**PA1-B实验报告**

**任务一、二：添加抽象类和局部类型推断**

由于ABSTRACT和VAR的添加与否不影响LL(1)文法的规范性，因而可以直接移植PA1-A的代码（因为两个关键字都加在推导式的最前方，在读取nextToken的过程中不会引起左递归、判断等问题）。同时，沿用PA1-A的关键字注册、类声明与注册。

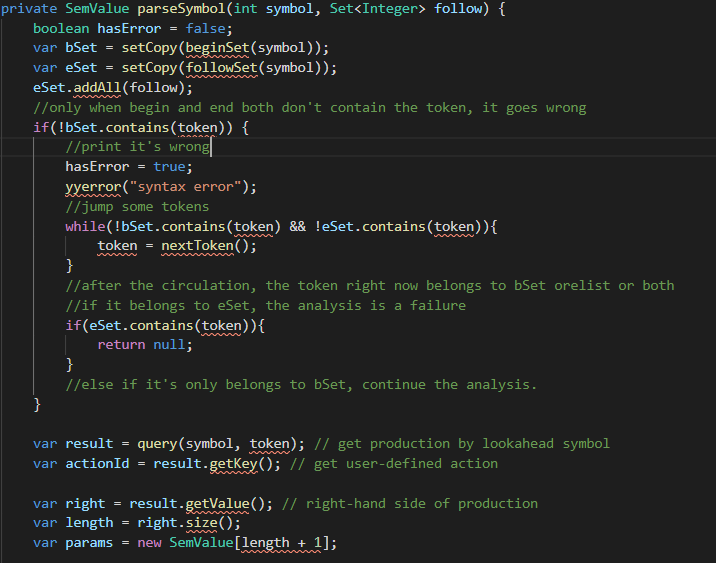
**任务三：添加简单的错误恢复**

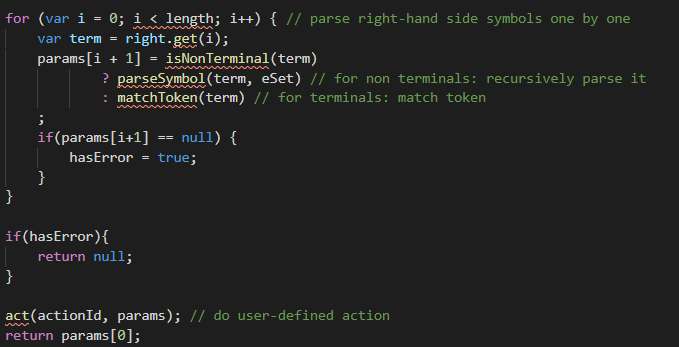
1.实现思路：

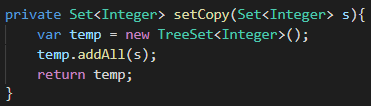
按照实验指导书-实验内容一节的内容，采用如下错误恢复方法：当分析非终结符A时，若当前输入符号a∉Begin(A) ，则先报错，然后跳过符号串中的一些符号，直至遇到Begin(A)∪End(A)中的符号：若遇到的是Begin(A)中的符号，可恢复分析A；若遇到的是End(A)中的符号，则A分析失败，继续分析A后面的

符号。若符号属于两者的交集，则也判断失败。

2.具体实现：



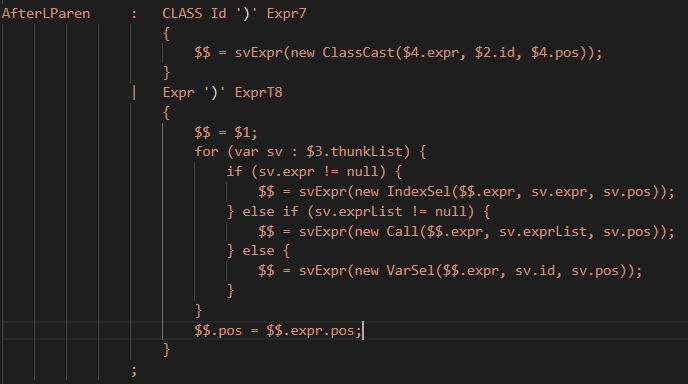


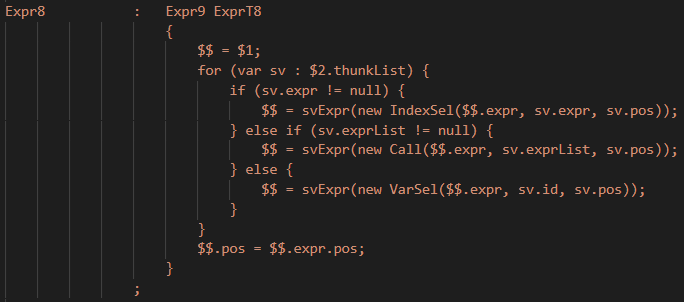
按照上述逻辑进行编写即可。式中提到的setCopy函数是对集合进行深拷贝的方法，具体实现如下：

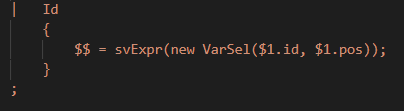
**任务四：添加函数类型、Lambda表达式及函数调用**

1°函数调用

函数调用的过程也可沿用PA1-A的方法，直接将当时更改过的Call类型的函数粘贴过来即可。然后在Decaf.spec文件中进行改动，一共有三处：AfterLParen中else if一处，将Id删去；Expr8中else if一处，将Id删去；Expr9中，将最后一个产生式含有Id的一句删去。

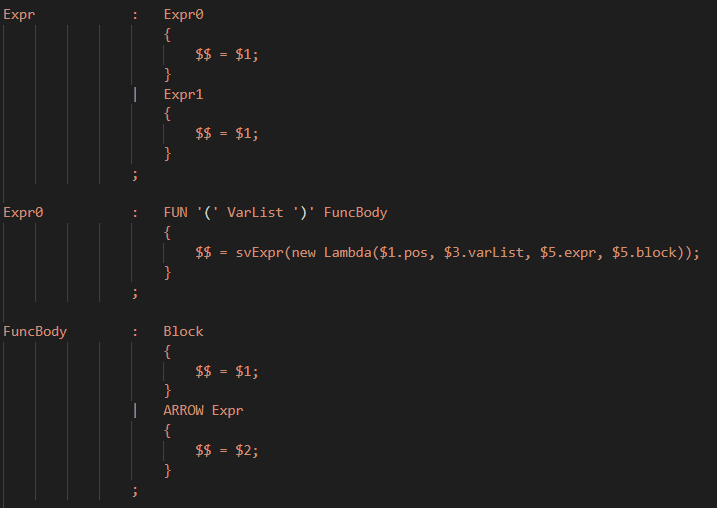






2°添加Lambda表达式：

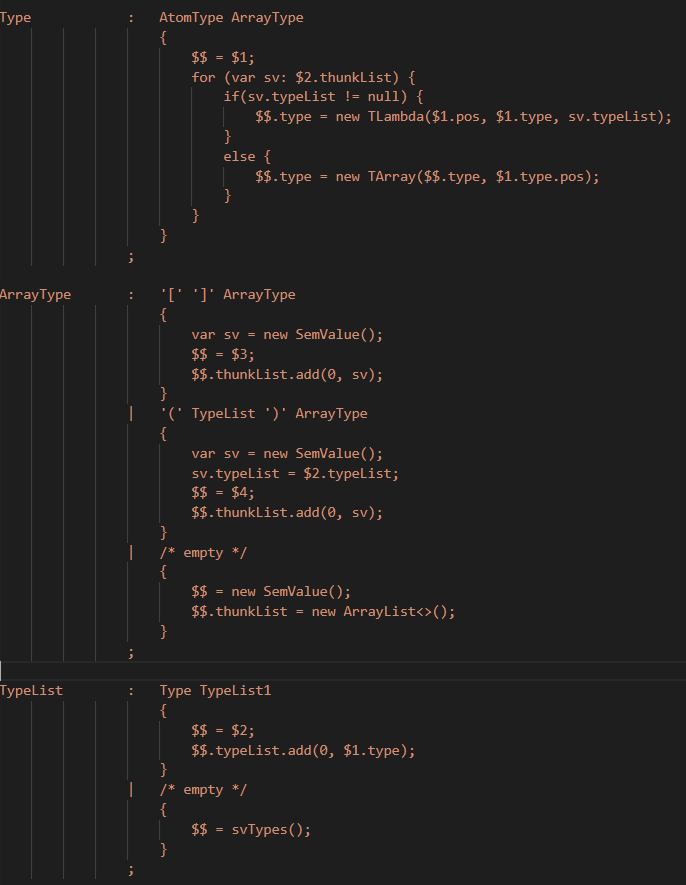
观察Op1~Op7定义的各运算符可发现，Expr1~Expr9是按照优先度从低到高的顺序排列的。在Lambda表达式中，优先度最低的是’=>’，因而创建了一个Expr0用来套入它。接下来分为后面接block和’=> expr’两种情况讨论。同时，为了避免左递归，进行了提取公因式的操作。

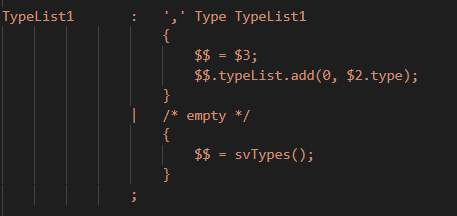


3°添加函数类型



通过观察注意到type的定义与AtomType(‘null’)\*类似，因此可以仿照后者与ExprT的构造方式进行构造。在第一个AtomType后利用一个函数遍历ArrayType的thunklist，从而生成一个type(,type)\*的序列来。TypeList的构造与PA1-A一致，进行LL(1)文法改造之后如下图所示。如果ArrayType的当前位置的typeList不为空，则由此产生一个Lambda表达式，否则产生一个TArray。





**实验中遇到的困难：**

实验中最大的困难在于LL(1)文法的相关改写，如同上面实验所述，TypeList的构造其实早在PA1-A就已经完成，但是当时的构造方式是由如下的上下文无关文法构造而来，存在左递归的问题，设计合适的LL(1)文法是一个很大的问题。

S -> (S) | A

A -> type | type ‘,’ A

其次，是在已有代码的框架中找寻类似结构的代码启示构造的过程，从Op1~Op7的定义找到优先度的关系再安排’=>’的位置，又或者是通过ExprT来构造type序列，这些都需要仔细阅读代码才能发现。

**思考题：**

1.本阶段框架是如何解决空悬else（dangling-else）问题的？

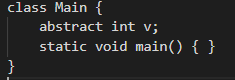
答：若碰到if if else形式的代码块，由于LL(1)文法每次只读取一个Token，当读到第一个if的时候会进到Stmt下面的产生式IF ‘(‘ Expr ‘)’ Stmt中。此时在进行ParseSymbol分析的时候，会将第二个if继续识别成一个Stmt并送到外层Stmt的if类的构造函数中，换言之，第二个if将成为第一个if的附属。此时按照正常的语义，第一个if的block已经结束，从而else将会和第一个if关联起来。空悬else问题得到解决。

2.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。

答：在本阶段的框架中，所有的运算符根据优先度从低到高的顺序分别写在了Op1~Op7中，同一个Op中的表达式优先度是一样的。从顶向下分析的过程是将匹配到的Expr进行拆分，一直拆分到最下层，再进行计算，每一层的计算由于优先度相同而可以进行结合，计算完成后返回到上一层的运算，这里体现出了优先级。对于左右结合，框架的一个做法是用thunkList控制循环是顺序还是倒序来控制是左结合还是右结合，比如buildBinaryExpr函数就是左结合。

3.无论何种错误分析方法，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。

答：abstract1.decaf：



文法分析程序运行到abstract的时候，会进入到FieldList下的ABSTRACT Type Id ‘(‘ VarList ‘)

‘;’ FieldList中。匹配完Type和Id之后，应该匹配’(‘，但是读到了’;’，第一次报错。’;’不属于Begin或End集合，于是被跳过。继续读，读到了static，属于End集合，‘(’分析失败，进到VarList分析，可认为匹配到/\* empty\*/。继续匹配’)’，此时Token还是static，于是同之前一样报错，并且跳过。然后进到最后的FieldList匹配，正确结束，整个过程报错两次。然而按照正常的逻辑应该只报错一次，即abstract int v是正常的int v声明前多写了一个abstract而已。这个问题本身是不可复原的，也从来没有一个正确的标准。所谓的正常理解也只是另一个角度的观点而已，误报只是在乎是否合乎人们的正常逻辑，而已，因此无法避免。