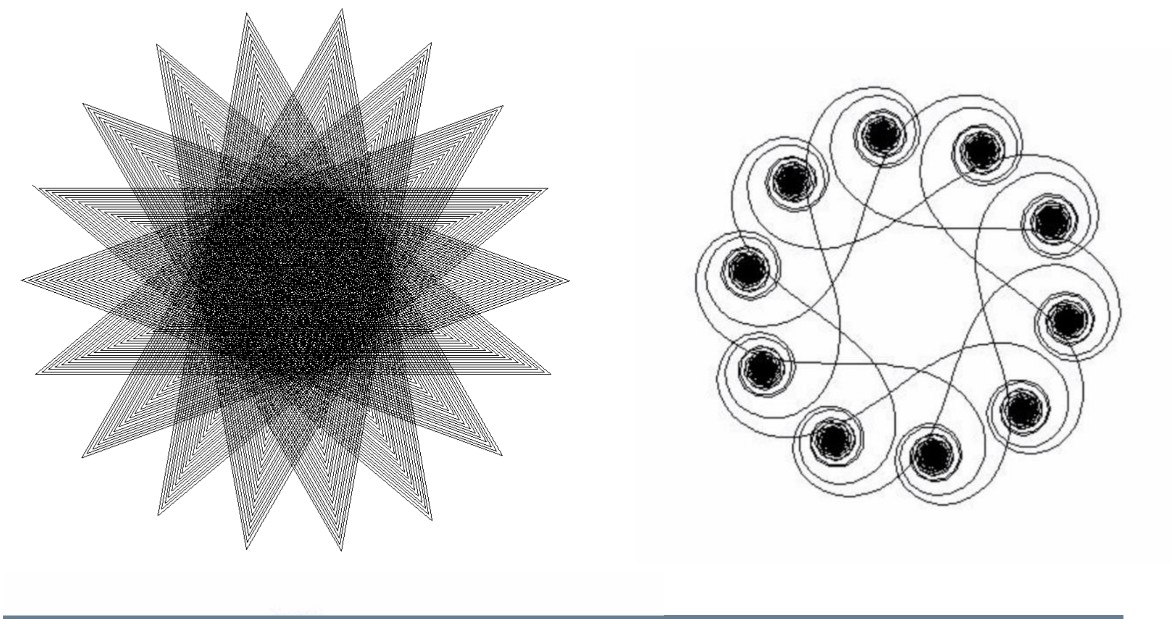


Report: Logo Project

Automata and Formal Languages (BTI7064ab) 20

Autor/en: Mac Müller
Advisor: Prof. Dr. Olivier Biberstein

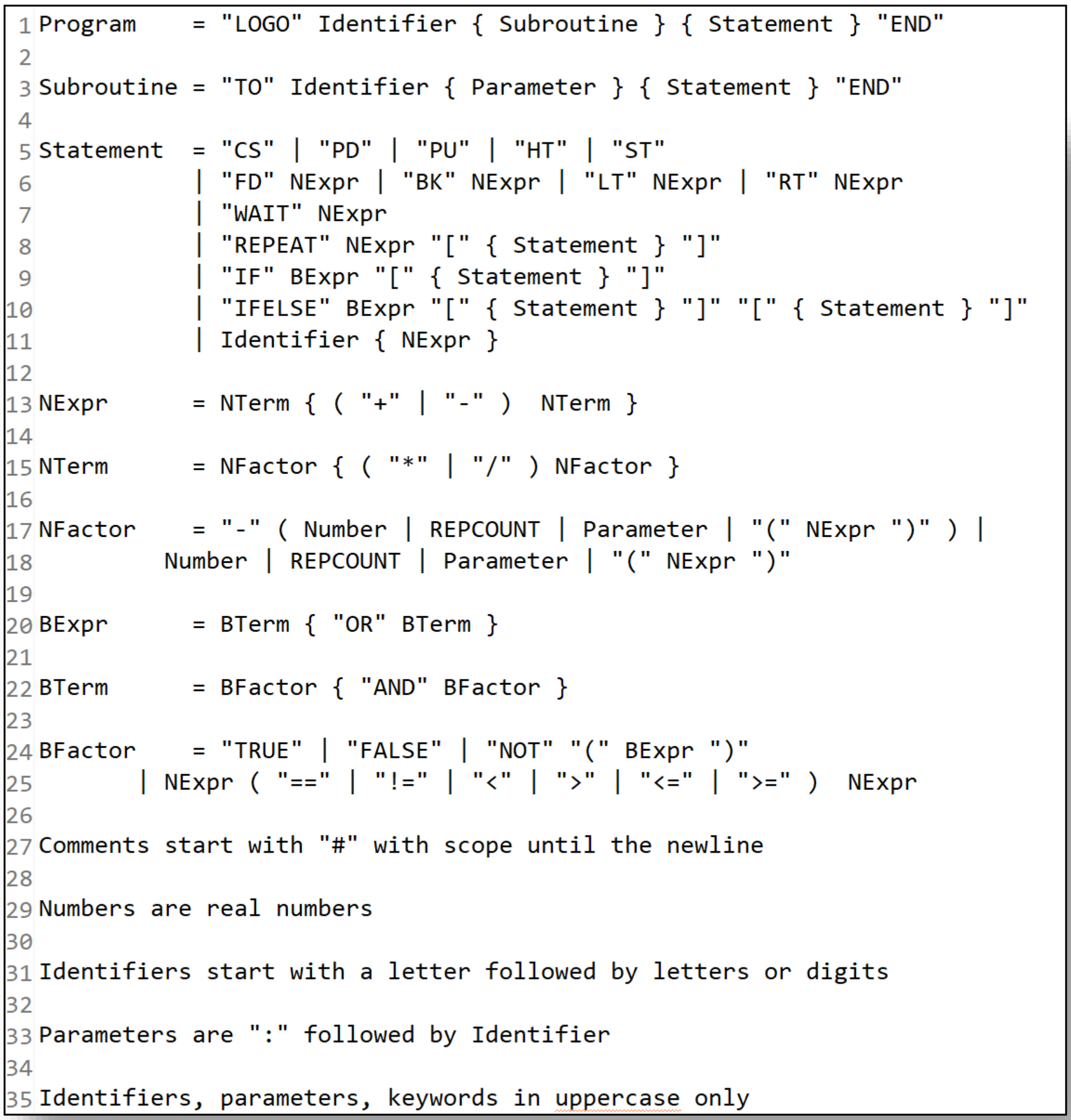
Version 01 / 12.06.2020

Bern University of Applied Sciences

Introduction

The objective of this project was to develop a parser/translator from a small subset of the Logo programming language into Java. The parser was developed by means of JavaCC. The grammar of translator was provided.

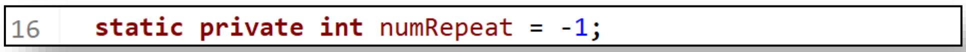
Grammar



```
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
```

REPCOUNT Handling

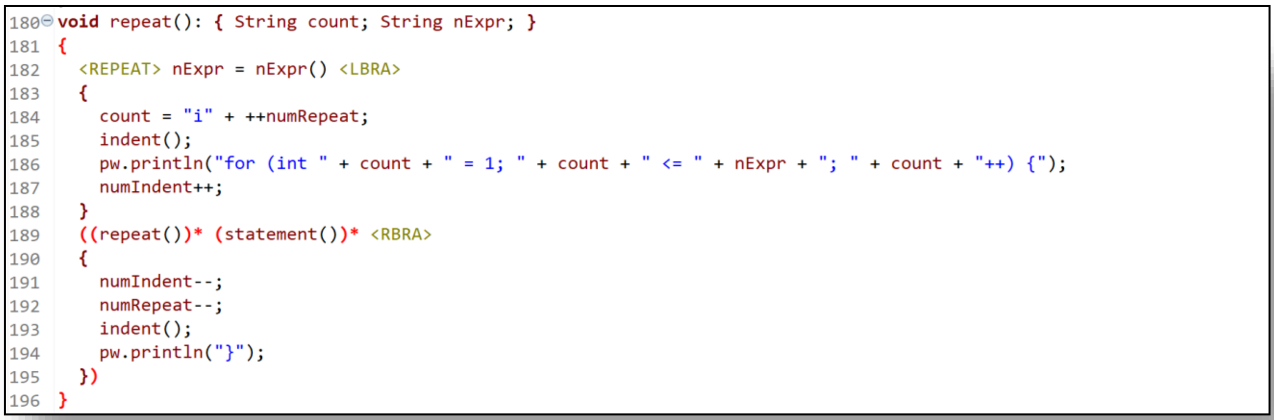
At First: "**numRepeat"** as global variable is initialized for counting.



```
static private int numRepeat = -1;
```

If "repeat()" function is called, "numRepeat" will be added by 1. If the function finished the work, "numRepeat" will be subtracted by 1.

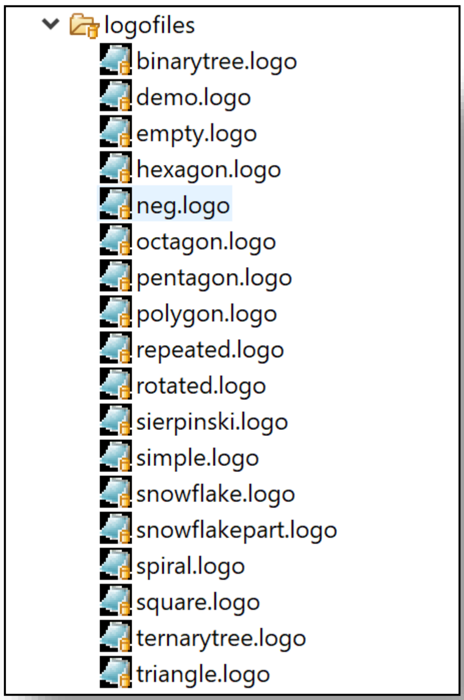
So "for loop" in JAVA code will be created the with combination of "numRepeat" varable.



```
void repeat(): { String count; String nExpr; }
{
  <REPEAT> nExpr = nExpr() <LBRA>
  {
    count = "i" + ++numRepeat;
    indent();
    pw.println("for (int " + count + " = 1; " + count + " <= " + nExpr + "; " + count + "++) {");
    numIndent++;
  }
  ((repeat())* (statement())* <RBRA>
  {
    numIndent--;
    numRepeat--;
    indent();
    pw.println("}");
  })
}
```

Test

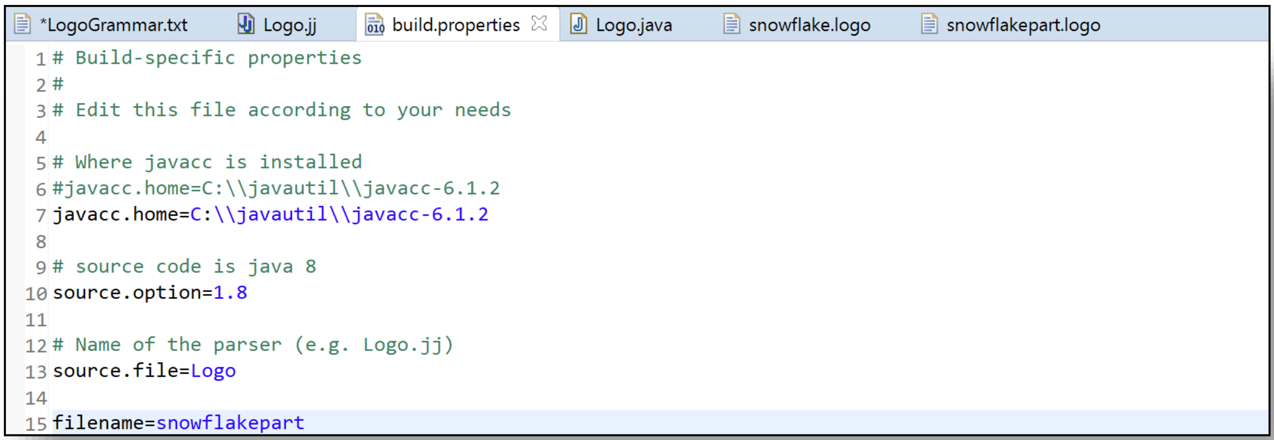
To verify the functionality of our parser following files are provided to test:



logofiles
- binarytree.logo
- demo.logo
- empty.logo
- hexagon.logo
- neg.logo
- octagon.logo
- pentagon.logo
- polygon.logo
- repeated.logo
- rotated.logo
- sierpinski.logo
- simple.logo
- snowflake.logo
- snowflakepart.logo
- spiral.logo
- square.logo
- ternarytree.logo
- triangle.logo

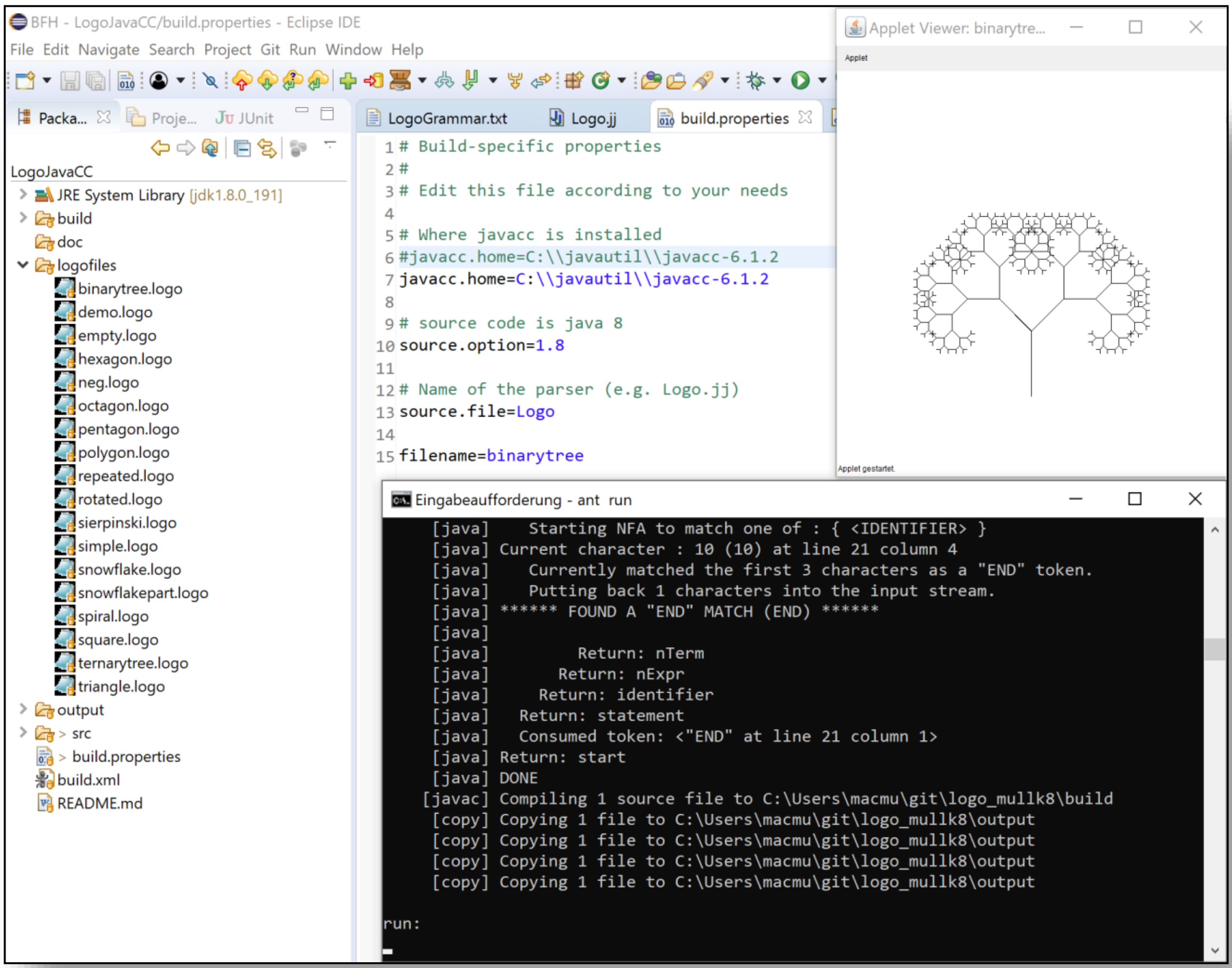
To testing has two possibilities of approaches:

1. With command line “ant run -Dfilename=xxx”. The "xxx" is the filename.
2. Change the "filename=" in "build.properties" file. Then type command line "ant run"
   This approach is more convenience in case of running-error. Debugging with other ant-commands was easier.



```
# Build-specific properties
#
# Edit this file according to your needs

# Where javacc is installed
#javacc.home=C:\\javautil\\javacc-6.1.2
javacc.home=C:\\javautil\\javacc-6.1.2

# source code is java 8
source.option=1.8

# Name of the parser (e.g. Logo.jj)
source.file=Logo

filename=snowflakepart
```

Example pictures of testing:



```
# Build-specific properties
#
# Edit this file according to your needs

# Where javacc is installed
#javacc.home=C:\\javautil\\javacc-6.1.2
javacc.home=C:\\javautil\\javacc-6.1.2

# source code is java 8
source.option=1.8

# Name of the parser (e.g. Logo.jj)
source.file=Logo

filename=binarytree
```

```
     [java]    Starting NFA to match one of : { <IDENTIFIER> }
     [java] Current character : 10 (10) at line 21 column 4
     [java]    Currently matched the first 3 characters as a "END" token.
     [java]    Putting back 1 characters into the input stream.
     [java] ****** FOUND A "END" MATCH (END) ******
     [java]
     [java]          Return: nTerm
     [java]        Return: nExpr
     [java]      Return: identifier
     [java]    Return: statement
     [java]    Consumed token: <"END" at line 21 column 1>
     [java] Return: start
     [java] DONE
    [javac] Compiling 1 source file to C:\Users\macmu\git\logo_mullk8\build
     [copy] Copying 1 file to C:\Users\macmu\git\logo_mullk8\output
     [copy] Copying 1 file to C:\Users\macmu\git\logo_mullk8\output
     [copy] Copying 1 file to C:\Users\macmu\git\logo_mullk8\output
     [copy] Copying 1 file to C:\Users\macmu\git\logo_mullk8\output

run:
```

Problems and difficulties of this project

My opinions, to start the project was very difficult to getting into JAVA CC programming. The setting up of development environment was not easy. The biggest difficulty was debugging the problems. However, this project was very interesting how a text file was read from the gramma and converted by JavaCC to Java.