Et si les preuves mathématiques n'étaient autres que des programmes ?

Hugo Herbelin

Maths en Mouvement

2 décembre 2023

Si je vous dis qu'une preuve est un programme, vous en pensez quoi ?

En tout cas, c'est le point de vue que Brouwer défendit au début du 20e siècle, créant une petite polémique dans le milieu des mathématiques de l'époque

La thèse de L. E. J. Brouwer est la suivante

Qu'est-ce qu'une preuve d'une conjonction, par exemple de la conjonction « *2 est un nombre pair et 3 un nombre impair* » ?

Et bien, c'est une juste la paire disons (p, q) d'une preuve p de la 1e proposition et d'une preuve q de la 2e proposition.

La thèse de L. E. J. Brouwer est la suivante (cas de l'implication)

Et qu'est-ce qu'une preuve d'une implication, par exemple « *si* $x > 2$ *est premier alors* x *est impair* » ?

Et bien, c'est une juste une fonction φ qui à toute preuve p de la prémisse associe une preuve $\varphi(p)$ de la conclusion.

La thèse de L. E. J. Brouwer est la suivante (cas de la quantification existentielle)

Et qu'est-ce que serait alors la preuve d'un énoncé existentiel, par exemple « *il existe un entier premier supérieur à 100* » ?

La thèse de L. E. J. Brouwer est la suivante (cas de la quantification existentielle)

Et qu'est-ce que serait alors la preuve d'un énoncé existentiel, par exemple « *il existe un entier premier supérieur à 100* » ?

Et bien ce serait aussi une paire, mais une paire hétérogène, une paire constituée d'un témoin, par exemple 101 ici, et d'une preuve que le témoin vérifie bien la propriété.

La thèse de L. E. J. Brouwer est la suivante (cas de la quantification universelle)

Et qu'est-ce que serait alors la preuve d'un énoncé universel, par exemple « *pour tout* x, $x^2 \geqslant 0$ » ?

La thèse de L. E. J. Brouwer est la suivante (cas de la quantification universelle)

Et qu'est-ce que serait alors la preuve d'un énoncé universel, par exemple « *pour tout* x, $x^2 \geqslant 0$ » ?

Et bien ce serait aussi une fonction, mais une fonction φ qui prend cette fois en argument un élément arbitraire x et qui renvoie une preuve $\varphi(x)$ que cet élément x vérifie la propriété donnée.

Bref, le slogan de L. E. J. Brouwer, c'est qu'une preuve, c'est comme un programme

Connaissez-vous le λ-calcul ?

Le λ-calcul

C'est un langage minimal, dû à Church (1932), pour parler des programmes.

Il est présent implicitement dans tous les langages de programmation courants, et plus explicitement dans des langages comme Lisp, Scheme, Haskell, Caml-Light, OCaml, λ-Prolog, ...

Le λ-calcul a seulement trois instructions !

$$
\begin{array}{rcll}
t, u, v & ::= & x & \textit{des variables} \\
& | & t\,u & \textit{des applications de fonctions « } f(u) \textit{ »} \\
& | & \lambda x.\, t & \textit{des définitions de fonction « } x \mapsto t \textit{ »}
\end{array}
$$

Et cela lui suffit à être aussi expressible que les machines de Turing !

En pratique, c'est utile de rajouter une diversité de types de données

t, u, v	$::=$	x	*des variables*
	$\mid$	$t\,u$	*des applications de fonctions « $f(u)$ »*
	$\mid$	$\lambda x.\, t$	*des définitions de fonction « $x \mapsto t$ »*
	$\mid$	(t, u)	*des paires*
	$\mid$	$fst(t) \mid snd(t)$	*des projections*
	$\mid$	$0 \mid n+1$	*des entiers*
	$\mid$	$rec[0 \mapsto u \mid n+1 \mapsto_x v]\ (t)$	*de la récurrence sur les entiers*
	$\mid$	$\ldots$	

Il est aussi utile de *typer* le λ-calcul

(c'est ce qui ne se passe pas dans Lisp ou Scheme, mais qui se passe dans Haskell ou OCaml)

Il est aussi utile de *typer* le λ-calcul

(c'est ce qui ne se passe pas dans Lisp ou Scheme, mais qui se passe dans Haskell ou OCaml)

Une règle de typage parle de *jugements* représentés par des *séquents* $\Gamma \vdash t : A$

Ici, Γ est un contexte de variables

Γ, Δ	$::=$	$\varnothing$	*le contexte vide*
	$\mid$	$\Gamma, x : A$	*le contexte étendu avec la déclaration que* x *a le type* A

De quels types parle-t-on ?

On va typer les fonctions avec des types de fonction $A \to B$

On va typer les paires avec des types produits $A \times B$

On va typer les entiers avec un type entier $\mathbb{N}$

$$\frac{}{\Gamma, x : A, \Delta \vdash A}\ \text{VARIABLE}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B}{\Gamma \vdash (t, u) : A \times B} \qquad \frac{\Gamma \vdash v : A \times B}{\Gamma \vdash \mathit{fst}(v) : A} \qquad \frac{\Gamma \vdash v : A \times B}{\Gamma \vdash \mathit{snd}(v) : B}$$

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash (\lambda x.t) : (A \to B)}$$

$$\frac{}{\Gamma \vdash 0 : \mathbb{N}} \qquad \frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash n + 1 : \mathbb{N}} \qquad \frac{\Gamma \vdash t : \mathbb{N} \qquad \Gamma \vdash u : A \qquad \Gamma, n : \mathbb{N}, x : A \vdash v : A}{\Gamma \vdash \mathit{rec}[0 \mapsto u \mid n + 1 \mapsto_x v]\ (t) : A}$$

Et maintenant, on peut changer notre regard !

$$\begin{array}{lcl} A \times B & = & A \wedge B \\ A \to B & = & A \Rightarrow B \\ \mathbb{N} & = & \mathbb{N} \end{array}$$

Notre λ-calcul peut être vu comme un langage de preuve !

$$\frac{}{\Gamma, x : A, \Delta \vdash A}\ \text{AXIOME}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B}{\Gamma \vdash (t, u) : A \wedge B} \qquad \frac{\Gamma \vdash v : A \wedge B}{\Gamma \vdash \mathit{fst}(v) : A} \qquad \frac{\Gamma \vdash v : A \wedge B}{\Gamma \vdash \mathit{snd}(v) : B}$$

$$\frac{\Gamma \vdash t : A \Rightarrow B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \Rightarrow B}$$

Et on peut aussi aller plus loin

Par exemple, on ajoutant un type $(n : \mathbb{N}) \times B(n)$ de *paires dépendantes*, ainsi qu'un type $(n : \mathbb{N}) \to B(n)$ de *fonctions dépendantes*, on obtient

$$\begin{aligned} (n : \mathbb{N}) \times B(n) &= \exists n^{\mathbb{N}}\, B(n) \\ (n : \mathbb{N}) \to B(n) &= \forall n^{\mathbb{N}}\, B(n) \end{aligned}$$

Conclusion

preuve	=	programme
proposition	=	type
hypothèse	=	variable
conjonction	=	type produit
implication	=	type de fonction
disjonction	=	type somme
récurrence	=	récursion
prouvabilité	=	habitabilité du type
lemme	=	fonction auxiliaire
logique propositionnelle	=	λ-calcul simplement typé
logique propositionnelle du 2e ordre	=	système F
logique des prédicats	=	λ-calcul à types dépendants

La théorie des types de Martin-Löf et les assistants à la preuve

Assistants à la preuve : des logiciels, tels que Coq, Lean, Agda, Mizar, Isabelle, HOL, qui permettent de prouver des énoncés mathématiques ou des spécifications de programme

Théorie des types de Martin-Löf : un formalisme né de la correspondance preuve-programme, dans les années 1970, qui est à la fois un système de preuve et un langage de programmation, et qui est au cœur des assistants Coq, Lean et Agda (alternatif à la théorie des ensembles)

Laissons le λ-calcul de côté pour l'instant

Nouveautés des 30 dernières années

raisonnement pas contradiction	$\simeq$	`goto` ou exceptions
axiome du choix	utilise des	affectations mémoires
l'égalité	$\simeq$	la notion de chemin dans un espace (HoTT)

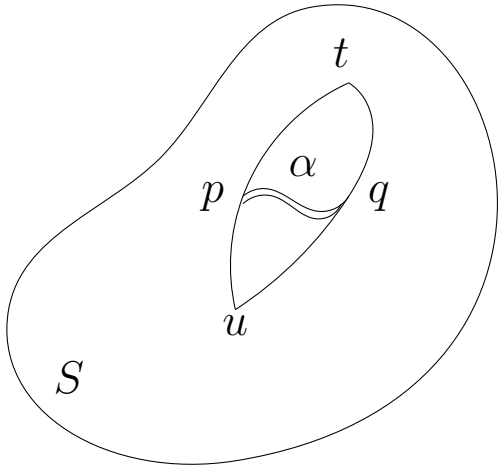
Plus généralement, par construction :

Tout axiome logique a un contenu calculatoire !

À homotopie près, tout espace, tout ensemble, peut être vu comme un type dans un langage de programmation !

Égalité et chemin (HoTT)

Un chemin p entre deux points t et u d'un espace topologique S, c'est la même chose qu'une preuve d'égalité de t et u:



Un chemin α entre deux chemins p et q reliant deux points t et u dans un espace topologique S, c'est la même chose qu'une preuve d'égalité de p et q:

$$\begin{array}{lcl} t, u & : & T \\ p, q & : & t =_T u \\ \alpha & : & p =_{(t=_T u)} q \end{array}$$

Vers une unité fondationnelle entre programmation, logique, algèbre et géométrie

La théorie des catégories est une théorie abstraite des structures mathématiques, avec des notions universelles de *type* telles que limites/colimites, algèbres/coalgèbres, ainsi qu'une notion d'adjonction généralisant la notion d'isormorphisme, ainsi aussi que de notions universelles de transformation appelées monade/comonade.

On peut connecter les langages de la logique, de la programmation et des catégories avec ce dictionnaire :

$A \Rightarrow B$	$=$	$A \to B$	$=$	B^A (exponentielle)
$A \wedge B$	$=$	$A \times B$	$=$	$A \times B$
$t = u$	$=$		$=$	espace des chemins entre t et u
connecteurs "positifs"	$=$	types inductifs	$=$	colimites / algèbres initiales
connecteurs "négatifs"	$=$	types coinductifs	$=$	limites / coalgèbres terminales
logique du 1er ordre	$=$	λ-calcul à types dépendants	$=$	hyperdoctrine
logique d'ordre supérieur	$=$	système F_ω	$\simeq$	tripos, topos élémentaire
	homotopy type theory		$\simeq$	∞-topos
logique polarisée	$=$	"call-by-push-value"	$=$	adjonction