Sujet de Stage M2 MPRI — 2024-2025

Optimisation interactive de programmes OCaml via des transformations source-à-source

Encadrant	Arthur Charguéraud, arthur.chargueraud@inria.fr, Directeur de recherche Inria
Lieu	Équipe Inria Camus, Laboratoire ICube, Strasbourg campus d'Illkirch

Sujet

Le langage OCaml permet de décrire des programmes à haut niveau. Les bénéfices de la programmation haut niveau sont bien connus : gain de productivité, meilleure réutilisabilité du code, plus de facilité pour vérifier formellement le code. La contrepartie est également bien connue : des performances moindres que pour un code bas niveau correspondant, typiquement exprimé en langage C. La question qui se pose est donc de savoir comment l'on pourrait bénéficier du meilleur des deux mondes.

L'idée centrale du projet OptiTrust est d'optimiser des programmes à travers une suite de transformations de code pilotées interactivement par le programmeur. Concrètement, le programmeur indique, à chaque étape, quelle transformation de code appliquer, et à quel endroit l'appliquer. Ces transformations sont décrites dans un script que l'on développe interactivement. À chaque étape, le programmeur peut visualiser le *diff* associé à la transformation, et continue à manipuler un code raisonnablement lisible sur lequel continuer à appliquer des optimisations. Pour établir un parallèle, là où Coq est un *assistant de preuves interactif*, OptiTrust est un *assistant de compilation interactif*.

Traditionnement, un programmeur écrit du code en réfléchissant déjà à implémenter certaines optimisations au niveau du code, et en passant du temps à essayer de déviner ce que les heuristiques du compilateur vont réussir à faire. Au contraire, avec OptiTrust, le programmeur peut d'abord se concentrer pleinement sur l'implémentation de ce que le code doit faire, puis se concentrer pleinement sur l'optimisation de ce code, et ce avec un degré de contrôle inégalé.

Une version du framework OptiTrust a déjà été développé au cours de quatre dernières années [1] pour appliquer des transformations sur du code C. L'objectif de ce stage est de généraliser le framework afin de permettre d'optimiser du code OCaml. Une bonne partie du framework, qui s'appuie en interne sur un λ-calcul impératif, pourra être réutilisée. Le but du stage est de développer les transformations qui sont spécifiques à l'optimisation de code haut niveau. Au-delà d'OCaml, on pourra aussi s'intéresser au raffinement de code non-exécutable, comme des spécifications Coq, vers du code exécutable haute performance qui satisfait, de manière inhérente, les spécifications de départ.

Un aspect essentiel des transformations de code est de s'assurer qu'elles préservent la sémantique du code. Pour garantir cela, OptiTrust s'appuie sur des spécifications et invariants exprimés en logique de séparation. Pour chaque nouvelle transformation, il faudra donc mettre au point des critères capturant une condition suffisante pour la préservation sémantique. En résumé, le stage consistera, pour une collection de transformations, à trouver des critères de préservation, en prouver la correction, les implémenter, et vérifier leur application en pratique sur des études de cas.

Études de cas

Comme exemples de transformations à viser, on peut citer : la mise à plat des structures algébriques (records et constructeurs de données) afin de minimiser le nombre d'indirections, le tassement de plusieurs champs de records contenant des petits entiers sur un seul champ, l'inlining des itérateurs d'ordre

1. https://www.chargueraud.org/research/2024/optitrust/optitrust.pdf

supérieur et leur transformation en des boucles, la fusion d'une suite d'appels à la fonction `map` sur des listes ou des arbres, le déroulement de cas de base pour des fonctions récursives, l'optimisation du pattern-matching, ou encore l'élimination de certaines allocations sur le tas en utilisant l'espace de pile. Toutes ces transformations sont absolument essentielles pour l'optimisation de code fonctionnel.

Concernant les applications, on pourra en considérer deux ou trois parmi les pistes suivantes.

— Génération de code pour représenter des listes avec cellules à nombre variable d'éléments, comme décrit dans cet exposé. Le code visé n'est pas vraiment réaliste à écrire à la main, ni à faire générer par un compilateur automatique, néanmoins les expériences préliminaires montrent qu'il peut donner des performances bien supérieures à celles des structures canoniques de la bibliothèque standard d'OCaml.

— Génération de code performant pour la structure de séquence "transiente" Sek, décrite dans le même exposé et dont le paquet est diffusé ici. Cette structure d'arbre avancée fournit une représentation pour les chaines de caractères avec des opérations de complexité asymptotique très attrayante. Il reste à obtenir de bons facteurs constants. Là encore, il n'est pas réaliste d'appliquer les optimisations requises à la main ni avec un outil automatiqu ; en revanche, on a bon espoir d'y arriver avec OptiTrust.

— Génération de code performant pour le langage Catala qui permet de décrire des textes de lois, et notamment les règles de calcul des impôts et des aides sociales. Les développeurs de Catala ont déjà élaboré une traduction de Catala vers le λ-calcul impératif. Ce qu'il reste à faire, c'est montrer comment utiliser OptiTrust pour produire du code haute performance permettant d'évaluer les impôts pour des dizaines de millions de contribuables, puis montrer comment le travail d'optimisation peut être réutilisé lorsque le ministère ou les parlementaires commandent une simulation visant à mesurer l'impact d'un projet de texte de loi sur l'ensemble des contribuables.

— Génération de code OCaml exécutable performance à partir de spécifications Coq non exécutables. Concrètement, le raffinement d'expression logiques (propositions) vers des fonctions calculatoires (fonctions booléenes) permet de passer de spécifications formelles à une implémentation. Dans les cas très simples, le processus peut être automatisé. Dans les cas les plus complexes, il faut exhiber un algorithme non trivial qui implémente la spécification. Mais pour tous les cas situés entre ces deux extrêmes, on peut espérer qu'un simple raffinement piloté interactivement permettrait d'obtenir, à moindre frais, une implémentation efficace qui satisfait la spécification formelle.

Profil de l'étudiant(e) pour ce stage

Pour mener à bien ce stage, l'étudiant(e) devra avoir un fort attrait pour le langage OCaml, avoir des bases en logique de séparation, et un intérêt pour la performance des programmes.

Équipe d'accueil

L'équipe Camus s'intéresse de manière générale à la compilation, à l'optimisation, à la vérification de programmes. Par exemple, les membres de l'équipe travaillent sur des techniques *polyédriques* pour optimiser des nids de boucles critiques ; sur la production de codes efficaces pour architectures modernes, notamment pour des simulations numériques (avec MLIR) ; sur le *scheduling* de programmes parallèles distribués ou multicœurs, que ce soit dans le modèle *fork-join* (Cilk) ou dans le modèle *graphe de tâches* (StarPU) ; sur la vérification interactive de programmes (CFML) ; et sur l'optimisation de programmes via des transformations source-à-source (OptiTrust).

Modalités pratiques

Le stage sera rémunéré selon la gratification standard. La résidence du Crous d'Illkirch propose des studios à très bon prix, à 5min du labo et 20min du centre ville—il suffit de réserver assez tôt.

Si le stage donne de bons résultats, il pourrait se poursuivre sur une thèse. Cette thèse pourra être orientée, selon le souhait de l'étudiant(e), soit plutôt vers les aspects "génération de code haute performance", soit plutôt vers les aspects "garanties de correction formalisées en Coq".