Exposé GdT logique

Sémantique dénotationnelle

Christine Tasson
tasson@pps.jussieu.fr

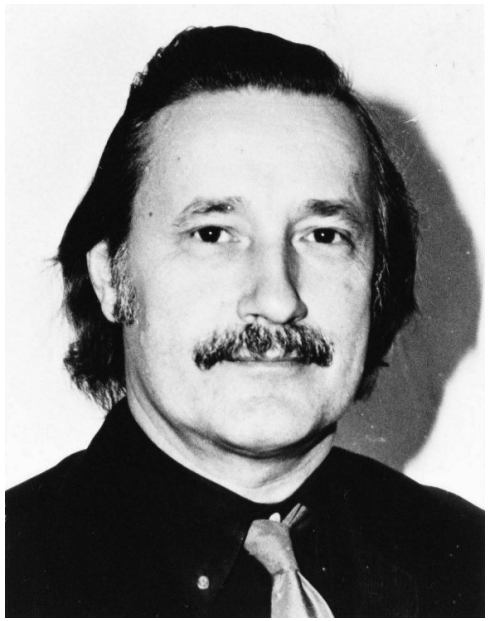
le 29 novembre 2010

Au sommaire

Syntaxe vs Sémantique

« *Decide what you want to say before you worry how you are going to say it.* »

Préface de The Scott-Strachey Approach to Programming Language Theory

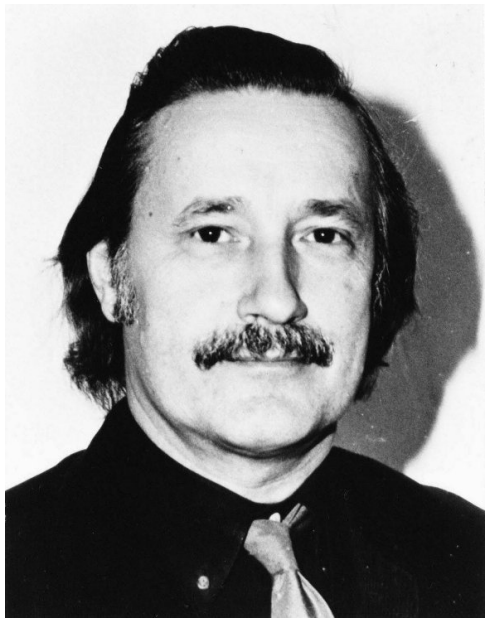


C. Strachey
(1916-1975)

- Qu'est-ce que la syntaxe ?
  L'alphabet et les règles de construction des programmes.

« *Decide what you want to say before you worry how you are going to say it.* »

Préface de The Scott-Strachey Approach to Programming Language Theory



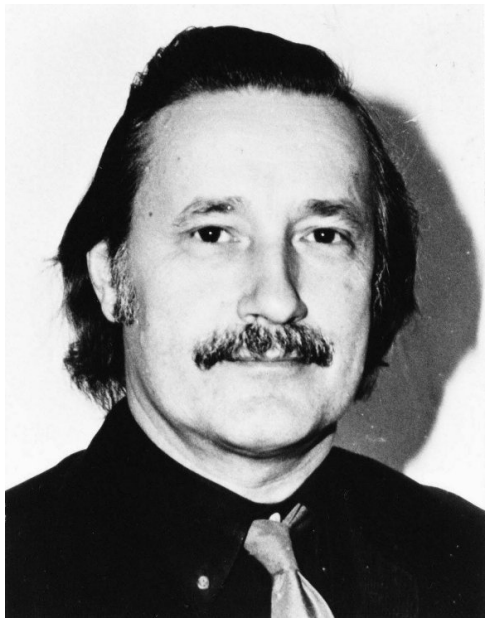
C. Strachey
(1916-1975)

- Qu'est-ce que la syntaxe ?
  L'alphabet et les règles de construction des programmes.
- Pourquoi la sémantique est-elle nécessaire ?
  Prolifération des syntaxes et manque de conventions.

De la syntaxe à la sémantique

« *Decide what you want to say before you worry how you are going to say it.* »

Préface de The Scott-Strachey Approach to Programming Language Theory



C. Strachey
(1916-1975)

- Qu'est-ce que la syntaxe ?
  L'alphabet et les règles de construction des programmes.
- Pourquoi la sémantique est-elle nécessaire ?
  Prolifération des syntaxes et manque de conventions.
- Qu'est-ce que la sémantique ?
  Le lien entre un programme et un objet mathématique conçu pour étudier certaines propriétés du langage.

Sémantiques plus ou moins abstraites

Un programme est représenté par son action sur son environnement.

Sémantique opérationnelle : Le programme est interprété par la suite des états de la machine l'exécutant.

Exemple : Machine de Turing.

Sémantiques plus ou moins abstraites

Un programme est représenté par son action sur son environnement.

Sémantique opérationnelle : Le programme est interprété par la suite des états de la machine l'exécutant.

Exemple : Machine de Turing.

Sémantique dénotationnelle : Le programme est interprété par une dénotation, c'est-à-dire une fonction mathématique qui représente l'effet du programme sur la dénotation de son entrée.

Sémantiques plus ou moins abstraites

Un programme est représenté par son action sur son environnement.

Sémantique opérationnelle : Le programme est interprété par la suite des états de la machine l'exécutant.

Exemple : Machine de Turing.

Sémantique dénotationnelle : Le programme est interprété par une dénotation, c'est-à-dire une fonction mathématique qui représente l'effet du programme sur la dénotation de son entrée.

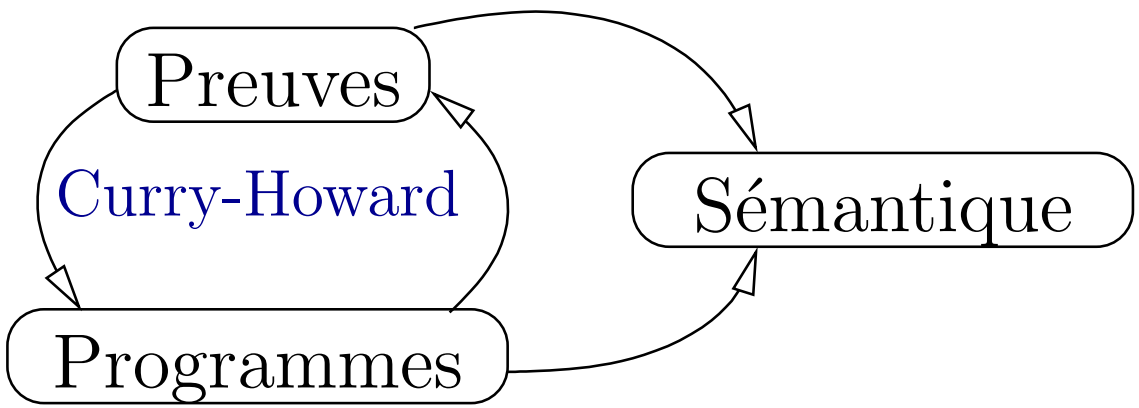
Sémantique axiomatique : Le programme est interprété par un opérateur transformant les propriétés logiques vérifiées par l'environnement avant exécution du programme, en les propriétés logique de l'environnement après exécution.

Exemple : logique de Hoare.

Principes de la sémantique dénotationnelle

- L'environnement est représenté par un objet mathématique et le programme est représenté par une fonction mathématique traduisant son effet sur l'environnement.
- Compositionnalité : La dénotation de deux programmes consécutifs est la composée de chacune des dénotations.
- La sémantique dénotationnelle $[\![\cdot]\!]$ est compatible avec la sémantique opérationnelle $\rightarrow$: Si $P \rightarrow Q$, alors $[\![P]\!] = [\![Q]\!]$.

Logique, Programmation et sémantique :



Théorie des modèles

Dans les années 60,

Strachey : Équation sémantiques impliquant la récursivité.

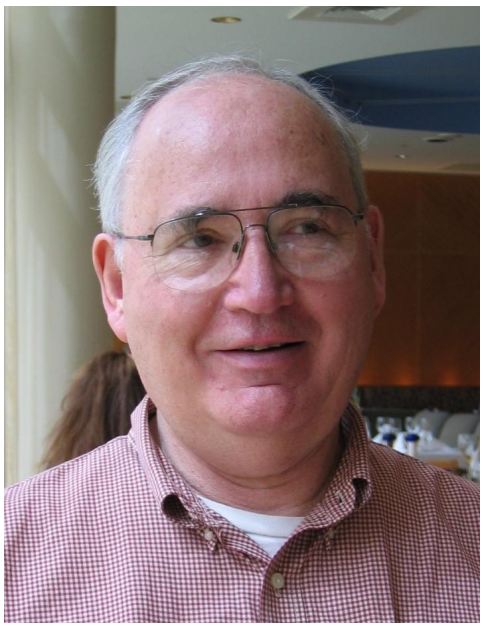
Scott : Solution à ces équations grâce aux domaines.

Domains and Lambda-Calculi, Roberto Amadio and Pierre-Louis Curien

Principes de la théorie des domaines

Toute partie finie du résultat d'un programme peut être atteinte en un nombre fini de calculs.

Exemple : Énumérer les entiers premiers.



D. Scott
Carnegie Mellon University

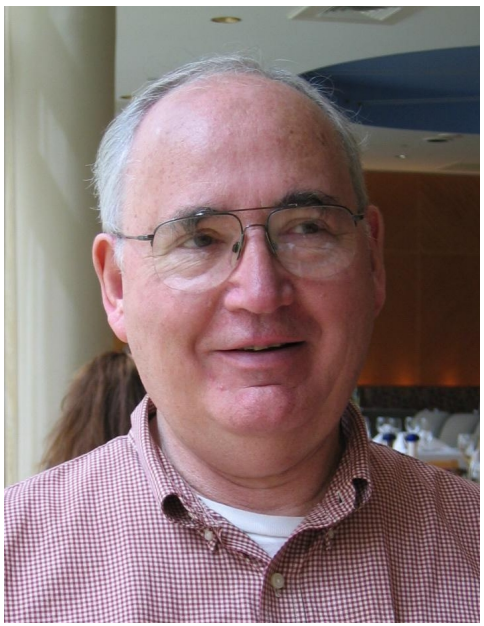
Domaine de Scott : Ensemble partiellement ordonné où l'ordre représente la quantité d'information connue.

Avec la bonne notion de fonction, on peut interpréter le λ-calcul.

Principes de la théorie des domaines

Toute partie finie du résultat d'un programme peut être atteinte en un nombre fini de calculs.

Exemple : Énumérer les entiers premiers.



D. Scott
Carnegie Mellon University

Domaine de Scott : Ensemble partiellement ordonné où l'ordre représente la quantité d'information connue.

Avec la bonne notion de fonction, on peut interpréter le λ-calcul.

Complete **P**artial **O**rder : Ensemble partiellement ordonné muni d'un plus petit élément et tel que toute suite croissante strictement ordonnée possède une borne supérieure.

Permet d'interpréter les points fixes et donc PCF.

Syntaxe et sémantique opérationnelle du λ-calcul

λ-calcul pur : $\quad M ::= x \mid M\,M \mid \lambda x.M$

Syntaxe et sémantique opérationnelle du λ-calcul

λ-calcul pur : $\quad M ::= x \mid M\,M \mid \lambda x.M$

λ-calcul simplement typé :

$$\begin{array}{rcl@{\qquad}rcl} At & ::= & \kappa \mid \kappa' \mid \dots & v & ::= & x \mid y \mid \dots \\ \sigma & ::= & At \mid \sigma \to \sigma & M & ::= & v \mid \lambda v : \sigma.M \mid M\,M \end{array}$$

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma} \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma.M : \sigma \to \tau} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau}$$

Syntaxe et sémantique opérationnelle du λ-calcul

λ-calcul pur : $\quad M ::= x \mid M\,M \mid \lambda x.M$

λ-calcul simplement typé :

$$\begin{array}{rcl@{\qquad}rcl} At & ::= & \kappa \mid \kappa' \mid \dots & v & ::= & x \mid y \mid \dots \\ \sigma & ::= & At \mid \sigma \to \sigma & M & ::= & v \mid \lambda v : \sigma.M \mid M\,M \end{array}$$

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma} \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma.M : \sigma \to \tau} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau}$$

β-réduction et règles de passage au contexte :

$$\frac{}{(\lambda x.M)\,N \to M[x := N]} \qquad \frac{M \to M'}{\lambda x.M \to \lambda x.M'} \qquad \frac{M \to M'}{M\,N \to M'\,N} \qquad \frac{N \to N'}{M\,N \to M\,N'}$$

Modèle du λ-calcul simplement typé

Catégorie Cartésienne Fermée :

est munie de

- objet terminal 1
- produit cartésien $\cdot \times \cdot$
- exponentiation $\cdot \Rightarrow \cdot$

$$\forall f : C \times A \to B$$
$$\exists! h : C \to A \Rightarrow B,$$
$$ev \circ (h \times id) = f$$

Exemple : CPO.

Catégorie Cartésienne Fermée :
est munie de

- objet terminal 1
- produit cartésien $\cdot \times \cdot$
- exponentiation $\cdot \Rightarrow \cdot$

$$\forall f : C \times A \to B$$
$$\exists! h : C \to A \Rightarrow B,$$
$$ev \circ (h \times id) = f$$

Exemple : CPO.

Interprétation du λ-calcul : Étant donnée l'interprétation des types atomiques : $[\![\kappa]\!]$, l'interprétation de la flèche est donnée par l'exponentiation :

$$[\![\sigma \to \tau]\!] = [\![\sigma]\!] \Rightarrow [\![\tau]\!].$$

$$\begin{aligned} [\![x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash x_i : \sigma_i]\!] &= \pi_{n,i} \\ [\![\Gamma \vdash \lambda x : \sigma . M : \sigma \to \tau]\!] &= \Lambda([\![\Gamma, x : \sigma \vdash M : \tau]\!]) \\ [\![\Gamma \vdash M\, N : \tau]\!] &= ev \circ ([\![\Gamma \vdash M : \sigma \to \tau]\!], [\![\Gamma \vdash N : \sigma]\!]) \end{aligned}$$

Il faut résoudre l'équation $D = D \Rightarrow D$:

$$\frac{x : \delta \in \Gamma}{\Gamma \vdash x : \delta} \qquad \frac{\Gamma, x : \delta \vdash M : \delta}{\Gamma \vdash \lambda x : \delta . M : \delta} \qquad \frac{\Gamma \vdash M : \delta \quad \Gamma \vdash N : \delta}{\Gamma \vdash M\, N : \delta}$$

Il faut résoudre l'équation $D = D \Rightarrow D$:

$$\frac{x : \delta \in \Gamma}{\Gamma \vdash x : \delta} \qquad \frac{\Gamma, x : \delta \vdash M : \delta}{\Gamma \vdash \lambda x : \delta . M : \delta} \qquad \frac{\Gamma \vdash M : \delta \quad \Gamma \vdash N : \delta}{\Gamma \vdash M\, N : \delta}$$

Objet réflexif : D d'une CCC est une rétraction (i, j) telle que :

$$i : D \Rightarrow D \to D, \qquad j : D \to D \Rightarrow D, \qquad j \circ i = id$$

Exemple : Modèle relationnel.

Il faut résoudre l'équation $D = D \Rightarrow D$:

$$\frac{x : \delta \in \Gamma}{\Gamma \vdash x : \delta} \qquad \frac{\Gamma, x : \delta \vdash M : \delta}{\Gamma \vdash \lambda x : \delta . M : \delta} \qquad \frac{\Gamma \vdash M : \delta \quad \Gamma \vdash N : \delta}{\Gamma \vdash M\, N : \delta}$$

Objet réflexif : D d'une CCC est une rétraction (i, j) telle que :

$$i : D \Rightarrow D \to D, \qquad j : D \to D \Rightarrow D, \qquad j \circ i = id$$

Exemple : Modèle relationnel.

Interprétation du λ-calcul pur : On reprend l'interprétation du λ-calcul simplement typé.

Idée : Interpréter les λ-termes par l'ensemble des approximations finies de la forme normale potentiellement infinie.

Définition :

$$BT(M) = \bigvee \{\omega(N) \mid M \to^* N\}$$

$$\omega(M) = \begin{cases} \Omega & \text{si } M = \lambda x_1 \cdots x_n.(\lambda x.M) M_1 \cdots M_p \\ \lambda x_1 \cdots x_n.x\omega(M_1) \cdots \omega(M_p) & \text{si } M = \lambda x_1 \cdots x_n.x M_1 \cdots M_p \end{cases}$$

Proposition : Les arbres de Böhm forment un modèle du λ-calcul pur.

Théorème de Böhm : Si M et N sont deux termes en forme normale et ne sont pas η-équivalents, alors ils sont séparables :

$$\forall x \neq y \, \exists C \;\; C[M] \to x \;\; C[N] \to y$$

Preuve : Utilise les arbres de Böhm.

Corollaire : Un modèle du λ-calcul n'égalise jamais deux formes normales.

Syntaxe et sémantique opérationnelle de PCF

Constantes de PCF : ι (entiers naturels), o (booléens)

$n \in \mathbb{N}$	:	ι	if	then	else	:	$o \to \iota \to \iota \to \iota$
tt, ff	:	o	if	then	else	:	$o \to o \to o \to o$
succ, pred	:	$\iota \to \iota$	Ω			:	$\sigma, \quad \forall\sigma$
zero?	:	$\iota \to o$	Y			:	$(\sigma \to \sigma) \to \sigma \quad \forall\sigma$

Sémantique opérationnelle :

$$(\lambda x.M)\,N \to M[x := N] \qquad Y\,M \to M\,(Y\,M)$$
$$\text{zero?}(0) \to tt \qquad \text{zero?}(n+1) \to ff$$
$$\text{succ}(n) \to n+1 \qquad \text{pred}(n+1) \to n$$
$$\text{if } tt \text{ then } N \text{ else } P \to N \qquad \text{if } ff \text{ then } N \text{ else } P \to P$$

$$\frac{M \to M'}{M\,N \to M'\,N} \qquad \frac{M \to M'}{\text{if } M \text{ then } N \text{ else } P \to \text{if } M' \text{ then } N \text{ else } P}$$

$$\frac{M \to M'}{f(M) \to f(M')} \quad (\text{pour } f \in \{\text{succ}, \text{pred}, \text{zero?}\})$$

Modèle standard de PCF : c'est un modèle de plus petit point fixe : une catégorie cartésienne close enrichie en CPO dans laquelle :

$$[\![\Omega]\!] = \bot \qquad [\![Y]\!] = \bigvee_{n \in \mathbb{N}} [\![\lambda f.f^n \Omega]\!]$$

Interprétation dans le modèle continu de PCF

$$\begin{array}{rcll} [\![o]\!] & = & \mathbb{B}_\bot & \text{où } \mathbb{B}_\bot = \{tt, ff, \bot\} \\ [\![\iota]\!] & = & \mathbb{N}_\bot & \text{domaine plat des entiers} \\ [\![\sigma \to \tau]\!] & = & [\![\sigma]\!] \Rightarrow [\![\tau]\!] & \text{fonctions continues} \end{array}$$

Les constantes succ, pred, $\mathrm{zero?}$ et $\mathrm{if\ \ then\ \ else}$ sont interprétées sans surprise.

À la recherche du meilleur modèle ? (1)

Le lemme d'adéquation : Dans tout modèle standard de PCF, pour tout programme P (terme clos de type de base, ι par exemple),

$$(\exists n\ P \to^* n) \Leftrightarrow [\![P]\!] = n$$

L'ordre d'observation : Si M et N ont le même type, alors

$$M \leq N \Leftrightarrow \forall C\ (C[M] \to^* c \Rightarrow C[N] \to^* c)$$

« Full abstraction » :

$$\boxed{M \leq N \Leftrightarrow [\![M]\!] \leq [\![N]\!]}$$

Contre-exemple : Ou-Parallèle.

À la recherche du meilleur modèle ? (2)

La course des années 80 :

1. Trouver une solution sémantique qui donne un éclairage mathématique sur l'essence de PCF.
2. Sémantique des jeux : P = Programme, O = Environnement. Point de vue interactif nouveau.
3. Hyland-Ong : liens avec les machines abstraites
4. Abramsky-Jagadeesan-Malakaria : liens avec la géométrie de l'interaction

Les autres approches :

1. Ajouter des constructions à PCF pour retrouver le modèle continu. Ou-parallèle / Opérateurs de contrôle
2. Comprendre la séquentialité pour éliminer le Ou-parallèle.
3. Hypercohérence : une caractérisation sémantique de la séquentialité.

Quelques applications de la Sémantique dénotationnelle

De la stabilité à la logique linéaire

Stabilité : raffinement des fonctions séquentielles
Espaces de cohérences : Sémantique découlant de l'étude de la stabilité.
Nouvelle décomposition : Les fonctions peuvent se décomposer grâce aux traces :

$$A \Rightarrow B = !A \multimap B$$

$A \multimap B$: programmes linéaires utilisant exactement une fois leur argument.

$!A$: exponentielle de A regroupant les copies des éléments de A.

Conséquences :

- Un nouveau domaine de recherche
- Une logique de ressources
- Bunched Logic

Intuition de la sémantique quantitative :

- Un programme de type $A \multimap B$ est interprété par une fonction linéaire d'un espace vectoriel A vers un espace vectoriel B.
- Un programme de type $A \Rightarrow B$ est interprété par une série entière

De la sémantique à la syntaxe : Qui dit série entière, dit développement de Taylor, dit dérivée.
Conséquence : Introduction du λ-calcul différentiel :

$$M \ ::= \ x \mid \lambda x.M \mid M\,N \mid DM\,N$$

Complexité, syntaxe et sémantique.

Constat : La complexité exponentielle des programmes provient de l'exponentielle ! et de $\Rightarrow$ qui permettent la duplication infinie des variables.

Remplacer l'exponentielle : Complexité implicite à la logique linéaire, on modifie les règles pour les connecteurs exponentiels et on borne le nombre de duplication.

J.-Y. Girard, Light linear logic

Remplacer la flêche intuitionniste : l'adjonction entre $\times$ et $\Rightarrow$ est mimée par une loi de distributivité entre produits et coproduits.

Pola : langage de programmation fonctionnelle dont le système de type garantit que les programmes terminent en temps polynomial.

« I am trying to find out where λ-calculus should come from, and the fact that the notion of a cartesian closed category is a late developing one (Eilenberg et Kelly (1966)) is not relevant : [. . .] we should look to it first ».

D. Scott, Relating theory of the λ-calculus

Idée des monades : Un programme est représenté par un morphisme de A (type des valeurs) dans TB (type des computations).

Définition : Une monade sur une catégorie C est la donnée de

- un foncteur $T : C \to C$
- une unité $\eta_A : A \to TA$
- une multiplication $\mu_A : TTA \to TA$

Exemples de monades :

- non-déterministe : $TA = \mathcal{P}(A)$, $\eta_A(a) = \{a\}$ et $\mu_A(X) = \cup X$.
- effets de bords : $TA = S \Rightarrow (A \times S)$ où $S \neq \emptyset$ est un ensemble de cases mémoire, $\eta_A(a) = \lambda s : S.\langle a, s\rangle$ et $\mu_A(f) = \lambda s : S.\mathrm{eval}(fs)$.

  Intuition : un calcul prend une case mémoire et renvoie sa valeur avec la case mémoire modifiée.

Bibliographie

- J. Stoy, *Denotational Semantics :The Scott-Strachey Approach to Programming Language Theory*, The MIT Press Series in Computer Science, 1977
- R. Amadio et P.-L. Curien, *Domains and Lambda-Calculi*, Cambridge Tracts in Theoretical Computer Science, 1998
- M. Hyland et L. Ong, *On full abstraction for PCF*, 1994
- S. Abramsky, R. Jafadeesan et P. Malacaria, *Full Abstraction for PCF*, 1995
- J.-Y. Girard, *Linear Logic, its syntax and semantics*, Advances in Linear Logic, 1995.

Bibliographie

- J.Y. Girard, *Normal functor power series*, Theoretical Computer Science, 1988
- T. Ehrhard, *Finiteness spaces*, Mathematical Structures in Computer Science, 2005
- T. Ehrhard et L. Regnier, *The differential λ-calculus*, Theoretical Computer Science, 2003
- E. Moggi, *Computational lambda-calculus and Monads*, LICS'89
- M. Burrell, R. Cockett et B. Redmond, *Pola : a language for PTIME programming*, Logic Computational Compleixty, Logic in Computer Science, 2009