Semantic Actions

- This is what we will use to construct ASTs

- Each grammar symbol may have attributes
  - An attribute is a property of a programming language construct
  - 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: An Example

- Consider the grammar

$$E \rightarrow \text{int} \mid E + E \mid (E)$$

- For each symbol X define an attribute X.val
  - For terminals, val is the associated lexeme
  - For non-terminals, val is the expression's value (which is computed from values of subexpressions)
- We annotate the grammar with actions:

$E \rightarrow \text{int}$	{ E.val = int.val }
$\mid E_1 + E_2$	{ E.val = E_1.val + E_2.val }
$\mid (E_1)$	{ E.val = E_1.val }

Semantic Actions: An Example (Cont.)

- String: 5 + (2 + 3)
- Tokens: int_5 '+' '(' int_2 '+' int_3 ')'

Productions	Equations
$E \rightarrow E_1 + E_2$	$E.val = E_1.val + E_2.val$
$E_1 \rightarrow int_5$	$E_1.val = int_5.val = 5$
$E_2 \rightarrow (E_3)$	$E_2.val = E_3.val$
$E_3 \rightarrow E_4 + E_5$	$E_3.val = E_4.val + E_5.val$
$E_4 \rightarrow int_2$	$E_4.val = int_2.val = 2$
$E_5 \rightarrow int_3$	$E_5.val = int_3.val = 3$

Semantic Actions: Dependencies

Semantic actions specify a system of equations

- Order of executing the actions is not specified

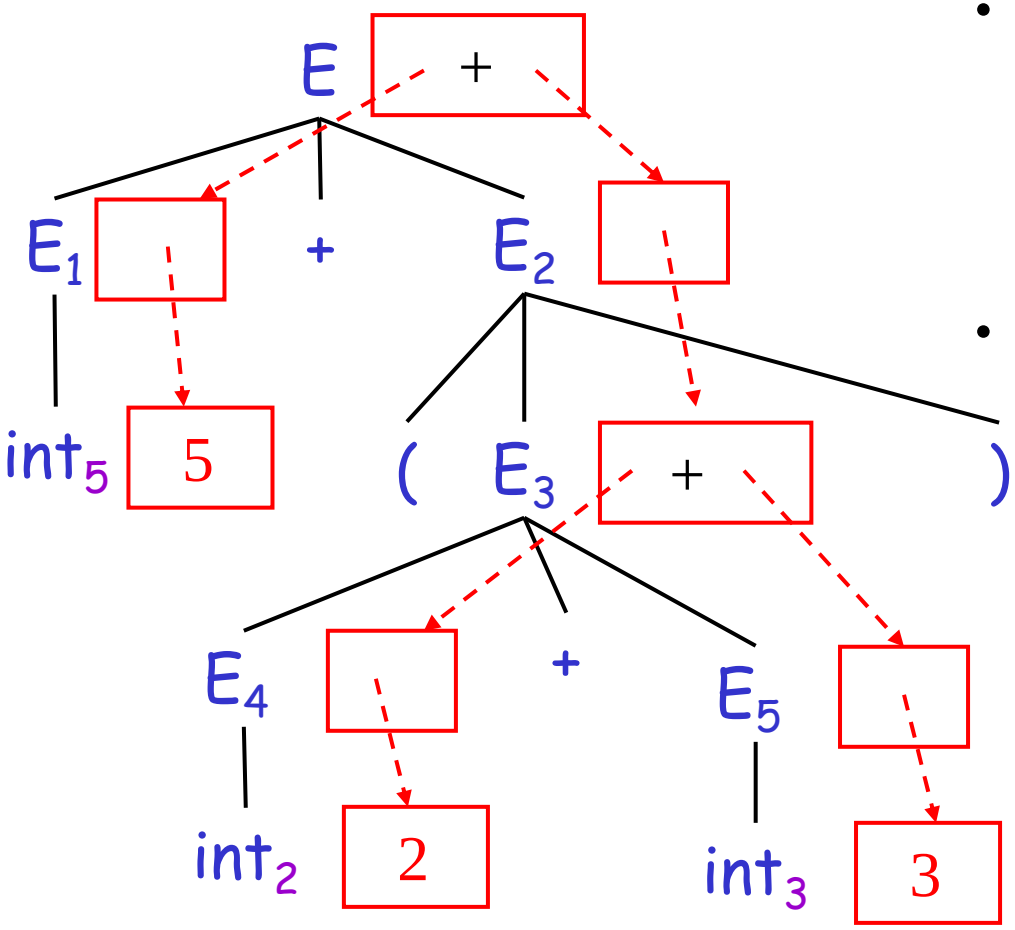
- Example:

  $E_3.val = E_4.val + E_5.val$

  - 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 with a non-terminal E has one slot for its val attribute
- Note the dependencies

Evaluating Attributes

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

- Such an order exists when there are no cycles
  - Cyclically defined attributes are not legal

Semantic Actions: Notes (Cont.)

- 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 frequent kinds of grammars

Inherited Attributes

- Another kind of attributes
- Calculated from attributes of the parent node(s) 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 the second form, the value of evaluation of the 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 val
  - Calculated as before
- Each L has a synthesized attribute val

  $L \rightarrow E$ = { L.val = E.val }

  | + E = { L.val = E.val + L.prev }
- We need the value of the previous line
- We use an inherited attribute L.prev

Attributes for the Line Calculator (Cont.)

- Each P has a synthesized attribute val
  - The value of its last line

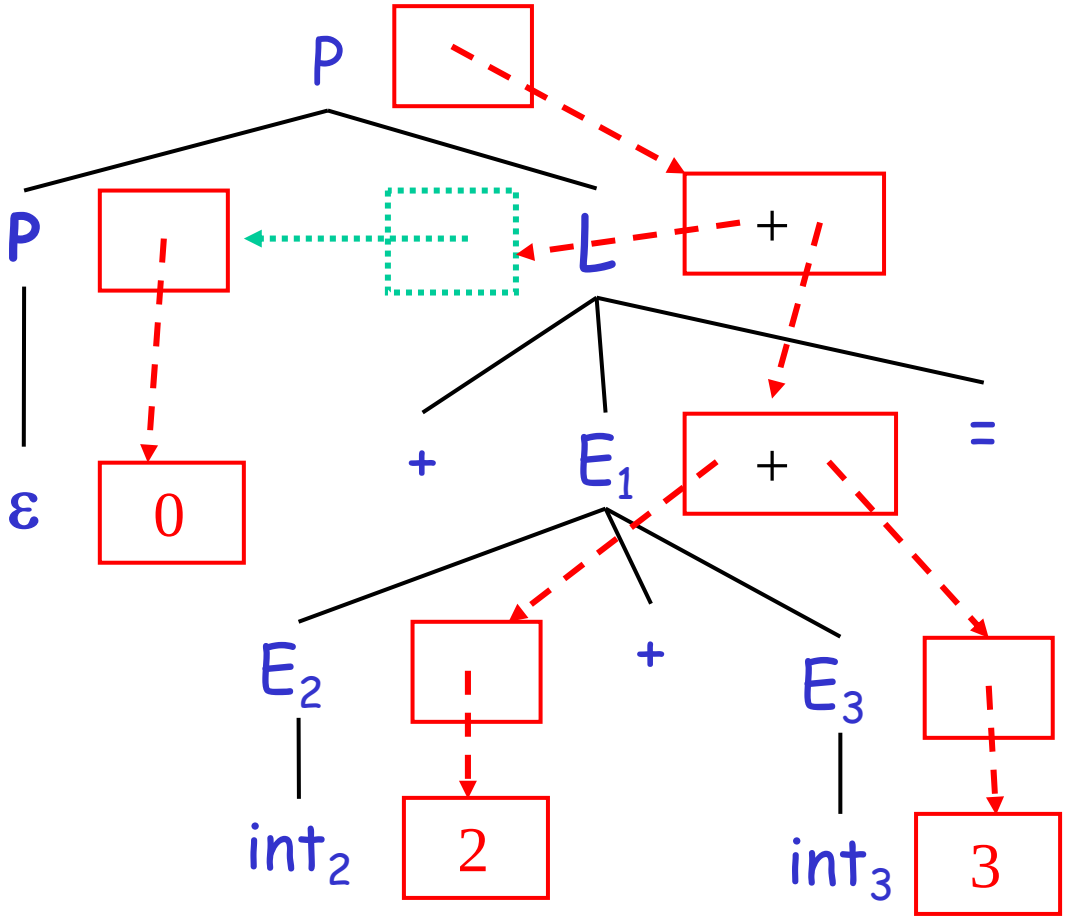
$P \rightarrow \varepsilon$ { P.val = 0 }

| P_1 L { P.val = L.val;
L.prev = P_1.val }

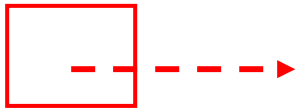
- Each L has an inherited attribute prev
  - L.prev is inherited from sibling P_1.val

- Example …

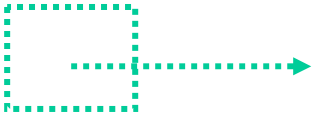
Example of Inherited Attributes



- val synthesized



- prev inherited



- All can be computed in depth-first order

Semantic Actions: Notes (Cont.)

- 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