

LICENCE SCIENCES & TECHNOLOGIES
1^{re} ANNÉE

UE INF201, INF231
ALGORITHMIQUE ET PROGRAMMATION FONCTIONNELLE
2020 / 2021

PROJET

ENSEMBLES, MULTI-ENSEMBLES ET CONTREPET

Objectifs. Ce projet vous permettra de vous exercer à manipuler les séquences, une structure de données récursive fondamentale en informatique en général, et en programmation fonctionnelle en particulier. Vous :

1. définirez d'abord des fonctions afin de manipuler des ensembles et des multi-ensembles ;
2. revisiterez ensuite certaines fonctions en utilisant l'ordre supérieur ;
3. appliquerez enfin ces fonctions pour étudier le contrepét.

Planning. Ce projet se fait en trinôme et se décompose en deux phases. Vous devez rendre selon les modalités fixées par vos enseignants de TD/TP :

- 1) au plus tard le 3 avril à 23h59 : Q1 (partie 2) et Q2 (partie 3) ;
- 2) au plus tard le 1^{er} mai à 23h59 : les autres questions.

Lors de la dernière séance de TP, vous devrez effectuer une soutenance de votre projet avec démonstration sur machine devant votre enseignant, éventuellement à distance.

Les détails des modalités de la soutenance et des rendus seront précisés par vos enseignants.

Table des matières

1	Introduction	2
2	Ensembles (8pt)	2
3	Réusinage : listes OCAML et ordre supérieur (6pt)	4
4	Multi-ensembles (6pt)	5
5	Contrepét	7
6	Optimisations et extensions	11

Rappels

- *définir* = donner une définition,
définition = spécification + réalisation ;
- *spécifier* = donner une spécification (le « quoi »),
spécification = profil + sémantique + exemples et/ou propriétés ;
- *réaliser* = donner une réalisation (le « comment »),
réalisation = algorithme (langue naturelle) + implémentation (OCAML) ;
- *implémenter* = donner une implémentation (OCAML).

Dans certains cas, certaines de ces rubriques peuvent être omises.

1 Introduction

Ce projet nécessite de représenter des groupements de valeurs. Les valeurs sont prises dans un *réservoir*, noté α . Par exemple, si $\alpha = \mathbb{N}$, on groupe des entiers naturels.

Il existe de multiples façons de grouper des valeurs. Ce projet en examine deux : les ensembles et les multi-ensembles.

Un *ensemble* $_{\alpha}$ est un groupement non ordonné et sans répétition de valeurs prises dans α . Cette façon de grouper les valeurs est étudiée dans la partie 2.

Dans un *multi-ensemble* (étudié dans la partie 4), on autorise les valeurs à être répétées ; il est donc possible qu'il y ait plusieurs exemplaires (on dit aussi occurrences) de la même valeur.

2 Ensembles (8pt)

Nous adopterons la définition récursive suivante des ensembles :

$$\text{ensemble}_{\alpha} \stackrel{\text{def}}{=} \{\emptyset\} \cup \{C_e(elt, ens) \mid elt \in \alpha, ens \in \text{ensemble}_{\alpha}, elt \notin ens\}$$

Dans cette définition :

- \emptyset dénote l'ensemble vide ;
- elt est un élément pris dans le réservoir α ;
- $C_e(elt, ens)$ dénote un ensemble non vide comportant (au moins) l'élément elt .

Par exemple, avec $\alpha = \mathbb{Z}$, le singleton (ensemble d'un seul élément) contenant l'entier 1 est noté : $C_e(1, \emptyset)$.

Afin de pouvoir manipuler des ensembles, nous spécifions les fonctions suivantes.

1. **Cardinalité** d'un ensemble :

Profil $\text{cardinal}_e : \text{ensemble}_{\alpha} \rightarrow \mathbb{N}$

Sémantique : $\text{cardinal}_e(ens)$ est le nombre d'éléments de ens .

2. **Appartenance** d'un élément à un ensemble :

Profil $\text{appartient}_e : \alpha \rightarrow \text{ensemble}_\alpha \rightarrow \mathbb{B}$

Sémantique : $(\text{appartient}_e \text{ elt } \text{ens})$ est vrai si et seulement si $\text{elt} \in \text{ens}$.

3. **Inclusion** de deux ensembles :

Profil $\text{inclus}_e : \text{ensemble}_\alpha \rightarrow \text{ensemble}_\alpha \rightarrow \mathbb{B}$

Sémantique : $(\text{inclus}_e \text{ ens}_1 \text{ ens}_2)$ est vrai si et seulement si $\text{ens}_1 \subset \text{ens}_2$

4. **Ajout** d'un élément à un ensemble :

Profil $\text{ajoute}_e : \alpha \rightarrow \text{ensemble}_\alpha \rightarrow \text{ensemble}_\alpha$

Sémantique : $(\text{ajoute}_e \text{ elt } \text{ens})$ est l'ensemble obtenu en ajoutant l'élément elt à l'ensemble ens en respectant la contrainte de non répétition des éléments.

5. **Suppression** d'un élément d'un ensemble :

Profil $\text{supprime}_e : \alpha \rightarrow \text{ensemble}_\alpha \rightarrow \text{ensemble}_\alpha$

Sémantique : $(\text{supprime}_e \text{ elt } \text{ens})$ supprime l'élément elt de l'ensemble ens .

6. **Égalité** de deux ensembles :

Profil $\text{egaux}_e : \text{ensemble}_\alpha \rightarrow \text{ensemble}_\alpha \rightarrow \mathbb{B}$

Sémantique : $(\text{egaux}_e \text{ ens}_1 \text{ ens}_2)$ est vrai si et seulement si ens_1 et ens_2 ont les mêmes éléments.

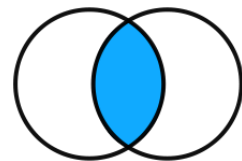
Exemple : $(\text{egaux}_e \text{ } C_e(1, C_e(2, \emptyset)) \text{ } C_e(2, C_e(1, \emptyset))) = \text{vrai}$

7. **Intersection** de deux ensembles :

Profil $\text{intersection}_e : \text{ensemble}_\alpha \rightarrow \text{ensemble}_\alpha \rightarrow \text{ensemble}_\alpha$

Sémantique : $(\text{intersection}_e \text{ ens}_1 \text{ ens}_2)$ est l'ensemble $\text{ens}_1 \cap \text{ens}_2$, c'est-à-dire l'ensemble des éléments appartenant à la fois à ens_1 et à ens_2 .

Une représentation graphique de l'intersection de deux ensembles est :

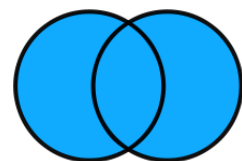


8. **Union** de deux ensembles :

Profil $\text{union}_e : \text{ensemble}_\alpha \rightarrow \text{ensemble}_\alpha \rightarrow \text{ensemble}_\alpha$

Sémantique : $(\text{union}_e \text{ ens}_1 \text{ ens}_2)$ est l'ensemble $\text{ens}_1 \cup \text{ens}_2$, c'est-à-dire l'ensemble des éléments appartenant à ens_1 ou à ens_2 .

Une représentation graphique de l'union de deux ensembles est :

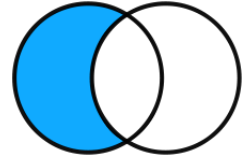


9. **Différence** de deux ensembles :

Profil $dif_e : ensemble_\alpha \rightarrow ensemble_\alpha \rightarrow ensemble_\alpha$

Sémantique : $(dif_e\ ens_1\ ens_2)$ est l'ensemble $ens_1 \setminus ens_2$ (se lit « ens_1 privé de ens_2 »), c'est-à-dire l'ensemble des éléments qui appartiennent à ens_1 mais pas à ens_2 .

Une représentation graphique de la différence de deux ensembles est :

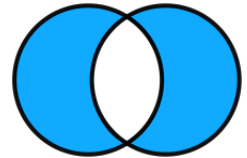


10. **Différence symétrique** de deux ensembles :

Profil $difsym_e : ensemble_\alpha \rightarrow ensemble_\alpha \rightarrow ensemble_\alpha$

Sémantique : $(difsym_e\ ens_1\ ens_2)$ est l'ensemble $ens_1 \Delta ens_2$ (se lit « ens_1 delta ens_2 »), c'est-à-dire l'ensemble des éléments qui appartiennent soit à ens_1 , soit à ens_2 , mais pas au deux à la fois.

Une représentation graphique de la différence symétrique de deux ensembles est :



Les ensembles seront implémentés par le type somme récursif polymorphe suivant :

```
type 'a ensemble = (* 'a est le réservoir d'éléments *)
  | Ve              (* V mis pour "vide", e pour "ensemble" *)
  | Ce of 'a * 'a ensemble (* C mis pour "constructeur" *)
```

Q1. En utilisant le type `'a ensemble`, réaliser les fonctions spécifiées ci-dessus.

Remarque De façon à limiter au maximum la duplication de code, vous veillerez à utiliser la composition fonctionnelle avec les fonctions que vous aurez déjà définies.

3 Réusinage : listes OCAML et ordre supérieur (6pt)

Le *réusinage* (ou refactoring) est l'opération consistant à retravailler le code source d'un programme informatique – sans toutefois y ajouter des fonctionnalités – de façon à en améliorer la lisibilité et par voie de conséquence la maintenance, ou à le rendre plus générique.¹

Q2. Réusiner le code de la partie précédente en implémentant le type `'a ensemble` grâce aux listes natives d'OCAML, mais sans utiliser les fonctions des bibliothèques d'OCAML.

1. Source wikipedia : fr.wikipedia.org/wiki/Réusinage_de_code

Q3. En utilisant les fonctions d'ordre supérieur vues en cours/TD/TP (*fold*, *map*, *for-all*, *exists*, *filter*, ...), réusiner les fonctions suivantes de la partie précédente :

- a) *cardinal_e*,
- b) *appartient_e*,
- c) *inclus_e*,
- d) *supprime_e*,
- e) *intersection_e*,
- f) *union_e*,
- g) *dif_e*.

4 Multi-ensembles (6pt)

Les multi-ensembles autorisant de multiples exemplaires d'une même valeur², nous introduisons la notion de multi-élément. Un *multi-élément* représente tous les exemplaires d'une même valeur appartenant à un multi-ensemble.

On peut représenter un multi-élément par un élément du réservoir α appelé *élément support*, ou plus simplement *support*, accompagné d'un entier naturel appelé *multiplicité* :

$$\text{multielement}_{\alpha} \stackrel{\text{def}}{=} \alpha \times \mathbb{N}$$

Par exemple, si $\alpha = \{ 'a', \dots, 'z' \}$, le multi-élément $('m', 3)$ est composé du support $'m'$ avec la multiplicité 3. La multiplicité d'un élément indique le nombre d'occurrences (d'exemplaires) de l'élément.

Un *multi-ensemble* est une collection non ordonnée de multi-éléments :

$$\text{multiensemble}_{\alpha} \stackrel{\text{def}}{=} \{ [] \} \cup \{ (x, m) :: \text{ens} \mid (x, m) \in \text{multielement}_{\alpha}, \text{ens} \in \text{multiensemble}_{\alpha}, \forall m' \in \mathbb{N}, (x, m') \notin \text{ens} \}$$

Dans cette définition :

- `[]` dénote le multi-ensemble vide ; `::` est l'opérateur OCAML d'ajout à gauche ;
- \in (respectivement \notin) dénote l'appartenance (resp. non appartenance) classique de la théorie des ensembles.

La dernière condition est appelée « *contrainte de non-répétition de multi-éléments de même support* ». Ainsi, les multi-éléments ne sont pas répétés, mais du fait des multiplicités, tout se passe comme si les supports apparaissaient plusieurs fois. Par exemple, le multi-ensemble $[('m', 3) ; ('u', 1)]$ « contient » 3 occurrences de $'m'$ et 1 occurrence de $'u'$.

Afin de pouvoir manipuler des multi-ensembles, nous spécifions les fonctions suivantes.

1. **Cardinalité** d'un multi-ensemble :

Profil $\text{cardinal}_m : \text{multiensemble}_{\alpha} \rightarrow \mathbb{N}^2$

-
2. Revoir l'introduction page 2

Sémantique : $\text{cardinal}_m(\text{mens})$ est le couple $(n_{\text{melt}}, n_{\text{tot}})$ où n_{melt} est le nombre de multi-éléments du multi-ensemble mens et n_{tot} est le nombre total d'occurrences des éléments.

2. **Occurrence** d'un support dans un multi-ensemble :

Profil $\text{occurrence} : \alpha \rightarrow \text{multiensemble}_\alpha \rightarrow \mathbb{N}$

Sémantique : $(\text{occurrence } x \text{ mens})$ est le nombre d'occurrences du support x dans mens .

3. **Appartenance** d'un multi-élément à un multi-ensemble :

Profil $\text{appartient}_m : \text{multielement}_\alpha \rightarrow \text{multiensemble}_\alpha \rightarrow \mathbb{B}$

Sémantique : $(\text{appartient}_m \text{ melt mens})$ est vrai si et seulement si la multiplicité de melt est inférieure ou égale au nombre d'occurrences de son support dans mens .

4. **Inclusion** de deux multi-ensembles :

Profil $\text{inclus}_m : \text{multiensemble}_\alpha \rightarrow \text{multiensemble}_\alpha \rightarrow \mathbb{B}$

Sémantique : $(\text{inclus}_m \text{ mens}_1 \text{ mens}_2)$ est vrai si et seulement si tout élément de mens_1 appartient à mens_2 .

Exemples :

(a) $(\text{inclus}_m [(\text{'u'}, 1)] [(\text{'u'}, 2)]) = \text{vrai}$

(b) $(\text{inclus}_m [(\text{'u'}, 2)] [(\text{'u'}, 1)]) = \text{faux}$

5. **Ajout** d'un multi-élément à un multi-ensemble :

Profil $\text{ajoute}_m : \text{multielement}_\alpha \rightarrow \text{multiensemble}_\alpha \rightarrow \text{multiensemble}_\alpha$

Sémantique : $(\text{ajoute}_m \text{ elt mens})$ est le multi-ensemble obtenu en ajoutant le multi-élément elt au multi-ensemble mens en respectant la contrainte de non répétition de multi-éléments de même support.

6. **Suppression** d'un multi-élément d'un multi-ensemble :

Profil $\text{supprime}_m : \text{multielement}_\alpha \rightarrow \text{multiensemble}_\alpha \rightarrow \text{multiensemble}_\alpha$

Sémantique : $(\text{supprime}_m (x, n) \text{ mens})$ supprime n occurrences du support x du multi-ensemble mens . Si n est supérieur ou égal au nombre d'occurrences de x dans mens , alors x disparaît complètement de mens . Selon le besoin, il pourra être pratique d'implémenter en plus la fonctionnalité suivante : si $n = 0$, toutes les occurrences de x sont supprimées ().

7. **Égalité** de deux multi-ensembles :

Profil $\text{egaux}_m : \text{multiensemble}_\alpha \rightarrow \text{multiensemble}_\alpha \rightarrow \mathbb{B}$

Sémantique : $(\text{egaux}_m \text{ mens}_1 \text{ mens}_2)$ est vrai si et seulement si mens_1 et mens_2 ont les mêmes multi-éléments.

Exemples :

(a) $(\text{egaux}_m [(\text{'a'}, 1), (\text{'b'}, 2)] [(\text{'a'}, 1), (\text{'b'}, 2)]) = \text{vrai}$

$$(b) \text{ (egaux}_m [('a',1), ('b',2)] [('b',2), ('a',1)]) = \text{vrai}$$

8. **Intersection** de deux multi-ensembles :

Profil $\text{intersection}_m : \text{multiensemble}_\alpha \rightarrow \text{multiensemble}_\alpha \rightarrow \text{multiensemble}_\alpha$

Sémantique : $(\text{intersection}_m \text{ mens}_1 \text{ mens}_2)$ est le multi-ensemble des éléments appartenant à la fois à mens_1 et à mens_2 .

Exemple : $(\text{intersection}_m [('m',3); ('u',1)] [('m',1); ('a',1)]) = [('m',1)]$

9. **Union** de deux multi-ensembles :

Profil $\text{union}_m : \text{multiensemble}_\alpha \rightarrow \text{multiensemble}_\alpha \rightarrow \text{multiensemble}_\alpha$

Sémantique : $(\text{union}_m \text{ mens}_1 \text{ mens}_2)$ est le multi-ensemble des éléments appartenant à mens_1 ou à mens_2 .

Exemple : $(\text{union}_m [('m',3); ('u',1)] [('m',1); ('a',1)]) = [('m',3); ('u',1); ('a',1)]$

10. **Différence** de deux multi-ensembles :

Profil $\text{dif}_m : \text{multiensemble}_\alpha \rightarrow \text{multiensemble}_\alpha \rightarrow \text{multiensemble}_\alpha$

Sémantique : $(\text{dif}_m \text{ mens}_1 \text{ mens}_2)$ est le multi-ensemble obtenu en supprimant les multi-éléments appartenant à mens_2 de mens_1 .

11. **Différence symétrique** de deux multi-ensembles :

Profil $\text{difsym}_m : \text{multiensemble}_\alpha \rightarrow \text{multiensemble}_\alpha \rightarrow \text{multiensemble}_\alpha$

Sémantique : $(\text{difsym}_m \text{ mens}_1 \text{ mens}_2)$ est le multi-ensemble des multi-éléments qui appartiennent soit à mens_1 , soit à mens_2 , mais pas aux deux à la fois.

Q4. Définir en OCAML les types $\text{multielement}_\alpha$ et $\text{multiensemble}_\alpha$.

Q5. Réaliser les fonctions spécifiées ci-dessus.

Remarque L'utilisation de l'ordre supérieur sera appréciée.

5 Contrepet

5.1 Introduction

Nous donnons d'abord quelques définitions :

contrepèterie (n. f.) : Inversion de l'ordre des syllabes, des lettres ou des mots.

L'objectif est de produire des phrases burlesques ou grivoises.

Par exemple, intervertir les lettres soulignées dans la phrase : « quelle ministre sèche ! » donne une nouvelle phrase ayant un tout autre sens.

contrepet (n. m.) Art de résoudre ou d'inventer les contrepèteries.

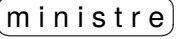
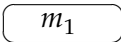
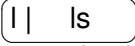
À partir d'une phrase donnée, nous souhaitons étudier la génération des contrepèteries possibles. Nous nous limiterons à la permutation de deux lettres pour former une phrase dont tous les mots sont des mots valides. Un mot est estimé *valide* si et seulement si il appartient à un certain dictionnaire³ défini par l'utilisateur. Nous commencerons donc par construire un dictionnaire qui regroupe l'ensemble des mots estimés valides.

Les phrases que nous manipulerons seront uniquement composés de lettres minuscules non accentuées :

$$\text{lettre} \stackrel{\text{def}}{=} \{'a', \dots, 'z'\}$$

Un **mot** est défini comme une séquence de lettres, une **phrase** comme une séquence de mots.

Q6. Implémenter les types `mot` et `phrase` grâce au type `list` d'OCAML.

Notation Dans la suite, pour alléger les notations, un mot pourra être représenté par un rectangle aux coins arrondis contenant la liste de ses lettres. Par exemple : . On pourra également noter l'identificateur du mot dans le rectangle ; l'ambiguïté sera levée par le contexte. Par exemple,  représente un mot d'identificateur m_1 . On pourra également isoler des lettres grâce à une barre verticale. Par exemple,  représente le mot dont la première lettre est identifiée par la variable `l`, et la suite des lettres par la variable `ls`.

5.1.1 Conversion de chaînes en mots

Pour fabriquer des jeux d'essais afin de tester vos algorithmes, il est pratique d'utiliser des fonctions de conversion entre chaînes de caractères et mots. On spécifie ainsi :

Profil $\text{chaîneVmot}, \text{cVm} : \text{string} \rightarrow \text{mot}$

Sémantique : $\text{chaîneVmot}(s)$ est le mot correspondant à la chaîne s .
 cVm est un synonyme de chaîneVmot .

Exemple : $(\text{cVm } \text{"algorithm"}) = ['a'; 'l'; 'g'; 'o'; 'r'; 'i'; 't'; 'h'; 'm'; 'e']$

La version courte de l'identificateur de cette fonction (cVm) pourra être utilisé dans le toplevel pour faire des tests rapides ; dans les sources, on privilégiera la version longue (chaîneVmot).

L'exemple montre à quel point il est plus rapide et pratique d'utiliser cVm , plutôt que d'écrire directement des séquences de caractères.

On donne l'implémentation de cette fonction afin que vous puissiez la copier-coller dans votre fichier source :

```
let chaîneVmot (ch:string) : char list =  
  List.of_seq (String.to_seq ch)  
  
let cVm : string -> char list =  
  chaîneVmot
```

Remarque Les fonctions `List.of_seq` et `String.to_seq` n'existent que dans OCAML ≥ 4.07 , et sont hors-programme.

3. Il ne s'agit pas des dictionnaires Python vus dans l'UE Inf101 au 1^{er} semestre.

Q7. Utiliser *chaineVmot* pour définir des constantes implémentant les mots des phrases :

- $QMS \stackrel{\text{def}}{=} \text{« quelle ministre sèche »}$
- $QSM \stackrel{\text{def}}{=} \text{« quelle sinistre mère »}$

(sans les accents : on rappelle qu'on ne manipule que des minuscules non accentuées). En déduire la définition de deux constantes `cstQMS` et `cstQSM` implémentant QMS et QSM en OCAML.

5.1.2 Conversion de phrases en séquences de chaînes

La lecture des phrases sous forme de séquences de séquences de caractères n'est pas aisée. En effet : `["quelle" ; "ministre" ; "seche"]` est plus facile à lire que `[['q' ; 'u' ; 'e' ; 'l' ; 'l' ; 'e'] ; ['m' ; 'i' ; 'n' ; 'i' ; 's' ; 't' ; 'r' ; 'e'] ; ['s' ; 'e' ; 'c' ; 'h' ; 'e']]`.

On spécifie donc :

Profil $\text{phraseVseqstring}, pVs : \text{phrase} \rightarrow \text{séq}(\text{string})$

Sémantique : $\text{phraseVseqstring}(ph)$ est la séquence de chaînes correspondant à ph .

Exemple : $(\text{phraseVseqstring } \text{cstQMS}) = [\text{"quelle"} ; \text{"ministre"} ; \text{"seche"}]$

La version courte de l'identificateur de cette fonction (pVs) pourra être utilisé dans le toplevel pour faire des tests rapides ; dans les sources, on privilégiera la version longue (phraseVseqstring).

On donne l'implémentation de cette fonction afin que vous puissiez la copier-coller dans votre fichier source :

```
let phraseVseqstring : phrase -> string list =
  List.map (fun m -> String.of_seq (List.to_seq m))

let pVs : phrase -> string list =
  phraseVseqstring
```

Remarque Les fonctions `String.of_seq` et `List.to_seq` n'existent que dans OCAML ≥ 4.07 , et sont hors-programme.

Il est conseillé d'utiliser cette fonction dans votre projet pour faciliter la lecture des valeurs de type *phrase*.

5.2 Dictionnaire

Un **dictionnaire** sera (naïvement) défini comme un ensemble de mots. On pose donc $\alpha = \text{mot}$ dans la définition du début de la partie 2.

Q8. Implémenter le type dictionnaire grâce au type `'a` ensemble.

Q9. Définir le dictionnaire appelé `cst_DICO` contenant les mots de la contrepèterie donnée en exemple plus haut.

Q10. Enrichir le dictionnaire `cst_DICO` afin de permettre plusieurs exemples de vos propres contrepèteries.

5.3 Vérificateur de contrepèteries

Dans un premier temps, nous voulons savoir si deux phrases sont contrepèteries l'une de l'autre. Pour cela, nous spécifions les fonctions suivantes :

Profil `supprimePrefixeCommun` : $\text{mot} \rightarrow \text{mot} \rightarrow \text{mot}^2$

Sémantique : Étant donnés deux mots m_1 et m_2 , posons $(m'_1, m'_2) = (\text{supprimePrefixeCommun } m_1 \ m_2)$. m'_1 (resp. m'_2) est le mot obtenu en supprimant de m_1 (resp. m_2) le préfixe commun à m_1 et m_2 .

Propriété : $\forall p, l_1s, l_2s \in \text{mot}, \forall l_1, l_2 \in \text{lettre}, l_1 \neq l_2 \Rightarrow$
 $(\text{supprimePrefixeCommun } (p \mid l_1 \mid l_1s) \ (p \mid l_2 \mid l_2s)) =$
 $((l_1 \mid l_1s), (l_2 \mid l_2s))$

Profil `suffixeEgaux` : $\text{mot} \rightarrow \text{mot} \rightarrow \mathbb{B}$

Sémantique : $(\text{suffixeEgaux } m_1 \ m_2)$ est vrai si et seulement si m_1 et m_2 ne diffèrent que par leur première lettre.

Q11. Implémenter les deux fonctions ci-dessus.

Q12. En déduire une implémentation de la fonction suivante :

Profil `motsSontContrepet` : $\text{mot}^2 \rightarrow \text{mot}^2 \rightarrow \mathbb{B}$

Sémantique : $(\text{motsSontContrepet } (m_1, m_2) \ (m'_1, m'_2))$ est vrai si et seulement si les couples de mots (m_1, m_2) et (m'_1, m'_2) forment deux contrepèteries.

Exemple : $(\text{motsSontContrepet } (\text{min}, \text{seche}) \ (\text{sin}, \text{meche})) = \text{vrai}$

Pour établir si deux phrases sont des contrepèteries, on spécifie la fonction suivante :

Profil `phrasesSontContrepet` : $\text{phrase} \rightarrow \text{phrase} \rightarrow \mathbb{B}$

Sémantique : $(\text{phrasesSontContrepet } \text{phr}_1 \ \text{phr}_2)$ est vrai si et seulement si phr_1 et phr_2 sont contrepèterie l'une de l'autre.

Exemple : $(\text{phrasesSontContrepet } [\text{quelle}; \text{min}; \text{seche}]$
 $[\text{quelle}; \text{sin}; \text{meche}]) = \text{vrai}$

Q13. Implémenter `phrasesSontContrepet` en se basant sur les opérations suivantes :

- suppression des préfixes et suffixes communs entre les deux phrases,
- vérification que les mots aux extrémités des phrases sont des contrepèteries,
- vérification que les autres mots (ceux hors extrémités) sont égaux.

5.4 Générateur de contrepets

Résoudre une contrepèterie consiste à générer l'ensemble des phrases contrepèteries d'une phrase donnée.

Dans un premier temps, il faut pouvoir décomposer un mot en préfixes et suffixes, séparés par une lettre :

Profil `decompose` : $\text{mot} \rightarrow \text{ensemble}_{\text{mot} \times \text{lettre} \times \text{mot}}$

Sémantique : $(\text{decompose } m)$ est l'ensemble des décompositions (p, l, s) telles que $m = \boxed{p \mid l \mid s}$, où p (resp. s) est un mot – le préfixe (resp. suffixe) – et l une lettre.

Exemple : $(\text{decompose } \boxed{\text{s i n}}) = [(\boxed{}, 's', \boxed{\text{i n}}) ; (\boxed{\text{s}}, 'i', \boxed{\text{n}}) ; (\boxed{\text{s i}}, 'n', \boxed{})]$

Q14. Implémenter `decompose`.

Indication. Soit m un mot non vide. m peut donc être écrit sous la forme $\boxed{x \mid xs}$. Pour obtenir les décompositions de m , on peut procéder en deux étapes :

- 1) calcul de l'ensemble `decomp_xs` des décompositions de xs ;
- 2) pour chaque élément (p, l, s) de `decomp_xs`, on obtient une nouvelle décomposition de m en ajoutant x en tête de p dans (p, l, s) .

La deuxième étape nécessite soit de définir (spécification + réalisation) une fonction intermédiaire, soit d'utiliser la fonction d'ordre supérieur `List.map`.

Dans un deuxième temps, il est nécessaire d'échanger des lettres :

Profil `echange` : $\text{mot} \times \text{lettre} \times \text{mot} \rightarrow \text{mot} \times \text{lettre} \times \text{mot} \rightarrow \text{mot}^2$

Sémantique : $(\text{echange } (p_1, l_1, s_1) (p_2, l_2, s_2))$ est le couple de mots $(\boxed{p_1 \mid l_2 \mid s_1}, \boxed{p_2 \mid l_1 \mid s_2})$.

Q15. Implémenter `echange`.

Q16. Dédurre des questions précédentes une implémentation de :

Profil `contrepeteries` : $\text{dictionnaire} \rightarrow \text{phrase} \rightarrow \text{ensemble}_{\text{phrase}}$

Sémantique : Étant donné un dictionnaire `dic`, $(\text{contrepeteries } \text{dic } \text{phr})$ est l'ensemble des contrepèteries de la phrase `phr`.

Cette fonction pourra par exemple être testée avec la phrase « il y a une rame, une mare et une tare », qui possède plusieurs contrepèteries différentes.

6 Optimisations et extensions

Si vous souhaitez aller plus loin, nous proposons quatre pistes, par ordre croissant de difficulté.

6.1 Ensembles « efficaces »

La recherche d'un élément dans une séquence triée ne nécessite pas le parcours de la totalité de la séquence : l'algorithme peut s'arrêter dès lors que la position où devrait se trouver l'élément s'il était présent est atteinte.

Implémenter les ensembles par des listes triées permet donc d'améliorer l'efficacité de certains traitements ; dans le cadre de ce projet, nous pouvons citer (non exhaustivement) : union, égalité, intersection, différence.

Q17. Proposer une version efficace des ensembles puis des multi-ensembles, ainsi que des fonctions associées.

6.2 Dictionnaires « efficaces » (*)

L'idée est la même que celle du paragraphe précédent : l'utilisation d'une structure de données plus sophistiquée permet d'éviter les parcours complets de la structure.

Q18. Proposer une implémentation plus efficace des dictionnaires et fonctions associées sous forme d'arbres.

6.3 Couple de lettres (**)

Q19. Proposer une extension permettant de traiter les contrepèteries sur des couples de lettres, comme par exemple :

- « le château du rat »,
- « la fleur du pic »,
- « le psychanaliste entra dans le flou ».

6.4 Phonétique (***)

Q20. Proposer une extension permettant de traiter les contrepèteries dans toute leur généralité comme dans l'exemple suivant : « à Bruxelles, il fait beau et chaud ».

QUEL BEAU METIER, PROFESSEUR !