UN ÉVALUATEUR D'EXPRESSIONS ARITHMÉTIQUES

On considère dans cette séance les formules syntaxiquement correctes que l'on construit habituellement avec les entiers et les trois opérations $+, -, \times$. Par exemple, $2+3$, $(1+2)\times 5$, $((1+3-5))$.

Plus formellement, on définit par induction structurelle les formules de la façon suivante :

- un entier naturel est une formule
- si u, v sont deux formules, alors $-u$, $u+v$, $u-v$, $u\times v$, (u) sont des formules.

La difficulté dans la signification de ces formules est de trois ordres.

Le premier est l'utilisation facultative des parenthèses : $2+3$ et $((2+((3))))$ sont deux formules différentes mais qui signifient la même chose (on dit qu'elles sont syntaxiquement différentes, mais sémantiquement équivalentes).

Le second est la priorité de la multiplication par rapport aux deux autres opérations.

Le troisième est l'associativité à gauche de la soustraction : $2-5-9$ doit être compris comme $(2-5)-9$ et non pas comme $2-(5-9)$.

Nous allons construire un évaluateur élémentaire qui calcule les valeurs associées aux formules.

1 Parenthésage correct

Nous représentons les formules par des chaînes de caractères (type `string`) : `"(12+3)-(5*6)"`.

La première étape de notre évaluateur est de vérifier que les parenthèses (s'il y en a) sont correctement disposées : `(1+3` est une formule mal parenthésée, et de supprimer les parenthèses extérieures inutiles : `"((2+(3+1)))"` est transformée en `"2+(3+1)"`.

Pour cela, on utilise l'algorithme suivant sur une chaîne sensée représenter une formule :

- on initialise un compteur à zéro ;
- on parcourt de droite à gauche (on verra ci-dessous que c'est mieux dans ce sens) tous les caractères de la chaîne : chaque fois qu'on rencontre une parenthèse fermante, on ajoute 1 au compteur ; si c'est une parenthèse ouvrante, on retire 1 ;
- la chaîne est correctement parenthésée si et seulement si le compteur n'est jamais strictement négatif et si sa valeur finale est 0.

Écrivez une fonction `bon_parenth` de type `string -> bool` qui évalue le bon parenthésage de la chaîne. Rappelons que le caractère d'indice i d'une chaîne notée `ch` est noté `ch.[i]` (avec des crochets) et qu'un caractère unique se note entre accents graves `'('` (symboles accessible par la touche `Alt Gr + 7`).

Écrivez ensuite une fonction récursive `nettoyer` de type `string -> string`, qui prend en paramètre une chaîne bien parenthésée et qui construit la chaîne équivalente en enlevant les parenthèses extérieures inutiles, selon que la chaîne ne commence pas ou ne finit pas par une parenthèse, ou possède des parenthèses au début et à la fin. Pour cela, on peut se rendre compte que quelque chose était indispensable quand on se plaint de ne plus l'avoir...

2 Recherche de l'opérateur de niveau minimal

Dans une formule correcte nettoyée (*i.e.* sans parenthèses extérieures inutiles), il se peut qu'il y ait zéro, un ou plusieurs opérateurs.

S'il n'y en a pas ou si c'est un $-$ en première position, la chaîne représente un entier : par exemple, `"-14"`. Dans ce cas, sa valeur est facile à donner, car la fonction CAML `int_of_string` transforme la chaîne en entier.

S'il y a un opérateur, il faut déterminer les opérateurs de niveau minimal 0 : si nous réitérons un algorithme du même type que celui sur les parenthèses, alors nous dirons qu'un caractère opérateur est de niveau i si la valeur du compteur juste après la lecture de ce caractère vaut i. S'il y a un opérateur additif ($+$ ou $-$) de niveau 0, alors la chaîne sera dite additive, sinon elle sera dite multiplicative.

Par exemple, `"(1+2)*3*5"` est multiplicative, tandis que `"1+2*3-6"` est additive. En notant en indice la valeur du compteur utilisé, on obtient :

- $({}_0\, 1_1\ +_1\ 2_1\ )_1\ *_0\ 3_0\ *_0\ 5_0$ est un chaîne multiplicative, car les seuls opérateurs de niveau 0 sont des multiplications
- $1_0\ +_0\ 2_0\ *_0\ 3_0\ -_0\ 6_0$ est additive car il existe au moins un opérateur additif de niveau 0

Remarque : on commence par la fin pour tenir compte de l'associativité à gauche du symbole $-$.

Écrivez une fonction op_min de type `string -> int * int` qui donne comme résultat

- le couple $(i, 1)$ si la chaîne est additive et l'opérateur additif de niveau 0 le plus à droite est un $+$ situé en position i ;
- le couple $(i, 2)$ si la chaîne est additive et l'opérateur additif de niveau 0 le plus à droite est un $-$ situé en position i ;
- le couple $(i, 3)$ si la chaîne est multiplicative et donc forcément l'opérateur de niveau 0 le plus à droite est un $*$ situé en position i ;
- le couple $(0, 4)$ si la chaîne représente un entier.

Remarque. On fera attention au cas du signe moins en première position : la chaîne "−15∗3" est multiplicative, car le signe moins initial n'est pas un symbole de loi de composition interne, mais le symbole d'opposé.

3 Évaluation récursive de la chaîne

Étant donnée une formule correcte, nous la calculons récursivement en la découpant en sous-formules.

D'abord, nous la nettoyons de ses parenthèses extérieures inutiles, en vérifiant au passage qu'elle est correctement parenthésée, sinon on lève une exception par failwith .

Ensuite nous tentons de couper la formule en deux sous-formules au niveau d'un opérateur. Si c'est impossible, alors on retourne la valeur numérique. Sinon, nous coupons la formule, nous évaluons les deux sous-formules et nous donnons la valeur de la formule initiale en utilisant le bon opérateur intermédiaire. Évidemment, le choix de l'opérateur n'est pas fait au hasard...

Par exemple, la formule "((1+3∗2)−6)" est nettoyée en "(1+3∗2)−6", doit être découpée en "(1+3∗2)" et "6", qui sont évaluées puis combinées par soustraction : calcul "((1+3∗2)−6)"= calcul "(1+3∗2)"− calcul "6". Puis récursivement, calcul "(1+3∗2)"= calcul "1" + calcul "3∗2", enfin calcul "3∗2" = calcul "3" ∗ calcul "2".

Écrivez cette fonction calcul , de type `string -> int`.