Decaf PA1-B

翁家翌 2016011446

2018.11

1

本阶段工作主要如下：

- 在 Parser.java 的 parse 函数中添加错误处理方法，按照实验指导的方法进行添加。具体为添加两个新的集合 “beginA” 和 “endA”，然后在开始的时候先判断是否要抛异常，如果是则跳过一些字符，剩下代码与原来一致。此外由于递归分析某条分支上报错，那么整个祖先分支都要报错，因此定义了一个全局变量 hasError 来更方便地处理这件事情。
- 在 Parser.spec 中加入 LL(1) 文法，除%%、++、取子数组、Default 和 Foreach 任务之外剩下与 PA1-A 基本一致。在通读 Parser.spec 代码之后发现优先级能够通过逐层递归分析得到解决，很受启发。
- 在 Tree.java 中加入生成相关文法的 AST 的类与函数，与 PA1-A 类似。我现在来看之前写的代码有很多冗余，因此做了一定程度上的修改。

2

由于 else 语句可以为空，因此在文法 G[S]：

```
1 S -> if C then S E
2 E -> else S | <empty>
```

有

```
1 PS(E -> else S) & PS(E -> <empty>) = {else}
```

因此 Decaf 不是严格的 LL(1) 语言。但是可以对不同的产生式赋予不同的优先级，比如优先产生 else，这样一来 PS 集合的交即为空，能够处理成 LL(1) 语言。

比如 test2.decaf 中第 21-23 行：

```
1 if (10 < n) {
2     n = 10;
3 }
```

此时 PS(E -> else S) = {else}，然而 else $\notin$ PS(E -> <empty>)，因此匹配后面那条规则，冲突得以解决。

3

原始编译结果如下所示：

```
1 table generation:
     [java] Warning: conflict productions at line 964:
     [java] ElseClause -> ELSE Stmt
     [java] ElseClause -> <empty>
     [java] Parser is successfully generated and written to "Table.java"
```

将 [| 和 |] 改成 [和] 之后，编译结果如下所示：

```
1 table generation:
     [java] Warning: conflict productions at line 777:
     [java] Expr9 -> Constant
     [java] Expr9 -> READ_INTEGER '(' ')'
     [java] Expr9 -> READ_LINE '(' ')'
     [java] Expr9 -> THIS
     [java] Expr9 -> NEW AfterNewExpr
     [java] Expr9 -> INSTANCEOF '(' Expr ',' IDENTIFIER ')'
     [java] Expr9 -> '(' AfterParenExpr
     [java] Expr9 -> IDENTIFIER AfterIdentExpr
     [java] Expr9 -> '[' Expr FOR IDENTIFIER IN Expr IfBoolExpr ']'
     [java] Warning: conflict productions at line 964:
     [java] ElseClause -> ELSE Stmt
     [java] ElseClause -> <empty>
     [java] Warning: unreachable production:
     [java] Expr9 -> '[' Expr FOR IDENTIFIER IN Expr IfBoolExpr ']'
     [java] predictive set is empty
     [java] Parser is successfully generated and written to "Table.java"
```

可以看到它和众多产生式冲突，并且预测集为空。具体而言，与数组常量的形式一致，都是[开头，有左公因子，无法直接转化为 LL(1) 文法。

4

将 S1+/test1.decaf 的代码第六行 int 改成 in：

```
class Main {
    bool field;

    class Main another;

    in[] foo(int y) {
        int[] q;
        int i;

        q = new int[y];
        i = 0;
        while (i < y) {
            q[i] = i;
```

```
14              i = i + 1;
15          }
16
17          return q;
18      }
19
20      static int main() {
21          return this.foo(5).length();
22      }
23  }
```

报错如下：

```
1  *** Error at (6,5): syntax error
2  *** Error at (6,19): syntax error
3  *** Error at (10,9): syntax error
4  *** Error at (10,21): syntax error
5  *** Error at (10,22): syntax error
6  *** Error at (11,9): syntax error
7  *** Error at (17,9): syntax error
```

理论上只要报第六行的 error，实际上它报了其他错。因为并没有文法以 in 开头，in 都是跟在 identifier 之后。根据提供的错误处理方法，在处理到 FieldList 的时候会报错，跳过第六行，7 和 8 当做类中变量声明，10 理应是函数实际上却是 Expr 所以报错，之后同理。