编译原理Project

ihciah 13307130364; fishtorres 13307130235

Java版尝试

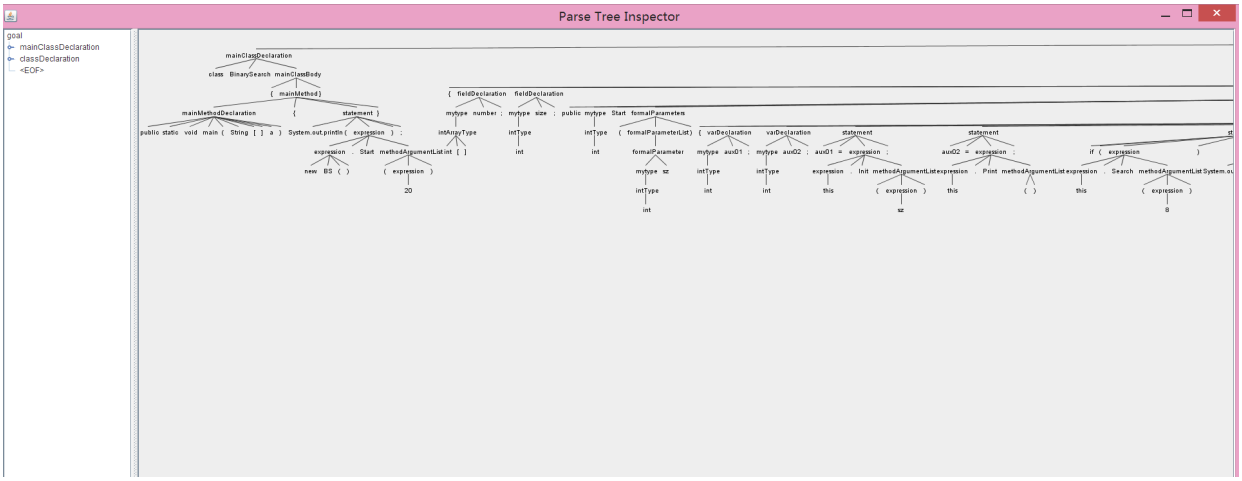
首先下载antlr4，然后把路径丢环境变量里。

然后根据 `MiniJava` 语法规则写 `MiniJava.g4` 。

运行 `java org.antlr.v4.Tool MiniJava.g4` 生成出一些 `MiniJava.java` 的代码文件。

编译： `javac MiniJava*.java`

然后可以用 `java org.antlr.v4.gui.TestRig MiniJava goal binarysearch.java -gui` 画出下图。
从下图我们可以看出 `binarysearch.java` 代码的树。



甚至我们可以把树结构输出出来：
`java org.antlr.v4.gui.TestRig MiniJava goal binarysearch.java -tree`
(输出文件见 `src/Java/binarytree.tree`)

要继续处理的话大概是要写 `Java` 的…并不会写java的我查了下真的可以用python….

Python版

首先运行 `java org.antlr.v4.Tool -Dlanguage=Python2 -visitor MiniJava.g4` 生成带有 `visitor` 的 `python` 代码。

错误处理

- 语法错误：
  删除掉 `binarysearch.java` 文件中11行行末的分号。
  运行脚本，则提示12行的开头缺少分号。

```
 9
10      class BS{
11          int[] number
12          int size ;
13
```

```
C:\Python27\python.exe C:/Users/Lemon/Desktop/MiniJava/python-test/my.py
line 12:4 missing ';' at 'int'

Process finished with exit code 0
```

- 词法错误：
  在 `binarysearch.java` 文件中第11行开头处添加一个减号。
  运行脚本，则提示对应词法错误。

```
10      class BS{
11          -int[] number ;
12          int size ;
13
```

```
C:\Python27\python.exe C:/Users/Lemon/Desktop/MiniJava/python-test/my.py
line 11:4 extraneous input '-' expecting {'public', '}', 'int', 'boolean', Identifier}
```

- 语义错误：
  将 `binarysearch.java` 文件中第135行的 `int j;` 注释掉。
  运行脚本提示语义错误。

```
123         // Initialize the integer array
124         public int Init(int sz){
125         //int j ;
126         int k ;
127         int aux02 ;
128         int aux01 ;
129 
130         size = sz ;
131         number = new int[sz] ;
132 
133         j = 1 ;
134         k = size + 1 ;
135         while (j < (size)) {
```

```
C:\Python27\python.exe C:/Users/Lemon/Desktop/MiniJava/src/Python/my.py
Error: 133:1 near j not defined.
Error: 139:5 near j not defined.

Process finished with exit code 0
```

该部分使用Python进行分析。首先读取语法树，并进行深度优先遍历，并维护一个list存储当前作用域下有效的变量名。由语法树结构可知，某个 `DeclarationContext` 的作用范围为其所有子孙，以及父节点的后续所有子孙。

```
    def ScopeCheck(mytree, active_vars=list()):
        if "children" not in dir(mytree):
            return
        local_vars = list()
        for node in mytree.children:
            # print str(type(node))
            if isinstance(node, MiniJavaParser.VarDeclarationContext) or
    isinstance(node, MiniJavaParser.FieldDeclarationContext):
                name = node.children[1]
                local_vars.append(str(name))
            ExpressionCheck(node, active_vars + local_vars)
            ScopeCheck(node, active_vars + local_vars)
```

这样就确定了每个阶段的有效变量名。

之后只需要在遍历的同时，过滤出所有表达式里的变量，并检查有效性即可。

```
def ExpressionCheck(node, xvars):
    EXPCLS = [MiniJavaParser.AssignStatementContext,
              MiniJavaParser.MulExpressionContext,
              MiniJavaParser.SubExpressionContext,
              MiniJavaParser.AddExpressionContext,
              MiniJavaParser.LtExpressionContext,
              MiniJavaParser.AndExpressionContext]
    if isinstance(node, tree.Tree.TerminalNodeImpl):
        if any([isinstance(node.parentCtx, cls) for cls in EXPCLS]):
            if str(node) not in ['*', '+', '-', '<', '&&', ';', '=']
 + xvars:
                print "Error: %d:%d near %s not defined." % (node.sym
bol.line, node.symbol.column, node.symbol.text)
                return False
    return True
```

额外功能

- 添加除法运算