Preuve d'un micro compilateur pour le lambda calcul non typé

Constantin Gierczak-Galle, Gabriel Doriath Döhler

Résultat

Nous avons fait toutes les questions sauf les questions 5.4 et 5.5. La question 5.4 a été partiellement abordé.

Il nous manque aussi la preuve d'un lemme pour la partie 1. Nous n'avons pas réussi à le prouver. Nous avons l'impression qu'il s'agit de la décidabilité de l'égalité sur les entiers. Ce qui nous bloque est les différentes définition de l'égalité (`=` et `?=`).

Changement par rapport au sujet

Nous avons rajouté le constructeur `Protect` au λ-calcul :

```
Inductive DeBruijn :=
    | Var : nat -> DeBruijn
    | Lambda : DeBruijn -> DeBruijn
    | Application : DeBruijn -> DeBruijn -> DeBruijn
    | Protect : DeBruijn -> DeBruijn.
```

Il empêche la subsitution des termes “protégés”. Cela permet de définir la subsitutition multiple très facilement :

```
Fixpoint substitution_multiple_aux (t: DeBruijn) (offset: nat) (u: list DeBruijn) : DeBruijn :=
    match u with
    | [] => t
    | hd :: tl =>
    substitution (substitution_multiple_aux t (offset + 1) tl) (offset) hd
    end
.

Definition substitution_multiple (t: DeBruijn) (base: nat) (u: list DeBruijn) :=
    substitution_multiple_aux t base u.

Notation "t [ n <-- l ]" := (deprotect (substitution_multiple t n l)) (at level 0)
```

Notez que les notations sont conformes aux définitions usuelles car on “déprotège” les termes à la toute fin. La propriété `S(t)`, qui est souvent hypothèse des théorèmes, affirme que le terme `t` ne contient pas de `Protect`. Ainsi l'ajout du constructeur `Protect` ne change pas les théorèmes.

Ce choix a facilité certaines preuves de la partie 1 mais cela a rendu les résultats de la partie 5 beaucoup plus durs à prouver (ainsi que l'inversion de la subsitution multiple)...