

# Moteur d'optimisation de programmes fonctionnelles

Contact: [Pierre-Evariste Dagand](#)

## Introduction générale

La programmation fonctionnelle offre la possibilité d'écrire des programmes de haut-niveau. Les programmeurs fonctionnels peuvent ainsi communiquer plus naturellement leurs intentions à la fois au compilateur mais également aux mainteneurs futurs, dans un style efficace et épuré proche des mathématiques. En conséquence, un compilateur de langage fonctionnel doit être en mesure de transformer agressivement les programmes afin de permettre leur exécution efficace.

Dans ce projet long, nous proposons d'étudier et d'implémenter un moteur d'optimisation pour un fragment du langage Core de [GHC](#). Il s'agit d'un langage pure (sans effet de bord), offrant des types algébriques, du typage explicite de second ordre et une sémantique paresseuse.

L'optimiseur fut le sujet de la [thèse de Andre Santos](#) ainsi que 3 publications de conférence :

- [Let-floating: Moving Bindings to Give Faster Programs](#); Peyton Jones, Partain et Santos
- [A Transformation-Based Optimiser for Haskell](#), Peyton Jones et Santos
- [Secrets of the Glasgow Haskell Compiler inliner](#); Peyton Jones et Marlow

L'objectif est de réduire le nombre de déclarations (`let`) ainsi que le nombre de `match` tout en évitant une explosion combinatoire de la taille du code généré. L'optimiseur fonctionne de source à source, en préservant le typage : il consomme un programme Core bien typé et produit un programme Core bien typé.

## Objectifs

L'objectif de ce projet long est d'apprendre les différentes techniques (et astuces) d'implémentation des langages fonctionnels. Il s'agira en particulier de modéliser une syntaxe avec lieux (`let`, `lambda`), d'implémenter un algorithme de typage et d'effectuer des transformations symboliques (optimisations) sur les termes de ce langage. Le travail sera effectué en OCaml.

Si le temps le permet, on abordera des optimisations avancées, en lien avec le flot de contrôle des programmes fonctionnels et basé sur la notion de ["join point"](#).

## Calendrier

Le calendrier prévisionnel est divisé en 3 phases d'approximativement 3 semaines chacune :

- modélisation du langage source en OCaml
- implémentation d'un vérificateur de type
- implémentation de transformations de programmes (inlining, let-floating)

## Références

- [Compilation by transformation for non-strict functional languages](#); Santos
- [Let-floating: Moving Bindings to Give Faster Programs](#); Peyton Jones, Partain et Santos
- [A Transformation-Based Optimiser for Haskell](#), Peyton Jones et Santos
- [Secrets of the Glasgow Haskell Compiler inliner](#); Peyton Jones et Marlow
- [Compiling without continuations](#); Maurer, Downen, Ariola et Peyton Jones
- [Compiling with continuations, or without? whatever](#); Cong, Osvald, Essertel et Rompf