Institut National Universitaire Champollion

Contrôle continu n°1

L3 Info
14 Octobre 2019

PROGRAMMATION FONCTIONNELLE CAML

Durée : 1h30 Aucun document autorisé

1. Fonctions booléennes

Écrire les fonctions booléennes suivantes par filtrage ou composition mais sans utiliser de sélection (if then else) ni les fonctions prédéfinies `not` et & de caml :

1. `non : bool -> bool` qui à un booléen b associe sa négation logique.
2. `et : bool * bool -> bool` qui vaut vrai ssi a et b sont vrais
3. `nand : bool * bool -> bool` qui vaut vrai ssi $et(a,b)$ est faux.

2. Fonctions sur les nombres réels et entiers

Écrire les fonctions suivantes. Pour chacune d'elle on utilisera obligatoirement au moins une définition locale.

1. `unité : float->int` qui à un réel x associe le chiffre de ses unités.

```
unité(56.78) ; ;                    unité(-57.14) ; ;
-  : int = 6                        -  : int = 7
```

2. `centième : float->int` qui à un réel x associe son chiffre des centièmes.

```
centieme(56.78) ; ;                    centieme(-57.14) ; ;
-  : int = 8                           -  : int = 4
```

3. `echange : float*int*int->flot` qui à un réel x et deux entiers n et p associe le réel de même partie décimale que x et dont la partie entière est le maximum de n et p. On définira localement la fonction de calcul du maximum.

```
echange(23.432,8,14) ; ;
-  : float = 14.432
echange(23.432,-8,-14) ; ;
-  : float = -8.432
echange(-23.432,8,14) ; ;
-  : float = 14.432
echange(-23.432,-8,-14) ; ;
-  : float = -8.432
```

3. Quelques fonctions sur les fractions

Une fraction $\frac{n}{d}$ de numérateur n et de dénominateur non nul d peut être représentée en caml par un couple d'entiers (n,d). Les fractions $\frac{-2}{3}$ et $\frac{1}{4}$ peuvent ainsi être définies par :

```
let f1=(-2,3) and f2=(1,4) ; ;
```

Le but de cette partie est d'écrire quelques fonctions manipulant les fractions.

1. Écrire une fonction `fraction : int * int -> string`, définie par un filtrage exhaustif qui à une fraction (n,d) associe respectivement les chaînes de caractères "zéro" si la fraction est nulle, "fraction entière" si le dénominateur d vaut 1, "fraction unaire" si le numérateur n vaut 1, (pour la fraction $\frac{1}{1}$ qui est entière et unaire, on privilégie "fraction entière"), qui exclut les fractions de dénominateur nul (par un $failwith$ ou en levant l'exception $denominateurNul$), et qui pour toutes les autres fractions, associe "fraction positive" si le numérateur et le dénominateur sont de même signe et "fraction négative" sinon.

```
fraction(f1) ; ;
- : string = "fraction négative"
```

```
fraction(f2) ; ;
- : string = "fraction unaire"
```

2. Écrire une fonction `produit : (int * int) * (int * int) -> int * int` qui calcule le produit

$$\frac{n_1}{d_1}.\frac{n_2}{d_2} = \frac{n_1.n_2}{d_1.d_2}.$$

On ne cherchera pas à simplifier la fraction résultat pour le moment.

```
produit(f1,f2) ; ;
- : int * int = -2, 12
```

3. Écrire une fonction `somme : (int * int) * (int * int) -> int * int` qui calcule la somme de deux fractions. (Doit-on vous rappeler que

$$\frac{n_1}{d_1} + \frac{n_2}{d_2} = \frac{n_1.d_2 + n_2.d_1}{d_1.d_2} ?$$

Je pense que non...)

```
somme(f1,f2) ; ;
- : int * int = -5, 12
```

On peut simplifier l'écriture d'une fraction en divisant le numérateur et le dénominateur par leur plus grand diviseur commun $pgcd(n,d)$. On peut calculer le pgcd de deux entier **positifs** par l'algorithme d'Euclide. L'idée (récursive) de cet algorithme est que a et b ont même pgcd que b et le reste de la division de a par b. Enfin n et 0 ont pour pgcd n.

4. Écrire une fonction **récursive** `pgcd : (int * int) -> int` qui calcule le pgcd de deux entiers, et tracer l'appel de pgcd(24,18).

```
pgcd <-- 24, 18
pgcd <-- 18, 6
pgcd <-- 6, 0
pgcd --> 6
pgcd --> 6
pgcd --> 6
- : int = 6
```

5. Écrire une fonction `simplifie : (int * int) -> int` qui simplifie et normalise l'écriture d'une fraction en simplifiant le numérateur et le dénominateur par leur pgcd et en imposant au dénominateur d'être positif.

```
simplifie(24,-18) ; ;
- : int * int = -4, 3
```

6. Écrire une fonction **récursive** `harmo : int -> int` donnant sous forme d'une fraction simplifiée le n ième terme de la série harmonique :

$$h(n) = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}.$$

```
harmo(6) ; ;
#harmo <-- 6
harmo <-- 5
harmo <-- 4
harmo <-- 3
harmo <-- 2
harmo <-- 1
harmo --> 1, 1
harmo --> 3, 2
harmo --> 11, 6
harmo --> 25, 12
harmo --> 137, 60
harmo --> 49, 20
- : int * int = 49, 20
```