PZ03CX - Language semantics

by T. Pratt and M. Zelkowitz
Prentice Hall, 2001

Section 4.2.1-4.2.3
Semantics overview

Language design has centered on context free grammars (LR(k) grammars).

- Parsing is not an interesting research question any more.

- Problem now is how to decide what a program means (semantics).

- Various approaches have been tried to develop semantic contents of programs.
Semantic approaches

Grammatical models: add extensions to the BNF grammar that defined the language. Given a parse tree for a program, additional information could be extracted from that tree. [e.g., attribute grammars, later]

Operational models: define how programs in the language are executed on a virtual computer. Compare that to the actual execution on a real computer. [Vienna Definition Language of the 1970s]

Applicative models: construct a definition of the function that each program in the language computes. This definition is built up hierarchically through definition of the function computed by each individual program construct. [Denotational semantics]

Axiomatic models: construct a formal logical proof theory to show that a program meets its specifications [to be described later].

Specification model: describe the relationship among the various functions implementing a program
An introduction to attribute grammars

- We will give a brief introduction to attribute grammars to show how these operate.
- Associate a function with each node in the parse tree of a program giving the semantic content of that node.
- An **inherited attribute** is a function that relates nonterminal values in a tree with nonterminal values higher up in the tree (i.e., the functional value for the nonterminals on the right of any rule are a function of the left-hand side nonterminal).
- A **synthesized attribute** is a function that relates the left-hand side nonterminal to values of the right-hand side nonterminals. These attributes pass information up the tree (i.e., were synthesized from the information below in the tree).
## Attributes for “value” of an expression

<table>
<thead>
<tr>
<th>Production</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>E → E+T</td>
<td>value(E₁) = value(E₂) + value(T)</td>
</tr>
<tr>
<td>E → T</td>
<td>value(E) = value(T)</td>
</tr>
<tr>
<td>T → T*P</td>
<td>value(T₁) = value(T₂) * value(P)</td>
</tr>
<tr>
<td>T → P</td>
<td>value(T) = value(P)</td>
</tr>
<tr>
<td>P → I</td>
<td>value(P) = value(I)</td>
</tr>
<tr>
<td>P → (E)</td>
<td>value(P) = value(E)</td>
</tr>
</tbody>
</table>

E₁ is first E in production and E₂ is second E in production

Technique often useful for passing data type information within a parse tree.
Example attributed tree
Use of attribute grammars

First need to develop parse tree. Attribute grammars assume you already have the derivation; it is not a parsing algorithm.

Functions for attributes can be arbitrary. Process to build attributes is mostly manual.

If only have synthesized attributes, and if parsing is LR(k), then attribute grammars can be used automatically along with parsing to generate intermediate code.

- This is how YACC works. Values are computed for each nonterminal.
- As the parse tree is built, information from one nonterminal is passed to another nonterminal higher in the tree.
- When parse is completed, all attribute values have already been computed.