Etude de la définition de fonctions récursives générales en théorie des types inductifs et construction d'outils de démonstration formelle pour raisonner sur ces fonctions.

Antonia Balaa
08 octobre 2001

Les systèmes de preuves basés sur la théorie des types sont des langages de programmation fonctionnels. Mais paradoxalement, la programmation de fonctions récursives peut paraître très difficile pour les utilisateurs. La difficulté n'apparait pas pour les fonctions récursives structurelles qui sont simples à décrire et simples à utiliser. En revanche elle devient flagrante pour les fonctions récursives générales. Pour de nombreux utilisateurs, même expérimentés, la seule technique de programmation qui ait été pratiquée est celle de la récursion structurelle, et la récursion bien fondée fait partie des outils dont on connait l'existence, mais dont on redoute le besoin.

Lorsque l'on programme une fonction récursive en théorie des types, on doit souvent fournir simultanément l'algorithme et la preuve qu'il termine. Pour les fonctions récursives structurelles, ceci se fait en isolant l'argument principal, tel que les appels récursifs ne pourront se produire que sur des sous-termes stricts de cet argument principal. Pour les fonctions récursives bien fondées, ceci se fait également en isolant un argument principal, mais aussi en exhibant une relation bien fondée et en montrant que les appels récursifs n'ont lieu que sur des arguments qui sont des prédécesseurs de l'argument principal pour cette relation bien fondée. Le système de preuve assure cette propriété au moment où il vérifie que l'algorithme est bien typé [6], donc on est encore dans la situation où l'algorithme et sa terminaison doivent être vérifiés simultanément.

Ces recherches sont menées suivant deux lignes directives:

- La première direction de ce travail est assez complexe, car elle repose fortement sur la notion de type dépendant. Pour définir une fonction récursive générale en Coq, l'usage est de fournir une fonction qui décrit les calculs effectués à chaque itération. Cette fonction a le type suivant

  ```
  F: (x:A)((y:A)(R y x)->B)->B
  ```

  Ce type est assez compliqué à lire, nous allons tenter de l'expliquer en langage usuel. La fonction définie, F, doit prendre deux arguments. Le premier est une valeur `x` de l'ensemble de départ `A`. Le deuxième est lui même une fonction à deux arguments: le premier est aussi une valeur `y` de l'ensemble de départ `A`, et le second doit être une preuve que cet élément est plus petit que `x`: `(R y x)`, `R` dénote l'ordre utilisé pour comparer `x` et `y` et on devra avoir prouvé par ailleurs que cet ordre ne contient pas de suites infinies décroissantes (on dit alors que `R` est un bon ordre:un ordre bien fondé et total, en anglais well-founded). La deuxième fonction représente les appels récursifs de la fonction en cours de définition. En imposant que cette fonction ait un deuxième argument qui démontre que le premier argument est plus petit que l'argument initial, on assure la terminaison de la fonction récursive.

  Ainsi, le programmeur doit fournir simultanément la description de la fonction et la démonstration qu'elle termine. Cette situation est considérée comme négative par la majeure partie des auteurs du domaine, car elle rend les données à fournir excessivement complexes. De plus, le programmeur ne dispose pas d'un théorème qui exprime comment la fonction fait ses calculs.

Nos travaux permettent d'obtenir un théorème "d'équation de point fixe" de la forme suivante:
$\forall x : A.(well\_founded\_induction\ x\ F) =$
$\quad (F\ x\ \lambda y : A.\lambda h : (y\ <\ x).\ (well\_founded\_induction\ y)).$

La fonction F fait intervenir des types dépendants et contient fréquemment des constructions de "filtrages dépendants" qui rendent la démonstration du théorème difficile.

- Dans la deuxième direction de ce travail, l'idée développée tire son origine du théorème du point fixe. On sait que toute fonction F monotone et continue dans un ordre partiel complet admet un plus petit point fixe. Il est de tradition en sémantique dénotationelle de justifier la construction des fonctions récursives par ce théorème. Le point le plus important est que ce plus petit point fixe est la limite, lorsque n tend vers l'infini de $F^n(\perp)$, ou $\perp$ est l'élément minimal de l'ordre partiel complet considéré. Notre première contribution est d'éviter de construire explicitement un ordre partiel complet en remplaçant l'utilisation de l'élément $\perp$ par une quantification universelle. Notre seconde contribution est de simuler la notion de limite par une suite de quantifications universelles et existentielles bien choisies.

  Le résultat de l'étude se concrétise en un outil automatisable (nous en avons construit une maquette) qui permet la programmation en Coq [2] de fonctions récursives générales presque aussi naturellement que dans un langage de programmation fonctionnel. Nous pensons que la technique s'appliquera avec des variations mineures dans tous les systèmes de démonstration basés sur la théorie des types.

Par comparaison, dans le premier travail [4], nous nous étions déjà intéressés à l'équation de point fixe, pour comprendre comment celle-ci pouvait être construite à partir de la fonction obtenue. La solution exposée dans [4] est excessivement complexe et n'atteint pas l'objectif de séparer les contraintes: description de l'algorithme, preuve de terminaison. Pourtant, cette solution reste valable car elle s'applique plus souvent que la solution décrite dans [10] et fournit bien une équation de point fixe. En outre, les travaux de [4] et la deuxième partie du travail [10] ont la même approche: la construction des différentes démonstrations nécessaires à la définition d'une fonction récursive suit la structure de la fonction récursive elle-même. Nous citons les travaux sur les fonctions imbriqués qui devront bien sûr être comparés aux travaux effectués par Bove et Capretta [8] et Dubois et Viguié Donzeau-Gouge [9] en théorie des types. Ces deux travaux proposent de faire apparaître la relation liant les arguments successifs des appels récursifs: le sous-ensemble du type de départ où la fonction partielle considérée est bien définie correspondant à la partie bien-fondée (selon Aczel [1]) de cette relation. D'autres travaux reliés sont ceux de Slind en logique d'ordre supérieur [5], qui fournit un outil plus complet que celui que nous présentons, mais dans un autre cadre.

Publications et présentations : La première partie du travail, a été décrite dans un article [4] publié à la conférence TPHOLs2000, ensuite présenté à TYPES2000. La deuxième partie a donné lieu à une soumission d'un article [10] à JFLA2002.

References

[1] Peter Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, 1977.

[2] B. Barras, S. Boutin, C. Cornes, J. Courant, J.C. Filliatre, E. Giménez, H. Herbelin, G. Huet, C. Mu noz, C. Murthy, C. Parent, C. Paulin, A. Saïbi, and B. Werner. The Coq Proof Assistant Reference Manual – Version V6.1. Technical Report 0203, INRIA, August 1997. revised version distributed with Coq.

[3] J. Harrison and M. Aagaard, editors. *Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000*, volume 1869 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[4] Antonia Balaa and Yves Bertot. Fix-point equations for well-founded recursion in type theory. In Harrison and Aagaard [3], pages 1–16.

[5] Konrad Slind. Another look at nested recursion. In Harrison and Aagaard [3], pages 497–517.

[6] Bengt Nordström. Terminating general recursion. *BIT*, 28, 1988.

[7] Richard J. Boulton and Paul B. Jackson, editors. *Theorem Proving in Higher Order Logics: 14th International Conference, TPHOLs 2001*, volume 2152 of *Lecture Notes in Computer Science*. Springer-Verlag, September 2001.

[8] Ana Bove and Venanzio Capretta. Nested general recursion and partiality in type theory. In Boulton and Jackson [7], pages 121–135.

[9] Catherine Dubois and Véronique Viguié Donzeau-Gouge. A step towards the mechanization of partial functions: domains as inductive predicates, July 1998. `http:/www.cs.bham.ac.uk/~ mmk/cade98-partiality`.

[10] Antonia Balaa et Yves Bertot. Fonctions récursives générales par itération en théorie des types. Article soumis à JFLA 2002, France.

[11] Antonia Balaa. Etude de la définition de fonctions récursives générales en théorie des types, Rapport de DEA Informatique, juillet 1998.