News & Info

• Scott Hunter
  • Friday 5/20 @ 11:30am SCI III 108

• SoCal Code Camp | UC San Diego, CA 6/25-6/26
  • http://www.socalcodecamp.com/
Administrivia

- Lab 07
  - Solution has been posted
- Lab 08
  - Due Thursday
Abstract Syntax Trees

- So far a parser traces the derivation of a sequence of tokens

- The rest of the compiler needs a structural representation of the program

- **Abstract syntax trees**
  - Like parse trees but ignore some details
  - Abbreviated as AST
Abstract Syntax Tree continued

- Consider the grammar
  \[ E \to \text{int} \mid (E) \mid E + E \]

- And the string
  \[ 5 + (2 + 3) \]

- After lexical analysis (a list of tokens)
  \[ \text{int}_5 \ ' + ' \ '(' \text{int}_2 \ ' + ' \text{int}_3 \ ')' \]

- During parsing we build a parse tree ...
Example of Parse Tree

- Traces the operation of the parser
- Does capture the nesting structure
- But too much info
  - Parentheses
  - Single-successor nodes
Example of AST

- Also captures the nesting structure
- But **abstracts** from the concrete syntax
  - more compact and easier to use
- An important data structure in a compiler
Semantic Actions

- This is what we’ll use to construct ASTs

- Each grammar symbol may have attributes
  - For terminal symbols (lexical tokens) attributes can be calculated by the lexer

- Each production may have an action
  - Written as: \( X \rightarrow Y_1 \ldots Y_n \) \{ action \}
  - That can refer to or compute symbol attributes
Semantic Actions: Example

- Consider the grammar
  \[ E \rightarrow \text{int} \mid E + E \mid (\ E) \]

- For each symbol \( X \) define an attribute \( X.val \)
  - For terminals, \( \text{val} \) is the associated lexeme
  - For non-terminals, \( \text{val} \) is the expression’s value (and is computed from values of subexpressions)

- We annotate the grammar with actions:
  
  \[
  E \rightarrow \text{int} \quad \{ E.val = \text{int.val} \} \\
  \mid E_1 + E_2 \quad \{ E.val = E_1.val + E_2.val \} \\
  \mid (\ E_1) \quad \{ E.val = E_1.val \}
  \]
Semantic Actions: Example continued

- **String:** $5 + (2 + 3)$
- **Tokens:** $\text{int}_5 \, '+' \, (\, \text{int}_2 \, '+' \, \text{int}_3 \, ')'$

**Productions**

$E \rightarrow E_1 + E_2$

$E_1 \rightarrow \text{int}_5$

$E_2 \rightarrow (E_3)$

$E_3 \rightarrow E_4 + E_5$

$E_4 \rightarrow \text{int}_2$

$E_5 \rightarrow \text{int}_3$

**Equations**

$E.\text{val} = E_1.\text{val} + E_2.\text{val}$

$E_1.\text{val} = \text{int}_5.\text{val} = 5$

$E_2.\text{val} = E_3.\text{val}$

$E_3.\text{val} = E_4.\text{val} + E_5.\text{val}$

$E_4.\text{val} = \text{int}_2.\text{val} = 2$

$E_5.\text{val} = \text{int}_3.\text{val} = 3$
Semantic Actions: Notes

- Semantic actions specify a system of equations
  - Order of resolution is not specified

- Example:
  - \( E_3\).val = \( E_4\).val + \( E_5\).val
  - Must compute \( E_4\).val and \( E_5\).val before \( E_3\).val
  - We say that \( E_3\).val depends on \( E_4\).val and \( E_5\).val

- The parser must find the order of evaluation
Dependency Graph

- Each node labeled E has one slot for the val attribute.
- Note the dependencies.
Evaluating Attributes

- An attribute must be computed after all its successors in the dependency graph have been computed
  - In previous example attributes can be computed bottom-up

- Such an order exists when there are no cycles
  - Cyclically defined attributes are not legal
Dependency Graph
Semantic Actions: Notes

- **Synthesized attributes**
  - Calculated from attributes of descendents in the parse tree
  - `E.val` is a synthesized attribute
  - Can always be calculated in a bottom-up order

- **Grammars with only synthesized attributes are called S-attributed grammars**
  - Most common case
Inherited Attributes

- Another kind of attribute
- Calculated from attributes of parent and/or siblings in the parse tree
- Example: a line calculator
A Line Calculator

- Each line contains an expression
  \[ E \rightarrow \text{int} \mid E + E \]
- Each line is terminated with the = sign
  \[ L \rightarrow E = \mid + E = \]
- In second form the value of previous line is used as starting value
- A program is a sequence of lines
  \[ P \rightarrow \varepsilon \mid P \; L \]
Attributes for the Line Calculator

- Each \( E \) has a synthesized attribute \texttt{val} calculated as before.
- Each \( L \) has an attribute \texttt{val}:
  \[
  L \rightarrow E = \{ \text{L.val} = E.val \} \\
  | + E = \{ \text{L.val} = E.val + L.prev \}
  \]

- We need the value of the previous line.
- We use an inherited attribute \texttt{L.prev}.
Attributes for the Line Calculator

- Each \( P \) has a synthesized attribute \( \text{val} \)
  - The value of its last line
    \[
    P \rightarrow \epsilon \quad \{ \ P.\text{val} = 0 \ \}
    \\
    | \ P_1 \ L \quad \{ \ P.\text{val} = L.\text{val}; \ L.\text{prev} = P_1.\text{val} \ \}
    \]
  - Each \( L \) has an inherited attribute \( \text{prev} \)
  - \( L.\text{prev} \) is inherited from sibling \( P_1.\text{val} \)

- Example ...
Example of Inherited Attributes

- All can be computed in depth-first order
- prev inherited
- val synthesized
Example of Inherited Attributes

\[
\begin{align*}
0 & \rightarrow 0 & \rightarrow 5 \\
\rightarrow 0 & \rightarrow E_3 & \rightarrow 5 \\
E_4 & \rightarrow 2 & + \rightarrow E_5 & \rightarrow 3 \\
int_2 & \rightarrow 2 & \rightarrow \text{int}_3 & \rightarrow 3
\end{align*}
\]

- val synthesized
- prev inherited
- All can be computed in depth-first order
Semantic Actions: Notes

- Semantic actions can be used to build ASTs

- And many other things as well
  - Also used for type checking, code generation, ...

- Process is called syntax-directed translation
  - Substantial generalization over CFGs
Constructing An AST

- We first define the AST data type
  - Supplied by us for the project
- Consider an abstract tree type with two constructors:

  \[
  \text{mkleaf}(n) = \begin{array}{c}
  n
  \end{array}
  \]

  \[
  \text{mkplus}(\begin{array}{c}
  T_1
  \end{array}, \begin{array}{c}
  T_2
  \end{array}) = \begin{array}{c}
  \text{PLUS}
  \end{array}
  \]

  \[
  \text{mkplus}(\begin{array}{c}
  T_1
  \end{array}, \begin{array}{c}
  T_2
  \end{array})
  \]

Constructing a Parse Tree

- We define a synthesized attribute \texttt{ast}
  - Values of \texttt{ast} values are ASTs
  - We assume that \texttt{int.lexval} is the value of the integer lexeme
  - Computed using semantic actions

\[
E \rightarrow \text{int} \quad E.\text{ast} = \text{mkleaf}(\text{int.lexval}) \\
| E_1 + E_2 \quad E.\text{ast} = \text{mkplus}(E_1.\text{ast}, E_2.\text{ast}) \\
| (E_1) \quad E.\text{ast} = E_1.\text{ast}
\]
Parse Tree Example

- Consider the string $int_5 \, '+' \, (' \, int_2 \, '+' \, int_3 \, ')$
- A bottom-up evaluation of the ast attribute:
  \[
  E.ast = mkplus(mkleaf(5), \mkplus(mkleaf(2), mkleaf(3)))
  \]