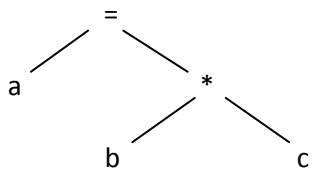
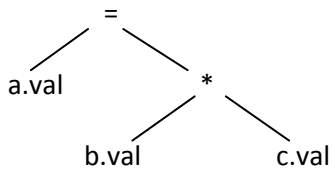
Attribute Grammar and Annotated Parse Tree

Parse Tree: Graphical representation of the source program
Example: a = b * c



Assign an attribute to each node



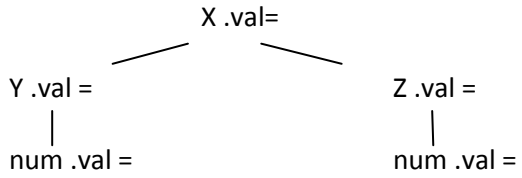
- a.val denotes that a has an attribute called val
- Value of a.val is found using the semantic rule corresponding to the production used

Annotated Parse Tree: A parse tree with attribute values at each node
Types of Attributes:
1. Synthesized attributes
2. Inherited attributes

Synthesized attributes: Derive their value from the attribute values of its children nodes

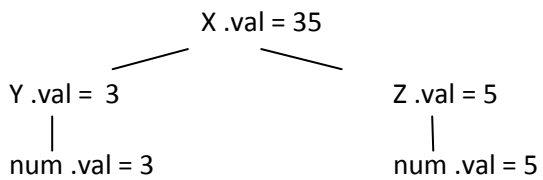
NOTE: In bottom-up approach, the values of the children node attributes are already known and the values of the parent node attribute can be calculated/derived.

1. X → YZ (Concatenation)
2. Y → num
3. Z → num



Assume num values are 3 and 5

1. X → YZ
2. Y → num
3. Z → num



Y.val was evaluated from num.val
Z.val was evaluated from num.val
X.val was evaluated from Y.val and Z.val (Concatenate)

Examples of attributes:

- Data type of a variable
- Value of an expression
- Location of a variable in the memory
- Object code of a procedure, etc.

NOTE:

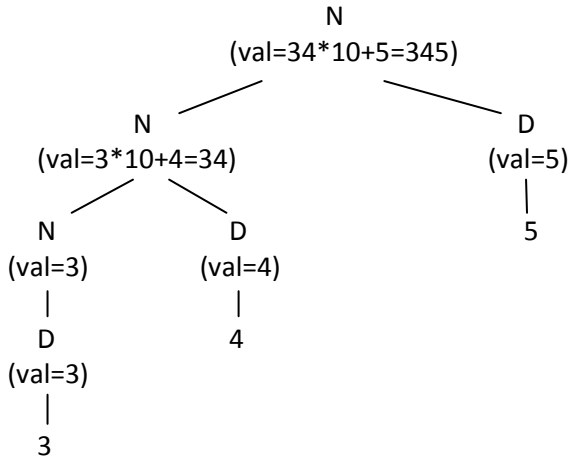
- Assign an attribute to each node
- Define value of that attribute using some RULE

$N \rightarrow ND \mid D$
$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Grammar Rule	Semantic Rule	
$N_1 \rightarrow N_2D$	$N_1.val = N_2.val * 10 + D.val$	(10=1*10+0,23=2*10+3, 345=34*10+5)
$N \rightarrow D$	N.val = D.val	
$D \rightarrow 0$	D.val = 0	
$D \rightarrow 1$	D.val = 1	
$D \rightarrow 2$	D.val = 2	
$D \rightarrow 3$	D.val = 3	
$D \rightarrow 4$	D.val = 4	
$D \rightarrow 5$	D.val = 5	
$D \rightarrow 6$	D.val = 6	
$D \rightarrow 7$	D.val = 7	
$D \rightarrow 8$	D.val = 8	
$D \rightarrow 9$	D.val = 9	

Attribute Grammar

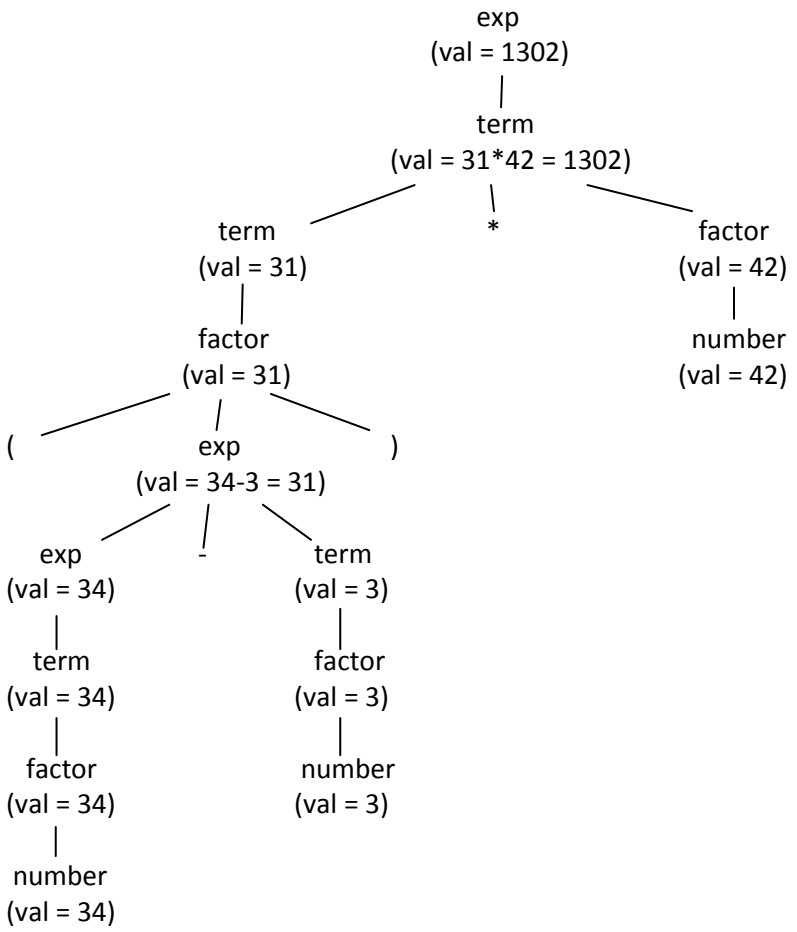
Tree for N = 345



exp → exp + term | exp - term | term
term → term * factor | factor
factor → (exp) | <u>number</u>

Grammar Rule	Semantic Rule
$exp_1 \rightarrow exp_2$ + term	exp_1.val = exp_2.val + term.val
$exp_1 \rightarrow exp_2$ – term	exp_1.val = exp_2.val - term.val
exp → term	exp.val = term.val
$term_1 \rightarrow term_2$ * factor	$term_1$.val = $term_2$.val * factor.val
term → factor	term.val = factor.val
factor → (exp)	factor.val = exp.val
factor → <u>number</u>	factor.val = <u>number.val</u>

Expression: (34-3)*42


exp
(val = 1302)
term
(val = 31*42 = 1302)
term
(val = 31)
*
factor
(val = 42)
factor
(val = 31)
number
(val = 42)
(
exp
(val = 34-3 = 31)
)
exp
(val = 34)
-
term
(val = 3)
term
(val = 34)
factor
(val = 3)
factor
(val = 34)
number
(val = 3)
number
(val = 34)

Attribute Grammar and Annotated Parse Tree - II

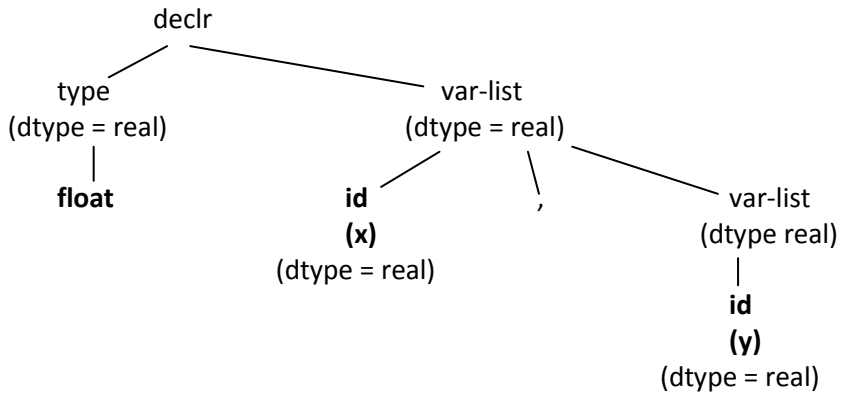
decl → type var-list example: int a, b, c;
type → **int** | **float**
var-list → **id**, var-list | **id** example: int a, b, c; int a;

Grammar Rule	**Semantic Rule**
decl → type var-list	var-list.dtype = type.dtype
type → **int**	type.dtype = integer
type → **float**	type.dtype = real
var-list_1 → **id**, var-list_2	**id**.dtype = var-list_1.dtype
	var-list_2.dtype = var-list_1.dtype
var-list → **id**	**id**.dtype = var-list.dtype

Statement: float x,y
Parse Table:

declr → type var-list	
Type → **float**	**float** var-list
var-list → **id**, var-list	**float** id, var-list
var-list → **id**	**float** x, var-list
var-list → **id**	**float** x, id
	float x, y

Parse Tree:



Grammar: number, examples: 23o (Octal), 23d (Decimal)

based-num → num basechar
basechar → **o** | **d**
num → num digit | digit
digit → **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

Attribute Grammar:

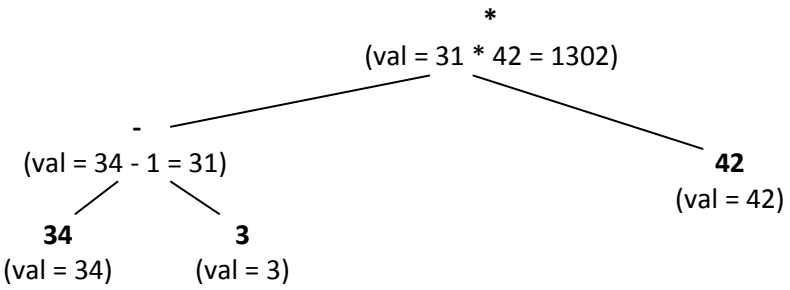
Grammar Rule	Semantic Rule
based-num → num basechar	based-num.val = num.val num.base = basechar.base
basechar → **o**	basechar.base = 8
basechar → **d**	basechar.base = 10
num_1 → num_2 digit	num_1.val = if digit.val=error or num_2.val=error then error else num_2.val * num_1.base + digit.val num_2.base = num_1.base digit.base = num_1.base
num → digit	num.val = digit.val digit.base = num.base
digit → **0**	digit.val = 0
digit → **1**	digit.val = 1
digit → **2**	digit.val = 2
digit → **3**	digit.val = 3
digit → **4**	digit.val = 4
digit → **5**	digit.val = 5
digit → **6**	digit.val = 6
digit → **7**	digit.val = 7
digit → **8**	digit.val = **if** digit.base=8 **then** error **else** 8
digit → **9**	digit.val = **if** digit.base=8 **then** error **else** 9

Grammar:
exp → exp + exp | exp - exp | exp * exp | (exp) | **number**

Attribute Grammar:

Grammar Rule	Semantic Rule
exp_1 → exp_2 + exp_3	exp_1.val = exp_2.val + exp_3.val
exp_1 → exp_2 - exp_3	exp_1.val = exp_2.val - exp_3.val
exp_1 → exp_2 * exp_3	exp_1.val = exp_2.val * exp_3.val
exp_1 → (exp_2)	exp_1.val = exp_2.val
exp → **number**	exp.val = **number**.val

Expression: (34 - 3) * 42
Parse Tree:



Grammar Rule	Semantic Rule
$exp_1 \rightarrow exp_2$ + term	exp_1.tree = mkOpNode(+,exp_2.tree,term.tree)
$exp_1 \rightarrow exp_2$ - term	exp_1.tree = mkOpNode(-,exp_2.tree,term.tree)
exp → term	exp.tree = term.tree
$term_1 \rightarrow term_2$ * factor	$term_1$.tree = mkOpNode(*,$term_2$.tree,factor.tree)
$term_1 \rightarrow term_2$ / factor	$term_1$.tree = mkOpNode(/,$term_2$.tree,factor.tree)
term → factor	term.tree = factor.tree
factor → (exp)	factor.tree = exp.tree
factor → **num**	factor.tree = mkNumNode(**num**.val)