HCMC University of Technology
Faculty of Computer Science & Engineering

Assignment 3

Static Checker

Author

Dr. Nguyen Hua Phung

December 25, 2018

Contents

Assignment 3

version 1.0

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 MP. To complete this assignment, you need to:

- Read carefully the specification of MP language
- Download and unzip file **assignment3.zip**
- If you are confident on your Assignment 2, copy your MP.g4 into src/main/mp/parser and your ASTGeneration.py into src/main/mp/astgen and you can test your Assignment 3 using MP input like the first two tests (400-401).
- 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 402-403).
- Modify StaticCheck.py in src/main/mp/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 MP language. The input of the checker is in the AST of a MP 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/mp/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/Procedure/Parameter:

An identifier must be declared before used. However, the declaration must be unique in its scope. Otherwise, the exception Redeclared(<kind>,<identifier>) is raised, where <kind> is the kind of the <identifier> in the second declaration. The scope of an identifier (variable, function, procedure, parameter) is informally described as in Section 8 of MP specification.

2.2 Undeclared Identifier/Function/Procedure:

The exception Undeclared(<kind>,<identifier>) is raised when there is an <identifier> is used but its declaration cannot be found. The <kind> is Identifier when the <identifier> is used as a variable in an expression. The <kind> is Procedure when the <identifier> is used as a procedure name in a call statement. In the case that the <identifier> is used as a function name in a call expression, the <kind> is Function.

2.3 Type Mismatch In Statement:

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

The type rules for statements are as follows:

- The type of a conditional expression in an **if** statement must be boolean.
- The type of <identifier>, <expression 1> and <expression 2> in a **for** statement must be integer.
- The type of condition expression in **while** statement must be boolean.
- In an assignment statement, there are many sub-assignments whose left-hand side (LHS) can be in any type except 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 MP, just the integer can coerce to the float.
- The **return** statement of a procedure must be without any expression while the **return** statement of a function must be with an expression whose type can be coerced to the return type of the enclosed function. When the return type of a function is an array, the expression of all **return** statement in the function must be in array type whose lower bound, uppper bound and element type must be the same as those of the return type of the function.

- For a procedure call <procedure name>(<args>), the number of the actual parameters must be the same as that of the formal parameters of the corresponding procedure. 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 which can be used as an actual parameter to pass to the corresponding parameter in an array type when their lower bound, upper bound and element type are the same.

2.4 Type Mismatch In Expression:

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

The type rules for expression are as follows:

- For an array subcripting E1[E2], E1 must be in array type and E2 must be integer.
- For a binary and unary expression, the type rules are described in the MP specification.
- The rule type for a function call is appled similar to that of procedure.

2.5 Function not return:

A function must return something in every its execution paths otherwise the exception FunctionNotReturn(<function name>) will be thrown.

2.6 Break/Continue not in loop:

A break/continue statement must be inside directly or indirectly a loop (**for** or **while** statement) otherwise the exception Break/ContinueNotInLoop() will be raised.

2.7 No Entry Point:

There must be a procedure **void main ()** somewhere in a MP program. Otherwise, the exception NoEntryPoint() is raised.

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.

The deadline is announced in course website and that is also the place where you MUST submit your code. The website www.cse.hcmut.edu.vn/onlinejudge is the place you can check your code before submitting in course website.

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