Assignment 4

Due: Wed. Nov 1 at 11:59pm

Implement an ASTVisitor to annotate and type check the abstract syntax tree generated by your parser. The following attribute grammar specifies the type system.

Program ::= name (Declaration | Statement)*

Program.name <= name

Declaration ::= Declaration_Image | Declaration_SourceSink | Declaration_Variable

Declaration_Image ::= name (xSize ySize | ε) Source

REQUIRE: symbolTable.lookupType(name) = ⊥

symbolTable.insert(name, Declaration_Image)

Declaration_Image.Type <= IMAGE

Declaration_SourceSink ::= Type name Source

REQUIRE: symbolTable.lookupType(name) = ⊥

symbolTable.insert(name, Declaration_Image)

Declaration_SourceSink.Type <= Type

Declaration_Variable ::= Type name (Expression | ε)

REQUIRE: symbolTable.lookupType(name) = ⊥

symbolTable.insert(name, Declaration_Image)

Declaration_Variable.Type <= Type

Statement ::= Statement_Assign | Statement_In | Statement_Out

Statement_Assign ::= LHS Expression

REQUIRE: LHS.Type == Expression.Type

StatementAssign.isCartesian == LHS.isCartesian

Statement_In ::= name Source

Statement_In.Declaration ::= name.Declaration

REQUIRE: (name.Declaration != null) & (name.type == Source.type)

Statement_Out ::= name Sink

Statement_In.Declaration ::= name.Declaration

REQUIRE: (name.Declaration != null)

REQUIRE: ((name.Type == INTEGER || name.Type == BOOLEAN) && Sink.Type == SCREEN)

|| (name.Type == IMAGE && (Sink.Type ==FILE || Sink.Type == SCREEN))

Expression ::= Expression_Binary | Expression_BooleanLit | Expression_Conditional |

Expression_FunctionApp | Expression_FunctionAppWithExprArg |
Expression_FunctionAppWithIndexArg | Expression_Ident | Expression_IntLit |
Expression_PixelSelector | Expression_PredefinedName _ Expression_Unary

Expression.Type <= Expression_X.Type

Expression_Binary ::= Expression$_0$ op Expression$_1$

REQUIRE: Expression0.Type == Expression1.Type && Expression_Binary.Type ≠ ⊥

Expression_Binary.type =

if op ∈{EQ, NEQ} then BOOLEAN

else if (op ∈{GE, GT, LT, LE} && Expression0.Type == INTEGER) then BOOLEAN

else if (op ∈{AND, OR}) &&

(Expression0.Type == INTEGER || Expression0.Type == INTEGER)

then Expression0.Type

else if op ∈{DIV, MINUS, MOD, PLUS, POWER, TIMES} && Expression0.Type == INTEGER

then INTEGER

else ⊥

Expression_BooleanLit ::= value

Expression_BooleanLit.Type = BOOLEAN

Expression_Conditional ::= Expression$_{condition}$ Expression$_{true}$ Expression$_{false}$

REQUIRE: Expressioncondition.Type == BOOLEAN &&

Expressiontrue.Type ==Expressionfalse.Type

Expression_Conditional.Type <= Expressiontrue.Type

Expression_FunctionApp ::= Expression_FunctionAppWithExprArg

| Expression_FunctionAppWithIndexArg

Expression_FunctionApp.Type <= Expression_FunctionAppWithXArg.Type

Expression_FunctionAppWithExprArg ::= function Expression

REQUIRE: Expression.Type == INTEGER

Expression_FunctionAppWithExprArg.Type <= INTEGER

Expression_FunctionAppWithIndexArg ::= function Index

Expression_FunctionAppWithExprArg.Type <= INTEGER

Expression_Ident ::= name

Expression_IntLit ::= value

Expression_IntLIt.Type <= INTEGER

Expression_PixelSelector ::= name Index

name.Type <= SymbolTable.lookupType(name)

Expression_PixelSelector.Type <= if name.Type == IMAGE then INTEGER

else if Index == null then name.Type

else ⊥

REQUIRE: Expression_PixelSelector.Type ≠ ⊥

Expression_PredefinedName ::= predefNameKind

Expression_PredefinedName.TYPE <= INTEGER

Expression_Unary ::= op Expression

Expression_Unary.Type <=

let t = Expression.Type in

if op ∈{EXCL} && (t == BOOLEAN || t == INTEGER) then t

else if op {PLUS, MINUS} && t == INTEGER then INTEGER

else ⊥

REQUIRE: Expression_ Unary.Type ≠ ⊥

Index ::= Expression_0 Expression_1

REQUIRE: Expression0.Type == INTEGER && Expression1.Type == INTEGER

Index.isCartesian = Expression0 == KW_R && Expression1 == KW_A

LHS ::= name Index

LHS.Declaration <= symbolTable.lookupDec(name)

LHS.Type <= LHS.Declaration.Type

LHS.isCarteisan <= Index.isCartesian

Sink ::= Sink_Ident | Sink_SCREEN

Sink.Type <= Sink_X.Type

Sink_Ident ::= name

Sink_Ident.Type <= symbolTable.lookupType(name)

REQUIRE: Sink_Ident.Type == FILE

Sink_SCREEN ::= SCREEN

Sink_SCREEN.Type <= SCREEN

Source ::= Source_CommandLineParam | Source_Ident | Source_StringLiteral

Source_CommandLineParam ::= $\text{Expression}_{\text{paramNum}}$

Source_CommandLineParam .Type <= ExpressionparamNum

REQUIRE: Source_CommandLineParam .Type == INTEGER

Source_Ident ::= name

Sink_Ident.Type <= symbolTable.lookupType(name)

REQUIRE: Sink_Ident.Type == FILE || Sink_Ident.Type == URL

Source_StringLiteral ::= fileOrURL

Source_StringLIteral.Type <= if isValidURL(fileOrURL) then URL else FILE

- Classes TypeCheckVisitor.java, TypeCheckTest.java, and TypeUtils.java have been provided. You will need to complete the implementations of TypeCheckVisitor.
- TypeCheckTest.java, as usual, provides a couple of Junit tests that illustrate how the pieces fit together. You will need to modify the AST classes previously provided with fields for attributes. The TypeUtils contains an enum Type. Do not change the names in the enum or reorder them. You probably will not need to modify TypeUtils for this assignment.
- If a type error is discovered, throw a SemanticException. The Token argument should be the firstToken of the AST node where the error was detected. The message should be a helpful error message.
- Note that some Nonterminals, such as Expression, Source, and Sink, which correspond to abstract classes in the AST, have attributes that come directly from their right hand sides. Fields to represent these attributes should generally be declared in the abstract classes so they will be inherited by all subclasses and can be accessed without needing a cast.
- In the specification, symbolTable is a global attribute, and corresponds to a field in the TypeCheckVisitor class. You will need to design an appropriate class or data structure. Note that our language does not have nested scopes, a fact that you can take advantage of. You may, of course, use classes from java.util in your implementation.
- TypeCheckTest.java contains three test cases. testSmallest should pass with the current implementation of the TypeCheckVisitor. The other two will fail in the current implementation with an UnsupportedOperationException, but should pass in the completed assignment.

Turn in a jar file containing your source code for TypeCheckVisitor.java, Parser.java, Scanner.java, all of the AST classes, TypeUtils.java, TypeCheckTest.java, and any classes that you have added.

Your TypeCheckTest will not be graded, but may be looked at in case of academic honesty issues. We will subject your parser to our set of unit tests and your grade will be determined solely by how many tests are passed.

Name your jar file in the following format: ***firstname_lastname_ufid_hw4.jar***

Comments and Suggestions

- Review the lecture on the Visitor Pattern before you begin.
- When a single attribute is computed (like type) it is convenient to let the visit method return it.
- As you implement the project, think about which attributes are synthesized and which are inherited. Would it be possible to incorporate this type checking with parsing?
- Remember that when you submit your assignment, you are attesting that have neither given nor received inappropriate help on the assignment. In this course, all assignments must be your own individual work, including the Scanner and Parser after they have been graded.