Programmation fonctionnelle Lambda-calcul (Suite) et Preuves

D.E ZEGOUR
École Supérieure d'Informatique
ESI

Programmation fonctionnelle / Lambda-calcul

Sommaire

Programmation fonctionnelle / Lambda-calcul

Forme normale

une λ -expression est dite en forme normale si elle ne contient aucun rédex. [$(\lambda x.M)$ N]

Dans l'exemple $(\lambda x.(\lambda y.yx)z)v) \Rightarrow (\lambda y.yv)z \Rightarrow zv$

zv est la forme normale de $(\lambda x.(\lambda y.yx)z)v)$

Dans l'exemple $(\lambda x.xxy)(\lambda x.xxy) \Rightarrow (\lambda x.xxy)(\lambda x.xxy)y \Rightarrow (\lambda x.xxy)(\lambda x.xxy)yy \Rightarrow$ ect...

il n y a pas de forme normale.

Programmation fonctionnelle / Lambda-calcul

Ordre de réduction

Par nom : Plus externe, plus à gauche dans le même niveau. (Ordre normal)

Par valeur : Plus interne, Plus à gauche dans le même niveau

Il existe d ’autres ordres

Deux chemins différents de réduction peuvent-ils aboutir à des formes normales différentes ?

Programmation fonctionnelle / Lambda-calcul

Ordre de réduction

Ordre normale (par nom)

(λx. (λy. + x ((λx. - x 3) y))) 5 6

=> (λy. + 5 ((λx. - x 3) y)) 6

=> + 5 ((λx. - x 3) 6)

=> + 5 (- 6 3)

=> + 5 3

=> 8

Par valeur

(λx. (λy. + x ((λx. - x 3) y))) 5 6

=> (λx. (λy. + x (- y 3))) 5 6

=> (λy. + 5 (- y 3)) 6

=> + 5 (- 6 3)

=> + 5 3

=> 8

Programmation fonctionnelle / Lambda-calcul

Théorème de Church-rosser

(Première forme)

Si U=>X et U=>Y, il existe Z tel que X=>Z et Y=>Z.

$$\begin{tikzcd} & U \arrow[dl] \arrow[dr] & \\ X \arrow[dr] & & Y \arrow[dl] \\ & Z & \end{tikzcd}$$

<u>Corollaire :</u>

Si U a des formes normales X et Y, alors X est congruent à Y (On peut passer de X à Y par des changements de variables)

(Deuxième forme)

Si E1 => E2 et Si E2 forme normale alors il existe une suite de réductions de E1 à E2 selon l 'ORDRE NORMAL.

L 'ordre normal garantit de trouver une forme normale si elle existe mais ne garantit pas de la trouver par un nombre minimum de réduction.

Programmation fonctionnelle / Lambda-calcul

Théorème de Church-rosser (exemple de non terminaison)

Soit à évaluer l'expression ((λx.y) (λx.(x x) λx.(x x)))

2 manières d'évaluation

Ordre par valeur (Plus interne)
((λx.y) (λx.($\underline{x}$ $\underline{x}$) λx.(x x)))
⇒ ((λx.y) (λx.(x x) λx.(x x)))
...
Boucle infinie

Ordre par nom (normal : plus externe)
((λx.y) (λx.(x x) λx.(x x)))
⇒ y

Programmation fonctionnelle / Lambda-calcul

Combinateurs

Combinateur = lambda-expression sans variable libres

Exemples:

I =	λx.x	Identité
App=	λf.λx.(f x)	Application
C=	λf.λg.λx.(f (g x))	Composition
L=	(λx.(x x) λx.(x x))	Boucle
Cur=	λf.λx.λy.((f x) y)	Curryfication
Car =	λc.c(λa. λb.a)	Premier élément d'une liste

Programmation fonctionnelle / Lambda-calcul

Fonctions récursives

Soit Fact = (λ n. (si (= n 0) 1 (* n (Fact (- n 1))))

De la forme Fact = (λn. (….Fact …))

Par une Béta-abstraction

Fact = (λ fac .(λn. (…fac …))) Fact

Fact = H Fact

[implique que Fact est un point fixe de H]

Avec H = (λ fac .(λn. (…fac …)))

Il suffit donc de trouver un point fixe de H.

On démontre que   YH = H (YH) , c ’est à dire que YH est un point fixe de H.

Y est appelé le combinateur du point fixe.

Solution Fact = YH.

Fact N = (YH) N = H (YH) N

Programmation fonctionnelle / Lambda-calcul

Fonctions récursives : Exemple : Calcul de Fact 1

YH 1 =

H (YH) 1

(λ fac. λn. si (= n 0) 1 (* n (fac (- n 1)))) YH 1

(λn. si (= n 0) 1 (* n (YH (- n 1)))) 1

si (= 1 0) 1 (* 1 (YH (- 1 1)))

* 1 (YH 0)

* 1 (H (YH) 0)

* 1 (λfac. λn. si (= n 0) 1 (* n (fac (- n 1)))) (YH) 0)

* 1 (λn. si (= n 0) 1 (* n (YH (- n 1)))) 0

* 1 (si (= 0 0) 1 (* 0 (YH (- 0 1)))

* 1 (1)

1

Fonctions récursives : Combinateur Y

$Y =(\lambda h . (\lambda x. h (x x)) (\lambda x.h (x x)))$

Montrons que $YH = H (YH)$

$YH = (\lambda h . (\lambda x. \underline{h} (x x)) (\lambda x.\underline{h} (x x))) H$

$YH = (\lambda x. H (\underline{x}\ \underline{x})) (\lambda x.H (x x))$

$YH = H (\lambda x.H (x x) (\lambda x.H (x x)$

$YH = H (YH)$

Donc YH point fixe de H.

Programmation fonctionnelle / Lambda-calcul

Modélisation des fonctions CONS, CAR et CDR

Concaténation de deux listes

CONS = (λa. λb. λf. fab)

Premier élément d'une liste

CAR = (λc.c(λa. λb.a))

Liste sans le premier élément

CDR = (λc.c(λa. λb.b))

Montrons que CAR (CONS p q) = p

$\underline{\text{CAR}}$ (CONS p q)

$=(\lambda c.\underline{c}(\lambda a.\ \lambda b.a))\ (\text{CONS p q})$

$=\underline{\text{CONS}}\ p\ q\ (\lambda a.\ \lambda b.a)$

$=(\ \lambda a.\ \lambda b.\ \lambda f.\ f\ \underline{a}\ b)\ p\ q\ \ (\lambda a.\ \lambda b.a)$

$=(\ \lambda b.\ \lambda f.f\ p\ \underline{b})\ \ q\ \ (\lambda a.\ \lambda b.a)$

$=(\lambda f.\ \underline{f}\ p\ q)\ \ (\lambda a.\ \lambda b.a)$

$= (\lambda a.\ \lambda b.\underline{a})\ p\ q$

$= (\lambda b.p)\ q$

$= p$

Programmation fonctionnelle / Preuve

Approche système formel (Rappel)

Soit **f(x;y) : μ** une déclaration de procédure de corps μ.

x : listes des paramètres valeurs(entrées)
y : listes des paramètres résultats(sorties)

Nous voulons démontrer

p(x) {f(x;y):μ} q(x,y)

p(x) vrai --> q(x,y) vrai

Théorème :

Supposer la correction partielle par rapport p et q de tous les appels internes
et montrer que
p(x) {f(x;y) : μ} q(x, y)
est vraie

Programmation fonctionnelle / Preuve

Concepts mathématiques : Notion de fonctionnelle et de points fixes

L'écriture f = λ n.u(n) a le même sens que l'écriture habituelle f : n-->u(n)

Cette notation permet d'exprimer d'une autre manière les définitions récursives.

Exemples :

f1 = λ n.si n =0 alors 1 sinon n*f1(n-1) Fsi

f2 = λ xy.
Si x=y alors y+1
Sinon f2(x,f2(x-1, y+1))
Fsi

$f2 = \tau(f2) = \lambda$ xy.
Si x=y alors y+1
Sinon f2(x,f2(x-1, y+1)) Fsi

Concepts mathématiques : Notion de fonctionnelle et de points fixes

Une fonctionnelle est une application τ d'un espace de fonction $\Re(E,F)$ dans lui-même. C'est à dire à une fonction f (de E dans F) on fait correspondre une fonction g (de E dans F). ce qui s'écrit $g = \tau(f)$.

Une fonction f de $\Re(E, F)$ est dite point fixe de si $\tau(f) = f$.

A chaque équation du type $f = \lambda$ n.u(n), on peut associer une fonctionnelle
$\tau = \lambda f.\lambda n.u(n)$ admettant comme points fixes les solutions de l'équation $\tau(f) = f$.

Par exemple la fonctionnelle f2 admet les 2 solutions suivantes (points fixes) :
$g = \lambda$ xy. x+1
$h = \lambda$ xy. Si x>=y alors x+1 Sinon y-1 Fsi
à vérifier ....

Programmation fonctionnelle / Preuve

Concepts mathématiques

Fonction moins définie qu'une autre (Inclusion)

Soient f et g deux fonctions de E dans F.
On dit que f est moins définie que g ($f \subseteq g$) si pour tout x de E tel que
- f(x) définie, g(x) est aussi définie et
- f(x)=g(x).

La relation $\subseteq$ est une relation d'ordre dans $\Re(E, F)$.

La fonction nulle part définie est le plus petit élément de cet ensemble.

Convergence

Une suite f1, f2, ..., fn converge vers une fonction f si:
$f1 \subseteq f2 \subseteq .....fn = f$

A partir d'un certain rang n
Domaine de fn = Domaine de f

Programmation fonctionnelle
/ Preuve

Un treillis complet (E,≤) : *toute partie A de E* admet une borne supérieure et une borne inférieure
E admet un plus grand élément et un plus petit élément.

Idée de la preuve

A=[x1, x2, ..., xn]
Continuite: f(SUP A) = SUP f(A)

$\Re$(E, F) muni de la relation $\subseteq$ est un treuilli complet

Son plus petit élément Ω est la fonction nulle part définie.

A tout programme fonctionnel récursif f = λ n.u(n) correspond une application
τ = λf.λn.u(n) de $\Re$ (E, F) dans $\Re$ (E, F)

τ est continue au sens des chaînes

Le programme est alors vu comme étant une équation fonctionnelle f = τ (f) dont le plus petit point fixe coïncide avec la fonction calculée par le programme.

Programmation fonctionnelle / Preuve

Idée de la preuve

La fonction x! est le plus petit point fixe (la plus petite solution de l 'équation
$f = \lambda$ x.si x=0 alors 1 sinon x*f(x-1) fsi

de la forme $f = \tau(f)$

C 'est à dire $x! = \tau\ (x!) = \lambda x.$ si x=0 alors 1 sinon x*(x-1)! fsi

Programmation fonctionnelle / Preuve

Idée de la preuve

Pour prouver qu' un programme fonctionnel (récursif) τ calcule bien (exactement) une fonction donnée f, il faut prouver que f est le pppf de τ. (Plus Petit Point Fixe)

Si f n 'est pas le pppf de τ mais un point fixe de τ alors on peut conclure que τ calcule partiellement f.

Si on ne connaît pas f on peut la trouver en calculant le pppf de τ qui est égal à SUP (τ^n (Ω)) n> 0.

Programmation fonctionnelle / Preuve

Exemple 1

f1 = τ(f1) =λ xy.Si x=y alors y+1
Sinon f1(x, f1(x-1, y+1) Fsi

Prouver que f1 calcule (au moins partiellement) la fonction
g = λ xy. x+1

Il suffit de montrer que g est un point fixe, c ’est à dire τ(g)=g

- **Démonstration**

Cas x=y : x+1 CQFD (Ce qu'il fallait démontrer)
Cas x#y :
calcul de g(x-1, y+1) = x
calcul de g(x, x) = x + 1 CQFD

Programmation fonctionnelle / Preuve

$f1 = \tau(f1) = \lambda$ xy.Si x=y alors y+1
Sinon f1(x, f1(x-1, y+1) Fsi

Exemple 2

On peut aussi prouver que f1 calcule la fonction
h = λ xy. Si x $\geq$ y alors x+1
Sinon y-1 Fsi.

Là aussi, il faut montrer que h est un point fixe de la fonctionnelle f1, c 'est à dire τ(h) = h

- **Démonstration**

Cas x=y : h(x, y) = x+1 = y +1 CQFD
Cas x#y :
si x-1 < y+1 ou x<y+2 ou x $\leq$ y+1
appel interne : h(x-1, y+1) = (y+1) - 1 = y
appel externe h(x, y) CQFD
Si x-1 $\geq$ y+1 ou x $\geq$ y+2 ou x > y+1
appel interne : h(x-1, y+1) = (x-1) + 1 = x
appel externe h(x, x) = x + 1 = h(x,y) CQFD
(Car si x >y+1 alors on a fortiori x $\geq$ y)

```
-------!--!-------------------
       y  y+1
```

(Toutes les valeurs sont couvertes)

Programmation fonctionnelle / Preuve

$f1 = \tau(f1) = \lambda$ xy.Si x=y alors y+1
Sinon f1(x, f1(x-1, y+1) Fsi

Exemple 3

On peut aussi prouver que :

m = λ xy. si x $\geq$y et pair (x-y) alors x+1 fsi

est point fixe de τ, c 'est à dire τ(m) = m

☐ **Démonstration**

τ(m) = si x = y alors y+1
sinon m(x, m(x-1, y+1))

Cas x=y : Pair(x-x) =Pair(0) et donc x + 1 = m (CQFD)

Cas x#y:

Appel interne

Si (x-1 $\geq$ y + 1) et Pair(x-1-y-1)Ou bien

Si x $\geq$ y+2 et pair(x-y-2) (qui est pair(x-y))

alors m(x-1, y+1) = x

Appel externe m(x, x) = x + 1 = m (CQFD)

τ(m) = x +1 si x=y ou x $\geq$ y+2 et pair(x-y)
= Ω sinon

Dans les 2 cas c 'est m.

Programmation fonctionnelle / Preuve

Exemple 4

Montrer que

$m \subseteq g$

$m \subseteq h$

m est pppf de τ.

Programmation fonctionnelle / Preuve

Exemple 5

Calculer le pppf de l ’équation

f = si x = 0 alors 1 sinon x * f(x-1) fsi

Solution

de la forme f = τ (f)

$\tau(\Omega)$ = si x = 0 alors 1 sinon x *Ω(x-1) fsi

= si x = 0 alors 1 sinon Ω fsi

$\tau^2(\Omega)$ = si x = 0 alors 1 sinon $\tau(\Omega)$ (x - 1) fsi

= si x = 0 alors 1 sinon (si x-1=0 alors 1 sinon (x-1)* Ω(x-1))

= si x = 0 alors 1 sinon (si x-1=0 alors 1 sinon Ω)

$\tau^3(\Omega)$ =si x = 0 alors 1 sinon si x=1 alors 1 sinon si x = 2 alors 1*2 sinon Ω fsi fsi fsi

$\tau^4(\Omega)$ =si x = 0 alors 1 sinon si x=1 alors 1 sinon si x = 2 alors 1*2 sinon si x=3 alors 1*2*3 sinon Ω fsi fsi fsi fsi

...

$\tau^n(\Omega)$ = 1* 2* 3* 4* 5 ……*n = n!

Reccurrence à démontrer