Semantic Rules (see also ch 2)

- Attach attributes to grammar
  - Defined by semantic rules

- Grammar directs the translation
- Attribute: piece of data associated with:
  - a node in the parse tree
  - a nonterminal in the grammar
  - each NT can have zero or more attributes.
  - terminals can get attributes from the lexer
Semantic Rules

• Annotated/decorated parse tree/grammar

• Associated with each production is:
  – semantic rules for evaluating attributes AND/OR
  – semantic rules for producing side-effects (e.g. updating a global variable).
Attributes

- Given: each grammar production $A \rightarrow \alpha$ has associated with it a set of semantic rules of the form $b := f(c_1, c_2, ..., c_k)$ where $f$ is a function
  
  EITHER
  
  - $b$ is a synthesized attribute of $A$ (LHS)
  
  OR
  
  - $b$ is an inherited attribute of one of the grammar symbols on the RHS of the production
Example

- Construct a simple grammar that can represent unsigned numbers.

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Semantic Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Example

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>decl -&gt; datatype list</td>
<td>list.att = datatype.type</td>
</tr>
<tr>
<td>datatype -&gt; int</td>
<td>datatype.type := integer</td>
</tr>
<tr>
<td>datatype -&gt; float</td>
<td>datatype.type := real</td>
</tr>
<tr>
<td>list -&gt; list1, id</td>
<td>list1.att := list.att</td>
</tr>
<tr>
<td></td>
<td>stAdd (id, list.att)</td>
</tr>
<tr>
<td>list -&gt; id</td>
<td>stAdd (id, list.att)</td>
</tr>
</tbody>
</table>
Implementation

• May be achieved with a top-down parse
  – Might add parameters/return values to functions
    `tdNonterminal()`
  – Might add some global data structures

  – Draw the parse tree and determine how the attributes flow!
    • Be aware of initialization!

```c
int foo(int);
foo(foo(foo(foo(9))));
```
Local Variables

Print the type of each local variable
Where is each piece of data available?

```
int x, y, *z;
int arrayX[100];
integer x
integer y
integer pointer z
integer array arrayX
```

What if we had float, char, int?
Practice

• Construct a CFG that allows unsigned integers to be expressed in octal or decimal notation. An octal number is succeeded with an O and a decimal number is succeeded with a D.
  – 18O
  – 10D
Other Examples

- Syntax Tree Printing
  - Using Dot (Graphviz) http://www.graphviz.org

```c
main()
{
    int localVar;
    input(localVar);
}
```

digraph {

    subgraph clusterTD
    {
        program -> externaldefs;
        program -> mainprogram;
        mainprogram -> main;
        mainprogram -> functionstmt;
        main [label="main()"]
        ...
    }
## Semantic Rules

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>program -&gt; externaldefs mainprogram</td>
<td>emit(&quot;program-&gt;externaldefs;’’); emit(&quot;program-&gt;mainprogram;’’);</td>
</tr>
<tr>
<td>mainprogram -&gt; main ( ) functionstmt</td>
<td>emit(&quot;mainprogram-&gt;main( )’’); emit(&quot;mainprogram-&gt;functionstmt’’);</td>
</tr>
</tbody>
</table>

OR

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>program -&gt; externaldefs mainprogram</td>
<td>externaldefs.parent = “program” mainprogram.parent =”program”</td>
</tr>
<tr>
<td>mainprogram -&gt; main ( ) functionstmt</td>
<td>emit(&quot;mainprogram.parent -&gt;mainprogram”); emit(&quot;mainprogram-&gt;main( )”); functionstmt.parent=&quot;mainprogram”;</td>
</tr>
</tbody>
</table>
## Type Checking (ch 6)

### Production and Semantic Rule

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>E → E + T</td>
<td>E.type = higher(E.type, T.type)</td>
</tr>
<tr>
<td>E → T</td>
<td>E.type = T.type</td>
</tr>
<tr>
<td>T → T * F</td>
<td>T.type = higher(T.type, F.type)</td>
</tr>
<tr>
<td>T → F</td>
<td>T.type = F.type</td>
</tr>
<tr>
<td>F → id</td>
<td>F.type =</td>
</tr>
<tr>
<td>A → id</td>
<td></td>
</tr>
<tr>
<td>S → A = E</td>
<td></td>
</tr>
</tbody>
</table>