CM 17-11

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

Rappel: Pour vérifier les arguments entrés :

```
match Sys.argv with
| [| _ ; "arg_0" ; "arg_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 / lié ?

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 ?

fun y -> (x, fun x -> x)

libre lié

car si je renomme x ça ne compile plus (en supposant ∃x différent avant globalement)

lié

let x = x in x

libre

Idem, si je renomme x ça ne fonctionne plus.

On note $t[x/v]$ pour la subst des occur libres de x par v dans t.

Exemple :

- $(x + y)[x/1]$
  $\equiv (1 + y)$

- $(\text{let } x = 0 \text{ in } x + y)[x/1]$
  $= (\text{let } x = 0 \text{ in } x + y)$ car x lié
  Rmq : On aurait pu écrire :
  $((\text{let } x = 0 \text{ in } x + y)[x/3])[x/1]$
  aurait rendu la même chose.

- $(x, \text{fun } x \to x)[x/1]$
  $\equiv (1, \text{fun } x \to x)$

- $(\text{let } x = x \text{ in } x+1)[x/1]$
  $\equiv (\text{let } x = 1 \text{ in } x+1)$

- $(\text{Fun } y \to x)[x/(y+1)]$

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

$\equiv (\text{Fun } z \to y+1)$ ← Solution: on renomme la variable liée en question.

(cf CM Logique 10/11)

Les valeurs en Ocaml:

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

   En gros ce qu'on renvoie.

2) Les fonctions de la forme

   $\text{Fun } x_1 \dots x_n \to e$

   avec aucune variable libre.

3) Les records dont les champs sont des valeurs.
4) Les constructeurs de données (ex: Bglbes, 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
(ssi le calcul termine)(l'éval donne un Fun x → e)
2) Evaluer e_2[x / v_1]

Exemple:

- let x = 1+2 in 2*x
  (e1 = 1+2, e2 = 2*x)

v_1 = 3 (= valeur de e1)
puis e2[x / v1=3] : 2*3 = 6

- let x = 2*x in 3*x = e
  (e1 = 2*x, e2 = 3*x)

⇒ il nous faut une valeur pour x (var libre)
donc e[x/3] : e1[x/3] = 2*3 = 6 = v1

donc e2[x/v1] : 3*v1 = 3*6 = 18

Pour les Fonctions :

F x y z = (((F x) y) z)
(e1 = ((F x) y), e2 = z)

Pour les match : match e with p1 → e1 | ... | pn → en

1) Evaluer e avec une valeur v.
2) Essayer de filtrer v par le motif p1
1. Si ça match, [x.i / v.i] où x.i est

une variable du pattern et v3 une valeur extraite 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 = e1 in e2

1) Evaluer e1 en une valeur v1. Cette valeur contient les occurrences libres de x.
2) Construire la valeur cyclique v = v1[x/v]
3) Evaluer e2[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 fermetures. 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.