Decaf PA2 实验报告

————计62 李祥凡 2016011262

实验总述

实现了README中描述的新的语法特性的语义分析和语义检查，并能正确处理一些典型的语义错误，准确定位错误发生的位置，对Testcases中的所有测试样例，均能得到与标准输出一致的结果。

具体实现

首先将frontend文件夹中的文件替换为PA_1A中对应的文件，并将PA_1A中Tree.java中新定义的语法节点拷贝过来

1.类的浅复制的支持

在OCStmt语法节点中新增一个Location型变量ident_loc，用于保存scopy语句第一个参数的Location

在Visitor类中定义访问scopy语句的方法visitOCStmt

在BuildSym类中不具体实现visitOCStmt，因为scopy语句并没有定义新的符号

在TypeCheck类中给出visitOCStmt的具体实现：

先用lookupBeforeLocation方法在符号表中当前位置之前查找ident(第一个参数)的定义：

1) 若没有找到，则报UndeclVarError; 然后访问expr（第二个参数），若得到expr的类型不是Class, 在expr的位置报BadScopyArgError;

2) 若找到了，判断找到的符号是不是一个Variable：

若不是，则在ident位置报BadScopyError；然后访问expr（第二个参数），若得到expr的类型不是Class, 在expr的位置报BadScopyArgError;

若是，再判断ident对应的符号类型与expr的类型是否一致，若不一致，报BadScopySrcError.

2. sealed的支持

在BuildSym中的visitTopLevel方法中新增一趟对program.classes列表的扫描，若发现某一个class拥有父类，则再在program.classes列表中查找该父类，判断该父类是否是sealed类型（通过判断对应的ClassDef节点的sealed变量是否为null），若是，则报BadSealedInherError.

3. 支持串行条件卫士

在Visitor类中定义新的方法visitGuardStmt和visitIfSubStmt，分别用来扫描条件卫士语句和条件卫士语句中的If分支.

扫描If分支时，对If分支的条件调用框架中原有的cheackTestExpr方法用于检查条件表达式是否是BOOL类型.

4. 支持简单的类型推导

在BaseType中新增基本类型UNKNOWN.

修改BuildSym中的visitAssign方法，判断赋值操作的 '=' 左端是否为VarIdent类型（即是否要进行自动类型推导），如果是的话，在符号表中查找 “var"之后的标识符名称，若在当前作用域下找到了同名的标识符，或当前作用域是局部作用域且函数参数中存在同名的标识符，说明该标识符已被定义，报DeclConflictError。

在Visitor类中定义新的方法visitVarIdent，用来扫描自动类型推导语句的左侧部分（"var" + 标识符 ）

BuildSym中实现visitVarIdent方法，在符号表中声明类型为BaseType.UNKNOWN的变量

修改TypeCheck中的visitAssign方法，先分别访问Assign操作左右两端对应的语法节点， 然后，当Assign的左端为VarIdent类型时，在符号表中将左标识符的类型替换为右端表达式的类型

5. 支持下列数组操作

1）数组初始化常量表达式

修改TypeCheck中的checkBinaryOp方法，在其中加入操作符为“%%”时的语义检查:

若两个操作数中有一个为ERROR类型，则返回以左操作数为元素的数组类型;

若右操作数不是INT类型，报BadArrTimesError;

若左操作数是VOID或UNKNOWN类型，报BadElementError;

若左右操作数均没有错误，返回以左操作数为元素的

数组类型，否则，返回ERROR类型.

2）数组下标动态访问表达式

在Visitor类中定义新的方法visitIndexedDefault方法，用于访问数组下标动态访问表达式;

在TypeCheck中实现visitIndexedDefault方法，调用新增的checkIndexedDefault方法对该表达式进行语义检查，并返回相应的类型

在checkIndexedDefault方法中，先分别对数组名，下标，默认值三者对应的语法节点进行访问，得到它们的确切类型, 接着进行类型检查:

若数组下标不是INT类型，报BadArrIndexError;

若数组名的真实类型不是数组，报BadArrOperArgError; 然后判断默认值的类型，若默认值类型为VOID或UNKNOWN，返回ERROR类型；否则返回默认值的类型

若数组名的真实类型是数组，就判断数组的元素类型与默认值的类型是否一致，若不一致，报BadDefError.最后，返回数组的元素类型.

3) 数组迭代语句

修改Tree.java中的ForeachStmt节点，在其中加入一个Block型变量scopeBlock来便于进行作用域的管理;

在ForeachStmt语法节点构建的时候，初始化一个元素为Tree类型的List，先后将boundVariable(类型 + 变量名），array(数组名)，condition（条件）对应的语法节点放入这个List中，然后，判断stmt是否是一个Block（用instanceof来判断），若是，则将这个Block中的所有子节点加入到List中，若不是，则将这个stmt本身加入到List中。最后，以这个List为参数进行scopeblock的初始化。这样，便实现了数组迭代语句中的所有成员共用一个作用域。

在Visitor类中定义新的方法visitForeachStmt和visitBoundVariable;

BuildSym中的visitForeachStmt方法会以ForeachStmt语法节点中的scopeBlock为参数构造一个LocalScope, 并将这个作用域与scopeBlock相关联，然后访问scopeBlock中的所有语法节点，最后关闭该作用域。

BiuldSym中的visitBoundVariable方法会先访问BoundVariable的type获得类型，然后在符号表中声明该标识符。

TypeCheck中的visitForeachStmt方法会先访问boundVariable, array, condition（scopeBlock中的前三个语法节点），获取它们的实际类型，然后进行类型检查，报告相应的语义错误。

并且，若boundVariable的类型为UNKNOWN，则将其类型修改为数组元素的类型，若发生错误，boundVariable中的标识符类型规定依照README中的描述。

当类型检查结束后，访问scopeblock中剩余的节点（可能是一条或多条stmt)。

实验总结

我花了大概一天半的时间完成了这次实验，感觉比上次简单一点，前两次PA不用关注Location的问题，所以我对语法节点的Location的初始化比较混乱，这次实验也做了一些Location的调整。我的PA_1A的实验架构还算比较清晰明了，这次在总体架构上没有什么调整。

通过本次实验，对语义分析、类型检查的整个过程有了较为清晰、直观的理解，对新增的语言特性的语义分析的实现，很大程度上借鉴了原有的架构，比如说，分析 数组迭代语句 时对作用域的处理，就借鉴了visitBlock中生成局部作用域的方法。

我认为本次实验的难点在于数组迭代语句中的作用域问题。