JavaCC Parser for Logo

Generation of JavaApplet

Project documentation in Module AFL

Field of Study:	Information Technology
Authors:	Emanuel Knecht (knece1@bfh.ch)
	David Aeschlimann (aescd5@bfh.ch)
Professor:	Olivier Biberstein
Date:	12.06.2015
Version:	Version 2.0

Table of Contents

1 Introduction

This document contains a description of the work done during the project for the class Automata and Formal Languages and the results that have been accomplished by our group.

1.1 Problem/Task

The objective of this project is to develop a parser/translator from a given subset of the Logo programming language into Java. The parser must be developed by means of JavaCC. The EBNF grammar was provided, as well as some examples of Logo programs, the primitives of Logo written in Java and a base ant project with a partly implemented JavaCC file. We are allowed to modify the grammar, but not to change the specification of the language. The developed parser/translator implements the provided grammar and must be able to run all the Logo programs provides as examples without modifications.

1.1.1 Logo

1.1.2 Deliverables

The deliverables are as follows:

- This written report documenting our work and the result of our project
- The parser/translator written in JavaCC
- A logo program we used to test our parser

1.2 Grammar

We didn't modify the grammar in any way.

```
Program    = "LOGO" Identifier { Subroutine } { Statement } "END"

Subroutine = "TO" Identifier { Parameter } { Statement } "END"

Statement  = "CS" | "PD" | "PU" | "HT" | "ST"
           | "FD" NExpr | "BK" NExpr | "LT" NExpr | "RT" NExpr
           | "WAIT" NExpr
           | "REPEAT" NExpr "[" { Statement } "]"
           | "IF" BExpr "[" { Statement } "]"
           | "IFELSE" BExpr "[" { Statement } "]" "[" { Statement } "]"
           | Identifier { NExpr }

NExpr      = NTerm { ( "+" | "-" )  NTerm }

NTerm      = NFactor { ( "*" | "/" ) NFactor }

NFactor    = "-" ( Number | REPCOUNT | Parameter | "(" NExpr ")" ) |
         Number | REPCOUNT | Parameter | "(" NExpr ")"

BExpr      = BTerm { "OR" BTerm }

BTerm      = BFactor { "AND" BFactor }

BFactor    = "TRUE" | "FALSE" | "NOT" "(" BExpr ")"
       | NExpr ( "==" | "!=" | "<" | ">" | "<=" | ">=" )  NExpr


Comments start with "#" with scope until the newline
Numbers are real numbers
Identifiers start with a letter followed by letters or digits
Parameters are ":" followed by Identifier
Identifiers, parameters, keywords in uppercase only
```

2 Implementation

2.1 Tests

2.2 Limitations

3 Conclusion