Semantic Analysis

CS 440: Programming Languages and Translators, Spring 2020

A. Semantic Analysis

- Next compiler phase after parsing
- May come before or as part of constructing internal representation of program
  - AST abstract syntax tree
    - Not $Expr \rightarrow Term \rightarrow Factor \rightarrow \ldots$
    - Maybe + node 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)

B. Attributes

- Attributes are properties associated with grammar symbols
  - Creation / calculation specified by semantic rules
- **Examples**
  - Value of an expression involving only constants.
  - The property "Does this expr. involve only constants?"
  - The set of identifiers/types were declared here
- 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 X_0, X_1, …
  - Examples: $Constant\,.\,val$ue, $Variable\,.\,type$
- Attribute rules attached to each grammar rule build and use attributes.

Synthesized Attribute

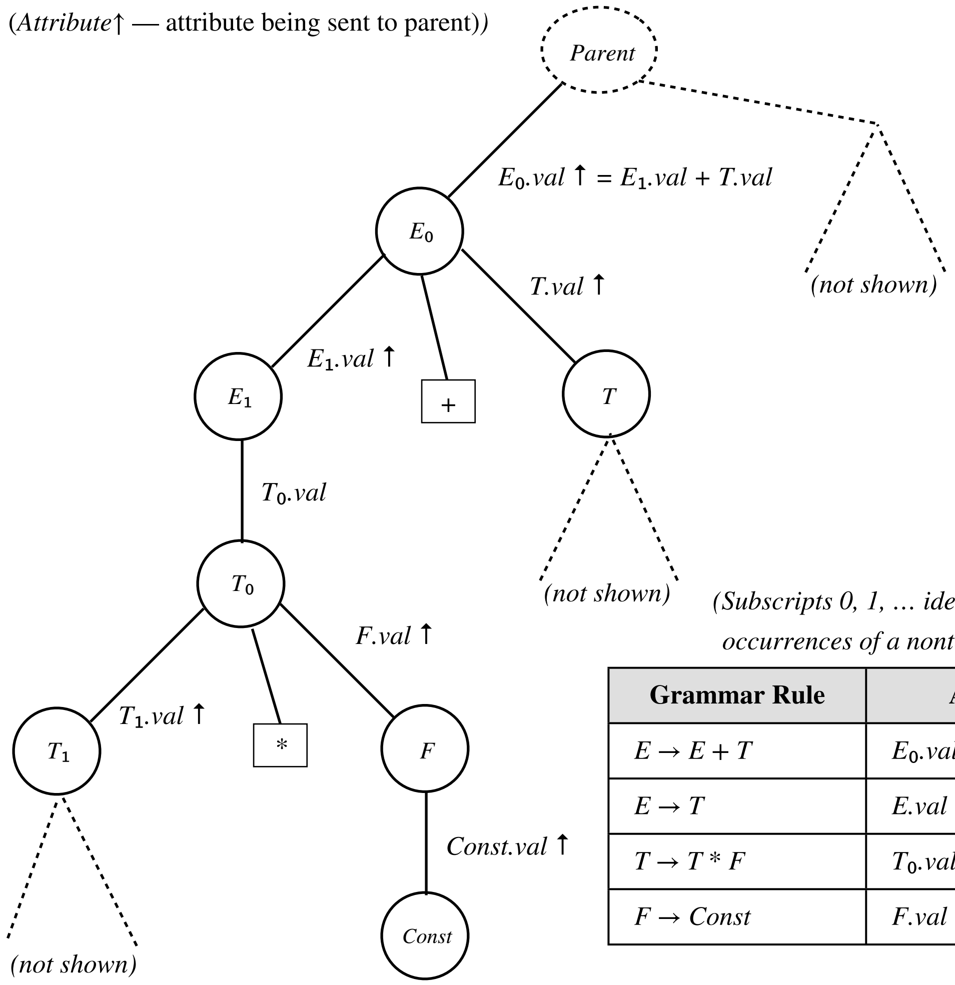
- A **synthetic attribute** is an attribute that doesn't rely on attributes of parent nodes. In terms of an attribute grammar, for a rule $A \rightarrow \alpha$, a synthetic property of A is calculated using only the attributes of symbols in α.

- In terms of the parse tree, for a synthetic attribute, values flow up from leaves toward parents. A node's attribute value is calculated using the values of the attributes of its children.

Example: Value of an expression

- Figure 1 below shows the general parse tree and attribute calculations for a couple of cases of the expression grammar.

(*Attribute*↑ — attribute being sent to parent)*)*



(Subscripts 0, 1, … identify different occurrences of a nonterminal, left-to-right.)

Grammar Rule	Attribute Rule
$E \rightarrow E + T$	$E_0.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T * F$	$T_0.val = T_1.val * F.val$
$F \rightarrow Const$	$F.val = Const.val$

Figure 1: Partial Parse Tree and Attribute Calculation for E

- Figure 2 below shows how the calculation of the value of 1 + 2 * 3 proceeds. (The leading dots with the parse tree nodes indicate depth in the parse tree.)

Parse Tree Node	***Attribute Calculation***
. . . *Const* 3	*Const.val* = 3
. . *F*	*F.val* = *Const.val* = 3
. . . . *Const* 2	*Const.val* = 2
. . . *F*	*F.val* = *Const.val* = 2
. . *T*	*T.val* = *F.val* = 2
. *T*	T_1*.val* = T_2*.val* * *F.val* = 2 * 3 = 6
. . . . *Const* 1	*Const.val* = 1
. . . *F*	*F.val* = *Const.val* = 1
. . *T*	*T.val* = *F.val* = 1
. *E*	*E.val* = *T.val* = 1
E	E_1*.val* = E_2*.val* + *T.val* = 1 + 6 = 7

*Figure 2: Attribute Evaluation for Value of Expression 1 + 2 * 3*

C. S-Attributed Grammar

- In an S-Attributed grammar, all attributes are synthetic.
- With a bottom-up (LR) parse, when you reduce $A \to \alpha$, you've already parsed α, so you have the attributes of the symbols of α.
- With a top-down (LL) parse, we prepare for calculating the attribute when we decide which $A \to \alpha$ rule we're going to use for *A*. When we've finished parsing α, we can calculate a synthetic attribute value then.
  - With a recursive descent parser, you save the attribute results for each recursive call and combine them before returning.

D. Inherited and General Attributes

Inherited Attribute

- An **inherited attribute** is an attribute passed down the parse tree. They're typically used when one part of a parse tree creates a value and we want that value passed on to the rest of the tree.
- **Example:** With a symbol table; each node receives a symbol table and can augment it and pass the result on to the rest of the parse.
- E.g., take a grammar with *Block* → *Decl Stmt* where *Decl* →* `int x;` and *Stmt* →* `x = x + 1;` The node for *Block* receives a symbol table, passes it on to the declaration, which adds an entry for `x`. *Block* then passes this new table to *Stmt* where eventually the assignment can make sure that the uses of `x` are type-correct.

With A → α B β, α doesn't get an inherited value from A because we haven't reduced yet so α doesn't know what parent it has. But if it can create its inherited attribute synthetically, then it can pass that value to B. So in an LR parse, you can use inherited properties that depend on your earlier siblings but not your parent.

L-Attributed Grammar

- In an **L-attributed grammar**, for a rule $A \to \alpha\ B\ \beta$, an inherited property of B can depend on inherited properties of A and inherited and synthetic properties of symbols in α.
- I.e., for a given parse tree node, the value of an inherited attribute can depend not only on the values of the subtrees of the node but also on values passed down from the parent node and also the node's left siblings.
  - (I.e., attribute values can be generated using a depth-first search of a parse tree.)
- Such inherited attributes can be calculated during an LL parse: At any point in the parse, we've looked at the nodes from the current node up to the root and also to the left within the same rule.
- L-attributed grammars are a strict superset of the 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.

- **Notation**: $A.attribute\downarrow$ and $A.attribute\uparrow$ denote attribute values inherited from / produced by A.

Example of Inherited Attribute: Symbol Table

- As we saw earlier, a symbol table is an inherited attribute: a declaration adds a binding to the table, use of variable inherits the table. Below is a more detailed syntax annotated with attribute rules

  $Program \to Block$

  $Block.symtab\downarrow$ = empty table

  $Block \to DeclList\ StmtList$

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

  $StmtList.symtab\downarrow = DeclList.symtab\uparrow$

  If the declarations in DeclList have a local scope, then we're dropping them from the symbol table as we go back to Block's parent, so we don't pass an updated symbol table to the parent. In that case, Block.symtab↑ = Block.symtab↓

  If the declarations in the block are nonlocal, then they stay in the symbol table when Block goes back to its parent, so then Block.symtab↑ = StmtList.symtab↑

  $DeclList_0 \to Decl\ DeclList_1$

  $Decl.symtab\downarrow = DeclList_0.symtab\downarrow$

  $DeclList_1.symtab\downarrow = Decl.symtab\uparrow$

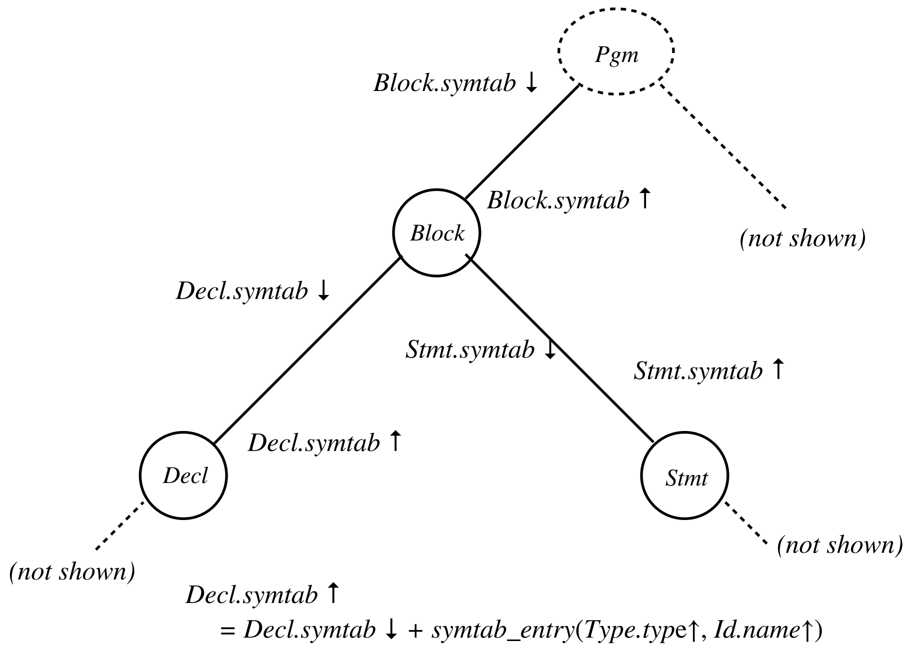
  $DeclList_0.symtab\uparrow = DeclList_1.symtab\uparrow$

  $Decl \to Type\ Id$

  $Decl.symtab\uparrow = Decl.symtab\downarrow + symtab\_entry(Type.type\uparrow, Id.name\uparrow)$

  (*symtab_entry* is a function that builds a symbol table entry)

- Figure 3 below shows the attribute flow attached to a parse tree. The inherited attributes are marked ↓ and are shown to the left of each line; the result attributes are marked ↑ and shown to the right of each line.



(For consistency, attributes being sent to a chid ↓ are shown to the left of a link; attributes being sent to a parent ↑ are shown to the right of a link)

Figure 3: Path of symbol table attribute

E. Adding Attributes to our Parsers

- With our recursive descent parsers, an inherited attribute is passed as an argument to the parser call and is also a result of the call. The input string being parsed is an example: We pass it as input to each parser call and (simulate) getting a updated string as part of the result of a call.
- A synthetic attribute is returned as a result without relying on inherited attributes (other than the input state). A returned parse trees are an example.
  - Roughly, *parser*(*input string* ↓) = Just (*input string* ↑, *parse tree* ↑)

Activity Questions for Lecture 26

1. (Various attribute questions)
   a. What is an attribute and how do we refer to them?
   b. What is an attribute rule? How might we write one for the rule X → A X ?
   c. What is an attribute grammar?
   d. What is a synthesized/synthetic attribute? Give an example.
   e. What is an inherited attribute? Give an example.
   f. Is the parser state a synthesized or inherited attribute?

2. The tree below uses grammar rules

   $S \to T$

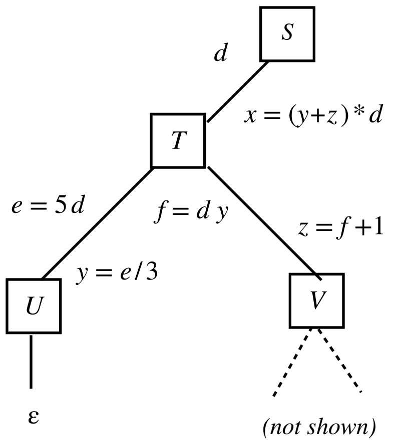
   $T \to U\ V$

   $U \to \varepsilon$

   $V \to$ not shown

   The attribute relationships are added as annotations to the links: Relative to a node, *node.attrib* ↓ is to the left of the link upward to the parent and *node.attrib* ↑ is to the right on the link upward. So for node *T*, it receives *d*, sends *e*, receives *y*, sends *f*, receives *z*, and sends *x*.

   Write out the attribute rules for *T* using *T.attrib* ↓ and ↑ notation. (E.g., instead of *e* = 5**d* we have *U.attrib* ↓ = 5 * *T.attrib* ↓)



Selected Solutions to Activity Questions

1. Omitted

2. Table version of answer:

Letter	***Name***	***equals***	***Equation***
d	*T.attrib↓*		
e	*U.attrib↓*	= *5*d*	*5 * T.attrib↓*
y	*U.attrib↑*	= *e/3*	*U.attrib / 3* typo U.attrib↓ / 3
f	*V.attrib↓*	= *d*y*	*T.attrib↓ * U.attrib↑*
z	*V.attrib↑*	= *f+1*	*V.attrib↓ + 1*
x	*T.attrib↑*	= *(y+z)*d*	*(U.attrib↑ + V.attrib↑) * T.attrib↓*