Abstract Syntax Trees

Lecture 14
Sections 5.1 - 5.4

Robb T. Koether

Hampden-Sydney College

Mon, Mar 2, 2009
Abstract Syntax Trees

2. Synthesized Attributes

3. Inherited Attributes

4. Expression Trees

5. Assignment
A parse tree shows the *grammatical* structure of a statement.

It includes all of the grammar symbols (terminals and nonterminals) that were encountered during parsing.
Abstract Syntax Trees

- An abstract syntax tree (AST) shows the *logical* structure of the statement.
- Each node represents an action to be taken by the program or an object to be acted upon.
- The syntax tree may introduce operations that were not in the source code or the grammar.
  - Dereferencing operations.
  - Type-casting operations.
  - Jump statements.
Consider the statement \( a = 2 \times b + c; \)
Syntax Trees vs. Parse Trees

- Our **TreeBuilder** program will “convert” the parse tree into the syntax tree.
- The parse tree never really exists, except insofar as the parser follows its logical order.
- **The TreeBuilder** will simply build the syntax tree from the information obtained by the parser.
- Then the code generator will write the assembly code from the syntax tree.
Recursive descent parsers generally create a single AST for the entire program.

Our parser will generate a separate AST for each statement.

- It will create a list of ASTs.
- This will allow us to generate assembly code as the ASTs are created.
- The trees will be connected both sequentially and through jump statements.
Definition

A syntax-directed definition is a context-free grammar with attributes added to the grammar symbols.

- These attributes are stored in the nodes of the syntax tree.
- Each node has
  - A set of synthesized attributes, and
  - A set of inherited attributes.
Synthesized Attributes

Definition

A synthesized attribute of a grammar symbol is a property that is determined by the attributes of the symbols below it in the parse tree.

- In other words, if $A \rightarrow \alpha$ is a production, then $A$’s synthesized attributes are determined by the attributes of the symbols in $\alpha$. 
Inherited Attributes

Definition
An inherited attribute is a property of a symbol (node) that is determined by its parent node and its siblings in the parse tree.

- In other words, if $\beta$ is symbol on the right side of the production $A \rightarrow \alpha \beta \gamma$, then $\beta$’s inherited attributes are determined by the attributes of $A$ and the other symbols in $\alpha$ and $\gamma$. 
Synthesized Attributes

- If the AST represents a numerical expression, then the value of the root node is determined by the values of the nodes below it in the tree.
- Thus, the value of the root node is a synthesized attribute.
Example

Example (Synthesized Attributes)

- Let the grammar be
  \[ E \rightarrow E + E \mid \text{num} \]
- Then \( E \) derives its value from the \text{num} tokens in the expression.
- This is expressed formally by the rules

\[
\begin{align*}
E.\text{val} & = E_1.\text{val} + E_2.\text{val} \\
E.\text{val} & = \text{num}.\text{lexval}
\end{align*}
\]
Example (Synthesized Attributes)

- The terminals get their values directly from the lexical analyzer.
- For example, a *num* token’s value attribute would be the numerical value of the string of digits in the token.
Example (Synthesized Attributes)

\[
E \cdot \text{val} \\
E_1 \cdot \text{val} \quad + \quad E_2 \cdot \text{val} \\
\text{num} \cdot \text{lexval} \quad 100 \\
\text{num} \cdot \text{lexval} \quad 250
\]
Example (Synthesized Attributes)

```
E.val
  /   \
/     \
E1.val + E2.val
  |     |
  |     |
num.lexval num.lexval
  |     |     |
  |     |     |
100   250
```

Example (Synthesized Attributes)

- `E.val`
- `E1.val` = `num.lexval` = 100
- `E2.val` = `num.lexval` = 250
- `+` is an operator from the lexer
- `E.val` is synthesized
- `E1.val` and `E2.val` are inherited

This example demonstrates how synthesized attributes can be computed based on inherited attributes and operations from the lexer.
Example (Synthesized Attributes)

Abstract Syntax Trees
Synthesized Attributes
Inherited Attributes
Expression Trees
Assignment

Example

Syntax Trees
Robb T. Koether
Example

Example (Synthesized Attributes)

\[ E \text{.val} \]

\[ E_1 \text{.val} \]

\[ + \]

\[ E_2 \text{.val} \]

\[ \text{synthesized from lexer} \]

\[ \text{synthesized from lexer} \]

\[ \text{synthesized from lexer} \]

\[ \text{from lexer} \]

\[ \text{from lexer} \]

\[ \text{num.lexval} \]

\[ 100 \]

\[ 250 \]
Example

Example (Inherited Attributes)

- Consider the grammar for a declaration containing one or more identifiers.

\[
D \rightarrow T \ L \\
L \rightarrow L, \ id \mid id \\
T \rightarrow \text{int} \mid \text{float}
\]

- For example, the declaration might be

```plaintext
float a, b;
```
Example

Example (Inherited Attributes)

- The attribute `float` first appears as the type of the `float` token.
- From there it is passed to the identifiers `a` and `b`. 
Example

Example (Inherited Attributes)

```
D
 / \  
T.type float L.type,
 / \          / 
float id2.type id1.type
```
Example

Example (Inherited Attributes)

```
D

T.type
float

L.type
L.type id
2.type id
1.type float
from lexer

D.T.type
float

L.type
L.type
,

id2.type

float

id1.type
```
Example

Example (Inherited Attributes)

```
D
  └── T.type
      └── float

L.type
  └── ,
      └── id2.type

id1.type
  └── float

from lexer

synthesized
```
Example

Example (Inherited Attributes)

```plaintext
D

T.type float,

T.type synthesized

L.type

L.type id2.type

id1.type inherited

from lexer

float

float

, 

Expression Trees

Inherited Attributes

Assignment

Abstract Syntax Trees

Robb T. Koether

Synthesized Attributes
```
Example

Example (Inherited Attributes)

Abstract Syntax Trees

rob t. koether

abstract syntax trees

synthesized attributes

inherited attributes

expression trees

assignment
Example

Example (Inherited Attributes)

- **T.type**: float
- **L.type**:,
- **id₂.type**: inherited
- **id₁.type**: inherited
- **L.type**: inherited
- **D**: synthesized

- **float**: from lexer

Abstract Syntax Trees
Synthesized Attributes
Inherited Attributes
Expression Trees
Assignment
Some Questions

- In an expression tree, is the type of the expression at the root inherited or is it synthesized?

- Is the type used in an arithmetic operation an inherited attribute or an synthesized attribute of the operator?

- In an assignment statement, is the type assigned by the operator an inherited attribute or a synthesized attribute of the operator?
We will describe how to build an AST for an expression.

We will use `TreeNode` constructors similar to the following.

- `TreeNode(op, left, right)`
  - Join two existing trees, placing `op` at the root node.
- `TreeNode(id, entry)`
  - Create a single-node tree with `id` at the root node.
- `TreeNode(num, value)`
  - Create a single-node tree with `num` at the root node.
Example (Expression Tree)

To construct a tree for the expression

\[ a - 4 + c \]

we do the following:

- \( \text{tree}_1 = \text{new} \ \text{TreeNode}(\text{id}, \ \text{idEntry}_a) \)
- \( \text{tree}_2 = \text{new} \ \text{TreeNode}(\text{num}, \ 4) \)
- \( \text{tree}_3 = \text{new} \ \text{TreeNode}(\text{minus}, \ \text{tree}_1, \ \text{tree}_2) \)
- \( \text{tree}_4 = \text{new} \ \text{TreeNode}(\text{id}, \ \text{idEntry}_c) \)
- \( \text{tree}_5 = \text{new} \ \text{TreeNode}(\text{plus}, \ \text{tree}_3, \ \text{tree}_4) \)
### Example (Expression Tree)

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E \rightarrow E_1 + E_2$</td>
<td>$E\text{.tree} = \textbf{new} \ \text{TreeNode}(\text{plus}, \ E_1\text{.tree}, \ E_2\text{.tree})$;</td>
</tr>
<tr>
<td>$E \rightarrow E_1 - E_2$</td>
<td>$E\text{.tree} = \textbf{new} \ \text{TreeNode}(\text{minus}, \ E_1\text{.tree}, \ E_2\text{.tree})$;</td>
</tr>
<tr>
<td>$E \rightarrow (E_1)$</td>
<td>$E\text{.tree} = \textbf{new} \ \text{TreeNode}(E_1\text{.tree})$;</td>
</tr>
<tr>
<td>$E \rightarrow \text{id}$</td>
<td>$E\text{.tree} = \textbf{new} \ \text{TreeNode}(\text{id}, \ \text{id}\text{.entry})$;</td>
</tr>
<tr>
<td>$E \rightarrow \text{num}$</td>
<td>$E\text{.tree} = \textbf{new} \ \text{TreeNode}(\text{num}, \ \text{num}\text{.val})$;</td>
</tr>
</tbody>
</table>
Assignment

Homework

- Read Sections 5.1 - 5.4, pages 279 - 301.