Number Sequences in trans-Classical Systems

Second-order Sequences of Fibonacci, Factorial and Stirling numbers

Rudolf Kaehr Dr.phil
 ISSN 2041-4358

Abstract

The project of trans-Classical systems as it was introduced by Gotthard Gunther as a theory of keno- and morphogrammatics was always understood as a second-order theory in the original sense of the attempts of Second-Order Cybernetics.
Nevertheless, there is no theoretical work up to now that risks a second-order arithmetics as it is prospected by the original endeavor of kenogrammatics.
This paper takes some preliminary steps to introduce second-order concepts of well known numerical sequences, like Factorial, Fibonacci and Stirling number sequences.
Hence, a kind of a polycontextural arithmetics is proposed that differs from the early approaches to trans-Classical number theory as they had been introduced originally by Gunther.
Firstly, there are polycontextural concepts of morphogrammatic sequences, and secondly, there are classical number sequences distributed and mediated over trans-Classical systems.
With a further step towards a second order arithmetics, new concepts and rules, unknown to the classical paradigm, have to be introduced. A connection between the second-order analysis of Stirling numbers and the concept, formulas and algorithm of the “refined”, multinomial coefficients of partitions is established. Some programming procedures in Standard ML are added.
(work in progress, vers. 0.4, Nov. 2013)

1. Trans-Classical Number Sequences 2.0

1.1. Number sequences in kenogrammatic systems

1.1.1. Trans-Classical systems

> *"Theories of trans-classic logic and many-valued ontologies are introduced for the very purpose of showing* not *the "essential unity" but the* essential differences *in the concept of a machine and what Wiener called the living tissue. This is crass heresy in the High Church of Cybernetics."* (G. Gunther)

In the late 60s Gotthard Gunther started the study of Peano sequences in trans-

Classical sytems. Trans-Classical systems had been defined as kenogrammatic systems.

Gotthard Günther, Natural Numbers in Trans-Classical Systems, Part I

> *"So far cybernetics has looked on living systems overwhelmingly from the view-point of **evolution** and, seen from here, the development of these systems seems – as we pointed out – to tend toward higher and higher integrated forms of unity and wholeness, implementing the principle of a "transcendental" superadditivity.*
> *On the other hand, if we look at the phenomenon Life as: a result of **emanation**, exactly the opposite type of properties seems to govern the development. Emanatively speaking, the development of systems of higher and higher organic complexity seems to accentuate a tendency towards internal disunity and disintegration. Seen from here the development seems to be guided by a principle which we might call that of supersubtractivity."*

http://www.vordenker.de/ggphilosophy/gg_natural-numbers.pdf

The mathematcis of trans-Classical systems is mainly based on the Stirling numbers of the second kind as they had been programmed and printed out by Alex M. Andrew at the BCL in the December of 1965.

An interesting property of the behavior of kenogrammatically based Peano sequences was the discovery of *mediation* numbers between the classical *cardinal* and *ordinal* numbers.

Number sequences in kenogrammatic systems are considered as distributed and mediated number sequences, hence representing a strong kind of parallelism and interaction between number systems.

Some features of classical sequences, like Peano, Fibonacci and Factorials, are sketched in this approach.

Thus the new situations are e.g., Fibonacci of Stirling, i.e. Fibonaccci sequences of Strirling defined morphograms.

Therefore, the combinatorics of such an approach are not defined by numbers and their functions but by *morphograms* that are themselves defined by Stirling numbers.

An interesting turn is given with an application of Stirling numbers onto themselves. As we know, morphograms of the trito-structure of graphematics are defined by the Stirling numbers Sn2 produced by the recursive formula for Sn.

It is natural to apply the same formula onto the Stirling based Tcontextures too.

Hence, we get a Stirling formula of the Stirling Tcontextures, producing a second-order sequence of Stirling based Tcontextures.

The present study is focused mainly on the '*evolutive*' aspect of kenogrammatics, and is not yet taking the '*emanative*' of patterns, and the zigzagging interaction between evolution and emanation into account.

An interesting new property of disseminated number systems is detected as the non-commutativity of its basic operations like 'addition' and 'multiplication'.

Therefore, to each kenogrammatic number sequence there exists a dual number system based on the duality of the non-commutativity of its operations.

This might sound trivial because disseminated number sequences are based on morphogrammatic patterns, tuples, and not on isolated numbers.

Classical operations in Peano number theory are, trivially, commutative.

What's the meaning of non-commutative kenogrammatic number sequences?

Retrograde recursivity of the successor operations

Again, what can we understand by a kenogram, a morphogram and its 'successor' functions?
A citation is a form of iteration, hence a just found note from the 1970s might contribute to elucidate the darkness of kenomics.

$$E^{(\nu)}(\alpha) = \alpha \text{ ^ } Z_\nu,\ Z_\nu \in zg \ \rightleftarrows\ u \cong v,\ X \cong Y \Longrightarrow Xu \cong Yv$$

"$E^{(\nu)}(\alpha)$ ist nicht eine abstrakte Nachfolgeropration, die zu (α) eine beliebige im K_{zg} produzierte Einheit, d.h. ein Zeichen hinzufügt. $\{Z_\nu\}$ wird durch die Struktur von (α) bestimmt. Es geht somit nicht um eine Verkettung, Konkatenation von vorgegebenen Zeichen, sondern um die Wiederholung, Unterstützung eines Elements in (α) oder der Hinzufügung eines neuen Elements. Da nur die Gleichheit oder Diversität bzw. das Unterstützen oder nicht Unterstützen eines Zeichens (einer Kontextur) das sich auf einem bestimmten Platz in der KG-Sequenz befindet, handelt, kommt es auf die semiotische Indivitualität der Indikatoren der Kenogramme nicht an, sie soll "übersehen" werden.
Die semiotische Indivitualität von Z_ν ist somit nicht primär. Primät ist, ob Z_ν ein Kenogramm wiederholt oder nicht oder ob Z_ν ein von allen KGs der KGSeq verschiedenes KG hinzufügt.
?? Daher ist es ausreichend, die Morphogramme in der Normalform zu untersuchen."

1.1.2. Non-commutativity and non-associativity

John Baez, Categories, Quantization, and Much More (2006)

"In quantum theory one thus learns to like noncommutative, but still associative, algebras.
It is interesting however to note why associativity without commutativity is studied so much more than commutativity without associativity.

Composition is always associative so the + operation is associative!
If we try to generalize the heck out of the concept of a group, keeping associativity as a sacred property, we get the notion of a category."

http://math.ucr.edu/home/baez/categories.html

Non-commutativity

At a first glance, they are establishing a duality between two different number sequences that are defined in the same general framework, say of Fibonacci

sequences, but are non-commutative by their basic operations of ‘addition’ or ‘mutiplication'.

Therefore, the general formula for Fibonacci sequences, $\mathbf{F_n = F_{n-1} +}_{\text{com}}$ $\mathbf{F_{n-2}}$, gets a dual formulation with $\mathbf{F_n = F_{n-2}+}_{\text{non-com}}$ $\mathbf{F_{n-1}}$, i.e. morphogrammatic Fibonacci sequences are defined as the two parallel sequences:

$$\text{mgFib [F]} = \begin{pmatrix} \mathbf{F}_{n-1}+\mathbf{F}_{n-2} \\ \\ \mathbf{F}_{n-2}+\mathbf{F}_{n-1} \end{pmatrix}.$$

This shall be depicted by the diagram for dual morphogrammatic Fibonacci sequences with the tuples $x_i = [n_1,n_2,...,n_{i-1},n_i]$.

x_5	x_6	x_7	x_8	x_9	x_{10}
5	**8**	**13**	**21**	34	55
□	⟵	⟵	$x_8 = x_7 + x_6$	$\boldsymbol{F}_{n-1}+\boldsymbol{F}_{n-2}$	□
□	⟶	⟶	$\overline{x}_8 = x_6 + x_7$	$\boldsymbol{F}_{n-2}+\boldsymbol{F}_{n-1}$	□

Table: x^[1] **Dual table: [1]^[x]**

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

Conceived as sets of sequences only, both systems are equal.

Especially, [1,1]+[1] = [1,1,2] is different to [1]+[1,1] = [1,2,2], and the asymmetry [1,2]+[1] = [1]+[1,1] differs too.

Non-commutativity procedure

- allkconcat (Tcontexture 2)(Tcontexture 3) = allkconcat(Tcontexture 3)(Tcontexture 2);
val it = false : bool

- map flat (allkconcat (Tcontexture 2)(Tcontexture 3)) = map flat (allkconcat (Tcontexture 3)(Tcontexture 2));
val it = false : bool

Non-associativity of kconcat

(1.) kconcat:

- kconcat[1,1][1,1];
val it = [[1,1,1,1],[1,1,2,2]]
- allkconcat[[**1,1**]][[1,1,1,1],[1,1,2,2]]; (a)
val it =
[[[1,1,1,1,1,1],[1,1,2,2,2,2]],[[1,1,1,1,2,2],[1,1,2,2,1,1],[1,1,2,2,3,3]]]
- allkconcat[[1,1,1,1],[1,1,2,2]][[**1,1**]]; (b)
val it =
[[[1,1,1,1,1,1],[1,1,1,1,2,2]],[[1,1,2,2,1,1],[1,1,2,2,2,2],[1,1,2,2,3,3]]]

Indexed kconcat

$\text{allkconcat}_i(a) = \text{allkconcat}_i(b)$, i=1,5 :[1,1,1,1,1,1]

allkconcat$_2$(a) = allkconcat$_3$(b) :[1,1,2,2,2,2]
allkconcat$_3$(a) = allkconcat$_2$(b) :[1,1,1,1,2,2]
allkconcat$_4$(a) = allkconcat$_3$(b) :[1,1,2,2,1,1] .

(2.) kconcat:
[[1,1]] [[1,2][1,1]] =? [[1,1][1,2]] [[1,1]]

- kconcat[1,2][1,1];
val it = [[1,2,1,1],[1,2,2,2],[1,2,3,3]]

- allkconcat[[**1,1**]][[1,2,1,1],[1,2,2,2],[1,2,3,3]];
val it =
[[[1,1,1,2,1,1],[1,1,2,1,2,2],[1,1,2,3,2,2]],
[[1,1,1,2,2,2],[1,1,2,1,1,1],[1,1,2,3,3,3]],
[[1,1,1,2,3,3],[1,1,2,1,3,3],[1,1,2,3,1,1],[1,1,2,3,4,4]]]

- allkconcat[[1,2,1,1],[1,2,2,2],[1,2,3,3]][[**1,1**]];
val it =
[[[1,2,1,1,1,1],[1,2,1,1,2,2],[1,2,1,1,3,3]],
[[1,2,2,2,1,1],[1,2,2,2,2,2],[1,2,2,2,3,3]],
[[1,2,3,3,1,1],[1,2,3,3,2,2],[1,2,3,3,3,3],[1,2,3,3,4,4]]]

Remark

The method to use indexed operations, kconcat$_i$, to adjust the 'additions' to associativity, is not working in the second case. Both sets of results are disjunct.

In the case of classical *word algebra*, commutativity of the concatenation operation doesn't hold, but associativity is accepted.

That is, "*xy is not guaranteed to be equal to yx*" but "*concatenation is associative • (xy)z= x(yz) = xyz .*"

Resumé at: CSE 460 Lectures
http://www.cse.msu.edu/~torng/Classes/Archives/cse460.03spring/Lectures/module0.pdf

The same holds for the 'multiplication' *kmul* too.

Non-commutativity of kmul
Head of kmul:

fun kmul [] b = [[]]
|kmul a [] = [[]]
|kmul a [1] = [a]
|kmul [1] b = [b]
But kmul ab ≠ kmul ba.

Examples

- flat(allkmul(Tcontexture 3)(Tcontexture 2)) = flat(allkmul(Tcontexture 2)(Tcontexture 3));
val it = false : bool

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

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

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

- flat(kmul[1,2,1][1,2]) = flat(kmul[1,2][1,2,1]);
val it = false : bool

val it = [1,2,1,2,1,2,1,2,1,3,1,3,1,2,1,2,3,2,1,2,1,3,4,3] : int list
- length it;
val it = 24 : int

- flat(kmul[1,2][1,2,1]);
val it = [1,2,2,1,1,2,1,2,3,1,1,2,1,2,2,3,1,2,1,2,3,4,1,2] : int list

But:
- **length**(flat(kmul[1,2][1,2,1])) = length(flat(kmul[1,2,1][1,2]));
val it = true : bool

Non-associativity of kmul

allkmul [[1,2][1,2,2]] [[1,1,2]] = allkmul [[1,2]] [[1,2,2][1,1,2]]
kmul[1,2][1,2,2];
val it = [[1,2,2,1,2,1],[1,2,3,1,3,1],[1,2,2,3,2,3],[1,2,3,4,3,4]]
allkmul[[1,1,2]][[1,2,2,1,2,1],[1,2,3,1,3,1],[1,2,2,3,2,3],[1,2,3,4,3,4]]

- kmul[1,2,2][1,1,2];
val it =
[[1,2,2,1,2,2,2,1,1],[1,2,2,1,2,2,3,1,1],[1,2,2,1,2,2,2,3,3],[1,2,2,1,2,2,3,4,4]]
- allkmul
[[1,2]][[1,2,2,1,2,2,2,1,1],[1,2,2,1,2,2,3,1,1],[1,2,2,1,2,2,2,3,3],[1,2,2,1,2,2,3,4,4]];

- flat(allkmul[[1,2]]
[[1,2,2,1,2,2,2,1,1],[1,2,2,1,2,2,3,1,1],[1,2,2,1,2,2,2,3,3],[1,2,2,1,2,2,3,4,4]])
= flat(allkmul[[1,1,2]][[1,2,2,1,2,1],[1,2,3,1,3,1],[1,2,2,3,2,3],[1,2,3,4,3,4]]);
val it = false : bool

Non-distributivity of kmul and kconcat

Distributivity: (x + y)z = xz + yz , z (x + y) = z x + z y

Example

x = z= [1,2], y = [1,1] : ([1,2] + [1,1]) * [1,2] =? [1,2]*[1,2] + [1,1]*[1,2]: ??

```
- kconcat[1,2][1,1];
val it = [[1,2,1,1],[1,2,2,2],[1,2,3,3]]

- allkmul[[1,2,1,1],[1,2,2,2],[1,2,3,3]] [[1,2]];
val it =
  [[[1,2,1,1,2,1,2,2],[1,2,1,1,3,1,3,3],[1,2,1,1,2,3,2,2],[1,2,1,1,3,4,3,3]],
   [[1,2,2,2,2,1,1,1],[1,2,2,2,3,1,1,1],[1,2,2,2,2,3,3,3],[1,2,2,2,3,4,4,4]],
   [[1,2,3,3,2,3,1,1],[1,2,3,3,3,1,2,2],[1,2,3,3,2,1,4,4],[1,2,3,3,2,4,1,1],
    [1,2,3,3,4,1,2,2],[1,2,3,3,3,1,4,4],[1,2,3,3,3,4,1,1],[1,2,3,3,4,3,1,1],
    [1,2,3,3,4,1,5,5],[1,2,3,3,4,5,1,1],[1,2,3,3,2,3,4,4],[1,2,3,3,3,4,2,2],
    [1,2,3,3,4,3,2,2],[1,2,3,3,2,4,5,5],[1,2,3,3,4,5,2,2],[1,2,3,3,3,4,5,5],
    [1,2,3,3,4,3,5,5],[1,2,3,3,4,5,6,6]]]

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

- allkmul[[1,2]][[1,1,1,2],[1,1,2,1],[1,1,2,3]] ;
val it =
  [[[1,2,1,2,1,2,2,1],[1,2,1,2,1,2,3,1],[1,2,1,2,1,2,2,3],[1,2,1,2,1,2,3,4]],
   [[1,2,1,2,2,1,1,2],[1,2,1,2,3,1,1,2],[1,2,1,2,2,3,1,2],[1,2,1,2,3,4,1,2]],
   [[1,2,1,2,2,1,3,4],[1,2,1,2,3,1,2,3],[1,2,1,2,3,1,2,4],[1,2,1,2,3,1,4,3],
    [1,2,1,2,3,1,4,5],[1,2,1,2,2,3,3,1],[1,2,1,2,2,3,4,1],[1,2,1,2,2,3,3,4],
    [1,2,1,2,2,3,4,5],[1,2,1,2,3,4,2,1],[1,2,1,2,3,4,4,1],[1,2,1,2,3,4,5,1],
    [1,2,1,2,3,4,2,3],[1,2,1,2,3,4,2,5],[1,2,1,2,3,4,4,3],[1,2,1,2,3,4,5,3],
    [1,2,1,2,3,4,4,5],[1,2,1,2,3,4,5,6]]]

 kmul[1,2][1,2];
val it = [[1,2,2,1],[1,2,3,1],[1,2,2,3],[1,2,3,4]]

- kmul[1,1][1,2];
val it = [[1,1,2,2]] : int list list

- allkconcat[[1,1,2,2]][[1,2,2,1],[1,2,3,1],[1,2,2,3],[1,2,3,4]] ;
val it =
  [[[1,1,2,2,1,2,2,1],[1,1,2,2,2,1,1,2],[1,1,2,2,1,3,3,1],[1,1,2,2,3,1,1,3],
    [1,1,2,2,2,3,3,2],[1,1,2,2,3,2,2,3],[1,1,2,2,3,4,4,3]],
   [[1,1,2,2,1,2,3,1],[1,1,2,2,1,3,2,1],[1,1,2,2,2,1,3,2],[1,1,2,2,2,3,1,2],
    [1,1,2,2,3,1,2,3],[1,1,2,2,3,2,1,3],[1,1,2,2,1,3,4,1],[1,1,2,2,3,1,4,3],
    [1,1,2,2,3,4,1,3],[1,1,2,2,2,3,4,2],[1,1,2,2,3,2,4,3],[1,1,2,2,3,4,2,3],
    [1,1,2,2,3,4,5,3]],
   [[1,1,2,2,1,2,2,3],[1,1,2,2,1,3,3,2],[1,1,2,2,2,1,1,3],[1,1,2,2,2,3,3,1],
    [1,1,2,2,3,1,1,2],[1,1,2,2,3,2,2,1],[1,1,2,2,1,3,3,4],[1,1,2,2,3,1,1,4],
    [1,1,2,2,3,4,4,1],[1,1,2,2,2,3,3,4],[1,1,2,2,3,2,2,4],[1,1,2,2,3,4,4,2],
    [1,1,2,2,3,4,4,5]],
   [[1,1,2,2,1,2,3,4],[1,1,2,2,1,3,2,4],[1,1,2,2,1,3,4,2],[1,1,2,2,2,1,3,4],
    [1,1,2,2,2,3,1,4],[1,1,2,2,2,3,4,1],[1,1,2,2,3,1,2,4],[1,1,2,2,3,1,4,2],
    [1,1,2,2,3,2,1,4],[1,1,2,2,3,2,4,1],[1,1,2,2,3,4,1,2],[1,1,2,2,3,4,2,1],
    [1,1,2,2,1,3,4,5],[1,1,2,2,3,1,4,5],[1,1,2,2,3,4,1,5],[1,1,2,2,3,4,5,1],
```

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

1.1.3. Parmetrization contextures

The simplest approach to an application of combinatorial formulas to define contextural sequences is given by the *parametrization* of the numerical function of the contextures.
The definition of contextures, say Tcontexture, implies a numeric value, hence *Tcontexture(n)* to realize the list of the contextures of complexity n.

This numerical parameter is accessible to a numeric definition that allows to apply combinatorial identities, like Fibonacci, Powers, sumUpTo, Factorials, and Stirling formulas.

The advantage is that the whole maneuver to develop a polycontextural combinatorics is moved to its parameters, and is not manipulating the burden of the complex lists of contextures represented by mophograms.

Hence, the general scheme for parametrized contextures is is given by the construct:

$$\text{Tcontexture}\left(\text{operation}\left(\text{n}\right)\right)$$

This construct answers the *question*: What is the polycontexturality of ‘operation(n)'?

E.g., what is the polycontexturality of the Fibonacci number 3:

Answer: Tcontexture(fib 3);
val it = [[1,1,1],[1,1,2],[1,2,1],[1,2,2],[1,2,3]] : int list list

The classical *question* is: What is the Fibonacci number for the natural number 3?

Answer: fib 3 = 1+2 = 3.

Therefore, instead of an addition of contextures for *kconcat*, just an addition of its parameters is implemented.

The contextural function **allTcontextureKconcat** *m n* is replaced by combinatorial function **allTcontextureKconcatNum** *m n,* i.e defined as **Tcontexture**(m+n).

fun **allTcontextureKconcat** n m = allkconcat(Tcontexture(n))(Tcontexture(m));
val allTcontextureKconcat = fn : int -> int -> int list list list

fun **allTcontextureKconcatNum** n m = Tcontexture(n+m);
val allTcontextureKconcatNum = fn : int -> int -> int list list

Are both approaches equivalent?

Obviously not!
fn : int -> int -> int list list list is not equal *fn : int -> int -> int list list*

The procedure **allTcontextureKconcat** has a more informative list structure than the direct numerical procedure **allTcontextureKconcatNum**.

The numeric approach has its practical advantages but is relying directly on classical combinatorics and is not emphasizing the contextural structures of the morphogrammatic approach.

The contextural approach is directly dealing with its morphogrammatic deep-structure and is relying on numeric combinatorics only in a secondary way.

It is obvious, that this morphogrammatic approach to polycontexturality has to be separated and compared to Gunther's polylogical introduction of contextures.

Nevertheless, a main algebraic rule can be extracted from the examples:

$$\textbf{operation}\left(\text{Tcontexture}\, n\right)\left(\text{Tcontexture}\, m\right) =$$

$$\text{Tcontexture}\left(\textbf{operation}\left(n, m\right)\right)$$

$$\text{for operation}\left(n, m\right) = z$$

$$\text{operation}\left(\text{Tcontexture}\, n\right) = \text{map Tcontexture}\left(\text{operation}\, n\right);$$

$$\textbf{map}\ \text{is used for operation}\left(n\right) = \left(\text{z1, z2, ..., zn}\right)$$

The term *Tcontexture* plays a double role:
in the formula *operation(Tcontexture n)(Tcontexture m)*, Tcontexture plays the role of an operand, while in the formula *Tcontexture(operation(n,m))*, Tcontexture takes the role of an operator.

This gets an operator reduction:
operator(Tcontexture(n)) = Tcontexture(operator(n))
operator(operator(operand n)) = operator(operator(operand n))

```
fun allTcontextureIota n  = map Tcontexture(iota n);
val allTcontextureIota = fn : int -> int list list list

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

fun allTIS n  = map TIS (Tcontexture n);
val it = fn : int -> int list list list
```

Stirling numbers of Stirling numbers

As a result for Stirling numbers, we may state: Stirling numbers are defining sequences of numbers. *Second-order* sequences are defining sequences of morphograms that are defined by Stirling numbers. Hence second-order sequences are sequences of sequences of kenograms, and not of numbers anymore.

A first attempt to implement Stirling numbers on Stirling morphograms is given by the following procedures:

- fun **allTcontextureStirling** n k = allkconcat(Tcontexture(rrr n k))(Tcontexture(sss n k));
val allTcontextureStirling = fn : int -> int -> int list list list

With the help of the functions:

- fun rrr n k = S(n-1,k-1);
val rrr = fn : int -> int -> int
- fun sss n k = k*S(n-1,k);
val sss = fn : int -> int -> int

The purely combinatorial result without an inscription of the order of the results is given by the procedure:

fun **TcontextureStirlingNum** n k = Tcontexture(S(n,k))

The correspondance
allkconcat(flat(Tcontexture(rrr n k))(Tcontexture(sss n k))) = Tcontexture(S(n,k))
is a further example for the operational rule:

operation(Tcontexture n)(Tcontexture m) = Tcontexture (operation(m,n)).

Again, the rule holds for the additive *kconcat.* It doesn't hold directly for multiplicative operations.

Limits of the approach

- allkmul(Tcontexture 2)(Tcontexture 2);
[[[1,1,1,1]],[[1,1,2,2]],[[1,2,1,2]],
[[1,2,2,1],[1,2,3,1],[1,2,2,3],[1,2,3,4]]]
- Tcontexture (2*2);
[[1,1,1,1],[1,1,2,2],[1,2,1,2],[1,2,2,1],**[1,1,1,2]**,**[1,1,2,1**],**[1,2,1,1]**,
[1,2,2,2],**[1,1,2,3**],**[1,2,1,3**],[1,2,3,1],[1,2,2,3],**[1,2,3,2]**,**[1,2,3,3]**,
[1,2,3,4]]

Different results between *allkmul(Tcontexture 2)(Tcontexture 2)* and *Tcontexture(2*2)*, because *kmul* is not covering the whole domain of *Tcontexture 4.*

In contrast *allkconcat(Tcontexture n)(Tcontexture m)* delivers the same set of morphograms as *Tcontexture(n+m)* but in a different order.

- allkconcat(Tcontexture 2)(Tcontexture 2);
[[[1,1,1,1],[1,1,2,2]],[[1,1,1,2],[1,1,2,1],[1,1,2,3]],
[[1,2,1,1],[1,2,2,2],[1,2,3,3]],
[[1,2,1,2],[1,2,2,1],[1,2,1,3],[1,2,3,1],[1,2,2,3],[1,2,3,2],[1,2,3,4]]]
- flat(allkconcat(Tcontexture 2)(Tcontexture 2)) = Tcontexture 4;

val it = false : bool

Further comparison

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

- **Tcontexture**(1+2);
val it = [[1,1,1],[1,1,2],[1,2,1],[1,2,2],[1,2,3]] : int list list

Are the results equal?

- length(Tcontexture(1+2)) = length(flat(allTcontextureKconcat 1 2));
val it = true : bool

As a result we observe that the numbers of produced morphograms are equal for both approaches. But they differ significantly in the structural information about the lists of morphograms that are reflecting the process of production.

Does this difference matter for the further studies of contextural number series? If yes, in what sense?

1.2. Recursion

1.2.1. Numerical analysis

```
scala> def factorial(number:Int) : Int = {
     |     if (number == 1)
     |        return 1
     |     number * factorial (number - 1)
     | }
factorial: (number: Int)Int
scala> println(factorial(5))    :120
```

"The new accumulator parameter stores the intermediate value, so we are no longer doing a calculation against the value returned from the function like we were before."

```
scala> def factorial(accumulator: Int, number: Int) : Int = {
     |   if (number == 1)
     |     return accumulator
     |   factorial (number * accumulator, number - 1)
     | }
factorial: (accumulator: Int, number: Int)Int
scala> println(factorial(1,5))     120
```

More for Java recursion at:
http://stackoverflow.com/questions/8183426/factorial-using-recursion-in-java

Recursion on the base of the retro-gradeness of morphograms

```
scala> def mg-factorial(morphogram:Morph List Int) : Morph = {
           |  if (morphogram == [ ])
           |     return [ ]
           |  kmul [morphogram (n)][morphogram (factorial(n-1))]
           | }
mg-factorial: (morphogram: Morph) Morph List
```

```
- kmul;
val it = fn : int list -> int list -> int list list
```

```
fun kmul [] b = [[]]
  |kmul a [] = [[]]
  |kmul a [1] = [a]
  |kmul [1] b = [b]
  |kmul a b =
```

Associativity and optimization

Because morphic languages, i.e. scripts, are neither *commutative* nor *associative*, several classical techniques to deal with complexity fail, and have to replaced by other methods.

Example for fact 3

fact(3) $\Longrightarrow$ 3 * fact(3-1)
$\Longrightarrow$ 3 *fact(2)
$\Longrightarrow$ 3 * (2 * fact(2-1))
$\Longrightarrow$ 3 * (2 * fact(1)
$\Longrightarrow$ 3 * (2 * (1 * fact(1-1)))
$\Longrightarrow$ 3 * (2 * (1 * fact(0)))
$\Longrightarrow$ 3 * (2* (1 * 1))
$\Longrightarrow$ 3 * (2 * 1)
$\Longrightarrow$ 3 * 2
$\Longrightarrow$ 6.

"Note how the multiplications get delayed until the recursion "bottoms out". We need to store these pending calculations somewhere (e.g. on the stack) and so the amount of storage required to evaluate fact n will be proportional to n.

We can make the function more efficient (but more obscure) by exploiting the ***associativity*** *of *.*

For example, we know that

3*(2*fact(1)) = (3*2)*fact(1) = 6*fact(1).

```
local fun ifact(0, p) = p
        | ifact(n, p) = ifact(n-1, n*p)
  in
     fun fact n = ifact(n, 1)
```

end;

Here, ifact(n,x) = n! × x.

"The evaluation of this version of fact uses constant space:

fact(3) ⟹ ifact(3,1)
⟹ ifact(3-1, 3*1))
⟹ ifact(2,3)
⟹ ifact(2-1, 2*3)
⟹ ifact(1,6)
⟹ ifact(1-1, 1*6)
⟹ ifact(0,6)
⟹ 6.

"This recursion has a special form: when a recursive call of the function is made, the caller returns the result of the recursive call directly, without further comupation; we say the call is tail-recursive, or iterative. In the original version of factorial the caller had to perform a multiplication using the result of the recursive call. Tail-recursive function calls can be implemented very efficiently."

http://homepages.inf.ed.ac.uk/mfourman/teaching/mlCourse/notes/L01.html

1.2.2. Contextural analysis

Following the 'naive' recursion scheme for fact 3, we get a corresponding multiplication "kmul" for 2*3.

kmul(Tcontexture 3 Tcontexture 2):

"2 * 3":

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

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

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

[1,2,3,2,3,4],[1,2,3,3,4,2],[1,2,3,4,3,2],[1,2,3,2,4,5],[1,2,3,4,5,2],
[1,2,3,3,4,5],[1,2,3,4,3,5],[1,2,3,4,5,6]] : 36

1.3. Tritogrammatics for factorials

1.3.1. Introducing morphogrammatic factorials

Example: Morphograms for mg-factorial 3!

mg(3) x mg(2) x mg(1):

```
fun allTcontextureFac 1 = []
| allTcontextureFac n = kmul((Tcontexture n) (Tcontexture (fac (n-1))));
 - Tcontexture 3;
val it = [[1,1,1],[1,1,2],[1,2,1],[1,2,2],[1,2,3]] : int list list
- Tcontexture 2;
val it = [[1,1],[1,2]] : int list list
- Tcontexture 1;
val it = [[1]] : int list list
- kmul[1,2][1];
val it = [[1,2]] : int list list
- kmul[1][1,2];
val it = [[1,2]] : int list list
```

kmul(Tcontexture 3 Tcontexture 2):

Does the equation **allTcontextureFac**(n) = **Tcontexture**(fac n) hold?

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

- length(flat(allTcontextureFac 3)); val it = 36 : int

- **Tcontexture**(fac 3);
Tcontexture (fac 3) = Tcontexture 6.
length it; val it = 203 : int, Tcard 6 =203

Procedure

```
fun allTcontextureFac n =
    allkmul (Tcontexture (n)) (Tcontexture (fac (n - 1)));
val allTcontextureFac = fn : int -> int list list list

fun TcontextureFacNum n = Tcontexture (fac n);
val TcontextureFacNum = fn : int -> int list list
```

Deuterogrammatics

```
- setof(map dnf(TcontextureFacNum 3));
 [[1,1,1,1,1,1],[1,1,1,2,2,2],[1,1,1,1,2,2],[1,1,2,2,2,2],[1,1,1,1,1,2],
  [1,2,2,2,2,2],[1,1,2,2,3,3],[1,1,1,2,2,3],[1,1,1,2,3,3],[1,1,2,2,2,3],
  [1,1,2,3,3,3],[1,2,2,2,3,3],[1,2,2,3,3,3],[1,1,1,1,2,3],[1,2,2,2,2,3],
  [1,2,3,3,3,3],[1,1,2,2,3,4],[1,1,2,3,3,4],[1,1,2,3,4,4],[1,2,2,3,3,4],
  [1,2,2,3,4,4],[1,2,3,3,4,4],[1,1,1,2,3,4],[1,2,2,2,3,4],[1,2,3,3,3,4],
  [1,2,3,4,4,4],[1,1,2,3,4,5],[1,2,2,3,4,5],[1,2,3,3,4,5],[1,2,3,4,4,5],
  [1,2,3,4,5,5],[1,2,3,4,5,6]] : int list list

setof(map dnf(flat(allTcontextureFac 3)));
[[1,1,1,1,1,1],
[1,1,1,1,2,2],[1,1,2,2,2,2], [1,1,1,2,2,2]
[1,1,1,2,3,3],[1,1,2,2,2,3],[1,1,1,2,2,3],[1,2,2,2,3,3],[1,1,2,2,3,3],
[1,2,2,3,4,4],[1,1,2,2,3,4],[1,1,2,3,3,4],[1,2,2,3,3,4],
[1,1,2,3,4,5],[1,2,2,3,4,5],[1,2,3,3,4,5],
[1,2,3,4,5,6]].
```

Protogrammatics

A further genuin reduction to its proto-structure is achieved with the following procedure:

```
- setof(map pnf(flat(allTcontextureFac 3)));
[[1,1,1,1,1,1],
[1,1,1,1,1,2],
[1,1,1,1,2,3],
[1,1,1,2,3,4],
[1,1,2,3,4,5],
[1,2,3,4,5,6]].
```

1.4. Fibonacci numbers in morphogrammatics

1.4.1. Fibonacci sequence for natural numbers.

"Let n be a positive integer. A composition of n is a way of writing n as an ordered sum of one or more positive integers (called

parts). For example, the compositions of 3 are 1 + 1 + 1, 1 + 2, 2 + 1, and 3. Let f (n) := the number of compositions of n in which all parts belong to the set {1, 2}." (Wagner)

Example for tritograms

Fibonacci sequences for trito-, deutero- and protograms. In this case, Fibonacci recursion is applied on morphic patterns of numbers, and not on isolated numbers as such.

Such patterns may be seen as complexions of sequences to model complex systems with complex distributed Fibonacci sequences, as an example.

The exercises of this paper are focused on the conceptual aspects of the constructions and not on any kind of optimizing the implemented procedures.

$$F_n = F_{n-1} + F_{n-2}$$

```
fun fib 1 = 1
| fib 2 = 1
| fib n = fib(n-1) + fib(n-2);
(* fib n is the nth Fibonacci number *)
```

1.4.2. Fibonacci sequences for tritograms

$$mgFib_{Tcontexture}\,n = mgFib_{Tcontexture}\,n-1 + mgFib_{Tcontexture}\,n-2$$

Numerical formulation

```
fun fib (n:int):int =
(case (n)
of 0 => 0
| 1 => 1
| n => fib(n-1) + fib(n-2));
```

Contextural formulation

The contextural formulation is modeling the clacssical version as far as possible. Nevertheless, there are, again two versions possible. One is contextural the other is numerical, i.e. *allTcontextureFib* and *TcontextureFibNum*(fib n).

```
fun allTcontextureFib n =
(case (n)
of 2 => []
| n  =>  allTcontextureFib (n));
val allTcontextureFib = fn : int -> 'a list
```

```
fun allTcontextureFib n  =
allkconcat (Tcontexture (fib(n-1))) (Tcontexture (fib(n-2)));
val it = fn : int -> int list list list
```

fun **lengthFib** n = length(flat(allTcontextureFib n));

Numeric formulation

fun **allTcontextureFibNum** n = Tcontexture(fib n);
val allTcontextureFibNum = fn : int -> int list list

```
Polycontextural Fibonacci allTcontextureFib

fun allTcontextureFib n =
    (case (n)
    of 2 = > []
    | n = >
     allkconcat (Tcontexture (fib (n - 1))) (Tcontexture (fib (n - 2))));

   val allTcontextureFib = fn : int -> int list list list

   fun lengthFib n = length (flat (allTcontextureFib n));
```

```
Polycontextural Fibonacci allTcontextureFib

fun TcontextureFibNum n =
    (case (n)
    of 2 = 1
     | n = > Tcontexture (fib (n));

   val TcontextureFib = fn : int -> int list list
```

1.5. Partitions

1.5.1. Partitioning contextures

Numerical partitions

```
fun P (n,1) = 1
  |P (n,k) =
    if k>n then 0
    else if k=n then 1
    else P(n-1,k-1) + P(n-k,k);
```

Combinatorics of partitions

http://www.artofproblemsolving.com/Resources/Papers/LaurendiPartitions.pdf

Contextural partitions

fun **TcontexturePartionNum** n k = Tcontexture(P(n,k))

- fun TcontexturePartitionNum n k = Tcontexture(P(n,k));
val TcontexturePartitionNum = fn : int -> int -> int list list

- TcontexturePartitionNum 5 2;
val it = [[1,1],[1,2]] : int list list
- TcontexturePartitionNum 10 3;
- length it;
val it = 4140 : int

allTcontexturePartition

fun **allTcontexturePartition** n k = allkconcat (Tcontexture(P(n-1,k-1))) (Tcontexture(P(n-k,k)))
- fun allTcontexturePartition n k = allkconcat (Tcontexture(P(n-1,k-1))) (Tcontexture(P(n-k,k)));
val allTcontexturePartition = fn : int -> int -> int list list list

- allTcontexturePartition 5 2;
val it = [[[1,1],[1,2]]] : int list list list

- allTcontexturePartition 10 3;
- length it;
val it = 225 : int

Some special functions for a 'combinatorics' of contextures:

Numeric

- **allpartitions** 10 3;

- map tnf(allpartitions 10 3);
val it = [[1,2,2],[1,1,2],[1,2,3],[1,2,3],[1,2,2],[1,2,3],[1,2,3],[1,2,2]]

Contextural

fun **TcontextureAllpartitions** n k = ???

Tuples of Tcontextures: [1,2,2] $\Longrightarrow$ allkconcat((Tcontexture 1), (Tcontexture 2), (Tcontexture 2))

Example

Properties: non-commutativity ans super-additivity

partition 5: [1,1,3] : allkconcat((Tcontexture 1), (Tcontexture 1), (Tcontexture 3)):

- allkconcat(Tcontexture 1)(Tcontexture 1);
val it = [[[1,1],[1,2]]] : int list list list

(a.) - allkconcat [[1,1,1],[1,1,2],[1,2,1],[1,2,2],[1,2,3]] [[1,1],[1,2]];
val it =
[[[1,1,1,1,1],[1,1,1,2,2]],
[[1,1,1,1,2],[1,1,1,2,1],[1,1,1,2,3]],
[[1,1,2,1,1],[1,1,2,2,2],[1,1,2,3,3]],

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

[1,2,3,3,2],[1,2,3,2,4],[1,2,3,4,2],[1,2,3,3,4],[1,2,3,4,3],[1,2,3,4,5]]]
- length it; val it = 10 : int
```

(b.) - allkconcat[[1,1],[1,2]][[1,1,1],[1,1,2],[1,2,1],[1,2,2],[1,2,3]];

```
val it =
  [[[1,1,1,1,1],[1,1,2,2,2]],
   [[1,1,1,1,2],[1,1,2,2,1],[1,1,2,2,3]],
   [[1,1,1,2,1],[1,1,2,1,2],[1,1,2,3,2]],
   [[1,1,1,2,2],[1,1,2,1,1],[1,1,2,3,3]],
   [[1,1,1,2,3],[1,1,2,1,3],[1,1,2,3,1],[1,1,2,3,4]],
   [[1,2,1,1,1],[1,2,2,2,2],[1,2,3,3,3]],
   [[1,2,1,1,2],[1,2,2,2,1],[1,2,1,1,3],[1,2,3,3,1],[1,2,2,2,3],[1,2,3,3,2],
[1,2,3,3,4]],

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

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

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

Comparison (a.) and (b.)

```
- flat(allkconcat[[1,1],[1,2]][[1,1,1],[1,1,2],[1,2,1],[1,2,2],[1,2,3]]) =
  flat(allkconcat[[1,1,1],[1,1,2],[1,2,1],[1,2,2],[1,2,3]] [[1,1],[1,2]]);
val it = false : bool
```

Numeric

- **allsums** 5 3;

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

Contextural

fun **TcontextureAllsums** n k = ??

1.5.2. Properties of contextural partitions

Does the Ferrers diagrams still hold for contextural partitions?

> *"We now have a partition of 10 with 3 parts. Now we can see that if we start off with a diagram of a partition whose largest part is r and count by the columns instead of by the rows we will end up with a partition of n with exactly r parts."*

http://www.artofproblemsolving.com/Resources/Papers/LaurendiPartitions.pdf

The presupposition of the example certainly is the equality of both partitions, i.e. (3+3+2+1+1) = (5+3+2).
For a Ferrers diagram to work, a commutativity between the colomns and the rows of the diagrams must hold.

1. column = rows

```
  3 2 1
3 • • •
2 • •
1 •
```

2. **row ≠ columns**

```
3 • • •
2 • •
1 •
1 •
```

```
4 2 1
• • •
• •
•
•
```

Numeric

7 = 3 + 2 + 1 + 1 = 4 + 2 + 1

Contextural

Tcontexture 3 + Tcontexture 2 + Tcontexture 1 + Tcontexture 1 =?
- allkconcat(Tcontexture 3)(Tcontexture 2);

Tcontexture 4 + Tcontexture 2 + Tcontexture 1

Simple case

[2,1]:

	2	1
2	•	•
1	•	

2 + 1 = (ab)c
= (ac)b

1.5.3. Partition of contextures by complexity

As much as a number in an arithmetical system gets its partition into parts, polycontextural constellations of contextures gets their partitions into contextural parts defined by the complexity of the parts.

This was shown as an example by the **Reflectional analysis of allTcontextureFac** 3 in the paper *Memristive Recursivity.*
What we get with this approach of a reflectional analysis of morphic factorials, *allTcontextureFac* 3, is a 6 layered system of mediated contextures. In this case they are separted by their complexity, i.e. the number of different kenograms. And not by the *type* of distribution as it is proposed for the morphic approach.

Partition of the contextures of complexity 6 of factorials into 6 parts of length:1+7+9+12+6+1 = 36

- length(flat(allTcontextureFac 3)); val it = 36 : int

Trito – partition : allTcontextureFac 3

$$= \sum_{i=1}^{6} S_i : 1 + 7 + 9 + 12 + 6 + 1 = 36$$

S1: [1,1,1,1,1,1],
S2:
[1,1,1,2,2,2],[1,1,2,1,1,2],[1,2,1,1,2,1],[1,1,2,2,2,1],[1,2,1,2,1,2],[1,2,2,1,2,2]
,[1,2,2,2,1,1],
S3:
[1,1,2,3,3,1],[1,1,2,2,2,3],[1,2,1,3,1,3],[1,2,1,2,3,2],[1,2,2,3,1,1],[1,2,2,2,3,3]
,[1,2,3,2,3,1],
[1,2,3,3,1,2],[1,2,3,1,2,3],
S4:
[1,2,1,3,4,3],[1,1,2,3,3,4],[1,2,2,3,4,4],[1,2,3,2,1,4],[1,2,3,2,4,1],[1,2,3,4,1,2]
,[1,2,3,3,1,4],[1,2,3,3,4,1],
[1,2,3,4,3,1],[1,2,3,2,3,4],[1,2,3,3,4,2],[1,2,3,4,3,2],
S5:
[1,2,3,4,1,5],[1,2,3,4,5,1],[1,2,3,2,4,5],[1,2,3,4,5,2],[1,2,3,3,4,5],[1,2,3,4,3,5]
,
S6: [1,2,3,4,5,6] .

A **reduction** of allTcontextureFac 3 to the **deutero**-structure is given by the following procedure.
setof(map dnf(flat(allTcontextureFac 3)));

Deutero-partition: dnf(allTcontextureFac 3) = $\sum_{i=1}^{6}$ s $_i$: 1+3+5+4+3+1 = 17

S1: [[1,1,1,1,1,1],
S2: [1,1,1,1,2,2],[1,1,2,2,2,2], [1,1,1,2,2,2]
S3: [1,1,1,2,3,3],[1,1,2,2,2,3],[1,1,1,2,2,3],[1,2,2,2,3,3], [1,1,2,2,3,3],
S4: [1,2,2,3,4,4],[1,1,2,2,3,4],[1,1,2,3,3,4],[1,2,2,3,3,4],
S5: [1,1,2,3,4,5],[1,2,2,3,4,5],[1,2,3,3,4,5],
S6: [1,2,3,4,5,6]].

- **Dcontexture** 6;
[[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]]

Numeric analysis

allpartitions 6 = P(6,i) = $\sum_{i=1}^{6}$: 1+3+3+2+1+1 = 11
P(6,1) = [[6]]
P(6,2) = [[3,3],[4,2],[5,1]]
P(6,3) = [[2,2,2],[3,2,1],[4,1,1]]
P(6,4) = [[2,2,1,1],[3,1,1,1]]
P(6,5) = [[2,1,1,1,1]]
P(6,6) = [[1,1,1,1,1,1]] .

Morphogrammatic analysis

The morphic analysis of the example for morphic factorials separates the parts of the partition by its morphogrammatic distribution. Thus, this approach gives a direct morphic interpretation of the numerical partitions of combinatorics.

With that, the question might be answered: *How does a partition of complex contextures work?*

The numerical partitions are mapped onto the morphograms of a contexture, here **allTcontextureFac** 3.

This procedure is naturally applied to other special contextures and to contextures in general.

Hence, the numerical partition of the number 6 is defined by P(6, i), while the contextures are partioned from the contextures of **allTcontextureFac** 3.

The general case of Tcontexture 6 would force to analyse 203 different parts of the whole contexture of complexity 6. The factorial

example produces morhograms of length 6 but with just 36 morphograms. With that a further analysis of the Ferrers diagrams of contextures is prepared.

1.5.4. Morphogrammatic interpretation of combinatorial partitions

No. Partition morphograms (1,3,1)
S1=[3]: [1,1,1],
S2=[2,1]: [1,1,2],[1,2,1],[1,2,2],
S3=[1,1,1]: [1,2,3].

- **Tcontexture** 4; NF 4 = 15 : 1+4+3+6+1
No. Partition morphograms
S1=[4]: [1,1,1,1], : (1)
S2=[3,1]: [1,1,1,2],[1,1,2,1],[1,2,1,1],[1,2,2,2], : (4)
S3=[2,2]: [1,1,2,2],[1,2,1,2],[1,2,2,1], : (3)
S4=[2,1,1]: [1,1,2,3],[1,2,1,3],[1,2,3,1],[1,2,2,3],[1,2,3,2],[1,2,3,3], : (6)
S5=[1,1,1,1]: [1,2,3,4]. (1)

No. Partition morphograms (1,5,10,10,15,10,1) (NF 5 = 52)
- **(5):** (1)
  [1,1,1,1,1],
- **(4,1):** (4+1)
  [1,1,1,1,2], [1,1,1,2,1],[1,1,2,1,1],[1,2,1,1,1],[1,2,2,2,2],
- **(3,2):** (6+4)
  [1,1,1,2,2],[1,1,2,1,2],[1,1,2,2,1],[1,2,1,1,2],[1,2,1,2,1],[1,2,2,1,1],
  [1,2,2,2,1],[1,2,2,1,2],[1,2,1,2,2],[1,1,2,2,2],
- **(3,1,1):** (6+3+1)
  [1,1,1,2,3],[1,1,2,1,3],[1,1,2,3,1],[1,2,1,1,3],[1,2,1,3,1],[1,2,3,1,1],
  [1,2,2,2,3],[1,2,2,3,2],[1,2,3,2,2],[1,2,3,3,3],
- **(2,2,1):** (12+2+1)
  [1,1,2,2,3],[1,1,2,3,2],
[1,1,2,3,3],[1,2,1,2,3],[1,2,1,3,2],[1,2,2,1,3],[1,2,2,3,1],[1,2,3,1,2],
  [1,2,3,2,1],[1,2,1,3,3],[1,2,3,1,3],[1,2,3,3,1],
  [1,2,2,3,3],[1,2,3,2,3],[1,2,3,3,2],
- **(2,1,1,1):** (4+3+3)
  [1,1,2,3,4],[1,2,1,3,4],[1,2,3,1,4],[1,2,3,4,1],
  [1,2,2,3,4],[1,2,3,2,4],[1,2,3,4,2],
  [1,2,3,3,4],[1,2,3,4,3],[1,2,3,4,4],
- **(1,1,1,1,1):** (1)
  [1,2,3,4,5] .

The Stirling partition for the number 5 produces 5 different parts with length (1,15,25,10,1) containing 52 morphograms.

In contrast, the partition of the number 5 delivers 7 parts, i.e. (1,5,10,10,15,10,1).

The previous analysis shows a *mix* of integer partion and Stirling partition, with the result that the Stirling based morphograms are differentiated into the number of integer partitions and not just into the number of Stirling classes.

Hence, this mix of partitioning makes a difference of the morphograms belonging to the class (4,1) and the class (3,2).

Further example for a differential analysis of the Stirling numbers of the partition p(6)

On this level of complexity 6, a differentiation between similar sequences is realized: **A080575** is not A036040 or A178867. A036040: [1, 6, 15, 10, 15, 60, 15, **20**, 45, 15, 1] differs with 20 instead of 15 at position 8.

A080575 for Tcontexture 6
1, 6, 15, 15, 10, 60, 20, 15, 45, 15, 1

p(6): 11 = Dcontexture 6. Tcard 6 =203

IntegerPartition[6] =

{{6}, {5, 1}, {4, 2}, {4, 1, 1}, {3, 3}, {3, 2, 1}, {3, 1, 1, 1},
{2, 2, 2}, {2, 2, 1, 1}, {2, 1, 1, 1, 1}, {1, 1, 1, 1, 1, 1}}

No. Partition morphograms

- **6 : 1** [1,1,1,1,1,1]
- **(5,1)** : 6 = 5+1
[**1,1,1,1,1**,2],[1,1,1,1,2,1],[1,1,1,2,1,1],[1,1,2,1,1,1],
[1,2,1,1,1,1],[1,**2,2,2,2,2**]
- **(4,2)** : 15=10+5
[**1,1,1,1**,2,2],[1,1,1,2,1,2],[1,1,1,2,2,1],[1,1,2,1,1,2],[1,1,2,1,2,1],
[1,1,2,2,1,1],[1,2,1,1,1,2],[1,2,1,1,2,1],[1,2,1,2,1,1],[1,2,2,1,1,1],
[1,**2,2,2,2**,1],[1,2,2,2,1,2],[1,2,2,1,2,2],[1,2,1,2,2,2],[1,1,2,2,2,2],
- **(4,1,1)** : 15 = 10+4+1
[1,1,1,1,2,3],[1,1,1,2,1,3],[1,1,1,2,3,1],
[1,1,2,1,1,3],[1,1,2,1,3,1],[1,1,2,3,1,1],[1,2,1,1,1,3],[1,2,1,1,3,1],
[1,2,1,3,1,1],[1,2,3,1,1,1],
[1,2,2,2,2,3],[1,2,2,2,3,2],[1,2,2,3,2,2],[1,2,3,2,2,2],[1,2,3,3,3,3],
- **(3,3)** : 10
[1,1,1,2,2,2],[1,1,2,1,2,2],[1,1,2,2,1,2],[1,1,2,2,2,1],[1,2,1,1,2,2],
[1,2,1,2,1,2],[1,2,1,2,2,1],[1,2,2,1,1,2],[1,2,2,1,2,1],[1,2,2,2,1,1],
- **(3,2,1)** : 60 =30+15+15
[**1,1,1**,2,2,3],[1,1,1,2,3,2],[1,1,1,2,3,3], (30)
[1,1,2,1,2,3],[1,1,2,1,3,2],[1,1,2,2,1,3],[1,1,2,2,3,1],[1,1,2,3,1,2],
[1,1,2,3,2,1],[1,1,2,1,3,3],[1,1,2,3,1,3],[1,1,2,3,3,1],[1,2,1,1,2,3],

[1,2,1,1,3,2],[1,2,1,2,1,3],[1,2,1,2,3,1],[1,2,1,3,1,2],[1,2,1,3,2,1],
[1,2,2,1,1,3],[1,2,2,1,3,1],[1,2,2,3,1,1],[1,2,3,1,1,2],[1,2,3,1,2,1],
[1,2,3,2,1,1],[1,2,1,1,3,3],[1,2,1,3,1,3],[1,2,1,3,3,1],[1,2,3,1,1,3],
[1,2,3,1,3,1],[1,2,3,3,1,1],
[1**,2,2,2**,1,3],[1,2,2,2,3,1],[1,2,2,1,2,3], (15)
[1,2,2,1,3,2],[1,2,2,3,2,1],[1,2,2,3,1,2],[1,2,1,2,2,3],[1,2,1,2,3,2],
[1,2,1,3,2,2],[1,2,3,2,2,1],[1,2,3,2,1,2],[1,2,3,1,2,2],[1,1,2,2,2,3],
[1,1,2,2,3,2],[1,1,2,3,2,2],
[1,1,2,**3,3,3**],[1,2,3,3,3,1],[1,2,3,3,1,3], (15)
[1,2,3,1,3,3],[1,2,1,3,3,3],[1,2,2,2,3,3],[1,2,2,3,2,3],[1,2,2,3,3,2],
[1,2,3,2,2,3],[1,2,3,2,3,2],[1,2,3,3,2,2],[1,2,3,3,3,2],[1,2,3,3,2,3],
[1,2,3,2,3,3],[1,2,2,3,3,3],

• **(3,1,1,1)** : 20 =10+6+3+1
[**1,1,1**,2,3,4],[1,1,2,1,3,4],[1,1,2,3,1,4],[1,1,2,3,4,1],[1,2,1,1,3,4],
[1,2,1,3,1,4],[1,2,1,3,4,1],[1,2,3,1,1,4],[1,2,3,1,4,1],[1,2,3,4,1,1],
[1,**2,2,2**,3,4],[1,2,2,3,2,4],[1,2,2,3,4,2],[1,2,3,2,2,4],[1,2,3,2,4,2],
[1,2,3,4,2,2],
[1,2,**3,3,3**,4],[1,2,3,3,4,3],[1,2,3,4,3,3],
[1,2,3,**4,4,4**],

• **(2,2,2)** :15
[1,1,2,2,3,3],[1,1,2,3,2,3],[1,1,2,3,3,2],[1,2,1,2,3,3],[1,2,1,3,2,3],
[1,2,1,3,3,2],[1,2,2,1,3,3],[1,2,2,3,1,3],[1,2,2,3,3,1],[1,2,3,1,2,3],
[1,2,3,1,3,2],[1,2,3,2,1,3],[1,2,3,2,3,1],[1,2,3,3,1,2],[1,2,3,3,2,1],

• **(2,2,1,1)** : 45=30+12+3
[**1,1**,2,2,3,4],[1,1,2,3,2,4],[1,1,2,3,4,2],
[1,1,2,3,3,4],[1,1,2,3,4,3],[1,1,2,3,4,4],[1,2,1,2,3,4],[1,2,1,3,2,4],
[1,2,1,3,4,2],[1,2,2,1,3,4],[1,2,2,3,1,4],[1,2,2,3,4,1],[1,2,3,1,2,4],
[1,2,3,1,4,2],[1,2,3,2,1,4],[1,2,3,2,4,1],[1,2,3,4,1,2],[1,2,3,4,2,1],
[1,2,1,3,3,4],[1,2,1,3,4,3],[1,2,1,3,4,4],[1,2,3,1,3,4],[1,2,3,1,4,3],
[1,2,3,3,1,4],[1,2,3,3,4,1],[1,2,3,4,1,3],[1,2,3,4,3,1],[1,2,3,1,4,4],
[1,2,3,4,1,4],[1,2,3,4,4,1],
[1,**2,2**,3,3,4],[1,2,2,3,4,3],[1,2,2,3,4,4],[1,2,3,2,3,4],[1,2,3,2,4,3],
[1,2,3,3,2,4],[1,2,3,3,4,2],[1,2,3,4,2,3],[1,2,3,4,3,2],[1,2,3,2,4,4],
[1,2,3,4,2,4],[1,2,3,4,4,2],
[1,2,**3,3**,4,4],[1,2,3,4,3,4],[1,2,3,4,4,3],

• **(2,1,1,1,1)** : 15=5+4+3+2+1
[**1,1**,2,3,4,5],[1,2,1,3,4,5],[1,2,3,1,4,5],[1,2,3,4,1,5],[1,2,3,4,5,1],
[1,**2,2**,3,4,5],[1,2,3,2,4,5],[1,2,3,4,2,5],[1,2,3,4,5,2],
[1,2,**3,3**,4,5],[1,2,3,4,3,5],[1,2,3,4,5,3],
[1,2,3,**4,4**,5],[1,2,3,4,5,4],[1,2,3,4,**5,5**],

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

"Refined" Stirling numbers of the second kind

Without doubt there is somewhere a mathematical formula for the described functions of Stirling numbers of partitions.

Partitions: p(6) = 11
Tcard 6 = 203

Stirling numbers: 203 = (1,31,90,65,15,1)
There are 11 groups for integer partition and just 6 groups for the Stirling partition of the number 6.

Hence, the integer partition of the Stirling partition delivers an additional *differentiation* of 5 groups. The Stirling approach unifies the groups (5,1), (4,2) and (3,3) of the morphic partition, as well as the classes (4,1,1), (3,2,1) and (2,2,2), and also the groups (3,1,1,1) and (2,2,1,1). This doesn't hold in this direct way for the general case.

Therefore, partitions of the same degree of partition are unified in the Stirling approach.

The number of groups of Stirling partitions is given by the value for n of the Stirling formula S(n,k).

My *intuitive and experimental* approach got some proper results as shown for the examples p(3) to p(6).

On the base of these *manually* elaborated sequences of 'partitions of Stirling partitions' it was finally possible to check it with the mathematical approaches and results as they are accessible by the OEIS organization.

It seems that this intuitively and experimentally achieved differentiation is similar to the concept of "*refined*" Stirling numbers of the second kind as collected at: http://oeis.org/**A080575.**

A relation between partition polynomials formed from these "refined" Stirling numbers of the second kind and umbral operator trees and Lagrange inversion is presented in the link **"Lagrange a la Lah".**

How to generate morphograms

As a next step of combinatorial analysis, the question, how to generate the individual sequences, i.e. morphograms out of the established partitions (integer partitions)?
The usual method is defined by the retrograde recursion of morphogrammatics. But this method is not giving combinatorial information about the number of partitions.

Bell polynomials

For example, we have

$\mathcal{B}_{6,2}(x_1,x_2,x_3,x_4,x_5) = 6x_5x_1 + 15x_4x_2 + 10x_3^2$

because there are

6 ways to partition a set of 6 as 5 + 1,

15 ways to partition a set of 6 as 4 + 2, and

10 ways to partition a set of 6 as 3 + 3.

http://en.wikipedia.org/wiki/Bell_polynomials

Following the entry for ***"Faà_di_Bruno's_formula"*** in WiKi, we get the interesting information.

"Faà di Bruno's formula may be stated in terms of Bell polynomials as follows:

$$\frac{d^n}{dx^n} f(g(x)) = \sum_{k=1}^{n} f^{(k)}(g(x)) B_{n,k}\left(g'(x), g''(x), \ldots, g^{(n-k+1)}(x)\right).$$

"The combinatorial form may initially seem forbidding, so let us examine a concrete case, and see what the pattern is:

$$\begin{aligned}(f\circ g)''''(x) &= f''''(g(x))g'(x)^4 + 6f'''(g(x))g''(x)g'(x)^2 \\ &+ 3f''(g(x))g''(x)^2 + 4f''(g(x))g'''(x)g'(x) \\ &+ f'(g(x))g''''(x).\end{aligned}$$

The pattern is

$$\begin{aligned} g'(x)^4 &\leftrightarrow 1+1+1+1 \leftrightarrow f''''(g(x)) \leftrightarrow 1 \\ g''(x)g'(x)^2 &\leftrightarrow 2+1+1 \leftrightarrow f'''(g(x)) \leftrightarrow 6 \\ g''(x)^2 &\leftrightarrow 2+2 \leftrightarrow f''(g(x)) \leftrightarrow 3 \\ g'''(x)g'(x) &\leftrightarrow 3+1 \leftrightarrow f''(g(x)) \leftrightarrow 4 \\ g''''(x) &\leftrightarrow 4 \leftrightarrow f'(g(x)) \leftrightarrow 1. \end{aligned}$$

"The factor g"(x)g'(x)^2 *corresponds to the partition 2 + 1 + 1 of the integer 4, in the obvious way.*
The factor f'''(g(x)) *that goes with it corresponds to the fact that there are three summands in that partition. The coefficient 6 that goes with those factors corresponds to the fact that there are exactly six partitions of a set of four members that break it into one part of size 2 and two parts of size 1."*
http://en.wikipedia.org/wiki/Faà_di_Bruno's_formula

The function T[n,m]: A080575

A080575: Triangle of multinomial coefficients
4: (1,4,3,6,1)
5: (1,5,10,10,15,10,1)
6: (1, 6, 15, 15, 10, 60, 20, 15, 45, 15, 1)
7: (1, 7, 21, 21, 35, 105, 35, 70, 105, 210, 35, 105, 105, 21, 1)

For n=4 the 5 integer partitions in canonical ordering with corresponding set partitions and counts are:

```
[4]      -> #{1234} = 1
[3,1]    -> #{123/4, 124/3, 134/2, 1/234} = 4
[2,2]    -> #{12/34, 13/24, 14/23} = 3
[2,1,1]  -> #{12/3/4, 13/2/4, 1/23/4, 14/2/3, 1/24/3, 1/2/34} = 6
[1,1,1,1] -> #{1/2/3/4} = 1
```

Thus row 4 is [1, 4, 3, 6, 1].

Triangle begins:
1;
1, 1;
1, 3, 1;
1, 4, 3, 6, 1;
1, 5, 10, 10, 15, 10, 1;
1, 6, 15, 15, 10, 60, 20, 15, 45, 15, 1;
1, 7, 21, 21, 35, 105, 35, 70, 105, 210, 35, 105, 105, 21, 1;

"Row 4 represents
1*k(4)+4*k(3)*k(1)+3*k(2)^2+6*k(2)*k(1)^2+1*k(1)^4 and T(4,4)=6 since there are six ways of partitioning four labeled items into one part with two items and two parts each with one item." (Tilman Neumann)
http://oeis.org/A080575

T[n,m] = count of set partitions of n with block lengths given by the m-th partition of n in the canonical ordering.
http://www.tilman-neumann.de/index.html

Alois P. Heinz, Rows n = 1..26, flattened
http://oeis.org/A080575/b080575.txt

http://en.wikipedia.org/wiki/Faà_di_Bruno's_formula
http://dida.sns.it/dida2/cl/10-11/folde0/pdf20

MATHEMATICA

```
<<DiscreteMath`Combinatorica`; runs[li:{__Integer}] := ((Length/@ Split[
# ]))&[Sort@ li]; Table[Apply[Multinomial, Partitions[w], {1}]/Apply[Times,
(runs/@ Partitions[w])!, {1}], {w, 6}]
```

Mathematica (w,10)
1.-5. {{1}, {1, 1}, {1, 3, 1}, {1, 4, 3, 6, 1}, {1, 5, 10, 10, 15, 10, 1},
7. {1, 6, 15, 15, 10, 60, 20, 15, 45, 15, 1},
8. {1, 7, 21, 21, 35, 105, 35, 70, 105, 210, 35, 105, 105, 21, 1},
9. {1, 8, 28, 28, 56, 168,56, 35, 280, 210, 420, 70, 280, 280, 840, 560, 56,

105, 420, 210, 28, 1},
10. {1, 9, 36, 36, 84, 252, 84, 126, 504, 378, 756, 126, 315, 1260, 1260, 1890, 1260, 126, 280, 2520,
840, 1260, 3780, 1260, 84, 945, 1260, 378, 36, 1}}

First and last position: 1
Second position: n+1: 3,4,5,...
Third: 3,10,15,21,28,36,45, ... : A112355, Triangular numbers that are the sum of three positive triangular numbers.

Comparison and corresondence

How are the morphograms corresponding to the 'fine' partitions defined?

fine-Stirling (Bell numbers) Stirling mor-phograms

[4] -> #{1234} = 1 :: {[1,1,1,1]}
[3,1] -> #{123/4, 124/3, 134/2, 1/234} = 4 :: {[1,1,1,2],[1,1,2,1],[1,2,1,1],[1,2,2,2]}
[2,2] -> #{12/34, 13/24, 14/23} = 3 :: {[1,1,2,2],[1,2,1,2],[1,2,2,1]}
[2,1,1] -> #{12/3/4, 13/2/4, 1/23/4, 14/2/3, 1/24/3, 1/2/34} = 6 :: {[1,1,2,3],[1,2,1,3],[1,2,3,1],[1,2,2,3],[1,2,3,2],[1,2,3,3]}
[1,1,1,1] -> #{1/2/3/4} = 1 :: {[1,2,3,4]}

Example (5,1)
• **(5,1)** : 6 = 5+1
[**1,1,1,1,1**,2],[1,1,1,1,2,1],[1,1,1,2,1,1],[1,1,2,1,1,1],[1,2,1,1,1,1],[1,**2,2,2,2,2**]
#{12345/6, 12346/5, 12356/4, 12456/3, 16345/2, 12345/6}

Both series are coinciding for the first 3 degrees, and for degree 4 and 5, there is a correspondence given by addition of components. This direct correspondence breaks down with degree higher 5.

fine-Stirling (Bell coefficients) Stirling Bell

1;
1, 1;
1, 3, 1;
1, **4, 3**, 6, 1; 1, 7, 6, 1 : 15
1, **5, 10**, **10, 15**, 10, 1; 1, 15, 25, 10, 1 : 52
1, 6, 15, 15, 10, 60, 20, 15, 45, 15, 1; 1, 31, 90, 65, 15, 1 : 203
1, 7, 21, 21, 35, 105, 35, 70, 105, 210, 35, 105, 105, 21, 1; 1, 63, 301, 350, 140, 21,1 : 877

Intrinsic fine analysis of the fine-analysis of Stirling numbers by partitions

A further step of the analysis of Stirling numbers, additionally to the 'refined' analysis (based on the Bell coefficients) is achieved with a kind of a fine-analysis of T[m,n], i.e. a fine-analysis of the fine-analysis, that takes the *different* representations of the partitions into account.

Example

S2=[3,1]: [1,1,1,2],[1,1,2,1],[1,2,1,1],[1,2,2,2], :
(4) = 3+1: ([1,1,1,2],[1,1,2,1],[1,2,1,1]) + ([1,2,2,2]).

The represenatant [1,2,2,2] is differentiated from the other representants with the value 1 as repetition. Hence, [1,2,2,2] is different from the representants ([1,1,1,2],[1,1,2,1],[1,2,1,1]). Therefore, this difference supports a further analysis, the *fine-analysis of the fine-analysis*.

Where is the formula for this case of differentiation?

Small table of second-order fine-analysis of partitions

3: 1, 2+1, 1
4: 1, 3+1, 3, 3+2+1, 1
5: 1, 4+1, 6+4, 6+3+1, 12+2+1, 4+3+3, 1
6: 1, 6=5+1, 15=10+5, 10, 15=10+4+1, 60=30+15+15,
20=10+6+3+1, 15, 45=30+12+3, 15=5+4+3+2+1, 1.

Example

4::6 -> 3+2+1:

[2,1,1] -> #{12/3/4, 13/2/4, 1/23/4, 14/2/3, 1/24/3, 1/2/34}
this corresponds to the morphograms:
[[**1,1**,2,3],[**1**,2,**1**,3],[**1**,2,3,**1**]; [1**,2,2**,3],[1,**2**,3,**2**];
[1,2,**3,3**]]

An analysis of the 'fine' analysis gives a hint how to define the further step of a 'fine-analysis of a fine-analysis'.

The partition [2,1,1] has 6 canonical results:
1. {12/3/4, 13/2/4, 1/23/4} : is producing a repetition of "1",
2. {14/2/3, 1/24/3} : is producing a repetition of "2",
3. {1/2/34} : is producing a repetition of "3".

Therefore, the second fine-analysis of "[2,1,1]" produces the partition: [3,2,1] out of the partition number 6.

Application

Morphograms of length 4 have a privileged status in the literature of morphogrammatics and morphogrammatically based polycontextural logic.

A first differentiation or classification was introduced as the distinction of "*junctional*" and "*transjunctional*" morphograms.

Hence, the 15 classical morphograms had been divided into the group of morphograms with 1 and two kenograms, and a second group with 3 or four morphograms.

Group of basic moprhograms as partitions: [4], [3,1], [2,2], [2,1,1],[1,1,1,1].

- **Tcontexture** 4;

type(1,7,6,1):

Junctional: [1,1,1,1],
[1,1,2,2],[1,2,1,2],[1,2,2,1],[1,1,1,2],
[1,1,2,1],[1,2,1,1],[1,2,2,2],
Transjunctional: [1,1,2,3],[1,2,1,3],[1,2,3,1],[1,2,2,3],[1,2,3,2],[1,2,3,3],
[1,2,3,4].

type(1,4,3,6,1):

total junctional: [1,1,1,1]
asymmetric junctional: [1,1,1,2],[1,1,2,1],[1,2,1,1],[1,2,2,2]
symmetric junctional: [1,1,2,2],[1,2,1,2],[1,2,2,1]
transjunctional: [1,1,2,3],[1,2,1,3],[1,2,3,1],[1,2,2,3],[1,2,3,2],[1,2,3,3]
total transjunctional: [1,2,3,4].

type(1,3+1,3,3+2+1,1):

total junctional: [1,1,1,1]
asymmetric junctional: [1,1,1,2],[1,1,2,1],[1,2,1,1],
total asymmetric: [1,2,2,2]
symmetric junctional: [1,1,2,2],[1,2,1,2],[1,2,2,1]
transjunctional: [1,1,2,3],[1,2,1,3],[1,2,3,1],
balanced [1,2,2,3],[1,2,3,2],
over-balanced [1,2,3,3]
total transjunctional: [1,2,3,4].

Reflector analysis

map kref
[[1,1,1,1],[1,1,2,2],[1,2,1,2],[1,2,2,1], [1,1,1,2],[1,1,2,1],[1,2,1,1],
[1,2,2,2],**[1,1,2,3]**,[1,2,1,3], [1,2,3,1],[1,2,2,3], [1,2,3,2],**[1,2,3,3],**
[1,2,3,4]] ;

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

Summary: Steps of analysis

1. Partitions,
2. Stirling numbers of the second kind, Bell coefficients,
3. ”refined” multinomial Stirling partition
4. fine-analysis of “refined” Stirling partitions.

1.6. Polycontextural Stirling and Mersenne numbers

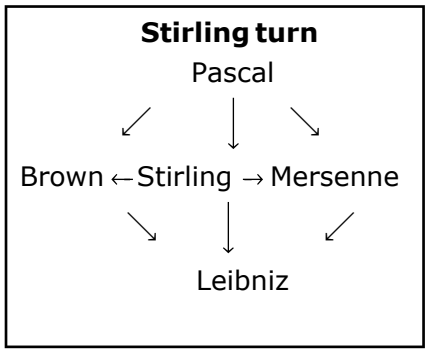
1.6.1. Polycontextural Stirling

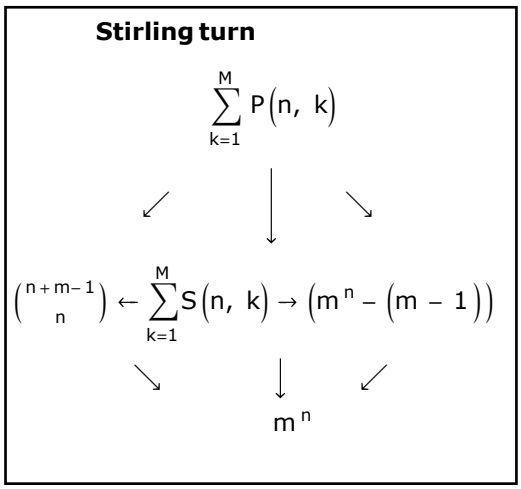
All writing systems of graphematics are accessible to a polycontexturalization of their fundamental frameworks that moves them from a first-order to a second order level of conceptualization and implementation.

Are there broken sequences of contextures or even broken contextures too?

This combinatorial study is neglecting partly the fact that contextures are not isolated objects but mediated into a web of polycontexturality.

This applies to the well known scheme of the **Stirling turn**:





Stirling

Stirling number sequences of Stirling number sequences.

Stirling S2n

fun **S** (n,1) = 1
|S (n,k) =

```
if k>n then 0
else if k=n then 1
else S(n-1,k-1) + k*S(n-1,k)
```

- fun **allTcontextureStirling** n k = allkconcat(Tcontexture(rrr n k))(Tcontexture(sss n k));
val allTcontextureStirling = fn : int -> int -> int list list list

With the help of the functions:
- fun rrr n k = S(n-1,k-1);
val rrr = fn : int -> int -> int
- fun sss n k = k*S(n-1,k);
val sss = fn : int -> int -> int

The purely combinatorial result without an inscription of the order of the results is given by the procedure:

fun **TcontextureStirlingNum** n k = Tcontexture(S(n,k))

- fun TcontextureStirlingNum n k = Tcontexture(S(n,k));
val TcontextureStirlingNum = fn : int -> int -> int list list

1.6.2. Polycontextural Mersenne universe

Mersenne universes are playing an important role in the general theory of graphematics. Here, contextures are set into a Mersenne sequence.

Mersenne universe of contextures: 2^n-1.

- fun **TcontextureMersenneNum** n = Tcontexture (subtract'i 1 (powers 2 n))
val TcontextureMersenneNum = fn : int -> int list list

- TcontextureMersenneNum 3;

```
[[1,1,1,1,1,1,1],[1,1,1,1,2,2,2],[1,1,1,2,1,2,2],[1,1,1,2,2,1,2],
 [1,1,1,2,2,2,1],[1,1,2,1,1,2,2],[1,1,2,1,2,1,2],[1,1,2,1,2,2,1],
 [1,1,2,2,1,1,2],[1,1,2,2,1,2,1],[1,1,2,2,2,1,1],[1,2,1,1,1,2,2],
 [1,2,1,1,2,1,2],[1,2,1,1,2,2,1],[1,2,1,2,1,1,2],[1,2,1,2,1,2,1],
 [1,2,1,2,2,1,1],[1,2,2,1,1,1,2],[1,2,2,1,1,2,1],[1,2,2,1,2,1,1],
 etcetera

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

length(TcontextureMersenneNum 3) : val it = 877 : int
Tcard 7: val it = 877 : int

Generalized Mersenne universe of contextures: m^n- (m-1).

- fun **TcontextureMersenneGenNum** m n = Tcontexture(subtract'i(powers m n) (m-1));
val TcontextureMersenneGenNum = fn : int -> int -> int list list

- fun all**TcontextureMersenneGen** *m n* = **all**subtract'i
(Tcontexture(powers m n)) (Tcontexture(m-1));

1.6.3. Polycontextural Brownian universe

fun **TcontextureBrownNum** m n = Tcontexture $\binom{n+m-1}{n}$

fun TcontextureBrownNum m n = Tcontexture(choose (n+m-1) n)
val TcontextureBrownNum = fn : int -> int -> int list list

Examples

- TcontextureBrown 1 1;
val it = [[1]] : int list list

- TcontextureBrownNum 2 1;
val it = [[1,1],[1,2]] : int list list

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

Functions

fun **choose** n k=
  (fak n) div ((fak k)* fak (n-k));

Catalan numbers

fun **catalan** 0 = 1
| catalan n = ((4 * n - 2) * catalan(n - 1)) div (n + 1);
val it = fn : int -> int

fun print_catalans(n) =
   if n > 15 then ()
   else (print (Int.toString(catalan n) ^ "\n"); print_catalans(n + 1));
print_catalans(0);

http://oeis.org/**A000108**
1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012

Catalan numbers: C(n) = binomial(2n,n)/(n+1) = (2n)!/(n!(n+1)!).

Catalan contextures

- fun **allTcontextureCatalanNum** n = Tcontexture(catalan n);

Graphics of Catalan Numbers
http://www.robertdickau.com/catalan.html

1.6.4. Polycontextural Leibniz universe

Leibniz universes are the basic universes of any formal languages. A polycntextural version of the semiotic or lingustic Leibniz universe is introdued by the polycontextural Leibniz universe *TcontexturLeibniz*.

fun TcontextureLeibnizNum m n = Tcontexture(powers m n)

```
- fun TcontextureLeibnizNum m n = Tcontexture(powers m n);
val TcontextureLeibnizNum = fn : int -> int -> int list list
- TcontextureLeibnizNum 2 2;
 [[1,1,1,1],[1,1,2,2],[1,2,1,2],[1,2,2,1],[1,1,1,2],[1,1,2,1],[1,2,1,1],
  [1,2,2,2],[1,1,2,3],[1,2,1,3],[1,2,3,1],[1,2,2,3],[1,2,3,2],[1,2,3,3],
  [1,2,3,4]]
```

fun allTcontextureLeibniz m n = allkmul (Tcontexture m) (Tcontexture(power(m,n-1)))

```
- fun allTcontextureLeibniz m n = allkmul (Tcontexture m)
(Tcontexture(power(m,n-1)));
val allTcontextureLeibniz = fn : int -> int -> int list list list
- allTcontextureLeibniz 2 2;

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

- map **ENstructureEN** (flat(allTcontextureLeibniz 2 2));

```
 [[[],[E],[E,E],[E,E,E]],
 [[],[E],[N,N],[N,N,E]],
 [[],[N],[E,N],[N,E,N]],
 [[],[N],[N,E],[E,N,N]],
 [[],[N],[N,N],[E,N,N]],
 [[],[N],[N,E],[N,N,N]],
 [[],[N],[N,N],[N,N,N]]] : EN list list list
```

2. Sequences in sequences

2.1. Reiterating the procedures

2.1.1. Conceptual context

The question is: *How are natural number series behaving in systems of trans-Classical number series?*

As a first step to study the behavior of number sequences in trans-

Classical systems it seems to be reasonable to apply the first-order operations onto the developed 'second-order' procedures.

Hence, the application of the basic kenogrammatic operations, like succession, *Tsucc*, addition, *kconcat* and multiplication, *kmul a*re iterated on the just defined operations.

The most basic function for all number theretic operations, obviously, is the successor function. In this context, the function *Tsucc*.

Again, the main feature of the successor function is its retrograde recursivity. This feature repeats and realizes itself in all further constructions.

2.1.2. Basic applications

- map **Tsucc**(Tcontexture 3);

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

- map Tsucc([[1,1,2],[1,2,1],[1,2,2],[1,2,3],[1,1,1,1]]);
val it = [[1,2,1],[1,2,2],[1,2,3],[1,1,1,1],[1,1,1,2]] : int list list

fun **TcontextureFibNumSucc** n = map Tsucc (Tcontexture (fib(n)));
val TcontextureFibNumSucc = fn : int -> int list list

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

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

- Tcontexture (fib 4);
val it = [[1,1,1],[1,1,2],[1,2,1],[1,2,2],[1,2,3]] : int list list

- map **Tsucc**(Tcontexture (fib 5));

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

- **allkconcat**(Tcontexture 3) (Tcontexture 2);

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

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

- **allkconcat**(TcontextureFibNum 3) (TcontextureFibNum 3);

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

- **allkconcat**(TcontextureFibNumSucc 3) (TcontextureFibNumSucc 3);

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

- **allkmul**(TcontextureFibNum 3) (TcontextureFibNum 3);

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

```
fun kconcatStirling m n k l = allkconcat (TcontextureStirlingNum m n)
(TcontextureStirlingNum k l);
val kconcatStirling = fn : int -> int -> int -> int -> int list list list
```

2.2. Discontextural number sequences

2.2.1. Evolutive movements

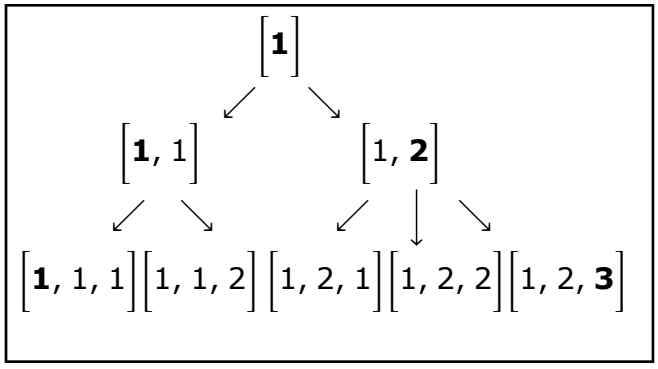
Morphograms, that are defining the basic formal structure of contextures, are patterns of kenograms.

It is an interesting question to ask how kenograms behave in sequences of evolving morphograms.

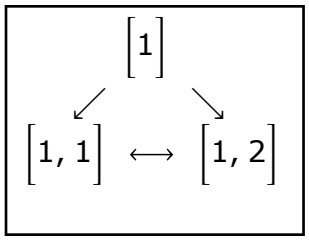
Patterns of kenograms might be used for strictly separated parallel developments of kenogrammatic sequences.

The first observation was studied by Gunther as the parallelism of sequences that behave as cardinal and ordinal sequences. And additionally, there are mediating sequences to observe. The sequences had been studied under the action of the successor operation only.

A new attempt is achieved if succession is additionally connected with ‘second-order’ prolongations as introduced before.



2.2.2. Emanative movements



2.3. Recursive arithmetics of contextures

2.3.1. Recursive arithmetics of numbers

S(x) ≠ x
S(x) = S(y) ⟶ x = y

x + 0 = x
x + S(y) = S(x + y)
x * 0 = x
x * S(y) = xy + x

With commutativity and assocuativity for addition "+".

Recursion theorem for number systems

Take two functions F: $\mathcal{N}$ ⟶ X and G: $\mathcal{N}$ ⟶ X such that:

F(0) = a
G(0) = a
F(n+1) = f(F(n))
G(n+1) = f(G(n))

By induction: F(n) = G(n) for all natural numbers n.

2.3.2. Recursive arithmetics for words

Similar scenario for words and lists. Except restrictions on commutativity.

2.3.3. Recursive arithmetics of contextures

TcontextureRecNum

-fun **TcontextureRec01Num** n = Tcontexture(succ n) = Tcontexture(n)

val TcontextureRec01Num = fn : int -> bool

fun **TcontextureRec02Num** n m = if ((Tcontexture(succ n)) = (Tcontexture(succ m))) then ((Tcontexture (n)) = (Tcontexture(m)));

- fun **TcontextureRec1Num** n = Tcontexture(n+0) = Tcontexture(n);
val **TcontextureRec1Num** = fn : int -> bool

- fun **TcontextureRec2Num** m n = Tcontexture(n+succ(m)) = Tcontexture(succ(m + n));
val TcontextureRec2Num = fn : int -> int -> bool

fun **TcontextureRec1** n = (allkconcat(Tcontexture 0)(Tcontexture(n))) = (Tcontexture(n));

fun **TcontextureRec1** n = (allkconcat(Tcontexture 2)(Tcontexture 0)) = (Tcontexture 2)

x + 0 = x, 0 + x= 0:

Non-commutativity

Kconcat

kconcat [x][] ≠ kconcat[][x]

- concat[][] = []
- kconcat[[]][] = []
- kconcat[][[]] = []

- kconcat[][1] = [[1]]
- kconcat[1][] = []

- kconcat[1,1][] = []
- kconcat[][1,1] = [[1,1]]

- Tcontexture 2;
val it = [[1,1],[1,2]] ,

- allkconcat(Tcontexture 0)(Tcontexture 2);
val it = [] : int list list list
- allkconcat(Tcontexture 2)(Tcontexture 0);
val it = []

- allkconcat[[1,1],[1,2]] [[]];
val it = [[],[]] : int list list list
- allkconcat [[]][[1,1],[1,2]];
val it = [[[1,1]],[[1,2]]]

Kmul

- kmul[][1] = [[]]
- kmul[1][] = [[]]
- kmul[][1,1] = [[]]
- kmul[1,1][] = [[]]

- kmul[1][1] = [[1]]

- kmul[1][1,1] [[1,1]]
- kmul[1,1][1] = [[1,1]]

- kmul[1][1,2] = [[1,2]]
- kmul[1,2][1] = [[1,2]]

- kmul[1,1][1,2] = [[1,1,2,2]]
- kmul[1,2][1,1] = [[1,2,1,2]] .

Hints towards a recursion theorem for kenogrammatics

The proof of *uniqueness* is disturbed by the distinction of *iterative* and *accretive* functions, i.e. by retrogradeness.

- TTS[];
val it = [[1]] : int list list

- TTS[1];
val it = [[1,1],[1,2]] : int list list

Analysis
TTS[1]: (tts_{iter}[1], tts_{accr}[1]) eq (tts_1[1], tts_2[1])

- map TTS[[1,1],[1,2]];
val it = [[**[1,1,1]**,[1,1,2]],[[1,2,1],[1,2,2],**[1,2,3]**]]

Analysis
TTS[1,1]: (tts_1[1,1], tts_2[1,1])

TTS[1,2]: (tts_1[1,2], tts_2[1,2], tts_3[1,2]).

- map TTS (flat [[[1,1,1],[1,1,2]],[[1,2,1],[1,2,2],[1,2,3]]]);
val it =
  [[[**1,1,1,1**],[1,1,1,2]],
  [[1,1,2,1],[1,1,2,2],[1,1,2,3]],
  [[1,2,1,1],[1,2,1,2],[1,2,1,3]],
  [[1,2,2,1],[1,2,2,2],[1,2,2,3]],
  [[1,2,3,1],[1,2,3,2],[1,2,3,3],[**1,2,3,4**]]] : int list list list

Pure iteration: [1,1,1] to [1,1,1,1]
Pure accretion: [1,2,3] to [1,2,3,4].

Pure iteration

F_{iter}(Tcontexture(0)) = Tcontexture(a)
G_{iter}(Tcontexture(0)) = Tcontexture(a)
F_{iter}(Tcontexture(n+1)) = f_{iter}(F_{iter}(Tcontexture(n)))
G_{iter}(Tcontexture(n+1)) = f_{iter}(G_{iter}(Tcontexture(n)))

Pure accretion

F_{accr}(Tcontexture(0)) = Tcontexture(a)
G_{accr}(Tcontexture(0)) = Tcontexture(a)

$F_{accr}(Tcontexture(n+1)) = f_{accr}(F_{accr}(Tcontexture(n)))$
$G_{accr}(Tcontexture(n+1)) = f_{accr}(G_{accr}(Tcontexture(n)))$

By induction

$F_{iter}(Tcontexture(n)) = G_{iter}(Tcontexture(n))$
$F_{accr}(Tcontexture(n)) = G_{accr}(Tcontexture(n))$.

Mixed cases

For $Tcontexture\ (n) > []$: $F_{iter}(Tcontexture\ (n)) \neq G_{accr}(Tcontexture\ (n))$
.

Analysis

Certainly, the mixed cases are the only one of real interest for a contextural analysis of induction.

As much as commutativity and associativity doesn't hold longer generally, the type of deviation, expressed by *negation* has to be specified. A general denial of such meta-theorems as usual is not of much use.

In a regular case, the complexity of the produced contextures differs by their iterative and accretive components but not by their combinatorial range or even 'length'.

The mixed cases are producing some zigzagging between iterative and accretive prolongations, it has therefore to be followed and compared. If the paths correspond then the induction holds. But there are also much more interesting cases where the paths are different but the journey end at the same station. Then the inductions are journey equivalent, albeit different. To realize such journeys properly, *emanative* differentiations shall be added to the iterative and accretive prolongations.

Hence two contextures might have the same evolutive level but differ in their emanative differentiation.

As an example, contexture [1,2,2,1] obviously differs from contexture [1,2,2,2] but their 'emanative length' is the same.

Hence, the negation involved in the statement, that they are *not* equal, is open for differentiation. The same holds for the cases of non-commutativity and non-associativity.
The emanative unary function TIS defines the difference between contexture [1,2,21] and contexture [1,2,2,2]:

```
TIS[1,2,21] = [1,2,2,2].
TIS(TIS(TIS(TIS[1,2,1,1]))) = [1,2,2,2];
val it = true : bool
```

Iterative prolongation

```
- fun iterTTS ts =
    map (fn i => ts@[i])
    (fromto 1 (AG ts));
val iterTTS = fn : int list -> int list list
- iterTTS[1,1];
val it = [[1,1,1]] : int list list
 - iterTTS[1,2];
val it = [[1,2,1],[1,2,2]] : int list list
- iterTTS[1,2,2];
val it = [[1,2,2,1],[1,2,2,2]] : int list list
- iterTTS[1,2,3];
val it = [[1,2,3,1],[1,2,3,2],[1,2,3,3]] : int list list
```

Mathematical induction

As a very first abbreviation we shall (re)write the axiom of mathematical induction in the framework of iteration and accretion

$$\mathrm{MI}: \forall\, \mathrm{P}\left[\left[\mathrm{P}\left(0\right) \wedge \forall\, \mathrm{k} \in \mathcal{N}\left(\mathrm{P}\left(\mathrm{k}\right) \Longrightarrow \mathrm{P}\left(\mathrm{k}+1\right)\right)\right)\right] \Longrightarrow \forall\, \mathrm{n} \in \mathcal{N}\left[\mathrm{P}\left(\mathrm{n}\right)\right]\right].$$

Polycontextural version

$$\forall\, \mathrm{P}\left[\left[\mathrm{P}\left(\mathrm{Tcontexture}\left(0\right)\right) \wedge \forall\, \mathrm{k} \in \mathcal{N}\left(\mathrm{P}\left(\mathrm{Tcontexture}\left(\mathrm{k}\right)\right) \Longrightarrow \mathrm{P}\left(\begin{array}{l}\mathrm{Tcontexture}\left(\mathrm{iter}\left(\mathrm{k}\right)\right)\\ \mathrm{Tcontexture}\left(\mathrm{accr}\left(\mathrm{k}\right)\right)\end{array}\right)\right)\right] \Longrightarrow \forall\, \mathrm{n} \in \mathcal{N}\left[\mathrm{P}\left(\begin{array}{l}\mathrm{Tcontexture}\left(\mathrm{iter}\left(\mathrm{n}\right)\right)\\ \mathrm{Tcontexture}\left(\mathrm{accr}\left(\mathrm{n}\right)\right)\end{array}\right)\right]\right]$$

3. Appendix: SML functions

Combinatorics

http://www.math.utk.edu/~wagner/papers/comb.pdf

Programming background:

System
SML/NJ: http://www.smlnj.org/,
Morphogrammatics
http://www.thinkartlab.com/pkl/SML-sources.NJ/ALL-MG-nov2012.sml
Book *Morphogrammatik*
http://www.thinkartlab.com/pkl/media/mg-book.pdf

List of ML functions (version 0.1)

SML additions (standard)

```
infix mem;
fun x mem[] = false
| x mem(y::l) = (x=y) orelse(x mem l);
val mem = fn : ''a * ''a list -> bool

fun newmem(x,xs) = if x mem xs then xs else x::xs;
val it = fn : ''a * ''a list -> ''a list

fun setof[]     = []
| setof(x::xs) = newmem(x,setof xs);
val it = fn : ''a list -> ''a list

- fun prod []      = 1
 | prod (n::ns) = n * (prod ns);
val prod = fn : int list -> int

fun conspair((x,y),(xs,ys))=(x::xs,y::ys);
fun split []          = ([],[])
| split(pair::pairs) =
conspair(pair, split pairs);
val conspair = fn : ('a * 'b) * ('a list * 'b list) -> 'a list * 'b list

fun split []          = ([],[])
|  split((x,y)::pairs) =
let val(xs,ys) = split pairs
in (x::xs, y::ys)
end;
val split = fn : ('a * 'b) list -> 'a list * 'b list

fun assoc ([], a)          = []
| assoc ((x,y)::pairs, a) =
if a=x then [y] else assoc(pairs, a);
val assoc = fn : (''a * 'b) list * ''a -> 'b list

fun nexts(a,[])          = []
| nexts(a, (x,y)::pairs) =
if a=x then y::nexts(a,pairs)
else    nexts(a,pairs);
```

```
val nexts = fn : ''a * (''a * 'b) list -> 'b list
```

```
fun iota 0 = []
| iota n = 1::(map increment (iota (n-1)))
and increment n = n+1;
```

Matrix

```
fun headcol[] = []
| headcol((x::_)::rows) = x :: headcol rows;
val headcol = fn : 'a list list -> 'a list
```

```
fun tailcols[] = []
| tailcols((_::xs)::rows) = xs::tailcols rows;
val tailcols = fn : 'a list list -> 'a list list
fun transp([]::rows) = []
| transp rows = headcol rows ::transp(tailcols rows);
val transp = fn : 'a list list -> 'a list list
```

```
fun cartprod([],   ys) = []
| cartprod(x::xs,ys) =
let val xsprod= cartprod(xs,ys)
fun pairx[]        = xsprod
| pairx(y::ytail) =
(x,y)::(pairx ytail)
in pairx ys
end;
val cartprod = fn : 'a list * 'b list -> ('a * 'b) list
```

```
fun dotprod([],[]) = 0.0
| dotprod(x::xs,y::ys)=x*y+dotprod(xs,ys);
val dotprod = fn : real list * real list -> real
```

```
fun rowprod(row,[])       = []
| rowprod(row,col::cols)=
dotprod(row,col)::rowprod(row,cols);
val rowprod = fn : real list * real list list -> real list
```

```
fun rowlistprod([],cols)       = []
| rowlistprod(row::rows,cols)=
rowprod(row,cols)::rowlistprod(rows,cols);
val rowlistprod = fn : real list list * real list list -> real list list
```

Sets

http://aleph0.clarku.edu/~djoyce/cs170/mlexample3.html

```
- fun union ([], ys)    = ys
 | union(x::xs,ys) = newmem(x,union(xs,ys));
val union = fn : ''a list * ''a list -> ''a list
```

```
- fun inter([],ys)    = []
 | inter(x::xs,ys) = if x mem ys
 then x::inter(xs,ys)
```

```
 else    inter(xs,ys);
val inter = fn : ''a list * ''a list -> ''a list

fun subset([],y) = true
 |   subset(a::x,y) =
       if member(a,y) then subset(x,y)
      else false;
val subset = fn : ''a list * ''a list -> bool

- fun equal(x,y) = subset(x,y) andalso subset(y,x);
val equal = fn : ''a list * ''a list -> bool

 fun difference([],y) = []
 |   difference(a::x,y) =
       if member(a,y) then difference(x,y)
       else a::difference(x,y);
val difference = fn : ''a list * ''a list -> ''a list

 fun intersection([],y) = []
 |   intersection(a::x,y) =
       if member(a,y) then a::intersection(x,y)
       else intersection(x,y);
val intersection = fn : ''a list * ''a list -> ''a list

- infix subs;
infix subs
- fun []      subs ys= true
 | (x::xs) subs ys= (x mem ys) andalso
 (xs subs ys);
val subs = fn : ''a list * ''a list -> bool

- infix seq;
infix seq
- fun xs seq ys= (xs subs ys) andalso
 (ys subs xs);
val seq = fn : ''a list * ''a list -> bool

- fun powset([],base)    = [base]
| powset(x::xs,base) =
 powset(xs,base) @ powset(xs,x::base);
val powset = fn : 'a list * 'a list -> 'a list list

fun power(x,n) = if (n=0) then 1 else power(x,n-1)*x;
val power = fn : int * int -> int
```

Kenogrammatics

```
fun allpairs xs ys=
    flat(map(fn x=> map (pair x) ys) xs);
val it = fn : 'a list -> 'b list -> ('a * 'b) list

fun allkmul xs ys=
    flat(map(fn x=> map (kmul x) ys) xs);
val it = fn : int list list -> int list list -> int list list list
```

```
fun allkconcat xs ys=
    flat(map(fn x=> map (kconcat x) ys) xs);
val it = fn : int list list -> int list list -> int list list list
```

```
fun allpowers xs ys=
    flat(map(fn x=> map (powers x) ys) xs);
val it = fn : int list -> int list -> int list
```

```
fun allsubtract'i xs ys=
    flat(map(fn x=> map (subtract'i x) ys) xs);
val allsubtract'i = fn : int list -> int list -> int list
```

ENstructureEN

```
fun deltaEN (i,j) z=
  if (pos i z) = (pos j z)
    then (E)
    else (N);
val it = fn : int * int -> ''a list -> EN
```

```
fun ENstructureEN z =
  map (fn trl => map (fn pair => deltaEN pair z)
                trl)
      (pairstructure (length z));
val it = fn : ''a list -> EN list list
```

```
fun allENstructureEN n =
 rd(map (fn ks => ENstructureEN ks)
   (Tcontexture n));
val it = fn : int -> EN list list list
```

```
fun alldnfENstructureEN n =
 rd(map (fn ks => ENstructureEN ks)
   (Dcontexture n));
val it = fn : int -> EN list list list
```

```
- fun allENstructureDNF n = map ENstructureEN (Dcontexture n);
val allENstructureDNF = fn : int -> EN list list list
```

```
fun allENstructure n =
 rd(map (fn ks => ENstructure ks)
   (Tcontexture n));
val it = fn : int -> (int * int * EN) list list list
```

```
fun allENcontextureEN m n = map ENstructureEN (flat(allkconcat
(Tcontexture n) (Tcontexture m)));
val it = fn : int -> int -> EN list list list
```

TNF, TTS, TIS

```
fun allTNF n =
 rd(map (fn ks => tnf ks)
   (Tcontexture n));
val it = fn : int -> int list list
```

```
fun TTS ts =
    map (fn i => ts@[i])
    (fromto 1 ((AG ts) + 1));
val it = fn : int list -> int list list

fun nTTS ts n =
    map (fn i => ts@[i])
    (fromto 1 ((AG ts) - n));
val it = fn : int list -> int -> int list list


fun allnTTS m n =
   rd(map (fn ks => nTTS ks m)
    (Tcontexture n));
val allnTTS = fn : int -> int -> int list list list  ??

fun allTTS n =
   rd(map (fn ks => TTS ks)
    (Tcontexture n));
val it = fn : int -> int list list list

fun iterTTS ts =
    map (fn i => ts@[i])
    (fromto 1 (AG ts));
val it = fn : int list -> int list list

fun allIterTTS n =
   rd(map (fn ks => iterTTS ks)
    (Tcontexture n));
val it = fn : int -> int list list list

fun allTIS n =
   rd(map (fn ks => TIS ks)
    (Tcontexture n));
val it = fn : int -> int list list

- map TIS [[1,1], [1,1,1]];
val it = [[1,2],[1,1,2]] : int list list
```

Palindromes

```
fun kref ks = tnf(rev ks);
fun palindrome l = (l = rev l);
fun ispalindrome l = (l = kref l);
fun dref ks = dnf(rev ks);
fun dnfispalindrome l = (l = dref l);

fun ENpalindromeEN n = map ENstructureEN(List.filter ispalindrome(Tcontexture n));
val it = fn : int -> EN list list list

fun ENpalindrome n = map ENstructure(List.filter ispalindrome(Tcontexture n));
val it = fn : int -> (int * int * EN) list list list
```

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

Combinatorial contextures

```
fun allTcontextureKconcat n m  = allkconcat(Tcontexture(n))(Tcontex-
ture(m));
val allTcontextureKconcat = fn : int -> int -> int list list list
```

```
fun allTcontextureKconcat n m  k = allkconcat((allkconcat(Tcontexture(n))(T-
contexture(m))) (Tcontexture(k)));
val allTcontextureKconcat = fn : int -> int -> int list list list
```

```
fun allTcontextureKconcatNum n m  = Tcontexture(n+m);
val allTcontextureKconcatNum = fn : int -> int -> int list list
```

```
fun allTcontextureIota n  = map Tcontexture(iota n);
val allTcontextureIota = fn : int -> int list list list
```

```
fun allTcontextureFac n =
      allkmul (Tcontexture (n)) (Tcontexture (fac (n – 1)));
val allTcontextureFac = fn : int –> int list list list
```

```
fun TcontextureFacNum n = Tcontexture (fac n);
val TcontextureFacNum = fn : int –> int list list
```

```
fun allTcontextureChooseNum n k = Tcontexture (choose n k);
val it = fn : int -> int -> int list list
fun allTcontextureChoose n k = allkmul(Tcontexture (choose n k));
val allTcontextureChoose = fn: int -> int -> int list list -> int list list list
```

```
fun allTcontextureFib n =
(case (n)
of 2 => []
| n  =>  allTcontextureFib (n));
val allTcontextureFib = fn : int -> 'a list
```

```
fun allTcontextureFib n  =
allkconcat (Tcontexture (fib(n-1))) (Tcontexture (fib(n-2)));
val it = fn : int -> int list list list
```

```
fun allTcontexturePartition n k = allkconcat (Tcontexture(P(n-1,k-1)))
(Tcontexture(P(n-k,k)))
val allTcontexturePartition = fn : int -> int -> int list list list
```

```
fun TcontextureStirlingNum n k = Tcontexture(S(n,k))
val TcontextureStirlingNum = fn : int -> int -> int list list
```

```
- fun allTcontextureStirling n k = allkconcat(Tcontexture(rrr n k))(Tcontex-
ture(sss n k));
val allTcontextureStirling = fn : int -> int -> int list list list
- fun rrr n k = S(n-1,k-1);
val rrr = fn : int -> int -> int
- fun sss n k = k*S(n-1,k);
```

```
val sss = fn : int -> int -> int

fun sumStirling n = sum 1 n (fn k => (S(n,k)));
val sumStirling = fn : int -> int  // (Bell numbers, Tcard)

- fun TcontextureMersenneNum n = Tcontexture (subtract'i 1 (powers 2 n))
val TcontextureMersenneNum = fn : int -> int list list

fun sumMersenne n = sum 1 n (fn k => subtract'i 1 (powers 2 n));
val sumMersenne = fn : int -> int

- fun TcontextureLeibnizNum m n = Tcontexture(powers m n);
val TcontextureLeibnizNum = fn : int -> int -> int list list

fun sumLeibniz m n = sum 1 n (fn k => powers m n);
val sumLeibniz = fn : int -> int -> int

- fun allTcontextureLeibniz m n = allkmul (Tcontexture m)
(Tcontexture(power(m,n-1)));
val allTcontextureLeibniz = fn : int -> int -> int list list list
```

Deutero- and proto-reductions

```
setof(map dnf(flat(allTcontexture"Operation" n)));
setof(map pnf(flat(allTcontexture"Operation" n)));

fun teq a b = (ENstructure a) = (ENstructure b);
val teq = fn : ''a list -> ''b list -> bool

- fun TcontextureTeq  a b = (map ENstructureEN(Tcontexture(a))) = (map
ENstructureEN(Tcontexture(b)));
val TcontextureTeq = fn : int -> int -> bool
```