Semantic Analysis - II

* Attribute Grammars:

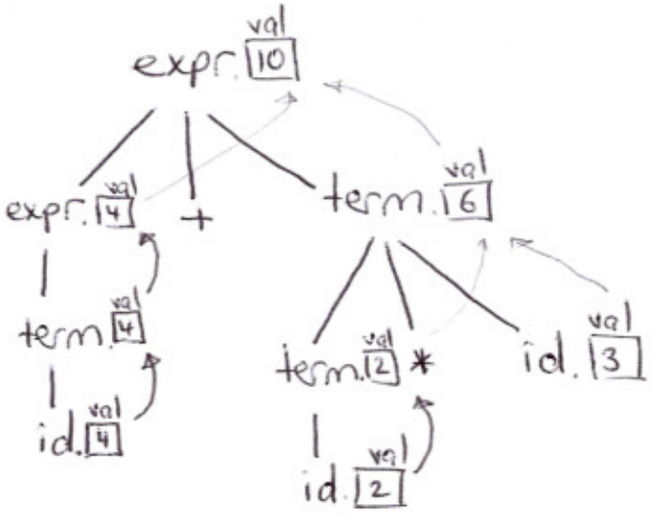
1) expr → expr + term
2) expr → term
3) term → term * id
4) term → id

- define a "val" attribute referring to the arithmetic value of each symbol
- invent a set of rules for each production to specify how the vals of different symbols are related.

1) $expr_1.val := sum(expr_2.val, term.val)$ → semantic function
2) $expr.val := term.val$ → copy rule
3) $term_1.val := product(term_2.val, id.val)$
4) $term.val := id.val$

when more than one symbol of a production has the same name, use subscripts to distinguish them.

example ① input string : 4 + 2 * 3



← Decoration of the parse tree

- In the example grammar above, each symbol has at most one attribute.
- And their values are calculated only in production in which their symbol appears on the left-hand side. → Synthesized Attributes

  They are calculated only from the symbols below them in the parse tree.

* If all attributes in an attribute grammar are synthesized, this grammar is called S-attributed.

* Inherited Attributes:

  In some cases, the values of attributes are calculated when their symbol is on the right hand side (meaning their value depend on symbols above or to the side of them in the parse tree). Such attributes are called → inherited.

Example: ② Implement left associativity for division

ej: 27 / 3 / 3 = 3

first

Second

1) expr ⟶ id A

2) A ⟶ / id A

3) A ⟶ ε

⇓

1 { expr.val = A.val
  { A.temp = expr.val

2 { A1.val = A2.val
  { A2.temp = division (A1.temp , id.val)

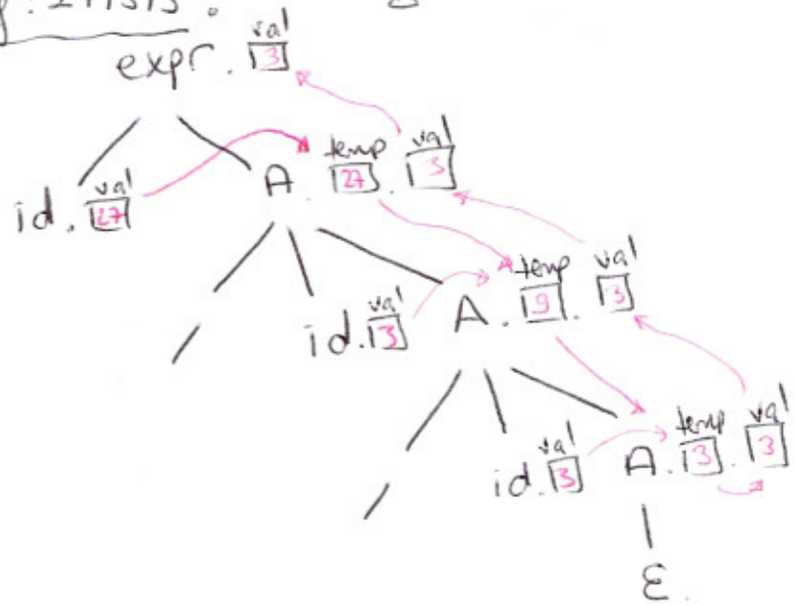
3 { A.val = A.temp

⇓

Attributes :

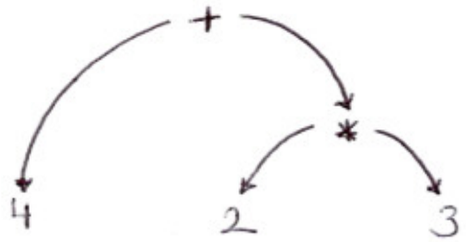
expr.val
A.val
A.temp
id.val

- for input string : 27/3/3 :



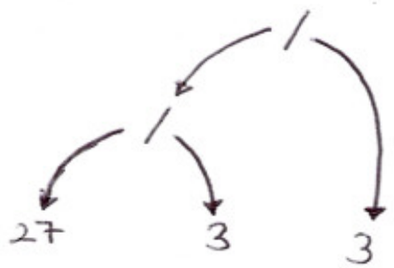
* Building a Syntax Tree: (pp. 175-178)

- Example: 1



1. make the values of ids the leaves of the new syntax tree
2. whenever you see a copy function, iterate the pointer
3. whenever you see a semantic function, generate a new node with that operator pointing to the ~~nds~~ corresponding nodes underneath.

- Example: 2



1. make the values of ids the leaves of the new syntax tree.
2. whenever you see a ~~semantic function~~ copy function, iterate the pointer.
3. whenever you see a semantic function, generate a new node with that operator pointing to the corresponding nodes underneath.