Proposition de stage de DEA dans le domaine des Isomorphismes de Types

Le "Tarski's High School Algebra Problem" et les types recursifs

Encadrants:

Roberto Di Cosmo Université de Paris 7 et INRIA Roquencourt
`http://www.dicosmo.org`, E-mail: `roberto@dicosmo.org`
Tel: 01 44 27 86 55.

Marcelo Fiore University of Cambridge
`http://http://www.cl.cam.ac.uk/~mpf23`.

Laboratoires d'accueil: PPS (Université de Paris 7) et Computer Laboratory (University of Cambridge).

Cadre de la recherche:
L'étude des "isomorphismes de types", c'est l'étude de ces types de données qui sont codifiables l'un dans l'autre sans perdre de l'information.

L'idée de base est très simple: une donnée a de type A peut être "codéé' par un programme (ou fonction) f sous la forme d'une donnée $f(a)$ de type B sans perte d'information si il est possible après de "décoder" ce $f(a)$ à l'aide d'un programme (ou fonction) f^{-1} tel que $f^{-1}(f(a))$ soit a lui-même. Or, on dit que A est isomorphe à B si toute donné de A peut être codée dans B sans perte d'information et viceversa. Autrement dit, on demande que le diagramme suivant commute:

$$\begin{tikzcd} A \arrow[loop left, "id_A"] \arrow[r, bend left, "f"] & B \arrow[l, bend left, "g"] \arrow[loop right, "id_B"] \end{tikzcd}$$

Dans le langage de la théorie des catégories, ces données sont des *objets isomorphes* (une notion absolument de base de cette théorie, et l'intérêt est celui de "caracteriser" tous les objets isomorphes dans certaines classes de catégories, comme cela a été fait par Mac Lane sur les catégories monoidales au debut des années soixantedix), par contre cette notion est plus récente pour la théorie de la preuve, et correspond à l'equivalence "forte " des propositions.

Un exemple simple de ces isomorphismes qui est très connu dans la programmation fonctionnel prend son nom de Curry:

$$A \to (B \to C) \cong (A \times B) \to C$$

Cela peut se lire comme "une fonction qui attend deux arguments en série de type A et B et renvoie un résultat de type C peut etre transformée en une fonction qui prend le couple de type $A \times B$ de ces arguments et renvoie le même résultat C (et vice versa)".

Si on se souvient de la formule qui donne le numéro de fonctions entre deux ensembles (ou types), il est facile de reconnaître là une identitée numerique bien connue: $C^{A^B} = C^{A \times B}$ (ce qui ouvre aussi les portes à des interactions avec la théorie des nombres, que n'a pas été developpé en profondeur pour l'instant). Comme dans le cas des nombres, quand il se présente ce phénomène en programmation fonctionnelle, les deux "types de donnée " A et B ne sont que différentes représentations d'une même entité sousjacente, et il faudrait soit les identifier, soit savoir que si on cherche quelque chose du premier type, elle peut se manifester avec le deuxième type.

Ces deux alternatives correspondent effectivement aux applications les plus répandues des isomorphismes de type:

- la recherche de fonctions dans des librairies fonctionnelles, en utilisant le type comme clé de recherche,
- l'identification automatique, à la compilation, de tous les types isomorphes possibles pour chaque fonction

Pour cela, il est souhaitable de pouvoir connaître *tous* les isomorphismes valides pour un langage et un système de types donnés. Un grand nombre de résultats connus sont exposés dans [4], mais il y au moins deux domaines dans lesquels beaucoup de travail reste à faire, et qui donnent lieu aux deux sujets de stage suivants:

Sujet 1: le "Tarski's High School Algebra Problem" pour $< I\!N, \cdot, \uparrow, +, 1, 0 >$

Tarski a posé en 1969 le problème de détérminer si les "egalités du lycée" que nous tous connaissons sont suffisantes pour prouver toutes les identités entre expressions entiéres que l'on peut écrire dans un langage fixé. Ce problème a occupé depuis un grand nombre d'illustres mathématiciens.
Si on considère les types construits à partir de la flèche, le produit, une constante "unit" et des variables seulement, il existe une remarquable coincidence entre les isomorphismes de types et les égalités entières dans $< I\!N, \cdot, \uparrow, 1 >$ (voir `http://www.dicosmo.org/Tarksi` pour plus de détails). Si on ajoute au langage la somme, on ne sait plus si la coincidence reste valide, mais on a recemment pu montrer que une remarquable famille d'identités entières [5] est aussi une famille d'isomorphismes de types [3], et aussi que les isomorphismes avec somme ne sont pas finiment axiomatisables.
On vous propose d'analyser en détail la preuve de Gurevich de non finie axiomatisabilité des identités entières, pour pouvoir obtenir un des résultats suivants:

- l'automatisation de la génération des contre-modèles décrite par Gurevich
- l'extension du Théorème de Gurevich au cas du langage incluant la constante 0
- l'étude de la décidabilité et la complexité de différentes extensions propres des "égalités du lycée"

Ce travail se place à la frontière entre théorie des nombres, logique et informatique théorique, et il demande un certain goût pour les mathématiques, les modèles de l'arithmetique et la théorie de la preuve.

Sujet 2: isomorphismes de types récursifs

Les types récursifs sont essentiels pour la définitions des types des données, et dans la théorie de la programmation orientée objets; ils ont fait l'objet d'études approfondies [2, 1] et il est très naturel de se poser la question des isomorphismes valides dans différentes systèmes de types une fois que l'on permet la formation de types récursifs.
On s'intéresse à caractériser, dans un prémier temps, *tous* les isomorphismes de types valides dans le système de types avec sommes, produits et types récursifs. Nous conjecturons que les isomorphismes naturels (associativite, commutativite et

distributivité) sont complets dans ce cadre une fois que l'on s'autorise l'axiome $\mu\alpha.F(\alpha) = F(\mu\alpha.F(\alpha))$ et la règle:

$$\frac{\tau = F(\tau)}{\tau = \mu\alpha.F(\alpha)}$$

Dans un deuxième temps, il sera intéressant d'étudier les complexités qui sont introduite par le constructeur des types flèches.

On pourra ensuite proposer un algorithme de décision pour lisomorphismes, dont on étudiera la complexité et réalisera une implémentation.

References

[1] M. Abadi and M. P. Fiore. Syntactic considerations on recursive types. In *Proceedings of the Symposium on Logic in Computer Science (LICS)*, number 11, pages 242–252, New Brunswick, New Jersey, 1996. IEEE computer society Press.

[2] R. M. Amadio and L. Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, September 1993.

[3] V. Balat, R. Di Cosmo, and M. Fiore. Remarks on isomorphisms in typed lambda calculi with empty and sum type. In *LICS*. IEEE, July 2002.

[4] R. Di Cosmo. *Isomorphisms of types: from λ-calculus to information retrieval and language design*. Birkhauser, 1995.

[5] R. Gurevič. Equational theory of positive numbers with exponentiation is not finitely axiomatizable. *Annals of Pure and Applied Logic*, 49:1–30, 1990.