Université de Strasbourg
Licence d'informatique – Semestre 2

Modèles de Calcul
2015/2016

TD 3 – Modélisation des constantes

1 Booléens

Les booléens peuvent s'exprimer en λ-calcul pur par *TRUE* $\stackrel{\text{def}}{=} \lambda x.\lambda y.x$ et *FALSE* $\stackrel{\text{def}}{=} \lambda x.\lambda y.y$. Dans ce modèle, la conditionnelle qui est une fonction ternaire, s'exprime par *COND* $\stackrel{\text{def}}{=} \lambda c.\lambda v.\lambda f.c\ v\ f$.

1. Vérifier que le λ-terme *COND* se comporte de la bonne façon, c'est-à-dire que pour tout λ-terme E_1 et E_2, *COND TRUE* $E_1\ E_2$ se réduit en E_1 et *COND FALSE* $E_1\ E_2$ se réduit en E_2.
2. En utilisant ce modèle, donner pour chacune des opérations booléennes classique (le "ou", le "et" et le "non" logique) un λ-terme le plus simple possible qui exprime cette opération.

2 Entiers de Church

1. Le mathématicien Alonzo Church a imaginé une façon d'exprimer les entiers dans le λ-calcul pur. Dans ce modèle, 0 est représenté par $\lambda f.\lambda x.x$, 1 est représenté par $\lambda f.\lambda x.f\ \ x$, 2 est représenté par $\lambda f.\lambda x.f\ \ (f\ \ x)$, 3 est représenté par $\lambda f.\lambda x.f\ \ (f\ \ (f\ \ x))$, etc. L'entier n est représenté par le λ-terme $\underline{n} \stackrel{\text{def}}{=} \lambda f.\lambda x.\underbrace{f\ (f\ ...\ (f}_{n\ \times}\ \ x)\ ...)$. Ce modèle permet d'exprimer les fonctions successeurs, addition, multiplication et puissance par les λ-termes suivants :

   SUCC $\stackrel{\text{def}}{=} \lambda n.\lambda f.\lambda x.f\ (n\ f\ x)$
   ADD $\stackrel{\text{def}}{=} \lambda n.\lambda m.\lambda f.\lambda x.n\ f\ (m\ f\ x)$
   MULT $\stackrel{\text{def}}{=} \lambda n.\lambda m.\lambda f.m\ (n\ f)$
   POW $\stackrel{\text{def}}{=} \lambda n.\lambda m.m\ n$

   Tester ce modèle en effectuant les calculs suivants :
   (a) Successeur de 2 : Réduire le λ-terme *SUCC* $\underline{2}$.
   (b) $3 + 2$: Réduire le λ-terme *ADD* $\underline{3}\ \underline{2}$.
   (c) 3×2 : Réduire le λ-terme *MULT* $\underline{3}\ \underline{2}$.
   (d) 3^2 : Réduire le λ-terme *POW* $\underline{3}\ \underline{2}$.
2. On s'intéresse au λ-terme Z défini par : Z $\stackrel{\text{def}}{=} \lambda x.x\ (\lambda x.\lambda c.c$ *FALSE* $x)\ (\lambda x.x)$ *TRUE*
   (a) Réduire sous forme normale (en utilisant NOR) les λ-termes suivants :
      i. Z $\underline{0}$
      ii. Z $\underline{1}$
      iii. Z $\underline{2}$
   (b) D'après vous, que modélise Z ?

3 Couples

1. On modélise un couple (a, b) par le λ-terme $\lambda t.t\ a\ b$.
   (a) Définir un λ-terme *PAIR* qui modélise la fonction qui, étant donné deux éléments a et b, retourne le couple (a, b).
   (b) Réduire les expressions (*PAIR* $a\ b$) $\lambda x.\lambda y.x$ et (*PAIR* $a\ b$) $\lambda x.\lambda y.y$.
   (c) En déduire une définition des λ-termes *FST* et *SND* qui modélisent les fonctions d'accès aux éléments d'un couple.

(d) Modéliser la fonction *CURRY* et son inverse *UNCURRY*. On rappelle que la fonction *CURRY* (resp. *UNCURRY*) permet, pour une fonction à deux arguments, de passer de la forme non curryfiée (resp. curryfiée) à la forme curryfiée (resp. non curryfiée).

2. En s'inspirant du modèle pour les couples, on souhaite modéliser les triplet :
   (a) Définir un λ-terme permettant de modéliser le triplet (a, b, c)
   (b) Définir un λ-terme *TRIPLET* qui modélise la fonction qui, étant donné trois éléments a, b et c, retourne le triplet (a, b, c)
   (c) Définir des λ-termes *ELT1*, *ELT2* et *ELT3* qui modélisent les fonctions d'accès aux éléments d'un triplet.

4 Récursivité

1. Le combinateur de point fixe de Turing est défini par : $\Theta \stackrel{\text{def}}{=} (\lambda x.\lambda y.y\ (x\ x\ y))\ (\lambda x.\lambda y.y\ (x\ x\ y))$. Montrer que pour tout λ-terme *H*, le λ-terme Θ *H* se réduit bien en *H* (Θ *H*).
2. On suppose que les λ-termes *COND*, *ISZERO*, *PRED* modélisent respectivement la conditionnelle, l'égalité à zéro et la fonction prédécesseur. Pour chacune des fonctions suivantes, écrire un λ-terme qui la modélise :
   (a) factorielle (*FAC* n)
   (b) égalité de deux entiers (*EG* n m)
   (c) inférieur strict entre deux entiers (*INF* n m)