TP noté de

Programmation impérative et fonctionnelle avec OCaml

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

3 janvier 2017

Consignes:

- Ouvrir une session locale avec le login enacaudio et le mot de passe enacaudio.
- La documentation peut être consultée en local en ouvrant avec un navigateur la page : /usr/share/doc/ocaml-doc/ocaml.html/index.html
- Déposer votre fichier de code source avant 10:00 sur e-campus.

Évaluation d'expressions arithmétiques avec adressage en mémoire

Les expressions arithmétiques considérées peuvent comporter les termes suivants :

- 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 :

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

s'évalue en 42.

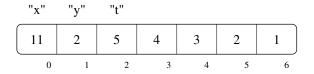


FIGURE 1 – Mémoire.

- 1. [3pt] Définir le type expr des expressions.
- 2. [1pt] Représenter l'expression 1 donnée en exemple dans l'introduction à l'aide du type expr.
- 3. [3pt] 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). L'adresse est simplement le numéro de la case du tableau mémoire où est stockée la valeur (entière) correspondante.

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 à lever une exception si l'identificateur n'existe pas dans l'environnement.

```
# adresse env "t";;
- : int = 2
```

4. [4pt] 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.

```
# eval env memory e;;
- : int = 42
```

- 5. [3pt] Écrire l'itérateur générique correspondant au type expr.
- 6. [3pt] Écrire une nouvelle version de l'évaluateur en utilisant l'itérateur générique.
- 7. [3pt] 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 e;;
- : int = 3
```

8. [Bonus] On souhaite rendre l'évaluateur générique vis-à-vis de l'implémentation de l'environnement. Soit la signature de module suivante :

```
module type EnvType = sig
  type id
  type t
  val adresse: t -> id -> int
end
```

Écrire un foncteur Expr qui prend un module Env de type EnvType en paramètre et qui définit le type expr et la fonction eval à l'aide du module Env. Écrire un module CoupleArray de type EnvType qui implémente le même environnement que celui défini dans les questions précédentes. Appliquer le foncteur Expr au module CoupleArray et vérifier que les résultats sont identiques après ouverture du nouveau module obtenu et évaluation de l'exemple.