

# 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 |  $\epsilon$  ) Source

REQUIRE: symbolTable.lookupType(name) =  $\square$

symbolTable.insert(name, Declaration\_Image)

Declaration\_Image.Type <= IMAGE

REQUIRE if xSize !=  $\epsilon$  then ySize !=  $\epsilon$  && xSize.Type == INTEGER && ySize.type == INTEGER

Declaration\_SourceSink ::= Type name Source

REQUIRE: symbolTable.lookupType(name) =  $\square$

symbolTable.insert(name, Declaration\_SourceSink)

Declaration\_SourceSink.Type <= Type

REQUIRE Source.Type == Declaration\_SourceSink.Type | Source.Type == null

Declaration\_Variable ::= Type name (Expression |  $\epsilon$  )

REQUIRE: symbolTable.lookupType(name) =  $\square$

symbolTable.insert(name, Declaration\_Variable)

Declaration\_Variable.Type <= Type

REQUIRE if (Expression !=  $\epsilon$  ) Declaration\_Variable.Type == Expression.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_Out.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 ::= Expression0 op Expression1

  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 == BOOLEAN)
      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 ::= Expressioncondition Expressiontrue Expressionfalse

  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_FunctionAppWithIndexArg.Type <= INTEGER

```

Expression\_Ident ::= name

Expression\_Ident.Type <= symbolTable.lookupType(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 ::= predefinedNameKind

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<sub>0</sub> Expression<sub>1</sub>

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.isCartesian <= 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 ::= Expression<sub>paramNum</sub>

```
Source_CommandLineParam.Type <= ExpressionparamNum.Type
REQUIRE: Source_CommandLineParam.Type == INTEGER
Source_CommandLineParam.Type <= null
REQUIRE: ExpressionparamNum.Type == INTEGER
```

Source\_Ident ::= name

```
Source_Ident.Type <= symbolTable.lookupType(name)
REQUIRE: Source_Ident.Type == FILE || Source_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\_ufile\_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.