PA1-B 实验报告

姓名：杨雅儒 班级：无73 学号：2017011071

1. 本阶段工作

本阶段工作主要分为两部分：**错误恢复**部分和**语法分析**部分。

1. 错误恢复部分

这部分的策略同步骤二中提出的错误处理方法，这里还是作一下归纳和介绍：

在parse方法中，对于当前symbol这个即将进行匹配的非终结符，首先这个非终结符的follow集合（代码中为followNow）加入到follow集合中。然后筛掉所有不在它的begin集合也不在follow集合中的lookahead（如果存在被筛掉的则报错）。

对于当前的lookahead，如果在begin集合中，则可以恢复匹配该非终结符【恢复匹配时，同正常匹配一样，不用特地考虑匹配失败，因为会在递归中处理】；如果在follow集合中，我们认为该非终结符已经匹配失败了，所以直接return null。

另外修改了Parser.java的parse方法，并且加入了一个Boolean变量fail，当有语法错误时不再触发act方法——这是调试时发现的一个bug，否则会出现空指针错误。

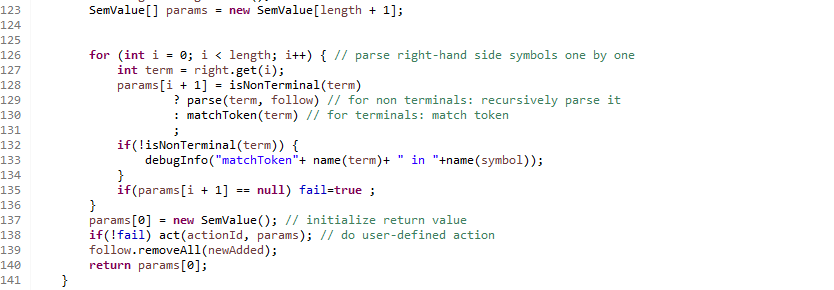
另外注意两点：

\* return时要从follow中删掉由该非终结符新添进去的

\* 匹配终结符时如果失败，并不会消耗掉该终结符，这样才可以回溯到正确处。

这里贴出代码如下：





1. 语法分析部分

* **特性1**

加入Scopy。

同PA1\_A的特性1。Parser.spec中新增对应的token以及文法，Tree.java中新增Scopy类型节点，其中加入一个字符串变量以及一个Expr类变量，Tree.java中新增SCOPY常量。

* **特性2**

加入Sealed。

同PA\_1A的特性2。Parser.spec中新增token和SealedOpt以及其对应的文法，通过SealedOpt返回的代表码的code判定是否有sealed参数，若是则将ClassDef中新增的isSealed变量赋值为true。并且在tree.java中修改了ClassDef节点，向其中加入boolean变量isSealed，并修改了构造函数。

（下面的特性中与PA1-A一样的就一笔带过不再赘述了，**主要叙述差别之处**）

* **特性3**

加入条件卫士语句。

在PA\_1A的特性3的基础上，由于发生了冲突，将IF作为左公因子提出，剩下的以IfSuf代替。

并且由于PA\_1A需要尽早规约，所以会出现直接左递归和其它一些conflict。在这里，IfBranches的产生式中将右侧的两个符号互换位置，在GuardedCont中同样互换位置。另外注意一下输出顺序即可。

* 特性4

支持简单的自动类型推导。

只用注意一下与PA\_1A语法上有区别。在Tree.java中的Ident节点上新增isVar变量并修改构造函数，修改其printTo函数，其余就是正常的加入token和相应常量等。

* **特性5**

1. 数组常量。

与PA1-A相比需要对parser.spec进行修改以改为LL(1)文法，另外新增常量和token同PA1-A，还有在Tree.java中新增ArrayConstant节点，并在SemValue.java中增加该类变量acons。

(2)、（3）数组初始化常量表达式和数组拼接表达式

形如 E%%n 和 E1++E2。

这部分主要需要处理的就是优先级与结合性，做法如下：

在Expr4和Expr5之间依次插入ExprDAdd 和 ExprDMod，以及对应的ExprDAddT和ExprDModT，另外再加入OperDAdd和OperDMod。对于ExprDMod同其它的Expr类似都是左结合的，而对于ExprDAdd是右结合，所以在ExprDAdd的产生式中应当从右向左使用for循环构建AST。这样做我们的AST就成功地构建了。

另外还需要在Binary节点上增加对两个操作符的输出。另外其它常量的添加不再赘述。

(4) 子数组表达式

原本做法：

由于ExprT8处有数组下标表达式，所以将其括号中的Expr后加上一个SubArrayExpr，这个非终结符的产生式有空和 ':' Expr。对于Tree.java需要新增节点ArrayRange，另外新增token和相应常量。

但是这样写会和(5)中的部分导致一些conflict，所以在做(5)的时候进行了修正。

(5) 支持default

调试地有些久了。

主要工作是在ExprT8中增加了内容，一并将上面的(4)进行了整体修改。主要是利用提取的方式，首先判断是子数组表达式还是普通数组（通过判断有无冒号确定），然后对于普通数组则考虑后面是否有default。对于普通数组则还可以继续使用ExprT8跟随其后，而对于default则后面不再有东西。实际上这样的话，冲突就在一次次讨论中被细化而解决了。

(6) comprehension 表达式

基本上同PA1-A，同样加入了IFExpr非终结符，以及加入了CompArrayExpr节点进行处理。唯一的区别就是将'['和']'改为了LOR和ROR两个词法符号。

(7) 数组迭代语句

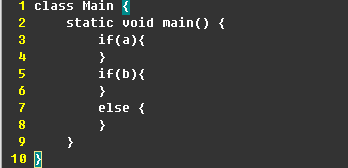
也基本上完全与PA1-A相同。不再赘述。



可以发现，在预测集合中，发生冲突的部分优先处理了else语句。而在给出的工具指示中也写明：“The default setting is unstrict mode. When conflicts appear, we assign higher priority to former defined productions”

也就是说，对于E：else S | /\* empty \*/ ，在非严格模式下，出现冲突时我们优先选择else S对应的产生式，这样就可以处理了。

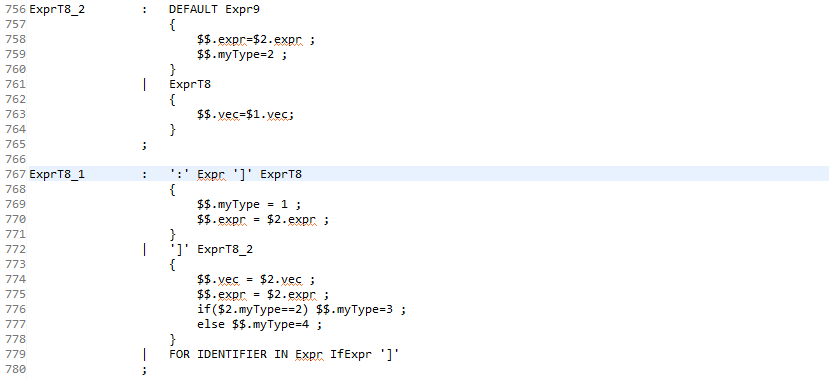
例如出现如下代码时：



则会由于优先级，使得else对应后者。

1. 为什么把原先的comprehension表达式文法改写成LL(1)比较困难？

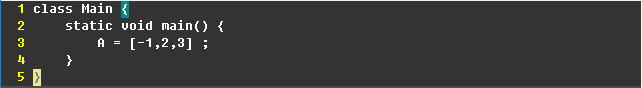
Emm其实在我的实现里面似乎不太难改，也可能是因为我忽略了一些问题，总之在先提取出 ‘[‘ 和 Expr 之后，我使用了如下方式：



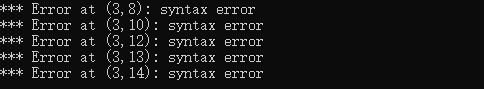
测试时并没有产生其它警告。

1. 无论何种错误处理方法，都无法完全避免误报的问题。请举出一个语法错误的Decaf  
   程序例子， 用你实现的 Parser 进行语法分析会带来误报。 根据你用的错误处理方法， 这些误报为什么会产生？

例如：



报错：



分析如下：

在语法分析过程中，当进入 = 后面时即进入 Expr 的匹配，到负号处时，由于负号不应该出现，所以正常报错。但是此时回到Oper5时将负号成功匹配了，接下来则进入了ExprT5: Oper5 Expr6 ExprT5的匹配，在将1作为Expr6直到Constant匹配完毕之后，接下来的逗号无法匹配，从而导致了误报。