TP 4 – Évaluation d'un mini langage d'expressions

Dans ce TP, nous allons construire un mini langage d'expressions mathématiques et nous allons implémenter son interprétation. Vous avez normalement déjà eu des cours de théories des langages les années précédentes, nous n'allons pas revenir sur les principes de parseurs et grammaires à proprement parler. Nous allons à la place construire un Arbre de Syntaxe Abstraite (Abstract Syntax Tree ou AST) et nous allons écrire un interpréteur récursif d'AST.

Exercice 1. Première version du langage

Le langage que nous allons définir est un simple langage d'expressions mathématiques. Nous pourrions dire qu'il sagit d'un Domain Specific Language (DSL). Nous lui donnerons le nom de *Mathematique Language* (si vous avez un meilleur nom...). MatL manipule uniquement des entiers (pour l'instant) et permet d'effectuer deux types d'opérations, les opérations binaires, avec deux opérandes, et les opérations unaires (avec une seule opérandes). Une expression MatL est soit une valeur directe, soit une opération binaire, soit une opération unaire. Une opération binaire est définie par une chaine de caractère représentant l'opération et deux sous-expressions alors qu'une opération unaire est définie par une chaine de caractère et une seule sous-expression.

1. Donner le type somme correspondant aux expressions MatL.

2. Écrire l'expressions OCaml permettant de représenter le calcul `1 + 2 + - 3` en MatL.

3. Écrire une fonction `evaluate_expr` qui prend une expression MatL et qui retourne le résultat de l'évaluation. On ne considère actuellement que les opérations d'addition, de multiplication, division et soustraction.

Exercice 2. Vers une version un peu plus flexible

La fonction d'évaluation d'expression codée est actuellement peu flexible. Lorsque l'on définit un nouvel opérateur, il est nécessaire de modifier la fonction d'évaluation, ce qui est loin d'être pratique. Pour éviter ça, nous allons définir associers des fonctions à des chaines de caractères et stocker le tout dans une liste.

1. Comment faire pour conserver le nom de l'opération et la fonction associée dans une liste ? Écrire une telle liste.

2. Écrivez une fonction de recherche qui retourne la fonction associées à une opération dont le nom est passé en paramètre.

3. Modifier la fonction `evaluate_expr` pour qu'elle prenne en paramètre la liste des opérations disponible et effectue le calcul avec les fonctions précédemment définies.

4. Ajouter une opération d'exposant à MatL.

Exercice 3. Variables

Actuellement, il est impossible de définir des variables en MatL. Nous allons donc rajouter ça à un nouveau type de donnée, les "statements". Les statements sont l'équivalent de ligne de code qui sont évaluées. Dans cette catégorie, nous trouvons l'expression des expressions ainsi que la définition des variables. Il convient de faire la différence entre deux utilisations : l'accés à la variable (qui est une expression) et son assignation (qui est un statement). L'utilisation d'une variable dans une expression se fait juste en faisant référence à son nom alors que l'assignation d'une variable se fait par association d'une sous-expression à une chaine de caractères.

1. Donner le type somme correspondant aux statements MatL (pour l'instant, 2 constructeurs doivent être présents).

2. Ajouter au type expression la possibilité de faire référence à une variable.

3. Écrire l'expression associée à l'expression suivante : `a = 4 * 5 + 6`.

Lorsqu'une assignation est exécutée, il est nécessaire d'associer la chaine de caractère représentant le nom de la variable au résultat de la sous-expression. Pour ça, il est nécessaire de conserver un

environnement d'exécution qui gardera ce type d'informations.
4. Comment représenter ces informations ? Est-il possible de conserver ces informations dans une liste ?
5. Donner une fonction permettant de rechercher la valeur d'une variable en fonction de son nom. Peut-on réutiliser la fonction utilisée pour rechercher une opération ?
6. Il faut maintenant écrire une fonction d'évaluation de statements. Que doit retourner cette fonction ?

Exercice 4. Séquence d'instructions
Comme vous avez pu le remarquer il est actuellement impossible en MatL de définir des suites d'instructions. Il est donc complexe d'assigner une valeur à une variable et de l'utiliser dans une expression par la suite.
1. Donner un nouveau type somme pour représenter des séquences de statements.
2. Écrire l'expression correspondante au code suivant :

```
a = 4 + 5
b = a + 1
a + b
```

3. En considérant qu'une séquence d'instructions MatL ne donne en résultat que le résultat de la dernières expression exécutée, écrire une fonction `evaluate_matl` qui permet d'évaluer une séquence d'instructions.

Exercice 5. Nouvelles fonctions
MatL est actuellement limité à des opérations unaire ou binaire. Il serait intéressant de l'étendre pour qu'il puisse manipuler des nouvelles fonctions (par exemple, `sin` ou `cos`). Si on regarder une fonction, on se rend compte qu'il s'agit de l'association d'une fonction à un label avec une liste d'arguments.
1. Modifier le type expression pour qu'il puisse prendre en compte les fonctions.
2. Ajouter un mécanisme proche de celui pour la définition des opérations binaires.
3. Modifier le mécanisme pour la gestion des opérations binaires pour qu'il soit aligné avec le mécanisme précédemment définit.
4. Ajouter de nouvelles fonctions à MatL.