Rudolf Kaehr
(1942-2016)

Title

Some Formal Aspects of Morphic Palindromes
Different Use and Characterizations of Palindromicity

Archive-Number / Categories

3_27 / K09, K08, K10, K12

Publication Date

2013

Keywords / Topics

- FORMAL ASPECTS OF PALINDROMES : Mathematical Concepts of Palindromes and DNA, Assembly Words - The Florida approach,
- MORPHOGRAMMATIC APPROACH TO PALINDROMES : DOWs and palindromes, 1.2.12. Constructing palindromes, 1.2.13. What are the advantages of the morphogrammatic understanding of palindromes?
- TOWARDS AN ALGORITHM FOR THE PRODUCTION OF MORPHIC PALINDROMES : Grammar for morphic palindromes, Programming aspects

Disciplines

Artificial Intelligence and Robotics – Cybernetics – Semiotics – Linguistics – Epistemology – Theory of Science - Other Languages, Societies, and Cultures

Abstract

Different notions of palindromy, palindromicity and palindromic words are of importance in the scientific literature. A short comparison of the semiotic, i.e. identity based concept as it occurs in linguistic, numeric and music compositions, the complementarity of the DNA-palindrome formulation (Kari), continued by the DOW approach with canonical representation (Jonoška), and the general morphogrammatic contribution, are analyzed in this paper "Formal Aspects of morphic Palindromes". The aim of this paper is not to elaborate a full-fledged mathematical theory but to collect some approaches and to give some further hints to what could and should be done to achieve it. A very first result is given here by a morpho-grammar for morphic palindromes.

Some Formal Aspects of morphic Palindromes

Different use and characterizations of Palindromicity

Rudolf Kaehr Dr.phil@

Abstract

Different notions of palindromy, palindromicity and palindromic words are of importance in the scientific literature. A short comparison of the *semiotic*, i.e. identity based concept as it occurs in linguistic, numeric and music compositions, the complementarity of the *DNA-palindrome* formulation (Kari), continued by the *DOW* approach with *canonical* representation (Jonoška), and the general *morphogrammatic* contribution, are analyzed in this paper *"Formal Aspects of morphic Palindromes"*. The aim of this paper is not to elaborate a full-fledged mathematical theory but to collect some approaches and to give some further hints to what could and should be done to achieve it. A very first result is given here by a morpho-grammar for morphic palindromes.
(Work in progress, vers. 0.4, July 2013)

1. Formal Aspects of Palindromes

1.1. General Concepts

1.1.1. Mathematical Concepts of Palindromes and DNA

'Given a finite word w = x1 x2 · · · xm (where each xi is a letter), the length of w, denoted by |w|, is equal to m. We denote by ~ w the reversal of w, given by ~ w = xm · · · x2 x1 (the "*mirror image*" of w). If w = ~ w, then w is called a *palindrome*. By convention, the empty word ε is assumed to be a palindrome.

'A finite word z is a factor of a finite or infinite word w if w = uzv for some words u, v. In the special case u = ε (resp. v = ε), we call z a prefix (resp. suffix) of w. If u != ε and v != ε, then we say that z is an interior factor of w = uzv. A proper factor (resp. proper prefix, proper suffix) of a word w is a factor (resp. prefix, suffix) of w that is shorter than w.''
http://arxiv.org/pdf/0807.2303.pdf

"Obviously, the palindromic language is closed under reversal, since Pal(L) = Pal($\tilde{L}$)."

Θ-palindromes
"Generalizations of *rich* words appeared soon. Instead of classical palindromes defined as words invariant under the reversal mapping one can consider *Θ-palindromes*, i.e., words invariant under an *involutive antimorphism* Θ.

"2.1 Antimorphisms and their fixed points

A mapping φ on A^* is called

- a *morphism* if φ(vw) = φ(v)φ(w) for any v, w ∈ A^* ;
- an *antimorphism* if φ(vw) = φ(w)φ(v) for any v, w ∈ A^* .

"We denote the set of all morphisms and *antimorphisms* on A^* by *AM* (A^*). Together with *composition*, it forms a *monoid* with the identity mapping *Id* as the unit element.

The set of all morphisms, denoted by *M*(A^*), is a submonoid of *AM*(A^*).

"The *reversal* mapping R defined by

$$R(w1\ w2 \cdots wn) = wn\ wn{-}1 \cdots w2\ w1 \text{ for all } w = w1 \cdots wn \in A^*$$
is an involutive antimorphism, i.e., R2 = Id. It is obvious that any antimorphism is a composition of R and a morphism. Thus

$$AM (A^*) = M (A^*) \cup R(M (A^*)) .$$

"A fixed point of a given antimorphism Θ is called Θ-palindrome, i.e., a word w is a Θ-palindrome if w = Θ(w). If Θ is the reversal mapping R, we say palindrome or classical palindrome instead of R-palindrome. One can see that if Θ has a fixed point containing all the letters of A, then Θ is an involution, and thus a composition of R and an involutive permutation of letters."

Edita Pelantová, Štěpán Starosta, *Palindromic richness for languages invariant under more symmetries*.
http://arxiv.org/pdf/1108.3042.pdf (21. Febr 2013)

Summary
Depending on the alphabet and the mapping function, different types of palindromes are defined.
An alphabet might be *finite* or *infinite*.
For a numeric alphabet, *numerical* palindromes are defined.
For a linguistic alphabet, *linguistic* palindromes are defined. And so on.
Mostly, numeric and linguistic palindromes are defined in the literature by R=1.
Depending on the morphism and anti-morphisms R, classical palindromes are defined for R=1=id. With R as a relabeling function (morphism), generalized palindromes in the sense of N. Jonoška are defined.

Morphogrammatic palindromes are not based on a pro-given alphabet, they evoke their alphabet of differentiations in the process of inscribing them. Given the system of such an inscription, different and more classical formalizations are accessible.

http://www.liafa.univ-paris-diderot.fr/~labbe/Publications/2011-codings.pdf

httpp://www.i.kyushu-u.ac.jp/~te104068/papers/CPM2011.pdf

http://www.deepdyve.com/lp/springer-journals/watson-crick-palindromes-in-dna-computing-UPvR0lM2I3/5

Kari
http://users.ics.aalto.fi/sseki/files/dnafoundall.pdf

Following Szilárd Zsolt Fazekas' et al concise paper, some important definitions are cited:

Palindromicity
"The central concept to the work presented here is *palindromicity*. First off, for a word *w* by w^R we denote its *reversal*, that is *w*[|w| . . . 1]. If $w = w^R$, the word is called a palindrome; for words of *even* length we have $w = uu^R$, while for *odd* length we have $w = uau^R$ with *u* a word and *a* some letter at their centre.
The set of all palindromes of a language L is denoted by Pal(L) = Pal ∩ *L*. If Pal(*L*) = *L*, the language *L* is called a *palindromic*.

"Theorem 1. A *regular* language L ⊆ Σ* is palindromic, if and only if it is a union of finitely many languages of the form $L_p = \{p\}$ or $L_{r,s,q} = qr(sr)^* q^R$ where p, r and s are palindromes, and q is an arbitrary word.

"We note here that the location of Pal - the language of all palindromes - in the Chomsky Hierarchy is well-known; it is *linear context-free*. Another fact worth noting is that the primitive root of every palindrome is again a palindrome.

"Trivially, every palindrome $p = aqa$, with q a (possibly empty) palindrome, has palindromic prefixes λ, a and aqa, therefore whenever we say a palindrome has a non-trivial palindromic prefix (suffix), we mean that it has a proper prefix (suffix) of length at least two which is a palindrome."
http://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/1809-15.pdf

Leopold about complementarity and multiple heads

"As Kuske and Weigel [6] observed, the *complementarity* does not really play any significant role in the operation of conventional Watson-Crick automata. All language classes of WK-automata are the same, even when the complementarity relation can only be the identity relation.

"Their argumentation will carry over to all types of automata defined here. Thus this simplification to plain strings does not really take us farther away from the motivation for these automata, but makes notations and definitions much less complicated.

"The idea of letting the two heads run in opposite ways was already mentioned by P ˘ aun et. al in the book on DNA Computing [11] and also in a later article [10], where these devices were called reverse WK-automata. However, their power was not investigated in more detail.

"Finally, let us note that conventional automata with several heads have been studied before extensively, one of the earliest works is by Ibarra [5] and research on the topic is still going on. However, it seems that no model equivalent to the one defined in this article has been defined, i.e., automata where the two heads run in oppositedirections and can turn only after reading the entire word.

Finally, it should be mentioned that the heads in our model do not sense each other. This means they cannot detect the step in which they are at the same position.

http://wortspieler.org/Dateien/Work/Pubs/NagyLeupold_WCAutomata.pdf

"What if finite automata worked on DNA strands instead of abstract strings of symbols?
A DNA strand is not just a simple string, but normally is a double strand with a *three-dimensional helix* structure.

"A run is a complete reading of the input string by both heads in their respective direction.
• There are start and end markers on the input string.
• After a run the heads turn *around* and can read the respectively other side of the strand - which is the same string in our case.
An input word is accepted in k runs, iff the automaton halts in an accepting state after k runs."

http://www.informatik.uni-giessen.de/ncma2009/files/Leupold.pdf

http://www.pps.univ-paris-diderot.fr/~mellies/tensorial-logic/6-braided-notions-of-dialogue-categories.pdf

1.1.2. Assembly Words - The Florida approach

Single occurrence trito-word: [1,2,3,4] is a trito-palindrome .

Partial completion examples: [1,2,3,4] to [1,2,3,4,2,1], [1,2,3,4,3,2,1] and .

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

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

Not all assembly words of 3 letters are trito-palindromic

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

Trito-Palindromic:

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

DOW-palindromic, includes ascending order ‘abstraction'

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

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

The ML-operation *tnf* in *kref = tnf(rev(w)),* corresponds *ReLabel* for positive integers only.
The decision for ‘ascending order’ by ‘relabeling’ in the DOW-approach has no *conceptual justification* except of its technical advantage for practical applications. DNA palindromes are not *per se* canonically ordered.

Assembly words and palindromes

"We consider an assembly word to be a permutation of {1,1,2,2,...,n,n} such that when the ith term appears for the first time in the word, it is preceeded by 1,2,...,i-1.
Ex: 112332 is an assembly word
113232 is not an assemby word (first 3 not preceeded by 2)
Assembly words which are invariant under *orientation reversal* are called *palindromes.*
Ex: 12341243 is the orientation reversal of 12342134
(reverse word and map 4->1, 3->2, 1->3, 2->4)
Ex: 121323 is a *palindrome.*” (J. Burns)

http://shell.cas.usf.edu/~saito/DNAweb/SimpleAssemblyTable.txt

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

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

- **ENstructure** [1,2,3,4,1,2,4,3] = ENstructure[1,2,3,4,2,1,3,4];
val it = false : bool

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

WEB tools of the Florida group

http://shell.cas.usf.edu/~jonoska/index.html
http://knot.math.usf.edu/data/index.html
http://knot.math.usf.edu/assembly/
http://knot.math.usf.edu/assembly/properties.html
http://knot.math.usf.edu/data/search.html
http://shell.cas.usf.edu/~saito/DNAweb/SimpleAssemblyTable.txt
http://arxiv.org/pdf/1105.2926v1.pdf

DOW-Palindromes
Search for
\`size\` = '3' AND \`pal\` = 'Yes' AND \`assembly\` = '1'
returned 7 results:
112233 122331 121323 123123 **123231** 123312 123321

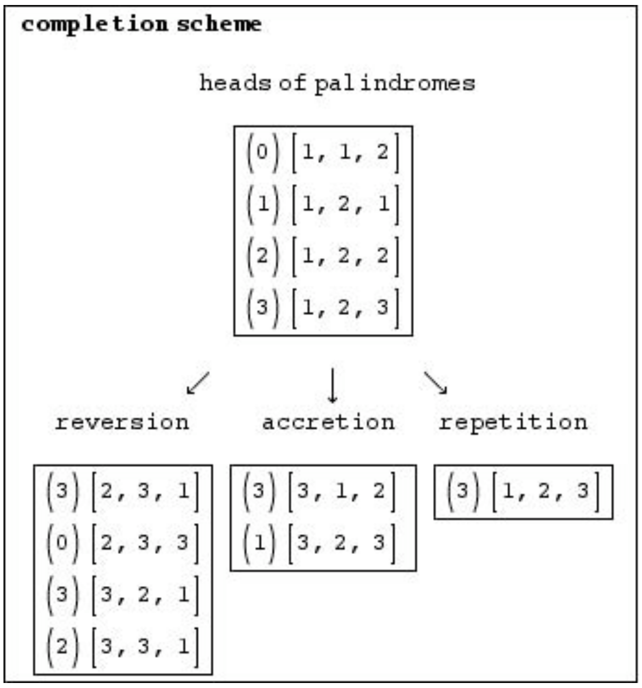
http://knot.math.usf.edu/data/search.html

Mathematica Code for Assambly Words:
AssemblyWords[1]={{1, 1}};
AssemblyWords[n_]:=Flatten[Map[Table[Join[{1},Insert[#,1,i]],{i,2n-1}]&,AssemblyWords[n-1]+1],1]
ReLabel[L_List]:=L/.Map[#[[1]]->#[[2]]&,Transpose[{DeleteDuplicates[L],Range[Length[Union[L]]]}]]
RevWord[L_List]:=ReLabel[Reverse[L]]

http://jtburns.myweb.usf.edu/index.html?http%3A//jtburns.myweb.usf.edu/assembly/mathematica.htm

AssemblyWords[3]
[1]:= AssemblyWords[3]
Out[1]:=
[[1,1,2,2,3,3],**[1,2,1,2,3,3]**,{**1,2,2,1,3,3**},{**1,2,2,3,1,3**},
{1,2,2,3,3,1},{**1,1,2,3,2,3**},{1,2,1,3,2,3},{1,2,3,1,2,3},
{**1,2,3,2,1,3**},{1,2,3,2,3,1},{**1,1,2,3,3,2**},{**1,2,1,3,3,2**},
{**1,2,3,1,3,2**},{1,2,3,3,1,2},{1,2,3,3,2,1}}

Palindromes with ‘ascending order’ operation (tnf)
112233 122331 121323 123123 123231 123312 123321



The bold **AssemblyWords** are not trito-palindromic.

List.filter ispalindrome “Tcontexture 6”;
[[1,1,1,1,1,1],*[1,1,1,2,2,2]*,*[1,1,2,1,2,2]*,*[1,2,1,2,1,2]*,*[1,2,2,1,1,2]*,
[1,1,2,2,1,1],*[1,2,1,1,2,1]*,[1,2,2,2,2,1],**[1,1,2,2,3,3]**,**[1,2,1,3,2,3]**,
[1,2,2,3,3,1],**[1,2,3,1,2,3]**,**[1,2,3,2,3,1]**,**[1,2,3,3,1,2]**,**[1,2,3,3,2,1]**,
[1,1,2,3,1,1],[1,2,1,1,3,1],[1,2,2,2,2,3],[1,1,2,3,4,4],[1,2,3,4,1,2],
[1,2,3,4,2,1],[1,2,1,3,4,3],[1,2,3,1,4,3],[1,2,3,3,4,1],[1,2,2,3,3,4],
[1,2,3,2,3,4],[1,2,3,3,2,4],[1,2,3,4,5,1],[1,2,3,4,2,5],[1,2,3,3,4,5],
[1,2,3,4,5,6]] : int list list

Classification of trito-palindromes in respect of DOW properties

Bold listed palindromes of *Tcontexture 6* are DOWs. Number is seven.
Emphazised listed palindrome are triple OWs, TOWs.
[1,1,2,2,1,1] is 1-2-DOW

DOW-word = (123435654126)
Trito-palindrome:
- ispalindrome[1,2,3,4,3,5,6,5,4,1,2,6];
val it = false : bool
isDOW: true,
DOW-palindromic: false,
(123435654126) != [123451564623]
Dow-word != ReLabel(DOW-word), reverse

Example
Trito-palindrome: yes
- ispalindrome[1,2,1];
val it = true : bool
DOW : false

palin[3] := AssemblyWords[3] = RevWord[AssemblyWords[3]]

DNA-Palindromic

"The meaning of “palindromic” in this context is different from what one might expect from its *linguistic* usage: 50-GTAATG-30 is not a palindromic DNA sequence, but 50-GTATAC-30 is (50-GTATAC-30 is WK complementary to 30-CATATG-50, which is the same as 50-GTATAC-30).

"It is exactly this *biological* meaning of the word “palindrome” that we attempt to model here, by the notion of **Watson-Crick palindrome.**
Using our formalization and convention on strand directionality, if WK denotes the Watson-Crick antimorphic involution, WK(GTATAC) = GTATAC.

"Thus, the study of h-palindromes for antimorphic involutions is interesting from two points of view: firstly, it may be desirable for certain DNA computing experiments to use DNA strands that contain h-palindromic enzyme restriction sites as subwords, and secondly, in general, a set of DNA codewords should be free of h-palindromic words, due to the intermolecular hybridizations that these would entail.” Kari

DOW-Palindrome: Double occurrence words DOW with ascending order

The common definition for palindromes as *“Palindromes are words that read from either the left or the right end are the same.”* gets a generalization by the operation of an “*ascending order*” over the words by a bijection, i.e. a relabeling, between the word and its reversal. With that a kind of “asymmetric” palindromes are accepted as palindromes.

"Definition 2.1 A *double occurrence* word is palindromic (or symmetric) if it is equivalent to its reverse. A double occurrence word that is palindromic is called a *palindrome*.

"1.4 Assembly words
Let A be an alphabet set. A double occurrence word (DOW) over A is a word which contains each symbol of A exactly 0 or 2 times. Words w1 over A1 and w2 over A2 are equivalent if there is a bijection between A1 and A2 under which w1 maps to w2. If w = a1 . . . ak is a word over A, its *reverse* w^R is defined by ak . . . a1. Two words w1, w2 are *reverse equivalent* if w1 is equivalent to w2 or $w^R 2$.

"An *assembly* word is a double occurrence word with reverse equivalence classes." (Jamies_thesis)

Ascending order
"Double occurrence words labeled by this convention are said to be in *ascending order*. Two double occurrence words are said to be equivalent if they are equal after being relabeled in ascending order. If two double occurrence words are not equivalent, they are said to be distinct.

"Definition 2.1 A double occurrence word is *palindromic* (or symmetric) if it is equivalent to its reverse. A double occurrence word that is palindromic is called a *palindrome*.
(we shall assume that all double occurrence words are in ascending order)."

Reverse DOW-word plus canonical form (= ascending order):
palindrome: canonical(reverse(word)) = word

Example
word = (122331)
rev(word) = (133221)
word $!=_{sem}$rev(word)
canonical(rev(word)) = (122331),
hence, word is palindrome.

The canonical form is similar to *tnf* for morphograms.

The tnf-operation in morphogrammatics is motivated and justified by the EN-structure of morphograms.

Relabeling
It seems that the operation of *canonical* representation by ascending order has no systematic justification by the DOW approach, and seems to be based on purely practical or technical motivations. It promotes an interesting decision but is not necessarily in tune with the other more classical approaches.

"For *convenience*, we let $\Sigma = \{1, 2,..., n\}$ and *relabel* each double occurrence word such that when *i* appears for the first time in the word, it is preceded by 1, 2, ..., i-1. *Double occurrence* words labeled by this convention are said to be in *ascending order*. Two double occurrence words are said to be *equivalent* if they are equal after being *relabeled* in ascending order." (Burns, Muche, p. 3, 2011)

"Let A be an alphabet set. A *double occurrence word* (DOW) over A is a word which contains each symbol of A exactly 0 or 2 times. Words w_1 over A_1 and w_2 over A_2 are *equivalent* if there is a bijection between A_1 and A_2 under which w_1 maps to w_2. If w = $a_1 . . . a_k$ is a word over A, its *reverse* w^R is defined by $a_k . . . a_1$. Two words w_1, w_2 are *reverse equivalent* if w_1 is equivalent to w_2 or w_2^R. An *assembly* word is a double occurrence word with reverse equivalence classes." (Jamie, p.3, 2013)
We say that two assembly words are *equivalent* if they are the same or if they are reverses.

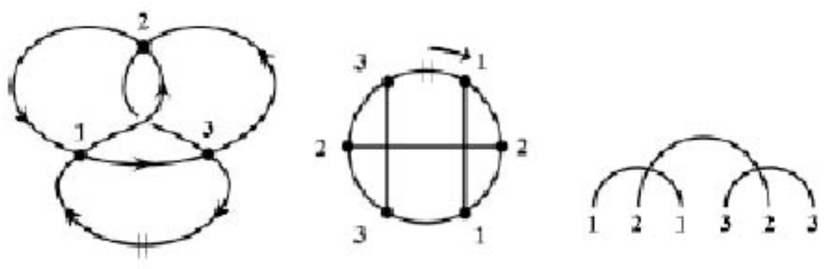


Figure 1: Self-intersecting closed curve (left), chord diagram (center), and linked diagram (right) representations of the double occurrence word 121323. Base points, indicating the starting point for reading the word, are marked by ||.

"An assembly word is a palindrome if it is equal to its reverse written in ascending order. The graph w is said to be palindromic if w is a palindrome." (Saito, 2012) http://math.usf.edu/~saito/DNAweb

Cyclic Permutations and deuterograms
Cyclic Permutations of 123321:
123321, 122133, 112332

```
- Dcontexture 6;
val it =
 [[1,1,1,1,1,1],[1,1,1,2,2,2],[1,1,1,1,2,2],[1,1,1,1,1,2],[1,1,2,2,3,3],
  [1,1,1,2,2,3],[1,1,1,1,2,3],[1,1,2,2,3,4],[1,1,1,2,3,4],[1,1,2,3,4,5],
  [1,2,3,4,5,6]] : int list list
```

The deuterogram **[1,1,2,2,3,3]** has trito-representations.

1.1.3. Semiotic Palindromes: linguistic, numeric, pictographic, sonic

"For Eliot *circularity* involves progression; it entails a quest that may carry the protagonist back to where he began, but that produces added acumen or insight.
Retraction is not the allure. Nor is it the attraction for Jerry in Albee's *The Zoo Story* who utters, *"Sometimes it's necessary to go a long distance out of the way in order to come back a short distance correctly."* Again in this case return to the beginning implies a new beginning." (Stan Fogel, Palinodes and Palindromes, 1984)
journals.hil.unb.ca/index.php/IFR/article/download/13657/14740

1.2. Morphogrammatic approach to palindromes

1.2.1. Morphogrammatic characterization of palindromes

Morphograms are not considered as sign sequences but as complexions of differentiations or differences. Therefore the concept of "relabeling' reverted morphograms is superfluous and covert at the very beginning of the definition of morphograms and palindromes by their ENstructure. The ENstructure is defining a morphogram by indicating its structure of differentiation.

In this sense, morphic palindromes that are semiotically asymmetric, are genuine. A relabeling in ascending order of the asymmetric semiotic palindromes by the DOW approach is then introducing an additional operation (abstraction) on semiotic palindromes.

The ENstructure of a morphogram is independet of any relabeling operation because a morphogram is not definied by its atomic signs but by the differentiation structure between signs, in fact by its kenograms.

In other words, the semiotic representation of a morphogram is by definition in “ascending order” and represented by its trito-normal form, tnf.

But that doesen’t exclude the fact that equivalent morphograms may occur at different locations in different realizations as repetitions.

With the additional bijective relabeling function to produce an ascending order on reversed palindromic words, the important property of a genuine *asymmetry* and its consequences gets eroded.

In the case of palindromes, the operation on a word, say reversion, is producing the reverted word. For non-symmetric palindromes, this reversion is producing asymmetric words that are not palindromes in the classical definition of palindromicity.

With the help of a bijection that is producing an ascending order on the reverted word, palindrome, the word is accepted as a palindrome.

The understanding of palindromes, especially for DOWs, in the sense of the Florida group, there are two operations involved to produce palindromes, the first are are reversion and repetition, the second, the relabeling operation. The operation of relabeling aims to give a *canonical* notation to the reversed word, so it can be read as a ‘symmetrical’ palindrome.

This strategy makes a difference to the common definitions of palindromes.
It is necessary a necessary technical step for words defined on an atomistic alphabet.

This delaying operation of relabeling is superfluous in the context of morphogrammatics.

Morphograms are defined by their differentiation and not by their signs.

This is not excluding the possibility to inscribe morphograms in a canonical notation as sign sequences. The trito-normal form operation, tnf, is just delivering that. Morphograms as sign sequences in ascending (lexical) order are useful for practical elaborations but are also disabeling importand features.

Morphograms are not special abstractions from sign systems but are based on the level of inscription defined by the Stirling approach of E/N-differences.

$$\begin{tikzcd}
\text{Morphogram}(mg) \arrow[r] \arrow[dr] & E/N-\text{structure}(mg) \arrow[d] \\
 & \text{tnf}(mg)
\end{tikzcd}$$

For technical reasons and for practical convenience, this is the approach practiced in this and other papers about morphogrammatics.

$$\begin{tikzcd}
\text{Word}(w) \arrow[r, "\text{operations}"] & \text{op}(w) \arrow[r, "\text{relabeling}"] & \text{properties}(\text{op}(w)) \\
\text{Morpogram}(mg) \arrow[r, "\text{operations}"] & \text{properties}(\text{op}(mg))
\end{tikzcd}$$

$$\begin{tikzcd}
\text{Word}(w) \arrow[rr, Rightarrow] \arrow[dr, "\text{ops}"'] & & \text{palin}(w) \\
& (\breve{w}) \arrow[ur, "\text{relabel}"'] &
\end{tikzcd}$$

$\text{ops} = \{\text{reversion, repetition}\}$

$$\text{morphogram}(mg) \xRightarrow{\text{operations}} \text{palin}(mg)$$

$\text{operations} = \{\text{reversion, repetition, accretion}\}$

Example

Words:

$w = (1122) \xrightarrow{\text{rev}} \text{op}(w) = (2211) \xrightarrow{\text{relabel}} \text{prop}(\text{op}(w)) = (1122) \in \text{Pal}$

Morphograms:

$mg = [1122] \xrightarrow{\text{reversion}} \text{prop}(\text{op}(w)) = [1122] \in \text{Pal}$

$\text{words} = \{\text{assembly words, non} - \text{assembly words})$

$$\begin{tikzcd}
\text{Word}(w) \arrow[rr, Rightarrow] \arrow[dr, "\text{ops}"'] & & \text{palin}(w\,\breve{w}) \\
& (\breve{w}) \arrow[ur, "\text{relabel}"'] &
\end{tikzcd}$$

$\text{ops} = \{\text{reversion, repetition}\}$

$$\begin{tikzcd}
\text{Word}(w) \arrow[r] & (\breve{w}w) \in \begin{pmatrix}\text{Pal}\\ \text{non} - \text{Pal}\end{pmatrix} \arrow[d, shift right=4] \arrow[d, shift left=4] \\
& \text{Pal}^{+} \quad \text{morphoPal}
\end{tikzcd}$$

$\text{Pal}^{+} = \text{Pal} + \text{relabel}, \ \text{morphoPal} = \text{Pal}^{+} \text{ plus accretion}$

General algorithm

write word w

repeat $w = ww$

reverse repeated $w = w\,\breve{w}$

read reverse $(w\,\breve{w}) = \breve{w}\,w$

relabel $(\breve{w}\,w) = w\,\breve{w}$

test palin $(w\,\breve{w}) :: (w\,\breve{w}) = \text{relabel}(\breve{w}\,w) : (\text{yes}/\text{no})$

yes : palindrome, no : non – palindrome.

$\text{Pal}(w) :: w = \mathbf{R}(w^{R}), \ w = uv, \ \mathbf{R} = \text{relabel}$

From an epistemological point of view it seems that the convenient strategy of relabeling reverted words is adapting the non-ecological approach of buffering and gluing.

The morphogrammatic approach to palindromes enables directly genuine asymmetric palindromes without detour.

This relabeling strategy might turn out to be an obstacle for an adequate and direct biological modeling of DNA processes as it is intended.

Hence, the grammatological status of morphograms should be cleared, again.

1.2.2. Programming aspects

ML-Modules (for concatenational morphogrammatics)

```
type keno = int;
type kseq = keno list;

fun tnf ks =
 let
   fun pos n [] = raise Nth
     |pos 1 (hd::tl) = hd
     |pos n (hd::tl) =pos (n-1) tl;

   fun firstocc item list =
     let
       fun place1 item [] n = raise Place
         |place1 item (x::xs) n = if item=x then n
                                  else place1 item xs n+1;
     in
       place1 item list 1
     end;

fun nfirst n [] =raise Place
     |nfirst 1 (hd::tl)=[hd]
     |nfirst n (hd::tl)=hd::nfirst (n-1) tl;

   fun tnf1 [] res n k = res
     |tnf1 (hd::tl) res 1 k = tnf1 tl [1] 2 2
     |tnf1 (hd::tl) res n k =
       if member (pos n ks) (nfirst (n-1) ks)
       then tnf1 tl
            (res@[pos (firstocc (pos n ks) ks) res])(n+1) k
       else tnf1 tl
            (res@[k]) (n+1) (k+1);
 in
   tnf1 ks [] 1 1
 end;

val tnf = fn : ''a list -> int list
```

```
type **enstruc** = (int*int*EN) list list;
fun **ENstructure** z =
  map (fn trl => map (fn pair => delta pair z)
                 trl)
     (pairstructure (length z));

fun **teq** a b = (ENstructure a) = (ENstructure b);
```

ML-Morphogrammatics, tnf, ks, ENstructure

```
- tnf[3, 2, 4, 1, 5];
val it = [1,2,3,4,5] : int list

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

***Mathematica*, ReLabel**

```
ReLabel[L_List] :=
L /. Map[#[[1]] -> #[[2]] &,
  Transpose[{DeleteDuplicates[L], Range[Length[Union[L]]]}]]

ReLabel[{3, 2, 4, 1, 5}]
{3, 2, 4, 1, 5} /.Transpose[
 DeleteDuplicates[{3, 2, 4, 1, 5}] -> {1, 2, 3, 4, 5}]
```

1.2.3. Gaps in palindromes

(pal, non-pal/pal, pal)

```
- ispalindrome[1,2,2,3,2,2,2,4,5,5,6];
val it = false : bool
```

Hence, (palin1, palin2,palin3)∉Palin

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

Hence, (palin1, palin2,palin3)∉Palin

```
- ispalindrome[1,2,2,3,4,5,5,4,5,5,6];
val it = false : bool
```

Henc, (palin1, non-palin2, palin3)∉Palin

Relabeling
w = [1,2,3,4,5,1,5,6,4,6,2,3]
rev(w) = [6,2,1,4,5,6,5,3,4,3,2,1]

```
- tnf[6,2,1,4,5,6,5,3,4,3,2,1] = [1,2,3,4,5,1,5,6,4,6,2,3];
val it = true : bool
```

ENstructure(tnf(rev(w))) = ENstructure(w)

1.2.4. Completion of palindromes

"We say that whenever a word has a palindrome as *prefix* or *suffix*, then it is extended with the reverse of the rest of the word.

"To illustrate this with an example, after palindromic completion from the word *abbb* we get both *abbba* and *abbbba*, depending on which of the suffixes *bb* or *bbb* we choose for the binding loop.

"Furthermore, we consider *iterated* palindromic completion, the successive application of palindromic completion, taken to the limit. Under these conditions we prove that one can obtain precise characterizations of both words and regular languages whose iterated palindromic completion is *regular*.

"The next immediate result says that, since each palindromic prefix of a word is also a suffix of it, the left and right palindromic completions are symmetric.

"*Lemma 2*. For palindromes, the right palindromic completion equals the left palindromic completion.

"Thus, whenever considering several steps of palindromic completion for some language L, it is enough to consider either the right, or the left, palindromic completion of L1." (Szilárd Zsolt Fazekas et al)

How does that apply to asymmetric palindromes of morphogrammatics?
For asymmetric palindromes the completion hast to involve the structural laws of morhogrammatics. At this point, completion shall be involved with accretive and iterative compositions and concatenations. In this case reduced to the concatenation of the inversion.

The DOW-palindromic completion, in contrast to the familiar completion, is the concatenation of the *reversed* suffix (prefix) in ascending order of the word. As a consequence of the ascending order, i.e the canonical representation of the reversion, the reversion might turn additionally to the reversion into the mode of a repetition or a accretion.

The morphic completion of a morphogram takes the polysemy of the concatenation operation, *kconcat*, i.e. its iterative and accretive mode, directly into account. Therefore the additional modi to *reversion*, the cases of *repetition* and *accretion* of the pre- or suffix, i.e. head or tail of the morphogram, are accepted primordially as parts of the definition.

[abbb] => [abbba], [abbbc]
[abbb] => [abbbba], [abbbbc].

Definition
A morphic assembly word is a palindrome if its suffix is either a *repetition*, *reversion* or *accretion* of its prefix (head). This concept of completion has to take the context rules for morphic palindromes (pal-CR) into account.

1.2.5. Partial, complete and totally complete completion

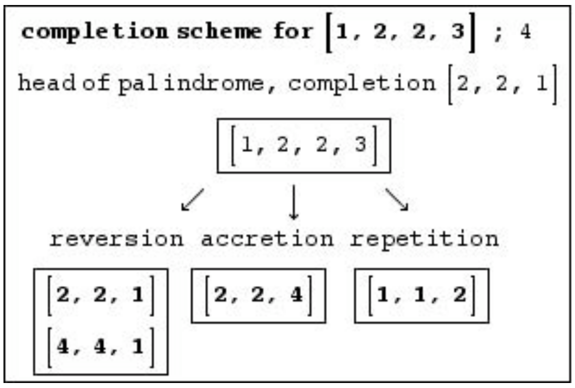
Partial completion of morphic palindromes

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

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

```
- kconcat[1,2,2,3][2,1];
val it =
  [[1,2,2,3,2,1],[1,2,2,3,1,2],[1,2,2,3,3,1],[1,2,2,3,1,3],[1,2,2,3,4,1],
   [1,2,2,3,1,4],[1,2,2,3,3,2],[1,2,2,3,2,3],[1,2,2,3,4,2],[1,2,2,3,2,4],
   [1,2,2,3,4,3],[1,2,2,3,3,4],[1,2,2,3,5,4]] : int list list
- List.filter ispalindrome it;
val it = [[1,2,2,3,3,1],[1,2,2,3,3,4]] : int list list
```

```
- kconcat[1,2,2,3][2,2,1];
val it =
  [[1,2,2,3,2,2,1],[1,2,2,3,1,1,2],[1,2,2,3,3,3,1],[1,2,2,3,1,1,3],
   [1,2,2,3,4,4,1],[1,2,2,3,1,1,4],[1,2,2,3,3,3,2],[1,2,2,3,2,2,3],
   [1,2,2,3,4,4,2],[1,2,2,3,2,2,4],[1,2,2,3,4,4,3],[1,2,2,3,3,3,4],
   [1,2,2,3,5,5,4]] : int list list
- List.filter ispalindrome it;
val it = [[1,2,2,3,2,2,1],[1,2,2,3,1,1,2],[1,2,2,3,4,4,1],[1,2,2,3,2,2,4]]
  : int list list
```


completion scheme for [1, 2, 2, 3] ; 4
head of palindrome, completion [2, 2, 1]
[1, 2, 2, 3]
reversion
accretion
repetition
[2, 2, 1]
[4, 4, 1]
[2, 2, 4]
[1, 1, 2]

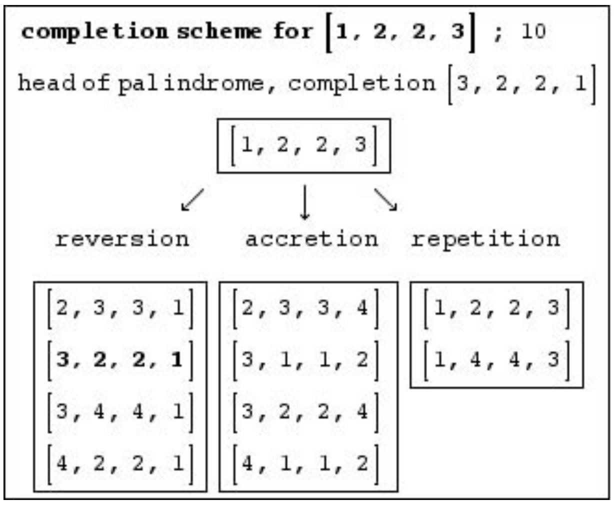
Complete completion

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

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

```
- List.filter ispalindrome it;
val it =
  [[1,2,2,3,3,2,2,1],[1,2,2,3,2,3,3,1],[1,2,2,3,3,1,1,2],[1,2,2,3,1,2,2,3],
   [1,2,2,3,4,2,2,1],[1,2,2,3,4,1,1,2],[1,2,2,3,3,4,4,1],[1,2,2,3,1,4,4,3],
   [1,2,2,3,3,2,2,4],[1,2,2,3,2,3,3,4]] : int list list
- length it;
val it = 10 : int
```

This case might be called *complete* completion.

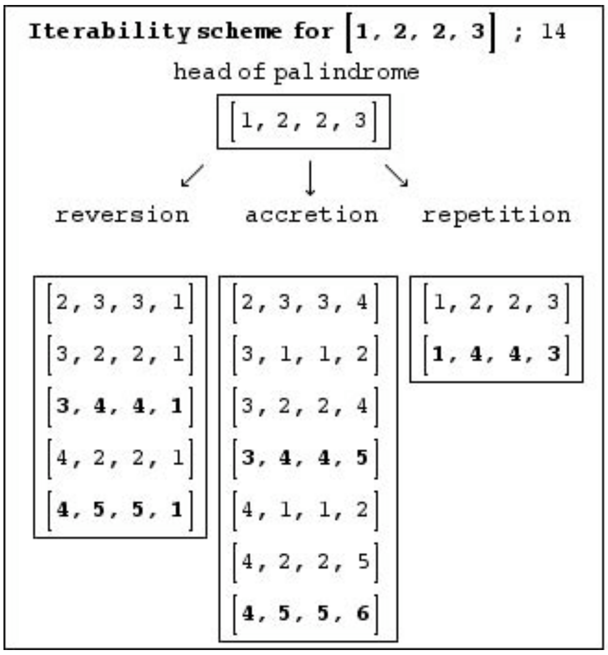


Totally complete completion

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

```
Palindromes:
[[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
- length it;
val it = 14 : int
```

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



|kconcat[1,2,2,3][3,2,2,1]|= |kconcat[1,2,2,3][1,2,2,3]| = 34
|palin(kconcat[1,2,2,3][3,2,2,1])| < |palin(kconcat[1,2,2,3][1,2,2,3])|

```
- length(kconcat[1,2,2,3][1,2,2,3]);
val it = 34 : int
- length(kconcat[1,2,2,3][3,2,2,1]);
val it = 34 : int
```

This modus might be called *totally complete* completion.

The difference between the two forms of completion, the complete and the totally complete, is the result of a difference in the application of the tnf-function for the complete completion, where morphograms of the form [1,2,2,3, 6,5,5,4] are not accepted as palindromes.
While tnf[1,2,2,3,6,5,5,4] = [1,2,2,3,4,5,5,6] is in ascending order and accepted as a morphic palindrome.

```
- tnf[1,2,2,3,6,5,5,4];
val it = [1,2,2,3,4,5,5,6] : int list
- ispalindrome it;
val it = true : bool
```

```
- length(kconcat[1,2,2,3][1,2,2,3]);
val it = 34 : int
- length(kconcat[1,2,2,3][3,2,2,1]);
val it = 34 : int
```

Other completion cases

Morphogram [1,1,2]: prefix is not a palindrome

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

"*Lemma 2*. For palindromes, the right palindromic completion equals the left palindromic completion."

This lemma holds for symmetric palindromes only.

word = abbb
prefix: abbb**ba**
suffix: **ab**bbba

For morphic palindromes we get:

```
kconcat[1,1][1,1];
Palindrome: val it = [[1,1,1,1],[1,1,2,2]] : int list list
- kconcat[1,2][2,1];
- List.filter ispalindrome it;
val it = [[1,2,2,1],[1,2,1,2],[1,2,3,1],[1,2,2,3]] : int list list
```

DOW palindromes L_2 = {**[1,1,2,2]**, **[1,2,2,1]**,**[1,2,1,2]**}

Partial completion examples: [1,2,3,4] to [1,2,3,4,2,1], [1,2,3,4,3,2,1].

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

Non-palindromic base

[1,1,2,3]∉ Pal to [1,1,2,3,1,1]∈Pal:

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

[1,1,2,3] to [3,2,1,1,2,3]

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

1.2.6. Depiction by assembly and differentiation graphs

Assembly graph abstraction

Assembly Graphs are abstracting from the underlying values of the alphabet of the word (palindrome).
Therefore, e.g., AssGraph[1,2,2,1] = AssGraph[3,4,4,3] = AssGraph[4,3,3,4].

That is, AssGraphs as such are depicting some aspects of the ENstruture of morphograms. They correspond to the operation of ascending relabeling. But they are not mirroring the number of differentiations between the elements, which is represented for morphograms by the number of subsystems of the morphogram.

$$\text{length (AssGraph)} = m,$$
$$\text{Number of subsystems of ENstructure} = \binom{m}{2}.$$

- ENstructure[3,4,4,3] = ENstructure[1,2,2,1].

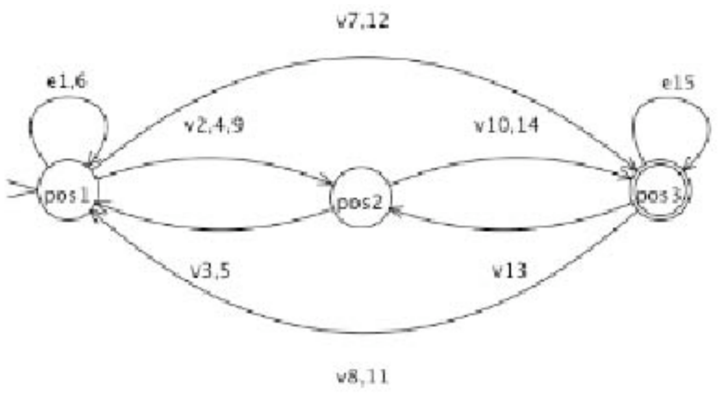
http://knot.math.usf.edu/assembly/index.html

Differentiation graph for the palindrome [1,1,2,2,3,3]

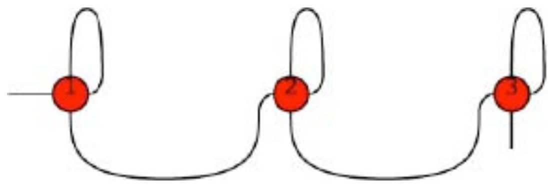
```
- ENstructure [1,1,2,2,3,3];
val it =
[[],
[(1,2,E)],
[(1,3,N),(2,3,N)],
[(1,4,N),(2,4,N),(3,4,E)],
[(1,5,N),(2,5,N),(3,5,N),(4,5,N)],
[(1,6,N),(2,6,N),(3,6,N),(4,6,N),(5,6,E)]] : (int * int * EN) list list
```

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

DiagrMorphFMS [1,1,2,2,3,3]



Assembly Graph for the palindrome (112233)



Comparison

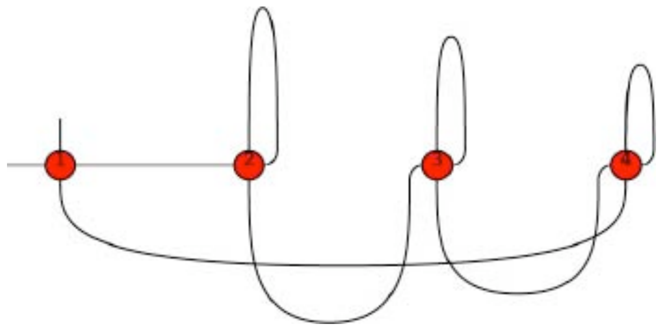
The d*ifferentiation* graph is answering the question: How are the ‘elements” of the palindrome differentiated? What makes the difference between the elements in respect of their property of equal/non-equal and their place (locus) in the grid of the palindrome? Each act of differentiation is creating the ‘elements’ of differentiations.

The *assembly* graph are answering the question: How are the elements of the palindrome connected in the linearly ordered succession structure? The elements of palindromes are pre-given as the elements of the word in question.

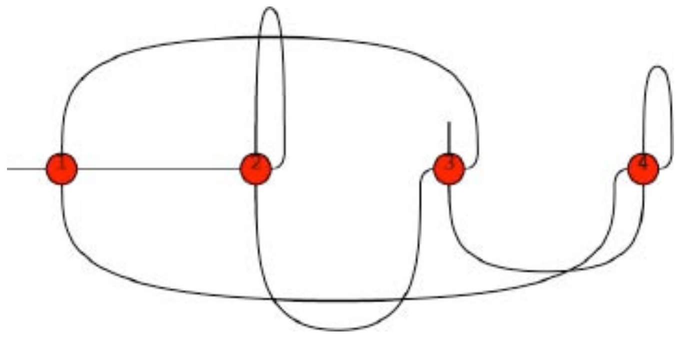
For the complexity of just 3 elements, both approaches, the semiotic and the morphogrammatic, coincide. There are just 3 differentiations for the differentiation graph and just 3 successions in the assembly graph.

Assembly Graphs

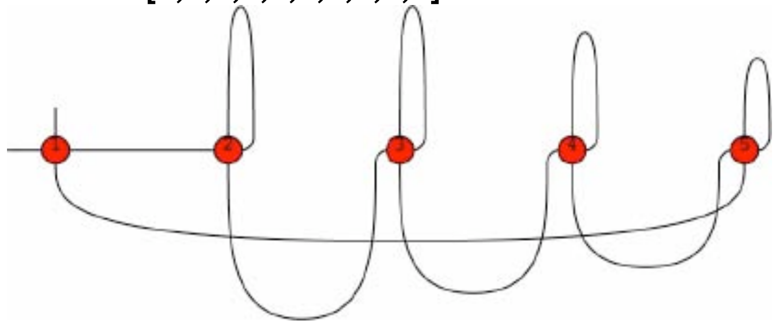
Palindrome P = [1,2,2,3,3,4,4,1]



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



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



1.2.7. Types of Separation: Reducible and irreducible DOWs

"Definition 2.2 If a double occurrence word w can be written as a product w = uv of two non-empty double occurrence words u; v, then w is called *reducible*; otherwise, it is called *irreducible*.

"Definition 2.3 A non-empty double occurrence word is *strongly-irreducible* if it does not contain a proper sub-word that is also a double occurrence word.

"The double occurrence word 12213434 is *reducible* because it can be written as the product of the two double occurrence words 1221 and 3434, but 12344123 is *irreducible*. However, since 44 is a proper sub-word of 12344123 it is not strongly-irreducible. The word 12132434 is *strongly-irreducible*. By definition, strongly-irreducible words are also irreducible, so 12132434 is irreducible as well. In particular 11 is strongly-irreducible.

"Lemma 2.4 Every double occurrence word contains a strongly-irreducible sub-word."

1.2.8. DOWs and palindromes

" [...] separate the classes of double occurrence words into palindromes *and* non-palindromes *and describe the construction of large double occurrence words from smaller double occurrence words."*

Classical palindromes are trivially symmetric (or specially, double) occurrence words.
DOW palindromes with relabeling operations are DOWs.
There are asymmetric palindromes that are not DOWs.

DOWs that are **non-palindromic**.

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

In contrast:

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

non-DOW

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

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

1.2.9. Modi of reducibility for morphic palindromes

A reducible palindromic morphogram is reducible either into *repetive*, *reversive* or *accretive* parts.

The word 12132434 is strongly-irreducible.

Properties
Assembly Word: 12132434
Reverse Word: 12132434
Palindromic: Yes
Weakly Irreducible: Yes
Strongly Irreducible: Yes

- **ispalindrome**[1,2,1,3,2,4,3,4];
val it = true : bool
- tnf[4,3,4,2,3,1,2,1];
val it = [1,2,1,3,2,4,3,4] : int list

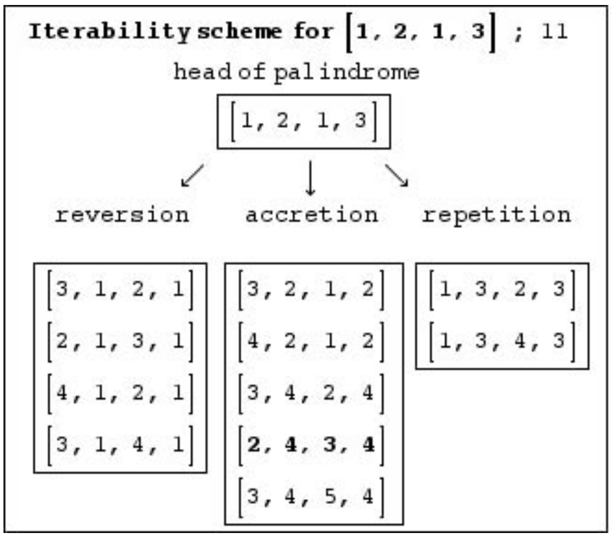
[1,2,1,3] => accr[1,2,1,3] = [2,4,3,4].

- **ispalindrome**[1,2,1,3];
val it = false : bool
- tnf[2,4,3,4];
val it = [1,2,3,2] : int list
- ispalindrome [1,2,3,2];
val it = false : bool

- **ENstructure**[1,2,1,3] = ENstructure[2,4,3,4];
val it = false : bool

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

- **kconcat**[1,2,1,3][3,1,2,1];
Palindrome: val it =   11
  [[1,2,1,3,3,1,2,1],[1,2,1,3,2,1,3,1],[1,2,1,3,3,2,1,2],[1,2,1,3,1,3,2,3],
   [1,2,1,3,4,1,2,1],[1,2,1,3,4,2,1,2],[1,2,1,3,3,1,4,1],[1,2,1,3,1,3,4,3],
   [1,2,1,3,3,4,2,4],**[1,2,1,3,2,4,3,4]**,[1,2,1,3,3,4,5,4]] : int list list

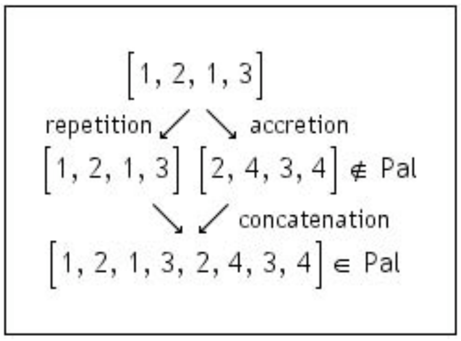


Following the Florida group (N. Jonoška), the modi *repetition* and *reversion* are corresponding generally, i.e. without the restriction to DOWs, to “*repeat words*” and “*return words*" (Ryan Arredondo). What is missing up to now, as far I can see, are the "*accretion words*" or as they may be called now the "*augmentation words*" of the morphogrammatic approach.

The word 12132434 is strongly-irreducible.

The word 12132434, read as a morphogram is decomposable into its non-palindromic parts [1,2,1,3] and [2,4,3,4]. The morphogram [1,2,1,3,2,4,3,4] then appears as an element of the morphic concatenation *kconcat* of [1,2,1,3] and [3,1,2,1], *kconcat*[1,2,1,3][3,1,2,1].
Hence, the palindromic morphogram [1,2,1,3,2,4,3,4] is decomposable into its head [1,2,1,3] and into the *accretion* of the head [2,4,3,4] as its body in context of the concatenation *kconcat*[1,2,1,3][3,1,2,1].
Also, [1,2,1,3] ∩ [3,1,2,1] != ∅, the word (morphogram) [1,2,1,3,2,4,3,4] is *reducible* by accretion. This is in conflict with the DOW analysis of the reducibility of words.

The given example, [1,2,1,3,2,4,3,4], is a palindromic 'double occurrence word' (DOW) but the parts are neither palindromic nor DOWs, and are therefore not reducible to 'double occurrence words'.


[1, 2, 1, 3]
repetition
accretion
[1, 2, 1, 3] [2, 4, 3, 4] ∉ Pal
concatenation
[1, 2, 1, 3, 2, 4, 3, 4] ∈ Pal

All palindromic morphograms are reducible by decomposition in the modi of *repetition*, *accretion* or *reversion*.

Strictly separable non-DOW palindromes
pal = (pal1, pal2)
pal1∩ pal2 = ∅

Example
pal = [1,2,2,3,4,5,5,6]: [1,2,2,3] ∩ [4,5,5,6] = ∅

DOW example for a strictly separable reducible word:
```
- ispalindrome[1,2,2,1,3,4,4,3];
val it = true : bool
```
[1,2,2,1] ∩ [3,4,4,3] = ∅

No classical non-DOW palindrome is strictly separable.
With relabeling in ascending order, *pal* is a palindrome too.

For fst=last(pal1,2):

ispalindrome(tnf[pali1minus frts-lst, palin2 minus frst-lst]) is palindrome:
Palindrome [1,**2,3,4**,1,**2,3,4**,1]:
```
- ispalindrome(tnf[2,3,4,2,3,4]); repetition
val it = true : bool
- ispalindrome(tnf[2,3,4,4,5,2]); reversion
val it = true : bool
- ispalindrome(tnf[2,3,4,5,6,2]); reversion
val it = true : bool
```

[1,2,**2,3,3,2**,2,1] to [2,3,3,2]:
```
- ispalindrome(tnf[2,3,3,2]);
val it = true : bool
```

[1,2,**2,3,4,2**,2,1] to [2,3,4,2]:
- ispalindrome(tnf[2,3,4,2]);
val it = true : bool

[1,2,2,**3,3,**2,2,1] to [3,3]: palindrome,
[1,2,2,**3,4,**2,2,1] to [3,4]: palindrome

1.2.10. Counting palindromes

The number of palindromes of *assembly* words

"If a *double occurrence* word w of size n >= 2 is a palindrome beginning and ending with 1, then the word formed by removing both 1s is also a palindrome. Hence there are Ln 1 palindromes with n letters that start and end with 1."

[1, 2,2,3,4,5,5,1]:
- ispalindrome (tnf[2,2,3,4,5,5]);
val it = true : bool

- ispalindrome(tnf[2,2,3,2,3,3]);
val it = true : bool

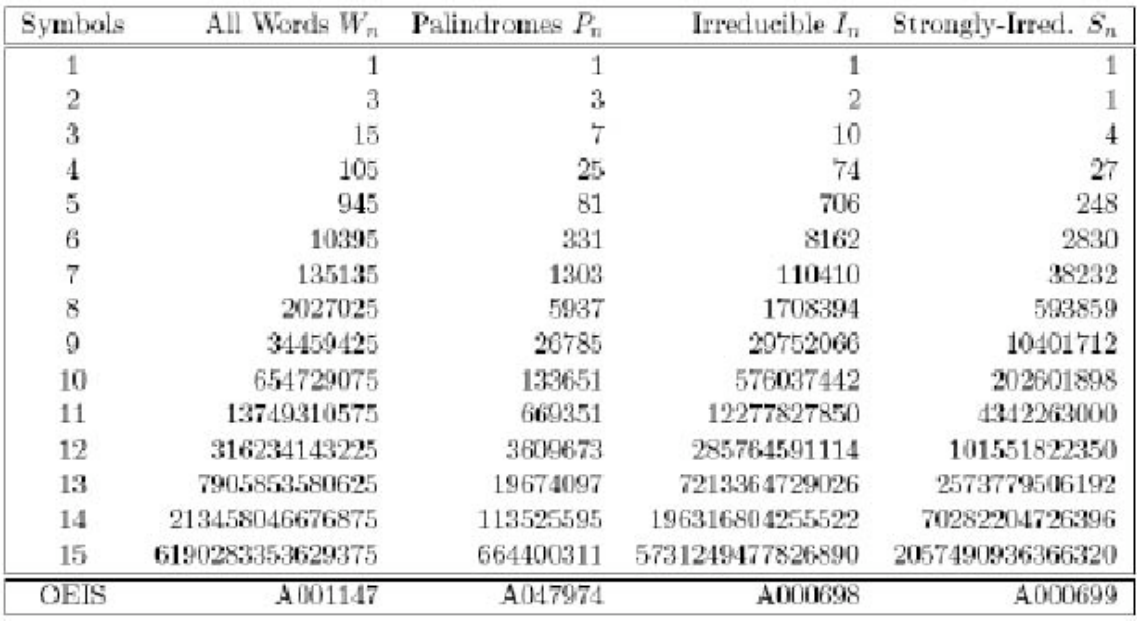
The recursive formula for DOW palindromes of length n is counted by:

$$\boxed{P_n = P_{n-1} + 2(n-1)P_{n-2} \text{ for } n > 1,\ P_0 = 1 \text{ and } P_1 = 1}$$

The number of DOW palindromes with n letters, for any positive integer n is given by the closed formula:

$$\boxed{Pn = \sum_{k=\left[\frac{n}{2}\right]}^{n} \binom{k}{n-k} \frac{n!}{k!}}$$

Sequences of numbers of distinct equivalence classes for various types of **assembly** words.



Symbols	All Words W_n	Palindromes P_n	Irreducible I_n	Strongly-Irred. S_n
1	1	1	1	1
2	3	3	2	1
3	15	7	10	4
4	105	25	74	27
5	945	81	706	248
6	10395	331	8162	2830
7	135135	1303	110410	38232
8	2027025	5937	1708394	593859
9	34459425	26785	29752066	10401712
10	654729075	133651	576037442	202601898
11	13749310575	669351	12277827850	4342263000
12	316234143225	3609673	285764591114	101551822350
13	7905853580625	19674097	7213364729026	2573779506192
14	213458046676875	113525595	196316804255522	70282204726396
15	6190283353629375	664400311	5731249477826890	2057490936366320
OEIS	A001147	A047974	A000698	A000699

Jonathan Burns, et al, Four-Regular Graphs with Rigid Vertices Associated to DNA Recombination, 2011

Number of *morphic* palindromes

n	Tcontexture Bell numbers	Palindromes	symPal	asymPalin
3	5	3	2	1
4	15	7	2	5
5	52	12	5	7
6	203	31	5	36
7	877	59	15	44
8	4140	164	15	159
9	21147	(339)	□	□
10	115975	(999)	□	□
□	Tcard	A080107	A205482 ?	□

A080107
"Number of fixed points of permutation of **SetPartitions** under {1,2,..,n}->{n,n-1,..,1}. Number of symmetric arrangements of non-attacking rooks on upper half of n X n chessboard."

1, 1, 2, 3, 7, 12, 31, 59, 164, 339, 999, 2210, 6841, 16033, 51790, 127643, 428131, 1103372, 3827967, 10269643, 36738144, 102225363, 376118747, 1082190554, 4086419601, 12126858113, 46910207114, 143268057587, 566845074703

http://oeis.org/A080107

Mathematica for A080107
<<DiscreteMath`NewCombinatorica`; Table[t = SetPartitions[n]; t= t /. Thread[Range[n] -> Range[n, 1, -1]]; t= 1 + RankSetPartition /@ t; t= ToCycles[t]; t= Cases[t, {_Integer}]; Length[t], {n, 7}]

bi-symmetric partitions
The number of *bi-symmetric partitions* was enumerated as the sequence A080107

***Mathematical* aspects of palindromes** as *"bi-symmetric partitions"* are pofoundly studied under different motivations.

Guoce Xin, Terence Y.J. Zhang, Enumeration of bilaterally symmetric 3-noncrossing partitions

"Theorem 1. For any given partition P and vacillating tableau V, $P^{refl} = V^{rev}$ if and only if φ(P) = V.

"A vacillating tableau V is said to be *palindromic* if $V = V^{rev}$. A partition P of [n] is said to be bilaterally symmetric (bi-symmetric for short) if $P = P^{refl}$. Theorem 1 implies that P is bi-symmetric if and only if V(P) is palindromic.

The enumeration of bi-symmetric partitions and matchings are not hard, but turns out to be very difficult if we also consider the statistic of crossing number or nesting number.

http://www.sciencedirect.com/science/article/pii/S0012365X08003919 (2008)

Christian Schröder, Palindromic and Even Eigenvalue Problems

"In mathematics it makes sense to talk about *palindromic polynomials* like $p(\lambda) = 5\lambda^4 + 7\lambda^3 + 2\lambda^2 + 7\lambda + 5$, where the *coefficients* form the palindromic sequence 5, 7, 2, 7, 5.

"Passing from scalar to *matrix* coefficients results in palindromic matrix polynomials, $P(\lambda) = \sum_{i=0}^{k} A_i\lambda^i$ where $A_i = A_{k-i} \in C^{n\times n}$.

"While these polynomials are interesting in themselves, a slight variation recently

received a lot more attention: the **-palindromic* polynomials $P(\lambda) = \sum_{i=0}^{k} A_i\lambda^i$, where $A_i = A^*_{k-1} \in C^{n\times n}$ and A* denotes the transpose or *conjugate* transpose of A.
So, *-palindromic polynomials are invariant under *reversing* the order of the coefficients -- and (conjugate-) transposing."

http://page.math.tu-berlin.de/~schroed/Publicat/FasMMS07.pdf

It seems to be quite natural to ask about the consequences for *palindromic polynomials* if morphic abstractions to the palindromic coefficients are applied.

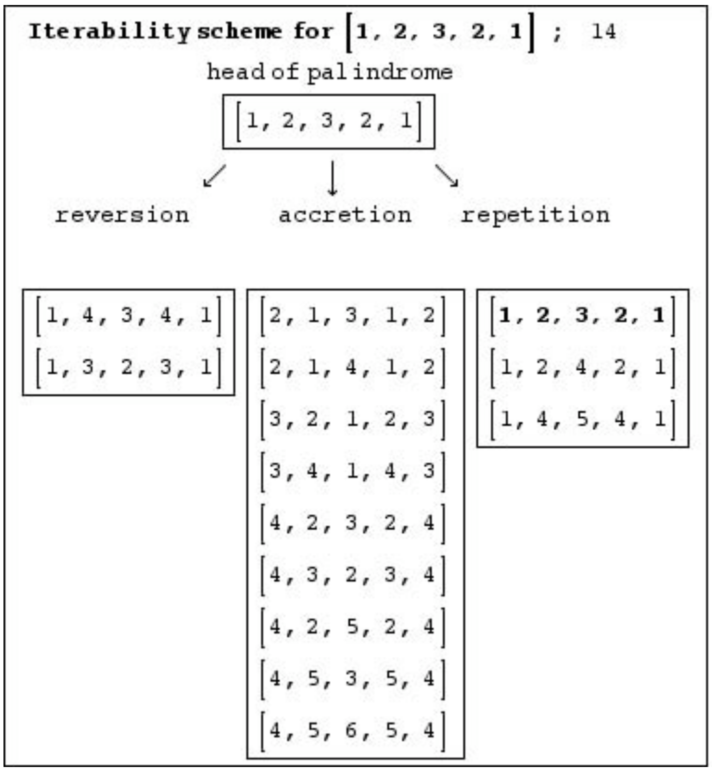
What happens to morpho(p(λ)) for the coefficients $(5\lambda^4 + 7\lambda^3 + 2\lambda^2 + 7\lambda + 5)$?

The sequence (5, 7, 2, 7, 5) is at first morphogrammatically equal to [1,2,3,2,1] as tnf[5,7,2,7,5].
But more interestingly, this sequence belongs morphogrammatically to a *field* of equivalent palindromic return-sequences of the 'root' palindrome [1,2,3,2,1].

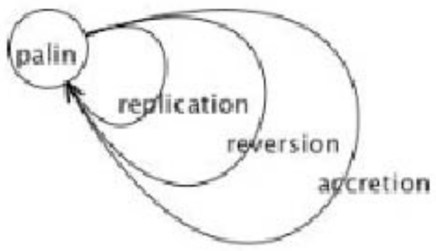
The example of the symmetric palindrome [1,2,3,2,1] has a *field* of 14 palindromes that are representing the possibilities of its 'backwards' reading without violating its structure and palindromicity.

symmetric	**asymmetric palindromes**
[1,2,3,2,1]	[1,2,3,2,1]
↑ ↓	↑ ↓
[1,2,3,2,1]	[3,4,1,4,3]

Palindromic iterability scheme for [1,2,3,2,1]



Palindromic fied of [1,2,3,2,1] aka [5,7,2,7,5]



Obviously, the forwards and the backwards reading of the palindrome are involving different contexts of *viewpoints* of the understanding of the formula. Only one viewpoint is confirming exactly as the same for both directions of reading the modus of the iterability is iteratve inversion.

1.2.11. Return and repeat words

Patterns translated

A *return* word is a word of the form a1 a2 · · · an an · · · a2 a1 , ai ∈ Σ for all i, and ai != aj for i != j
A *repeat* word is a word of the form a1 a2 · · · an a1 a2 · · · an , ai ∈ Σ for all i, and ai != aj for i != j

Special cases of morphic equivalence of return and repeat words

Both cases, *return* and *repeat* of the prefix [1,2,3,4], are DOW words and palindromic. Their ENstructure is different. Obviously, the repeat words need a relabeling mapping to install palindromicity between the word and its repetition.

Ryan Arredondo, Reductions on Double Occurrence Words (03/08/2013)http://131.247.211.200/multimedia/Boca.pdf

DOW palindromes(3)
112233 122331 121323 123123 **123231** 123312 123321
direct:
repeat: 123123
return: 123321
with *relabeling*:
repeat: 121323, 123312, 123231,
return: 122331, 112233

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

```
- ENstructure[1,2,3,4,1,2,3,4] = ENstructure[1,2,3,4,4,3,2,1];
val it = false : bool
```

non-DOW palindromes

repetition as augmentation:
[1,2,2,3,3,2,2,4]

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

The Florida model of palindromes is not giving a *constructive* method of constructing palindromes.

Applied to the restricted case of DOWs, the operations of "*repeat*" and "*direct*" are not directly covering all cases. The property of palindromicity via relabeling of *repeat* and *return* has secoundarily to check the composed word for palindromicity. Not all possible combinations are delivering palindromes.

1.2.12. Constructing palindromes

Given the head of a possible palindrome. How to construct out of the given head a palindrome?

Procedures are "*repeat*" (repetition) and "*return*" (reversion), completed with the *relabeling* procedure and the *test* of palindromicity. This approach holds for double occurence words (DOW) and adapts to the general case too.

Are there morphic palindromes not definied by this procedures?

Given the head [1,2,2,3], how is the DOW morphic palindrome [1,2,2,3,1,4,4,3] constructed?
The body = [1,4,4,3] is a repetition of the head = [1,2,2,3] modulo relabeling. Therefore it is accepted as the body of the palindrome.
The case for a body = [4,5,5,6] is accepted by relabeling: rel[4,5,5,6] = [1,2,2,3]. But the accepted morphic palindrome [1,2,3,4,5,5,6] is not a DOW palindrome. Hence, the DOW property is restricting the range of combinations.

Hence, the main condition that words are DOWs has to be respected as the first distinction of the constructions.

Repetion or reversion as *augmentation* in the sense of accretion is therefore not necessary for DOWs with relabeling function.

Without the restrinction on DOWs, a morphogram [1,2,2,3,4,5,5,6] is well accepted as a morphic palindrome.

But not the morphogram [1,2,2,3,2,4,4,5]:

```
- ispalindrome [1,2,2,3,2,4,4,5];
val it = false : bool
```

To construct all general morphic palindrome it seems to be necessary to add more rules to support repetition, reversion and accretion. This is covered by the *context rules* for palindromes (pal-CR).

Similar to the morphogrammatic rules for multiplication where context rules are limiting the combinatorial possibilities given by an atomistic or symbolic approach to the multiplication of sign sequences, the building of palindromes as a special kind of concatenation has to apply the context rules on the morphogrammatic operation **kconcat**[-][-].

Reversion
kconcat [mg](rev[mg])
Repetition
kconcat[mg][mg]

The test shows that the reversive **kconcat** [mg](rev[mg]) is part of the repetive **kconcat**[mg][mg].

Hence, for morphic palindromes, reversion is a special case of repetition for morphogrammatic concatenation.

Again,
```
- Tcard 8;
val it = 4140 : int
```

```
- length(kconcat[1,2,2,3] [3,2,2,1]);
val it = 34 : int
```

```
- length(kconcat[1,2,2,3][1,2,2,3]);
val it = 34 : int
```

```
Palindrome:  val it =
  [[1,2,2,3,3,2,2,1],[1,2,2,3,2,3,3,1],[1,2,2,3,3,1,1,2],[1,2,2,3,1,2,2,3],
   [1,2,2,3,4,2,2,1],[1,2,2,3,4,1,1,2],[1,2,2,3,3,4,4,1],[1,2,2,3,1,4,4,3],
   [1,2,2,3,3,2,2,4],[1,2,2,3,2,3,3,4]] : int list list
- length it;
val it = 10 : int
```

```
Palindrome: 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
- length it;
val it = 14 : int
```

Asymmetry with reverse and relabel

symmetric and asymmetric words

word = [1,2,2,3,1,3]
word = DOW, non-palindrome

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

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

tnf(rev(word)): DOW, non-palindrome
word != tnf(rev(word))

word = [1,2,1,3,2,3]
word = DOW, palindrome

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

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

tnf(rev(word)): DOW, palindrome
word = tnf(rev(word))

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

Equality, equivalence and ‘canonical forms'

"Two DOWs are said to be equal if they have the same canonical form.”
This corresponds to some morphogrammatic approaches:
Two morphograms are equivalent iff they have the same (equal) E/N-structure.

There is a difference too in constucting palindromes or to filter them out of a set of words (morphograms), say with the procedure *List.filter ispalindrome "Tcontexture n"* for morphic palindromes.

Hence, where are the asymmetric palindromes gone?

The asymmetric morphic palindromes are asymmetric because they insist on the difference between semiotic or symbolic constructions, i.e. words, and the kenogrammatic scriptural level of morphogrammatic constructions, i.e. morphograms.

Accepting this basic difference in the mode of writing, the case for asymmetric palindromes is clearly established. There is no such fallacy of having forgotten to apply the *relabeling* fixture.

It is just a secondary, and obviously convenient and practical decision, to employ the operation of *relabeling* in a *second* step to produce on a semiotic level symmetry where there is in fact asymmetry.

1.2.13. What are the advantages of the morphogrammatic understanding of palindromes?

Neither the identitive semiotic approach nor the more abstract approach of equivalences of relabeled palindromes offers a mechanism to deal with *distributed* palindromes in contextural complexions.

Morphograms, and therefore palindromes too, are not just simple sequences of signs but involved into a distribution and mediation mechanism, called morphic *dissemination*.

Hence a single morphogram or a single palindrome is therefore just a morphogram at a single locus.

At least two different modi of distribution of palindromes have to be considered: first the *intra-contextural* and second the *trans-contextural* distribution and its *mediations*.

Therefore palindromes are not just appearing in a intra-contextural distribution but also in disjunct, i.e. *discontextural* distributions. As a consequence, a new type of possible palindromes occurs: the palindromes, i.e. palindromic paths, in fact *journeys*, between contextures.

In short: A palindrome is not a just palindrome *per se*. A palindrome is situated in a context that frames it as a palindrome at the distinguished place or locus it occurs.

The classical concept of palindromes is based on the *uniqueness* of its alphabet. Because this uniqueness is unique it has not to be inscribed, noted or reflected. It is as it is.

Morphograms are not given in the is-abstraction mode but in the mode of "X as Y is Z", that is the *as-abstraction*.

1.3. Towards an algorithm for the production of morphic palindromes

1.3.1. Definitions

$w = w^R$ $\quad$ $R(w) = rev(uv) = rev(v)rev(u)$
$w = uv$

relabeling: pal:: rel(rev(w)) = w
rel(rev(w)) = rel(rev(v) rev(u))

uv = u o v, o = accretive, iterative concatenation (composition)

$$u \circ v = \begin{pmatrix} \text{repetition of } u \\ \text{reversion of } u \\ \text{augmentation of } u \end{pmatrix} \times \begin{pmatrix} \text{iteration} \\ \text{accretion} \end{pmatrix}$$

u = pal or non-pal

frame
repetition = frame of v = frame of u
reversion = frame of v = accr(rev(frame of u)) and lst(v) = frst(u)
augmentation: frame of v != frame of u or rev(frame(u))
core
repetition: core(v) = accr(core(u)), core(v) != frame(u)
reversion:
augmentation:

First attempt

For all w∈Pal : tnf[v] = u
u = palindrome.
1. v = direct repetition or v = direct reversion,
2. v = direct accretive repetition or v = direct accretive reversion,
3. v = direct accretion of u.

Example
u = palindrome = [1,2,3,4]

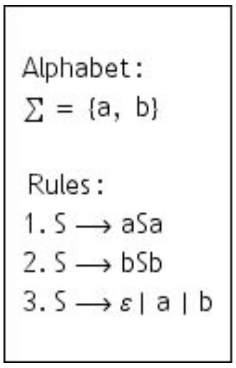
1. v = direct *repetition* :	[1,2,3,4];
or v = direct *reversion*:	[4,3,2,1], [5,3,2,1];
2. v = direct accretive *repetition*:	[1,3,5,4], [1,5,6,4];
inverse accretive repetition:	[1,3,2,4], [1,5,2,4];
or v = direct accretive *reversion*:	[4,5,6,1], [5,6,7,1], [3,4,5,1],
repetive accretive revers:	[2,4,5,1], [5,2,3,1], [4,3,5,1];
rev-core:	[4,2,3,1]; [5,2,3,1], [5,3,2,1];
3. v = direct accretion of u:	[2,3,4,5], [3,4,5,6], [4,5,6,7], [5,6,7,8];
repetive accretion:	[4,1,2,3], [5,1,2,3];
rev-core	[2,4,5,1], [2,5,4,1]; [4,3,2,5], [4,2,3,5];
	[5,2,3,6], [5,3,2,6];

1.3.2. Recursive generation of palindromes

Production rule for Palindromes

A palindrome is a word that reads the same both forward and backwards, such as rotor. An algorithm to determine whether or not a word is a palindrome can be expressed recursively. Simply strip off the first and last letters; if they are different, the word is not a palindrome. If they are, test the remaining string (after the first and last letters have been removed) to see if it is a palindrome.

Classical grammar for palindromes over the alphabet $\Sigma = \{a, b\}$.
Context Free Grammar (CFG) production rules for palindromes:



Alphabet:
$\Sigma = \{a,\ b\}$

Rules:
1. $S \longrightarrow aSa$
2. $S \longrightarrow bSb$
3. $S \longrightarrow \varepsilon \mid a \mid b$

http://www.univ-orleans.fr/lifo/Members/Mirian.Halfeld/Cours/TLComp/l3-CFG.pdf

Example
S -->3 a -->2 bab -->1 ababa -->1 aababaa.

1.3.3. List of morphic palindromes from [1] tp [1,2,3,4,5,6,7]

[[**1**],
[1,1],[1,**2**], even; iter, accr
odd(3): [1,1,1],[1,2,1],[1,2,**3**], odd; iter, accr-sym, accr-asym

even(4): [1,1,1,1],[1,1,2,2],[1,2,1,2], [1,2,2,1],[1,2,2,3],[1,2,3,1],[1,2,3,**4**],

odd(5): [1,1,**1**,1,1], [1,1,**2**,1,1],[1,1,**2**,3,3],[1,2,**1**,2,1],[1,2,**1**,3,1],[1,2,**2**,2,1],[1,2,**2**,2,3],
[1,2,**3**,1,2],[1,2,**3**,2,1],[1,2,**3**,2,4],[1,2,**3**,4,1],[1,2,**3**,4,**5**],

even(6): [1,1,1,1,1,1],[1,1,1,2,2,2],[1,1,2,1,2,2],[1,1,2,2,1,1],[1,1,2,2,3,3],[1,1,2,3,1,1],[1,1,2,3,4,4],
[1,2,1,1,2,1],[1,2,1,1,3,1],[1,2,1,2,1,2],[1,2,1,3,2,3], [1,2,1,3,4,3],
[1,2,2,1,1,2],[1,2,2,2,2,1], [1,2,2,2,2,3],[1,2,2,3,3,1],[1,2,2,3,3,4],
[1,2,3,1,2,3],[1,2,3,1,4,3],[1,2,3,2,3,1],[1,2,3,2,3,4], [1,2,3,3,1,2],
[1,2,3,3,2,1], [1,2,3,3,2,4],[1,2,3,3,4,1],[1,2,3,3,4,5],
[1,2,3,4,1,2], [1,2,3,4,2,1],[1,2,3,4,2,5],[1,2,3,4,5,1],[1,2,3,4,5,**6**],

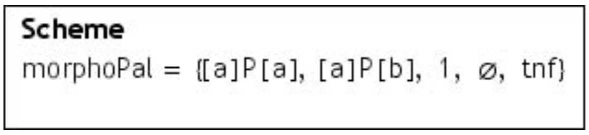
odd(7): [1,1,1,**1**,1,1,1],
[1,1,1,**2**,1,1,1],[1,1,1,**2**,3,3,3],[1,1,2,**1**,2,1,1],
[1,1,2,1,3,1,1],[1,1,2,2,2,1,1],[1,1,2,2,2,3,3],[1,1,2,3,1,2,2],
[1,1,2,3,2,1,1],[1,1,2,3,2,4,4],[1,1,2,3,4,1,1],[1,1,2,3,4,5,5],
[1,2,1,1,1,2,1],[1,2,1,1,1,3,1],[1,2,1,2,1,2,1],[1,2,1,2,3,2,3],
[1,2,1,3,1,2,1],[1,2,1,3,1,4,1],[1,2,1,3,2,1,2],[1,2,1,3,4,2,4],
[1,2,1,3,4,5,4],[1,2,2,1,2,2,1],[1,2,2,1,3,3,1],[1,2,2,2,2,2,1],
[1,2,2,2,2,2,3],[1,2,2,3,1,1,2],[1,2,2,3,2,2,1],[1,2,2,3,2,2,4],
[1,2,2,3,4,4,1],[1,2,2,3,4,4,5],[1,2,3,1,2,3,1],[1,2,3,1,3,2,1],
[1,2,3,1,3,4,1],[1,2,3,1,4,2,1],[1,2,3,1,4,5,1],[1,2,3,2,1,2,3],
[1,2,3,2,3,2,1],[1,2,3,2,3,2,4],[1,2,3,2,4,2,1],[1,2,3,2,4,2,5],
[1,2,3,3,3,1,2],[1,2,3,3,3,2,1],[1,2,3,3,3,2,4],[1,2,3,3,3,4,1],
[1,2,3,3,3,4,5],[1,2,3,4,1,2,3],[1,2,3,4,1,5,3],[1,2,3,4,2,3,1],
[1,2,3,4,2,3,5],[1,2,3,4,3,1,2],[1,2,3,4,3,2,1],[1,2,3,4,3,2,5],
[1,2,3,4,3,5,1],[1,2,3,4,3,5,6],[1,2,3,4,5,1,2],[1,2,3,4,5,2,1],
[1,2,3,4,5,2,6],[1,2,3,4,5,6,1],[1,2,3,4,5,6,**7**].

Morphogrammatic ‘grammar’ for palindromes

In contrast to classical grammars, morphogrammatic production systems are contextual in the specific sense that they are relating the range of their production to the just produced products (strings, objects, morphograms) and not to a pre-given alphabet manipulated by stable rules.

Again, the operation of ‘concatenation’ for palindromes, with w = u o v, is not atomically defined as for atomistic formal languages but is retro-recursive entangled with the operands of the previous operation. Hence, the operator of morphic concatenation or specifically, the operator of *prolongation*, and here, for palindromes, of double, forwards and backwards, prolongations, is reflecting the contextual realizations that have to be prolongated.

Mimickry of a *production system* for palidromes.


Scheme
morphoPal = {[a]P[a], [a]P[b], 1, ∅, tnf}

Palindrome grammar

$$P = \left[\text{w1w2}\right],\ \text{w} = \text{w1 w2}$$

$$\text{Rule1}:\ \text{w1 P w2}\ :\ \left[\text{w1w1w2w2}\right]$$

$$\text{Rule2}:\ \text{w2 P w1}\ :\ \left[\text{w2w1w2w1}\right]$$

$$\text{Rule3}:\ \text{w3 P w3}\ :\ \left[\text{w3w1w2w3}\right]$$

$$\text{Rule4}:\ \text{w3 P w4}\ :\ \left[\text{w3w1w2w4}\right].$$

Defs

$$\text{w3} = \text{add}\left(\left|\text{w1}\right|,\ 1\right)$$

$$\text{w4} = \text{add}\left(\left|\text{w3}\right|,\ 1\right) = \text{add}\left(\text{add}\left(\left|\text{w2}\right|,\ 1\right),\ 1\right).$$

Palindrome production rules

$$\text{Rule1}:\ \left[P\right] \Longrightarrow \left[\text{w1Pw2}\right]$$

$$\text{Rule2}:\ \left[P\right] \Longrightarrow \left[\text{w2Pw1}\right]$$

$$\text{Rule3}:\ \left[P\right] \Longrightarrow \left[\text{w3Pw3}\right]$$

$$\text{Rule4}:\ \left[P\right] \Longrightarrow \left[\text{w3Pw4}\right].$$

Defs

$$P = \left[\text{w}\right] = \left[\text{w1 w2}\right]$$

$$\text{w3} = \text{add}\left(\left|\text{w1}\right|,\ 1\right)$$

$$\text{w4} = \text{add}\left(\left|\text{w3}\right|,\ 1\right) = \text{add}\left(\text{add}\left(\left|\text{w2}\right|,\ 1\right),\ 1\right).$$

Categorial rules

$$\text{morphoPal rules} = \begin{pmatrix} \text{repetition: } \begin{pmatrix} \text{direct : rule1} \\ \text{inverse: rule2} \end{pmatrix} \\ \\ \text{accretion: } \begin{pmatrix} \text{symmetric: rule3} \\ \text{asymmetric: rule4} \end{pmatrix} \end{pmatrix}$$

Some productions

[∅]	1P1	P = [∅]	[1,1]	: iter, rep	: rule1
	1P2		[1,2]	: accr, asym	: rule4
	(2P1		[2,1],	: but [1,2] = [2,1])	
[1]	1P1	P = [1]	[1,1,1]	: iter, rep	: rule 1
	2P2	[2,1,2]	[1,2,1]	: accr, rep	: rule3
	2P3	[2,1,3]	[1,2,3]	; accr, asym	: rule4
	(3P2	[3,1,2])			
[1,1]	1P1		[1,**1**,**1**,1]	: iter	: rule1
	2P2	[2,**1**,**1**,2] ->	[1,2,2,1]	: accr, sym	: rule3
	2P3	[2,**1**,**1**,3] ->	[1,2,2,3]	: accr, asym	: rule4
	(3P2	[3,**1**,**1**,2] ->	[1,2,2,3]	: accr, asym	: rule4')

[1,2]	1P2	[1,**1,2**,2]	: repetition	: rule1
		[2,**1,2**,1] -> [1,2,1,2]	: reversion	: rule2
		[3,**1,2**,3] -> [1,2,3,1]	: accr, sym	: rule3
		[3,**1,2**,4] -> [1,2,3,4]	: accr, asym	: rule4

[1,1,1,1]   [1,**1,1,1**,1] =
[2,1,1,1,1,2] -> [1,2,2,2,2,1] =
[2,1,1,1,1,3] -> [1,2,2,2,2,3] =
([3,1,1,1,1,2] -> [1,2,2,2,2,3])

[1,2,2,1]   [1,1,2,2,1,1] =
[2,1,2,2,1,2] -> [1,2,1,1,2,1] =
[3,1,2,2,1,3] -> [1,2,3,3,2,1] =
[3,**1,2,2,1**,4] -> [1,2,3,3,2,4] =

[1,2,2,3]   [1,1,2,2,3,3] =
[3,1,2,2,3,1] -> [1,2,3,3,1,2] =
[4,1,2,2,3,5] -> [1,2,3,3,4,5] =
[2,**1,2,2,3**,2] -> [1,2,1,1,3,1] =
[4,1,2,2,3,4] -> [1,2,3,3,4,1] =

[1,1,2,2,]   [1,1,1,2,2,2] =
[2,1,1,2,2,1] -> [1,2,2,1,1,2] =
[3,1,1,2,2,4] -> [1,2,2,3,3,4] =
[3,1,1,2,2,3] -> [1,2,2,3,3,1] =

[1,2,1,2]   [1,1,2,1,2,2] =
[2,1,2,1,2,1] -> [1,2,1,2,1,2] =
[3,1,2,1,2,4] -> [1,2,3,2,3,4] =
[3,1,2,1,2,3] -> [1,2,3,2,3,1] =

[1,2,3,1]   [1,1,2,3,1,1] =
[4,1,2,3,1,4] -> [1,2,3,4,2,1] =
[4,1,2,3,1,5] -> [1,2,3,4,2,5] =
[2,1,2,3,1,3] -> [1,2,1,3,2,3] =
[3,1,2,3,1,2] -> [1,2,3,1,2,3] =

[1,2,3,4]	1P4	[1,1,2,3,4,4] =	frame	iter
	4P1	[4,1,2,3,4,1] -> [1,2,3,4,1,2] =	rev
	2P3	[2,1,2,3,4,3] -> [1,2,1,3,4,3] =	core
	3P2	[3,1,2,3,4,2] -> [1,2,3,1,4,3] =	core
		[5,1,2,3,4,6] -> [1,2,3,4,5,6] =	accr, asym
		[5,1,2,3,4,5] -> [1,2,3,4,5,1] =	accr, sym

Productions for odd palindromes

=> [1]
[1]   [1,1,1]
[2,1,2] -> [1,2,1]
[2,1,3] -> [1,2,3].
[1,1,1]   [1,1,1,1,1]
[2,1,1,1,2] -> [1,2,2,2,1]
[2,1,1,1,3] -> [1,2,2,2,3]
[1,2,1]   [1,1,2,1,1]
[2,1,2,1,2] -> [1,2,1,2,1]
[3,1,2,1,3] -> [1,2,3,2,1]
[3,1,2,1,4] -> [1,2,3,2,4]
[1,2,3]   [1,1,2,3,3]
[3,1,2,3,1] -> [1,2,3,1,2]
[2,1,2,3,2] -> [1,2,1,3,1]
[4,1,2,3,4] -> [1,2,3,4,1]
[4,1,2,3,5] -> [1,2,3,4,5].

[1,2,3] : $[\boxed{2},1,\mathbf{2},3,\boxed{2}]$
$[\boxed{1},\mathbf{1},2,\mathbf{3},\boxed{3}]$,
$[\boxed{3},\mathbf{1},2,\mathbf{3},\boxed{1}]$
$[\boxed{\mathbf{4}},1,2,3,\boxed{\mathbf{4}}]$
$[\boxed{\mathbf{4}},1,2,3,\boxed{\mathbf{5}}]$

sum(pal(5)) = 12
[1,1,1,1,1], [1,1,2,1,1],[1,1,2,3,3],[1,2,1,2,1],[1,2,1,3,1],[1,2,2,2,1],[1,2,2,2,3],
[1,2,3,1,2],[1,2,3,2,1],[1,2,3,2,4],[1,2,3,4,1],[1,2,3,4,5]

Further exemplification

P = [1,2,3,4,5,6] : u o v : [1,2,3] o [4,5,6]
succ(u) o succ(v) : {1,2,3} {4,5,6} {7,8}

P = [1,2,3,4,5,6]: **tnf**
$[\boxed{3},1,2,\mathbf{3},\mathbf{4},5,6,\boxed{4}]$ -> [1,2,3,1,4,5,6,4]
$[\boxed{4},1,2,\mathbf{3},\mathbf{4},5,6,\boxed{3}]$ -> [1,2,3,4,1,5,6,4]
$[\boxed{2},1,\mathbf{2},3,4,\mathbf{5},6,\boxed{5}]$ -> [1,2,1,3,4,5,6,5]
$[\boxed{5},1,\mathbf{2},3,4,\mathbf{5},6,\boxed{2}]$ -> [1,2,3,4,5,1,6,3]
$[\boxed{1},\mathbf{1},2,3,4,5,\mathbf{6},\boxed{6}]$ -> [1,1,2,3,4,5,6,6]
$[\boxed{6},\mathbf{1},2,3,4,5,\mathbf{6},\boxed{1}]$ -> [1,2,3,4,5,6,1,2]
$[\boxed{\mathbf{7}},1,2,3,4,5,6,\boxed{\mathbf{7}}]$ -> [1,2,3,4,5,6,7,1]
$[\boxed{\mathbf{7}},1,2,3,4,5,6,\boxed{\mathbf{8}}]$ -> [1,2,3,4,5,6,7,8].

P = [1,2,3,1,4,5,6,4] : **tnf**
[1,1,2,3,1,4,5,6,4,4] -> [1,1,2,3,1,4,5,6,4,4]
[4,1,2,3,1,4,5,6,4,1] -> [1,2,3,4,2,1,5,6,1,2]
[2,1,2,3,1,4,5,6,4,6] -> [1,2,1,3,2,4,5,6,4,6]
[6,1,2,3,1,4,5,6,4,2] -> [1,2,3,4,2,5,6,1,5,3]
[3,1,2,3,1,4,5,6,4,5] -> [1,2,3,1,2,4,5,6,4,5]
[5,1,2,3,1,4,5,6,4,3] -> [1,2,3,4,2,5,1,6,5,4]
[7,1,2,3,1,4,5,6,4,7] -> [1,2,3,4,2,5,6,7,5,1]
[7,1,2,3,1,4,5,6,4,8] -> [1,2,3,4,2,5,6,7,5,8] .

[1,2,2,3]: $[\boxed{2},1,\mathbf{2},\mathbf{2},3,\boxed{2}]$ -> [1,2,1,1,3,1]
$[\boxed{1},\mathbf{1},2,2,\mathbf{3},\boxed{3}]$ -> [1,1,2,2,3,3]
$[\boxed{3},\mathbf{1},2,2,\mathbf{3},\boxed{1}]$ -> [1,2,3,3,1,2]
$[\boxed{\mathbf{4}},1,2,2,3,\boxed{\mathbf{4}}]$ -> [1,2,3,3,4,1]
$[\boxed{\mathbf{4}},1,2,2,3,\boxed{\mathbf{5}}]$ -> [1,2,3,3,4,5] .

tnf(succ([mg])) = succ(tnf[mg])
[2,1,2] [2,2,1,2,2] -> [1,1,2,1,1]
[1,2,1,2,1]
[3,2,1,2,3] -> [1,2,3,2,1]
[3,2,1,2,4] -> [1,2,3,2,4]

1.3.4. Grammar for morphic palindromes

Palindrome grammar

$P = [w1w2],\ w = w1\, w2$

Rule1 : $w1\, P\, w2$: $[w1w1w2w2]$

Rule2 : $w2\, P\, w1$: $[w2w1w2w1]$

Rule3 : $w3\, P\, w3$: $[w3w1w2w3]$

Rule4 : $w3\, P\, w4$: $[w3w1w2w4]$.

Defs

$w3 = \mathrm{add}(|w1|, 1)$

$w4 = \mathrm{add}(|w3|, 1) = \mathrm{add}(\mathrm{add}(|w2|, 1), 1)$.

Palindrome production system

Alphabet

$\Sigma = \mathrm{Int}$

Axioms

Axiom01 : $\Longrightarrow [\varnothing]$

Axiom02 : $\Longrightarrow [1]$

Rules

Rule1 : $[w] \Longrightarrow [w1Pw2]$

Rule2 : $[w] \Longrightarrow [w2Pw1]$

Rule3 : $[w] \Longrightarrow [w3Pw3]$

Rule4 : $[w] \Longrightarrow [w3Pw4]$.

Defs

$P = [w] = [w1\, w2]$

$w3 = \mathrm{add}(|w1|, 1)$

$w4 = \mathrm{add}(|w3|, 1) = \mathrm{add}(\mathrm{add}(|w2|, 1), 1)$.

Some starts :

$[\varnothing] \longrightarrow [1, 1],\ [1, 2]$: R1, R4

$[1] \longrightarrow [1, 1, 1],\ [2, 1, 2],\ [2, 1, 3]$: R1, R3, R4

Production example

([w1=1, w2=2], [w1=1,w2=1]): P = [1,2] and P = [1,1]

```
P = [1,1]:   w1Pw1                 [1,1,1,1]  ; rule1 (rule2)
             w3Pw3   [2,1,1,2] -> [1,2,2,1]  ; rule3
             w3Pw4   [2,1,1,3] -> [1,2,2,3]  ; rule4

P = [1,2]:   w1Pw2   [1,1,2,2]               ; rule1 : direct repetition
             w2Pw1   [2,1,2,1] -> [1,2,1,2]  ; rule2 : inverse repetition
             w3Pw3   [3,1,2,3] -> [1,2,3,1]   ; rule3 : symmetric accretion
             w3Pw4   [3,1,2,4] -> [1,2,3,4]  ; rule4 : asymmetric accretion
```

Easy test for palindromicity of a morphogram

Like for classical palindromes there is a method to reduce palindrome to its atomic elements by using the rules in reverse.

This goes back to Palul Lorenzen's Principle of Inversion for formal calculi.

The difference to morphic calculi is defined by their context rules that don't exist for formal calculi (systems, languages).

Classical rules
Ex. S --> {∅,|a|, |b|, |c|, aSa, bSb, cSb}

Example
(acbcaacbca) is palindrome?
Rule-a: (cbcaacbc)
Rule-c: bcaacb
Rule-b: caac
Rule-c: aa
Rule-a: a.

Hence, (acbcaacbca) is a classical palindrome.

Morphic rules

Negative examples
Given the morphogram [1,2,3,1,1,2]. Is it a palindrome?
One method is to apply the grammar rules in reverse. If the morphogram is reducible to [1] (or [1,1], [1,2]) following the reverted rules, then it is a palindrome.

rule2: [1,2,3,1,1,2] -> [2,3,1,1]
rule2: [2,3,1,1] -> [3,1]
?? : [3,1] -> [1] : there is no rule to produce [1] from [3,1].
Hence, [1,2,3,1,1,2] is not a palindrome.

Further example:
rule1: [1,2,3,1,1,1,1] -> [2,3,1,1,1]
rule2: [2,3,1,1,1] -> [3,1,1]
no rules for [3,1,1] -> [1].
Hence, [1,2,3,1,1,1,1] is not a palindrome.

Positive example

Morphogram [1,2,3,4,1,1,3,4,5,1]:
rule1: [1,2,3,4,1,1,3,4,5,1] -> [2,3,4,1,1,3,4,5] :
rule4: [**2**,3,4,1,1,3,4,**5**] -> [3,4,1,1,3,4] : (2,5) , no rule for [2,5] applicable!!
rule4: [3,4,1,1,3,4] -> [4,1,1,3]
rule4: [4,1,1,3] -> [1,1]
rule1: [1,1] -> [1].

Reformulation
Hence, the morphogram has to be re-writen in a form that is accessible to the rules.
This is shown with the following examples.

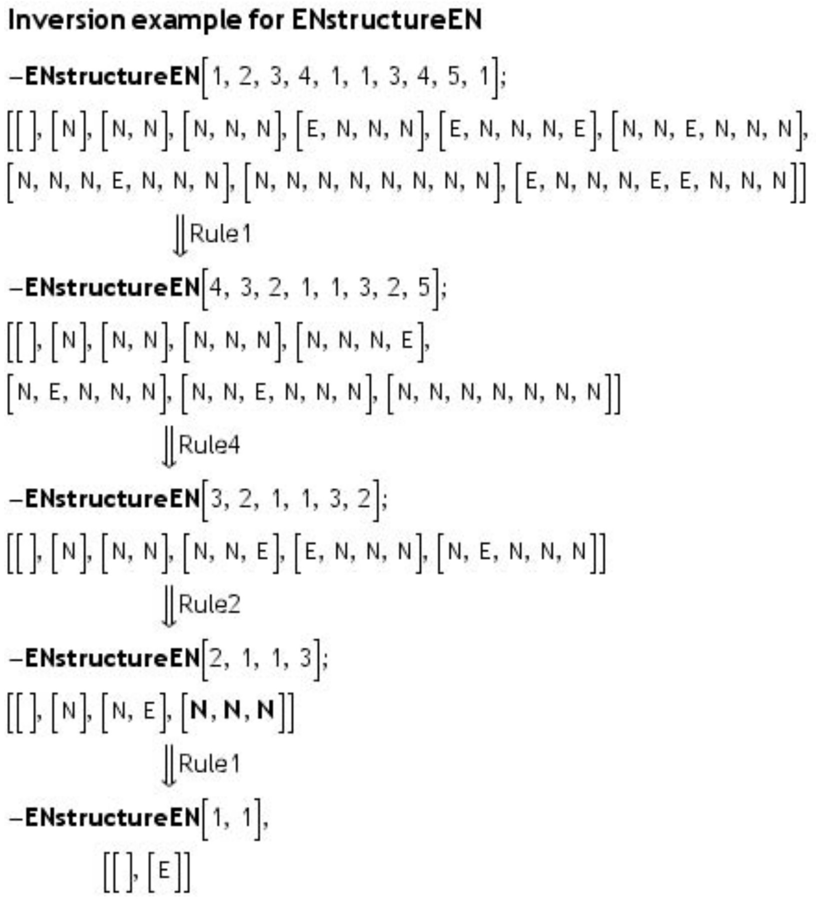
Morphogram [1,2,3,4,1,1,3,4,5,1] -> [1,4,3,2,1,1,3,2,5,1] tnf
rule1: [1,4,3,2,1,1,3,2,5,1] -> [4,3,2,1,1,3,2,5]
rule4: [4,3,2,1,1,3,2,5] -> [3,2,1,1,3,2]
rule4: [3,2,1,1,3,2] -> [2,1,1,3]
rule4: [2,1,1,3] -> [1,1]

rule1: [1,1] -> [1].
Hence, the morphogram [1,2,3,4,1,1,3,4,5,1] is a palindrome.

The rules and their inversion are not directly applicable to the morphogram read as a sign sequence. The rules are written in a *canonical* form. Hence, the notation of the morphogram has to be *adjusted* to the rules to be directly applied. This is possible without conflicts by the tnf-function.

A more direct solution is achieved by the choice for the E/N-notation.
The EN-notation is neutral to the list-representation of the morphograms.

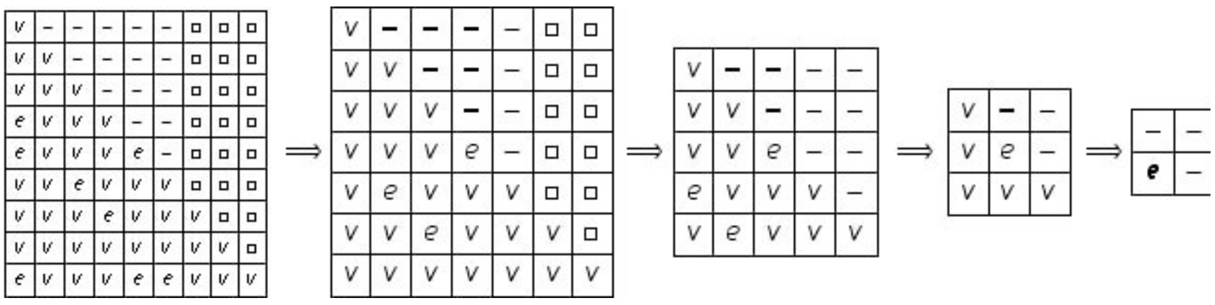
Hence,
ENstructureEN[1,2,3,4,1,1,3,4,5,1] = ENstructureEN[1,4,3,2,1,1,3,2,5,1].



Inversion example for ENstructureEN

```
–ENstructureEN[1, 2, 3, 4, 1, 1, 3, 4, 5, 1];
[[ ], [N], [N, N], [N, N, N], [E, N, N, N], [E, N, N, N, E], [N, N, E, N, N, N],
[N, N, N, E, N, N, N], [N, N, N, N, N, N, N, N], [E, N, N, N, E, E, N, N, N]]
                ⇓Rule1
–ENstructureEN[4, 3, 2, 1, 1, 3, 2, 5];
[[ ], [N], [N, N], [N, N, N], [N, N, N, E],
[N, E, N, N, N], [N, N, E, N, N, N], [N, N, N, N, N, N, N]]
               ⇓Rule4
–ENstructureEN[3, 2, 1, 1, 3, 2];
[[ ], [N], [N, N], [N, N, E], [E, N, N, N], [N, E, N, N, N]]
               ⇓Rule2
–ENstructureEN[2, 1, 1, 3];
[[ ], [N], [N, E], [N, N, N]]
               ⇓Rule1
–ENstructureEN[1, 1],
      [[ ], [E]]
```

ENstructureEN tables: direct cutting



v	–	–	–	–	–	□	□	□
v	v	–	–	–	–	□	□	□
v	v	v	–	–	–	□	□	□
e	v	v	v	–	–	□	□	□
e	v	v	v	e	–	□	□	□
v	v	e	v	v	v	□	□	□
v	v	v	e	v	v	v	□	□
v	v	v	v	v	v	v	v	□
e	v	v	v	e	e	v	v	v

$\Longrightarrow$

v	–	–	–	–	□	□
v	v	–	–	–	□	□
v	v	v	–	–	□	□
v	v	v	e	–	□	□
v	e	v	v	v	□	□
v	v	e	v	v	v	□
v	v	v	v	v	v	v

$\Longrightarrow$

v	–	–	–	–
v	v	–	–	–
v	v	e	–	–
e	v	v	v	–
v	e	v	v	v

$\Longrightarrow$

v	–	–
v	e	–
v	v	v

$\Longrightarrow$

–	–
e	–

More at:
http://memristors.memristics.com/Grammars%20and%20Programs
/Grammars%20and%20Programs.html

Production of [1,2,3,1,2,3]   [3,1,2,3,1,2])
-> ∅
rule4: [∅] -> [1,2]
rule3: [1,2] -> [3,1,2,3] -> [1,2,3,1])
rule4: [3,1,2,3] -> [4,3,1,2,3,5] -> [1,2,3,1,2,3].

Revesion of production has to consider the reversion of the tnf-operation as part of the context-rules. This reformulation is not bijective. Different realisations that are compatible to the rules are possible.

For the morphogram [1,2,3,1,2,3], which is also a DOW palindrome, there are 4 realisations possible that are conform with the rules:
[1,2,3,1,2,3]:
    [3,1,2,3,1,2]
    [2,1,3,3,2,1]
    [1,3,2,1,3,2]
    [2,3,1,2,3,1]

Again, with the E/N-notation that is not depending on a canonical representation, this distinction would disapear. The disadvantage of the E/N-approach is its higher complexity that would have to be considered.

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

```
- ENstructureEN;
val it = fn : ''a list -> EN list list
```

```
- ENstructureEN[1,2,3,1,2,3];
val it = [[],[N],[N,N],[E,N,N],[N,E,N,N],[N,N,E,N,N]] : EN list list
```

ENstructureEN [1, 2, 3, 1, 2, 3]	(1, *i*)	(2, *i*)	(3, *i*)	(4, *i*)	(5, *i*)
□	*N*	–	–	–	–
□	*N*	*N*	–	–	–
□	*E*	*N*	*N*	–	–
□	*N*	*E*	*N*	*N*	–
□	*N*	*N*	*E*	*N*	*N*

N	–	–	–	–
N	*N*	–	–	–
E	*N*	*N*	–	–
N	*E*	*N*	*N*	–
N	*N*	*E*	*N*	*N*

Is the morphogram [1,2,3,1,2,3] a palindrome?
version1: tnf: [1,2,3,1,2,3] -> [3,1,2,3,1,2]
rule4: [3,1,2,3,1,2] -> [1,2,3,1]
rule3: [3,1,2,3] -> [1,2]
rule4: [1,2] -> [∅].

Morphogram [1,2,3,1,2,3]
version2: tnf: [1,2,3,1,2,3] -> [2,1,3,2,1,3]
rule4: [2,1,3,2,1,3] -> [1,3,2,1]
rule1: [1,3,2,1] -> [3,2]
rule4: tnf[3,2] -> [2,1] -> [∅]
[∅].

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

Negative example

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

tnf: [1,2,3,2,1,3] -> [2,1,3,1,2,3]
rule4: [2,1,3,1,2,3] -> [1,3,1,2]
rule4: [1,3,1,2] -> [3,1]
[3,1] : no rule

There is no rule to change adequately [3,1] into an acceptable form, like [2,1], [1,2], [2,3] etc.
Hence, the morphogram [1,2,3,2,1,3] is not a palindrome.

ML-test

```
- ispalindrome[1,2,3,1,1,2];
val it = false : bool
- ispalindrome[1,2,3,1,1,1,1];
val it = false : bool
- ispalindrome [1,2,3,4,1,1,3,4,5,1];
val it = true : bool
- ispalindrome[1,2,3,1,2,3];
val it = true : bool
```

Comparison
The classical rule system for palindromes is given by (S --> ∅, |a|, |b|, aSa, bSb) for the alphabet Σ = {a, b}.

Hence, for each new situation of the complexity of a running system, the classical grammar with its alphabet and rules has to be re-formulated from an external observer of the grammar in dependence of the new alphabet.

For Σ = {a, b, c}
S --> {∅, |a|, |b|, aSa, bSb} ∪ {|c|, cSc}

The morphic rules for palindromes are based on *differentiations*, independent of an alphabet. Hence, the rules for morphic palindromes are covering all morphic palindromes.

"Being a palindrome is a lexical *property rather than a mathematical one*." Dan Dyler

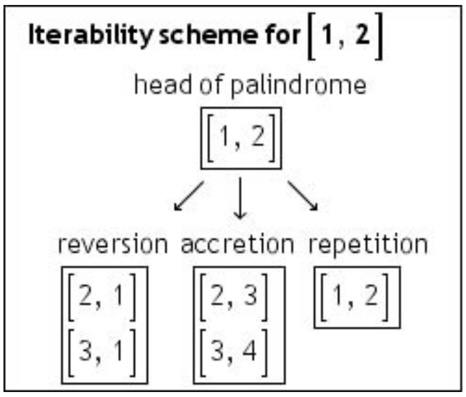
This might be obvious for classical palindromes. It stops to be an ephemeric property of lexicography if morphograms are involved.

1.3.5. Representation of the modi

All modi of iterability of palindromes, i.e. the modi of *repetition*, *reversion* and *accretion*, are representable by the 4 morphogrammatic grammar rules, Rule1-Rule4.

Question: Given a head, are all its palindromic completion derivable?

The *source* of the derivation is not necessarily identical with the head of the palindrome.
Therefore, the palindromes with the head = [1,2] have not to be derivable from its head.
It is sufficient if all palindromes with the head [1,2] are derivable from the sources [1,1] and [1,2].
It is a sligthly different question, how to complete the head of a palindrome to get the complete palindrome.



reversion
P = [1,1]   w3Pw3   [2,1,1,2] -> [1,2,2,1] ; rule3
P = [1,2]   w3Pw3   [3,1,2,3] -> [1,2,3,1] ; rule3

accretion
P = [1,1]   w3Pw4   [2,**1**,**1**,3] -> [1,2,2,3] ; rule4
P = [1,2]   w3Pw4   [3,**1**,**2**,4] -> [1,2,3,4] ; rule4

repetition
P = [1,2]   w2Pw1   [2,**1**,**2**,1] -> [1,2,1,2] ; rule2.

1.3.6. Production by complementation of palindromes

Given the palindrome [1,2,1] as a head, what are the resulting possible palindromes?

Palindromes [1,2,1,x,x,x]

[1,2,1] Q [x_1 x_2 x_3], with x_1= x 3

[1,2,1]   : rule1,1,1 : repetition
[1,3,1]   : rule1,2,1 : repetition
[2,1,2]   : rule3,3,3 : reversion
[3,2,3]   : rule3,1,3 : accretion
[3,4,3]   : rule3,4,3 : accretion

Hence, the head [1,2,1] allows to generate the 5 following palindromes by retro-recursive prolongation with the palindrome grammar rules: [1,2,1,1,2,1],[1,2,1,1,3,1],[1,2,1,2,1,2],[1,2,1,3,2,3], [1,2,1,3,4,3].

Are there applications of the rules on the head = [1,2,1] that are violating palindromicity?

[head] + [empty body] x rules = palindromes

1.3.7. Palindromization of morphograms

Given an *arbitary* morphogram *mg* as a head of a possible palindrome, what are the resulting palindromes?

Repetition : ??
Reversion: [1,2,2] Q x x x

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

1.3.8. DOW palindromes produced by morphoPal grammar

DOW-PAl(3): 112233 12233 121323 123123 123231 123312 123321

P = [1,1] :   [2,**1**,**1**,3] -> [1,2,2,3]   : rule4
[2,2,1,1,3,3] -> [1,1,2,2,3,3] : rule4
[2,1,1,2] -> [1,2,2,1]   : rule3
[3,2,1,1,2,3] -> [1,2,3,3,2,1]   : rule3

```
P = [1,2]
            [2,1,2,1]   ->    [1,2,1,2]        : rule2
          [3,1,2,1,2,3] -> [1,2,3,2,3,1]      : rule3
[1,1,2,2]                                   : rule1
[3,1,1,2,2,3] -> [1,2,2,3,3,1]              : rule3
[3,1,2,3] -> [1,2,3,1]                      : rule3
[2,1,2,3,1,3] -> [1,2,1,3,2,3]              : rule1
[3,1,2,3,1,2] -> [1,2,3,1,2,3]              : rule2
  [2,1,1,3] -> [1,2,2,3]                    : rule4
[3,1,2,2,3,1] -> [1,2,3,3,1,2]              : rule4
```

non-DOW palindromes

```
P = [1,1] :
          [1,1,1,1]   -> [1,1,1,1,1,1]          : rule1
          [2,1,1,3] : rule4  ->  [1,2,1,1,3,1] : rule1
          [2,1,1,2] : rule3  ->  [1,2,1,1,2,1] : rule1 .
```

DOW-palindromes are produced on the base of [1,1] and [1,2], or: just on [1] and [∅], and the rules of the morphoPal grammar.

How has the application of the morphoPal rules to be ruled to produce just the DOW palindromes?

The morphoPal rules are delivering all possible morphic palindromes of a given length. Hence, a *restriction* to the ‘double occurence words’ property has to be applied.

The morphoPal rules are divided in two sets, one is repeating the elements of a production in *direct* or in *inverse* form. The other rules are augmenting the the ‘alphabet’, i.e. the differentiations, of the words by *symmetrical* or *asymmetrical* augmentation (accretion) of the elements of the production.

1.4. Programming aspects

1.4.1. Recursive programming of the morphoRules

Programming classical palindromes is straight forwards, easy to access and realized in all programming languages.

http://rosettacode.org/wiki/Palindrome_detection

In general there are 2 approaches to consider:
1. The non-recursive and
2. The recursive approach.

The non-recursive works with the construct “reverse”, the recursive works over the constructs “head” and “last” of a string.

For the *morphogrammatic* approach, the descriptive approach has to completed by
a) reversion
b) repetition and
c) accretion.

The (retro-)recursive morphogrammatic approach has to deal additionally with the concept of trito-normal form, tnf, also called in other contexts an “OrderedCollection” or a “relabeling by ascending order” and, more important, the *variability* of the head (first) and last (tail) function for strings.

This variability is ruled by the **morphoRules** of the grammar for morphic palindromes.

Warning
The following tables had been manually produced on the base of normed (canonized) palindromes in trito-normal form, tnf, as it is used in the ML implementation.

The Scala program for the recursive production of palindromes, *MorphoGrammar*, is not yet accepting this approach. It is based purely, as it is defined, on non-canonized palindromes.

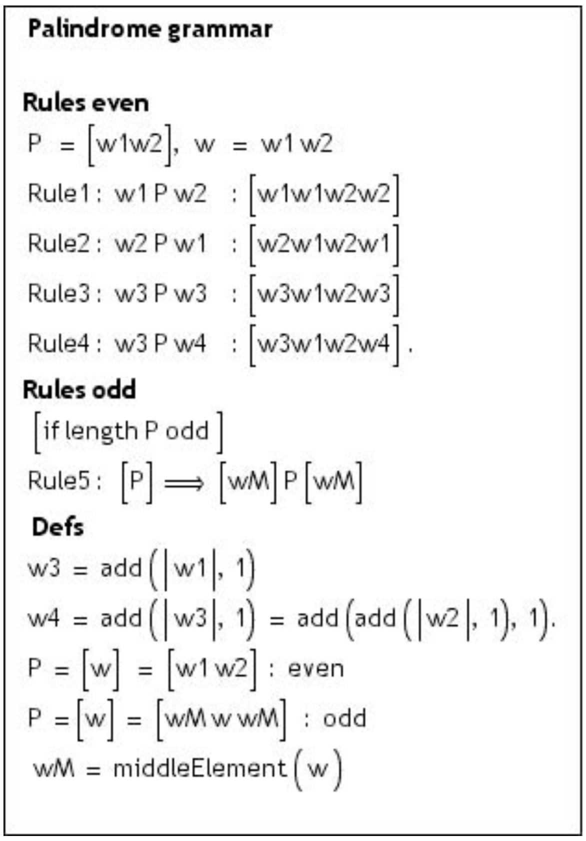
Hence, a morphogram [1,2,3] is not accepted as a palindrome by the MorphoGrammar program. Written as the list (1,2,3) it is not recognized as a morphogram that is written as [1,2,3].

```
scala> isPalindrome2(List(1,2,3))
res17: Boolean = false
```

With the list written in the form as it is produced, i.e. as (2,1,3) or (3,1,2), the morphogram [1,2,3] is accepted by the MorphoGrammar as a palindrome.

```
scala> isPalindrome2(List(2,1,3))
res2: Boolean = true
```

Hence, the approach of the tables is some kind of *zigzagging* between produced and normed palindromes.



Palindrome grammar

Rules even

$P = [w1w2],\ w = w1\,w2$

$\text{Rule1}:\ w1\,P\,w2 \ : [w1w1w2w2]$

$\text{Rule2}:\ w2\,P\,w1 \ : [w2w1w2w1]$

$\text{Rule3}:\ w3\,P\,w3 \ : [w3w1w2w3]$

$\text{Rule4}:\ w3\,P\,w4 \ : [w3w1w2w4].$

Rules odd

$[\text{if length P odd}]$

$\text{Rule5}:\ [P] \Longrightarrow [wM]\,P\,[wM]$

Defs

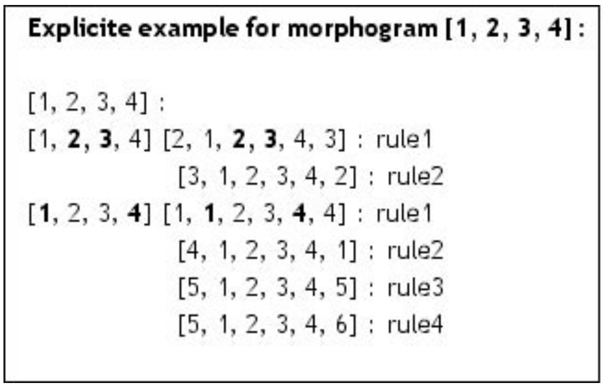
$w3 = \text{add}(|w1|, 1)$

$w4 = \text{add}(|w3|, 1) = \text{add}(\text{add}(|w2|, 1), 1).$

$P = [w] = [w1\,w2] : \text{even}$

$P = [w] = [wM\,w\,wM] : \text{odd}$

$wM = \text{middleElement}(w)$



Explicite example for morphogram [1, 2, 3, 4] :

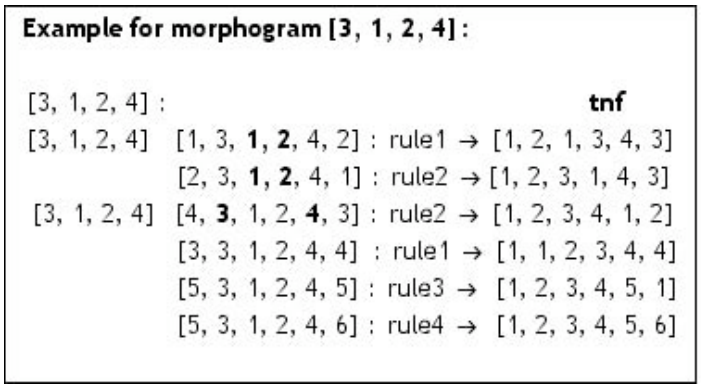
[1, 2, 3, 4] :
[1, **2**, **3**, 4] [2, 1, **2**, **3**, 4, 3] : rule1
[3, 1, 2, 3, 4, 2] : rule2
[**1**, 2, 3, **4**] [1, **1**, 2, 3, **4**, 4] : rule1
[4, 1, 2, 3, 4, 1] : rule2
[5, 1, 2, 3, 4, 5] : rule3
[5, 1, 2, 3, 4, 6] : rule4

1.4.2. Comments on the implementations

Context rules vs. implementation

The *intuitive* and *manual* use of the rules follows some *context rules* (conditions)that are not yet implementet in the proposed grammar and the production program.

Recursion on words is not yet running round with the Scala grammar and program ’genPalindrome”.
Additionally, a strict 'mechanical' application of the rules on the length of the 'words' is producing automatically some redundancy.
On the other hand, the manual application of the rules is delivering correctly the list of palindromes. Obviously, some intuitive properties are not yet formalized.
This is discussed with the following example.



Example for morphogram [3, 1, 2, 4] :

[3, 1, 2, 4] : **tnf**
[3, 1, 2, 4] [1, 3, **1**, **2**, 4, 2] : rule1 → [1, 2, 1, 3, 4, 3]
[2, 3, **1**, **2**, 4, 1] : rule2 → [1, 2, 3, 1, 4, 3]
[3, 1, 2, 4] [4, **3**, 1, 2, **4**, 3] : rule2 → [1, 2, 3, 4, 1, 2]
[3, 3, 1, 2, 4, 4] : rule1 → [1, 1, 2, 3, 4, 4]
[5, 3, 1, 2, 4, 5] : rule3 → [1, 2, 3, 4, 5, 1]
[5, 3, 1, 2, 4, 6] : rule4 → [1, 2, 3, 4, 5, 6]

exclusion:
[3, **1**, **2**, 4] → [**3**, 3, **1**, **2**, 4, **3**]: rule3, because of **fst=scnd** but not **last$_{n-1}$ = last$_n$** like with [1, 2, 2, 1] | [1, 1, 2, 2, 1, 1] | rule1 .
Hence, rule3(1,2) is not applicable to [3, **1**, **2**, 4].
An application produces non-palindromic morphograms.

redundancy:
[3, **1**, **2**, 4] → [**3**, 3, **1**, **2**, 4, **4**]: rule4 on (1,2) is applicable. But the result is also produced by rule1 on (3,4):
[**3**, 1, 2, **4**] → [**3**, 3, **1**, **2**, 4, **4**]: rule1 on (3,4).
Hence, the application rule4 on (1,2) can be omitted.

Example for redundancy

[1, **1**, **1**, 1]	[1, 1, 1, 1, 1, 1]	rule1(w1w2)
[**1**, 1, 1, **1**]	[1, 1, 1, 1, 1, 1]	rule1(w3w4)
[1, **1**, **1**, 1]	[2, 1, 1, 1, 1, 2]	rule3(w1w2)
[**1**, 1, 1, **1**]	[2, 1, 1, 1, 1, 2]	rule3(w3w4)
[1, **1**, **1**, 1]	[2, 1, 1, 1, 1, 3]	rule4(w1w2)
[**1**, 1, 1, **1**]	[2, 1, 1, 1, 1, 3]	rule4(w3w4)
[1, **1**, **1**, 1]	[3, 1, 1, 1, 1, 2]	rule4 " (w1w2)
[**1**, 1, 1, **1**]	[3, 1, 1, 1, 1, 2]	rule4 "(w3w4)

With rule4[1, 1, 1, 1] = rule4 "[1, 1, 1, 1]

limitation:

The implemented rules are not yet covering recursively all correct applications on a word (morphogram).

The present implementation is not yet considering cases like:

[1, 2] -- -> [3, **1**, **2**, 3] -- ->

[**1**, 3, 1, 2, 3, **2**]
[**2**, 3, 1, 2, 3, **1**]

: missing.

Manual examples

typ::odd	start	result : odd	rules	tnf
odd(1)	[1]	[1, 1, 1]	rule1	[1, 1, 1]
–	–	[2, 1, 2]	rule3	[1, 2, 1]
–	–	[2, 1, 3]	rule4	[1, 2, 3]
odd(3)	[1, 1, 1]	[1, 1, 1, 1, 1]	rule1	[1, 1, 1, 1, 1]
–	–	[2, 1, 1, 1, 2]	rule3	[1, 2, 2, 2, 1]
–	–	[2, 1, 1, 1, 3]	rule4	[1, 2, 2, 2, 3]
–	[1, 2, 1]	[1, 1, 2, 1, 1]	rule1	[1, 1, 2, 1, 1]
–	rule5	[2, **1**, 2, **1**, 2	rule5(2)	[1, 2, 1, 2, 1]
–	–	[3, 1, **2**, 1, 3]	rule3	[1, 2, 3, 2, 1]
–	–	[3, 1, 2, 1, 4]	rule4	[1, 2, 3, 2, 4]
–	[1, 2, 3]	[1, 1, 2, 3, 3]	rule1	[1, 1, 2, 3, 3]
–	–	[3, 1, 2, 3, 1]	rule2	[1, 2, 3, 1, 2]
–	rule5	[2, 1, **2**, 3, 2]	rule5(2)	[1, 2, 1, 3, 1]
–	–	[4, 1, 2, 3, 4]	rule3	[1, 2, 3, 4, 1]
–	–	[4, 1, 2, 3, 5]	rule4	[1, 2, 3, 4, 5]

ML-Test odd(3): sum(pal(5)) = 12
[1,1,1,1,1], [1,1,2,1,1],[1,1,2,3,3],[1,2,1,2,1],[1,2,1,3,1],
[1,2,2,2,1],[1,2,2,2,3],[1,2,3,1,2],[1,2,3,2,1],[1,2,3,2,4],
[1,2,3,4,1],[1,2,3,4,5].

```
ML : val it = 59 : int , corrected
```

typ::odd	**start : Σ = 12**	**result : odd(7)**	**rules**	**tnf, Σ = 59**
odd(5)	[1, 1, 1, 1, 1]	[1, 1, 1, 1, 1, 1, 1]	rule1	[1, 1, 1, 1, 1, 1, 1]
–	–	[2, 1, 1, 1, 1, 1, 2]	rule3	[1, 2, 2, 2, 2, 2, 1]
–	–	[2, 1, 1, 1, 1, 1, 3]	rule4	[1, 2, 2, 2, 2, 2, 3]
–	[1, 2, 2, 2, 1]	[1, 1, 2, 2, 2, 1, 1]	rule1	[1, 1, 2, 2, 2, 1, 1]
–	–	[2, 1, 2, **2**, 2, 1, 2]	rule5(2)	[1, 2, 1, 1, 1, 2, 1]
–	–	[3, 1, **2**, 2, **2**, 1, 3]	rule3(2, 2)	[1, 2, 3, 3, 3, 2, 1]
–	–	[3, 1, 2, 2, 2, 1, 4]	rule4	[1, 2, 3, 3, 3, 2, 4]
–	[1, 2, 2, 2, 3]	[1, 1, 2, 2, 2, 3, 3]	rule1	[1, 1, 2, 2, 2, 3, 3]
–	–	[3, 1, 2, 2, 2, 3, 1]	rule2	[1, 2, 3, 3, 3, 1, 2]
–	–	[2, 1, 2, 2, 2, 3, 2]	rule3	[1, 2, 1, 1, 1, 3, 1]
–	–	[4, 1, 2, 2, 2, 3, 4]	rule3	[1, 2, 3, 3, 3, 4, 1]
–	–	[4, 1, 2, 2, 2, 3, 5]	rule4	[1, 2, 2, 3, 2, 2, 4]
□	[1, 1, 2, 1, 1]	[1, 1, 1, 2, 1, 1, 1]	□	[1, 1, 1, 2, 1, 1, 1]
□	□	[2, 1, 1, 2, 1, 1, 2]	□	[1, 2, 2, 1, 2, 2, 1]
□	□	[3, 1, 1, 2, 1, 1, 3]	□	[1, 2, 2, 3, 2, 2, 1]
□	□	[3, 1, 1, 2, 1, 1, 4]	□	[1, 2, 2, 3, 2, 2, 4]
□	[2, 1, 2, 1, 2]	[1, 2, 1, 2, 1, 2, 1]	r1	[1, 2, 1, 2, 1, 2, 1]
□	□	[2, 2, 1, 2, 1, 2, 2]	r5	[1, 1, 2, 1, 2, 1, 1]
□	□	[3, 2, 1, 2, 1, 2, 3]	r3	[1, 2, 3, 2, 3, 2, 1]
□	□	[3, 2, 1, 2, 1, 2, 4]	r4	[1, 2, 3, 2, 3, 2, 4]
□	[3, 1, 2, 1, 3]	[1, 3, 1, 2, 1, 3, 1]	□	[1, 2, 1, 3, 1, 2, 1]
□	□	[2, 3, 1, 2, 1, 3, 2]	□	[1, 2, 3, 1, 2, 3, 1]
□	□	[3, 3, 1, 2, 1, 3, 3] [4, 3, 1, 2, 1, 3, 4]	r3(3, 3)	[1, 1, 2, 3, 2, 1, 1] **[1, 2, 3, 4, 3, 2, 1]**
□	□	[4, 3, 1, 2, 1, 3, 5]	□	[1, 2, 3, 4, 3, 2, 5]
□	[3, 1, 2, 1, 4]	[1, 3, 1, 2, 1, 4, 1]	r1(1, 1)	[1, 2, 1, 3, 1, 4, 1]
□	□	[2, 3, 1, 2, 1, 4, 2]	r5(2)	[1, 2, 3, 1, 3, 4, 1]
□	□	[3, 3, 1, 2, 1, 4, 4]	r1(3, 4)	[1, 1, 2, 3, 2, 4, 4]

□	□	[4, 3, 1, 2, 1, 4, 3]
		[5, 3, 1, 2, 1, 4, 5]
			r2(3, 4)
			r3(3, 4)
				[1, 2, 3, 4, 3, 1, 2]
				[1, 2, 3, 4, 3, 5, 1]

□	□	[5, 3, 1, 2, 1, 4, 6]	r4	[1, 2, 3, 4, 3, 5, 6]
□	[1, 1, 2, 3, 3]	[2, 1, 1, 2, 3, 3, 2]	□	[1, 2, 2, 1, 3, 3, 1]
□	□	[1, 1, 1, 2, 3, 3, 3]	□	[1, 1, 1, 2, 3, 3, 3]
□	□	[3, 1, 1, 2, 3, 3, 1]	□	[1, 2, 2, 3, 1, 1, 2]
□	□	[4, 1, 1, 2, 3, 3, 4]	□	[1, 2, 2, 3, 4, 4, 1]
□	□	[4, 1, 1, 2, 3, 3, 5]	□	[1, 2, 2, 3, 4, 4, 5]
□	[3, 1, 2, 3, 1]	[2, 3, 1, 2, 3, 1, 2]	□	[1, 2, 3, 1, 2, 3, 1]
□	□	[3, 3, 1, 2, 3, 1, 1]	□	[1, 1, 2, 3, 1, 2, 2]
□	□	[1, 3, 1, 2, 3, 1, 3]	□	[1, 2, 1, 3, 2, 1, 2]
□	□	[4, 3, 1, 2, 3, 1, 4]	□	[1, 2, 3, 4, 2, 3, 1]
□	□	[4, 3, 1, 2, 3, 1, 5]	□	[1, 2, 3, 4, 2, 3, 5]
□	[2, 1, 2, 3, 2]	[1, 2, **1**, 2, **3**, 2, 3]	r1	[1, 2, 1, 2, 3, 2, 3]
□	□	[3, 2, **1**, 2, **3**, 2, 1]	r2	[1, 2, 3, 2, 1, 2, 3]
□	□	[2, 2, 1, **2**, 3, 2, 2]	r5	[1, 1, 2, 1, 3, 1, 1]
□	□	[4, **2**, 1, 2, 3, **2**, 4]	r3	[1, 2, 3, 2, 4, 2, 1]

ML, Palindromes odd (7), sum 59

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

Number of direct successors

How many direct successors of *a* palindrome can be generated by the 4 production rules?

For *a* palindrome of length 5, there are maximal 7 = 1 + 2 + 2 + 1 + 1 direct successors. This corresponds to

type::even	**start, 7**	**result : even(6)**	**rules**	**tnf, 31**
even(4)	[1, 1, 1, 1]	[1, 1, 1, 1, 1, 1]	rule1	[1, 1, 1, 1, 1, 1]
–	–	[2, 1, 1, 1, 1, 2]	rule3	[1, 2, 2, 2, 2, 1]
–	–	[2, 1, 1, 1, 1, 3]]	rule4	[1, 2, 2, 2, 2, 3]
–	[1, 2, 2, 1]	[1, 1, 2, 2, 1, 1]	rule1	[1, 1, 2, 2, 1, 1]
–	–	[2, 1, 2, 2, 1, 2]	rule3	[1, 2, 1, 1, 2, 1]
–	–	[3, 1, 2, 2, 1, 3]	rule3	[1, 2, 3, 3, 2, 1]
–	–	[3, 1, 2, 2, 1, 4]	rule4	[1, 2, 3, 3, 4, 1]
–	[1, 2, 2, 3]	[1, 1, 2, 2, 3, 3]	rule1	[1, 1, 2, 2, 3, 3]
–	–	[3, 1, 2, 2, 3, 1]]	rule2	[1, 2, 3, 3, 1, 2]
–	–	[4, 1, 2, 2, 3, 5]	rule4	[1, 2, 3, 3, 4, 5]
–	–	[2, 1, 2, 2, 3, 2]	rule1	[1, 2, 1, 1, 3, 1]
–	–	[4, 1, 2, 2, 3, 4]	rule3	[1, 2, 3, 3, 4, 1]
–	[1, 1, 2, 2]	[1, 1, 1, 2, 2, 2]	rule1	[1, 1, 1, 2, 2, 2]
–	–	[2, 1, 1, 2, 2, 1]	rule2	[1, 2, 2, 1, 1, 2]
–	–	[3, 1, 1, 2, 2, 4]	rule4	[1, 2, 2, 3, 3, 1]
–	–	[3, 1, 1, 2, 2, 3]	rule3	[1, 2, 2, 3, 3, 1]
–	[1, 2, 1, 2]	[1, 1, 2, 1, 2, 2]	rule1	[1, 1, 2, 1, 2, 2]
–	–	[2, 1, 2, 1, 2, 1]	rule2	[1, 2, 1, 2, 1, 2]
–	–	[3, 1, 2, 1, 2, 4]	rule4	[1, 2, 3, 2, 3, 4]
–	–	[3, 1, 2, 1, 2, 3]	rule3	[1, 2, 3, 2, 3, 1]
–	[1, 2, 3, 1]	[1, 1, 2, 3, 1, 1]	rule1	[1, 1, 2, 3, 1, 1]
–	–	[4, 1, 2, 3, 1, 4]	rule3	[1, 2, 3, 4, 2, 1]
–	–	[4, 1, 2, 3, 1, 5]	rule4	[1, 2, 3, 4, 2, 5]
–	–	[2, 1, 2, 3, 1, 3]	rule1	[1, 2, 1, 3, 2, 3]
–	–	[3, 1, 2, 3, 1, 2]	rule2	[1, 2, 3, 1, 2, 3]
–	[1, 2, 3, 4]	[1, 1, 2, 3, 4, 4]	rule1	[1, 1, 2, 3, 4, 4]
–	–	[4, 1, 2, 3, 4, 1]	rule2	[1, 2, 3, 4, 1, 2]
–	–	[2, 1, 2, 3, 4, 3]	rule1	[1, 2, 1, 3, 4, 3]
–	–	[3, 1, 2, 3, 4, 2]	rule2	[1, 2, 3, 1, 4, 3]
–	–	[5, 1, 2, 3, 4, 6]	rule4	[1, 2, 3, 4, 5, 6]
–	–	[5, 1, 2, 3, 4, 5]	rule3	[1, 2, 3, 4, 5, 1]

ML – Test : even(6) : 31 : OK

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

Scala – Test : even(6) : 37

OK, minus 6 morphograms not in CR :
[2, 2, 1, 1, 2, 3], [2, 2, 1, 1, 3, 2], [2, 2, 1, 2, 1, 2], [2, 2, 1, 2, 1, 3], [3, 3, 1, 2, 3, 4], [3, 3, 1, 2, 4, 3]

Reminder: PALINDROME is not a regular language.

"In the automata theory, a set of all palindromes in a given alphabet is a typical example of a language that is *context-free*, but *not regular*. This means that it is, in theory, impossible for a computer with a finite amount of memory to reliably test for palindromes.

In addition, the set of palindromes may not be reliably tested by a deterministic pushdown automaton which also means that they are not LR(k)-parsable or LL(k)-parsable. When reading a palindrome from left-to-right, it is, in essence, impossible to locate the "*middle*" until the entire word has been read completely." (WiKi)

DOW palindromes

Restrictions for *P* = [*i*, *i*]

DOW Palindrome grammar

```
P  = [w1w2],  w  = w1 w2
Rule1 :  P ≠ [i, i]  : w1 P w2  : [w1w1w2w2]
Rule2 :  w2 P w1   : [w2w1w2w1]
Rule3 :  w3 P w3    : [w3w1w2w3]
Rule4 :  w3 P w4   : [w3w1w2w4] .
```
Defs
```
w3 = add (|w1|, 1)
w4 = add (|w3|, 1) = add (add (|w2|, 1), 1).
```

DOW palindromes

DOW – palin(6)

type::even	start	result : even(6)	rules	tnf
even(4)	[1, 2, 2, 1]	[3, 1, 2, 2, 1, 3]	rule3	[1, 2, 3, 3, 2, 1]
–	#[1, 2, 2, 3]	[1, 1, 2, 2, 3, 3]	rule1	[1, 1, 2, 2, 3, 3]
–	–	[3, 1, 2, 2, 3, 1]	rule2	[1, 2, 3, 3, 1, 2]
–	[1, 1, 2, 2]	[3, 1, 1, 2, 2, 3]	rule3	[1, 2, 2, 3, 3, 1]
–	[1, 2, 1, 2]	[3, 1, 2, 1, 2, 3]	rule3	[1, 2, 3, 2, 3, 1]
–	#[1, 2, 3, 1]	[2, 1, 2, 3, 1, 3]	rule1	[1, 2, 1, 3, 2, 3]
–	–	[3, 1, 2, 3, 1, 2]	rule2	[1, 2, 3, 1, 2, 3]
–	#[1, 2, 3, 4]	–	–	–

```
– ENstructureEN[1, 2, 3, 3, 2, 1];
val it = [[], [N], [N, N], [N, N, E], [N, E, N, N], [E, N, N, N, N]]
– ENstructureEN[1, 1, 2, 2, 3, 3];
val it = [[], [E], [N, N], [N, N, E], [N, N, N, N], [N, N, N, N, E]]
– ENstructureEN[1, 2, 3, 3, 1, 2];
val it = [[], [N], [N, N], [N, N, E], [E, N, N, N], [N, E, N, N, N]]
– ENstructureEN[1, 2, 2, 3, 3, 1];
val it = [[], [N], [N, E], [N, N, N], [N, N, N, E], [E, N, N, N, N]]
– ENstructureEN[1, 2, 1, 3, 2, 3];
val it = [[], [N], [E, N], [N, N, N], [N, E, N, N], [N, N, N, E, N]]
– ENstructureEN[1, 2, 3, 1, 2, 3];
val it = [[], [N], [N, N], [E, N, N], [N, E, N, N], [N, N, E, N, N]]
```

type::DOW	start : DOW(6)	result : DOW(8)	rules	tnf
even(6)	[3, 1, 2, 2, 1, 3]	[4, 3, 1, 2, 2, 1, 3, 4]	rule3	[1, 2, 3, 4, 4, 3, 2, 1]
–	[1, 1, 2, 2, 3, 3]	[4, 1, 1, 2, 2, 3, 3, 4]	rule1	[1, 2, 2, 3, 3, 4, 4, 1]
–	[3, 1, 2, 2, 3, 1]	[4, 3, 1, 2, 2, 3, 1, 4]	rule2	[1, 2, 3, 4, 4, 2, 3, 1]
–	[3, 1, 1, 2, 2, 3]	[4, 3, 1, 1, 2, 2, 3, 4]	rule3	[1, 2, 3, 3, 4, 4, 2, 1]
–	[3, 1, 2, 1, 2, 3]	[4, 3, 1, 2, 1, 2, 3, 4]	rule3	[1, 2, 3, 4, 3, 4, 2, 1]
–	[2, 1, 2, 3, 1, 3]	[4, 2, 1, 2, 3, 1, 3, 4]	rule3	[1, 2, 3, 2, 4, 3, 4, 1]
–	[3, 1, 2, 3, 1, 2]	[4, 3, 1, 2, 3, 1, 2, 4]	rule3	[1, 2, 3, 4, 2, 3, 4, 1]