PA1-A 实验报告

计科70 徐明宽 2017011310

工作内容

抽象类

我修改了Decaf.jflex, Decaf.jacc, Tokens.java, JaccParser.java等文件以加入`abstract`关键字（以及后面要用到的`var`、`fun`关键字与`=>`操作符）（后来为了后面的PA1-B阶段又在Decaf.spec和LLParser.java中加入了这些关键字与操作符），并照着给出的EBNF语法规范修改了Decaf.jacc。在Tree.java中，由于`body`这个成员不一定要有，我尝试为`Block`类加入空构造函数发现输出与答案不一样之后将`body`的类型更改为`Optional<Block>`，然后发现需要修一下TacGen.java, Namer.java, Typer.java, Decaf.spec这些与本阶段无关的文件的依赖关系。

在Tree.java中，我为`Modifiers`类加入了`abstract`修饰符，并为`MethodDef`和`ClassDef`类增加了这一修饰。

局部类型推断

与抽象类类似，我照着给出的EBNF语法规范修改了Decaf.jacc，然后我以为需要写类型推断，就开始写一个新的构造函数试图通过`Expr initVal`来推断类型，直到我发现答案中的类型是`<none>`。于是我只能将`typeLit`的类型改为`Optional<typeLit>`，并修一下Namer.java这一与本阶段无关的文件中用到`typeLit`的地方。

First-class Functions

函数类型

我参考Decaf.jacc中的`VarList`实现了`TypeList`，参考FunType.java和Tree.java中的`TArray`类实现了`TLambda`类，在SemValue.java中增加了`typeList`的判断，在Visitor.java中添加了`visitTLambda`（以及后面的`visitLambda`），在AbstractParser.java中参考`protected SemValue svVars(Tree.LocalVarDef... vars)`实现了`protected SemValue svTypes(Tree.TypeLit... types)`~~并发现前者是从`svFields`复制过来的但忘了把`FIELD_LIST`改掉了于是发了个pr~~。

Lambda 表达式

我在Tree.java中参考其他类实现了继承`Expr`类的`Lambda`类，其中指导书中的`paramList`我沿用了代码中的`List<LocalVarDef>`（毕竟是按照`LocalVarDef`来打印）。对于两种Lambda 表达式，我用了`Optional<Expr>`和`Optional<Block>`两个变量，并保证其中有且仅有一个是`isPresent()`的。然后我发现`var f = fun (int x) => x + 1;`被解析成`var f = (fun (int x) => x) + 1;`了，于是改了一下运算符优先级。

函数调用

我把`Call`类的`receiver`和`method`改成一个变量`methodExpr`之后，出现了大量依赖关系问题，我只能先把Decaf.spec, TacEmitter.java和Typer.java中的部分代码注释掉，以后再修。

运算符优先级

在我完成PA1-A的实验内容之后，申奥同学与我讨论了decaf语言的二义性问题，我们由`(class a)b(1)`这条语句的二义性发现decaf语言的文档并没有规定强制类型转换的优先级。我们发现原有的框架会把`(class a)b(1)`解析成`(class a)(b(1))`（这是因为原有的框架的Call不支持另一种解析方式），却会把`(class a)b.c(1)`解析成`((class a)b).c(1)`，会把`a.b.c(1)`解析成`(a.b).c(1)`。将这一现象反馈给助教后，助教及时更新了文档，将强制类型转换的优先级规定为与负号一样（和C++语言中对C风格强制类型转换的规定一样）。

修改Decaf.jacc、加入一行`%prec`即可指定这一优先级。

这里有个坑：在这一修改后，可能需要删掉build文件夹或者更新src文件夹下代码的修改日期来彻底重新编译，才能使修改生效（否则会编译成功但运行时报syntax error）。

回答问题

Q1. (Java) 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?

若`A`继承了`B`，那么语法上`A`一定是一个`B`（`B`可被解析成`A`），如`LocalVarDef`是`Stmt`，`TInt`是`TypeLit`。

Q2. 原有框架是如何解决空悬 else (dangling-else) 问题的?

框架为`ElseClause`的空解析用`%prec`指定了`EMPTY`的优先级，比`ELSE`的优先级更低，因此`if...if...else`会优先将后者的`ElseClause`解析为`else{...}`而不是空，从而解决空悬else问题。

Q3. 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?

区别在于我们没有显式地构建出具体语法树，而是通过Decaf.jacc生成的parser由单词流直接构建抽象语法树。我们的具体语法树只在Decaf.jacc里，用以指示构建抽象语法树的规则。

致谢

我完成词法分析部分以后阅读了实验指导书却还不知道接下来应该去修改哪个文件以实现文法分析部分的逻辑，在此感谢罗承扬同学告诉我应该去看Tree.java这一文件（在此之后，有若干个其他同学也问了我这个问题）。

感谢申奥同学（学号2017012518）与我讨论decaf语言的二义性问题，并一起检查每个移进/归约冲突以发现`(class a)b(1)`这条语句的二义性问题。