Lycée Chateaubriand
Informatique MP 2002/03
Mathieu Giraud

TP Caml n° 5
Automates, expressions arithmétiques

Les deux parties du TP d'aujourd'hui sont indépendantes. Avant de commencer à coder, récupérez le fichier `TP5.ml`.

1 Simulation d'automates

Un automate déterministe peut se définir par son état initial, sa table de transition ainsi que par la liste de ses états finals. Les états seront ici représentés par des entiers.

```
type automate =
        { initial : int ;
          arcs    : int -> char -> int ;
          finals  : int list } ;;
```

a. Écrire la table de transition `arcs1 : int -> char -> int` d'un automate reconnaissant le langage $L_1 = a(a|b)^*b$ sur l'alphabet $\{a, b\}$.

b. Ecrire une fonction `reconnaitre : automate -> string -> bool` qui simule le comportement de l'automate et qui renvoie un booléen indiquant si le mot donné en entrée est reconnu. Tester votre fonction avec l'automate de la question précédente. Quelle est sa complexité ?

c.* Comment pourrait-on simuler un automate indéterministe ?

2 Expressions arithmétiques

On désire réaliser quelques fonctions pour traiter les expressions arithmétiques composées de constantes entières (`0`, `14`, `42`...) et d'opérateurs pris parmi `+`, `-`, `*`, et `/` (division entière). Le type suivant sera utilisé :

```
type expr =
    | Const of int
    | Add of expr * expr | Sub of expr * expr
    | Mul of expr * expr | Div of expr * expr ;;
```

On manipulera donc des expressions de la forme `Add(Const 2, Const 2)`.

2.1 Évaluation simple

d. Écrire une fonction `ecrit : expr -> string` qui "affiche" une expression arithmétique avec un parenthèsage correct.

e. Écrire une fonction `evalue : expr -> int` qui évalue récursivement une expression arithmétique. Le cas de la division par zéro sera traité par la levée d'exception `raise Division_par_zero`.

2.2 Gestion des variables locales : les environnements

On souhaite enrichir les expressions arithmétiques par l'ajout de déclarations de variables. Pour cela, ajoutez au type des expressions les constructeurs suivants :

```
type expr =
  | ...
  | Var of string
  | Let of string * expr * expr
```

L'expression `let x = 4 in x + 1` sera ainsi représentée par `Let ("x", 4, Add(Var "x", 1 ))`.

f. Modifier la fonction `ecrit` pour gérer les nouveaux cas.

La modification de `evalue` est un peu plus complexe : lors de l'évaluation de `let x = 4 in x + 1`, le programme doit se souvenir que "`x` vaut `4`" pour évaluer `x+1`. On utilise pour cela un *environnement* qui associe à un nom de variable (la clé) une valeur. Cet environnement initialement vide sera passé récursivement à chaque appel de `evalue`. Nous allons ici étudier deux solutions.

... avec des listes d'association

Une liste d'association est une liste de couples `('a, 'b) list` associant à chaque clé de type `'a` une valeur de type `'b`. Le type environnement sera donc `type env1 == (string * int) list`.

g. Écrire la fonction `associe : ('a, 'b) list -> 'a -> 'b` qui renvoie la valeur associée à la clé donnée en entrée. On devra lever une exception si la clé ne figure pas dans la liste d'association (`raise Variable_non_definie`).

h. Modifier la fonction `evalue`.

... avec des environnements fonctionnels

Puisqu'en Caml les fonctions sont des valeurs comme les autres, on peut tout à fait construire des environnements fonctionnels en posant `type env2 == string -> int`.

i. Écrire une fonction `env_vide : string -> int` qui correspond à l'environnement vide et une fonction `etend2 : env2 -> string -> int -> env2` qui étend un environnement.

j. Modifier la fonction `evalue` en conséquence.

2.3 S'il vous reste du temps...

k. Écrire une fonction `correct : expr -> bool` qui vérifie que les variables utilisées dans une expression sont bien gardées par un `let x = ... in ...` englobant.

l. Proposer un mécanisme pour évaluer le coût du calcul d'une opération (en prenant par exemple un coût 1 pour `+` et `-` et un coût 2 pour `*` et `/`...).

m.* Ajouter des instructions et l'opérateur séquence pour pouvoir écrire des mini-programmes permettant de lire et d'écrire en mémoire.

n.* Que pensez vous des variables libres (non encadrées par un `let in`)?

Sources

La partie sur les automates reprend un TP de l'année dernière; celle sur les expressions arithmétiques s'inspire d'un TP de Nicolas Ollinger.