

Programmation Comparée

Interpréteurs

23 février 2010

Ceci est une version préliminaire du sujet. Je ne sais pas encore si toutes les questions sont faisables facilement. Essayez de le faire et dites-moi où cela pose problème...

Pour simplifier la correction, répondez d'abord aux questions suivantes les unes après les autres de la manière la plus simple possible sans faire le moindre ajout ni amélioration (pas de types sommes, pas de foncteurs, etc.). Vous pourrez ensuite proposer des versions plus sophistiquées de vos programmes si vous le souhaitez, en les justifiant.

1. Écrire en OCaml en style direct quatre versions d'un interpréteur du λ -calcul avec produits binaires :
 - la première implémente l'appel par valeur de gauche à droite ;
 - la deuxième implémente l'appel par nom ;
 - la troisième implémente l'appel par nécessité ;
 - la quatrième suit les conventions du langage dans lequel l'interpréteur est implémenté.
2. Dans cette question (seulement), et uniquement pour l'interpréteur en appel par valeur, ajoutez un `let` et un `let rec` au langage. On n'autorisera le `let rec` que pour une fonction. Est-ce que vos interpréteurs optimisent les appels terminaux ? Montrez des tests qui permettent de le détecter et expliquez.
3. En vous inspirant de la monade d'état, écrire des versions monadiques des interpréteurs CBV et CBN qui permettent de compter le nombre d'appels de fonctions lors de l'exécution du programme interprété. Donnez un exemple de programme pour lequel les résultats sont différents en CBV et CBN.
4. Implémentez un interpréteur en CPS, qui fait de l'appel par valeur. N'utilisez pas de monade dans cette question. La fonction d'évaluation prend en paramètre le terme, l'environnement et la continuation.
5. Écrivez une version de l'interpréteur en CPS/appel par valeur où les continuations sont défonctionnalisées.
6. Implémentez un interpréteur en CPS, qui fait de l'appel par nom, en utilisant une monade.
7. Combinez la monade précédente avec celle qui compte le nombre d'appels de fonctions pour obtenir un nouvel interpréteur.
8. Ré-implémentez en Haskell le précédent interpréteur.