

Option Informatique

TP 2: Arbres dictionnaires

David Baelde

david.baelde@ens-lyon.org

Dans ce TP nous continuons de nous familiariser avec l'utilisation des types variants, atout essentiel de Caml qu'il faut maîtriser. Nous développerons une représentation pratique d'un ensemble de mot, sous forme d'un arbre, en mettant en oeuvre l'insertion et la recherche d'un mot. A la fin, nous calculerons des préfixes univoques minimaux dans un dictionnaire.

1 Listes associatives triées

Avant de construire et manipuler des arbres, nous allons créer un outil préliminaire : des listes associatives triées. Une liste associative est une liste de couples vue comme un ensemble d'associations $k \mapsto v$: [(k₁, v₁) ; .. ; (k_n, v_n)].

La librairie standard de Caml fournit des fonctions pour manipuler de telles listes :

```
# List.assoc 3 [ 1,"un" ; 2,"deux" ; 12,"douze" ] ;;
- : string = "deux"
```

En guise d'exercice nous allons ré-écrire les fonctions de manipulations de listes associatives de façon plus efficace, puisque nous maintiendrons des listes triées. De plus nous assurerons qu'il y a au plus une association par clé.

Pour vous simplifier la tâche, nous nous restreindrons au cas où la fonction de comparaison pour les tris est $<$. Mais les clés et les valeurs associées sont de type quelconque, j'insisterai donc sur le polymorphisme dans les types de fonctions demandées.

Nous utiliserons le type `option` pour gérer le cas où aucune association n'est trouvée, au lieu des exceptions que vous n'avez pas encore abordées. Ce type est déjà défini dans la librairie standard de Caml, comme suit :

```
type 'a option = None | Some of 'a
```

Le type `option` sert quand on n'est pas sûr de pouvoir renvoyer une valeur du bon type. On renvoie `Some v` quand on le peut, et `None` sinon.

Les types paramétrés. Vous découvrez peut-être les types paramétrés, le *polymorphisme de type*. Sur l'exemple précédent cela signifie simplement qu'on peut utiliser le constructeur `Some` sur n'importe quel paramètre de type `t`, le résultat sera alors de type `t option`.

```
# Some 12 ;;
- : int option = Some 12
# Some "douze" ;;
- : string option = Some "douze"
# None ;;
- : 'a option = None
```

► Ecrivez une fonction `assoc` de type `('a*'b) list -> 'a -> 'b option`, telle que `assoc l k` renvoie le premier `v` tel que `k,v` est dans `l`. Vous supposerez que la liste est triée par ordre croissant (pour la fonction de comparaison `<`) et veillerez à en tirer parti.

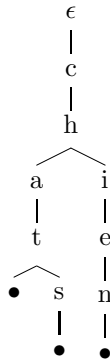
► Ecrivez une fonction d'insertion `add_assoc` de type `('a*'b) list -> 'a -> 'b -> ('a*'b) list` telle que pour toute liste d'association triée `l`, `add_assoc l k v` renvoie la liste d'association triée contenant les mêmes associations que `l`, sauf pour `k` auquel sera associé `v`. L'association éventuelle de `k` dans `l` sera effacée.

► Ecrivez une fonction qui à une liste associative non triée associe la liste associative triée, en utilisant l'insertion triée. Quand plusieurs associations apparaissent dans la liste non triée, laquelle est préservée? Savez vous évaluer la complexité de cet algorithme de tri?

2 Arbres dictionnaires

Donnons nous un *alphabet* A , *a priori* un ensemble quelconque mais ici l'alphabet romain, ou les valeurs de type `char` en Caml. Un *mot* sur l'alphabet A est une suite d'éléments de A . Le mot vide est noté ϵ . Un *langage* est un ensemble de mots. On note cw le mot $cc_1 \cdots c_n$ si w est le mot $c_1 \cdots c_n$. Pour un langage A on notera par extension $cA = \{cw | w \in A\}$.

Un dictionnaire est une représentation d'un ensemble de mots permettant une recherche facile. Nous allons utiliser pour cela une représentation arborescente. Par exemple, l'arbre suivant représente l'ensemble $\{\text{chien}, \text{chat}, \text{chats}\}$.



► Montrez que tout langage peut s'écrire sous la forme $E \cup \bigcup_{1 \leq i \leq n} c_i A_i$ où E est vide ou restreint à $\{\epsilon\}$, les A_i sont des langages et les c_i sont des caractères distincts.

Par induction sur la taille d'un langage L en utilisant la question précédente, nous savons maintenant que tout langage peut se représenter par une valeur du type suivant en Caml. La liste d'associations en paramètre représente les c_i, A_i . Le constructeur `Plein` dénotera la présence de ϵ dans l'ensemble, `Vide` sera utilisé pour un langage ne contenant pas le mot vide.

```
type dico = Plein of (char*dico) list | Vide of (char*dico) list
```

(* On retrouve ainsi l'exemple précédent *)

```
let exemple =
  Vide [('c', Vide [('h', Vide
    [('a', Vide [('t', Plein [('s', Plein [])])]);
    ('i', Vide [('e', Vide [('n', Plein [])])])]))])]
```

3 Insertion et recherche

Nous programmons maintenant les fonctions élémentaires de manipulation de dictionnaires. Vous les testerez sur quelques exemples couvrant plusieurs cas.

► Programmez la fonction `appartient` de type `dico -> char list -> bool` qui indique si un mot est présent dans un dictionnaire.

► Définissez maintenant la fonction `insertion` de type `dico -> char list -> dico` telle que `insertion dico mot` renvoie le nouveau dictionnaire correspondant à `dico` auquel on a ajouté `mot`.

4 Préfixes univoques minimaux

Pour finir nous allons calculer l'ensemble des *préfixes univoques minimaux* d'un langage. Un *préfixe* de $c_1 \dots c_n$ est un mot de la forme $c_1 \dots c_i$ pour $0 \leq$

$i \leq n$. Le préfixe est dit strict si $i < n$. Le préfixe u est dit *univoque* dans un langage L s'il est préfixe d'un unique mot du langage L . Enfin, il est *minimal* s'il n'a pas de préfixe strict qui soit encore univoque.

En français il s'agit des plus courtes abréviations d'un mot qui ne soient pas ambiguës. Pour l'exemple de langage cité plus haut, l'ensemble des préfixes univoques minimaux est : $\{chi, chats\}$

Nous allons voir que la structure de dictionnaire rend facile le calcul de l'ensemble des préfixes univoques minimaux d'un langage, noté $pum(L)$.

- ▶ Montrez que :
 - $pum(\{\epsilon\}) = \{\epsilon\}$
 - $\forall L \forall c, pum(L) = \{\epsilon\} \Rightarrow pum(cL) = \{\epsilon\}$
- ▶ Pour un langage $L = E \cup \bigcup_{1 \leq i \leq n} c_i L_i$ avec $n > 1$, exprimez $pum(L)$ en fonction des $pum(L_i)$.
- ▶ Exprimez $pum(L)$ en fonction de $pum(L_1)$ pour
 - $L = c_1 L_1$ quand $pum(L_1) \neq \{\epsilon\}$
 - $L = \{\epsilon\} \cup c_1 L_1$
- ▶ Vous pouvez maintenant définir la fonction `univoques` de type `dico -> char list list` qui renvoie la liste des préfixes univoques minimaux d'un dictionnaire.