Programmation fonctionnelle 2024-2025

Feuille 9 : Dictionnaires

Exercice 9.1 La première implantation à réaliser représente un dictionnaire par une fonction, de type `'k -> 'v option`, où `'k` est le type des clés et `'v` le type des valeurs.

1. Définir un type `('k,'v) dict` permettant de représenter un dictionnaire par une fonction de type `'k -> 'v option`.
2. Définir le dictionnaire vide `empty_dict`, de type `('k,'v) dict`.
3. Écrire une fonction `find` de type `'k -> ('k,'v) dict -> 'v option` telle que `find key dict` calcule la valeur à la clé `key` dans le dictionnaire `dict` si cette clé existe, et `None` sinon.
4. Écrire une fonction `add_or_update` de type `'k -> 'v -> ('k,'v) dict -> ('k,'v) dict`, telle que `add_or_update key val dict` renvoie le dictionnaire `dict` dans lequel on a associé la clé `key` à la valeur `val`. Si la clé `key` n'était pas dans le dictionnaire, l'appel `add_or_update key val dict` ajoute cette clé. Sinon, elle la nouvelle valeur associée à la clé `key` déjà présente est `val` dans le dictionnaire renvoyé. Autrement dit, on considère que l'adjonction d'une donnée à une clé déjà présente met à jour l'ancienne donnée (en l'écrasant).
5. Écrire une fonction `remove` de type `'k -> ('k,'v) dict -> ('k,'v) dict` telle que `remove k dict` renvoie le dictionnaire `dict` dans lequel on a enlevé la l'entrée correspondant à la clé `key`, si cette clé est présente dans `dict` (et renvoie `dict` s'il ne contient pas la clé `key`).

Exercice 9.2 La seconde implantation demandée représente un dictionnaire comme une liste de couples (clé, valeur). Dans cette version, lorsqu'on veut mettre à jour une valeur associée à une clé déjà présente dans le dictionnaire, on la masquera simplement par un nouvel ajout. Par exemple, la liste suivante représente un dictionnaire à 3 entrées :

```
[(1,"Lundi"); (2,"Mardi"); (3,"Jeudi")]
```

Si on veut remplacer la valeur associée à la clé `3` par `"Mercredi"`, on obtiendra la liste suivante :

```
[(3,"Mercredi"); (1,"Lundi"); (2,"Mardi"); (3,"Jeudi")]
```

L'entrée `(3, "Jeudi")` de cette liste est masquée par l'entrée `(3, "Mercredi")` qui arrive plus tôt dans la liste (parce qu'elle a été insérée plus récemment). La valeur associée à l'entrée 3 est donc `"Mercredi"`.

Reprendre les questions 1 à 5 de l'exercice précédent avec cette nouvelle structure de données.

Vous devez d'abord redéfinir le type `dict`. Une fois ce type défini, la constante `empty_dict` et les fonctions doivent avoir le **même type** que dans l'exercice précédent. Vous avez le choix, pour la question 1 (définition du type) :

- soit de faire porter aux secondes composantes des couples des valeurs de type `'v`,
- soit de leur faire porter des valeurs de type `'v option`.

Dans le premier cas, la fonction `remove` ne peut pas être implantée, on la définira ainsi :

```
let remove key dict = raise (Failure "Unimplemented")
```

Cette écriture signifie que la fonction provoque une exception (les exceptions seront abordées ultérieurement). Dans le second cas (utilisation du type `'v option`), pour simuler la suppression de l'élément ayant la clé `key`, on il suffira d'insérer `(key, None)`.

Exercice 9.3 Reprendre les questions 1 à 5 de l'exercice 1 en utilisant comme structure de données des listes de couples `(clé,valeur)` **sans répétition de clé** : une clé se trouvera 0 ou 1 fois dans le dictionnaire.

Ainsi, la suppression dans le dictionnaire `[(1,"Lundi"); (2,"Mardi"); (3,"Jeudi")]` de la clé `3` renverra le dictionnaire `[(1,"Lundi"); (2,"Mardi")]`.

Ensuite, l'ajout de `(3, "Mercredi")` au dictionnaire `[(1,"Lundi"); (2,"Mardi")]` renverra le dictionnaire `[(3,"Mercredi"); (1,"Lundi"); (2,"Mardi")]`.

Exercice 9.4 On a vu dans l'exercice 1 qu'on peut utiliser des fonctions pour représenter des dictionnaires. Dans cet exercice, on utilise des fonctions pour représenter des listes.

1. Par quelle fonction pourrait-on représenter la liste `["Hello"; "World!"]` ? La liste `[0;2;4;6;8]` ?
2. En utilisant le type `'a option`, proposer un type `'a fun_list` pour représenter une liste d'éléments par une fonction.
3. Quelle est la condition sur une fonction du type écrit pour qu'elle représente effectivement une liste finie d'éléments ? La représentation devra permettre de calculer la longueur de la liste représentée par une fonction. Dans les questions suivantes, sauf dans les deux dernières, on suppose cette condition réalisée.
4. Écrire une fonction `fun_nil : 'a fun_list` qui représente la liste vide.
5. Écrire une fonction `fun_is_nil : 'a fun_list -> bool` qui teste si son argument représente la liste vide.
6. Écrire une fonction `fun_nth : int -> 'a fun_list -> 'a option` telle que `fun_nth n l` renvoie le n-ième élément de la liste `l`.
7. Écrire une fonction `fun_cons : 'a -> 'a fun_list -> 'a fun_list` telle que `fun_cons x l` renvoie la liste construite en ajoutant l'élément `x` en tête de la liste `l`.
8. Écrire une fonction `fun_tail : 'a fun_list -> 'a fun_list` telle quel `fun_tail l` renvoie la liste représentée par `l` privée de son 1er élément.
9. Écrire une fonction `fun_length : 'a fun_list -> int` qui calcule la longueur de la liste représentée par son premier argument.
10. Écrire une fonction `fun_map : ('a -> 'b) -> 'a fun_list -> 'b fun_list` telle que `fun_map f l` renvoie la liste dont les éléments sont ceux de `l` auxquel on a appliqué la fonction `f`.
11. Écrire une fonction `list_of_fun_list : 'a fun_list -> 'a list` telle que `list_of_fun_list l` calcule la liste OCaml représentée par la liste fonctionnelle `l`.
12. Écrire une fonction `fun_list_of_list : 'a list -> 'a fun_list` réalisant la conversion inverse.
13. Le type que vous avez défini permet aussi de représenter des listes infinies. Écrire la représentation de la liste infinie des carrés d'entiers.
14. Écrire une fonction `truncate : 'a fun_list -> int -> 'a fun_list` telle que `truncate l n` renvoie la liste `l` dans laquelle les éléments aux positions supérieures ou égales à `n` ont été suprimés.

Exercice 9.5 1. Évaluer la complexité de recherche d'un élément dans un dictionnaire implanté avec la structure de données de l'exercice 4.3.

Pour réduire le temps de recherche d'un élément, on organise les dictionnaires dans des arbres. Un tel arbre aura le type

```
type ('k,'v) abr = Empty | Node of ('k,'v) abr * 'k * 'v * ('k,'v) abr
```

Pour accélérer la recherche dans un tel arbre, on range les entrées (clé,valeur) en respectant la règle suivante : en tout nœud interne ayant une clé k, les nœuds internes du sous-arbre gauche ont une clé strictement inférieure à k, et les nœuds internes du sous-arbre droit ont une clé strictement supérieure à k. Toutes les opérations produisant des arbres devront respecter cette règle, et l'opération de recherche d'un élément pourra l'utiliser.

Cela suppose donc qu'on dispose d'un ordre total sur l'ensemble des clés. Comme on ne sait pas à l'avance quel sera le type des clés, cet ordre total sera représenté par une fonction de comparaison de clés, qui retournera une valeur indiquant si la première clé est inférieure, égale ou supérieure à la seconde. On définit donc le type

```
type comparison = Lt | Eq | Gt
```

où `Lt` signifie inférieur (lower than), `Eq` signifie égal, et `Gt` signifie plus grand que (greater than).

2. Définir un type `type ('k,'v) dict` utilisant le type type `('k,'v) abr`.
3. Définir une **fonction** `empty_dict: ('k -> 'k -> comparison) -> ('k,'v) dict` qui construit un dictionnaire vide incorporant une fonction de comparaison donnée en premier argument.
4. Écrire une fonction `find : 'k -> ('k,'v) dict -> 'v option` telle que `find key dict` renvoie `None` si la clé `key` n'est pas trouvée dans le dictionnaire, et la valeur appariée à la clé `key` si elle est présente dans le dictionnaire.
5. Écrire une fonction `add : 'k -> 'v -> ('k, 'v) dict -> ('k, 'v) dict` telle que `add key value dict` renvoie le dictionnaire obtenu à partir de `dict` dans lequel la clé `key` a été ajoutée si elle n'était pas présente, et dans lequel sa valeur associée est `value` (qu'elle soit présente ou non dans `dict`).
6. ($\star$) Écrire une fonction `remove : 'k -> ('k,'v) dict -> ('k,'v) dict` telle que `remove key dict` renvoie le dictionnaire `dict` dans lequel la clé `key` a été supprimée. Si la clé n'existe pas, on terminera la fonction en lançant une exception `failwith "Should not happen"`. Cette question est plus difficile : réfléchir aux différents cas de figure, selon que les fils du nœud à supprimer sont 2 feuilles, une feuille et un nœud interne, ou deux nœuds internes (cas le plus compliqué).