MV6 2015 – TP du 10 mars

Benoît Valiron

Exercice 1 : Ajout de variables dans Myrte

Dans cet exercice, vous utiliserez le fichier `mv_myrte_v2.ml`. Avec emacs, ouvrez le fichier et faites « C-c C-e » pour chaque déclaration afin de l'exécuter dans le buffer caml associé.

Le fichier contient ce qui a été vu jusqu'à la séance dernière, excepté les variables. L'objectif est de compléter le code pour prendre en compte les définitions de variables.

1. Complétez la fonction `machine` avec l'instruction `Acc n` vue en cours.
2. Complétez la fonction `compil` comme vue en cours pour supporter les constructeurs `Var` et `Let`.
3. Interpretez, compilez et exécuter la MV sur les termes sur papier puis sur machine :

```
let x = 1 in 2 + x

let x = 1 in x + 2

let x = 1 in let y = x in let x = y + 3 in y + x

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

   Est-ce que vous obtenez le résultat attendu ?

4. Voici le terme d'une slide du cours précédant que nous avons sauté :

$$\texttt{let x} = \left(\ \texttt{let x} = \texttt{1 in x} + \texttt{3}\ \right)\ \texttt{in}\\ \left(\ \left(\ \texttt{let x} = \left(\begin{array}{l}\texttt{let x} = \texttt{x} + \texttt{4}\\ \texttt{in x} + \texttt{5}\end{array}\right)\ \texttt{in x} + \texttt{2}\ \right) + \texttt{x}\ \right)$$

   Sur machine, interprétez-le, compilez-le et faites tourner la machine sur les instructions produites : comprenez-vous ce qui se passe ?

Exercice 2 : La machine virtuelle Ocamlrun

Dans cet exercice, vous allez manipuler des instructions de la machine virtuelle Ocamlrun.

1. 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

   (a) `int tree`

(b) `(int * bool array * int array)`

(c) `(int * (int * bool * unit) * (int * unit * bool))`

(d) `bool tree`

avec le type `tree` défini comme

```
type 'a tree = Node of 'a * 'a tree * 'a tree | Leaf ;;
```

2. Toutes les listes d'instructions suivantes ont été générées avec la commande

   `ocamlc -dinstr [fichier].ml`

   Comme dit en cours, sauf indiqué on omet dans la liste les deux dernières instructions. Trouvez un contenu de fichier pour générer les instructions suivante.

   (a)

   ```
   const 30
   offsetint 40
   push
   const 10
   addint
   ```

   (b)

   ```
   const 1
   push
   acc 0
   push
   const 2
   addint
   pop 1
   ```

   (c)

   ```
   const 1
   push
   acc 0
   push
   acc 1
   addint
   pop 1
   ```

   (d) Ici, les deux dernières lignes du fichier sont écrites.

   ```
   const 2
   push
   const 3
   gtint
   branchifnot L2
   const 1
   branch L1
   L2: const 4
   L1: makeblock 0, 0
   setglobal Test!
   ```

(e)

```
const [0: 1 2]
push
acc 0
getfield 1
push
acc 1
getfield 0
addint
pop 1
```

(f)

```
const 1
push
const 2
push
acc 0
push
acc 2
makeblock 2, 0
pop 2
```

3. Soit le type

   `type t = A | B of int | C | D | E of t | F of t * t | G | H of int`

   À quelle valeur de type `t` correspondent les instructions

   (a)

   ```
   const 2a
   push
   acc 0
   makeblock 1, 1
   pop 1
   ```

   (b)

   ```
   const 2a
   push
   const 3a
   push
   acc 0
   push
   acc 2
   makeblock 2, 2
   pop 2
   ```

Et si vous avez fini...

Encore plus d'instructions Ocamlrun !

1. Comme dit en cours, sauf indiqué on omet dans la liste les deux dernières instructions. Trouvez un contenu de fichier pour générer les instructions suivante.

   (a)

   ```
   const 10
   offsetint 30
   offsetint 40
   ```

   (b)

   ```
   const 40
   push
   const 30
   push
   const 10
   mulint
   mulint
   ```

   (c)

   ```
   const 1
   push
   acc 0
   pop 1
   offsetint 2
   ```

   (d)

   ```
   const 1
   push
   const 0a
   push
   acc 1
   makeblock 2, 0
   pop 1
   ```

   (e)

   ```
   const [0: 1 2]
   push
   const 0a
   push
   acc 1
   getfield 1
   makeblock 2, 0
   push
   acc 1
   getfield 0
   makeblock 2, 0
   pop 1
   ```

(f)

```
const 1
push
acc 0
push
const 2
eqint
branchifnot L2
const 3
branch L1
L2: const 4
L1: push
acc 0
push
acc 2
addint
pop 2
```

(g)

```
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
pop 3
```

2. Soit le type

   `type t = A | B of int | C | D | E of t | F of t * t | G | H of int`

   À quelle valeur de type `t` correspondent les instructions

   (a)

```
const 2
push
acc 0
```

```
push
const 3
addint
push
acc 0
makeblock 1, 3
pop 2
```

(b)

```
const 42
push
const 2a
push
acc 1
makeblock 1, 0
push
acc 1
makeblock 1, 1
push
acc 0
push
acc 2
makeblock 2, 2
pop 4
```

Et enfin, n'hésitez pas à commencer le projet !