— vordenker-archive —

Rudolf Kaehr
(1942-2016)

Title

Towards Programming Morphic Palindromes

A first presentation of morphic grammers and programs for palindromes

Archive-Number / Categories

3_28 / K09, K08, K10, K12

Publication Date

2013

Keywords / Topics

Programming Aspects of Palindromes – See also: 3_27

Disciplines

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

Abstract

Morphic palindromes had been introduced and studied on two levels of representation:

1. the alpha-numeric level with tuples or lists, esp. of integers,
2. the differentiational approach of the ε/v-structure.

The tuple approach got some specification by a grammar and by a production program written in Scala.

The production system for morphic palindromes is characterized a) by the morphic production rules and b) by the context rule that refers backwards to the produced palindrome and is therefore adding to the recursive production rules an aspect of retro-grade recursion.

The tuple approach is divided into a conceptual and a recursive formulation and formalization.

The conceptual thematization is considering palindromes under the operations of reversion, repetition and accretion, This is augmenting the classical definition of palindromes that are characterized by reversion (and repetition)by the operation of accretion.

The relabeling concept for DOW palindromes is analyzed.

The ε/v-approach that is emphasizing the differential structure of a morphogram and its morphic palindromes by equality or non-equality (ε/v, or E/N) specifies palindromes independently of the elements of the tuple approach.

The ε/v-approach allows different presentations of morphic palindromes.

The matrix or table presentation of the ε/v-approach enables to proof that a morphogram is a palindrome iff its matrix is symmetric. This might be seen as a fundamental theorem for palindromes in general.

Furthermore, a new form of duality for ε/v-palindromes is introduced.

Connections with graph-theoretic analysis are sketched.

Citation Information / How to cite

Rudolf Kaehr: "Towards Programming Morphic Palindromes", www.vordenker.de (Sommer Edition, 2017) J. Paul (Ed.), http://www.vordenker.de/rk/rk_Towards-Programming-Morphic-Palindromes_2013.pdf

Towards Programming morphic Palindromes

A first presentation of morphic grammars and programs for palindromes

Rudolf Kaehr Dr.phil[@]

 ISSN 2041-4358

Abstract

Morphic palindromes had been introduced and studied on two levels of representation:
1. the alpha-numeric level with tuples or lists, esp. of integers,
2. the differentiational approach of the ε/ν-structure.

The tuple approach got some specification by a grammar and by a production program written in Scala.
The production system for morphic palindromes is characterized a) by the morphic production rules and b) by the context rule that refers backwards to the produced palindrome and is therefore adding to the recursive production rules an aspect of retro-grade recursion.

The tuple approach is divided into a conceptual and a recursive formulation and formalization.
The conceptual thematization is considering palindromes under the operations of reversion, repetition and accretion, This is augmenting the classical definition of palindromes that are characterized by reversion (and repetition)by the operation of accretion.

The relabeling concept for DOW palindromes analyzed.

The ε/ν-approach that is emphasizing the differential structure of a morphogram and its morphic palindromes by equality or non-equality (ε/ν, or E/N) specifies palindromes independently of the elements of the tuple approach.

The ε/ν-approach allows different presentations of morphic palindromes.
The matrix or table presentation of the ε/ν-approach enables to proof that a morphogram is a palindrome iff its matrix is symmetric. This might be seen as a fundamental theorem for palindromes in general.

Furthermore, a new form of duality for ε/ν-palindromes is introduced.
Connections with graph-theoretic analysis are sketched.
(work in progress, vers. 0.5, Aug 2013)

1. Programming Aspects of Palindromes

1.1. Motivations for asymmetric palindromes

"But Warren S. McCulloch has already explained in his work from 1945, which we referred to in the beginning, that the knowing and wanting neural network of the brain works not only with hierarchy*, but also with a* heterarchical *functionality of logical values. This requires - as we have emphasized throughout the course of this analysis - an additional language-axis for the description of consciousness. While along the hierarchical axis the beginning value can no longer change, for the heterarchical axis the same value can never be repeated.*

"The possibility of self-reflection *in computable systems thereby suggests itself. In opposition to hierarchy, which in the movement from concept to concept never allows a self-relation and which if traveled in the opposite direction only undoes the achieved result, the heterarchical axis produces an increasing number of circles of backwards referential Being, provided that the number of values is continuously increased."* (G. Gunther, trnsl. J. Paul, J. Newbury)

http://www.vordenker.de/ggphilosophy/gg_identity-neg-language_biling.pdf

This note gives the first grammar for asymmetric palindromes as they had been introduced in previous papers.

Palindromes in DNA research
Again, palindromes in the field of DNA research are, in a philosophical sense, '*object*'-oriented. They are focused on the relational **structures** between objects, elements, entities of identifiable and measurable 'observables' (R. Rosen).

Morphic palindromes are not focused on objects, they are, in a philosophical sense, '*subject*'-oriented, studying the dynamics, processuality, evocation of events, i.e. of **structurations**.

Hence, morphic palindromes are thematizing the *praxa* of DNA events. And are therefore *praxeological* (or praxeo-grammatical)and not in any way informational or in-formational.

Why are 'asymmetric' palindromes of importance?

Everybody knows the famous palindromes in phonetic writing systems.

The simplest western example is the name "*anna*". It reads forwards and backwards the same and it has for both ways of reading the same meaning.

There are competitions about the longest palindrome, and there are even novels written as a palindrome.

But again, their meaning is invariant of the reading direction.

Therefore they are sometimes called symmetric palindromes. But in fact, all palindromes are symmetric.

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 $\sum$ = {a, b}.
Context Free Grammar (CFG) production rules for palindromes:

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

Rules :
1. S ⟶ aSa
2. S ⟶ bSb
3. S ⟶ ε | a | b

Inductive definition
Basis: ∅, 0 and 1 are palindromes
Induction: If *w* is a palindrome, so are 0*w*0 and 1*w*1. No string is palindrome of 0 and 1, unless it follows from this basis and inductive rule.

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

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

As a formal system

- The language of *formulae* *words*
- The set of *axioms* (or assumptions) *a-z,ϵ*

- The language of *proofs*

$$(\text{ax})\ \frac{\text{P is an axiom}}{\text{P is a palindrome}}$$

$$(\text{ext})\ \frac{\text{P is a palindrome}}{\text{x P x is a palindrome}}$$

$$(\text{concat})\ \frac{\text{P is a palindrome}\ \ \text{Q is a palindrome}}{\text{Q P Q is a palindrome}}$$

- *Theorems* are formulae that have proofs.

http://tel.archives-ouvertes.fr/docs/00/67/26/99/ANNEX/slides.pdf

Now, what is an example for an **asymmetric palindrome**?

I don't know a single asymmetric palindrome in a linguistic or numeric version of what ever length and elaboration.

Chinese palindrome

友朋小吃 （you meng xiaochi : a snack bar named *You-Peng*

吃小朋友 （chi xiao pengyou ：*“Eat little kids”*

http://blog.chinesehour.com/

Hence, this very short palindrome is asymmetric in its meaning, albeit it is scripturally symmetric: 友朋小吃 吃小朋友.

And, again, I don't know of a single Western example of this kind of palindromes.

It might be argued that Martin Gardners “*Semordnilap* is a name coined for a word or phrase that spells a different word or phrase backward. “Semordnilap" is itself "palindromes" spelled backward.” is contradicting my last statement.

But as it is introduced, it is a specially coined word for such a purpose. Obviously, there are no limits to introduce such artificial words and constructs. The meaning of "semordnilap” is its inverse and it seems to have no own meaning.

Obviously, there are a few examples from natural languages too, like “deserts” that reads as “stressed” and represents a different meaning. But these are nevertheless rare and arbitrary exceptions with highly restricted length. And there is no genuine linguistic-semantic base for “asymmetric” palindromes in the linearity of Western languages.

Now, I introduced the concept of asymmetric palindromes of arbitrary length that are neither linguistic nor numeric or pictographic.

A simple example is the name "**Annabelle**". Taken as a name, it isn't palindromic at all.

Funny enough, it consists of 3 palindromes: "*anna*, *b*” and "*elle*". But as a composition it isn't a palindrome.

Taken as a pattern of *differentiations* it is a palindrome. It reads forwards and backwards as the same.

OK, it is an asymmetric palindrome which reads the same independently of the reading direction albeit it is inscribed differently.

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

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

The **differential** approach is not counting on the elements of the string but on the *differences* between the elements.

The function **ENstructure** is calculating those differences in respect of their *locations*.

```
- ENstructure[1,2,2,1,3,4,5,5,4];
val it =
  [[],[(1,2,N)],
   [(1,3,N),(2,3,E)],
   [(1,4,E),(2,4,N),(3,4,N)],
   [(1,5,N),(2,5,N),(3,5,N),(4,5,N)],
   [(1,6,N),(2,6,N),(3,6,N),(4,6,N),(5,6,N)],
   [(1,7,N),(2,7,N),(3,7,N),(4,7,N),(5,7,N),(6,7,N)],
   [(1,8,N),(2,8,N),(3,8,N),(4,8,N),(5,8,N),(6,8,N),(7,8,E)],
   [(1,9,N),(2,9,N),(3,9,N),(4,9,N),(5,9,N),(6,9,E),(7,9,N),(8,9,N)]]
  : (int * int * EN) list list
```

It is easy to see that the differentiational equation holds:

```
ENstructure[1,2,2,1,3,4,5,5,4] = ENstructure[4,5,5,3,1,2,2,1]
```

Therefore, the morphogram [1,2,2,1,3,4,5,5,4] is a morphic palindrome.

ENstructureEN

The ENstructureEN function summarizes the pattern of ENstructure by omitting the explicite numeric notation for the loci of the differentiations:

```
- ENstructureEN[1,2,2,1,3,4,5,5,4];
val it =
[[],[N],
[N,E],
[E,N,N],
[N,N,N,N],
[N,N,N,N,N],
[N,N,N,N,N,N],
[N,N,N,N,N,N,E],
[N,N,N,N,N,E,N,N]] : EN list list
```

ML ENstructureEN

```
fun ENstructureEN z =
  map (fn trl => map (fn pair => deltaEN pair z)
                     trl)
      (pairstructure (length z));

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

This is nicely tabularized with the table of the EN-pattern:

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

From a *symbolic* point of view the morphogram [1,2,2,1,3,4,5,5,4] is obviously an asymmetric *list*.

On the other hand, the EN-table of the morphic and asymmetric palindrome [1,2,2,1,3,4,5,5,4] shows clearily its hidden *symmetry*.

That's the reason why such differentiational, i.e. morphogrammatic palindromes are called,

paradoxically, *asymmetric* palindromes. They are asymmetric on a semiotical (symbolic) or mathematical level but symmetric on the level of morphogrames and their EN-structure.

Using a purely formal mathematical approach, we also might say: *Morphic palindromes are invariant under relabeling.*

It turned out that this relabeling approach is purely combinatorial and is not telling much about the character of palindromes.
A later and more profund formulation will be:

A morphogram is a palindrome iff its EN – table is symmetric.

Diagrams of palindromes
A well known palindrome in the DOW literature, i.e. the study of *double occurrence words* under relabeling, is the sequence [1,1,2,2,3,3].

Thematized as a differential structure, the palindrome gets its E/N-structure.

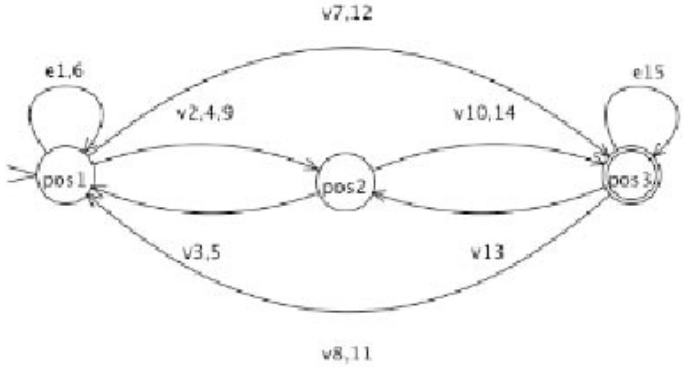
- ENstructure [1,1,2,2,3,3];
val it =
[[],
[(1,2,E)],
[(1,3,N),(2,3,N)],
[(1,4,N),(2,4,N),(3,4,E)],
[(1,5,N),(2,5,N),(3,5,N),(4,5,N)],
[(1,6,N),(2,6,N),(3,6,N),(4,6,N),(5,6,E)]] : (int * int * EN) list list

A simple palindrome test confirms its palindromicity. That is,
ENstructure [1,1,2,2,3,3] = Enstructure[3,3,2,2,1,1].

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

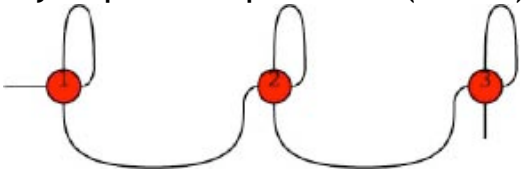
The differential diagram is given by the DiagrMorphoPalin:

DiagrMorphoPalin [1,1,2,2,3,3]



The classical interpretation is depicted by the following diagram.

Assembly Graph for the palindrome (112233)



More at:
http://memristors.memristics.com/Morphospheres/Asymmetric%20Palindromes.html
http://the-chinese-challenge.blogspot.co.uk/2012/12/morphospheres-asymmetric-

palindromes.html

1.2. Towards a formal grammar for morphic palindromes

Programming classical palindromes is straight forwards, easy to access and realized in all programming languages.
http://rosettacode.org/wiki/Palindrome_detection

Morphic Palindromes
http://memristors.memristics.com/Formal%20Aspects/Formal%20Aspects.html
http://memristors.memristics.com/Formal%20Aspects/Formal%20Aspects.pdf

In general
There are 2 approaches for programming to consider:
1. The *non-recursive* and
2. The *recursive* approach.

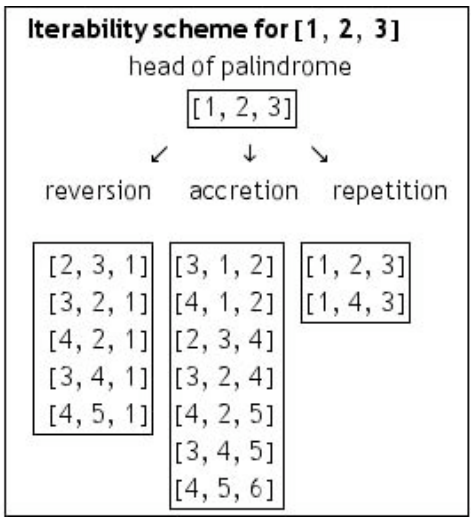
The non-recursive approach works with the construct "*reverse*", the recursive approach works over the constructs "*head*" and "*last*" of a list.

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

The non-recursive approach answers the question: *How to produce a palindrome out from a given "head"*. This head might be itself a palindrome or not.

From the palindromic head [1,2,3] the bodies of the possible palindromes are produced by the rules of *reversion*, *repetition* and *accretion*.

Those rules are here not yet given explicitly. An example shows how they work.



The scheme covers correctly the list of morphic palindromes with head [1,2,3]:

Palindromes : valit =
{{1, 2, 3, 1, 2, 3}, {1, 2, 3, 2, 3, 1}, {1, 2, 3, 3, 1, 2}, {1, 2, 3, 3, 2, 1}, {1, 2, 3, 4, 1, 2},
{1, 2, 3, 4, 2, 1}, {1, 2, 3, 1, 4, 3}, {1, 2, 3, 3, 4, 1}, {1, 2, 3, 4, 5, 1}, {1, 2, 3, 2, 3, 4},
{1, 2, 3, 3, 2, 4}, {1, 2, 3, 4, 2, 5}, {1, 2, 3, 3, 4, 5}, {1, 2, 3, 4, 5, 6}} : intlistlist

The recursive morphic approach
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.

The *morphoRules* are defining the production of morphic palindromes. The full grammar and program will produce the palindromes on the base of the *morphoRules* and print them as a list of 'rough' palindromes and additionally as a list of palindromes of canonical order, i.e. as palindromes in trito-normal form defined by the operation tnf.

Recursive definition of morphic palindromes
Basis: [⌀] and [1] are morphic palindromes
Induction: If for [P] = [*w*] = [w1w2], [P] is a palindrome,
even palindromes for w1!=w2, and
odd palindromes for w1=w2,
so are

Rules even
R1: [P] --> w1[P]w2
R2: [P] --> w2[P]w1
R3: [P] --> w3[P]w3
R4: [P] --> w3[P]w4.

Rule odd
R5 [if length w odd] [P] => wM [P] wM
wM = middleElement(P)

Defs
w3 = add(|w1|,1)
w4 = add(|w3|,1)
wM = middleElement(P)

Closure
No string is a morphic palindrome of ∑(w), unless it follows from this basis and the inductive rules R1 - R5.

With that, inductive proofs of properties of *morphoGrammars* are introduced.

Hence, as the first palindrome productions, we get:
[⌀] --> [1,1], [1,2] : R1, R4: *even* palindromes
[1] --> [1,1,1], [2,1,2], [2,1,3] : R1 (R5), R3, R4: *odd* palindromes.

Palindrome grammar – properties

Rules even

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

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

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

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

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

Rule odd

$R5\,[\text{if length } w \text{ odd}]\ [w] => wM\,[w]\,wM$

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

Defs

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

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

even : $w1 \neq w2$,

odd : $w1 = w2$

Non – recursive test

fun ispalindrome l = (l = kref l);

fun kref ks = tnf(rev ks);

The palindrome grammar is applied as a palindrome *tester*. If a morphogram accepts the properties defined with the rules, it is a palindrome.

Palindrome grammar – production

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

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

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

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

Rule5 : $[\text{if length } w \text{ odd}]$

$[P] \Longrightarrow wM\,[P]\,wM$

Defs

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

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

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

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

The palindrome production system is producing recursively the domain of morphic palindromes.

For short:

If P ∈ Palin, then Rules1 – 5 (P) ∈ Palin

Test-Example

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

tnf might be replaced by **ReLabel.**

```
- tnf;
val it = fn : "a list -> int list
```

Examples for tnf

```
- tnf["a", "c"];
val it = [1,2] : int list
- tnf["&", "*"];
val it = [1,2] : int list
- tnf[3,1,4];
val it = [1,2,3] : int list
```

Production example for even palindromes

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

	rules	result	tnf
P = [1,1]:	w1Pw1	[1,**1**,**1**,1] -> [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] -> [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

Production example for odd palindromes

P: w1=w2: [1]

	rules	result	tnf
P = [1]:	w1Pw1	[1,1,1] -> [1,1,1]	; rule5
	w3Pw3	[2,1,2] -> [1,2,1]	; rule3
	w3Pw4	[2,1,3] -> [1,2,3]	; rule4

Example for morphogram in tnf [1, 2, 3, 4] : even

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

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

[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]

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

odd(3)	[1, 2, 3]	[2, 1, **2**, 3, 2] [1, **1**, 2, **3**, 3]	rule5 rule1	[1, 2, 1, 3, 1] [1, 1, 2, 3, 3]
–	–	[3, **1**, 2, **3**, 1]	rule2	[1, 2, 3, 1, 2]
–	–	[4, **1**, 2, **3**, 4]	rule3	[1, 2, 3, 4, 1]
–	–	[4, **1**, 2, **3**, 5]	rule4	[1, 2, 3, 4, 5]

1.3. Production tables

1.3.1. Table for Palin(1-3-5)

ATTENTION
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.

As mentioned before,

Rules(palindromes) $=_{MG}$ Rules(tnf(Palindromes)).

type::odd	start	result: odd: (w1 = w2)	rules	tnf	ENstructureEN
odd(1)	[1]	[1, 1, 1]	rule1	[1, 1, 1]	e – / e e
–	–	[2, 1, 2]	rule3	[1, 2, 1]	v – / e v
–	–	[2, 1, 3]	rule4	[1, 2, 3]	v – / v **v**
odd(3)	[1, 1, 1]	[1, 1, 1, 1, 1]	rule1	[1, 1, 1, 1, 1]	e – – – / e **e** – – / e **e** **e** – / e e e e
–	–	[2, 1, 1, 1, 2]	rule3	[1, 2, 2, 2, 1]	v – – – / v **e** – – / v **e** **e** – / e v v v
–	–	[2, 1, 1, 1, 3]	rule4	[1, 2, 2, 2, 3]	v – – – / v **e** – – / v **e** **e** – / v v v v .
–	[1, 2, 1]	[1, **1**, 2, **1**, 1]	rule1	[1, 1, 2, 1, 1]	etc.
–	–	[2, 1, **2**, 1, 2]	rule5	[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]	□
–	–	[2, 1, **2**, 3, 2]	rule 5	[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].

scala > prnt(genPalindrome(5)) : 12 OK

Test – Scala odd(3) morphoProRule : [2, 1, 2]

```
scala > morphoProdRule1(List(2, 1, 2))
res345 : List[Int] = List(2, 2, 1, 2, 2)
scala > morphoProdRule2(List(2, 1, 2))
res346 : List[Int] = List(2, 2, 1, 2, 2)
scala > morphoProdRule3(List(2, 1, 2))
res347 : List[Int] = List(3, 2, 1, 2, 3)
scala > morphoProdRule4(List(2, 1, 2))
res348 : List[Int] = List(3, 2, 1, 2, 4)
scala > morphoProdRule5(List(2, 1, 2))
res349 : List[Int] = List(1, 2, 1, 2, 1)
```

Test – Scala odd(3) : [2, 1, 3]

```
scala > morphoProdRule1(List(2, 1, 3))
res353 : List[Int] = List(2, 2, 1, 3, 3)
scala > morphoProdRule2(List(2, 1, 3))
res359 : List[Int] = List(3, 2, 1, 3, 2)
scala > morphoProdRule3(List(2, 1, 3))
res355 : List[Int] = List(4, 2, 1, 3, 4)
scala > morphoProdRule4(List(2, 1, 3))
res356 : List[Int] = List(4, 2, 1, 3, 5)
scala > morphoProdRule5(List(2, 1, 3))
res361 : List[Int] = List(1, 2, 1, 3, 1)
```

1.3.2. Patterns, Tables and ENstructures

ENpatterns

The symmetric structure of morphic palindromes becomes obvious with the E/N-pattern representation calculated by **ENstructureEN** (palin). Each palindrome *palin(n)* of such an E/N-structure is contained in the pattern of palin(n+1).

All patterns are symmetric in the sense of an equality of rows and columns of the square pattern. This holds for even as well as for odd morphic palindromes.

A morphogram is palindromic if its E/N – structure is symmetric.

The symmetry of the E/N-pattern (matrix) is a 'row/column' or right/left symmetry.

reversion : [ks] ⟶ [pal]
↕ ↕
symmetry : [EN] ⟶ [bi – sym]

$$\text{Palin}: w =_{MG} w^{rev} \text{ iff } M = M^{refl}$$

(morphogrammatically reversed word, bi-symmetric EN-matrix)

A similarity to be checked:

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)

The advantage of the EN-abstraction is the fact that it allows to emphasize much more directly than it is possible with other approaches, the genuine morphogrammatic properties of morphic palindromes.

The EN-table presentation of morphic palindromes, ENpalindromes, or ε/v-Palindromes, has the great advantage of a mathematical structure that is more directly accessible to analysis, elaboration and perception than its topological counterparts for standard palindromes in canonical form with its complex diagrams of knotted graphs.

Example for a EN-pattern

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

[1, **2**, **3**, 1]	[**3**, 1, 2, 3, 1, **2**]	rule2(2, 3)	[1, 2, 3, 1, 2, 3]

rule2(2, 3):

v	–	–	–
v	**v**	–	–
e	**v**	**v**	–

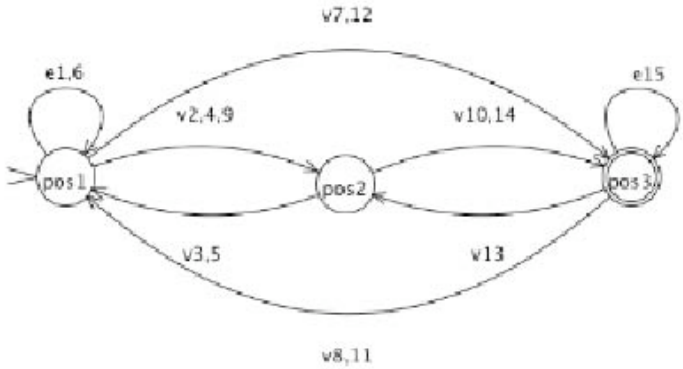
⟹

v	–	–	–	–
v	**v**	–	–	–
e	**v**	**v**	–	–
v	**e**	**v**	**v**	–
v	v	e	v	v

1.3.3. Palindromes in different approaches and notations

The **numeric list** notation gets different presentations as a *list*, a *graph*, a *EN-structure* and a EN-symmetry *table* and the classic *assembly graph,* and more.

1.Differentiation graph of palindrome [1,1,2,2,3,3] :



2. EN-structureEN of palindrome [1,2,2,3,1,4,4,3]:

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

val it =

[[],[N],[N,E],[N,N,N],[E,N,N,N],[N,N,N,N,N],[N,N,N,N,N,E],[N,N,N,E,N,N,N]]

: EN list list

3. EN-symmetry table of palindrome [1,2,2,3,1,4,4,3] :

V	–	–	–	–	–	–
V	**e**	–	–	–	–	–
V	V	V	–	–	–	–
e	V	V	V	–	–	–
V	V	V	V	V	–	–
V	V	V	V	V	**e**	–
V	V	V	**e**	V	V	V

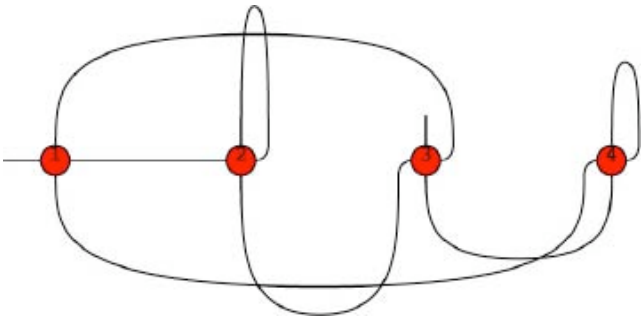
EN-symmetry table [1,2,1,3,2,3]

V	–	–	–	–
e	V	–	–	–
V	V	V	–	–
V	**e**	V	V	–
V	V	V	**e**	V

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

val it = [[],[N],[E,N],[N,N,N],[N,E,N,N],[N,N,N,E,N]] : EN list list

4. Assembly graph of (1,2,2,3,1,4,4,3):



EN-symmetry table [1,2,1,3,2,3]

V	–	–	–	–
e	V	–	–	–
V	V	V	–	–
V	**e**	V	V	–
V	V	V	**e**	V

- **ENstructureEN**[1,2,1,3,2,3];

val it = [[],[N],[E,N],[N,N,N],[N,E,N,N],[N,N,N,E,N]] : EN list list

Diagrams for numeric palindrome (1,2,1,3,2,3)

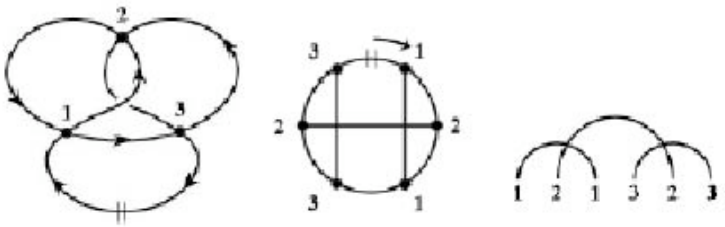


Figure 1: Self-intersecting closed curve (left), chord diagram (center), and linked diagram (right) representations of the double occurrence word 121323. Base points, indicating the starting point for reading the word, are marked by ||.

1.4. Towards production rules for ENpalindromes

1.4.1. Production rules for ENpalindromes

EN-palindromes are focused on the E/N-structure of morphograms and morphic palindromes.

The EN-approach is *presenting* palindromes, while the *list* approach of keno-sequences for morphograms is just *re-presenting* palindromes in a canonical notation form.

Production rules for ENpalindromes

Alphabet : $\Sigma = \{\varnothing,\ e,\ v\}$

Axioms

Axiom0 : $\boxed{\varnothing} \Longrightarrow \begin{array}{|c|c|}\hline - & - \\ \hline e & - \\ \hline \end{array}$: $[1]$

Axiom1 : $\boxed{\varnothing} \Longrightarrow \begin{array}{|c|c|}\hline - & - \\ \hline e & - \\ \hline \end{array}$: $[1, 1]$

Axiom2 : $\boxed{\varnothing} \Longrightarrow \begin{array}{|c|c|}\hline - & - \\ \hline v & - \\ \hline \end{array}$: $[1, 2]$

Odd elementary rules

Axiom0 = $[1]$

Rule1 : $\begin{array}{|c|c|}\hline - & - \\ \hline e & - \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|}\hline e & - \\ \hline e & e \\ \hline \end{array}$: $[1, 1, 1]$

Rule2 : $\begin{array}{|c|c|}\hline - & - \\ \hline e & - \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|}\hline v & - \\ \hline \mathbf{e} & v \\ \hline \end{array}$: $[2, 1, 2]$

Rule3 : $\begin{array}{|c|c|}\hline - & - \\ \hline v & - \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|}\hline \mathbf{v} & - \\ \hline \mathbf{v} & \mathbf{v} \\ \hline \end{array}$: $[2, 1, 3]$.

Even elementary rules

Axiom1 . 1 = $[1, 1]$

Rule1 . 1 : $\begin{array}{|c|c|}\hline - & - \\ \hline \mathbf{e} & - \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|c|}\hline e & - & - \\ \hline e & e & - \\ \hline e & e & e \\ \hline \end{array}$: $[1, 1, 1, 1]$

Rule1 . 2 : $\begin{array}{|c|c|}\hline - & - \\ \hline \mathbf{e} & - \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|c|}\hline v & - & - \\ \hline v & e & - \\ \hline e & v & v \\ \hline \end{array}$: $[2, 1, 1, 2]$

Rule1 . 3 : $\begin{array}{|c|c|}\hline - & - \\ \hline \mathbf{e} & - \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|c|}\hline v & - & - \\ \hline v & e & - \\ \hline v & v & v \\ \hline \end{array}$: $[2, 1, 1, 3]$

Axiom1 . 2 = $[1, 2]$

Rule2 . 1 : $\begin{array}{|c|c|}\hline - & - \\ \hline \mathbf{v} & - \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|c|}\hline e & - & - \\ \hline v & v & - \\ \hline v & v & e \\ \hline \end{array}$: $[1, 1, 2, 2]$

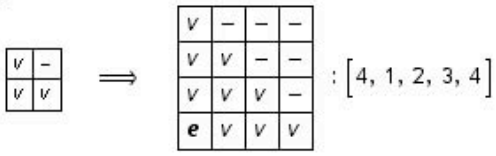
Rule2 . 2 : $\begin{array}{|c|c|}\hline - & - \\ \hline \mathbf{v} & - \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|c|}\hline v & - & - \\ \hline e & v & - \\ \hline v & e & v \\ \hline \end{array}$: $[2, 1, 2, 1]$

Rule2 . 3 : $\begin{array}{|c|c|}\hline - & - \\ \hline \mathbf{v} & - \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|c|}\hline v & - & - \\ \hline v & v & - \\ \hline e & v & v \\ \hline \end{array}$: $[3, 1, 2, 3]$

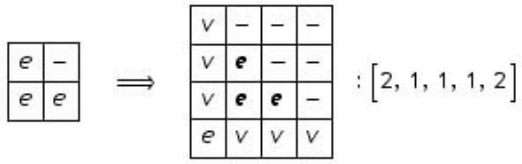
Rule2 . 4 : $\begin{array}{|c|c|}\hline - & - \\ \hline \mathbf{v} & - \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|c|}\hline v & - & - \\ \hline v & v & - \\ \hline v & v & v \\ \hline \end{array}$: $[3, 1, 2, 4]$

EN – Rules applications

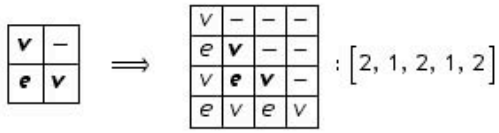
Odd ENpalindromes

Rule4 :  : [4, 1, 2, 3, 4]

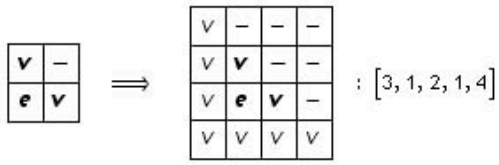
Rule4 = Rule2 (Rule3 (Axiom0))

Rule5 :  : [2, 1, 1, 1, 2]

Rule5 = Rule2 (Rule1 (Axiom1))

Rule6 :  : [2, 1, 2, 1, 2]

Rule6 = Rule3 (Rule2 (Axiom0))

Rule7 :  : [3, 1, 2, 1, 4]

Rule7 = Rule4 (Rule2 (Axiom0))

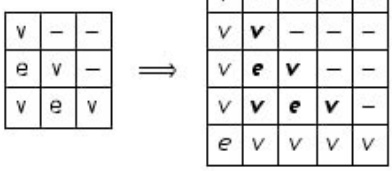
Example for an even ENpalindrome

– ispalindrome (tnf[3, 2, 1, 2, 1, 3]);

val it = true : bool

– ENstructureEN[3, 2, 1, 2, 1, 3];

val it = [[], [N], [N, N], [N, E, N], [N, N, E, N], [E, N, N, N, N]] : EN list list

Rule1 .2 : 

rule 3 : [2, 1, 2, 1] ⟹ [3, 2, 1, 2, 1, 3].

Even elementary rules

Axiom2 = [1,2]:
[1,2] -> [1,1,2,2]
-> [2,1,2,1] : [1,2,1,2]
-> [3,1,2,3] : [1,2,3,1]
-> [3,1,2,4] : [1,2,3,4]

```
- ENstructureEN[1,1,2,2];
val it = [[],[E],[N,N],[N,N,E]] : EN list list
- ENstructureEN[2,1,2,1];
val it = [[],[N],[E,N],[N,E,N]] : EN list list
- ENstructureEN[3,1,2,3];
val it = [[],[N],[N,N],[E,N,N]] : EN list list
- ENstructureEN[3,1,2,4];
val it = [[],[N],[N,N],[N,N,N]] : EN list list
```

ML definition of ENpalindrome

```
fun ENpalindrome l = (ENstructureEN (l) = ENstructureEN (kref l));
val ENpalindrome = fn : "a list -> bool
```

```
fun ENpalindrome ENstructureEN(l) = (ENstructureEN (l) = ENstructureEN (kref l));
val ENpalindrome = fn : (int list -> "a) -> int list -> bool
```

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

Axiom1 = [1,1]
P = [1,1]: -> [1,**1,1**,1]
-> [2,1,1,2] : [1,2,2,1]
-> [2,**1,1**,3] : [1,2,2,3]

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

Applications for odd ENpalindromes

Rule2 .1 : $\begin{array}{|c|c|}\hline - & - \\ \hline \boldsymbol{v} & - \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|c|}\hline e & - & - \\ \hline v & \boldsymbol{v} & - \\ \hline v & v & e \\ \hline \end{array} : [1, 1, 2, 2]$

rule2 : $[1, 1, 1] \Longrightarrow [2, 1, 1, 1, 2]$

Rule5 : $\begin{array}{|c|c|}\hline e & - \\ \hline e & e \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|c|c|}\hline v & - & - & - \\ \hline v & \boldsymbol{e} & - & - \\ \hline v & \boldsymbol{e} & \boldsymbol{e} & - \\ \hline e & v & v & v \\ \hline \end{array}$: accretion, sym.

chain : $(1) \Longrightarrow (3) \Longrightarrow (10)$

Rule2 : $\begin{array}{|c|c|}\hline e & - \\ \hline e & e \\ \hline \end{array} \Longrightarrow \begin{array}{|c|}\hline v \\ \hline v \\ \hline v \\ \hline e \\ \hline \end{array} \begin{array}{|c|c|}\hline \boldsymbol{e} & - \\ \hline \boldsymbol{e} & \boldsymbol{e} \\ \hline \end{array} : \begin{array}{|c|c|}\hline N & - \\ \hline \boldsymbol{E} & N \\ \hline \end{array}$

$\begin{array}{|c|c|c|c|}\hline e & v & v & v \\ \hline \end{array}$

(vvve) ↓

1 v1, 2	–	–	–
2 v1, 3	3 **e2, 3**	–	–
4 v1, 4	5 **e2, 4**	6 **e3, 4**	–
7 e1, 5	8 v2, 5	9 v3, 5	10 v4, 5

: (row = inv(column)) = palin

← (evvv)

rule1 : [1, 1, 1] ⟹ [1, 1, 1, 1, 1]

Rule1 :

e	–
e	e

⟹

e	–	–	–
e	**e**	–	–
e	**e**	**e**	–
e	**e**	**e**	**e**

rule4 : [1, 1, 1] ⟹ [2, 1, 1, 1, 3]

Rule3 :

e	–
e	e

⟹

v	–	–	–
v	**e**	–	–
v	**e**	**e**	–
v	v	v	v

accretion, asym

rule1 : [1, 2, 1] ⟹ [1, 1, 2, 1, 1] ⟹ [1, 1, 1, 2, 1, 1, 1]

Rule1 :

v	–
e	**v**

⟹

e	–	–	–
v	**v**	–	–
e	**e**	**v**	–
e	e	v	**e**

⟹

e	–	–	–	–	–
e	**e**	–	–	–	–
v	**v**	**v**	–	–	–
e	**e**	**e**	**v**	–	–
e	**e**	**e**	**v**	**e**	–
e	e	e	v	e	**e**

ENstructureEN[1,1,1,2,1,1,1]:
[[],[E],[E,E],[N,N,N],[E,E,E,N],[E,E,E,N,E],[E,E,E,N,E,E]]

Next example

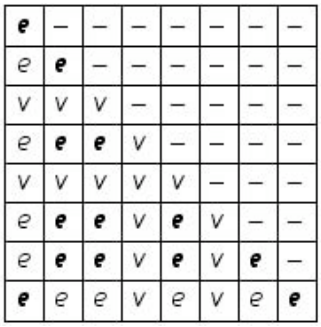
pal = [1,1,2,1,3,1,1] rule4(pal) = [4,1,1,2,1,3,1,1,5]

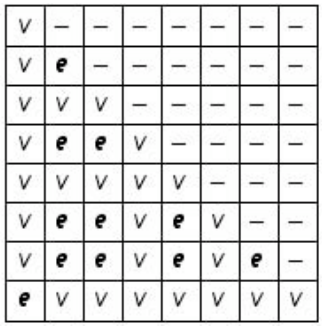
e	–	–	–	–	–
v	v	–	–	–	–
e	**e**	v	–	–	–
v	v	v	v	–	–
e	**e**	v	**e**	v	–
e	**e**	v	**e**	v	**e**

⟹

v	–	–	–	–	–	–	–
v	**e**	–	–	–	–	–	–
v	v	v	–	–	–	–	–
v	**e**	**e**	v	–	–	–	–
v	v	v	v	v	–	–	–
v	**e**	**e**	v	**e**	v	–	–
v	**e**	**e**	v	**e**	v	**e**	–
v	v	v	v	v	v	v	v

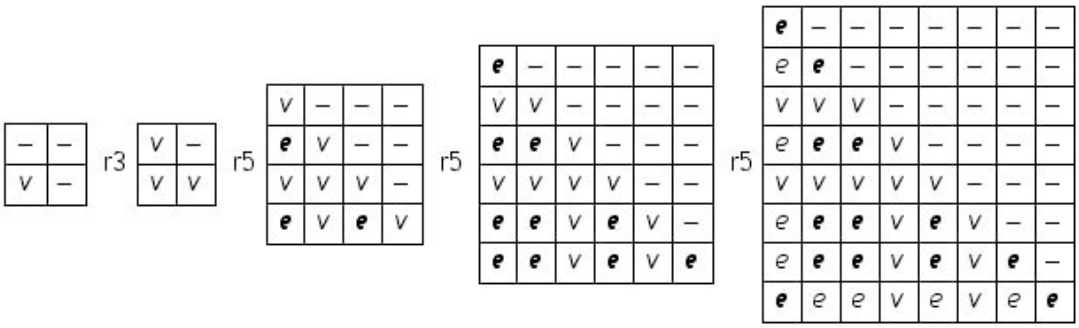
rule1(pal) = [1,1,1,2,1,3,1,1,1] rule3(pal) = [4,1,1,2,1,3,1,1,4]





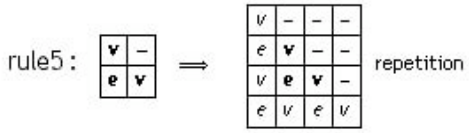
- **ENstructureEN**[1,1,1,2,1,3,1,1,1];
  [[],[E],[E,E],[N,N,N],[E,E,E,N], [N,N,N,N,N],[E,E,E,N,E,N],[E,E,E,N,E,N,E],
  [E,E,E,N,E,N,E,E]]
- **ENstructureEN**[4,1,1,2,1,3,1,1,4];
  [[],[N],[N,E],[N,N,N],[N,E,E,N], [N,N,N,N,N],[N,E,E,N,E,N],[N,E,E,N,E,N,E],
  [E,N,N,N,N,N,N,N]]

Pal1 : $\left[1,\ 1,\ 1,\ 2,\ 1,\ 3,\ 1,\ 1,\ 1\right] = \text{r5}\left(\text{r5}\left(\text{r5}\left(\text{r3}\left(\left[1\right]\right)\right)\right)\right)$

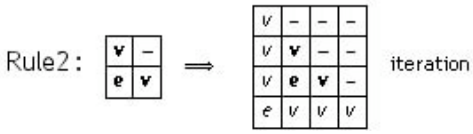


[v] as [1] [2,1,3] [1,2,1,3,1] [1,1,2,1,3,1,1] [1,1,1,2,1,3,1,1,1]

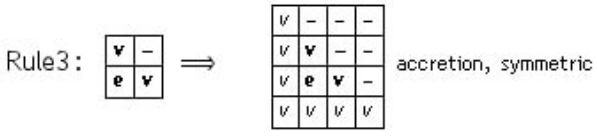
rule5 : $\left[1,\ 2,\ 1\right] \Longrightarrow \left[2,\ 1,\ 2,\ 1,\ 2\right]$



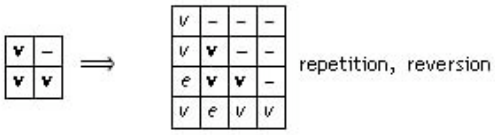
rule3 : $\left[1,\ 2,\ 1\right] \Longrightarrow \left[3,\ 1,\ 2,\ 1,\ 3\right]$



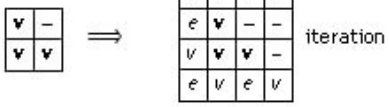
rule4 : $\left[1,\ 2,\ 1\right] \Longrightarrow \left[3,\ 1,\ 2,\ 1,\ 4\right]$



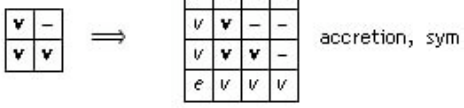
rule2 : $\{1, 2, 3\} \Longrightarrow \{3, 1, 2, 3, 1\}$

rule3 :  repetition, reversion

rule3 : $\{1, 2, 3\} \Longrightarrow \{2, 1, 2, 3, 2\}$

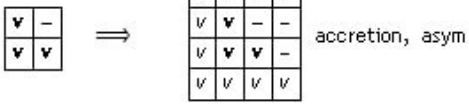
rule2 :  iteration

rule4 : $\{1, 2, 3\} \Longrightarrow \{4, 1, 2, 3, 4\}$

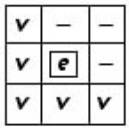
rule4 :  accretion, sym

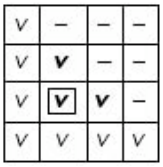
Null

rule4 : $\{1, 2, 3\} \Longrightarrow \{4, 1, 2, 3, 5\}$

Rule3 :  accretion, asym

Even and odd centers

center palin(odd) = 

center palin(even) = 

1.4.2. Inversion of productions

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]]

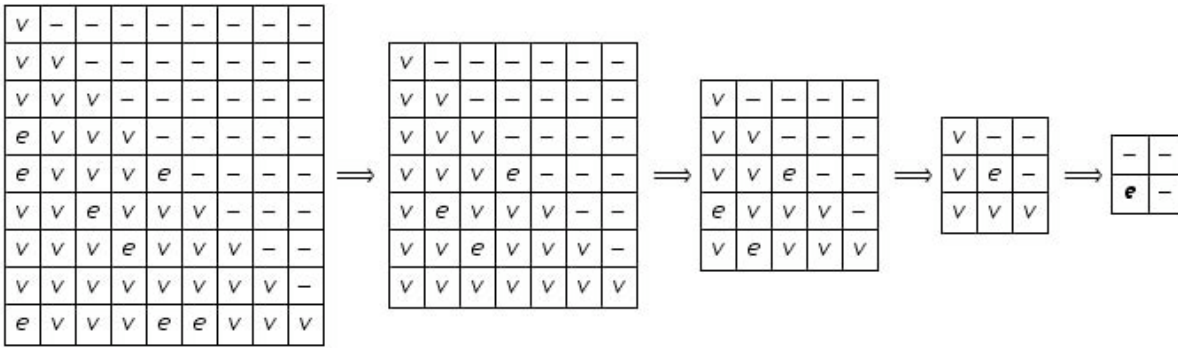
Rule2

-**ENstructureEN**[2,1,1,3];
[[],[N],[N,E],[N,N,N]]

Rule1

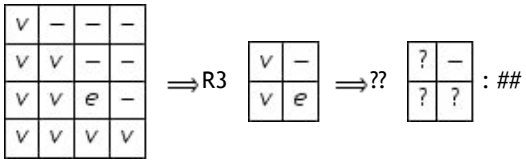
-**ENstructureEN**[1,1];
[[],[E]]

ENstructureEN tables : direct cutting



Contrary example

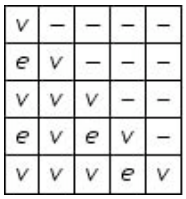
- **ENstructureEN**[3,1,2,2,4];
val it = [[],[N],[N,N],[N,N,E],[N,N,N,N]] :



Albeit the first row/column is symmetric, there is no rule to finally reduce the morphogram to an ENpalindrome axiom. Hence, the morphogram fails the test, and is therefore not a palindrome.

- **ispalindrome** [1,2,1,3,1,3];
val it = false : bool
- **ENstructureEN**[1,2,1,3,1,3];
val it = [[],[N],[E,N],[N,N,N],[E,N,E,N],[N,N,N,E,N]] : EN list list

The table for **ENstructureEN**[1,2,1,3,1,3] is not *symmetric*. Hence it is not a palindrome.



1.4.3. Classic symmetric palindromes

Symmetric palindromes from [1] to [1,2,3,4,5,6,7]: 31
[[1],[1,1],
[1,1,1],[1,2,1],
[1,1,1,1],[1,2,2,1],
[1,1,1,1,1],[1,1,2,1,1],[1,2,1,2,1],[1,2,2,2,1],[1,2,3,2,1],
[1,1,1,1,1,1],[1,1,2,2,1,1],[1,2,1,1,2,1],[1,2,2,2,2,1],[1,2,3,3,2,1],
[1,1,1,1,1,1,1],[1,1,1,2,1,1,1],
[1,1,2,1,2,1,1],[1,1,2,2,2,1,1],[1,1,2,3,2,1,1],[1,2,1,1,1,2,1],

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

Examples: symmetric Palin(5)

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

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

[1,1,1,1,1]

e	–	–	–
e	e	–	–
e	e	e	–
e	e	e	e

[1,1,2,1,1]

e	–	–	–
v	v	–	–
e	e	v	–
e	e	v	e

[1,2,1,2,1]

v	–	–	–
e	v	–	–
v	e	v	–
e	v	e	v

[1,2,2,2,1]

v	–	–	–
v	e	–	–
v	e	e	–
e	v	v	v

[1,2,3,2,1]

v	–	–	–
v	v	–	–
v	e	v	–
e	v	v	v

Out of 35 morphic palindromes, 5 palindromes are filtered out as *symmetric* in the classical sense. There is a total of 203 morphograms of length 5. Hence, there are still 30 morphic palindromes that are symbolically asymmetric but structurally symmetric.

Obviously, for all classical palindromes, position (1,5) has to be marked by equality, i.e by *e*.

Distribution of v is:
v([1,1,1,1,1]) = 0,
v([1,1,2,1,1]) = 4,
v([1,2,1,2,1]) = 6,
v([1,2,2,2,1]) = 6,
v([1,2,3,2,1]) = 8,

This shows, again, the enormeous structural poverty of the classical approach to palindromes and their specific iterability.

EN-structure of DOW palindromes of length 6:

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

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

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

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

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

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

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

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

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

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

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

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

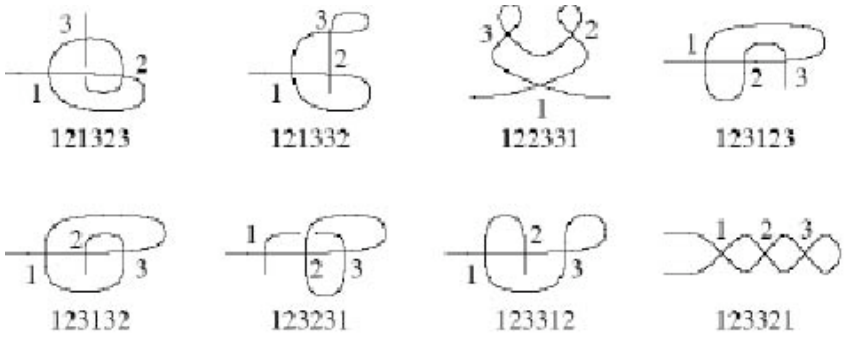
v	–	–	–	–
v	v	–	–	–
e	v	v	–	–
v	e	v	v	–
v	v	e	v	v

Duality between DOW palindromes

[1,2,3,3,2,1] dual [1,2,3,1,2,3]
[1,2,3,3,1,2] dual [1,2,3,2,3,1]
[1,1,2,2,3,3] dual [1,1,2,2,3,3]
[1,2,2,3,3,1] dual [1,2,2,3,3,1]
[1,2,1,3,2,3] dual [1,2,1,3,2,3].

e(DOW-palin(3)) = 3.

Representatives of isomorphism classes of irreducible graphs of size 3 (Jonathan Burns et al, 2011)
(palindromic and non-palindromic, [1,2,1,3,3,2], [1,2,3,1,3,2], [1,2,1,3,3,2], graphs)



Component analysis
Following the dissertation of Jamie Sprecher, a structural analogy to the ENstructure-analysis of this paper is noted. This graph-theoretical analysis might be translated into the context of EN-structures and their tabulations.

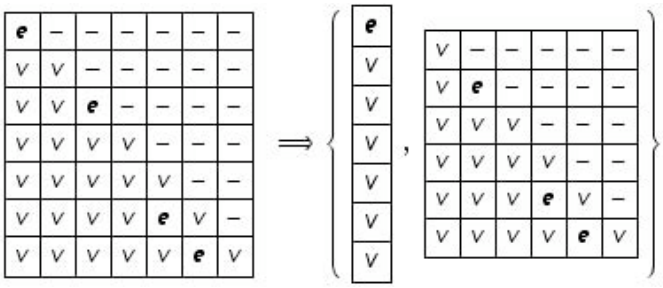
Because morphograms are patterns and not sequentiell strings, i.e. directed graphs, the inter-connectedness of the parts is 'remembered' by the reductive cut into different components.

The assembly number of a graph is the minimum number of polygonal paths needed to create a Hamiltonian set for that graph.

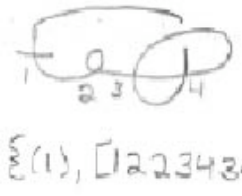
Jamie Sprecher, Multicomponent Assembly Graphs in DNA Recombination
http://knot.math.usf.edu/multimedia/jamies_thesis_2013.pdf

This might be seen as a very first step of a comparison of the *multicomponent assembly graphs* and corresponding morphograms and their decomposition into palindromeic monomorphies.

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

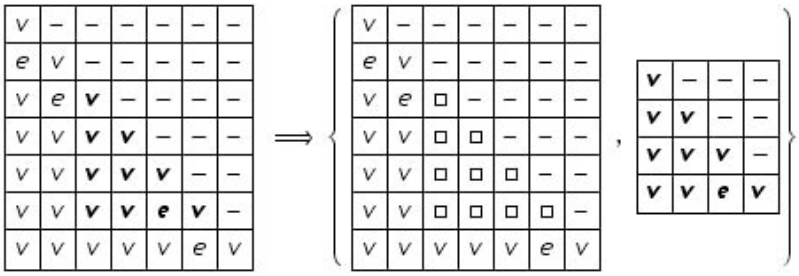


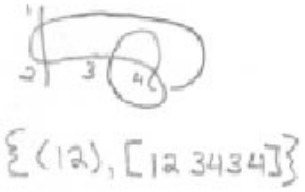
[1,1,2,2,3,4,3,4] ==> {[1], [1,2,2,3,4,3,4]}



- **ENstructureEN**[1,1,2,2,3,4,3,4];
  [[],[E],[N,N],[N,N,E],[N,N,N,N],[N,N,N,N,N],[N,N,N,N,E,N],[N,N,N,N,N,E,N]]
- **ENstructureEN**[1,2,2,3,4,3,4];
  [[],[N],[N,E],[N,N,N],[N,N,N,N],[N,N,N,E,N],[N,N,N,N,E,N]]

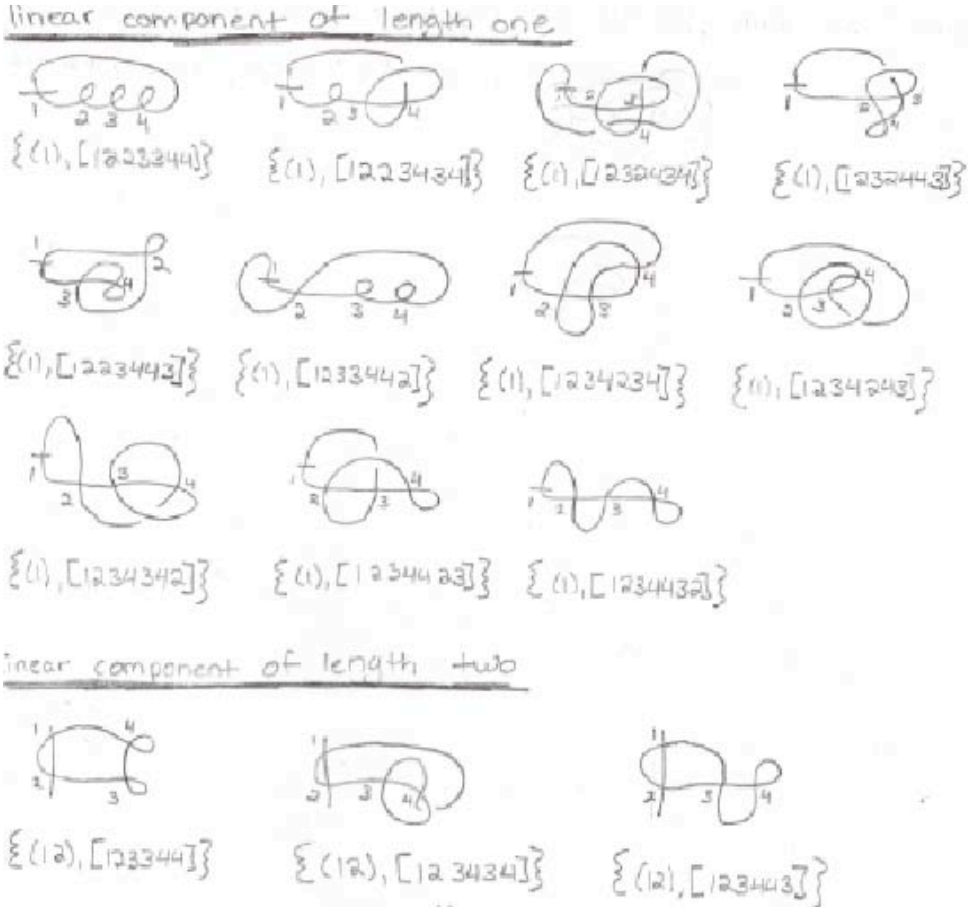
Example [1,2,1,2,3,4,3,4]





- **ENstructureEN**[1,2,1,2,3,4,3,4];
  [[],[N],[E,N],[N,E,N],[N,N,N,N],[N,N,N,N,N],[N,N,N,N,E,N],[N,N,N,N,N,E,N]]
- **ENstructureEN**[1,2,3,4,3,4];
[[],[N],[N,N],[N,N,N],[N,N,E,N],[N,N,N,E,N]]
- **ENstructureEN**[1,2]; [[],[N]]

Further decompositions (Jamie Sprecher)



http://knot.math.usf.edu/multimedia/jamies_thesis_2013.pdf

1.4.4. Duality principle for ENpalindromes

There is a complementarity principle for DNA palindromes related to their *elements* which directly leads to Herrlich's economic principle of *"two for one"* of duality in category theory.

Two for one

Genetics is cleverly applying the duality principle to reduce the costs of observation. Because of the duality (DNA complementarity) of A and T and C and G, *"one is justified in following the evolutionary development of one strand and ignoring the other strand"* (Kimura's DNA Substitution Model).

"*For example, if one strand contains the block -ACCGT- of bases, then the other strand contains the complementary block -TGGCA- of bases.*"

http://www.genetics.ucla.edu/courses/hg236b/Lange_Chapter_PopGenetics.pdf

One more

"DNA (deoxyribonucleic acid) is found in any living organism, it consists of polymer chains, called DNA strands. DNA strands are composed of nucleotides (sometimes called bases): A (adenine), G (guanine), C (cytosine), and T (thymine). A and G are called purines, C and T are called pyrimidines.

"In nature the strands form the well-known double helix by a bondage of two separate strands. This bonding always takes place by pairwise attraction of the bases: A bonds with T, G bonds with C. This phenomenon is called ***Watson-Crick complementarity*** *and the pairs and are called complementary pairs.*

"Watson-Crick complementarity *gives us some fundamental information for free: whenever in a double strand a bondage takes place, we know that the bases at the corresponding positions are complementary. Moreover, for a single DNA strand nature gives its*

complementary pair; therefore if we have one member of a double strand, then we have its complementary pair as well."

http://www.cs.bme.hu/~csima/phd1/node25.html#watson

This kind of element-oriented complementarity (duality, complementary strand) is focused on its elements and is not yet taking the internal, symmetric and asymmetric, *structures* of the palindromes into account.

In contrast, the *ENpalindrome* concept is purely structural and algebraic, and not based on the duality of its elements, equal = E and non-equal = N, of the palindromes.

The Watson-Crick complementarity is working with 4 elements, the ENpalindromic complementarity works with 2 elements {E, N} and the *grid* (matrix) of their distribution.

Obviously, the patterns $\begin{bmatrix} \mathbf{v} & - \\ e & \mathbf{v} \end{bmatrix}$ and $\begin{bmatrix} e & - \\ \mathbf{v} & e \end{bmatrix}$ are not complementary. Albeit there might be a duality between the elementary E and N. The latter pattern even doesn't exist.

A kind of complementation might exist between homogeneous EN-patterns like: $\begin{bmatrix} \mathbf{v} & - \\ \mathbf{v} & \mathbf{v} \end{bmatrix}$ and $\begin{bmatrix} e & - \\ e & e \end{bmatrix}$ based on the elementary duality of E and N.

Structural complementarity

Because of the structural symmetry of palindromes expressed in the ENtables of palindromes, a new specific duality (complementarity) of palindromes is introduced. The table or matrix duality of palindromes.

Slogan

Instead of the *pheno*-type bargain shopping at the CatShop we get a *geno*-type interaction of "two for one" at the deep-structural level of the morphic game.

$$\text{Palin} = \text{pal} \cup \overline{\text{pal}}$$

Duality of the symmetric order on the matrix

1, 2	–	–	–
1, 3	2, 3	–	–
1, 4	2, 4	3, 4	–
1, 5	2, 5	3, 5	4, 5

:

+	–	–	–
+	+	–	–
+	**2, 4**	3, 4	–
1, 5	2, 5	3, 5	4, 5

dual

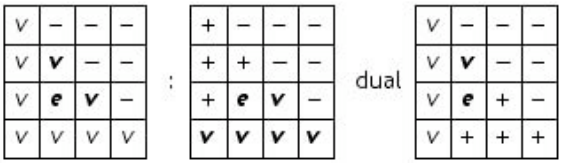
1, 2	–	–	–
1, 3	2, 3	–	–
1, 4	**2, 4**	+	–
1, 5	+	+	+

Symmetry scheme

(1, 2) dual (4, 5)
(1, 3) dual (3, 5)
(2, 3) dual (3, 4)
(2, 4) dual (2, 4)
(1, 4) dual (2, 5)
(1, 5) dual (1, 5)

Examples

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



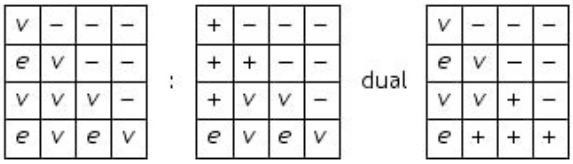
$(1, 2, N)$ dual $(4, 5, N)$
$(1, 3, N)$ dual $(3, 5, N)$
$(2, 3, N)$ dual $(3, 4, N)$
$(2, 4, E)$ dual $(2, 4, E)$
$(1, 4, N)$ dual $(2, 5, N)$
$(1, 5, N)$ dual $(1, 5, N)$.
The loci $(1, 5)$ and $(2, 4)$ are self – dual.

full relational $([3, 1, 2, 1, 4]) \Longrightarrow$ dually scrapped $([3, 1, 2, 1, 4])$

Duality under the condition of palindromic symmetry

– **ENstructureEN** $[3, 1, 2, 1, 4]$;
val it = $[[], [N], [N, N], [N, \mathbf{E}, \mathbf{N}], [\mathbf{N}, \mathbf{N}, \mathbf{N}, \mathbf{N}]]$
val it = $[[], [\mathbf{N}], [\mathbf{N}, \mathbf{N}], [\mathbf{N}, \mathbf{E}, N], [\mathbf{N}, N, N, N]]$

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



EN – Symmetry
$[[], [N], [E, N], [N, \mathbf{N}, \mathbf{N}], [\mathbf{E}, \mathbf{N}, \mathbf{E}, \mathbf{N}]]$
$[[], [\mathbf{N}], [\mathbf{E}, \mathbf{N}], [\mathbf{N}, \mathbf{N}, N], [\mathbf{E}, N, E, N]]$.

Symmetry table

$(1, 2, N)$ dual $(4, 5, N)$
$(1, 3, E)$ dual $(3, 5, E)$
$(2, 3, N)$ dual $(3, 4, N)$
$(2, 4, N)$ dual $(2, 4, N)$
$(1, 4, N)$ dual $(2, 5, N)$
$(1, 5, E)$ dual $(1, 5, E)$.

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

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

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

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

Symmetry for $[2, 1, 2, 3, 2]$

$[[N], [E, N], [N, N, N], [E, N, E, N]]$ =
$[[N], [E, N], [N, \boldsymbol{N}, \boldsymbol{N}], [\boldsymbol{E}, \boldsymbol{N}, \boldsymbol{E}, \boldsymbol{N}]]$
$[[\boldsymbol{N}], [\boldsymbol{E}, \boldsymbol{N}], [\boldsymbol{N}, \boldsymbol{N}, N], [\boldsymbol{E}, N, E, N]]$

1.4.5. More ENstructureEN and tables

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

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

- **ENstructureEN**[1,1,1,1,1];
val it = [[],[E],[E,E],[E,E,E],[E,E,E,E]] : EN list list

e	–	–	–
e	e	–	–
e	e	e	–
e	e	e	e

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

v	–	–	–
v	e	–	–
v	e	e	–
e	v	v	v

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

e	–	–	–	–	–
v	**v**	–	–	–	–
v	**v**	**e**	–	–	–
v	**v**	**e**	**e**	–	–
e	**e**	**v**	**v**	**v**	–
e	e	v	v	v	e

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

v	–	–	–
v	e	–	–
v	e	e	–
v	v	v	v

ENstructureEN[4,2,1,1,1,3,5];
val it = [[],[N],[N,N],[N,N,E],[N,N,E,E],[N,N,N,N,N],[N,N,N,N,N,N]]

Rule3 :

v	–	–	–
v	e	–	–
v	e	e	–
v	v	v	v

==>

v	–	–	–	–	–
v	v	–	–	–	–
v	v	**e**	–	–	–
v	v	**e**	**e**	–	–
v	v	v	v	v	–
v	v	v	v	v	v

: accretion asym

- **ENstructureEN**[4,2,1,1,1,3,4];
val it = [[],[N],[N,N],[N,N,E],[N,N,E,E],[N,N,N,N,N],[E,N,N,N,N,N]]

Rule3:

v	–	–	–
v	e	–	–
v	e	e	–
v	v	v	v

==>

v	–	–	–	–	–
v	v	–	–	–	–
v	v	**e**	–	–	–
v	v	**e**	**e**	–	–
v	v	v	v	v	–
e	v	v	v	v	v

: accretion, sym.

rule3 : [1, 2, 3] ⟹ [2, 1, 2, 3, 2]

rule2 :

v	–
v	**v**

⟹

v	–	–	–
e	**v**	–	–
v	**v**	**v**	–
e	v	e	v

iteration

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

v	–	–	–	–	–
v	**v**	–	–	–	–
v	**e**	**v**	–	–	–
v	**v**	**e**	**v**	–	–
v	**e**	**v**	**e**	**v**	–
e	v	v	v	v	v

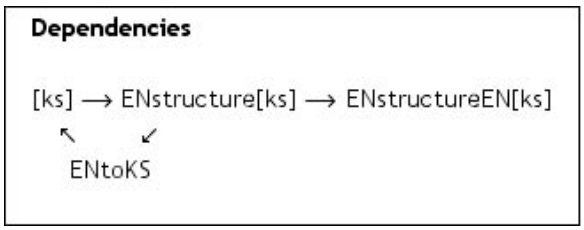
ENstructureEN
- ENstructureEN[1,1,2,1,1];
val it = [[],[E],[N,N],[E,E,N],[E,E,N,E]] : EN list list
- ENstructureEN[2,1,2,1,2];
val it = [[],[N],[E,N],[N,E,N],[E,N,E,N]] : EN list list
- ENstructureEN[3,1,2,1,3];
val it = [[],[N],[N,N],[N,E,N],[E,N,N,N]] : EN list list

```
- ENstructureEN[3,1,2,1,4];
val it = [[],[N],[N,N],[N,E,N],[N,N,N,N]] : EN list list
- ENstructureEN[1,1,2,3,1];
val it = [[],[E],[N,N],[N,N,N],[E,E,N,N]] : EN list list
- ENstructureEN[3,1,2,3,1];
val it = [[],[N],[N,N],[E,N,N],[N,E,N,N]] : EN list list
- ENstructureEN[2,1,2,3,2];
val it = [[],[N],[E,N],[N,N,N],[E,N,E,N]] : EN list list
- ENstructureEN[4,1,2,3,4];
val it = [[],[N],[N,N],[N,N,N],[E,N,N,N]] : EN list list
- ENstructureEN[4,1,2,3,5];
val it = [[],[N],[N,N],[N,N,N],[N,N,N,N]] : EN list list

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

Translations: **Conflict with the enummerations**

Dependencies



There is a conflict of enummeration between the functions *Enstructure* and *subsystems*.

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

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

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

Enumeration table

1 v1, 2	–	–	–
2 v1, 3	3 **e2, 3**	–	–
4 v1, 4	5 **e2, 4**	6 **e3, 4**	–
7 e1, 5	8 v2, 5	9 v3, 5	10 v4, 5

```
– subsystems 5;
val it =
 [(1, [1, 2]), (2, [2, 3]), (3, [1, 3]), (4, [3, 4]), (5, [2, 4]), (6, [1, 4]), (7, [4, 5]),
  (8, [3, 5]), (9, [2, 5]), (10, [1, 5])] : (int * int list) list
```

Correspondence
subsystems: (1, [1, 2]), **(2, [2, 3]), (3, [1, 3])** corresponds to
ENstructure: [(1, 2, N)], **[(1, 3,** N), **(2, 3**, E)].

1.4.6. Summary

Morphic palindromes had been introduced and studied on two levels of representation:
1. the alpha-numeric level with tuples or lists, esp. of integers,
2. the differentiational approach of the ε/v-structure.

The tuple approach got some specification by a grammar and by a production program written in Scala.
The production system for morphic palindromes is characterized a) by the morphic

production rules and b) by the context rule that refers backwards to the produced palindrome and is therefore adding to the recursive production rules an aspect of retro-grade recursion.

The ε/v-approach that is emphasizing the differential structure of a morphogram and its morphic palindromes by equality or non-equality (ε/v, or E/N) specifies palindromes independently of the elements of the tuple approach.

The ε/v-approach allows different presentations of morphic palindromes.
The matrix or table presentation of the ε/v-approach enables to proof that a morphogram is a palindrome iff its matrix is symmetric.

Furthermore, it make directly accessible a new form of duality for ε/v-palindromes.

2. Programming MorphoGrammars

2.1. Test programs

Ossip's Scala program for MorphoGrammar
This Scala program is not yet producing the list of palindromes of arbitrary length but is functioning as a recursive palindrome tester for the lists defined by the m*orphoRules*.

MorphoGrammar in Scala

```
def morphoRules (a : List[Int]) : Boolean =

   ((a.head == a.last) ||
(a.head + 1 == a.last) ||
(a.head + 1 == a.last + 1) ||
 (a.head + 1 == a.last + 2))

def isPalindrome (a : List[Int]) : Boolean =
     a match {
        case a if a.length > 1 = > morphoRules (a) &&
             isPalindrome (a.slice (1, a.length − 1))
        case _ = > true
 }
```

2.2. Production rules and programs

2.2.1. Production rules

Palindrome production rules

Alphabet

Σ = Int

Axioms

Axiom01 : $\Longrightarrow [\varnothing]$

Axiom02 : $\Longrightarrow [1]$

Rules even

Rule1 – 4 :

$[P] \Longrightarrow [w1Pw2] \,||\, [w2Pw1] \,||\, [w3Pw3] \,||\, [w3Pw4]$

Rules odd

[if length P odd]

Rule5 : $[P] \Longrightarrow [wM]\, P\, [wM]$

Defs

P = [w] = [w1 w2] : even

P = [w] = [wM w wM] : odd

wM = middleElement(w)

w3 = add(|w1|, 1)

w4 = add(|w3|, 1) = add(add(|w2|, 1), 1).

2.2.2. Production program (first version)

```
Morpho production program

def isOdd (i : Int) : Boolean = i % 2 != 0
def middle (s : List[Int]) = s (s.size / 2)

def genPalindrome (maxSize : Int, s : List[Int] = Nil) = {
 def genP (s : List[Int]) : Set[List[Int]] = {
  s match {
    case x if x.size >= maxSize - 1 = > Set (x)
    case x if x.isEmpty = >
         genP (List (1)) ++ genP (List (1, 1)) ++ genP (List (1, 2))
    case x = >
     Set (x) ++
     genP (List (x.head) ++ x ++ List (x.last)) ++
     genP (List (x.last) ++ x ++ List (x.head)) ++
     genP (List (x.last + 1) ++ x ++ List (x.last + 1)) ++
     genP (List (x.last + 1) ++ x ++ List (x.last + 2)) ++
     (if (isOdd (x.size))
       genP (List (middle (x)) ++ x ++ List (middle (x))) else Nil)
    }
 }
 genP (s)
}
```

Use
genPalindrome(5)
genPalindrome(5, List(2,3))

Print
```
def prnt(s:Set[List[Int]]) =  s.map("[" + _.mkString(",") +"]").toList.sorted.mkString("\n")
prnt(genPalindrome(n))
```
Size
genPalindrome(n).size

Test:
genPalindrome(5).map(x => x +":"+ isPalindrome(x).toString).mkString("\n")

2.2.3. Scala morpho production rules

Rules 1 – 4 : morphoProdRules for *even* palindromes

```
def morphoProdRule1 (s : List[Int]) = List (s.head) ++ (s) ++ List (s.last)
def morphoProdRule2 (s : List[Int]) = List (s.last) ++ (s) ++ List (s.head)
def morphoProdRule3 (s : List[Int]) = List (s.last + 1) ++ (s) ++ List (s.last + 1)
def morphoProdRule4 (s : List[Int]) = List (s.last + 1) ++ (s) ++ List (s.last + 2)
```

Rule 5 : morphoProdRule for *odd* palindromes

```
def isOdd (i : Int) : Boolean = i % 2 != 0
def middle (s : List[Int]) = s ( s.size / 2)

def morphoProdRule5 (s : List[Int]) = (if (isOdd (s.size))
  | (List (middle (s)) ++ (s) ++ List (middle (s))) else Nil)
```

Definition and test

```
def morphoProdRule1(s:List[Int]) = List(s.head)++(s)++List(s.last);
morphoProdRule1: (s: List[Int])List[Any]
scala> morphoProdRule1(List(1,2,3))
res53: List[Any] = List(1, 1, 2, 3, 3)

scala> def morphoProdRule2(s:List[Int]) = List(s.last)++(s)++List(s.head);
morphoProdRule2: (s: List[Int])List[Any]
scala> morphoProdRule2(List(1,2,3))
res56: List[Any] = List(3, 1, 2, 3, 1)

def morphoProdRule3(s:List[Int]) = List(s.last+1)++(s)++List(s.last+1)
scala> def morphoProdRule3(s:List[Int]) = List(s.last+1)++(s)++List(s.last+1)
morphoProdRule3: (s: List[Int])List[Any]
scala> morphoProdRule3(List(1,2,3))
res62: List[Any] = List(4, 1, 2, 3, 4)

scala> def morphoProdRule4(s:List[Int]) = List(s.last+1)++(s)++List(s.last+2)
morphoProdRule4: (s: List[Int])List[Any]
scala> morphoProdRule4(List(1,2,3))
res63: List[Any] = List(4, 1, 2, 3, 5)


scala> def morphoProdRule5 (s : List[Int]) = (if (isOdd(s.size))
     (List(middle(s)) ++ (s) ++ List(middle(s))) else Nil)
scala> morphoProdRule5(List(1,2,3))
res342: List[Int] = List(2, 1, 2, 3, 2)
```

2.2.4. Comments on the implementations

The *intuitive* and *manual* use of the rules follows some *context rules* (conditions) that are not yet implemented 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 for even palindromes.

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

P = [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]

Restriction:

[3, **1, 2**, 4] → [**3**, 3, **1, 2**, 4, **3**]: rule3, because **fst=scnd** but not $\mathbf{last_{n-1} = last_n}$ like with

[3, **1, 2**, 4] →	[3, 3, 1, 2, 4, 4]	rule1

.

Hence, rule3 is not applicable to [3, **1, 2**, 4]

An application of rule3 on (1,2) produces a non-palindromic morphogram.

[2,1,1,2] --> [2,2,1,1,2,3]: rule4

An application of rule4(2,2) of [2,1,1,2] produces a non-palindromic morphogram.

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**] : is missing.

Consequences

As a consequence, the list constructs **List(x.head)** and **List(x.last)** are much too static and have to be replaced by a more dynamic (functional) approach to realize the *recursive* application of the rules on the length of the words.

Additionally, some *context rules* that are restricting the use of the rules have to be implemented.

Because of possible redundancy, an elimination procedure for redundant items has to be added too.

Towards a re-formulation of the rules

Length of the word
Type: Odd, Even
w**ord scheme**: w = $[w_1, w_2, \ldots, w_i, w_j, \ldots, w_{n-1}, w_n]$
middle of odd word:

```
def isOdd (i : Int) : Boolean = i % 2 != 0
def middle (s : List[Int]) = s ( s.size / 2)
```

Tuple notation
word tuple scheme: w = $[(w_1, w_n), (w_2, w_{n-1}), \ldots, (w_i w_j), \ldots, (w_{n-1}, w_2), (w_n, w_1)]$
Re-enummeration: ∀i,j: n ==> n+2, 2<=i,j<=n ==> 2<=i,j<=n+2
$[w_1, w_2, \ldots, w_i, w_j, \ldots, w_{n-1}, w_n]$ ==> $[w_1, w_2, \ldots, w_i, w_j, \ldots, w_{n-1}, w_{n+2}]$

New Set of Rules

Production rules for morphic palindromes

rule1 (i, j) :

$$\frac{\left[\left(w_1, w_n\right), \left(w_2, w_{n-1}\right), \ldots, \left(w_i w_j\right), \ldots, \left(w_{n-1}, w_2\right), \left(w_n, w\right.\right.}{\left[\left(w_i, w_j\right), \left(w_1, w_{n+2}\right), \left(w_2, w_{n+1}\right), \ldots, \left(w_i w_j\right), \ldots, \left(w_{n+1}, w_2\right), \left(w_{n+2},\right.\right.}$$

rule2 (i, j) :

$$\frac{\left[\left(w_1, w_n\right), \left(w_2, w_{n-1}\right), \ldots, \left(w_i w_j\right), \ldots, \left(w_{n-1}, w_2\right), \left(w_n, w\right.\right.}{\left[\left(w_j, w_i\right), \left(w_1, w_n\right), \left(w_2, w_{n-1}\right), \ldots, \left(w_i w_j\right), \ldots, \left(w_{n-1}, w_2\right), \left(w_n, w\right.\right.}$$

rule3 (i, j) :

$$\frac{\left[\left(w_1, w_n\right), \left(w_2, w_{n-1}\right), \ldots, \left(w_i w_j\right), \ldots\right.}{\left[\left(\left|w_j\right|+1, \left|w_j\right|+1\right), \left(w_1, w_n\right), \left(w_2, w_{n-1}\right), \ldots, \left(w_i w_j\right), \ldots\right.}$$

rule4 (i, j) :

$$\frac{\left[\left(w_1, w_n\right), \left(w_2, w_{n-1}\right), \ldots, \left(w_i w_j\right), \ldots,\right.}{\left[\left(\left|w_i\right|+1, \left|w_i\right|+2\right), \left(w_1, w_n\right), \left(w_2, w_{n-1}\right), \ldots, \left(w_i w_j\right), \ldots\right.}$$

rule5 (middle) :

$$\frac{\left[\left(w_1, w_n\right), \left(w_2, w_{n-1}\right), \ldots, \left(w_{mid}\right), \ldots, \left(w\right.\right.}{\left[\left(w_{mid}, w_{mid}\right), \left(w_1, w_n\right), \left(w_2, w_{n-1}\right), \ldots, \left(w_{mid}\right), \ldots, \left(w\right.\right.}$$

Context Rule CR

rule 3 $(w_i, w_j) : (w_i w_j) \bigcap (w_1, w_n) = \emptyset$

$$\frac{\left[(w_1, w_n), (w_2, w_{n-1}), \ldots, (w_i w_j), \ldots, (w_{n-1}, w_2), (w_n, w_1)\right]}{\left[(w_i w_j), (w_1, w_n), (w_2, w_{n-1}), \ldots, (w_i w_j), \ldots, (w_{n-1}, w_2), (w_n, w_1)\right]}$$

In general :

CR : $(w_i, w_j) = (w_1, w_n)$ or $(w_i, w_j) \bigcap (w_1, w_n) = \emptyset$

Examples

P = [1, 2, 3, 4, 5, 6], rule1(i, j):

rule1(3, 4): [3, 1, 2, **3, 4**, 5, 6, 4] → [1, 2, 3, 1, 4, 5, 6, 4]

rule1(2, 5): [2, 1, **2**, 3, 4, **5**, 6, 5] → [1, 2, 1, 3, 4, 5, 6, 5]

rule1(1, 6) : [1, **1**, 2, 3, 4, 5, **6**, 6] → [1, 1, 2, 3, 4, 5, 6, 6]

P = [1, 2, 3, 4, 5, 6]:
length(P) = 6
type: even
re-nummeration: n ==> n+2
form: linear or tuple
linear: [w_1, w_2, ,w_3, w_4, w_5, w_6]
tupel :[(w1, w6), (w2,w5), (w3, w4)]

rule scheme, tuple form:

$$[(w1, w6), (w2, w5), (w3, w4)] \Longrightarrow [(w1, w8), (w2, w7), (w3, w6), (w4, w5)]$$

or:

$$\frac{\left[(w1, w6), (w2, w5), (w3, w4)\right]}{\left[(w1, w8), (w2, w7), (w3, w6), (w4, w5)\right]}$$

Rules

rule1(3,4): [(w1, w6), (w2,w5), (w3, w4)]
==>
[**(w3, w4)**, (w1, w6), (w2,w5), (w3, w4)]
[(w1, w8), (w2, w7), (w3,w6), (w4, w5)] : re-enumeration

rule1(3,4): [(1, 6), (2, 5), (3, 4)]
==>
[**(3, 4)**], (1, 6), (2, 5), (3, 4)] ==> [**3**,1,2,3,4,5,6,**4**] : (3, 4) ∩ (1, 6) = ∅

rule1(2,5): [(w1, w6), (w2,w5), (w3, w4)]
==>
[(w2, w5), (w1, w6), (w2,w5), (w3, w4)]

rule1(2,5): [(1, 6), (2, 5), (3, 4)]
==>
[(**2, 5**), (1, 6), (2, 5), (3, 4)]] ==> [**2**,1,2,3,4,5,6,**5**] : (2, 5) ∩ (1, 6) = ∅

rule1(1,6): [(w1, w6), (w2,w5), (w3, w4)]
==>
[(w1, w6), (w1, w6), (w2,w5), (w3, w4)]

rule1(1,6): [(1, 6), (2, 5), (3, 4)]
==>

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

Restriction for rule 3

rule3(3,4): [(1, 6), (2, 5), (3, 4)]
==>
[**(5, 5)**], (1, 6), (2, 5), (3, 4)] ==> [**5**,1,2,3,4,5,6,**5**] : (5,5) ∩ (1,6) = ⌀

rule3(2,5): [(1, 6), (2, 5), (3, 4)]
==>
[**(6, 6)**], (1, 6), (2, 5), (3, 4)] ==> [**6**,1,2,3,4,5,6,**6]** : (6,6) ∩ (1,6) != ⌀

rule3(1,6): [(1, 6), (2, 5), (3, 4)]
==>
[**(7, 7)**], (1, 6), (2, 5), (3, 4)] ==> [**7**,1,2,3,4,5,6,**7**] : : (7,7) ∩ (1,6) = ⌀

Mismatch:
rule3(1,2):
[3,1,2,4] ⟹ [3, 3, **1, 2**, 4, 3] ==> [**(3,3)**, **(3,4)**, (1,2)] : normal form, test: (3,3) ∩ (3,4) != ⌀

rule3(2,3):
[1,2,3,4] ⟹ [4, 1, **2, 3**, 4, 4] ==> [**(4,4**), **(1,4)**, (2,3)] : numeric form, test: (4,4) ∩ (1,4) != ⌀

rule4(1,1)
[2,1,1,2] ⟹ [2,2,1,1,2,3] ⟹ [(2,3), (2,2), (1,1)] : (2,3) ∩ (2,2) != ⌀

rule3(1,1):
[2,1,1,3] ⟹ [2,2,1,1,3,2] ⟹ [(2,2), (2,3), (1,1)] : (2,2) ∩ (2,3) != ⌀

rule2(2,3):
[2,1,1,3] ⟹ [3,2,1,1,3,2] ⟹ [(3,2), (2,3), (1,1)] : (3,2) ∩ (2,3) = ⌀

Context Rule for genPalindrome

```
def rule3(x:List[Int]) = {
   (0 until x.size/2).flatMap(i => genP(List(x(x.size -i-1)+1) ++ x ++ List(x(x.size -i-1)+1) ) )
```

rule3(2,5): [(1, 6), (2, 5), (3, 4)]
==>
[**(6, 6)**], (1, 6), (2, 5), (3, 4)] ==> [**6**,1,2,3,4,5,6,**6]** : (6,6) ∩ (1,6) != ⌀

x = [(1, 6), (2, 5), (3, 4)]
List(x(x.size -i-1)+1) = 6
List(x(x.size -i-1)+1) = 6
head(x) = 1,
last(x) = 6
hence,
(List(x(x.size -i-1)+1) , head(x)) ∩ (List(x(x.size -i-1)+1) , last(x)) != ⌀
is (6, 1) ∩ (6, 6) != ⌀.
Therefore, [**6**,1,2,3,4,5,6,**6]** or [**(6, 6)**], **(1, 6)**, (2, 5), (3, 4)] is not a palindrome.

Context rules for genPalindrome (approx.)

$$\text{def CR} - \text{rule3}(x : \text{List[Int]}) = (\text{List}(x(x.\text{size} - i - 1) + 1), \text{head}(x)) \cap (\text{List}(x(x.\text{size} - i - 1) + 1), \text{last}(x)) = \emptyset$$

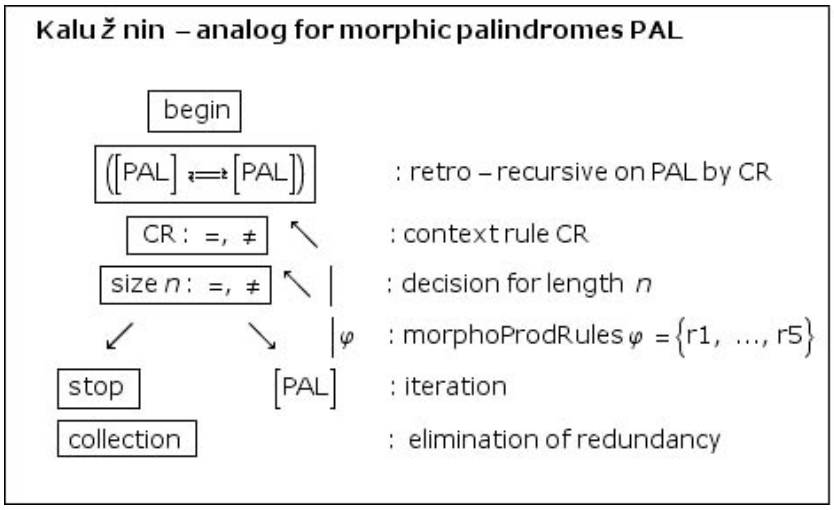
$$\text{def CR} - \text{rule4}(x : \text{List[Int]}) = (\text{List}(x(x.\text{size} - i - 1) + 1), \text{head}(x)) \cap (\text{List}(x(x.\text{size} - i - 1) + 2), \text{last}(x)) = \emptyset$$

In general:

$$\text{def CR} - \text{rule}(x : \text{List[Int]}) = (\text{List}(\text{rule}), \text{head}(x)) \cap (\text{List}(\text{rule}), \text{last}(x)) = \emptyset$$

2.2.5. Recursion and reflection for palindromes

Hence, the retrograde recursion over the morphic palindromes gets a meta-rule, context rule CR, which controls the palindrome-procedures in retrograde respect of the structure of the prolonged palindrome. Therefore, the *morphoProdRules* $\varphi = \{r1, \ldots, r5\}$ are not just directly applied on the palindromes but are also reflecting the structure of the produced palindromes by the context rule to decide its further application.



Some Palindrome productions

begin : [bac], length (6), context rule , stop : (accept, reject), collect

$$[\text{bac}] \xrightarrow{\varphi=r1} [\text{bbacc}] \xrightarrow{\varphi=r1} [\text{bbbaccc}] \in CR \Longrightarrow [\text{accept}],\ n = 6,\ \text{stop}$$

$$\downarrow \text{repeat}$$

$$[\text{bbacc}] \xrightarrow{\varphi=r2} [\text{cbbaccb}] \in CR \Longrightarrow [\text{accept}]$$

$$\downarrow \text{repeat}$$

$$[\text{bbacc}] \xrightarrow{\varphi=r3} [\text{dbbaccd}] \in CR \Longrightarrow [\text{accept}]$$

$$\downarrow \text{repeat}$$

$$[\text{bbacc}] \xrightarrow{\varphi=r4} [\text{dbbacce}] \in CR \Longrightarrow [\text{accept}]$$

$$\downarrow \text{repeat}$$

$$[\text{bbacc}] \xrightarrow{\varphi=r5} [\text{abbacca}] \in CR \Longrightarrow [\text{accept}]\ .$$

$$[\text{bac}] \xrightarrow{\varphi=r2} [\text{cbacb}] \xrightarrow{\varphi=r1} [\text{ccbacbb}] \in CR \Longrightarrow [\text{accept}]$$

$$\downarrow \text{repeat}$$

$$[\text{cbacb}] \xrightarrow{\varphi=r2} [\text{bcbacbc}] \in CR \Longrightarrow [\text{accept}]$$

$$\downarrow \text{repeat}$$

$$[\text{cbacb}] \xrightarrow{\varphi=r4(b,c)} [\text{ccbacbd}] \notin CR \Longrightarrow [\text{reject}]\ \#\#$$

$$\downarrow \text{repeat}$$

$$[\text{cbacb}] \xrightarrow{\varphi=r4} [\text{dcbacbe}] \in CR \Longrightarrow [\text{accept}]$$

$$\downarrow \text{repeat}$$

$$[\text{cbacb}] \xrightarrow{\varphi=r5} [\text{acbacba}] \in CR \Longrightarrow [\text{accept}].$$

$$[\text{bac}] \xrightarrow{\varphi=r3} [\text{cabad}] \xrightarrow{\varphi=r1} [\text{ccabadd}] \in CR \Longrightarrow [\text{accept}]$$

$$\downarrow \text{repeat}$$

$$[\text{cabad}] \xrightarrow{\varphi=r2} [\text{dcabadc}] \in CR \Longrightarrow [\text{accept}]$$

$$\downarrow \text{repeat}$$

$$[\text{cabad}] \xrightarrow{\varphi=r3} [\text{e cabade}] \in CR \Longrightarrow [\text{accept}]$$

$$\downarrow \text{repeat}$$

$$[\text{cabad}] \xrightarrow{\varphi=r4} [\text{ecabadf}] \in CR \Longrightarrow [\text{accept}]$$

$$\downarrow \text{repeat}$$

$$[\text{cbacb}] \xrightarrow{\varphi=r5} [\text{a cabada}] \in CR \Longrightarrow [\text{accept}].$$

$$[\text{bac}] \xrightarrow{\varphi=r4} [\text{dbace}] \xrightarrow{\varphi=r1} [\text{ddbacee}] \in CR \Longrightarrow [\text{accept}]$$

$$\downarrow \text{repeat}$$

Morphogrammatics and Computational Reflection
Applying insights from the retro-grade recursivity concept of morphogrammatics to questions of reflectionality and interactionality of programming

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

2.2.6. Reformulation of the Scala program genPalindrome (2nd version)

This reformulation is considering the recursive application of the rules on the length of the word.
The *context* rule is not yet implemented.

```
object Palindrome {
def morphoRules(s:List[Int]) : Boolean =
  ((s.head == s.last) ||
  (s.head+1 == s.last) ||
  (s.head+1 == s.last+1) ||
  (s.head+1 == s.last+2) )
def isPalindrome(s : List[Int]) : Boolean =
  s match {
    case s if s.length > 1 => morphoRules(s) && isPalindrome(s.slice(1, s.length - 1))
    case _ => true
  }
def isOdd(i:Int) : Boolean = i % 2 != 0
def middle(s:List[Int]) = s( s.size / 2)
def genPalindrome1(maxSize:Int, s:List[Int] = Nil) = {
  def genP(s:List[Int]) : Set[List[Int]] = {
    s match {
      case x if x.size >= maxSize -1 => Set(x)
      case x if x.isEmpty =>  genP(List(1)) ++ genP(List(1,1)) ++ genP(List(1,2))
      case x =>
         Set(x) ++
        genP(List(x.head) ++ x ++ List(x.last)) ++
        genP(List(x.last) ++ x ++ List(x.head)) ++
        genP(List(x.last+1) ++ x ++ List(x.last+1)) ++
        genP(List(x.last+1) ++ x ++ List(x.last+2)) ++
        (if (isOdd(x.size)) genP(List(middle(x)) ++ x ++ List(middle(x))) else Nil)
    }
  }
  genP(s)
}
def genPalindrome(maxSize:Int, s:List[Int] = Nil) = {
  def rule1(x:List[Int]) = {
    (0 until x.size/2).flatMap(i => genP(List(x(i)) ++ x ++ List(x(x.size -i-1)) ) )
  }
  def rule2(x:List[Int]) = {
    (0 until x.size/2).flatMap(i => genP(List(x(x.size -i-1)) ++ x ++ List(x(i)) ) )
  }
  def rule3(x:List[Int]) = {
    (0 until x.size/2).flatMap(i => genP(List(x(x.size -i-1)+1) ++ x ++ List(x(x.size -i-1)+1) ) )
  }
  def rule4(x:List[Int]) = {
    (0 until x.size/2).flatMap(i => genP(List(x(x.size -i-1)+1) ++ x ++ List(x(x.size -i-1)+2) ) )
  }
  def genP(s:List[Int]) : Set[List[Int]] = {
    s match {
      case x if x.size >= maxSize -1 => Set(x)
      case x if x.isEmpty =>  genP(List(1)) ++ genP(List(1,1)) ++ genP(List(1,2))
      case x =>
        Set(x) ++
        rule1(x) ++ rule2(x) ++ rule3(x) ++ rule4(x) ++
        (if (isOdd(x.size)) genP(List(middle(x)) ++ x ++ List(middle(x))) else Nil)
```

```
    }
  }
  genP(s)
}
def prnt(s:Set[List[Int]]) =  s.map("[" + _.mkString(",") + "]").toList.sorted.mkString("\n")
// prnt(genPalindrome(5))
// genPalindrome(5).map(x => x +":"+  isPalindrome(x).toString)
def main(args:Array[String]) {
  val len = if(args.length >0) args(0).toInt else 5
  val pal = genPalindrome(len)
   println("genPalindrome("+len+") => "+pal.size+"..\n" + prnt(pal))
}
}
// println("genPalindrome("+6+")..\n" + prnt(genPalindrome(6)))
```

2.2.7. Scala program with context rule CR (3rd version)