Categorical Explicit Substitutions

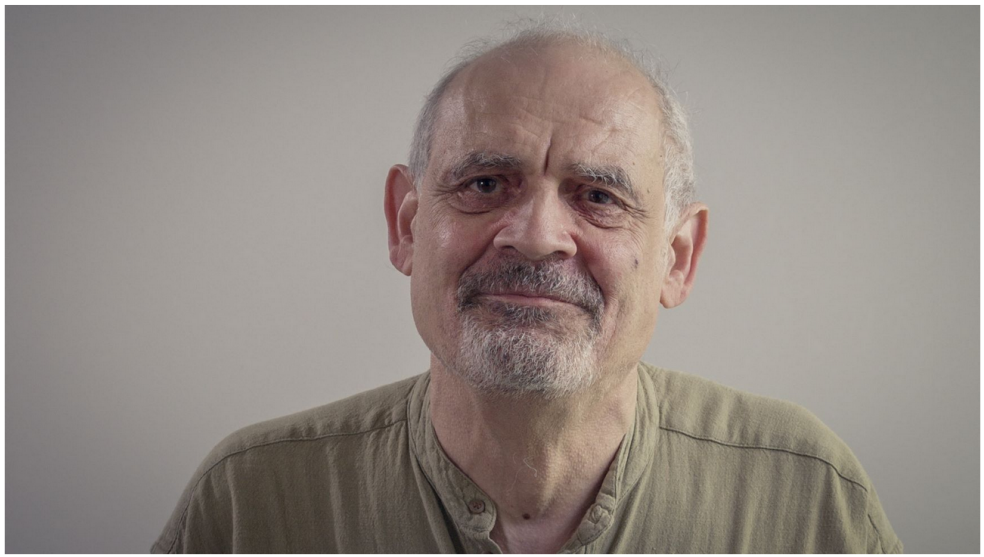
Valeria de Paiva

Topos colloquium
August 2021

Thanks for all the fish!

CAM (Categorical Abstract Machine)

Creativity and interdisciplinarity are watchwords for Pierre-Louis Curien's fundamental research in theoretical computer science, which has spawned a range of innovations and applications. Let's look back at the career of this researcher and winner of the Inria - French Academy of Sciences Grand Prize 2020.



© Inria / Photo B. Fourrier

(INRIA's announcement)

xSLAM: eXplicit Substitutions Linear Abstract Machine

- Ritter's PhD thesis on categorical combinators for the Calculus of Constructions (Cambridge 1992, TCS 1994)
- de Paiva's PhD thesis on models of Linear Logic & linear lambda-calculus (Cambridge 1990)
- Put the two together for Categorical Abstract Machines for Linear Functional Programming
- Project xSLAM Explicit Substitutions for Linear Abstract Machines (EPSRC 1997-2000)

xSLAM: eXplicit Substitutions Linear Abstract Machine

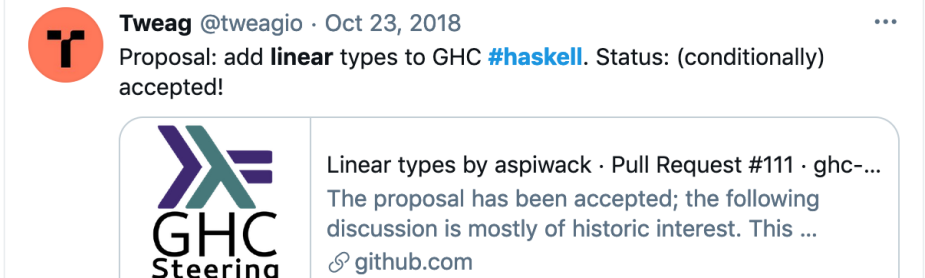
Twenty years later:

- More than 10 papers submitted to conferences
- Only two in journals: Linear Explicit Substitutions. (Ghani, de Paiva, Ritter, 2000) and Relating Categorical Semantics for Intuitionistic Linear Logic (Maietti, Maneggia, de Paiva, Ritter 2005)
- Master's thesis: An Abstract Machine based on Linear Logic and Explicit Substitutions. Francisco Alberti, 1997
- Two international workshops: Logical Abstract Machines (Saarbruecken, 1998), Logical Abstract Machines (Birmingham, 1999)
- International Dagstuhl meeting: Linear Logic and Applications (1999)

xSLAM Project Work

1. E. Ritter. A calculus for Resource allocation. Technical Report, University of Birmingham, 2000.
2. Maietti, de Paiva, Ritter. Linear primitive recursion. manuscript 2000.
3. Maietti, de Paiva, Ritter. Categorical Models for Intuitionistic and Linear Type Theory. FoSSaCS 2000.
4. Cervesato, de Paiva, Ritter. Explicit Substitutions for Linear Logical Frameworks. LFM 1999
5. E. Ritter. Characterising Explicit Substitutions which Preserve Termination. TLCA 1999
6. Ghani, de Paiva, Ritter. Explicit Substitutions for Constructive Necessity. ICALP 1998
7. Ghani, de Paiva, Ritter. Categorical Models for Explicit Substitutions. FoSSaCS 1999.
8. de Paiva, E. Ritter. On Explicit Substitutions and Names. ICALP 1997.
9. Nesi, de Paiva, Ritter. Rewriting Properties of Combinators for Intuitionistic Linear Logic. Higher Order Algebra, Logic and Term Rewriting, HOA'1993.
10. Maietti, de Paiva and Ritter. Normalization Bounds in Rudimentary Linear Logic ICC, 2002.
11. de Paiva and Eike Ritter. Variations on Linear PCF. WESTAPP, 1999.

Twenty years later



Categorical what?

Explicit substitutions!

> *In computer science, lambda calculi are said to have* explicit substitutions *if they pay special attention to the formalization of the process of substitution. This is in contrast to the standard lambda calculus where substitutions are performed by beta reductions in an implicit manner which is not expressed within the calculus.*

The concept of explicit substitutions has become notorious (despite a **large number of published calculi** of explicit substitutions in the literature with quite different characteristics) because the notion often turns up (implicitly and explicitly) in formal descriptions [...] *W*ikipedia, Feb 2021

Syntactic calculi, plenty of them. No models?

Categorical Explicit Substitutions

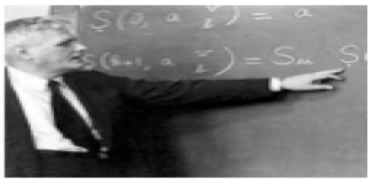
From Abadi, Cardelli, Curien and Levy 1989:

> *Substitution is the eminence grise of λ-calculus. The classical β rule*
>
> $$(\lambda x.a)b \to_\beta a\{b/x\}$$
>
> *uses substitution crucially though informally. Here a and b denote two terms, and $a\{b/x\}$ represents the term a where all free occurrences of x are replaced with b. This substitution does not belong in the calculus proper, but rather in an informal meta-level. Similar situations arise in dealing with all binding constructs, from universal quantifiers to type abstractions. [...] The correspondence between the theory and its implementations becomes highly non-trivial, and the correctness of the implementations can be compromised.*

Why Categorical Semantics?

+ Curry-Howard Correspondence

1963



Lambda-calculus

1965

Cartesian Closed Categories ⟺ Intuitionistic Propositional Logic

- Syntactic calculus should have a categorical semantics
- Mathematics as a source of intuitions
- System implementation proved correct by construction
- Hierarchy of systems
- Showing this for IPL and ILL (works for CS4$_{\Box,\Diamond}$ too)

Why functional languages?

Programs as mathematical functions transforming inputs into outputs
Work on inductively defined data structures like lists, trees
$\Rightarrow$ no side effects
$\Rightarrow$ can employ mathematical reasoning and substitute equals for equals
$\Rightarrow$ function definition and application central part of languages like Haskell, OCaml, cakeML, Rust, Scala
...
(usually) Have strong typechecking
$\Rightarrow$ can detect many errors already at compile-time

What you will NOT see in this talk

- Intersection types (idempotent or not)
- Evaluation strategies
- Dynamic types
- Patterns
- Proof-nets
- Nominal syntax
- Geometry of Interaction
- explicit substitutions Reductions, etc

Prototypical language λ-calculus

Terms:

$$M ::= x \mid \lambda x\colon A.M \mid MM$$

Types:

$$A ::= G \mid A \to A$$

Meaning of terms:
x: Variable (placeholder)
$\lambda x\colon A.M$: Function expecting input of type A
MM: Function application

Single out well-formed terms (check whether correct kind of argument supplied)

Lambda calculus

Judgements $\Gamma \vdash M : A$

$$\frac{\Gamma, x : A \vdash x : A \quad \Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \to B}$$

$$\frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

To obtain a Turing-complete language, add recursively defined functions via `letrec` and add means for defining inductively defined data structures like lists, trees, etc

Computation done via β-reduction

$$(\lambda x : A.M)N \rightsquigarrow M[N/x]$$

where $M[N/x]$ is M with x replaced by N

Environment machines

Implementing β reduction efficiently is hard
Traditional solution:
Reduce λ-terms in an environment
storing bindings of terms for variables separately
$\Rightarrow$ have two kinds of expressions:

- *Environments* $\langle M_i/x_i \rangle$: lists of bindings
- *Terms*: as before plus new closures $f * M$
- Modified β-reduction stores substitution in the environment
  $(\lambda x.t)u \Rightarrow \langle u/x \rangle * t$
- New **rewrite rules** to eliminate substitutions
  $\langle u/x \rangle * t \quad \Rightarrow^* \quad t[u/x]$

Transforming this '**trick**' of implementation into a logical calculus
$\Rightarrow$ the reason for 'explicit substitutions'

The $\lambda\sigma$-Calculus (Abadi, Cardelli, Curien, Lévy 1989)

- Terms of the λ-calculus $+$
  - Application of a substitution to a term: $f * t$
  - Start substitution: $\langle\rangle$
  - Parallel and sequential composition: $\langle f, t/x\rangle$, $f; g$
- **Typing Rules:**
  Substitutions are judgements $\Gamma \vdash f : \Delta$

$$\frac{\Gamma' \subseteq \Gamma}{\Gamma \vdash \langle\rangle : \Gamma'} \qquad \frac{\Gamma \vdash f : \Delta \qquad \Gamma \vdash t : A}{\Gamma \vdash \langle f, t/x\rangle : \Delta, x : A}$$

$$\frac{\Gamma \vdash f : \Delta \qquad \Delta \vdash t : A}{\Gamma \vdash f * t : A} \qquad \frac{\Gamma \vdash f : \Delta \qquad \Delta \vdash g : \Psi}{\Gamma \vdash f; g : \Psi}$$

- **Key Idea 1:** Every $\lambda\sigma$-term is equivalent to a λ-term.
- **Key Idea 2:** Contexts $z : A \times B$ and $x : A, y : B$ are not equal

Delia Kesner on Explicit Substitutions

‘ A Theory of Explicit Substitutions with Safe and Full Composition’ LMCS 2009: better understanding of execution models of higher-order languages

- transform the equalities of the specification into a set of rewriting rules;
- composition of substitutions allows more sophisticated interactions;
- a logical system where the reduction rules behave like cut elimination transformations;
- Melliès counter-example 1995 shows ‘a flaw in the design of ES calculi’

Kesner ES (2009)

- ways to avoid Melliès counter-example and recover the SN property;
  - weak lambda calculi. not good for proof assistants;
  - terms with explicit weakening constructors;
- Her way: a perpetual reduction strategy and a principle "Implicit substitution implies normalisation of Explicit substitution" = IE property;
- This extends ideas in [Kes07, Kes08], by the use of intersection types as well as the use of the Z-property of van Oostrom [vO] to show confluence.

What do we want from a Categorical Semantics?

- **Motivation:** What should calculi of explicit substitutions contain?
  - What term constructs should we require
  - What equations should we require
  - how to discover this for different logics?
- **Methodology:** Extend the Curry Howard correspondence
  - Category theory as syntax debugging
  - Internal language gives term constructs & equational theory
- **Key Ideas:**
  - ES model should contain a model of the underlying λ-calculus
  - Contexts $z : A \times B$ and $x : A, y : B$ are isomorphic

Models of Explicit Substitutions — Indexed Categories

- **Contexts:** Modelled by a cartesian category
  - Objects are contexts, morphisms are substitutions
  - Cartesian structure is given by parallel composition
- **Terms:** For every context Γ, define the category $D(\Gamma)$
  - Objects are variable-type pairs $x : A$
  - Morphisms $(x : A) \longrightarrow (y : B)$ are judgements $\Gamma, x : A \vdash t : B$
- **Re-indexing:** $\Gamma \vdash f : \Delta$ induces a functor $D(f) : D(\Delta) \to D(\Gamma)$
- **Summary:** An indexed category is a functor $D : C^{op} \to Cat$

What's Wrong with Indexed Categories?

- **Plus Points:** Indexed categories induce $\lambda\sigma$-equations

$$
\begin{array}{lcll}
\langle\rangle; h & = & h & \text{Identity of } C \\
h; \langle\rangle & = & h & \text{Identity of } C \\
\langle\rangle * t & = & t & \text{Functoriality of } D \\
 & & & \\
(f; g); h & = & f; (g; h) & \text{Assoc of } C\text{-comp} \\
(f; g) * t & = & f * (g * t) & \text{Functoriality of } D
\end{array}
$$

- **Lemma:** Soundness and completeness
- **Problem 1:** Indexed Categories are inherently non linear
  - Identities in the fibres require judgements $\Gamma, x : A \vdash x : A$
- **Problem 2:** No syntax for forming substitutions from terms

Context-Handling Categories

- **Solution 1:** Remove identities by changing codomain to **Set**
- **Solution 2:** Add two new natural transformations
- **Defn:** Let $\mathcal{B}$ be a SMC (CCC) with $\mathcal{T} \subseteq |\mathcal{B}|$. A *linear (cartesian) context handling category* is a functor $L : \mathcal{B}^{op} \to Sets^{\mathcal{T}}$ and for each type, natural isomorphisms

  $\mathtt{Sub}_A : L(-)_A \Rightarrow \mathcal{B}(-, A)$ $\qquad$ $\mathtt{Term}_A : \mathcal{B}(-, A) \Rightarrow L(-)_A$

- **Yoneda lemma:** There exists $\mathtt{Var}_A \in L(A)_A$ with $\mathtt{Term}_A(f) = f * \mathtt{Var}_A$
- **Equations:** We get the following equations

$$\begin{array}{lcll} \langle (f * t)/x \rangle & = & f ; \langle t/x \rangle & \text{Naturality of } \mathtt{Sub} \\ \langle t/x \rangle * x & = & t & \text{One half of the iso} \\ \langle (f * x)/x \rangle & = & f & \text{Other half of the iso} \end{array}$$

Adding Type Structure — E-Categories

- Context handling categories model the behaviour of explicit substitutions, eg their formation and application to terms
- **Types:** Since terms are defined on the fibres, modelling type constructors requires extra structure on the fibres.
- **Examples:** Given types $A, B \in \mathcal{T}$, there are types $A \to B, A \times B \in \mathcal{T}$. In addition there are isomorphisms, natural in Γ

$$\frac{E((\Gamma, A))_B}{E(\Gamma)_{A \to B}} \qquad \frac{E(\Gamma)_A \times E(\Gamma)_B}{E(\Gamma_{A \times B})}$$

- **Naturality:** Equations distribute explicit subs over terms

$$f * \lambda x.t = \lambda x.f * t \qquad f * (tu) = (f * t)(f * u)$$

- **Model Collapse:** Contexts $z : A \times B$ and $x : A, y : B$ are isomorphic

Adding Linear Type Structure — L-Categories

- **Functions:** Given types $A, B \in \mathcal{T}$, there is a type $A \multimap B \in \mathcal{T}$ and isomorphisms, natural in Γ

$$\frac{E((\Gamma, A))_B}{E(\Gamma)_{A \multimap B}}$$

- **Tensor:** But the following doesn't give an iso $z : A \otimes B \cong x : A, y : B$

$$\frac{E((\Gamma, A, B))_C}{E(\Gamma, A \otimes B)_C}$$

- **Solution:** Demand $x : A, y : B \vdash \langle (x \otimes y)/z \rangle : A \otimes B$ has inverse.

$$\frac{\Gamma, x : A, y : B \vdash f : \Delta}{\Gamma, z : A \otimes B \vdash \texttt{let}\ z\ \texttt{be}\ x \otimes y\ \texttt{in}\ f : \Delta}$$

- **Key Idea:** Category theory guiding the design of the calculus

Modelling the !-type Constructor

- **Key Idea:** !-arises as the comonad of a *monoidal adjunction* between CCCs and SMCCs
- **Recall:** Models of explicit substitutions
  - E categories model calculi with $\rightarrow, \times, 1$ types
  - L categories model calculi with $\multimap, \otimes, I$ types
- **Defn:** A $!L$-category is a monoidal adjunction where C is an E category, B is an L-category and F, G preserve types.
- **Crucially:** Requires an isomorphism $x : A|\_ \cong \_|z :!A$

$$\frac{\Gamma, x : A|\Delta \vdash f : \Delta}{\Gamma|\Delta, z :!A \vdash \texttt{let } z \texttt{ be } !x \texttt{ in } f : \Delta}$$

Categorical Theorems

Have three theorems for E-categories (CCCs), three theorems for L-categories (SMCCs), two for putting the E- and L-categories together into a !L-category.

- (right defn) Every E-category contains an underlying CCC and every CCC extends to an E-category, in an iso way;
- (soundness) We can model the $\lambda\sigma$ calculus in an E-category;
- (completeness) If for every E-category, the interpretation of two morphisms f, f' is the same, then the equality of the morphisms is proved by the $\lambda\sigma$ theory.

Similarly for L-categories.
For !L-categories, we use Barber's DILL calculus with explicit substitutions xDILL, described in *Linear Explicit Substitutions*

First Conclusions

- Indexed categories are inherently non-linear but we can use presheaves
- Explicit substitutions are tricky to model (extensionally they are equivalent to their underlying λ-calculus)
- Our models reflect this by "taking isomorphisms seriously"
- We get an extension of Curry-Howard correspondence justifying our calculus
- Have a working indexed-cat model for linear, cartesian, both as well as the original for the Calculus of Constructions

Explicit substitutions are not simply a hack: find maths of clever implementation

ILT: no modality for better behaviour

- Better type theory for better implementations
- Which type theory? why better?
- Categorical Models for Linear and Intuitionistic Type Theory (Maietti, Ritter, de Paiva, FOSSACS, LNCS Springer, vol 1784, 2000)

Linear Type Theories with Categorical Models

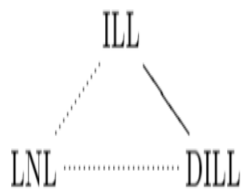
- Linear λ-calculus (Benton, Bierman, de Paiva, Hyland, 1992) https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-262.html
- DILL system (Plotkin and Barber, 1996) http://www.lfcs.inf.ed.ac.uk/reports/96/ECS-LFCS-96-347/
- LNL system (Benton, 1995) `https://www.researchgate.net/publication/221558077_A_Mixed_Linear_and_Non-Linear_Logic_Proofs_Terms_and_Models`

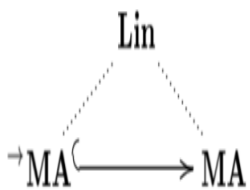
Their Categorical Models

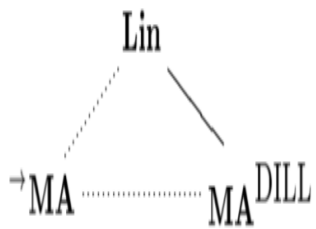
- Linear λ-calculus (ILL)
  A linear category is a symmetric monoidal closed (smc) category equipped with a linear monoidal comonad, such that the co-Kleisli category of the comonad is a Cartesian closed Category (CCC). (loads of conditions)
- DILL system
  A DILL-category is a pair, a symmetric monoidal closed category and a cartesian category, related by a (symmetric) monoidal adjunction.
- LNL system
  An LNL-model is a (symmetric) monoidal adjunction between a smcc and a ccc.

Linear Curry-Howard Isos

"Relating Categorical Semantics for Intuitionistic Linear Logic"(with P. Maneggia, M. Maietti and E. Ritter), Applied Categorical Structures, vol 13(1):1–36, 2005.







Take home: Categorical models need to be more than sound and complete. Need to provide *internal languages* for the theories they model.

Why a new calculus?

(ILT, Fossacs 2000)

Choosing between the three type theories:

- DILL is best, less verbose than ILL, but closer to what we want to do than LNL.
- Easy formulation of the promotion rule
- Contains the usual lambda-calculus as a subsystem
- however, most FPers would prefer to use a variant of DILL where instead of !, one has two function spaces, $\to$ and $\multimap$
- But then what's the categorical model?

Intuitionistic and Linear Type Theory

(ILT, Calculus)

If we have two function spaces, but no modality !, how can we model it?
All the models discussed before have a notion of !, a comonad created by the adjunction.
Well, we need to use deeper mathematics, i.e. **fibrations** or indexed categories.

Calculus ILT

$$\Gamma \mid a : A \vdash a : A \qquad \Gamma, x : A \mid \_ \vdash x : A$$

$$\frac{\Gamma \mid \Delta, a : A \vdash M : B}{\Gamma \mid \Delta \vdash \lambda a^{A}.M : A \multimap B} \qquad \frac{\Gamma, x : A \mid \Delta \vdash M : B}{\Gamma \mid \Delta \vdash \lambda x^{A}.M : A \to B}$$

$$\frac{\Gamma \mid \Delta_1 \vdash M : A \multimap B \quad \Gamma \mid \Delta_2 \vdash N : A}{\Gamma \mid \Delta \vdash M \,{}_{!}N : B} \qquad \frac{\Gamma \mid \Delta \vdash M : A \to B \quad \Gamma \mid \_ \vdash N : A}{\Gamma \mid \Delta \vdash M \,{}_{¡}N : B}$$

ILT Models

Idea goes back to Lawvere's hyperdoctrines satisfying comprehension

Model of ILT should modify this setting to capture the separation between intuitionistic and linear variables.

A base category B, which models the intuitionistic contexts of ILT, objects in B model contexts $(\Gamma|\_)$.

Each fibre over an object in B modelling a context models terms $\Gamma|\Delta \vdash M : A$ for any context Δ

The fibres are now symmetric monoidal closed categories with tensor products and model the linear constructions of ILT

ILT Models and internal language theorems

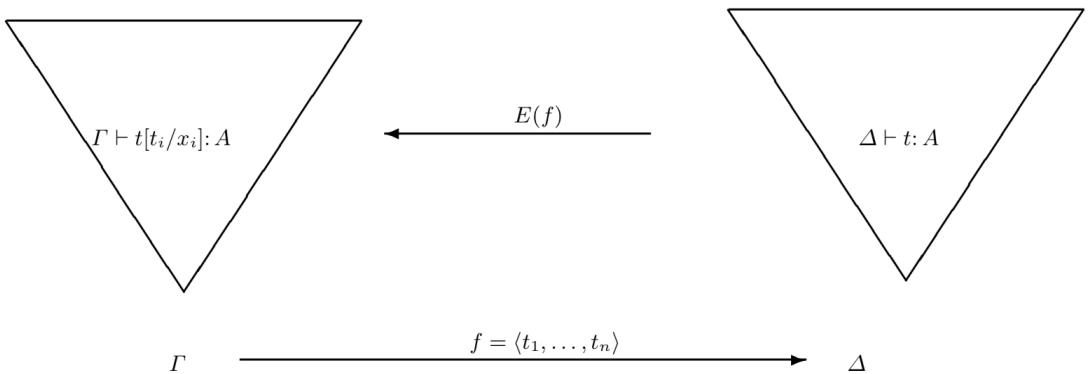


Fig. 1. Modelling the simply-typed λ-calculus in a D-category

Just as in the previous 'example' can prove soundness, completeness and internal language theorems.
(or so they say)
Missing an understanding of what is essential, what can be changed, and limitations of the methods.

Conclusions

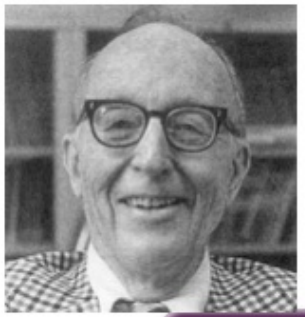
- Explicit substitutions discussions stuck on issues:
  - equations-in-context vs. raw reductions
  - de Bruijn indices vs. variable names
  - untyped vs. typed calculi
  - calculus for abstract machines or for proof assistants, etc..
- our explicit substitution calculus has been driven by the correspondence with the (well-established) cat semantics
- Ritter showed that every reduction used in abstract machines terminates, thus SN is not necessary for their design. $\Rightarrow$ not worried about SN or metavariables
- still: what can we say about ES calculi we skipped? what more do we need to say? which other logics can we do?

Thanks!

Curry-Howard Enablers

Curry-Howard Correspondence TriangleS

Functional Programming



Category Theory

Proof Theory

Intuitionistic Type Theory

Curry-Howard Correspondence Triangles

Lambda-calculus

1963

Cartesian Closed Categories

Intuitionistic Propositional Logic

1965

Linear Type Theory

Curry-Howard Correspondence



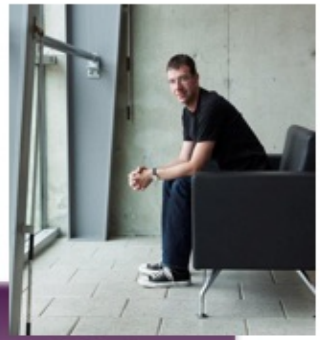
Linear Lambda-Calculus

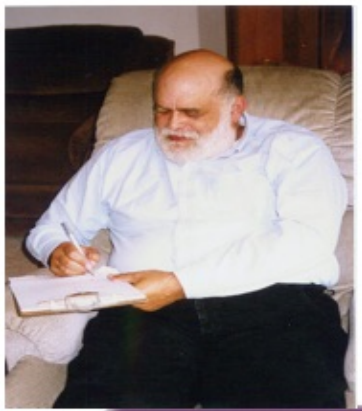


Linear Categories

Linear Logic

Modal (S4) Curry-Howard Correspondence





Modal S4 Lambda-Calculus

CCC+monoidal comonad

Constructive S4 Modal Logic

Lambek calculus Curry-Howard Correspondence

2-sided Linear Lambda-Calculus

Monoidal bi-closed Categories

Lambek Calculus