TP n°2

Bytecode OCaml: Arithmétique et données structurées

Le but de ce TP est de manipuler le code-octet généré par `ocamlc` et interprété par `ocamlrun`, en se restreignant aux instructions arithmétiques et de manipulation de valeurs structurées. 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 (Golf des valeurs)**.** Donner le type et la représentation en mémoire des valeurs suivante. Donner la suite d'instructions la plus courte possible qui permet de les allouer dans le tas et d'obtenir dans A un pointeur vers elles.

1. (1, 0), [|1; 0|], [1]
2. [(1, 2); (3, 4)]
3. ((), ((), ()), ())
4. Some (Some None)
5. **type** α `tree` = Leaf | Bin **of** $\alpha \times \alpha$ `tree` $\times \alpha$ `tree`;;
   Bin (1, Bin (2, Leaf, Bin (3, Leaf, Leaf)), Bin (4, Leaf, Leaf))
6. **type** α `t` = Node **of** ($\alpha \times \alpha$ `t`) `list`
   Node [(1, Node [(2, Node []); (3, Node [])]); (4, Node [])]
7. **type** `e` = Int **of** `int` | Add **of** `e` × `e` | Inv **of** `e` | Pi;;
   Add (Inv (Add (Int 1, Int 2)), Pi)
8. **let rec** `l` = 0 :: `l` **in** `l`

Exercice 2 (Lego des valeurs)**.** Soit la valeur représentée sur le tas par les trois blocs:

[0: 1 [0: 2 0 0] [0: 3 0 0]]

De quelle valeur OCaml sont issus ces blocs sachant que son type est

1. `int tree`
2. (`int` × `bool array` × `int array`)
3. (`int` × (`int` × `bool` × `unit`) × (`int` × `unit` × `bool`))
4. `bool tree`

Exercice 3 (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`.

1. 1-2+5
2. **if** $1 = 2$ **then** [] **else** [42]
3. **if** $1 = 2$ **then** [] **else true**
4. **type** α `ref` = {**mutable** `contents` : α};;
   let `x` = {`contents` = 42} **in** `x.contents` ← 43; `x`
5. **type** `t` = C1 | C2 **of** `int` | C3 **of** `int` × `int`;;
   match C3 (2,3) **with** C1 → 0 | C2 `n` → `n` | C3 (`a`,`b`) → `a`+`b`
6. **match** [1;2] **with** [] → -1 | [`x`] → 0 | `x` :: `xs` → `x`
7. **match** [] **with false** → 1 | **true** → 2
8. **let** (`x`, `y`) = (1, 2) **in** `x`+`y`
9. **let** (`x`, `y`) = 1 **in** `y`

Exercice 4 (Décompilation)**.** Décrivez l'état de la machine virtuelle lors de l'execution du programme suivant. Qu'y a-t-il dans *A* à la fin de son execution? De quelle expression peut-il être la compilation?

```
    const 0
    push
    acc 0
    push
    const 1
    makeblock 2, 0
    push
    acc 0
    branchifnot L2
    acc 0
    getfield 0
    push
    const 1
    neqint              (* "neq" dans la documentation des instructions *)
    branchif L2
    acc 0
    getfield 1
    branchif L2
    const 1
    branch L1
L2: const 0
L1: pop 2
    makeblock 0, 0
```