Semantic Analysis

Semantic Analysis

- **Semantic Analysis** is the third phase of Compiler design.
- 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.

cont..

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

cont..

For example:

```
int a = “value”;
```

should not issue an error in lexical and syntax analysis phase, 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.

Functions of Semantic Analysis:

1. **Type Checking** –

   Ensures that data types are used in a way consistent with their definition.
2. **Label Checking** –

   A program should contain labels references.
3. **Flow Control Check** –

   Keeps a check that control structures are used in a proper manner.(example: no break statement outside a loop)

```
float x = 10.1;
```

```
float y = x*30;
```

In the above example integer 30 will be typecasted to float 30.0 before multiplication, by semantic analyzer.

Tasks in semantic analysis

The following tasks should be performed in semantic analysis:

- Check the meaning of the source code

    - variables are used correctly

    - variable declarations

    - incorrect function call

    - syntax structure (if else, switch case, regular expression)

Tasks in semantic analysis

- Check meaning of labels

  Flow, controles, loops and switches

- Check data type, constraints, expressions and assignments
- Check number of parameters in function call

● Semantic analysis give meaning for each and every elements like tokens and syntax structures

Semantic Errors

some of the semantics errors that the semantic analyzer is expected to recognize:

- 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 form of context-free grammar where some additional information (attributes) are appended to one or more of its non-terminals in order 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 → 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.

Static and Dynamic Semantics

1. **Static Semantics –**

   It is named so because of the fact that these are checked at compile time. The static semantics and meaning of program during execution, are indirectly related.

2. **Dynamic Semantic Analysis –**

   It defines the meaning of different units of program like expressions and statements. These are checked at runtime unlike static semantics.

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. Based on the way the attributes get their values, they can be broadly divided into two categories :

synthesized attributes and **inherited attributes**.

Synthesized attributes

These attributes get values from the attribute values of their child nodes.

To illustrate, assume the following production:

$$S \rightarrow ABC$$

If S is taking values from its child nodes (A,B,C), then it is said to be a synthesized attribute, as the values of ABC are synthesized to S.

As in our previous example ($E \rightarrow E + T$), the parent node E gets its value from its child node. Synthesized attributes never take values from their parent nodes or any sibling nodes.

Inherited attributes

In contrast to synthesized attributes, inherited attributes can take values from parent and/or siblings. As in the following production,

$$S \rightarrow ABC$$

A can get values from S, B and C. B can take values from S, A, and C. Likewise, C can take values from S, A, and B.