A multicategorical approach to mixed linear-non-linear substitution

Journées nationales du GDRIM 2022 - Villeneuve d'Ascq

Christine Tasson (christine.tasson@lip6.fr)

Semantics and Programming Languages

Semantics is a key tool for **designing** *programming languages,* **formalization** *and* **proof** *of programs all through the continuum from machine code to high level programming languages.*

Semantics and Programming Languages

Semantics is a key tool for **designing** *programming languages,* **formalization** *and* **proof** *of programs all through the continuum from machine code to high level programming languages.*

This talk is about

- Substitution
- Linearity and Non-linearity
- (multi-)Category Theory

Computer Science **Mathematics**

$$\text{Programs} \xrightarrow[\text{Semantics}]{\text{Denotational}} \text{Interpretation}$$

Computer Science $\circlearrowleft$ Mathematics

```python
import random
def flip(p:float) -> int:
  if random.random()<p:
    return 0
  else:
    return 1
```

$$
\begin{array}{ccc}
[0,1] & \xrightarrow{f} & \mathcal{V}([0,1]) \\
0.3 & \mapsto & 0.3\,\delta_0 + 0.7\,\delta_1
\end{array}
$$

$$
\begin{tikzcd}
\text{Programs} \arrow[rr, bend left, "\text{Denotational Semantics}"] & & \text{Interpretation}
\end{tikzcd}
$$

Computer Science $\circlearrowleft$ Mathematics

```python
import random
def flip(p:float) -> int:
  if random.random()<p:
    return 0
  else:
    return 1
```

$$\longrightarrow \quad \begin{array}{ccc} [0,1] & \xrightarrow{f} & \mathcal{V}([0,1]) \\ 0.3 & \mapsto & 0.3\,\delta_0 + 0.7\,\delta_1 \end{array}$$

$$\begin{tikzcd}
\text{Programs} \arrow[r, bend left, "\text{Denotational Semantics}"] & \text{Interpretation} \\
\text{Constructions} & \text{Structures} \arrow[l, bend left, "\text{Computational content}"]
\end{tikzcd}$$

Computer Science $\circlearrowright$ Mathematics

```python
import random
def flip(p:float) -> int:
  if random.random()<p:
    return 0
  else:
    return 1
```

$$\begin{array}{ccc} [0,1] & \xrightarrow{f} & \mathcal{V}([0,1]) \\ 0.3 & \mapsto & 0.3\,\delta_0 + 0.7\,\delta_1 \end{array}$$

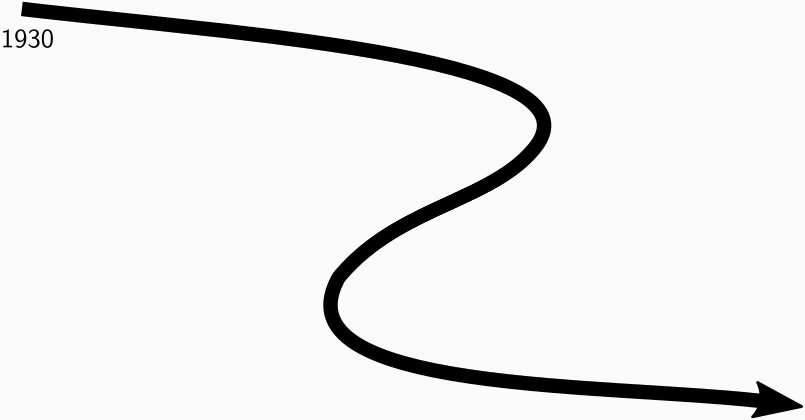
$$\begin{tikzcd}
\text{Programs} \arrow[r, bend left, "\text{Denotational Semantics}"] & \text{Interpretation} \\
\text{Constructions} & \text{Structures} \arrow[l, bend left, "\text{Computational content}"]
\end{tikzcd}$$

$$\text{Lambda} \longleftarrow \text{Functions}$$

```python
def shift(n:int) :
    return lambda s:s+n
```

Lambda-Calculus

Church



1930: Church

Lambda-terms represent computable functions.

	Programs	Functions	
	M, N	$f, g : \mathbb{N} \longrightarrow \mathbb{N}$	
Variable	x	x	Variable
Abstraction	$\lambda x.M$	$f : x \mapsto 5 * x * (x + 1) + 3$	Map
Application	$(\lambda x.M)N$	$f(g) = 5 * g * (g + 1) + 3\,)$	Substitution

Church

Lambda-Calculus

Computers

1930

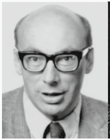
1940

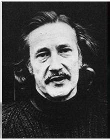
1950

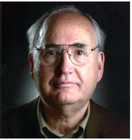
Lambda-Calculus

Church

Operational and Denotational Semantics

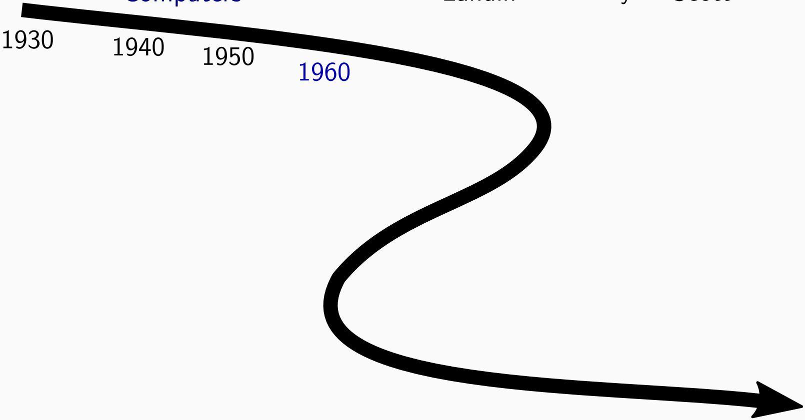






Landin Strachey Scott

Computers



1960: From syntax to semantics

Syntax describes how to write programs,
Semantics describes how and what programs compute.

Operational semantics describes program execution as transition system. (Landin 1966)

For λ-calculus, **substitution** in contexts

$$(\lambda x.M)N \to M[N/x]$$

Denotational Semantics denotes programs as functions acting on *values* and on *memory state*.

(Strachey 1960) (Scott 1969)
For pure λ-calculus, solving **equation**

$$D \stackrel{?}{=} Var + [D \to D] + \cdots$$

1960: From syntax to semantics

Syntax describes how to write programs,
Semantics describes how and what programs compute.

Operational semantics describes program execution as transition system. (Landin 1966)

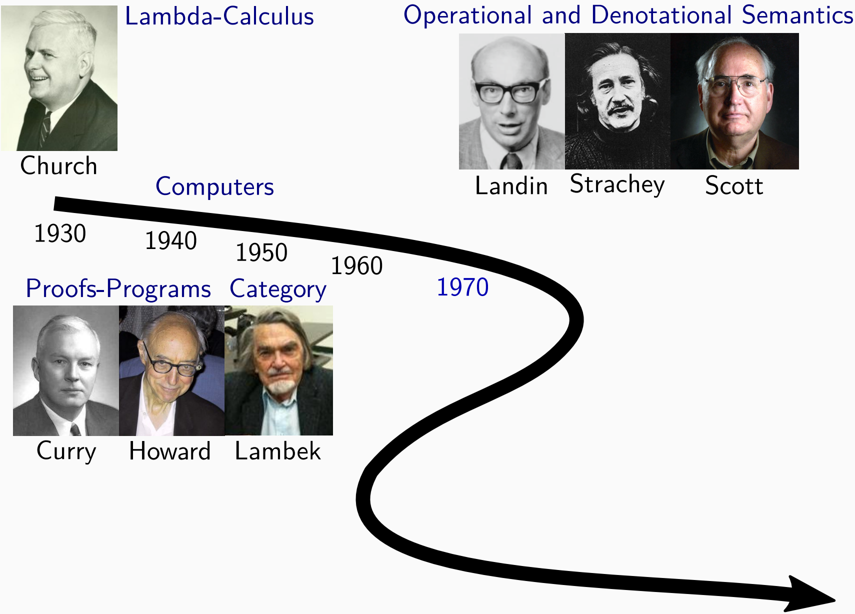
For λ-calculus, **substitution** in contexts

$$(\lambda x.M)N \to M[N/x]$$

Denotational Semantics denotes programs as functions acting on *values* and on *memory state*.

(Strachey 1960) (Scott 1969)
For pure λ-calculus, solving **equation**

$$D \stackrel{\checkmark}{=} Var + \underbrace{[D \to D]}_{\text{Continuous}} + \cdots$$


Lambda-Calculus
Church
Operational and Denotational Semantics
Landin
Strachey
Scott
Computers
1930
1940
1950
1960
1970
Proofs-Programs
Category
Curry
Howard
Lambek

1970: Computer Science - Logic - Category

Curry-Howard

λ-calculus	**Logic**
Program :Type	Proof : Formula
$M : A \Rightarrow B$	$\dfrac{\pi}{A \Rightarrow B}$

1970: Computer Science - Logic - Category

Curry-Howard

λ-calculus

Program :Type

$M : A \Rightarrow B$

Logic

Proof : Formula

$$\frac{\pi}{A \Rightarrow B}$$

Lambek

Cartesian Closed Categories

$[\![M]\!]$: Morphism from context to output

$[\![A \Rightarrow B]\!]$: Object with

- evaluation: $ev : (A \Rightarrow B) \times A \rightarrow B$
- curryfying: $f : C \times A \rightarrow B$ corresponds to $\Lambda f : C \rightarrow (A \Rightarrow B)$

1970: Computer Science - Logic - Category

Curry-Howard

λ-calculus

Program :Type

$M : A \Rightarrow B$

Logic

Proof : Formula

$$\frac{\pi}{A \Rightarrow B}$$

Lambek

Cartesian Closed Categories

$[\![M]\!]$: Morphism from context to output

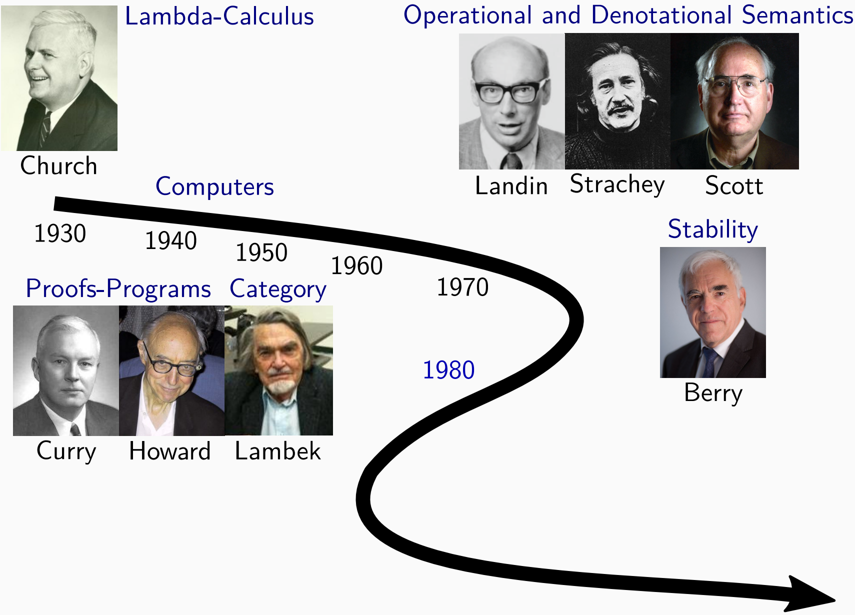
$[\![A \Rightarrow B]\!]$: Object with

- evaluation: $ev : (A \Rightarrow B) \times A \to B$
- curryfying: $f : C \times A \to B$ corresponds to $\Lambda f : C \to (A \Rightarrow B)$

OCaml

Categorical Abstract Machine influenced prototypes

(Milner, Cousineau, Curien, Leroy, . . .)


Lambda-Calculus
Church
Operational and Denotational Semantics
Landin
Strachey
Scott
Computers
1930
1940
1950
1960
1970
1980
Proofs-Programs
Category
Curry
Howard
Lambek
Stability
Berry

1980: Sequential algorithms

PCF a typed functional languages such as `Haskell` or `ML`

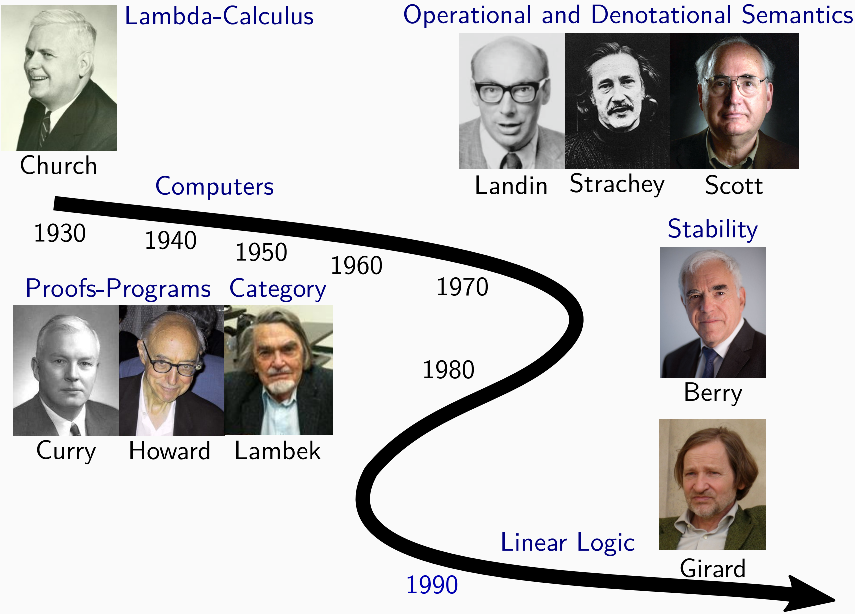
Denotational Semantics

Scott Domains contain non sequential functions such as Parallel-Or.

Stability gets rid of this example, but does not characterize *sequentiality*

Sequential algorithm model uses the language of category (Berry-Curien 1982)

The Full Abstraction quest generates new models *Hypercoherence* (Ehrhard 1993) and *Game semantics* (Abramsky-Jagadeesan-Malacaria 1994), (Hyland-Ong 1995)



Lambda-Calculus
Church
Operational and Denotational Semantics
Landin
Strachey
Scott
Computers
1930
1940
1950
1960
1970
1980
1990
Stability
Berry
Proofs-Programs
Category
Curry
Howard
Lambek
Linear Logic
Girard

1990: Linear Logic

Semantical observation: (Girard 1987)

$$A \overset{Stable}{\Rightarrow} B \quad \simeq \quad !A \overset{Linear}{\multimap} B$$

Categorical models

$$\begin{tikzcd}
\textbf{Linear} & & \textbf{Non-Linear} \\
\begin{matrix}(\mathcal{L}, 1, \otimes, \multimap) \\ \text{SMCC}\end{matrix} \arrow[rr, bend left] \arrow[rr, phantom, "\perp"] & & \begin{matrix}(\mathcal{M}, \top, \times, \Rightarrow) \\ \text{CCC}\end{matrix} \arrow[ll, bend left]
\end{tikzcd}$$

1990: Linear Logic

Semantical observation: (Girard 1987)

$$A \overset{\textit{Stable}}{\Rightarrow} B \quad \simeq \quad !A \overset{\textit{Linear}}{\multimap} B$$

Categorical models

$$\begin{tikzcd}[column sep=huge] \textbf{Linear} & \textbf{Non-Linear} \\ \begin{array}{c}(\mathcal{L}, 1, \otimes, \multimap)\\ \text{SMCC}\end{array} \arrow[r, bend left] \arrow[r, phantom, "\perp"] & \begin{array}{c}(\mathcal{M}, \top, \times, \Rightarrow)\\ \text{CCC}\end{array} \arrow[l, bend left] \end{tikzcd}$$

Rust

Linear Type Systems influenced prototypes

(Berry, Girard, Pym, O'Hearn, Dreyer, Graydon Hoare. . .)

1991: Monads

Eugenio Moggi wrote Notions of computation and Monads and formalized computation and effects in categorical semantics.

Monad T

maybe monad $TA = \text{Nothing or } A$

TA denotes the computations of type A and come with

- multiplication $\mu : T^2A \to TA$ describes how to compose effects
- unit $\eta : A \to TA$ embeds effect-free computations into effectful computations

1991: Monads

Eugenio Moggi wrote Notions of computation and Monads and formalized computation and effects in categorical semantics.

Monad T

maybe monad TA = Nothing or A

TA denotes the computations of type A and come with

- multiplication $\mu : T^2A \to TA$ describes how to compose effects
- unit $\eta : A \to TA$ embeds effect-free computations into effectful computations

---

Haskell

Monads influenced the design of Haskell

(Plotkin, Power, Wadler, Jones...)

Influence of semantics on the design of programming languages

Challenges

- **Machine learning and Automatic Differentiation** (Tensorflow, Julia, Swift, Pytorch, JAX,. . .),
- **Statistical Learning** (Anglican, Gen, Birch, Pyro,. . .),
- **Quantic** (Quipper, Choir, ZX-calculus, . . .).

Influence of semantics on the design of programming languages

Challenges

- **Machine learning and Automatic Differentiation** (Tensorflow, Julia, Swift, Pytorch, JAX,...),
- **Statistical Learning** (Anglican, Gen, Birch, Pyro,...),
- **Quantic** (Quipper, Choir, ZX-calculus, ...).

Key concepts

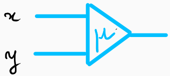
- **Substitution** (replace) and evaluation
- **Compositionnality** (compute from smaller blocks)
- **Higher-order** (programs with programs as parameters)

Theory of Substitution

Examples

What is a group in Set?

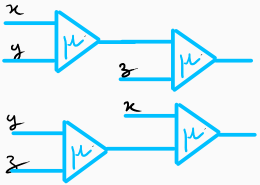
A Monoid



$x \cdot y = \mu(x,y)$

e

Associativity law

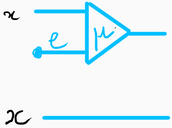


$(x\cdot y)\cdot z$

$=$

$x\cdot(y\cdot z)$

Unit law



$x\cdot e$

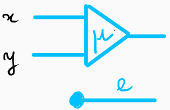
$=$

x

Hopf Algebras

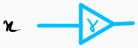
What is a group in Set?

A Monoid with an Inverse



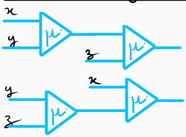
$x \cdot y = \mu(x,y)$

e



$x^{-1} = \gamma(x)$

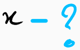
Associativity law

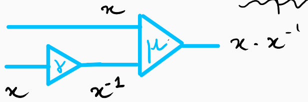


Inverse law $x \cdot x^{-1} = e$

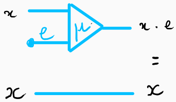
Input

Output



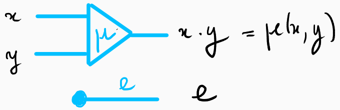


Unit law

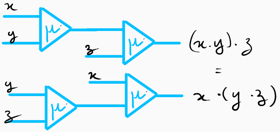


What is a group in Set?

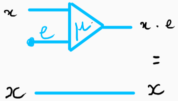
A Monoid with an Inverse



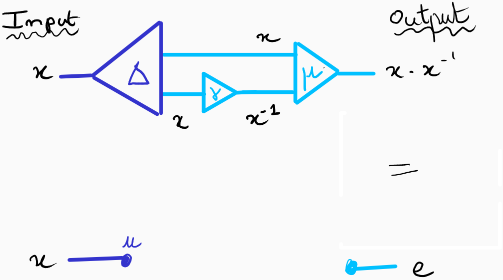
Associativity law



Unit law

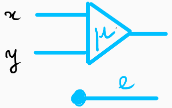


Inverse law $x \cdot x^{-1} = e$



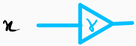
What is a group in Set?

A Monoid with an Inverse



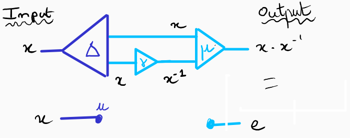
$x \cdot y = \mu(x,y)$

e

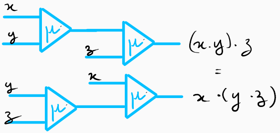


$x^{-1} = \gamma(x)$

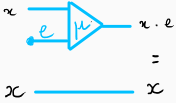
Inverse law $x \cdot x^{-1} = e$



Associativity law



Unit law



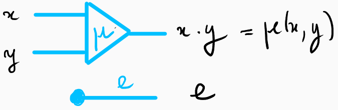
$\mu : X \times X \to X$
$e : T \to X$
satisfying unit, associativity, inverse laws.

$\Delta : X \to X \times X$
$u : X \to T$
Exists because Set is cartesian

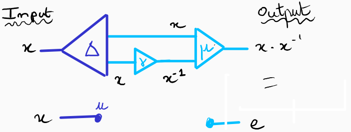
$\gamma : X \to X$

What is a group in ~~Set~~? Vect

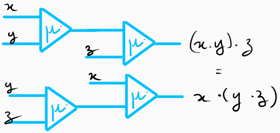
A Monoid with an **Inverse**



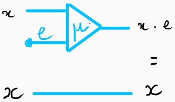
$\boxed{\text{Inverse law } \; x \cdot x^{-1} = e}$



$\boxed{\text{Associativity law}}$



$\boxed{\text{Unit law}}$



HOPF ALGEBRA

$\mu : X \otimes X \to X$

$e : 1 \to X$

$\Delta : X \to X \otimes X$

$u : X \to 1$

$\gamma : X \to X$

satisfying unit, associativity, inverse laws.

~~Exists because Set is cartesian~~

Ask for it

Consumable computational resources

```
L = 5*[0]
M = []
for i in range(5):
    M.append(L)
M[0][0] = 1

>>> M
[[1, 0, 0, 0, 0],
 [1, 0, 0, 0, 0],
 [1, 0, 0, 0, 0],
 [1, 0, 0, 0, 0],
 [1, 0, 0, 0, 0]]
```

Consumable computational resources

- Allocated memory
- Open file handles
- Socket connections

Linear type influence

- Uniqueness types, Ownership, borrowed pointers
- `Mezzo`, `Cyclone`, `Rust`, `Idris`...

Yet, programs and proofs are not linear in generaly as one needs to copy values and to reuse lemmas.

What is a linear-non-linear semantics ?

Equivalent Probabilistic Programs?

```
let x = sample (flip p)
in if x
   then if x
        then 1
        else 2
   else if x
        then 2
        else 3
```

outputs:

- 1 with probability p
- 2 with probability 0
- 3 with probability $(1-p)$

```
let x = flip p
in if sample x
   then if sample x
        then 1
        else 2
   else if sample x
        then 2
        else 3
```

outputs:

- 1 with probability p^2
- 2 with probability $2\,p\,(1-p)$
- 3 with probability $(1-p)^2$

Equivalent Probabilistic Programs?

```
let x = sample (flip p)
in if x
   then if x
        then 1
        else 2
   else if x
        then 2
        else 3
```

outputs:

- 1 with probability p
- 2 with probability 0
- 3 with probability $(1-p)$

```
let x = flip p
in if sample x
   then if sample x
        then 1
        else 2
   else if sample x
        then 2
        else 3
```

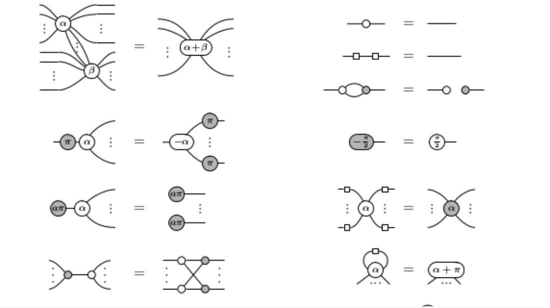
outputs:

- 1 with probability p^2
- 2 with probability $2\,p\,(1-p)$
- 3 with probability $(1-p)^2$

Substitution of effectful constructs is subtile due to duplication.

Quantum Programming

ZX-Calculus (https://zxcalculus.com/)



Linear and non-linear resources

- Quantum ressources can be used once : they are Linear
- Classical ressource can be used at will : they are non-Linear

Differential λ-Calculus

(Ehrhard-Regnier 2003, Ehrhard 2018)

Semantical observation: λ-terms can be interpreted by smooth functions, hence differentiation.

	Programs	**Functions**	
	M, N	f, g	
Variable	x	x	Variable
Abstraction	$\lambda x.M$	$f : x \mapsto f(x)$	Map
Application	$(\lambda x.M)N$	$f \circ g$	Composition
Linear Application	$D\lambda x.M \cdot N$	$Df_x \circ g$	Derivation

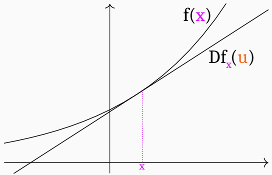
Linear and Non-Linear substitutions in Differential λ-calculus

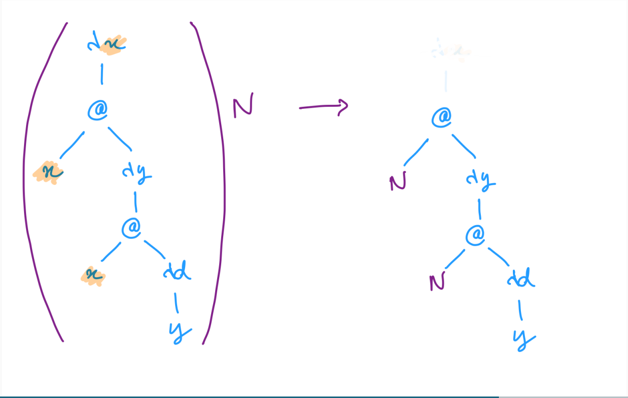
Linear and Non Linear Substitutions

$$(\lambda x.M)N \quad \rightarrow \quad M[x \backslash N]$$

$$D\lambda x.M \cdot N \quad \rightarrow \quad \lambda x.\left(\frac{\partial M}{\partial x} \cdot N\right)$$

Linear approximation





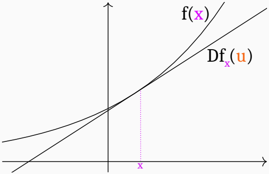
Linear and Non-Linear substitutions in Differential λ-calculus

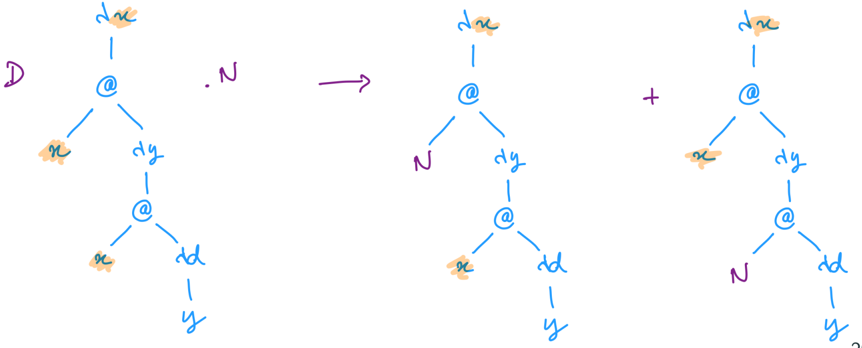
Linear and Non Linear Substitutions

$$(\lambda x.M)N \quad \rightarrow \quad M[x\backslash N]$$

$$D\lambda x.M \cdot N \quad \rightarrow \quad \lambda x.\left(\frac{\partial M}{\partial x} \cdot N\right)$$

Linear approximation





Theory of Substitution

Linear-non-linear Multicategories

Multiplicative Linear Logic

Linear Type system

$$\frac{}{x : a \quad \vdash x : a} \qquad \frac{\Gamma, x : a \quad \vdash t : b}{\Gamma \quad \vdash \lambda x^a.t : a \multimap b}$$

$$\frac{\Gamma \quad \vdash s : a \multimap b \qquad \Gamma' \quad \vdash t : a}{\Gamma, \Gamma' \quad \vdash \langle s \rangle t : b}$$

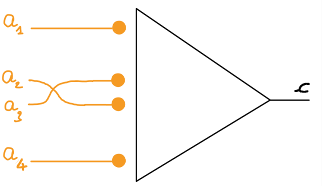
Multicategorical axiomatization

$$x_1 : a_1, \ldots, x_\ell : a_\ell \vdash t : c$$

denoted as

$$a_1, \ldots, a_\ell \multimap c.$$

symetric multicategory:



Multiplicative Linear Logic

Linear Type system

$$\frac{}{x : a \quad \vdash x : a} \qquad \frac{\Gamma, x : a \quad \vdash t : b}{\Gamma \quad \vdash \lambda x^a.t : a \multimap b}$$

$$\boxed{\frac{\Gamma \quad \vdash s : a \multimap b \qquad \Gamma' \quad \vdash t : a}{\Gamma, \Gamma' \quad \vdash \langle s \rangle t : b}}$$

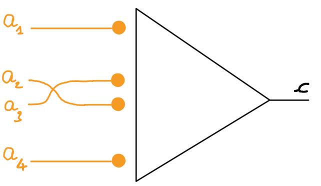
Multicategorical axiomatization

$$x_1 : a_1, \ldots, x_\ell : a_\ell \vdash t : c$$

denoted as

$$a_1, \ldots, a_\ell \multimap c.$$

symetric multicategory:



Lambda-calculus

Non-Linear Type system

$$\frac{}{\Delta, x : \underline{b} \vdash x : \underline{b}} \qquad \frac{\Delta\,,\, x : \underline{a} \vdash t : b}{\Delta \vdash \lambda x^{\underline{a}}.t : \underline{a} \longrightarrow b}$$

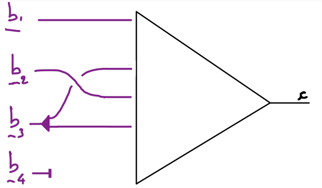
$$\frac{\Delta \vdash s : a \longrightarrow b \qquad\qquad \Delta \vdash t : a}{\Delta \vdash (s)t : b}$$

Multicategorical axiomatization: cartesian multicategory (a clone):

$$y_1 : \underline{b}_1, \ldots, y_n : \underline{b}_n \vdash t : c$$

denoted as

$$\underline{b}_1, \ldots, \underline{b}_n \longrightarrow c.$$



Lambda-calculus

Non-Linear Type system

$$\frac{}{\Delta, x : \underline{b} \vdash x : \underline{b}} \qquad \frac{\Delta\,,\, x : \underline{a} \vdash t : b}{\Delta \vdash \lambda x^{\underline{a}}.t : \underline{a} \longrightarrow b}$$

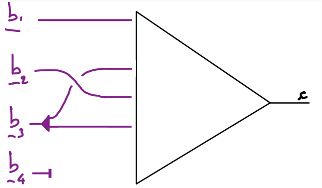
$$\boxed{\frac{\Delta \vdash s : a \longrightarrow b \qquad\qquad \Delta \vdash t : a}{\Delta \vdash (s)t : b}}$$

Multicategorical axiomatization: cartesian multicategory (a clone):

$$y_1 : \underline{b}_1, \ldots, y_n : \underline{b}_n \vdash t : c$$

denoted as

$$\underline{b}_1, \ldots, \underline{b}_n \longrightarrow c.$$



A term calculus for Linear-non-linear Logic

(Benton-Bierman-de Paiva-Hyland 1993, Barber 1996)

$$\boxed{\underbrace{x_1 : a_1, \ldots, x_\ell : a_\ell}_{\text{Linear}} \mid \underbrace{\underline{y}_1 : \underline{b}_1, \ldots, \underline{y}_n : \underline{b}_n}_{\text{Non-Linear}} \vdash t : c}$$

Linear rules:
$$\frac{}{x : a \mid \Delta \vdash x : a} \qquad \frac{\Gamma, x : a \mid \Delta \vdash t : b}{\Gamma \mid \Delta \vdash \lambda x^a.t : a \multimap b}$$

$$\frac{\Gamma \mid \Delta \vdash s : a \multimap b \qquad \Gamma' \mid \Delta \vdash t : a}{\Gamma, \Gamma' \mid \Delta \vdash \langle s \rangle t : b}$$

Non-linear rules:
$$\frac{}{\Gamma \mid \Delta, x : \underline{b} \vdash x : \underline{b}} \qquad \frac{\Gamma \mid \Delta, x : \underline{a} \vdash t : b}{\Gamma \mid \Delta \vdash \lambda x^{\underline{a}}.t : \underline{a} \to b}$$

$$\frac{\Gamma \mid \Delta \vdash s : a \to b \qquad \cdot \mid \Delta \vdash t : a}{\Gamma \mid \Delta \vdash (s)t : b}$$

Linear-non-linear rule:
$$\frac{\Gamma, x : a \mid \Delta \vdash t : b}{\Gamma \mid \Delta, x : \underline{a} \vdash t : b}$$

A term calculus for Linear-non-linear Logic

(Benton-Bierman-de Paiva-Hyland 1993, Barber 1996)

$$\boxed{\underbrace{x_1 : a_1, \ldots, x_\ell : a_\ell}_{\text{Linear}} \mid \underbrace{\underline{y}_1 : \underline{b}_1, \ldots, \underline{y}_n : \underline{b}_n}_{\text{Non-Linear}} \vdash t : c}$$

Linear rules: $$\frac{}{x : a \mid \Delta \vdash x : a} \qquad \frac{\Gamma, x : a \mid \Delta \vdash t : b}{\Gamma \mid \Delta \vdash \lambda x^a.t : a \multimap b}$$

$$\boxed{\frac{\Gamma \mid \Delta \vdash s : a \multimap b \qquad \Gamma' \mid \Delta \vdash t : a}{\Gamma, \Gamma' \mid \Delta \vdash \langle s \rangle t : b}}$$

Non-linear rules: $$\frac{}{\Gamma \mid \Delta, x : \underline{b} \vdash x : \underline{b}} \qquad \frac{\Gamma \mid \Delta, x : \underline{a} \vdash t : b}{\Gamma \mid \Delta \vdash \lambda x^{\underline{a}}.t : \underline{a} \to b}$$

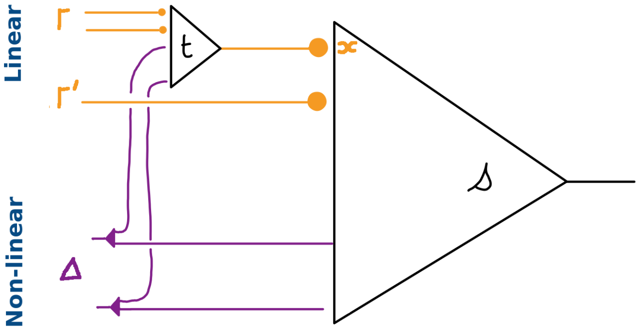
$$\frac{\Gamma \mid \Delta \vdash s : a \to b \qquad \cdot \mid \Delta \vdash t : a}{\Gamma \mid \Delta \vdash (s)t : b}$$

Linear-non-linear rule: $$\frac{\Gamma, x : a \mid \Delta \vdash t : b}{\Gamma \mid \Delta, x : \underline{a} \vdash t : b}$$

A term calculus for Linear-non-linear Logic

(Benton-Bierman-de Paiva-Hyland 1993, Barber 1996)

$$\frac{\Gamma \mid \Delta \vdash t \qquad x, \Gamma' \mid \Delta \vdash s}{\Gamma, \Gamma' \mid \Delta \vdash s[t/x]}$$



A term calculus for Linear-non-linear Logic

(Benton-Bierman-de Paiva-Hyland 1993, Barber 1996)

$$\boxed{\underbrace{x_1 : a_1, \ldots, x_\ell : a_\ell}_{\text{Linear}} \mid \underbrace{\underline{y}_1 : \underline{b}_1, \ldots, \underline{y}_n : \underline{b}_n}_{\text{Non-Linear}} \vdash t : c}$$

Linear rules:
$$\frac{}{x : a \mid \Delta \vdash x : a} \qquad \frac{\Gamma, x : a \mid \Delta \vdash t : b}{\Gamma \mid \Delta \vdash \lambda x^a.t : a \multimap b}$$

$$\frac{\Gamma \mid \Delta \vdash s : a \multimap b \qquad \Gamma' \mid \Delta \vdash t : a}{\Gamma, \Gamma' \mid \Delta \vdash \langle s \rangle t : b}$$

Non-linear rules:
$$\frac{}{\Gamma \mid \Delta, x : \underline{b} \vdash x : \underline{b}} \qquad \frac{\Gamma \mid \Delta, x : \underline{a} \vdash t : b}{\Gamma \mid \Delta \vdash \lambda x^{\underline{a}}.t : \underline{a} \to b}$$

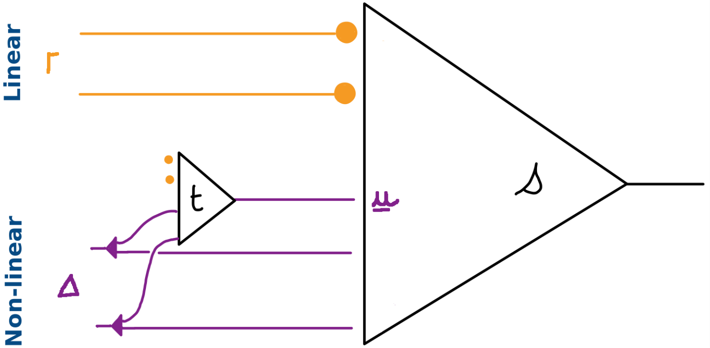
$$\boxed{\frac{\Gamma \mid \Delta \vdash s : a \to b \qquad \cdot \mid \Delta \vdash t : a}{\Gamma \mid \Delta \vdash (s)t : b}}$$

Linear-non-linear rule:
$$\frac{\Gamma,\ x : a \mid \Delta \vdash t : b}{\Gamma \mid \Delta,\ x : \underline{a} \vdash t : b}$$

A term calculus for Linear-non-linear Logic

(Benton-Bierman-de Paiva-Hyland 1993, Barber 1996)

$$\frac{\cdot \mid \Delta \vdash t \qquad \Gamma \mid \underline{u}, \Delta \vdash s}{\Gamma \mid \Delta \vdash s[t/\underline{u}]}$$



A term calculus for Linear-non-linear Logic

(Benton-Bierman-de Paiva-Hyland 1993, Barber 1996)

$$\underbrace{x_1 : a_1, \ldots, x_\ell : a_\ell}_{\text{Linear}} \mid \underbrace{\underline{y}_1 : \underline{b}_1, \ldots, \underline{y}_n : \underline{b}_n}_{\text{Non-Linear}} \vdash t : c$$

Linear rules:

$$\frac{}{x : a \mid \Delta \vdash x : a} \qquad \frac{\Gamma, x : a \mid \Delta \vdash t : b}{\Gamma \mid \Delta \vdash \lambda x^a.t : a \multimap b}$$

$$\frac{\Gamma \mid \Delta \vdash s : a \multimap b \qquad \Gamma' \mid \Delta \vdash t : a}{\Gamma, \Gamma' \mid \Delta \vdash \langle s \rangle t : b}$$

Non-linear rules:

$$\frac{}{\Gamma \mid \Delta, x : \underline{b} \vdash x : \underline{b}} \qquad \frac{\Gamma \mid \Delta, x : \underline{a} \vdash t : b}{\Gamma \mid \Delta \vdash \lambda x^{\underline{a}}.t : \underline{a} \to b}$$

$$\frac{\Gamma \mid \Delta \vdash s : a \to b \qquad \cdot \mid \Delta \vdash t : a}{\Gamma \mid \Delta \vdash (s)t : b}$$

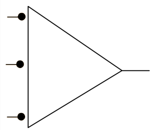
Linear-non-linear rule: $$\frac{\Gamma, x : a \mid \Delta \vdash t : b}{\Gamma \mid \Delta, x : \underline{a} \vdash t : b}$$

Multicategories

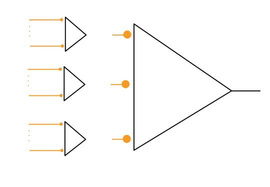
Profunctors and context 2-Monads

A Multicategory

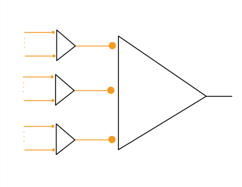
Operations and



Substitution



$\Rightarrow$



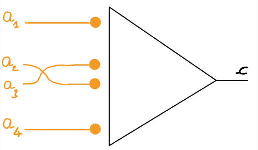
$\mathcal{T}A$ represent sequences of input types $\langle a_1, \ldots, a_n \rangle$

- Monad: Concatenate sequences of sequences $\mathcal{T}\mathcal{T}A \to \mathcal{T}A$
- Extend: $\mathcal{T}$ from inputs to operations

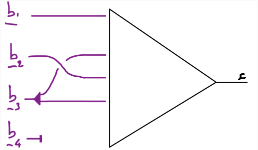
The multicategory $M : \mathcal{T}A \mathrel{\rightarrow\!\!\!\!\!\!+\;\;} A$ is given by the set of operations $M(\langle a_1, \ldots, a_n \rangle, b)$

How to build a mixed multicategory ?

Symmetric Multicategory

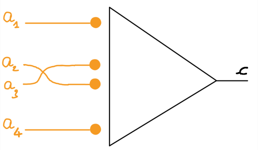


Cartesian Multicategory



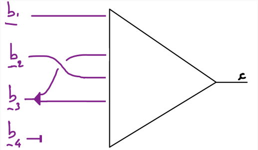
How to build a mixed multicategory ?

Symmetric Multicategory



$\mathcal{L}(X)$ free Symetric Monoidal Categories over X: $\langle a_1, \ldots, a_\ell \rangle$

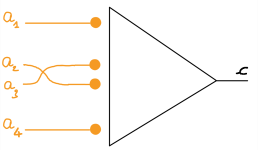
Cartesian Multicategory



$\mathcal{M}(X)$ free category with finite products over X: $\langle \underline{b}_1, \ldots, \underline{b}_n \rangle$

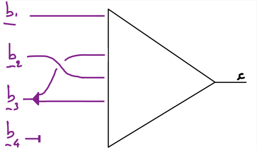
How to build a mixed multicategory ?

Symmetric Multicategory



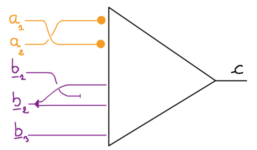
$\mathcal{L}(X)$ free Symetric Monoidal Categories over X: $\langle a_1, \ldots, a_\ell \rangle$

Cartesian Multicategory



$\mathcal{M}(X)$ free category with finite products over X: $\langle \underline{b}_1, \ldots, \underline{b}_n \rangle$

Mixed Linear-Non-Linear Multicategory



What is the mixed Linear-non-linear context monad ?

$$\mathcal{Q}(X) : \langle a_1, \ldots, a_\ell, \underline{b}_1, \ldots, \underline{b}_n \rangle$$

What is $\mathcal{Q}$-substitution?

A Colimit construction

To build the Linear-non-linear monad

colimits induced by a map in a category $\mathcal{K}$

If $\lambda : A \to B$ is a map in $\mathcal{K}$, then the induced colimit is
$$\begin{tikzcd} A \arrow[d, "\lambda"'] \arrow[dr, bend left, "k"] & \\ B \arrow[r, "\ell"'] & C \end{tikzcd}$$

- for any
$$\begin{tikzcd} A \arrow[d, "\lambda"'] \arrow[dr, bend left, "f"] & \\ B \arrow[r, "g"'] & D \end{tikzcd} = \begin{tikzcd} A \arrow[d, "\lambda"'] \arrow[dr, bend left, "k"] & & \\ B \arrow[r, "\ell"'] & C \arrow[r, dotted, "\exists ! r"] & D \end{tikzcd}$$

Colax colimits induced by a map in a 2-category $\mathcal{K}$

If $\lambda : A \to B$ is a map in $\mathcal{K}$, then the induced colax colimit is
$$\begin{tikzcd} A \arrow[d, "\lambda"'] \arrow[dr, bend left, "k"] \arrow[dr, phantom, "\Uparrow\alpha"] & \\ B \arrow[r, "\ell"'] & C \end{tikzcd}$$

- for any $\begin{tikzcd} A \arrow[d, "\lambda"'] \arrow[dr, bend left, "f"] \arrow[dr, phantom, "\Uparrow\phi"] & \\ B \arrow[r, "g"'] & D \end{tikzcd}$ $=$ $\begin{tikzcd} A \arrow[d, "\lambda"'] \arrow[dr, bend left, "k"] \arrow[dr, phantom, "\Uparrow\alpha"] & & \\ B \arrow[r, "\ell"'] & C \arrow[r, dotted, "\exists! r"] & D \end{tikzcd}$

Colax colimits induced by a map in a 2-category $\mathcal{K}$

If $\lambda : A \to B$ is a map in $\mathcal{K}$, then the induced colax colimit is
$$\begin{tikzcd} A \arrow[d, "\lambda"'] \arrow[dr, bend left, "k"] \\ B \arrow[r, "\ell"'] \arrow[ur, phantom, "\Nearrow\alpha"] & C \end{tikzcd}$$

There are two universal aspects for 1-cells and 2-cells

- for any $\begin{tikzcd} A \arrow[d, "\lambda"'] \arrow[dr, bend left, "f"] \\ B \arrow[r, "g"'] \arrow[ur, phantom, "\Nearrow\phi"] & D \end{tikzcd} = \begin{tikzcd} A \arrow[d, "\lambda"'] \arrow[dr, bend left, "k"] \\ B \arrow[r, "\ell"'] \arrow[ur, phantom, "\Nearrow\alpha"] & C \arrow[r, dotted, "\exists! r"] & D \end{tikzcd}$

- for any $\begin{tikzcd} A \arrow[d, "\lambda"'] \arrow[dr, bend left=40, "f'"] \arrow[dr, bend right, "f"'] \arrow[dr, phantom, "\Nearrow\rho"] \\ B \arrow[r, "g"'] \arrow[ur, phantom, "\Nearrow\phi"] & D \end{tikzcd} = \begin{tikzcd} A \arrow[d, "\lambda"'] \arrow[dr, bend left=40, "f'"] \\ B \arrow[r, "g"'] \arrow[r, bend left, "g'"] \arrow[r, phantom, "\sigma\Uparrow"] \arrow[ur, phantom, "\phi'\Nearrow"] & D \end{tikzcd}$, $\exists!\, r \overset{\tau}{\Rightarrow} r'$ s.t.

$$\begin{tikzcd} A \arrow[r, bend left, "f'"] \arrow[r, bend right, "f"'] \arrow[r, phantom, "\rho\Uparrow"] & D \end{tikzcd} = \begin{tikzcd} A \arrow[r, "k"] & C \arrow[r, bend left, "r'"] \arrow[r, bend right, "r"'] \arrow[r, phantom, "\tau\Uparrow"] & D \end{tikzcd} \qquad \begin{tikzcd} B \arrow[r, bend left, "g'"] \arrow[r, bend right, "g"'] \arrow[r, phantom, "\sigma\Uparrow"] & D \end{tikzcd} = \begin{tikzcd} B \arrow[r, "\ell"] & C \arrow[r, bend left, "r'"] \arrow[r, bend right, "r"'] \arrow[r, phantom, "\tau\Uparrow"] & D \end{tikzcd}$$

Mixing Linear and Non-Linear contexts via a colimit

Remark:

- Every category with products is a symmetric strict monoidal category

$$\lambda_X : \langle a_1, \ldots, a_\ell \rangle \in \mathcal{L}X \mapsto \langle \underline{a}_1, \ldots, \underline{a}_\ell \rangle \in \mathcal{M}X$$

Solution: Colax Colimit over λ in the 2-category of SymStMonCat

$$\begin{tikzcd}
\mathcal{L}X \arrow[d, "\lambda_X"'] \arrow[dr, bend left, "k_X"] & \\
\mathcal{M}X \arrow[r, "\ell_X"'] \arrow[ur, phantom, "\Nearrow \alpha_X"] & \mathcal{Q}X
\end{tikzcd}$$

Mixing Linear and Non-Linear contexts via a colimit

Remark:

- Every category with products is a symmetric strict monoidal category

$$\lambda_X : \langle a_1, \ldots, a_\ell \rangle \in \mathcal{L}X \mapsto \langle \underline{a}_1, \ldots, \underline{a}_\ell \rangle \in \mathcal{M}X$$

Wanted

- $\mathcal{Q}X$ is in SymStMonCat and objects are $\langle a_1, \ldots, a_\ell \mid \underline{b}_1, \ldots, \underline{b}_n \rangle$

Solution: Colax Colimit over λ in the 2-category of SymStMonCat

$$\begin{tikzcd}
\mathcal{L}X \arrow[d, "\lambda_X"'] \arrow[dr, bend left, "k_X"] & \\
\mathcal{M}X \arrow[r, "\ell_X"'] \arrow[ur, phantom, "\Nearrow \alpha_X"] & \mathcal{Q}X
\end{tikzcd}$$

Mixing Linear and Non-Linear contexts via a colimit

Remark:

- Every category with products is a symmetric strict monoidal category

$$\lambda_X : \langle a_1, \ldots, a_\ell \rangle \in \mathcal{L}X \mapsto \langle \underline{a}_1, \ldots, \underline{a}_\ell \rangle \in \mathcal{M}X$$

Wanted

- $\mathcal{Q}X$ is in SymStMonCat and objects are $\langle a_1, \ldots, a_\ell \mid \underline{b}_1, \ldots, \underline{b}_n \rangle$
- $\mathcal{Q}X$ contains Linear objects

$$k_X : \langle a_1, \ldots, a_\ell \rangle \in \mathcal{L}X \mapsto \langle a_1, \ldots, a_\ell \mid \cdot \ \rangle \in \mathcal{Q}X$$

Solution: Colax Colimit over λ in the 2-category of SymStMonCat

$$\begin{tikzcd}
\mathcal{L}X \arrow[d, "\lambda_X"'] \arrow[dr, bend left, "k_X"] & \\
\mathcal{M}X \arrow[r, "\ell_X"'] \arrow[ur, phantom, "\Nearrow \alpha_X"] & \mathcal{Q}X
\end{tikzcd}$$

Mixing Linear and Non-Linear contexts via a colimit

Remark:

- Every category with products is a symmetric strict monoidal category

$$\lambda_X : \langle a_1, \ldots, a_\ell \rangle \in \mathcal{L}X \mapsto \langle \underline{a}_1, \ldots, \underline{a}_\ell \rangle \in \mathcal{M}X$$

Wanted

- $\mathcal{Q}X$ is in SymStMonCat and objects are $\langle a_1, \ldots, a_\ell \mid \underline{b}_1, \ldots, \underline{b}_n \rangle$
- $\mathcal{Q}X$ contains Linear objects

$$k_X : \langle a_1, \ldots, a_\ell \rangle \in \mathcal{L}X \mapsto \langle a_1, \ldots, a_\ell \mid \cdot \rangle \in \mathcal{Q}X$$

- $\mathcal{Q}X$ contains Non-Linear ones

$$\ell_X : \langle \underline{b}_1, \ldots, \underline{b}_n \rangle \in \mathcal{M}X \mapsto \langle \cdot \mid \underline{b}_1, \ldots, \underline{b}_n \rangle \in \mathcal{Q}X$$

Solution: Colax Colimit over λ in the 2-category of SymStMonCat

$$\begin{tikzcd}
\mathcal{L}X \arrow[d, "\lambda_X"'] \arrow[dr, bend left, "k_X"] & \\
\mathcal{M}X \arrow[r, "\ell_X"'] \arrow[ur, phantom, "\Nearrow \alpha_X"] & \mathcal{Q}X
\end{tikzcd}$$

Mixing Linear and Non-Linear contexts via a colimit

Remark:

- Every category with products is a symmetric strict monoidal category

$$\lambda_X : \langle a_1, \ldots, a_\ell \rangle \in \mathcal{L}X \mapsto \langle \underline{a}_1, \ldots, \underline{a}_\ell \rangle \in \mathcal{M}X$$

Wanted

- $\mathcal{Q}X$ is in SymStMonCat and objects are $\langle a_1, \ldots, a_\ell \mid \underline{b}_1, \ldots, \underline{b}_n \rangle$
- $\mathcal{Q}X$ contains Linear objects

$$k_X : \langle a_1, \ldots, a_\ell \rangle \in \mathcal{L}X \mapsto \langle a_1, \ldots, a_\ell \mid \cdot \rangle \in \mathcal{Q}X$$

- $\mathcal{Q}X$ contains Non-Linear ones

$$\ell_X : \langle \underline{b}_1, \ldots, \underline{b}_n \rangle \in \mathcal{M}X \mapsto \langle \cdot \mid \underline{b}_1, \ldots, \underline{b}_n \rangle \in \mathcal{Q}X$$

Solution: Colax Colimit over λ in the 2-category of SymStMonCat

$$\begin{tikzcd}
\mathcal{L}X \arrow[d, "\lambda_X"'] \arrow[dr, bend left, "k_X"] & \\
\mathcal{M}X \arrow[r, "\ell_X"'] \arrow[ur, phantom, "\Nearrow \alpha_X"] & \mathcal{Q}X
\end{tikzcd}$$

$$\alpha_{X, \langle a_1, \ldots, a_\ell \rangle} \in \mathcal{Q}X(\langle \cdot \mid \underline{a}_1, \ldots, \underline{a}_\ell \rangle \,,\, \langle a_1, \ldots, a_\ell \mid \cdot \rangle)$$

Mixing Linear and Non-Linear contexts via a colimit

Remark:

- Every category with products is a symmetric strict monoidal category

$$\lambda_X : \langle a_1, \ldots, a_\ell \rangle \in \mathcal{L}X \mapsto \langle \underline{a}_1, \ldots, \underline{a}_\ell \rangle \in \mathcal{M}X$$

Wanted

- $\mathcal{Q}X$ is in SymStMonCat and objects are $\langle a_1, \ldots, a_\ell \mid \underline{b}_1, \ldots, \underline{b}_n \rangle$
- $\mathcal{Q}X$ contains Linear objects

$$k_X : \langle a_1, \ldots, a_\ell \rangle \in \mathcal{L}X \mapsto \langle a_1, \ldots, a_\ell \mid \cdot \rangle \in \mathcal{Q}X$$

- $\mathcal{Q}X$ contains Non-Linear ones

$$\ell_X : \langle \underline{b}_1, \ldots, \underline{b}_n \rangle \in \mathcal{M}X \mapsto \langle \cdot \mid \underline{b}_1, \ldots, \underline{b}_n \rangle \in \mathcal{Q}X$$

Solution: Colax Colimit over λ in the 2-category of SymStMonCat

$$\begin{tikzcd} \mathcal{L}X \arrow[d, "\lambda_X"'] \arrow[dr, bend left, "k_X"] & \\ \mathcal{M}X \arrow[r, "\ell_X"'] \arrow[ur, phantom, "\Nearrow \alpha_X"] & \mathcal{Q}X \end{tikzcd}$$

$$\alpha_{X, \langle a_1, \ldots, a_\ell \rangle} \in \mathcal{Q}X(\langle \cdot \mid \underline{a}_1, \ldots, \underline{a}_\ell \rangle \ , \ \langle a_1, \ldots, a_\ell \mid \cdot \rangle)$$

$$\boxed{\dfrac{x_1 : a_1, \ldots, x_\ell : a_\ell \mid \cdot \vdash t : b}{\cdot \mid x_1 : \underline{a}_1, \ldots, x_\ell : \underline{a}_\ell \vdash t : b}}$$

A Colimit construction

$\mathcal{Q}$ is a 2-monad on Cat and $\mathcal{Q}$-algebras

A general colax colimit construction on 2-monads

Let $\lambda : \mathcal{L} \to \mathcal{M}$ a map of 2-monad on **Cat**.
If $\mathcal{L}$-algebras has colimits, then the colimit is

$$\begin{tikzcd} \mathcal{L}X \arrow[d, "\lambda_X"'] \arrow[dr, bend left, "k_X"] & \\ \mathcal{M}X \arrow[r, "\ell_X"'] \arrow[ur, phantom, "\Uparrow \alpha_X"] & \mathcal{Q}X \end{tikzcd}$$

Theorem

- $\mathcal{Q}$ is a 2-monad on **Cat**.
- Characterization of $\mathcal{Q}$-algebras

On going work (Galal-Hyland)

- $\mathcal{Q}$-multicategories enjoy correct substitution

Take home:

- **Semantics** influence design of Programming Languages
- **Substitution** is subtile and important
- New combination of 2-monads through a 2-**colimit**
- A first step towards understanding substitution with **mixed** variables through multicategories

The linear-non-linear substitution 2-monad (Hyland, T. 2020)

https://arxiv.org/abs/2005.09559

"The purpose of abstraction is *not* to be vague, but to create a new semantic level in which one can be absolutely precise".

(E. Dijkstra, The Humble Programmer, ACM Turing Lecture, 1972)