Exercices de révision

Jean-Pierre Becirspahic
Lycée Louis-Le-Grand

Exercice 1

Rédiger une fonction Caml qui :

- renvoie le dernier élément d'une liste non vide (et déclenche l'exception **`Not_found`** sinon) ;

Exercice 1

Rédiger une fonction CAML qui :

- renvoie le dernier élément d'une liste non vide (et déclenche l'exception **`Not_found`** sinon) ;
- détermine si un élément est présent dans une liste ;

Exercice 1

Rédiger une fonction CAML qui :

- renvoie le dernier élément d'une liste non vide (et déclenche l'exception `Not_found` sinon) ;
- détermine si un élément est présent dans une liste ;
- concatène deux listes (sans utiliser `@`) ;

Exercice 1

Rédiger une fonction Caml qui :

- renvoie le dernier élément d'une liste non vide (et déclenche l'exception **`Not_found`** sinon) ;
- détermine si un élément est présent dans une liste ;
- concatène deux listes (sans utiliser `@`) ;
- calcule la somme des éléments d'une liste d'entiers ;

Exercice 1

Rédiger une fonction Caml qui :

- renvoie le dernier élément d'une liste non vide (et déclenche l'exception **`Not_found`** sinon) ;
- détermine si un élément est présent dans une liste ;
- concatène deux listes (sans utiliser `@`) ;
- calcule la somme des éléments d'une liste d'entiers ; Comment modifier la fonction précédente pour la rendre polymorphe ? Précisez alors le type de votre fonction.

Exercice 1

Rédiger une fonction CAML qui :

- renvoie le dernier élément d'une liste non vide (et déclenche l'exception **`Not_found`** sinon) ;
- détermine si un élément est présent dans une liste ;
- concatène deux listes (sans utiliser **`@`**) ;
- calcule la somme des éléments d'une liste d'entiers ;
- insère un élément dans une liste triée ;

Exercice 1

Rédiger une fonction CAML qui :

- renvoie le dernier élément d'une liste non vide (et déclenche l'exception **`Not_found`** sinon) ;
- détermine si un élément est présent dans une liste ;
- concatène deux listes (sans utiliser `@`) ;
- calcule la somme des éléments d'une liste d'entiers ;
- insère un élément dans une liste triée ;
- trie par insertion une liste ;

Exercice 1

Rédiger une fonction Caml qui :

- renvoie le dernier élément d'une liste non vide (et déclenche l'exception **`Not_found`** sinon) ;
- détermine si un élément est présent dans une liste ;
- concatène deux listes (sans utiliser `@`) ;
- calcule la somme des éléments d'une liste d'entiers ;
- insère un élément dans une liste triée ;
- trie par insertion une liste ;
- calcule l'image miroir d'une liste ;

Exercice 1

Rédiger une fonction Caml qui :

- renvoie le dernier élément d'une liste non vide (et déclenche l'exception **`Not_found`** sinon) ;
- détermine si un élément est présent dans une liste ;
- concatène deux listes (sans utiliser `@`) ;
- calcule la somme des éléments d'une liste d'entiers ;
- insère un élément dans une liste triée ;
- trie par insertion une liste ;
- calcule l'image miroir d'une liste ;
- calcule le *shuffle* de deux listes ℓ_1 et ℓ_2, c'est-à-dire la liste de tous les mélanges de ℓ_1 et ℓ_2 préservant l'ordre relatif des éléments de chacune de ces deux listes. Par exemple,

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

Exercice 2

Dans cet exercice on considère des ensembles d'entiers représentés par des listes triées (sans répétition), comme par exemple :

```
# let l1 = [1; 3; 5; 7; 9] ;;
l1 : int list = [1; 3; 5; 7; 9]

# let l2 = [2; 3; 6; 8; 9] ;;
l2 : int list = [2; 3; 6; 8; 9]
```

Rédiger des fonctions calculant :

- l'intersection de deux ensembles ;

```
# intersection l1 l2 ;;
- : int list = [3; 9]
```

Exercice 2

Dans cet exercice on considère des ensembles d'entiers représentés par des listes triées (sans répétition), comme par exemple :

```
# let l1 = [1; 3; 5; 7; 9] ;;
l1 : int list = [1; 3; 5; 7; 9]

# let l2 = [2; 3; 6; 8; 9] ;;
l2 : int list = [2; 3; 6; 8; 9]
```

Rédiger des fonctions calculant :

- l'intersection de deux ensembles ;
- la différence symétrique de deux ensembles ;

```
# difference l1 l2 ;;
- : int list = [1; 2; 5; 6; 7; 8]
```

Exercice 2

Dans cet exercice on considère des ensembles d'entiers représentés par des listes triées (sans répétition), comme par exemple :

```
# let l1 = [1; 3; 5; 7; 9] ;;
l1 : int list = [1; 3; 5; 7; 9]

# let l2 = [2; 3; 6; 8; 9] ;;
l2 : int list = [2; 3; 6; 8; 9]
```

Rédiger des fonctions calculant :

- l'intersection de deux ensembles ;
- la différence symétrique de deux ensembles ;
- le produit cartésien de deux ensembles (trié par ordre lexicographique).

```
# produit l1 l2 ;;
- : (int * int) list =
 [(1, 2); (1, 3); (1, 6); (1, 8); (1, 9); (3, 2); (3, 3); (3, 6); (3, 8);
  (3, 9); (5, 2); (5, 3); (5, 6); (5, 8); (5, 9); (7, 2); (7, 3); (7, 6);
  (7, 8); (7, 9); (9, 2); (9, 3); (9, 6); (9, 8); (9, 9)]
```

Exercice 3

Donnez le type des expressions suivantes :

```
let rec f x y z = match x with
  | t::q -> y t (f q y z)
  | _    -> z ;;
```

Exercice 3

Donnez le type des expressions suivantes :

```
let rec f x y z = match x with
  | t::q -> y t (f q y z)
  | _    -> z ;;
```

```
exception exn of int ;;

let rec a b c d =
  try match b with
      | []               -> d
      | t::q when c t d -> a q c t
      | t::q             -> raise (exn t)
  with exn f -> f ;;
```

Exercice 4

Arithmétique de Peano

La définition axiomatique des entiers naturels de Peano peut être décrite de manière informelle par les axiomes suivants :

- 0 est un entier naturel ;
- tout entier naturel n possède un unique successeur noté Sn ;
- aucun entier naturel n'a pour successeur 0 ;
- deux entiers naturels ayant même successeur sont égaux ;
- tout ensemble d'entiers naturels qui contient 0 et le successeur de chacun de ses éléments est égal à $\mathbb{N}$.

Proposer un type Caml pour représenter les entiers de Peano.

Exercice 4

Arithmétique de Peano

La définition axiomatique des entiers naturels de Peano peut être décrite de manière informelle par les axiomes suivants :

- 0 est un entier naturel ;
- tout entier naturel n possède un unique successeur noté Sn ;
- aucun entier naturel n'a pour successeur 0 ;
- deux entiers naturels ayant même successeur sont égaux ;
- tout ensemble d'entiers naturels qui contient 0 et le successeur de chacun de ses éléments est égal à $\mathbb{N}$.

Proposer un type Caml pour représenter les entiers de Peano.

```
type nat = Zero | S of nat ;;
```

Exercice 4

Arithmétique de PEANO

La définition axiomatique des entiers naturels de PEANO peut être décrite de manière informelle par les axiomes suivants :

- 0 est un entier naturel ;
- tout entier naturel n possède un unique successeur noté Sn ;
- aucun entier naturel n'a pour successeur 0 ;
- deux entiers naturels ayant même successeur sont égaux ;
- tout ensemble d'entiers naturels qui contient 0 et le successeur de chacun de ses éléments est égal à $\mathbb{N}$.

Proposer un type CAML pour représenter les entiers de PEANO.

```
type nat = Zero | S of nat ;;
```

Définir les entiers **Un**, **Deux** et **Trois**.

Exercice 4

Arithmétique de Peano

La définition axiomatique des entiers naturels de Peano peut être décrite de manière informelle par les axiomes suivants :

- 0 est un entier naturel ;
- tout entier naturel n possède un unique successeur noté Sn ;
- aucun entier naturel n'a pour successeur 0 ;
- deux entiers naturels ayant même successeur sont égaux ;
- tout ensemble d'entiers naturels qui contient 0 et le successeur de chacun de ses éléments est égal à $\mathbb{N}$.

Proposer un type Caml pour représenter les entiers de Peano.

```
type nat = Zero | S of nat ;;
```

Définir les entiers **Un**, **Deux** et **Trois**.
Définir les fonctions de conversion **int_of_nat** et **nat_of_int**.

Exercice 4

Arithmétique de PEANO

```
type nat = Zero | S of nat ;;
```

Dans le système de PEANO l'addition est définie par les axiomes :

$$\forall a \in \mathbb{N}, \qquad a + 0 = a\ ;$$
$$\forall (a,b) \in \mathbb{N}^2, \qquad a + Sb = S(a+b).$$

Définir la fonction correspondante **add** de type *nat –> nat –> nat*.

Exercice 4

Arithmétique de Peano

```
type nat = Zero | S of nat ;;
```

Dans le système de Peano l'addition est définie par les axiomes :

$$\forall a \in \mathbb{N}, \qquad a + 0 = a\,;$$
$$\forall (a,b) \in \mathbb{N}^2, \qquad a + Sb = S(a+b).$$

Définir la fonction correspondante **add** de type *nat –> nat –> nat*.

Faire de même avec la multiplication, définie par :

$$\forall a \in \mathbb{N}, \qquad a \times 0 = 0\,;$$
$$\forall (a,b) \in \mathbb{N}^2, \qquad a \times Sb = (a \times b) + a.$$

Exercice 5

Dérivation formelle

On représente une expression arithmétique à l'aide du type :

```
type expr =
  | Cons of int
  | Var of char
  | Add of expr * expr
  | Mul of expr * expr ;;
```

- Comment se définit l'expression $e = 3xy + x^2$ à l'aide de ce type ?

Exercice 5

Dérivation formelle

On représente une expression arithmétique à l'aide du type :

```
type expr =
  | Cons of int
  | Var of char
  | Add of expr * expr
  | Mul of expr * expr ;;
```

- Comment se définit l'expression $e = 3xy + x^2$ à l'aide de ce type ?

```
let e = Add (Mul (Cons 3, Mul (Var 'x', Var 'y')),
             Mul (Var 'x', Var 'x')) ;;
```

Exercice 5

Dérivation formelle

On représente une expression arithmétique à l'aide du type :

```
type expr =
  | Cons of int
  | Var of char
  | Add of expr * expr
  | Mul of expr * expr ;;
```

- Comment se définit l'expression $e = 3xy + x^2$ à l'aide de ce type ?
- Définir une fonction **deriv** de type *char -> expr -> expr* qui réalise la dérivation formelle d'une expression vis-à-vis d'une variable.

Exercice 5

Dérivation formelle

On représente une expression arithmétique à l'aide du type :

```
type expr =
  | Cons of int
  | Var of char
  | Add of expr * expr
  | Mul of expr * expr ;;
```

- Comment se définit l'expression $e = 3xy + x^2$ à l'aide de ce type ?
- Définir une fonction **deriv** de type *char –> expr –> expr* qui réalise la dérivation formelle d'une expression vis-à-vis d'une variable.

```
# deriv `x` e ;;
- : expr =
 Add
  (Add
    (Mul (Cons 0, Mul (Var `x`, Var `y`)),
     Mul (Cons 3, Add (Mul (Cons 1, Var `y`), Mul (Var `x`, Cons 0)))),
   Add (Mul (Cons 1, Var `x`), Mul (Var `x`, Cons 1)))
```

Exercice 5

Dérivation formelle

On représente une expression arithmétique à l'aide du type :

```
type expr =
  | Cons of int
  | Var of char
  | Add of expr * expr
  | Mul of expr * expr ;;
```

- Comment se définit l'expression $e = 3xy + x^2$ à l'aide de ce type ?
- Définir une fonction **deriv** de type *char -> expr -> expr* qui réalise la dérivation formelle d'une expression vis-à-vis d'une variable.
- Rédiger une fonction **simplifie** de type *expr -> expr* qui réalise les simplifications suivantes au sein d'une expression :
  - $0 + e \longrightarrow e$;
  - $1 \times e \longrightarrow e$;
  - $0 \times e \longrightarrow 0$.

Exercice 5

Dérivation formelle

On représente une expression arithmétique à l'aide du type :

```
type expr =
  | Cons of int
  | Var of char
  | Add of expr * expr
  | Mul of expr * expr ;;
```

- Comment se définit l'expression $e = 3xy + x^2$ à l'aide de ce type ?
- Définir une fonction **deriv** de type *char –> expr –> expr* qui réalise la dérivation formelle d'une expression vis-à-vis d'une variable.
- Rédiger une fonction **simplifie** de type *expr –> expr* qui réalise les simplifications suivantes au sein d'une expression :
  - $0 + e \longrightarrow e$;
  - $1 \times e \longrightarrow e$;
  - $0 \times e \longrightarrow 0$.

```
# simplifie (deriv `x` e) ;;
- : expr = Add (Mul (Cons 3, Var `y`), Add (Var `x`, Var `x`))
```

Exercice 6

Recherche d'un élément majoritaire

On considère un tableau à n éléments représenté par le type *'a vect*, et on cherche à savoir s'il existe un élément majoritaire, c'est-à-dire dont le nombre d'occurrences dépasse strictement $n/2$.

Exercice 6

Recherche d'un élément majoritaire

On considère un tableau à n éléments représenté par le type *'a vect*, et on cherche à savoir s'il existe un élément majoritaire, c'est-à-dire dont le nombre d'occurrences dépasse strictement $n/2$.

- Rédiger une fonction **`occurrences x t i j`** qui retourne le nombre d'occurrences d'un élément x dans le tableau t entre les indices i et j (inclus).

Exercice 6

Recherche d'un élément majoritaire

On considère un tableau à n éléments représenté par le type *'a vect*, et on cherche à savoir s'il existe un élément majoritaire, c'est-à-dire dont le nombre d'occurrences dépasse strictement $n/2$.

- Rédiger une fonction **`occurrences x t i j`** qui retourne le nombre d'occurrences d'un élément x dans le tableau t entre les indices i et j (inclus).
- En déduire une fonction **`majoritaire t`** qui détermine s'il existe un élément majoritaire dans t. Quel est son coût ?

Exercice 6

Recherche d'un élément majoritaire

On considère un tableau à n éléments représenté par le type *'a vect*, et on cherche à savoir s'il existe un élément majoritaire, c'est-à-dire dont le nombre d'occurrences dépasse strictement $n/2$.

- Rédiger une fonction **`occurrences x t i j`** qui retourne le nombre d'occurrences d'un élément x dans le tableau t entre les indices i et j (inclus).
- En déduire une fonction **`majoritaire t`** qui détermine s'il existe un élément majoritaire dans t. Quel est son coût ?
- Proposez une seconde version de la fonction **`majoritaire`** suivant cette fois le paradigme «diviser pour régner». Quel est son coût ?

Exercice 6

Recherche d'un élément majoritaire

On considère un tableau à n éléments représenté par le type *'a vect*, et on cherche à savoir s'il existe un élément majoritaire, c'est-à-dire dont le nombre d'occurrences dépasse strictement $n/2$.

- Rédiger une fonction **`occurrences x t i j`** qui retourne le nombre d'occurrences d'un élément x dans le tableau t entre les indices i et j (inclus).
- En déduire une fonction **`majoritaire t`** qui détermine s'il existe un élément majoritaire dans t. Quel est son coût ?
- Proposez une seconde version de la fonction **`majoritaire`** suivant cette fois le paradigme «diviser pour régner». Quel est son coût ?
- Rédiger une fonction **`pseudo_majoritaire t`** *de coût linéaire* qui retourne un élément x possédant la propriété suivante :

  il existe un entier $c_x \geqslant n/2$ tel que x apparaisse au plus c_x fois dans t et tout autre élément au plus $n - c_x$ fois.

Exercice 6

Recherche d'un élément majoritaire

On considère un tableau à n éléments représenté par le type *'a vect*, et on cherche à savoir s'il existe un élément majoritaire, c'est-à-dire dont le nombre d'occurrences dépasse strictement $n/2$.

- Rédiger une fonction **`occurrences x t i j`** qui retourne le nombre d'occurrences d'un élément x dans le tableau t entre les indices i et j (inclus).
- En déduire une fonction **`majoritaire t`** qui détermine s'il existe un élément majoritaire dans t. Quel est son coût ?
- Proposez une seconde version de la fonction **`majoritaire`** suivant cette fois le paradigme «diviser pour régner». Quel est son coût ?
- Rédiger une fonction **`pseudo_majoritaire t`** *de coût linéaire* qui retourne un élément x possédant la propriété suivante :

  il existe un entier $c_x \geqslant n/2$ tel que x apparaisse au plus c_x fois dans t et tout autre élément au plus $n - c_x$ fois.

- En déduire une fonction de coût linéaire déterminant s'il existe un élément majoritaire dans t.