Devoir de compilation : Rapport

Waelkens Dimitri, Lempereur Martin

22 mai 2016

Département d'Informatique
Université de Mons

Consigne

Il nous était demandé dans le cadre du cours de compilation de réaliser un moteur de templating simplifié. Ce moteur doit recevoir un fichier de données ainsi que un ficher de template qui sera utilisé pour venir y insérer ces données. Ce genre de moteur de templating permet à partir d'un même modèle d'afficher plusieurs fichiers avec des données différentes sans devoir pour autant tout réécrire.

Grammaire

Nous avons d'abord implémenter la grammaire décrite dans les consignes. Ensuite nous avons insérer plusieurs règles pour gérer les opérations sur entier ainsi que certaines expressions booléennes. La grammaire résultante est décrite ci-dessous :

Rule 0 : $S' \rightarrow prog$
Rule 1 : $prog \rightarrow TXT$
Rule 2 : $prog \rightarrow dumbo\_block$
Rule 3 : $prog \rightarrow TXT\ prog$
Rule 4 : $prog \rightarrow dumbo\_block\ prog$
Rule 5 : $dumbo\_block \rightarrow BEGIN\ expression\_list\ END$
Rule 6 : $expression\_list \rightarrow\ expression\ ;$
Rule 7 : $expression\_list \rightarrow expression\ ;\ expression\_list$
Rule 8 : $expression \rightarrow PRINT\ string\_expression$
Rule 9 : $expression \rightarrow PRINT\ int\_expression$
Rule 10 : $string\_expression \rightarrow STRING$
Rule 11 : $string\_expression \rightarrow string\_expression\ .\ string\_expression$
Rule 12 : $expression \rightarrow ID\ ASSIGN\ string\_expression$
Rule 13 : $string\_expression \rightarrow ID$
Rule 14 : $string\_list \rightarrow (\ string\_list\_interior\ )$
Rule 15 : $string\_list\_interior \rightarrow STRING\ ,\ string\_list\_interior$
Rule 16 : $string\_list\_interior \rightarrow STRING$
Rule 17 : $expression \rightarrow ID\ ASSIGN\ string\_list$
Rule 18 :
$expression \rightarrow FOR\ ID\ IN\ string\_list\ DO\ expression\_list\ ENDFOR$
Rule 19 : $expression \rightarrow FOR\ ID\ IN\ ID\ DO\ expression\_list\ ENDFOR$
Rule 20 : $int\_expression \rightarrow ID$
Rule 21 : $int\_expression \rightarrow NBR$
Rule 22 : $expression \rightarrow ID\ ASSIGN\ int\_expression$
Rule 23 : $int\_expression \rightarrow int\_expression\ -\ int\_expression$
Rule 24 : $int\_expression \rightarrow int\_expression\ +\ int\_expression$
Rule 25 : $int\_expression \rightarrow int\_expression\ *\ int\_expression$
Rule 26 : $int\_expression \rightarrow int\_expression\ /\ int\_expression$
Rule 27 : $bool\_expression \rightarrow int\_expression\ >\ int\_expression$
Rule 28 : $bool\_expression \rightarrow int\_expression\ <\ int\_expression$
Rule 29 : $bool\_expression \rightarrow !\ bool\_expression$
Rule 30 : $bool\_expression \rightarrow int\_expression\ NEQ\ int\_expression$
Rule 31 : $bool\_expression \rightarrow bool\_expression\ OR\ bool\_expression$

Rule 32 : $bool\_expression \rightarrow bool\_expression\ AND\ bool\_expression$
Rule 33 : $bool\_expression \rightarrow TRUE$
Rule 34 : $bool\_expression \rightarrow FALSE$
Rule 35 : $expression \rightarrow IF\ bool\_expression\ DO\ expression\_list\ ENDIF$

Dans notre parser qui reprend cette grammaire nous avons à chaque règle trouvée, créer un tuple qui décrit l'opération en question, ce tuple sera utilisé par notre interpréteur.

Exemple :
Pour la règle 18, on crée un tuple qui contient

```
('FOR',p1,p2,p3)
```

où
p1 : variable qui va évoluer,
p2 : variable sur laquelle on va itérer,
p3 : instructions à réaliser.

For, If et explications

Explications

Nous avons créer une variable global dans notre programme qui contient des informations sur l'environnement des variables. Cet environnement peut être enrichi en connaissance comme lorsque un ID est assigné alors on ajoute l'ID et sa valeur à la variable globale, ou bien on peut retirer des information lorsqu'une variable est initialisé dans une boucle et ne peut plus être appelée ensuite.

For

Comme décrit dans la section d'avant l'interpréteur va recevoir un tuple contenant plusieurs éléments d'information sur la boucle For.

L'interpréteur va d'abord aller chercher le contenu de la variable cible.

Ensuite pour chaque élément e de cette variable, une variable accessible par les instructions de la boucle va être assignée avec la valeur de e. Une fois la valeur assignée et ajouter a la variable global environnent, on va exécuter le corps de la boucle, ainsi de suite jusqu'à ce que chaque élément e sont parcouru.

If

Le If est réalisé comme toutes les autres opérations à l'aide de la description contenue dans un tuple. L'interpréteur va évaluer la condition si celle-ci est vérifié va interpréter l'intérieur du if sinon va retourner une chaine vide.

Problèmes rencontrés

Nous avons essayer de construire petit a petit notre programme en regardant à ce que chaque partie implémentée soit fonctionnelle avant de passer à la suite. Mais dès lors que l'on veut changer la grammaire, on se rend compte que

certaines parties du lexeur par exemple ont été mal pensées mais on ne peut pas forcement déterminer de quel partie du compilateur le problème provient. Dans notre projet on a par exemple eu une regex qui était légèrement mal construite mais fonctionnait correctement avec notre lexer, ainsi que notre grammaire, lorsque l'on a voulu faire évoluer notre grammaire des erreurs sont apparues mais ne provenait pas de la nouvelle description de la grammaire.