Semantic Analysis

CS 440: Programming Languages and Translators
Lecture 21, Mon Apr 15

Semantic Analysis

- Next compiler phase after parsing
- May come before or as part of constructing internal representation of program
  - AST abstract syntax tree
  - Not Expr -> Term -> Factor -> …, maybe + with two children, 2 and x
  - Construct separately or interleaved with parse
- Internal representation makes it easier to check and process program
  - Enforce static semantic rules (e.g. typechecking)
  - Intermediate code generation (later)

Attributes

- Properties associated with grammar symbols
  - Creation / calculation specified by semantic rules
  - E.g., value of an expression involving only constants
  - or property "Does this expr. involve only constants?"
  - What identifiers/types were declared by this?
- Annotate / decorate parse tree or AST
  - Attribute grammars - formal technique for annotation

Attribute References

- To refer to property of grammar symbol X, use X.property
  - If multiple X's in rule, distinguish using X₁, X₂, …
  - Examples: Constant.value, Variable.type
- Attribute rules attached to each grammar rule build and use attributes.

Synthesized Attribute

- Value depends only the attribute values of the children of this node.
  - Can calculate value of synthesized recursively
  - Rule A -> rhs, a synthetic property of A can use properties of symbols on rhs.
- Works nicely with top-down parsing.
S-Attributed Grammar

- In an S-Attributed grammar, all attributes are synthetic
  - Attribute of node depends only attributes of children.
  - Can calculate attributes along with an LR parse.
    - When you reduce $A \rightarrow \alpha$, you’ve already parsed $\alpha$, so you have the attributes of its symbols.

Example: Value of an expression

\[
\begin{align*}
E & \rightarrow E_1 + T & E_0.val = E_1.val + T.val \\
E & \rightarrow T & E.val = T.val \\
T & \rightarrow T_1 * F & T_0.val = T_1.val * F.val \\
F & \rightarrow \text{Const} & F.val = \text{Const.val} \\
\end{align*}
\]

- For expression $1 + 2 \times 3$, values flow up from leaves to root so we break up larger expressions into smaller ones and get the val property of those.

Expr

\[
\begin{align*}
E_0.val & = E_1.val + T.val = 1 + 6 = 7 \\
E.val & = T.val = 1 \\
T.val & = F.val = 1 \\
F.val & = \text{Const.val} = 1 \\
\text{Const.val} & = 1 \\
T_1.val & = T_2.val * F.val = 2 * 3 = 6 \\
T.val & = F.val = 2 \\
F.val & = \text{Const.val} = 2 \\
\text{Const.val} & = 2 \\
F.val & = \text{Const.val} = 3 \\
\text{Const.val} & = 3 \\
T.val & = F.val = 2 \\
T_1.val & = T_2.val * F.val = 2 * 3 = 6 \\
\text{Const.val} & = 1 \\
F.val & = \text{Const.val} = 1 \\
T.val & = F.val = 1 \\
E.val & = T.val = 1 \\
E_1.val & = E_2.val + T.val = 1 + 6 = 7 \\
\end{align*}
\]

Attribute evaluation order: From more-deeply nested nodes to less-deeply nested nodes (synthetic value)

\[
\begin{align*}
\text{Const} & = 1 \\
\text{Factor} & = \text{Const} = 3 \\
\text{Term} & = \text{Factor} = \text{Const} = 2 \\
\text{Expr} & = \text{Term} = \text{Factor} = \text{Const} = 3 \\
\end{align*}
\]
Inherited Attribute

- Example: Declaration of a variable (`int x;`) adds a type binding to the symbol table.
  - Uses of x (later in the tree) inherit the symbol table.
  - One part of tree builds up symbol table attribute, another uses the symbol table.
- With rule $A \to \alpha B \beta$, inherited property of $B$ can use
  - Inherited properties of $A$, inherited and synthetic properties of symbols in $\alpha$

General Attributes

- Another possibility is to build the entire abstract syntax tree and then traverse it to build/use the attributes.
  - But the traversal after you build the tree follows some part of the traversal used to build the tree.
  - (Duplicate traversal work.)
  - On the other hand, making this a separate traversal means that it can more complicated than (e.g.) depth-first search.

L-Attributed Grammar

- Attributes can be calculated along with an LL parse
  - Attribute of node can be calculated using depth-first search of the tree.
    - In an LL parse, at any point in the parse, we've looked at the nodes from you up to the root and also to your left within the same rule.
  - Inherited and synthetic properties.
  - Strict superset of S-attributed grammars.
- In practice, LR parsers can use some inherited attributes
  - But you have to be careful to make sure all the inherited properties you need (at any point in the parse) have already been calculated.

Example: Symbol Table (Inherited Attribute)

- A symbol table is an inherited attribute: a declaration adds a binding to the table, use of variable inherits table.
- **Notation**: $A.attribute \downarrow = \text{attribute inherited by nonterminal } A$, $A.attribute \uparrow = \text{attribute produced by } A$

  Program $\to$ Block
  
  Block.symtab$\downarrow = \text{empty table}$

  Block $\to$ DeclList StmtList

  DeclList.symtab$\downarrow = Block.symtab \downarrow$ (and top-level block inherits empty symbol table)
  
  StmtList.symtab$\downarrow = DeclList.symtab \uparrow$

  DeclList$_0 \to$ Decl DeclList$_1$

  Decl.symtab$\downarrow = DeclList$_0.symtab \downarrow$
DeclList₁.symtab↓ = Decl.symtab↑
DeclList₀.symtab↑ = DeclList₁.symtab↓

Decl → Type Id
Decl.symtab↑ = Decl.symtab↓ plus binding of identifier and type

**Example: Value of Expression (As An Inherited Attribute)**

- Attribute val = value attached to symbol
- Look at compile-time value determination

\[
\begin{align*}
E \rightarrow T TL & \quad T.val ↓ = E.val ↓ \\
& \quad TL.val ↓ = T.val ↑ \\
& \quad E.val ↑ = TL.val ↑ \\
T₀ \rightarrow T₁ TL & \quad T₁.val ↓ = T₀.val ↓ \\
& \quad TL.val ↓ = T₁.val ↑ \\
& \quad T₀.val ↑ = TL.val ↑ \\
TL \rightarrow ε & \quad TL.val ↑ = TL.val ↓ \\
\end{align*}
\]

\[
\begin{align*}
TL₀ \rightarrow Op T TL₁ & \quad T.val ↓ = 0 \quad \text{-- Op is + or -} \\
& \quad TL₁.val ↓ = \text{function}(TL₀.val ↓, T.val ↑) \quad \text{-- function is addition or subtraction} \\
& \quad TL₀.val ↑ = TL₁.val ↑ \\
\end{align*}
\]

- Parse/eval expression 1 + 2

\[
\begin{align*}
1. & \quad E \rightarrow T TL \quad E.val ↓ = 0, T.val ↓ = E.val ↓ = 0 \\
2. & \quad T \rightarrow Const \quad Const.val = 1, T.val ↑ = 1 \\
& \quad \quad \text{so in (1), } TL.val ↓ = T.val ↑ = 1 \\
3. & \quad TL₀ \rightarrow + T TL₁ \quad TL₀ ↓ = 1 \text{ Op = +, } T.var ↓ = 0 \\
4. & \quad T \rightarrow Const \quad Const.val = 2, T.val ↑ = 2 \\
& \quad \quad \text{so in (3), } T.val ↑ = 2, TL₁.val ↓ = \text{addition}(TL₀ ↓, T.val ↑) = 1+2 = 3 \\
5. & \quad TL \rightarrow ε \quad TL.val ↓ = TL.val ↑ = 3 \\
& \quad \quad \text{so in (1) } TL.val ↑ = 3, \text{ so } E.val ↑ = TL.val ↑ = 3 \\
\end{align*}
\]

(Next example moved to lecture 22.)