

TP n°3

Bytecode OCaml: Fonctions et fermetures

Le but de ce TP est de manipuler le code-octet généré par `ocamlc` et interprété par `ocamlrun`, en particulier le traitement des fonctions et des fermetures. On pourra se servir de l'option `-dinstr` de `ocamlc` ou de l'outil `ocamldumpobj` pour visualiser le code-octet généré. On pourra aussi consulter la spécification des instructions dans le document <http://cadmium.x9c.fr/distrib/caml-instructions.pdf>.

Exercice 1 (Compilation). Donner la suite d'instruction correspondant à la compilation de chacune de ces expressions, puis décrivez l'évolution de l'état de la machine virtuelle lors de son exécution. Enfin comparer avec la sortie d'`ocamlc -dinstr`. Si nécessaire, enlever le sucre syntaxique avant de les compiler.

1. `1+2`
2. `let x = 1 in x + 2`
3. `let x = 1 and y = 2 in x + y`
4. `let f x = x+1 in f 42 + 0`
5. `let f x = x+1 in f 42`
6. `(fun x → x+1) 42`
7. `let f x y = x+y-1 in f 2 3`
8. `let f x y = x+y-1 in let g = f 2 in g 3`
9. `let f = (let a = 1 in (fun x → x + a)) in f 1`
10. `let rec fact n = if n=0 then 1 else fact (n-1) × n`
11. `let rec fact a n = if n=0 then a else fact (a×n) (n-1)`
12. `let gensym = let c = ref 0 in fun () → c.contents ← c.contents + 1; !c in gensym ()`
13. `let id x = x in id (fun x → x+1) 10`
14. `let id x = x in id id id 10`

Exercice 2 (Examen 2011–2012). Soit le code-octet OCaml suivant :

<pre>closurerec L1, 0 const [0: 1 [0: 2 [0: 3 0a]]] push acc 1 appterm 1, 3 L1: acc 0 branchifnot L2 acc 0 getfield 1 push acc 0 branchifnot L3</pre>	<pre>offsetclosure 0 acc 0 push apply 1 push const 1 addint return 2 L2: const 0 return 1 L3: const 1 return 2</pre>
---	--

1. Exécutez ce code-octet en partant de l'état initial où la pile contient une fermeture qui, quand elle est appliquée, quitte le programme.
2. Proposez une expression OCaml pour laquelle le code-octet de la question 1 pourrait être le code compilé.

Exercice 3 (`List.map` en place). Écrire en bytecode une fonction récursive prenant en argument une liste `l` et une fonction `f`, *modifiant en place* tous les éléments `x` de `l` par `f x` et renvoyant `()`. Cette fonction est-t-elle récursive terminale? Pouvez-vous écrire cette fonction en OCaml? Quel type aurait-t-elle? Écrire un programme bien typé utilisant cette fonction et qui produit une erreur de segmentation.