Stage: Recherche de fonctions par types

Gabriel RADANNE, Inria CASH/LIP

2020

1 Contexte

Une question récurrente pour le programmeur est « Quelle fonction dois-je utiliser ? ». Souvent, le programmeur à une idée précise du type de la fonction désirée : « Cette fonction reçoit au moins une voiture, une date et renvoie un booléen ». Le programmeur va alors chercher à différents endroits spécifiques (le module `Voiture` par exemple). Cette tâche est rendue plus difficile avec la taille de l'écosystème et la complexité du langage. On pourrait imaginer un outil qui, étant donné un type, cherche automatiquement les fonctions appropriées. Un tel outil devrait alors ignorer des détails ennuyeux tels l'ordre et le nombre d'arguments et se concentrer sur la forme du type. Les résultats d'une telle recherche se doivent d'être corrects (la fonction respecte la signature) et complets (on trouve toutes les fonctions) et rapides à calculer.

Plus concrètement, dans le contexte d'un langage tel OCaml, on pourrait vouloir chercher une fonction de type `int * voiture list -> voiture`. Cette recherche pourra trouver des fonctions de types `int -> voiture list -> voiture`, ou encore la fonction `List.nth`, de type `'a list -> int -> 'a`. On souhaite ainsi à la fois trouver des fonctions plus générales, plus spécialisées, et permettre certaines transformations tel que réordonner les arguments.

Cette idée a été initiée il y a 25 ans sous le nom de « recherche modulo isomorphisme de types » [6, 4]. Malheureusement, les algorithmes développés dans ce but étaient appropriés pour les écosystèmes des langages typés de l'époque qui étaient petits. Ces algorithmes ne passent pas à l'échelle sur les écosystèmes modernes et les approches plus récentes, tel Hoogle[1], ne sont ni correctes ni complètes.

Cette notion de recherche est également particulièrement utile dans le cas des langages de preuves. En effet, via l'isomorphisme de Curry-Howard, elle permet de rechercher des preuves, ce qui a déjà été largement mis en application dans l'écosystème Coq. De plus, les langages de preuves font souvent un usage très large de la synthèse de code dirigée par les types, qui est un cadre plus général de cette idée.

2 Objectifs du Stage

Le but de ce stage est d'implémenter un outil de recherche par isomorphisme de type suffisamment efficace pour une recherche dans un écosystème complet, par exemple celui d'OCaml. On peut distinguer trois volets.

Un premier volet consiste en l'implémentation d'une procédure simple, non optimisée, sur un système de type simple tel ML. Nous utiliserons cet outil afin d'évaluer l'efficacité pratique des algorithmes d'unification modulo isomorphisme sur les fonctions présentes dans l'écosystème OCaml. Cette partie s'appuiera sur les travaux déjà effectués sur l'unification modulo isomorphisme [2, 3]. Ce prototype permettra également de construire un ensemble de query servant de benchmark pour évaluer l'efficacité et l'expressivité de l'outil.

Dans un deuxième temps, on considérera différentes approches pour optimiser cette recherche, notamment en utilisant des techniques d'indexation utilisées dans les bases de données et dans la synthèse de code.

Enfin, on pourra étendre les fonctionnalités de types supportées par notre outil de recherche, par exemple aux variants polymorphes [5].

Le stage aura pour objectif principal d'obtenir un outil capable d'effectuer une recherche sur l'ensemble de l'écosystème OCaml (via OPAM) en un temps raisonnable. Ceci correspond au volet 1 et une partie du volet 2. Dans tous les cas, le but est de faire avancer les aspects théoriques et pratiques main dans la main : toute implémentation doit se baser sur des solides fondements théoriques et la théorie doit s'inspirer des besoins pratiques découverts durant l'implémentation.

LIP – UMR CNRS / ENS Lyon / UCB Lyon 1 / Inria – 69007 Lyon
E-mail : gabriel.radanne@inria.fr

3 Profil recherché

Le candidat devra idéalement être familier avec les approches formelles des langages de programmation, notamment les systèmes de types et les bases de logiques.

Côté pratique, une expérience du développement logiciel est bienvenue, ainsi qu'une connaissance minimale des outils de collaboration tels `git`. Une connaissance du langage de programmation OCaml est nécessaire à la poursuite du stage.

Références

[1] Hoogle. URL https://hoogle.haskell.org/.

[2] Alexandre Boudet. Competing for the AC-Unification Race. *J. Autom. Reasoning*, 11(2) :185–212, 1993. doi : 10.1007/BF00881905. URL https://doi.org/10.1007/BF00881905.

[3] Evelyne Contejean and Hervé Devie. An efficient incremental algorithm for solving systems of linear diophantine equations. *Inf. Comput.*, 113(1) :143–172, 1994. doi : 10.1006/inco.1994.1067. URL https://doi.org/10.1006/inco.1994.1067.

[4] Roberto Di Cosmo. A short survey of isomorphisms of types. *Mathematical Structures in Computer Science*, 15(5) :825–838, 2005. doi : 10.1017/S0960129505004871. URL https://doi.org/10.1017/S0960129505004871.

[5] Jacques Garrigue. Programming with polymorphic variants. In *ML Workshop*, volume 13, page 7. Baltimore, 1998.

[6] Mikael Rittri. Using types as search keys in function libraries. *J. Funct. Program.*, 1(1) :71–89, 1991. doi : 10.1017/S095679680000006X. URL https://doi.org/10.1017/S095679680000006X.