DEUG MIAS 24

Types et Structures de données

TD 10

1.Les rationnels

Nous voulons créer un module implantant une représentation des rationnels et des fonctions pour les manipuler.

1.1.Le type des rationnels

Définir un type nommé **t** correspondant à un enregistrement d'étiquettes `num` et `den` de type `int`.

1.2.Écriture d'un rationnel

Définir la fonction `to_string` telle que `(to_string r)` retourne la représentation décimale du numérateur de `r`, suivie d'une barre de fraction puis de la représentation décimale du dénominateur de `r`, représentation du dénominateur parenthésée si le dénominateur est négatif.

1.3.Plus grand diviseur commun de 2 entiers

Écrire une définition de la fonction `pgdc` telle que `(pgdc n1 n2)` retourne le plus grand diviseur commun des entiers `n1` et `n2`.
Indication : le pgdc de deux entiers n et m est défini récursivement par

- pgdc n 0 = n,
- pgdc n m = pgdc m (n mod m) si m est différent de 0.

Tester votre définition.

1.4.Représentation normalisée

Pour qu'un rationnel **r** n'aie qu'une seule représentation, on le représente sous sa forme irréductible avec un dénominateur positif.
Écrire une définition de la fonction `norm` telle que `(norm r)` retourne la représentation normalisée du rationnel `r` ou lève l'exception `Failure "norm"` si le dénominateur de `r` est nul.

1.5.Opérations sur les rationnels

Écrire les quatre fonctions `add`, `sub`, `mul`, `div` implantant les opérations sur les rationnels et retournant le résultat sous forme normalisée.

1.6.Égalité de deux rationnels

Écrire une définition du prédicat `equal` tel que `(equal r1 r2)` retourne `true` si et seulement si les rationnels `r1` et `r2` sont égaux.

2.Un module simple : les rationnels (TME)

Dans un fichier de nom ratio.ml, recopier toutes les définitions de la question 1.
Ne pas inclure les phrases d'essai du top-level, supprimer les doubles point virgule en fin de phrase.
Sauvegarder le fichier ratio.ml.

2.1.Compilation du module

Dans un terminal, taper la commande permettant de créer un code objet pour ce module (sans édition de liens) :

```
ocamlc -c ratio.ml
```

s'il n'y a pas d'erreur, la commande suivante :

```
ls ratio.*
```

doit afficher en retour :

```
ratio.cmi        ratio.cmo        ratio.ml
```

où

- ratio.ml est le source du module,
- ratio.cmi, l'interface compilée du module
- ratio.cmo, le code objet du module.

2.2.Interface du module

Vous pouvez obtenir l'impression de toutes les déclarations du module par la commande

```
ocamlc -c -i ratio.ml
```

et leur sauvegarde dans un fichier d'interface par la commande

```
ocamlc -c -i ratio.ml > ration.mli
```

Il vous reste à insérer, dans ce dernier fichier les spécifications des définitions.

3.Utilisation du module

3.1.Le problème

On considère deux évèments: **A** l'alarme se déclenche; **I** un incident se produit.
On recueille, sous forme de rationnels, les données suivantes :

- p_1 probabilité que l'alarme se déclenche sans incident, $P(\mathbf{A} \cap \sim\mathbf{I})$: $p_1 = 1/50$.
- p_2 probabilité qu'un incident se produise sans alarme, $P(\sim\mathbf{A} \cap \mathbf{I})$: $p_2 = 1/500$
- p_3 probabilité qu'un incident se produise, $P(\mathbf{I})$: $p_3 = 1/100$.

Calculer la probabilité qu'un incident se soit produit alors que l'alarme s'est déclenchée (probabilité de "**I** sachant **A**"),

$$P(\mathbf{I}/\mathbf{A}) = P(\mathbf{A} \cap \mathbf{I}) / P(\mathbf{A})$$

avec

$$P(\mathbf{A} \cap \mathbf{I}) = P(\mathbf{I}) - P(\sim\mathbf{A} \cap \mathbf{I}) \text{ et } P(\mathbf{A}) = P(\mathbf{A} \cap \mathbf{I}) + P(\mathbf{A} \cap \sim\mathbf{I})$$

3.2.Le calcul (à reprendre en TME)

Écrire des phrases (déclarations ou expressions) permettant de résoudre le problème 3.1.