



Compiler Construction

Lecture Notes

Semantic Analysis and Symbol Table

Semantics

Semantics of a language provide meaning to its constructs. Semantics help interpret symbols, their types, and their relations with each other. Semantic analysis decides whether the syntax structure of the source program derives any meaning or not.

CFG + semantic rules = Syntax Directed Definitions

For example: `int a = "value";`

Is not an error in lexical and syntax analysis phases, as it is lexically and structurally correct, but it should generate a semantic error as the type of the assignment differs. These rules are set by the grammar of the language and evaluated in semantic analysis. Semantic analysis performs following tasks:

- Scope resolution
- Type checking
- Array-bound checking

Semantic Errors

Some of the semantics errors that the semantic analyzer recognizes:

- Type mismatch
- Undeclared variable
- Reserved identifier misuse.
- Multiple declaration of variable in a scope.
- Accessing an out of scope variable.
- Actual and formal parameter mismatch.

Attribute Grammar

Attribute grammar is a special context-free grammar with some additional attributes appended to one or more of its non-terminals to provide context-sensitive information. Each attribute has well-defined domain of values, such as integer, float, character, string, and expressions.

Attribute grammar is a medium to provide semantics to the context-free grammar and it can help specify the syntax and semantics of a programming language. Attribute grammar (when viewed as a parse-tree) can pass values or information among the nodes of a tree.

Example:

$$E \rightarrow E + T \{ E.value = E.value + T.value \}$$

The right part of the CFG contains the semantic rules that specify how the grammar should be interpreted. Here, the values of non-terminals E and T are added together and the result is copied to the non-terminal E.

Semantic attributes may be assigned to their values from their domain at the time of parsing and evaluated at the time of assignment or conditions.

Program Semantics and Symbol Table

Information from the symbol-table entry is needed for semantic analysis and code generation.

$position = initial + rate * 60$

For the above expressions, the tokens generated will be:

$\langle id, 1 \rangle \langle = \rangle \langle id, 2 \rangle \langle + \rangle \langle id, 3 \rangle \langle * \rangle \langle int_const, 4 \rangle$

Here position for example, is a lexeme that would be mapped into a token $\langle id, 1 \rangle$ where id is an abstract symbol standing for identifier and 1 points to the symbol-table entry for position.

The symbol-table entries generated for the above expression should be:

Table 1: Symbol-table for the expression above

1	position	float	...
2	initial	float	...
3	rate	int	...
4	60	inst_const	...
...			

The semantic analyzer uses the information in the symbol table to check the source program for semantic consistency with the language definition. It also gathers type information and saves it in the symbol table, for subsequent use during intermediate-code generation.

An important part of semantic analysis is type checking, where the compiler checks that each operator has matching operands. For example, many programming language definitions require an array index to be an integer; the compiler must report an error if a floating-point number is used to index an array.

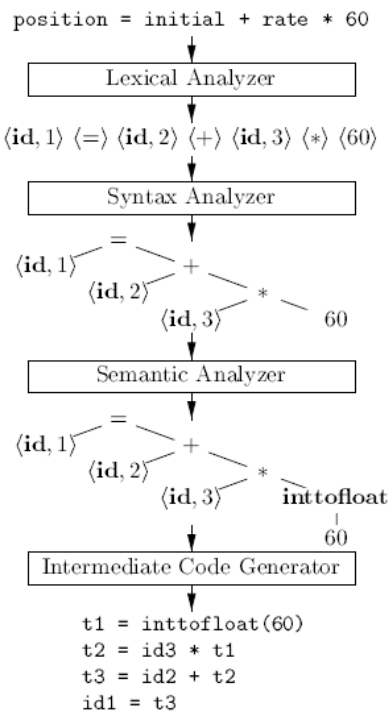


Figure 1: Translation of the expression code above in the front-end

Symbol-Table Management:

Compiler records the variable names used in the source program and collect information about various attributes of each name. These attributes may provide information about the storage allocated for a name, its type, its scope, and in the case of procedure names, such things as the number and types of its arguments, the method of passing each argument (for example, by value or by reference), and the type returned.

The symbol table is a data structure containing a record for each variable name, with fields for the attributes of the name. The data structure should be designed to allow the compiler to find the record for each name quickly and to store or retrieve data from that record quickly.

Symbol Table

Symbol tables are data structures that hold information about identifiers. Information is put into the symbol table when the declaration of an identifier is analyzed. A semantic action gets information from the symbol table when the identifier is subsequently used, for example, as a factor in an expression.

The information is collected incrementally by the analysis phases of a compiler and used by the synthesis phases to generate the target code. Entries in the symbol table contain information about an identifier such as its lexeme, its type, its scope, and any other relevant information.

The scope of a declaration is the portion of a program to which the declaration applies. You can implement scopes by setting up a separate symbol table for each scope. A program block with declarations will have its own symbol table with an entry for each declaration in the block. This approach also works for other constructs that set up scopes; for example, a class would have its own table, with an entry for each field and method.

To implement Symbol Table, we can create a new class `SymbolTable` and use a `HashTable` to store the symbols. Each entry in the table is a key-value pair, with key as the token lexeme and value as the symbol class object. The symbol class object contains the information about the lexeme including its type, scope and any other relevant information.

The `SymbolTable` class also contains methods to put method to insert a new symbol into the table and get method to retrieve a symbol from the table.

The Use of Symbol Tables

The role of a symbol table is to pass information from declarations to uses. A semantic action “puts” information about identifier x into the symbol table, when the declaration of x is analyzed. Subsequently, a semantic action associated with a production such as $\text{factor} \rightarrow \text{id}$ “gets” information about the identifier from the symbol table.