Bern University of Applied Sciences

Automata and Formal Languages
Semester Project
Bern
BTI7064r

Logo Project

Bolaños Mayta Raul Alberto
bolar1@bfh.ch

Roland Rytz
roland.rytz@gmail.com

15. Januar 2016

Inhaltsverzeichnis

1 Introduction

Logo is a easy to lean not case sensitive programing language in which his commands represents movements of a virtual turtle that produces lines graphics.

In this Project we develop a `parser / translator` of the Logo programming language into Java. This logo translator should be able to run all the Logo programs provides as examples without modifications. Logo

2 Grammar

For this project we did not modified the Grammar

```
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
```

3 Solution

First, we added syntax checking for LOGO, without actually implementing an interpreter. This was fairly straight-forward after we got used to javaCC's syntax.

After this, functionality of an interpreter was added step-by-step, starting at the most basic statements and building upon them.

There were some problems, like ensuring that the output is always proper Java syntax. At first, REPCOUNT didn't work with nested REPEAT statements. We then decided to implement REPCOUNT as a stack; with each deeper level of REPEAT nesting, another element is pushed to that stack, and popped on leaving the loop.

We also experienced problems in getting the turtle to display correctly. Originally, the turtle would only be drawn, never erased - This lead to the turtle being left over after every single instruction. First, we tried getting around this by drawing the turtle on a separate canvas from the actual drawing. This would have brought various advantages, such as draw speed and leaving the drawing intact when erasing the turtle.

Limitations in java.applet and our knowledge of java lead to this idea being shelved; instead, we found that the method pd() in LogoPrimitives.java always sets the draw color to black, which made erasing the turtle impossible. After fixing this, the turtle was erased properly, but since it's merely erased by drawing over it in white, it will damage drawings under the turtle.

4 Test

We test the translator with all the given logo examples and some more specific tests.

That test ensure that REPCOUNT still works and can be complied even with more nested REPEAT statements.

```
# Test
LOGO TEST

   ST
   REPEAT 5 [
   FD 5
      RT REPCOUNT
      WAIT 20
           REPEAT 5 [
                      FD 5
                      RT REPCOUNT
                      WAIT 20
                      REPEAT 5 [
                                 REPEAT 5 [
                                            FD 5
                                            RT REPCOUNT
                                            WAIT 20
                                  ]
                       ]
             ]
   ]

END
```

5 Limitations

The most limiting factor was time; We experienced some of the typical pitfalls in software development, such as lack in communication among collaborators and bad time management.

While our compiler implements all the required features, there are definitely some aspects that could still be polished.

In the end, we did learn a lot about compiler design and JavaCC in particular, and did also get some enjoyment from that along the way.