Programmation TP 2
Mini-ML : évaluation, suite

{ jrobert, fbouchez, apardon } @ens-lyon.fr
http://perso.ens-lyon.fr/florent.bouchez/
3 octobre 2006

Dans ce TP, on va continuer à étendre l'évaluateur du précédent TP : gestion des aspects impératifs (références, effets de bord).

3 Fonctions et application

Dans le TP précédent, on a rajouté deux constructions : les fonctions et l'application.

```
type expr =
  | TInt of int                      (* i *)
  | TAdd of expr * expr              (* e1+e2 *)
  | TMul of expr * expr              (* e1*e2 *)
  | TDiv of expr * expr              (* e1/e2 *)
  | TLet of string * expr * expr     (* let x = e1 in e2 *)
  | TVar of string                   (* x *)
  | TFun of string * expr            (* fun x → e *)
  | TApp of expr * expr              (* e1(e2) *)
```

La fonction d'évaluation pouvait alors renvoyer quatre chose qu'un entier comme dans le cas de `TFun("x",TMul(TVar "x",TVar "x"))`.

Q 3.5 Testez votre fonction d'évaluation sur différents exemples (dont ceux du fichier `td02_example.ml`).

Q 3.6 Précédemment, on a *compilé* Mini-ML dans OCaml en utilisant un type `value` fonctionnel. Quel doit être le type `value` si l'on s'interdit cela ? Le but est que notre compilation soit plus *bas-niveau*. En particulier, réfléchissez à ce qui se passe lorsque l'on évalue le terme suivant :
« `(let y=3 in fun x →x*y)` ».

4 Aspects impératifs : références et affectations

Ajoutez les références implique de rajouter la déréférentiation, l'affectation et une constante « `unit` » : on redéfinit le type `expr` comme suit :

```
type expr =
  | TUnit                            (* () *)
```

```
  | TSeq of expr * expr              (* e1;e2 *)
  | TInt of int                      (* i *)
  | TAdd of expr * expr              (* e1+e2 *)
  | TMul of expr * expr              (* e2*e2 *)
  | TVar of string                   (* x *)
  | TLet of string * expr * expr     (* let x = e1 in e2 *)
  | TApp of expr * expr              (* e1(e2) *)
  | TFun of string * expr            (* fun x → e *)
  | TRef of expr                     (* ref e *)
  | TDeref of expr                   (* !e *)
  | TAffect of expr * expr           (* e1 := e2 *)
```

Remarquez que l'on n'a pas écrit « `TAffect of string * expr` » (i.e. « `x := e` ») pour le dernier constructeur : en Ocaml, on peut écrire le terme « `(f 2) := 3` », il suffit que `f` renvoie une référence.

Désormais, la fonction d'évaluation doit manipuler une *mémoire* : créer de nouvelles références, y accéder, les modifier. Comme on veut rester fonctionnel, on va écrire une fonction d'évaluation qui prend en argument un environnement et un état mémoire, et qui renverra non seulement la valeur obtenue, mais aussi le nouvel état de la mémoire.

Récapitulons :

- l'*environnement* contient les variables liées par un « let » (**`let`** `x=...` **`in`** `...`) ou une fonction (**`fun`** `x →...`). Il n'est jamais modifié.
- la *mémoire* contient l'ensemble des références manipulées par un terme. Elle est modifiée lors de l'évaluation d'un terme.

Le fichier que vous avez récupéré contient une implantation possible d'une représentation des états mémoire. Elle satisfait la signature suivante :

```
type address                    (* le type des adresses memoire *)
type 'a mem                     (* etat memoire contenant des 'a *)

(* memoire vide *)
val empty_mem

(* lecture d'une case de la memoire *)
val get_mem: 'a mem → address → 'a

(* modification d'une case de la memoire *)
val set_mem: 'a mem → address → 'a → 'a mem

(* allocation d'une nouvelle case memoire (malloc),
   initialisee a une valeur donnee *)
val alloc_mem: 'a mem → 'a → address * 'a mem
```

Nous avons deux nouvelles sortes de valeurs : les références, et la valeur « `()` ». Il nous faut donc étendre aussi le type `value` :

```
type value =
  | VUnit
  | VInt of int
```

```
  | VFun of symbol * expr
  | VRef of address
```

Q 4.1 Ce type est faux : il vous faut corriger le cas de la valeur fonctionnelle, comme pour la question 3.

Q 4.2 Écrivez la fonction d'évaluation :

```
eval: value env →value mem →expr →value * value mem
```

Q 4.3 Testez votre fonction sur les termes se trouvant dans le fichier `tp02_test.ml`.

Q 4.4 Trichez : réécrivez le type `value` et la fonction d'évaluation en utilisant les fonctions Ocaml pour représenter les fonctions miniML.