Compilers

3rd year
Spring term

Mick O’Donnell: michael.odonnell@uam.es
Alfonso Ortega: alfonso.ortega@uam.es

Topic 5: Semantic analysis

5.0 Introduction
What is the Semantic Analyser?
- Set of routines used by the Compiler.
- The input is the parse tree, either fully built, or in construction.
- The routines perform semantic error checking:
  - Check variables declared before use
  - Check variables initialised before reference
  - Test type restrictions in expressions and assignments.
  - Check number of arguments in function calls,
  - etc.

Introduction

Semantic analysis

What is the Semantic Analyser?
- Note: Semantic analysis sometimes presented as performed AFTER Syntactic analysis
- In practice, it is usually performed in parallel,
- Routines called as part of syntactic analysis
Output of Semantic Analysis

- The “output” of semantic analysis is the syntactic tree with semantic annotations.
- These annotations serve to:
  - Determine the semantic correctness of the program.
  - Prepare the next step: the code generation
- In some cases, code generation performed as part of semantic analysis
Semantic analysis

Introduction

Semantic Analysis is non-local checking

- Lexical Analysis and Syntactic Analysis produce a “context free” analysis of the input:
  - They check that each local part of the parse tree is well formed.

- Semantic Analysis performs context sensitive checks:
  - a series of non-local checks, to make sure that, e.g., declarations made in one part of the parse tree are respected in other parts of the parse tree

```
int A ;     ….     A = 2 ;
```

Topic 5: Semantic analysis

5.1 Attributes
Semantic analysis

Introduction

Information Propagation

• Semantic Analysis requires that information held in one part of the parse tree is compared to information held in another part.
• E.g., comparing information in a declaration statement to that in an assignment statement.
• To allow comparison, information in one branch of a tree can be ‘propagated up’ to some common node, and then propagated down to the place it is needed:

```
int A ;     ….     A = 2 ;
```

Semantic analysis

Introduction

Attributes on Nonterminals

• To support information movement like this, we extend our grammar to allow nonterminal symbols to carry information.
• Each Nonterminal thus has an associated set of “attributes”, which can be considered slots or fields that hold information about that nonterminal.
• Notation: an attribute of a nonterminal can be referred to as follows:

```
E.val
```

• …referring to the ‘val’ attribute of symbol E
• Similar to how Java accesses the variables of an object.
Attribute Grammar

• An Attribute Grammar is an extension of a context-free grammar which allows attributes to be associated with each nonterminal.
• The attribute grammar also includes a set of rules to assign values to the attributes (next class).

Attributes: informal description

Example

• Consider the following grammar for arithmetic expressions.
• No variables are used in this language.

\[
\begin{align*}
E & \rightarrow E + E \\
E & \rightarrow E - E \\
E & \rightarrow -E \\
E & \rightarrow E \times E \\
E & \rightarrow E / E \\
E & \rightarrow E^E \\
E & \rightarrow (E) \\
E & \rightarrow c
\end{align*}
\]

• \(i\) refers to any identifier, and \(c\) refers to any numeric constant.
• We will deal with two attributes here,

• $\text{val} (v)$: the value of the expression represented by the node

• $\text{type} (t)$: the data type represented by the expression (e.g., integer, float, string, etc.)
Expression: $(3+4)^5$ indicates the work performed by the lexical analyser.
Attributes: informal description

Examples

• Expression: \((3+4)^5\) indicates the work performed by the lexical analyser
Attributes: informal description

Examples

- Expression: (3+4)*5  
  indicates the work performed by the lexical analyser

Example 1: Conclusions

- In this case, to obtain the correct value of the whole expression, the “value” attribute needs to be propagated from the bottom up.
- Also, the “type” of the whole expression needs to be propagated up from the leaves.
Attributes: informal description

Example 2

• Consider the rules for declaring variables:

\[
\begin{align*}
D & \rightarrow T \ L \\
T & \rightarrow \text{int} \\
T & \rightarrow \text{real} \\
L & \rightarrow L, i \\
L & \rightarrow i
\end{align*}
\]

• E.g., \( \text{int } A, B \)

• We shall assume that:
  • \( i \) refers to identifiers.
  • \( D \) represents a declarative sentence (a declaration).
  • \( T \) indicates the type of the variables.
  • \( L \) represents a list of comma-separated identifiers (,)

Attributes: informal description

Example 2

• In the following declaration: \( \text{int } \text{var1, x, y} \)
• In the following declaration: `int var1, x, y`
Example 2

- In the following declaration: `int var1, x, y`
In the example 1, values were propagated up from the bottom of the parse tree.

In Example 2, type information needs to be propagated from the top of a declaration statement down.

So, some cases call for upward propagation of attributes, others call for downward propagation.

The previous examples have illustrated two kinds of attributes:

- **Synthesised attributes:**
  - The attributes of the node are calculated by looking at the attributes of CHILDREN of that node (in other words, the attributes of the LHS are calculated by looking at the attributes of the symbols on the RHS).
  - Example: value attribute in the example or arithmetic expressions:
    - $E \rightarrow E_1 + E_d \{ E\text{.val} = E_1\text{.val} + E_d\text{.val} \}$

- **Inherited attributes:**
  - Attributes of the node are calculated using the the attributes of the PARENT (or sister) node in the tree.
  - An example is the *type* attribute, in the second example:
    - $D \rightarrow TL \{ D\text{.type} = T\text{.type}; L\text{.type} = D\text{.type} \}$
**Attributes: informal description**

**Kinds of attributes, and attribute propagation**

**Inherited attributes:**
- The term “inherited” is used because in the basic case, attributes come from the parent.
- However, the case of receiving attributes from a sister can be re-expressed as a mixed case of synthesis and inheritance.

<table>
<thead>
<tr>
<th>Synthesis: T passes its type information to parent D</th>
<th>Synthesis Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inheritance: D passes its type information to child L</td>
<td>Inheritance Diagram</td>
</tr>
</tbody>
</table>

- For convenience, passing of information from a node directly to its sister is also called ‘inheritance’.