

1. Definition klassischer Turing-Maschinen

1.1. Motivation für eine Dekonstruktion von TMs

Ziel ist es, in einem ersten Schritt einer Dekonstruktion der Konzepte und Techniken der klassischen Turingmaschine, eine Idee einer dazu äquivalenten Konzeption und Apparatur einer morphogrammatischen (Turing-)Maschine einzuführen.

Eine Dekonstruktion in diesem Sinne geht von einem etablierten klassischen Gebilde (Konzept, Konstruktion, Konvention) aus und hinterfragt schrittweise alle seine Grundvoraussetzungen.

Die Mechanismen der Dekonstruktion (Derrida), *Verschiebung* und *Verkehrung* (renversement, déplacement), werden hier nicht speziell thematisiert und kommen nur hintergündig ins Spiel. Sie sind in aller Ausführlichkeit in früheren Publikationen zur Genüge expliziert worden.

Dekonstruktion heisst hier vorerst Demontage, gefolgt von dem Versuch eines 'subversiven' Re-Engineering.

Daher haben sich alle Elemente einer Turing-Maschine einer solchen Prozedur der Dekonstruktion zu unterziehen.

Die **Elemente** einer Turing-Maschine sind:

Band,
Les- und Schreibkopf,
Bewegung des Lesekopfes, L, R
Zustände, z,
Anfangs- und Endzustand
Übergangsfunktion, $f(z, n)$.

Erste Dekonstruktionstabelle

Dekonstruktion Strategien	Turingmaschine TM	morphoTM
Tabularisierung	lineares Band	Topologie
poly – Crossbar	1 – Les- / Schreibkopf	Doppelkopf
multi – path	Bewegung L, R, S	$\{L, R, S\}_i, i \in s(m)$
Thematisierung	Zustände	Differenzen
Trans – Formation	Übergangsfunktion	Differenzierung

In einem aller *ersten* Schritt einer solchen Dekonstruktion ist es nicht abwegig, Morphogramme erst einmal bloss als einen speziellen, eventuell neuen *Datentyp* einer TM anzusehen. Und daher vom klassischen Datentyp einer *Zeichenreihe* definiert über einem Alphabet über die Abstraktionsklasse der Stirlingverteilung von Zeichenreihen zum postulierten 'Datentyp' der Morphogramme der Tritostufe überzugehen.

Im Gegensatz zu Zeichenreihen der TM werden Konstellationen von Kenogrammen für die morphogrammatische Maschine eingesetzt. Solche Konstellationen sind Relationsgebilde, d.h. Ordnungsstrukturen, allerdings nicht zwischen Zeichen, sondern zwischen Kenogrammen, und müssen daher tabular und nicht linear notiert werden. Die neuen Konstellationen werden über einer kenogrammatisch definierten Topologie definiert.

Es wird allerdings nicht leicht sein, diesen Datentyp via "Gödelisierung" im Nachhinein wieder zu linearisieren.

Ein *zweiter* Schritt weist unweigerlich neue strukturelle Eigenschaften des Verhaltens einer morphogrammatischen Turing Maschine, genannt, *morphoTM*, auf.

Diese betreffen vorerst die *Übergangsfunktion*. Aus der Übergangsfunktion als Abbildung (mapping) wird eine Aktion des Differenzierens (differentiation).

Solche *morphoTMs* gehören der Klasse der morphogrammatischen Finiten Automaten (Finite State Machines) an, und wurden in früheren Arbeiten als *morphoFSM* konzipiert und analysiert. Diese sind nicht eigentlich *State Machines*, sondern *Differentiation Machines*.

Zwangsläufig ergibt sich aus der Struktur der Morphogramme und ihrer Gesetzmäßigkeiten, dass die Vorgabe eines *Maschinenbandes* für eine Folge von linear geordneten Zeichenreihen, also Sukzessionen mit linearer Struktur, nicht mehr tragfähig ist.

Im Zuge einer solchen Dekonstruktion lässt sich die Unschuld des *Lesekopfes* nicht mehr halten. Morphogramme verlangen zu ihrer Lesbarkeit mindestens einen Doppelkopf. Was gelesen wird, um bei der Metapher zu verbleiben, ist nicht ein Zeichen in seiner atomaren Provinienz und Isolation, sondern einzig und allein Differenzen. Durchaus auch Differenzen zwischen Zeichen.

Damit entfällt ein weiterer wichtiger Baustein für eine morphogrammatische *Analogiebildung*.

Die einfachen *Leseoperationen* R und L , für die Bewegung des Lesekopfes nach rechts und nach links, setzen trivialerweise eine *lineare* Struktur des Lesebandes wie der zu lesenden Daten, Zeichenreihen, voraus.

Eine Dekonstruktion dieser Linearität hat die *Ordnungsstruktur* der Leseoperationen mitanzugeben, also $ord(L)$ und $ord(R)$. Die Ordnungsstruktur ord muss per Konvention nummeriert sein, num , die Gesamtoperation des Lesens und Schreibens ist somit $num(ord(Morphogram))$.

Damit ist auch klar, dass das simple L und R einer TM, ein Spezialfall von $num(ord)$ ist. Für num linear $succ(n) = n+1$ und $ord = 1$, ist $ord(num)$ reduziert auf L und R .

Die *Nummerierung* der Ordnungsstruktur ermöglicht eine Linearisierung. Diese basiert aber auf einer Konvention, und kann daher auch anders vorgenommen werden.

Das heisst also, dass die Bewegung rechts oder links, L oder R , der nummerierten Ordnungsstruktur entlang läuft, die für Morphograme nicht notwendigerweise linear sein muss, wie dies für den Fall der Zeichenreihen der TM notwendigerweise gilt.

Dies ist wiederum im strikten Gegensatz zur klassischen Definition. Weitere konservative Erweiterungen und Modifikationen der ursprünglichen Konzeption der TM, wie Mehr-Band, Mehr-Kopf oder Orakel Maschinen, usw., werden hier nicht thematisiert.

Ebenso ist die *Schrittoperation*, die von einem Zustand zu einem nächsten Zustand führt, für eine morphoTM nicht mehr eindeutig. Dieser Unterschied zwischen Eindeutigkeit und 'Mehrdeutigkeit' erscheint auf der Ebene einer (deterministischen) Turing-Maschine und ist nicht erst auf der Ebene nicht-deterministischer TMs angesiedelt.

Schon aus diesen wenigen Dekonstruktionschritten folgen weitreichende Konsequenzen für das Verhalten von *morphoTMs*. Insbesondere sind die verschiedenen *Interaktionen* (Koalitionen, Kooperationen, usw.) zwischen verschiedenen morphoTMs grundlegend verschieden vom klassischen Vorbild.

Die **schematische Differenz** zwischen dem Lesekopf einer Turing Maschine und einem morphoTM Doppelkopf wird im folgenden Diagramm dargestellt.

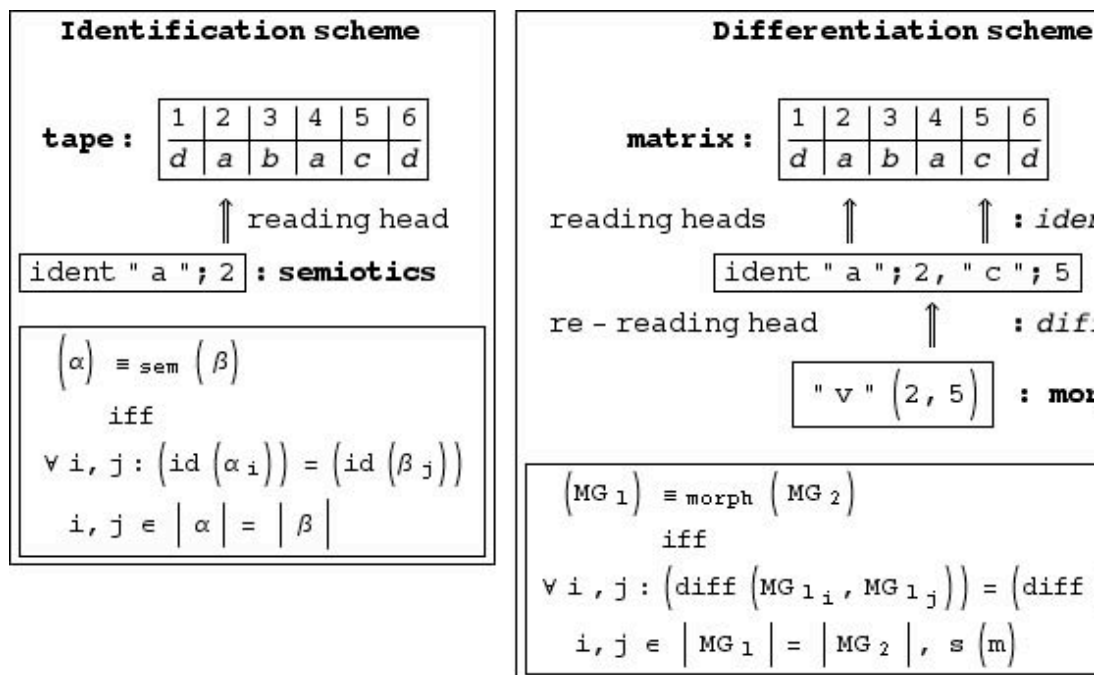
Für das Identifikationsschema gilt:

Zwei Zeichenreihen sind identisch, wenn sie in jedem Teilzeichen der Zeichenreihen übereinstimmen.

Für das Differenzenschema gilt:

Zwei Morphogramme sind äquivalent, wenn sie für jede Differenz der Morphogramme übereinstimmen.

Für beide gilt in diesem Falle, der Metamorphosen ausschliesst, dass ihre Inskriptionen semiotisch von gleicher Länge sind.



<http://memristors.memristics.com/MorphoFSM/Finite%20State%20Machines%20and%20Morphogramatics.pdf>

1.2. Definition einer Turingmaschine

1.2.1. Programm einer Turingmaschine

Aufgabe:

Einfügen eines Zeichens am Anfang und Ende der Zeichenkette (010011).

Band am Anfang (Anfangszustand): 010011

Band am Ende (Endzustand): a010011a .

Das **Turingmaschinenprogramm** besteht aus den sechs Befehlen (für 'leer' wird das

Zeichen # verwendet):

(z1, 0, R) := f(z1, 0) Lies und schreibe 0 und gehe nach rechts

(z1, 1, R) := f(z1, 1) Lies und schreibe 1 und gehe nach rechts

(z2, a, L) := f(z1, #) Lies Leerzeichen, schreibe a und gehe nach links

(z2, 0, L) := f(z2, 0) Lies und schreibe 0 und gehe nach links

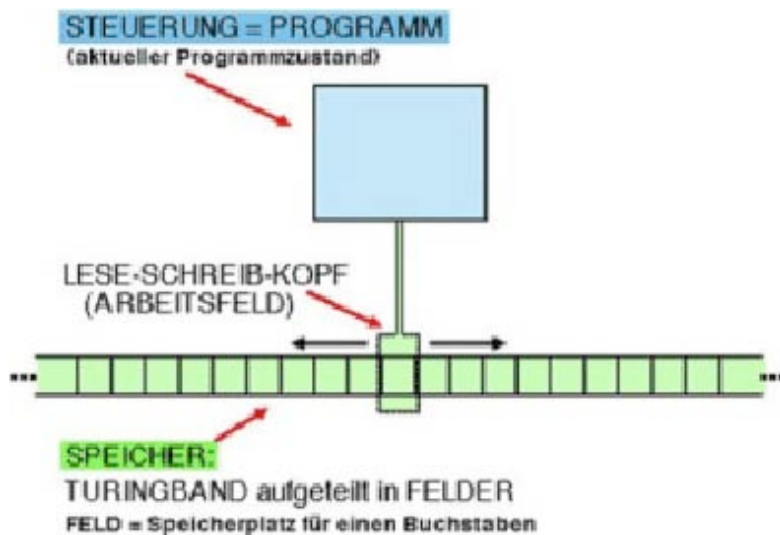
(z2, 1, L) := f(z2, 1) Lies und schreibe 1 und gehe nach links

(z3, a, R) := f(z2, #) Lies Leerzeichen, schreibe a und gehe nach rechts.

bedeutet das Leerzeichen (unbeschriebene Bandzelle).

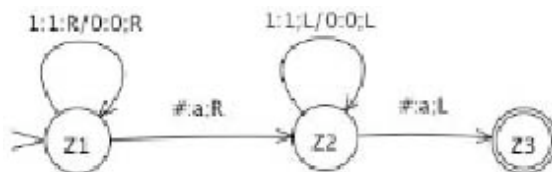
Ich folge hier der didaktischen Darstellung, die von Goldammer in Folie 005 gegeben hat.

http://www.vordenker.de/vgo/anmerkungen_leibniz_b.pdf



1.2.2. Zustandsgraph des Programmes einer Turingmaschine

Der **Zustandsgraph** veranschaulicht die beiden Aktionen, dem Wort am Ende ein Zeichen hinzuzufügen, wie auch rückwärts, am Anfang des Wortes das Zeichen "a" anzufügen.



Den Graphen liest oder arbeitet man wie folgt ab: (Ablauftabelle).

1.2.3. Ablauftabelle des Programmes einer Turingmaschine

Die Zeichenfolge lautet (Band im Anfangszustand): **010011**

t01 : Wir beginnen bei z1 : wir lesen 0 und schreiben 0 und bewegen uns nach R
 t02 : Wir sind noch in z1 : wir lesen 1 und schreiben 1 und bewegen uns nach R
 t03 : Wir sind noch in z1 : wir lesen 0 und schreiben 0 und bewegen uns nach R
 t04 : Wir sind noch in z1 : wir lesen 0 und schreiben 0 und bewegen uns nach R
 t05 : Wir sind noch in z1 : wir lesen 1 und schreiben 1 und bewegen uns nach R
 t06 : Wir sind noch in z1 : wir lesen 1 und schreiben 1 und bewegen uns nach R
 t07 : Wir sind noch in z1 : wir lesen # und schreiben a und bewegen uns nach L

Zwischenresultat : 010011a

t08 : Wir sind jetzt in z2 : wir lesen 1 und schreiben 1 und bewegen uns nach L
 t09 : Wir sind noch in z2 : wir lesen 1 und schreiben 1 und bewegen uns nach L
 t10 : Wir sind noch in z2 : wir lesen 0 und schreiben 0 und bewegen uns nach L
 t11 : Wir sind noch in z2 : wir lesen 0 und schreiben 0 und bewegen uns nach L
 t12 : Wir sind noch in z2 : wir lesen 1 und schreiben 1 und bewegen uns nach L
 t13 : Wir sind noch in z2 : wir lesen 0 und schreiben 0 und bewegen uns nach L
 t14 : Wir sind noch in z2 : wir lesen # und schreiben a und bewegen uns nach R
 t15 : Wir sind jetzt in z3 : in z3 hält die Maschine an.

Resultat: Die Zeichenfolge lautet (Band im Endzustand): **a010011a** .

1.2.4. Nachbemerungen

Für jemanden, der sich bis dahin nicht mit der Materie beschäftigt hat, ist es beruhigend zu wissen, dass sich Turing-Maschinen für kompliziertere Aufgaben modular aus einfachen Teilmaschinen zu "zusammengesetzten Turingmaschinen" aufbauen lassen. (Hermes, 1961, p. 45)

So ist direkt aus dem obigen Beispiel leicht ersichtlich, dass sich sowohl eine Rechts- wie eine Links-Lese/Schreibmaschine als unabhängige Module separieren lassen. Beide Module sind miteinander seriell verknüpft.

Hans Hermes hat in seinem wegweisenden Buch "*Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit*" schon 1961, basierend auf Arbeiten aus den fünfziger Jahren, alle entscheidenden Definitionen, Beweise und Konstruktionen für eine Theorie der Berechenbarkeit in aller Ausführlichkeit in deutscher Sprache verfasst.

Auf den Seiten 50/51 sind die wichtigsten Module für zusammengesetzte Turingmaschinen in einer formalen und einer graphischen Form aufgelistet.

Eine lesenswerte Darstellung mathematischer Art auf Deutsch gibt z.B. Jürgen Dassow mit:
http://theo.cs.uni-magdeburg.de/lehre03w/ti_if/ti_if02_f03.pdf

Oder auch:

Klaus Ambos-Spies, Turingmaschinen

http://www.math.uni-heidelberg.de/logic/md/lehre/fohlen_03_dvg2-pdf.pdf

Ebenso lesenswert in Englisch ist:

<http://webloria.loria.fr/~blackbur/courses/math/l1.pdf>, lectures (l1-l6)

1.3. Definition morphogrammatischer Turing-Maschinen

1.3.1. Morphogrammatisches Programm einer Turing-Maschine

Das **Turing Maschinenprogramm** für Morphogramme besteht aus den sechs Befehlen (für 'leer' wird das Zeichen # verwendet):

$(z1, E, R(s(m)))$	$:= f(z1, E)$	Lies und schreibe E und gehe nach rechts im Graphen
$(z1, N, R(s(m)))$	$:= f(z1, N)$	Lies und schreibe N und gehe nach rechts
$(z2, new, L(s(m)))$	$:= f(z1, \#)$	Lies Leerzeichen #, schreibe new und gehe nach links
$(z2, E, L(s(m)))$	$:= f(z2, E)$	Lies und schreibe E und gehe nach links
$(z2, N, L(s(m)))$	$:= f(z2, N)$	Lies und schreibe N und gehe nach links
$(z3, new, R(s(m)))$	$:= f(z2, \#)$	Lies Leerzeichen #, schreibe new und gehe nach rechts

Der Term *new* entspricht dem Zeichen "a" der TM, entstammt jedoch nicht aus einem vorgegebenen Alphabet, sondern wird in Abhängigkeit zur Struktur des vorgefundenen Wortes in der Topologie des TM-Bandes berechnet.

Die Funktion *new* ist daher nicht eindeutig, sondern kontextuell abhängig vom Morphogramm (MG) und wird als Aggregationsgrad "AG" durch $AG(MG)$ berechnet.

Beispiel:

$MG = [1,0,1,1,0]$

$AG([1,0,1,1,0]) = \{0,1\}$

$AG_{accr} = AG + 1.$

Der Trick der vorgestellten Implementierung besteht allerdings darin, dass die retro-grad rekursive Prozedur vorerst ausgeklammert, und der Wert für "new" vorgegeben, statt berechnet wird.

Morphogrammatische Interpretation des 'Ausgangszustandes'

Gegeben sei das Morphogram [010011], produziert werde das Morphogramm [a010011a].

Bevor die morphoTM etwas berechnen kann, muss sie das Morphogram als Anfangszustand lesen.

Diese Prozedure wird durch die EN-Struktur des Morphogrammes unterstützt. Daher soll als Erstes die EN-Struktur des Morphogrammes angegeben (berechnet) werden.

Das Morphogram [010011] hat die EN-Struktur:

- **ENstructure** [0,1,0,0,1,1];

val it =

[[],

[(1,2,N)],

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

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

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

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

Die EN-Struktur von [010011] als **Tabelle** mit **Nummerierung**, mit $N \equiv v$ und $E \equiv e$ ist:

[010011]	2	3	4	5	6
[]	v_1	e_2	e_4	v_7	v_{11}
□	□	v_3	v_5	e_8	e_{12}
□	□	□	e_6	v_9	v_{13}
□	□	□	□	v_{10}	v_{14}
□	□	□	□	□	e_{15}

Die EN-Struktur von [010011] nummeriert, ist die Linearform:

($N_1, E_2, N_3, E_4, N_5, E_6, N_7, E_8, N_9, N_{10}, N_{11}, E_{12}, N_{13}, N_{14}, E_{15}$).

Diagramm der FSM-Maschine der morphoTM[010011]

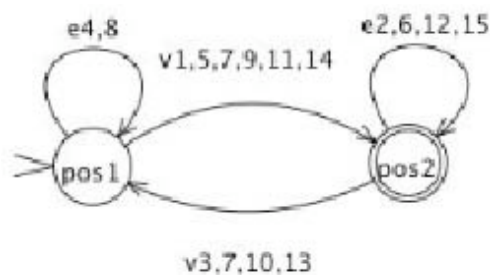
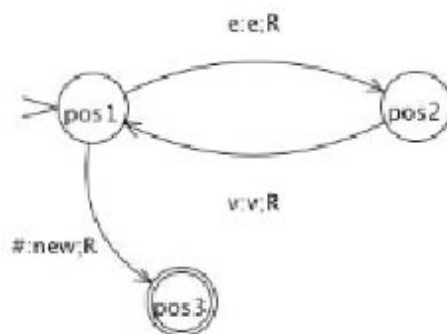
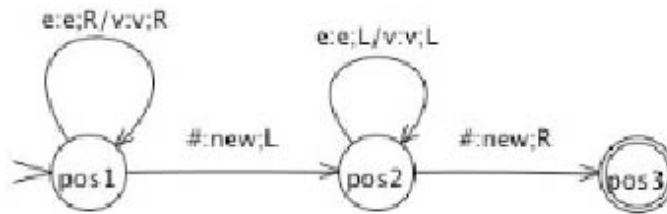


Diagramm der morphoTM-[01011]new (rechts)



Vollständiger Zustandsgraph für morphoTM-(new[010011]new)

(rechts+links)**Nummerierung der Subsysteme und Verortung (Position)**

Die Nummerierung der Subsysteme und deren Positionen des Morphogrammes $[0,1,0,0,1,1]$ werden durch die ML-Funktion *subsystem (n)* berechnet:

subsystems 6;

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])] : (int * int list) list

```

Die Berechnungen basieren auf dem Programmpaket:

System: SML/NJ: <http://www.smlnj.org/>,

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

Book Morphogramatik: <http://www.thinkartlab.com/pkl/media/mg-book.pdf>

Damit sind nun die Vorbereitungen gegeben, um den Job, dem Morphogram $[0,1,0,0,1,1]$ ein Element an das "Ende" und in einem zweiten Durchlauf (run) an den "Beginn" anfügen zu können.

Es wird noch angenommen, dass "a" als *new* weder "0" noch "1" ist, es kann also als "2" notiert werden.

Somit ist das Morphogramm, das gesucht oder konstruiert werden soll, das Morphogramm $[20100112]$.

Dieses Morphogramm hat die Trito-Standard-Normalform *tnf* $[1,2,3,2,2,3,3,1]$.

Diese Funktion, *tnf*, führt eine beliebige Notation für ein Morphogramm in seine kanonische Normalform zurück.

Die Normalform für $[2,0,1,0,0,1,1,2]$ ist berechnet durch:

- *tnf* $[2,0,1,0,0,1,1,2]$;

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

Somit ist endlich der **Task** für die morphogrammatistische Turing Maschine, *morphoTM*, klar definiert. Sie soll das Morphogramm $[1,2,3,2,2,3,3,1]$ aus dem Morphogramm $[0,1,0,0,1,1]$ erzeugen.

Diese Normalisierung der Morphogramme in eine kanonische Normalform erfolgt mehr in didaktischer denn in systematischer Absicht.

Berechnung des Resultats "rechts" gemäss vollständiger Zustandsgraph:

Die EN-Struktur für $[0,1,0,0,1,1]$ ist:

$(N_1, E_2, N_3, E_4, N_5, E_6, N_7, E_8, N_9, N_{10}, N_{11}, E_{12}, N_{13}, N_{14}, E_{15})$

Das Morphogramm lautet (Band im Anfangszustand): **$[0,1,0,0,1,1]$**

t01 : Wir beginnen bei pos1 : wir lesen N_1 , schreiben N , bewegen uns nach R

t02 : Wir sind noch in pos1 : wir lesen E₂, schreiben E, bewegen uns nach R
 t03 : Wir sind noch in pos1 : wir lesen N₃, schreiben N, bewegen uns nach R
 t04 : Wir sind noch in pos1 : wir lesen E, schreiben E, bewegen uns nach R
 t05 : Wir sind noch in pos1 : wir lesen N, schreiben N, bewegen uns nach R
 t06 : Wir sind noch in pos1 : wir lesen E, schreiben E, bewegen uns nach R
 t07 : Wir sind noch in pos1 : wir lesen N, schreiben N, bewegen uns nach R
 t08 : Wir sind jetzt in pos1 : wir lesen E, schreiben E, bewegen uns nach R
 t09 : Wir sind noch in pos1 : wir lesen N, schreiben N, bewegen uns nach R
 t10 : Wir sind noch in pos1 : wir lesen N, schreiben N, bewegen uns nach R
 t11 : Wir sind noch in pos1 : wir lesen N, schreiben N, bewegen uns nach R
 t12 : Wir sind noch in pos1 : wir lesen E, schreiben E, bewegen uns nach R
 t13 : Wir sind noch in pos1 : wir lesen N, schreiben N, bewegen uns nach R
 t12 : Wir sind noch in pos1 : wir lesen N, schreiben N, bewegen uns nach R
 t14 : Wir sind noch in pos1 : wir lesen E, schreiben E, bewegen uns nach R
 t15 : Wir sind noch in pos1 : wir lesen #, schreiben *new*, bewegen uns nach R

(*new*: hier sollte der Aggregationsgrad AG des Morphogrammes berechnet werden.)

... zweiter Durchlauf (links) ...

t30 : Wir sind noch in pos2 : wir lesen #, schreiben *new*, bewegen uns nach R
 t31 : Wir sind jetzt in pos3 : in pos3 hält die Maschine für den *zweiten* Durchlauf an.

Das Resultat "rechts" (erster Durchlauf) ist: [0,1,0,0,1,1,2]

Dasselbe wird nun wiederholt mit "links", allerdings mit dem neuen Start **[0,1,0,0,1,1,2]** :
Erstes, iteratives, Resultat für Durchlauf "rechts+links" in tnf: [1,2,3,2,2,3,3,1].

Dieser neue Start des ersten Durchlaufs [0,1,0,0,1,1,2] muss für den zweiten Durchlauf akzeptiert werden, da die zwei Durchläufe (runs), nach Vorlage, *hintereinander* ausgeführt werden sollen.

Endzustand

Die Morphogramme im *Endzustand* lauten (Band im Endzustand):

- a) im *iterativen* Durchgang: [2,0,1,0,0,1,1,2], d.h. in tnf **[1,2,3,2,2,3,3,1]** und
 b) im *akkretiven* Durchgang : [3,0,1,0,0,1,1,2], d.h. in tnf **[1,2,3,2,2,3,3,4]**.

Die gestellte Aufgabe, dem Morphogramm [0,1,0,0,1,1] am "Anfang" und am "Ende" ein neues Kenogramm anzufügen, hat somit *zwei* Lösungen. Die Anzahl der Lösungen wird durch AG_{accr}(MG) berechnet.

Standardform

- tnf [2,0,1,0,0,1,1,2];
 val it = [1,2,3,2,2,3,3,1] : int list

Die **EN-Struktur** des *ersten* Resultats, [1,2,3,2,2,3,3,1], ist gegeben durch die ML-Funktion *ENstructure*.

```
ENstructure[1,2,3,2,2,3,3,1]
- ENstructure [1,2,3,2,2,3,3,1] ;
val it =
[[],
[(1,2,N)],
[(1,3,N),(2,3,N)],
[(1,4,N),(2,4,E),(3,4,N)],
[(1,5,N),(2,5,E),(3,5,N),(4,5,E)],
[(1,6,N),(2,6,N),(3,6,E),(4,6,N),(5,6,N)],
```

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

Linearform des ersten Resultats

(N NN NEN NENE NNENN NNENE ENNNNNN).

Standardform des zweiten Resultats

```
- tnf[3,0,1,0,0,1,1,2];
val it = [1,2,3,2,2,3,3,4] : int list
```

Die **EN-Struktur** des *zweiten* Resultats, [1,2,3,2,2,3,3,4], ist gegeben durch die ML-Funktion *ENstructure*

```
- ENstructure [1,2,3,2,2,3,3,4];
val it =
[[],
[(1,2,N)],
[(1,3,N),(2,3,N)],
[(1,4,N),(2,4,E),(3,4,N)],
[(1,5,N),(2,5,E),(3,5,N),(4,5,E)],
[(1,6,N),(2,6,N),(3,6,E),(4,6,N),(5,6,N)],
[(1,7,N),(2,7,N),(3,7,E),(4,7,N),(5,7,N),(6,7,E)],
[(1,8,N),(2,8,N),(3,8,N),(4,8,N),(5,8,N),(6,8,N),(7,8,N)]]
: (int * int * EN) list list
```

Zweites Resultat "links":

ENstructure für [1,2,3,2,2,3,3,4] als Resultat "links" in Linearform:

(N, N,N, N,E,N, N,E,N,E, N,N,E,N,N N,N,E,N,N,E N,N,N,N,N,N).

Gesamtresultat der morphoTM für [0,1,0,0,1,1] (ohne explizite Nummerierung):

Erster iterative Durchgang:

(N, N,N N,E,N N,E,N,E, N,N,E,N,N N,N,E,N,N,E, **E**₂₂,N,N,N,N,N,N)

Zweiter akkretiver Durchgang:

(N, N,N, N,E,N, N,E,N,E, N,N,E,N,N N,N,E,N,N,E, **N**₂₂,N,N,N,N,N,N).

Die zwei Durchläufe, der iterative und der akkretive, unterscheiden sich einzig, jedoch signifikant, im Wert für das 22. Subsystem, $E_{22} \neq N_{22}$, alle anderen Differenzen sind gleich.

Nummerierung der Subsysteme

Die numerische Länge der beiden Resultate ist 8, ihre jeweilige Subsystemnummer wird errechnet durch die ML-Funktion *subsystems* (8):

```
- subsystems 8;
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]),(22,[7,8]),(23,[6,8]),(24,[5,8]),(25,[4,8]),(26,[3,8]),
(27,[2,8]),(28,[1,8])] : (int * int list) list
```

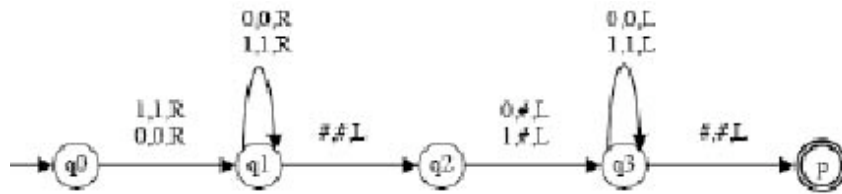
ACHTUNG: Diese Nummerierung durch "*subsystems* (n)" entspricht nicht der morphoFSM-Konvention!!!

1.3.2. The opposite operation: Simple Eraser

Simple Eraser.

"This Turing machine reads strings in the language given by the expression $(0,1)^*$ and replaces the right-most symbol by a blank (#)."

(Following Dr. Natalio Krasnogor, <http://www.cs.nott.ac.uk/~nxk/TEACHING/G53COM/G53COMLecture5.pdf>)



TM = ($\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, \#\}, \delta, q_0, \{p\}$) where δ is given by:

Q	Σ	0	1	#
q_0		$(q_1, 0, R)$	$(q_1, 1, R)$	$\emptyset$
q_1		$(q_1, 0, R)$	$(q_1, 1, R)$	$(q_2, \#, L)$
q_2		$(q_3, \#, L)$	$(q_3, \#, L)$	$\emptyset$
q_3		$(q_3, 0, L)$	$(q_3, 1, L)$	$(p, \#, L)$

Then, the above Turing machine processes the input string "1110" as follows:

q_0 1110# $\rightarrow$ #1 q_1 110# $\rightarrow$ #11 q_1 10# $\rightarrow$ #111 q_1 0# $\rightarrow$ #1110 q_1 # $\rightarrow$
 #111 q_2 0# $\rightarrow$ #11 q_3 1## $\rightarrow$ #1 q_3 11## $\rightarrow$ # q_3 111## $\rightarrow$ q_3 #111## $\rightarrow$
 p##111##

1.3.3. Simple morpho-Eraser

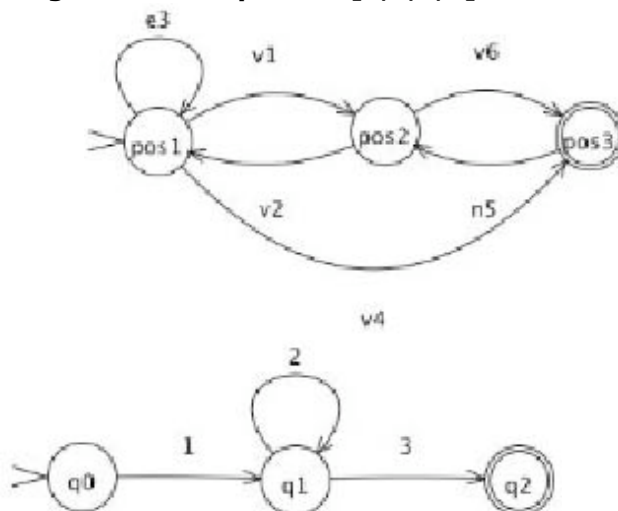
This morpho-Turing machine reads morphograms in the language given by the expression $\Sigma^n(\text{AG}(\text{MG}) + 1)$, and replaces the 'right-most' differentiation by a blank (-).

Because the 'alphabet' of this *Simple morpho-Eraser* depends on the applied morphograms only and not on a pre-given set of signs, say $\{0, 1\}$, its definition works for all kinds of morphograms.

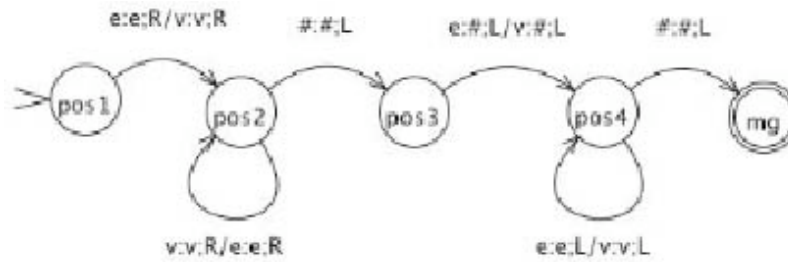
Examples [1,2,2,3], [3,4,4,5], [$\diamond$, O, O, $\blacktriangle$], [μ , ψ , ψ , β], etc.

Diagram for morphoFSM[1,2,2,3]

Diagram for FSM(1,2,2,3)



Again, this diagram depicts the differentiation structure of the morphogram [1,2,2,3] in respect of E- and N-differentiations representing its morphoFSM. The second diagram depicts the FSM for the numeric sequence (1,2,2,3). While the machine diagram for the Simple morpho-Eraser depicts the machine steps of left and right movements in respect to the EN-structure of the morphogram and its task to erase the last differentiation.

Erasing machine diagram**Morphic erasing procedure**

morphoTM = ($\{\text{pos1}, \dots, \text{pos4}\}$, $\{E, N, s(m)\}$, $\{\{AG(MG)+1\}, \#\}$, δ , $\text{pos1}, \{p\}$) where δ is given by the transition table.

Transition table

pos Σ	v_i	e_i	#
pos ₁	(pos ₂ , v, R)	(pos ₂ , e, R)	–
pos ₂	(pos ₂ , v, R)	(pos ₂ , e, R)	(pos ₂ , #, L)
pos ₃	(pos ₄ , #, L)	(pos ₄ , #, L)	–
pos ₄	(pos ₄ , v, L)	(pos ₄ , e, L)	(mg, #, L)

Differentiation Table

pos pos	pos ₁	pos ₂	pos ₃	pos ₄	pos ₅
pos ₁	#[MG]#	$e_1 : e_1, R$ $v_1 : v_1, R$	–	–	–
pos ₂	–	$e_2 : e_2, R$ $v_2 : v_2, R$	# : #, L	–	–
pos ₃	–	–	–	$e_3 : \#, L$ $v_3 : \#, L$	–
pos ₄	–	–	–	$e_4 : e_4, L$ $v_4 : v_4, L$	# : #, L
pos ₅	–	–	–	–	#[mg]#

Elimination procedure for v_6 of morphogram [1,2,2,3]

[1, 2, 2, 3]	2	3	4
□	v_1	e_3	v_4
□	□	v_2	v_5
□	□	□	v_6

 $\Rightarrow$

[1, 2, 2]	2	3
□	v_1	e_3
□	□	v_2

Then, the above morpho-Turing machine processes the input morphogram "[1,2,2,3]" towards an erasing of the 'last' differentiation of the morphogram with the resulting morphogram [1,2,2], and right as $R=\uparrow$ and left as $L=\downarrow$, as follows:

#↑[1,2,2,3]#

	v_1	e_3	v_4
# pos1	□	v_2	v_5
	□	□	v_6

 $\rightarrow$
#[1,↑,2,2,3]#**#[1,2,↑,2,3]#**

$$\# \begin{array}{c|c} e_3 & v_4 \\ \hline v_2 & v_5 \\ \hline \square & v_6 \end{array} \# \quad \rightarrow \quad \# \begin{array}{c|c} v_1 & - \\ \hline \square & v_2 \end{array} \text{pos2} \begin{array}{c|c} e_3 & v_4 \\ \hline - & v_5 \\ \hline - & v_6 \end{array} \# \quad \rightarrow$$

#[1,2,2,↑,3]#

#[1,2,2,3]↑#

$$\# \begin{array}{c|c} v_1 & e_3 \\ \hline \square & v_2 \end{array} \text{pos2} \begin{array}{c|c} - & v_4 \\ \hline - & v_5 \\ \hline - & v_6 \end{array} \# \quad \rightarrow \quad \# \begin{array}{c|c|c} v_1 & e_3 & v_4 \\ \hline \square & v_2 & v_5 \\ \hline \square & \square & v_6 \end{array} \text{pos2} \# \quad \rightarrow$$

#[1,2,2,↓,3]#

$$\# \begin{array}{c|c} v_1 & e_3 \\ \hline \square & v_2 \end{array} \text{pos3} \begin{array}{c|c} - & v_4 \\ \hline - & v_5 \\ \hline - & v_6 \end{array} \# \quad \rightarrow$$

#[1,2,↓,2,-]##

#[1,↓,2,2,-]##

$$\# \begin{array}{c|c} v_1 & - \\ \hline \square & v_2 \end{array} \text{pos4} \begin{array}{c|c} e_3 & v_4 \\ \hline - & v_5 \\ \hline - & v_6 \end{array} \# \quad \rightarrow \quad \# \begin{array}{c|c} v_1 \text{pos4} & \begin{array}{c|c} e_3 & v_4 \\ \hline v_2 & v_5 \\ \hline \square & v_6 \end{array} \end{array} \# \quad \rightarrow$$

↓[1,2,2,-]##

##[1,2,2,-]##

$$\# \text{pos4} \begin{array}{c|c|c} v_1 & e_3 & - \\ \hline \square & v_2 & - \\ \hline \square & \square & - \end{array} \# \quad \rightarrow \quad \text{pos4} \# \# \begin{array}{c|c|c} v_1 & e_3 & - \\ \hline \square & v_2 & - \\ \hline \square & \square & - \end{array} \# \# \quad \rightarrow$$

##[1,2,2]##

$$[\text{mg}] \# \# \begin{array}{c|c} v_1 & e_3 \\ \hline \square & v_2 \end{array} \# \# .$$

1.4. Spekulative Aspekte einer transTM

1.4.1. Weitere Dekonstruktion von Grundbegriffen

Nachdem die vorläufigen ersten Dekonstruktionsschritte sich sklavisch ("sheepish") an die klassische Vorlage der Definition der Turing Maschine gehalten haben, ist es nun angebracht, einen subversiveren Entwurf zu wagen, der sich nicht mehr an einem Maschinenmodell, sondern eher an einem Modell lebender Materie orientiert.

Dabei entfallen die gängigen konstitutiven Dichotomien des Konzepts der Turing Machine. Die Dichotomien von *Maschine* (Apparat) und *Programm*, *Programm* und *Alphabet*, *Lesekopf* und *Band*, *Alphabet* und *Skript*, *Befehl* und *Daten*, sind restlos zu dekonstruieren und aufzulösen. Ebenso Konzepte wie *analog* versus *digital*, usw.

Lebewesen rechnen nicht. Diese These ist in Opposition zur Leibniz'schen Weltauffassung wie sie heute in den Kognitionswissenschaften und der Neurobiologie vorherrscht, und entspricht eher dem Pascal'schen Weltbild.

Dass heute Milliarden Dollars/Euro für die Leibniz'sche Version vom Staat via Steuerzahler bereitgestellt werden, beweist in keinsten Weise seine Richtigkeit.

Auch wenn Gotthard Günther in seinem Moskauer Vortrag der Hegel-Gesellschaft 1970 schreibt, "*Der Weltgeist rechnet*", hat er damit gewiss nicht an eine Turing Maschine gedacht.

Wenn in einem ersten Dekonstruktionsschritt in *negativer*, d.h. abgrenzender Absicht postuliert wurde, dass die neue Maschine weder rechnet, noch liest oder schreibt, usw. ist es nun an der Zeit, positiv, und soweit als möglich, *konstruktiv* anzugeben, was ihre Aktivitäten und Leistungen darstellen könnten.

Spekulativ lässt sich sagen, die Maschine liest nicht, sie *sieht*. Ein 'Wort' wird nicht mehr

schrittweise gelesen, sondern in seiner historisch vermittelten Ganzheit augenblicklich wahrgenommen, d.h. gesehen. Dies ist für digitale Maschinen sicherlich absurd, scheint jedoch für Analogmaschinen weniger fremd zu klingen.

Die Komplexität des Sehapparates steht in direkter Abhängigkeit von der Struktur des Gesehenen, und wandelt sich in seiner Aktivität je nach der Komplexität des zu Sehenden. Damit ist die klassische Dichotomie von Sehendem und Gesehenem, d.h. von Operator und Operand der Seh-Operation dynamisiert und aufgelöst.

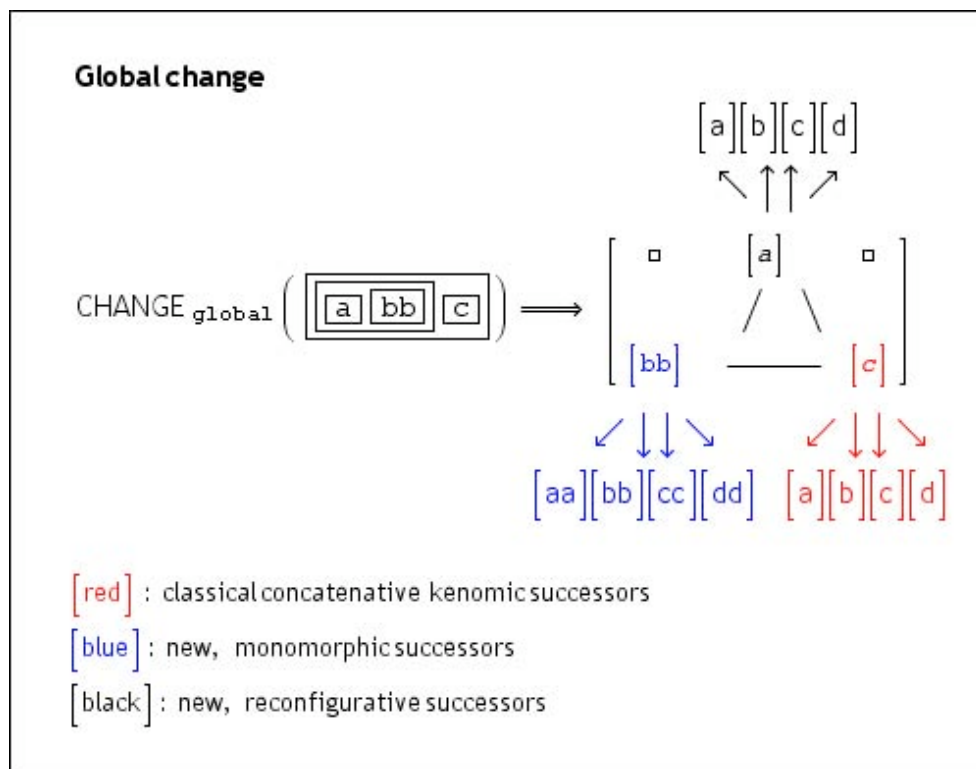
Es ist bestimmt hilfreich, sich etwa auf den früheren Text zu den *Morphosphere(s)*, *Morphosphere(s): Asymmetric Palindromes as Keys*, zu beziehen, um triviale Missverständnisse ausklammern zu helfen.

Zur vorgeschlagenen Maschinenkonzeption sei auf die Arbeit verwiesen:

<http://memristors.memristics.com/MorphoFSM/Finite%20State%20Machines%20and%20Morphogramatics.pdf>

Eine Idee wie sich eine Nachfolgeroperation auf ein ganzheitlich thematisiertes Morphogramm auswirkt, ist in dem foldenden Diagramm angegeben.

Anstelle einer sequentiellen Interpretation mit $\text{Succ}([abbc]) = \{[abbca], [abbc b], [abbcc], [abbc d]\}$, wird im 'holistischen' Modell die Nachfolgeoperation auf jede einzelne Monomorphie des Morphogrammes angewandt.



<http://memristors.memristics.com/Dominos/Domino%20Approach%20to%20Morphogramatics.html>

Die neue Maschine rechnet nicht, sondern transformiert die wahrgenommenen und erkannten Konstellationen von einer Konstellation in eine andere Konstellation gleicher oder verschiedener Kompliziertheit und Komplexität. Sie ist somit keine Rechenmaschine mehr, sondern eine *Umformungsmaschine*.

Es dürfte nicht schwer fallen, zu erkennen, dass eine Rechenmaschine im Sinne einer Turing-Maschine, ein Spezialfall einer Umformungsmaschine, darstellt. Dies führt zu dem Satz: *Eine Turing-Maschine ist durch eine Umformungsmaschine darstellbar*.

Die Modi der Umformung sind die bekannten Ver-Operationen, wie *Verkehrung*, *Verdichtung*, *Verschiebung*, *Verformung* (Kaehr, 1976). Die Wege solcher Umformungen

sind nicht als sukzessive und linearisierte *paths* oder *multi-paths*, sondern als *journeys* zu verstehen, die eher der Strategien von Hamiltonkreisen folgen.

Weiteres zum Verhältnis von Semiotik und Kenogrammatik, siehe auch: R. Matzka, *Klassische Kenogrammatik. Eine semiotische Ortsbestimmung* (2012)
<http://www.rudolf-matzka.de/dharma/kenogrammatik.pdf>

Als leitende Metaphern, anstelle von Informationsverarbeitung und Programmierung, erweisen sich die *Konzepte* von *Wartung* und *Perturbation* (Maturana, Varela) eines Gebildes.

Morphogramme sind seit ihrer Einführung durch Günther (1962) immer als ganzheitliche Pattern bzw. Muster ("hologram", 1965), d.h. als Ganzheiten verstanden worden. Das heisst nicht, dass sie auch immer dementsprechend thematisiert und formalisiert worden wären.

Die EN-Struktur eines Morphogrammes betont in einfacher und kalkulierbarer Weise die ganzheitliche Struktur eines Morphogrammes. Wobei allerdings die Aufmerksamkeit auf die Simultaneität der Differenzen gesetzt, und die Nummerierung der Positionen (Orte) der Differenzen als technisches Hilfsmittel verstanden werden muss.

Die Simultaneität ist auch Ausdruck der retro-grad rekursiven Erzeugung des Morphogrammes.

Die EN-Struktur und ihre Positionen schreiben die generischen Eigenschaften des Morphogrammes ein.

Damit ist auch klar, dass die positionierten EN-Ereignisse *technisch* nicht einfach durch eine mikro-elektronische Realisation wie "on-off" oder durch einen binär-dual gedeuteten Gencode gegeben sind, sondern wegen ihrer reflektionalen, d.h. rückwärts- wie vorwärts gerichteten und positionierten Orientierung durch nano-elektronische Eigenschaften charakterisiert werden müssen, die z.Z. wohl am besten mit *Memristoren* und memristiven System realisiert werden könnten.

Durch die Nummerierung der Differenzierungen wurde diese Ganzheitlichkeit der Morphogramme aus formalisierungs-technischen Gründen teilweise, wenn auch konventionell, wieder linearisiert.

Alle EN-Unterscheidungen, die ganzheitlich das Morphogramm bestimmen, werden in einem 'Blick' von der neuen Maschine 'gesehen'.

Eine natürliche Konsequenz dieses ganzheitlichen Ansatzes, bei dem keine Position eine Auszeichnung erhält, ist es, dass eine solche Maschine weder durch die Kategorien eines Inputs noch eines Outputs als primäre Bestimmung charakterisiert werden kann.

Damit ist auch ein Kriterium angegeben, maschinentheoretisch den Anforderungen einer autonomen und autopoietischen Maschine (Maturana, Varela) zu entsprechen.

Analog heisst die Metapher des Sehens für die Maschine, dass sie im Gegensatz zu einer Turing-Maschine nicht nur einen Lese-Schreibkopf hat, sondern sovieler simultan aktive Lese-Schreibköpfe wie es die EN-Struktur des Morphogrammes jeweils erfordert.

Nach dem Skizzierten ist einsichtig, dass es sich hier nicht um die Situation von Mehrkopf-, Mehrband-Maschinen handelt, die sich wie bekannt in ihrer Berechenbarkeit auf den klassischen Fall reduzieren lassen.

Auch die von Peter Wegner vorgeschlagenen *Interaktionsmaschinen* sind nicht in der Lage, ihre Fixierung auf das klassische Denkmodell aufzulösen.

Eine weitere Metapher könnte sein, dass an jeder Position eines kenogrammatischen Feldes ein Lese-Schreibkopf positioniert ist, der je nach Konstellation der Differenzen für *E* und *N* liest oder schreibt. Es liegt nun nicht fern, anzunehmen, dass alle Köpfe *simultan* ihre Konfiguration von EN-Differenzen lesen, da sie nicht linear, sondern tabulart angeordnet sind, und so zusammen die Gesamtkonstellation wahrnehmen, d.h. 'sehen' können.

Diese Art des Wahrnehmens und Sehens ist offensichtlich nicht mit einer Weise von Perzeption von Daten und deren Komposition zu verwechseln.

Ohne auf eine Phänomenologie des Lesens, Schreibens und Sehens eingehen zu wollen, lässt es sich doch erwähnen, dass das Wahrnehmen als Sehen dem Lesen und Schreiben vorangeht.

Eine Simultaneität des 'Wahrnehmens' ist möglich, da ein Morphogramm nicht sequentiell, sondern differentiell. d.h. auch, tabular definiert ist.

Eine solche Sicht der Dinge schliesst nun eine *konventionelle* Nummerierung aus technischen Gründen durchaus nicht aus. Denn Ganzheitlichkeit und Nummerierung der Teile des Ganzen stehen nicht in einem ausschliesslichen Gegensatz.

Die Figur [aba] z.B. wird simultan von den Positionen pos1, pos2 und pos3 gelesen, und zwar als Folge von E/Ns, d.h. als $[v_1e_2, v_3]$, womit das Morphogramm [aba] als Ganzes erkannt wird.

Die früheren Entwürfe zu einer morphogrammatistischen Turing-Maschine, wie sie oben skizziert wurden, sind daher in Hinsicht einer Metapher des *Sehens* (Wahrnehmens) statt des Lesens und Schreibens neu zu interpretieren.

Re-interpretation: (read1=v1, read2=e2, read3=v3) => see[v1,e2,v3].

Ganzheitliches Sehen ist der strategische Gegensatz zum sukzessiven Lesen und Schreiben. Das sukzessive Lesen und Schreiben von Ganzheiten ist eine sekundäre Möglichkeit der Thematisierung von Ganzheiten.

Das Modell der Sukzession wurde vom *Konnektionismus* erweitert durch das Konzept der massiven Parallelität. Trotz der Vorteile der Parallelität, multipliziert diese auch die Nachteile des Sukzessionsmodells.

Ein Modell, das die Simultaneität der Realisation der verschiedenen Differenzierungen unterstützt, könnte durch die *Tabularität* einer kenomischen, bzw. eine kontextualen Matrix versucht werden.

Ein Argument für die simultane Tabularität der Differenzen ist gegeben durch die Tatsache der verschiedenen Nummerierungsmöglichkeiten, die sich aufgrund der Konventionalität der Nummerierung ergeben.

Diese Möglichkeit ist offensichtlich bei Zeichenreihen(gestalten) wie sie für die Turing-Maschine, äquivalent auch für LISP und jede andere auf Zeichen basierende Programmiersprache, vorausgesetzt wird, definitionsgemäss ausgeschlossen.

Zeichenreihen werden rekursiv sequentiell über einem vorgegebenen Alphabet schrittweise aufgebaut. Der Operator (oder Relator) dieses Aufbaues ist die Konkatenation (Verkettungsoperation), die Operanden sind die Zeichen(reihen) basierend auf dem Alphabet.

Damit ist die Linearität und Einzigkeit der Nummerierung vorgegeben. Auf ein Zeichen folgt schrittweise ein anderes Zeichen. Dies geschieht nach dem Modell eines Strichkalküls oder, spezieller, nach dem Modell der natürlichen Zahlen. (cf. SKIZZE-0.9.5)

Alle Differenzen einer morphogrammatistischen Ganzheit gelten simultan.

Diese Einsicht unterstützt die biologische Metapher eines lebenden Systems.

Lebende Systeme sind konzeptionell schon sehr früh als Ganzheiten erkannt worden. Andererseits ist immer wieder die Erfahrung gemacht worden, dass sich Ganzheiten als solche einer Mathematisierung entziehen.

Historisch sind bei einer Interpretation und Abgrenzung dieses neuen Ansatzes an konzeptionelle Unterschiede von symbolischer Formalisierung, neuromorphen Netzen (Neurocomputing) und anderer Abweichungen, wie DNA-Computing, Quanten- und Fuzzy-Logiken zu erinnern.

Von spezieller Wichtigkeit für eine konzeptionelle Charakterisierung des non-computational Ansatzes ist auf das *Actor Model* von Carl Hewitt hinzuweisen.

Eine Addition ist dort eher als eine *Interaktion* zwischen Aktanten/Actors verstanden als

eine formalistische Berechnung im Sinne rekursiver Funktionen.

Die Aktionen der *Wartung* und *Perturbation* eines lebenden Systems sind interaktiver Natur. Insofern lassen sich die klassischen berechenbaren Funktionen mithilfe von Interaktionen definieren.

<http://what-is-computation.carlhewitt.info/>

<http://www.vordenker.de/blog/?p=643#more-643>

Lebende Systeme rechnen nicht, sie leben. Leben ist jedoch nicht Rechnen.

Oder in Analogie zu Heidegger: *Leben lebt*.

Chiastische Metamorphose der Interaktion

Gewiss erreicht die bisher angedeutete Subversion noch nicht ihre eigenen Grenzen. Die vorgeschlagenen Dekonstruktionen beziehen sich konsequenterweise erst einmal einzig auf die intra-kontextualen Begriffsbildungen und formalen Konstruktionen (von Turing-Maschinen), und lassen ihre eigenen kontextualen Rahmenbedingungen noch ausserhalb jeglicher Dekonstruktion.

Der definitorische Rahmen, d.h. die generelle Definition einer TM, als solcher ist dabei noch nicht tangiert. Die angedeutete "*Wartung*" (maintenance) bezieht sich darauf, diesen Rahmen gegenüber Perturbationen aufrechtzuerhalten. Woher die Perturbationen kommen, und wie weit diese von "ausserhalb" kommen können, und was ein solches "ausserhalb" bedeuten könnte, ist noch weitgehend offen.

Eine Dekonstruktion des Konzepts der "*Zustände*" eröffnet den Zugang zum Verständniss der Entstehung der Zustände als Teil der *History* der Maschine. Zustände einer Turing-Maschine sind im morphogrammatischen Kontext spezielle angehaltene Histories. Histories sind demnach das Resultat der Verarbeitung von Perturbationen im Prozess der Selbst-Wartung der morphogrammatischen Maschine.

Damit wird auch eine Ablösung von Ontologien durch '*Ontogenesen*' (Fabian Kostadinov) für formale und Programmier-Sprachen wie auch für Automaten- und Maschinenkonzeptionen angedeutet. Wie schon des öfteren dargelegt, ein Kenogramm erscheint als die Einschreibung des Prozesses seiner Semiose/Kenose.

Was bei Maturana als Metapher der *strukturellen Kopplung* erscheint, ist hier als chiastische Interaktion zwischen diskontextualen morphogrammatischen Maschinen aufzufassen. Deren Mechanismus wurde in früheren Arbeiten im Rahmen der *Diamondtheorie* skizziert.

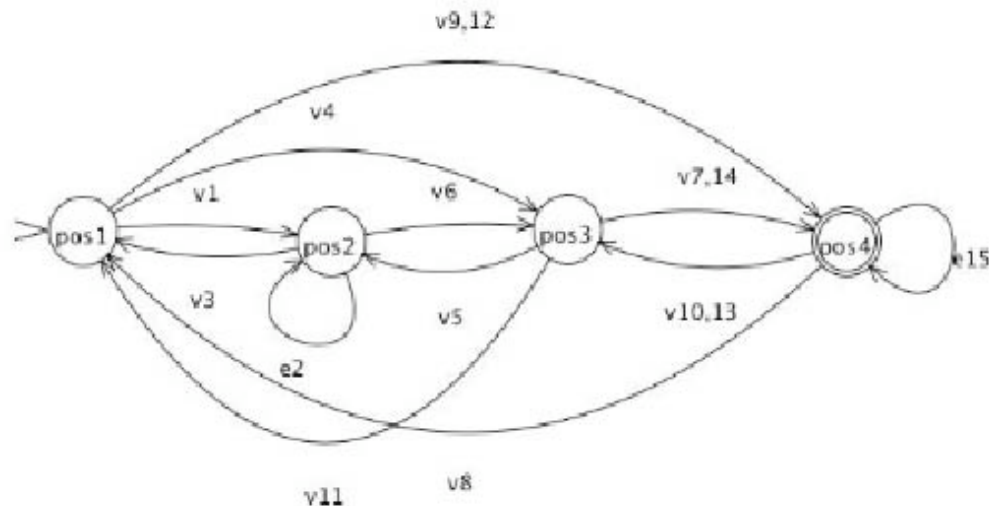
1.4.2. Formale Metapher einer nicht-rechnenden TM

Zweite Dekonstruktionstabelle

Dekonstruktions Strategien	Turing Maschine TM	morphoTM
Tabularisierung	lineares Band	Topologie
poly – Crossbar	Lese – Schreibkopf	Tabularität
Journey	Bewegung, <i>L</i> , <i>R</i>	Simultaneität
Thematisierung	Zustände	History
Trans – Formation	Übergangsfunktion	Umformung
Charakteristik	Definition der TM	Selbst – Wartung

ENstructure automaton table of [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 [abacdd]**1.5. Anmerkungen****1.5.1. Wozu der Aufwand?**

Die Frage, wozu ein solcher Aufwand betrieben werden soll, um an ein Morphogramm ein Kenogramm anzufügen oder wie im Fall des *Simple morpho-Eraser*, eine Differenz zu löschen, stellt sich zu Recht.

Ein Morphogramm repräsentiert in diesem vorläufigen Sinne eines neuen *Datentyps* allerdings eine abzählbar unendliche Menge von strukturgleichen Zeichenketten. Insofern soll auch die Isomorphie des morphogrammatistischen "Zustandsgraphen", *Diagr-morphoTM*, mit dem klassischen TM-Zustandsgraphen, *Diagr-TM*, des Beispiels, nicht irritieren.

Der letztere ist über einem Alphabet definiert, der morphogrammatistische über Differenzen.

Zur Erinnerung und als *Kriterium* zur Bestimmung einer Turing-Maschine:

That's it

"A Turing machine is a *mathematical* construct. It models a machine that can be built in reality.

"The machine has a *tape head* that can *read* and/or *write* a symbol on a tape. It can also *shift* the tape one position to the left and to the right.

"At any *time*, the machine is in some *internal* state.

"The machine's operation is defined by a finite set of **transitions**.

A *transition* is specified by four items: a *state*, a *symbol*, another *state*, and an *action*.

The transition is applicable whenever the machine is in the first state, and the head is on the specified symbol.

"It is applied by moving into the second state and performing the action. The action specifies either a move to the *left*, or a move to the *right*, or a *symbol* (to be written on the tape, overwriting whatever symbol was there)."

"That's it. Anything that behaves according to these specifications is a Turing machine. The transitions completely define the machine's operation." (Lucy-S)

<http://everything2.com/title/Turing+Machine>

Die Frage, definieren die beiden oben genannten Diagramme, *Diagr-TM* und das *Diagr-morphoTM*, dieselbe Maschine, lässt sich nun durch den "*That's it*"-Katalog entscheiden und so eine mögliche Irritation verhindern. *Hence, check it!*

Als eine erste Antwort auf die Frage nach dem Unterschied zwischen einer TM und einer morphoTM, erhalten wir somit: Die *morphoTM* berechnet, wenn sie semiotisch-symbolisch interpretiert wird, nicht eine, sondern beliebig viele gleichgestaltete semiotische Figuren in einem einzigen Durchgang.

Es wird also nicht bloss das vorgegebene symbolische Beispiel (0,1,0,0,1,1) mit "a" ergänzt, sondern es wird die Struktur $[x,y,x,x,y,y]$ mit $x,y \in \Sigma = \{\text{Menge von Notationen aller Art}\}$ mit *new* erweitert.

"Wozu das Ganze" kriegt damit zumindest die Qualität einer enormen *Komplexitätsreduktion*.

Konkreter: Es geht um den Übergang von der Potenzfunktion für Zeichenketten m^n zur Summe der Stirling Zahlen 2. Art für Morphogramme, d.h den Bell-Zahlen: $\sum S(n,k)$.

Die Abstraktion, z.B. von $11^{11} = 285'311'670'611$ nach $B_{11} = 678'570$ liefert eine Komplexitätsreduktion um den Faktor 420'460.1892... (Wolfram|Alpha)

Eine naheliegende Konsequenz einer differenztheoretischen Definition einer morphoTM ist ihre Alphabet-Unabhängigkeit. Diese hat zur Folge, dass eine Situation wie die Aktion einer *Nicht-Anerkennung* eines Wortes durch die Maschine wegen eines nicht zu ihrer Definition passenden Zeichensatzes nicht existiert.

Die Alphabet-Abhängigkeit einer Turing-Maschine (wie jeder formalen Sprache auch) scheint trivial zu sein. Für eine Theorie der Berechenbarkeit gibt es jedoch keine Trivialitäten, die unreflektiert akzeptiert werden müssen. Jeder Bestimmungspunkt eines Kalküls muss expliziert, und was nicht expliziert werden kann, wie die Identität der vorausgesetzten Zeichen, muss postuliert werden.

So ist es nicht redundant, wenn *Hans Hermes* bei der Charakterisierung der Elementarmaschinen für rechts-, links- und stopp-Aktionen schreibt:

"Vorausgesetzt wird ein festes Alphabet $\{a_1, \dots, a_N\}$. Die betrachteten Maschinen hängen von diesem Alphabet ab." (Hermes, S. 41)

Wichtig ist auch die Charakterisierung des Alphabets als ein "festes". D.h., das Alphabet verändert sich nicht im Verlauf einer Berechnung (Goguen, Gurewicz, Kaehr).

Die Feststellung, dass Morphogramme kein Alphabet voraussetzen, heisst in einer komplementären Formulierung, dass Morphogramme ihre je eigenen Alphabete im Prozess der Operationen selbst definieren, und sich daher über einem lokal-dynamischen Alphabet einschreiben.

Eine solche Bestimmung eines Alphabetes widerspricht jedoch vollständig der Definition eines Alphabetes im Sinne formaler Sprachen, Logik und Turing-Maschinen.

Siehe auch: R. Kaehr, Anfangen, in: *Marc Jongen (Hrsg.) Was wird Denken heißen? Kognition und Psyche im posthumanen Zeitalter*, Wilhelm Fink Verlag, 2013

Jede differenzierbare Struktur wird von einer morphoTM akzeptiert.

Morphogrammatische Maschinen, die auf diesem Prinzip der morphogrammatischen *Differenzierbarkeit* beruhen, sind im Sprachrahmen der "konkatenativen" Kenogrammatik formalisiert. Die Modi der *Überlagerung* wie *Verschmelzung* und *Verknüpfung* (im Gegensatz zu *Verkettung*=Konkatenation), die bei einer weiteren Dekonstruktion anzugehen sind, werden hier noch nicht in Betracht gezogen.

Damit wird vorerst auch jegliche Form von *Metamorphose* als Modus einer neuen "Berechenbarkeit" im Sinne lebender Systeme ausgeklammert.

Von einem *epistemologischen* Standpunkt muss auch gesehen werden, dass die Definitionen einer TM zwar sehr einfach aussehen, ihre Einfachheit jedoch der etablierten Intuition eines Mathematikers bedarf, und als solche selbst, weder begrifflich noch mathematisch, in der Definition der TM expliziert und reflektiert sind.

1.5.2. Iterative und akkretive Mehrdeutigkeit

Das doppelte Resultat der Berechnung mag etwas überraschen, da es von der vor-gegebenen Aufgabestellung, zumindest in einem Teilresultat, $[1,2,3,2,2,3,3,4]$, abweicht.

Vom Standpunkt der Morphogrammatik ist dieses abweichende Resultat zwingend. Denn die Operation *new* hat keinen vor-gegebenen Bereich, sondern hat ihren Wirkungsbereich situativ aus dem manipulierten Morphogramm zu errechnen. Die Errechnung erfolgt mit der Aggregations-Funktion *AG*.

Die Anwendung der TM in eine Richtung, hier *R*, liefert ein Resultat, das das Wort, Morphogramm, gemäss Jobdefinition verändert. Auf dem Weg (zurück) in Richtung *L*, muss dieses Resultat jedoch in Rechnung gestellt werden. Damit ändern sich die kontextuellen Angaben für die Übergangsfunktion (Umformungsfunktion).

Der klassische Fall ist neutral zu einer solchen kontextbezogenen Veränderung. Was auch immer das Resultat des ersten Durchganges ist, definiert dieser nicht die möglichen Resultate des zweiten Durchganges. Die Aufgabe ist, unabhängig vom ersten Resultat, das Element "a" and das vorgegebene Wort anzufügen.

Trivialerweise lässt sich diese kontextuelle Mehrdeutigkeit per *Dekret* verhindern. Womit nur das erste oder zweite Resultat des zweiten Durchganges, d.h. die iterative oder die akkretive Übergangsfunktion, akzeptiert wird, und das Resultat $[1,2,3,2,2,3,3,1]$ oder $[1,2,3,2,2,3,3,4]$ liefert.

Vorausgesetzt, man weiss das Resultat der ersten Berechnung im Voraus, lässt sich adhoc die Aufgabe für den Weg zurück als iterative vorgeben.

Es liesse sich auch monieren, dass das neu angehängte Element "a", hier als die "1" erscheint. Diese ist nun gewiss kein neues Element bezüglich des Ausgangswortes, bzw. des Anfang-Morphogrammes.

Dieses Argument ist gewiss richtig, wenn wir im Rahmen der klassischen Turingdefinitionen, mit ihren Identitätsvoraussetzungen operieren, und nicht im Rahmen der Morphogrammatik, wo die Identität eines einzelnen Zeichens bzw. eines Kenogrammes prinzipiell keine Rolle spielt.

D.h., das angefügte *new* als Kenogramm kann sehr wohl die Gestalt der vorhergehenden "1" annehmen.

Worauf es in der Morphogrammatik einzig und allein ankommt, sind die *Differenzen* und deren *Verortung* (Position), kurz: (e, v, pos), und nicht die atomaren Vorkommnisse von Zeichen der Semiotik eines Zeichenvokabulars einer Turing-Maschine.

Insofern ist auch die Sprechweise der Hinzufügung eines Kenogrammes irreführend.

Der Term *new* entspricht dem Zeichen "a" des TM-Beispiels, entstammt jedoch nicht aus einem vorgegebenen Alphabet, sondern wird in Abhängigkeit der Struktur des vorgefundenen Wortes, d.h. Morphogrammes, auf dem TM-Band berechnet.

AG ist der Aggregationsgrad eines Morphogrammes, also die Anzahl der verschiedenen Kenogramme des Morphogrammes, wobei für die *Akkretion* eine Einheit dazu addiert wird.

$$AG([1,0,1,1,0]) = \{0,1\}$$

$$AG_{accr}(MG) = AG(MG) + 1.$$

1.5.3. Weiterer Vergleich von FSA und morphoFSM

Die Abarbeitung eines Wortes geschieht beim Finite State Automaton, FSA, direkt linear sukzessiv nach Vorgabe der Struktur des Wortes. Abgearbeitet werden sukzessive die einzelnen linear geordneten *Elemente* eines Wortes.

Die Abarbeitung eines Morphogrammes geschieht bei dem morphogrammatischen Finite State Automaton, morphoFSA, nach Massgabe der Nummerierung der

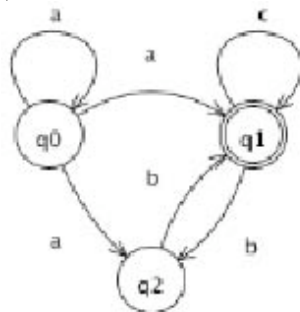
Subsysteme des als Anfangszustand gegebenen Morphogrammes. Diese Nummerierung ist konventionell, und kann auch anders definiert werden. Abgearbeitet werden durch die morphoFSA sukzessive die einzelnen *E/N-Differenzen* des Morphogrammes.

Der Datentyp der FSA sind lineare Zeichenreihen, gebildet über einem festen Alphabet. Damit ist dieser Datentyp definiert durch die Zeichen seines Alphabets und der Position der Zeichen in der linearen Zeichenreihe, also $(\text{Alph}, Z_i, i \in \text{length}(\text{ZR}))$.

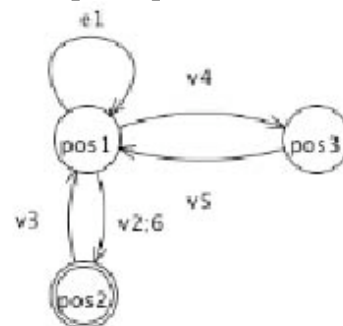
Der Pseudo-Datentyp der morphoFSA repräsentiert Strukturen von verteilten und verorteten E/N-Differenzen in einem System von Positionen, $(\text{pos}, \{e, v\}_i, i \in s(m))$. Kenogramme als Teile von Morphogrammen sind somit generiert durch die Interaktionen der Differenzen an den durch die Interaktionen erschlossenen Orten (Positionen).

So wird die semiotische Zeichenreihe (aabc) durch $\text{FSA}(\text{aabc})$ erkannt. Entsprechend wird das Morphogramm [aabc] von dem morphogrammatischen FSA, $\text{morphoFSA}[\text{aabc}]$, 'erkannt'. Die Erkennung des Morphogrammes [aabc] durch den Automaten $\text{morphoFSA}[\text{aabc}]$ unterscheidet sich offensichtlich von dem semiotischen Automaten FSA. Der Automat $\text{FSA}(\text{aabc})$ für (aabc) akzeptiert die Zeichenreihe (aabc) als vorgegebene, während der morphogrammatische Automat $\text{morphoFSA}[\text{aabc}]$ das Morphogramm [aabc] durch den Prozess des differentiellen Erkennens eher interaktiv erzeugt denn als vorgegebene Reihe wieder-erkennt. M.a.W., das Morphogramm [aabc] steht für jede mögliche struktur-gleiche Zeichenkette.

FSA(aabc)



morphoFSA[aabc]



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

1.5.4. Palindrome da und dort

Ein weiterer interessanter Vergleich zwischen TMs und morphoTMs lässt sich mit Hilfe von *Palindromen* herstellen.

Palindrome sind in diesem Falle Wörter, deren Leseresultat unabhängig von der Leserichtung ist. Beispiele sind etwa “Anna” und “elle”. “Anna” wie “elle” sind sich selbst vorwärts wie rückwärts gelesen identisch. Dabei wird von der Gross- und Kleinschreibung abstrahiert. Ein einzelnes Zeichen ist in sich selbst palindromisch, also etwa “b”.

Derselbe Gleichheitsspruch für Palindrome gilt auch für Morphogramme. Sie sind palindromisch, wenn ihre Lesungen richtungsneutral sind. Dabei wird eine durch Konvention bestimmte Nummerierung der Differenzen miteinbezogen. Und vorausgesetzt, dass diese Nummerierung für beide Leserichtungen gilt. Als Leserichtung gilt hier die Vorwärts- und Rückwärts-Gerichtetheit.

Witzig ist nun, dass ein Wort wie “Annabelle”, das zusammengesetzt ist aus den drei Palindromen, “Anna”, “b” und “elle”, gewiss kein klassisches Palindrom darstellt, morphogrammatisch jedoch leserichtungs-unabhängig ist, und daher nach Definition, zur Klasse der Palindrome gehört.

Linguistic interpretations of asymmetric palindromes

Why looking for such sophisticated palindromes like “Oh Cello voll Echo” (Pfeiffer) if we have such an intriguing example like the asymmetric palindrome “Annabelle”?

Annabelle gets a palindromic interpretation by the *asymmetric* morphogram [1,2,2,1,3,4,5,5,4]:

“anna” : num(anna) = [1,7,7,1]

” 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
```

ENstructure of “Annabelle”: [1,2,2,1,3,4,5,5,4]

Again, morphograms are not defined over an alphabet but by differentiations. The ENstructure is calculating the differentiations, N, E, of a morphogram at the subsystem place (i, j).

- ENstructure [1,2,2,1,3,4,5,5,4] = ENstructure [4,5,5,4,3,1,2,2,1]:

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

ENstructure [1,2,2,1,3,4,5,5,4] $\cup$ subsystems 9 $\cup$ positions{1,2,3,4,5}
 (int * int * EN) list list $\cup$ (int * int list) list $\cup$ int

Mapping

Diagr: (ENstructure, subsystems, positions) --> DiagrMorphoFSM

ENtable: ENtable --> subsystems

```
[[],
 [1,(1,2,N)],
 [3,(1,3,N), 2,(2,3,E)],
 [6,(1,4,E), 5,(2,4,N), 4,(3,4,N)], : pos{1,2}
 [10,(1,5,N),9,(2,5,N),8,(3,5,N),7,(4,5,N)], : pos{1,2,3}
 [15,(1,6,N),14,(2,6,N),13,(3,6,N),12,(4,6,N),11,(5,6,N)], : pos{1,2,3,4}
 [21,(1,7,N),20,(2,7,N),19,(3,7,N),18,(4,7,N),17,(5,7,N),16,(6,7,N)], : pos{1,2,3,4,5}
 [28,(1,8,N),27,(2,8,N), 26,(3,8,N),25,(4,8,N),24,(5,8,N),23,(6,8,N),22,(7,8,E)], : {1,2,3,4,5}
 [38,(1,9,N),37(2,9,N),36(3,9,N),35,(4,9,N),34,(5,9,N),31,(6,9,E),30,(7,9,N),29,(8,9,N)]] :
pos{1,2,3,4,5}
: (int * int * int list * EN) list list list
```

DiagrTable: (EN, subsystems) --> (pos,pos).

MorphoFSA	pos ₁	pos ₂	pos ₃	pos ₄	pos ₅
[1, 2, 2, 1, 3, 4, 5, 5, 4]					
pos ₁	e6	v1, 4	v7	□	□
pos ₂	v3,5,19,24	e2	□	□	□
pos ₃	v9	v8	□	(to do)	□
pos ₄	□	□	□	□	□
pos ₅	□	□	□	□	□

That is, the trito-normal form *tnf* of the numeric interpretations of the words “anna” and “elle”, num(*anna*) = [1,7,7,1] and num(*elle*) = [4,5,5,4] are equivalent: tnf[1,7,7,1] = [1,2,2,1] = tnf[4,5,5,4]. But *localized* in the context of the whole morphogram, hence contextualized, they are different.

1.5.5. How to decide palindromicity with TMs and morphoTMs?

General procedure for TM

"We remember the first letter, delete, move to the last letter, and either delete it if it matches or return *false* if it doesn't.

We repeat this process on the undeleted part of the string and return *true* iff the string becomes empty by deleting its *last* letter."

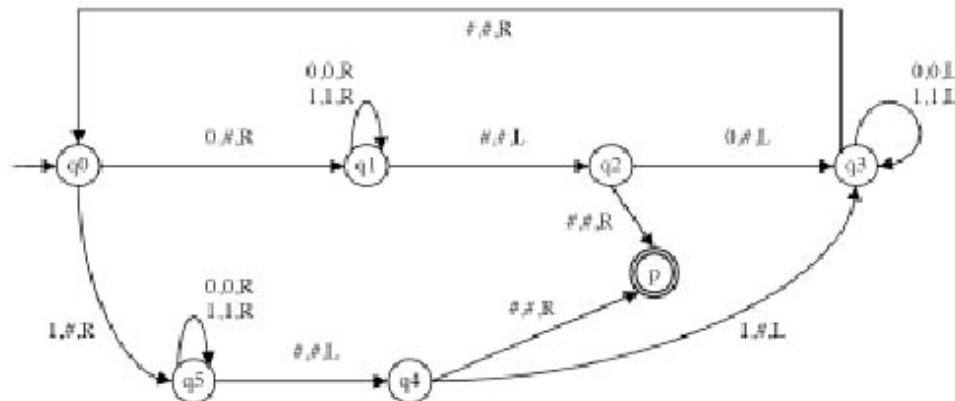
Example(10101)

10101 --> x010x --> xx1xx --> xxxx.

Procedure

With $10101 = 1_1 0_2 1_3 0_4 1_5$: $1_1 = 1_5 \rightarrow 0_2 = 0_4 \rightarrow 1_3 = 1_3$.

Explicit TM processing:



<http://www.cs.nott.ac.uk/~nxk/TEACHING/G53COM/G53COMLecture6.pdf>

More at:

Problem Solving, Ding-Zhu Du, Ker-Iko, 2001, p.166/67

General procedure for morphoTM

Paraphrase

We remember (mark it) the first difference (of the original ENstructure), ‘delete’ (mark it), move to the last difference (of the second inverse ENstructure), and either ‘delete’ it (mark it) if it matches or return *false* if it doesn’t.

We repeat this process on the ‘undeleted’ (unmarked) part of the EN-Structures and return *true* iff the morphogram becomes ‘empty’ by virtually ‘deleting’ its *last* difference.

This approach is a paraphrase of the classical TM approach, and is not yet taking the intrinsic ‘memorization’ property of the retro-grade recursive structure of morphograms and therefore, morphic palindromes, into account.

The idea is that the build-up of the morphic palindrome entails the information of its ‘history’ and a test of its palindromicity can thus be based on it, and has not specially be saved in an ‘external’ memory.

In other words:

The procedures for the decision of the palindromicity of *asymmetric* palindromes are defined by the involvement of a *duplication*, i.e. the original morphogram and the reversed morphogram have to be considered. The comparison proceeds then between the morphogram (palindrome) and its reversion.

This duplication is *technically* reasonable to support the procedure of reading and canceling the differentiations in the forward and backward orientation of palindromes. It makes more explicit what happens without duplication.

The comparison between the first and the last, etc. differentiation involves both morphograms. Therefore the first differentiation of the original morphogram and the last differentiation of the inverted morphogram are compared.

Asymmetric palindromes might be easier modeled and computed by a 2- or 3-tape

TM. One tape for the forward, one tape for the backward reading, and one tape more for the comparison of the results of both tapes.

For the case of *symmetric* palindromes, this distinction collapses with the original morphogram, where the first differentiation and the last differentiation, etc., of the same morphogram, are compared. This corresponds to the classical symmetric case, and is directly modeled and computed by a 1-tape TM.

Palindromes and complexity

"What we do gain by using more complicated machines is speed and efficiency.

For instance, Barzdin (1965) has proved that no machine with one head can tell whether or not a word is symmetric, that is a palindrome, in less than a time that increases proportionally with the square of the length of the word; whereas it is easy to see if it has two heads, we may move the heads to the opposite end of the tape and have them move in to the middle, reporting back to the control, so that when they meet, in a time that only goes up with the length of the tape, they will have been able to tell whether or not the string is a palindrome." (Michael A. Arbib, Brains, Machines and Mathematics, 1987, p.132)

Example: asymmetric even palindrome [1,2,2,3]

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

val it = true : bool

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

val it = [[],[(1,2,N)],[(1,3,N),(2,3,E)],[(1,4,N),(2,4,N),(3,4,N)]]

ENstructure[3,2,2,1];

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

Hence, palindromicity of *asymmetric* palindromes involves the equality of the palindrome and its reversion:

ENstructure[1,2,2,3] = ENstructure[3,2,2,1].

This is trivial, only for *symmetric* palindromes.

Palindromicity for [1,2,2,3]

forwards: backwards:

$(N,N,E,N,N,N)_1 = (N,N,E,N,N,N)_2$

Compare the EN-structure: n_1-n_6 , n_2-n_5 , e_3-e_4 , n_4-n_3 , n_5-n_2 , n_6-n_1

of the Subsystems values: 1,2-3,2, 1,2-3,2, 2,2-2,2, 1,3-3,1, 2,3-2,1, 2,3-2,1.

Result

$N_1 = N_6 \rightarrow N_2 = N_5 \rightarrow E_3 = E_4 \rightarrow N_4 = N_3 \rightarrow N_5 = N_2 \rightarrow N_6 = N_1$.

Hence, [1,2,2,3] is a palindrome.

Elimination procedure

The elimination of the 'last' kenogram of a morphogram is legitim, and is not in conflict with its contextual definition. In fact, the legitimation takes not a kenogram but the 'last' differentiation that is defined by this kenogram.

Elimination procedure from v_6 to v_1 :

[1, 2, 2, 3]	2	3	4
[]	v_1	e_3	v_4
□	□	v_2	v_5
□	□	□	v_6

 $\Rightarrow$

[1, 2, 2]	2	3
[]	v_1	e_3
□	□	v_2

 $\Rightarrow$

[1, 2]	2
[]	v_1

Elimination procedure from v_1 to v_6 :

[1, 2, 2, 3]	2	3	4
□	v ₁	e ₃	v ₄
□	□	v ₂	v ₅
□	□	□	v ₆

 $\Rightarrow$

[2, 2, 3]	2	3
□	v ₅	e ₄
□	□	v ₆

 $\Rightarrow$

[2, 3]	2
□	v ₆

Both elination procedure together:

$N_1=N_6$, thus cancel N_6 ,

$N_2=N_5$, thus cancel N_5 ,

$E_3=E_4$, thus cancel E_4 ,

$N_4=N_3$, thus cancel N_3 ,

$N_5=N_2$, thus cancel N_2 ,

$N_6=N_1$, thus cancel N_1 .

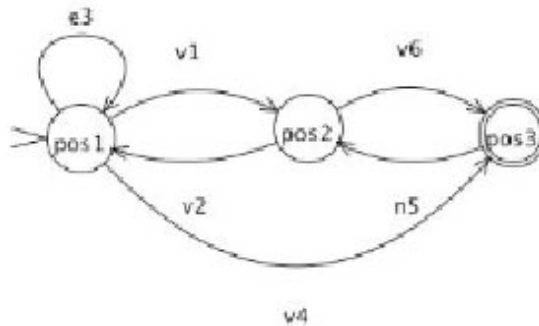
Hence, [1,2,2,3] is a palindrome.

Comparison

Compare the first differentiation with last differentiation,
second-first differentiation with second-last differentiation,
etc.

of the ENstructures of the morphogram and the ENstructure of its inversion,
if equal, then mark it as true (ispalindrome), else as false.

Diagram for morphoFSM[1,2,2,3]



Transformation of the TM example (0110):

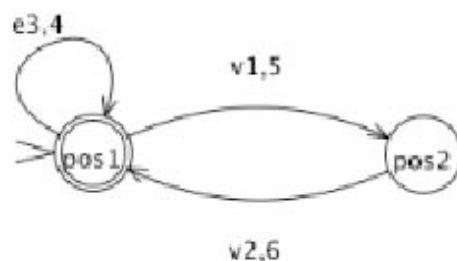
- tnf[0,1,1,0];

val it = [1,2,2,1] : int list

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

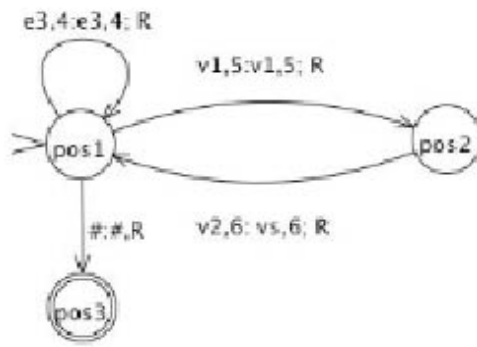
val it = [[], [(1,2,N)], [(1,3,N),(2,3,E)], [(1,4,E),(2,4,N),(3,4,N)]]
: (int * int * EN) list list

Diagram of MorphoFSM[1,2,2,1]



MorphoFSA[1, 2, 2, 1]	pos ₁	pos ₂
pos ₁	e _{3,4}	v _{1,5}
pos ₂	v _{2,6}	—

Diagram of morphoTM[1,2,2,1]



Short version of transition rules for morphoTM[1,2,2,1]

```
q0; #: #; R, q1
q1; v : v; R, q2
q2; v : v; R, q1
q1; e : e; R, q1
q1; #: #; R, q3
(q ≡ pos)
```

Explicit transition rules for morphoTM[1,2,2,1]

```
q0; #: #; R, q1
q1; v1 : v1; R, q2
q2; v2 : v2; R, q1
q1; e3 : e3; R, q1
q1; e4 : e4; R, q1
q1; v5 : v5; R, q2
q2; v6 : v6; R, q1
q1; #: #; R, q3
```

Morphogram[1,2,1,2,1] (symmetric, odd)

```
- tnf[1,0,1,0,1];
val it = [1,2,1,2,1] : int list
- ENstructure[1,2,1,2,1];
val it =
  [[], [(1,2,N)], [(1,3,E), (2,3,N)], [(1,4,N), (2,4,E), (3,4,N)],
  [(1,5,E), (2,5,N), (3,5,E), (4,5,N)]] : (int * int * EN) list list
```

Compare the EN-structures: [1,2,1,2,1]

```
forwards:                backwards:
(N,E,N,N,E,N,E,N,E,N)1 = (N,E,N,N,E,N,E,N,E,N)2
  n1-n10, e2-e9, n3-n8, n4-n7, e5-e6, n6-n5, e7-e4, n8-n3, e9-e2, n10-n1
  1,2-1,2, 1,1-1,1, 2,1-2,1, 1,2-3,1, 2,2-2,1, 1,2-2,1 1,1-2,2, 2,1-2,1, 1,1-1,1, 2,1-2,1
'values' of the Subsystems.
```

Result

$N_1 = N_{10} \rightarrow E_2 = E_9 \rightarrow N_3 = N_8 \rightarrow N_4 = N_7 \rightarrow E_5 = E_6 \rightarrow$

$N_6 = N_5 \rightarrow E_7 = E_4 \rightarrow N_8 = N_3 \rightarrow E_9 = E_2 \rightarrow N_{10} = N_1.$

Hence, [1,2,1,2,1] is a palindrome.

Morphogram [1,2,1], symmetric, odd

```
- ENstructure[1,2,1];
val it = [[], [(1,2,N)], [(1,3,E), (2,3,N)]] : (int * int * EN) list list
```

(N,E,N):
 $n_1-n_3, \quad e_2-e_2, \quad n_3-n_1$
 1,2-1,2, 1,1-1,1, 2,1-2,1.

Result

$N_1 = N_3 \rightarrow E_2 = E_2 \rightarrow N_3 = N_1$.
 Hence, [1,2,1] is a palindrome.

Symmetric Palindrome

The procedures for the decision of the palindromicity of *symmetric* palindromes are defined in strict analogy to the procedures for the classical symbolic palindromes.

Example: symmetric palindrome [1,2,2,1]

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

 - ENstructure[1,2,2,1];
 val it = [[],[(1,2,N)],[(1,3,N),(2,3,E)],[(1,4,E),(2,4,N),(3,4,N)]]

 (N,N,E,E,N,N):
 $n_1-n_6 \rightarrow n_2-n_5 \rightarrow e_3-e_4 \rightarrow e_4-e_3 \rightarrow n_5-n_2 \rightarrow n_6-n_1$.
 Hence, [1,2,2,1] is a palindrome.

Counter example

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

 - ENstructure [1,2,3,3];
 val it = [[],[(1,2,N)],[(1,3,N),(2,3,N)],[(1,4,N),(2,4,N),(3,4,E)]]

 - ENstructure[3,3,2,1];
 val it = [[],[(1,2,E)],[(1,3,N),(2,3,N)],[(1,4,N),(2,4,N),(3,4,N)]]

Comparison: (N,N,N,N,N,E) != (E,N,N,N,N,N).

The relation is a *reversion* but not a palindrome.

1.5.6. Gödel's Palindroms

Gödel's Games within Palindromes

Morphic Palindromicity as a Measure for Self-Reference

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

Kodierung von Turingmaschinen

"Werden Turingmaschinen zu Codewörtern kodiert, so kann ein solches Codewort als Eingabe für eine Turingmaschine benutzt werden, wodurch es möglich wird, mit dem allgemeinen Kalkül der Turingmaschine eine Aussage zu treffen über eine konkrete Turingmaschine, denn jede beliebig abstrakte Turingmaschine kann verwendet werden, um auf das Codewort einer konkreten Turingmaschine angesetzt zu werden." (V. Wolowski)

http://vincent-wolowski.net/content/computer_science/entscheidbarkeitstheorie.pdf

Selbst-Anwendung

Selbst-Anwendung als die Anwendung der Anwendung (H. von Foerster). D.h., die Anwendung der Anwendung auf sich selbst.

Oder als Formel: Die Anwendung, angewandt auf die Anwendung.

Damit ist der ganze Zirkus der Problematik der Iterabilität, Wiederholbarkeit und Zitierbarkeit, jedoch auch der Palindromie und der Zirkularität, aufgefahren.

"Um zu zeigen, daß eine Sprache nicht entscheidbar ist, d.h. daß es eine „Nicht-Terminierung“ gibt, wird eine Turingmaschine auf sich selbst angewandt, man benutzt also eine reflexiv operierende Turingmaschine." (ibd., p. 4)

Für die Selbst-Anwendung $X \text{ rX}$, gilt gewiss, dass X und das zitierte X semiotisch identisch sind. Also: $X =_{\text{semiot}} \text{rX}$.

X: *Codewort als Eingabe für eine Turingmaschine*

$X \text{ rX}$: *jede beliebig abstrakte Turingmaschine kann verwendet werden, um auf das Codewort einer konkreten Turingmaschine angesetzt zu werden.*

Die Logik, die angewandt wird, benutzt die *Spezifikation*, und erlaubt zu erklären: Was für alle gilt, gilt auch für ein Spezielles. Auch wenn dieses Spezielle ein Allgemeines ist. Dass sich damit Widersprüche erzeugen lassen, sollte selbstverständlich sein.

Und die Verplüffung, die dabei allerortens erfahren wird, sollte nicht vergessen, dass der Trick, den Bertrand Russell, um Georg Cantor zu irritieren, letztendlich nichts weiteres war als ein raffinierter Witz, den andere auch schon kannten, sich jedoch nicht entblöden wollten, daraus Publizität zu machen.

The episode was devastating to Frege and Cantor. The story catapulted Russell to the top of intellectual community, and everyone had great fun formulating new versions of the liar's paradox.

<http://descmath.com/diag/russell.html>

Modi der Wiederholbarkeit

Die Differenzen in der Wiederholung lassen sich durch den palindromischen Ansatz der Morphogrammatik weiter spezifizieren. Gotthard Günther hat morphogrammatische Wiederholung vorwiegend durch die Unterscheidung von *Iteration* und *Akkretion* charakterisiert.

Auf der Ebene der der Morphogrammatik lassen sich jedoch weitere Modi der Wiederholbarkeit unterscheiden, speziell wenn diese mit dem Verhalten morphogrammatischer Palindrome in Verbindung gebracht werden.

Wiederholung als *Repetition*, d.h. Wiederholung desselben (Iteration),

Wiederholung als Umkehrung, d.h. als *Reversion* und

Wiederholung als Umformung, d.h. als *Akkretion*.

Die Modi der Iteration und Reversion lassen sich kombinieren mit dem Modus der Akkretion. Es gilt also auch die akkretive Iteration und die akkretive Reversion.

Morphogrammatische Repetition, kombiniert mit Akkretion, ist somit nicht beschränkt auf den Modus der *persevativen* Repetition ("persevative repetitions", Christian Schrei, 2005).

<http://www.nook.at/minimal/en/repetition/>

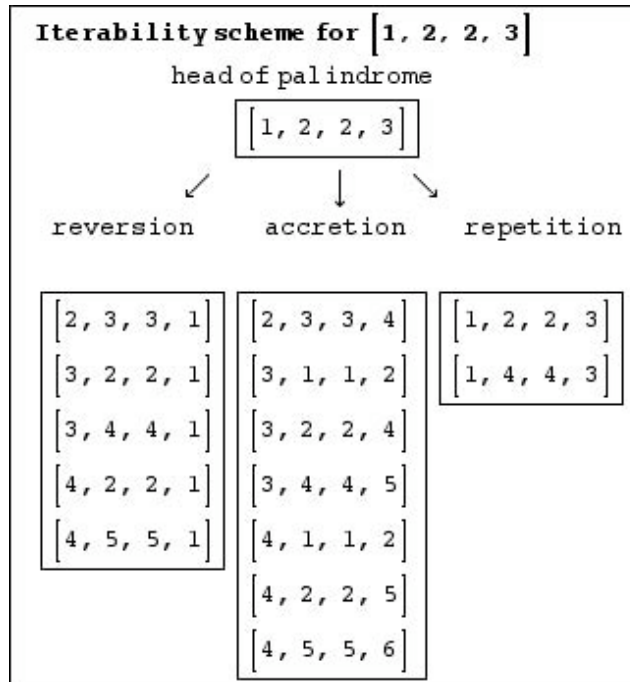
Als Beispiel sei die Palindromizität des Morphogrammes $MG = [1,2,2,3]$ betrachtet.

Modi for head=[1,2,2,3]

repetition: [1,2,2,3,1,2,2,3],

reversion: [1,2,2,3,3,2,2,1],

accretion: [1,2,2,3,2,3,1], [1,2,2,3,3,1,1,2],
 [1,2,2,3,1,4,4,3], [1,2,2,3,3,4,4,1], [1,2,2,3,4,1,1,2], [1,2,2,3,4,2,2,1], [1,2,2,3,2,3,3,4],
 [1,2,2,3,3,2,2,4]
 [1,2,2,3,4,5,5,1], [1,2,2,3,4,2,2,5], [1,2,2,3,3,4,4,5],
 [1,2,2,3,4,5,5,6],



Tcontexture 8

- Tcard 8;

val it = 4140 : int

Palindromes (8)

val it =

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

```
[1,1,2,3,3,4,1,1],[1,2,1,3,4,1,2,1],[1,2,3,1,1,4,2,1],[1,2,1,3,3,1,4,1],
[1,2,3,1,1,3,4,1],[1,2,2,3,4,2,2,1],[1,2,3,2,2,4,2,1],[1,2,3,3,3,3,4,1],
[1,2,2,3,3,2,2,4],[1,2,3,2,2,3,2,4],[1,2,3,3,3,3,2,4],[1,1,2,3,4,2,5,5],
[1,1,2,3,3,4,5,5],[1,2,1,3,4,5,2,5],[1,2,2,3,4,5,5,1],[1,2,3,4,5,1,2,3],
[1,2,3,4,5,2,3,1],[1,2,3,4,5,3,1,2],[1,2,3,4,5,3,2,1],[1,2,3,1,4,5,2,4],
[1,2,3,2,4,5,4,1],[1,2,3,4,1,5,2,4],[1,2,3,4,2,5,4,1],[1,2,3,4,4,5,1,2],
[1,2,3,4,4,5,2,1],[1,2,1,3,3,4,5,4],[1,2,3,1,4,3,5,4],[1,2,3,3,4,4,5,1],
[1,2,3,4,1,3,5,4],[1,2,3,4,3,4,5,1],[1,2,3,4,4,1,5,3],[1,2,3,4,4,3,5,1],
[1,2,2,3,3,4,4,5],[1,2,3,2,4,3,4,5],[1,2,3,3,4,4,2,5],[1,2,3,4,2,3,4,5],
[1,2,3,4,3,4,2,5],[1,2,3,4,4,2,3,5],[1,2,3,4,4,3,2,5],[1,1,2,3,4,5,1,1],
[1,2,1,3,4,1,5,1],[1,2,3,1,1,4,5,1],[1,2,2,3,4,2,2,5],[1,2,3,2,2,4,2,5],
[1,2,3,3,3,3,4,5],[1,1,2,3,4,5,6,6],[1,2,3,4,5,6,1,2],[1,2,3,4,5,6,2,1],
[1,2,1,3,4,5,6,5],[1,2,3,4,5,1,6,3],[1,2,3,4,5,3,6,1],[1,2,3,1,4,5,6,4],
[1,2,3,4,1,5,6,4],[1,2,3,4,4,5,6,1],[1,2,2,3,4,5,5,6],[1,2,3,4,5,2,3,6],
[1,2,3,4,5,3,2,6],[1,2,3,2,4,5,4,6],[1,2,3,4,2,5,4,6],[1,2,3,4,4,5,2,6],
[1,2,3,3,4,4,5,6],[1,2,3,4,3,4,5,6],[1,2,3,4,4,3,5,6],[1,2,3,4,5,6,7,1],
[1,2,3,4,5,6,2,7],[1,2,3,4,5,3,6,7],[1,2,3,4,4,5,6,7],[1,2,3,4,5,6,7,8]]
```

Symmetric Palindromes: 15

```
val it =
```

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

dnfispalindrome (8) : 16

```
val it =
```

```
[[1,1,1,1,1,1,1,1],[1,1,1,1,2,2,2,2],[1,1,1,2,2,3,3,3],[1,1,2,2,2,2,3,3],
[1,2,2,2,2,2,2,3],[1,1,2,2,3,3,4,4],[1,1,1,2,3,4,4,4],[1,2,2,2,3,3,3,4],
[1,1,2,3,3,4,5,5],[1,2,2,3,3,4,4,5],[1,2,3,3,3,3,4,5],[1,1,2,3,4,5,6,6],
[1,2,2,3,4,5,5,6],[1,2,3,3,4,4,5,6],[1,2,3,4,4,5,6,7],[1,2,3,4,5,6,7,8]]
: int list list
```

MG1 = tnf(MG2) and [MG1, rev(MG2)] = rev[MG1, rev(MG2)].

```
head!=palin, [head, kref(head)]=palin
```

```
- ispalindrome[1,1,2,1];
```

```
val it = false : bool
```

```
- kref[1,1,2,1];
```

```
val it = [1,2,1,1] : int list
```

```
- ispalindrome[1,1,2,1,1,2,1,1];
```

```
val it = true : bool
```

```
- ispalindrome [1,2,3,3];
```

```
val it = false : bool
```

```
- kref[1,2,3,3];
```

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

```
- ispalindrome[1,2,3,3,1,1,2,3];
```

```
val it = true : bool
```

```
- ispalindrome[1,2,2,3,3,4,4,2];
```

```
val it = false : bool
```

```
- [1,2,2,3] = tnf[3,4,4,2];
```

```
val it = true : bool
```

```
head!=palin, [head, rev(head)]=palin
```

```
- ispalindrome[1,2,1,3,3,1,2,1,3];
```

```
val it = false : bool
```

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

Semiotische Palindrome

Die Frage ist wohl, welches sind die Bedingungen dafür, dass die Wiederholung eines Morphogrammes ein Palindrom zu definieren vermag?

Für den *semiotischen* Fall ist die Bedingung recht einfach: Die wiederholte Zeichenreihe ist eine exakte Umkehrung, d.h. Inversion oder Reversion, der ursprünglichen Zeichenreihe: $\text{Palindrome} = (\text{ZR}, \text{rev}(\text{ZR}))$. D.h. der Wiederholungsmodus der Reversion ist ausreichend um ein Palindrom semiotisch zu charakterisieren.

Rekursive Definition für semiotische Palindrome

Die induktive bzw. rekursive Definition für Palindrome sieht wie folgt aus:

1. Das leere Wort ϵ (das Wort der Länge 0, der „Leerstring“) ist ein Palindrom.
2. Jedes Wort x der Länge 1 ist ein Palindrom.
3. Ist α ein Symbol und x ein Palindrom, so ist $\alpha x \alpha$ ein Palindrom.
4. Kein anderes Wort ist ein Palindrom. (WiKi, Palindrom)

Morphogrammatischer Fall

Die Konstruktion von morphogrammatischen Palindromen lässt eine weitere interessante Abstraktion definieren.

Bestimmung von Palindromen

Es lassen sich mindestens zwei interessante Vorgehensweisen zur Bestimmung eines morphogrammatischen Palindromes unterscheiden:

1. Gegeben sei ein beliebiges Morphogramm. Wie lässt sich aus diesem beliebigen Morphogramm ein Palindrom konstruieren?

Die ersten zu unterscheidenden Modi scheinen einfach definierbar zu sein: Repetition und Reversion. Schwieriger zu bestimmen sind die verschiedenen Varianten des Modus der Akkretion. Dieser ist allerdings auch für die akkretive Repetition und akkretive Reversion zu beachten.

2. Gegeben sei ein Palindrom beliebiger Länge. Wie lässt sich daraus die Menge aller Palindrome gleicher Länge (mit gleichem Kopf) konstruieren?

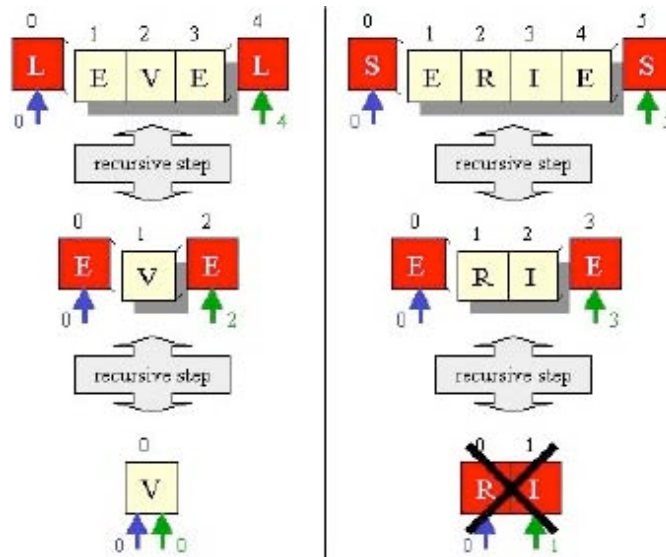
Es kann wohl angenommen, dass alle aus dem gegebenen Palindrom konstruierten Palindrome bezüglich ihres Kopfes, d.h. der ersten Hälfte des gegebenen Palindromes übereinstimmen. Dieser Fall lässt sich als eine spezielle Form der Emanation von Palindromen verstehen.

3. Zudem lässt sich das nicht-konstruktive Verfahren der Filterung benutzen. Es wird die *Tcontexture* der Länge n aufgerufen, womit alle Tritogramme der Länge n produziert werden. Daraus werden durch Filterung, *List.filter ispalindrome "Tcontexture (n)"*, die gültigen Palindrome selektiert. Dies gibt jedoch kein konstruktives Verfahren zur Bestimmung von Palindromen an.

Die Auflistung der aller Morphogramme einer Kontextur durch "*Tcontexture(n)*", kann allerdings in Größenordnungen führen, die für Computer des gemeinen Hausgebrauchs nicht mehr zugänglich sind. Dem lässt sich ein wenig begegnen, wenn die Abstraktion der Deutrogramme eingeführt wird, berechnet durch "*Dcontexture(n)*".

Eine andere Zugangsweise ist gegeben durch die additive Konkatination, “kconcat”, von Morphogrammen und Palindromen.

Rekursive Definition für semiotische Palindrome



<http://jade-cheng.com/hpu/2012-spring/csci-2912/recursion/>

Wenn MG1 morphogrammatisch equivalent Morphogramm MG2 ist, und zusätzlich das zusammengesetzte Morphogramm [MG1, MG2] palindromische ist, dann ist MG1 und MG2 palindromisch äquivalent: $MG1 =_{\text{palin}} MG2$.

IF $MG1 =_{\text{MG}} \text{tnf}(MG2)$ and $[MG1, \text{rev}^+(MG2)] =_{\text{MG}} \text{rev}^+[MG1, \text{rev}^+(MG2)]$

THEN

$MG1 =_{\text{palin}} MG2$.

Die zwei Konstituenten, MG1 und MG2, eines morphogrammatischen Palindromes, $MG1 =_{\text{MG}} \text{rev}(MG1)$, sind morphogrammatisch equivalent, wenn sie ein Palindrom definieren.

Die Frage ist, welche Umformungen MG2 von MG1 konstituieren zusammen mit MG1 ein morphogrammatische Palindrome?

Die Antwort mag leicht zirkulär erscheinen.

MG2 konstituiert mit MG1 ein Palindrom, wenn MG1 zusammen mit MG2 ein Palindrom definiert. Dann ist auch MG2 eine zulässige Umformung von MG1.

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

MG1 = [1,1,2,3]

MG2 = [4,2,1,1]

kref(MG2) = MG1

MG2 = kref(MG1).

- ispalindrome[1,1,2,3,3,4,2,2];

val it = **false** : bool

MG2 = [3,4,2,2]

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

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

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

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

Test

```

- ispalindrome[1,1,2,3,1,2,3,3];
val it = true : bool
- ispalindrome[1,1,2,3,3,1,2,2];
val it = true : bool
- ispalindrome[1,1,2,3,3,2,4,4];
val it = true : bool
- ispalindrome[1,1,2,3,3,4,2,2];
val it = false : bool
- ispalindrome[1,1,2,3,3,2,4,4];
val it = true : bool
- ispalindrome[1,1,2,3,3,2,4,4];
val it = true : bool
- ispalindrome[1,1,2,3,3,1,2,2];
val it = true : bool
- ispalindrome[1,1,2,3,4,1,3,3];
val it = false : bool
- ispalindrome[1,1,2,3,4,2,3,3];   tl h1 = tl h2
val it = false : bool
- ispalindrome[1,1,2,3,4,1,2,2];
val it = true : bool

- ispalindrome[1,1,2,3,4,5,6,6];
val it = true : bool
- ispalindrome[1,1,2,3,4,5,3,3];   h1 ∩ h2 != ∅, CR
val it = false : bool
- ispalindrome[1,1,2,3,4,5,1,1];
val it = true : bool

- Tcontexture 4;
val it =
[[[1,1,1,1],[1,1,2,2],[1,2,1,2], [1,2,2,1],
  [1,1,1,2],[1,1,2,1],[1,2,1,1], [1,2,2,2],[1,1,2,3], [1,2,1,3] non-rev,
  [1,2,3,1], [1,2,2,3],
  [1,2,3,2] accr-rev- , [1,2,3,3],
  [1,2,3,4]]] : int list list

```

Palindromes

```

[[[1,1,1,1], rep [1,1,2,2], rep-accr
[1,2,1,2], rep
[1,2,2,1], rev , [1,2,3,1], rev-accr, [1,2,2,3], rep-accr, [1,2,3,4] rep-accr.

```

Alle Palindrom-konforme Umformungen MG2 von MG1 sind Akkretionen von Repetitionen oder Inversionen von MG1.

Es wird also zwischen *semiotischer*, *morphogrammatischer* und *palindromischer* Äquivalenz unterschieden.

Die palindromischer Äquivalenz ist für semiotische Palindrome unter der Reversion *rev* trivial. Für sie gilt: mit Zeichenreihe ZR1 = (1,2,2,3) und ZR2 = rev(ZR1), gilt das Palindrom $ZR^{rev} = (ZR1, rev(ZR1))$. Was der klassischen Definition eines Palindromes entspricht.

Damit gilt auch die palindromische Äquivalenz: $ZR1 =_{\text{palin}} ZR2$.

Nicht jede Repetition erzeugt ein Palindrom:

```

- ispalindrome[1,1,1,2,1,1,1,2];
val it = false : bool

```

Der Grund liegt darin, dass das Morphogramm [1,1,1,2] kein Palindrom darstellt.

Ist jedoch das zu repetierende Morphogramm ein Palindrom, dann ist trivialerweise auch die Repetition ein Palindrom:

head=palin, [palin, palin]=palin

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

ispalindrome [1,2,3,4];
val it = true : bool
- ispalindrome[1,2,3,4,1,2,3,4];
val it = true : bool
```

Palin = [palin, rep(palin)]

Jedoch erzeugt jede Reversion auch auf dem morphogrammatischen Level ein Palindrom:

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

Die interessantesten Wiederholungen sind gewiss die Wiederholungen im Sinne von Umformungen. Nach dem Motto "*iterations alter*" sind Umformungen nicht mehr an die getreue Wiederholung des Originals, sei es durch Repetition oder durch Umkehrung, gebunden.

Nichtsdestotrotz sind Umformung immer noch morphogrammatisch äquivalent ihren Originalen gegenüber.

Jedoch gilt auch hier, nicht jede morphogrammatisch äquivalente Umformung bildet zusammen mit dem Original ein Palindrom.

Beispiel

Gegeben sei das Palindrome = [1,2,1,3,3,1,4,1].

Hier ist MG1 = [1,2,1,3] kein Palindrome.

MG2 = [3,1,4,1] ist auch nicht einfach $\text{tnf}(\text{MG1})$, sondern verlangt eine Reversion: $\text{tnf}(\text{rev}(\text{MG1}))$.

MG1 und MG2 sind keine Palindrome. Jedoch ist die Komposition [MG1, MG2] ein Palindrom.

```
- ispalindrome [1,2,1,3,3,1,4,1];
val it = true : bool
- [1,2,1,3] = tn timer (rev[3,1,4,1]);
val it = true : bool
```

Jedoch:

```
- [1,2,1,3] = rev(tn timer [3,1,4,1]);
val it = false : bool
```

Und:

```
- tn timer [3,1,4,1];
val it = [1,2,3,2] : int list
- ispalindrome[1,2,3,2];
val it = false : bool
```

Der Fall, das MG1 ein Teil-Palindrom im Gesamt-Palindrom ist, stellt sich somit als Spezialfall heraus.

Es ist nicht notwendig, dass MG1 selbst ein Palindrom ist, um durch seine Umformung ein Palindrom zu bestimmen.

Damit nähert sich diese Untersuchung dem klassischen Ausgangspunkt zu, nachdem jedwede Zeichenreihe zum Palindrom wird, wenn sie mit ihrer Verdoppelung und Reversion verknüpft wird.

Klassische Situation

Nehme eine beliebige Zeichenreihe, füge ihre Inversion an, und das Resultat ist ein Palindrom.

Wenn die beliebige Zeichenreihe selbst ein Palindrome ist, gilt die Verknüpfung der Inversion trivialerweise.

Also gilt das Lemma:

Nehme ein Palindrom, wiederhole es, und das Resultat ist ein Palindrom.

Morphogrammatische Situation

Nehme ein Palindrom, wiederhole es in Trito-Normalform, und das Resultat ist ein Palindrom.

Nehme ein beliebiges Morphogramm, füge seine Inversion an, und das Resultat ist ein Palindrom.

Palindrome = [1,2,3,4,4,5,1,2]

MG1 = [1,2,3,4], MG1 ∈ Palin

MG2 = [4,5,1,2]

tnf[MG2] = MG1

- ispalindrome[1,2,3,4,1,4,2,3];

val it = false : bool

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

val it = true : bool

- ispalindrome[1,2,3,4];

val it = true : bool

- ispalindrome[5,1,2,3];

val it = false : bool

- tnf[5,1,2,3];

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

Allgemeines Schema

$$\text{Palindrom}_{\text{MG}} = \left(\begin{array}{l} \text{repetition : } \omega = \omega^{\text{rep}} \\ \text{reversion : } \omega = \omega^{\text{rev}} \\ \text{accretion : } \omega = \omega^{\text{accr}} \end{array} \right) \text{ modulo palin}$$

1.5.7. Palindrom (8)

Ordnung muss sein. Speziell für Palindrome, die noch wenig erforscht sind.

Vergleich

- kconcat[1,1][1,1];

val it = [[1,1,1,1],[1,1,2,2]] : int list list

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

val it =

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

: int list list-

Bespiel

ispalindrome "kconcat [1,2,2,3][1,2,2,3]" = ispalindrome "Tcontexture/head=[1,2,2,3]" = card 14

- kconcat [1,2,2,3][1,2,2,3]; : 34

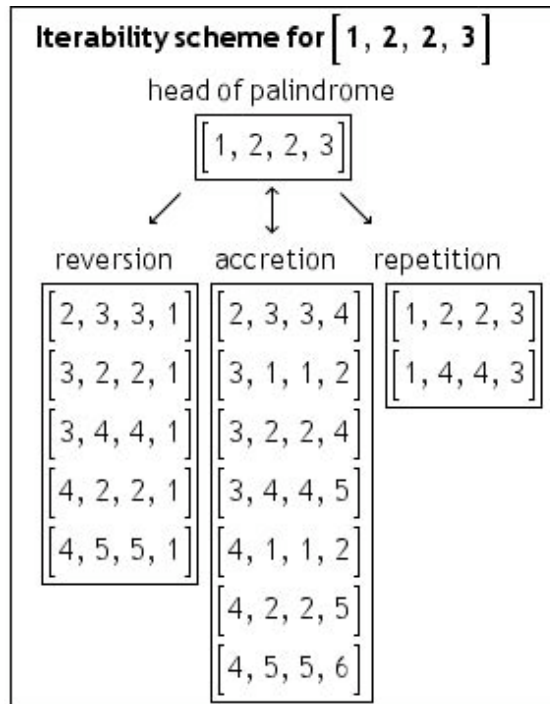
val it =

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

List.filter ispalindrome "kconcat [1,2,2,3][1,2,2,3]" : 14

val it =

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

**Some Palindrome Tables based on kconcat**

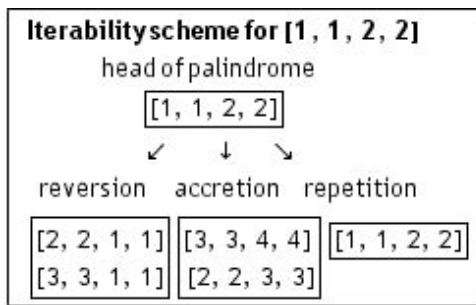
- kconcat[1,1,2,2][1,1,2,2];

val it =

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

Palindromes: val it =

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



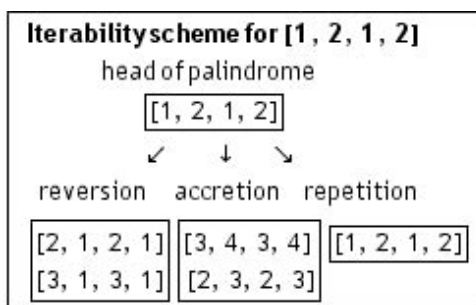
- kconcat[1,2,1,2][1,2,1,2];

val it =

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

Palindromes: val it =

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



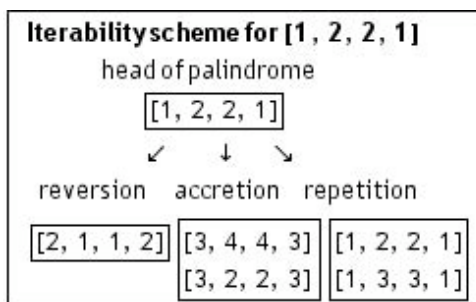
- kconcat[1,2,2,1][1,2,2,1];

val it =

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

Palindromes: val it =

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



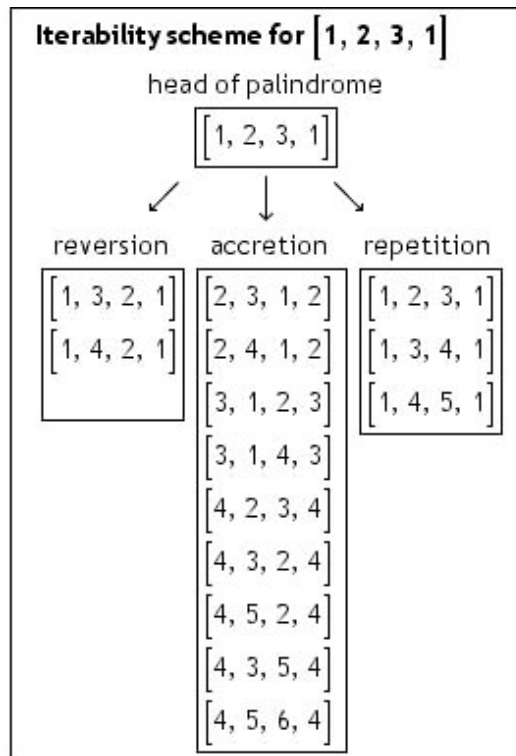
- kconcat[1,2,3,1][1,2,3,1];

val it =

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

Palindromes: val it =

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



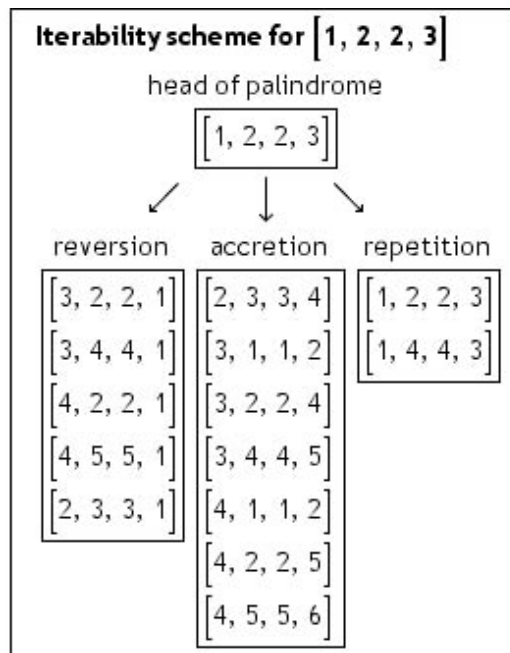
```
- kconcat[1,2,2,3][1,2,2,3];
```

```
val it =
```

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

```
Palindromes: val it =
```

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



- kconcat[1,2,3,4][1,2,3,4];

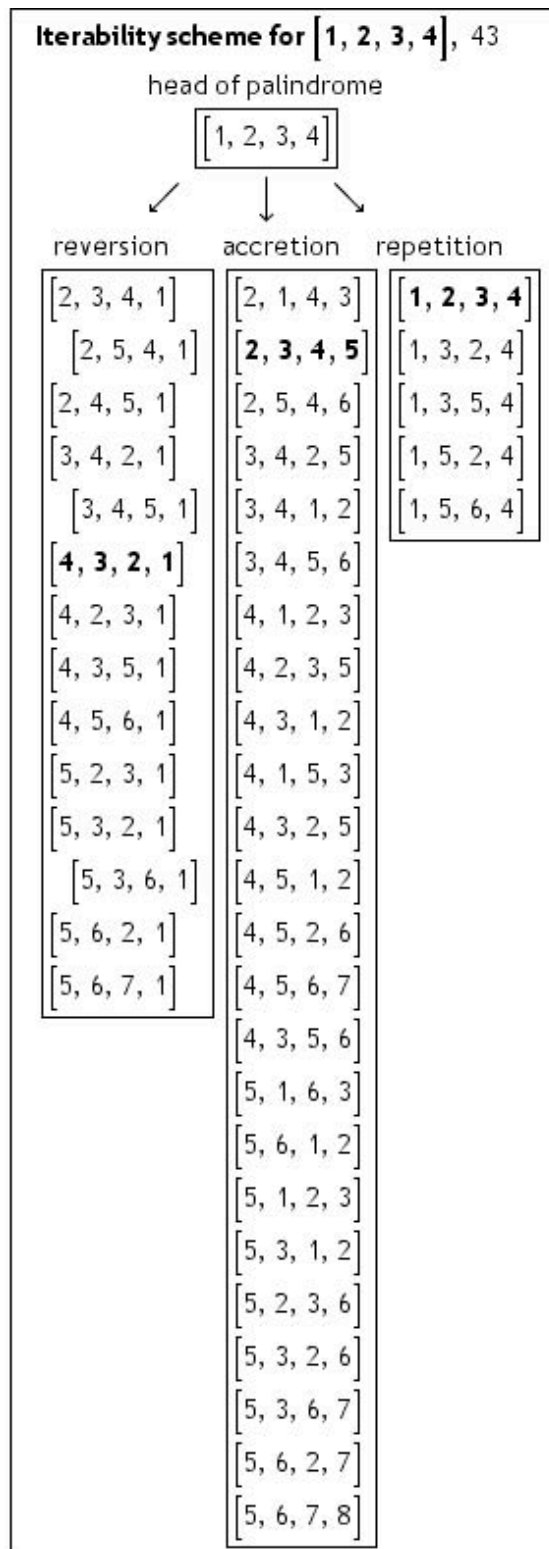
val it =

[1,2,3,4,1,2,3,4],[1,2,3,4,1,2,4,3],[1,2,3,4,1,3,2,4],[1,2,3,4,1,3,4,2],
 [1,2,3,4,1,4,2,3],[1,2,3,4,1,4,3,2],[1,2,3,4,2,1,3,4],[1,2,3,4,2,1,4,3],
 [1,2,3,4,2,3,1,4],[1,2,3,4,2,3,4,1],[1,2,3,4,2,4,1,3],[1,2,3,4,2,4,3,1],
 [1,2,3,4,3,1,2,4],[1,2,3,4,3,1,4,2],[1,2,3,4,3,2,1,4],[1,2,3,4,3,2,4,1],
 [1,2,3,4,3,4,1,2],[1,2,3,4,3,4,2,1],[1,2,3,4,4,1,2,3],[1,2,3,4,4,1,3,2],
 [1,2,3,4,4,2,1,3],[1,2,3,4,4,2,3,1],[1,2,3,4,4,3,1,2],[1,2,3,4,4,3,2,1],
 [1,2,3,4,1,2,3,5],[1,2,3,4,1,2,5,3],[1,2,3,4,1,3,2,5],[1,2,3,4,1,3,5,2],
 [1,2,3,4,1,5,2,3],[1,2,3,4,1,5,3,2],[1,2,3,4,2,1,3,5],[1,2,3,4,2,1,5,3],
 [1,2,3,4,2,3,1,5],[1,2,3,4,2,3,5,1],[1,2,3,4,2,5,1,3],[1,2,3,4,2,5,3,1],
 [1,2,3,4,3,1,2,5],[1,2,3,4,3,1,5,2],[1,2,3,4,3,2,1,5],[1,2,3,4,3,2,5,1],
 [1,2,3,4,3,5,1,2],[1,2,3,4,3,5,2,1],[1,2,3,4,5,1,2,3],[1,2,3,4,5,1,3,2],
 [1,2,3,4,5,2,1,3],[1,2,3,4,5,2,3,1],[1,2,3,4,5,3,1,2],[1,2,3,4,5,3,2,1],
 [1,2,3,4,1,2,4,5],[1,2,3,4,1,2,5,4],[1,2,3,4,1,4,2,5],[1,2,3,4,1,4,5,2],
 [1,2,3,4,1,5,2,4],[1,2,3,4,1,5,4,2],[1,2,3,4,2,1,4,5],[1,2,3,4,2,1,5,4],
 [1,2,3,4,2,4,1,5],[1,2,3,4,2,4,5,1],[1,2,3,4,2,5,1,4],[1,2,3,4,2,5,4,1],
 [1,2,3,4,4,1,2,5],[1,2,3,4,4,1,5,2],[1,2,3,4,4,2,1,5],[1,2,3,4,4,2,5,1],
 [1,2,3,4,4,5,1,2],[1,2,3,4,4,5,2,1],[1,2,3,4,5,1,2,4],[1,2,3,4,5,1,4,2],
 [1,2,3,4,5,2,1,4],[1,2,3,4,5,2,4,1],[1,2,3,4,5,4,1,2],[1,2,3,4,5,4,2,1],
 [1,2,3,4,1,2,5,6],[1,2,3,4,1,5,2,6],[1,2,3,4,1,5,6,2],[1,2,3,4,2,1,5,6],
 [1,2,3,4,2,5,1,6],[1,2,3,4,2,5,6,1],[1,2,3,4,5,1,2,6],[1,2,3,4,5,1,6,2],
 [1,2,3,4,5,2,1,6],[1,2,3,4,5,2,6,1],[1,2,3,4,5,6,1,2],[1,2,3,4,5,6,2,1],
 [1,2,3,4,1,3,4,5],[1,2,3,4,1,3,5,4],[1,2,3,4,1,4,3,5],[1,2,3,4,1,4,5,3],
 [1,2,3,4,1,5,3,4],[1,2,3,4,1,5,4,3],[1,2,3,4,3,1,4,5],[1,2,3,4,3,1,5,4],
 [1,2,3,4,3,4,1,5],[1,2,3,4,3,4,5,1],[1,2,3,4,3,5,1,4],[1,2,3,4,3,5,4,1],
 [1,2,3,4,4,1,3,5],[1,2,3,4,4,1,5,3],[1,2,3,4,4,3,1,5],[1,2,3,4,4,3,5,1],
 [1,2,3,4,4,5,1,3],[1,2,3,4,4,5,3,1],[1,2,3,4,5,1,3,4],[1,2,3,4,5,1,4,3],
 [1,2,3,4,5,3,1,4],[1,2,3,4,5,3,4,1],[1,2,3,4,5,4,1,3],[1,2,3,4,5,4,3,1],
 [1,2,3,4,1,3,5,6],[1,2,3,4,1,5,3,6],[1,2,3,4,1,5,6,3],[1,2,3,4,3,1,5,6],
 [1,2,3,4,3,5,1,6],[1,2,3,4,3,5,6,1],[1,2,3,4,5,1,3,6],[1,2,3,4,5,1,6,3],
 [1,2,3,4,5,3,1,6],[1,2,3,4,5,3,6,1],[1,2,3,4,5,6,1,3],[1,2,3,4,5,6,3,1],
 [1,2,3,4,1,4,5,6],[1,2,3,4,1,5,4,6],[1,2,3,4,1,5,6,4],[1,2,3,4,4,1,5,6],
 [1,2,3,4,4,5,1,6],[1,2,3,4,4,5,6,1],[1,2,3,4,5,1,4,6],[1,2,3,4,5,1,6,4],
 [1,2,3,4,5,4,1,6],[1,2,3,4,5,4,6,1],[1,2,3,4,5,6,1,4],[1,2,3,4,5,6,4,1],
 [1,2,3,4,1,5,6,7],[1,2,3,4,5,1,6,7],[1,2,3,4,5,6,1,7],[1,2,3,4,5,6,7,1],

```
[1,2,3,4,2,3,4,5],[1,2,3,4,2,3,5,4],[1,2,3,4,2,4,3,5],[1,2,3,4,2,4,5,3],
[1,2,3,4,2,5,3,4],[1,2,3,4,2,5,4,3],[1,2,3,4,3,2,4,5],[1,2,3,4,3,2,5,4],
[1,2,3,4,3,4,2,5],[1,2,3,4,3,4,5,2],[1,2,3,4,3,5,2,4],[1,2,3,4,3,5,4,2],
[1,2,3,4,4,2,3,5],[1,2,3,4,4,2,5,3],[1,2,3,4,4,3,2,5],[1,2,3,4,4,3,5,2],
[1,2,3,4,4,5,2,3],[1,2,3,4,4,5,3,2],[1,2,3,4,5,2,3,4],[1,2,3,4,5,2,4,3],
[1,2,3,4,5,3,2,4],[1,2,3,4,5,3,4,2],[1,2,3,4,5,4,2,3],[1,2,3,4,5,4,3,2],
[1,2,3,4,2,3,5,6],[1,2,3,4,2,5,3,6],[1,2,3,4,2,5,6,3],[1,2,3,4,3,2,5,6],
[1,2,3,4,3,5,2,6],[1,2,3,4,3,5,6,2],[1,2,3,4,5,2,3,6],[1,2,3,4,5,2,6,3],
[1,2,3,4,5,3,2,6],[1,2,3,4,5,3,6,2],[1,2,3,4,5,6,2,3],[1,2,3,4,5,6,3,2],
[1,2,3,4,2,4,5,6],[1,2,3,4,2,5,4,6],[1,2,3,4,2,5,6,4],[1,2,3,4,4,2,5,6],
[1,2,3,4,4,5,2,6],[1,2,3,4,4,5,6,2],[1,2,3,4,5,2,4,6],[1,2,3,4,5,2,6,4],
[1,2,3,4,5,4,2,6],[1,2,3,4,5,4,6,2],[1,2,3,4,5,6,2,4],[1,2,3,4,5,6,4,2],
[1,2,3,4,2,5,6,7],[1,2,3,4,5,2,6,7],[1,2,3,4,5,6,2,7],[1,2,3,4,5,6,7,2],
[1,2,3,4,3,4,5,6],[1,2,3,4,3,5,4,6],[1,2,3,4,3,5,6,4],[1,2,3,4,4,3,5,6],
[1,2,3,4,4,5,3,6],[1,2,3,4,4,5,6,3],[1,2,3,4,5,3,4,6],[1,2,3,4,5,3,6,4],
[1,2,3,4,5,4,3,6],[1,2,3,4,5,4,6,3],[1,2,3,4,5,6,3,4],[1,2,3,4,5,6,4,3],
[1,2,3,4,3,5,6,7],[1,2,3,4,5,3,6,7],[1,2,3,4,5,6,3,7],[1,2,3,4,5,6,7,3],
[1,2,3,4,4,5,6,7],[1,2,3,4,5,4,6,7],[1,2,3,4,5,6,4,7],[1,2,3,4,5,6,7,4],
[1,2,3,4,5,6,7,8]] : int list list
```

Palindromes: val it =

```
[[1,2,3,4,1,2,3,4],[1,2,3,4,1,3,2,4],[1,2,3,4,2,1,4,3],[1,2,3,4,2,3,4,1],
[1,2,3,4,3,4,1,2],[1,2,3,4,3,4,2,1],[1,2,3,4,4,1,2,3],[1,2,3,4,4,2,3,1],
[1,2,3,4,4,3,1,2],[1,2,3,4,4,3,2,1],[1,2,3,4,5,1,2,3],[1,2,3,4,5,2,3,1],
[1,2,3,4,5,3,1,2],[1,2,3,4,5,3,2,1],[1,2,3,4,1,5,2,4],[1,2,3,4,2,5,4,1],
[1,2,3,4,4,5,1,2],[1,2,3,4,4,5,2,1],[1,2,3,4,5,6,1,2],[1,2,3,4,5,6,2,1],
[1,2,3,4,1,3,5,4],[1,2,3,4,3,4,5,1],[1,2,3,4,4,1,5,3],[1,2,3,4,4,3,5,1],
[1,2,3,4,5,1,6,3],[1,2,3,4,5,3,6,1],[1,2,3,4,1,5,6,4],[1,2,3,4,4,5,6,1],
[1,2,3,4,5,6,7,1],[1,2,3,4,2,3,4,5],[1,2,3,4,3,4,2,5],[1,2,3,4,4,2,3,5],
[1,2,3,4,4,3,2,5],[1,2,3,4,5,2,3,6],[1,2,3,4,5,3,2,6],[1,2,3,4,2,5,4,6],
[1,2,3,4,4,5,2,6],[1,2,3,4,5,6,2,7],[1,2,3,4,3,4,5,6],[1,2,3,4,4,3,5,6],
[1,2,3,4,5,3,6,7],[1,2,3,4,4,5,6,7],[1,2,3,4,5,6,7,8]] : int list list
```



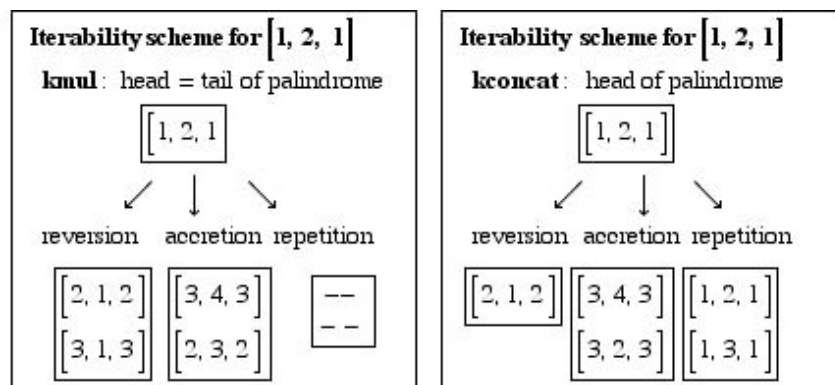
- kconcat[1,2,2,3,1,2,2,3][1,2,2,3,1,2,2,3]; : 34

val it =

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

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

```
[[1, 2, 1, 2, 1, 2, 1, 2, 1], [1, 2, 1, 3, 1, 3, 1, 2, 1], [1, 2, 1, 2, 3, 2, 1, 2, 1],  
[1, 2, 1, 3, 4, 3, 1, 2, 1]] : intlistlist
```



Sommerliche Anmerkungen

Gewiss sind solche Untersuchungen äusserst pingelig und langweilig. Umso mehr erstaunt es, welche überraschende Konsequenzen für eine Theorie der Berechenbarkeit sich ergeben, wenn diese Konzepte der asymmetrischen Palindromie auf die entsprechenden Konstruktionen der Selbstanwendung angewandt werden, die klassischerweise unweigerlich zu den bekannten Unentscheidbarkeitstheoremen führen.

Es fragt sich, ob nicht gerade diese stolzen Konstruktionen im Modus semiotischer Identität jegliche vorstellbare Langweiligkeit bei weitem übertreffen.

Es scheint mir nichts Lächerlicheres zu geben, als der jeweilige stolze Hinweis auf den eben mithilfe der Selbstanwendungsmethode konstruierte *logische* Widerspruch bei der Beweisführung der Unentscheidbarkeitstheoreme.

1.5.8. Graphematics of the Stirling Turn: From equivalence classes to iterability

With this approach of the Stirling Turn, a sign sequence, string or word, is conceived as a repetition (Wiederholung) of itself. Such a string might be build of other strings that are repetitions of themselves. Finally, what is called in semiotics an atomic sign of an alphabet appears as the smallest string of self-repetition, i.e. repetition of itself.

Obviously, this approach tries not to refer to the physical graphemic occurrence of letters that leads to the definition of words as equiform appearances of physical inscriptions, i.e. of equivalence classes over equiform inscriptions.

Letters of the semiotic approach are in fact abstractions over equiform physical inscriptions.

Stability of semiotic repetition then is a consequence of the abstractness and ideality of the form of equiform inscriptions or notations.

Taking graphematics and especially the *Stirling Turn* (kenogrammatics) into account, repetitions are unmasked just as one very special case of iterability (Derrida). Iterability involves at least the modi of repetition (iteration), reversion, accretion and metamorphosis.

According to Derrida, a sign that is not repeatable can not be considered a sign. The consequences, that repeatability is defining signs had not been drawn further.

The proposed Stirling Turn intends to make a further step in taking iterability first. Signs then are realizations of iterability, and iterability is inscribed by morphograms. This turn corresponds the transition from perception-oriented to

cognition-oriented thematizations of iscription and writing systems, i.e. the subversion from semiotics to graphematics.

Still very close to the repetition of strings with its maximal operational semiotic *security* and stability are the modi of iterability like reversion (reversion) and alterations (accretions) in contrast to metamorphosis.

Hence, the whole theory of computability shall start, in a first attempt, with the idea, concept, simulation and construction of *unrestricted iterability* of repetitions.

Until now, iterability is restricted to the concept of iteration of pre-given signs by the iterated application of pre-given rules (operators).

This distinction may still be correlated to the difference of abstract algebra for symbolic constructions (structures) and its 'dual' co-algebra for the structuration of streams.

Equality of strings are algebraically defined by equivalence classes of equiform sign sequences. Equality in co-algebras are defined as equivalences of events.

But both concepts or paradigms, the algebraic and the co-algebraic, remain in the secure culture of semiotics (and category theory) as there common ground.

Morphogrammatic equivalence is therefore defined as behavioral sameness, independently of algebraic or co-algebraic grounding.

The proposed morphogenetic turn, basically supported by the 'Stirling turn', is the viewpoint from which semiotics appears as a non-reflected positivism of manipulated pre-given signs.

The Stirling approach is not presupposing atomic signs and sign sequences but, 'paradoxically', their iterability.

Morphograms, therefore, are not ultra-abstract semiotic constructions based on semiotic or truth-value strings, but unrestricted iterations of themselves.

With the strategy of introducing palindromes, and palindromy or palindromicity, as one of the fundamental categories of kenomic scripturality, an important step towards a concretization of morphogrammatic conceptualization and operationalization is achieved.

Therefore, the identity of a sign sequence is defined by its iterative palindromicity.

This suggests that each sign sequence is palindromic in itself, i.e. a symmetric iteration of itself.

The identity of a string of signs is a special case of iterability.

This might be in conflict with the semiotic definition of the equality of signs and sign-sequences.

But it becomes more obvious on the level of morphogrammatcs.

Two sign sequences are equal iff the interaction (behavior) of the sequence and its re-reading are not distinguishable.

$A \equiv A$ iff $A \wedge A = A$

GSB, Mersenne

The identity of a sign sequence is not changing in the process of reading. A re-read

text is equal to the read text.

Morphogrammatically, this is a very special case of repetition, i.e. iterability.

$w \equiv_{MG} w$ iff $w = \text{iter}(w)$

Imitons l'Ingénieur

"L'abbé physicien Marin Mersenne (1586-1648), avec qui Descartes échangea une correspondance abondante, écrit dans ses Questions philosophiques : << *c'est faire acte de vertu que de connaître par la science l'ordre exact des phénomènes, parce que par cette connaissance nous imitons l'Ingénieur divin qui lui-même a pris plaisir à agencer cette machine [le monde]* >> [cité par Lenoble, op. cit. : 322]."

Suzanne Chappaz-Wirthner, Le thème de la connaissance interdite dans *L'Homme au sable* de E.T. Hoffmann

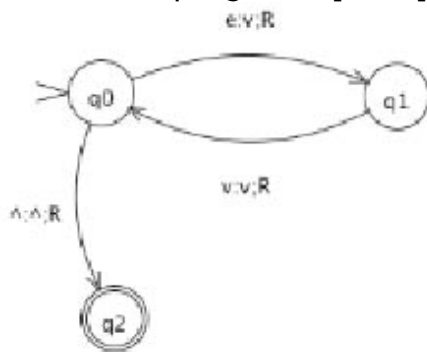
<http://www.cairn.info/>

2. Weitere Beispiele

2.1. Beispiele für morphoTM

1. Beispiel: Übersetzung

Übersetze das Morphogramm [AAAA] in das Morphogramm [ABCD]



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] $\Rightarrow$ [BAAA] $\Rightarrow$ [BACA] $\Rightarrow$ [BACD] $\Rightarrow$ [ABCD]
 $\Rightarrow$ ^: accepted

Explanation

read "e": write "v"; goto R.

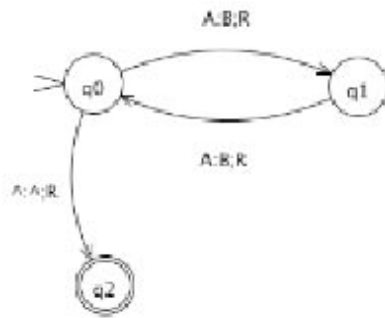
read "v": write "v"; goto R.

EN-notation, plus linearized morphogram in trito-normal form (tnf) with [AAAA] to [ABCD].

run: [AAAA] to [ABCD].

2. Beispiel: Selbstabbildung

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

q0; # : #; R, q1
q1; e1 : v1; R, q2
q2; e2 : v2; R, q1
q1; e3 : v3; R, q1
q1; v4 : e4; R, q1
q1; v5 : e5; R, q2
q2; v6 : e6; R, q1
q1; # : #, R, q3

Linearized notation, plus EN-structure.

run: [AAAA] to [BBBB]

run= [AAAA] => [BAAA] => [BABA] => [BBBA] => [BBBB] => ^:
accepted.

$$\text{EN: } \begin{pmatrix} e1 & e2 & e4 \\ e3 & e5 & - \\ e6 & - & - \end{pmatrix} \begin{pmatrix} \mathbf{v1} & v2 & v4 \\ e3 & e5 & - \\ e6 & - & - \end{pmatrix} \begin{pmatrix} v1 & \mathbf{e2} & v4 \\ v3 & v5 & - \\ e6 & - & - \end{pmatrix} \begin{pmatrix} e1 & e2 & v4 \\ \mathbf{e3} & v5 & - \\ v6 & - & - \end{pmatrix} \begin{pmatrix} e1 & e2 & e4 \\ e3 & e5 & - \\ \mathbf{e6} & - & - \end{pmatrix}$$

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] => [BABA] => [BBBA] => [BBBB] =_{MG} [AAAA]" is self-applicative:

morphoTM([AAAA]) =_{MG} [AAAA].

One more turn:

run= [AAAA] => [BAAA] => [BABA] => [BBBA] => [BBBB] =>
[BBBB] => [ABBB] => [ABAB] => [AAAB] => [AAAA] => ^: accepted.

Hence:

morphoTM(morphoTM([AAAA])) =_{SEM} [AAAA].

2.2. Beispiele für morphoFSM

2.2.1. Einfache Beispiele für Recognizer morphoFSM

Beispiel 1

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:

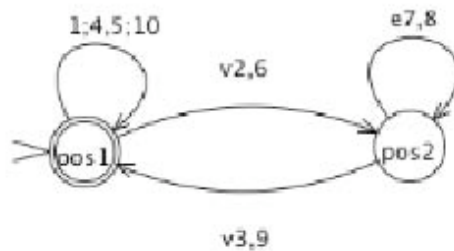
ENTable[abac]	1	2	3
1	v ₁	e ₂	v ₄
2	–	v ₃	v ₅
3	–	–	v ₆

ENstructure automaton table of [abac] :

MorphoFSA[abac]	pos1	pos2	pos3
pos1	initial	v ₁	v ₄
pos2	v ₃	e ₂	v ₆
pos3	–	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 of {e, v}_j, j ∈ s(m).

Beispiel 2



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}

2.2.2. Ausführliches Beispiel eines Recognizers morphoFSM

Morphogram[abbcddcdd]

tnf[a, b, b, c, d, d, c, d, d] = [1, 2, 2, 3, 4, 4, 3, 4, 4].

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

val it =

[[],

[(1, 2, *N*),

[(1, 3, *N*), (2, 3, *E*) = [**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*) = [**dd**]],

[(1, 7, *N*), (2, 7, *N*), (3, 7, *N*), (4, 7, *E*) = [**c' 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*),

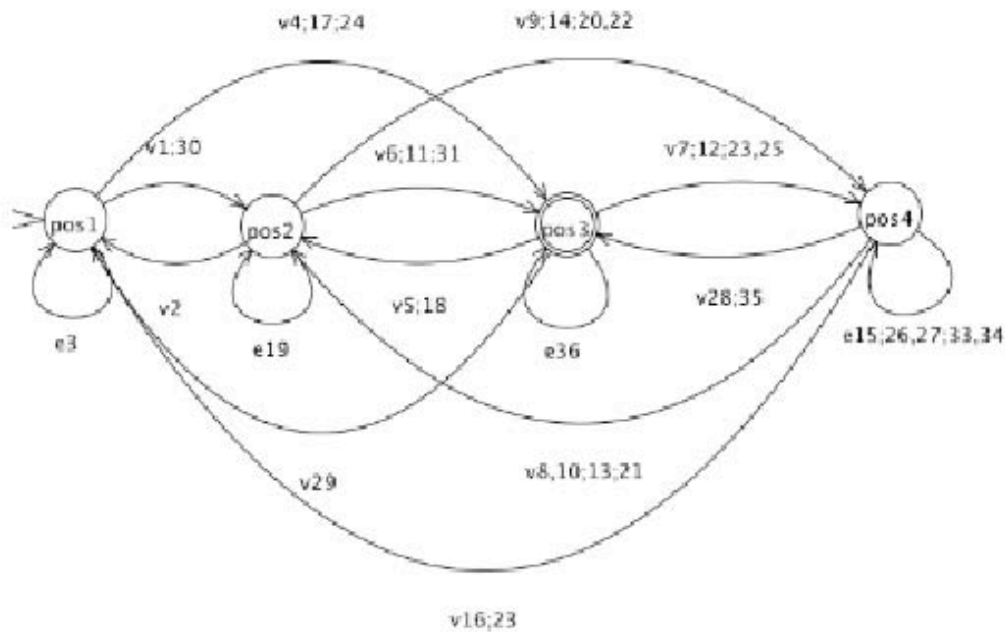
[(1, 9, *N*), (2, 9, *N*), (3, 9, *N*), (4, 9, *N*), (5, 9, *E*) = [**d' d**], (6, 9, *E*) = [**d' d**], (7, 9, *N*), (8, 9, *E*) = [**d**],

: (int * int * EN) list list

ENstructure table with enumeration

EN[abbcbbcb]	1	2	3	4	5	6	7	8	9
1	[]	v1	v2	v4	v7	v11	v16	v22	v29
2	□	□	e3	v5	v8	v12	v17	v23	v30
3	□	□	□	v6	v9	v13	v18	v24	v31
4	□	□	□	□	v10	v14	e19	v25	v32
5	□	□	□	□	□	e15	v20	e26	e33
6	□	□	□	□	□	□	v21	e27	e34
7	□	□	□	□	□	□	□	v28	v35
8	□	□	□	□	□	□	□	□	e36

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

Decomposition of the main table **MorphoFSA[abbcddcdd]** into its *sub-tables* by three layered runs.

First run

MorphoFSA[abbcddcdd]	pos ₁	pos ₂	pos ₃	pos ₄
pos ₁	e3	v1	v4	–
pos ₂	v2	e19	v6	v9
pos ₃	v29	v5	e36	v7
pos ₄	v16	v8, 10	v28	e15

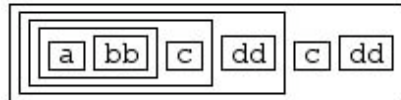
Second run

MorphoFSA[abbcddcdd]	pos ₁	pos ₂	pos ₃	pos ₄
pos ₁	–	v30	v17	–
pos ₂	–	–	v11	v14
pos ₃	–	v18	–	v12
pos ₄	v23	v13	v35	e26, 27

Third run

MorphoFSA[abbcddcdd]	pos ₁	pos ₂	pos ₃	pos ₄
pos ₁	–	–	v24	–
pos ₂	–	–	v31	v20, 22
pos ₃	–	–	–	v23, 25
pos ₄	–	v21	–	e33, 34

Monomorphic decomposition of the morphogram [abbcddcdd]



=

MG	loc ₁	loc ₂	loc ₃	loc ₄	loc ₅	loc ₆
Dec	mg ₁	mg ₂	mg ₃	mg ₄	mg ₃	mg ₄
Ken	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>c</i>	<i>d</i>
	∅	<i>b</i>	∅	<i>d</i>	∅	<i>d</i>

- ENstructure[1,2,3,4,3,4];

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,N),(3,6,N),(4,6,E),(5,6,N)]] : (int * int * EN) list list

```

MorphoFSA[MG]	pos ₁	pos ₂	pos ₃	pos ₄
pos ₁	–	v1	v7	v4; 11
pos ₂	–	–	v3; 8	v12
pos ₃	v2	–	e9	v6; 13, 15
pos ₄	–	v5	–	v10; 14

3. Where are the diamonds left?

3.1. What can we learn from earlier approaches?

"Living systems exist as singular entities that operate as totalities in interactions in a medium where each conserves its individual identity under the form of a unicellular or a multicellular organism. That is, living systems exist as organisms in the realization of their living." (Maturana, p.12)

Humberto Maturana Romesin, AUTOPOIESIS, STRUCTURAL COUPLING AND COGNITION, 1999

<http://www.isss.org/maturana.htm>

Maturana's concept of structural coupling which is enabling interactions and communication between different closed, i.e. autopoietic systems got many explanations.

A more advanced explanation to the concept related to the interaction between different living systems that are defining and enabling each other is offered with the *diamond category* approach.

First, autopoietic systems shouldn't start with one autopoietic system alone and then extending the framework to more than one autopoietic system. There are no genuine concepts to realize structural coupling within the means of the conception of a singular autopoietic systems.

Second, because structural coupling is focused on *structures* and *organization* and not on any information processing, a diamond approach which is dealing genuinely with structures and environment of structures might offer a mechanism of coupling.

Third, diamonds are intrinsically separated into complementary interacting parts, the system and its environment, represented by categories and saltatories.

Fourth, structural coupling is the chiasmic interplay between the environment of one system and the system of the other system, and vice versa. That is, the saltatories of the first system are accepted as the environment of the second system by the categories of the second system.

Fifths, this mechanism is conceptualized in diamond semiotics as the interactionality of bi-signs of textemes.

Structure of structural coupling	
$\left(\text{MG} \xrightarrow{s_i} \text{MG} \right) \diamond \left(\text{MG} \xrightarrow{s_j} \text{MG} \right)$	: autopoietic system AUT1
$\left(\text{MG} \xrightarrow{s_i} \text{MG} \right) \circ \left(\text{MG} \xrightarrow{s_j} \text{MG} \right) \left \bar{\text{M}}\bar{\text{G}} \xleftarrow{\bar{h}} \bar{\text{M}}\bar{\text{G}} \right.$	: system environment of AUT1
X	: metamorphic chiasm
$\bar{\text{M}}\bar{\text{G}} \xleftarrow{\bar{h}} \bar{\text{M}}\bar{\text{G}} \left \left(\text{MG} \xrightarrow{s_i} \text{MG} \right) \circ \left(\text{MG} \xrightarrow{s_j} \text{MG} \right) \right.$	: environment system of AUT2
$\left(\text{MG} \xrightarrow{s_i} \text{MG} \right) \diamond \left(\text{MG} \xrightarrow{s_j} \text{MG} \right)$	: autopoietic system AUT2

As just explained, the environment of a system might be considered as an externalization of the *matching* conditions of the operators of the system into the system itself. These matching conditions are complementarily mirrored in the system of *saltatories*, i.e. the environment of the categorical compositions of the system. Between categories and saltatories of diamonds there exists a complementarity. Hence, a system might be thematized from the point of view of its saltatories or complementarily from the point of view of its categories.

This holds at once for both systems involved in the interaction of structural coupling. The mechanism of interaction between both systems is ruled by the operation of a *metamorphic chiasm*. What was perturbing anonymously the system as an irritation from the environment becomes now an interaction between two autopoietic systems.

Therefore, the structural coupling happens between the environments of the autopoietic systems AUT.

The complexity and complication of the possible coupling is defined by the interacting autopoietic systems mediated by their environments under the rules of a metamorphic chiasm. Anything else, like information isn't in the game yet.

A structural change of the saltatorial environment of a systems is enabling or forcing a change of its categorical systems insofar as the matching conditions of the categorical system which are out-sourced to its saltatorial environment have been changed. This change is realized in the system. Hence the system is answering to the perturbation by the environment, this might be connected with different types of structural learning.

Obviously, and again, the environments of a diamond system are its own environments, and that are its possibilities of existence, formally realized by the matching conditions of the morphisms of the system. At this stage, there are no informational, physical or energetic environments in play.

The conditions of the existence of systems are the environment of the systems.

Diamond systems theory is 'in-sourcing' those out-sourced conditions as the immanent environment of the system itself. The 'out-sourced' conditions are the externally set conditions that preface and pre-define a system axiomatically.

Therefore, the maintenance of a system is self-maintenance, i.e. the maintenance of the maintenance of the system. The function of maintenance of a system is maintaining the maintenance of the system.

With that, the whole aspect of *circularity* is at work. But instead of a classical recursive modeling as it is established for autopoietic systems, the constellation of the interplay is modeled by the diamond approach as a *closed chiasm* of inside/outside and system/environment.

The ‘in-sourcing’ of ‘matching conditions’ has many faces. It might happen *intra*-contexturally for compositions of any kind and it might happen *trans*-contexturally as the interplay of discontextual systems, machines, or structurations.

Any composed action occurs within the diamond approach with its own intrinsic environment.

<http://www.thinkartlab.com/pkl/lola/Diamond-Category-Theory.pdf>

http://www.thinkartlab.com/pkl/media/Generalized_Diamonds/Generalized_Diamonds.html

Contrasts in different directions:

I. C. Baianu and J. F. Glazebrook

CATEGORICAL ONTOLOGY OF COMPLEX SYSTEMS, META-SYSTEMS AND LEVELS: THE EMERGENCE OF LIFE, HUMAN CONSCIOUSNESS AND SOCIETY, 2010

<http://cogprints.org/7754/8/BRAINpp89ICBjfg242.pdf>

“The point is being often made that it is not the component atoms that are preserved in organisms (and indeed in ‘living fossils’ for geological periods of time), but the structure-function relational pattern, or indeed the associated organismic categories/supercategories. This is a very important point: only the functional organismic structure or pattern persists as it is being conserved and transmitted from one generation to the next. Biomolecules turn-over in an organism, and not infrequently, but the structure-function patterns/organismic categories remain unchanged /are conserved over long periods of time through repeated repairs and replacements of the molecular parts that need repairing, as long as the organism lives. Such stable patterns of relations are, at least in principle, amenable to logical and mathematical representation without tearing apart the living system.

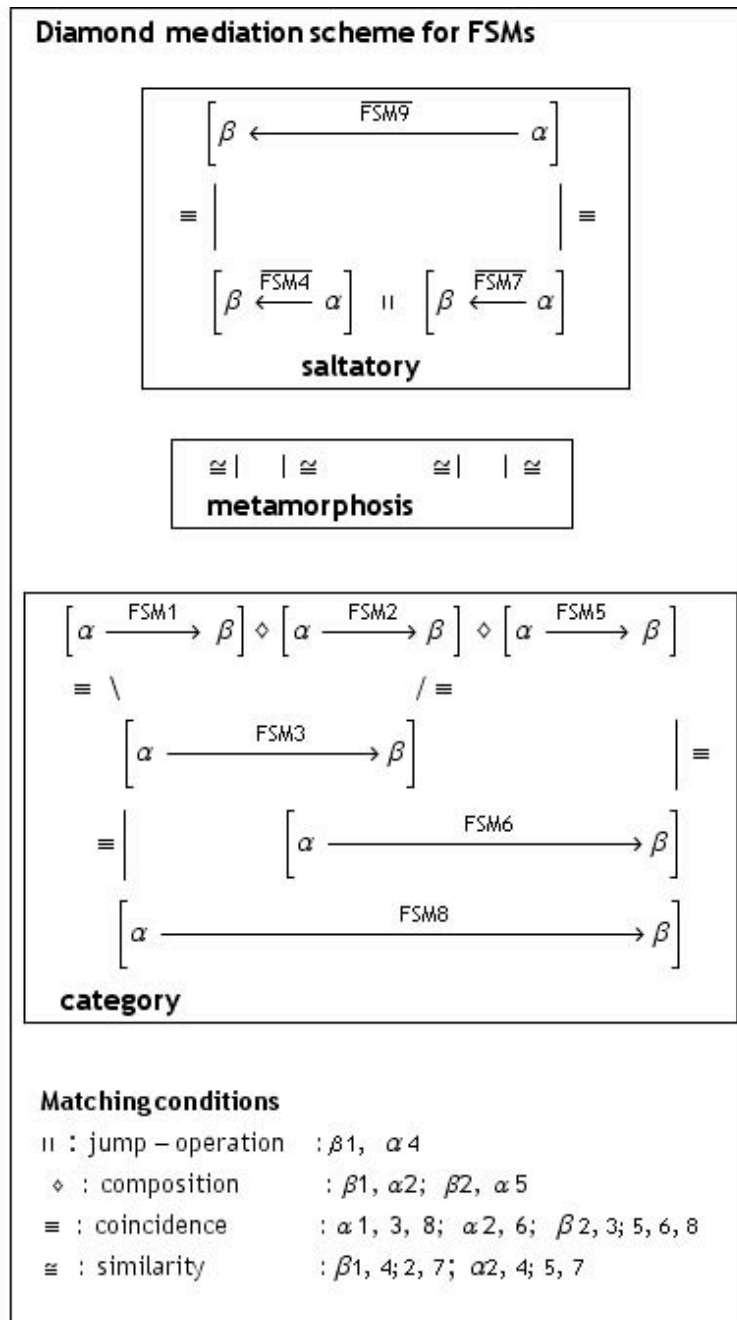
“To sum it up, the operating/operational logics at both the top and the fundamental levels are non-commutative (the ‘invisible’ actor(s) who- behind the visible scene- make(s) both the action and play possible!).” (Baianu, 2010, p. 164)

3.2. How to apply this mechanism to morphic machines?

3.2.1. Introductory sketch

Diamond category theory was introduced 2007 as the insight into the *in-sourcing* of the matching conditions of compositions of morphisms, functors and categories into the very definition of compositions, functorial mappings and natural transformations of categories leading to the *complementarity* of categories and saltatories (saltitions, jumpoids) as a metamorphic interplay of categorical and saltatorial structurations.

An interplay between a category and its saltatory, or a saltatory with its category, is depicted by the following diagram:



There are two different types of complex diamonds to consider: the *transitive* and the *intransitive* modeling of saltitions.

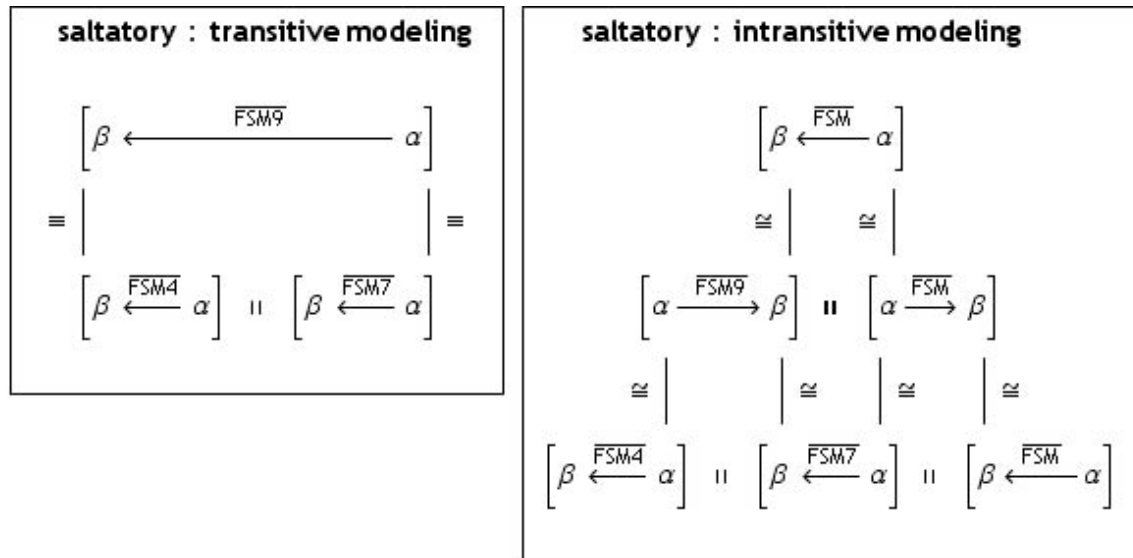
Transitive saltitions are bridging the gap between saltitions by a *pseudo-commutative closure* of the basic saltitions with the help of *coincidence* relations between the domains and codomains of the involved saltitions (hetero-morphisms). This approach conserves the aspects of commutativity and transitivity of categories in the real of saltatories. The orientation of the hetero-morphisms is constantly in one direction.

Intransitive saltitions are bridging the gap by opening up new gaps. With that, the characteristics of saltatories as *jumpoids* becomes more prominent. Hence, the conservation of categorical properties is abandoned by the intransitive version of

saltatories. What counts with this approach are saltitions and their jumps (salti) only.

The 'direction' of intransitive saltitions gets changed from level to level of heteromorphic compositions. This also hints to the fact that saltitions are not just inverse morphisms.

It seems that the complementarity property of saltitions in respect to categories gets some more concretization by the throughout domination of jumps (||) in saltatories.



The principle *andromicity* of categories and saltatories is therefore distinguished as a mono- and a cascading directionality, corresponding to the single direction for transitive and the double directions for intransitive saltatories.

3.2.2. Intra-contextural interplay of morphic machines

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

A morphism is defined by the transition function $\delta(r_i, \omega_{i+1})$:

$q_i \in Q, \Sigma = \{\text{signs}\} \quad \text{pos}_i \in Q', \Sigma = \{e, v\}$

$\text{FSM } M = (Q, \Sigma, \delta, s_0, F) \quad \overline{\text{FSM}} M = (Q, \Sigma, \delta', s_0, F)$

- | | |
|---|--|
| 1. $r_0 = q_0$, | 1'. $\omega_0 = q_0$, |
| 2. $\delta(r_i, \omega_{i+1}) = r_{i+1}$ for $0 \leq i < n$ | 2'. $\delta'(\omega_i, r_{i+1}) = \omega_{i+1}$ for $0 \leq i < n$ |
| 3. $r_n \in F$ | 3'. $\omega_n \in F$ |

Also written as $r_0 \xrightarrow{a_1 a_2 \dots a_n} r_n \mid r'_n \xleftarrow{a_1 a_2 \dots a_n} r'_0$