DV1655/6 - Assignment 2

—

Semantic Analysis

Suejb Memeti
Blekinge Institute of Technology

January 31, 2024

1 Introduction

In this assignment you are going to :

1. Construct the symbol table (described in Section 2), and
2. Perform the semantic analysis (type checking) (described in Section 3)

Accordingly, this assignment is split into two parts, (1) Constructing the Symbol Table and (2) Semantic Analysis.

The goal of the first part is to traverse the AST to construct the symbol table, which is a data structure that contains information related to identifiers, such as name, type, and scope.

The goal of the second part of this assignment is to traverse the AST and use the information stored in the symbol table to perform the semantic analysis.

The examination of this project is done through demonstrations during the lab sessions. It is expected that you submit the source code in Canvas. The source code should be compressed in a zip/tar file.

The solution should be implemented using C or C++, and it should be compiled and executed correctly on a Unix-based operating system.

1.1 Laboratory groups

Collaborative work in pairs of two students for this assignment is suggested. Please note that groups larger than two students are not permitted. While individual submissions are accepted, I strongly recommend working in pairs, as the project is designed for groups of two students.

Collaboration and discussions between laboratory groups are encouraged. However, it is essential to maintain the distinction between collaborative learning and plagiarism, as outlined in section 1.3 below.

1.2 Lecture support

In Canvas (the course management page), you should be able to find the Semantic Analysis lecture, which is related to this assignment. In that lecture we introduce the general concepts and theory for semantic analysis. We briefly describe the tree traversal algorithms and their role with respect to the construction of the symbol table and semantic analysis. We also describe the role of the symbol table and semantic analysis. Furthermore, we provide practical examples of how the symbol table is created and how it can be used for semantic analysis.

1.3 Plagiarism

We emphasize the importance of academic integrity. Any work that is not your own must be appropriately referenced. Failure to do so will be considered plagiarism and reported to the university disciplinary board.

2 Part 1: Constructing the symbol table

2.1 Problem Description

The goal of this part of the assignment is to traverse the AST to construct the symbol table (ST). The ST is a data structure for storing information about the identifiers, such as the name, type, and scope. Note that, in MiniJava, we have three types of identifiers: *class identifiers*, *method identifiers*, and *variable identifiers*. Each of those types represent the scope of such identifiers.

It is common for an identifier to have different type or value on a different scope in the program. So, the implementation of the symbol table needs to keep track of the current scope when analyzing our program. It needs to provide functionality to enter and exit scopes, add new identifiers to the symbol table, as well as functionality to lookup a given identifier.

Every time we start a new traversal of the AST, we need to reset the ST, hence we need to provide such functionality. For more implementation details on how we can construct a symbol table I refer you to the Semantic Analysis lecture.

2.1.1 Tasks to Complete

You are supposed to do the following:

- Write a tree traversal algorithm, which will visit all the nodes of the AST. Note that we already have a tree traversal algorithm implemented for the print-tree function, which basically visits each node of the tree and prints their name and type.
- Design a data-structure for the symbol table, which is able to store all the necessary information for all identifiers.
- Perform a single left-to-right tree traversal and populate the symbol table.
- For debugging and demonstration purposes, write a method that prints the symbol table (the name and the type for each record in the symbol table). You may as well generate a graphviz (.dot) file if you want to visualize the symbol table, similarly to what we do for the AST. In that case, a command `make st` to generate the pdf version of the symbol table is expected in the Makefile.

2.1.2 Recommended approach

The recommended approach for constructing the symbol table is as follows:

- For each type of the identifiers (e.g., classes, methods, and variables) use a different type of record in the symbol table. A record represents an identifier in the symbol table. I therefore suggest a hierarchy of record types.
- The symbol table should be able to deal with the scopes of the identifiers. For instance, the variables defined inside a class could be accessed from anywhere in the class, whereas identifiers defined inside a method could only be accessed from that method. Note that in MiniJava, we can not define variables inside of a block-statement scope, hence we do not consider the cases where we define new variables inside the body of the *if-else* and *while* statements (in such cases we would need a nested level of scopes).
- Note that the symbol table can be constructed using a single left-to-right traversal of the AST.
- Note that the construction of the ST is concerned only with the declarations of identifiers (including variables, methods, and classes) in our program. The way those identifiers will be used is considered in the semantic analysis phase.
- Write a method for printing the symbol table and use it to verify the correctness of your solution. An example of the symbol table for one of the programs in the set of valid Java classes is provided in slide 14 of the Semantic Analysis lecture.

3 Part 2: Semantic analysis

3.1 Problem Description

The semantic analysis verifies the semantic correctness of the program. The main task of the semantic analysis include type checking, checking if identifiers are declared before they are used, checking for duplicate identifiers, and others.

The semantic analysis uses the information in the symbol table to perform the above tasks. It is expected for the semantic analysis to be implemented as a separate phase of the compiler, which means that you need to perform another traversal of the AST.

3.1.1 Tasks to Complete

You are supposed to do the following:

- Write another tree traversal function for performing the semantic analysis.
- Note that you may need to perform multiple tree traversals for different aspects of the semantic analysis. Also, note that the semantic analysis requires evaluation of types of sub-trees in a post-order fashion.
- Verify that all identifiers are declared and that there are no duplicate identifiers.
- Perform type checking for expressions, statements, methods, method calls, array accesses, and so on.
- In comparison to the syntax analysis phase, where the compiler stops after the first syntax error is found, during semantic analysis, it is expected that the compiler handles all the semantic errors. I.e., the analysis phase should not stop at the first semantic error.

3.1.2 Recommended approach

For this assignment, I suggest to start with the functionality that checks for undeclared identifiers, as this is the simplest case of semantic analysis. For example, start checking if a variable of a method is declared inside the same class. Then you may extend the functionality to check for function calls outside the current class, and so on.

Then, you may start performing the type checking analysis. You may start with the simplest cases and then move to the more complex parts. For instance,

- Expressions: the type of all terminals inside an expression should be the same. For example, in `a+b;` both `a` and `b` should be integers. `10 + false` should report a semantic error.
- Statements: check that the left-hand-side and right-hand-side of assignment statements are of the same type; check that the condition inside `if-statements` and `while-statements` is `Boolean` type; check that the type of the expression inside the print-statement is of type `integer`;. For example, in `a=b;` type of `a` and `b` should be the same.
- Method declaration: check that the return type of the method is in accordance with the declared return type. In `int a(int c)... return b;` the type of return expression `b` should be the same as the type of the method declaration (int).
- Method calls: verify that the number of parameters and the type of parameters inside a method call matches the number and type of parameters of the method declaration. For example, in `a(x);` the type of `x` and `c` should be the same as in their definition (see previous item).
- Array access: check that the expression inside the `int [ ]` is of type integer; verify that the left hand side of the expression that has the `.length` member is an array of integer. For example, in `b[a]; b.length;` the type of `a` must be integer; the type of `b` must be `int[]`.
- and more ...

4 Testing

You may utilize the test script provided as part of Assignment 1. To execute all semantic analysis test cases, apply the `-semantic` flag. This flag prompts the script to execute test cases specifically designed to identify semantic errors, which are located in the `test_files/semantic_errors` directory.

Additionally, the `test_files/valid` directory comprises various valid Java test classes. When these files are processed through your compiler, no semantic errors should be detected. To further validate the absence of semantic errors, it is advisable to also utilize the `-valid` flag with the test script.

You are advised to thoroughly examine the files within the aforementioned directories to understand the types of semantic checks required. It is important to note that the provided test classes may not cover all possible scenarios. Therefore, you are encouraged to extend this collection with your own test classes, aimed at testing a broader range of functionalities.

5 Examination

During the demonstration:

- Your compiler should be easily compiled using the Makefile.
- You should be able to explain briefly how you have constructed the symbol table and how specific type checking steps work. For example, "Show me the code related to type checking of return statements?". Show me the code that does ...
- Have a set of valid test classes and invalid test classes ready. During the demonstration, you should use those test classes to demonstrate that the compiler is able to show semantic errors for the invalid examples. **Failing to demonstrate particular functionality will be considered as not-implemented.**
- You should print the symbol table and explain the information and how they relate to the input source code.

5.1 Minimum requirements to complete the assignment

Minimum requirements to pass the assignment:

- **Construction of the Symbol Table**
  - This task constitutes the first part of the assignment (Part 1) and involves the creation of a symbol table. The symbol table is a crucial data structure used in the compilation process to store information about the scope and declaration of identifiers (variables, functions, classes, etc.).
- **Identification of Semantic Errors**. Specifically, students must implement functionality within their compiler to:
  - *Detect duplicate identifiers*: This involves identifying cases where a single identifier (e.g., variable name, method name, class name) is declared more than once within the same scope, which is not permissible.
  - *Identify undeclared identifiers*: This requires recognizing instances where an identifier is used without being previously declared or is out of scope.