

<p>info401 : Programmation fonctionnelle TP 4 : expressions arithmétiques</p>

Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 22, poste : 94 22
email : Pierre.Hyvernat@univ-savoie.fr
www : <http://www.lama.univ-savoie.fr/~hyvernat/>

Ce TP, comme les suivants, sera noté. Je ne demande aucun compte rendu à part, mais seulement un *unique* fichier Caml, commenté.

- Votre fichier doit contenir du code valide et ne doit pas provoquer d'erreur lorsqu'on l'évalue avec l'interprète Caml. (Testez avant de me l'envoyer : si votre fichier ne s'évalue pas correctement, vous perdez automatiquement 5 points sur votre note finale.)
- Votre fichier devra contenir un commentaire contenant votre nom, prénom et filière. (Idem, si ce n'est pas le cas, vous perdez 5 points sur votre note finale.)

Partie 1 : exceptions (45min)

Question 1. En utilisant `List.fold_left`, programmez une fonction `somme : int list -> int` et une fonction `produit : int list -> int` qui calculent respectivement la somme et le produit des éléments d'une liste.

Question 2. Idem, mais pour une fonction `produit7` qui calcule le produit dans $\mathbf{Z}/7\mathbf{Z}$ (modulo 7).

Question 3. Testez `produit7` sur des listes aléatoires d'éléments de $\mathbf{Z}/7\mathbf{Z}$ (en utilisant `Random.int 7` pour générer un tel élément) de grande taille. Que constatez-vous ? Expliquez.

Question 4. Réécrivez `produit7-bis` avec `List.fold_left`, mais en utilisant une exception pour améliorer le calcul.

Partie 2 : expressions arithmétiques (2h)

Le but de cet exercice est d'écrire un petit ensemble de fonctions pour manipuler les expressions arithmétiques telles que $3 + (4 \times (2 + 3))$ ou $(a + b) \times (a + b) - 2 \times a \times b$.

Question 1. Donnez un type de données `exp` pour les expressions arithmétiques. Une expression arithmétique est construite à partir :

- de valeurs entières (0, 1, ...)
- de variables numérotées (X_1, X_2, \dots)
- des opérations arithmétiques habituelles (+, - et \times).

Question 2. La fonction `print_int : int -> unit` permet d'afficher un entier à l'écran. En utilisant cette fonction ainsi que la fonction `print_char` et l'opérateur ";" pour séquentialiser les commandes, écrivez une fonction `print_expr : expr -> unit`.

Par exemple, votre fonction pourra afficher la chaîne `((7) * (X1)) + ((0) * (X2))` pour l'expression $(7 \times X_1) + (0 \times X_2)$. (Version simple.)

Si vous avez le temps de faire une fonction un peu plus sophistiquée, n'hésitez pas...

Question 3. Écrivez une fonction `calcule : expr -> int` qui calcule la valeur entière d'une expression arithmétique sans variables.

Modifiez cette fonction pour qu'elle prenne un argument supplémentaire `env` censé contenir les valeurs des variables dans une liste.

Pouvez-vous facilement transformer votre fonction en fonction récursive terminale ?

Question 4. La notation polonaise préfixée fonctionne de la manière suivante : le symbole arithmétique (binaire) vient avant ces deux arguments. Par exemple, au lieu de noter

$$3 * ((12 - 17) / x),$$

on notera

$$* 3 / - 12 17 x .$$

Écrivez une fonction `transforme : string -> expr` qui transformera une chaîne de caractère en expression arithmétique.

Les fonctions suivantes peuvent vous être utiles :

- `int_of_string : string -> int` qui transforme une chaîne de caractères en entier.
- `String.sub : string -> int -> int -> string` qui permet de récupérer une sous chaîne,
- `Str.split : Str.regexp -> string -> string list` qui permet de séparer une chaîne en morceaux, en utilisant une expression régulière.

Pour utiliser `Str.split`, il faut soit lancer Caml avec `ocaml str.cma`, soit ajouter

```
#load "str.cma";;
```

Si vous voulez découper une chaîne selon les espaces, vous pouvez utiliser :

```
# let split s = Str.split (Str.regexp_string " ") s;;  
val split : string -> string list = <fun>  
# split "Salut, je m'appelle Bob !";;  
- : string list = ["Salut,","je"; "m'appelle"; "Bob"; "!"]
```

Pour tester, la fonction `read_line` qui lit une chaîne de caractères sur l'entrée standard pourra vous servir...

Partie 3 : comparaison d'expression arithmétiques (...)

Question 1. Comment pouvez tester l'équivalence de deux expressions arithmétiques sans variable ? Écrivez la fonction correspondante `equiv_sans_var : expr -> expr -> bool`.

À votre avis, comment peut-on tester l'équivalence de deux expressions arithmétiques avec des variables. (Pas la peine de le faire, donnez simplement des idées.)

Question 2. Un polynôme à coefficients entiers et à plusieurs variables peut être représenté par le type `type poly = (int * (int list)) list` :

- on représente un polynôme par la liste de ces monômes,
- on représente un monôme par son coefficient, la liste des variables avec leur multiplicité.

Par exemple, le polynôme $X_1^2 + 2X_1X_2 + X_2^2$ est représenté par

$$[(1, [1; 1]); (2, [1; 2]); (1, [2; 2])].$$

Écrivez une fonction `normalize : expr -> poly` qui développe une expression arithmétique pour la transformer en polynôme correspondant.

Question 3. Essayez de faire en sorte que deux expressions arithmétiques équivalentes (comme $(X_1 + X_2) \times (X_1 + X_2)$ et $X_1 \times X_1 + 2 \times X_1 \times X_2 + X_2 \times X_2$) soient normalisées en polynômes égaux.

Documentez vos découvertes et améliorations.