Thèse : Définition et implémentation de schémas calculatoires pour la preuve de programmes

Marc Coiffier

Contents

À faire :

- renseigner machine CAM
- développer le COq -> Mu
- (XAH pour taper les caractères unicode)
- Justifier LAP à partir de Ltac / SsReflect
  - Portée dynamique pourquoi ? En 1 et deux mots
- Réordonner : motivations concrètes d'abord

Ma thèse se faisant dans le cadre du project VOCAL (Verified OCAml Library, pour les intimes), j'ai tout d'abord découvert l'univers de la preuve formelle assistée par ordinateur, en me familiarisant avec le code dudit projet. Cette familiarisation a été principalement dirigée par la volonté d'y intégrer une preuve de la correction de l'algorithme MergeSort implémenté dans la librairie standard OCaml.

Une fois familiarisé avec les bases d'un assistant de preuve (Coq), et de son modèle de calcul, j'ai exploré différentes modélisations de théories mathématiques établies, afin de me faire une idée des limites que l'on pouvait rencontrer, et peut-être de trouver une contribution intéressante à apporter.

Limites du CIC simple

Lors de ces explorations, j'ai voulu modéliser des ω-catégories (si je ne m'abuse sur les nomenclatures), qui peuvent être définies intuitivement à l'aide des structures suivantes :

Soient O un type d'objets, et pour toute paire d'objets x et y, un type Mxy des 0-morphismes de x vers y. On aimerait définir les familles de types inductifs $V_n : Type$ et $M_n : V_n \to V_n \to Type$ (resp. des n-objets et n-morphismes de notre ω-catégorie), dotés des constructeurs suivants :

$$
\begin{aligned}
&v_0 : O \to V_0 \\
&v_S : \forall n (xy : V_n), M_n xy \to V_{Sn} \\
&m_0 : \forall (xy : O), Mxy \to M_0 (v_0 x)(v_0 y) \\
&m_S : \forall n (xyzt : V_n)(f : M_n xy)(g : M_n zt), M_n xz \to M_n yt \to M_{Sn}(v_S\, n\, x\, y\, f)(v_S\, n\, z\, t\, g)
\end{aligned}
$$

Coq (et d'autres assistants basés sur le CIC) ne permet pas la définition de familles mutuellement inductives (comme les V_n et M_n définis ci-dessus) si l'une des familles doit servir d'index à l'autre. Cette limite est justifiée par l'apparente impossibilité de faire référence à un constructeur dans le type d'un autre constructeur si les deux appartiennent à la même famille.

Il est curieux de trouver de telles limites sur les familles inductives mutuellement récursives, puisqu'il est plutôt simple – dans un contexte de types dépendants – d'en définir un encodage de Church ($O_n \equiv \forall(O : \mathbb{N} \to Type)(M : \forall i, Oi \to Oi \to Type)(o0 : o \to O0)(oS : ...)..., On$, par exemple).

Bien entendu, l'encodage de Church n'est pas une panacée, car il ne donne pas accès à un grand nombre des propriétés qui nous intéressent dans l'étude de valeurs inductives (la discrimination des constructeurs, pour n'en citer qu'une). Le CIC, et ses dérivés, ne disposent pas à ce jour de façon satisfaisante de définir ces familles.

Solution en travail : une autre extension du CoC, capable de gérer "plus" d'inductifs

Dans l'optique de combler cette lacune, je cherche à donner aux CoC la capacité de prouver des principes d'induction directement sur des encodages de Church, plutôt que sur des types "définis inductifs".

Pour ce faire, je me fie à une observation qui semble se vérifier en pratique : les principes d'induction ont une forme qui reflètent celle de l'encodage de Church de l'inductif sur lequel ils travaillent. Par exemple, le principe d'induction des booléens (`Boolean_rect` en Coq) a le type suivant :

$$\forall(P : Boolean \to Type), Ptrue \to Pfalse \to \forall(b : Boolean), Pb$$

Le paramètre b ne dépendant d'aucun des paramètres précédents, on peut le décaler à gauche du $\forall$ principal, et l'abstraire dans le contexte :

$$\forall(P : Boolean \to Type), Ptrue \to Pfalse \to Pb$$

À présent, si on regarde l'encodage de Church des mêmes booléens, on a le type suivant :

$$\forall(P : Type), P \to P \to P$$

Autrement dit, la même structure, moins un paramètres booléen ! Bien sûr, les choses sont plus compliquées dans le cas de types récursifs (tels les naturels), car il s'agit de mêler discrimination et récursion dans la sémantique opérationnelle des principes d'induction.

Étant donné un encodage de Church, on peut faire le travail inverse, et générer automatiquement le type du principe d'induction canonique de cet encodage. Dans ma thèse, j'introduis un nouvel opérateur, noté $\mu(x)$, dont c'est le rôle : étant donné une valeur inductive x, produire un terme dont la structure "reflète"

celle de x, en y ajoutant les paramètres nécessaires à la preuve du principe d'induction spécialisé sur x.

Un interpréteur de ce nouveau modèle de calcul – CoC + μ, appelé le Calcul des Constructions Prismatiques pour rappeler la métaphore des "reflets with extra steps" – est rendu disponible sur une plateforme Web interactive, ainsi que sous une forme plus classique, à l'adresse https://wiqee.curly-lang.org .