

CM 17-11

Pour le projet, aucune structure impérative
seul `Sys.argv` pour récupérer les arguments.

Rappel: Pour vérifier les arguments entrés :

`match Sys.argv with`

`| [1 _; "arg_0"; "arg_1"] -> ...`

`| [1 _; "arg"] -> ...`

`| _ -> ...`

le premier argument est (comme en C) le
nom de la commande.

Evaluation en Ocaml: (Sémantique)

① - Première approche haut niveau:

La substitution:

Var libre / liée ?

En Ocaml, une variable est liée si elle
est définie puis réutilisable après.

On peut le voir si on peut renommer
la variable.

`match ... with`

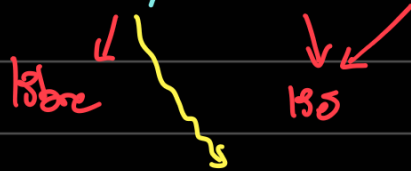
`| Some x -> ...`

`| y -> ...`

les variables
sont liées.

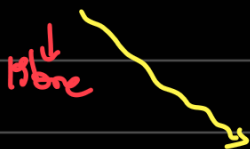
Quand est-ce que c'est libre ?

$\text{fun } y \rightarrow (x, \text{fun } z \rightarrow z)$



car si je renomme z ça ne compile plus
(en supposant $\exists x$ après avant globalement)

$\text{let } x = x \text{ in } x$



Idem, si je renomme x ça ne fonctionne plus.

On note $\tau[x/u]$ pour la subst des
occurs libres de x par u dans τ .

Exemple:

$$\begin{aligned} & (x + y) [x/1] \\ &= (1 + y) \end{aligned}$$

$$\begin{aligned} & (\text{let } x = 0 \text{ in } x + y) [x/1] \\ &= (\text{let } x = 0 \text{ in } x + y) \quad \text{car } \underline{x \text{ libre}} \end{aligned}$$

Rmq: On aurait pu écrire :

$$\begin{aligned} & ((\text{let } x = 0 \text{ in } x + y) [x/z]) [x/1] \\ & \text{aurait rendu la même chose.} \end{aligned}$$

$$\begin{aligned} & (x, \text{fun } z \rightarrow z) [x/1] \\ &= (1, \text{fun } z \rightarrow z) \end{aligned}$$

$$\bullet \text{ (let } x = e_1 \text{ in } e_2) [x / 1] \\ \equiv (\text{let } x = 1 \text{ in } e_2)$$

$$\bullet (\text{Fun } y \rightarrow e) [x / (y+1)]$$

⚠ On doit renommer le y WS pour faire la substitution ⚠

$$\equiv (\text{Fun } z \rightarrow y+1) \quad \downarrow \text{ Solution: on renomme la variable WS en question.}$$

(cf CM Logique 10/11)

Les valeurs en Ocaml:

- 1) • Les constantes / littéraux (1, 2, ...)
- Booléens
- String
- ...

En gras ce qu'on renvoie.

- 2) Les fonctions de la forme

$$\text{Fun } x_1 \dots x_n \rightarrow e$$
 avec aucune variable libre.

- 3) Les records dont les champs sont des valeurs.

- 4) Les constructeurs de données (ex: Piles, arbres, ...)

↳ Comment évaluer le let.

let $x = e_1$ in e_2 avec e_1, e_2 des expressions. Ce n'est pas une valeur donc comment l'évaluer.

- 1) Evaluer e_{-1} en sa valeur v_{-1}
(si le calcul termine) (l'éval donne un Fun $x \rightarrow e$)
- 2) Evaluer $e_{-2}[x / v_{-1}]$

Exemple:

$$\bullet \text{ let } x = \underbrace{1+2}_{e_1} \text{ in } \underbrace{2 * x}_{e_2}$$

$$v_{-1} = 3 \text{ (= valeur de } e_1)$$

$$\text{puis } e_2[x / v_1 = 3] : 2 * 3 = 6$$

$$\bullet \text{ let } x = \underbrace{2 * x}_{e_1} \text{ in } \underbrace{3 * x}_{e_2} = e$$

! nous faut une valeur pour x (var libre)

$$\text{donc } e[x/3] : e_1[x/3] = 2 * 3 = 6 = v_1$$

$$\text{donc } e_2[x/v_1] : 3 * v_1 = 3 * 6 = 18$$

Pour les Fonctions:

$$F \ x \ y \ z = \left(\underbrace{(F \ x) \ y}_{e_1} \right) \uparrow \underbrace{z}_{e_2}$$

Pour les match: match e with $p_1 \rightarrow e_1 \mid \dots \mid p_n \rightarrow e_n$

- 1) Evaluer e avec une valeur v .
- 2) Essayer de lier v par le motif p_i
1. si ça match, $[x_i / v_i]$ où x_i est

une variable du pattern et v_i
une valeur extrait de v .

2. Sinon on passe à la branche suivante.
Jusqu'à la fin.

Pour les exemples, voir poly.

Pour les let rec :

$let\ rec = e_1\ in\ e_2$

- 1) Évaluer e_1 en une valeur v_1 . Cette valeur contient les occurrences libres de x .
- 2) Construire la valeur cyclique $v = v_1[x/v]$
- 3) Évaluer $e_2[x/v]$

② - En machine:

On se rappelle des valeurs dans le tas au lieu de toujours les substituer.

↳ Et pour les fonctions ?

On représente la fonction par des fermatures. C'est un bloc alloué ds le tas qui débute par un pointeur de code (le code de la fonction). Ce bloc contient aussi les valeurs des variables libres de la fonction.