TP NF11

Génération d'un analyseur lexical et syntaxique pour le langage Logo

Jean Vintache, Adrien Jacquet

25 mai 2016

Table des matières

utc
Université de Technologie
Compiègne

1 Introduction

Le Logo est un langage de programmation à vocation principalement éducationnelle pouvant être utilisé pour réaliser des dessins sur un tableau blanc numérique. L'objectif des différents TP suivis lors de ce semestre était d'implémenter un analyseur lexical et syntaxique permettant d'utiliser une sous-partie des instructions de ce langage.

D'un point de vue technologique, nous avons utilisé le framework libre ANTLR pour générer l'analyseur lexical et la structure principale de l'analyseur syntaxique à partir d'une grammaire.

2 Grammaire

On commence par définir les expressions régulières permettant d'identifier respectivement un nombre entier, un identifiant de fonction ou de variable et les caractères ignorés par l'analyseur lexical :

```
INT : '0' | [1-9][0-9]* ;
ID : [_a-zA-Z][_a-zA-Z0-9]* ;
WS : [ \t\r\n]+ -> skip ;
```

Un programme est constitué d'une liste de déclaration de fonction et/ou de procédure, suivi d'une liste d'instruction, chacun de ces deux élément pouvant être optionnel :

```
programme :
    liste_declarations?
    liste_instructions?
;

liste_declarations :
    (declarationProcedure|declarationFunction)+
;

liste_instructions :
    (instruction)+
;
```

On donne ensuite la structure des déclarations de procédure et de fonction. La différence entre les deux réside dans le fait qu'une fonction doit se terminer par une action retournant une valeur. Contrairement à la syntaxe originale du Logo, nous avons choisi de délimiter la liste des paramètres par des parenthèses pour des raisons qui seront explicitées plus loin dans ce document.

```
declarationProcedure :
    'pour' ID '(' liste_params? ')'
        liste_instructions
    'fin'
;

declarationFunction :
```

```
    'pour' ID '(' liste_params? ')'
        liste_instructions?
        'rends' exp
    'fin'
;

liste_params :
    (':'ID)+
;
```

On défini ensuite la structure des différentes instructions standard ainsi que celle des appels de procédure.

```
instruction :
    ID '(' (exp)* ')' # procedureCall
    | 'av' exp # av
    | 'td' exp # td
    | 'tg' exp # tg
    | 'lc' # lc
    | 'bc' # bc
    | 've' # ve
    | 're' exp # re
    | 'fpos' exp exp # fpos
    | 'fcc' exp # fcc
    | 'repete' exp '['liste_instructions']' # repete
    | 'si' exp '['liste_instructions']' ('['liste_instructions']')? # si
    | 'tantque' exp '['liste_instructions']' # tantque
    | 'donne' '"' ID exp # donne
;
```

On clôture finalement la grammaire en donnant la structure des expressions mathématiques et des appels de fonctions utilisés comme paramètres pour les instructions précédentes.

```
exp :
    ID '(' (exp)* ')' # functionCall
    | 'hasard' exp # hasard
    | exp ('*'|'/') exp # mult
    | exp ('+'|'-') exp # sum
    | exp ('=='|'>='|'<='|'>'|'<'|'!=') exp # test
    | '!' exp # neg
    | atom # arule
;

atom :
    INT # int
    | ':'ID # id
    | 'loop' # loop
    | '('exp')' # parent
;
```