TP n°4

Ocamlrun, opérations de base et valeurs algébriques

Le but de ce TP est de comprendre certaines instructions de la machine virtuelle Ocamlrun et d'observer la compilation de certaines expressions OCAML. N'hésitez pas à consulter la description du jeu d'instructions disponible à :

<http://cadmium.x9c.fr/distrib/caml-instructions.pdf>

Exercice 1. Toutes les listes d'instructions suivantes ont été générées dans le toplevel `ocaml -dinstr`. Trouvez des expressions OCaml pour générer les instructions suivantes (on ignore event `"/toplevel/"...`).

(a) <code>const 30</code>
<code>offsetint 40</code>
<code>push</code>
<code>const 10</code>
<code>addint</code>
<code>return 1</code>	(c) <code>const 1</code>
<code>push</code>
<code>const 2</code>
<code>gtint</code>
<code>branchifnot L1</code>
<code>const 3</code>
<code>return 1</code>
L1: <code>const 42</code>
<code>return 1</code>
(b) <code>const 30</code>
<code>offsetint 40</code>
<code>push</code>
<code>acc 0</code>
<code>offsetint 10</code>
<code>return 2</code>	(d) <code>const 3</code>
<code>push</code>
<code>acc 0</code>
<code>offsetint 1</code>
<code>pop 1</code>
<code>offsetint 1</code>
<code>return 1</code>

Exercice 2. Donnez la représentation sur le tas des valeurs suivantes, en ayant défini le type :

```
type 'a t = {head : 'a list; tail : 'a list}
```

Remarque : les enregistrements sont représentés comme des n -uplets (attention à l'ordre!).

- (a) `(1, 2, 3)`
- (b) `[1; 2; 3]`
- (c) `{head = []; tail = []}`
- (d) `{head = [("x",2)]; tail = [("y",1);("z",0)]}`
- (e) `{tail = [("y",1);("z",0)]; head = [("x",2)]}`
- (f) `[|1; 2; 3|]`

Vérifiez vos réponses à l'aide de la commande `ocaml -dinstr`.

Exercice 3. Soit les valeurs représentées sur le tas par les blocs :

```
[0: [0: [1: 2]]]
[2: [1: 0] [0: 1]]
```

Quelles sont-elles quand leur type t est défini par :

- (a) `type t = A | B | C | D of t | E of int | F of t*t`
 (b) `type t = A of t | B | C of int | D of t * t | E | F`

Exercice 4. Toutes les listes d'instructions suivantes ont été générées dans le toplevel `ocaml -dinstr`. Trouvez des expressions OCaml pour générer les instructions suivantes.

(a)	<pre>const [0: 2 4] push acc 0 getfield 0 offsetint 1 return 2</pre>	<pre>push const 0a push acc 1 makeblock 2, 0 push acc 2 makeblock 2, 0 return 3</pre>
(b)	<pre>const [0: 2 4] push acc 0 getfield 0 pop 1 offsetint 1 return 1</pre>	(e)
(c)	<pre>const 10 push acc 0 push const 4 ltint pop 1 branchifnot L1 const "hello" return 1 L1: const "world" return 1</pre>	<pre>const [0: 1 5] push const [0: 6 7] push acc 1 getfield 0 push const 2 eqint branchifnot L2 const 3 branch L1 L2: acc 0 getfield 1 L1: push acc 0 push acc 3 getfield 1 addint return 4</pre>
(d)	<pre>const 'a' push const 'b'</pre>	

Exercice 5. Expérimentez d'abord avec la représentation des tableaux dans la machine virtuelle. Comment un tableau d'entier est construit par des instructions de la machine virtuelle? Est-ce que c'est différent des listes d'entiers? Par exemple, observez la différence entre

```
let x = 1 in let y = 2 in [x;y]
```

et

```
let x = 1 in let y = 2 in [|x;y|]
```

Soit le type

```
type t = A | B of t | C of t array | D
```

À quelle valeur de type t correspondent les instructions :

<p>(a) <code>const [0: 0]</code>
 <code>return 1</code></p> <p>(b) <code>const [0: 0]</code>
 <code>makeblock 1, 0</code>
 <code>makeblock 1, 1</code>
 <code>return 1</code></p> <p>(c) <code>const 0</code>
 <code>makeblock 1, 0</code>
 <code>makeblock 1, 1</code>
 <code>makeblock 1, 0</code>
 <code>return 1</code></p> | <p>(d) <code>const [0: 0]</code>
 <code>push</code>
 <code>const 0</code>
 <code>push</code>
 <code>const 1</code>
 <code>makeblock 2, 0</code>
 <code>makeblock 1, 1</code>
 <code>push</code>
 <code>const 1</code>
 <code>makeblock 3, 0</code>
 <code>return 1</code></p> |
|---|---|

Exercice 6 (Facultatif). Le but de cet exercice est de commencer à comprendre comment Ocamlrun représente en mémoire les fonctions sur quelque exemple.

1. Regardez le code de la MV générés par les expressions suivantes :

```
let f x = x in f 1
let f x = x + 2 in f 1
```

Comprenez-vous toutes les instructions? Quelle partie du code est associée au corps définissant la fonction `f` et quelle partie à l'application `f 1`?

2. Êtes-vous donc capable de compiler à la main les expressions suivantes?

```
let f x = x+x in f 3
let f x = x+2 in (f 3)+(f 1)
```

3. Maintenant regardez les codes MV générés par les expressions suivantes :

```
let a = 1 in let f x = a+x in f 1
let a = 1 in let b = 3 in let f x = (x+a)*b in f 1
```

Quelle est la différence fondamentale entre ces expressions et celles du point (1)? Comprenez-vous les nouvelles instructions qui apparaissent?

4. Enfin, êtes-vous capable de compiler à la main les expressions suivantes?

```
let a = (1,2) in let f x = if x=0 then fst a else snd a in f 0
```

Pour plus sur les fonctions, il faudra attendre le prochain cours!