编译原理实验报告

PA1-A

姓名：谢兴宇
学号：2017011326

2019 年 10 月

本次实验我选择了 Scala 框架和 PA1-A 的专门代码，对于框架有了初步的认识，之后的 PA 我计划切换到整体框架。

1 实验思路

对于每一种新特性，所需做之事可大致归纳如下：

- 如有需要，添加词法信息并注册 token；
- 修改或添加语法；
- 修改或添加其在树种节点的定义；
- 修改或添加其在`Parser`中的 parse 流程；
- 修复由此处修改导致的其他更改，对于目前尚不会用到的部分（比如`tac`和`typecheck`），简单起见我直接删去了会出错的部分。

当然，这是事后的总结，在实际的实验过程中，我的策略是先改好所有我认为需要更改的地方；再试图进行编译，修复编译错误；将自己对于测例的运行情况与正确的答案相比较，思考可能导致不一致的错误及漏改之处。

1.1 抽象类

- 在 DecafLexer.g4 中添加`abstract`。

- 在 DecafLexer.token4 中为`abstract`注册 token。
- 在 DecafParser.g4 中添加以`abstract`修饰类和函数的语法。
- 将`abstract`添加为一个新的 modifier。
- 为了不影响其他部分对`ClassDef`的使用，将`modifiers`作为其一个默认参数，为了匹配格式化打印，需要将参数顺序重排，将`modifiers`置于最先遍历的位置。

1.2 局部类型推断

- 在 DecafLexer.g4 中添加`var`，由于 var(case-insensitive) 在 DecafParser.g4 中已被使用，为避免不必要的麻烦，我使用了`AUTO`作为`'var'`的 TokenName。
- 在 DecafLexer.token4 中为`var`注册 token。
- 在 DecafParser.g4 中添加以`var`修饰局部变量的语法。
- 在`decaf.frontend.tree.TreeTmpl`中添加自动类型`TVar`。
- 修改`decaf.frontend.parsing.Parser.visitLocalVarDef`使之能解析推断类型的局部变量。

1.3 First-class Functions

1.3.1 函数类型

- 在 DecafParser.g4 中添加函数类型的语法。
- 在`decaf.frontend.tree.TreeTmpl`中添加函数类型`TLambda`。
- 添加做 tree transformation 的函数`decaf.frontend.parsing.Parser.visitLambdaType`，对于`List`的处理借鉴了`visitMethodDef`。

1.3.2 Lambda 表达式

- 在 DecafLexer.g4 中添加`fun`。
- 在 DecafLexer.token4 中为`fun`注册 token。

- 在 DecafParser.g4 中添加以`fun`修饰局部变量的语法。
- 方便起见我将 block lambda 和 expression lambda 作为不同的节点`ExpressionLambda`和`BlockLambda`处理，将其`productPrefix`都重写为`"Lambda"`以保持 AST 格式化打印时的一致性。
- 添加 tree transformation 所需函数`visitExpressionLambda`和`visitBlockLambda`。

1.3.3 函数调用

- 修改文法,将变量和函数调用的文法分开,抛却了非终结符`VarSelOrCall`。
- 修改`decaf.frontend.tree.SyntaxTree`中的`Call`使之符合新语法。
- 将`decaf.frontend.parsing.Parser`中的`visitVarSelOrCall`改为`visitCall`，重写了`visitPath`和`visitSinglePath`。

2 疑难解决

2.1 环境配置

由于这是我第一次接触 Scala（甚至这是我用过的第一个 JVM 语言），配置环境对我来说并非易事，经过一夜的尝试才终于配置成功，并发现了存在于Decaf 实验文档中的一些错误：

- “安装好 JDK8 或更高版本的开发环境”：我们的代码要求使用 JDK12。
- “Visual Studio Code 可安装 sbt 插件为集成开发环境”：经过一段时间的探索之后，我并没有学会如何使用这一插件。此插件并无文档，在其评论区中有人指出其需要在使用 VSCode 打开 Scala 工程前先运行 sbt 才能使用此插件，不过由于其过于繁琐我并未尝试。
- “在项目根目录下运行`sbt build`完成构建”：应是`sbt compile`。且 sbt 是一交互式工具，如此使用并非官方推荐的用法，会浪费较多时间用于 sbt 的初始化和设定载入。

2.2 字符串解析

有时对于字符串的判断会出现问题。最初，字符串末尾的双引号被识别为了`DecafLexer.UNTERM_STRING`，我通过更改词法的行序修复了这个问题，但我并不能解释它为何能得以修复；之后，字符串中的新行又被识别为了`DecafLexer.UNTERM_STRING`，我将 token 与助教的发布版本保持兼容后，此问题得以修复，猜想这是因为 scala 代码中存在对于 token 的 hardcode 所致。

我花了大量时间调试字符串的词法解析，我想这多少是因为 antlr4 对于 EOF 并未妥善处理和此框架对于 EOF 过于技巧性的处理所致。`DecafLexer.UNTERM_STRING`考虑的是在字符串中遇到了 EOF 的情况，但根据最新的 POSIX 标准（POSIX.1-2017），Text File 被定义为（POSIX.1-2017 3.403）：

> A file that contains characters organized into zero or more **lines**. The lines do not contain NUL characters and none can exceed {LINE_MAX} bytes in length, including the <newline> character. Although POSIX.1-2017 does not distinguish between text files and binary files (see the ISO C standard), many utilities only produce predictable or meaningful output when operating on text files. The standard utilities that have such restrictions always specify "text files" in their STDIN or INPUT FILES sections.

Line 被定义为（POSIX.1-2017 3.206）：

> A sequence of zero or more non- <newline> characters plus a terminating <newline> character.

故对于 Text File——Decaf 源代码文件当属此类——EOF 前必有 endline。故在解析字符串时在遇到 EOF 前必然会遇到 <newline>。即在被`DecafLexer.UNTERM_STRING`捕捉到之前必先被`DecafLexer.ERROR_NEWLINE`‘捕捉到，故此处对于 EOF 的判断在 POSIX 标准下是完全无需的。

3 问题回答

问题 1. AST 结点间是有继承关系的。若结点 A 继承了 B，那么语法上会不会 A 和 B 有什么关系？限用 100 字符内一句话说明。
回答 在本框架的 AST 中，结点 A 继承结点 B 仅当消去 Unit 产生式后

- B 有多个产生式
- 且 A 是 B 的产生式之一

问题 2. 原有框架是如何解决空悬 else(dangling-else) 问题的？限用 100 字符内说明。
回答 通过采用贪心策略（使用尽量多的输入）的形如`(...)?`的 EBNF 子规则：

```
stmt:
    ...
    | IF '(' cond = expr ')' trueBranch = stmt (
        ELSE falseBranch = stmt
    )?
    ...
```

问题 3. PA1-A 在概念上，如下图所示：

作为输入的程序（字符串）
–> lexer –> 单词流 (token stream)
–> parser –> 具体语法树 (CST)
–> 一通操作 –> 抽象语法树 (AST)

输入程序 lex 完得到一个终结符序列，然后构建出具体语法树，最后从具体语法树构建抽象语法树。这个概念模型与框架的实现有什么区别？我们的具体语法树在哪里？限用 120 字符内说明。
回答 从输入字符串到 CST 的过程交由 Antlr 来实现，其是按需（by need）解析单词的，“一通操作”指遍历 Antlr 生成的 CST 将其变换为 AST。CST 由 Antlr 生成，具体到代码，是`decaf.frontend.parsing.Parser.transform`中的`parser`。