HCMC University of Technology
Faculty of Computer Science & Engineering

Assignment 3

Static Checker for Gifted Students

Author Dr. Nguyen Hua Phung

November 15, 2020

Contents

Assignment 3

version 1.0

1 Specification

This is the extra work for gifted students. Some gifted students who passed this subject but did not pass the extra work yet just submit your work in the extra website. The other gifted students who just study this subject in the first time or have not passed this subject yet or would like to improve your performance in this subject need to submit your work in **both website**. The requirements in the extra work has some constraints that are the same as those in the assignment for normal students but the testcases for these constraints are different.

To complete this assignment, you need to:

- Read carefully the specification of BKIT language
- Download and unzip file **assignment3.zip**
- If you are confident on your Assignment 2, copy your BKIT.g4 into src/main/bkit/parser and your ASTGeneration.py into src/main/bkit/astgen and you can test your Assignment 3 using BKIT input like the first three tests (400-402).
- Otherwise (if you did not complete Assignment 2 or you are not confident on your Assignment 2), don't worry, just input AST as your input of your test (like test 403-405).
- Modify StaticCheck.py in src/main//checker to implement the static checker and modify CheckSuite.py in src/test to implement 100 testcases for testing your code.

2 Static Checker

A static checker plays an important role in modern compilers. It checks in the compiling time if a program conforms to the semantic constraints according to the language specification. In this assignment, you are required to implement a static checker for BKIT language. The input of the checker is in the AST of a BKIT program, i.e. the output of the assignment 2. The output of the checker is nothing if the checked input is correct, otherwise, an error message is released and the static checker will stop immediately

For each semantics error, students should throw corresponding exception given in StaticError.py inside folder src/main/bkit/checker/ to make sure that it will be printed out the same as expected. Every test-case has at most one kind of error. The semantics constraints required to check in this assignment are as follows.

2.1 Redeclared Variable/Function/Parameter:

An identifier must be declared before used. However, the declaration must be unique in its scope. Otherwise, the exception Redeclareds(<kind>,<identifier>) is released, where <kind> is the kind of the <identifier> (Variable/Function/Parameter) in the second declaration. The scope of an identifier (variable, parameter) is informally described as in Section 5 of BKIT specification. All function declarations are in global scope and their scope is the entire program. That means a function can be invoked before its declaration. All function names are unique that means a function name cannot be similar to any other **global** variables, built-in function names or other function names.

2.2 Undeclared Identifier/Function:

The exception "Undeclared(<kind>,<identifier>) is released when there is an <identifier> is used but its declaration cannot be found. The identifier can be a variable or parameter or function. The kind Function is used when the identifier is used as a function name of a function call but there is no such a function declared in the program. The kind Identifier is used in other cases.

2.3 Type Cannot Be Inferred:

BKIT does not require to declare the type of variables or functions but it is able to infer their types. If an identifier is used but its type has not been inferred yet, the exception TypeCannotBeInferred(<statement>) will be released. To infer the type of a variable or a function, BKIT reads the program from the beginning to the end and applies the following rules:

- When a variable is initialized in its declaration, the type of the variable is also the type of the initialized literal.
- The type of an identifier (variable, parameter or function) must be inferred in the first appearance of the identifier's usage of and cannot be changed. If its type cannot be inferred in the first use, the innermost statement containing the first use of the identifier is sent with the exception.
- If an expression can be inferred to some type but some of its components cannot be inferred to any type, the innermost statement containing the type-unresolved component will be associated with the exception. For example, the expression in the right

hand side of the statement **y = a + foo(x)** can be inferred to type int as the result of + is in type int and y, a and the return type of foo can also be inferred to type int, but we cannot infer the type of x, then the exception is raised with the assignment statement.

- A call statement to a type-unresolved function is valid when all its parameter types can be inferred by the corresponding argument types and its return type can be inferred to VoidType. If there exists at least one type-unresolved parameter, the exception is raised with the call statement. Note that if the number of the arguments is not the same as the number of the arguments, the exception concerned in Section 2.4 is raised.

- A function call to a type-unresolved type function is valid if all its parameter types and the return type can be resolved. Otherwise, the innermost statement containing the function call is associated to the exception. Note that if the number of the arguments is not the same as the number of the arguments, the exception concerned in Section 2.5 is raised.

- The types of both sides of an assignment must be the same (i.e. same scalar type or same element type and same list of dimensions for array type) so that if one side has resolved its type, the other side can be inferred to the same type. If both sides cannot be resolved their types, the exception is raised with the assignment.

- For each statement, all variables appear in the statement must have type resolved otherwise the innermost statement containing the type-unresolved variable will be associated with the raised exception.

2.4 Type Mismatch In Statement:

A statement must conform the corresponding type rules for statements, otherwise the exception TypeMismatchInStatement(<statement>) is released.

The type rules for statements are as follows:

- The type of a conditional expression in an **if** statement must be boolean.

- The type of index variable, expression 1 and expression 3 in a **for** statement must be integer while the type of expression 2 is boolean.

- The type of condition expression in **do while** and **while** statements must be boolean.

- For an assignment, the left-hand side (LHS) can be in any type except **VoidType**. The right- hand side (RHS) is the same type as that of the LHS.

- For a call statement <method name>(<args>), the callee must have VoidType as return type. The number of arguments and the number of parameters must be the same. In addition, the type of each argument must be the same as the corresponding parameter.

- For a **return** statement, if the return type of the enclosed function is **VoidType**, the expression in the return statement must be empty. Otherwise, the type of the return expression must be the same as the return type of the function.

2.5 Type Mismatch In Expression:

An expression must conform the type rules for expressions, otherwise the exception TypeMismatchInExpression(<expression>) is released.

The type rules for expression are as follows:

- For an array indexing E[E1]...[En], E must be in array type with n dimensions and E1...En must be integer.

- For a binary and unary expression, the type rules are described in the BKIT specification.

- For a function call <function name>(<args>), the number of the actual parameters must be the same as that of the formal parameters of the corresponding function. The type of each argument must be same as the type of the corresponding parameter.

2.6 No Entry Point:

There must be a function whose name is **main** in a BKIT program. Otherwise, the exception NoEntryPoint() is released.

2.7 Break/Continue not in loop:

A break/continue statement must be inside directly or indirectly a loop otherwise the exception NotInLoop(<statement>) will be released where <statement> is the break/continue statement not in a loop.

2.8 Invalid Array Literal:

An array type in BKIT requires that all elements in an array must be in the same type. An array literal must conform this rule, otherwise, the exception InvalidArrayLiteral(<ArrayLiteral>) is raised where <ArrayLiteral> is the innermost array literal whose elements are in different types.

2.9 Function not return:

A function whose return type is not **VoidType** must return something in every its execution paths. If there exists one path where there is nothing returned, the exception FunctionNotReturn(<function name>) will be released.

2.10 Unreachable function:

A function, except **main** function, must be invoked by another function, otherwise, the exception UnreachableFunction(<function name>) is released.

2.11 Unreachable statement:

An unreachable statement is a statement in a function which the control flow cannot reach. For example, the statement after a return statement is an unreachable statement. The value of an expression is ignored when determining an unreachable statement i.e. the statements in if, for, while statements are assumed to be executed regardless of the value of the condition expression. The exception UnreachableStatement(<statement>) will be released when the <**statement**> is an unreachable statement.

2.12 Index out of range:

An index expression will cause this exception IndexOutOfRange(<ArrayCell>) when there is at least one its indexes is a constant expression and its value is lower than 0 or equal to or greater than the size of the corresponding dimension in the array type. An expression is a constant expression when all its operands are constants (no variables, no call expression, no index expression, etc.). For example x[n][3-4] has the second index which is a constant expression and its value is negative so exception IndexOutOfRange is raised with AST of x[n][3-4].

3 Submissions

This assignment requires you submit 2 files: **StaticCheck.py** containing class **StaticChecker** with the entry method **check**, and **CheckSuite.py** containing 100 testcases.

File **StaticCheck.py** and **CheckSuite.py** must be submitted in **"Assignment 3 Submission"**.

The deadline is announced in course website and that is also the place where you MUST submit your code.

4 Plagiarism

- You must complete the assignment by yourself and do not let your work seen by someone else.

If you violate any requirement, you will be punished by the university rule for plagiarism.

5 Change Log