TP6 Caml : Plus grand préfixe commun

Le but de ce TP est de réaliser un programme qui recherche le plus grand préfixe commun dans un ensemble de mots noté M. Par exemple, le plus grand préfixe commun de la liste `["indice";"initial";"initier"]` est `"in"`.
Ce problème a des applications pratiques : certains environnements informatiques offrent la possibilité de compléter les noms de fichiers autant que possible tant qu'il n'y a pas d'ambiguïté. Si par exemple, mon répertoire courant contient les trois fichiers

`recursivite.ml  recurrence.ml  arbres.ml`

alors la chaîne de caractères `re` peut être complété en `recur`, `recurs` complété directement en `recursivite.ml` et `a` en `arbres.ml`.

1 Liste d'association

Une liste d'association est une liste de type `('a*'b) list`, associant des objets de types `'b` à des objets de type `'a`. On peut ainsi associé des noms à des numéros de télephone :

```
let list_num = ["Pierre",4131;
                "Paul",2134;
                "Jacques",7697;
                "Arthur",2964;
                "Michel",9871];;
```

1. Ecrire une fonction `appartient: 'a -> ('a*'b) list -> bool` qui teste si un objet de type `'a` apparait dans une liste d'association.
Exemple :

```
#appartient "Arthur" list_num;;
- : bool = true
#appartient "David" list_num;;
- : bool = false
```

2. Ecrire une fonction `recherche: 'a -> ('a*'b) list -> 'b` qui retourne l'objet de type `'b` associé à un objet `a` de type `'a` dans une liste d'association (et une erreur si `a` n'est pas présent dans la liste).
Exemple :

```
#recherche "Paul" list_num;;
- : int = 2134
```

3. Ecrire une fonction `change : 'a -> 'b -> ('a*'b) list -> ('a*'b) list` qui modifie l'objet associé à une entrée `a` dans une liste d'association (ou crée une nouvelle asso-

ciation si `a` n'était pas présent dans la liste). On supposera qu'une entrée `a` ne peut pas apparaître plus d'une fois dans la liste.
Exemple :

```
#change "Jacques" 5555 list_num;;
- : (string * int) list =
 ["Pierre", 4131; "Paul", 2134; "Jacques", 5555; "Arthur", 2964;
  "Michel", 9871]
#change "David" 9292 list_num;;
- : (string * int) list =
 ["David", 9292; "Pierre", 4131; "Paul", 2134; "Jacques", 7697;
  "Arthur", 2964; "Michel", 9871]
```

2 Recherche du plus grand préfixe commun

2.1 Conversion chaîne de caractères ↔ liste de caractères

4. Ecrire une fonction `char_list_of_string : string -> char list` qui convertit une chaîne de caractères en une liste de caractères.
Exemple :

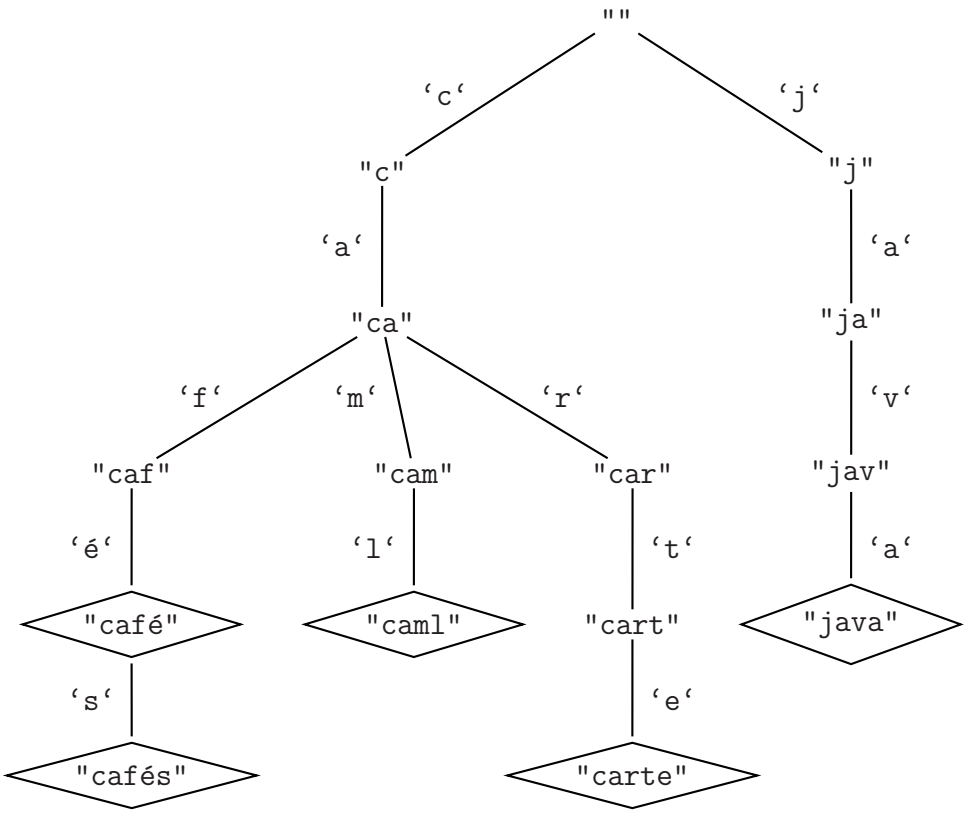
```
char_list_of_string "Caml";;
#- : char list = char_list_of_string [‘C‘; ‘a‘; ‘m‘; ‘l‘];;
```

5. Ecrire une fonction `string_of_char_list : char list -> string` qui convertit une liste de caractères en une chaîne de caractères.
Exemple :

```
#string_of_char_list [‘C‘; ‘a‘; ‘m‘; ‘l‘];;
- : string = "Caml"
```

2.2 Utilisation d'un arbre $n-$aire

Pour pouvoir faire une recherche efficace, on construit à partir des mots de M un arbre $n-$aire dont les branches sont étiquetées avec des lettres et les nœuds avec des mots, qui sont soit des préfixes de M, soit des mots de M.
Exemple : si M =`["caml";"café";"cafés";"carte";"java"]`, alors l'arbre est le suivant :



Nous considérerons le type `arbre` suivant :

```
type etiquette = Mot of string
               | Prefixe of string ;;

type arbre = Noeud of etiquette * (char * arbre) list ;;
```

Les différentes branches partant d'un nœud sont représentées par une liste d'association qui associe des étiquettes avec des arbres fils.

2.2.1 Recherche dans l'arbre

Vous testerez les fonctions de cette section sur les exemples donnés dans le fichier `TP6.squ`. Les arbres donnés sont les suivants :

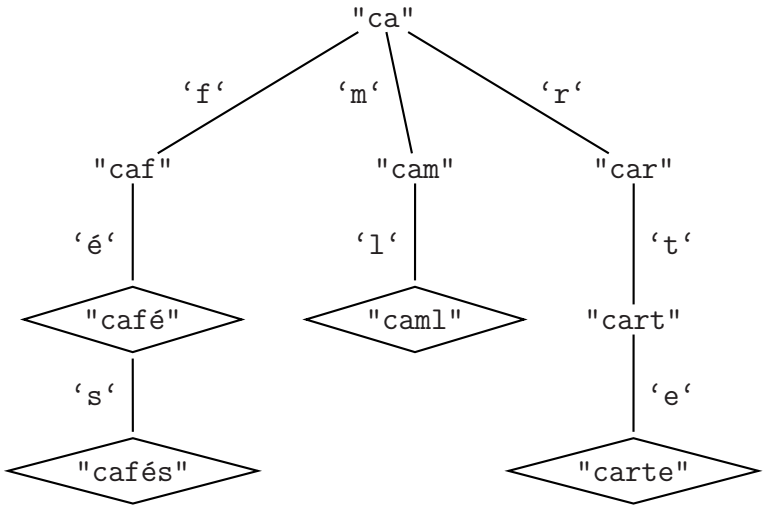
arbre	ensemble
arbre1	["caml";"café";"cafés";"carte";"java"]
arbre2	["cantine";"canard";"canari";"candide"]
arbre3	["abricot";"allumette";"allumer"; "caml";"café";"cafés";"carte";"java"; "cantine";"canard";"canari";"candide"]

6. Ecrire une fonction `plus_grand_prefixe : arbre -> string` qui calcule le plus grand préfixe commun d'un ensemble de mots.

Exemple :

```
#plus_grand_prefixe arbre2;;
- : string = "can"
```

7. Ecrire une fonction `arbre_prefixe : string -> arbre -> arbre` qui renvoie, pour un mot p, le sous-arbre des mots de M dont p est un préfixe. Cette fonction renvoie une erreur si le préfixe donné ne correspond à aucun mot dans l'arbre.
Exemple :
`arbre_prefixe "ca" arbre1 =`



8. Ecrire une fonction `complete : arbre -> string -> string` recherchant le plus grand complément possible d'un mot p, c'est-à-dire le plus grand préfixe commun à tous les mots de M dont p est un préfixe.
Exemple :

```
#complete arbre3 "al";;
- : string "allume"
```

9. Ecrire une fonction `trouve_complements : arbre -> string -> string list` qui renvoie, pour un mot p, tous les mots de M dont p est préfixe.
Pour cette fonction vous aurez besoin de plusieurs sous-fonctions dont la fonction `map` de Caml et une fonction `concat_list : 'a list list -> 'a list` (à écrire) qui concatène les listes d'une liste.
Exemple :

```
#concat_list [[1;2];[3];[];[4;5;6]];;
- : int list = [1; 2; 3; 4; 5; 6]
```

2.2.2 Construction de l'arbre

L'arbre de recherche est construit en partant d'un arbre initialement vide (`Noeud (Prefixe "",[])`) et en y insérant les mots de M un à un.
10. Ecrire une fonction `insere_mot_liste : arbre -> char list -> arbre` qui insère un mot, donné sous forme de liste de caractères, dans un arbre. En déduire une fonction `insere_mot : arbre -> string -> arbre` qui insère un mot de type `string`.

11. En déduire une fonction `construit_arbre : string list -> arbre` qui construit l'arbre associé à une liste de mots. Tester votre programme.