Proposition de TER pour la Maîtrise d'Informatique : sujets liés à la Compilation et la programmation fonctionnelle

Encadrants :

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

Laboratoires d'accueil : PPS (Université de Paris 7).

Cadre de la recherche : Dans le domaine de la compilation, et de la programmation fonctionnelle, ils existent des nombreux travaux de recherche recents qui, sans être d'une très grande difficulté d'accès, se prêtent bien à la réalisation d'un TER sous forme de lecture et explication d'un article, plus un petit travail d'implantation qui prouve que l'on a bien compris l'article.

Voici une liste non exhaustive des sujets proposes :

Calcul incremental : on connaît en litérature un ensemble de techniques qui se prêtent à l'écriture d'algorithmes "incrementaux", dans le sens que si une partie seulement des données en entrée est modifiée, alors on ne recalcul pas entiérement le résultat, mais seulement la partie qui est affectée par le changement. Par exemple, si on cherche le minimum d'une liste par l'agorithme naturel, et que l'on a déjà calculé le minimum 1 sur la liste `l` qui vaut

$$[1\,;2\,;3\,;2\,;3\,;7\,;10\,;12\,;3\,;1]$$

et maintenant on *modifie* `l` en ajoutant à la fin de la liste un élément de valeur 12, on souhaiterait n'avoir à calculer que le minimum entre 1 et 12, sans reéxaminer toute la liste `l`, et cela sans avoir à trop modifier la structure du programme qui calcul le minimum.
Une technique bien connue est celle dite de `change propagation`, qui prévoit que l'utilisateur manipule les tructures modifiables seulement à travers des primitives particulières, ce qui permet d'effectuer le calcul incremental sans besoin d'autre aide du programmeur.
Le but de ce TER est de lire l'article "Monads for incremental computing", disponible sur `http://www.cse.ogi.edu/~magnus/papers/icfp-2002.pdf`, et de réaliser une petite librairie en OCaml en se basant sur la libreiaire Haskell presentée dans l'article.

Utilisation du partage dans les structures fonctionnelles Dans l'article "Trouble shared is trouble halved", paru comme "Functional Pearl" dans les "Proceedings of the ACM SIGPLAN workshop on Haskell" en 2003,

Richard Bird et Ralf Hinze proposent un usage originel des structures partagées que l'on crée souvent en programmation fonctionnel, pour réaliser de la `mémoisation` efficace. La mémoisation est un procédé dual du `change propagation` et cet article l'explore d'un point de vue nouveau.
On vous demande de lire et comprendre l'article et de traduire en OCaml l'un des deux exemples présentés.