

— vordenker-archive —

Rudolf Kaehr

(1942-2016)

Title

Lambda-Calculi in Polycontextural Situations

Archive-Number / Categories

2_19 / K08

Publication Date

2005

Keywords

Lambda Calculus, Polycontexturality, Algebras and Co-Algebras

Disciplines

Artificial Intelligence and Robotics, Continental Philosophy, Computer Sciences, Cybernetics,
Programming Languages, Systems Architecture and Theory and Algorithms,

Abstract

Sketch of the Lambda Calculus Essentials – General framework for Lambda Calculi in Contextures –
Lambda Calculi in a 3-contextural situation – Annoy the deadheads! – More to Bore: From Y to Why
– Don't halt the halting problem to halt – Towards General Architectonics for Lambda Calculi –
Some ends are just beginnings

Citation Information / How to cite

Rudolf Kaehr: "Lambda-Calculi in Polycontextural Situations", www.vordenker.de (Sommer Edition, 2017) J. Paul
(Ed.), URL: http://www.vordenker.de/rk/rk_Lambda-Calculi-in-Polycontextural-Situations_2005.pdf

Categories of the RK-Archive

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

DERRIDA'S MACHINES PART III

BYTES & PIECES

of

PolyLogics, m-Lambda Calculi,
ConTeXtures

Lambda Calculi in Polycontextural Situations



© by Rudolf Kaehr

ThinkArt Lab Glasgow Hallowe'en 2005

***"Interactivity is all there is to write about:
it is the paradox and
the horizon of realization."***

Lambda Calculi in Polycontextural Situations

A. Architectonics for Disseminated Formal Systems

- 1 General considerations 5**
 - 1.1 Signatures, place-holders and architectonics 6
 - 1.2 Knots, the Celtic Connection 7
 - 1.3 Tree patterns: linear, arboreal and stars 7
- 2 Advanced Architectonics for tabular PolyLogics 8**
 - 2.1 A simple star-pattern 8
 - 2.2 A simple star-line-pattern 9
 - 2.3 Distributed cyclic patterns 10
 - 2.4 Birkhoff Arithmetics of Chiasms 11

B. Lambda Calculi in Polycontextural Situations

- 1 Remembering the beginnings 14**
- 2 Sketch of the Lambda Calculus Essentials 15**
 - 2.1 Assembling the elements 15
 - 2.2 Reductions vs. abstraction rules 18
 - 2.3 Church-Rosser-Theorem 20
 - 2.4 Connecting the lambda calculus to the rest of the formal world 21
 - 2.5 Combinators 22
 - 2.6 Fixed-point theorem: the Y-operator 23
- 3 Main results 24**
- 4 General framework for Lambda Calculi in Contextures 25**
 - 4.1 Lambda Calculi disseminated 26
 - 4.2 ARS, ConTeXtures and Lambda Calculi 27
 - 4.3 *Free* and *bound* in contextures 28
 - 4.4 Algebras and co-algebras 28
 - 4.5 General Syntax for distributed lambda calculi 29
 - 4.6 Signatures for m-lambda calculi 35
 - 4.7 Lambda calculi as trees 36
- 5 Lambda Calculi in a 3-contextural situation 40**
 - 5.1 Syntactic rules 40
 - 5.2 Super-operators 43
 - 5.3 A family of substitutions 44
 - 5.4 General Constellations of Transformations 46
 - 5.5 Connecting poly-lambda calculi to a complex formal world 56
- 6 Annoy the deadheads! 57**
 - 6.1 A simple identical constellation 57
 - 6.2 Internal vs. external super-operators 58
 - 6.3 Internal vs. external super-operators 59
 - 6.4 A permutational constellation 61
 - 6.5 A reductional constellation 61
 - 6.6 An interactional constellation 61
 - 6.7 A reflectional constellation 62
- 7 More to Bore: From Y to Why 63**
 - 7.1 Remembering 63
 - 7.2 Distributed Y-Operators 64
 - 7.3 Why not some Why-operators? 65
 - 7.4 Graph diagrams for Y and WHY 66
 - 7.5 Iterability in Y- and WHY-operators 67
 - 7.6 Combinators in the general context of iterability 73

8	Don't halt the halting problem to halt	79
9	Reductional closure	80
9.1	Combining super-operators	80
10	Towards General Architectonics for Lambda Calculi	81
10.1	Architectonics with 4 contextures: One more stroke!	81
10.2	Architectonics with 5 and more strokes (contextures)	85
11	Some ends are just beginnings	87

C. Types and Contextures

1	Types in Lambda Calculus	90
2	Towards chiasmic types in $LC^{(m)}$	
2.1	Basic chiasms of types	91
3	Further developments: Type Derivations	96
3.1	Monocontextural type derivations	96
3.2	Meta-theoretic results	96
3.3	Complexity as polymorphism of types	97
3.4	Complexity as polycontexturality of calculi	98
3.5	Metamorphic type transformations	99
4	Types and Paradoxes	99

A. Architectonics for Disseminated Formal Systems

(PolyLogics, ConTeXtures, m-Lambda Calculi, m-Combinatory Logics)

1 General considerations

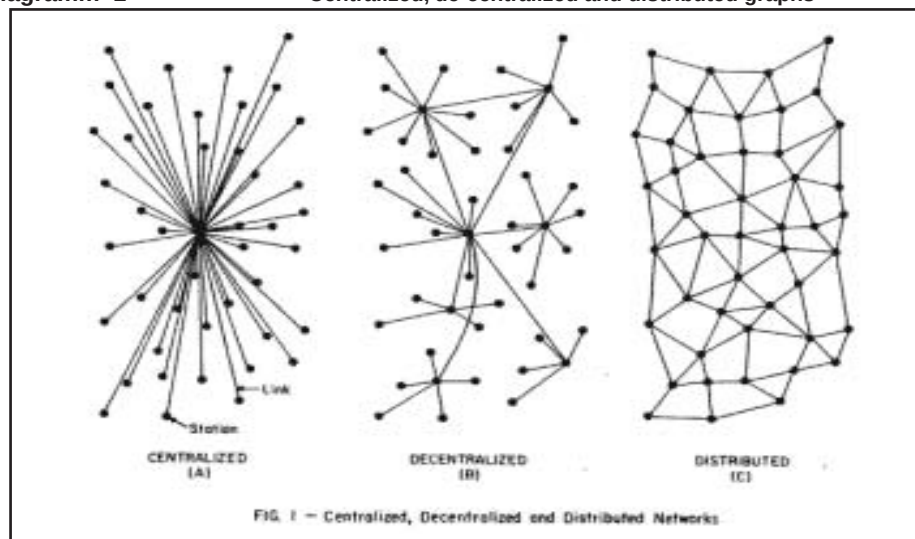
The idea of polycontextuality is based on compositions and decompositions of structures and their interpretation by formal systems, logics, arithmetics, semiotics. Each part of a complex compound can be logificated, that is, interpreted as a base for a logical system. Each sub-graph of a complex graph can be logificated. This turns the general sub-graph into a directed sub-graph. Sub-graphs are composed to a graph by means of the proemial relation. This is emphasizing the fact that the end of a connected sub-graph is the beginning of another sub-graph. What is considered as a component of a compound is a decision of modeling and has no ontological status. Thus, a compound can become a component and a component can be modeled as a compound following the rules of proemiality and chiasm in an iterative or recursive sense.

As an example of a chiasm between intra- and trans-contextual patterns the negational systems of linear architectonics produced by polycontextual negations can be thematized as regular decentralized architectonic patterns of new polycontextual systems delivering all sorts of shortest path and Hamiltonian cycles. Architectonic circularity should not be confused with self-referentiality of sentences.

To apply graph systems to polycontextuality they have to be interpreted as directed graph systems because contextures have locally, intra-contextual, a linear ordered structure. Especially logical systems are linear by their basic terms.

Additional to linear and arboreal graphs, the structure of graph systems can be classified as centralized, de-centralized and distributed.

Diagramm 1 Centralized, de-centralized and distributed graphs



http://thekla-guk.ch/hyperfiction05/index_hyperfic05.html

Non-linear mediated polylogics can be linked to m-categories and modal logics for further explanation and formalization. On the other hand it may turn out that such non-linear, tabular polylogics can serve as logical foundations of m-categories which shouldn't be based on predicate logic only.

Graph-theoretic studies on polycontextural, morphogrammatic and negational systems had been done in the 70th esp. by Gerhard Thomas. They also had been connected to Birkhoff numbers by Schadach, Thomas and Kaehr. Special interest was put on Hamilton cycles and other "labyrinths" following Gunther's project of developing a "negative language". But there was no working concept to interpret those structures logically.

1.1 Signatures, place-holders and architectonics

A term, object, entity belongs to a formal system—or it doesn't belong to it. Then it is not a term of the formal system. In a situation of a multitude of formal systems or calculi, a term which doesn't belong to a calculus, may belong to another one. To indicate this situation we can use signed terms and formulas. These signatures are not necessarily connected with logical meanings. They can represent all kinds of basic framework related oppositions in formal systems, like true/false, acceptance/rejection, opponent/proponent, antecedent/succedent, marked/unmarked, etc.

Signatures are a kind of place-holders, locators, not only for formulas but for formal systems as such. They are positioning formal systems in the polycontextural grid. Thus, disseminated formal systems are located, marked by their locators. The metaphor of inscription gets an explication by signatures as place-holders, or locators, of the inscribed formal systems.

A classical formal system is a system with a place-holder of value 1, which, obviously, can be omitted.

$$\begin{array}{lll}
 \forall^{(m)} H^{(m)} : H^{(m)} \in LC^{(m)} & \forall^{(3)} H^{(3)} : H^{(3)} \in LC^{(3)} & \begin{bmatrix} T_1, \emptyset, T_3 \\ F_1, T_2, \emptyset \\ \emptyset, F_2, F_3 \end{bmatrix} \\
 T_i H_i \text{ iff } H_i \in LC_i & H_1 \notin LC_1 \text{ iff } H_2 \in LC_2 & \\
 F_i H_i \text{ iff } H_i \notin LC_i, i \in s(m) & H_1 \in LC_1 \text{ iff } H_3 \in LC_3 & \\
 H_i \notin LC_i \text{ iff } H_{i+1} \in LC_{i+1} & H_2 \notin LC_2 \text{ iff } H_3 \notin LC_3 & T_1 H_1 \cong T_3 H_3 \\
 T_i H_i, T_i K_i \Rightarrow (H_i K_i) \in LC_i & & F_1 H_1 \approx T_2 H_2 \\
 H_i \in LC_i \Rightarrow H_i \equiv H_i & & F_2 H_2 \cong F_3 H_3
 \end{array}$$

1.2 Knots, the Celtic Connection

Knots as an architectonic pattern would introduce a new dimension into polycontextuality. As far as knots can be represented by their parts they are accessible to a polycontextual modeling. Proemiality is not excluding knot-structures. Therefore, polycontextual systems based on knot-architectonics could be constructed by the means of a specific application of the proemial relation.

<http://www.earlham.edu/~peters/knotlink.htm>

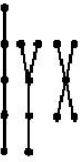
<http://www.maths.warwick.ac.uk/~bjs/MA3F2-page.html>

1.3 Tree patterns: linear, arboreal and stars

A tabular and matrix approach to polylogics opens up a quite natural semantic interpretation of non-linear mediated logical systems.

The numbers of tree structures are given by $b(m)$. Tree structures are abstracted from rooted trees.

Interestingly, for $m > 6$, tree structures are growing rapidly, so for $m=13$, we have 1301 different tree structures, producing 1301 different types of architectonics of the mediation of formal systems. These numbers are growing further with the introduction of rooted trees, where we have to consider some kind of origins of the tree.

Baumstrukturen :				Anzahlen :	
m	3	4	5	m	b(m)
					
b(m)	1	2	3		
				1	1
				2	1
				3	1
				4	2
				5	3
				6	6
				7	11
				8	23
				9	47
				10	106
				11	235
				12	551
				13	1301

2 Advanced Architectonics for tabular PolyLogics

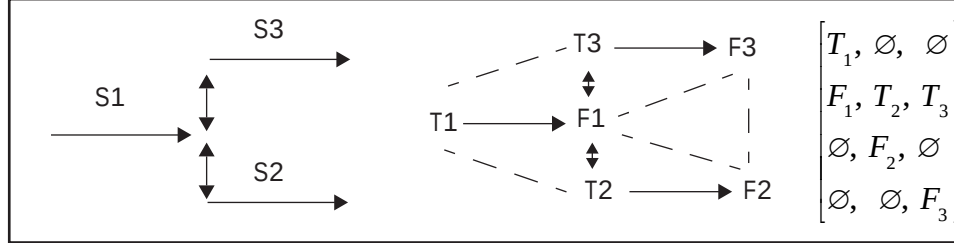
New notational challenges of complexities are naturally occurring for the case of non-linearly ordered mediation of logics. Additional to the known linear chiasms a new kind of over-determination enters the game. The examples shows simple star-pattern for distributed logical systems. The numeration is different from the linear case and the relations are reduced to the order and exchange relation excluding mediated systems.

2.1 A simple star-pattern

Graph-patterns can be valuated by polylogical truth-values delivering the polycontextural semantic matrix of the pattern.

$$\begin{aligned} \text{val}(\text{graph}) &= \text{sem}[\text{matrix}] \\ \text{val} : \{T_i, F_i, i \in N\} &\rightarrow [\text{semantic} - \text{matrix}] \rightarrow [\text{syntax} - \text{pattern}] \end{aligned}$$

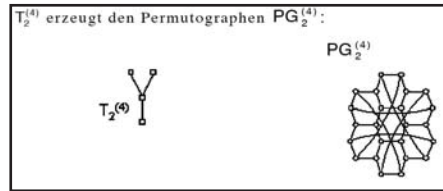
Diagramm 2 star-pattern without mediating sub-systems



$$N_1 : \begin{bmatrix} T_1, \emptyset, \emptyset \\ F_1, T_2, T_3 \\ \emptyset, F_2, \emptyset \\ \emptyset, \emptyset, F_3 \end{bmatrix} \Rightarrow \begin{bmatrix} F_1, T_2, T_3 \\ T_1, \emptyset, \emptyset \\ \emptyset, F_2, \emptyset \\ \emptyset, \emptyset, F_4 \end{bmatrix}$$

$$N_2 : \begin{bmatrix} T_1, \emptyset, \emptyset \\ F_1, T_2, T_3 \\ \emptyset, F_2, \emptyset \\ \emptyset, \emptyset, F_3 \end{bmatrix} \Rightarrow \begin{bmatrix} T_1, \emptyset, \emptyset \\ \emptyset, F_2, \emptyset \\ F_1, T_2, T_3 \\ \emptyset, \emptyset, F_3 \end{bmatrix}$$

$$N_3 : \begin{bmatrix} T_1, \emptyset, \emptyset \\ F_1, T_2, T_3 \\ \emptyset, F_2, \emptyset \\ \emptyset, \emptyset, F_3 \end{bmatrix} \Rightarrow \begin{bmatrix} T_1, \emptyset, \emptyset_2 \\ \emptyset, \emptyset, F_3 \\ \emptyset, F_2, \emptyset \\ F_1, T_2, T_3 \end{bmatrix}$$



Proemiality of (S1, S2, S3):

crossing of ((S1, S2), (S3))

with:

$\text{ord}(T_i, F_i), i=1,2,3$

$\text{cross}(\text{exch}(F_1, T_2), \text{exch}(F_1, T_3))$

$\text{cross}(\text{coinc}(T_1, T_2), \text{coinc}(T_1, T_3))$

$\text{cross}(\text{coinc}(F_1, F_2), \text{coinc}(F_1, F_3))$

The above semantic interpretation of the star-patterns allows to define straight forward logical operations based on it.

As an example the negations N1, N2 and N3 are introduced to explain the notational approach to star-patterns.

Negation system for $PG_2^{(4)}$:

$\text{Neg}(PG_2^{(4)}) = [N_1, N_2, N_3]$

Negation cycles:

$N_1(N_2(N_3)) = N_3(N_2(N_1))$

In contrast to linear negational systems star-systems have no direct commutative cycles, like $N_1(N_3) \neq N_3(N_1)$.

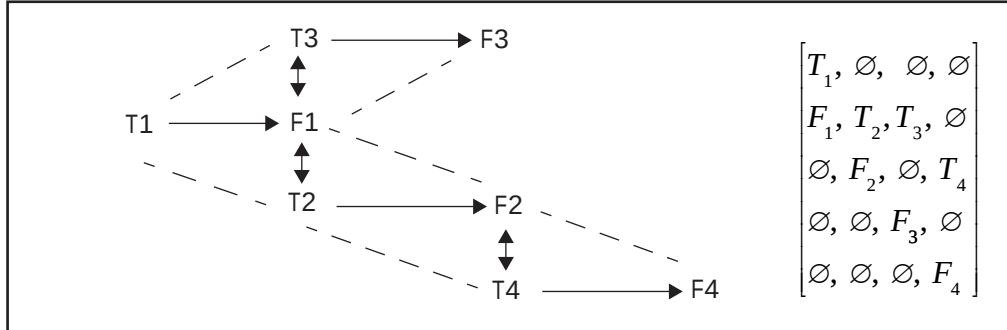
Gerhard Thomas: http://math.unipa.it/~circmat/indice_11_1985.htm

2.2 A simple star-line-pattern

The following example shows a crossing of two proemial relations.

Diagramm 3

Star-line pattern without mediating sub-systems



$$N_1 : \begin{bmatrix} T_1, \emptyset, \emptyset, \emptyset \\ F_1, T_2, T_3, \emptyset \\ \emptyset, F_2, \emptyset, T_4 \\ \emptyset, \emptyset, F_3, \emptyset \\ \emptyset, \emptyset, \emptyset, F_4 \end{bmatrix} \Rightarrow \begin{bmatrix} F_1, T_3, T_2, \emptyset \\ T_1, \emptyset, \emptyset, \emptyset \\ \emptyset, F_3, \emptyset, \emptyset \\ \emptyset, \emptyset, F_2, T_4 \\ \emptyset, \emptyset, \emptyset, F_4 \end{bmatrix}$$

$$N_2 : \begin{bmatrix} T_1, \emptyset, \emptyset, \emptyset \\ F_1, T_2, T_3, \emptyset \\ \emptyset, F_2, \emptyset, T_4 \\ \emptyset, \emptyset, F_3, \emptyset \\ \emptyset, \emptyset, \emptyset, F_4 \end{bmatrix} \Rightarrow \begin{bmatrix} T_1, \emptyset, \emptyset, \emptyset \\ \emptyset, F_2, \emptyset, T_4 \\ F_1, T_2, T_3, \emptyset \\ \emptyset, \emptyset, F_3, \emptyset \\ \emptyset, \emptyset, \emptyset, F_4 \end{bmatrix}$$

$$N_3 : \begin{bmatrix} T_1, \emptyset, \emptyset, \emptyset \\ F_1, T_2, T_3, \emptyset \\ \emptyset, F_2, \emptyset, T_4 \\ \emptyset, \emptyset, F_3, \emptyset \\ \emptyset, \emptyset, \emptyset, F_4 \end{bmatrix} \Rightarrow \begin{bmatrix} T_1, \emptyset, \emptyset, \emptyset \\ \emptyset, \emptyset, F_3, \emptyset \\ \emptyset, F_2, \emptyset, T_4 \\ F_1, T_2, T_3, \emptyset \\ \emptyset, \emptyset, \emptyset, F_4 \end{bmatrix}$$

$$N_4 : \begin{bmatrix} T_1, \emptyset, \emptyset, \emptyset \\ F_1, T_2, T_3, \emptyset \\ \emptyset, F_2, \emptyset, T_4 \\ \emptyset, \emptyset, F_3, \emptyset \\ \emptyset, \emptyset, \emptyset, F_4 \end{bmatrix} \Rightarrow \begin{bmatrix} T_1, \emptyset, \emptyset, \emptyset \\ F_1, T_2, T_3, \emptyset \\ \emptyset, \emptyset, \emptyset, F_4 \\ \emptyset, \emptyset, F_3, \emptyset \\ \emptyset, F_2, \emptyset, T_4 \end{bmatrix}$$

Crossing of two linear proemial relations:

`cross(PR(S1, S3), PR(S1, S2, S4))`

with:

`ord(Ti, Fi), i=1,2,3,4`

`cross(exch(T3, F1), excl(F1, T2))`

`cross(coinc(T1, T2, T4), coinc(T1, T3))`

`cross(coinc(F1, F2, F4), coinc(F1, F3))`

Non-linear mediated polylogics can be linked to m-categories and the relational semantics of modal logics for further explanation and formalization. On the other hand it may turn out that non-linear, tabular polylogics can serve as logical foundations of m-categories (Tom Henster) which shouldn't be based on predicate logics only.

On the base of the given semantic interpretation of the graphs all kinds of *Birkhoff arithmetics* are accessible now to a polylogical interpretation.

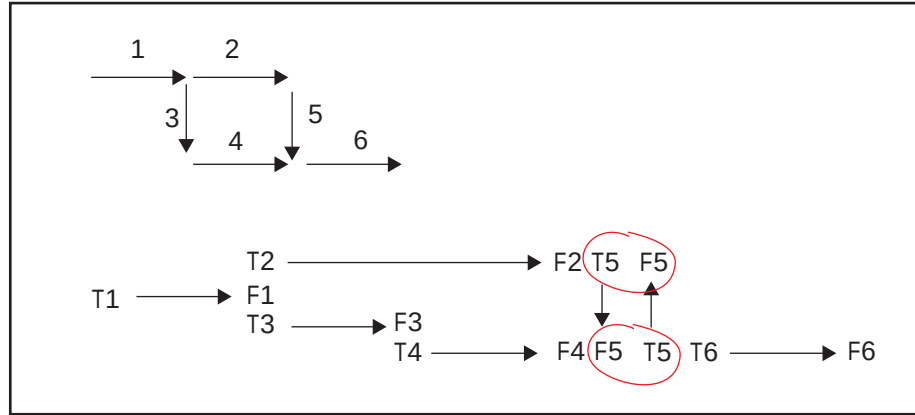
More information can be found in the chapter "Combinatorics". This "m-categorical" approach is supporting the proposed tabular and textual strategies of ConTeXtures. Only for the elementary case of $m=3$, star and line structures are coinciding. This simple structural coincidence may be the hidden reason of profound epistemological controversies in philosophy and sociology.

2.3 Distributed cyclic patterns

Double interpretations

The following example deals with conflicts of modeling the exchange relation. As a result two different, partly incomplete interpretations have to be accepted.

Diagramm 4 Double interpretations



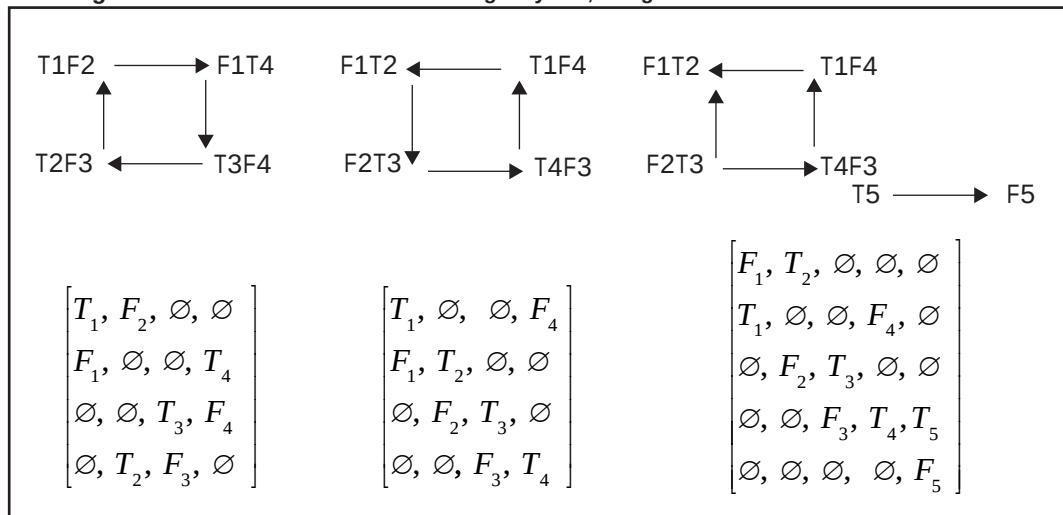
Between F1T2T3 and F3T3, F4T6 proper exchange relations are realized.

This unambiguous situation of mediation is disturbed for (F2T5, F4F5) and (F2F5, F4T5). Both interpretations are correct in realizing one exchange and one coincidence relation, thus both have to be considered.

Cyclic patterns

Cyclic patterns are of special interest because they introduce some kind of circularity into the structure of architectonics which is independent to the circularity, say, of self-referential sentences on a linguistic level of logics. Again, only elementary examples can be given here and full-fledged theory of the *general dynamics of architectonics* has to be developed elsewhere.

Diagramm 5 Left and right cycles, a flag and its semantics

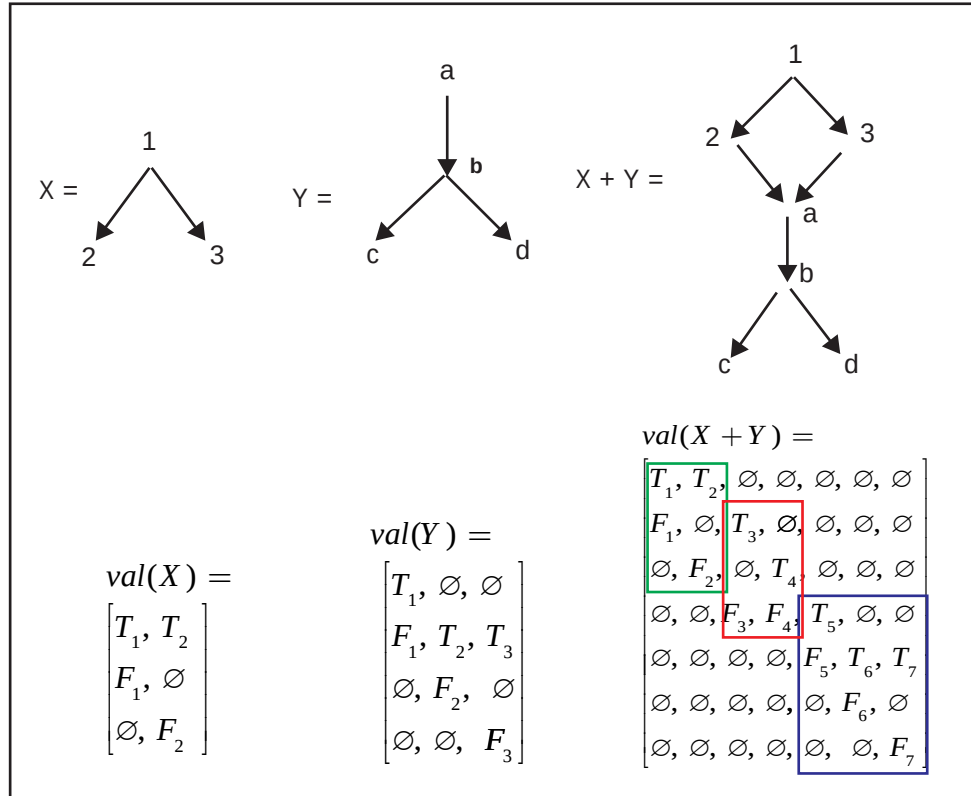


2.4 Birkhoff Arithmetics of Chiasms

Birkhoff arithmetics are used as skeletal structures of the dynamics of composed poly-logical systems. As mentioned above, the dynamic aspects of architectonics, *general dynamics of architectonics*, are not considered for further development of polylogics.

Diagramm 6

Ordinal sum of X and Y



How are formal systems of different architectonics interacting and reflecting each other?

B. Lambda Calculi in Polycontextural Situations

From Lambda to Samba's

Goal: Church's lambda-calculus provides a convenient way of representing meanings, whether meanings of programs or of expressions in English or some other natural language. As a result, it is ubiquitous in computer science, logic, and formal approaches to the semantics of natural language.

The lambda-calculus consists of two things: a *formal language* and an associated notion of *REDUCTION* (roughly equivalent to "computation"). In the context of the lambda calculus, reduction is specifically called lambda-reduction.

<http://ling.ucsd.edu/~barker/Lambda/#lambdareduction>

Paraphrase: The proposed ideas and constructions of "lambda calculi in polycontextural situations", or short: *poly-Lambda Calculus*, aims the possibility to provide a convenient way of representing and calculating ambiguous and complex meanings, especially for common sense thinking and computations. Because in computer science and mathematical linguistics basic ambiguous terms are disallowed by the principle of disambiguation the ideas of poly-Lambda Calculi are not yet perceived seriously.

The proposed poly-Lambda Calculi consists of complex, i.e., distributed and mediated, *formal languages* and associated notions of *REDUCTION* and *METAMORPHOSIS* (roughly similar to trans-computations). In the polycontextural game of disseminated Lambda Calculi, reductions are specifically called poly-lambda-reductions and metamorphic transformations poly-lambda-metamorphosis. Additional to computational reductions, disseminated lambda calculi are involved in interactional and reflectional activities depending on their societal architectonics.

Lambda Calculus, Combinatory Logic and Category Theory are structurally very close. Dissemination and proemiality of them evokes to consider n-category and poly-mathematics as scientific metaphors and source of techniques to study polycontextural formalisms.

Poly-Lambda Calculi starts with the simple idea that complexity is first, simplicity is last. Following the Chinese model of scripturality. But this is not a simple duality of the same. Scientific thinking as invented and promoted by Leibniz is based on a stroke, one and only one, and the identity principle for the linear repetition of the stroke. And its absence as non-stroke. Arithmetically interpreted as one and zero. A stroke is a stroke. In Chinese writing, and as developed in poly-contextural formalisms, a contexture gives place to strokes, is placing strokes and strokes in their motion are enabling contextures. Strokes are written, painted, inscribed, they are not in the mind, they are in the world, unclosing the horizons of the world. These patterns are called morphograms. Lambda Calculi in polycontextural systems are situated, they occur in situations, situs, placed in scriptural contextures. Their meaning has to be interpreted, negotiated.

Svend Ostergaard argues that "the stroke...is pure presentation...The stroke is 'marked' by the incidental since the stroke results from an act that selects within endless possibilities...The stroke... a 'cut' in a continuum..."

<http://www.stefanarteni.net/writings/Polycontextuality/Polycontextuality.html>

In contrast to the lambda calculus and its scriptural linearity and atomicity of its marks – *finite linear sequences of abstract symbols from its alphabet*–, disseminated calculi are written in a tabular complexity playing with ambiguous and paradox inscriptions.

1 Remembering the beginnings

A SET OF POSTULATES FOR THE FOUNDATION OF LOGIC.¹

BY ALONZO CHURCH.²

1. **Introduction.** In this paper we present a set of postulates for the foundation of formal logic, in which we avoid use of the free, or real, variable, and in which we introduce a certain restriction on the law of excluded middle as a means of avoiding the paradoxes connected with the mathematics of the transfinite.

In consequence of this abstract character of the system which we are about to formulate, it is not admissible, in proving theorems of the system, to make use of the meaning of any of the symbols, although in the application which is intended the symbols do acquire meanings. The initial set of postulates must of themselves define the system as a formal structure, and in developing this formal structure reference to the proposed application must be held irrelevant. There may, indeed, be other applications of the system than its use as a logic.

From: Alonzo Church, *Annals of Mathematics*, ser. 2, vol. 33, (1932); in: Philip Wadler.



From foundational studies to main model of computation

Today it is easy forgotten that the leading motivations behind the lambda calculus, but also behind its equivalents, like combinatory logic, Turing machines, and many other formalisms, was to build an absolute foundation of security for mathematics, excluding the appearance of catastrophic paradoxes and antinomies. This aim failed. But interesting insights and techniques into the nature of formalisms, calculus, logic and computation, had been developed and are today everywhere in use. The main strategy to achieve this goal of ultimate foundation was reduction of complexity, as far as possible. This would exclude all kinds of ambiguity, meaning related interpretations, not written on paper. Surprisingly enough, the result of these enormous

stringent conceptualizations and formalizations was that there is no such thing as an absolute security in the fundaments (Entscheidungsproblem/Hilbert). One of the last ideology of scientific reasoning was unmasked. The strict calculus of marks unmasked the mask of mathematics. As Joseph Goguen pointed out, that facing the enormous complexity of different approaches, methods, trends, etc., today, nobody beliefs that there should or could be something like a general ground to secure uniqueness of mathematical and computational thinking.

"In der ursprünglichen Absicht der Begründer (Schönfinkel, Curry und Church) lag nicht nur eine Axiomatisierung des Anwendungsbegriffes für allgemeine Funktionen, sondern eine funktionale Begründung der gesamten Logik und Mathematik überhaupt. Insbesondere Curry und Church haben ursprünglich Systeme aufgestellt, die mit durchaus vernünftig erscheinenden Zusätzen zur kombinatorischen Algebra auch logische Gesetze und Teile der Mathematik miteinbezogen. Diese erweiterten Systeme stellten sich dann als widerspruchsvoll heraus." Engeler, *Metamathematik der Elementar-Mathematik*, Springer 1983, p. 104

2 Sketch of the Lambda Calculus Essentials

2.1 Assembling the elements

The operator terminology of the lambda calculus gives us also a hint how to build a poly-lambda calculus as the mathematical conception and apparatus behind the programming language ARS and its distribution and mediation to the polycontextural notion of ConTextures as poly-ARS.

Lambda calculus as a rewrite system: Barendregt's intro

"A *functional program* consists of an expression E (representing both the algorithm and the input). This expression E is subject to some rewrite rules. Reduction consists of replacing a part P of E by another expression P' according to the given rewrite rules.

In schematic notation $\mathbf{E[P] \rightarrow E[P']}$, provided that $P \rightarrow P'$ is according to the rules.

This process of reduction will be repeated until the resulting expression has no more parts that can be rewritten. This so called normal form E^* of the expression E consists of the output of the functional program." Barendregt, p.8

Syntax

$$x \in V \Rightarrow x \in \Lambda$$

$$M, N \in \Lambda \Rightarrow (MN) \in \Lambda$$

$$M \in \Lambda, x \in V \Rightarrow (\lambda x M) \in \Lambda$$

$$V = \{v, v', v'', \dots\}$$

Set of free variables of $M : FV(M)$

$$FV(x) = \{x\}$$

$$FV(M, N) = FV(M) \cup FV(N)$$

$$FV(\lambda x M) = FV(M) - \{x\}$$

M is a closed λ -term

$$\text{if } FV(M) = \emptyset$$

Substitution,

N for free occurrence of x in M , $M[x := N]$

$$x[x := N] \equiv N;$$

$$y[x := N] \equiv y, \text{ if } x \neq y;$$

$$(M_1 M_2)[x := N] \equiv (M_1[x := N])(M_2[x := N]);$$

$$(\lambda x. M_1)[x := N] \equiv \lambda y. (M_1[x := N])$$

"Nothing happens until a lambda-binding form occurs in construction with an argument, thus:

((lambda var body) argument).

Once a lambda-based binding form occurs with an argument like this, it is possible to reduce the expression to a simpler." Barker

"The lambda calculus is a computational system composed of (1) a term algebra, (*lambda-terms; apply; lambda-abstractions*); (2) a description of the *reduction* steps making up computations; (3) a description of the *meaning* of terms." W. Richard Stark

It seems, for linguistic and philosophic reasons, to be more appropriate and also more suggestive to

name "variable" as "name", thus: *((lambda name body) argument)*. This corresponds to the understanding of abstraction as to give a name. And a name, obviously, is identifying and re-presenting the named or referred object, hiding all details of the named.

To run the scripture of the calculus technical marks, like bracket are included.

(cf. WEB: Chris Hankin, Introduction to Lambda Calculus, autumn 2003, and Henk Barendregt/Erik Barendson, Introduction to Lambda Calculus, 1994)

Definition of the Lambda Calculus

(i) Principle Axiom

$$(\lambda x.M)N = M[x := N] \quad (\beta)$$

for all $M, N \in \Lambda$.

(ii) Logical rules

$$\begin{aligned} \text{Equality :} \quad & M = M \\ & M = N \Rightarrow N = M \\ & M = N, N = L \Rightarrow M = L. \end{aligned}$$

$$\text{Compatibility : } M = M' \Rightarrow MZ = M'Z$$

$$M = M' \Rightarrow ZM = ZM'$$

$$M = M' \Rightarrow \lambda x.M = \lambda x.M'. \quad (\xi)$$

(iii) If $M = N$ is provable in the λ -calculus, then
we sometimes write $\lambda \vdash M = N$.

$$(iv) \lambda x.M = \lambda y.M[x := y], y \notin M \quad (\alpha).$$

Some wordings

"The beta-reduction says that lambda-variables may be replaced by their arguments. A computation is a series of terms $t_0 \rightarrow t_1 \rightarrow \dots \rightarrow t_i \rightarrow t_{i+1}, \dots$ such that each t_i reduces to t_{i+1} ." W.Richard Stark

"An expression may occur in three positions as a component of a larger expression:

1. in the operator position,
2. in the operand position,
3. as the body of another lambda expression.

The lambda expression is the second basic method of assembling a new expression. In their most austere form the expression under consideration may be characterized as follows.

An expression is

either *simple* and is an identifier
or a *lambda expression*
and has a *bound variable* which is an identifier
and a body which is an expression,
or it is *composite*

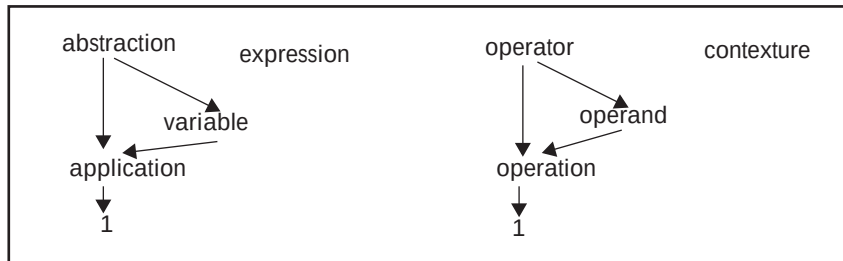
and has an *operator* and an *operand*, both of which are expressions.

A rule is needed for recognizing when the body of a lambda expression ends. The rule is that the body extends as far as it can until it is terminated by a closing bracket, comma, or the end of the whole expression. It follows that parenthesis are only needed to enclose the body if it is a list although they may be used if this improves readability."

W.H. Burge, Recursive Programming Techniques, 1975, p

Variables are identifying identitive objects, objects or operands which are simple and identified by a simple identification. Thus, such objects are the abstract objects, which had been called by Haskell Curry "obs" in the context of his combinatory logic. *Abstractions* and *applications*, operators and operands, are defined over these abstract objects "obs", they inherit their principle of identity. From the linguistics of the lambda calculus, the term-terminology is preferred to the abstract entity-terminology of "obs".

Diagramm 7



The idea of disseminating lambda calculi in polycontextural situations is by no way to challenge those results of foundational studies and applications to computer science.

Uniqueness and identity

Because of the uniqueness of the lambda calculus its operations are always in the super-operator modus of identity, i.e., [id]. Thus the lambda calculus is a morphism of the form: [id]: LC → LC. Because this identity is ubiquitous for LC, it can be neglected; it's trivial. Things are changing dramatically if we have to deal with a multitude of mediated lambda calculi.

Lambda calculus vs. Actor model

The lambda calculus of Alonzo Church can be viewed as the earliest *message passing* programming language. For example the lambda expression below implements a tree data structure when supplied with parameters for a leftSubTree and rightSubTree. When such a tree is given a parameter message "getLeft", it returns leftSubTree and likewise when given the message "getRight" it returns rightSubTree.

```

lambda...(leftSubTree,rightSubTree)
  lambda...(message)
    if (message == "getLeft") then leftSubTree
    else if (message == "getRight") then rightSubTree

```

However, the semantics of the lambda calculus were expressed using variable substitution in which the values of parameters were substituted into the body of an invoked lambda expression. The substitution model is unsuitable for *concurrency* because it does not allow the capability of sharing of changing resources. Inspired by the lambda calculus, the interpreter for the programming language Lisp made use of a data structure called an environment so that the values of parameters did not have to be substituted into the body of an invoked lambda expression. This allowed for sharing of the effects of updating shared data structures but did not provide for concurrency. (Carl Hewitt)

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

2.2 Reductions vs. abstraction rules

The inverse or dual rules to the reduction rules are the abstraction rules.

"Reversing beta-reduction produces beta-abstraction rule." Kenneth Slonneger,
Formal syntax and semantics for programming languages, §5, p.149, 1995

$$\begin{aligned}(\lambda x.M) N &\Rightarrow M [x := N] & (\beta - \text{reduction}) \\(\lambda x.M) N &\Leftarrow M [x := N] & (\beta - \text{abstraction}) \\(\lambda x.M) N &\Leftrightarrow M [x := N] & (\beta - \text{conversion})\end{aligned}$$

for all $M, N \in \Lambda$.

Syntax of Lambda Expressions

1. $t = x, x \in \text{Var}$: variable : operands
2. $t = \lambda x M, x \text{ and } M \text{ are expressions}$: abstraction : operator
3. $t = (MN), M, N \text{ expressions}$: application : operation

β - Reduction Rules.

$$a. (\lambda x.u) t \Rightarrow u [t / x]$$

$$b. \langle t, u \rangle_0 \Rightarrow t$$

$$c. \langle t, u \rangle_1 \Rightarrow u$$

plus "logical rules" for "="

another notation for a :

$$(\lambda x.M) N \xrightarrow{\beta} M [x \longrightarrow N]$$

Lambda calculus BNF syntax

$\langle \lambda \text{ term} \rangle ::=$

$\langle \text{variable} \rangle \mid \langle \text{abstraction} \rangle \mid \langle \text{application} \rangle$

$\langle \text{variable} \rangle := v^*$

$\langle \text{abstraction} \rangle := (\lambda \langle \text{variable} \rangle \langle \lambda \text{ term} \rangle)$

$\langle \text{application} \rangle := (\langle \lambda \text{ term} \rangle \langle \lambda \text{ term} \rangle)$

Or:

variable=reference,

application=synthesis,

abstraction=abstraction in the sense of ARS

(not considering intrinsic differences between ARS and the Lambda Calculus.)

Summary of the reduction rules (Barendregt, 1994)

4.1. DEFINITION. (i) A binary relation R on Λ is called *compatible* (with the operations) if

$$\begin{aligned} M R N &\Rightarrow (ZM) R (ZN), \\ &(MZ) R (NZ) \text{ and} \\ &(\lambda x.M) R (\lambda x.N). \end{aligned}$$

(ii) A *congruence* relation on Λ is a compatible equivalence relation.

(iii) A *reduction* relation on Λ is a compatible, reflexive and transitive relation.

4.2. DEFINITION. The binary relations $\rightarrow_\beta$, $\twoheadrightarrow_\beta$ and $=_\beta$ on Λ are defined inductively as follows.

- (i) 1. $(\lambda x.M)N \rightarrow_\beta M[x := N]$;
2. $M \rightarrow_\beta N \Rightarrow ZM \rightarrow_\beta ZN, MZ \rightarrow_\beta NZ$ and $\lambda x.M \rightarrow_\beta \lambda x.N$.
- (ii) 1. $M \twoheadrightarrow_\beta M$;
2. $M \rightarrow_\beta N \Rightarrow M \twoheadrightarrow_\beta N$;
3. $M \twoheadrightarrow_\beta N, N \twoheadrightarrow_\beta L \Rightarrow M \twoheadrightarrow_\beta L$.
- (iii) 1. $M \twoheadrightarrow_\beta N \Rightarrow M =_\beta N$;
2. $M =_\beta N \Rightarrow N =_\beta M$;
3. $M =_\beta N, N =_\beta L \Rightarrow M =_\beta L$.

These relations are pronounced as follows.

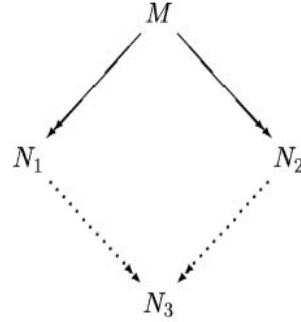
$$\begin{aligned} M \twoheadrightarrow_\beta N &: M \beta\text{-reduces to } N; \\ M \rightarrow_\beta N &: M \beta\text{-reduces to } N \text{ in one step}; \\ M =_\beta N &: M \text{ is } \beta\text{-convertible to } N. \end{aligned}$$

By definition $\rightarrow_\beta$ is compatible, $\twoheadrightarrow_\beta$ is a reduction relation and $=_\beta$ is a congruence relation.

2.3 Church-Rosser-Theorem

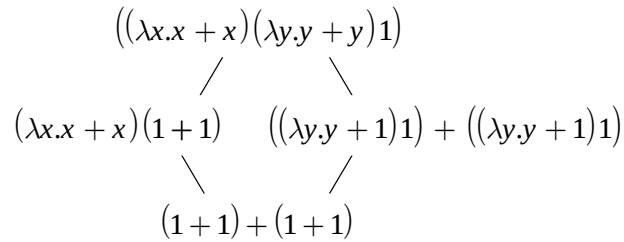
The Church-Rosser Theorem says that if two terms are convertible in the lambda calculus, then there is a term to which they both reduce.

4.9. CHURCH-ROSSER THEOREM. *If $M \rightarrow_{\beta} N_1$, $M \rightarrow_{\beta} N_2$, then for some N_3 one has $N_1 \rightarrow_{\beta} N_3$ and $N_2 \rightarrow_{\beta} N_3$; in diagram*

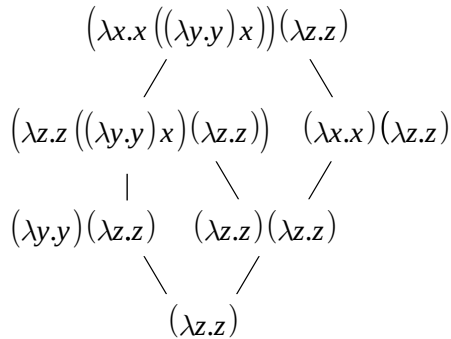


In fact, when a form contains more than one lambda that can be reduced, it does not matter which one is reduced first, the result will be the same. This is known as the Church-Rosser property, or, informally, as the diamond property. Barker

An example



Another example



The reduction rules of the lambda calculus are dictating the strategy of reduction. There is some degree of freedom in the succession of the reduction steps. But if there is a normal form of the expression at all, the reduction steps results in the same one and only one normal form. The normal forms are terminal objects to the initial objects of the axioms of the calculus. This reduction to normal forms is proven with the Church-Rosser Theorem. For modern proof details (cf. Barendregt).

2.4 Connecting the lambda calculus to the rest of the formal world

2.4.1 Connecting to Logic

The AND function of two arguments can be defined as

$$\wedge \equiv \lambda xy.xy(\lambda uv.v) \equiv \lambda xy.xyF$$

The OR function of two arguments can be defined as

$$\vee \equiv \lambda xy.x(\lambda uv.u)y \equiv \lambda xy.xTy$$

Negation of one argument can be defined as

$$\neg \equiv \lambda x.x(\lambda uv.v)(\lambda ab.a) \equiv \lambda x.xFT$$

The negation function applied to “true” is

$$\neg T \equiv \lambda x.x(\lambda uv.v)(\lambda ab.a)(\lambda cd.c)$$

2.4.2 Connecting to Numbers

$$\begin{aligned} 1 &\equiv \lambda sz.s(z) \\ 2 &\equiv \lambda sz.s(s(z)) \\ 3 &\equiv \lambda sz.s(s(s(z))) \\ S &\equiv \lambda wyx.y(wyx) \end{aligned}$$

The lambda calculus is dealing with functions. Thus, natural numbers have to be interpreted as functions. This kind of numbers are called Church Numerals.

<http://www.rbjones.com/rbjpub/>

logic/cl/cl017.htm

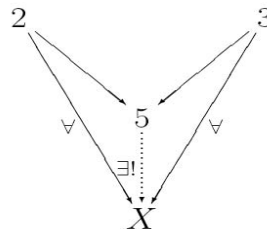
<http://ling.ucsd.edu/~barker/Lambda/#lambdareduction>

Understanding equations

Any interesting equation is really a summary of an interesting *process*. For example:

$$\begin{array}{c} 2 + 3 \\ \parallel \\ 5 \end{array}$$

is short for:



From platonist to constructivist and to constructionist understanding of an equation?

2.5 Combinators

Combinators

$I \equiv \lambda x.x$

$K \equiv \lambda xy.x$

$K_* \equiv \lambda xy.y$

$S \equiv \lambda xyz.xz(yz)$

$W \equiv \lambda fx.fxx$

$K \equiv \langle t, u \rangle_0$

$K_* \equiv \langle t, u \rangle_1$

$S \equiv \lambda xyz.xz(yz)$

$true \equiv K$

$false \equiv K_*$

"Lambda-reduction is a complicated syntactic transformation whose complete and explicit description is quite complex, and whose execution is full of subtle pitfalls that catch even experienced semanticists. You might think, therefore, that the popularity of the lambda-calculus is due to there being no simpler alternative.

But you would be wrong. Combinatory Logic (CL), invented by Moses Schön-

finkel and developed by Haskell Curry and others in the 1920's (note: before the lambda-calculus!), is equivalent in expressive power to the lambda calculus, but much simpler."

<http://ling.ucsd.edu/~barker/Lambda/ski.html>

But if you would think, that the combinatory logic is the simplest possible calculus, you would be wrong again. And the winner, until now, is: Barker, with his Iota-calculus. See for a competition at:

<http://ling.ucsd.edu/~barker/Iota/>

http://www.thinkartlab.com/pkl/media/SUSHIS_LOGICS.pdf

The Y combinator

The formulation of the Y operator in combinatory logic shows very neatly and explicit the character of the involved kind of iterability. Below is the classic formulation of Curry and Feys 1958. An interesting analysis of this formulation can be found at Fitch 1960.



Combinatory definition of **Y**

$$\begin{aligned} Y f &= (W(B f))(W(B f)) \\ &= WS(BWB) f \end{aligned}$$

Proof of **Y** in Combinatory Logic

Proof of $Y f = f(Y f)$:

$$\begin{aligned} Y f &= WS(BWB) f \\ &= S(BWB) ff \\ &= BWB f (BWB f) \\ &= W(B f)(BWB f) \\ &= B f(BWB f)(BWB f) \\ &= f(Y f) \end{aligned}$$

"The combinator **W**, when applied to a function f of two arguments, produces the function of one argument obtained by identifying the two arguments." W.H. Burge

2.6 Fixed-point theorem: the Y-operator

"A fixed point combinator is a *higher-order* function which computes fixed points of other functions. A fixed point of a function on *values* is another value which is left unchanged by that function; for example, 0 and 1 are fixed points of the squaring function. Formally, a value x is a fixed point of a function f if $f(x) = x$.

Fixed point combinators on the other hand are functions on functions. The fix point of a function is another function that is left unchanged by further applications of the fix point combinator."

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

Also the lambda calculus is per se not recursive it is possible to define fixed point functions in terms of non-recursive lambda abstractions. The Y-operator, introduced as a combinator by Haskell Curry, is a famous example of a fixed point operator, also called *paradoxical operator*. Applied to logic it produces paradoxies. The Y-operator is of importance to define recursive functions and other self-referential applications in the framework of the lambda calculus and its application to programming languages. Y is a *second-order function* constructed by lambda-f of lambda-x-of-f-of-x.

FIXEDPOINT THEOREM

(i) $\forall F \exists X FX = X$.

(ii) *There is a fixed point combinator*

$$Y \equiv \lambda f. (\lambda x. f(x x)) (\lambda x. f(x x))$$

such that

$$\forall F F(Y F) = Y F.$$

"Every recursively defined function can be seen as a fixed point of some other suitable function, and therefore, using Y, every recursively defined function can be expressed as a lambda expression. In particular, we can now cleanly define the subtraction, multiplication and comparison predicate of natural numbers recursively."

Thus, the connection between the lambda calculus and recursive number theory is established. As a result we will have the theorem "*All recursive functions are lambda-definable.*" And with that, we have a general model or explication of computability.

Proof. (i) Define $W \equiv \lambda x. F(x x)$ and $X \equiv W W$. Then

$$X \equiv W W \equiv \lambda x. F(x x) W \equiv F(W W) \equiv F X.$$

(ii) *By the proof of (i). (Barendregt)*

$$Y = (\lambda f. (\lambda x. f(x x)) (\lambda x. f(x x)))$$

$$YF = (\lambda f. (\lambda x. f(x x)) (\lambda x. f(x x))) F$$

$$\Rightarrow (\lambda x. F(x x)) (\lambda x. F(x x))$$

$$\Rightarrow F(\lambda x. F(x x)) (\lambda x. F(x x))$$

$$\Rightarrow F(Y F).$$

The re-entry of " $f(x, x)$ " into " $\lambda x. f(x, x)$ " is producing an endless loop.

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

http://people.cs.uchicago.edu/~odonnell/Teacher/Lectures/Formal_Organization_of_Knowledge/Examples/combinator_calculus/

3 Main results

Logic

The Curry-Howard isomorphism implies a relationship between logic and programming: Every valid proof of a theorem of logic corresponds directly to a reduction of a lambda term, and vice versa. Theorems themselves are identified with function type signatures. Specifically, typed combinatory logics correspond to Hilbert systems in proof theory.

Undecidability of equivalence

There is no algorithm which takes as input two lambda expressions and outputs TRUE or FALSE depending on whether or not the two expressions are equivalent. This was historically the first problem for which the unsolvability could be proven. Of course, in order to do so, the notion of algorithm has to be clearly defined; Church used a definition via recursive functions, which is now known to be equivalent to all other reasonable definitions of the notion.

Church's proof first reduces the problem to determining whether a given lambda expression has a normal form. A normal form is an equivalent expression which cannot be reduced any further. Then he assumes that this predicate is computable, and can hence be expressed in lambda calculus.

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

The whole history together

Inge Bethke, Commented bibliography of the lambda calculus, 1999, 587pp.
<http://a9.com/?q=abramski+mccusker+1999&sourceid=mozilla-search>

4 General framework for Lambda Calculi in Contextures

The general framework of poly-Lambda Calculus is very similar to the general framework of ConTeXtures. This is obvious, because ConTeXtures are based on ARS-systems, which are themselves a generalized interpretation of the Lambda Calculus.

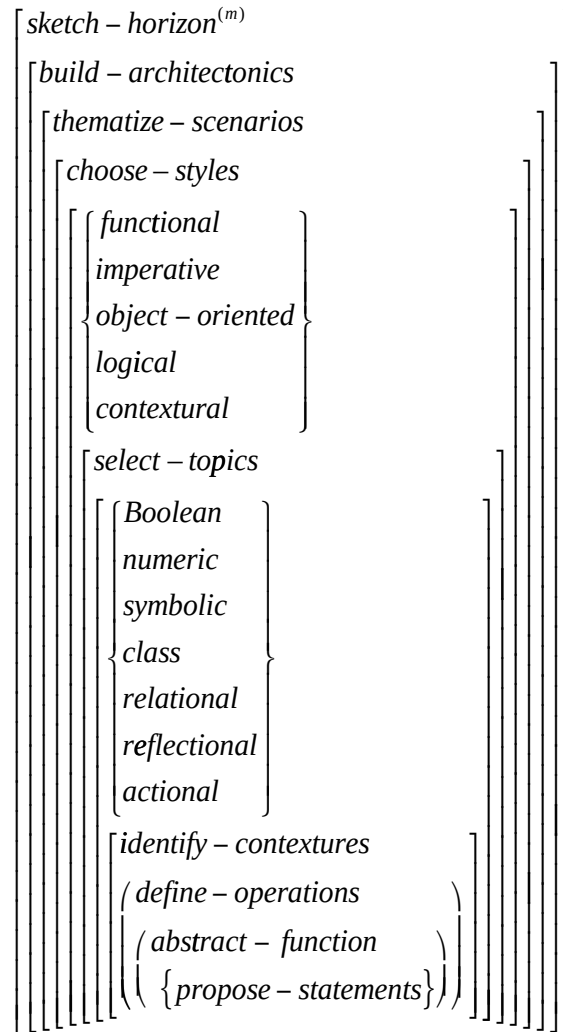
It is not the aim of this introduction of the idea and some formalisms of a poly-Lambda Calculus to repeat the arguments and strategies of ConTeXtures. To understand the whole manoeuvre I recommend to read together the texts on *PolyLogics*, *ConTeXtures* and this one on *poly-Lambda Calculi*. In German there is a well elaborated "*Strukturationen der Interaktivität*" to read, too.

Architectonics

Architectonics in polycontextural systems are, metaphorically, describing and reconstructing the different types of Chinese writings, glyphs and morphograms. From linear chiasms to highly complex interwoven ambiguous patterns.

To learn about poly-Lambda Calculi we follow step by step to more complex writings, interweaving step by step more contextures...

ConTeXtures



Thematization as as-abstraction

Abstraction as giving something a name is identifying something as something; it should be called *is-abstraction*. In contrast, the *as-abstraction* is identifying something as something else. That is, as-abstraction is thematizing something as something in a specific context (situation, constellation, environment). As-abstraction, i.e., thematization, is naming something as something and giving the context of its identification. The context of identification is designed by the architectonics of polycontexturality. Abstraction as giving something a name is emphasizing the act of classification in contrast to creation. As-abstraction is not naming something pre-given but evoking new creations.

as-abstraction = [evocation, contextualization]

is-abstraction = [identification, classification]

4.1 Lambda Calculi disseminated

Here too, the substitution abstraction has to be deliberated from the identity principle to an abstraction in the mode of the *as-category*. Thus, the substitution can happen in the mode of identity or in the mode of sameness. Identity happens if there is no sub-system change and no simultaneous substitution into other sub-systems. The example shows (more or less), how substitutions into different systems happens at once. That is, S1 to S1S2, and S2 to S2S3, and S3 to S3, according to the super-operators id (identity) and bif (bifurcation).

$$\begin{array}{c}
 \text{samba}^{(3)} \\
 (\text{bif}, \text{bif}, \text{id}) \\
 \left[\begin{array}{ccc}
 S_1 : (\lambda x.u) t & \xrightarrow{\beta_1} & u [t \longrightarrow x] \\
 \vdots & \vdots & \vdots \\
 S_2 : (\lambda x.u) t & \xrightarrow{\beta_2} & u [t \longrightarrow x] \\
 \vdots & \vdots & \vdots \\
 S_3 : (\lambda x.u) t & \xrightarrow{\beta_3} & u [t \longrightarrow x]
 \end{array} \right]
 \end{array}$$

Ambiguity, first!

Strictly m-contextural parallel total substitution example as a graphic, non-operable and not animated, metaphor.

Iterability between iteration and alteration

To start with ambiguity is an option, there are no ultimate and natural reasons for it; neither for its opposite, the naturality of simplicity. The iterability of the distributed lambda calculi is defined, in this study, by a reflectional and an interactional dimension of repeatability. The whole study is to understand as an exercise in a new paradigm of iterability, including iteration, repetition, alteration, recursion, disreption, transformation, etc. Terms to be deconstruct and deliberated from their logocentric linearity of scriptural iteration.

"Il n'y a pas de mot, ni en general de signe, qui ne soit construit par la possibilite de se repeter. Un signe qui ne se repete pas, qui n'est pas deja divise par la repetition dans sa premiere fois n'est pas un signe." Derrida

Derrida brought together in his concept of iterability, not only the stream of non-founded events—the mis en abym, but also the alterity of the "iter", the "alter".

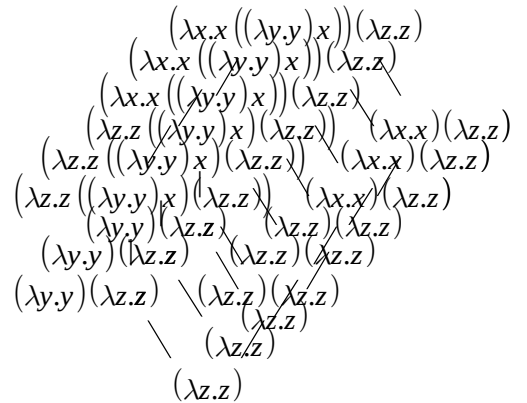
Citation goes together, like translation, with jumps to different contextures, all organized by their own and different rules and different origins. More technically, this procedure of citation is possible only in connection with the interplay of an identity and a neighbor function of a mark.

The problematics are getting more virulent. Behind this generalized idea of iterability and repeatability which is based on a generalized concept of signs and marks, there is something like the strict non-iterability of the non-signs and non-marks, the *kenograms*.

DERRIDA'S MACHINES

<http://www.thinkartlab.com/pkl/media/DERRIDA/Iterability of Zero.html>

The Lambda Calculus as a formal theory of substitution can be embedded into a *grid of contextures*. As a consequence we have to study different sorts of substitutions—partial/total, homogeneous/heterogeneous—, from identive parallel to metamorphic bifurcational transformations (substitutions, reductions, computations). All these different sorts of substitutions also have to be reflected in the logical rules of equality (=) and proof theory in a new calculus.

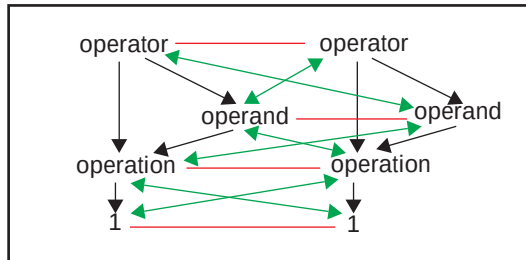


4.2 ARS, ConTeXtures and Lambda Calculi

In the same sense as ARS (a proto-programming language based on lambda calculus) systems can be mapped onto the polycontextural matrix, especially onto reflectional and interactional dimensions, Lambda Calculi can be distributed and mediated. The template of construction will be more or less the same as it is developed here for ConTeXtures. That is, the construction of mediation and the rules of transformation by super-operators, like identity, permutation, reduction and bifurcations, will appear again.

ARS-syntax	Operator-terminology
<expression> ::= <abstraction> <reference> <synthesis>	linguistic ARS- contexture ::= <operator> <operand> <operation>
<abstraction> ::= '(' define <variable> <expression> ')' '(' lambda (' {<variable>}')' <expression> { <expressions> }')'	<operator> ::= operator of operator (operator as operator) operator of operand (operator as operand) operator of operation (operator as operation)
<reference> ::= <variable> <variable> ::= <symbol>	<operand> ::= programming operand <operand> ::= linguistic operand
<synthesis> ::= '(' <expression> { <expression> }')	<operation> ::= operator (operation)

The operator terminology of the lambda calculus gives us also a hint how to build up a poly-lambda calculus as the mathematical conception and apparatus behind the programming language ARS and its distribution and mediation to the polycontextural notion of ConTeXtures as poly-ARS.



In the same sense as ARS systems can be mapped onto the polycontextural matrix, especially onto reflectional and interactional dimensions, Lambda Calculi can be distributed and mediated. The templates of construction will be more or less the same as it is developed for ConTeXtures. That is, the construction of mediation and the rules of

transformation by super-operators, like identity, permutation, reduction and bifurcations, will appear again.

$$\begin{aligned}
 S^1 : P[E] &\rightarrow P[E'] \\
 S^2 : P[E] &\rightarrow P[E'] \\
 S^3 : P[E] &\longrightarrow P[E']
 \end{aligned}$$

"Reduction consists of replacing a part P of E by another expression P' according to the given rewrite rules. In schematic notation **E[P] → E[P']**, provided that $P \rightarrow P'$ is according to the rules." Barendregt, p.8

This kind of rewrite system is distributed over 3 places, including one mediating system. It is mediated as an operator/operand chain realizing the conditions of the proemial relationship, which are the order, exchange and coincidence relations over the operator/operand distinction and 3 contextual loci.

4.3 Free and bound in contextures

The distinction of free and bound variables is itself bound or embedded into/by contextures. A free variable is "bound" by its contexture. The syntactical distinction of free and bound variables is an intra-contextural distinction. Contextures are binding the boundness of intra-contextural distinctions. Free variables are bound by contextures.

The boundness and closure notion has to be distributed over the openness of different contextures. The purpose of the lambda calculus was to formalize the use of functions in a non ambiguous way. To avoid paradoxes and to give a formal explication of the notion of computability. That is, to give a formal model of the vague notion of algorithm. Maybe, the purpose of poly-lambda calculi is to formalize the use of reflectionality and interactivity between distributed formal systems. To domesticate different kinds of self-referential constructions and to give a formal explication of the notion of complex and ambiguous computation as a step to a formal theory of living systems.

All that will have far reaching consequences for further developments, say programming languages, as started with ConTeXtures. As a beginning it seems obvious, that there is some kind of an asymmetry between existing formalism and their applicability to the real world in the sense of Wolfram's *Equivalence Thesis* and the proposed poly-contextural formalisms. This leads to the thesis that there is no such equivalence, say for socio-biological and reflectional situations. And that, maybe, polycontexturality will be more appropriate to develop a formal model for living beings than the known mono-contextural approaches. Artificial life studies and the theory of living systems are focusing on very different topics. Gunther's *"Life as Polycontexturality"* gives us a first hint.

"Almost all processes that are not obviously simple can be viewed as computations of equivalent sophistication (Wolfram 2002, pp. 5 and 716-717).

More specifically, the principle of computational equivalence says that systems found in the natural world can perform computations up to a maximal ("universal") level of computational power, and that most systems do in fact attain this maximal level of computational power. Consequently, most systems are computationally equivalent. For example, the workings of the human brain or the evolution of weather systems can, in principle, compute the same things as a computer. Computation is therefore simply a question of translating inputs and outputs from one system to another."

<http://mathworld.wolfram.com/PrincipleofComputationalEquivalence.html>

4.4 Algebras and co-algebras

"The stroke... a 'cut' in a continuum..."

There is no final stroke... An ink stroke has a beginning and an end, and ones end is the others beginning. No origin, but beginnings of beginnings and ends of ends...

Until now, Lambda Calculi are, from a systematic point of view, algebras (Engeler). They are constructed out of axioms, rules and equations. They form a formal language with its formal rules of reduction, deduction, computation. There is no continuum in the architectonics of such algebras, they are strictly structural in the sense of the word.

The proposed poly-Lambda Calculi are intra-contexturally algebras, too. But they are embedded in a "continuum" of neighbor calculi. The structure of such a continuum of calculi is not algebraic but co-algebraic. There is no first calculus, no initial object as an ultimate and natural origin in a paradigmatic sense. Beginnings, and ends, are chosen by decision. Thus, there is always a beginning and an end, in the realization of concrete polycontextural situation and constellation. Thus, poly-Lambda Calculi are introduced as an interplay of algebras and co-algebras, not necessarily based on category theory but probably on n-categories and polycontexturality. See for first steps in this direction at: <http://www.thinkartlab.com/pkl/media/SKIZZE-0.9.5-medium.pdf>

4.5 General Syntax for distributed lambda calculi

"1.1.1. *Biases and loci*. The basic analytical artifacts (designs) are located "somewhere".

We shall therefore build a system of locations." Jean-Yves Girard, Locus Solum

From locus solus to locus solum; a multitude of disseminated loci in contextural grids.

4.5.1 Tabular rewrite systems

We introduce tabular lambda calculi as a distribution of rewrite systems,

$E[P] \rightarrow E[P']$, over epistemic dimensions, here, reduced to two: the *reflectional* and the *interactional*.

Matrix for a balanced 3-contextural reflectional / interactional rewrite system scheme.

$PM^{(3)}$	S_1	S_2	S_3
S_1	$E[P] \rightarrow E[P']$	$E[P] \rightarrow E[P']$	$E[P] \rightarrow E[P']$
S_2	$E[P] \rightarrow E[P']$	$E[P] \rightarrow E[P']$	$E[P] \rightarrow E[P']$
S_3	$E[P] \rightarrow E[P']$	$E[P] \rightarrow E[P']$	$E[P] \rightarrow E[P']$

The main *computational* rewrite systems are *located* at the diagonal: $S_{i,j}$, $i=j$, and highlighted in red. These systems are mediated by the proemial relationship.

Paraphrase of Barendregt's intro

A *m*-tabular functional programing constellation consists of m -expressions $E^{(m)}$ (representing both the complexion of algorithms and the corresponding output-systems). Constelleations are representing the architectonics and types of complexity of the mediated lambda calculi. Mediated calculi are "chained" and "glued" by the proemial relationship according to the underlying architectonics. These m -expression are subject, intra-contexturally, to some rewrite rules, and object some trans-contextural transformations and displacements according to the super-operator rules of *reflectionality* and *interactionality*. Reflectionality is displacing rewrite systems along the i -axis, interactionality is interacting along the j -axis of the *polycontextural matrix*.

Poly-reduction consists of replacing intra-contexturally a part $P_{i,j}$ of $E^{(m)}$ by another equivalent expression $P_{i,j}'$ or trans-contexturally by an analogue expression $P_{i,k}$ of $E^{(m)}$. Thus, a system can have a prolongation in itself, and prolongations into other, neighboring places, along reflectional and interactional rules of prolongations.

This so called *complexion of normal forms* $E^{(m)*}$ of the m -expression $E^{(m)}$ consists of the output of the introduced (functional) programming constellation. A normal form $NF^{(m)}$ represents the end form of a complex reduction chain. But such a normal form $NF_{i,j}$ can be *shifted* to another contexture, as $NF_{i,k}$, without loosing its characteristics as a NF. Such a shift can happen in both modi, reflective and interactive. Therefore, normal forms are, in a strict sense, not necessarily terminal objects.

A *singular* computation can start at a computational locus $M_{i,j}$, $i=j$, or at any other place of the matrix. A primary computation at a diagonal place can become a secondary system in a transformed matrix as a result of the *architectonic dynamics* of the general matrix. (Not considered in this paper.)

Computation can happen *at once* at different places of the matrix. A *multitude* of computations can process strictly in parallel or in interwoven forms of interactions and reflections.

4.5.2 Reflection and interaction in the polycontextural Matrix

Super-operators are mappings between complex lamda calculi.

Super-operators sops

$$\begin{aligned}
 & \text{mapping sops} : LC^{(m)} \longrightarrow LC^{(m)} : \\
 & id(i, j) : \forall i, j \in s(m) : (LC^{i,j}) \xrightarrow{id} (LC^{i,j}) \\
 & perm(i, j) : \forall i, j \in s(m) : (LC^i, LC^j) \xrightarrow{perm} (LC^j, LC^i) \\
 & red(i, j) : \forall i, j \in s(m) : (LC^i, LC^j) \xrightarrow{red} (LC^i, LC^i) \\
 & bif(i, j) : \forall i, j \in s(m) : (LC^i, LC^j) \xrightarrow{bif} ((LC^i \parallel LC^j), LC^j) \\
 & repl(i, j) : \forall i, j \in s(m) : (LC^i, LC^j) \xrightarrow{repl} ((LC^i \mid LC^j), LC^j) \\
 & sops = \{id, perm, red, bif, repl\}
 \end{aligned}$$

Two examples of iterated reflexional and interactional patterns in bracket and matrix notation for balanced 3-contextural systems.

Bracket and matrix notation

$$\begin{array}{c}
 (O1O2O3) \\
 \left[\begin{array}{c}
 \left[\begin{array}{c} O1 \\ \left(\begin{array}{c} M1M2M3 \\ (G111) \end{array} \right) \end{array} \right] \\
 \left[\begin{array}{c} O2 \\ \left(\begin{array}{c} (G100) \\ M1 \left(\begin{array}{c} M3 \\ M2 \left(\begin{array}{c} M3 \\ (G003) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right] \\
 \left[\begin{array}{c} O3 \\ \left(\begin{array}{c} M1M2M3 \\ (G033) \end{array} \right) \end{array} \right]
 \end{array} \right]
 \end{array}
 \quad
 \begin{array}{c}
 (O1O2O3) \\
 \left[\begin{array}{c}
 \left[\begin{array}{c} O1 \\ \left(\begin{array}{c} M1M2M3 \\ M1 \left(\begin{array}{c} M1 \\ (G110) \left(\begin{array}{c} M1 \\ (G010) \left(\begin{array}{c} M1 \\ (G010) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right] \\
 \left[\begin{array}{c} O2 \\ \left(\begin{array}{c} M1M2M3 \\ (G222) \end{array} \right) \end{array} \right] \\
 \left[\begin{array}{c} O3 \\ \left(\begin{array}{c} M1M2M3 \\ (G033) \end{array} \right) \end{array} \right]
 \end{array} \right]
 \end{array}$$

<i>PM</i>	<i>O1</i>	<i>O2</i>	<i>O3</i>
<i>M1</i>	S_1	$S_{2,1}$	$\emptyset$
<i>M2</i>	S_1	$S_{2,0}$	S_3
<i>M3</i>	S_1	$S_{2,3}$	S_3

reflectional and interactional
patterns and iterations of
reflections.

<i>PM</i>	<i>O1</i>	<i>O2</i>	<i>O3</i>
<i>M1</i>	S_1	S_2	$\emptyset$
<i>M2</i>	$S_{1,1,1,1}$	S_2	S_3
<i>M3</i>	$\emptyset$	S_2	S_3

This bracket and matrix approach is widely developed in ConTeXtures.

4.5.3 Tabular syntax

Full poly-Lambda Calculi are distributed over reflectional and interactional dimensions, building the polycontextural matrix. Thus the syntax has to reflect this kind of 2-dimensionality in a tabular syntactic structure. For short we can say, that tabular syntactics are products of the pre-given classic syntax of lambda calculus modulo the rules of mediation. Mediation is restricting the full range of combinatorial possibilities of combining syntactical systems (formal languages).

$$\begin{aligned}
 \text{poly} - \text{Lambda Calculus} &:= \\
 &[architectonics] \parallel [dissemination] \parallel [interactionality] \parallel [reflectionality] \\
 [architectonics] &:= (\langle \text{complexity} \rangle \langle \text{structuration} \rangle) \\
 [dissemination] &:= (\langle \text{distribution} \rangle \langle \text{mediation} \rangle \langle \text{lambda calculus} \rangle) \\
 [interactionality] &:= (\langle \text{super} - \text{operators} \rangle \langle \lambda \text{ term} \rangle) \\
 [reflectionality] &:= (\langle \text{super} - \text{operators} \rangle \langle \lambda \text{ term} \rangle) \\
 [\text{lambda calculus}] &:= \langle \lambda \text{ term} \rangle
 \end{aligned}$$

Lambda calculi terms belong to the textuality, intra-textuality of the lambda calculus. They are terms of the formal language of the lambda calculus.

Architectonics and dissemination are describing the trans-linguistic and inter-textual properties and processes between different mediated lambda calculi, i.e., between different and discontexturally separated formal languages.

The classic lambda calculus is identical with 1-lambda calculus. The architectonics and dissemination of a 1-calculus are reduced to simple uniqueness. Architectonics in the classical lambda calculus is reduced to an unitarian tectonic, describing intra-textually the construction levels of its formal language. The meta-language of classical lambda calculus is the universal common language, called U-language by Curry. Its structure is mono-contextural. But it would be wrong to think that the meta-language or proto-language of poly-lambda calculi is an U-language, also it would be misleading to presuppose that it is itself mono-contextural.

This situation was discussed intensely in the past in the context of many-valued logics and their interpretation in a meta-language, arising the endless debate about the two-valuedness of the meta-language which would reduce the logico-philosophical relevance of these many-valued logics (Rescher).

$$\begin{aligned}
 PM^{(m)} &:= [reflectional, interactional] \\
 LC^{(m)} &\in PM^{(3)} \Leftrightarrow \\
 \langle \lambda \text{ term} \rangle^{(m)} &\in PM^{(m)} \\
 Syn_{LC}^{(m)} &= diss [Syn_{LC}^1, Syn_{LC}^1, ..., Syn_{LC}^n] \\
 Syn_{LC}^{(m)} : [Syn_{LC}^{(m)}]_{refl, act} &\xrightarrow{\text{super-ops}} [Syn_{LC}^{(m)}]_{refl, act}
 \end{aligned}$$

Again, this study is understanding dissemination restricted to the dimension of interactionality and reflectionality, excluding other behavioral dimensions like *intervention* and *anticipation* of a general of rationality.

Some philosophical and logical reflections about formal systems in general, developed by Michael J. O'Donnell, involving Curry's approach, can be found at:
<http://arxiv.org/abs/cs/9911010>

4.5.4 Gernal syntax of distribution

This general syntax of distributed lambda calculi is not yet very informative because it is simply stating abstractly that the syntax is distributed over all loci of the polycontextural matrix. That is, at each locus we have to consider the rules of the syntax as we know it from the classical case.

Set of free variables of $M^{(m)}$:

$$\begin{aligned} \forall i, j \in s(m) : & FV^{i,j}(M^{i,j}) \\ \left[\begin{array}{l} FV(x) = \{x\} \\ FV(M, N) = FV(M) \cup FV(N) \\ FV(\lambda x M) = FV(M) - \{x\} \end{array} \right] \end{aligned}$$

$\forall i, j \in s(m) :$

$$\left[\begin{array}{l} M^{i,j} \text{ is a closed } \lambda^{i,j} \text{ - term} \\ \text{if } FV^{i,j}(M^{i,j}) = \emptyset \end{array} \right]$$

$\forall i, j \in s(m) : \text{Syn}_{LC}^{i,j}$

$$\left[\begin{array}{l} x \in V \Rightarrow x \in \Lambda \\ M, N \in \Lambda \Rightarrow (MN) \in \Lambda \\ M \in \Lambda, x \in V \Rightarrow (\lambda x M) \in \Lambda \\ V = \{v, v', v'', \dots\} \end{array} \right]$$

This general syntax of distributed lambda calculi is not yet very informative because it is simply demonstrating abstractly that the syntax is distributed over all loci of the polycontextural matrix. That is, at each locus we have to consider the rules of the syntax as we know it from the classical case.

This free abstract distribution of the full syntax over all loci has to be restricted by the rules of mediation. Second, the syntax itself appears more complex with the application of the super-operators, esp. with the interactional operators of bifurcations. The bracket inclosing definitions says, that this package as a whole is distributed over the reflectional and interactional axis of the matrix, denoted simply with the indices i, j . The bracket reflects some kind of contextural closure of the distributed syntactic systems. In another design of dissemination, additional dimensions could be introduced, like *intervention* and *anticipation*.

The sets of the terms of different syntactical systems are not only disjunct but *discontextural*, belonging to different contextures.

Substitution^(m)

$\forall i, j \in s(m) : \text{Subst}^{i,j}$

$$\left[\begin{array}{l} N \text{ for free occurrence of } x \text{ in } M, M[x := N] \\ x[x := N] \equiv N; \\ y[x := N] \equiv y, \text{ if } x \neq y; \\ (M_1 M_2)[x := N] \equiv (M_1[x := N])(M_2[x := N]); \\ (\lambda x. M_1) \lambda x[x := N] \equiv \lambda y. (M_1[x := N]) \end{array} \right]$$

$$\text{Syntax}^{(m)} = \left\{ \begin{array}{l} \text{local distribution :} \\ \forall i, j \in s(m) : \left[\begin{array}{l} \text{syntactic} \\ \text{rules} \end{array} \right]^{i,j} \\ \text{global interaction / reflection :} \\ \text{sops : Syntax}^{i,j} \longrightarrow \text{Syntax}^{i,j} \end{array} \right.$$

Therefore, I have to use in the meta-language some kind of polycontextural quantifiers, *all* and *exists*, but also bifurcational operators, like *at once* and set theoretical operators, implying that the meta-language itself is polycontextural. The syntactic rules as a whole are disseminated over the polycontextural matrix. But this distribution, reflecting the local aspects only, is not yet including the syntactic properties of interaction and reflection.

To add more salt and pepper to the construction, consult Hankin and Barendregt.

4.5.5 General definition of m-lambda calculi

Definition of the m – Lambda Calculus

a. Intra – contextual rules

$\forall i, j \in s(m):$

(i) *Principle Axiom*

$$\left[\begin{array}{l} (\lambda x.M) N = M [x := N] \\ \text{for all } M, N \in \Lambda. \end{array} \right]^{i,j} (\beta)$$

(ii) *Logical rules*

$$\text{Equality : } \left[\begin{array}{l} M = M \\ M = N \Rightarrow N = M \\ M = N, N = L \Rightarrow M = L \end{array} \right]^{i,j}$$

$$\text{Compatibility : } \left[\begin{array}{l} M = M' \Rightarrow MZ = M'Z \\ M = M' \Rightarrow ZM = ZM' \\ M = M' \Rightarrow \lambda x.M = \lambda x.M' \end{array} \right]^{i,j}$$

$$(iii) \left[\begin{array}{l} \text{If } M = N \text{ is provable in the } \lambda\text{-calculus, then} \\ \text{we sometimes write } \lambda \mapsto M = N \end{array} \right]^{i,j} (\xi)$$

$$(iv) \left[\lambda x.M = \lambda y.M [x := y], y \notin M \right]^{i,j} (\alpha).$$

b. Trans – contextual rules

$[M = M]^{i,j}$ as a complex ion, not a simple identity.

4.5.6 Super-operators

As the general syntactic rules this definition of m-lambda calculi is not telling us much about the trans-contextual rules which are yielding between different calculi. One step to fill the gap is done by the super-operators which are operating as identity, replication, reduction, permutation and bifurcation between different distributed calculi.

Super-operators applied on lambda calculi and their syntax.

$$\begin{aligned}
 & \text{Syn}_{LC}^{(m)} : [\text{Syn}_{LC}^{(m)}]_{\text{refl}, \text{act}} \xrightarrow{\text{super-ops}} [\text{Syn}_{LC}^{(m)}]_{\text{refl}, \text{act}} \\
 & \text{super-ops} = \{id, \text{perm}, \text{red}, \text{bif}, \text{repl}\} \\
 & id : \forall i, j \in s(m) : (\text{Syn}_{LC}^{i,j}) \xrightarrow{id} (\text{Syn}_{LC}^{i,j}) \\
 & \text{perm}(i, j) : \forall i, j \in s(m) : (\text{Syn}_{LC}^i, \text{Syn}_{LC}^j) \xrightarrow{\text{perm}} (\text{Syn}_{LC}^j, \text{Syn}_{LC}^i) \\
 & \text{red}(i, j) : \forall i, j \in s(m) : (\text{Syn}_{LC}^i, \text{Syn}_{LC}^j) \xrightarrow{\text{red}} (\text{Syn}_{LC}^i, \text{Syn}_{LC}^i) \\
 & \text{bif}(i, j) : \forall i, j \in s(m) : (\text{Syn}_{LC}^i, \text{Syn}_{LC}^j) \xrightarrow{\text{bif}} ((\text{Syn}_{LC}^i \parallel \text{Syn}_{LC}^j), \text{Syn}_{LC}^j) \\
 & \text{repl}(i, j) : \forall i, j \in s(m) : (\text{Syn}_{LC}^i, \text{Syn}_{LC}^j) \xrightarrow{\text{repl}} ((\text{Syn}_{LC}^i \mid \text{Syn}_{LC}^i), \text{Syn}_{LC}^j)
 \end{aligned}$$

4.6 Signatures for m-lambda calculi

A lambda term, object, entity belongs to a lambda calculus—or it doesn't belong to it. Then it is not a lambda term. In a situation of a multitude of calculi, a term which doesn't belong to a calculus, may belong to another one. To indicate this situation we can use signed terms and formulas as introduced in the chapter "architectonics" and in ConTeXtures. These signatures are not necessarily connected with logical meanings. They can represent all kinds of framework related basic oppositions in formal systems, like true/false, acceptance/rejection, opponent/proponent, antecedent/succedent, marked/unmarked, etc.

4.6.1 Signatures as locators

Signatures are a kind of place-holders, locators, not only for formulas but for formal systems. They are positioning formal systems in the polycontextural grid.

$$\begin{aligned} \forall^{(m)} H^{(m)} : H^{(m)} \in LC^{(m)} \\ T_i H_i \text{ iff } H_i \in LC_i \\ F_i H_i \text{ iff } H_i \notin LC_i, i \in s(m) \\ H_i \notin LC_i \text{ iff } H_{i+1} \in LC_{i+1} \end{aligned}$$

Intra – contextural Postulates :

Uniqueness

$$T_i H_i, T_i K_i \Rightarrow (H_i K_i) \in LC_i$$

$$H_i \in LC_i \Rightarrow H_i = H_i$$

Symmetry and Transitivity for =,

Composition

$$H_i^1 = H_i^2, H_i^3 = H_i^4 \Rightarrow (H_i^1 H_i^3) = (H_i^2 H_i^4),$$

Extensionality

$$H_i^1, H_i^2 \in LC_i \text{ and } \forall H_i^3 \in LC_i :$$

$$(H_i^1 H_i^3) = (H_i^2 H_i^3) \Rightarrow H_i^1 = H_i^2.$$

Intra-Extensionality:

This means that H_1 and H_2 are the same operator iff they always yield the same result when applied to an arbitrary entity H_3 . This postulate yields intra-contexturally for all positioned lambda calculi.

Orthogonality for sameness

What is yet missing is an *orthogonal* extensionality postulate which yields between terms of different calculi. That is a kind of a trans-contextural extensionality, not for identity, but considering the sameness of terms.

Additional to the intra-contextural equality and comparability postulates for orthogonal features of disseminated calculi have to be developed.

4.6.2 Signatures and Conditions of Mediation

Locators are placing terms in a complexion but they don't rule the composition of the complexion. Depending on the architectonics of the complex formal system combinations have to realize the condition of mediation defined by the architectonics. Thus, not all formal combinations are accepted to build a complexion.

4.7 Lambda calculi as trees

Syntactic systems have representations as trees. The root of the tree marks, additionally to its function as the root of the following branches, the *location* of the tree in the grid. This location is marked by a signature, which makes the tree a signed tree. This fully in the sense of Smullyan's approach to logics, but not for trees only but for forests and rhizomes of trees. The picture below, from O'Donnell, shows well the tree structure of the basic combinators, K and S, and thus also for the lambda calculus. Obviously, K and S, are producing one and only one tree. Therefore it is superfluous to sign it in its uniqueness. The uniqueness of formal systems is independent of their graphical representation as trees or as linear structures. What counts alone, is their uniqueness.

Diagramm 8

Figur 2: Derivation rules for Combinatory Calculus

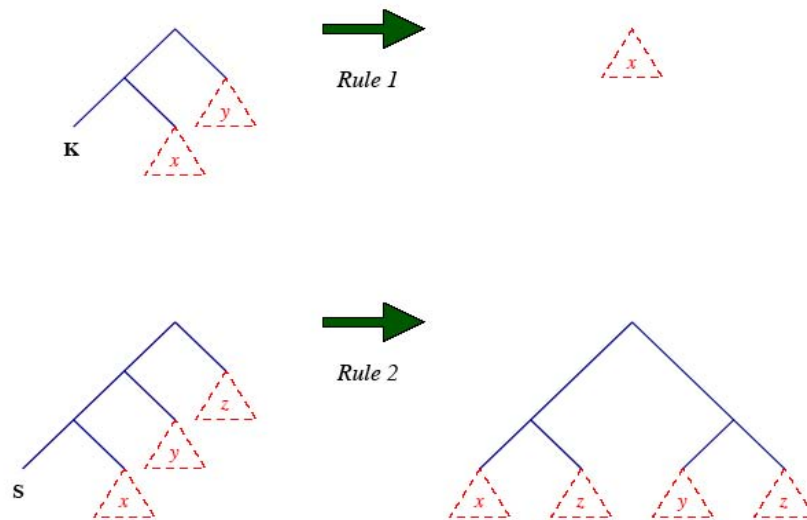


Figure 2: Derivation rules for the Combinator Calculus

- The system deals entirely with finite binary branching tree diagrams, where the end of each path is labelled with exactly one of the symbols 'S' or 'K'. Such a tree diagram is called a *combinator*.
- You may start with any combinator.
- In Figure 2, the x , y , and z in dashed triangles may be replaced by any combinators, as long as in each application of a rule, each of the x triangles is replaced by a copy of the same combinator, similarly for each of the y triangles and each of the z triangles.
- When a structure of the form given by the left-hand side of one of the two rules in Figure 2 appears anywhere within a combinator, you may replace that structure by the corresponding structure on the right-hand side of the same rule.

Such tree-presentations are well known for graph-reduction techniques. O'Donnell resumes his interpretation of combinatory logic by trees as follows:

The useful interpretations of the Combinator Calculus are too long to describe here. The interesting qualities of this formal system for our purposes are

- it is best described in terms of binary branching tree diagrams, instead of sequences of symbols;
- although it has only two rather simple rules, it is capable of deriving values of every function that is computable by every other formal system.

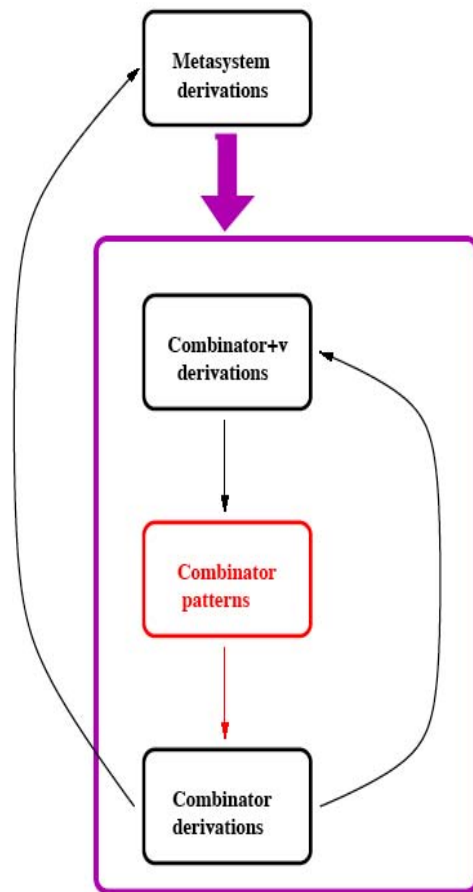
O'Donnell goes on by emphasizing the immense self-reflectional power of formal systems, mainly based on the fact that they are dealing with patterns and not on the manipulation of concrete chains of marks.

The idea of a schematic derivation is worth some attention, as it illustrates the highly reflexive way in which formal systems provide reasoning power. Most of the intuitively important observations about formal systems are schematic—they are observations of *patterns* in the derivations of the formal system, rather than individual derivations. But, there is another formal system containing the derivations of the Combinator Calculus, and also derivations with formal variable symbols. Individual derivations in the Combinator Calculus with variables correspond to schematic patterns of derivations in the Combinator Calculus, in a rigorous way. There is yet another formal system that models the correspondence between schematic derivations in the Combinator Calculus and derivations in the Combinator Calculus with variables.

But, the trickiest twists are yet to come. The Combinator Calculus was designed specifically to be able to simulate the behavior of systems with variables, in a variable-free style. So, the Combinator Calculus contains a precise model of the behavior of the Combinator Calculus with variables, and therefore single derivations in the Combinator Calculus can demonstrate the behaviors of schematic derivations in the Combinator Calculus. And, there's a formal system that models the correspondence between the Combinator Calculi with and without variables, and the Combinator Calculus contains a model of that system, and Figure 5 suggests the systems and relations described above, but of course the real picture is infinitely large, and infinitely more complicated.

From: Michael J O'Donnell, *The Sources of Certainty in Computation and Formal Systems*, 1999
<http://arxiv.org/abs/cs/9911010>

Diagramm 9 Figur 5. Variation on Combinator Calculus and their modeling relations



This abstractive characteristics of formal systems, enabling highly complex reflectionality is fully appreciated by my approach to distribute lambda calculi and combinatory logics. They are in a strict sense formal systems and understood as formal systems.

It seem that this kind of reflectionality is hierarchic in a strict sense. This is not excluding some kind of "tangled reflexivity" but it is strictly not heterarchic, not allowing strict simultaneity of formal systems of different levels of reflectionality. Neither there is any kind of interactionality between simultaneous acting formal systems at all.

O'Donnell gives a very clear explanation of Haskell Curry's world in "*Outline of a Formalistic Philosophy of Mathematics*". It may be understood as intra-contextural reflectionality, like introspection (Smith).

At the end of the exercise O'Donnell brings us back to a definition of a formal system, confirming all the elements of logocentric formalisms: linearity, atomicity and non-ambiguity. The very obstacles of liveliness, creativity, complexity beyond the crystalline security of the "ultimate lambda powers".

Liveliness appears as the ultimate security strategy for living beings.

(cf. http://www.thinkartlab.com/pkl/media/SUSHIS_LOGICS.pdf)

Definition 1 (Formal system)

- A *formal alphabet* is a finite set of discrete symbols, reliably distinguishable from one another.
- A *formal language* is a set of finite discrete arrangements of symbols from a given formal alphabet, with a clear and unambiguous characterization of the relevant qualities of an arrangement.
- A *formal system* is a system of rules for deriving some of the arrangements of symbols from a given formal language, with a clear and unambiguous characterization of the manipulations that are and are not allowed as steps in such derivations.

4.7.1 Lambda calculi in forests

Our job is to hold up the abstractness and operativity of formal systems and nevertheless involve them into the complex interplay of interaction and reflection.

Disseminated lambda calculi or combinatory logics are characterized by their location. They don't come as unique but as distributed uniqueness, that is, as multitudes. Locally, they are unique, globally, they are pluralities. It is convenient to sign their plurality, to mark it with a signature. These signatures are distributed over the structure of their architectonics. Depending on the architectonics, step by step, some small forests are constructed. There is no doubt, that problems of circularity in the constructions are arising: Which calculus is calculating the calculi? But at least there is some epistemological space stretched to tackle the question. The question is similar to Jean-Yves Girard's *From the Rules of logic to the Logic of rules*. (Locus Solum)

At first we have something like trees of trees, second-order trees: Trees of lambda calculi and lambda calculi as trees of formulas. This is not in focus now, but the answer lies in the understanding of morphogramatics; the "calculus" of "locations".

Liveliness and undecidability

Jumping between trees creates new forms of undecidability, new kinds of unpredictability as liveliness. Liveliness is the new kind of undecability in complexions of formal systems. Liveliness is not a loss of control and security but a guaranty of viability.

Equality of formulas vs. bisimulation of behaviors

5 Lambda Calculi in a 3-contextural situation

5.1 Syntactic rules

local Syntax $Syn_{LC}^{(3)}$

$$x^{(3)} = [x^1, x^2, x^3]$$

$$V^{(3)} = [V^1, V^2, V^3]$$

$$V = \{v, v', v'', \dots\}$$

$$V^{(3)} = [\{v, v', v'', \dots\}^1, \{v, v', v'', \dots\}^2, \{v, v', v'', \dots\}^3]$$

$$\Lambda^{(3)} = [\Lambda^1, \Lambda^2, \Lambda^3]$$

$$\in^{(3)} = [\in^1, \in^2, \in^3]$$

$$x^{(3)} \in^{(3)} V^{(3)} \Rightarrow \Rightarrow \Rightarrow x^{(3)} \in^{(3)} \Lambda^{(3)}$$

$$M^{(3)}, N^{(3)} \in^{(3)} \Lambda^{(3)} \Rightarrow \Rightarrow \Rightarrow (M^{(3)} N^{(3)}) \in^{(3)} \Lambda^{(3)}$$

$$M^{(3)} \in^{(3)} V^{(3)}, x^{(3)} \in^{(3)} V^{(3)} \Rightarrow \Rightarrow \Rightarrow (\lambda x^{(3)} M^{(3)}) \in^{(3)} \Lambda^{(3)}$$

partial

$$\forall i, j \in s(3): M^{(3)} \in V^{(3)}, x^{(3)} \in V^{(3)} \Rightarrow \Rightarrow \Rightarrow (\lambda x^{i,j}. M^{(3)}) \in \Lambda^{(3)}$$

$$(\lambda x^i. M^{(3)}) \in \Lambda^{(3)} = [M^1 \in V^1, (\lambda x^i. M^i) \in \Lambda^2, M^3 \in V^3]$$

If the contextural situation is known, all kinds of notational abbreviation can be introduced to avoid tedious pedantry and to be called a *formalistic maniac* (Paul Feyerabend, Against Method).

Set of free variables of $M^{(3)}$:

$$\forall i, j \in s(3): FV^{i,j}(M^{i,j})$$

$$FV(x^{(3)}) = [\{x\}^1, \{x\}^2, \{x\}^3]$$

$$FV(M^{(3)}, N^{(3)}) = FV(M) \cup^{(3)} FV(N)$$

$$FV(\lambda x M^{(3)}) = FV(M^{(3)}) - - - \{x\}^{(3)}$$

$$\forall i, j \in s(3):$$

$M^{i,j}$ is a closed $\lambda^{i,j}$ - term

if $FV^{i,j}(M^{i,j}) = \emptyset^{i,j}$

This suggests a balanced, fully parallel syntax for 3-contextural systems. The syntactical influence of the super-operator is not yet modeled.

5.1.1 General definition of 3-lambda calculi

Definition of the m – Lambda Calculus

a. Intra – contextual rules

$\forall i, j \in s(3):$

(i) *Principle Axiom*

$$\left[\begin{array}{l} (\lambda x.M)N = M[x := N] \\ \text{for all } M, N \in \Lambda. \end{array} \right]^{i,j} (\beta)$$

(ii) *Logical rules*

$$\text{Equality : } \left[\begin{array}{l} M = M \\ M = N \Rightarrow N = M \\ M = N, N = L \Rightarrow M = L \end{array} \right]^{i,j}$$

$$\text{Compatibility : } \left[\begin{array}{l} M = M' \Rightarrow MZ = M'Z \\ M = M' \Rightarrow ZM = ZM' \\ M = M' \Rightarrow \lambda x.M = \lambda x.M' \end{array} \right]^{i,j}$$

$$(iii) \left[\begin{array}{l} \text{If } M = N \text{ is provable in the } \lambda\text{-calculus, then} \\ \text{we sometimes write } \lambda \mapsto M = N \end{array} \right]^{i,j} (\xi)$$

$$(iv) \left[\lambda x.M = \lambda y.M[x := y], y \notin M \right]^{i,j} (\alpha).$$

b. Trans – contextual rules

Again, the combinatorial distribution of over the reflectional and interactional dimensions of the full general definition of 3-lambda calculi has to be restricted by the conditions of mediation. Thus the combination is *modulo* condition of mediation (mod CM). Otherwise it wouldn't be a mediated system.

$$LC^{(3)} = \pi^{i,j} \left[\begin{array}{l} \text{General definition} \\ \text{of 3-lambda calculi} \\ \text{a. intra : (i) – (iv)} \end{array} \right]_{// \text{MOD CM}}$$

The condition of mediation is a parameter which has different definitions. One classic definition is given by the conditions of "linear chiasms" as realized by the proemial relation. But other definitions are possible and in dynamic situations, not all systems have to be mediated. That is, the *process of mediation* can be in focus, while here, mediated systems are prevailing the study, letting the dynamics to another study.

Without any mediation, systems would simply be a repetition of the known unitarian calculus, maybe as a n-tupel collection or as different applications of it.

5.1.2 Notational variants

$$X^{(3)}, Y^{(3)} \in CL_{linear}^{(3)} \Rightarrow \left[\begin{array}{c} \langle x^1, y^1 \rangle \langle x^2, y^2 \rangle \\ \langle x^3, y^3 \rangle \end{array} \right]$$

Basic linear-matrix of 3-contextural pairs.

$$\Rightarrow \left[\langle x^1, y^1 \rangle \langle x^2, y^2 \rangle \langle x^3, y^3 \rangle \right]$$

$$\left[\begin{array}{c} \langle t, u \rangle^{1.1}, \langle t, u \rangle^{2.1}, \langle t, u \rangle^{3.1} \\ \langle t, u \rangle^{1.2}, \langle t, u \rangle^{2.2}, \langle t, u \rangle^{3.2} \\ \langle t, u \rangle^{1.3}, \langle t, u \rangle^{2.3}, \langle t, u \rangle^{3.3} \end{array} \right]$$

General pair-matrix

The pair-matrix shows the full distribution of <t, u>-pairs over the tabular matrix of complexity 3.

5.1.3 Signatures and Conditions of Mediation

$$\begin{aligned} \forall^{(3)} H^{(3)} : H^{(3)} &\in LC^{(3)} \\ H_1 &\notin LC_1 \text{ iff } H_2 \in LC_2 \\ H_1 &\in LC_1 \text{ iff } H_3 \in LC_3 \\ H_2 &\notin LC_2 \text{ iff } H_3 \notin LC_3 \end{aligned}$$

$$\begin{aligned} T_1 H_1 &\cong T_3 H_3 \\ F_1 H_1 &\approx T_2 H_2 \\ F_2 H_2 &\cong F_3 H_3 \end{aligned}$$

Locators are placing terms in a complexion but they don't rule the composition of the complexion. Depending on the architectonics of the complex formal system combinations have to realize the condition of mediation defined by the architectonics. Thus, not all formal combinations are accepted to build a complexion. Standard combinations, as shown below, can have several permutations. If we start with the full pattern S1S2S3, all other patterns are results of the super-operators permutation (perm) and reduction (red). But this is obviously only halve the story because we are excluding the tabular patterns of interactional/reflectional constellations.

$$\left[\begin{array}{c} X^{1.3} \\ X^{1.2} \\ X^{2.3} \end{array} \right] \equiv \left[\begin{array}{c} T_1, \emptyset, T_3 \\ F_1, T_2, \emptyset \\ \emptyset, F_2, F_3 \end{array} \right]$$

$$\left[\begin{array}{c} X^{1.1} \\ X^{1.2} \\ X^{2.1} \end{array} \right] \equiv \left[\begin{array}{c} T_1, \emptyset, T_1 \\ F_1, T_2, \emptyset \\ \emptyset, T_2, F_1 \end{array} \right]$$

$$\left[\begin{array}{c} X^{1.1} \\ X^{1.1} \\ X^{1.1} \end{array} \right] \equiv \left\{ \left[\begin{array}{c} T_1, \emptyset, T_1 \\ F_1, F_1, \emptyset \\ \emptyset, F_1, F_1 \end{array} \right], \left[\begin{array}{c} T_1, \emptyset, T_1 \\ F_1, F_1, \emptyset \\ \emptyset, T_1, T_1 \end{array} \right], \left[\begin{array}{c} T_1, \emptyset, T_1 \\ T_1, T_1, \emptyset \\ \emptyset, F_1, F_1 \end{array} \right], \left[\begin{array}{c} T_1, \emptyset, T_1 \\ T_1, T_1, \emptyset \\ \emptyset, T_1, T_1 \end{array} \right] \right\}$$

$$(X^{(3)}) \in CM^{(3)} :$$

$$\left\{ \left[\begin{array}{c} X^{1.3} \\ X^{1.2} \\ X^{2.3} \end{array} \right] \left[\begin{array}{c} X^{1.1} \\ X^{1.2} \\ X^{2.1} \end{array} \right] \left[\begin{array}{c} X^{1.1} \\ X^{1.1} \\ X^{1.1} \end{array} \right] \right\}$$

The main decision is, on which tectonic level of a formal system are we establishing the "inside/outside"-relation of mediated contextures.

From this, other mediations can be derived by permutation.

5.2 Super-operators

global Syntax⁽³⁾ :

super – ops = {*id*, *perm*, *red*, *bif*, *repl*}

id (*i*, *j*) : $\forall i, j \in s(3) : (Syn_{LC}^{i,j}) \xrightarrow{id} (Syn_{LC}^{i,j}), i, j \in s(3)$

perm (*i*, *j*) : $\forall i, j \in s(3) : (Syn_{LC}^i, Syn_{LC}^j) \xrightarrow{perm} (Syn_{LC}^j, Syn_{LC}^i)$

red (*i*, *j*) : $\forall i, j \in s(3) : (Syn_{LC}^i, Syn_{LC}^j) \xrightarrow{red} (Syn_{LC}^i, Syn_{LC}^i)$

bif (*i*, *j*) : $\forall i, j \in s(3) : (Syn_{LC}^i, Syn_{LC}^j) \xrightarrow{bif} ((Syn_{LC}^i \parallel Syn_{LC}^j), Syn_{LC}^j)$

repl (*i*, *j*) : $\forall i, j \in s(3) : (Syn_{LC}^i, Syn_{LC}^j) \xrightarrow{repl} ((Syn_{LC}^i \mid Syn_{LC}^i), Syn_{LC}^j)$

$X^{(m)} \in CM \Rightarrow sops(X^{(m)}) \in CM :$

$X^{(m)} \in CM \Rightarrow id(X^{(m)}) \in CM$

$X^{(m)} \in CM \Rightarrow repl(X^{(m)}) \in CM$

$X^{(m)} \in CM \Rightarrow perm(X^{(m)}) \in CM$

$X^{(m)} \in CM \Rightarrow red(X^{(m)}) \in CM$

$X^{(m)} \in CM \Rightarrow bif(X^{(m)}) \in CM$

5.3 A family of substitutions

total / partial, local / global, simple / multiple, conservative / transformative

Substitution⁽³⁾

$\forall i, j \in s(3): \text{Subst}^{i,j}$

N for free occurrence of x in M , $M[x := N]$

$$x^{(3)} \begin{bmatrix} x^1 := N^1 \\ x^2 := N^2 \\ x^3 := N^3 \end{bmatrix} \equiv [N^1, N^2, N^3] \equiv N^{(3)};$$

$$x^{(3)} \begin{bmatrix} x^1 := N^1 \\ x^2 := x^2 \\ x^3 := N^3 \end{bmatrix} \equiv [N^1, x^2, N^3];$$

$$y^{(3)} \begin{bmatrix} x^1 := N^1 \\ x^2 := N^2 \\ x^3 := N^3 \end{bmatrix} \equiv y^{(3)}, \text{ if } x^{(3)} \neq y^{(3)};$$

$$y^{(3)} \begin{bmatrix} x^1 := N^1 \\ x^2 := N^2 \\ x^3 := N^3 \end{bmatrix} \equiv [N^1, y^2, y^3], \text{ if } x^{(3)} = y^{(3)};$$

$$\left(M^{(3)}_1 M^{(3)}_2 \right) \begin{bmatrix} x := N \\ x := N \\ x := N \end{bmatrix} \equiv \left(M^{(3)}_1 \begin{bmatrix} x := N \\ x := N \\ x := N \end{bmatrix} \right) \left(M^{(3)}_2 \begin{bmatrix} x := N \\ x := N \\ x := N \end{bmatrix} \right);$$

$$\left(M^{(3)}_1 M^{(3)}_2 \right) \begin{bmatrix} x := x \\ x := N \\ x := N \end{bmatrix} \equiv \left(M^{(3)}_1 \begin{bmatrix} x := x \\ x := N \\ x := N \end{bmatrix} \right) \left(M^{(3)}_2 \begin{bmatrix} x := x \\ x := N \\ x := N \end{bmatrix} \right);$$

$$\left(\lambda x. M^{(3)}_1 \right) \lambda x \begin{bmatrix} x := N \\ x := N \\ x := N \end{bmatrix} \equiv \lambda y. \left(M^{(3)}_1 \begin{bmatrix} x := N \\ x := N \\ x := N \end{bmatrix} \right);$$

$$\left(\lambda x. M^{(3)}_1 \right) \lambda x \begin{bmatrix} x := x \\ x := N \\ x := N \end{bmatrix} \equiv \lambda y. \left(M^{(3)}_1 \begin{bmatrix} x := x \\ x := N \\ x := N \end{bmatrix} \right);$$

5.3.1 Metamorphic substitutions: False citations?

A transformative or metamorphic substitution can happen on the base of the internal structure of a full term with its contexture/head/body tectonics. The switch from one contexture to another can be connected with a change in the head and/or in the body of the term. Such a change is not necessarily changing the content (statement) of the term but its significance or relevance. Depending on the architectonics, it may involve a change in the topics: from Numeric to List or Boolean, etc. or it may remain inside the topic, changing only some internal parameters.

$$term = (head, body)$$

$$subst : term \longrightarrow term$$

$$subst_{conservative} : \begin{cases} head \xrightarrow{id} head \\ body \xrightarrow{id} body \end{cases}$$

$$subst_{metamorph} : \begin{cases} head \xrightarrow{chiasm} body \\ body \xrightarrow{chiasm} head \end{cases}$$

Again, all depends on the notion of abstraction.

Polycontextural abstractions are basically "*as-abstractions*".

That is, a name1 as a name2 *is* a name3; short: *X as Y is Z*.

Lambda Calculus is reducing as-abstractions to a single *is-abstraction*: *X is X*.

A head in a contexture as a body in another contexture is a term of that contexture. Something as something else.

5.4 General Constellations of Transformations

There are at least 5 main modi of transformation for distributed lambda calculi. These modi of transformation are ruled by the super-operators (sops). The *identical* modus is intra-contextual and is repeating the classical transformation in each of the distributed calculi. The *permutational* mode is keeping the internal structure of the calculi intact, but is changing their order in the matrix. Similar is the *reductional* mode, but reducing the complexity of the situation. *Replications* are repeating a system at its locus. This replication (reflection) can be iterative (introspective) or metamorphic, transforming some aspects of the situation. *Bifurcational* transformations are interactional realizing themselves at once at their place and at other places. Bifurcational transformations can be seen as structural patterns of co-operation and concurrency.

5.4.1 Notational conventions

$$t, u \in \text{expression} \Rightarrow \langle t, u \rangle \in \text{expression}$$

$$\text{Reduction } b. (\lambda xy.x) : \langle t, u \rangle_0 \Rightarrow t$$

$$\text{Reduction } c. (\lambda xy.y) : \langle t, u \rangle_1 \Rightarrow u$$

$$\begin{aligned} [id, id, id] \langle t, u \rangle^{(3)} &= \begin{bmatrix} \langle t, u \rangle, \emptyset, & \emptyset \\ \emptyset, & \langle t, u \rangle, \emptyset \\ \emptyset, & \emptyset, \langle t, u \rangle \end{bmatrix} \\ &= [\langle t, u \rangle^1, \langle t, u \rangle^2, \langle t, u \rangle^3] \\ [\langle t, u \rangle^1, \langle t, u \rangle^2, \langle t, u \rangle^3]_{111} &= (u^1, u^3, u^2) \\ [\langle t, u \rangle^1, \langle t, u \rangle^2, \langle t, u \rangle^3]_{000} &= (t^1, t^3, t^2) \end{aligned}$$

5.4.2 Modi of interactions

1. Identical
2. Permutational
3. Reductional
4. Bifurcational (interactional)
5. Replicational (reflectional)

5.4.3 Some Beta-reduction rules

$\beta^{(3)}$ – Reduction Rules.

$a_0.$

$samba^{(3)}$

$$\left[\begin{array}{l} (id, id, id) \\ \left[\begin{array}{l} (\lambda x.u) t \Rightarrow \left[\begin{array}{l} u \left[\begin{array}{c} t^1 \xrightarrow{id} x^1 \end{array} \right] \\ u \left[\begin{array}{c} t^2 \xrightarrow{id} x^2 \end{array} \right] \\ u \left[\begin{array}{c} t^3 \xrightarrow{id} x^3 \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right]$$

$a_1.$

$samba^{(3)}$

$$\left[\begin{array}{l} (id, perm, perm) \\ \left[\begin{array}{l} (\lambda x.u) t \Rightarrow \left[\begin{array}{l} u \left[\begin{array}{c} t^1 \xrightarrow{id} x^1 \end{array} \right] \\ u \left[\begin{array}{c} t^2 \xrightarrow{perm} x^3 \end{array} \right] \\ u \left[\begin{array}{c} t^3 \xrightarrow{perm} x^2 \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right]$$

$a_2.$

$samba^{(3)}$

$$\left[\begin{array}{l} (id, bif, id) \\ \left[\begin{array}{l} (\lambda x.u) t \Rightarrow \left[\begin{array}{l} \left[u \left[\begin{array}{c} t^1 \xrightarrow{id} x^1 \end{array} \right] \parallel u \left[\begin{array}{c} t^2 \xrightarrow{bif} x^2 \end{array} \right] \right] \\ u \left[\begin{array}{c} t^2 \xrightarrow{bif} x^2 \end{array} \right] \\ u \left[\begin{array}{c} t^3 \xrightarrow{id} x^3 \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right]$$

$a_3.$

$samba^{(3)}$

$$\left[\begin{array}{l} (id, red_1, id) \\ \left[\begin{array}{l} (\lambda x.u) t \Rightarrow \left[\begin{array}{l} u \left[\begin{array}{c} t^1 \xrightarrow{id} x^1 \end{array} \right] \\ u \left[\begin{array}{c} t^2 \xrightarrow{red} x^1 \end{array} \right] \\ u \left[\begin{array}{c} t^3 \xrightarrow{id} x^3 \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right]$$

The following beta-reduction rules are not explicitly formalized but should be clear enough, at the moment, to understand the mechanisms of the applications of the super-operators to substitutions.

(id, id, id) :

The rules for the identity mapping of substitution are simply repeating the known definition of substitution, distributed over 3 loci, (S123)→(S123).

$(id, perm, perm)$:

This rule involves the substitution into a permutation of the sub-systems S2 and S3, (S123)→(S132).

(id, bif, id) :

This simple application of bifurcation is producing a prolongation of sub-system S2 into S1. Sub-systems S1 and S3 are not touched by this mapping, (S123)→(S1S2, S2, S3).

(id, red, id) :

Here, the operator red1 is reducing, transferring sub-system S2 to the place S1. S2 is repeated at place S1. Thus at place S1, two sub-systems are realized S1 and S2. Sub-system S3 remains at its locus, (S123)→(S1S1, 0, S3).

It is more a question of combinatorics to write down all the possible singular reduction rules. The above example are clear enough to guide to complete the set of rules.

5.4.4 Identical constellations

$$\begin{aligned}
 \text{pattern } [id, id, id] : S_{123} &\Rightarrow S_{123} \\
 b_0. \langle t, u \rangle_{000} &\Rightarrow (t^1 t^2 t^3) \\
 b_1. \langle t, u \rangle_{001} &\Rightarrow (ttu) \\
 b_2. \langle t, u \rangle_{010} &\Rightarrow (tut) \\
 b_3. \langle t, u \rangle_{011} &\Rightarrow (tuu) \\
 c_0. \langle t, u \rangle_{111} &\Rightarrow (uuu) \\
 c_1. \langle t, u \rangle_{110} &\Rightarrow (uut) \\
 c_2. \langle t, u \rangle_{101} &\Rightarrow (utu) \\
 c_3. \langle t, u \rangle_{100} &\Rightarrow (utt)
 \end{aligned}$$

Two main sets of operators: *substitutions* and *super-operators*.

This general approach is not yet answering the questions of fulfilling the *conditions of mediation* (CM). A first step to introduce the conditions of mediation is to restrict the substitution rules towards a more balanced approach. If the brackets are representing ordered pairs of terms then their distribution over the different contextures has to be restricted to fulfill the conditions of mediation. This restriction simply will be a restriction of the possible combinations of the substitution types of the terms of the ordered pair. If we would give up these CM-restrictions the whole enterprise would simply boil down to some

kind of a product calculus as we know it from product logics or combined logics.

$$\begin{array}{ccc}
 (\lambda z. \langle y_{111}, x_{000} \rangle) \langle y^{(3)}, x^{(3)} \rangle & & (\lambda z. \langle y_{100}, x_{001} \rangle) \langle y^{(3)}, x^{(3)} \rangle \\
 \Downarrow & & \Downarrow \\
 \langle \langle y, x \rangle_{111} \langle y, x \rangle_{000} \rangle & & \langle \langle y, x \rangle_{100} \langle y, x \rangle_{001} \rangle \\
 \Downarrow & & \Downarrow \\
 \langle x^{(3)}, y^{(3)} \rangle & & \langle x^1, y^2, y^3 \rangle \langle y^1, y^2, x^3 \rangle \\
 & & \Downarrow \\
 & & [\langle x^1, y^1 \rangle \langle y^2, y^2 \rangle \langle y^3, x^3 \rangle]
 \end{array}$$

All these mediations will be involved in dissolving the unique stability of the Church-Rosser-Theorem, which will, for itself and in itself, nevertheless, remain rock solid, intra-contexturally at each locus, and between the multitude of the loci, of the distribution.

Additionally we have to study the interplay between substitution and super-operators.

$$\begin{array}{c}
 (\lambda z. \langle y_{100}, x_{011} \rangle) \langle y^{(3)}, x^{(3)} \rangle \\
 \Downarrow \\
 \langle \langle y, x \rangle_{100} \langle y, x \rangle_{011} \rangle \\
 \Downarrow \\
 \langle x^1, y^2, y^3 \rangle \langle y^1, x^2, x^3 \rangle \\
 \Downarrow \\
 [\langle x^1, y^1 \rangle \langle y^2, x^2 \rangle \langle y^3, x^3 \rangle]
 \end{array}$$

5.4.5 Permutational constellations

$$\begin{aligned} \text{pattern } [id, perm, id] : S_{123} &\Rightarrow S_{1.3.2} \\ b_0. < t, u >_{111} &\Rightarrow (u^1, u^3, u^3) \end{aligned}$$

$$\begin{aligned} < t, u >^{(3)} &= [< t, u >^1, < t, u >^2, < t, u >^3] \\ [id, perm, id] < t, u >^{(3)} &= [< t, u >^1, < t, u >^3, < t, u >^2] \\ [< t, u >^1, < t, u >^3, < t, u >^2]_{1.1.1} &= (u^1, u^3, u^2) \end{aligned}$$

$$\begin{aligned} \text{pattern } [inv, perm, perm] : S_{123} &\Rightarrow S_{132} \\ b_0. < t, u >_{111} &\Rightarrow (u^1, u^3, u^3) \end{aligned}$$

$$\begin{aligned} < t, u >^{(3)} &= [< t, u >^1, < t, u >^2, < t, u >^3] \\ [inv, perm, perm] < t, u >^{(3)} &= [< u, t >^1, < t, u >^3, < t, u >^2] \\ [< u, t >^1, < t, u >^3, < t, u >^2]_{1.1.1} &= (t^1, u^3, u^2) \end{aligned}$$

$$\begin{aligned} \text{pattern } [inv, perm, perm] : S_{123} &\Rightarrow S_{132} \\ [inv, perm, perm] < t, u >^{(3)} &= [< u, t >^1, < t, u >^3, < t, u >^2] \\ [< u, t >^1, < t, u >^3, < t, u >^2]_{1.1.1} &= (t^1, u^3, u^2) \end{aligned}$$

$$\begin{aligned} \text{pattern } [perm, inv, perm] : S_{123} &\Rightarrow S_{321} \\ [perm, inv, perm] < t, u >^{(3)} &= [< t, u >^3, < u, t >^2, < t, u >^1] \\ [< t, u >^3, < u, t >^2, < t, u >^1]_{1.1.1} &= (u^3, t^2, u^1) \end{aligned}$$

5.4.6 Reductional constellations

$$\begin{aligned}
\langle t, u \rangle^{(3)} &= [\langle t, u \rangle^1, \langle t, u \rangle^2, \langle t, u \rangle^3] \\
[id, red_1, id] \langle t, u \rangle^{(3)} &= [\langle t, u \rangle^1, \langle t, u \rangle^1, \langle t, u \rangle^3] \\
[\langle t, u \rangle^1, \langle t, u \rangle^1, \langle t, u \rangle^3]_{111} &= (u^1, u^1, u^3)
\end{aligned}$$

$$\begin{aligned}
&pattern [id, red, red] : S_{123} \Rightarrow S_{111} \\
b_0. \langle t, u \rangle_{000} &\Rightarrow (t^1 t^1 t^1) \\
b_1. \langle t, u \rangle_{001} &\Rightarrow (ttu) \\
b_2. \langle t, u \rangle_{010} &\Rightarrow (tut) \\
b_3. \langle t, u \rangle_{011} &\Rightarrow (tuu) \\
c_0. \langle t, u \rangle_{111} &\Rightarrow (uuu) \\
c_1. \langle t, u \rangle_{110} &\Rightarrow (uut) \\
c_2. \langle t, u \rangle_{101} &\Rightarrow (utu) \\
c_3. \langle t, u \rangle_{100} &\Rightarrow (utt)
\end{aligned}$$

$$\begin{aligned}
&[id, red, red] (\lambda z. \langle y_{100}, x_{011} \rangle) \langle y^{(3)}, x^{(3)} \rangle \\
&\quad \Downarrow \beta a. \\
&\langle \langle y^{(3)}, x^{(3)} \rangle_{100} \langle y^{(3)}, x^{(3)} \rangle_{011} \rangle \\
&\quad \Downarrow subst \\
&\langle x^1, y^2, y^3 \rangle \langle y^1, x^2, x^3 \rangle \\
&\quad \Downarrow [id, red, red] \\
&\langle x^1, y^1, y^1 \rangle \langle y^1, x^1, x^1 \rangle \\
&\quad \Downarrow collection \\
&[\langle x^1, y^1 \rangle \langle y^1, x^1 \rangle \langle y^1, x^1 \rangle]
\end{aligned}$$

Reduction is changing the context, i.e., the contextual position of the system but not its content. The syntax remains untouched, but is dislocated. To change also the content, the reduction would involve metamorphosis, not considered at the moment.

$$\begin{aligned}
& [id, red, red] \left(\lambda z. \langle y_{100}, x_{011} \rangle \right) \langle y^{(3)}, x^{(3)} \rangle \\
& \quad \Downarrow \beta a., [id, red, red] \\
& \langle \langle y^{111}, x^{111} \rangle_{100} \langle y^{111}, x^{111} \rangle_{011} \rangle \\
& \quad \Downarrow subst \\
& \langle x^1, y^1, y^1 \rangle \langle y^1, x^1, x^1 \rangle \\
& \quad \Downarrow collection \\
& \left[\langle x^1, y^1 \rangle \langle y^1, x^1 \rangle \langle y^1, x^1 \rangle \right]
\end{aligned}$$

Pattern: [id1, red1, red1], S123 → S111,
CM??? <1,2><2,1><2,1> , <1,2><2,1><1,1>

$$\begin{aligned}
& [id, red, red] \left(\lambda z. \langle y_{101}, x_{010} \rangle \right) \langle y^{(3)}, x^{(3)} \rangle \\
& \quad \Downarrow \beta a., [id, red, red] \\
& \langle \langle y^{111}, x^{111} \rangle_{101} \langle y^{111}, x^{111} \rangle_{010} \rangle \\
& \quad \Downarrow subst \\
& \langle x^1, y^1, x^1 \rangle \langle y^1, x^1, y^1 \rangle \\
& \quad \Downarrow collection \\
& \left[\langle x^1, y^1 \rangle \langle y^1, x^1 \rangle \langle x^1, y^1 \rangle \right]
\end{aligned}$$

A more explicite notation is the following:

$$\begin{aligned}
& samba^{(3)} [id, red, red] \\
& \left[\begin{array}{l}
thematize (substitution) \\
identify contextures^{(3)} \\
\left[\begin{array}{l}
\left(\lambda z. \langle y_{100}, x_{011} \rangle \right) \langle y^{(3)}, x^{(3)} \rangle \\
\quad \Downarrow \beta a., [id, red, red] \\
\langle \langle y^{111}, x^{111} \rangle_{100} \langle y^{111}, x^{111} \rangle_{011} \rangle \\
\quad \Downarrow subst \\
\langle x^1, y^1, y^1 \rangle \langle y^1, x^1, x^1 \rangle \\
\quad \Downarrow collection \\
\left[\langle x^1, y^1 \rangle \langle y^1, x^1 \rangle \langle y^1, x^1 \rangle \right]
\end{array} \right]
\end{array} \right]
\end{aligned}$$

5.4.7 Interactional constellations

$$[id, bif_{123}, id] \langle t, u \rangle^{(3)} = \begin{bmatrix} \langle t, u \rangle^{1.1} & , \emptyset & , \emptyset \\ \langle t, u \rangle^{2.1} & , \langle t, u \rangle^{2.2} & , \langle t, u \rangle^{2.3} \\ \emptyset & , \emptyset & , \langle t, u \rangle^{3.3} \end{bmatrix}$$

$$\begin{aligned} \langle t, u \rangle^{(3)} &= [\langle t, u \rangle^1, \langle t, u \rangle^2, \langle t, u \rangle^3] \\ [id, bif, id] \langle t, u \rangle^{(3)} &= [\langle u, t \rangle^1, \langle \langle t, u \rangle^1 \langle t, u \rangle^2 \langle t, u \rangle^3 \rangle, \langle t, u \rangle^2] \\ [\langle t, u \rangle^1, \langle \langle t, u \rangle^1 \langle t, u \rangle^2 \langle t, u \rangle^3 \rangle, \langle t, u \rangle^2]_{000} &= (t^1, (t^1 t^2 t^3), t^3) \end{aligned}$$

$$\begin{aligned} pattern [id, bif, id] : S_{123} &\Rightarrow S_{1,231,3} \\ b_0. \langle t, u \rangle_{0,000,0} &\Rightarrow (t^1, (t^2 t^3 t^1), t^3) \end{aligned}$$

$$\begin{aligned} pattern [id, id, bif] : S_{123} &\Rightarrow S_{1,2,123} \\ b_0. \langle t, u \rangle_{0,0,000} &\Rightarrow (t^1, t^2, (t^1 t^2 t^3)) \end{aligned}$$

$$\begin{aligned} pattern [bif, id, id] : S_{123} &\Rightarrow S_{123,2,3} \\ b_0. \langle t, u \rangle_{000,0,0} &\Rightarrow ((t^1 t^2 t^3), t^2, t^3) \end{aligned}$$

$$\begin{aligned} pattern [id, bif, bif] : S_{123} &\Rightarrow S_{1,123,1233} \\ b_0. \langle t, u \rangle_{0,000,000} &\Rightarrow (t^1, (t^1 t^2 t^3), (t^1 t^2 t^3)) \end{aligned}$$

$$\begin{aligned} pattern [bif, bif, id] : S_{123} &\Rightarrow S_{123,2,123.1} \\ b_0. \langle t, u \rangle_{000,0000,0} &\Rightarrow ((t^1 t^2 t^3), (t^1 t^2 t^3), t^3) \end{aligned}$$

$$\begin{aligned} pattern [bif, bif, bif] : S_{123} &\Rightarrow S_{123,123,123} \\ b_0. \langle t, u \rangle_{000,000,000} &\Rightarrow ((t^1 t^2 t^3), (t^1 t^2 t^3), (t^1 t^2 t^3)) \end{aligned}$$

The same scheme for the substitution of u.

$$\begin{aligned} pattern [id, bif, id] : S_{123} &\Rightarrow S_{1,231,3} \\ b_0. <t, u>_{1,111,1} &\Rightarrow (u^1, (u^2 u^3 u^1), u^3) \end{aligned}$$

And along this line all the combinations of t and u, restricted by mediation.

$$\begin{aligned} pattern [id, bif, id] : S_{123} &\Rightarrow S_{1,231,3} \\ b_0. <t, u>_{1,000,0} &\Rightarrow (u^1, (t^2 t^3 t^1), t^3) \end{aligned}$$

5.4.8 Modi of interaction between contexture/head/body

The general pattern of interactionality can be analyzed in respect of the different modi involved in the process of interaction. The matrix itself gives only the place-holder system of interaction, also of reflection, and not how the interaction/reflection is realized.

Different modi of interaction produced by the operator of alteration can be defined with the help of the distinction contextures/heads and body. Where *lambda* <variable> represents the head and <lambda-term> the body of the formula, *lambda*<variable>. <lambda-term>.

1. *Metamorphic* interaction: Alteration of contextures at a single locus including different head/body/statements.
2. *Alterational* interaction: Alteration of heads into itself including bodies under single contextures.
3. *Replicational* interaction: Alteration of bodies into itself including statements under single heads.
4. *Iterations* of bodies in interactional situations which are ruled by a head under a single contexture are not interactions but *superpositions*. That is, at each position of the polycontextural matrix superpositions can take place.

5.4.9 Reflectional constellations

If a system is reflecting upon itself, making an inner model of itself, entering introspection and other self-referential processes, the replicated system has to be positioned, it needs a locus in the architecture of the reflecting system which is part of a societal constellation. The space for such placements is offered by the reflectional axis of the polycontextural matrix which is part of the architectonics of the poly-lambda calculus under consideration.

Two examples of reductions in reflectional constellations

$$\begin{aligned} \text{pattern } [repl_{1.1,1.2,1.3}, id_{2.2}, id_{3.3}] : S_{123} &\Rightarrow S_{1.1,1.2,1.3,2,3} \\ b_0. <t, u>_{000.1.1} &\Rightarrow (u^{1.1}u^{1.2}u^{1.3}), t^2, t^3 \end{aligned}$$

$$\begin{aligned} [repl_{1.1,1.2,1.3}, id_{2.2}, id_{3.3}] <t, u>^{(3)} &= [\langle <t, u>^{1.1} <t, u>^{1.2} <t, u>^{1.3} \rangle, <t, u>^{2.2}, <t, u>^{3.3}] \\ [\langle <t, u>^{1.1} <t, u>^{1.2} <t, u>^{1.3} \rangle, <t, u>^{2.2}, <t, u>^{3.3}]_{000.1.1} &= (u^{1.1}u^{1.2}u^{1.3}), t^2, t^3 \end{aligned}$$

$$\begin{aligned} \text{pattern } [id_{1.1}, repl_{2.1,2.2,2.3}, id_{3.3}] : S_{123} &\Rightarrow S_{1.1,2.1,2.2,3,3,3,3} \\ b_0. <t, u>_{0.1.1} &\Rightarrow (t^1, (u^{2.1}u^{2.2}u^{2.3}), t^3) \end{aligned}$$

$$\begin{aligned} [id_{1.1}, repl_{2.1,2.2,2.3}, id_{3.3}] <t, u>^{(3)} &= [<t, u>^{1.1}, \langle <t, u>^{2.1} <t, u>^{2.2} <t, u>^{2.3} \rangle, <t, u>^{3.3}] \\ [<t, u>^{1.1}, \langle <t, u>^{2.1} <t, u>^{2.2} <t, u>^{2.3} \rangle, <t, u>^{3.3}]_{0.111.1} &= (t^2, (u^{2.1}u^{2.2}u^{2.3}), t^3) \end{aligned}$$

Short form of the results of replications and reductions

$$\begin{aligned} [\langle <t, u> \rangle^{1.1,1.2,1.3}, <t, u>^{2.2}, <t, u>^{3.3}]_{000.1.1} &= (u^{1.1}u^{1.2}u^{1.3}), t^2, t^3 \\ [<t, u>^{1.1}, \langle <t, u> \rangle^{2.1,2.2,2.3}, <t, u>^{3.3}]_{0.111.1} &= (t^2, (u^{2.1}u^{2.2}u^{2.3}), t^3) \end{aligned}$$

Matrix notation of the super-operator effects

$$\begin{aligned} [repl_{1.1,1.2,1.3}, id_{2.2}, id_{3.3}] <t, u>^{(3)} &= \begin{bmatrix} <t, u>^{1.1}, & \emptyset, & \emptyset \\ <t, u>^{1.2}, <t, u>^{2.2}, & \emptyset \\ <t, u>^{1.3}, & \emptyset, & <t, u>^{3.3} \end{bmatrix} \\ [id_{1.1}, repl_{2.1,2.2,2.3}, id_{3.3}] <t, u>^{(3)} &= \begin{bmatrix} <t, u>^{1.1}, <t, u>^{2.1}, & \emptyset \\ \emptyset, & <t, u>^{2.2}, & \emptyset \\ \emptyset, & <t, u>^{2.3}, <t, u>^{3.3} \end{bmatrix} \end{aligned}$$

5.4.10 Modi of reflection

The general pattern of reflectionality can be analyzed in respect of the different modi involved in the process of reflection. The matrix itself gives only the place-holder system of reflection, also of interaction, and not how the reflection is realized.

Different modi of reflection produced by the operator of replication can be defined with the help of the distinction contextures/heads and body.

1. *Metamorphic* reflection: Replication of contextures at a single locus including different head/body/statements.
2. *Alterational* reflection: Replication of heads into itself including bodies under single contextures.
3. *Replicational* reflection: Replication of bodies into itself including statements under single heads.
4. *Iterations* of bodies in reflectional situations which are ruled by a head under a single contexture are not reflections but *superpositions*. That is, at each position of the polycontextural matrix superpositions can take place.

5.5 Connecting poly-lambda calculi to a complex formal world

5.5.1 PolyLogics

Boolean

$$\begin{aligned} & \textit{Booleans}^{(3)} \\ & \textit{true}_i \equiv K^i \\ & \textit{false}_i \equiv K^i_* \\ & \textit{true}_1 \equiv \textit{true}_3 \\ & \textit{false}_1 \approx \textit{true}_2 \\ & \textit{false}_2 \equiv \textit{false}_3 \end{aligned}$$

Junctions

Transjunctions

5.5.2 Numbers in trans-classic systems

Numbers

Some poly-arithmetics

Computation and poly-lists

There is no doubt, that all this tedious stuff has to be programmed to be properly explored and as a result, formalized again, to another step of ultra-pedantic perfectionism. But to start programming, we have to pass some rounds of the intuition-formalization merry-go-round. Also the idea of disseminated lambda calculi doesn't fit into a programming paradigm, if taken seriously and literally, there is no problem to model or simulate these patterns in a common programming language, say Scheme or ML. Some people are not aware about such differences in the interplay between conceptualization and realization, and start to worry.

$$id(i, j): \forall i, j \in s(3): (Syn_{LC}^{i,j}) \xrightarrow{id} (Syn_{LC}^{i,j}), i, j \in s(3)$$
$$\begin{array}{l}
\text{samba}^{(3)}_{\text{linear}} [id, id, id] \\
\text{thematize} (\text{parallel}, \text{numeric}, \text{arithmetics}) \\
\text{identify contexts}^{(3)} \\
\left[\begin{array}{l} ((\lambda x.x^{(3)} ++ -x^{(3)}) (\lambda y.y^{(3)} ++ -y^{(3)}) 111) \\ \textcolor{red}{/} \quad \textcolor{red}{/} \quad \textcolor{red}{/} \qquad \textcolor{red}{\backslash} \quad \textcolor{red}{\backslash} \quad \textcolor{red}{\backslash} \\ \left[\begin{array}{l} (\textcolor{red}{\lambda x.x + x})(\textcolor{red}{1 + 1}) \\ (\textcolor{green}{\lambda x.x + x})(\textcolor{green}{1 + 1}) \\ (\lambda x.x - x)(1 - 1) \end{array} \right] \left[\begin{array}{l} ((\textcolor{red}{\lambda y.y + 1}) 1) + ((\textcolor{red}{\lambda y.y + 1}) 1) \\ ((\lambda y.y + 1) 1) + ((\lambda y.y + 1) 1) \\ ((\lambda y.y - 1) 1) - ((\lambda y.y - 1) 1) \end{array} \right] \\ \textcolor{red}{\backslash} \quad \textcolor{green}{\backslash} \quad \textcolor{green}{\backslash} \qquad \textcolor{red}{/} \quad \textcolor{red}{/} \quad \textcolor{red}{/} \\ \left[\begin{array}{l} (\textcolor{red}{1 + 1}) + (\textcolor{red}{1 + 1}) \\ (\textcolor{green}{1 + 1}) + (\textcolor{green}{1 + 1}) \\ (1 - 1) - (1 - 1) \end{array} \right]
\end{array} \right]
\end{array}$$

Substitution onto itself or into another are two modi made accessible by poly-Lambda Calculi. A different story is introduced, if something of a contexture is substituted for something else, different or similar, in another contexture, say from red to green. Such *trans-contextural* operations are not in conflict with the above statement of the inacces-

sibility of, say, numbers of one contexture from another contexture by a counting alone. But it splits the homogeneous identity of isomorphism into heteromorphisms, as a result some displacements of the uniqueness of the Church-Rosser-Theorem are on the way. What we get is something like: If $\text{red}(\text{red}(X))$, $\text{green}(\text{red}(X))$, then $X\text{red}$ simil $X\text{green}$.

The two equations are representing the same arithmetical situation, but they are not identical because they are realized at two different contextual loci. They are the same, but different. Mono-contextural thinking is focussed on one and only one contextual locus and is therefore not aware of it. The equation as green and the equation as red, like a formula as a program or as a set of data, both at once, but without any self-referential conflicts, or *circuli creativi*, but distributed and mediated, embedded in a working complex scriptural patterns, an ultra-formalism.

Reductional diamonds are classically hierarchic. The reduction path is identical to the construction path. This symmetry is no longer compulsory if we introduce slippery reductions like colored reductions. A new asymmetry is possible, ruled by the different but similar substitutions. In other words, the reductions as computations are discontexturally concurrent. They are also co-operative in the special sense, that if one computation is not working, the other one can do the job, because they are doing essentially the same.

6.2 Internal vs. external super-operators

As long as the super-operators are only active externally it is easier to decompose the formulas into their sub-system parts and then apply the super-operators.

This is not working for internal super-operator activities. A good example for internal bifurcations are transjunctions. Internal permutations are internally forced by negations. Internal super-operators have to be introduced into the different topics, like poly-contextural arithmetics, lists, logics, etc.

How are external super-operators motivated? External super-operators can be motivated as global operators, ruling the local systems independently of their internal structure. Thus, local sub-systems can be permuted, reduced, bifurcated as such, even if there are no internal operators forcing this behavior.

The motivation for internal super-operators is given by the behavior of the internal operators under consideration. How should we understand the interplay between internal/external operators?

6.3 Internal vs. external super-operators

6.3.1 External super-operators [id, red1, id]

$$S^{(3)} : [id, red_1, id] \left((\lambda x.x^{(3)} + + / x^{(3)}) (\lambda y.y^{(3)} + + / y^{(3)}) 123 \right)$$

$$S^{1.1} : \begin{aligned} & ((\lambda x.x + x) (\lambda y.y + y) 1) \\ & (\lambda x.x + x) (1 + 1) \mid ((\lambda y.y + y) 1) + ((\lambda y.y + y) 1) \\ & (1 + 1) + (1 + 1) \end{aligned}$$

$$[id] : S^{1.1} \longrightarrow S^{1.1}$$

$$S^{2.2} : \begin{aligned} & ((\lambda x.x + x) (\lambda y.y + y) 2) \\ & (\lambda x.x + x) (2 + 2) \mid ((\lambda y.y + y) 2) + ((\lambda y.y + y) 2) \\ & (2 + 2) + (2 + 2) \end{aligned}$$

$$[red_1] : S^{2.2} \longrightarrow S^{1.2}$$

$$S^{3.2} : \begin{aligned} & ((\lambda x.x / x) (\lambda y.y / y) 3) \\ & (\lambda x.x / x) (3 / 3) \mid ((\lambda y.y / y) 3) / ((\lambda y.y / y) 3) \\ & (3 / 3) / (3 / 3) \end{aligned}$$

$$[id] : S^{3.3} \longrightarrow S^{3.3}$$

$$result : \begin{aligned} & [id, red_1, id] : S^{123} \longrightarrow S^{1.1.1.2.3.3} : \left[\begin{array}{lll} \langle (1 + 1) + (1 + 1) \rangle, & \langle \emptyset \rangle, & \langle \emptyset \rangle \\ \langle (2 + 2) + (2 + 2) \rangle, & \langle \emptyset \rangle, & \langle \emptyset \rangle \\ \langle \emptyset \rangle, & \langle \emptyset \rangle, & \langle (3 / 3) / (3 / 3) \rangle \end{array} \right] \end{aligned}$$

short :

$$[id, red_1, id] : S^{123} \longrightarrow S^{1.1.1.2.3.3} : \left[\langle (1 + 1) + (1 + 1) \rangle, \langle (2 + 2) + (2 + 2) \rangle, \langle (3 / 3) / (3 / 3) \rangle \right].$$

S123 → S1 | S2, S3.

The external super-operator pattern [id, red1, id] is at the start of the constellation. Thus, the the formula in sub-system S2 is moved to S1: S2→S1. This move from S2 to S1 is a reduction of the position S2 to S1. That is, S2 abandons its position in favor of position S1. But the content of the formula of sub-system S2 remains, what is changing, its location, can be interpreted as a change in relevance. The viepoint, from which the formula is stated has shifted to another viepoint. But this is not a subsumtion under the position of S1 because S2 receives its own location at the position S1 as S1.2, i.e., S2.2→S1.2. In this example of reductional behavior, the mathematical content of the formala is not considere.

6.3.2 Internal super-operators [id, bif1, id]

$$S^{(3)} : [id, id, id] \left((\lambda x.x^{(3)} + \oplus / x^{(3)}) (\lambda y.y^{(3)} + \oplus / y^{(3)}) 123 \right)$$

$$S^{1,1} : \begin{aligned} & ((\lambda x.x + x)(\lambda y.y + y)1) \\ & (\lambda x.x + x)(1+1) \mid ((\lambda y.y + 1)1) + ((\lambda y.y + 1)1) \\ & (1+1) + (1+1) \end{aligned}$$

$$[id] : S^1 \longrightarrow S^1$$

$$S^{2,1} : \begin{aligned} & ((\lambda x.x \oplus x)(\lambda y.y \oplus y)2) \\ & (\lambda x.x \oplus x)(2 \oplus 2) \mid ((\lambda y.y \oplus 2)2) \oplus ((\lambda y.y \oplus 2)2) \\ & (2 \oplus 2) \oplus (2 \oplus 2) \end{aligned}$$

$$S^{2,2} : \begin{aligned} & ((\lambda x.x \oplus x)(\lambda y.y \oplus y)2) \\ & (\lambda x.x \oplus x)(2 \oplus 2) \mid ((\lambda y.y \oplus 2)2) \oplus ((\lambda y.y \oplus 2)2) \\ & (2 \oplus 2) \oplus (2 \oplus 2) \end{aligned}$$

$$[bif_1] : S^2 \longrightarrow S^2 \parallel S^1$$

$$S^{3,3} : \begin{aligned} & ((\lambda x.x / x)(\lambda y.y / y)3) \\ & (\lambda x.x / x)(3 / 3) \mid ((\lambda y.y / 3)3) / ((\lambda y.y / 3)3) \\ & (3 / 3) / (3 / 3) \end{aligned}$$

$$[id] : S^3 \longrightarrow S^3$$

$$result :$$

$$[id, bif_1, id] : S^{123} \longrightarrow S^{1,1,2,1,2,2,,3,3} : \left[\begin{array}{ccc} \langle (1+1) + (1+1) \rangle, & \langle \emptyset \rangle, & \langle \emptyset \rangle \\ \langle (2 \oplus 2) \oplus (2 \oplus 2) \rangle, & \langle (2 \oplus 2) \oplus (2 \oplus 2) \rangle, & \langle \emptyset \rangle \\ \langle \emptyset \rangle, & \langle \emptyset \rangle, & \langle (3 / 3) / (3 / 3) \rangle \end{array} \right]$$

$S^{123} \rightarrow S^1, S^2, S^3 \rightarrow S^1, S^2 \mid S^1, S^3.$

The external super-operator pattern [id, id, id] is the start constellation. But the internal structure of the formula includes bifurcational (transjunctional) operators. Thus, the the formula in sub-system S2 splits: $S^2 \rightarrow S^2 \mid S^1$. Because we are not yet dealing with the internal structure of the formulas and their topics the notation is not giving much information, its purpose is only to show the functioning of bifurcations and not to give an interpretation about the interesting features of poly-arithmetic systems.

6.4 A permutational constellation

$$perm(i, j): \forall i, j \in s(3): (Syn_{LC}^i, Syn_{LC}^j) \xrightarrow{perm} (Syn_{LC}^j, Syn_{LC}^i)$$

6.5 A reductional constellation

$$red(i, j): \forall i, j \in s(3): (Syn_{LC}^i, Syn_{LC}^j) \xrightarrow{red} (Syn_{LC}^i, Syn_{LC}^i)$$

$$\begin{array}{l} \textit{samba}^{(3)}_{\textit{linear}} \left[\textit{id}, \textit{red}_1, \textit{id} \right] \\ \textit{thematize} \left(\textit{parallel}, \textit{numeric}, \textit{arithmetics} \right) \\ \textit{identify contextures}^{(3)} \\ \left[\left[\left(\left(\lambda \mathbf{x}. \mathbf{x}^{(3)} \right) ++ / \mathbf{x}^{(3)} \right) \left(\lambda \mathbf{y}. \mathbf{y}^{(3)} \right) ++ / \mathbf{y}^{(3)} \right) 123 \right] \\ \quad // / \quad \textit{subst}[\textit{id}, \textit{red}_1, \textit{id}] \quad \backslash \backslash \backslash \\ \left[\left\langle \left(\lambda \mathbf{x}. \mathbf{x} + \mathbf{x} \right) (1 + 1), \left(\lambda \mathbf{x}. \mathbf{x} + \mathbf{x} \right) (2 + 2) \right\rangle \right] \left[\left\langle \left(\left(\lambda \mathbf{y}. \mathbf{y} + 1 \right) 1 \right) + \left(\left(\lambda \mathbf{y}. \mathbf{y} + 1 \right) 1 \right), \left(\left(\lambda \mathbf{y}. \mathbf{y} + 2 \right) 2 \right) + \left(\left(\lambda \mathbf{y}. \mathbf{y} + 2 \right) 2 \right) \right\rangle \right] \\ \quad \left(\lambda \mathbf{x}. \mathbf{x} / \mathbf{x} \right) (3 / 3) \quad \left[\left(\left(\lambda \mathbf{y}. \mathbf{y} / 3 \right) 3 \right) - \left(\left(\lambda \mathbf{y}. \mathbf{y} / 3 \right) 3 \right) \right] \\ \quad \backslash \backslash \backslash \quad // / \\ \left[\left\langle (1 + 1) + (1 + 1), (2 + 2) + (2 + 2) \right\rangle \right] \\ \left(3 / 3 \right) / \left(3 / 3 \right) \end{array}$$

S123—>S1.1S1.2,S3.3

6.6 An interactional constellation

$$bif(i, j): \forall i, j \in s(3): (\text{Syn}_{LC}^i, \text{Syn}_{LC}^j) \xrightarrow{bif} ((\text{Syn}_{LC}^i \parallel \text{Syn}_{LC}^j), \text{Syn}_{LC}^j)$$

The same as bifurcation example.

6.7 A reflectional constellation

$$repl(i, j) : \forall i, j \in s(3) : (Syn_{LC}^i, Syn_{LC}^j) \xrightarrow{repl} ((Syn_{LC}^i | Syn_{LC}^i), Syn_{LC}^j)$$

$$S^{123} \longrightarrow S^{1.1}S^{1.2}, S^{2.2}, S^{3.3}$$

$$S^{(3)} : [repl, id, id]((\lambda x.x^{(3)} ++ / x^{(3)})(\lambda y.y^{(3)} ++ / y^{(3)})123)$$

$$S^{1.1} : \begin{aligned} & ((\lambda x.x + x)(\lambda y.y + y)1) \\ & (\lambda x.x + x)(1 + 1) | ((\lambda y.y + 1)1) + ((\lambda y.y + 1)1) \\ & (1 + 1) + (1 + 1) \end{aligned}$$

$$S^{1.2} : \begin{aligned} & ((\lambda x.x + x)(\lambda y.y + y)1) \\ & (\lambda x.x + x)(1 + 1) | ((\lambda y.y + 1)1) + ((\lambda y.y + 1)1) \\ & (1 + 1) + (1 + 1) \end{aligned}$$

$$[repl] : S^{1.1} \longrightarrow S^{1.1}S^{1.2}$$

$$S^{2.2} : \begin{aligned} & ((\lambda x.x + x)(\lambda y.y + y)2) \\ & (\lambda x.x + x)(2 + 2) | ((\lambda y.y + 2)2) + ((\lambda y.y + 2)2) \\ & (2 + 2) + (2 + 2) \end{aligned}$$

$$[id] : S^{2.2} \longrightarrow S^{2.2}$$

$$S^{3.3} : \begin{aligned} & ((\lambda x.x / x)(\lambda y.y / y)3) \\ & (\lambda x.x / x)(3 / 3) | ((\lambda y.y / 3)3) / ((\lambda y.y / 3)3) \\ & (3 / 3) / (3 / 3) \end{aligned}$$

$$[id] : S^{3.3} \longrightarrow S^{3.3}$$

result :

$$[repl, id, id] : S^{123} \longrightarrow S^{1.1.1.2.2.2.3.3} : \left[\begin{array}{lll} \langle (1+1) + (1+1) \rangle, & \langle \emptyset \rangle, & \langle \emptyset \rangle \\ \langle (1+1) + (1+1) \rangle, & \langle (2+2) + (2+2) \rangle, & \langle \emptyset \rangle \\ \langle \emptyset \rangle, & \langle \emptyset \rangle, & \langle (3/3) / (3/3) \rangle \end{array} \right]$$

short :

$$[repl, id, id] : S^{123} \longrightarrow S^{1.1.1.2.2.2.3.3} :$$

$$\left[\langle (1+1) + (1+1) \rangle, \langle (1+1) + (1+1) \rangle, \langle (2+2) + (2+2) \rangle, \langle (3/3) / (3/3) \rangle \right].$$

7 More to Bore: From Y to Why

7.1 Remembering

FIXEDPOINT THEOREM

(i) $\forall F \exists X FX = X$.

(ii) *There is a fixed point combinator*

$$Y \equiv \lambda f. (\lambda x. f(x x)) (\lambda x. f(x x))$$

such that

$$\forall F F(Y F) = Y F.$$

„Etymologically speaking, correct opinion is orthodox; paradox, however, lies beyond opinion. Unfortunately, orthodox attempts to establish the orthodoxy of the orthodox results in paradox, and, conversely, the appearance of paradox within the orthodox puts an end to the orthodoxy of the orthodox. In other words, paradox is the apostle of sedition in the kingdom of the orthodox.“

„As long as was possible, logical ortho-

doxy attempted to treat such seditious intrusions just as would any other orthodoxy, that is, to dismiss them as cranks, as (syntactic) pathologies, (semantic) freaks, in short, as aberrations (of thought).“ R. H. Howe, H. von Foerster, *Introductory Comments to Francisco Varela's Calculus for Self-Reference*, Int. J. General Systems, 1975, Vol. 2, p. 1.

Proof. (i) Define $W \equiv \lambda x. F(x x)$ and $X \equiv W W$. Then

$$X \equiv W W \equiv \lambda x. F(x x) W \equiv F(W W) \equiv F X.$$

(ii) *By the proof of (i). (Barendregt)*

To define X with WW means an iteration of W to WW. This is, naturally, a innocent operation as

long as we restrict ourselves to strictly identical systems, i.e., to non-ambiguous situations. Iteration, repetition, recurrence in a complex domain is always involved with the chance of change. Iteration as selection or iteration as election. In other words, iteration as iteration and at once as alteration. The same, realized in a different contexture, is different.

In complex situations, like polycontextural, the reference in a self-referential process can slip to another, not diverse but similar domain without getting inconsistent to the definitions—nor lost in the abyss of the unknown. Thinking on slippery grounds is supported by nets of polycontextural constructions.

$$\begin{aligned} Y &= (\lambda f. (\lambda x. f(x x)) (\lambda x. f(x x))) \\ YF &= (\lambda f. (\lambda x. f(x x)) (\lambda x. f(x x))) F \\ &\Rightarrow (\lambda x. F(x x)) (\lambda x. F(x x)) \\ &\Rightarrow F(\lambda x. F(x x)) (\lambda x. F(x x)) \\ &\Rightarrow F(Y F). \end{aligned}$$

In YF, the term Y is an operator and F is an operand of the application YF. Because of the highly abstract definition of the Lambda Calculus it is possible to change the operand, step by step, to an operator. Now, F is an operator to (YF) and also an operand to Y in (YF). This double-functionality of Y is saved in the mind of the reader, the difference is nullified by notional abstraction.

Polycontextural strategy tries, in contrast and additionally, to inscribe such a notational abstraction into a graphematic play. Because there is no trust in mental representations we have to write it down.

The iteration of "*lambda x.f(x x)*" in the definition of Y can easily, without violating the rules, turn into a metamorphic alteration in polycontextural situations. Because all objects, that is, all terms in polycontextural systems are, from the very beginning, complexions, —even atomic terms are complexions—, the very beginning of the machinery can be involved into metamorphosis.

7.2 Distributed Y-Operators

A strictly parallel distribution of the Y-operator over different contextures is a first step to play with self-referential operators in poly-Lambda Calculi. In this case, Y-operators operate strictly isolated to each other, without any interaction and reflection, and not sharing any common terms. Nevertheless they are involved in their common framework of polycontextuality. The sets of their terms are not only disjunct but discontextual.

3 – FIXEDPOINT THEOREM

(i) $\forall \forall \forall F^{(3)} \exists \exists \exists X^{(3)} F^{(3)} X^{(3)} = X^{(3)}$.

(ii) *There is a fixed point combinator*

$$Y^{(3)} \equiv \lambda f^{(3)}. (\lambda x. f^{(3)}(x x)) (\lambda x. f^{(3)}(x x))$$

such that

$$\forall \forall \forall F^{(3)} F^{(3)} (Y^{(3)} F^{(3)}) = Y^{(3)} F^{(3)}.$$

Proof. (i)

Define $W^{(3)} \equiv \lambda x. F^{(3)}(x x)$ and $X^{(3)} \equiv W^{(3)} W^{(3)}$.

Then

$$\begin{aligned} X^{(3)} &\equiv W^{(3)} W^{(3)} \equiv \lambda^{(3)} x. F^{(3)}(x x) W^{(3)} \\ &\equiv \lambda^{(3)} x. (F^{(3)}(x x)) (\lambda^{(3)} x. F^{(3)}(x x)) \\ &\equiv \lambda^{(3)} x. F^{(3)}(x x) W^{(3)} \equiv F^{(3)}(W^{(3)} W^{(3)}) \\ &\equiv F^{(3)} X^{(3)}. \\ Y^{(3)} &= (\lambda f^{(3)}. (\lambda x. f^{(3)}(x x)) (\lambda x. f^{(3)}(x x))) \\ Y^{(3)} F^{(3)} &= (\lambda f^{(3)}. (\lambda x. f^{(3)}(x x)) (\lambda x. f^{(3)}(x x))) F^{(3)} \\ &\Rightarrow (\lambda x. F^{(3)}(x x)) (\lambda x. F^{(3)}(x x)) \\ &\Rightarrow F^{(3)} (\lambda x. F^{(3)}(x x)) (\lambda x. F^{(3)}(x x)) \\ &\Rightarrow F^{(3)} (Y^{(3)} F^{(3)}) \end{aligned}$$

This doesn't sound specially interesting, but guaranties autonomous recursions and recursive functions in each contexture.

As a result, we can deal with a kind of parallelism unknown in existing implementations, the *architectonic parallelism*. Architectonic parallelism is also introducing a new form of graph reduction preserving at each locus the known methods of graph reduction. Some first applications of both, the 3-contextural combinator $Y^{(3)}$ and the architectonic parallelism, applied to an implementation of the Fibonacci numbers, *define FIB* (YYY) $FIB^{(3)}$, in contrast to other parallel implementations, is proposed at:

<http://www.thinkartlab.com/research.html>

```
samba(3) ( id, id, id )    ;;< architectonics : id – super – operator, complexity = 3 >
[ style functional
  [ topics numeric        ;;< fibi : inti --> inti, i = 1, 2, 3 >
    [ thematize FIB(3)    ;;< scenario : FIB(3) = ( fib1, fib2, fib3 )
      [ identify contexture1
        [ define fib1
          [ lambda ( n )
            [ if ( equaln zero )
              [ one
                [ ( fib1 ( sub n one ) ) ]
              [ ( elect3 ) ]
            ]
          ]
        ]
      [ identify contexture2
        [ define fib2
          [ lambda ( n )
            [ if ( equaln zero )
              [ one
                [ ( fib2 ( sub n two ) ) ]
              [ ( elect3 ) ]
            ]
          ]
        ]
      [ identify contexture3
        [ define fib3
          [ lambda ( n )
            [ lambda ( add )
              [ if ( equaln zero )
                [ one
                  [ ( add fib1 fib2 ) ]
                [ ( elect1 elect2 ) ]
              ]
            ]
          ]
        ]
      ]
    ]
  ]
]
```

According to the analogous principles of sameness as introduced by the *as-abstractions*, substitution of the same has not to be necessarily substitution of the identical term.

We also don't have to accept the general formalistic trick as the only way of thinking, namely, that a part is nothing else than the whole and the whole is not worth more than a part because they are both simply terms in an abstract general calculus and nothing more.

$$\begin{array}{l} \text{thematize } (\text{Why}^1 \emptyset \emptyset) \\ \left[\text{lambda } (f^{1.1} f^{1.2} f^{1.3}) \right] \\ \left[\text{identify contexture}^1 \right] \\ \left[\text{define Why}^1 \right] \\ \left[\text{lambda } (f^{1.1}) \right] \\ \left(\begin{array}{l} \text{lambda } (x) \\ (f (x \ x)) \end{array} \right) \\ \left[\text{lambda } (f^{1.2}) \right] \\ \left(\begin{array}{l} \text{lambda } (x) \\ (f (x \ x)) \end{array} \right) \\ \left[\text{lambda } (f^{1.3}) \right] \\ \left(\begin{array}{l} \text{lambda } (x) \\ (f (x \ x)) \end{array} \right) \end{array}$$

A shift (slip) in the reference.
From identity to samenes.

(ii) *There is a fixed point combinator*

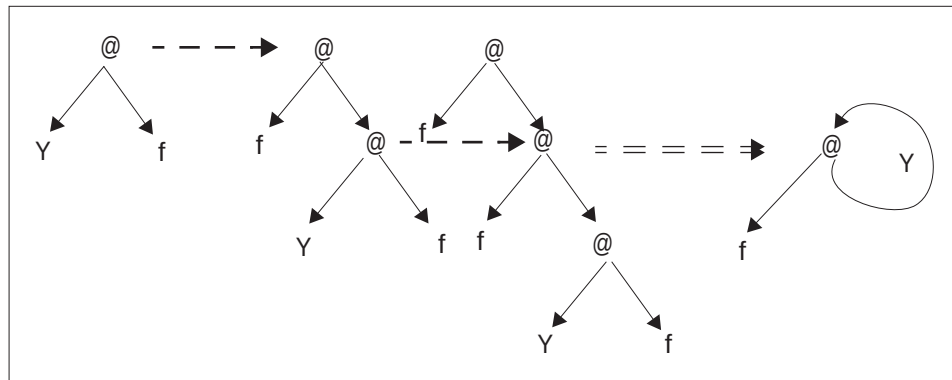
such that

$$\mathbb{V}^{(3)} F^{(3)} F^{(3)} (WHY F^{(3)}) = WHY F^{(3)}.$$

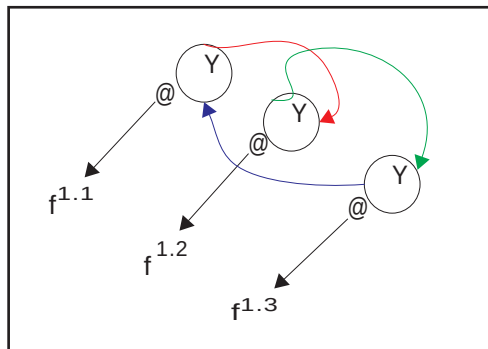
© Rudolf Kaehr

7.4 Graph diagrams for Y and WHY

"The Y combinator can be implemented by using the reduction sequence $Y(f)=f(Y)f$ i.e. as follows:



But, as a copy of the original graph appears a sub-graph of sequent graphs in the (infinite) reduction sequence, we can tie the knot by making the sub-graph coincide with the graph itself." A J T Davie, p. 160



"the sub-graph coincide with the graph itself" this is a highly complex and paradox statement. The sub-graph "coincides" with the "graph itself", thus it is not identical. Because the "graph itself" can also be understood as the same and analogues but not identical graph, belonging therefore at once to different contextures, reduction and substitution can be distributed over different loci. The diagram shows, by its indices, a reflectional constellation. That is, iteration is modeled as replication.

With other indices it could also be an interactional or mixed constellation. The process of iteration, i.e., the repetition of the identical term, slips to another contextures, repeating itself as another but analogues term. To speak or to write about the mechanism of a term changing its contexture *itself* needs an own contexture to give space for the abstraction mentioning, identifying, a term moving from one to another contextual realization.

To calm the tempers, again, at each locus the classic construction holds. No chances have disappeared. Depending on the structure of the connection between different loops a typology of new fixed-point operators, transcontextual fix-points, can be established.

From the aspects of circularity involved in the distributed Y-operators two circles are now in the game, the classic internal circle, repeated at different loci, and a queer, i.e., orthogonal/transversal circle between the distributed Y-operators.

Re-entry of/into identity or of/into sameness—that makes the difference.

A lot of work has been published in the past about non-vicious logical circularity, some are collected at: <http://www.thinkartlab.com/pkl/media/dissemin-final.pdf>

7.5 Iterability in Y- and WHY-operators

7.5.1 Mono-contextural combinators

Collection of the essentials for the journey

Combinators	Proof of $Y f = f(Y f)$:	
$I \equiv \lambda x.x$	$Y f = WS(BWB) f$	This is the classic proof of the fixed point theorem in combinatory logic using the operator Y.
$K \equiv \lambda xy.x$	$= S(BWB) ff$	
$K_* \equiv \lambda xy.y$	$= BWB f (BWB f)$	The exercise in self-referentiality is more direct to the point in combinatory logic terms than in the lambda calculus version.
$S \equiv \lambda xyz.xz(yz)$	$= W(B f)(BWB f)$	
$W \equiv \lambda fx.fxx$	$= B f(BWB f)(BWB f)$	
$B \equiv \lambda fgx.f(gx)$	$= f(Y f)$	
$Y \equiv WS(BWB)$		

$$\begin{aligned}
 W^i &\equiv \lambda^i fx.fxx \\
 Q^i &\equiv \lambda^{i+1} fx.fx \\
 W^i f^i x &\equiv f^i xx, \quad \text{iteration} \\
 Q^i f^i x &\equiv f^{i+1} x, \quad \text{accretion}
 \end{aligned}$$

As in the lambda formulation everything depends on the modi of repetition, i.e., the kinds of iterability. This crucial functionality of W doesn't mean that the operator W is prior to the other operators S, B and defining them. The importance is defined by the structure of iterability involved in Y and WHY. And this is realized by W. The operator W is a *Wiederholungs*-operator. The German term *Wiederholung*

appears in philosophy as a twofold term: *Wiederholung des Alten* and *Wiederholung des Neuen* (Kierkegaard, Weber). *Schöpfung als Wiederholung* (Günther, Gehlen). Repetition of the old and repetition of the new. Or in other words: repetition as iteration and repetition as accretion. The German term *Wiederholung* is incorporating at once the difference of iter/alter in iterability. Thus, repetition as iteration is only one side of the coin. Additional to the intra-contextural operator of repetition W I have to introduce a trans-contextural operator Q which shifts formulas from one contexture to another.

Jumps between gaps: From W to Q

(W mult) => mult (mult). This correspondence transforms easily to:
 (Q mult) => multⁱ(mult^j), $i \neq j$ in a polycontextural situation.

Again, there is no unambiguous definition which forces to refer to one and only one domain of application in applying the combinator W to a referenced term, say *mult*. There is an *interpretational gap* between the abstract combinators and the concrete references of the operators. As long as we *believe* in one and only one domain of reference of an application both are coinciding, the uniqueness of the calculus and the uniqueness of the domain.

Because combinators are applying on themselves the same argument as for the interpretational gap has to be repeated in respect of the operator/operand difference, which constitutes a *dis-contextural gap*.

The following formula stuff (§ 7.5.1–7.5.4) is material from manuscripts dating back to the 70th and 80th. Maybe in the meantime, they matured in the filing cabinet. Another approach has been introduced at: http://www.thinkartlab.com/pkl/lola/Godel_Games.pdf

7.5.2 Paradoxical constructions: Between Russian Dolls and Uroboros

These are topics from the 70th, they can be studied in full at the right place.

$$\begin{array}{ccc}
 S_1 : part_1 & \longrightarrow & whole_1 \\
 \parallel & & | : duplication \\
 S_2 : & & part_2 \longrightarrow whole_2 \\
 \parallel & & \parallel \\
 S_3 : part_3 & \xrightarrow{\text{mediation}} & whole_3
 \end{array}$$

$$\begin{array}{ccc}
 \forall t_1 \exists G_1 : t_1 & \longrightarrow & G_1 \\
 | & & \\
 \forall t_2 \exists G_2 : & & t_2 \longrightarrow G_2
 \end{array}$$

$$PR(\alpha) = [\exists, \forall, F, f]$$

Proemiality between :

1. quantors : $\exists$ and $\forall$
2. functor F , argument f



In the metaphor of the graph-reduction for Y , the part/whole relation comes as "sub-graph" and "graph itself" and the mystery lies in the magic formula "we can tie the knot by making the sub-graph coincide". A possible resolution of such magics is tried by another magical device, the proemial relation, here, between parts and wholes distributed and mediated over different loci embedded in the horizon of sameness.

Nobody should feel stupid if an understanding does not happen to conceive that the biggest Russian doll is part of the smallest Russian doll, and at once, that the smallest Russian doll is encapsulating the biggest Russian doll. In computer science we have not only to accept that but we have to live with and from it. Why not change to chiasms?

Substitution in the mode of identity

1. $\exists F \forall f : F(f) \equiv \neg^1 f(f) : \text{postulation}$
2. $\forall f : F_0(f) \equiv \neg^1 f(f) : \text{specification}$
3. $f \equiv F_0 : \forall f : F_0(f) \equiv \neg^1 f(f) : \text{substitution } W$
4. $: F_0(F_0) \equiv \neg^1 F_0(F_0) : \text{contradiction}$

Strategies of avoiding this kind of contradiction had traditionally been:

1. hierarchical type theories like Russell's to avoid the contradiction,
2. Acceptance of contradiction and splitting the formal system into
 - a) meaningful semantic propositions and
 - b) meaningless abstract objects "obs" (Curry).

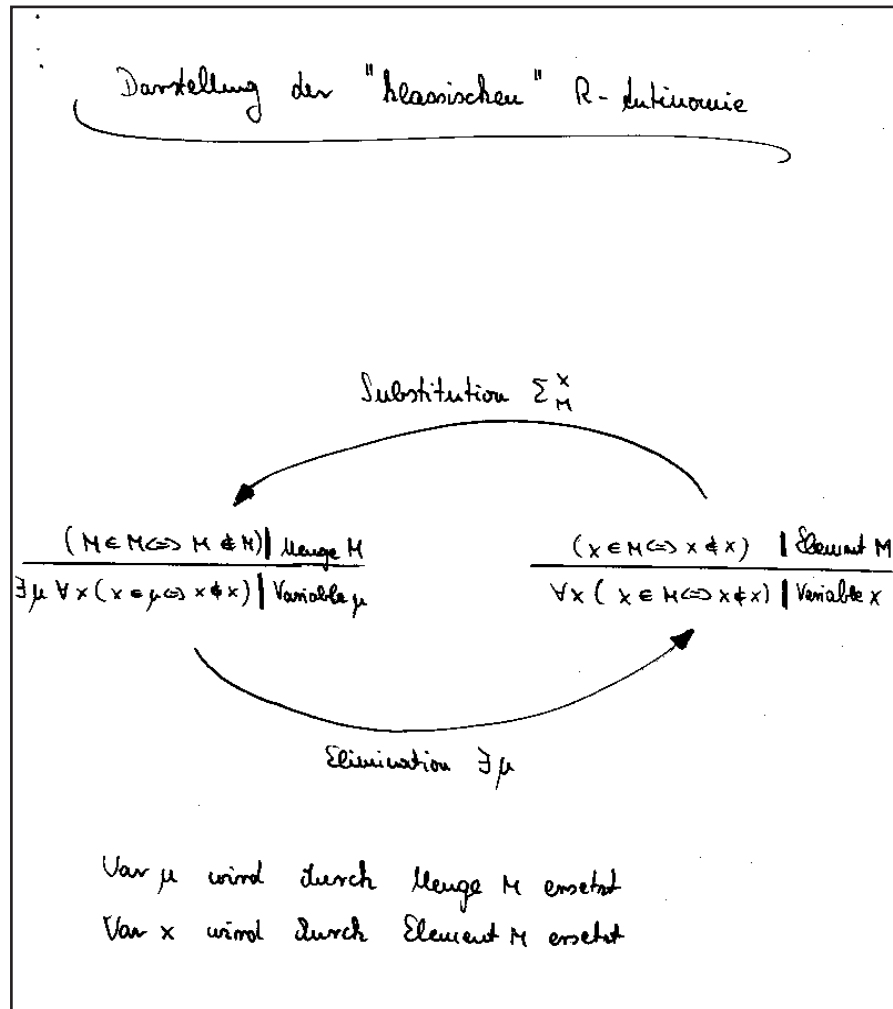
The new strategy obviously, is dissemination, ruled by proemiality.

7.5.3 Some other pictures around paradoxes

non ExProof(x,g,g). Such a proof, where a two-variable predicate is given the same value for both its arguments, is called a proof by diagonalization, and it crops up frequently in the theory of infinite sets and mathematical logic.

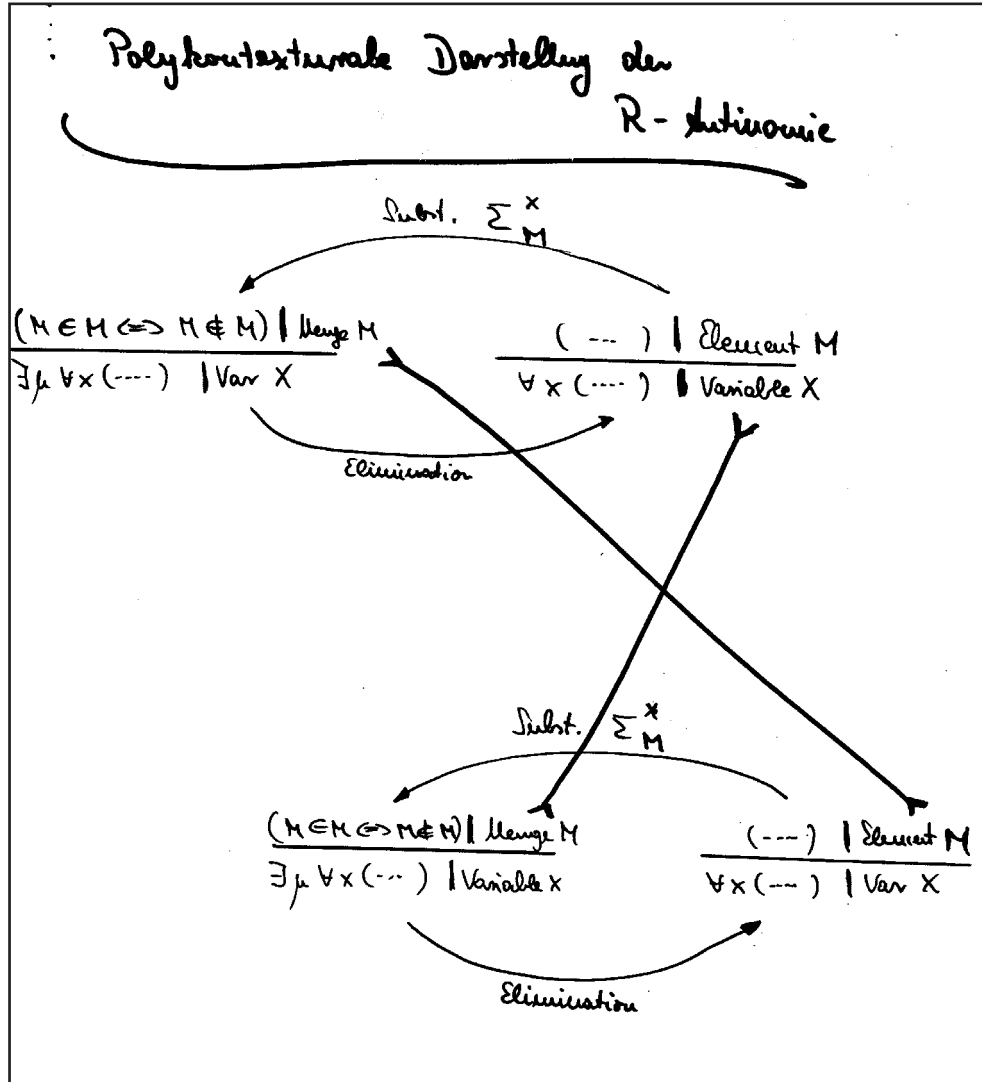
A.K. Dewdney, The New Turing Omnibus, A5, p. 35, 2001

Presentation of the "classic" R-antinomy



These two diagrams goes back to a very inspirational seminar with Prof. Wolfgang Nielg, Informatik, Universität der Bundeswehr München, 1988.

<http://www.thinkartlab.com/pkl/media/SKIZZE-0.9.5-Prop.book.pdf>



Substitution in the mode of sameness

1. $\exists^{(3)} F^{(3)} \forall^{(3)} f^{(3)} : F^{(3)}(f^{(3)}) \equiv \neg^1 f^{(3)}(f^{(3)}) : \text{postulation of constellation}$
2. $\forall^1 f^1 : F_0^1(f^1) \equiv \neg^1 f^1(f^1) : \text{specification + contextural selection}$
3. $f^2 \equiv F_0^1 : \forall^2 f^2 : F_0^2(f^2) \equiv \neg^1 f^2(f^2) : \text{accretive substitution } Q$
4. $: F_0^2(F_0^1) \equiv \neg^1 F_0^1(F_0^1) : \text{comparision : non - contradiction}$

Cf. Kaehr, Materialien 1973-75, in: Günther, Grundriss, 1978, p. 57

7.5.4 Modeling self-referentiality in poly-combinatory logics

Some more constructions from the filing cabinet.

Bracket modeling of Fitch's construction

$$\left[\begin{array}{l} W(BN)(W(BN)) : hyp \\ \left[\begin{array}{l} BN(W(BN))(W(BN)) : W\ elim \\ N(W(BN)(W(BN))) : B\ elim \end{array} \right] \\ W(BN)(W(BN)) = N(W(BN)(W(BN))) : ext \end{array} \right]$$

$$\begin{aligned} F_0^1 &\equiv W^1(B^1 \neg^1) \equiv B^1 W^1 B^1 \neg^1 \\ F_0^2 &\equiv W^2(B^2 \neg^1) \equiv B^2 W^2 B^2 \neg^1, f^2 \equiv Q^1(F_0^1) \\ F^2 f^2 &\equiv W^2(B^2 \neg^1) f^2 \geq W^2(B^2 \neg^1) F^1 \\ &\geq W^2(B^2 \neg^1)(W^1(B^1 \neg^1)) \\ &\geq B^2 W^2 B^2 \neg^1(B^1 W^1 B^1 \neg^1) \\ &\geq S^{2,1}(B^2 W^2 B^2)(B^1 W^1 B^1) \neg^1. \end{aligned}$$

$$SR^{(3)} \equiv W^{2,1} S^{2,1}(B^{(3)} W^{(3)} B^{(3)})$$

Theorem :

$$\forall^{(3)} X^{(3)} \exists^{(3)} Y^{(3)} : SR^{(3)} X^{(3)} \geq (Y^1 Y^2) \geq X(Y^1 Y^2)$$

Modeling along Fitch

" $W(BN)$ is a member of $W(BN)$ "

$$\alpha) W_{\text{iterative}} : W(BN)(W(BN))$$

$$\begin{aligned} \beta) W_{\text{accretive}} : Q(B^{(3)} N_1)(W^1(B^1 N_1)) \\ \leq B^{(3)} N_1 (W^1(B^1 N_1))(W^2 B^2 N_1) \\ \leq N_1 (W^1(B^1 N_1))(W^2(B^2 N_1)) \\ \leq N_1 (W^1 B^1 N_1)(W^2 B^2 N_1) \\ \leq N_1 (W^{1,2} S^{1,2}(B^{(3)} W^{(3)} B^{(3)})) \end{aligned}$$

$\alpha) \Rightarrow \text{contradiction}$

$\beta) \Rightarrow \text{distribution}$

Again, the construction of Fitch is producing a contradiction. But in his case only if the axiom of the excluded middle (TND= X or non X) for ppositions is added to the combinatory system.

Futurism from the 50th

"It follows from this [fixed point] theorem that Y can be used to construct obs of a more or less paradoxical nature. Given any X , YX is an ob which is unchanged by X . Thus YN , where N is negation, is the FF of the Russell paradox; YK is a combinator which cancels infinitely many variables; [...]. For such purposes Y promises to be useful in the future. The famous argument of Gödel may, evidently, be thought of as an application of Y .

The obs of YX all have the property that they cannot be reduced to a form which is not further reducible, i.e., which does not contain a component which can form the left side of an instance of a reduction rule." Curry/Feys, p.178

This polycontextural construction is producing no contradiction even if TNDs are added to the singular contextural combinatory systems. Instead of a possible contradiction we get an incomparability of a certain distance, here: distance=1.

7.5.5 Visualizing the metaphor of interweaving formula developments

$$\left[\begin{array}{l} \text{thematize self – referentiality} \\ \text{identify contextures}^{(m)} \\ \left[\begin{array}{l} W(BN)(W(BN)); W = Q \\ BN(W(BN))(W(BN)) \\ N(W(BN)(W(BN))) \\ N(W(BN)(W(BN))) \end{array} \right] \left[\begin{array}{l} W(BN)(W(BN)) \\ BN(W(BN))(W(BN)) \\ N(W(BN)(W(BN))) \\ N(W(BN)(W(BN))) \end{array} \right] \left[\begin{array}{l} W(BN)(W(BN)) \\ BN(W(BN))(W(BN)) \\ N(W(BN)(W(BN))) \\ N(W(BN)(W(BN))) \end{array} \right] \end{array} \right]$$

At each locus we may have a formula development involved in proving the classic self-referential construction of the Y-operator. But this distributed developments can also be involved into a transversal movement constructing a formulation of the Y-operator by crossing the borders of the contextures in use. This queer movement of crossing the borders transcontexturally is shown below. The equation could be placed at an own locus.

$$\left[\begin{array}{l} \text{thematize distributed self – referentiality} \\ \text{identify contextures}^{(m)} \\ \left[\begin{array}{l} Q(BN)(W(BN)) \\ BN(W(BN))(W(BN)) \\ \emptyset \\ \emptyset \\ \text{elect contexture2} \end{array} \right] \left[\begin{array}{l} \emptyset \\ BN(W(BN))(W(BN)) \\ \emptyset \\ \emptyset \\ \text{elect contexture3} \end{array} \right] \left[\begin{array}{l} \emptyset \\ \emptyset \\ \emptyset \\ N(W(BN)(W(BN))) \\ N(W(BN)(W(BN))) = \\ Q(BN)(W(BN)) \end{array} \right] \end{array} \right]$$

Towards poly-contextural Y-operators

$$Y = W^i S(BW^j B), \text{ mono – } Y \text{ for } i = j = 1$$

$$Y^{(m)} = W^{(m)} S^{(m)} (B^{(m)} W^{(m)} B^{(m)}), \text{ poly – } Y^{(m)}$$

$$\forall t \in s(m) : Y_t^{(m)} = W^t S^{(m)} (B^{(m)} W^t B^{(m)}), \text{ intra – } Y_t^{(m)}$$

$$WHY^{(m)} = W^i S^{(m)} (B^{(m)} W^j B^{(m)}), \text{ trans – } Y, i \neq j \in s(m)$$

A little catalogue of a mix of operators for self-referential constellations in formal systems.

Obviously, all these highly interesting polycontextural designs and modeling of self-referentiality, introducing poly-Ys and **WHY**-operators, are only a very first step into an unknown world far away from all known second-order circular magicks.

7.6 Combinators in the general context of iterability

7.6.1 Interdefinability among combinators (Curry/Feys)

$Ix = x$	$Y = WS(BWB)$	$I = WK$
$Kxy = x$	$W = CSI$	$I = SKK = SKS$
$Sxyz = xz(yz)$	$S = B(B(BW)C)(BB)$	$W = SS(KI)$
$Wx = xx$	$S = B(B(B(CSI))C)(BB)$	$B = S(KS)K$
$Cfxy = fyx$	$W = C(B(B(BW)C)(BB))I$	$C = S(BBS)(KK)$
$Bfgx = f(gx)$	$Y = (CSI)S(B(CSI)B)$	

An extensive table of SKI-defined operators as Mockingbirds is collected in the bird-cage *Combinator Birds* –They may sing but they don't mate.– at:
<http://www.angelfire.com/tx4/cus/combinator/birds.html>

Others, –with tickets only–, at *Combinatorial Ornithology*:
<http://library.wolfram.com/infocenter/MathSource/4862/>

7.6.2 Types of iterability in SKI

Combinators can be considered as operators of iterability. In this sense we can introduce different strength or modi of iterability realized by different combinators.

Operator **I** is iterating itself as an *identification* in a constellation (formula).
 Operator **K** is iterating the *one* ob as a *confirmation* and *eleminating* all *other* obs.
 Operator **S** is iterating the obs as a *distribution* of it in a constellation,
 Operator **W** is iterating an ob as a *duplication* of the ob in a constellation,
 Operator **C** and **B** are iterating the obs for *reorganization* in a constellation,
 Operator **Y** is iterating the constellation of **Y** itself as a *self-iteration*.

Names of operators:

After Curry (Menne): **I** is an elementary *identificator*, **C** a *permutator*, **W** a *duplicator* (*repetitor*), **B** a *compositor*, **K** a *cancellator* (*eleminator*) and **S** is a distributor.

After Smullyan, **I** is the bird who loves (identifies) all birds, himself included.

$Y = WS(BWB)$

| | |

2 – 1 – 0 – iter

infinite iteration of Y

Depending on the number of repeated obs iterability can have different strength. The operators **B** and **C** are of strength 0, the operator **W** of strength 1, and the operator **S** is of strength 2.

The infinit iterability of the **Y** operator can be defined on the base of a 1-2-0-1-0-iterability operator. These levels of strength may be compared with other orders and with Smul-

lyans ranks of combinatory birds.

In a **SKI**-system of combinatory logic the operator **S** is guarantying the system full iterability. The distribution of obs by **S** is possible only with the implication of iteration as repetition. Because **S** is fundamental to the **SKI**-combinatory system, the type of iterability in combinatory logic at all is *repetition*.

This becomes quite obvious if combinatory logic is connected to recursive number theory. But the iterability as repetition (iteration) in combinatory logic is not restricted by the exclusion of circular or self-referential constructions. For that, the **Y** operator and its equivalents are proof for unrestricted self-referentiality in combinatory logic. This is extending elementary repetition to recursion and other types of iteration. But, because **Y**-operators are based on **SKI**-operators, esp. the combinator **S**, they are realized as repetition only. Repetition only means iterability in the modus of identity, excluding all traits of accretive repeatability or altering disreption. That is, iterability is restricted to the *ITER*, excluding the *ALTER* of the poly-notion *iter/alter-ability*. This decision for identical iterability guarantees strict dis-ambiguity of formal systems. The challenge to introduce the non-concept of iter/alterability is the basic decision to start computation from the very beginning with complex writing and introducing the game of ambiguous calculations.

It seems that the very nature of combinatory logic (and lambda calculus) consists in a complete explication and formalization of the notion of iterability in the modus of non-ambiguous identity. It therefore doesn't come as a surprise that the combinatory and lambda calculus are the ultimate and natural formal foundations of computer programming.

The intuitive notions of algorithm and computability are codified by combinatory logic and, in the same sense, by the lambda calculus.

Iterability in Algorithms

Following A.A. Markov an explanation of the intuitive notion of algorithm has to cover some criteria of iterability:

Abstraction of identification, abstraction of potential realizability

„5. Elementary signs are signs that we shall consider as not having parts. The content of this concept depends upon the conventions that are assumed. (...)

6. In simultaneous consideration of any two elementary signs, we determine whether they are the same or different. These concepts are also conditional.

abstraction of identification

7. The possibility of determining when two elementary signs are the same permits us, applying an abstraction of identification, to speak of two identical elementary signs or of one and the same elementary sign. On this basis, we introduce the concept of an abstract elementary sign, that is, of an elementary sign, considered up to identity.

Concrete elementary signs will be considered as representatives of the corresponding abstract elementary sign. Two concrete elementary signs represent one and the same abstract elementary sign if and only if they are identical.

8. Lists of elementary signs are called alphabets. We shall call two alphabets equal if every elementary sign appearing in the first alphabet is identical with a certain elementary sign appearing in the second alphabet, and conversely. Alphabets considered up to equality will be called abstract alphabets.“

potential realizability

„11. Another abstraction, (...), is abstraction of potential realizability. This consists in departing from real limits of our constructive possibilities and beginning to discuss arbitrarily long abstract words as if they were constructible. Their realizability is potential: their representatives could be practically realized if we had at our disposal sufficient time, space, and materials.“

A. A. Markov

Especially the principle of potential realizability of the process of repetition is of importance for the study of the realization of iterability by combinatory logic. It is the principle which is responsible for serious problems if thinking as use of signs is connected with technology: the generous use of unlimited time, space and material.

My studies are not too much concerned about the different types of infiniteness in the concept of iterability but much more with the simple question of the beginning and the plurality of beginnings of whatever infiniteness.

Iter/alter-ability in poly-algorithms

Thus, combinatory logics in polycontextual situations (as well as lambda calculi in polycontextual situations) have to implement the non-notion of iter/alter-ability at the very base of its constructions including all types of operators and introducing additionally the reflectional and interactional operators of the whole system complex.

7.6.3 Y-operator and implicit circularity in combinator definitions

S implies **W** and **W** implies **S**.

"The definition of **I** in terms of **W** and **K** may be obtained thus

$x < \mathbf{K}xx < \mathbf{W}Kx$

so that we have the definition

I = **WK**." Curry, p. 157

Identification is defined as elimination of duplication.

Thus, identification is iteration of itself.

A sketch of a poly-combinatorial system can be found, soon, at:
http://www.thinkartlab.com/pkl/lola/poly-combinatory_logics.pdf

7.6.4 Identification vs. thematization

Now we may be prepared to introduce polycontextural strategies at the very beginning of our calculus, combinatory as well as lambda:

$Ix=x$, identity is often excluded from the calculus, because it is obvious and it can be defined by **S** and **K**. (But this is the same trick as to define the unary negation in logic with the binary Sheffer Stroke, which surely implies negation.)

Because of the complexity of identification in polycontextural systems, the operator **I** deserves its own arena of presentation.

Ix means, identification of x as x , thus $Ix=x$.

Therefore, identification is a special case of thematization. Identification is thematization of something as something and not as something similar or different.

Identification in poly-combinatorial systems is involved in *elective* decisions, and has to decide as what something is identified. *Elective* decisions are decisions between contextures, *selectiv* decisions are decisions made inside of contextures.

Identification of something as something or something else. Identification as what? A step further has to take account of the question "Identification by whom?" because polycontextural systems are societal systems, involving a multitude of acting agents. Classic calculus is "subjectless". It doesn't matter who, where, when ect. the operations are operated. Therefore, in polycontextural constellations, the operator identification **I** is realized in different modi, from the identical $I^i x^i = x^i$ for all sub-systems S^i to the different *transversal* identifiers:

$$I^i x^{(m)} = x^i.$$

Thematization as interpretation and/or thematization as identification. Identification, again is, "*giving something a name*", that is, identification is abstraction, abstracting identity, an identical property, out of complexity and diversity. Abstraction as identification is the sense of and behind the lambda calculus. To identify is to iterate the same as the identical. And this kind of identification determines the kind of iterability of the operations.

What is *abstraction* for the lambda calculus is *identification* for combinatory logic. And both are, in an abstract sense, equivalent. At least isomorphic. *Thematization* is (the working title) for polycontextural calculi or formal games in general. Another game starts with the process of *morphic* abstraction and subversion of morphogrammatcs.

Thus, the meta-language identification or identifier *Ident* is realizing itself as different kinds of specific identifiers I^i .

$I^i x^{(m)} = x^i$, means the complexion $x^{(m)}$ identified as x^i .

Or: x^i identified as a part of $x^{(m)}$.

$I^{i..j} x^{(m)} = x^{i..j}$, means the complexion $x^{(m)}$ identified as sub-complexion $x^{i..j}$.

With involvement of the super-operators [id, perm, red, repl, bif] a more complex definition of identifiers in polycontextural situations is possible.

Iterability and difference

Iterability as repetition is based on the identity of its signs, here the name of its operators. For $I(I(I)) = I$, all occurrences of the name **I** for the identity operator are identical. Now, we learn, that this constellation is a very special case for iter/alter-ability in the modus of sameness. The identical signs are the same without intrinsic differences.

The same is different. $I^i(I^j(I^k)) \neq I$ for $i \neq j \neq k$

Signs, terms, are realized at locations, they occur at semiotic places, they have an index of their occurrence. Thus, signs or marks are not anymore abstract objects, written down, by accident, on paper, living in the mind or logospere of the thinker. Written marks are presentations, designs of thoughts and not re-presentations of idealities.

The difference between token/type, occurrence/abstractness, use/mention is deplaced and deconstructed.

Petitio in principii? Yes! Why not?

Obviously, classic thinking is defending its position easily. My construction is simply a *petitio in principii*. I start with the complex constellation, and reduce the classic non-complex constellation from it. Even worse, I am using the classic notions, methods and techniques to do it. If we can learn something from the German philosophers and logician from the 60/70th, then it is this one thing: there is no justification, no legitimation, no proof for the exclusiveness of the existing rationality of logic. There is only the chance of *reconstruction* and no *foundation*, there is no such thing as a Letztbegründung. Maybe, the methods of Paul Lorenzen had been "amateurish" (Jean-Yves Girard) but we should learn from his experiences. There is no logical reason for a justification of THE "Logic of Rules" for the "Rules of Logic". There are surely all sorts of other reasons to restrict thinking and computation to the classic paradigm. But there are no reasons of whatever rationality to deny the possibility to start, for a beginning, with complex in contrast to the simple options. Circularity of my option? For whom-as what? Perhaps it is better to be aware of ones circularity than to deny ones fundamentalism.

Variants of K and S

For classical combinatory logic the identificator operator **I** seems to be quite superfluous. For transclassic combinatory logic the multitude of different identificators I^i are basic. Variants of identificators opens up variant definitions of the main operators **S** and **K**. Because each operator is identical with itself $I(K)=K$ and $I(S)=S$, different kinds of operators **K** and **S** can be defined depending on different identificators:

$I^i(S^{(m)}) = S^i$. This operation is self-applicable: $I^i(I^{(m)}) = I^i$.

$$\left[\begin{array}{c} \text{thematize } CL (SKI)^{(3)} \\ \left[\begin{array}{c} \text{contextures}^{(3)} \\ \left[\begin{array}{c} \text{identify } CL^1 \\ \left[\begin{array}{c} \text{operators} \\ \left[\begin{array}{c} \text{terms} \end{array} \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right] \left[\begin{array}{c} \text{identify } CL^2 \\ \left[\begin{array}{c} \text{operators} \\ \left[\begin{array}{c} \text{terms} \end{array} \end{array} \right] \end{array} \right] \end{array} \right] \left[\begin{array}{c} \text{identify } CL^3 \\ \left[\begin{array}{c} \text{operators} \\ \left[\begin{array}{c} \text{terms} \end{array} \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right]$$

This kind of specification is an election of a contexture out of a compound contextures. In other words, also classic formulas are "bound" by the operation "identify". Because there is only one identity and one way to identify in classic systems this operation can be omitted. But it is nevertheless decisive in an implicit

way. Transclassic systems with many options of identification, that is thematizations, have to identify their contextures and formal systems explicitly.

8 Don't halt the halting problem to halt

As shown before, and especially in Gödel Games, iteration can happen in different ways. Iterability has realizations as iterating the identical or as iterating the same, while the sameness is not identical to identity. Substitution as well as quotation or normation is a form of iterability or repeatability and can therefore be realized in different forms. Because of the tabular structure of λ -calculus problems of decidability like the halting problem has to be distributed over the interactional and reflectional dimensions.

The proof that the halting problem is noncomputable relies on the same device illustrated in the preceding paradoxes and used by Gödel to prove his Incompleteness Theorem: self-reference.

Suppose there is an algorithm H to solve the halting problem: given an encoding of a program P and an input string x , it returns 'true' if program P halts on input x , and 'false' otherwise. Use H as a subroutine to perform the conditional test in the following program H' (as formulated by Floyd and Beigel, p. 479):

```
input x, a string which encodes a program
if program x halts on input x then
  loop forever
else
  halt
```

Note that this program passes its input string to the subroutine H both as the encoding of a program (or, equivalently, a Turing machine) and as the input string for which we are to determine if program x halts. This means that H' halts on input x only if x doesn't halt on input x .

What happens if we give H' its own description as its input string?

Hoare and Allison re-state the conclusion this way (in "Incomputability", Computing Surveys 4, no. 3 [Sept. 1972]):

Any language containing conditionals and recursive function definitions which is powerful enough to program its own interpreter cannot be used to program its own 'terminates' function.

http://www.augustana.ab.ca/~mohrj/courses/1998.fall/csc110/lecture_notes/turing_machines.html

<http://www2.informatik.uni-erlangen.de/Forschung/Publikationen/download/comput.pdf?language=de>

9 Reductional closure

Thesis. The reduction procedure is preserving the conditions of mediation. If a complex formula is in CM, then the reduction to its normal form is in CM.

Proof.

Inductively over the formulas and substitution rules.

Additional:

Condition is that, if X is in CM, then $\text{super-ops}(X)$ is in CM

Combination of super-operators have to fulfil CM.

That is, not all combinations are in CM.

$$\begin{aligned} t^{(m)} \in CM, u^{(m)} \in CM &\Rightarrow \langle t, u \rangle^{(m)} \in CM \\ \langle t, u \rangle^{(m)} \in CM &\Rightarrow \langle t, u \rangle_0^{(m)} \text{ and } \langle t, u \rangle_1^{(m)} \in CM \\ (\lambda x.u)^{(m)} t^{(m)} \in CM &\Rightarrow u^{(m)} [t / x]^{(m)} \in CM \end{aligned}$$

9.1 Combining super-operators

Identity. Trivial

Replication. Trivial??

Permutation. Obvious.

Bifurcation. Not specially problematic, because CM-formulas are distributed over different places preserving the CM-properties of the distributed formulas or parts of formulas.

Reduction. Problematic. Not all reductions are preserving the CM-properties. That is, the reduction operators have to be defined properly.

$$\begin{aligned} X^{(m)} \in CM &\Rightarrow \text{sops} (X^{(m)}) \in CM : \\ X^{(m)} \in CM &\Rightarrow \text{id} (X^{(m)}) \in CM \\ X^{(m)} \in CM &\Rightarrow \text{repl} (X^{(m)}) \in CM \\ X^{(m)} \in CM &\Rightarrow \text{perm} (X^{(m)}) \in CM \\ X^{(m)} \in CM &\Rightarrow \text{red} (X^{(m)}) \in CM \\ X^{(m)} \in CM &\Rightarrow \text{bif} (X^{(m)}) \in CM \end{aligned}$$

To verify these consequences clear sepecific definitions of the single super-operator are needed.

Even such weak transformations like replications and reductions can violate the conditions of mediation.

10 Towards General Architectonics for Lambda Calculi

10.1 Architectonics with 4 contextures: One more stroke!

Architectonic designs for 4 contextures are still quite simple: the linear and the arbo-
ral distribution of calculi. Because of the super-additivity principle, in both cases we
get 6 contextual compounds for a complexity of 4. That is, 3 basic and 3 mediated
contextures and their calculi.

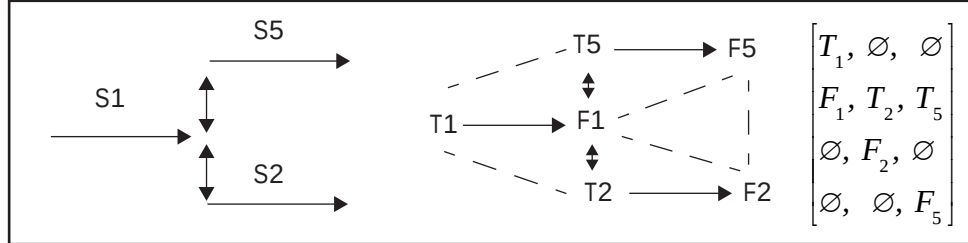
10.1.1 Linear distribution of lambda calculi

$$\begin{aligned}
 & \text{samba}_{\text{linear}}^{(4)} [id, red_1, red_1, id_6, id_6, id_6] \\
 & \left[\begin{array}{l} \text{identify contextures}^{(4)} \\ \text{thematize (reduction)} \\ \left[\begin{array}{l} (\lambda z. \langle y_{100001}, x_{011110} \rangle) \langle y^{(4)}, x^{(4)} \rangle \\ \Downarrow \beta a. \\ \langle \langle y^{(3)}, x^{(3)} \rangle_{100001} \langle y^{(3)}, x^{(3)} \rangle_{011110} \rangle \\ \Downarrow \text{subst} \\ \langle x^1, y^2, y^3, y^4, y^5, x^6 \rangle \langle y^1, x^2, x^3, x^4, x^5, y^6 \rangle \\ \Downarrow [id, red, red, id, id] \\ \langle x^1, y^1, y^1, y^6, y^6, x^6 \rangle \langle y^1, x^1, x^1, x^6, x^6, y^6 \rangle \\ \Downarrow \text{collection} \\ [\langle x^1, y^1 \rangle \langle y^1, x^1 \rangle \langle y^1, x^1 \rangle \langle y^6, x^6 \rangle \langle y^6, x^6 \rangle \langle x^6, y^6 \rangle] \end{array} \right] \end{array} \right] \\
 & (\lambda z. \langle y_{100001}, x_{011110} \rangle) \langle y^{(4)}, x^{(4)} \rangle_{[id, red_1, red_1, id_6, id_6, id_6]} \Rightarrow \left[\begin{array}{l} \langle x^1, y^1 \rangle_1 \langle y^1, x^1 \rangle_2 \langle y^6, x^6 \rangle_4 \\ \langle y^1, x^1 \rangle_3 \langle y^6, x^6 \rangle_5 \\ \langle x^6, y^6 \rangle_6 \end{array} \right] \\
 & (\lambda z. \langle y_{100001}, x_{011110} \rangle) \langle y^{(4)}, x^{(4)} \rangle \\
 & [id, red_1, red_1, id_6, id_6, id_6] \\
 & \Rightarrow \left[\begin{array}{l} \langle x^1, y^1 \rangle_1 \langle y^1, x^1 \rangle_2 \langle y^6, x^6 \rangle_4 \\ \langle y^1, x^1 \rangle_3 \langle y^6, x^6 \rangle_5 \\ \langle x^6, y^6 \rangle_6 \end{array} \right]
 \end{aligned}$$

10.1.2 Arboreal distribution of lambda calculi

With a complexity of 4, new architectonic possibilities are entering the game. Additional to the known linear order of distribution, some structural differences enter into the dynamics of architecture. The possibility of an architectonic simultaneity of 2 contexts are introduced.

3-Star pattern without mediating sub-systems



The main sub-systems are S1, S2 and S5. Sub-systems, mediating the main sub-systems are, in the full matrix, S3=(T3, F3), S4=(T4, F4) and S6=(T6, F6).

An entity of an 4-arboreal architecture which is not belonging to, say, sub-system1 can belong at once to sub-system2 and to sub-system5.

$$\begin{bmatrix} T_1, \emptyset, T_3, \emptyset, \emptyset, T_6 \\ F_1, T_2, \emptyset, \emptyset, T_5, \emptyset \\ \emptyset, F_2, F_3, T_4, \emptyset, \emptyset \\ \emptyset, \emptyset, \emptyset, F_4, F_5, F_6 \end{bmatrix}$$

$$\begin{aligned} S^5 : & \quad P[E] \rightarrow P[E'] \\ S^1 : & \quad P[E] \rightarrow P[E'] \\ S^2 : & \quad P[E] \rightarrow P[E'] \end{aligned}$$

This *architectonic simultaneity* or parallelity is not to confuse with the kind of simultaneity produced by transjunctions and bifurcations. Architectonic parallelism, in contrast to many other intra-contextual kinds of parallelism, is based on the dis-contextuality of a multitude of different mediated

contexts. Transjunctional simultaneities are generated intra-systemically by the super-operators of the system, like BIF, independent of the grounding architectonics of the system itself. Therefore, linear architectonics are containing transjunctions but no architectonic parallelisms. Architectonic parallelity contains additionally all kind of transjunctional operators. This difference has to be studied in detail. (In earlier papers the term *architectonic parallelism* was used also for transjunctional parallelism in contrast to common parallelity notions in programming.)

10.1.2.1 Syntax of 4-arboreal languages

$$X^{(4)}, Y^{(4)} \in CL_{arboreal}^{(4)} \Rightarrow \left[\begin{array}{l} \langle x^1, y^1 \rangle \langle \langle x^2, y^2 \rangle \langle x^5, y^5 \rangle \rangle \\ \langle x^3, y^3 \rangle \langle x^4, y^4 \rangle \\ \langle x^6, y^6 \rangle \end{array} \right]$$

$$X^{(4)}, Y^{(4)} \in CL_{arboreal}^{(4)} \Rightarrow \left[\langle x^1, y^1 \rangle \langle \langle x^2, y^2 \rangle \langle x^5, y^5 \rangle \rangle \langle x^3, y^3 \rangle \langle x^4, y^4 \rangle \langle x^6, y^6 \rangle \right]$$

10.1.2.2 Reduction rules for a 4-arboreal lambda calculus

$$\begin{array}{l} \text{samba}_{arboreal}^{(4)} [id^{(4)}] \\ \left[\begin{array}{l} \text{identify contextures}^{(4)} \\ \text{thematize (reduction)} \\ \left[\begin{array}{l} (\lambda z. \langle y_{100001}, x_{011110} \rangle) \langle y^{(4)}, x^{(4)} \rangle \\ \downarrow \beta a. \\ \langle \langle y^{(4)}, x^{(4)} \rangle_{100001} \langle y^{(4)}, x^{(4)} \rangle_{011110} \rangle \\ \downarrow \text{subst} \\ \langle x^1, \langle y^2, y^5 \rangle, y^3, y^4, x^6 \rangle \langle y^1, \langle x^2, x^5 \rangle, x^3, x^4, y^6 \rangle \\ \downarrow \text{collection} \\ \left[\langle x^1, y^1 \rangle \langle \langle y^2, x^2 \rangle \langle y^5, x^5 \rangle \rangle \langle y^3, x^3 \rangle \langle y^4, x^4 \rangle \langle x^6, y^6 \rangle \right] \end{array} \right] \end{array} \right] \end{array}$$

$$\Rightarrow \left[\begin{array}{l} \langle x^1, y^1 \rangle \langle \langle y^2, x^2 \rangle \langle y^5, x^5 \rangle \rangle \\ \langle y^3, x^3 \rangle \langle y^4, x^4 \rangle \\ \langle x^6, y^6 \rangle \end{array} \right]$$

$$\begin{array}{l}
\text{lambda}_{arboral}^{(4)} [id^{(4)}] \\
\left[\begin{array}{l}
\text{identify contextures}^{(4)} \\
\text{thematize (reduction)} \\
\left[\begin{array}{l}
\left(\lambda z. \langle y_{101001}, x_{010110} \rangle \right) \langle y^{(4)}, x^{(4)} \rangle \\
\Downarrow \beta a. \\
\langle \langle y^{(4)}, x^{(4)} \rangle_{101001} \langle y^{(4)}, x^{(4)} \rangle_{010110} \rangle \\
\Downarrow \text{subst} \\
\langle x^1, \langle y^2, x^5 \rangle, y^3, y^4, x^6 \rangle \langle y^1, \langle x^2, y^5 \rangle, x^3, x^4, y^6 \rangle \\
\Downarrow \text{collection} \\
\left[\langle x^1, y^1 \rangle \langle y^2, x^2 \rangle \langle x^5, y^5 \rangle \langle y^3, x^3 \rangle \langle y^4, x^4 \rangle \langle x^6, y^6 \rangle \right]
\end{array} \right]
\end{array} \right]
\end{array}$$

$$\Rightarrow \left[\begin{array}{l}
\langle x^1, y^1 \rangle \langle y^2, x^2 \rangle \langle x^5, y^5 \rangle \\
\langle y^3, x^3 \rangle \langle y^4, x^4 \rangle \\
\langle x^6, y^6 \rangle
\end{array} \right]$$

10.2 Architectonics with 5 and more strokes (contextures)

It seems to be clear now, how to map lambda calculi onto architectonic structures. Also the definition of the architectonic structures with their proemial relations are well introduced. All that seems to be not only quite simple but maybe even trivial. This tedious approach is motivated by the idea to give a concrete realization of distribution and mediation. A much more general approach could be chosen by the use of fibration, indexing and other constructions well known from category theory and general algebra. But as mentioned somewhere else before, the abstract approach is not guiding necessarily a concrete construction. After the work is done, it is appropriate and helpful to involve a more abstract approach to the topology of disseminated lambda calculi.

10.2.1 Some more stuff to do!

Diagramm 11

3-Star-1-line pattern without mediating sub-systems

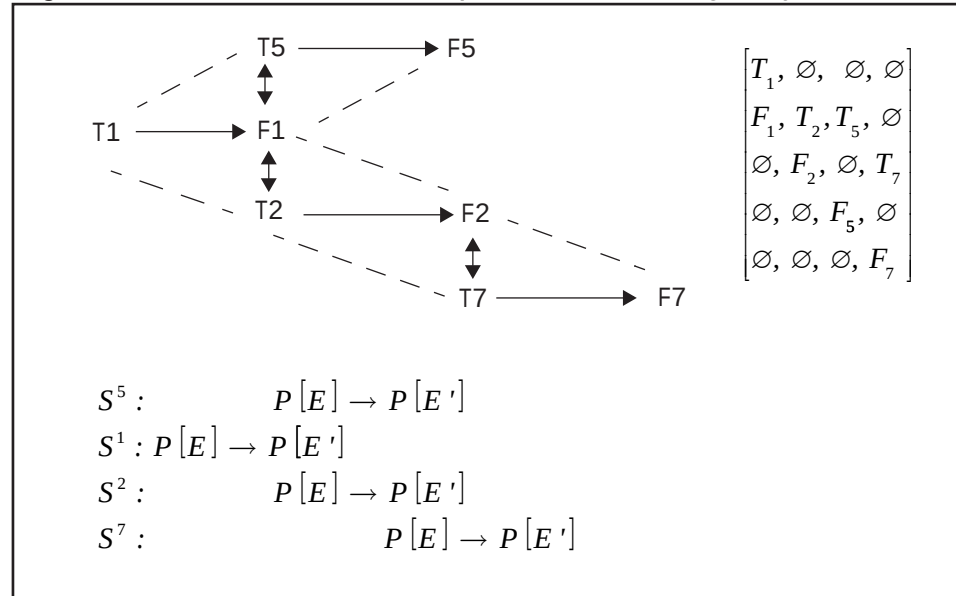
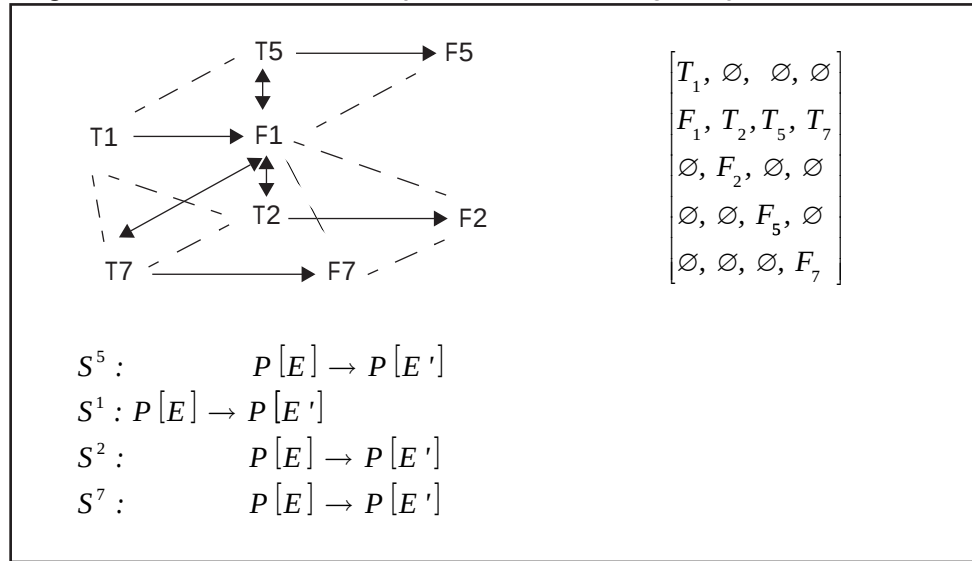


Diagramm 12

4-Star pattern without mediating sub-systems



Systems having more than one direct neighbor.

• • •

$$sketch - horizon^{(m)}$$
$$\left[\left[\left[\left[\left[\begin{array}{l} \textit{build} - \textit{architectures} \\ \textit{thematize} - \textit{scenarios} \\ \textit{identify} - \textit{frameworks} \\ \textit{define} - \textit{operations} \\ \textit{abstract} - \textit{function} \\ \{ \textit{propose} - \textit{statements} \} \end{array} \right] \right] \right] \right] \right]$$

C. Types and Contextures

To restart the game again we should introduce some more specific differentiations. Until now, our objects *obs* and complex polycontextural objects *c-obs* had not been differentiated into different sorts, structures, modules or types. They had been used as abstract syntactic objects and had no semantic specification. Like in first order logic where we have a general universal domain of individuals without any intrinsic differentiation and then all kind of sorts defined over this abstract domain, lambda calculus is introducing types over the abstract *obs* and functions. Thus, it is called *typed lambda calculus*. Obviously, computer programming, programming languages, need sorts and types or data structures to achieve some flexibility and efficiency for calculations.

The introduction of types into programming allows to differentiate between mono-morphic and poly-morphic types. And a whole machinery of techniques of realizations can start on the base of these important distinctions.

From a polycontextural point of view all these *type assignments* are of hierarchical order. Types are assigned to untyped *obs*. Thus, first are the (untyped) uniform and homogeneous objects *obs* and then, based on them, a system of type assignment is introduced. These type systems themselves may have a complex structure and interesting properties of interdependences between types can be studied.

Between terms (or *obs*) and types a stable relation of strict order is installed. An *ob* will not become a type and a type will not become an *ob* in a consistent calculus. Thus the whole systems crystalline unambiguity is ultimately realized. This is the nature of eternal truth. This may even be beautiful, at least per se. But is it not at the same time the ultimate "Killerapplication" if we have to live with it?

Where there is difference there should be change, too.

In other words, it should be possible to reverse hierarchies. This is not the beginning of anarchy but the event of chiasmic interplays of metamorphosis, i.e. heterarchy.

On the base of the concept of chiasmic types the neutrality and abstractness of the abstract objects "*obs*" is broken. Abstract objects *obs* can play the role of types too. And other types can play the role of *obs*. Thus, *obs* are not *obs*. *Obs* are *obs* as *obs*. But *obs* as types are not *obs* but types.

How does it work?

1 Types in Lambda Calculus

"A type denotes a collection of values. A function's argument type specifies which values could be returned as a result." Paulson, p.55

5.1. DEFINITION. The set of *types* of $\lambda \rightarrow$, notation $\text{Type}(\lambda \rightarrow)$, is inductively defined as follows. We write $\mathbb{T} = \text{Type}(\lambda \rightarrow)$. Let $\mathbb{V} = \{\alpha, \alpha', \dots\}$ be a set of *type variables*. It will be convenient to allow *type constants* for basic types such as Nat , Bool . Let $\mathbb{B}$ be such a collection. Then

$$\begin{aligned}\alpha \in \mathbb{V} &\Rightarrow \alpha \in \mathbb{T}, \\ \mathbf{B} \in \mathbb{B} &\Rightarrow \mathbf{B} \in \mathbb{T}, \\ \sigma, \tau \in \mathbb{T} &\Rightarrow (\sigma \rightarrow \tau) \in \mathbb{T} \quad (\text{function space types}).\end{aligned}$$

For such definitions it is convenient to use the following abstract syntax to form $\mathbb{T}$.

$$\mathbb{T} = \mathbb{V} \mid \mathbb{B} \mid \mathbb{T} \rightarrow \mathbb{T}$$

with

$$\mathbb{V} = \alpha \mid \mathbb{V}' \quad (\text{type variables}).$$

NOTATION. (i) If $\sigma_1, \dots, \sigma_n \in \mathbb{T}$ then

$$\sigma_1 \rightarrow \sigma_2 \rightarrow \dots \rightarrow \sigma_n$$

stands for

$$(\sigma_1 \rightarrow (\sigma_2 \rightarrow \dots \rightarrow (\sigma_{n-1} \rightarrow \sigma_n) \dots));$$

that is, we use association to the right.

(ii) $\alpha, \beta, \gamma, \dots$ denote arbitrary type variables.

5.2. DEFINITION. (i) A *statement* is of the form $M : \sigma$ with $M \in \Lambda$ and $\sigma \in \mathbb{T}$. This statement is pronounced as ' M in σ '. The type σ is the *predicate* and the term M is the *subject* of the statement.

(ii) A *basis* is a set of statements with only distinct (term) variables as subjects.

Barendregt 1994, p. 36

2 Towards chiasmic types in $LC^{(m)}$

2.1 Basic chiasms of types

"The type σ is the *predicate* and the term M is the *subject* of the statement."

The statement " $M : \sigma$ " is pronounced as 'M in σ '.

M is a *formula*, σ is a *type*, and $M : \sigma$ is a *statement* of the typed LC.

$M : \sigma$, obviously, this term is ruled by an order relation between M and σ , $M \rightarrow \sigma$.

And more obviously, $M : \sigma$ belongs to one and only one system of lambda calculus LC.

And again, things are dramatically different for systems like $LC^{(m)}$ where typed or not typed lambda calculi are disseminated over different kenomic loci.

Chiasmic types in $LC^{(3)}$

$LC^1 : M : \sigma$



$LC^2 : M : \sigma$

$LC^1 : M : \sigma$



$LC^2 : M : \sigma$

A kind of a formula for chiasm (M, σ)

$$M^3 \triangleq M^1 \longrightarrow \sigma^1 \Downarrow M^2 \longrightarrow \sigma^2 \triangleq \sigma^3$$

or

$$M^1 \longrightarrow \sigma^1 \Downarrow M^2 \longrightarrow \sigma^2$$

$$\triangleq$$

$$M^3 \longrightarrow \sigma^3$$

The diagram models the simple chiasmic relations between formula and type of a statement distributed over two loci, representing two different lambda calculi. At each locus,

formula and type are in an order relation. Between formula and type of different loci an exchange relation holds. To not to exchange into the blue sky, coincidence relations between types (formulas) of different levels have to be realized. All together are fulfilling the condition of proemiality (PR) or chiasm, thus the construction is sound.

This simple presentation is modeling the wording "*types becomes formulae and formulae becomes types*". (And types fit together and terms fit together.) It can be considered as an abbreviation of the complex presentation realized by the *as-abstraction* developed below. System LC^3 is, additionally, mediating the two systems LC^1 and LC^2 .

$$\left[\begin{array}{c} \text{thematize chiasm}(M, \sigma)^{(3)} \\ \left[\begin{array}{c} \text{contextures}^{(3)} \\ \left[\begin{array}{c} \text{identify } LC^1 \\ \left[\begin{array}{c} M \\ \left[\begin{array}{c} \sigma \\ \left[\text{elect } M^2 \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right] \left[\begin{array}{c} \text{identify } LC^2 \\ \left[\begin{array}{c} M \\ \left[\begin{array}{c} \sigma \\ \left[\text{elect } \sigma^1 \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right] \left[\begin{array}{c} \text{identify } LC^3 \\ \left[\begin{array}{c} M \\ \left[\begin{array}{c} \sigma \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right]$$

This first and simple kind of modeling the chiasm between subject and predicate, formula and types, is put into the bracket formalism of distributed lambda calculi or lambda based programming languages ARS.

The operator "elect" is a trans-contextual selector as the operator *sel* is an intra-contextual selector of ARS.

Lambda calculi LC in complex situations have to be identified, thus *identify* LC. And the thematization has to be chosen, thus *thematize chiasm* (*term*, *type*) in the constellation $LC^{(3)}$.

Full explanation of chiasms for types

O ₁			O ₂			O ₃		
M1	M2	M3	M1	M2	M3	M1	M2	M3
M		#	M	#	#	#	#	M
↓ σ	↓ σ	↓ σ	↓ σ	↓ σ	↓ σ	↓ σ	↓ σ	↓ σ
G ₁₂₀			G ₁₂₀			G ₀₀₃		

PM	O1	O2	O3
M1	S ₁	S ₁	∅
M2	S ₂	S ₂	∅
M3	∅	∅	S ₃

The wording here is not only "types becomes terms and terms becomes types" but "a type as a term becomes a term" and, at the same time, "a type as type remains a type". Thus, "a type as a term becomes a term and as a type it remains a type". And the same round for terms.

Full wording for a chiasm between terms and types over two loci

Explicitly, first the green round,

"A type $\sigma^{1.1}$ as a term $M^{2.1}$ becomes a term $M^{2.1}$ and as a type $\sigma^{1.1}$ it remains a type $\sigma^{1.1}$ ".

And,

"A type $\sigma^{2.2}$ as a term $M^{1.2}$ becomes a term $M^{1.2}$ and as a type $\sigma^{2.2}$ it remains a type $\sigma^{2.2}$ ".

And simultaneously, the second round in red, the same for terms:

"A term $M^{1.1}$ as a type $\sigma^{2.1}$ becomes a type $\sigma^{2.1}$ and as a term $M^{1.1}$ it remains a term $M^{1.1}$ ".

And,

"A term $M^{2.2}$ as a type $\sigma^{1.2}$ becomes a type $\sigma^{1.2}$ and as a term $M^{2.2}$ it remains a term $M^{2.2}$ ".

And finally, between terms $M^{1.1}$ and $M^{2.2}$, and types $\sigma^{1.1}$ and $\sigma^{2.2}$, a categorial coincidence is realized. To round up, the same coincidence holds for terms and types of $LC^{1.2}$ and $LC^{2.1}$.

Thus, a type has two functionalities at once, a type as a type and a type as a term.

Therefore, this double meaning has to be distributed over different localization of the complex constellation. Otherwise it simply would produce unnecessary conflictive overlapping. The *matrix* shows clearly the kind of distribution, the *diagram* is visualizing the process of the chiasm.

Putting the rounds together

$$\left[\begin{array}{c} \text{thematize chiasm}(M, \sigma)^{(3)} \\ \left[\begin{array}{c} \text{contexture}^{(3)} \\ \left[\begin{array}{c} \left[\begin{array}{c} \text{identify } LC^{1.1} \\ \left[\begin{array}{c} M \\ \sigma \\ \left[\text{elect } M^{2.1} \end{array} \right] \end{array} \right] \\ \text{identify } LC^{1.2} \\ \left[\begin{array}{c} M \\ \sigma \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right] \left[\begin{array}{c} \left[\begin{array}{c} \text{identify } LC^{2.1} \\ \left[\begin{array}{c} M \\ \sigma \end{array} \right] \end{array} \right] \\ \text{identify } LC^{2.2} \\ \left[\begin{array}{c} M \\ \sigma \\ \left[\text{elect } \sigma^{1.2} \end{array} \right] \end{array} \right] \end{array} \right] \left[\begin{array}{c} \text{identify } LC^{3.3} \\ \left[\begin{array}{c} M \\ \sigma \end{array} \right] \end{array} \right] \end{array} \right]$$

Final round up of chiaistification

We can round up the whole procedure of chiaistification of types and terms in disseminated lambda calculi by introducing *elect LC*. The exchange between types and terms is involving a switch of sub-systems and not only a switch of the categories "type / term", thus the full mechanism of chiasms, that is **PR=[term, type, LC1, LC2]**, has to implement this system change, this happens with *elect LC*.

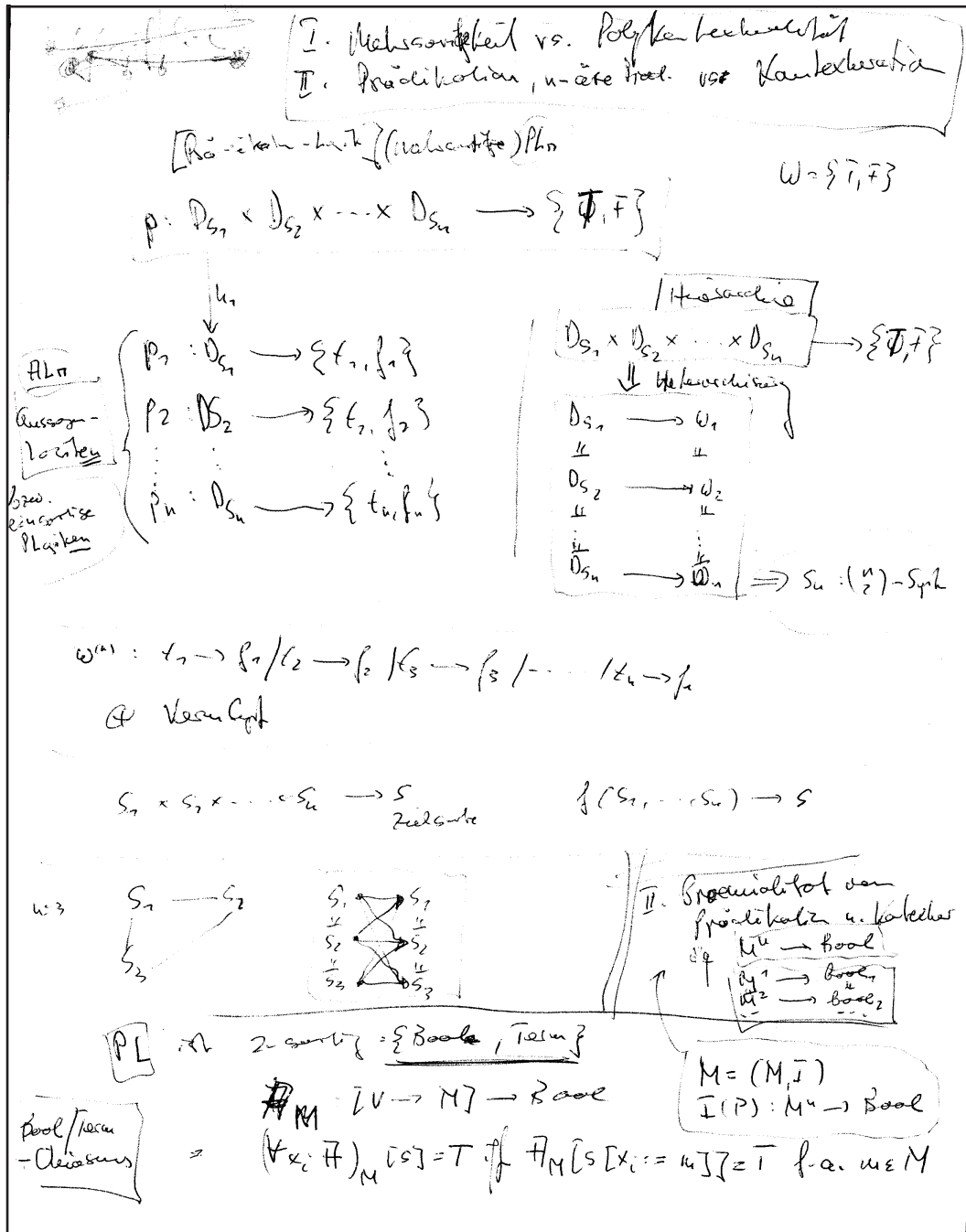
$$\left[\begin{array}{c} \text{thematize chiasm}(M, \sigma)^{(3)} \\ \left[\begin{array}{c} \text{contexture}^{(3)} \\ \left[\begin{array}{c} \left[\begin{array}{c} \text{identify } LC^{1.1} \\ \left(\begin{array}{c} M \\ (\sigma) \end{array} \right) \end{array} \right] \\ \left[\text{elect } LC^{2.1} \right] \\ \left[\begin{array}{c} \text{identify } LC^{1.2} \\ \left(\begin{array}{c} M \\ (\sigma) \end{array} \right) \end{array} \right] \end{array} \right] \left[\begin{array}{c} \left[\begin{array}{c} \text{identify } LC^{2.1} \\ \left(\begin{array}{c} M \\ (\sigma) \end{array} \right) \end{array} \right] \\ \text{identify } LC^{2.2} \\ \left(\begin{array}{c} M \\ (\sigma) \end{array} \right) \\ \left[\text{elect } LC^{1.2} \right] \end{array} \right] \left[\begin{array}{c} \text{identify } LC^{3.3} \\ \left(\begin{array}{c} M \\ (\sigma) \end{array} \right) \end{array} \right] \end{array} \right]$$

A first lesson

Polycontextural systems are, per se, untyped systems. They can contain typed systems of all kind. But they are, as such, not reducible to typed systems.

Again, the difference is between hierarchic and heterarchic order of complexities. Typed systems are in a strict sense hierarchic, polycontextural systems are heterarchic. Heterarchic systems are distributing and mediating hierarchic systems. Also heterarchic systems are not reducible to hierarchic systems it is possible to *model* and *simulate* them in hierarchic environments.

An early chiasm between universes and sorts in discontextual first-order logics.



The modeling strategy for chiasmic types in polycontextual situations is similar to the modeling strategy for chiasms in many-sorted logics. There, the chiasm is between universe(s) and sorts of disseminated logics. Sorts in one logic can change to become universes in another mediated logic. And in reverse, universes can change to sorts. Thus, chiasms are equally operating on many-sorted algebras as on typed calculi.

Classification paradigms: Sorts and Types

To contrast the very strict introduction of types in lambda calculus a more philosophical approach, offered by Peter Wegner, can help to deliberate the mind to understand the quite speculative adventures of polycontextural constructions of chiasms between statements and sorts.

A very broad and deep methodological analysis of typed systems in computer programming, not reduced to functional programming languages, is given to read by:

Peter Wegner, *The Object-Oriented Classification Paradigm*, pp.479-560, in: *Research Direction in Object-Oriented Programming*, ed. B. Shriver, P. Wegner, Computer Systems Series, MIT Press 1987

Internal and external use of the as-abstraction

In the context of typed lambda calculus the use of the as-abstraction is of importance for the interpretation of the calculus. But this procedure of taking something as something else is external and not implemented into the calculus itself.

Some as-interpretations:

- types and terms as programs and specifications,
- propositions-as-types and proofs-as-terms (Bruijn, Howard),
- types and terms as category theoretic notions.

Obviously, the lambda calculus as such is interpreted, used as something for some other purposes. Vaguely, it is a semantic interpretation of the syntactical system lambda.

In contrast, the as-abstraction is basic for the paradigm of "*lambda calculi in polycontextural situations*", short poly-lambda calculi. Thus, it is a kind of an *internal* realization of the as-abstraction, while the semantic interpretation is some kind of an *external* use of the as-abstraction. The theory of such an external use of the as-abstraction is moved to a "modeling theory" or theory of modeling, not to confuse with logical model theory.

3 Further developments: Type Derivations

On the base of the introduction of the concept of chiasmic types the whole machinery of typed lambda calculus can be reconsidered and involved into a chiasmification of its concepts and techniques.

This becomes more obvious if we introduce notions like "*function space types*" and "*derivations*" because these terms are in some sense more dynamic than the purely structural notions of types and terms and their chiasms.

3.1 Monocontextual type derivations

Typed languages are of high importance for the design of programming languages. Functional languages, which are based on lambda calculus, are directly connected with typed lambda calculus.

5.3. DEFINITION. Type *derivations* in the system $\lambda \rightarrow$ are built up from assumptions $x:\sigma$, using the following inference rules.

$$\frac{M : \sigma \rightarrow \tau \quad N : \sigma}{MN : \tau} \qquad \frac{\begin{array}{c} x : \sigma \\ \vdots \\ M : \tau \end{array}}{\lambda x.M : \sigma \rightarrow \tau}$$

5.4. DEFINITION. (i) A statement $M : \sigma$ is *derivable from* a basis Γ , notation

$$\Gamma \vdash M : \sigma$$

(or $\Gamma \vdash_{\lambda \rightarrow} M : \sigma$ if we wish to stress the typing system) if there is a derivation of $M : \sigma$ in which all non-cancelled assumptions are in Γ .

(ii) We use $\vdash M : \sigma$ as shorthand for $\emptyset \vdash M : \sigma$.

Barendregt, 1994, p. 35

3.2 Meta-theoretic results

Interesting questions about the procedure of type assignment can be put and answered. Topics are "*typechecking*", "*typability*" and "*inhabitation*".

Decidability of type assignment

For the system of type assignment several questions may be asked. Note that for $\Gamma = \{x_1:\sigma_1, \dots, x_n:\sigma_n\}$ one has

$$\Gamma \vdash M : \sigma \Leftrightarrow \vdash (\lambda x_1:\sigma_1 \dots \lambda x_n:\sigma_n.M) : (\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow \sigma),$$

therefore in the following one has taken $\Gamma = \emptyset$. Typical questions are

- (1) Given M and σ , does one have $\vdash M : \sigma$?
- (2) Given M , does there exist a σ such that $\vdash M : \sigma$?
- (3) Given σ , does there exist an M such that $\vdash M : \sigma$?

The procedure of *type assignment* is decidable. This is of great importance for theoretical and practical (programming) reasons.

These three problems are called *type checking*, *typability* and *inhabitation* respectively and are denoted by $M : \sigma?$, $M : ?$ and $? : \sigma$.

Type checking and typability are decidable. This can be shown using the following result, independently due to Curry (1969), Hindley (1969), and Milner (1978).

5.13. THEOREM. (i) *It is decidable whether a term is typable in $\lambda \rightarrow$.*

(ii) *If a term M is typable in $\lambda \rightarrow$, then M has a principal type scheme, i.e. a type σ such that every possible type for M is a substitution instance of σ . Moreover σ is computable from M .*

5.14. COROLLARY. *Type checking for $\lambda \rightarrow$ is decidable.*

3.3 Complexity as polymorphism of types

"Generally speaking, an object is polymorphic if it can be regarded as having multiple types." Paulson, p.55

Polymorphism

Note that in $\lambda \rightarrow$ one has

$$\vdash \mathbf{I} : \sigma \rightarrow \sigma \quad \text{for all } \sigma \in \mathbb{T}.$$

In the polymorphic lambda calculus this quantification can be internalized by stating

$$\vdash \mathbf{I} : \forall \alpha. \alpha \rightarrow \alpha.$$

5.15. DEFINITION. The set of *types* of $\lambda 2$ (notation $\mathbb{T} = \text{Type}(\lambda 2)$) is specified by the syntax

$$\mathbb{T} = \mathbb{V} \mid \mathbb{B} \mid \mathbb{T} \rightarrow \mathbb{T} \mid \forall \mathbb{V}. \mathbb{T}.$$

5.16. DEFINITION. The rules of type assignment are those of $\lambda \rightarrow$, plus

$$\frac{M : \forall \alpha. \sigma}{M : \sigma[\alpha := \tau]} \quad \frac{M : \sigma}{M : \forall \alpha. \sigma}$$

In the latter rule, the type variable α may not occur free in any assumption on which the premiss $M : \sigma$ depends.

These "*bytes and pieces*" should be helpful to clarify the conceptual difference between typed lambda calculi and polycontural dissemination of lambda calculi

The real stuff about Lambda: *Summer Lambda School, Lambda Calculus, Nijmegen, July 8-12, 1991, University of Nijmegen, The Netherlands.*

3.4 Complexity as polycontextuality of calculi

Typed languages are helping to avoid collisions with incommensurable sorts, like *"adding 3 volts with 2 amperes"*. Type-free languages are not restricting such use. Thus all sorts of confusion and bad self-referentiality can happen. Polycontextual calculi, typed or untyped, are not in contradiction to such house-holding strategies but also don't agree with the denial of not avoidable incommensurability. Obviously, non-avoidable incommensurability (Paul Feyerabend) have to be distributed heterarchically over different contexts, thus avoiding confusional conflicts as well as accepting incommensurable in complex situations.

3.5 Metamorphic type transformations

As shown in ConTeXtures, different kinds of metamorphic abstractions, changing from types to types, crossing different contextures in reflectional and interactional behaviors, are possible. ConTeXtures are dealing with types as *topics*, mono- and poly-topics of complex constellations of programming languages.

$samba^{(3)}(repl, \emptyset, \emptyset) - horizon$

$$\left[\begin{array}{l} \left[\begin{array}{l} \left[\begin{array}{l} \text{identify contexture}_{1.1} \\ \left(\begin{array}{l} \text{define zero} \\ \left(\begin{array}{l} \text{lambda (f)} \\ \left(\begin{array}{l} \text{lambda (x)} \\ x \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right] \\ \emptyset \quad \emptyset \end{array} \right] \\ \left[\begin{array}{l} \left[\begin{array}{l} \text{identify contexture}_{1.2} \\ \left(\begin{array}{l} \text{define zero}_{1.1} \text{ as nil} \\ \left(\begin{array}{l} \text{lambda (l)} \\ \text{true} \end{array} \right) \end{array} \right) \end{array} \right] \\ \emptyset \quad \emptyset \end{array} \right] \\ \left[\begin{array}{l} \left[\begin{array}{l} \text{identify contexture}_{1.3} \\ \left(\begin{array}{l} \text{define nil}_{1.2} \text{ as false} \\ \left(\begin{array}{l} \text{lambda (a b)} \\ b \end{array} \right) \end{array} \right) \end{array} \right] \\ \emptyset \quad \emptyset \end{array} \right] \end{array} \right]$$

This reflectional metamorphic transformation example shows a polytopic situation with the topics *Number*, *List* and *Boolean*. Thus, "define $name_i$ " is an abbreviation of "define $name_i$ as $name_j$ " with $i=j$.

– replication *repl*, in this example, is a metamorphic replication and not replicating isolated configurations.

Exchange relations:

– "define zero" is "define zero as zero", as the start of the levels. It could itself be produced by a predecessor level.

as: define zero in contexture_{1.1} as zero in contexture_{1.1}

– "define nil" is "define zero as nil", as: define zero from contexture_{1.1} as nil in contexture_{1.2}

– "define false" is "define nil as false". as: define nil from contexture_{1.2} as false in contexture_{1.3}.

Obviously, transcontextural type transformations are not identical with intra-contextural type derivations. The first are crossing the borders of contextures, from types in one contextures to other types in other contextures. This can happen successively, from one contexture to another contexture, or simultaneously, from a multitude of types in one or more contextures to a multitude of different types of different contextures.

4 Types and Paradoxes

In hierarchic type systems paradoxes can be avoided by disallowance. Paradox sentences are simply wrongly typed and therefore not part of the game. As a result, all strict self-referential statements and constructions are excluded from typed systems. This strategy is well known by Russell's "*vicious circle principle*".

Again, self-referential statements are not to be confused with recursive constructions and "bootstrapping".

<http://www.doc.ic.ac.uk/~rah03/suprema/index.cgi?section=church>