PA2 实验报告

工作内容

实验部分

PA2 实验要求对抽象语法树进行语义分析，具体来说需要做这些事情：

1. 为类添加抽象的属性，判断抽象方法是否被重载、是否实例化一个抽象类
2. 局部类型推导，用递归的方式实现
3. 添加函数类型，实现 Lambda 函数的返回类型推导
4. 实现类型的上下界的计算
5. 添加 Lambda 作用域，修改变量声明时的符号查找逻辑
6. 修改调用函数时的检查逻辑

抽象类

实现了要求的错误判断：第一个判断是，抽象方法必须在抽象类中；第二个是具体类需要重载所有的抽象类，只要对每个类都遍历一遍继承关系，然后统计各个方法的情况即可；第三个则是在 new 的时候判断一下 abstract ，比较容易实现。

局部类型推断

在 Symbol Pass 的时候，var 类型还是 var 类型，然后到 Type Pass 的时候，递归地把 VarDef 的 init 部分的类型求出来，然后针对 void 进行报错。

为了递归地获得类型，对几个遍历 AST 的函数都添加了 Ty 的返回值，这样就可以知道 expression 最终的类型。

First-class functions

函数类型

函数类型本来已经有了，就是 ret + 数个 param 的形式。现在给它添加了 Lambda 的支持，从 Lambda 的声明中生成对应的 Ty。

然后针对函数调用的类型进行相应地检查，因为这个时候调用可能是一个函数类型的变量，所以添加了相应的处理。也针对 void 进行了判断。

Lambda 作用域

类似于函数，函数有一层 scope 用于参数，然后再是内部的 local scope。 Lambda 也是类似的处理办法，把参数放在一个 scope 里，然后 body 是另一个 scope 。并且按照样例要求，也在 lambda 所在的 scope 中注册了一个 lambda@(row,col) 名字的符号，因为有 '@' 所以和正常的不会冲突。

比较麻烦的就是赋值给正在定义的符号的时候的判断。原框架中代码只考虑了简单的情况，现在可能是 Lambda 嵌套了，并且还有不能把外面的变量用作左值的规定。所以我修改了原来的往上搜+Loc在前面的规则，而是在每一层都记录了当前正在定义的变量，然后跳过它和它后面的内容，同时记录在寻找符号的时候是否跨越了 Lambda 边界。

返回类型推导

大概的思路就是把 Lambda 中遇到的所有 return 的类型记录下来，收集之后取上界做 fold 。

寻找上界的方法就是按照 GitBook 上的要求，递归地进行，传进去一个变量纪录着是找上界还是下界。如果是类，上界就 LCA，下界就看共通祖先是否二者其一；如果是函数类型，就对参数求下界，返回值求下界。

函数调用检查

因为调用的函数不再仅限于 VarSel ，所以把一些检查进行了移动，分别处理了样例要求的各种情况，实现不难，但是比较繁琐，有挺多重复代码。

符号表打印

这里涉及到 Lambda 符号的打印的问题。由于 Lambda 和 上一层 Scope 之间可能隔了几层 AST 结点，所以在 print 的时候还是重新遍历了一遍 AST ，找到 Lambda 后再去打印它。这样并不是很优雅，其实可以在所属的 scope 中记录一下，不过想到的时候已经做得差不多了。

问题回答

Q1. 实验框架中是如何实现根据符号名在作用域中查找该符号的？在符号定义和符号引用时的查找有何不同?

用 ScopeStack 实现了一个栈，每个栈是一个 Map，从栈顶往下查找找第一个。进入作用域时压栈，退出作用域时弹栈，然后特殊处理了继承的情况。符号定义时会跳过符号定义本身的符号去找前面，而符号引用的时候正常 lookup 即可。

Q2. 对 AST 的两趟遍历分别做了什么事？分别确定了哪些节点的类型?

第一趟遍历把各个 Scope 建立起来，然后把符号都插入到 Scope 中，第二趟则对代码进行具体的检查，利用第一趟得到的符号信息去找每个符号引用对应的定义。第一趟把类方法、非 var 的变量的类型确定了，第二趟把推导的类型都推导出来。

Q3. 在遍历 AST 时，是如何实现对不同类型的 AST 节点分发相应的处理函数的？请简要分析。

和之前的 PA 一样，就是对 AST 的类型进行 pattern matching ，然后按照类型分别调用对应的处理函数。比如 Expr 就是对 ExprKind 进行匹配，Stmt 就是对 StmtKind 进行匹配。