

PASD - Compte Rendu Projet

Nicolas Hiot - Darenn Keller - Virgil Pia

28 octobre 2016

1 État d'avancement du Projet

Aucune extension supplémentaire n'a été réalisée par module. La réécriture des termes n'est pas complètes et des erreurs mémoire sont encore présentes dans valuate.

2 Répartitions du travail

Ci-dessous, un point de vu global du travail réalisé, pour plus de détail la page du dépôt (<https://github.com/Neplex/jubilant-octo-funicular>) sera mise en public à la fin du projet.

Si le temps le permet, le développement d'extension est envisagé.

1. Module utilitaire : Virgil Pia
2. Termes : Nicolas Hiot
3. Entrées et sorties : Nicolas Hiot
4. Variables : Nicolas Hiot
5. Donner des valeurs à des variables : Virgil Pia (avec l'aide de Nicolas Hiot)
6. Unication : Darenn Keller
7. Ré-écriture de terme : Darenn Keller

3 Termes

Question 1 : Comment fonctionnerait une fonction servant à tester la validité complète d'un terme ?

L'algorithme consisterait à tester la validité du symbole, ensuite, il faudrait tester si l'arité du terme est conforme avec son nombre d'arguments. L'algorithme est récursif sur les enfants du terme. Soit a le nombre de symboles dans le terme et l le nombre moyen de lettres d'un symbole. La complexité pour tester la validité du symbole est de $O(l)$. Pour a symboles, on a donc une complexité de $O(al)$.

4 Unification

Question 2 : Pourquoi cet algorithme s'arrête ? Par quoi peut-on borner le nombre le nombre d'égalité que l'algorithme va traiter ?

Cet algorithme s'arrête quand il n'y a plus d'égalité à traiter. On rajoute une égalité entre les arguments des termes lorsqu'ils ne sont pas des variables et qu'ils ont le même symbole et la même arité. Il est donc certain qu'au bout d'un certain nombre d'ajouts d'égalités, les égalités ajoutées auront soit une variable sur un des côtés, soit deux termes avec leurs symboles ou leur arités différentes. Dans ce cas, on ne rajoute pas d'égalité à traiter, ce qui permet à l'algorithme de s'arrêter.

Appelons `unify` la procédure exécutant cet algorithme.

Soit :

- n le nombre d'égalités données en entrée à `unify`,
- i_n le terme gauche de l'égalité n ,
- j_n le terme droit de l'égalité n ,
- $t(x)$, une fonction qui retourne l'arité du terme x plus la somme des arités de ses arguments récursivement.

Avec X le nombre d'égalités traitées par la procédure `unify`, on a :

$$X \leq n + \sum_{x=0}^n (t(i_x) + t(j_x))$$

5 Bilan Personnel

La difficulté principale rencontrée a été sur la partie **valuate**. Elle correspondait à la non-compréhension de l'utilité de la pile présente dans le code source. Une fois l'utilité trouvée (empilement des variables permettant le traitement des termes avec la dernière valeur de la variable enregistrée), l'utilisation de la procédure **term_replace_variable** du module **term_variable**, étant récursive, a posé des problèmes de réécriture des termes **set**. La solution trouvée a été de parcourir manuellement tous les termes : si un **set** est trouvé, on empile la variable et sa valeur dans la pile, sinon si c'est une variable, on essaie de la remplacer par toutes les valeurs de la pile.

Nous avons pu découvrir ou redécouvrir comment un langage peut être représenté sous forme d'AST (Arbre de Syntaxe Abstraite).

De plus, grâce à ce projet, nous avons pu commencer à développer d'une manière un peu plus orientée objet, en n'accédant pas directement aux attributs d'une structure mais en utilisant des fonctions développées pour ; ce qui rappelle l'encapsulation des données avec la notion de privée/publique en C++ par exemple.