PA1-B实验报告

计73 林俊峰

2017011303

实验目的

使用LL(1)文法将PA1-A重做一次并增加错误处理

0 Merge

PA1-A使用的分离框架，PA1-B直接换成完整框架了，本次任务开始前，先将上一次作业合并到总框架上，用idea自带的compare很方便

1 LL(1)语法分析

1，2 abstract关键字和局部类型推断

这两部分基本同PA1-A

3 First-class Functions

3.1 函数类型

实现思路

将对应文法修改为LL(1)文法即可。

这个地方有个难点就是如何识别数组和Lambda表达式交错的串，如int []，感觉是这部分最难的地方

在查看其它文法的时候发现可以使用thunklist + sv来实现一个类似栈的功能，完美解决

文法实现

> Type -> AtomType ArrayType
>
> ArrayType -> [ ] ArrayType
>
> | (TypeList) ArrayType
>
> | epsilon
>
> TypeList -> Type TypeList1
>
> TypeList1 -> , Type TypeList1 | epsilon

3.2 Lambda 表达式

实现思路

根据题目要求，再多做一步提取左公因式即可

文法实现

> Expr -> Expr1
>
> | FUN '(' VarList ')' RLambda
>
> RLambda -> '=>' Expr | Block

####

3.3 函数调用

实现思路

这部分比较复杂的地方在于有好多个Expr，仔细观察不难发现Expr1-7都是在解决符号优先级的问题，主要看8和9即可。观察带有Call的文法，发现Expr9自带的Id ExprOpt是与目标文法冲突的，删去ExprOpt分支后在T8中加入对应文法即可。同时在这步操作做完后会发现输出比答案少了一个 `ValSel` ，那是因为我们需要在8中将含有exprlist的表达式中的 `expr` 与 `id` 通过 `ValSel` 绑定成一个 `expr` 。

文法实现

> ExprT8 -> ... | (ExprList) ExprT8

2 错误恢复

实现思路

按照文档上的步骤一通操作之后发现LLTable自带beginSet和followSet函数，那就很方便了。每次进入parse程序后判断是否符合语法规范，若报错就连续读入token直到token属于begin或end集合并进行相应处理就可以。

注意的地方有两点: 若存在报错，则不进入act函数；递归时需要将参数follow更新。

问题解答

Q1: 本框架是如何解决空悬 else (dangling-else) 问题的？

答: 在对输入串进行自左向右的扫描时，由递归下降的过程可知，`Stmt` 的规约在 `ElseClause` 之前，最靠近的 `if-else` 会被规约成 `Stmt` ,也就是 `else` 与最近的 `if` 匹配

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。

举例:

```
Expr1 -> Expr2 ExprT1                                   --Expr1中为优先级最低的运算，递归下降分析的过程保证了先进行Expr2和ExprT1中的更高优先级的运算
ExprT1-> Op1 Expr2 ExprT1                               --用右递归描述右结合，并且算符优先级联描述优先级
Expr2 -> Expr3 ExprT2                                   --Expr2中为优先级更高的运算
ExprT2-> Op2 Expr3 ExprT2

Op1    -> OR
Op2    -> AND
```

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。

Decaf 例子

```
class Main{
   voi f(){
        return ;
   }
   static void main() {

   }
}
```

报错

> * **Error at (2,4): syntax error** * Error at (5,4): syntax error

理论上应该只有(2,4)处报错，但是多报了(5,4)的报错

错因分析:

第一步推导为

```
TopLevel        ->   ClassDef ClassList
```

第一个 `token` 是 `class` , 应用文法

```
ClassDef    ->   CLASS Id ExtendsClause '{' FieldList '}'
```

右侧前几项都成功解析，开始解析 `FieldList`

首先 `voi` 规约为成 `Id` ，然而 `FieldList` 的所有产生式右侧并不存在以 `Id` 开头的句型，解析失败，报错并继续读入 `token` 读到 `'}'` 后 `FieldList` 规约为空，同时 `ClassDef` 的解析结束，开始解析 `ClassList` ，由文法

```
ClassList       ->   ClassDef ClassList | epsilon
```

可知，接下来解析 `ClassDef` ，然而遇到的第一个 `token` 是 `static` ，解析失败报错，继续读入输入串直到结束