Curry-Howard et choix dépendant

Jean-Louis Krivine

IRIF - Université Paris-Diderot

krivine@irif.fr

Journées PPS, Paris, 8 novembre 2018

Introduction

La *correspondance de Curry-Howard* consiste, en principe, à associer
un programme à chaque démonstration mathématique.
Depuis sa découverte, dans les années 70, elle connaît de nombreux
développements et avatars : à quelles preuves s'applique-t-elle ?
D'abord limitée à la logique intuitionniste, on est passé à la logique classique
en 1990 (T. Griffin), quand on a enfin pris conscience, en théorie,
de la *dualité programme-environnement*, bien connue en pratique.

Introduction

Finalement, pour mettre en œuvre cette correspondance
beaucoup de systèmes très élaborés ont été développés :
sémantique dénotationnelle, système F, théorie de Martin-Löf,
réalisabilité, topos, types homotopiques, …
Or, aucun ne s'appuie sur la formalisation usuelle des mathématiques
à savoir la théorie des ensembles avec choix : ZFC.
Sauf celui dont je me servirai pour cet exposé :
la théorie de la *réalisabilité classique*.
Quel rapport entre la réalisabilité inventée par Kleene en 1950
pour l'Arithmétique intuitionniste, et la correspondance preuves-pogrammes ?

Réalisabilité

C'est que la correspondance de Curry-Howard ne donne un programme
à partir d'une preuve *qu'en logique pure* c'est-à-dire *sans axiome*.
Il faut la compléter par des programmes convenables pour les axiomes
(on va voir qu'il y a un prix à payer pour cela).
C'est le rôle des diverses théories de la *réalisabilité*.
Il y en a plusieurs suivant les divers systèmes d'axiomes possibles.
La *réalisabilité classique* traite des axiomes de ZF en logique classique.
Toutefois, elle ne s'occupe que de ZF. Il manque *l'axiome du choix*.
Est-ce grave ? Et si oui, comment s'en sortir ?

Splendeurs et misères de Curry-Howard

En passant d'une preuve à un programme, il y a perte d'information, principalement à cause de la traduction des axiomes utilisés.
Cette perte est énorme en Arithmétique. Par exemple, elle est *totale* pour les formules Π^0_1 (théorème de Fermat, hypothèse de Riemann) !
Pour une formule Π^0_2 c'est-à-dire $\forall m \exists n (f(m,n) = 0)$, on obtient seulement un programme de calcul d'une fonction g telle que $f(m, g(m)) = 0$.
Même s'il est meilleur que le programme trivial, c'est très décevant.
C'est en Analyse que cette traduction peut montrer son véritable intérêt.
Après tout, c'est là que se trouvent les applications des mathématiques : E.D.P., Fourier, espaces de Hilbert et E.V.T., ondelettes, probas-stats, etc.

Le choix dépendant

Bien, mais en Analyse, on a besoin de l'axiome du choix AC :

$$\forall a \exists f (\forall x \in a) \forall y (F[x,y] \to F[x, f(x)])$$

Hélas, on n'a pas encore réussi à trouver un programme pour lui.

On va donc faire provisoirement une croix sur Zorn et Zermelo

(qui, par ailleurs, ont des conséquences assez abracadabrantes).

Pratiquement toujours, on peut se contenter du *choix dépendant DC* :

$$\forall x \exists y\, F[x,y] \to \exists f (\forall n \in \mathbb{N}) F[f(n), f(n+1)]$$

Celui-là, pas question de s'en passer. En Analyse, on s'en sert tout le temps,

sans y penser, comme d'une règle logique. Et heureusement,

on a trouvé un programme pour cet axiome, et même plusieurs !

Le choix dépendant

Je vais vous en montrer trois ; le deuxième est tout récent.
Mais, à la réflexion, plusieurs programmes, c'est moins bien qu'un seul.
Explication :
Pour chaque axiome de ZF, on a un programme, en λ_c-calcul pur,
valable pour tous les langages de programmation.
D'une certaine façon, les axiomes de ZF se comportent donc
un peu comme les règles logiques.
Par contre, pour l'axiome DC, il y a diverses contraintes sur le langage,
et on a ainsi différents programmes, suivant le choix de ces contraintes.

La réalisabilité classique en bref

Pour écrire des programmes, il faut un langage. On va au plus simple : le *λ-calcul*.
Les programmes, dits λ-termes, sont construits à partir des *variables* $x, y, \ldots$
au moyen de *l'application* : $t, u \mapsto (t)u$ et de *l'abstraction* : $t \mapsto \lambda x\, t$.
C'est tout pour le *λ-calcul "pur"*.
Pour parler d'environnements (logique classique oblige),
on ajoute des *piles* (suites finies de termes)
et une instruction cc pour les sauvegarder dans des termes (*continuations*).
On a alors le *λ_c-calcul pur*.

La réalisabilité classique en bref

Pour pouvoir exécuter un programme (terme),
il faut lui adjoindre un environnement (pile). On a alors un *processus*.
On peut définir des contraintes sur le langage :

• En ajoutant des instructions ; une horloge, par exemple
ou bien `quote` comme en LISP, ou `random`, ou `read`, `write`, etc.

• En choisissant un ensemble $\perp\!\!\!\perp$ de processus "distingués", par exemple
ceux qui s'arrêtent, ou bien ceux qui ne s'arrêtent pas, ou bien ...

$\perp\!\!\!\perp$ fixe le "mode d'exécution" (appel par nom, par valeur, impératif, etc.)
On note $t \Vdash F$ pour dire que le terme t réalise la formule F.
On cherche donc un terme qui *réalise le choix dépendant*.

Bar récursion

En λ-calcul pur, on écrit des programmes invraisemblables,
bien plus facilement que dans d'autres langages. Exemples :
Le célèbre *point fixe* (Curry,Turing) :

$$\mathsf{Y} = (\lambda a \lambda x (x)(a)ax)\lambda a \lambda x (x)(a)ax$$

Beaucoup moins connu, mais plus étrange encore,
l'opérateur de *bar récursion* (Brouwer, Spector) :

$$\mathsf{B} = \lambda g \lambda u (\mathsf{Y}) \lambda h \lambda k \lambda f (u)(\chi k f)(g)\lambda z (h k^+)(\chi) k f z$$

où $\chi k f = \lambda z \lambda i ((i < k)(f)i)z$ (prolongement d'une fonction finie par une constante) ;
$k^+ = \lambda f \lambda x (f)(k) f x$ (successeur de l'entier k) ;
$(i < k) = ((k\, \lambda x \lambda y\, yx)\lambda d \lambda x \lambda y\, y)(i\, \lambda x \lambda y\, yx)\lambda d \lambda x \lambda y\, x$ (comparaison d'entiers).
Personne ne peut dire ce que fait vraiment ce programme.
Il est pourtant fort intéressant, comme on va voir.

Bar récursion et choix dépendant

Il est temps de vous montrer les trois programmes promis
pour l'axiome du choix dépendant.
Ils sont très différents, car chacun suppose d'utiliser
le langage de programmation (ici, le λ-calcul) d'une certaine façon.
Le premier est justement l'opérateur B de bar récursion.
Il réalise le choix dépendant à condition de n'utiliser *que le λ_c-calcul pur*
et de considérer comme "distingués" *les processus qui s'arrêtent*.
Par contre, on peut réduire n'importe quel redex dans le processus
mais on n'a pas droit à de nouvelles instructions.
Ce sont donc des contraintes très fortes
qui empêcheraient, par exemple, de programmer un ordonnanceur.

Remarques sur la bar récursion

La *bar induction* fut introduite par Brouwer. Intuitivement,
elle consiste à dire que, si un arbre n'a que des branches finies
(d'où la "barre") il est bien fondé ; ce qui permet l'induction.
Spector s'en est servi pour une "preuve" de la consistance de l'Analyse.
Dans un article bien connu, Berardi, Bezem & Coquand
ont écrit le programme et montré qu'il réalisait DC, en logique du second ordre.
Berger & Oliva l'ont adapté à la théorie des domaines.
Streicher l'a adapté à la réalisabilité classique
et montré ainsi qu'il réalisait DC *dans ZF*.
Notons enfin, qu'à l'aide de B, on peut écrire un λ-terme pur réalisant
l'hypothèse du continu, c'est-à-dire un bon ordre sur $\mathbb{R}$ de type $\aleph_1$.

Petite page de publicité

C'est un avantage de la théorie de la réalisabilité classique
de pouvoir *énoncer* un tel résultat, sans même parler de le démontrer.
Bon, il n'y aura pas une foule d'applications, mais ça donne à réfléchir
sur ce que veut vraiment dire cette fameuse hypothèse du continu.
Par ailleurs, les deux autres programmes dont je vais parler
supposent de nouvelles façons d'utiliser le λ-calcul
qui ne sont apparues qu'avec la réalisabilité classique.

Variables *fraîches*

Ajoutons au λ_c-calcul pur deux instructions κ, e portant sur des variables $x_0, \ldots, x_i, \ldots$

$\kappa\, t$ se réduit en $t\, x_n$ où x_n est la première variable qui n'apparaît pas dans t
(introduction d'une variable fraîche) ;

e $x_i\, x_j\, t\, u\, v$ se réduit en t si $i = j$, en u si $i < j$ et en v si $i > j$
(élimination des variables).

Noter que les deux premiers arguments de e *doivent être des variables*.

L'exécution se limite donc à la *réduction de tête faible*.

Par contre, on a droit à de nouvelles instructions,
pourvu qu'elles ne distinguent pas les variables x_i ;
et une grande variété de choix possibles pour $\perp\!\!\!\perp$.

Choix dénombrable

Et maintenant, le programme pour le choix dépendant ? Prenons plutôt l'axiome du *choix dénombrable* qui est plus simple à expliquer (et presqu'aussi utile) : $\exists f(\forall n \in \mathbb{N})\forall x(F(n,x) \to F(n,f[n]))$.

On définit une fonction ϕ dans ZF et on réalise trois formules qui expriment que $\phi[n] = \{f[n]\}$ (singleton d'une fonction de choix) :

$$(\forall n \in \mathbb{N}) \begin{cases} \text{i) } (\forall x, y \in \phi[n])(x = y)\,; \\ \text{ii) } \exists x\, F(n,x) \to \exists x(x \in \phi[n])\,; \\ \text{iii) } (\forall x \in \phi[n])F(n,x)\,; \end{cases}$$

(i) est réalisée par e, (ii) par $\kappa \circ \mathsf{Y}$ et (iii) par $\lambda x \lambda y \lambda z\, z$.

Ce programme est donc fort simple et son exécution facile à comprendre.

Et cependant, son rapport avec l'axiome reste énigmatique.

Quote ou horloge

On ajoute cette fois au λ_c-calcul pur une instruction ι qui fournit un entier :

$\iota t u$ se réduit à $t\nu$, où l'entier ν peut être calculé de diverses façons.

a) ν est le numéro du terme u (analogue à quote) ;

b) ν est l'heure courante ;

c) toute autre méthode, pourvu que $u \neq u' \Rightarrow \nu \neq \nu'$

par exemple la signature SHA2 de u (si on exclut les collisions)

ou tout simplement `random`.

On doit définir $\perp\!\!\!\perp$ pour avoir une exécution *impérative* :

réduction de tête faible avec un instant initial (boot).

On a droit à une foule de nouvelles instructions : `read, write,...`

Quote ou horloge

On obtient encore l'axiome du choix dénombrable
en réalisant les mêmes trois formules.
(ii) est réalisée par $\iota \circ \mathsf{Y}$, (i) et (iii) par deux λ-termes purs simples.
C'est certainement cette troisième "implémentation" de DC
qui se rapproche le plus des applications usuelles de l'Analyse,
qui sont souvent effectuées en langage impératif sur une station de travail ;
par exemple, résolution numérique d'une équation différentielle
ou compression d'images par ondelettes.
Il reste que, comme dans le cas précédent, le rôle de ce programme
dans les calculs effectifs n'est pas encore bien compris.

Références

S. Berardi, M. Bezem, T. Coquand *On the computational content of the axiom of choice.* J. Symb. Log. 63 (1998) p. 600-622.

U. Berger, P. Oliva *Modified bar recursion and classical dependent choice.* Proc. Logic Colloquium 2001 - Springer (2005) p. 89-107.

J.-L. Krivine *Realizability algebras II : new models of ZF + DC.* Logical Methods in Comp. Sc., vol. 8, 1:10 (2012) p. 1-28.

J.-L. Krivine *Bar recursion in classical realizability : dependent choice and continuum hypothesis.* Proceedings CSL 2016, LIPIcs vol. 62 (2016) pp. 25:1-11.

J.-L. Krivine *Curry-Howard, dependent choice and fresh variables.* En préparation.

T. Streicher *A classical realizability model arising from a stable model of untyped λ-calculus.* Logical Methods in Comp. Sc., vol. 13, 4 (2017).