Polymorphisme d'ordre ω avec inférence partielle mis en pratique

Sujet proposé par Didier Rémy* à Boris Yakobowski

29 septembre 2004

Contexte

Dans le domaine de la programmation, les systèmes de types permettent d'écrire des programmes plus sûrs ; en détectant statiquement certaines classes d'erreurs, ils évitent des erreurs à l'exécution. Toutefois, un bon système de types demande un compromis subtil entre son expressivité et sa complexité.

Parmi les langages communément utilisés, ceux de la famille ML représentent un excellent compromis : le système de types est relativement expressif, et l'inférence de types (qui évite au programmeur d'avoir à écrire les types lui-même) est décidable ; ces deux points sont probablement pour beaucoup dans le succès des langages de cette famille (OCaml, SML...).

Un utilisateur expérimenté de ML se heurte néanmoins rapidement aux restrictions imposées par le système de types, notamment le fait que toutes les variables de types quantifiées doivent l'être en tête du type. Ainsi, la fonction `fun f => (f 1, f true)` n'est pas typable, car la fonction `f` reçue en argument ne peut pas être rendue suffisamment polymorphe. De fait, les évolutions récentes dans les langages de la famille ML et leurs applications font ressortir le besoin croissant de polymorphisme d'ordre supérieur dans un système de types moderne.

Un système supportant des types avec quantification arbitraire (et donc cette forme de polymorphisme) est le système F de Girard [Gir72], qui permet par exemple de donner à la fonction précédente le type $(\forall\alpha.\,\alpha \to \alpha) \to \mathsf{int} \times \mathsf{bool}$. Toutefois, bien que l'expressivité de ce système fasse de lui un candidat intéressant, il souffre d'un défaut rédhibitoire : l'inférence de types dans le système F a été prouvée indécidable.

*INRIA Rocquencourt, Projet Cristal

Sujet

Récemment, un compromis entre ML et le système F, appelé ML^{F}, a été proposé [LBR03] ; il est intéressant pour plusieurs raisons. Tout d'abord c'est une extension conservative de ML : tout programme typable dans ML est typable dans ML^{F}. Ensuite tout terme typable dans le système F peut être encodé dans ML^{F} ; de plus, les annotations de types nécessaires à l'encodage sont peu nombreuses, et faciles à déterminer par l'utilisateur. ML^{F} est donc un excellent candidat pour accroître l'expressivité des langages de la famille ML.

Toutefois, ML^{F} n'est pas utilisable en l'état en tant que langage de programmation. Plusieurs fonctionnalités essentielles doivent au préalable lui être ajoutées, ce qui constitue l'objectif principal de cette thèse. Elles peuvent être séparées en deux classes : celles obligatoires dans un langage réaliste, et celles plus avancées ou expérimentales, qui sont souhaitables mais pas indispensables.

Les constructions que nous jugeons obligatoires comprennent les références (variables mutables), les types récursifs, et les variables de rangées ; ces deux derniers points sont nécessaires entre autre pour encoder la partie objet de OCaml [RV97]. Ces constructions ne devraient a priori poser que des problèmes techniques, qui demandent toutefois à être résolus dans une implémentation réaliste ; une étude formelle est dans tous les cas nécessaire.

D'autres traits de langages nous paraissent importants, mais poseront sans doute des difficultés théoriques bien plus grandes. Nous pensons tout particulièrement à la quantification sur les opérateurs de types, pour atteindre le pouvoir expressif de F_ω (qui permettrait de puissants encodages directement dans le langage). Les autres pistes à explorer comprennent notamment les types algébriques gardés [XCC03], qui ont récemment été proposés comme ajout possible au système de types de ML, et qui permettent d'intéressantes formes d'abstraction. Enfin, il nous semble également prometteur d'étudier la possibilité d'ajouter du sous-typage au langage, pour atteindre par exemple l'expressivité de $\mathrm{F}_\omega^{<:}$ [PT93].

Par ailleurs les types de ML^{F} sont actuellement traités de manière purement syntaxique, faute d'une sémantique adéquate, ce qui rend leur formalisation plus difficile que l'intuition ne le suggère. La recherche d'une telle sémantique, ainsi que la formalisation de l'inférence sous forme de résolution de contraintes [PR03], est donc un objectif à part entière.

Au final, tout ceci demandera à être validé par une implémentation en vraie grandeur, et il est légitime de s'attendre à être confronté à des problèmes de passage à l'échelle, qu'il faudra résoudre ou contourner.

Cadre

Ce travail pourra avoir une dimension européenne, voire internationale, étant donné l'intérêt actuel au niveau mondial pour ces thèmes de recherche. Une coopération avec les développeurs du langage Haskell à Cambridge est par

exemple envisageable, ceux-ci abordant des problématiques très proches dans le développement de leur langage [PJS04].

Cette thèse aura lieu dans le projet Cristal de l'INRIA Rocquencourt, où est développé le langage OCaml. Elle s'insère parfaitement dans les objectifs plus vastes de l'équipe, qui sont la conception, l'implémentation et l'établissement des fondements théoriques des langages de programmation fortement typés.

Signatures

Boris Yakobowski Didier Rémy

Références

[Gir72] Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. Thèse d'état, University of Paris VII, 1972.

[LBR03] Didier Le Botlan and Didier Rémy. MLF : Raising ML to the power of System-F. In *Proceedings of the International Conference on Functional Programming (ICFP 2003), Uppsala, Sweden*, pages 27–38. ACM Press, aug 2003.

[PJS04] Simon Peyton Jones and Mark Shields. Practical type inference for arbitrary-rank types. *Journal of Functional Programming*, 2004. Submitted.

[PR03] François Pottier and Didier Rémy. The essence of ML type inference. Draft manuscript, to appear in Benjamin C. Pierce, ed., *Advanced Topics in Types and Programming Languages*, September 2003.

[PT93] Benjamin C. Pierce and David N. Turner. Object-oriented programming without recursive types. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Charleston, South Carolina*, pages 299–312, January 1993.

[RV97] Didier Rémy and Jerôme Vouillon. Objective ML : A simple object-oriented extension to ML. In *Proceedings of the 24th ACM Conference on Principles of Programming Languages*, pages 40–53. ACM Press, January 1997.

[XCC03] Hongwei Xi, Chiyan Chen, and Gang Chen. Guarded recursive datatype constructors. In *Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 224–235. ACM Press, 2003.