SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

PCOMP(LU3IN032)
Année 2022/2023
Séance 2 de TD/TME

Objectifs

1. Comparaison des styles de programmation impérative et fonctionnelle
   - manipulation de matrices
   - enregistrement avec des champs fonctionnels
   - parcours d'arbres
2. Code et données :
   - fermetures,
   - taille des valeurs
3. mécanisme d'évaluation, immédiat et retardé

Travaux Dirigés

Exercice 1 : Comptage des points à la Belote

Q.1.1 Définir un type `couleur` pour représenter les couleurs d'un jeu de cartes.

Q.1.2 Définir un type `carte` pour représenter les cartes d'un jeu de belote : as, roi, dame, valet et petites cartes.

Q.1.3 Écrire une fonction `valeur` qui prend la couleur de l'atout et une carte et renvoie la valeur de la carte. On rappelle les règles de comptage suivantes : l'as vaut 11, le roi 4, la dame 3, le valet 20 s'il a la couleur de l'atout, 2 sinon, le 10 vaut 10, le 9 vaut 14 s'il a la couleur de l'atout, 0 sinon, les autres cartes valent 0.

Q.1.4 Écrire une fonction `valeur_jeu` qui prend en paramètre la couleur de l'atout et une liste de carte, et renvoie la valeur du jeu.

Q.1.5 Tester la fonction précédente sur le jeu suivant : valet de carreau, 10 de trèfle, 7 de cœur, 8 de carreau, 9 de pique avec atout à carreau (on doit trouver 30) .

Exercice 2 : Jeu de la vie

Nous allons implanter le célèbre *jeu de la vie* de Conway. Dans un tel jeu, le programme possède un quadrillage de cellules qui peuvent être mortes ou vivantes. En partant d'une configuration donnée (une valeur morte ou vivante pour chaque cellule), le jeu passe d'étape en étape en appliquant simultanément un ensemble de règles qui décident de la survie et de la naissance des cellules d'une étape à l'autre.

Q.2.1 Donnez un type pour représenter un tel quadrillage et donnez un exemple de situation initiale `init_gen`.

Q.2.2 Donnez la fonction `next_gen` qui calcule la génération suivante avec les règles de vie et de mort suivantes :

- Une cellule meurt d'isolement si elle a moins de deux voisins.
- Une cellule meurt d'étouffement si elle a plus de trois voisins.
- Une cellule naît si elle a exactement 3 voisins.

Vous donnerez aussi la fonction auxiliaire `neighbours` qui calcule le nombre de cellules voisines vivantes d'une cellule donnée.

Exercice 3 : Enregistrements à champs fonctionnels

On cherche à définir des déplacements sur des mobiles dans un plan. Pour cela on se donne les types suivants en OCaml (l'exercice peut être résolu en Python, Java ou C).

```
type point = {xc :int ; yc :int} ;;
type mobile = {pos :point ; depl : point -> mobile} ;;
```

Q.3.1 Définir la fonction translate de type int -> int -> point -> point qui prend deux entiers `dx` et `dy` et un point et qui rend un nouveau point déplacé relativement de (dx, dy).

Q.3.2 Définir le déplacement immobile qui correspond à un point immobile.

Q.3.3 Définir le déplacement uniforme qui correspond à un mouvement uniforme défini par deux incréments `dx` et `dy`

Q.3.4 Définir un déplacement uniforme sur un nombre de pas n défini à l'avance, ce déplacement sera suivi par un autre déplacement *cont* que l'on passera en argument :

```
avance : int (* dx *) -> int (* dy *)
         -> int (* n *) -> deplacement (* cont *)
         -> deplacement
```

Q.3.5 Définir une fonctionnelle `compose` sur les déplacements qui prend un entier n et deux déplacements, effectue le premier durant n pas puis effectue le second. Redéfinir le déplacement précédent à partir de cette fonction.

Exercice 4 : Evaluation retardée

On reprend l'exemple du cours 2 d'implantation de l'évaluation paresseuse qui est une optimisation de l'évaluation retardée. Une expression est évaluée à sa première utilisation et retournera la valeur calculée pour les autres utilisations.

```
# type 'a v =
    Imm of 'a
  | Ret of (unit -> 'a);;

# type  vm = {mutable c :  v };;

# let  eval e = match e.c with
  Imm a -> a
| Ret f -> let u = f () in
           ( e.c <- Imm u;u);;
val eval : 'a vm -> int = <fun>

# let si_ret c e1 e2 =
  if eval c then eval e1 else eval e2;;
val si_ret : bool vm -> 'a vm -> 'a vm -> 'a = <fun>

# let rec facr n =
  si_ret {c=Ret(fun () -> n = 0)}
         {c=Ret(fun () ->  1)}
         {c=Ret(fun () -> n*(facr(n-1)))};;
val facr : int -> int = <fun>

# facr 5;;
- : int = 120
```

Q.4.1 Ecrire la fonction `fib` (Fibonacci) dans ce style.

Q.4.2 Indiquer alors comment implanter le trait `lazy` (OCaml) qui permet de ne pas évaluer une expression à sa définition mais uniquement à sa première utilisation.

Q.4.3 Traduire en Python la structure de données, la fonction `eval` et l'exemple de la factorielle.

Travaux sur Machines Encadrés

Exercice 5 : Taille mémoire des valeurs

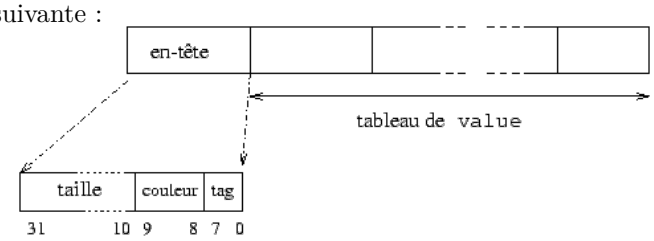
En Python on utilisera getsizeof qui calcule (en octets) la taille de surface d'une valeur :

```
>>> from sys import getsizeof
>>> getsizeof(0)
```

et en OCaml on utilisera la fonction suivante reachable_words qui calcule la taille en mots d'une valeur OCaml :

```
# let size o = Obj.reachable_words(Obj.repr o);;
val size : 'a -> int = <fun>
```

En OCaml les constructeurs paramétrés, les enregistrements et les fermetures sont allouées dans le tas sous la forme suivante :



le tableau de `value` correspond aux champs pour un enregistrement ou un n-uplet, ou l'environnement et le code d'une fermeture. L'entête contient la taille du tableau de `value`, 2 bits pour le gestionnaire mémoire, et un `tag` correspondant au constructeur qui a alloué cette valeur.

Q.5.1 Exécuter le code Python suivant et commenter les résultats.

```
>>> liste = [1,2,3]
>>> getsizeof(liste)
>>> l2 = [1, 2, 3, 4]
>>> getsizeof(l2)
>>> getsizeof(liste,l2)
```

Q.5.2 Même question pour le code OCaml suivant.

```
# let l1 = [2;4;6];;
# size l1 ;;
# let l2 = 0::l1;;
# size l2;;
# let dl = (l1,l2);;
# size dl;;
```

Q.5.3 Dessiner la représentation mémoire de `l1`, `l2` et `dl`.

Q.5.4 Calculer les tailles des expressions suivantes et expliquer les résultats obtenus.

```
let add = function x -> function y -> x + y
let c = 3
let addc = function z -> c + z
let addr = function xr -> function yr ->
            xr.contents + yr.contents
let r = {contents = 3}
let addrr = function zr -> r.contents + y.contents
let add3 = add 3
let addr3 = addr {contents = 3}
```

Q.5.5 (question optionnelle) Comment feriez-vous pour calculer la taille complète d'une valeur Python, et d'une valeur Java ?

Exercice 6 : Mini-Interprète

On considère le mini-langage suivant :

	expression
$e ::=$	
$\|c$	constante
$\|x$	variable
$\|+$	addition
$\|=$	égalité

	constante
$c ::=$	
$\|n$	constante entière (1,2,-42,...)
$\|b$	constante booléenne (true, false)

	instructions
$s ::=$	
$\|x \leftarrow c$	affectation
$\|SKIP$	skip
$\|SEQ\ s\ s$	sequence
$\|ITE\ e\ s\ s$	conditionnelle
$\|While\ e\ do\ s$	boucle

Q.6.1 Écrire un interprète en Java pour ce langage. Reprendre celui vu en cours et l'étendre avec les opérations de comparaison et les conditionnelles et boucles. Le tester avec des programmes simples.

Q.6.2 Écrire un interprète en OCaml. Reprendre celui vu en cours et l'étendre avec les opérations de comparaison et les conditionnelles et boucles. Le tester avec des programmes simples.

Quelques Liens

— en OCaml :

- Cours02b [MU4IN511], représentation mémoire : `https://www-apr.lip6.fr/~chaillou/Public/enseignement/2020-2021/ouv/Cours02b.pdf`
- "exploration des valeurs OCaml" de DAOC : `https://www-apr.lip6.fr/~chaillou/Public/DA-OCAML/book-ora116.html`

— en Python : getsizeof : `https://docs.python.org/3/library/sys.html#sys.getsizeof`