Lambda calculs et catégories

Paul-André Melliès

Master Parisien de Recherche en Informatique

Ecole Normale Supérieure

Plan de la séance

1 – Lambda-calcul typé du second ordre

2 – Une sémantique opérationnelle

3 – Une topologie

4 – Les variétés comme type sémantique

5 – Théorème fondamental

6 – Application: théorème de normalisation

Part One

Second order lambda calculus

The expressive power of polymorphism

The simply-typed λ-calculus

The simple types A, B are constructed by the grammar:

$$A, B \ ::= \ \alpha \mid A \Rightarrow B.$$

A **typing context** Γ is a finite sequence

$$\Gamma = (x_1 : A_1, ..., x_n : A_n)$$

where each x_i is a variable and each A_i is a simple type.

A **sequent** is a triple

$$x_1 : A_1, ..., x_n : A_n \vdash P : B$$

where

$$x_1 : A_1, ..., x_n : A_n$$

is a typing context, P is a λ-term and B is a simple type.

The simply-typed λ-calculus

Variable

$$\frac{}{x : A \vdash x : A}$$

Abstraction

$$\frac{\Gamma, x : A \vdash P : B}{\Gamma \vdash \lambda x.P : A \Rightarrow B}$$

Application

$$\frac{\Gamma \vdash P : A \Rightarrow B \qquad \Delta \vdash Q : A}{\Gamma, \Delta \vdash PQ : B}$$

Weakening

$$\frac{\Gamma \vdash P : B}{\Gamma, x : A \vdash P : B}$$

Contraction

$$\frac{\Gamma, x : A, y : A \vdash P : B}{\Gamma, z : A \vdash P[x, y \leftarrow z] : B}$$

Exchange

$$\frac{\Gamma, x : A, y : B, \Delta \vdash P : C}{\Gamma, y : B, x : A, \Delta \vdash P : C}$$

Subject reduction

A λ-term P is simply typed when there exists a sequent

$$\Gamma \quad \vdash \quad P : A$$

which may be obtained by a derivation tree.

One establishes that the set of simply typed λ-terms is closed under β-réduction:

Subject Reduction:

If $\Gamma \vdash P : A$ and $P \longrightarrow_\beta Q$, then $\Gamma \vdash Q : A$.

Girard 1972: the second-order λ-calculus

The simple types are extended by second-order quantification on the type variables

$$A, B \ ::= \ \alpha \ \mid A \Rightarrow B \ \mid A \times B \ \mid \forall \alpha . A$$

A typing context Γ is a finite list constructed by the grammar

$$\Gamma \ = \ \texttt{nil} \ \mid \ \Gamma, x : A \ \mid \ \Gamma, \alpha : Type$$

where

- $\texttt{nil}$ is the empty list
- x is a term variable and A is a type
- α is a type variable and $Type$ is a symbol.

Two families of sequents

A type sequent is a triple

$$\alpha_1 : Type\,,\; \ldots\,,\; \alpha_n : Type \quad \vdash \quad A : Type$$

where

- $\alpha_1 : Type\,,\; \ldots\,,\; \alpha_n : Type$ is a context of type variables
- A is a type.

Type derivation

Variable

$$\frac{}{\alpha : Type \;\vdash\; \alpha : Type}$$

Implication

$$\frac{\Gamma \;\vdash\; A : Type \qquad \Delta \;\vdash\; B : Type}{\Gamma , \Delta \;\vdash\; A \Rightarrow B : Type}$$

Quantification

$$\frac{\Gamma , \alpha : Type \;\vdash\; A : Type}{\Gamma \;\vdash\; \forall\alpha.A : Type}$$

Weakening

$$\frac{\Gamma \;\vdash\; A : Type}{\Gamma , \alpha : Type \;\vdash\; A : Type}$$

Contraction

$$\frac{\Gamma , \alpha : Type , \beta : Type , \Delta \;\vdash\; A : Type}{\Gamma , \gamma : Type , \Delta \;\vdash\; A[\alpha,\beta \leftarrow \gamma] : Type}$$

Exchange

$$\frac{\Gamma , \alpha : Type , \beta : Type , \Delta \;\vdash\; A : Type}{\Gamma , \beta : Type , \alpha : Type , \Delta \;\vdash\; A : Type}$$

Term derivation

Variable

$$\frac{\Gamma \vdash A : Type}{\Gamma, x : A \vdash x : A}$$

Abstraction [term]

$$\frac{\Gamma, x : A \vdash P : B}{\Gamma \vdash \lambda x.P : A \Rightarrow B}$$

Application [term]

$$\frac{\Gamma \vdash P : A \Rightarrow B \qquad \Delta \vdash Q : A}{\Gamma, \Delta \vdash PQ : B}$$

Abstraction [type]

$$\frac{\Gamma, \alpha : Type \vdash P : A}{\Gamma \vdash P : \forall \alpha.A}$$

Application [type]

$$\frac{\Gamma \vdash P : \forall \alpha.A \qquad \Delta \vdash B : Type}{\Gamma , \Delta \vdash P : A[\alpha := B]}$$

Term derivation

Weakening [term]

$$\frac{\Gamma \vdash P : A \qquad \Delta \vdash B : Type}{\Gamma\,,\ \Delta\,,\ x : A \vdash P : B}$$

Weakening [type]

$$\frac{\Gamma \ \vdash\ P : A}{\Gamma,\ \alpha\ :\ Type\ \vdash\ P : A}$$

Contraction [term]

$$\frac{\Gamma\,,\ x : A\,,\ y : A\,,\ \Delta\ \vdash\ P : B}{\Gamma\,,\ z : A\,,\ \Delta\ \vdash\ P[x, y \leftarrow z] : B}$$

Contraction [type]

$$\frac{\Gamma\,,\ \alpha\ :\ Type\,,\ \beta\ :\ Type\,,\ \Delta\ \vdash\ P : A}{\Gamma\,,\ \gamma\ :\ Type\,,\ \Delta[\alpha, \beta \leftarrow \gamma]\ \vdash\ P : A[\alpha, \beta \leftarrow \gamma]}$$

Term derivation

Exchange [term-term]

$$\frac{\Gamma, x : A, y : B, \Delta \;\vdash\; P : C}{\Gamma, y : B, x : A, \Delta \;\vdash\; P : C}$$

Exchange [type-type]

$$\frac{\Gamma, \alpha : Type, \beta : Type, \Delta \;\vdash\; P : B}{\Gamma, \beta : Type, \alpha : Type, \Delta \;\vdash\; P : A}$$

Exchange [term-type]

$$\frac{\Gamma, x : A, \alpha : Type, \Delta \;\vdash\; P : B}{\Gamma, \alpha : Type, x : A, \Delta \;\vdash\; P : B}$$

Exchange [type-term]

$$\frac{\Gamma, \alpha : Type, x : A, \Delta \;\vdash\; P : B}{\Gamma, x : A, \alpha : Type, \Delta \;\vdash\; P : B}$$

The Exchange [type-term] rule is allowed only when α is not free in A.

Term derivation (calculus with pairs)

Pair

$$\frac{\Gamma \vdash P : A \qquad \Gamma \vdash Q : B}{\Gamma \vdash \langle P, Q \rangle : A \times B}$$

Left projection

$$\frac{\Gamma \vdash P : A \times B}{\Gamma \vdash \mathtt{fst}(P) : A}$$

Right projection

$$\frac{\Gamma \vdash P : A \times B}{\Gamma \vdash \mathtt{snd}(P) : B}$$

Unit

$$\frac{}{\Gamma \vdash * : 1}$$

Properties of the second-order λ-calculus

A λ-term P is typed in second-order λ-calculus when there exists a typing context Γ and a type A such that:

$$\Gamma \quad \vdash \quad P : A$$

The set of simply-typed λ-terms is closed under β-reduction:

Subject reduction: If $\Gamma \vdash P : A$ and $P \longrightarrow_\beta Q$, then $\Gamma \vdash Q : A$.

A λ-term P is strongly normalizing when there exists no infinite path

$$P \longrightarrow_\beta P_1 \longrightarrow_\beta P_2 \longrightarrow_\beta \cdots \longrightarrow_\beta P_n \longrightarrow_\beta \cdots$$

of β-reductions.

Strong normalisation: Every λ-term P typed in the second-order λ-calculus is strongly normalising.

Curry-Howard

Second order intuitionistic logic

Variable

$$\frac{\Gamma \vdash A : Type}{\Gamma, \quad A \vdash \quad A}$$

Abstraction [term]

$$\frac{\Gamma, \quad A \vdash \quad B}{\Gamma \vdash \quad A \Rightarrow B}$$

Application [term]

$$\frac{\Gamma \vdash \quad A \Rightarrow B \qquad \Delta \vdash \quad A}{\Gamma, \Delta \vdash \quad B}$$

Abstraction [type]

$$\frac{\Gamma, \alpha : Type \ \vdash \quad A}{\Gamma \vdash \quad \forall \alpha . A}$$

Application [type]

$$\frac{\Gamma \ \vdash \quad \forall \alpha . A \qquad \Delta \ \vdash \ B : Type}{\Gamma \ , \ \Delta \ \vdash \quad A[\alpha := B]}$$

Curry-Howard

Second order λ-calculus

Variable

$$\frac{\Gamma \vdash A : Type}{\Gamma, x : A \vdash x : A}$$

Abstraction [term]

$$\frac{\Gamma, x : A \vdash P : B}{\Gamma \vdash \lambda x.P : A \Rightarrow B}$$

Application [term]

$$\frac{\Gamma \vdash P : A \Rightarrow B \qquad \Delta \vdash Q : A}{\Gamma, \Delta \vdash PQ : B}$$

Abstraction [type]

$$\frac{\Gamma, \alpha : Type \ \vdash P : A}{\Gamma \vdash P : \forall \alpha.A}$$

Application [type]

$$\frac{\Gamma \ \vdash \ P : \forall \alpha.A \qquad \Delta \ \vdash \ B : Type}{\Gamma \ , \ \Delta \ \vdash \ P : A[\alpha := B]}$$

Encoding of the natural numbers

The type Nat of the natural numbers is defined as

$$Nat \quad = \quad \forall\alpha \quad . \quad (\alpha \Rightarrow \alpha) \quad \Rightarrow \quad (\alpha \Rightarrow \alpha)$$

Exercise: show that every Church numeral n is of type Nat:

$$\vdash \quad \lambda f.\lambda a.\ \underbrace{f \cdots f}_{n \text{ times}}\ (a) \quad : \quad \forall\alpha.(\alpha \Rightarrow \alpha) \Rightarrow (\alpha \Rightarrow \alpha)$$

Encoding of the finite lists

The type $List$ of the finite lists of elements of A is defined as

$$List(A) \quad = \quad \forall\alpha \quad . \quad (\, A \Rightarrow \alpha \Rightarrow \alpha \,) \quad \Rightarrow \quad (\, \alpha \Rightarrow \alpha \,)$$

The list $[a_1, a_2, \cdots, a_n]$ is encoded as

$$[a_1, \cdots, a_n] \quad = \quad \lambda f.\lambda x.f a_1 (f a_2 (\cdots f a_n x) \cdots)$$

while the empty list is encoded as

$$nil \quad = \quad \lambda f.\lambda x.x$$

Property. The encoding of a finite list is of the expected type:

$$\vdash \quad [a_1, \cdots, a_n] \quad : \quad List(A)$$

Concatenation

The λ-term

$$Append \quad = \quad \lambda list_1.\ \lambda list_2.\ \lambda f.\ \lambda x.\ list_1\ f\ (list_2\ f\ x)$$

appends two finite lists.

Exercise: check that the λ-term Append has the type

$$\vdash \quad Append \quad : \quad \forall\gamma \quad . \quad List(\gamma) \quad \Rightarrow \quad List(\gamma) \quad \Rightarrow \quad List(\gamma)$$

and the expected behaviour:

$$Append[a_1,\cdots,a_k][a_{k+1},\cdots a_n] \quad \longrightarrow_\beta \quad \cdots \quad \longrightarrow_\beta \quad [a_1,\cdots,a_n]$$

Show that the number of β-réductions does not depend on the size of the lists.

Map

Suppose given a λ-term h of type

$$h \quad : \quad A \Rightarrow B$$

Then, the λ-term

$$Map \quad = \quad \lambda h.\ \lambda list.\ \lambda f.\ \lambda x.\ list\,(\lambda a.f(ha))\,x$$

transforms every list

$$[a_1, \cdots, a_n]$$

of elements of A in the list

$$[ha_1, \cdots, ha_n]$$

of elements of B.

Map

Exercise: show that the λ-term Map has the type

$$\vdash \quad Map \quad : \quad \forall\alpha \;.\; \forall\beta \;.\; (\alpha \;\Rightarrow\; \beta) \;\Rightarrow\; (List(\alpha) \;\Rightarrow\; List(\beta))$$

and the expected behaviour

$$Map\;\; h\;\; [a_1, \cdots, a_n] \quad \longrightarrow_\beta \quad \cdots \quad \longrightarrow_\beta \quad [ha_1, \cdots, ha_n]$$

Multiplication

Every λ-term a of type A induces the λ-term

$$Map(\lambda b.\langle a, b\rangle) \quad : \quad List(B) \quad \Rightarrow \quad List(A \times B)$$

Multiplication is encoded as the λ-term

$$Mult = \lambda list_1.\lambda list_2.\ list_1(\lambda a.\lambda list.Append(\underline{Map(\lambda b.\langle a, b\rangle)(list_2)}, list))(\lambda f.\lambda x.x)$$

Exercise: show that $Mult$ has the type

$$\vdash \quad Mult \quad : \quad \forall\alpha\ .\ \forall\beta\ .\quad List(\alpha) \times List(\beta) \quad \Rightarrow \quad List(\alpha \times \beta)$$

et le comportement suivant:

$$Mult[a_1, \cdots, a_m][b_1, \cdots b_n] \quad \longrightarrow_\beta \quad \cdots \quad \longrightarrow_\beta \quad [(a_1, b_1), (a_1, b_2), \cdots, (a_m, b_n)]$$

Exponentiation

We start from the λ-term

$$\vdash \quad Head \quad = \quad \lambda a.\lambda list.\lambda f.\lambda x.\ fa(list\, f\, x) \quad : \quad \forall\alpha\ .\ \alpha \Rightarrow List(\alpha) \Rightarrow List(\alpha)$$

which adds an element of type α to a list of type $List(\alpha)$.

The λ-term Exp is then defined as follows:

$$\lambda list_1.\lambda list_2.\ list_1\{\lambda a.\lambda list.Map\underline{(\lambda b.\lambda list'.Head\langle a,b\rangle(list'))}\ \underline{(Mult(list_2, list)}\}\{[nil]\}$$

where

$$\alpha : Type, \beta : Type \quad \vdash \quad [nil] \quad = \quad \lambda f.\lambda x.f(\lambda g.\lambda y.y)x \quad : \quad List(List(\alpha \times \beta))$$

Exponentiation

Exercise: show that Exp has the type

$$\vdash \quad \mathit{Exp} \quad : \quad \forall\alpha \;.\; \forall\beta \;.\quad \mathit{List}(\alpha) \times \mathit{List}(\beta) \quad \Rightarrow \quad \mathit{List}(\mathit{List}(\alpha \times \beta))$$

and constructs a list of length q to the power p where

- p is the length of the list list_1 in $\mathit{List}(\alpha)$
- q is the length of the list list_2 in $\mathit{List}(\beta)$.

Encoding of the binary trees

The type $BinTree$ of binary trees with leaves of type A is defined as

$$BinTree(A) \quad = \quad \forall\alpha \quad . \quad (\,\alpha \times \alpha \Rightarrow \alpha\,) \quad \Rightarrow \quad (\,A \Rightarrow \alpha \;\Rightarrow \alpha\,)$$

Exercise: define the λ-term associated to a binary tree and construct the λ-term

$$\vdash \quad Flatten \quad : \quad \forall\alpha.\; BinTree(\alpha) \quad \Rightarrow \quad List(\alpha)$$

which transforms every binary tree in the list of its leaves, ordered from left to right.

Part Two

An operational semantics

Terms and evaluation contexts

Second-order calculus with boolean tests

In order to build a programming language, the syntax of second-order λ-calculus may be extended with a type

$$Bool$$

two constants

$$\mathtt{true} \qquad\qquad \mathtt{false}$$

as well as the term constructor

$$\mathtt{if}\ M\ \mathtt{then}\ P\ \mathtt{else}\ Q$$

and the three typing rules:

$$\frac{}{\vdash \mathtt{true} : Bool} \qquad\qquad \frac{}{\vdash \mathtt{false} : Bool}$$

$$\frac{\Gamma \vdash M : Bool \qquad \Delta \vdash P : A \qquad \Delta \vdash Q : A}{\Gamma, \Delta \ \vdash\ \mathtt{if}\ M\ \mathtt{then}\ P\ \mathtt{else}\ Q : A}$$

Goal: show that every typed term has a normal form

The terms

The **untyped** λ-terms extended with **pairs** and **boolean tests**.

P	$::=$	x	variable
	$\mid$	$\lambda x.P$	abstraction
	$\mid$	$P\,Q$	application
	$\mid$	(P, Q)	pair
	$\mid$	$\mathtt{fst}(P)$	first projection
	$\mid$	$\mathtt{snd}(P)$	second projection
	$\mid$	$\mathtt{if}\ M\ \mathtt{then}\ P\ \mathtt{else}\ Q$	boolean test
	$\mid$	$\mathtt{true}$	true constant
	$\mid$	$\mathtt{false}$	false constant

Evaluation contexts

The evaluation contexts are **stacks** or **finite lists** of operations:

$$
\begin{array}{lcll}
\mathrm{E} & ::= & \texttt{nil} & \text{empty context} \\
 & \mid & P \cdot \mathrm{E} & \text{application} \\
 & \mid & \texttt{fst} \cdot \mathrm{E} & \text{first projection} \\
 & \mid & \texttt{snd} \cdot \mathrm{E} & \text{second projection} \\
 & \mid & (\texttt{if}\ P, Q) \cdot \mathrm{E} & \text{boolean test}
\end{array}
$$

The evaluation bracket

The combination of a term M and of a context E induces a term

$$\langle M \mid \mathrm{E} \rangle$$

defined as follows:

$$\begin{array}{lcl}
\langle M \mid \texttt{nil} \rangle & = & P \\
\langle M \mid P \cdot \mathrm{E} \rangle & = & \langle M\, P \mid \mathrm{E} \rangle \\
\langle M \mid \texttt{fst} \cdot \mathrm{E} \rangle & = & \langle \texttt{fst}(M) \mid \mathrm{E} \rangle \\
\langle M \mid \texttt{snd} \cdot \mathrm{E} \rangle & = & \langle \texttt{snd}(M) \mid \mathrm{E} \rangle \\
\langle M \mid (\texttt{if}\ P, Q) \cdot \mathrm{E} \rangle & = & \langle \texttt{if}\ M\ \texttt{then}\ P\ \texttt{else}\ Q \mid \mathrm{E} \rangle
\end{array}$$

The dynamics

Five rewriting rules:

- the β-rule:

$$\langle \lambda x.M \mid P \cdot \mathrm{E} \rangle \rightarrow \langle M[x := P] \mid \mathrm{E} \rangle$$

- two projection rules for the pair:

$$\begin{array}{rcl} \langle (P, Q) \mid \mathtt{fst} \cdot \mathrm{E} \rangle & \rightarrow & \langle P \mid \mathrm{E} \rangle \\ \langle (P, Q) \mid \mathtt{snd} \cdot \mathrm{E} \rangle & \rightarrow & \langle Q \mid \mathrm{E} \rangle \end{array}$$

- two rules for the boolean test:

$$\begin{array}{rcl} \langle \mathtt{true} \mid (\mathtt{if}\ P, Q) \cdot \mathrm{E} \rangle & \rightarrow & \langle P \mid \mathrm{E} \rangle \\ \langle \mathtt{false} \mid (\mathtt{if}\ P, Q) \cdot \mathrm{E} \rangle & \rightarrow & \langle Q \mid \mathrm{E} \rangle \end{array}$$

Sums

Possible to extend the language of **terms** with three operators

$$\mathtt{inl}(M) \qquad \mathtt{inr}(M) \qquad \mathtt{caseof}(M, P, Q)$$

the language of **contexts** with one operator

$$(\mathtt{case}\ P, Q) \cdot \mathrm{E}$$

Then, add the equation:

$$\langle M \mid (\mathtt{case}\ P, Q) \cdot \mathrm{E} \rangle \ = \ \langle \mathtt{caseof}(M, P, Q) \mid \mathrm{E} \rangle$$

and the two rewriting rules:

$$\langle \mathtt{inl}(M) \mid (\mathtt{case}\ P, Q) \cdot \mathrm{E} \rangle \ \longrightarrow \ \langle P\, M \mid \mathrm{E} \rangle$$
$$\langle \mathtt{inr}(M) \mid (\mathtt{case}\ P, Q) \cdot \mathrm{E} \rangle \ \longrightarrow \ \langle Q\, M \mid \mathrm{E} \rangle$$

Part Three

A pretopology

Well-typed terms cannot go wrong

Accepting set of terms

Definition. A set of terms

$$\perp\!\!\!\perp$$

is called **accepting** when it is closed by **anti-reduction**, that is:

for all M, N $\quad M \rightarrow N \quad$ and $\quad N \in \perp\!\!\!\perp$

implies

$$M \in \perp\!\!\!\perp$$

Illustration: the safe terms

Definition. A λ-term M is called **safe** when:

- ▷ the term M rewrites into the boolean constant $\texttt{true}$
- ▷ the term M rewrites into the boolean constant $\texttt{false}$
- ▷ or M produces an infinite rewriting path:

$$M \longrightarrow M_1 \longrightarrow M_2 \longrightarrow \cdots \longrightarrow M_n \longrightarrow \cdots$$

The existence of the infinite path ensures that there is no syntax error.

examples of safe terms: $(\lambda x.xx)(\lambda x.xx)$ $\qquad$ $\texttt{true}$ $\qquad$ $(\lambda x.x)\texttt{true}$

example of non safe terms: $\texttt{fst}(\texttt{true})$ $\qquad$ (*syntax error* !)

Illustration: the safe and normalising terms

Property. The set $\perp\!\!\!\perp_s$ of **safe** terms is accepting.

Property. The set $\perp\!\!\!\perp_n$ of **safe** and **normalising** terms is accepting.

Remark:
the set of **safe** and **strongly normalising** terms is not accepting...

Hence, the method below does not imply that the typed λ-terms are strongly normalizing in the second-order λ-calculus.

However, the method may be easily adapted to establish strong normalization, thanks to the notion of **reducibility candidate**.

Orthogonality

We suppose given an accepting set $\perp\!\!\!\perp$ of terms.

Definition.

A term M is called **orthogonal** to a context E, what we write

$$M \perp \mathrm{E}$$

when the term

$$\langle M \mid \mathrm{E} \rangle$$

is an element of $\perp\!\!\!\perp$.

Orthogonality

Suppose given a set S of evaluation contexts.

Notation: one writes

$$M \perp S$$

when

$$M \perp \mathrm{E}$$

for every context $\mathrm{E} \in S$.

Semantic varieties

Definition.

A **semantic variety** is a set of terms of the form

$$S^{\perp} \quad = \quad \{ \ M \ \mid \ M \perp S \ \}$$

for some set S of evaluation contexts.

Interpretation:

$S^{\perp}$ contains the terms which combine well with every context of S.

Closure operator

A **closure operator** on a set A is a function

$$\sigma \quad : \quad \wp(A) \quad \longrightarrow \quad \wp(A)$$

from the powerset $\wp(A)$ of A to itself, such that

- ▷ σ is monotone:

$$X \subseteq Y \quad \Rightarrow \quad \sigma(X) \subseteq \sigma(Y)$$

- ▷ σ is increasing:

$$X \subseteq \sigma(X)$$

- ▷ σ is idempotent:

$$\sigma(\sigma(X)) \quad = \quad \sigma(X)$$

Closure operators

Property.

A closure operator

$$\sigma \quad : \quad \wp(A) \quad \longrightarrow \quad \wp(A)$$

is entirely described by the set

$$fix(\sigma)$$

of its fixpoints, which is closed by arbitrary intersections:

- if $(X_i)_{i \in I}$ is a family of subsets of A, then

$$\forall i \in I, \quad X_i \in fix(\sigma) \qquad \Rightarrow \qquad \bigcap_{i \in I} X_i \in fix(\sigma)$$

Closure operators

Conversely,

Property. Every set $\mathcal{F}$ of subsets of A closed by arbitrary intersection

$$\forall i \in I, \quad X_i \in \mathcal{F} \qquad \Rightarrow \qquad \bigcap_{i \in I} X_i \in \mathcal{F}$$

defines a unique closure operator σ such that

$$\mathcal{F} \quad = \quad fix(\sigma).$$

Closure operators

Corolary.

The set $\mathcal{F}$ of semantic varieties is closed by intersection:

$$\bigcap_{i \in I} S_i^{\perp} \quad = \quad \Big(\bigcup_{i \in I} S_i \Big)^{\perp}$$

and thus defines a closure operator, computed by **biorthogonality**:

$$X \mapsto X^{\perp\perp}$$

Corolary. A set X of terms is a semantic variety if and only if

$$X = X^{\perp\perp}.$$

In algebraic geometry

λ-terms	$\sim$	points
evaluation contexts	$\sim$	polynomials
$\langle x \mid P \rangle$	$\sim$	$P(x)$
accepting set $\perp\!\!\!\perp$	$\sim$	equality to zero
$S^{\perp}$	$\sim$	$V(S)$

Definition. The **Zariski topology** has the varieties $V(S)$ as closed sets.

In algebraic geometry

This definition of the Zariski topology requires the following property:

Proposition. The union $X \cup Y$ of two varieties X and Y is a variety.

This property is a consequence of the equality:

$$S^{\perp} \cup T^{\perp} \quad = \quad (ST)^{\perp}$$

where

$$ST \quad = \quad \{ \quad P_1 P_2 \quad | \quad P_1 \in S, P_2 \in T \quad \}$$

The **product** $P_1 P_2$ of two polynomials P_1 et P_2 behaves as a disjunction:

$$\langle x \mid P_1 P_2 \rangle = 0 \quad \Longleftrightarrow \quad \langle x \mid P_1 \rangle = 0 \quad \text{or} \quad \langle x \mid P_2 \rangle = 0.$$

which tests P_1 and P_2 in a **parallel** and **independent** fashion.

By way of comparison...

Suppose given two semantic varieties X and Y.

The set $X \cup Y$ is a variety if and only if, for all terms M,

$$M \in (X \cup Y)^{\perp\perp} \quad \Rightarrow \quad M \in X \cup Y,$$

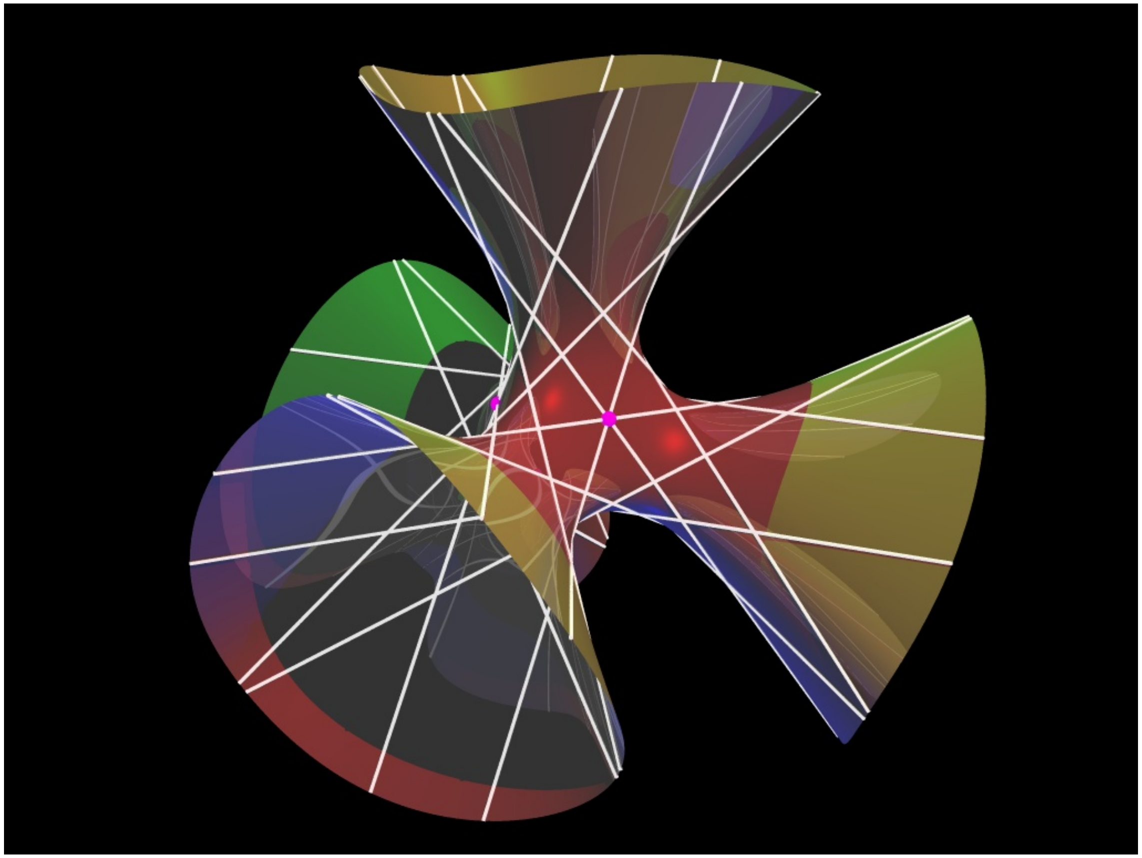
or equivalently:

$$M \perp X^{\perp} \cap Y^{\perp} \quad \Rightarrow \quad M \in X \cup Y,$$

This is true when there exits an operation $\otimes$ on contexts such that

$$\langle M \mid \mathrm{E}_1 \otimes \mathrm{E}_2 \rangle \in \perp\!\!\!\perp \quad \Longleftrightarrow \quad \langle M \mid \mathrm{E}_1 \rangle \in \perp\!\!\!\perp \ \ \textit{or} \ \ \langle M \mid \mathrm{E}_2 \rangle \in \perp\!\!\!\perp$$

Remark: Such an operator $\otimes$ does not exist in the λ-calculus, either for safety $\perp\!\!\!\perp_s$ or normalization $\perp\!\!\!\perp_n$.



Part fourth

Varieties as semantic types

A modern account of “reducibility candidates”

Arrow type

For every two sets X and Y of terms, one defines

$$X \Rightarrow Y$$

as the set of terms M such that:

$$\forall P \in X, \qquad MP \in Y.$$

Proposition: $X \Rightarrow Y$ is a variety when Y is a variety.

Arrow type

Proposition. $X \Rightarrow Y$ is a variety when Y is a variety $Y = S^{\perp}$

Proof.	$M \in X \Rightarrow Y$
$\Longleftrightarrow$	$\forall P \in X,\ \forall \mathrm{E} \in S, \quad MP \perp \mathrm{E}$
$\Longleftrightarrow$	$\forall P \in X,\ \forall \mathrm{E} \in S, \quad M \perp P \cdot \mathrm{E}$
$\Longleftrightarrow$	$M \perp X \cdot S$

where the set $X \cdot S$ is defined as

$$X \cdot S \quad = \quad \{ \quad P \cdot \mathrm{E} \quad | \quad P \in X, \quad \mathrm{E} \in S \quad \}$$

Product type

For every two sets X and Y of terms, one defines

$$X \times Y$$

as the set of terms

$$X \times Y \quad = \quad \left(\quad \{ \ \mathtt{fst} \cdot \mathrm{E} \mid \mathrm{E} \in S \ \} \quad \cup \quad \{ \ \mathtt{snd} \cdot \mathrm{E} \mid \mathrm{E} \in T \ \} \quad \right)^{\perp}$$

Product type

In the case where $\perp\!\!\!\perp = \perp\!\!\!\perp_n$ is the set of safe and normalizing terms,

Proposition. Suppose that

$$X = S^{\perp} \qquad\qquad Y = T^{\perp}$$

are two varieties defined by the sets $S \neq \emptyset$ et $T \neq \emptyset$.

In that case, $X \times Y$ contains the terms M which rewrite into a pair

$$(P, Q)$$

where $P \in X$ and $Q \in Y$.

Universal quantification

Given a family of varieties $(X_\alpha)_{\alpha \in W}$ parametrized by W, one defines the variety

$$\forall \alpha . X_\alpha \quad = \quad \bigcap_{\alpha \in W} X_\alpha$$

It should be noted that

$$\forall \alpha . X_\alpha \quad = \quad \Big(\bigcup_{\alpha \in W} S_\alpha \Big)^\perp$$

when

$$\forall \alpha \in W, \qquad X_\alpha = S_\alpha^\perp .$$

Part five

The fundamental theorem of realizability

A semantic proof of normalization

Negative interpretation

We suppose given an accepting set $\perp\!\!\!\perp$ of terms.

Definition. a **typing environment** ξ is a function which associates to every type variable $\alpha : Type$ a set of **evaluation contexts.**

Idea: every type

$$\alpha_1 : Type, \cdots, \alpha_n : Type \quad \vdash \quad A : Type$$

is interpreted as a function

$$\xi \quad \mapsto \quad \| A \|_\xi$$

which transports every environment ξ to a set $\| A \|_\xi$ of contexts.

Negative interpretation

The function

$$\|A\| \quad : \quad \xi \quad \mapsto \quad \|A\|_\xi$$

is defined by induction on the size of the type:

$$\|\alpha\|_\xi \quad = \quad \xi(\alpha)$$

$$\|A \Rightarrow B\|_\xi \quad = \quad \{ \ P \cdot \mathrm{E} \ \mid \ P \perp \|A\|_\xi \ \ \text{et} \ \ \mathrm{E} \in \|B\|_\xi \ \}$$

$$\|A \times B\|_\xi \quad = \quad \{ \ \mathtt{fst} \cdot \mathrm{E} \mid \mathrm{E} \in \|A\|_\xi \ \} \ \cup \ \{ \ \mathtt{snd} \cdot \mathrm{E} \mid \mathrm{E} \in \|B\|_\xi \ \}$$

$$\|\forall \alpha . A\|_\xi \quad = \quad \bigcup_{S \in \wp(\Pi)} \|A\|_{\xi + \alpha \mapsto S}$$

Notation: the set of contexts is denoted by Π.

Fundamental theorem

Theorem. Suppose that the term M is typed by the sequent:

$$\vdash \quad M \quad : \quad A$$

Then,

$$M \quad \perp \quad \| A \|_{\xi}$$

Corolary. The λ-term M is an element of the variety $\| A \|_{\xi}^{\perp}$.

Remark.
The interpretation $\| A \|_{\xi}$ does not depend on the environment ξ.

Application: the normalization theorem

Take the accepting set

$$\perp\!\!\!\perp \quad = \quad \perp\!\!\!\perp_n$$

consisting of the safe and normalizing terms.

First, one shows that the set of contexts

$$\| A \|_{\xi}$$

is nonempty for every closed type.

From this follows that the variety

$$\| A \|_{\xi}^{\perp}$$

contains only normalizing terms.

Corolary: every typed λ-terms is normalizing.

Application

One would like to understand the property of terms

$$\vdash \; M \; : \; \forall\alpha.\alpha \Rightarrow \alpha$$

For a given accepting set $\perp\!\!\!\perp$, one has

$$\| \forall\alpha.\alpha \Rightarrow \alpha \| \quad = \quad \{ \; P \cdot \mathrm{E} \mid P \perp \mathrm{E} \; \}$$

It is possible to extend the syntax of the pure λ-calculus with

- ▷ a generic term $\mathbf{P}$
- ▷ a generic context $\mathbf{E}$

Proposition.

The term $\langle M \mid \mathbf{P} \cdot \mathbf{E} \rangle$ rewrites into the term $\langle \mathbf{P} \mid \mathbf{E} \rangle$.

Proof. Define the accepting set

$$\perp\!\!\!\perp \quad = \quad \left\{ \quad \text{the terms which rewrite into } \langle \mathbf{P}, \mathbf{E} \rangle \quad \right\}$$

By definition of $\perp\!\!\!\perp$ one has that

$$\mathbf{P} \perp \mathbf{E}$$

From this, it follows that

$$M \perp \mathbf{P} \cdot \mathbf{E}$$

We conclude.

Classical logic

One extends the syntax of the pure λ-calculus with an operator

$$callcc$$

and a constant

$$k_{\mathrm{E}}$$

for every evaluation context (in a recursive way).

One considers the rewriting rules:

$$\langle callcc \mid P \cdot \mathrm{E} \rangle \quad \rightarrow \quad \langle P \mid k_{\mathrm{E}} \cdot \mathrm{E} \rangle$$

$$\langle k_{\mathrm{E}} \mid P \cdot \mathrm{E}' \rangle \quad \rightarrow \quad \langle P \mid \mathrm{E} \rangle$$

Classical logic

Proposition.
The fundamental theorem remains true in the presence of

Peirce Law $$\frac{\Gamma \vdash M : (A \Rightarrow B) \Rightarrow A}{\Gamma \vdash callcc\, M : A}$$

Proof. Let ξ be an environment associated to Γ. Suppose that

$$M_\xi \quad \perp \quad \| (A \Rightarrow B) \Rightarrow A \|_\xi$$

We want to show that

$$callcc\, M_\xi \quad \perp \quad \| A \|_\xi$$

Suppose that

$$\mathrm{E} \in \| A \|_\xi$$

The sequence of rewriting steps

$$\langle callcc\, M_\xi \mid \mathrm{E} \rangle \rightarrow \langle callcc \mid M_\xi \cdot \mathrm{E} \rangle \rightarrow \langle M_\xi \mid k_{\mathrm{E}} \cdot \mathrm{E} \rangle$$

establishes that its is sufficient to show that

$$k_{\mathrm{E}} \quad \perp \quad \| A \Rightarrow B \|_\xi$$

So, consider a term P and context E' such that

$$P \perp \| A \|_\xi \qquad\qquad E' \in \| B \|_\xi$$

The property follows from the rewriting step

$$\langle k_{\mathrm{E}} \mid P \cdot \mathrm{E}' \rangle \rightarrow \langle P \mid \mathrm{E} \rangle$$

and from the fact that

$$\langle P \mid \mathrm{E} \rangle \in \perp\!\!\!\perp$$

We conclude that

$$callcc\, M_\xi \quad \perp \quad \| A \|_\xi$$

Extension: subtyping

$$X \subseteq Y$$

Zermelo-Fraenkel set theory

Starting from a model of ZF set theory, one defines

$$\|a \not\in b\| \quad = \quad \{ \quad \mathrm{E} \quad | \quad (a, \mathrm{E}) \in b \quad \}$$

A generalization of Paul Cohen's forcing.

Extension: intersection and union types

$$X \wedge Y \quad = \quad X \cap Y$$

$$X \vee Y \quad = \quad (X \cup Y)^{\perp\perp}$$

One needs to close in order to obtain a variety

Extension: existential quantification

Given a family of varieties $(X_i)_{i \in I}$ parametrized by I,

$$\forall \alpha . X_\alpha \quad = \quad \bigcap_{\alpha \in W} X_\alpha$$

$$\exists \alpha . X_\alpha \quad = \quad \Big(\bigcup_{\alpha \in W} X_\alpha \Big)^{\perp\perp}$$

Note that we need to close the union here!

VAR-ACCESS

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$$

APP

$$\frac{\Gamma \vdash e_1 : \tau_2 \to \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\, e_2 : \tau_1}$$

ABS

$$\frac{\Gamma, x : \tau_2 \vdash e : \tau_1}{\Gamma \vdash \lambda x.e : \tau_2 \to \tau_1}$$

PAIR

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}$$

FST

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \texttt{fst}(e) : \tau_1}$$

SND

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \texttt{snd}(e) : \tau_2}$$

CONSTANT TRUE

$$\Gamma \vdash \texttt{true} : \mathit{Bool}$$

CONSTANT FALSE

$$\Gamma \vdash \texttt{false} : \mathit{Bool}$$

CONDITIONAL

$$\frac{\Gamma \vdash e_1 : \mathit{Bool} \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : \tau}$$

FIXPOINT

$$\frac{\Gamma \vdash e : \tau \to \tau}{\Gamma \vdash Y\, e : \tau}$$

ALL-INTRO

$$\frac{\Gamma, \alpha \vdash e : \tau}{\Gamma \vdash e : \forall \alpha.\tau}$$

ALL-ELIM

$$\frac{\Gamma \vdash e : \forall \alpha.\tau}{\Gamma \vdash e : \tau[\tau'/\alpha]}$$

EXISTS-INTRO

$$\frac{\Gamma \vdash e : \tau[\tau'/\alpha]}{\Gamma \vdash e : \exists \alpha.\tau}$$

EXISTS-ELIM

$$\frac{\Gamma \vdash e : \exists \alpha.\tau' \qquad \Gamma, \alpha, x : \tau' \vdash \langle x \mid \mathrm{E} \rangle : \tau \qquad \alpha \notin FV(\tau) \qquad x \notin FV(\mathrm{E})}{\Gamma \vdash \langle e \mid \mathrm{E} \rangle : \tau}$$

SUB

$$\frac{\Gamma \vdash e : \tau' \qquad \tau' <: \tau}{\Gamma \vdash e : \tau}$$