HCMC University of Technology
Faculty of Computer Science & Engineering

Assignment 3

Static Checker

Author Dr. Nguyen Hua Phung

October 24, 2019

Contents

Assignment 3

version 1.2

After completing this assignment, you will be able to

- explain the principles how a compiler can check some semantic constraints such as type compatibility, scope constraints,... and
- write a medium (300-500LOC) Python program to implement that.

1 Specification

In this assignment, you are required to write a static checker for a program written in MC. To complete this assignment, you need to:

- Read carefully the specification of MC language
- Download and unzip file **assignment3.zip**
- If you are confident on your Assignment 2, copy your MC.g4 into src/main/mc/parser and your ASTGeneration.py into src/main/mc/astgen and you can test your Assignment 3 using MC 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/mc/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 MC language. The input of the checker is in the AST of a MC 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/mc/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> in the second declaration. The scope of an identifier (variable, function, parameter) is informally described as in Section 10 of MC specification.

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.

2.3 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 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** statement must be boolean.
- For a **return** statement, if the return type of the enclosed function is void, the expression in the return statement must be empty. Otherwise, the type of the return expression must be equal to or be coerced to the return type of the function. An exception is an array pointer or array can be returned to an array pointer, i.e., the type of return expression can be in array pointer or array type while the return type of the enclosed function is in array pointer type with the same element type.

2.4 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 subcripting E1[E2], E1 must be in array type or array pointer type and E2 must be integer.

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

- For an assignment expression, the left-hand side (LHS) can be in any type except void, array pointer type and array type. The right- hand side (RHS) is either in the same type as that of the LHS or in the type that can coerce to the LHS type. In MC, just the integer can coerce to the float.

- 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 rule for an assignment is applied to parameter passing where a formal parameter is considered as the LHS and the corresponding actual parameter is the RHS. An exception is an array pointer or array can be passed to an array pointer, i.e., the actual parameter can be in array pointer or array type while the corresponding formal parameter is in array pointer type with the same element type.

2.5 Function not return:

A function that does not return **void** 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.

2.6 Break/Continue not in loop:

A break/continue statement must be inside directly or indirectly a loop otherwise the exception BreakNotInLoop()/ContinueNotInLoop() will be released.

2.7 No Entry Point:

There must be a function **void main ()** somewhere in a MC program. Otherwise, the exception NoEntryPoint() is released.

2.8 Unreachable function:

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

2.9 Not Left Value:

The exception NotLeftValue(<expression>) will be raised when the LHS of assignment operator is not accessible storage, i.e. a variable or an index expression.

The following constraints are requires just for **gifted students (KSTN)**

2.10 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. For instance, the **stmt** in **if (false) stmt** is reachable as the value of the condition expression is ignored. The exception UnreachableStatement(<statement>) will be released when the <**statement**> is an unreachable statement.

2.11 Index out of range:

An index expression will cause this exception IndexOutOfRange(<index expression>) when it is a constant expression and its value is lower than 0 or equal to or greater than the size of the corresponding array type(not array pointer type). An expression is a constant expression when all its operands are constants (no variables, no call expression, no index expression, etc.).

2.12 Uninitialized Variable:

All variables must be explicitly assigned before its use, otherwise the exception UnitializedVariable(<string>) is raised where <string> is the name of the uninitialized variable.

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** must be submitted in **"Assignment 3 Submission"** while file **CheckSuite.py** must be submitted in **"CheckSuite.py Submission"**.

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

4 Peer Assessment

You are required to assess 3 other CheckSuite.py files before the deadline given in **CheckSuite.py Submission**. The rubrics to assess CheckSuite.py is similar to the rubrics was used to assess ASTGenSuite.py. If you don't complete assess 3 other files, your result of this assignment is limited to 70%.

5 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.

6 Change Log