La sémantique, un point de vue mathématique sur les programmes

Une brève histoire, partielle et subjective

Thomas Ehrhard

IRIF
CNRS et Université Paris Diderot

6 février 2018

Maurice Nivat a défendu toute sa vie l'idée

- que l'informatique est une science
- et que c'est une *science mathématique*.

Contrairement aux objets dont parle la physique, ceux de l'informatique sont de pures créations mentales et formelles.

Même s'ils se concrétisent dans la réalité (machines, programmes).

Le travail de Nivat en sémantique, et le rôle crucial qu'il a eu sur le développement de ce sujet en France, l'illustrent parfaitement.

Un bon point de départ : les fonctions récursives

Dans les années 1930, sous la pression de questions fondationnelles (programme de Hilbert), les logiciens ont été amenés à formaliser la notion de fonction calculable $\mathbb{N}^k \to \mathbb{N}$.

- λ-calcul pur (Church)
- systèmes d'équations récursives de Herbrand-Gödel
- machines de Turing
- fonctions générales récursives de Kleene
- ...

Toutes ces notions sont équivalentes.

Ici, on voit déjà la distinction entre

- le *programme* (le λ-terme, les équations de Herbrand-Gödel, la machine de Turing etc)
- et la *fonction mathématique* $\mathbb{N}^k \to \mathbb{N}$, partielle en général, qu'il calcule.

Mais on peut dire que les logiciens ont privilégié le second point de vue, par souci d'économie conceptuelle.

Et aussi à cause de la thèse de Church, puisque ce second point de vue est intrinsèque (et pas le premier).

Et après ?

La calculabilité a continué de jouer un rôle essentiel en logique, par exemple :

- Réalisabilité (Kleene, Kreisel) : sémantique des formules logiques comme des ensembles de programmes.
- Élimination des coupures (Gentzen) : contenu calculatoire des preuves, les preuves sont des programmes. Curry-Howard.
- Interprétations fonctionnelles (Gödel) : λ-calculs typés étendus (système T, *bar-recursion* de Spector, système F de Girard).
- Fonctionnelles récursives (Kleene, Platek, Gandy).
- ...

Pendant ce temps-là...

Développement massif des ordinateurs dans les années 1940-1960 (avec l'invention du transistor en 1947, du circuit intégré en 1958) sur la base des idées de Turing et Von Neumann.

Premiers bugs...

Premiers langages de programmation (abstraction, portabilité) : COBOL (Hopper), FORTRAN (Backus), LISP (McCarthy), ALGOL 60.

Vers une science des programmes

Un programme est un objet formel et indépendant de la machine.

Suite de symboles obéissant à une certaine *grammaire* définie de façon non ambiguë. On peut le transformer en un *arbre de syntaxe abstraite* : analyse lexicale et syntaxique.

Il devrait être possible de spécifier de façon tout aussi précise ce que *fait* un programme, ce qu'il calcule, sa *sémantique*.

Cette question est déjà considérée par McCarthy *A Basis for a Mathematical Theory of Computation* en 1963 qui introduit le *amb* (non-déterminisme) et propose un traitement pour les définitions récursives.

Vers la sémantique dénotationnelle : Strachey

Après avoir été physicien puis professeur d'école, Christopher Strachey s'intéresse vraiment à la programmation au début des années 1950, d'abord dans différents centres de recherche puis comme professeur : Cambridge puis Oxford.

1958 - 1966 : Strachey participe notamment à la mise au point de différents langages de programmation, dont CPL.

CPL $\rightarrow$ BCPL $\rightarrow$ B $\rightarrow$ C !

1964 : avec *Towards a Formal Semantics*, Strachey jette les bases d'une sémantique des programmes comme fonctions.

Principe général de la SD : les programmes représentent, *dénotent*, des *fonctions*.

Intuitivement, ces fonctions agissent sur des valeurs, ou sur des états de la machine.

Mais précisément, quelles fonctions mathématiques, agissant sur quels espaces ? Cette question attendra l'arrivée de Scott en 1969 pour trouver une première réponse satisfaisante.

Le papier de Strachey manque de rigueur, utilise des ensembles définis récursivement dont l'existence n'est pas évidente.

$$V = N + [(V \times S) \to (V \times S)] + \cdots$$

Impossible dans les ensembles à cause de la cardinalité !

Bases de la sémantique opérationnelle

1966 : avec *The Mechanical Evaluation of Expressions*, Peter Landin jette les bases de la sémantique opérationnelle : λ-calcul, opérateur de point fixe, machine abstraite.

$\rightsquigarrow$ ISWIM (la source des langages fonctionnels), machine SECD.

Il faut aussi mentionner Corrado Böhm qui présente en 1964 le langage CuCh basé sur le λ-calcul, avec une machine abstraite.

Principe général de la SO : *système de transitions* dont les états peuvent être des états d'une machine (automate à piles, machine de Turing, SECD, machine de Krivine, ZAM etc), des termes (réécriture), des processus etc.

1969

Dana Scott (élève de Church !) visite Amsterdam et travaille avec de Bakker sur les *schémas de programmes récursifs* (ils simplifient Paterson, 1968).

Il construit le 1er modèle du λ-calcul pur : $D_\infty = D_\infty \Rightarrow D_\infty$.

A l'automne, il visite Oxford (où Strachey est prof depuis 1966). Ensemble ils mettent en place un programme de sémantique mathématique des programmes basée sur les idées utilisées par Scott pour son interprétation du λ-calcul.

$\rightsquigarrow$ *Towards a Mathematical Semantics of Computer Languages*, Scott and Strachey, 1971.

Scott invente LCF (et donc PCF) — cf. Robin Milner, ML.

PCF

Noyau purement fonctionnel très simplifié de ML.

λ-calcul simplement typé avec

- un type de base ι pour les entiers
- des primitives de calcul (successeur, prédecesseur, conditionnelle)
- un opérateur de point fixe pour la récursivité ($\mathrm{let\ rec}$ de Ocaml).

Les idées fondamentales de Scott et Strachey

Les idées mathématiques de base sont très bien résumées dans *Outline of a Mathematical Theory of Computation*, Scott, 1970 (8 pp).

Scott y reconnaît qu'elles sont déjà largement présentes en logique, mais bien sûr pas sous la forme qu'il va leur donner.

Il cite Lacombe, Nerode, Kleene, Kreisel et Platek.

- Types de données $\rightsquigarrow$ ordres partiels complets (toute suite croissante a un sup) dont les éléments sont des informations *partielles* : plus un élément est grand, plus il est défini.
- Programmes $\rightsquigarrow$ fonctions croissantes préservant les sup des suites croissantes (continuité).
- Les définitions récursives de programmes sont interprétées comme des plus petits points fixes de fonctions continues.

Analogie : fonctions récursives partielles ordonnées par l'inclusion de leurs graphes.

Mais ici on se donne les unions de *toutes* les suites croissantes de fonctions, et donc *toutes* les fonctions partielles $\mathbb{N} \to \mathbb{N}$.

Continuité de Scott et Théorème de Rice Shapiro

Que reste-t-il de la récursivité si on enlève... la récursivité ?

Théorème (Rice-Shapiro)

Soit F un ensemble de fonctions récursives partielles tel que $\{n \mid \varphi_n \in F\}$ soit récursivement énumérable. Alors $\psi \in F$ si et seulement s'il existe une fonction **finie** *$\theta \subseteq \psi$ telle que $\theta \in F$.*

Autrement dit : le programme qui calcule F ne peut utiliser qu'une partie *finie* de son argument (qui est une fonction) pour produire un résultat.

C'est une propriété indépendante de la récursivité : continuité de Scott.

Maurice Nivat et la sémantique

En 1969 (toujours !), Maurice Nivat découvre les schémas de définition récursifs (version de Bakker et Scott) et s'y intéresse car il voit tout de suite l'analogie avec la théorie des langages.

Exemple :

$$\mathrm{fact}(x) \Leftarrow \mathrm{if}\, x = 0\, \mathrm{then}\, 1\, \mathrm{else}\, x * \mathrm{fact}(x-1)$$
$$F(x) \Leftarrow H(x, F(p(x)))$$

peut être vu comme une règle pour engendrer un ensemble "algébrique" d'arbres, qu'il voit plutôt comme des mots, par réécriture :

$$F(x), H(x, F(p(x))), H(x, H(p(x), F(p(p(x))))), \ldots$$

Les contributions de Maurice à la sémantique

Études des schémas de programmes récursifs, puis extension aux programmes non déterministes et parallèles.

Par exemple : son papier de 1974 *On the interpretation of recursive program schemes* contient des résultats qui préfigurent des résultats fondamentaux en Sémantique (O et D) :

- Continuité syntaxique (en lien avec des “arbres de Böhm”)
- Confluence, standardisation
- Adéquation.

Sémantique algébrique, arbres infinis : toute une école.

Remarque : PCF généralise les schémas récursifs

Les schémas récursifs peuvent être vus comme des termes particuliers de PCF :

ce sont les termes de dont le type est de la forme

$$\overbrace{\iota \to \cdots \to \iota \to}^{k} \iota \qquad \text{pour un } k \in \mathbb{N}$$

et dont toutes les variables libres ont un type de la même forme.

Un terme général de PCF peut avoir un type plus compliqué, comme $((\iota \to \iota \to \iota) \to \iota) \to \iota$.

À la même époque, Jean-Marie Cadiou et Jean Vuillemin préparent leur PhD à Stanford (Manna) sur les stratégies de calcul dans les schémas récursif : correction des stratégies d'évaluation, optimalité.

Puis (1973), Vuillemin introduit la *séquentialité* qui implique la correction de l'"appel par nécessité" dans les schémas.

Sous la houlette de Maurice : des schémas au λ-calcul

- Jean-Jacques Lévy étend les idées de Cadiou et Vuillemin au λ-calcul (stratégies de réduction sûre) — même si Maurice n'était pas un fanatique du λ-calcul !
- Gérard Berry développe une notion de stratégie bottom-up pour l'évaluation des schémas récursifs.

Au cours de ce travail, Berry introduit la *stabilité*, une notion sémantique qui relâche et généralise la séquentialité de Vuillemin.

Berry prouve des théorèmes de stabilité et de séquentialité syntaxique pour le λ-calcul. Les interactions avec Lévy ont été essentielles dans ce travail.

Important : IRIA, bât. 8, où on s'intéresse beaucoup aux travaux de Scott (et de Strachey, Milner, Plotkin etc).

Stabilité et séquentialité : de la syntaxe à la sémantique

Berry généralise la stabilité à tous les types dans la sémantique et obtient de nouveaux modèles dénotationnels de PCF avec des propriétés très différentes de celui de Scott.

Les objets sont toujours des ordres partiels complets, mais maintenant on s'intéresse aussi aux infs, pas seulement aux sups...

Intuition sur la stabilité...

Soit un programme $F : (\iota \to \iota) \to \iota$ où ι représente le type des entiers.

Imaginez que $F(f) = 7$ pour un certain programme $f : \iota \to \iota$.

Scott nous dit qu'il y a une fonction partielle $f_0 : \iota \to \iota$ à domaine *fini* et telle que $f_0 \subseteq f$ (comme graphes) pour laquelle on a déjà $F(f_0) = 7$: F n'a pu utiliser qu'une partie finie de f pour arriver au résultat 7.

Berry nous dit qu'il n'y a qu'un seul tel f_0 minimal : son domaine est l'ensemble des entiers auxquels F a dû appliquer f pour arriver au résultat 7. Cet ensemble dépend de F **et** de f.

Le déterminisme du calcul nous assure que cet ensemble est bien défini.

. . . et sur la séquentialité

Le *premier* entier auquel F a appliqué f ne dépend que de F : c'est l'*indice de séquentialité* de F (sur la fonction indéfinie).

Mais on n'arrive pas à refaire avec la séquentialité ce que Berry a fait avec la stabilité.

Difficultés inhérentes à la séquentialité

- En fait, l'indice de séquentialité de F ne dépend pas que de F (en tant que fonction), mais aussi du *programme qui définit* F.
- Pour cette raison, la notion même d'indice perd son sens quand on "monte dans les types" dans le cadre de Scott (ou de Berry).

Exemple : $F_1(f) = f(1) + f(2)$ $\qquad$ $F_2(f) = f(2) + f(1)$

où "$+$" est une addition qui évalue d'abord son argument de gauche.

F_1 et F_2 sont deux fonctions séquentielles $(\iota \to \iota) \to \iota$ identiques, mais des programmes avec des indices de séquentialité différents.

Algorithmes séquentiels : Berry et Pierre-Louis Curien

Peut-on trouver une notion intermédiaire entre “fonctions” et “programmes” qui permette de généraliser la séquentialité aux types fonctionnels ?

Les CDS. Gilles Kahn et Gordon Plotkin avaient étendu la séquentialité à des domaines assez généraux (Concrete Data Structures, CDS) pour interpréter les *réseaux de Kahn* (calcul distribué à base de streams).

Intuitivement, une CDS est un domaine (à la Scott) avec une notion d'indices : une donnée est une sorte de tableau avec des cellules remplies par des valeurs élémentaires.

En général, pour pouvoir remplir une cellule, il faut en avoir rempli d'autres avant.

En 1977, Pierre-Louis Curien commence une thèse avec Berry à Sophia Antipolis (centre de maths app de l'École des Mines).

En 1978, ils arrivent à la notion d'*algorithme séquentiel* sur les CDS : ce sont des fonctions séquentielles munies d'une fonction de choix explicite d'indices de séquentialité (cellules).

$\rightsquigarrow$ CDS0, langage de bas niveau.

Une grande nouveauté : les programmes sont interprétés par des morphismes qu'il est très difficile de voir comme de simples fonctions (maintenant on sait...).

D'où la *nécessité* d'introduire le formalisme des catégories en sémantique dénotationnelle : catégories cartésiennes fermées etc.

Et Maurice ?

Il cesse de travailler directement sur la sémantique à la fin des années 1970.

Mais il continue de montrer beaucoup d'intérêt pour le sujet : TCS, EATCS, recrutements universitaires etc.

1985 : retour de la logique

En 1970, Jean-Yves Girard avait introduit le *système F*, un λ-calcul avec types polymorphes et prouvé sa normalisation forte.

Il a cherché ensuite à ramener cette preuve (qui entraîne la cohérence de PA_2) à une *récurrence ordinale*.

Pour cela il a développé un généralisation catégorique des notations ordinales : les *dilatateurs* (foncteurs sur les ordinaux).

Ces foncteurs vérifient des formes catégoriques de la continuité de Scott et de la stabilité. C'est essentiel pour donner une *représentation finitaire* des dilatateurs.

Entre temps Girard et Jean-Louis Krivine avait commencé à s'intéresser à l'informatique, notamment après la redécouverte du système F par John Reynolds.

Girard récupère cette idée de stabilité pour construire un modèle dénotationnel du système *F* : *The system F of variable types, fifteen years later*, TCS, 1986.

Les types (clos) sont interprétés par des domaines de Scott très particuliers (domaines qualitatifs). Les termes deviennent des fonctions stables au sens de Berry.

Dans l'*Appendix C* de ce papier, Girard observe qu'il peut se restreindre à des espaces encore plus spécifiques : les *espaces cohérents*.

La logique linéaire

C'est la remarque fondamentale qui mènera Girard à la logique linéaire en 1986.

Dans ce modèle, l'espace cohérent $X \Rightarrow Y$ des fonctions stables de X vers Y s'écrit

$$X \Rightarrow Y = (!X) \multimap Y$$

- $Z \multimap Y$ est l'espace des fonctions *linéaires* de X vers Y (fonctions stables qui commutent à toutes les unions qui existent).
- $!X$ est une nouvelle construction.

Analogie entre SD et algèbre linéaire

La catégorie des espaces cohérents et des fonctions linéaires ressemble à celle des espaces vectoriels de dimension finie et des fonctions linéaires :

- dualité $X^* = (X \multimap \bot)$ avec $X^{**} = X$
- en moins dégénéré : $(X \otimes Y)^*$ n'est pas isomorphe à $X^* \otimes Y^*$.

Dualité = négation linéaire.

Intuition opérationnelle :

- Une fonction linéaire est un programme qui utilise son argument exactement une fois.
- $!X$, c'est "du X autant de fois qu'on veut".

$\rightsquigarrow$ *Linear Logic*, TCS, 1987. Un des articles les plus cités de TCS.

C'est le début d'un long développement, Girard introduit plein de nouvelles idées qui découlent de la découverte de LL :

- réseaux de preuves
- géométrie de l'interaction
- systèmes à complexité bornée
- logique polarisée
- etc.

LL et sémantique dénotationnelle

La LL a complètement changé le point de vue sur la sémantique dénotationnelle.

On retrouve cette structure linéaire dans de nombreux modèles dénotationnels, et même dans la sémantique de Scott.

À chaque fois, la linéarité fait apparaître des structures nouvelles et surprenantes.

Ma première expérience en SD...

En 1988, avec Antonio Bucciarelli, on commence à travailler sur la séquentialité.

En retournant la définition de Vuillemin dans tous les sens, on se rend compte qu'elle est équivalente à la conservation de certains infs, plus que les infs de familles bornées de Berry.

$\rightsquigarrow$ **stabilité forte**

Contrairement à celle de Vuillemin, cette définition s'étend aux types fonctionnels : on a une catégorie cartésienne fermée où les fonctions "du 1er ordre" sont Vuillemin-séquentielles !

La cerise sur le gâteau

Ensuite j'essaie d'exprimer, et de prouver, que ces fonctions fortement stables sont "représentables" par des algorithmes séquentiels.

Pour cela je suis amené à faire des hypothèses supplémentaires sur nos objets.

Je finis par en trouver une classe particulière qu'on peut décrire comme des hypergraphes, qui suffit pour interpréter PCF (et de nombreux autres langages), et étend la séquentialité de Vuillemin.

Et la surprise assez incroyable, c'est que c'est aussi un modèle de LL !

$\rightsquigarrow$ 3 caractérisations indépendantes de la stabilité forte.

Les jeux : 1992

François Lamarche et Curien revoient les algorithmes séquentiels à la lumière de la logique linéaire.

La négation linéaire devient l'échange entre les cellules et les valeurs des CDS, qui s'organisent maintenant en séquences alternées (CDS filiformes).

On peut voir alors les algorithmes séquentiels comme des stratégies déterministes dans des jeux à deux joueurs : cellule/valeur (ou opposant/joueur).

À partir du même genre d'idée, 3 groupes

- Hanno Nickau
- Martin Hyland et Luke Ong
- Samson Abramsky, Radha Jagadeesan et Pasquale Malacaria

développent indépendamment des modèles de jeux déterministes "dénotationnellement complets" au sens où toute stratégie finie est définissable par un terme de PCF.

Modulo un "collapse extensionnel", c'est une solution (non effective, cf. Ralf Loader) au problème de la full abstraction posé par Milner et Plotkin autour de 1975.

Depuis, le sujet reste très actif (impossible de citer des noms, il y en a trop !).

Autres développements

Les effets. Comment parler des “effets” en SD : mémoire modifiable, lecture/écriture, non déterminisme, choix probabilistes, continuations (c’est un peu du Prévert) ?

Eugenio Moggi, en 1986, propose une méthode générale consistant à les encapsuler dans une “monade forte”.

$\rightsquigarrow$ langage Haskell, un des langages fonctionnels les plus populaires.

Nombreux développements théoriques :

- powerdomains comme monades d'effets (pour le non déterminisme, par exemple)
- effets algébriques (John Power, Gordon Plotkin, Martin Hyland, Paul-André Melliès)
- effets et logique linéaire (Masahito Hasegawa, Paul Levy, Alex Simpson, Guillaume Munch Maccagnoni)
- etc.

Logique classique. En 1990, Timothy Griffin fait une découverte fondamentale : on peut typer la construction “appel avec continuation courante” avec la loi de Peirce

$$((A \to B) \to A) \to A$$

une tautologie classique, non prouvable en logique intuitionniste.

- $\lambda\mu$-calcul (Michel Parigot)
- Réalisabilité classique (Jean-Louis Krivine)
- Logique linéaire polarisée (Girard puis toute une école, logique tensorielle de Melliès)
- Interprétation calculatoire du calcul des séquents (Hugo Herbelin, Curien...)

Sémantique quantitative. La sémantique de Scott, de Berry etc est *qualitative* : l'interprétation d'un programme $\iota \to \iota$ ne dit pas combien de fois il utilise son argument.

Ce n'est pas le cas des jeux N, HO, AJM.

Ni de la *sémantique quantitative* introduite en 1987 (en fait quelques années avant) par Girard.

Programmes $\rightsquigarrow$ séries entières à coefficients ensemblistes.

- Parfaitement compatible avec LL.
- Lien avec les espèces de structures de Joyal (Martin Hyland etc)
- Sémantique quantitative avec coefficients numériques (Girard, E., Vincent Danos, Christine Tasson, Michele Pagani etc)
- Applications aux langages non déterministes, probabilistes, quantiques.
- λ-calcul et logique linéaire différentiels (E. et Laurent Regnier).

Conclusion

Presque 50 ans de recul par rapport à la SD. Impossible de faire un bilan ici, mais quelques remarques :

- La SD est d'abord un formidable outil heuristique : Logique Linéaire, langage Haskell, sémantique des jeux etc.
- Retour des schémas : avec les travaux récents de Ong, Kobayashi, Serre, Salvati, Walukiewicz, Melliès, Grellois etc, la SD est apparue comme un outil essentiel pour analyser le *model checking* des schémas récursifs d'ordre supérieur.
- Il faudrait plus de méthodes effectives de vérification, dans l'esprit de l'interprétation abstraite (Cousot et Cousot), issues de la SD.
- Je suis convaincu que la SD aura aussi son mot à dire sur les langages pour l'apprentissage, l'IA etc.

Merci

- à Jean-Jacques Lévy pour toutes les informations qu'il m'a données sur ce qui se manigançait au bâtiment 8 de l'IRIA Rocquencourt dans les années 70
- à Pierre-Louis Curien, Antonio Bucciarelli, Lorenzo Tortora de Falco et Paul-André Melliès pour leurs commentaires sur des versions préliminaires de ces transparents

et à Maurice Nivat pour avoir créé et toujours soutenu l'école de sémantique des langages de programmation en France !