Examen de programmation fonctionnelle en Objective Caml

Christian Rinderknecht

Mercredi 28 avril 2004

Durée : 2 heures. Documents et calculatrices ne sont pas autorisés.

1 Syntaxe et sémantique

Pour chacun des programmes suivants **dans l'ordre du sujet**,

1. construisez l'arbre correspondant (**à l'encre**),
2. dans chaque arbre, reliez les variables dans les expressions à leur lieur (**dans une encre différente**),
3. identifiez les variables libres (**à l'encre dans l'arbre et dans une phrase**),
4. décrivez formellement son exécution à l'aide de la sémantique suivante, où $[\![e]\!]\rho$ est la valeur de l'expression e dans l'environnement ρ :

$$
\begin{aligned}
[\![\overline{n}]\!]\rho &= \dot{n} \quad \text{où } \overline{n} \text{ est un entier mini-ML et } \dot{n} \in \mathbb{N} \\
[\![e_1 \texttt{+} e_2]\!]\rho &= [\![e_1]\!]\rho + [\![e_2]\!]\rho \quad \text{etc.} \\
[\![(e)]\!]\rho &= [\![e]\!]\rho \\
[\![x]\!]\rho &= \rho(x) \quad \text{(la première liaison de } x \text{ dans } \rho) \\
[\![\textsf{fun } x \to e]\!]\rho &= \langle \textsf{fun } x \to e, \rho \rangle \\
[\![\textsf{let } x = e_1 \textsf{ in } e_2]\!]\rho &= [\![e_2]\!]((x \mapsto [\![e_1]\!]\rho) \oplus \rho) \\
[\![e_1\ e_2]\!]\rho &= [\![e]\!]((x \mapsto [\![e_2]\!]\rho) \oplus \rho') \quad \text{où} \quad [\![e_1]\!]\rho = \langle \textsf{fun } x \to e, \rho' \rangle
\end{aligned}
$$

   L'évaluation consiste à appliquer les équations de la gauche vers la droite jusqu'au résultat ou une impossibilité (c.-à-d. erreur à l'exécution).

Programme 1

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

Programme 2

```
fun y -> x + (fun x -> x) y;;
```

Programme 3

```
let x = 1 in
  let f = fun y -> x + y in
  let x = 2
in f(x);;
```

Programme 4

```
let x = 0 in
  let id = fun x -> x in
  let _ = let y = 2 in id (y) in
  let x = (fun x -> fun y -> x + y) 1 2
in x+1;;
```

Programme 5

```
let compose = fun f -> fun g -> fun x -> f (g x) in
  let square = fun f -> compose f f in
  let double = fun x -> x + x in
  let quad = square double
in square quad;;
```

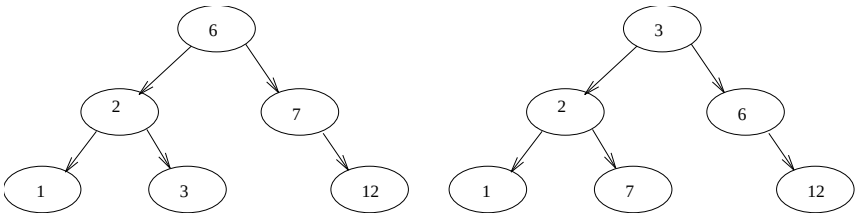
2 Programmation

Le type des arbres binaires est défini par :

```
type 'a arbre = Vide | Noeud of 'a * 'a arbre * 'a arbre
```

Un arbre binaire de recherche est un arbre binaire tel que :

- toutes les étiquettes du sous-arbre de gauche (respectivement, de droite) sont inférieures (respectivement, supérieures) à l'étiquette de la racine,
- récursivement, les deux sous-arbres sont eux-mêmes des arbres de recherche.



Le premier arbre ci-dessus est un arbre de recherche. Le second n'en est pas un car 7 est plus grand que 3 mais il se trouve dans le sous-arbre de gauche de la racine.

- Écrire une fonction `minimum` qui renvoie le plus petit élément d'un arbre binaire de recherche. On déclenchera l'exception `Not_found` si l'arbre est vide.
- Écrire une fonction `appartient` currifyée qui indique si une étiquette donnée apparaît dans un arbre de recherche donné. On tirera parti du fait que l'arbre est classé pour que la recherche soit plus rapide que dans un arbre binaire quelconque.
- Écrire une fonction `ajoute` qui ajoute un nœud dans un arbre de recherche de façon à ce que celui-ci reste un arbre de recherche. Cela se fait en rajoutant une feuille à l'arbre.