Semantic Analysis

1 Semantic Analysis

- Ensure that the program has a well defined meaning
- Verify properties aren't caught during earlier phases e.g.
  - Variables are declared before they are used
  - Expressions have the right types
  - Number and types of arguments of a procedure call agree with the procedure declaration
- To check these properties
  - We need to check context conditions imposed by the language specification
  - They can't be checked by context free grammars
  - We enhance our grammars: attribute grammars

2 Syntax-Directed Definition

These additional data needed for these checks are stored as attributes on the nodes of the parse tree, forming an annotated parse tree.

An attribute may represent any quantity, e.g. type, string, memory allocation ...

Definition: Syntax-Directed Definition

A context free grammar in which:

- Every grammar symbol has an associated set of attributes
- Every production has an associated set of semantic rules for computing the attribute values

3 Syntax-directed translation

Definition: Syntax-directed translation

- Process an input string x using a syntax-directed definition
- First build the parse tree for the string x
- For every node N of the parse tree that is labelled by a grammar symbol X denote X.a the attribute a of X at node N
- If X is a non-terminal the value of attribute X.a is determined using a semantic rule associated with some production
- If X is terminal the value of X.a is determined by the lexical analyser

4 Types of attributes

There are two main types of attributes

Definition: Synthesized attribute

An attribute whose value at a node N is defined in terms of attributes at the children of N and at N itself

Definition: Inherited attribute

An attribute whose value at node N is defined in terms of attributes at N's parent, at N's siblings and at N itself

If an attribute a is computed by attributes of parent/siblings and children of N, then add a new inherited attribute b at N and express a as a synthesized attribute (using b)

5 Evaluation of attributes

Before we evaluate an attribute at a node we must first evaluate all the attributes upon which its value depends

Examples:

- Only synthesized attributes (S-attributed SDD)
  - Evaluate first the children, then the node itself
  - All the nodes can be evaluated in any bottom-up order
- Only inherited attributes
  - Evaluate first the parent and all (needed) attributes of siblings and of itself, then evaluate the node itself
  - Evaluate the tree in some top-down order
  - However: never only inherited attributes! (compare terminals vs root)

With both synthesized and inherited attributes there is no guarantee that there exists an order to evaluate the attributes of all nodes

PRODUCTION	SEMANTIC RULES
$A \rightarrow B$	$A.s = B.i$ $B.i = A.s + 1$

These routes are circular as it is impossible to evaluate either A.s or B.i without first evaluating the other

6 Dependency Graph

To check whether there exists an evaluation order for the attributes of an annotated parse tree, we construct the dependency graph

A directed graph $G = (N, E)$ with a set of nodes N and a set of directed edges E, with

- A node for each attribute in the annotated parse tree
- A directed edge $X.c \rightarrow A.b$ whenever in order for compute $A.b$ we have to compute first $X.c$

There exists an evaluation order for all the attributes iff the dependency graph has no directed cycles

7 L-attributed SDD

Some classes of SDD always admit an evaluation order for the attributes

Definition: S-Attributed SDD

- The annotated parse tree has only synthesized attributes
- Evaluate attributes with any bottom-up order

Definition: L-attributed SDD

- Both synthesized and inherited attributes
- In the parse tree, the dependency graph edges between siblings are only ”left to right”

8 Scopes of declarations

Definition: Scope of declaration (for a variable x)

The region of the program, in which the uses of x refer to this declaration of x

- A new declaration for x in the same scope may hide older declarations of x
- A symbol table is a mapping from a name to what the name refers to
- As we run our semantic analysis we continuously update the symbol table with information about what is in scope

In order to keep track of what is visible we implement a stack of tables:

- Each table corresponds to a particular scope
- Stack allows for easy ”enter” and ”exit” operations

Symbol table operations are:

- **Push** scope: enter a new scope
- **Pop** scope: remove a scope, discarding all its declarations
- **Insert** symbol: add a new entry to the current scope
- **Lookup** symbol: find the name this symbol corresponds too

9 Spaghetti stack

- Treat the symbol table as a linked structure of scopes
- Each scope stores a pointer to its parents, but not vice versa
- From any point in the program the symbol table appears to be a stack
- This is called a spaghetti stack