CSE 504: Compiler Design

Semantic Analysis
Attribute Grammar

Pradipta De
pradipta.de@sunykorea.ac.kr
Current Topic

- Semantic Analysis
  - What is the use of this phase?
- Type Systems
- Attribute Grammar
- Adhoc Syntax directed translation
What is Attribute Grammar

• An attribute grammar consists of a CFG augmented by a set of rules that specify computations
  – Each rule defines one value, or attribute, in terms of values of other attributes
  – The rule associates attributes to a specific grammar symbol
  – An attribute is a value attached to one or more nodes in a parse tree
Attribute Grammar for Signed Binary Number

Grammar for SBN

\[ P = \begin{align*} 
  Number & \rightarrow \ Sign \ List \\
  Sign & \rightarrow \ + \ \\
  & \mid - \\
  List & \rightarrow \ List \ Bit \\
  & \mid Bit \\
  Bit & \rightarrow \ 0 \\
  & \mid 1 
\end{align*} \]

\[ T = \{+,-,0,1\} \]

\[ NT = \{Number, Sign, List, Bit\} \]

\[ S = \{Number\} \]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>value</td>
</tr>
<tr>
<td>Sign</td>
<td>negative</td>
</tr>
<tr>
<td>List</td>
<td>position, value</td>
</tr>
<tr>
<td>Bit</td>
<td>position, value</td>
</tr>
</tbody>
</table>

SBN generates all signed binary numbers -110, -01, +1000, -10001

For each node in the parse tree of the grammar, attributes are selected. In this case, terminal symbols do not need an attribute.
## Attribute Grammar for Signed Binary Number

<table>
<thead>
<tr>
<th>Production</th>
<th>Attribution Rules</th>
</tr>
</thead>
</table>
| 1 $\text{Number} \rightarrow \text{Sign List}$ | \begin{align*}
\text{List}.\text{position} & \leftarrow 0 \\
\text{if Sign}.\text{negative} & \\
\text{then Number}.\text{value} & \leftarrow -\text{List}.\text{value} \\
\text{else Number}.\text{value} & \leftarrow \text{List}.\text{value}
\end{align*} |
| 2 $\text{Sign} \rightarrow +$ | $\text{Sign}.\text{negative} \leftarrow \text{false}$ |
| 3 $\text{Sign} \rightarrow -$ | $\text{Sign}.\text{negative} \leftarrow \text{true}$ |
| 4 $\text{List} \rightarrow \text{Bit}$ | \begin{align*}
\text{Bit}.\text{position} & \leftarrow \text{List}.\text{position} \\
\text{List}.\text{value} & \leftarrow \text{Bit}.\text{value}
\end{align*} |
| 5 $\text{List}_0 \rightarrow \text{List}_1 \text{ Bit}$ | \begin{align*}
\text{List}_1.\text{position} & \leftarrow \text{List}_0.\text{position} + 1 \\
\text{Bit}.\text{position} & \leftarrow \text{List}_0.\text{position} \\
\text{List}_0.\text{value} & \leftarrow \text{List}_1.\text{value} + \text{Bit}.\text{value}
\end{align*} |
| 6 $\text{Bit} \rightarrow 0$ | $\text{Bit}.\text{value} \leftarrow 0$ |
| 7 $\text{Bit} \rightarrow 1$ | $\text{Bit}.\text{value} \leftarrow 2^{\text{Bit}.\text{position}}$ |

- A rule specifies the value of an attribute in terms of constants and other attributes.
- A rule can pass information from LHS to RHS of a production, and vice versa. (Production 4 passes information in both directions)

Given a string in SBN grammar, the rules set $\text{Number}.\text{value}$ to the decimal value.
Attributed Parse Tree

Parse tree with attributes specified for each node ➜ Annotated Parse Tree

(a) Parse Tree for -101
Evaluate Attributed Parse Tree

- Each rule defines a set of dependences → forms an attribute dependence graph

dependence graph captures the flow of values that the evaluator must follow

- Circular dependency possible

- If AG is non-circular, then dependence graph imposes a partial order of evaluation → can find the order using a topological sort of the nodes
Types of attributes

• Synthesized attribute
  – Information flow is bottom up
  – An attribute defined only in terms of attributes of the node, its children, and constants

• Inherited attribute
  – Information flow is top-down or lateral
  – An attribute defined in terms of node’s own attributes and those of its siblings or its parent in the parse tree

In SBN attribute grammar:
value and negative ➔ synthesized attributes
position ➔ inherited attribute
Evaluation Methods for AG

• The rules and attributes in Attribute grammar specify a way to compute in a valid parse tree.

• How to write an evaluator given a AG as input?
  – Can be an adhoc program written by compiler writer
  – Can use an evaluator generator (like scanner or parser generators)
Evaluation Methods for AG

- Techniques to design evaluators that can interpret high level rules and automatically evaluate an instance of a parse tree

- Dynamic Methods
  - Follow the topological sorted order in a parse tree to evaluate

- Rule Based Methods
  - Determine a fixed ordering of evaluation based on compile time analysis

- Oblivious Methods
  - Pick a convenient ordering of evaluation at design time
Evaluation → Manage Circularity

- Dependence graph may have circular dependency → compiler cannot complete the evaluation

- Approaches to handle circularity
  - Avoidance: Restrict the grammar to a class that cannot have circularity
    - eg. Grammar uses only synthesized attributes or constant attributes
  - Evaluation: Use an evaluation method that assigns a default value to an attribute to break circularity

Use of non-circular grammar is the common practice
Attribute Grammar to Infer Expression Types

All attributes are synthesized attributes
This is an example of S-attributed grammar

Simple rule-based evaluation method work
Suitable for bottom-up parsing
 Problems with Attribute Grammar

- Handling non-local information
  - A rule can name only values associated with a grammar symbol that appears in same production
  - When non-local values are required, introduce copy rules → increases size of attribute grammar

- Storage management
  - Use of copy rules multiples the number of attribute instances

- Solution → use a central repository to store global information about attributes
  - Communication through anything other than attribution rules, the implicit dependence embedded in the rules is broken

- Syntax Directed Definitions (SDD) → allows rules with side effects, unlike Attribute Grammars which prohibits rules with side effects
  - Side effects can be writing to central repository, or write to console, or generate machine code directly while parsing