Programmation Fonctionnelle, L3 Informatique, 2022-2023

Projet - Simplification d'expressions arithmétiques

Le but de ce projet est d'écrire un simplificateur (très naïf, le problème de la simplification étant très complexe) d'expressions arithmétiques entières. Pour pouvoir disposer confortablement d'expressions arithmétiques suffisamment complexes, on utilisera l'une des méthodes les plus simple pour les représenter, la notation polonaise inversée, dans laquelle, on représente les opérandes d'un opérateur avant l'opérateur lui-même. L'intérêt de ce langage est qu'il ne nécessite pas l'introduction des priorités relatives entre les opérateurs, et que l'on peut exprimer toute expression arithmétiques en utilisant aucune parenthèse. L'inconvénient est que chaque opérateur doit avoir une arité bien déterminée (pas question d'utiliser le signe moins pour la soustraction et pour l'opposé). Voici une expression dans ce langage : $13\ 2\ 5 \times 1\ 0/ - +$ qui représente l'expression habituelle $13 + (2 \times 5 - 1/0)$. Comme dit précédemment, si l'on veut un opérateur de soustraction de deux nombres et un opérateur d'opposés alors il faut les représenter par des signes distincts. Dans notre cas, on supposera que les opérateurs possibles sont :

- $\times, +, -, /$, tous les quatre d'arité 2,
- $\tilde{}$ d'arité 1, représentant le moins unaire,
- les constantes sont les entiers positifs,
- les variables sont les lettres minuscules.

Question 1 (Analyse)
Pour simplifier l'analyse lexicale, qui n'est pas l'objet premier de ce projet, l'expression arithmétique sera saisie sous la forme d'une listes de chaînes de caractères, où chaque chaîne représente un élément (opérateur, constante ou variable). Ainsi par exemple, l'opérateur $\times$ sera représenté par la chaîne `"*"`, la constante 26 par la chaîne `"26"` et la variable x par la chaîne `"x"`.

Écrire un type `tree` qui permet de représenter les arbres de syntaxe abstraite des expressions arithmétiques telles que décrites ci-dessus.

Écrire une fonction `parse : string list -> tree` qui construit l'arbre de syntaxe abstraite d'une expression arithmétique en notation polonaise. Cette fonction lèvera une exception lorsque l'une des chaînes de caractères de la liste n'est pas un élément d'une expression, et une autre exception lorsque la liste d'éléments est incohérente (par manque ou excès d'opérandes).

Question 2 (Simplification des expressions)
La simplification d'une expression à partir de son arbre de syntaxe abstraite se fait en appliquant les règles qui sont présentées ci-dessous (plus éventuellement d'autres si vous avez du courage à revendre). Ces règles correspondent à des simplifications élémentaires ; dans ce qui suit, x désigne une sous expression quelconque :

- une sous-expression sans variable sera évaluée ;
- une sous-expression de la forme $1 \times x$ ou $0 + x$ (respectivement $x \times 1$ ou $x + 0$) sera simplifiée en x ; de même l'expression $0 \times x$ (respectivement $x \times 0$) sera simplifiée en 0 ;
- une sous-expression de la forme $x - x$ sera simplifiée en 0 ;
- de manière analogue, une sous-expression de la forme x/x sera simplifiée en 1 (bien que ce soit un peu abusif).

Attention, on ne demande pas de simplifier une expression de la forme $x + y - x$ en y, car cela fait intervenir la commutativité de l'addition, et peu conduire facilement à des programmes qui bouclent... disons qu'il faut faire un peu de théorie avant de s'attaquer à ces choses là.

Question 3 (Affichage du résultat)
Une fois les simplifications terminées, il faudra afficher l'expression sous la forme simplifiée. Cet affichage devra éliminer le plus possible de parenthèses inutiles, et au moins les parenthèses associatives, ce qui signifie par exemple, qu'une expression comme $((a \times b) \times c) \times (e + f)$ sera affichée sous la forme `a * b * c * (c + f)`. On n'exige pas que soient supprimées toutes les parenthèses rendues inutiles par la priorité relative des opérateurs (par exemple afficher $a + (b \times c)$ comme `a + b * c`, mais si le coeur vous en dit...).

Travail à rendre

Le projet est à réaliser par groupes de 3. Les noms des auteurs doivent être clairement indiqués en entête du fichier rendu.

Il n'est pas demandé de rapport de projet, mais chaque type et fonction codés doivent être dûment documentés.

Des tests unitaires et fonctionnels seront fournis dans un fichier séparé.