DM COMP

Aurèle Barrière & Antonin Garret

29 septembre 2016

1 Introduction

Le but de ce projet est d'implémenter un parseur d'expressions arithmétiques en `OCaml`, en utilisant les `stream parsers` de l'extension `Camlp4`. Le but est ainsi de convertir une chaîne de caractère (écrite sur un alphabet comportant des nombres et des opérateurs arithmétiques) en un arbre syntaxique abstrait donnant une représentation correcte (c'est-à-dire qui respecte les propriétés de priorité et d'associativité voulues) de l'expression.

On effectuera une analyse descendante.

Dans une première partie, nous verrons comment aboutir à une grammaire d'expressions arithmétiques prête à être implémentée. Ensuite, nous nous intéresserons à la vérification empirique de notre implémentation.

2 Une grammaire pour des expressions arithmétiques

Pour implémenter notre parseur avec analyse descendante avec les `stream parsers`, il faut définir un ensemble de fonctions mutuellement récursives qui décomposent la chaîne de caractère souhaitée. Dans un premier temps, la chaîne de caractère aura été parcourue par un *scanner* (ou `lexer`) qui repère les mot-clés (ou `tokens`), et qui convertit donc une chaîne de caractères en une chaîne de mot-clés.

Les fonctions mutuellement récursives prennent donc en entrée une chaîne de mot-clés, et renvoient un arbre syntaxique abstrait d'expression, qu'il nous faut définir. Une expression sera ainsi : un nombre ou une addition ou soustraction ou multplication ou division ou puissance de deux expressions. On aura donc en `Ocaml` :

```
type expr = Num of int | Add of expr * expr | Sub of expr * expr | Mul
    of expr * expr | Div of expr * expr | Pow of expr * expr
```

.

2.1 Une première version

Pour parser correctement des expressions arithmétiques dans lesquelles figurent l'addition, la soustraction, la multiplication et des parenthèses, en respectant les propriétés suivantes :

— L'addition, la soustraction et la multiplication sont associatives à gauche.

— Les parenthèses sont priotitaires, puis la multiplication, puis l'addition (ou soustraction).

On dispose d'une première grammaire. Les récursivités gauches témoignent de l'associativité gauche des opérateurs. On a décomposé notre expressions en plusieurs sous-expressions : (E : expressions, F : facteurs, A : atomes) en suivant l'ordre inverse de priorité. Les dépendances cycliques entre ces différentes fonctions récursives nous obligeront à les déclarer comme des fonctions mutuellement récursives.

```
E -> E + F | E - F | F
F -> F * A | A
A -> constante | ( E )
```

2.2 Ajout de la division

La division est un opérateur qui doit être associatif à gauche et avoir la même priorité que la multiplication. On ajoute donc la règle `F -> F / A`.

```
E -> E + F | E - F | F
F -> F * A | F / A | A
A -> constante | ( E )
```

2.3 Récursivités gauches

Cependant, la méthode utilisée est l'analyse descendante. On ne peut donc pas garder nos règles avec récursivité gauche. Il faut donc créer de nouvelles règles pour les transformer en récursivités droites.

```
E      -> F E_aux
E_aux -> + F E_aux | - F E_aux | ϵ
F      -> P F_aux
F_aux -> * A F_aux | / A F_aux | ϵ
A      -> constante | ( E )
```

C'est ce qu'on a de déjà implémenté.

2.4 Ajout de l'opérateur de puissance

Ajout entre les facteurs et les atomes : priorité.
Récursivité droite pour associativité droite.
On a donc

```
E      -> F E_aux
E_aux -> + F E_aux | - F E_aux | ϵ
F      -> P F_aux
F_aux -> * P F_aux | / P F_aux | ϵ
P -> A ^ P | A
A      -> constante | ( E )
```

2.5 Changement de la règle de puissance

Dû à l'implémentation : deux règles ne peuvent pas commencer par le même élément et P se réduit en A ^ P | A qui commencent tous les deux par A.

On remplace donc la règle par

P -> A P_aux
P_aux -> ^ P | ϵ

2.6 Grammaire finale

E -> F E_aux
E_aux -> + F E_aux | - F E_aux | ϵ
F -> P F_aux
F_aux -> * P F_aux | / P F_aux | ϵ
P -> A P_aux
P_aux -> ^ P | ϵ
A -> constante | (E)

3 Implémentation

Quelques mots sur stream parsers.

Préciser pourquoi on envoie e1 en argument aux fonctions auxiliaires.

4 Tests

Automatiser ?

Gestion d'erreurs ? Stream parsers reconnaissent le début d'une expression correcte et s'en satisfont. Ca peut etre le rôle d'autre chose je ne suis pas sur.

5 Conclusion