

Typage et Analyse Statique

Exercices cours 1

Emmanuel Chailloux

Parcours Science et Technologie du Logiciel
Master mention Informatique
Sorbonne Université

année 2019-2020

Évaluation du λ -calcul

Évaluation formelle : β -réduction :

1. On choisit un redex $(\lambda x. T_1) T_2$ dans l'expression,
2. on remplace x par T_2 dans T_1 ,
3. on remplace le redex par ce résultat.
4. **Normalisation** : on continue tant qu'il y a des redexes.

Stratégies d'évaluation du λ -calcul

Stratégies d'évaluation :

- ▶ réduction standard : le redex le plus externe gauche est réduit en premier
- ▶ réduction par valeur : le redex le plus interne à droite est réduit en premier
- ▶ sans stratégie : n'importe quel redex peut être réduit

Appels dans les langages de programmation :
quelle stratégie est suivie ?

Extensions du λ -calcul

Par encodage (ex : les couples) :

- ▶ **Construction** : $CONS := \lambda x.\lambda y.(\lambda f.f x y)$
- ▶ **Projection 0** : $P0 := \lambda c.c (\lambda a.\lambda b.a)$
- ▶ **Projection 1** : $P1 := \lambda c.c (\lambda a.\lambda b.b)$
- ▶ **Échange** : $SWAP := \lambda c.c (\lambda x.\lambda y.CONST y x)$

Par ajout de termes/opérations de base (ex : entiers) :

- ▶ $val ::= var \mid int \mid add \mid sub$
- ▶ $term ::= \lambda var.term \mid term term \mid val$
- ▶ Ex : $\lambda x.\lambda y.add x (sub y 3)$

Comment évaluer $CONS\ 1\ 2$ en pratique ?

- ▶ Réécriture de termes : $CONS\ 1\ 2 = \lambda f.f\ 1\ 2$
en pratique, difficile de modifier le code du programme.

- ▶ Fermetures :

$CONS\ 1\ 2$

$\rightarrow (\lambda x.\lambda y.\lambda f.f\ x\ y)[]\ 1\ 2$

$\rightarrow (\lambda y.\lambda f.f\ x\ y)[(x,1)]\ 2$

$\rightarrow (\lambda f.f\ x\ y)[(x,1);(y,2)]$

On crée une **fermeture** :

- ▶ corps de la fonction,
- ▶ environnement : valeurs des variables lors de l'abstraction.

Lors de l'appel, on exécute le corps dans l'environnement, augmenté de la valeur du paramètre.

Exemple en OCaml

```
# let f x y z = x + y + z ;;
val f : int -> int -> int -> int = <fun>
# f 1 ;;
- : int -> int -> int = <fun>
# let g = f 1 2 ;;
val g : int -> int = <fun>
# g 10 ;;
- : int = 13
# g 10 20 ;;
Error: This function is applied to too many arguments;
maybe you forgot a ';'
#
```

Un Évaluateur de λ -calcul

Fabriquer une valeur calculable de la forme $\text{terme}_{\text{env}}$.

env	terme	pile	\rightarrow env	term	pile
e	$F A$	S	$\rightarrow e$	F	$A_e :: S$
e	$\lambda x.C$	$a :: S$	$\rightarrow (x, a) :: e$	C	S
$(= x, A_{e'}) :: e$	x	S	$\rightarrow e'$	A	S
$(\neq x, _) :: e$	x	S	$\rightarrow e$	x	S

Une machine fonctionnelle : la machine de Krivine

- ▶ Exécute du code-octet, compilé depuis un lambda terme,
- ▶ code-octet complètement linéaire (suite d'opcodes),
- ▶ trois opcodes très simples.

De quoi a-t'on besoin ?

env	terme	pile		env	term	pile
e	$F A$	S	\rightarrow	e	F	$A_e :: S$
e	$\lambda x.C$	$a :: S$	\rightarrow	$(x, a) :: e$	C	S
$(= x, A_{e'}) :: e$	x	S	\rightarrow	e'	A	S
$(\neq x, _) :: e$	x	S	\rightarrow	e	x	S

Machine virtuelle

```
type closure = C of int * closure list
```

```
let interprete code =
```

```
  let rec interp env pc stack =
```

```
    match (nth code pc) with
```

```
    | ACCESS n ->
```

```
      begin try
```

```
        let (C (n,e)) = nth env n in
```

```
          interp !e n stack
```

```
        with ex -> (C (pc, ref env))
```

```
    | PUSH n ->
```

```
      interp env (pc+1) ((C (n,ref env))::stack)
```

```
    | GRAB ->
```

```
      begin match stack with
```

```
      | [] -> C (pc,ref env)
```

```
      | so::s -> interp (so::env) (pc+1) s
```

```
in
```

```
  interp [] 1 []
```

Compilation vers la machine de Krivine

Assembleur avec étiquettes :

```
type instr =  
  | IPUSH of lbl  
  | IGRAB  
  | IACCESS of int  
  | ILABEL of lbl
```

Schéma de compilation \mathcal{C} :

$$\mathcal{C}_e(T_1 T_2) = \text{IPUSH } l ; \mathcal{C}_e(T_1) ; \text{ILABEL } l ; \mathcal{C}_e(T_2)$$

$$\mathcal{C}_e(\lambda x. T) = \text{IGRAB} ; \mathcal{C}_{x::e}(T)$$

$$\mathcal{C}_e(x) = \text{IACCESS } nth(x, e)$$

Puis on fait une passe de suppression des étiquettes.