Rattrapage du TP noté de Programmation fonctionnelle avec OCaml

Durée: 2h
Tous documents autorisés.
Le barème est donné à titre indicatif.

23 juin 2020

Consignes :

- Vous pouvez consultez toute documentation (papier ou numérique), mais il est interdit de se faire aider par une tierce personne.
- Il est interdit d'utiliser des structures de contrôle impérative : boucles `for` et `while`, séquence '`;`'.
- Déposer votre fichier de code source avant 12:00 sur e-campus.

Expressions arithmétiques et tableaux

On souhaite évaluer des expressions arithmétiques contenant des tableaux, eux-mêmes indexés par une expression, pouvant contenir :

- des **constantes** entières ;
- des **variables**, nommées par leur identificateur (chaîne de caractères) ;
- un **tableau** nommé par son identificateur et indexé par une expression (le premier élément du tableau étant indexé par 0) ;
- la **somme** de deux expressions ;
- le **produit** de deux expressions.

Ces expressions sont évaluées avec un **environnement** qui permet d'associer l'identificateur désignant une variable ou un tableau à une **adresse dans la mémoire**. La mémoire sera implémentée à l'aide d'un simple tableau d'entiers.

Avec l'environnement de couples `(identificateur, adresse)` suivant : `("x", 0), ("y", 1), ("t", 2)` et la mémoire représentée dans la figure 1, l'expression $t[0]$ s'évalue donc en 5, $t[1]$ en 4, etc. On pourra également vérifier que l'expression :

$$5 \times (x - 3) + t[y + 1] \tag{1}$$

s'évalue en 42.

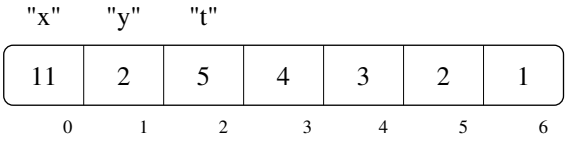

FIGURE 1 – Mémoire.

1. [3pt] Définir le type `expr` des expressions.
2. [1pt] Avec le type `expr`, représenter :
   - l'expression de l'équation 1 donnée en exemple dans l'introduction à l'aide du type `expr` ;
   - l'expression $t[t[2]]$.
3. [1pt] Lors de l'évaluation d'une expression, **l'environnement** sera représenté par un **tableau de couples** `(identificateur, adresse)` (à ne pas confondre avec le tableau mémoire, qui est un simple tableau d'entiers). L'adresse est simplement le numéro de la

case du tableau mémoire où est stockée la valeur (entière) correspondante. Représentez l'environnement indiqué dans l'exemple de l'introduction.

4. [3pt] Écrire la fonction `adresse: (string * int) array -> string -> int` qui prend en paramètres l'environnement et un identificateur, et renvoie l'adresse mémoire associée. On veillera à ne pas parcourir de cases inutilement et la fonction devra lever une exception si l'identificateur n'existe pas dans l'environnement. Avec l'environnement donné en exemple dans l'introduction, on obtient :

```
# adresse env "x";;
- : int = 0
# adresse env "t";;
- : int = 2
# adresse env "i";;
Exception: Not_found.
```

Indication : écrire une fonction récursive locale à la fonction `adresse` qui prend l'index en paramètre.

5. [5pt] Écrire la fonction `eval: (string * int) array -> int array -> expr -> int` qui prend en paramètres une expression, un environnement et un tableau mémoire, et renvoie la valeur de l'expression. Sur l'expression de l'équation 1 (`e1`) et sur l'expression $t[t[2]]$ (`e2`), on obtient respectivement :

```
# eval env memory e1;;
- : int = 42
# eval env memory e2;;
- : int = 2
```

6. [3pt] Écrire l'itérateur générique correspondant au type `expr`.
7. [2pt] Écrire une nouvelle version de l'évaluateur en utilisant l'itérateur générique.
8. [2pt] Écrire une fonction `nbvars: expr -> int` qui calcule le nombre d'identificateurs (de variable ou tableau) présents dans une expression en utilisant l'itérateur générique.

```
# nbvars e1;;
- : int = 3
```