Semantic Analysis

Overview

Performing the semantic checking involves the following steps:

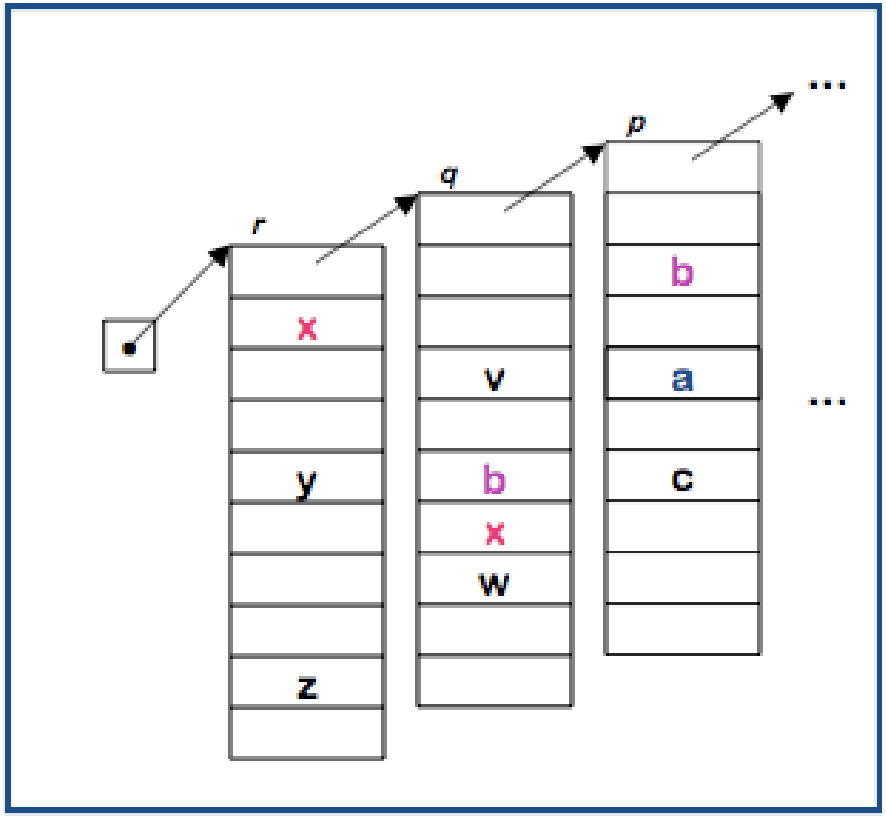
- Build inheritance graph & check to see there are no cycles.
- Build Symbol tables for each class.
- Perform Type checking based on the inheritance tree and the symbol tables.

You may find it easier to perform these steps in 3 different passes of the AST tree or just one.

SymbolTable class

SymbolTable can be used to perform the following:

- Adding Attributes
- Managing and checking scope
- Type Checking




```
B0: procedure b {
        int a, b, c
B1:     {
            int v, b, x, w

B2:         {
                int x, y, z
                ....
            }

B3:         {
                int x, a, v
                ...
            }
            ...
        }
    ...
}
```

semanticAnalyzer.Semant

- The main method that calls the semantic analyzer for the Program.
- The semant() method of the Program class calls

  ClassTable classTable = new ClassTable(classes);
- This installs the basic classes (Object, IO, Int, Bool and Str) look at *semanticAnalyzer.ClassTable* for more information

treeNodes.Program

- Once the basic classes are installed, walk the AST for all the classes, and call semant on those classes

```
for (Class_ c : classes) {
c.semant(new SymbolTable<Info>(), classTable, c);
}
```

- This creates a new scope for each class.

Semant() for a class

The following steps need to be done when the semant method of a class is called:

- Check if the class is present in the inheritance graph.
- Start a new scope for the symbol table " st.enterScope()"
- Add all the variables into the symbol table.
- Call semant for each method in that class.
- Exit the scope "st.exitScope()"

Variables and functions

- Perform similar passes to build the symbol table for functions.
- During this phase you will have to implement the semant method for all the treeNodes.* classes

Finally

- Section 12 of the cool manual would provide you with all the details of type checking for the 3rd pass of the AST tree.