Boîte à outils pour l'inférence de type

Sujet de TRE (M1)
Yann Régis-Gianas (yrg@pps.jussieu.fr)
Université Paris 7

4 février 2009

1 Contexte

Le typage statique permet de garantir la sûreté d'exécution des programmes grâce à deux propriétés fondamentales : (i) le type d'une expression est invariant au cours de son évaluation, (ii) toute expression bien typée est soit une valeur valide, soit une expression non bloquée (*ie* qui peut être évaluée).

Dans de nombreux langages, la forme d'une expression et son contexte d'utilisation sont suffisants pour déduire son type canonique, c'est-à-dire un type plus général que tous ceux que l'on peut donner à cette expression. Cette remarque a donné lieu à la conception d'algorithmes d'inférence de type. Parmi eux, l'algorithme $\mathcal{W}$ [1] a été popularisé par son implémentation dans les langages de la famille ML (CAML, HASKELL, ...) et s'est révélé être un point d'équilibre entre simplicité, généralité et efficacité.

Une exposition moderne de cet algorithme consiste à le reformuler comme la résolution de contraintes d'égalités sous préfixes mixtes [3, 2]. Pour déterminer le type σ le plus général d'une expression e, on produit une contrainte $(\!|\, \Gamma \vdash e : \sigma \,|\!)$ qui signifie que, dans le contexte Γ, l'expression e admet le (schéma de) type σ. Un solveur générique de contraintes détermine alors la satisfiabilité de cette contrainte.

En réutilisant ce solveur de contraintes, on se dispense d'un travail fastidieux et technique lorsque l'on veut implanter l'inférence de type (similaire à celle de ML) pour un nouveau langage de programmation. En effet, il suffit uniquement de spécifier la façon dont ces contraintes sont produites à partir des expressions du langage et d'utiliser ensuite une « boîte à outils » générale de résolution de contraintes.

2 Problématique

Si l'équivalence entre inférence de types et résolution de contrainte a déjà été formalisée et implantée, aucune boîte à outils pour l'inférence des types n'a été réalisée jusqu'à maintenant. Deux problèmes doivent encore être résolus :

(i) Savoir si la contrainte de typage est satisfiable ou non n'est pas suffisant pour reconstruire un terme explicitement typé à partir d'un terme implicitement typé. Comment doit-on augmenter le langage de contraintes pour que la résolution des contraintes produisent non seulement un booléen mais aussi une expression du langage de programmation ?

(ii) Quand une expression est mal typée, dans quelle mesure des messages d'erreur peuvent être élaborés à partir de la contrainte de types et d'une expression partiellement typée ?

3 Travail demandé

L'objectif de ce projet est de concevoir, dans le langage O'CAML, une boîte à outils pour l'inférence des types. On se focalisera d'abord sur une simplification du langage de contraintes habituel de façon à obtenir une première version répondant aux deux questions précédentes dans le cadre des langages de programmation monomorphes et non extensibles. On généralisera ensuite le cadre aux langages polymorphes et on étendra enfin, si le temps le permet, l'algèbre de types aux rangées et aux fonctions injectives par exemple.

Les étapes du stage pourront suivre la progression suivante :

❶ Familiarisation avec la littérature et les implantations existantes.

❷ Conception et spécification de l'architecture générale.

❸ Implantation de la restriction aux langages monomorphes.

❹ Application à un langage monomorphe.

❺ Vérification des hypothèses (nouvelle itération si besoin).

❻ Généralisation aux langages à polymorphisme paramétrique.

❼ Application à ML.

❽ Vérification de la robustesse de la méthode dans le cadre polymorphe (nouvelle itération si besoin).

❾ Extension aux rangées, aux familles inductives de types, à l'inférence partielle ou locale, ...

4 Prérequis

Une bonne maîtrise du langage de programmation O'Caml est conseillée (ainsi que des outils de développement TRAC, SVN, ...). Une bonne compréhension du typage de ML est un plus mais, dans tous les cas, j'assurerai une mise à niveau sur les problèmes d'inférence des types auprès de l'étudiant.

Références

[1] Luis Damas and Robin Milner. Principal type-schemes for functional programs. pages 207–212, 1982.

[2] Martin Odersky, Martin Sulzmann, and Martin Wehr. Type inference with constrained types. 5(1) :35–55, 1999.

[3] François Pottier and Didier Rémy. The essence of ML type inference. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 10, pages 389–489. 2005.