CS323 Compliers - Project 2

> *张佳晨*
>
> *11713020 ZHANG, Jiachen*

1. Introduction

Semantic Analysis is the third phase of Compiler. Semantic Analysis makes sure that declarations and statements of program are semantically correct. It is a collection of procedures which is called by parser as and when required by grammar. Both syntax tree of previous phase and symbol table are used to check the consistency of the given code.

Type checking is an important part of semantic analysis where compiler makes sure that each operator has matching operands.

Semantic Analyzer:
It uses syntax tree and symbol table to check whether the given program is semantically consistent with language definition. It gathers type information and stores it in either syntax tree or symbol table. This type information is subsequently used by compiler during intermediate-code generation.

Semantic Errors:
Errors recognized by semantic analyzer are as follows:

- Type mismatch
- Undeclared variables
- Reserved identifier misuse

2. Type Definition

In order to check the type defined in SPL, we need a well defined data structure to record the nessessary information.

2.1 Primary Type

For the primary types, such as int, char and float, the only information which needs to be recorded is the type itself.

2.2 Array Type

For the array type, we need to record not only the primaray type of the array, but the size information for the declared instance. (Because at least for the variable assignment, we need to ensure that, the type and size of the array should be the same).

2.3 Structure Type

For the user defined structure, (as we use name equivanlence,) we need to record the name of the defined struct. Meanwhile, as we need to check the member of the defined structure for the accessibility as well as type checking, so we need to keep the information of the structure memebers, which at least contains the names and the types for each member.

2.4 Variable Type

For the function definition, it a special type as we need to keep the information about the function name, function return value as well as the type information of its arguments. So we need to maintain a list of types to record these kind of information.

3. Type Checking

3.1 Type recording

As we assume all the variables are defined in the same scope, so we can simplify the recording stage into two hash map, one for variable, one for user defined types (struct).

Every time when we iteration the syntax tree and see the variable definition, we can update the variable table, which is, construct the variable type instance to record its name and type informations, (types of return value and arguments for function declaration). And when we meet the struct declaration, we need to construct the structure type instance to record its name and type information of members.

We can also do some semantic checking when recording the type information. For example, we can check the existence of the type to judge whether the identifier is declared with duplication.

3.2 Type retrieving

Every time when the statements is going to use the identifier, we need to check its existence and type information. Thus, we can easily find the variable not defined exception.

3.3 Type checking

As we have introduced the type recording and retrieving, then it's quite easy to implement the type checking. Every time when the syntax tree iterator find the expresson computation or assignment, we need to check the type coorsponding to the identifier and report any error.

4. Conclusion

In this project, the most challenging steps is to design the type recording data structure to record the key information for each kind of types in SPL.

In order to reduce the complexity, I have migrated this project from C to C++, thus I can use many well defined data structure and object relations.

I use plenty of assertion in the program so that it can report specific errors and remind me the unfinished parsing and the semantic error I've not handled yet.

Reference

1. Semantic Analysis in Compiler Design