PA1-B 实验报告

计科70 徐明宽 2017011310

工作内容

LL(1) 语法分析

抽象类

对`methodDef`，我在Decaf.spec中仿照`STATIC Type Id '(' VarList ')' Block FieldList`写了`ABSTRACT Type Id '(' VarList ')' ';' FieldList`（`classDef`已在PA1-A中改好）。

局部类型推断

与PA1-A相同。

First-class Functions

函数类型

我参考Decaf.spec中的`VarList`实现了`TypeList`，并写了非终结符`AfterAtomType`。我在SemValue.java中添加了`List<MutablePair<List<Tree.TypeLit>, Integer>> typeListList`以同时存储形如`int(int, int)[](int, int, int)`中的参数列表`int, int`和`int, int, int`以及其中的`[]`的个数，并在AbstractParser.java中添加了`protected SemValue svTypess()`。

然后我发现对`new`的处理也用到了`Type`的定义，需要手动加入函数类型，类比`AfterAtomType`实现了`AfterNewAtomType`。

Lambda 表达式

我参照实验说明写了`FUN '(' VarList ')' AfterFunExpr`，并在`AfterFunExpr`的产生式中实现了Lambda表达式的两种情况。

函数调用

对于`AfterLParen`和`Expr8`中的函数调用，我重写了`ExprT8`使其对应成员字段选择、数组索引、函数调用这一优先级~~，同时发现函数类型部分可以使用`SemValue`的`thunkList`而无需开一个`List<MutablePair<List<Tree.TypeLit>, Integer>> typeListList`~~。对于`Expr9`中的函数调用，我直接将其删除了，因为函数调用并不属于这一优先级。

错误恢复

我实现了“实验内容”一节中推荐的算法，当遇到$Begin(A)$中的符号时恢复分析A，当遇到$End(A)$中的符号时返回`null`（并且使得A的各父节点均返回`null`），否则跳过该符号（消耗终结符）。

在实验过程中，我发现`parseSymbol`的参数`symbol`为`int`类型，调试时不能直观看出输出的`symbol`具体表示哪个非终结符，于是我**读入**了`LLTable.java`，在里面查找字符串`public static final int`，以此建立一个`symbol`与非终结符字符串的对应关系表，方便调试。

运算符结合性

我发现框架中对大小比较运算符实现的是左结合，而语言规范中写的是不结合，于是将产生式`ExprT4 -> Op4 Expr5 ExprT4`改成了`ExprT4 -> Op4 Expr5`。

回答问题

Q1. 本阶段框架是如何解决空悬 else (dangling-else) 问题的?

如https://github.com/paulzfm/ll1pg/wiki/2.-Resolving-Conflicts所述，当冲突发生时写在前面的产生式优先级更高，所以`E : else S | /* empty */`会优先将`E`解析为`else S`而不是空，从而解决空悬else问题。

Q2. 使用 LL(1) 文法如何描述二元运算符的优先级与结合性？请结合框架中的文法，举例说明。

框架中用多个非终结符来描述二元运算符的优先级，如`Expr6`表示"项"，即只含乘除模及更高优先级运算符的表达式；`Expr5`则表示只含加减及更高优先级运算符的表达式。对于左结合的二元运算符如加减（`Op5`），框架中实现了产生式`Expr5 -> Expr6 ExprT5`与`ExprT5 -> Op5 Expr6 ExprT5 | /* empty */`；右结合的只需将左结合的代码中在`thunkList`的开头插入改为在结尾插入；不结合的可以写`ExprT4 -> Op4 Expr5 | /* empty */`。

Q3. 无论何种错误恢复方法，都无法完全避免误报的问题。 请举出一个具体的 Decaf 程序（显然它要有语法错误），用你实现的错误恢复算法进行语法分析时会带来误报。 并说明该算法为什么无法避免这种误报。

```
class Main {
    static void main() {
        int(, int) x;
    }
}
```

我的输出：

```
*** Error at (3,13): syntax error
*** Error at (3,18): syntax error
*** Error at (5,1): syntax error
```

其中`Error at (5,1)`明显属于误报。对比我在PA1-A阶段的输出：

```
*** Error at (3,13): syntax error
```

这是因为集合$End(A)$太大了，应跳过一些符号时我们却选择了分析失败、不跳过符号直接返回。如$',' \in follow(AfterAtomType)$，但我们在应用展开式`Var -> Type Id`时并不期望`Id`以`','`开头。因此，虽然$',' \in follow(AfterAtomType)$，我们也不应直接返回，因为$',' \in follow(AfterAtomType)$是因为有`TypeList -> Type TypeList1`、`TypeList1 -> ',' Type TypeList1`、`Type -> AtomType AfterAtomType`这些产生式，而我们*在分析到`AfterAtomType`时*并没有遇到过`TypeList`这个非终结符，所以可以预见的是这个`','`将会导致大量非终结符分析失败，但我们在这里甚至不能排除这个`','`是`Block`后面的这一可能性：**我们无法权衡应跳过一个终结符还是回溯若干层非终结符**，因而**无法避免**这种误报。

我们可能会想，

我们的错误恢复算法在读入`int(`后看到`','`，分析非终结符`TypeList`、`AfterAtomType`、`Type`时均没有消耗这个`','`就分析失败了，于是接下来试图分析`Type`后面的`Id`看到`','`失败，于是分析`Var`失败，分析`Initializer`看到`','`失败，于是分析`SimpleStmt`失败，分析`Stmt`失败，分析`StmtList`失败，分析`Block`失败，返回到`FieldList`时才意识到必须要跳过这个`','`了，于是应用了我们并不期望的产生式`FieldList -> Type Id AfterIdField FieldList`，消耗了`int`然后看到`')'`于是分析`Id`失败，跳过`')' x`后消耗了`';'`，回到`ClassDef`消耗了第一个`'}'`，于是在第二个`'}'`处再次报错。

致谢

无。