Fast

8 décembre 2025

Je propose une théorie axiomatique de l'algèbre de composition des applications. Son originalité tient à son système de types, permis par des théorèmes exclusifs. En application, je présente un prototype en assembleur x86-64 d'un langage de programmation fonctionnel.

1 Introduction

Dans la théorie ensembliste de Zermelo-Fraenkel avec choix (ZFC).
THÉORÈME. Soient les relations d'équivalence $\sim$, $\approx$ sur les ensembles S, T respectivement, les ensembles quotients S_i, T_j et les surjections canoniques $i : S \to S_i$, $j : T \to T_j$. J'appelle homomorphisme $f : i \to j$ toute application $f : S \to T$ qui satisfait

$$\forall xy \in S \ (y \sim x \Rightarrow fy \approx fx).$$

Alors, avec la loi $\circ$ de composition des applications,

$$\exists! g : S_i \to T_j \ (g \circ i = j \circ f).$$

THÉORÈME. Soient les ensembles S, T, l'application $f : S \to T$ et l'équivalence $\approx$ sur T. Est équivalence sur S toute relation $\sim$ définie par

$$\forall xy \in S \ (y \sim x \Leftrightarrow fy \approx fx).$$

Dans le système axiomatique de von Neumann-Bernays-Gödel (NBG).
THÉORÈME. Je note $\mathrm{B} = \{\top\ \bot\}$ la classe des constantes booléennes, respectivement truisme et contradiction. Soient :

— la composition, opérateur binaire $\circ$ qui, aux relations $g : T \times S \to \mathrm{B}$ et $h : S \times R \to \mathrm{B}$, associe la relation $f = g \circ h : T \times R \to \mathrm{B}$ telle que

$$\forall x \in R \ \forall z \in T \ (zfx \Leftrightarrow \exists y \in S \ (z\, g\, y \wedge y\, h\, x)) ;$$

— les classes $\{i : \mathrm{Eq}\, C\}$ de toutes les relations d'équivalence i sur toutes les classes C ;
— les classes, $\{r : i \to j\}$, de toutes les relations $r : C \times D \to \mathrm{B}$ qui commutent avec les relations d'équivalences $i \in \mathrm{Eq}\, C$, $j \in \mathrm{Eq}\, D$ des classes C, D :

$$r \circ i = j \circ r ;$$

Alors je dispose de la catégorie M des applications, dont les relations d'équivalence sont les identités et les relations commutatives les morphismes :

$$\mathrm{Obj\ M} = \{i : \mathrm{Eq}\, C\},\ \mathrm{Mor\ M} = \{r : i \to j\}$$

Dans la catégorie Set, seule subsiste l'égalité, relation d'équivalence $=$, donc un unique objet par classe ; ses morphismes sont les seules applications qui commutent avec l'égalité. J'ai construit une catégorie M qui généralise la notion d'application, la catégorie Set et déclasse la primitive d'égalité — or la théorie des catégories déclasse de même $\in$, la relation d'appartenance… La question qui vient : dans quelle mesure les résultats connus dans Set, c'est-à-dire les mathématiques selon les axiomes ZFC, sont applicables à M. Mais l'étude, centrale dans ma *Théorie algébrique des mathématiques*, amène des développements qui ne font pas l'objet de cette présentation ; ci-après j'explore un système axiomatique simplifié et plus opératoire dans le cadre informatique.

2 Mathématique

Les entités mathématiques sont typées, autrement dit je déclare par « $\forall x : t$ » la variable x de type t. Je me dote d'une logique classique de prédicats à valeur dans le type B des booléens et munie des constantes de truisme et contradiction, respectivement $\top\bot : B$.

Chaque type t s'identifie à une relation d'équivalence, application $\sim : t \times t \to B$. Ainsi le type des booléens est assimilé à l'équivalence logique $\Leftrightarrow : B \times B \to B$. Une application est un homomorphisme de types : elle conserve, ou commute avec, les équivalences. Autrement dit, étant donnés les types s, t associés aux équivalences $\sim_s$, $\sim_t$, j'appelle application notée $f : s \to t$, ou plus rigoureusement $f : \sim_s \to \sim_t$, une relation $f : t \times s \to B$ qui vérifie

$$\forall ax : s \ \forall by : t \ (bfa \Rightarrow (yfx \Rightarrow (a \sim_s x \Rightarrow b \sim_t y))).$$

On peut donc, au sens de le relation d'équivalence $\sim_t$ identifier y et b à fx (ou fa) ; ainsi la conservation s'écrit plus lisiblement

$$\forall ax : s \ (a \sim_s x \Rightarrow fa \sim_t fx)$$

Todo. Il existe une équivalence sur les types et routines : tout deux sont typés.

Todo. Exemple des opérateurs booléens notés $\Leftrightarrow \to \Leftrightarrow$ ou encore $B \times B \to B$.

3 Informatique

L'originalité de la démarche est de définir théoriquement un type comme une équivalence sur les états de la machine. Ce faisant un type n'est pas une représentation en mémoire, c'est-à-dire une bijection entre un ensemble fini quelconque et N_n, ensemble des entiers $i < n \in \mathrm{N}$. En pratique le type spécifie complètement l'interface de (co)routines, qui sont la modélisation informatique, c'est-à-dire l'implémentation, des applications mathématiques.

Le paradigme est remarquable par l'abandon de l'approche orthodoxe de cloisonnement de la mémoire, avec des performances attendues supérieures aux standards actuels. L'interface et les données transmises des routines sont en mémoire partagée. En pratique, pour des raisons de concurrence d'accès, on implémentera un espace mémoire privé dévolu à chaque instance de routine.

Les entrées-sorties, sont associées à un type $\emptyset$ « nil », identifié à une relation toujours fausse. Pour tout type t, une routine $\emptyset \to t$ est productrice pure, une routine $t \to \emptyset$ consommatrice pure. Un programme est une routine évaluable c'est-à-dire de type $\emptyset \to \emptyset$. Une coroutine f à n arguments est une famille $\{c_{i,j}\}$ de blocs d'instructions. À chaque indice $i \in \mathrm{N}_n$ correspond un argument, l'appelant est par convention $i = 0$. Aux indices $j \in \{0, 1\}$ sont associés les appels terminaux (1er appel ou retour final) et les appels intermédiaires.

Car les types sont typés, une implémentation à typage dynamique est envisageable. Car les routines sont typées elle sont dynamiquement instanciables et manipulable, autrement dit la composition est une opération réalisable à l'exécution.