Université Paris Diderot — Programmation Fonctionnelle
Licence — Année 2010-2011

TP n°3

Un générateur de spam en OCaml

La génération de texte pseudo-aléatoire est un problème bien connu des producteurs de spams. Une méthode pour générer ce type de texte consiste à partir d'un texte réel (*e.g.* le contenu d'une ou plusieurs pages web), et de construire pour ce texte une *table de succession*, sous forme de liste d'association.

Table de succession

Pour chaque mot `m` du texte, une table de succession contient un couple de la forme `(m, ls)`, où `ls` est une liste de mots contenant, pour chaque occurrence de `m` dans le texte, un exemplaire du mot qui suit cette occurrence. Voici par exemple un texte dû à Aristote :

> *"Le faux et le vrai ne sont pas dans les choses, comme si le bien était le vrai et le mal, en lui-même le faux, mais dans la pensée, et en ce qui regarde les natures simples et les essences, le vrai et le faux n'existent pas même dans la pensée."*

En convertissant les majuscules en minuscules, et considérant l'espace comme l'unique séparateur de mots, voici le début sa table de succession :

```
[("le",   ["faux"; "vrai"; "faux,"; "mal,";
           "vrai"; "bien"; "vrai"; "faux"]);
 ("faux", ["n'existent"; "et"]);
 ("et",   ["le"; "les"; "en"; "le"; "le"]); ...]
```

Le mot *"le"* a huit occurrences dans le texte. Trois d'entre elles sont suivies par le mot *"vrai"*, qui apparaît donc trois fois dans sa liste associée. Noter que le mot *"pensée."* a une seule occurrence, et celle-ci est la dernière du texte : le couple correspondant dans la table est donc `("pensée.", []`).

Génération du texte

La génération d'un texte aléatoire à partir d'une table de succession se fait par la méthode suivante : le premier mot du texte généré est choisi au hasard dans la table. A chaque étape, on prend la liste associé au mot courant dans la table, et le mot suivant est choisi au hasard dans cette liste. Si le mot courant est le dernier, le mot suivant est à nouveau choisi au hasard dans la table.

Noter que plus un mot à tendance à suivre le mot courant dans le texte initial, et plus il a de chances d'être le mot suivant. Le résultat ressemble donc à l'original, tout en étant différent :

> *"pas même dans la pensée, et le bien était le mal, en ce qui regarde les choses, comme si le faux n'existent pas même dans la pensée, comme si le vrai et les essences, le bien était le vrai"*

On considèrera l'espace comme l'unique séparateur des mots d'un texte. Il peut avoir un ou plusieurs espaces entre deux mots, ainsi qu'avant le premier et le dernier mot d'un texte.

Exercice 1 Les fonctions utilitaires suivantes sont deux à deux indépendantes.

- Écrire une fonction `simplifier : 'a list -> 'a list` renvoyant, à partir d'une liste quelconque, la liste obtenue en ne gardant qu'une seule occurrence de chaque élément. Exemple :
  ```
  simplifier  ["ab"; "cd"; "ab"; "ab"; "ef"] = ["ab"; "cd"; "ef"]
  (* ou ["ef"; "cd"; "ab"] selon l'implementation *);;
  ```
- Écrire une fonction `successeurs : 'a -> 'a list -> 'a list` telle que `successeurs x l` renvoie la liste des éléments situés après chacune des occurrences de `x` dans `l`. Exemples :
  ```
  successeurs "cd" ["ab"; "cd"; "ab"; "ab"; "ef"] = ["ab"]
  successeurs "ab" ["ab"; "cd"; "ab"; "ab"; "ef"] = ["cd"; "ab"; "ef"]
  successeurs "ef" ["ab"; "cd"; "ab"; "ab"; "ef"] = []
  ```
- A partir des fonctions prédéfinies `String.contains`, `String.index`, `String.sub`, écrire une fonction `decouper : string -> string list`. Appliquée à un texte, cette fonction doit renvoyer la liste formée de la suite des mots du texte, dans l'ordre où ils apparaissent. Exemples :
  - `decouper " abc abc   def " = ["abc"; "abc"; "def"]`
  - `decouper " abc" = ["abc"]`
  - `decouper "" = []`

Exercice 2 En vous servant de `simplifier`, `successeurs` et `List.map`, écrire une fonction :

```
table : string list -> (string * (string list)) list
```

telle que si `lm` est un texte découpé en liste de mots, `table lm` renvoie la table de succession de ce texte.

Exercice 3 Appliquée à un entier positif `n`, la fonction `Random.int` renvoie un entier compris entre `0` et `n-1` inclus. A partir de cette fonction, de `List.length` et `List.nth`, écrire :

```
piocher : 'a list -> 'a
```

telle que pour toute liste `l` non vide, `piocher l` renvoie un élément choisi au hasard dans `l`.

Exercice 4 À partir de `piocher` et de `List.assoc`, écrire une fonction :

```
generer : int -> (string * (string list)) list -> string
```

telle que si `n` est un entier et `t` est une table de successions, `generer n t` renvoie à partir de cette table et suivant la méthode décrite dans l'introduction, un texte aléatoire à `n` mots, séparés par des espaces. Vous pouvez écrire une ou plusieurs fonctions auxiliaires.

Exercice 5 En déduire une fonction

```
spam : int -> string -> string
```

telle que si `n` est un entier et `s` est un texte, `spam n s` renvoie un texte aléatoire de `n` mots séparés par des espaces, construit à partir de `s`. Pour de meilleurs résultats, commencez par convertir le texte initial en minuscules, à l'aide de `String.lowercase`.