INFO085 - Compiler
Semantic Analysis

Thibaut Chavet Maxim Henry Maxime Massart

April 19, 2017

1 Introduction

This project is done for the compiler course given by Prof. Geurts during the academic year 2016-2017. We are asked to implement a vsop compiler. The vsop language is defined on the course website.

2 Our compiler

We made improvments during the holidays to keep on improving on good basis. We will present the improvments by the previous impacted parts already done for this project.

2.1 Lexical Analysis

We had an issue with this lexical part. As our C++ bison yyparse() calls yylex itself, when the input of this part is not correct in the vsop formalism, all the tokens are not read until the end, but only until the syntax error. We could have a first parsing phase that would be done twice but for performances, it is not a good option. If you ask for it, we can do it but it is not optimal.

2.2 Syntax Analysis

We checked this part because we tried not to break the previous step so we were not up to date. Now we succeeded in all automated tests. We also refactored the classes and expressions to get Nodes in the nodes/ directory.

2.3 Semantic parsing

For this part, we implemented this step in three phases. First we implemented the Classes checking, then the types checking while retrieving them. After that, we determined the non-redefinition of methods or variables.

2.3.1 Classes checking

In this first part, we used a 2D array that contains the Class ID (string) and the ClassNode associated to it. It is useful to know if this type has been declared in the following part. It also allows to know if all classes extending others are all declared.

2.3.2 Types retrieving and checking

We needed mechanisms to know the closest common parent that we use to determine the output of a dynamic instanciation. This step has the purpose of checking that all types are correctly implemented, and that all expressions have the correct type. This is done using the existing tree of nodes, instead of building a separate scope tree.

2.3.3 Redefinition

We have to determine if the fields are not redefined. The methods can be overwritten if they agree to certain conditions. We implemented this by going through the nodes and asking the parent classes (if needed) to find out if it is a redefinition. We also check the arguments of the methods during this step.

3 Conclusion

We had fun doing this project even though we found very difficult to collaborate as the two steps were to be done and as it depends on the previous test that has also been improved. We also changed our implementation structure quite a lot giving the fact that we could improve the given solution.