PA2实验报告

2017011307 张晨

实验原理

抽象类

- 在`methodSymbol`、`classSymbol`里分别记录当前函数/类是否是抽象的。
- 在`classScope`里定义`hasAbstract`函数，以判断类内是否有抽象函数。判断方法是看所有自己和所有父类是否定义的abstract函数在当前classScope里是否有定义。
- 在`typer`的`visitNew`里加一条特判判断不能实例化抽象类。

局部类型推导

- 将`var`视为一种`builtInType`。
- 在`typer`里进行类型推导，并更新`scopeStack`里的`symbol`表，更新与查找`symbol`的过程很像。
- 判断类型推导为`void`的情况，报错。

函数类型

- 调整输出格式，以区分`(int => int) => int`和`int => int => int`

lambda表达式

- 模仿`FormalScope`建立`LambdaScope`，其中嵌套一个`localScope`。模仿`VarSymbol`建立`LambdaSymbol`。
- 将项目原有的判断是否是`methodScope`都改成判断是否是`methodScope`或`lambdaScope`
- 重名问题处理：
  - `namer`里，如果是lambda，则先声明名字再visit初始值，以防止重名。
  - 在`lambdaScope`内嵌的`localScope`里记录这个`lambda`表达式的定义者的名称，在搜索时通过特判去掉以伪造出这个名字未被声明的假象。为了实现这个特判，要重写`lookUpBefore`。
- 判断作用域是否是lambda子作用域，变量是否定义在最内层lambda内部，变量是否是类的成员变量来判断是否是捕获变量，从而对赋值进行报错。
- 返回类型推导
  - 收集所有的`return`语句的类型。
  - 如果最后一个语句块中不含`return`，则再加一个`void`。
  - 按文档中的方法求返回语句。求`ClassType`的上界的方法是看是否存在一个是所有类派生类的类。

函数变量和函数调用

- 为每个expr增加symbol
- 在`typer`的`visitcall`里判断是否`callable`和参数是否匹配，剩余工作迁移至`varsel`

小问题的修复

- 如果在同一个类作用域里定义两个同名函数是“多重定义”错误。
- 一些位置没有检查“之前没有出错”。

符号表的打印

- 增加对`LambdaScope`的支持。

  除此之外，要在`typer`和`namer`里新定义一些visit函数。

思考题

1. 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？

   作用域根据嵌套的顺序被组织在同一个`ScopeStack`中。框架根据符号名，自小到大查询每个作用于里的`Map<Key, Symbol>`，并返回第一个找到的合法`Symbol`。

   定义时使用`findConflict`，

   `FormalScope`和`LocalScope`不能和`FormalScope`、`LocalScope`、`GlobalScope`中已定义名称冲突。

   `ClassScope`和`GlobalScope`不能和任何东西冲突。

   引用时使用`lookUpBefore`，仅查找这个位置之前的定义。

2. 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？

   第一趟遍历定义了各种`Symbol`和`Scope`，在`Scope`里声明符号名并与`Symbol`关联，检查重复定义、没有`Main`等问题。

   第二趟遍历做类型检查，将变量的引用链接到定义时的`Symbol`。

   第一趟确定了`MethodDef`和`LocalVarDef`(扩展后的`var`类型在第二遍确认)的类型，第二趟确定了剩余有类型节点的类型，如`Unary`、`Binary`、`NewArray`、`NewClass`、`This`、`VarSel`、`IndexSel`、`Call`、`ClassCast`、`ClassTest`。

3. 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。

   使用访问者模式，AST中每个节点都重载`accept`函数，在`accept`中调用以自己名字命名的处理函数。`Visitor`可以重载这些以AST节点命名的函数，在想要调用某个节点的处理函数时，调用该节点的`accept`。