# Heuristique d'inlining complexe

Adrien Simonnet

Avril - Septembre 2023

**Abstract**

Ceci constitue la synthèse (bilan) de mon stage de fin de cursus Science et Technologie du Logiciel (STL) à Sorbonne Université réalisé chez OCamlPro au sein de l'équipe flambda. Ce stage a consisté à proposer des heuristiques d'inlining pour le compilateur du langage OCaml, spécialité de l'entreprise. Découvrir et travailler sur un compilateur complexe comme celui-ci n'a pas été jugé envisageable par mes tuteurs de stage, Vincent Laviron et Pierre Chambart, c'est la raison pour laquelle j'ai évolué sur un langage "jouet". Mon stage se terminant après la soutenance, mon rapport rend compte essentiellement de mon travail sur les différentes représentations intermédiaires et analyses nécessaires à l'inlining. J'apporte néanmoins quelques idées d'heuristiques que je vais être amener à étudier en détails d'ici la fin du stage.

L'inlining consiste à injecter le corps d'une fonction en lieu et place d'un appel vers celle-ci dans l'objectif d'accélérer l'exécution du code (ou dans certains cas en diminuer sa taille). Néanmoins copier le corps d'une fonction peut faire augmenter la taille du code et conduire à de grosses pertes de performances lorsque certains seuils sont franchis. Vu la difficulté que serait de faire une analyse approfondie du meilleur choix d'inlining l'idée a été de se concentrer sur des heuristiques qui fonctionneront bien la plupart du temps. Cette optimisation est actuellement effectuée dans le compilateur natif par la série d'optimisations flambda, qui sera plus tard remplacé par flambda2 actuellement en développement. Le langage "jouet" sur lequel j'ai travaillé n'est rien d'autre que la partie intéressante d'OCaml pour y appliquer l'inlining.

J'ai beaucoup travaillé sur les représentations intermédiaires essentielles pour à la fois réaliser la meilleure analyse possible et me permettre d'avoir tous les outils en main pour inliner. J'ai commencé avec 2 représentations, l'AST et la forme CPS (désormais le CFG) puis j'en ai créé 2 nouvelles, l'AST rafraîchi et le CFG exécutable pour alléger les différentes transformations. Toutes ont beaucoup évoluées pour me rapprocher de la sémantique la plus simple et la plus natuelle, ce qui a demandé beaucoup de choix pas toujours évidents. Le fait de les avoir formalisées dans le rapport m'a permis de corriger certaines erreurs qui auraient été difficile de détecter dans le code.

J'ai évidemment passé beaucoup de temps sur l'interprétation abstraite et rencontré de nombreuses difficultés dont certaines n'ont pas été surmontées. En particulier certaines erreurs rencontrées avec les abstractions de pile n-CFA n'apparaissant pas avec l'abstraction

par motifs laissent supposer que je n'exploite pas toujours de la bonne manière la pile abstraite dans mon algorithme. J'ai également des erreurs qui peuvent apparaître lorsque je réalise au moins 2 analyses consécutives mais uniquement avec une étape de propagation entre les deux. Exceptées ces deux problèmes rencontrés, l'analyse tient la plupart de ses promesses pour ce que j'attends d'elle, c'est à dire me donner de manière la plus exhaustive possible les blocs candidats pour les appels indirects, et d'assez bonnes précisions sur les valeurs taggées. Il est néanmoins très important de noter que durant ce stage j'ai fait totalement abstraction de la complexité de mon analyse. J'essaye de garantir de mon mieux qu'elle termine mais il est fort probable qu'elle soit exponentielle et la toute petite taille de mes tests ne me permet de m'en faire une idée.

Concernant l'inlining, d'après les premiers tests que j'ai réalisés, il apparaît que spécialiser systématiquement les blocs de très petite taille (de l'ordre d'1 instruction) permet à la fois de réduire grandement la taille du code tout en diminuant les instructions exécutées lors de l'évaluation. Augmenter légèrement ce seuil (de l'ordre de 2-3 instructions) permet généralement de gagner quelques instructions exécutées mais augmente aussi la taille du code, ce qui fait que les bénéfices sont durs à déterminer. Au delà de quelques instructions, sans autre forme d'heuristiques comme par exemple l'élimination de branchements en connaissant le motif, il semblerait que cela n'ait plus aucun ou très peu d'impact sur le nombre d'instructions exécutées tout en augmentant considérablement la taille du code, ce qui n'est évidemment pas souhaitable.

Je suis assez satisfait de ce que j'ai appris et produit durant ce stage, à la fois d'un point de vue théorique et pratique. J'ai sous la main un ensemble de phases de compilations cohérentes pour un petit langage certes mais très expressif et je pense être capable sans trop de difficultés d'implémenter et tester de nouvelles heuristiques d'inlining. Pour mener au mieux de telles expérimentations il me sera par contre nécessaire de disposer d'une base de tests exhaustifs et d'en assurer l'automatisation, ce qui en soit ne sera probablement pas la chose la plus aisée de ce stage.