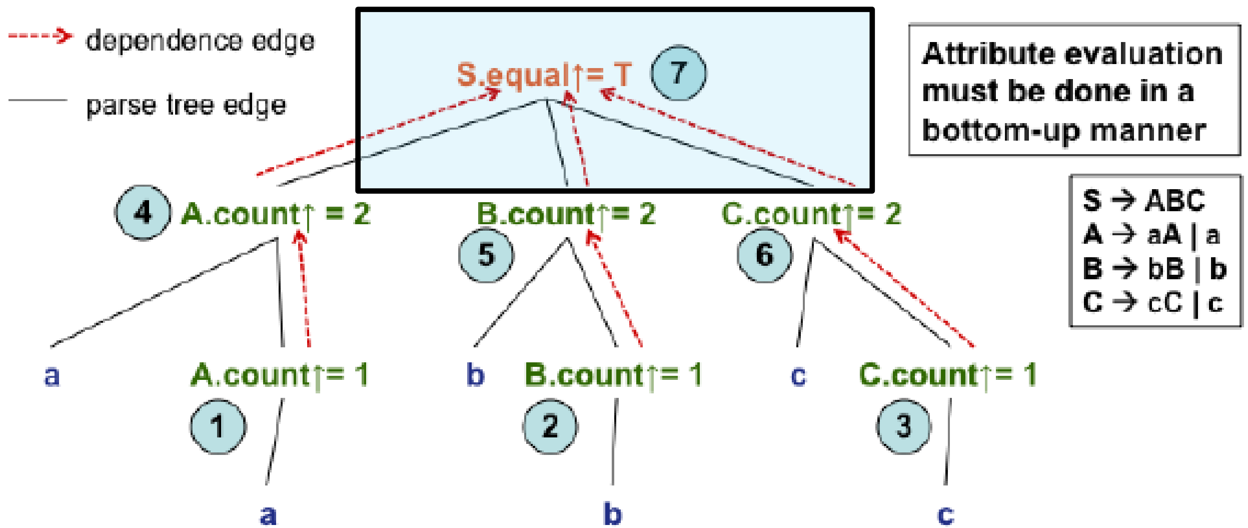
CC Lecture 20

Prepared for: 7th Sem, CE, DDU

Prepared by: Niyati J. Buch

Attribute Grammar - Example 1



1. $S \rightarrow ABC\ \{S.equal \uparrow := \text{if } A.count \uparrow = B.count \uparrow\ \&\ B.count \uparrow = C.count \uparrow \text{ then } T \text{ else } F\}$
2. $A_1 \rightarrow aA_2\ \{A_1.count \uparrow := A_2.count \uparrow + 1\}$
3. $A \rightarrow a\ \{A.count \uparrow := 1\}$
4. $B_1 \rightarrow bB_2\ \{B_1.count \uparrow := B_2.count \uparrow + 1\}$
5. $B \rightarrow b\ \{B.count \uparrow := 1\}$
6. $C_1 \rightarrow cC_2\ \{C_1.count \uparrow := C_2.count \uparrow + 1\}$
7. $C \rightarrow c\ \{C.count \uparrow := 1\}$

Attribute Dependence Graph

- Let T be a parse tree generated by the CFG of an AG, G.

- The **attribute dependence graph** (dependence graph for short) for T is the directed graph, DG(T) = (V, E), where

  V = {b|b is an attribute instance of some tree node}

  E = {(b, c)|b, c ∈ V, b and c are attributes of grammar symbols in the same production p of B, and the value of b is used for computing the value of c in an attribute computation rule associated with production p}

Attribute Dependence Graph

- An AG(attribute grammar) G is **non-circular**, if and only if for all trees T derived from G, DG(T) is acyclic
  - Non-circularity is very expensive to determine (exponential in the size of the grammar)
  - Therefore, our interest will be in subclasses of AGs whose non-circularity can be determined efficiently

- Assigning consistent values to the attribute instances in DG(T) is attribute evaluation.

Attribute Evaluation Strategy

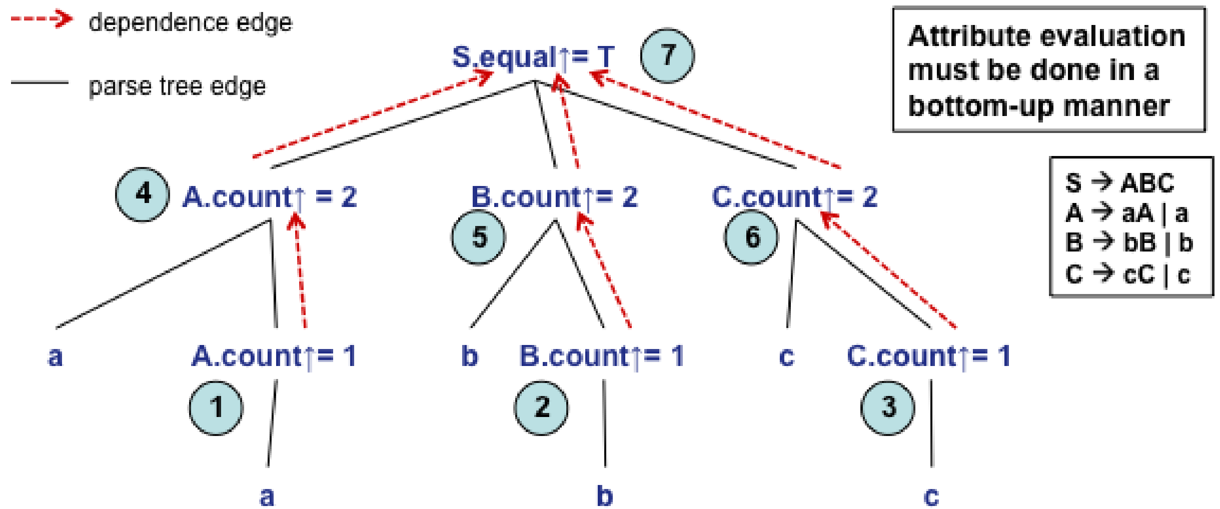
- Construct the parse tree
- Construct the dependence graph
- Perform topological sort on the dependence graph and obtain an evaluation order
- Evaluate attributes according to this order using the corresponding attribute evaluation rules attached to the respective productions
- Multiple attributes at a node in the parse tree may result in that node to be visited multiple number of times
  - Each visit resulting in the evaluation of at least one attribute

Attribute Evaluation Algorithm

Input: A parse tree T with unevaluated attribute instances
Output: T with consistent attribute values

$\{$ Let $(V, E) = DG(T)$;
(W is a queue) Let $W = \{b \mid b \in V \;\&\; indegree(b) = 0\}$;
  while $W \neq \phi$ do
    $\{$ remove some b from W;
      $value(b)$:= value defined by appropriate attribute
                computation rule;
      for all $(b, c) \in E$ do
        $\{$ $indegree(c) := indegree(c) - 1$;
          if $indegree(c) = 0$ then $W := W \cup \{c\}$;
        $\}$
    $\}$
$\}$

Dependence Graph for Example 1



1,2,3,4,5,6,7 and 2,3,6,5,1,4,7 are two possible evaluation orders. 1,4,2,5,3,6,7 can be used with LR-parsing. The right-most derivation is below (its reverse is LR-parsing order)

S => ABC => ABcC => ABcc => AbBcc => Abbcc => aAbbcc => aabbcc

1. A.count = 1 {A → a, {A.count := 1}}
4. A.count = 2 {A_1 → aA_2, {A_1.count := A_2.count + 1}}
2. B.count = 1 {B → b, {B.count :=1}}
5. B.count = 2 {B_1 → bB_2, {B_1.count := B_2.count + 1}}
3. C.count = 1 {C → c, {C.count :=1}}
6. C.count = 2 {C_1 → cC_2, {C_1.count := C_2.count + 1}}
7. S.equal = 1 {S → ABC, {S.equal := *if* A.count = B.count &
B.count = C.count *then* T else F}}

Syntax Directed Translation

=

Grammar + Semantic Rules

- S → ABC
- A → aA|a
- B→ bB|b
- C →cC|c

+

1. $S \to ABC$ $\{S.equal \uparrow := \text{if } A.count \uparrow = B.count \uparrow \;\&\; B.count \uparrow = C.count \uparrow \text{ then } T \text{ else } F\}$
2. $A_1 \to aA_2$ $\{A_1.count \uparrow := A_2.count \uparrow + 1\}$
3. $A \to a$ $\{A.count \uparrow := 1\}$
4. $B_1 \to bB_2$ $\{B_1.count \uparrow := B_2.count \uparrow + 1\}$
5. $B \to b$ $\{B.count \uparrow := 1\}$
6. $C_1 \to cC_2$ $\{C_1.count \uparrow := C_2.count \uparrow + 1\}$
7. $C \to c$ $\{C.count \uparrow := 1\}$

Example 2

- Write an attribute grammar for the evaluation of a real number from its bit-string representation.

- Example: **$(110.101)_2 = (6.625)_{10}$**

110	.	101
110 → **6**		101→5 (decimal value)/(2^ no. of bits) = 5 / 2^3 = 5 / 8 = **0.625**

Example 2

- Write an attribute grammar for the evaluation of a real number from its bit-string representation.

N → L . L

L → BL | B

B → 0|1

Example 2

$N \rightarrow L.L$

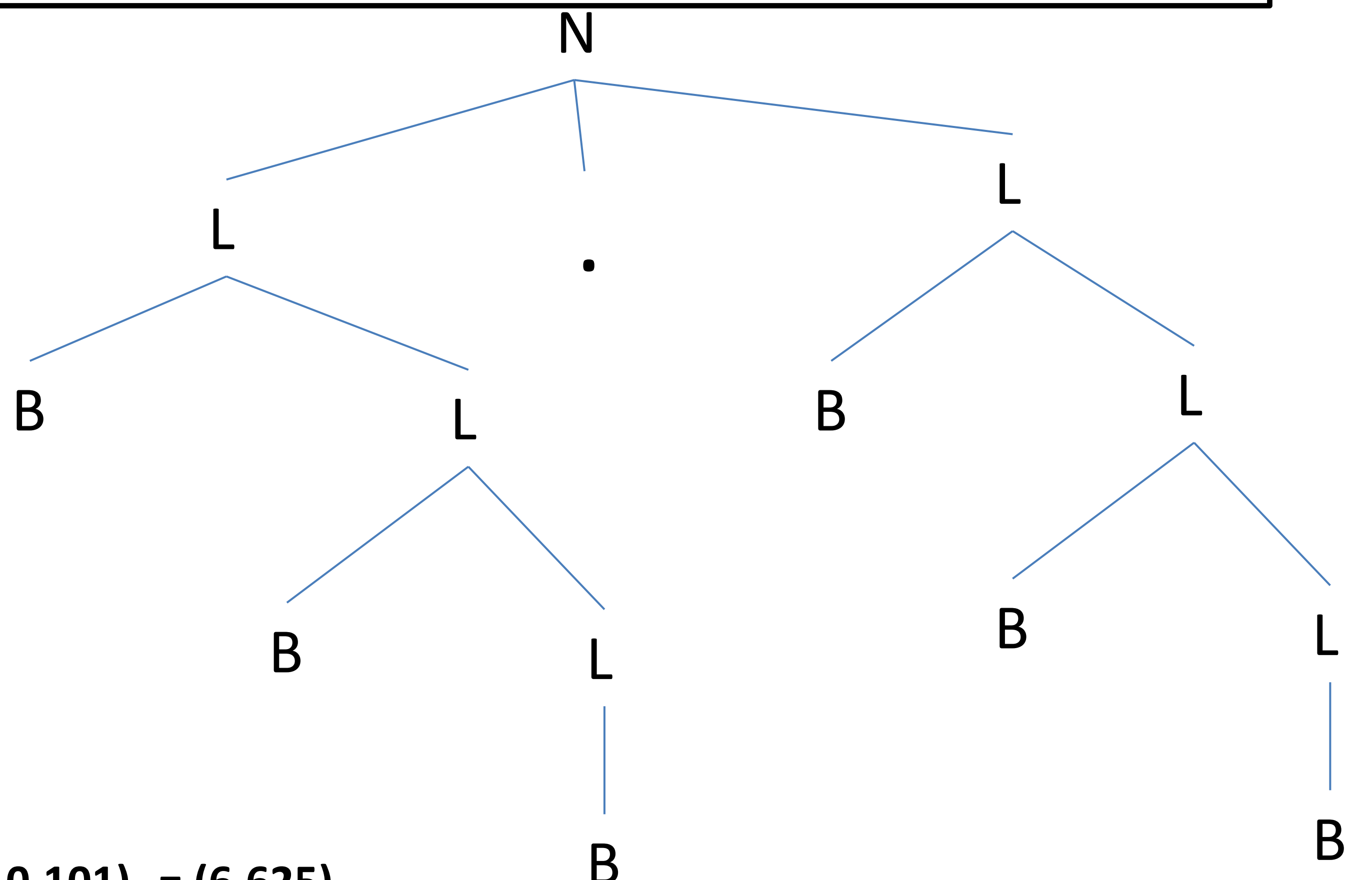
$L \rightarrow BL \mid B$

$B \rightarrow 0|1$

- $AS(N)= AS(B) =\{val\uparrow:real\}$
- $AS(L) =\{cnt\uparrow:integer, val\uparrow:real\}$

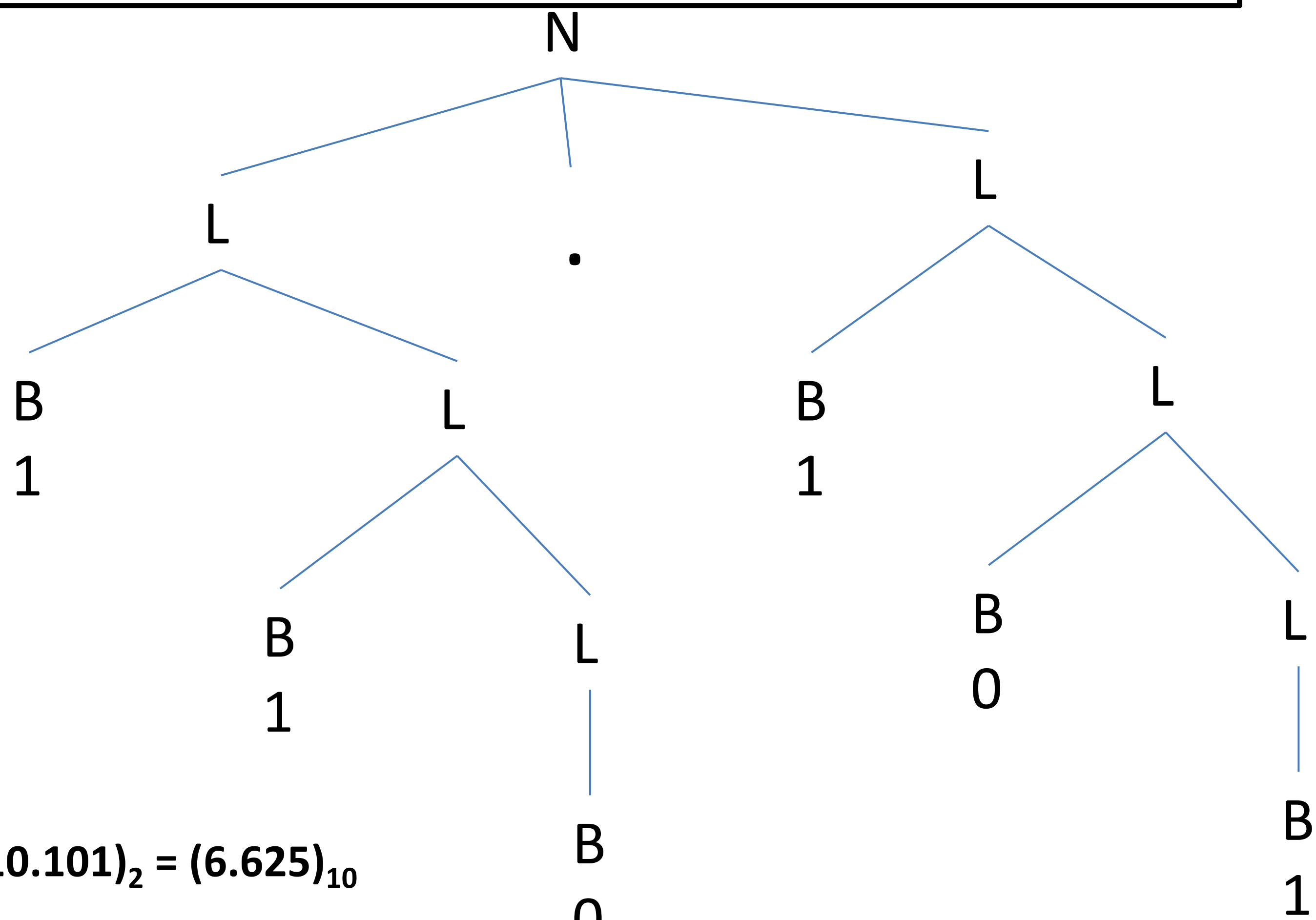
1. $N \rightarrow L_1.L_2$ $\{N.val = L_1.val + (L_2.val / 2\verb|^|L_2.cnt)\}$
2. $L \rightarrow BL_1$ $\{L.cnt=L_1.cnt+1; L.val=L_1.val+(B.val* 2\verb|^|L_1.cnt)\}$
3. $L \rightarrow B$ $\{L.cnt = 1 ; L.val = B.val\}$
4. $B \rightarrow 0$ $\{B.val = 0\}$
5. $B \rightarrow 1$ $\{B.val = 1\}$

1.	$N \rightarrow L_1.L_2$	$\{N.val = L_1.val + (L_2.val / 2\text{^}L_2.cnt)\}$
2.	$L \rightarrow BL_1$	$\{L.cnt=L_1.cnt+1; L.val=L_1.val+(B.val * 2\text{^}L_1.cnt)\}$
3.	$L \rightarrow B$	$\{L.cnt = 1 ; L.val = B.val\}$
4.	$B \rightarrow 0$	$\{B.val = 0\}$
5.	$B \rightarrow 1$	$\{B.val = 1\}$



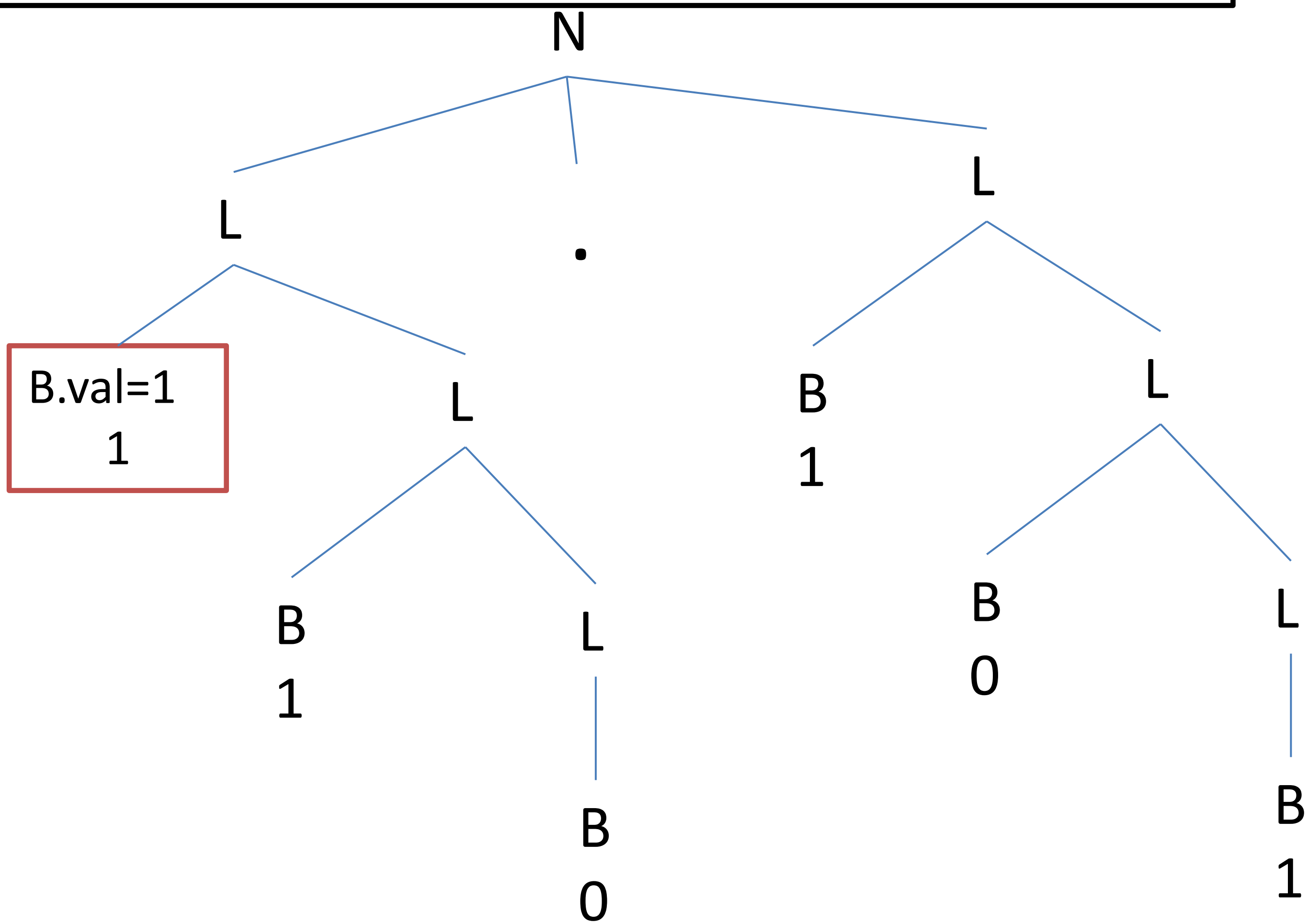
Example: $\mathbf{(110.101)_2 = (6.625)_{10}}$

1.	$N \rightarrow L_1.L_2$	{$N.val = L_1.val + (L_2.val / 2$^$L_2.cnt)$}
2.	$L \rightarrow BL_1$	{$L.cnt=L_1.cnt+1$; $L.val=L_1.val+(B.val*$ 2^$L_1.cnt)$}
3.	$L \rightarrow B$	{$L.cnt = 1$; $L.val = B.val$}
4.	$B \rightarrow 0$	{$B.val = 0$}
5.	$B \rightarrow 1$	{$B.val = 1$}

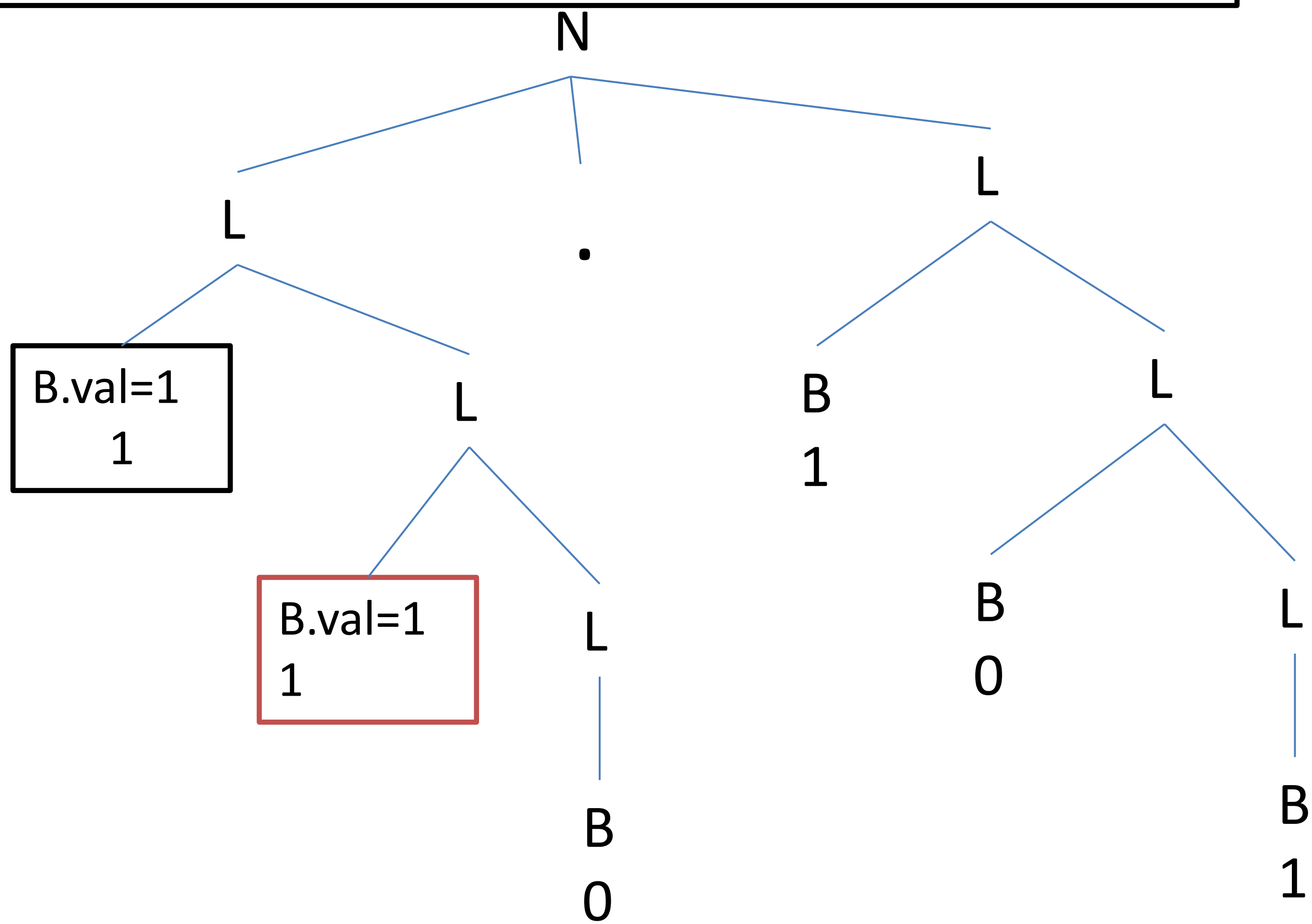


Example: $(\mathbf{110.101})_2 = (\mathbf{6.625})_{10}$

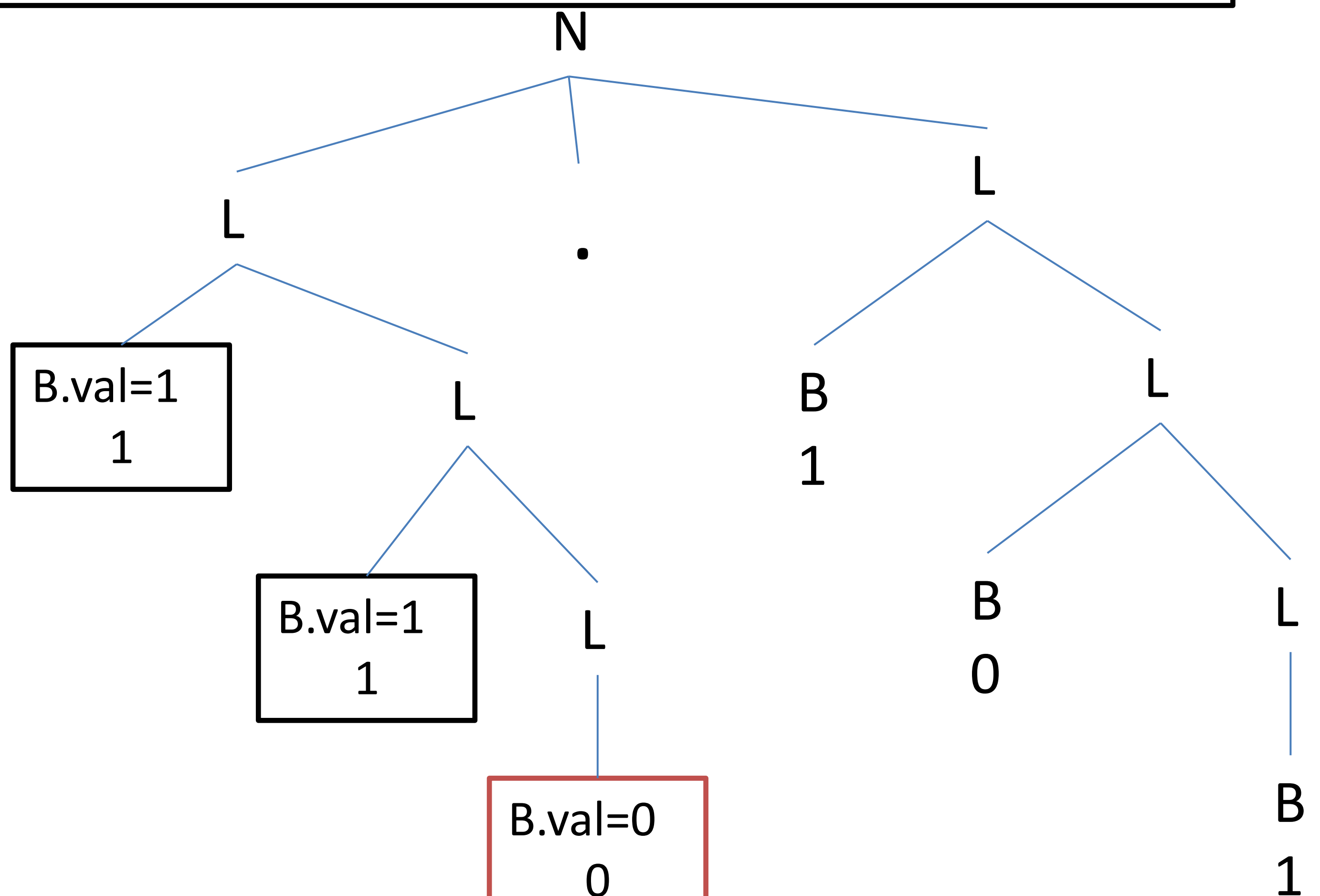
1. $N \rightarrow L_1.L_2$ $\{N.val = L_1.val + (L_2.val / 2\text{^}L_2.cnt)\}$
2. $L \rightarrow BL_1$ $\{L.cnt=L_1.cnt+1; L.val=L_1.val+(B.val * 2\text{^}L_1.cnt)\}$
3. $L \rightarrow B$ $\{L.cnt = 1 ; L.val = B.val\}$
4. $B \rightarrow 0$ $\{B.val = 0\}$
5. $B \rightarrow 1$ $\{B.val = 1\}$



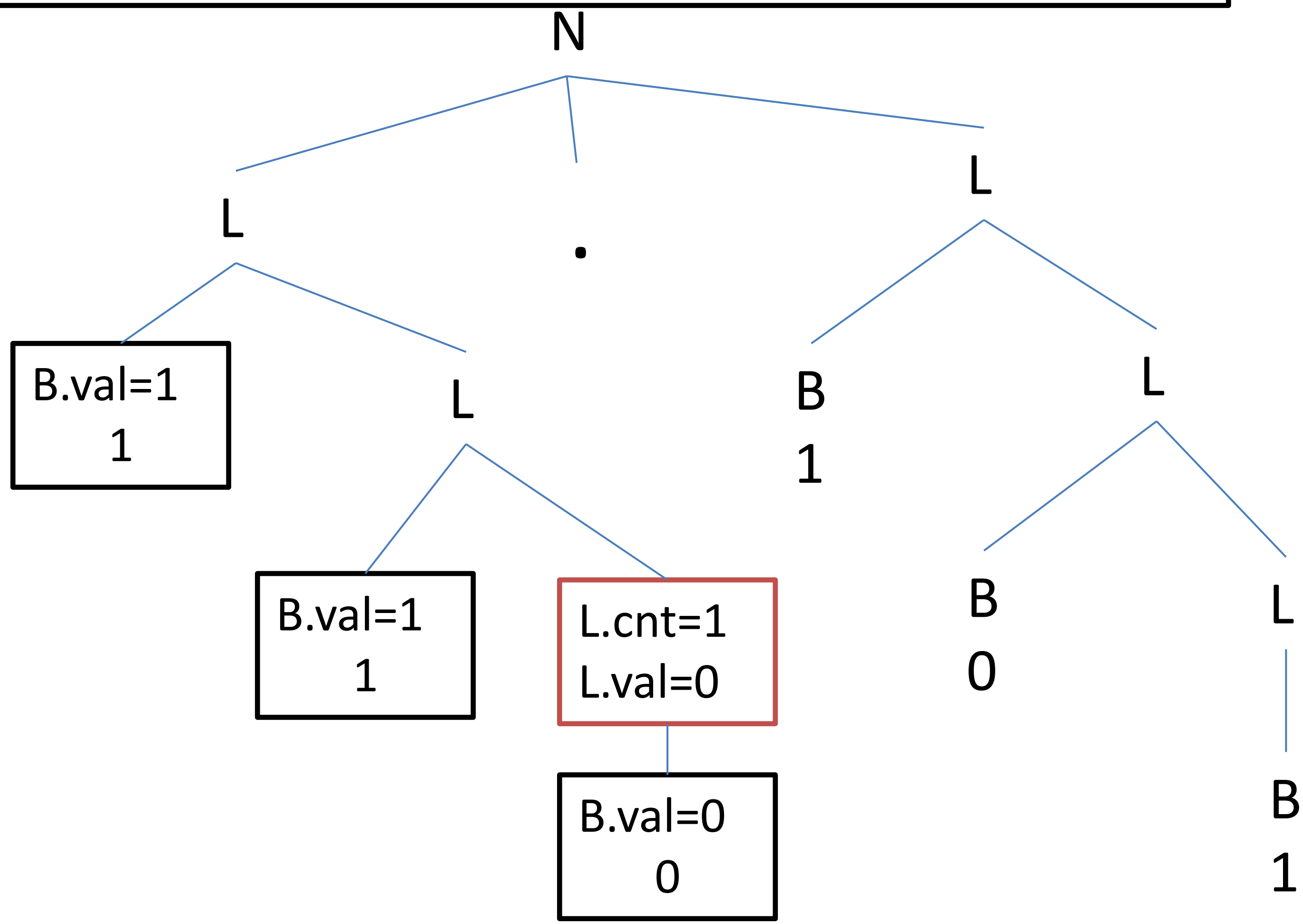
1. $N \rightarrow L_1.L_2$ $\{N.val = L_1.val + (L_2.val / 2\text{^}L_2.cnt)\}$
2. $L \rightarrow BL_1$ $\{L.cnt=L_1.cnt+1; L.val=L_1.val+(B.val * 2\text{^}L_1.cnt)\}$
3. $L \rightarrow B$ $\{L.cnt = 1 ; L.val = B.val\}$
4. $B \rightarrow 0$ $\{B.val = 0\}$
5. $B \rightarrow 1$ $\{B.val = 1\}$



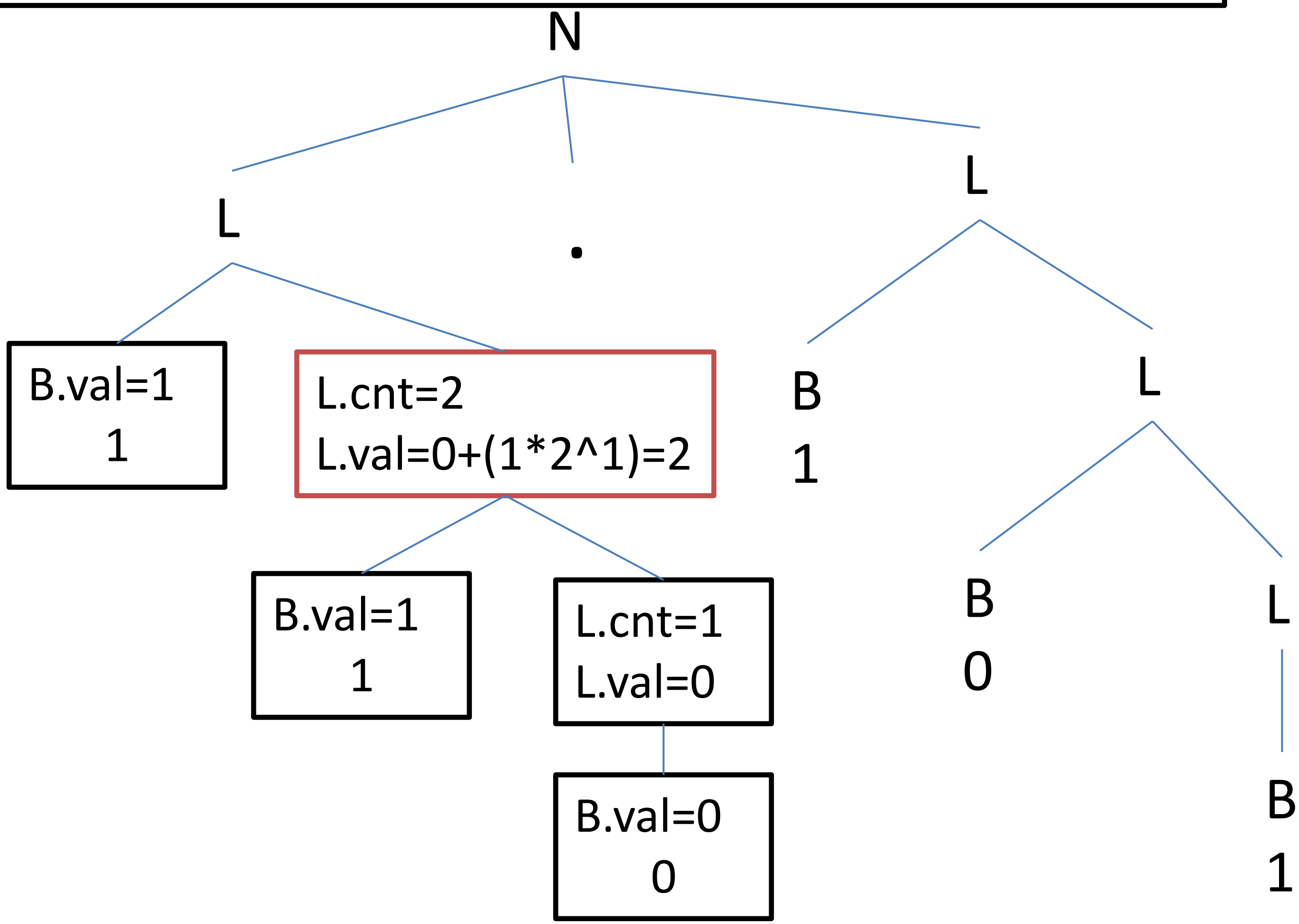
1.	$N \rightarrow L_1.L_2$	$\{N.val = L_1.val + (L_2.val / 2\string^L_2.cnt)\}$
2.	$L \rightarrow BL_1$	$\{L.cnt=L_1.cnt+1; L.val=L_1.val+(B.val{*}\ 2\string^L_1.cnt)\}$
3.	$L \rightarrow B$	$\{L.cnt = 1 ; L.val = B.val\}$
4.	$B \rightarrow 0$	$\{B.val = 0\}$
5.	$B \rightarrow 1$	$\{B.val = 1\}$



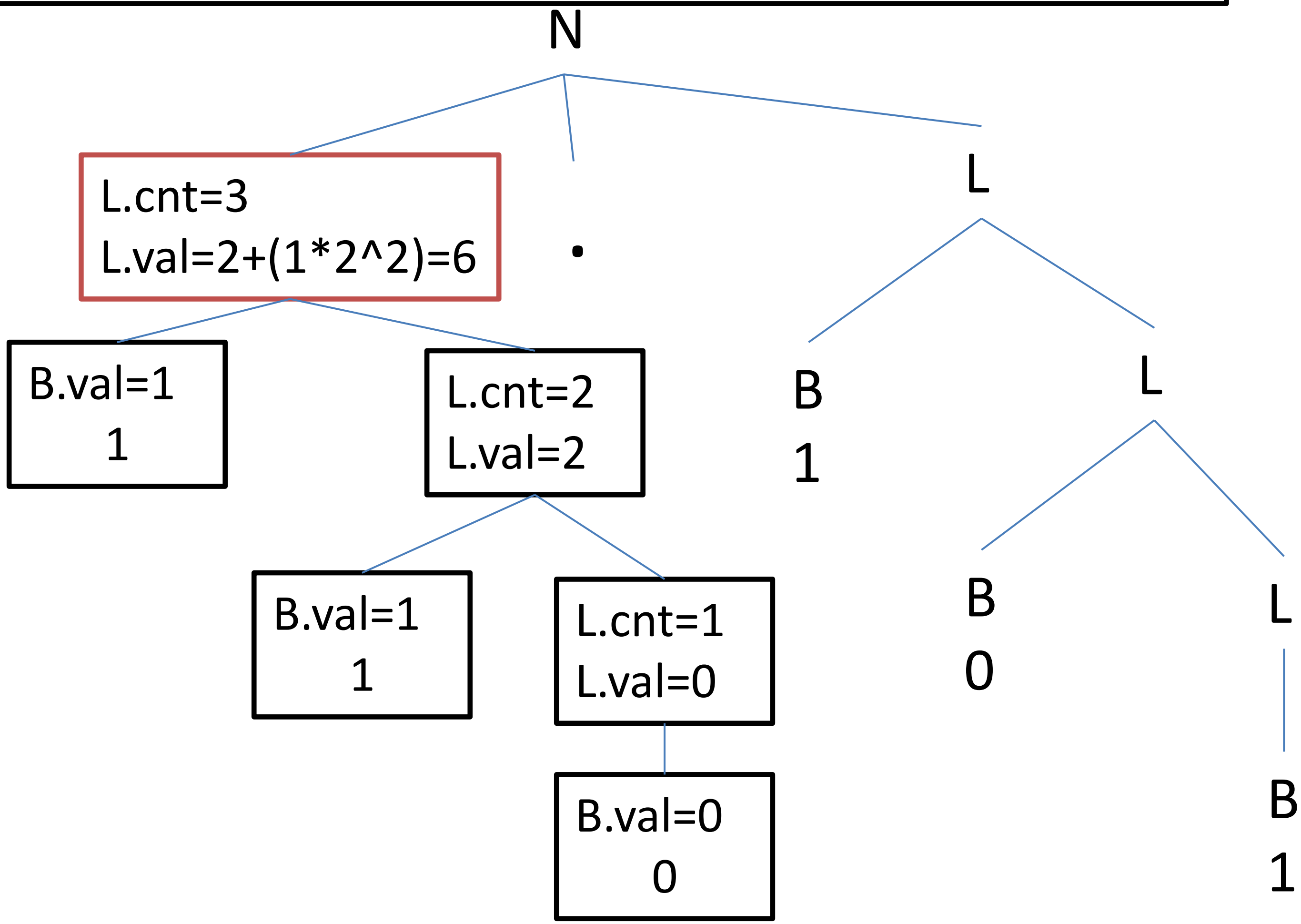
1.	$N \rightarrow L_1.L_2$	{N.val = L_1.val + (L_2.val / 2^L_2.cnt)}
2.	$L \rightarrow BL_1$	{L.cnt=L_1.cnt+1; L.val=L_1.val+(B.val* 2^L_1.cnt)}
3.	$L \rightarrow B$	{L.cnt = 1 ; L.val = B.val}
4.	$B \rightarrow 0$	{B.val = 0}
5.	$B \rightarrow 1$	{B.val = 1}



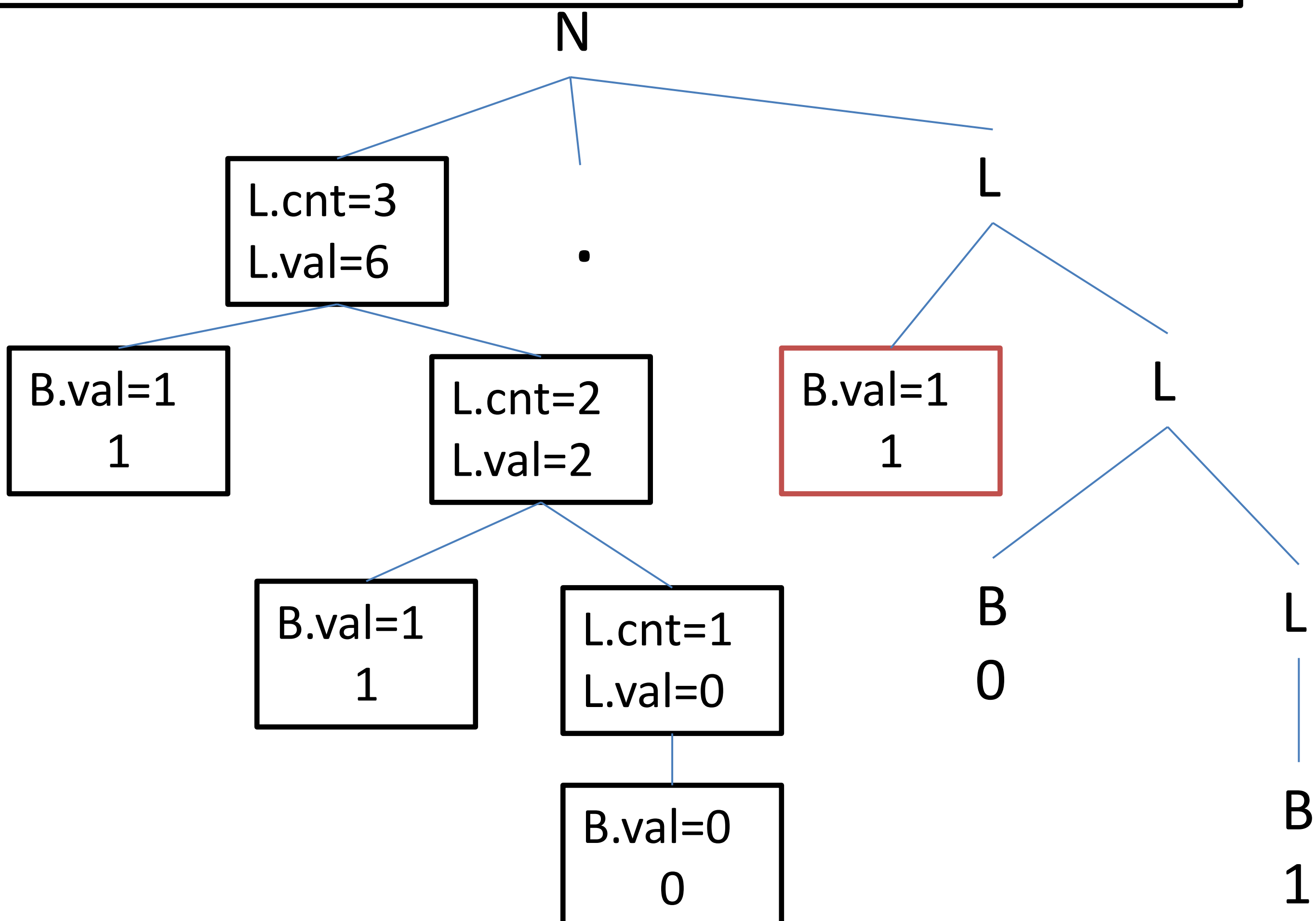
1.	$N \rightarrow L_1.L_2$	$\{N.val = L_1.val + (L_2.val / 2^\wedge L_2.cnt)\}$
2.	$L \rightarrow BL_1$	$\{L.cnt=L_1.cnt+1; L.val=L_1.val+(B.val* 2^\wedge L_1.cnt)\}$
3.	$L \rightarrow B$	$\{L.cnt = 1 ; L.val = B.val\}$
4.	$B \rightarrow 0$	$\{B.val = 0\}$
5.	$B \rightarrow 1$	$\{B.val = 1\}$



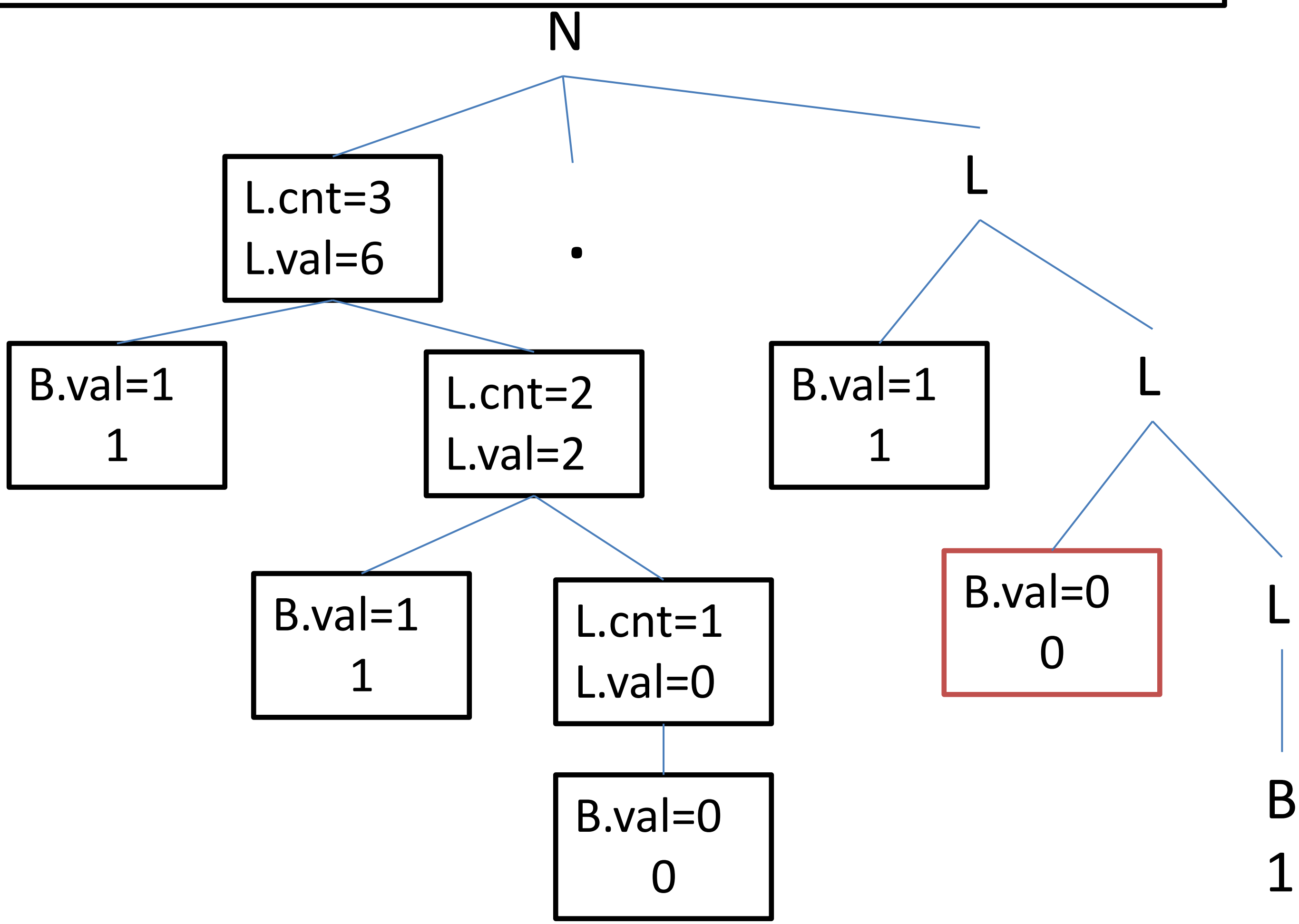
1.	$N \rightarrow L_1.L_2$	$\{N.val = L_1.val + (L_2.val / 2\text{^}L_2.cnt)\}$
2.	$L \rightarrow BL_1$	$\{L.cnt=L_1.cnt+1; L.val=L_1.val+(B.val * 2\text{^}L_1.cnt)\}$
3.	$L \rightarrow B$	$\{L.cnt = 1 ; L.val = B.val\}$
4.	$B \rightarrow 0$	$\{B.val = 0\}$
5.	$B \rightarrow 1$	$\{B.val = 1\}$



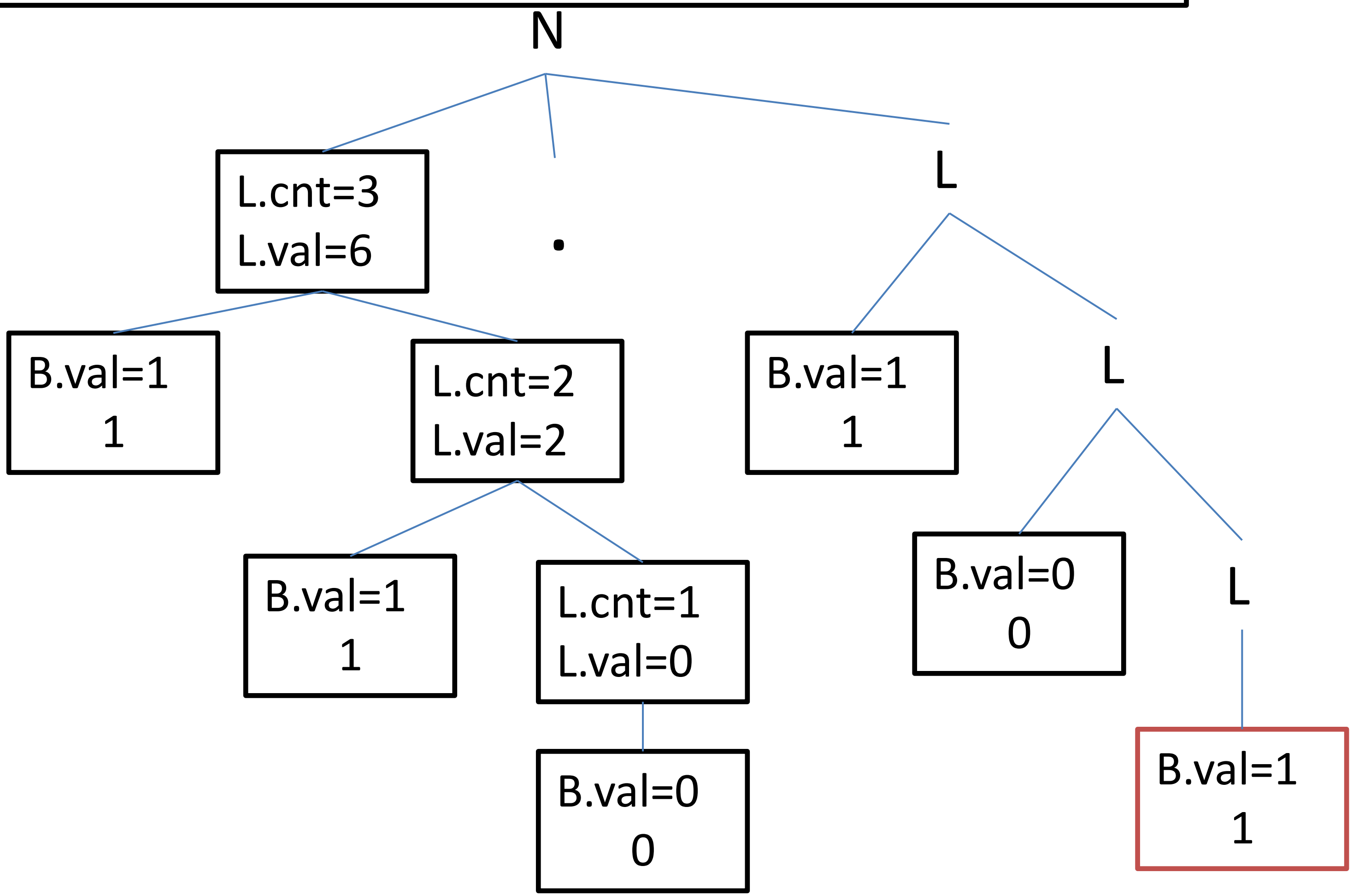
1.	$N \rightarrow L_1.L_2$		{N.val = L_1.val + (L_2.val / 2^L_2.cnt)}
2.	$L \rightarrow BL_1$		{L.cnt=L_1.cnt+1; L.val=L_1.val+(B.val* 2^L_1.cnt)}
3.	$L \rightarrow B$		{L.cnt = 1 ; L.val = B.val}
4.	$B \rightarrow 0$		{B.val = 0}
5.	$B \rightarrow 1$		{B.val = 1}



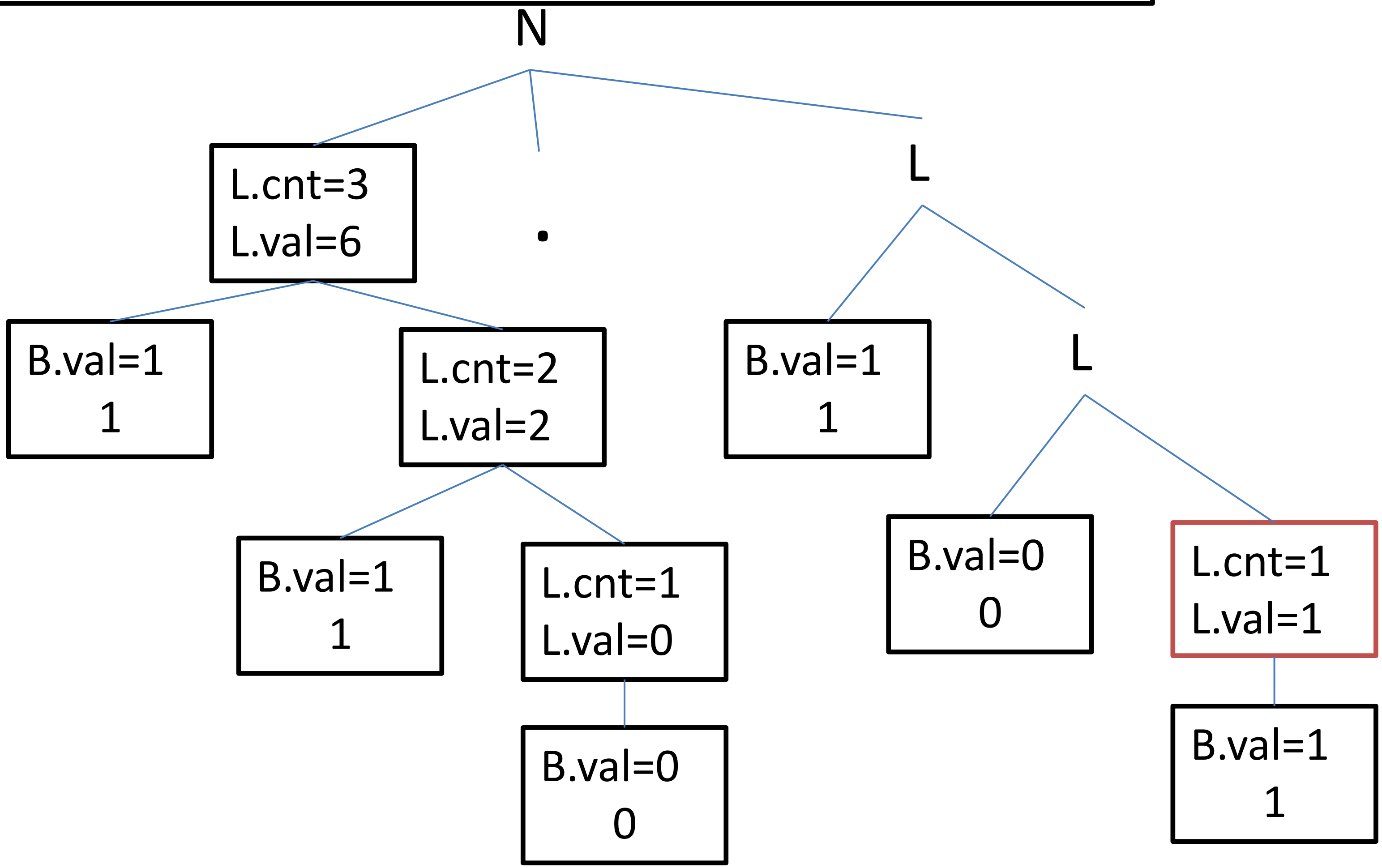
1.	$N \rightarrow L_1.L_2$	{N.val = L_1.val + (L_2.val / 2^L_2.cnt)}
2.	$L \rightarrow BL_1$	{L.cnt=L_1.cnt+1; L.val=L_1.val+(B.val* 2^L_1.cnt)}
3.	$L \rightarrow B$	{L.cnt = 1 ; L.val = B.val}
4.	$B \rightarrow 0$	{B.val = 0}
5.	$B \rightarrow 1$	{B.val = 1}



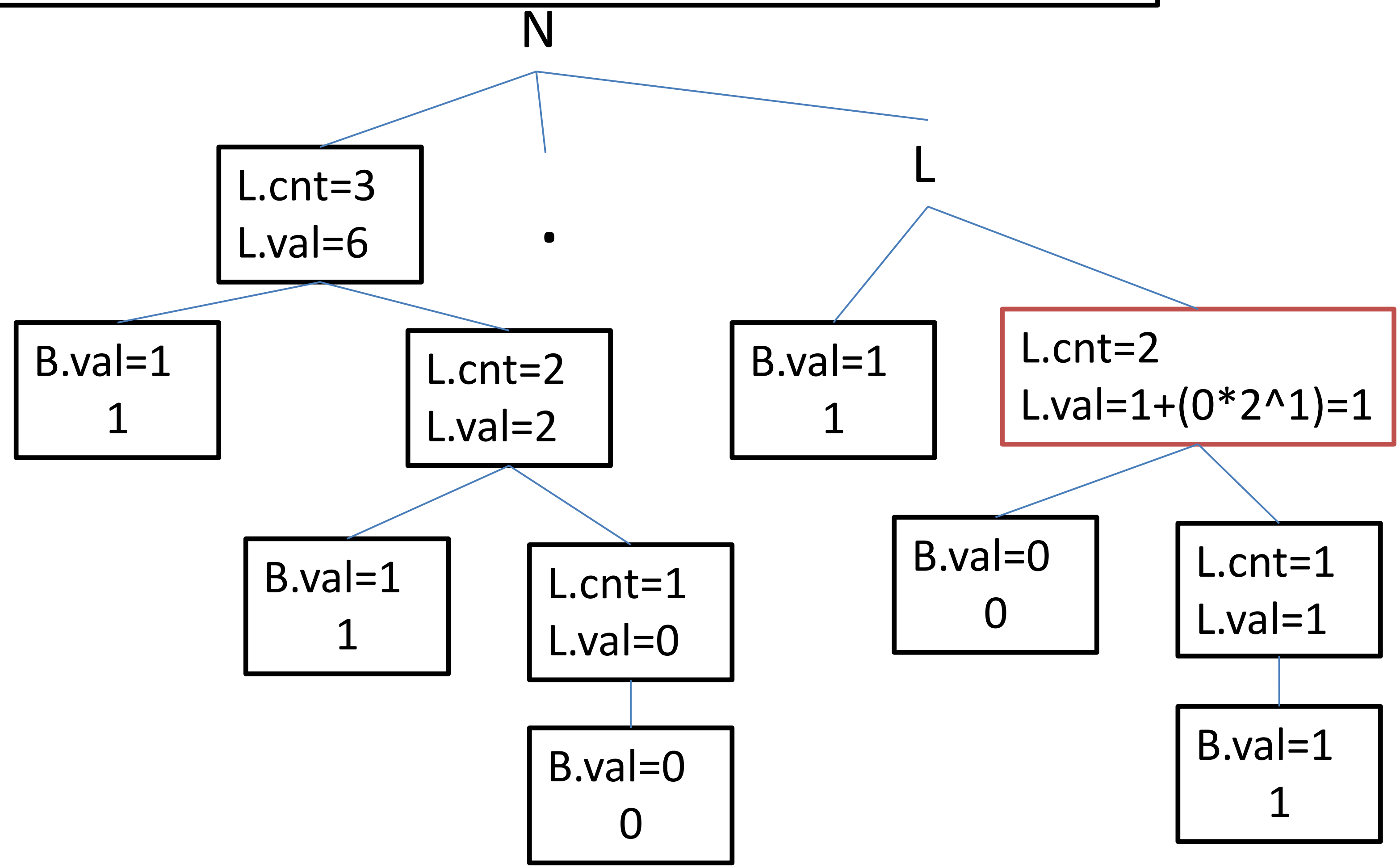
1. $N \rightarrow L_1.L_2$ {N.val = L_1.val + (L_2.val / 2^L_2.cnt)}
2. $L \rightarrow BL_1$ {L.cnt=L_1.cnt+1; L.val=L_1.val+(B.val* 2^L_1.cnt)}
3. $L \rightarrow B$ {L.cnt = 1 ; L.val = B.val}
4. $B \rightarrow 0$ {B.val = 0}
5. $B \rightarrow 1$ {B.val = 1}



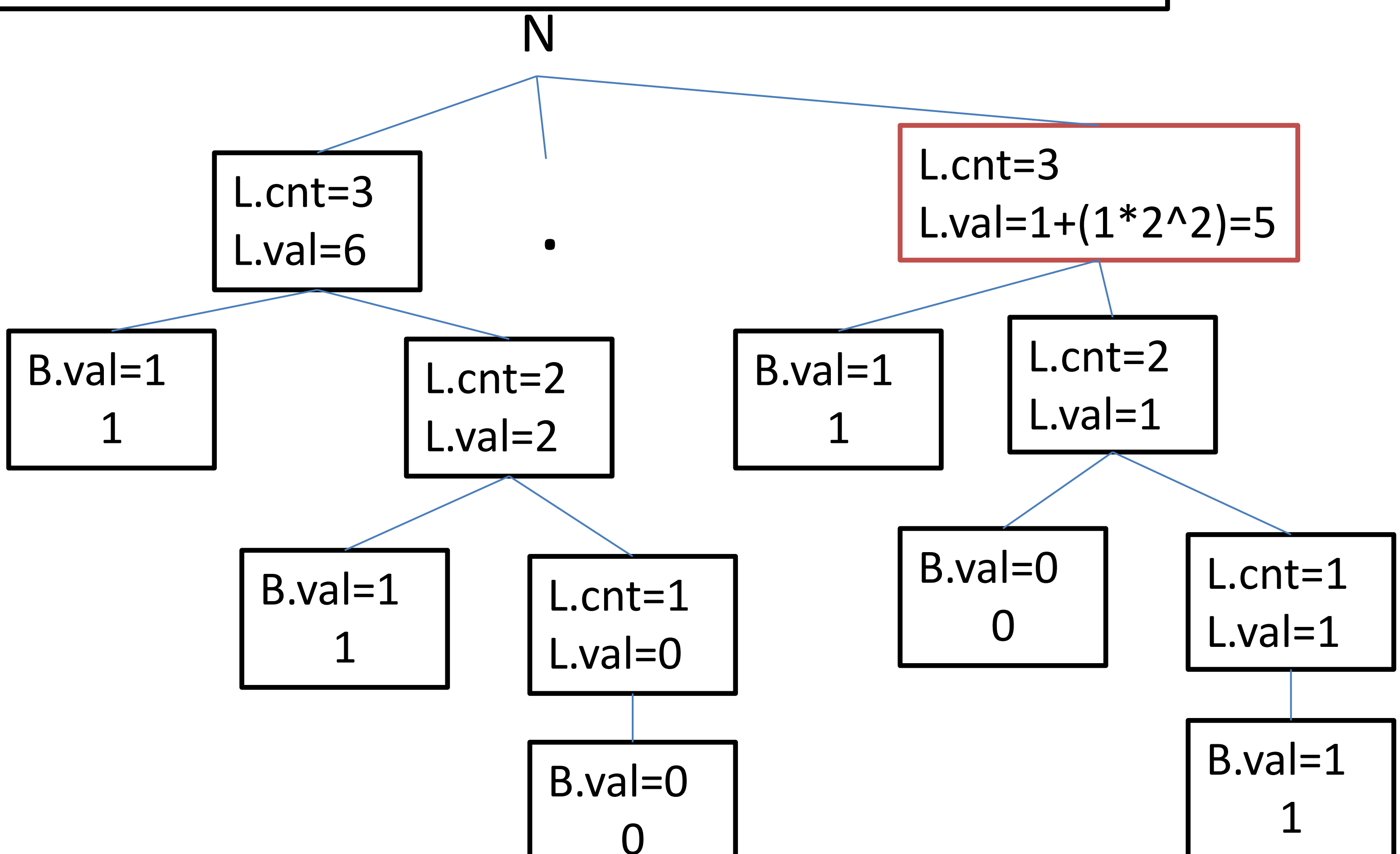
1.	$N \rightarrow L_1.L_2$	{N.val = L_1.val + (L_2.val / 2^L_2.cnt)}
2.	$L \rightarrow BL_1$	{L.cnt=L_1.cnt+1; L.val=L_1.val+(B.val* 2^L_1.cnt)}
3.	$L \rightarrow B$	{L.cnt = 1 ; L.val = B.val}
4.	$B \rightarrow 0$	{B.val = 0}
5.	$B \rightarrow 1$	{B.val = 1}



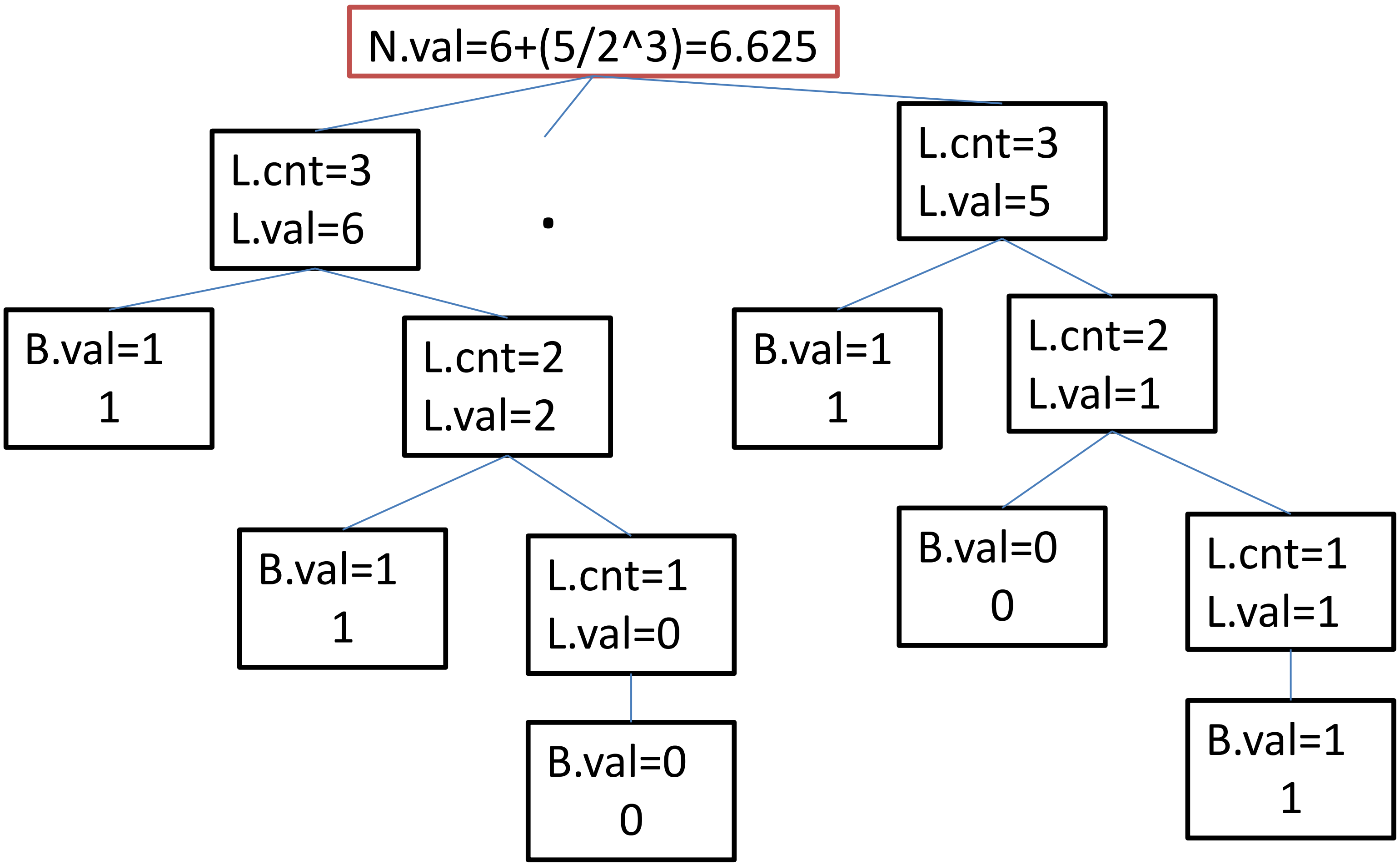
1.	$N \rightarrow L_1.L_2$	$\{N.val = L_1.val + (L_2.val / 2\text{^}L_2.cnt)\}$
2.	$L \rightarrow BL_1$	$\{L.cnt=L_1.cnt+1; L.val=L_1.val+(B.val* 2\text{^}L_1.cnt)\}$
3.	$L \rightarrow B$	$\{L.cnt = 1 ; L.val = B.val\}$
4.	$B \rightarrow 0$	$\{B.val = 0\}$
5.	$B \rightarrow 1$	$\{B.val = 1\}$



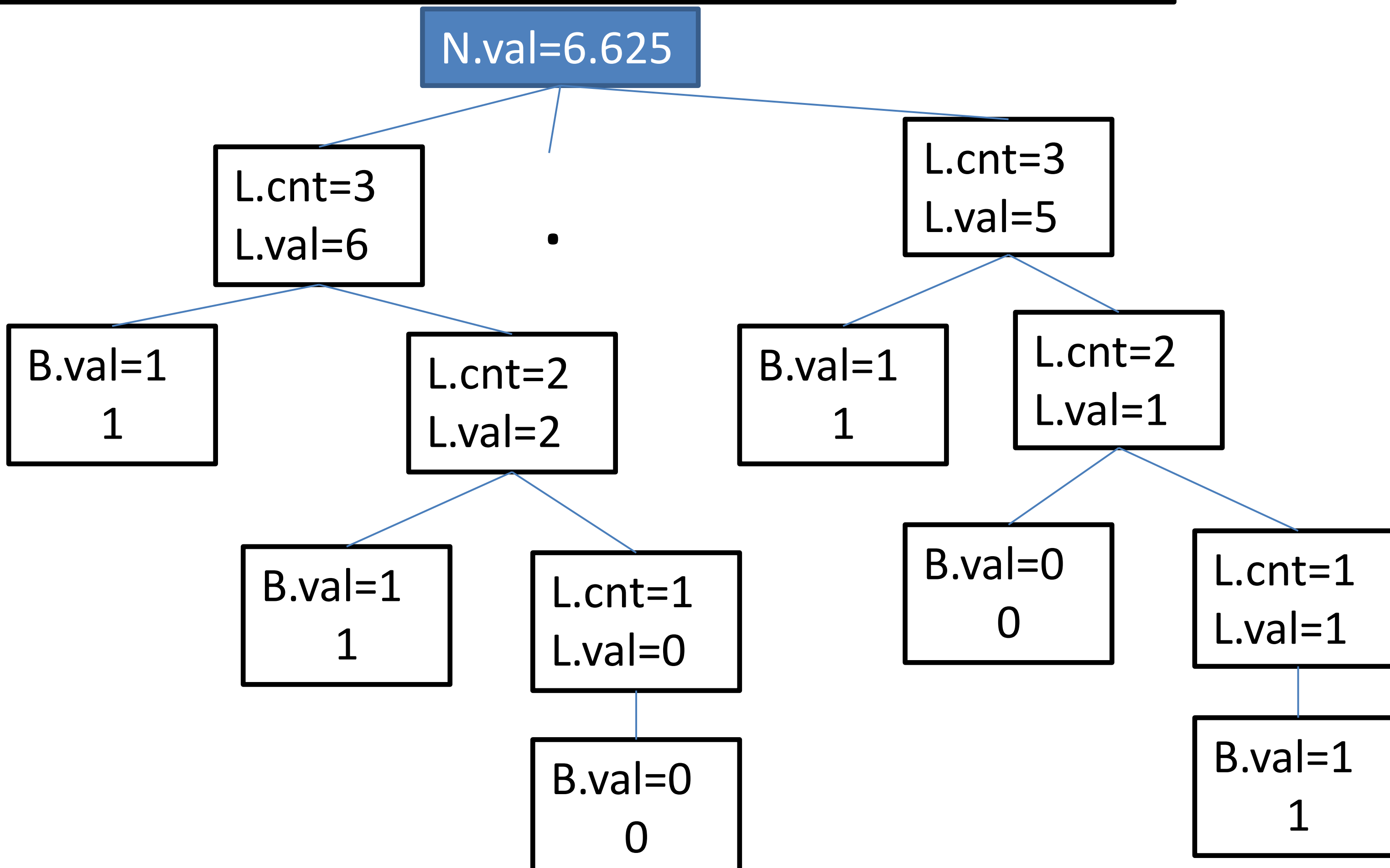
1.	$N \rightarrow L_1.L_2$		$\{N.val = L_1.val + (L_2.val\ /\ 2\text{^}L_2.cnt)\}$
2.	$L \rightarrow BL_1$		$\{L.cnt=L_1.cnt+1;\ L.val=L_1.val+(B.val*\ 2\text{^}L_1.cnt)\}$
3.	$L \rightarrow B$		$\{L.cnt = 1\ ;\ L.val = B.val\}$
4.	$B \rightarrow 0$		$\{B.val = 0\}$
5.	$B \rightarrow 1$		$\{B.val = 1\}$



1. $N \rightarrow L_1.L_2$ $\{N.val = L_1.val + (L_2.val / 2\hat{}L_2.cnt)\}$
2. $L \rightarrow BL_1$ $\{L.cnt=L_1.cnt+1; L.val=L_1.val+(B.val * 2\hat{}L_1.cnt)\}$
3. $L \rightarrow B$ $\{L.cnt = 1 ; L.val = B.val\}$
4. $B \rightarrow 0$ $\{B.val = 0\}$
5. $B \rightarrow 1$ $\{B.val = 1\}$



1.	$N \rightarrow L_1.L_2$	$\{N.val = L_1.val + (L_2.val\ /\ 2\text{^}L_2.cnt)\}$
2.	$L \rightarrow BL_1$	$\{L.cnt=L_1.cnt+1;\ L.val=L_1.val+(B.val*\ 2\text{^}L_1.cnt)\}$
3.	$L \rightarrow B$	$\{L.cnt = 1\ ;\ L.val = B.val\}$
4.	$B \rightarrow 0$	$\{B.val = 0\}$
5.	$B \rightarrow 1$	$\{B.val = 1\}$



Example 2 (second method)

- Write an attribute grammar for the evaluation of a real number from its bit-string representation.

- Example: **$(110.101)_2 = (6.625)_{10}$**

110	.	101
110 → **6**		(decimal value)/(2^ no. of bits) = 5 / 2^3 = 5 / 8 = **0.625**

Example 2 (second method)

- Write an attribute grammar for the evaluation of a real number from its bit-string representation.

$N \rightarrow L \, . \, L$

$L \rightarrow LB \mid B$

$B \rightarrow 0 \mid 1$

Example 2 (second method)

$N \rightarrow L.L$

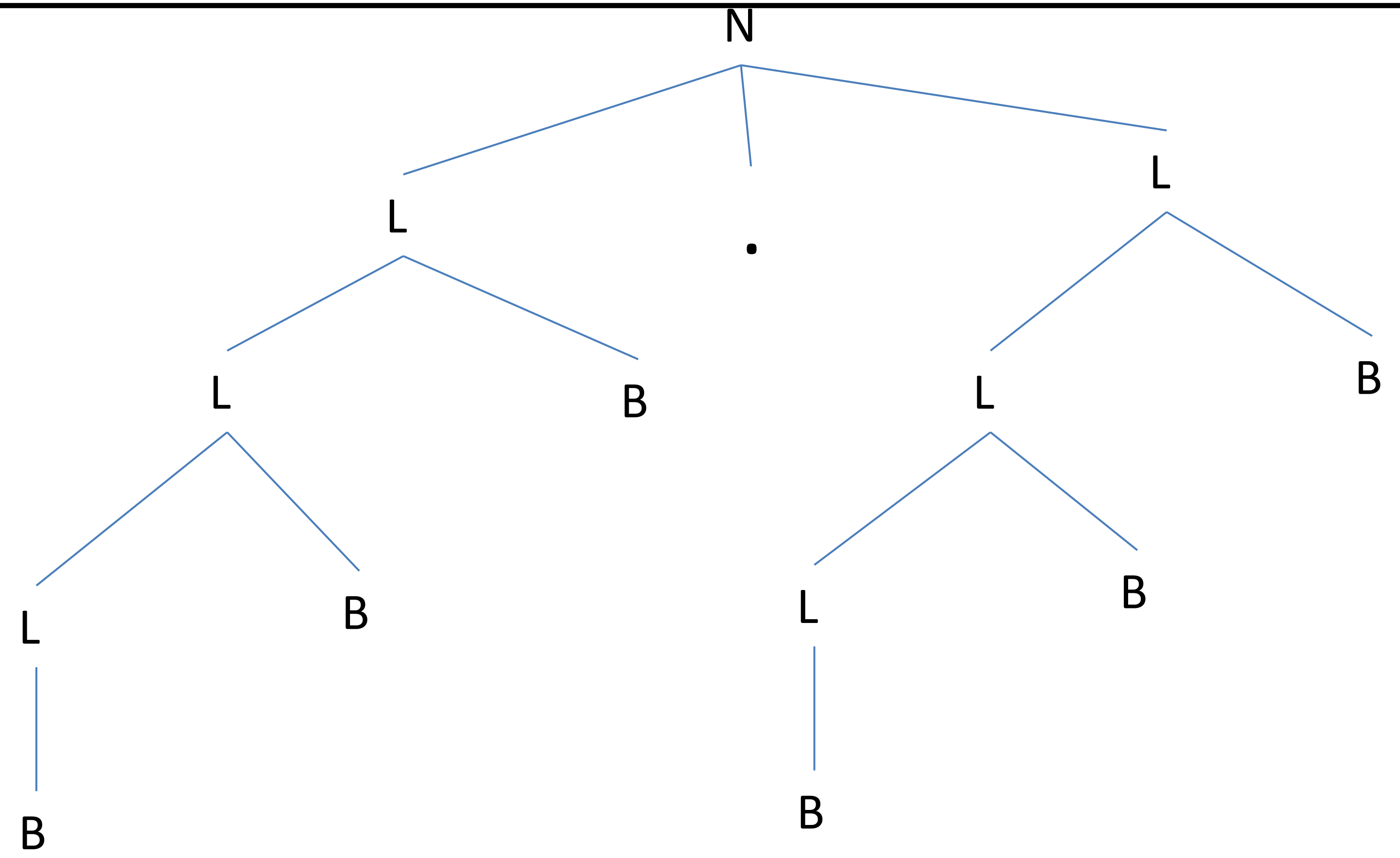
$L \rightarrow LB \mid B$

$B \rightarrow 0 \mid 1$

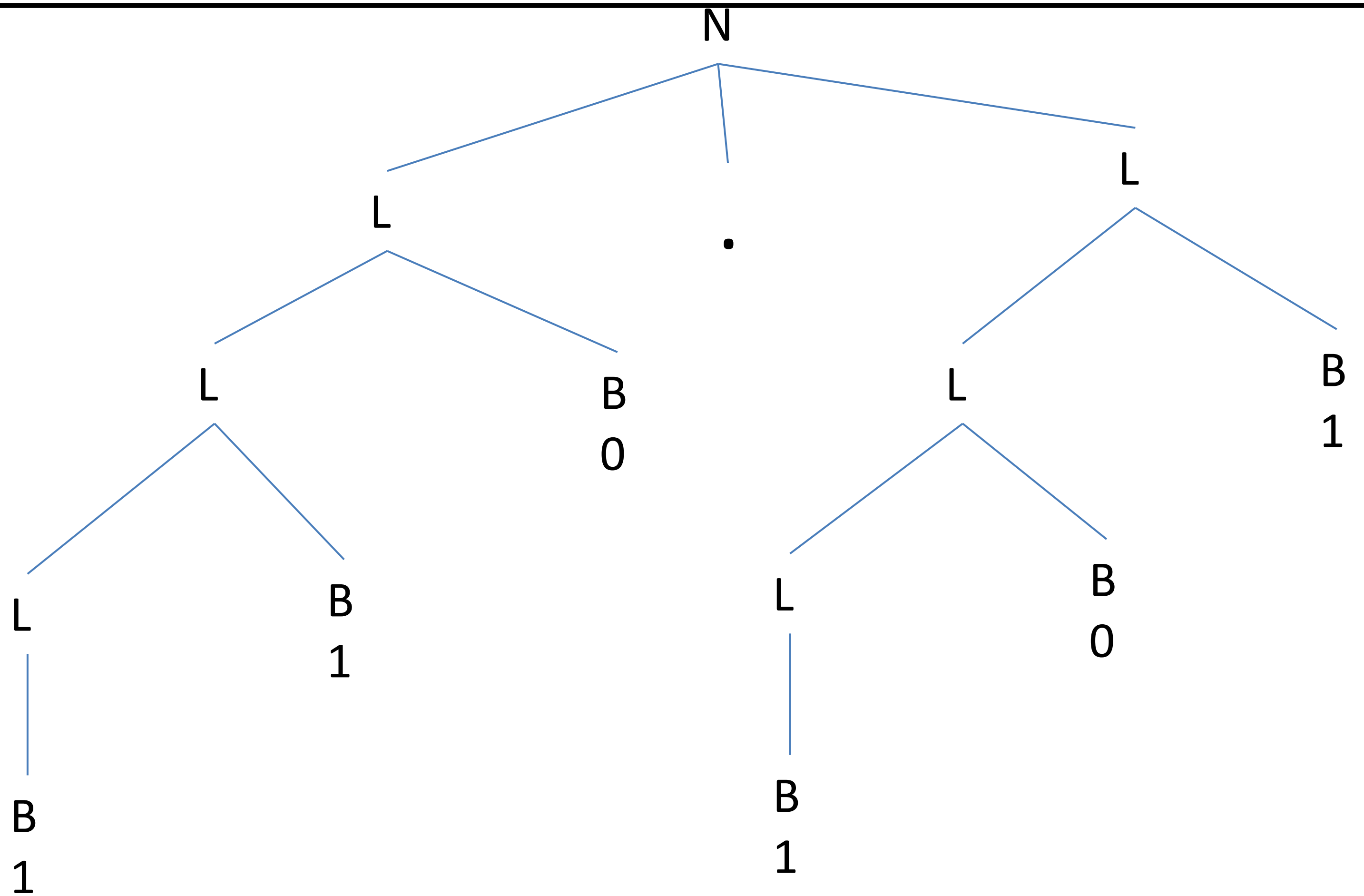
- AS(N)={val↑:real}
- AS(L) = AS(B) ={cnt↑:integer, val↑:real}

1. $N \rightarrow L_1.L_2$ {$N.val = L_1.val + (L_2.val / 2$^$L_2.cnt)$}
2. $L \rightarrow L_1B$ {$L.cnt=L_1.cnt+1$; $L.val=L_1.val*2 + B.val$}
3. $L \rightarrow B$ {$L.cnt = 1$; $L.val = B.val$}
4. $B \rightarrow 0$ {$B.val = 0$}
5. $B \rightarrow 1$ {$B.val = 1$}

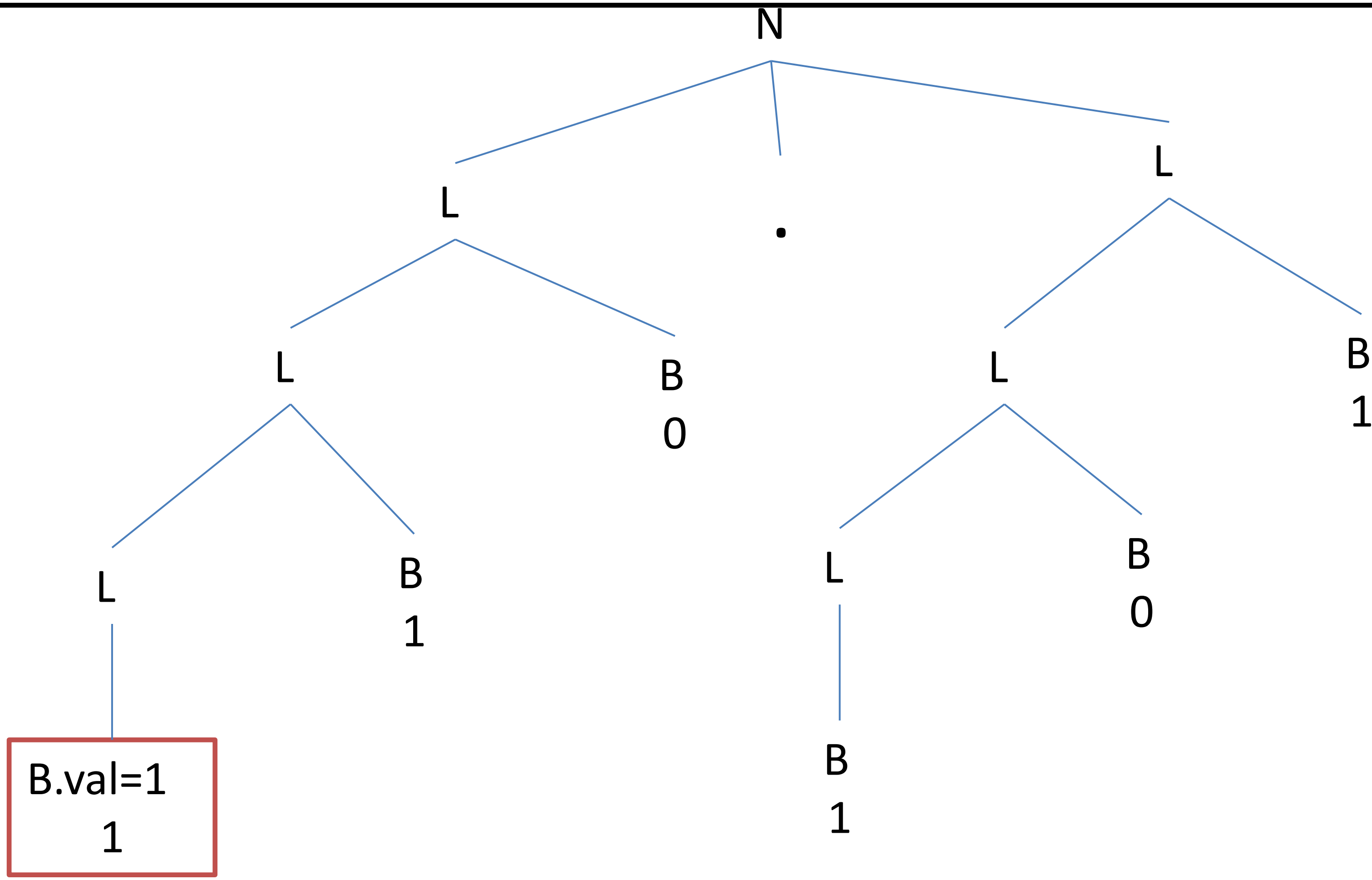
1. $N \rightarrow L_1.L_2$ {$N.val = L_1.val + (L_2.val / 2\verb|^|L_2.cnt)$}
2. $L \rightarrow L_1B$ {$L.cnt=L_1.cnt+1; L.val=L_1.val*2 + B.val$}
3. $L \rightarrow B$ {$L.cnt = 1 ; L.val = B.val$}
4. $B \rightarrow 0$ {$B.val = 0$}
5. $B \rightarrow 1$ {$B.val = 1$}



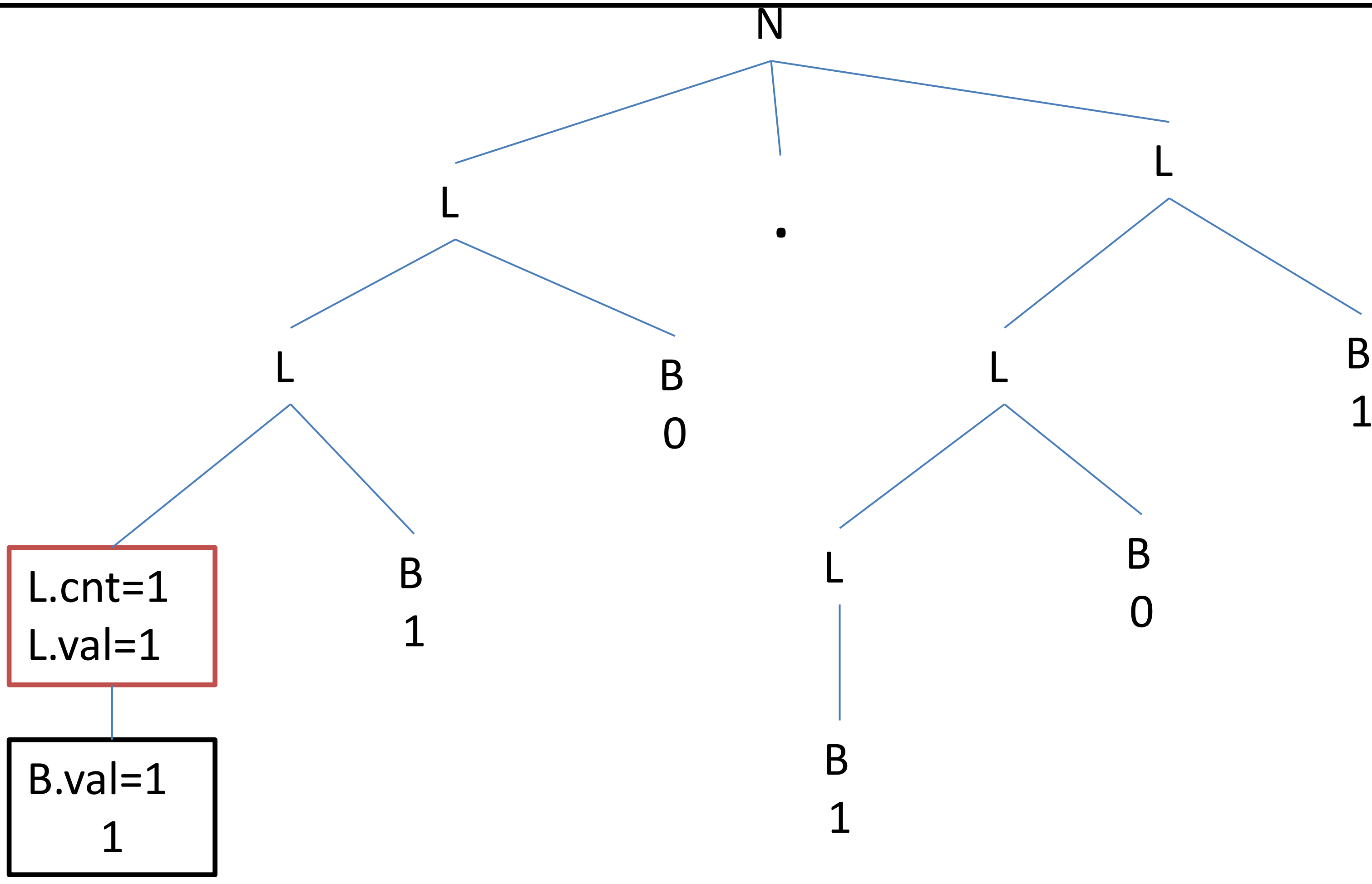
1. $N \rightarrow L_1.L_2$ {$N.val = L_1.val + (L_2.val / 2$^$L_2.cnt)$}
2. $L \rightarrow L_1B$ {$L.cnt=L_1.cnt+1$; $L.val=L_1.val*2 + B.val$}
3. $L \rightarrow B$ {$L.cnt = 1$; $L.val = B.val$}
4. $B \rightarrow 0$ {$B.val = 0$}
5. $B \rightarrow 1$ {$B.val = 1$}



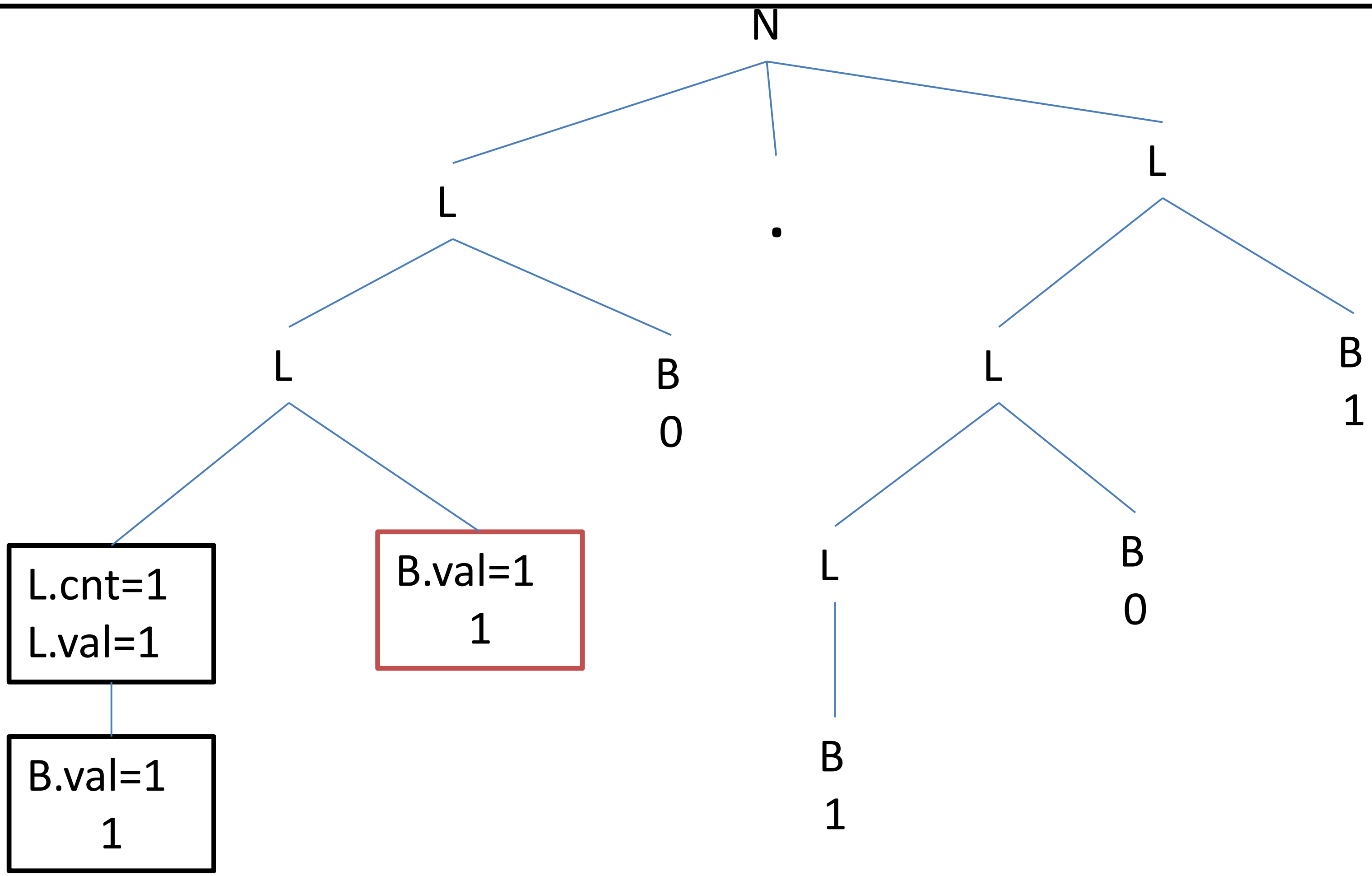
1. $N \rightarrow L_1.L_2$ $\{N.val = L_1.val + (L_2.val / 2\text{^}L_2.cnt)\}$
2. $L \rightarrow L_1B$ $\{L.cnt=L_1.cnt+1; L.val=L_1.val*2 + B.val\}$
3. $L \rightarrow B$ $\{L.cnt = 1 ; L.val = B.val\}$
4. $B \rightarrow 0$ $\{B.val = 0\}$
5. $B \rightarrow 1$ $\{B.val = 1\}$



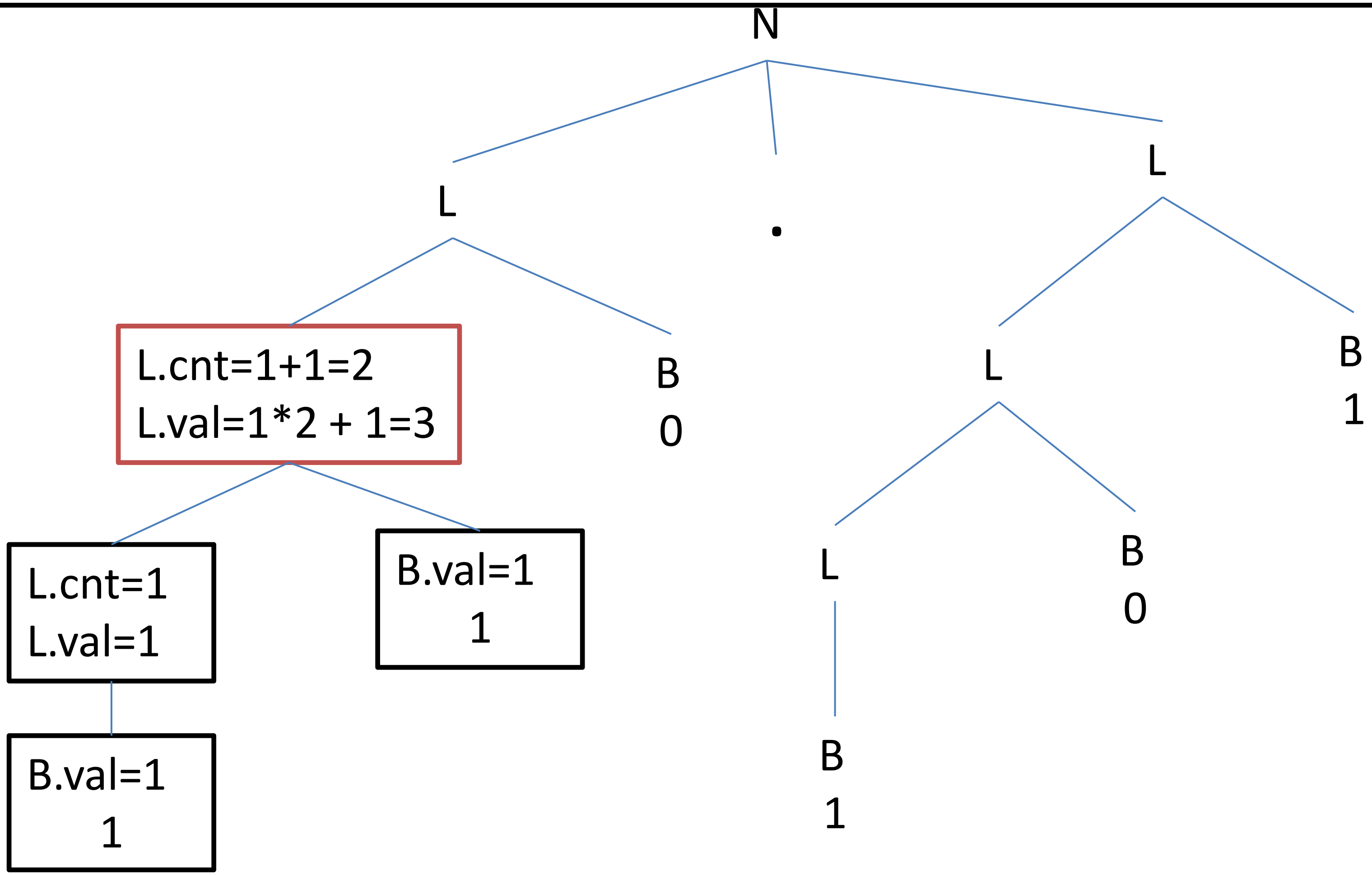
1. $N \rightarrow L_1.L_2$ {$N.val = L_1.val + (L_2.val / 2$^$L_2.cnt)$}
2. $L \rightarrow L_1B$ {$L.cnt=L_1.cnt+1; L.val=L_1.val*2 + B.val$}
3. $L \rightarrow B$ {$L.cnt = 1 ; L.val = B.val$}
4. $B \rightarrow 0$ {$B.val = 0$}
5. $B \rightarrow 1$ {$B.val = 1$}



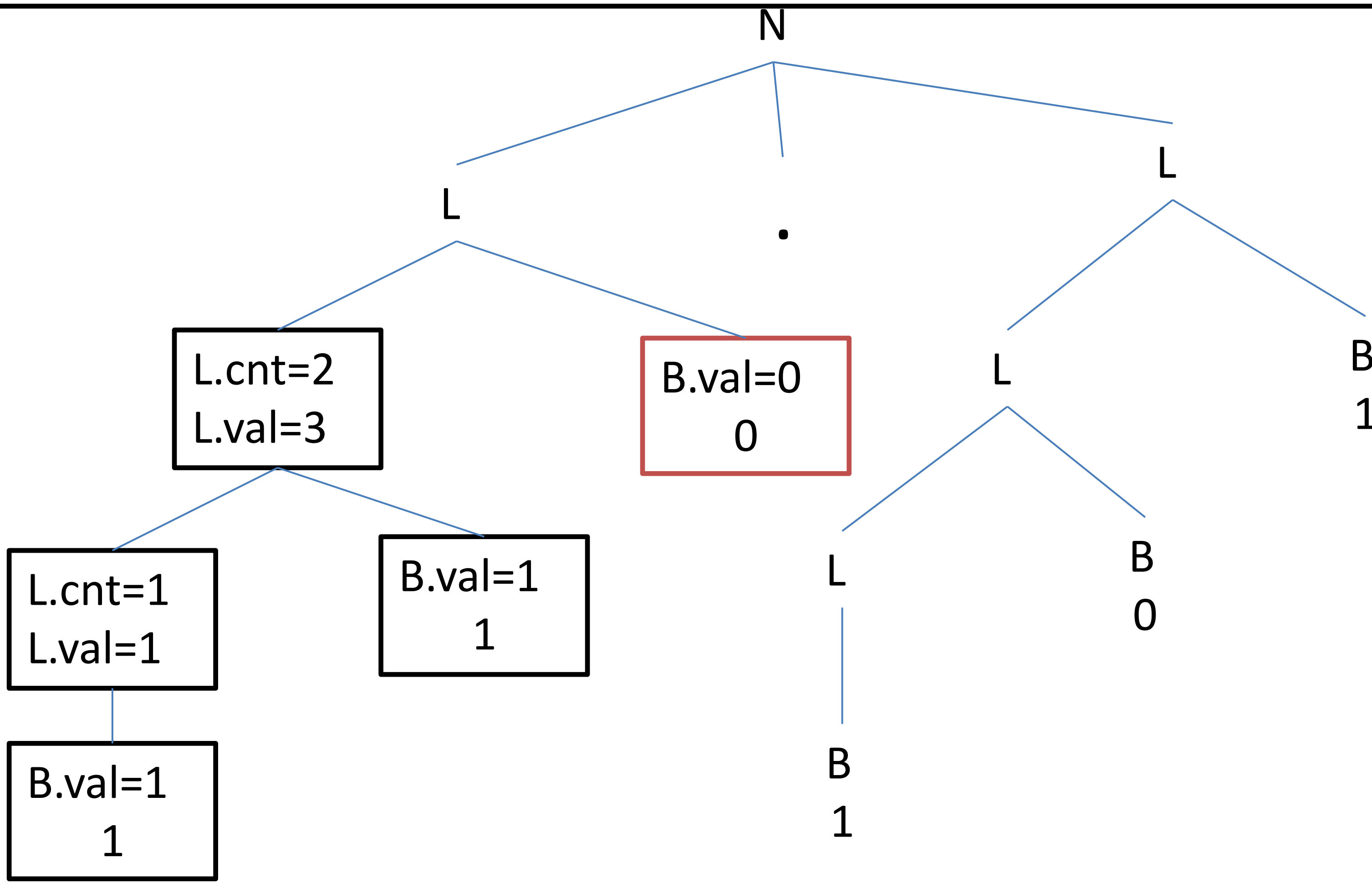
1.	$N \rightarrow L_1.L_2$	$\{N.val = L_1.val + (L_2.val / 2\text{^}L_2.cnt)\}$
2.	$L \rightarrow L_1B$	$\{L.cnt=L_1.cnt+1; L.val=L_1.val*2 + B.val\}$
3.	$L \rightarrow B$	$\{L.cnt = 1 ; L.val = B.val\}$
4.	$B \rightarrow 0$	$\{B.val = 0\}$
5.	$B \rightarrow 1$	$\{B.val = 1\}$



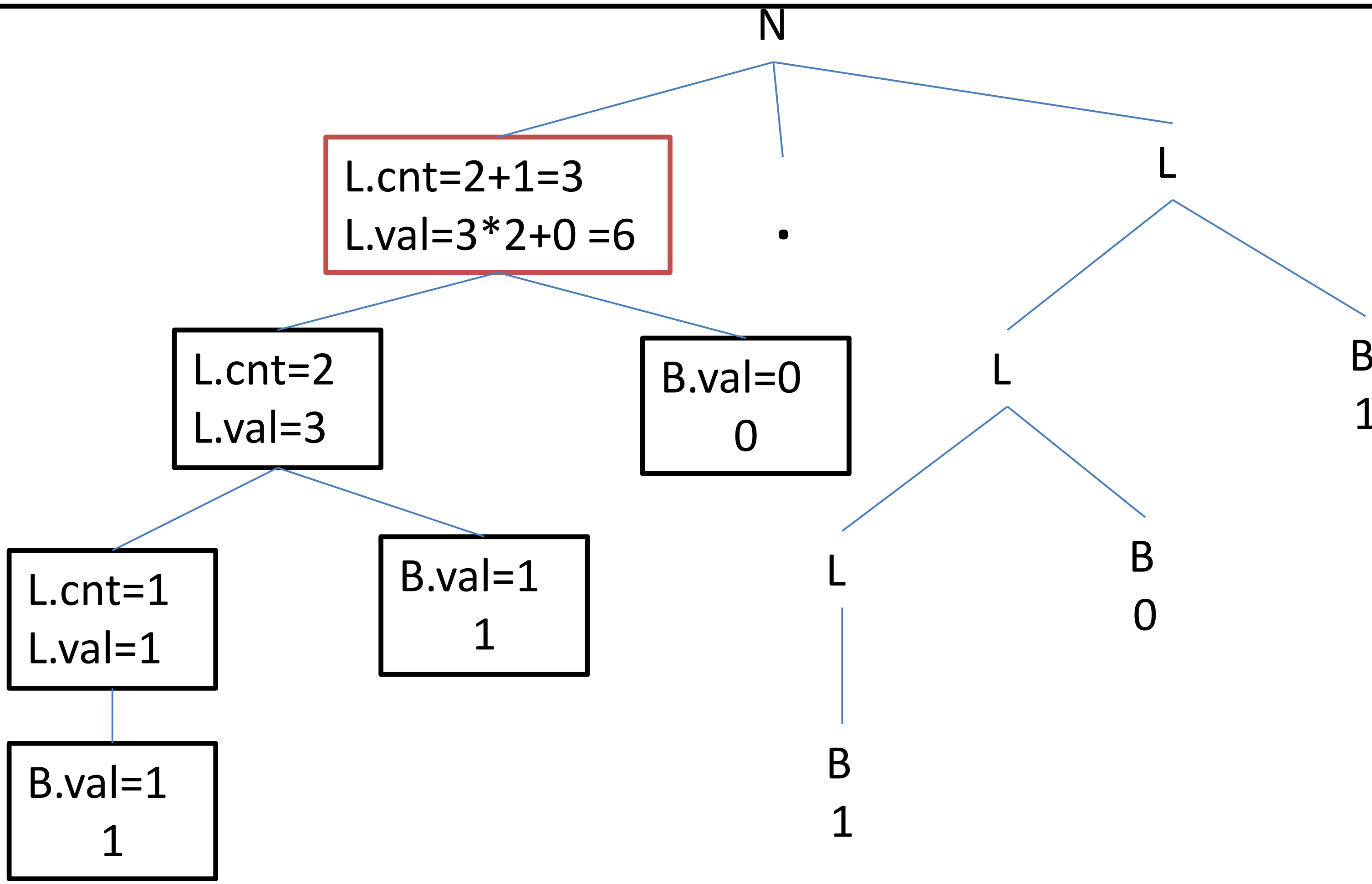
1.	$N \rightarrow L_1.L_2$	{$N.val = L_1.val + (L_2.val / 2$^$L_2.cnt)$}
2.	$L \rightarrow L_1B$	{$L.cnt=L_1.cnt+1; L.val=L_1.val*2 + B.val$}
3.	$L \rightarrow B$	{$L.cnt = 1 ; L.val = B.val$}
4.	$B \rightarrow 0$	{$B.val = 0$}
5.	$B \rightarrow 1$	{$B.val = 1$}



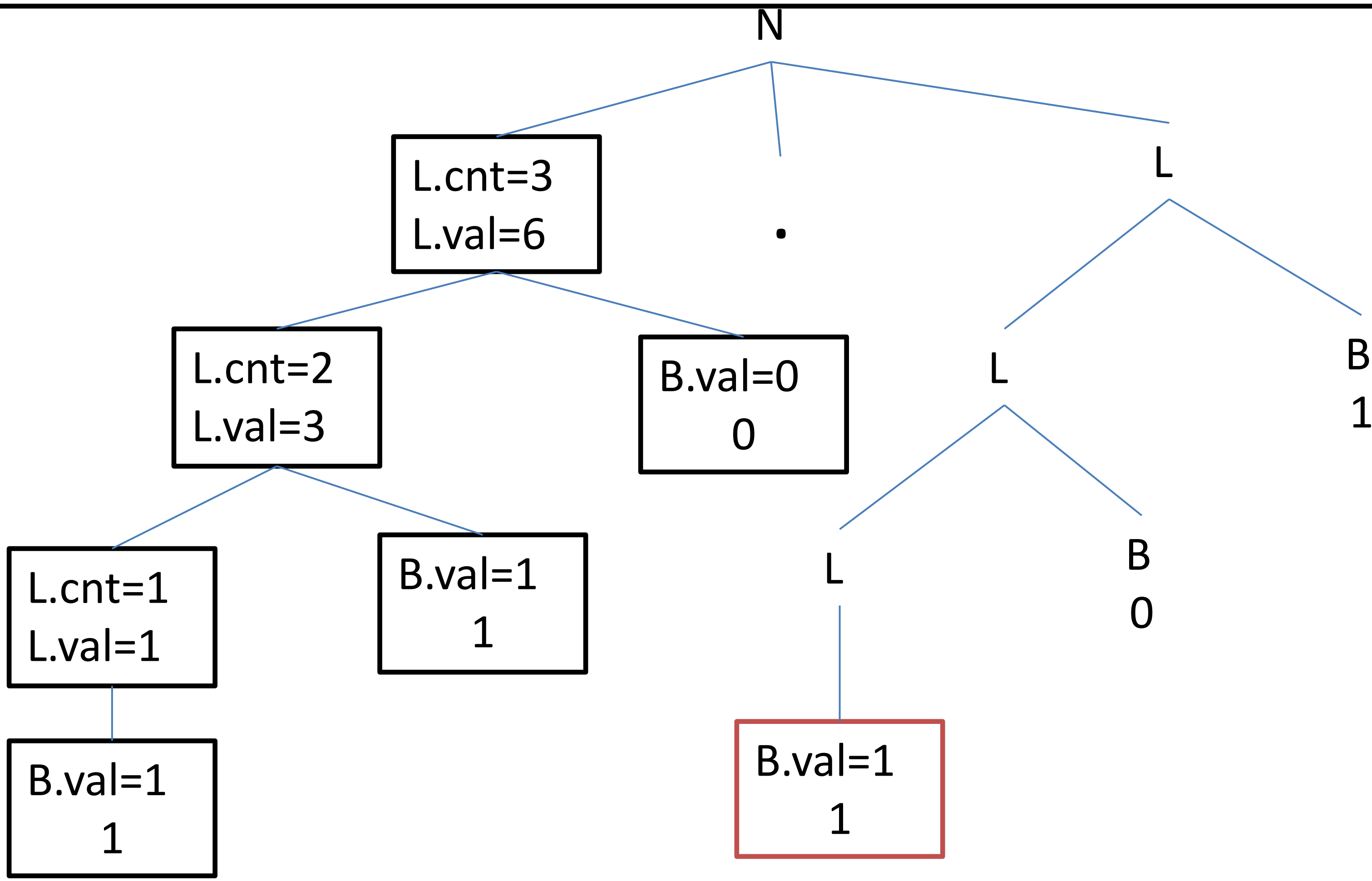
1.	$N \rightarrow L_1.L_2$		{N.val = L_1.val + (L_2.val / 2^L_2.cnt)}
2.	$L \rightarrow L_1B$		{L.cnt=L_1.cnt+1; L.val=L_1.val*2 + B.val}
3.	$L \rightarrow B$		{L.cnt = 1 ; L.val = B.val}
4.	$B \rightarrow 0$		{B.val = 0}
5.	$B \rightarrow 1$		{B.val = 1}



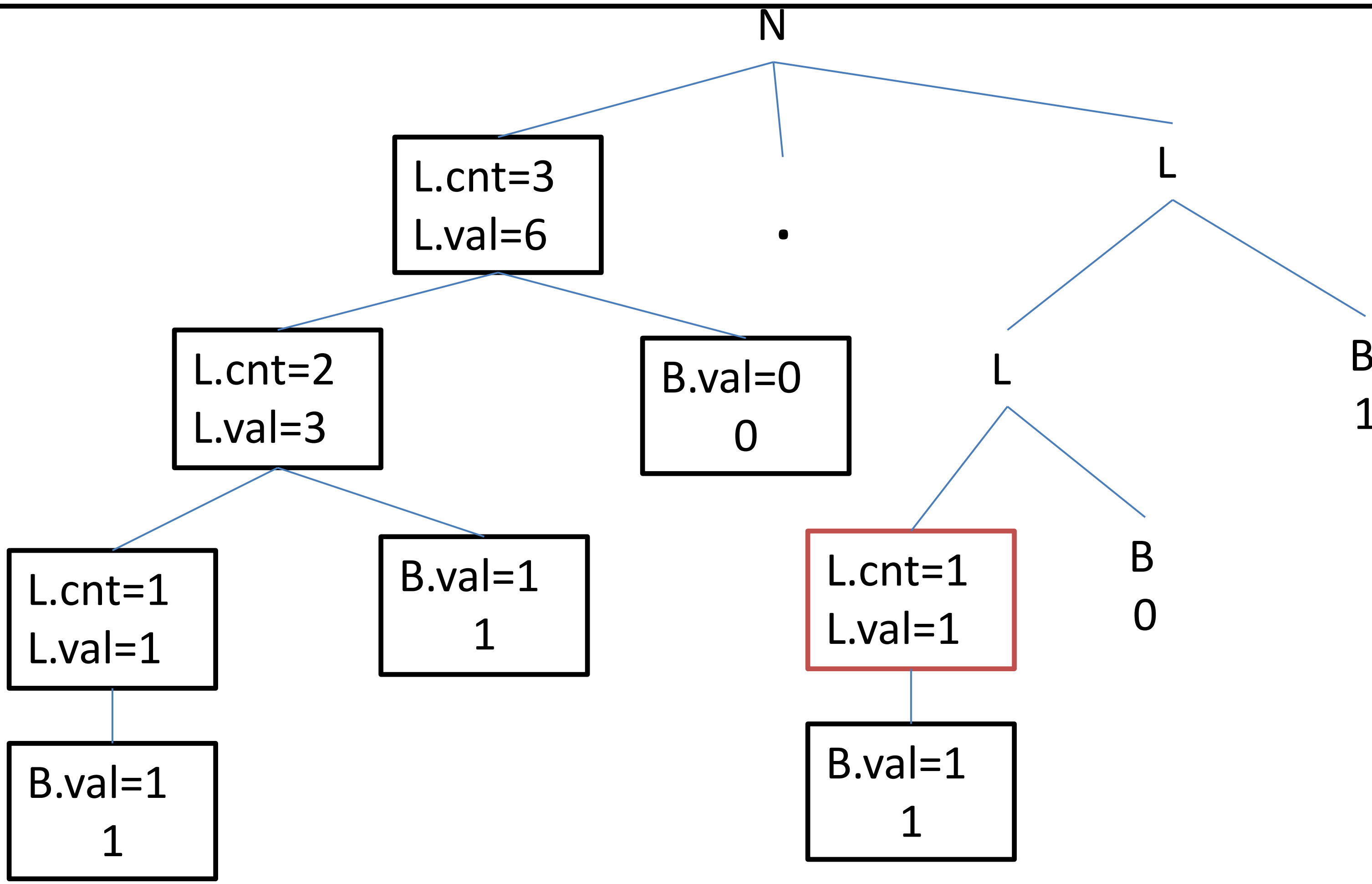
1. $N \rightarrow L_1.L_2$ {$N.val = L_1.val + (L_2.val / 2\hat{\ }L_2.cnt)$}
2. $L \rightarrow L_1B$ {$L.cnt=L_1.cnt+1$; $L.val=L_1.val*2 + B.val$}
3. $L \rightarrow B$ {$L.cnt = 1$; $L.val = B.val$}
4. $B \rightarrow 0$ {$B.val = 0$}
5. $B \rightarrow 1$ {$B.val = 1$}



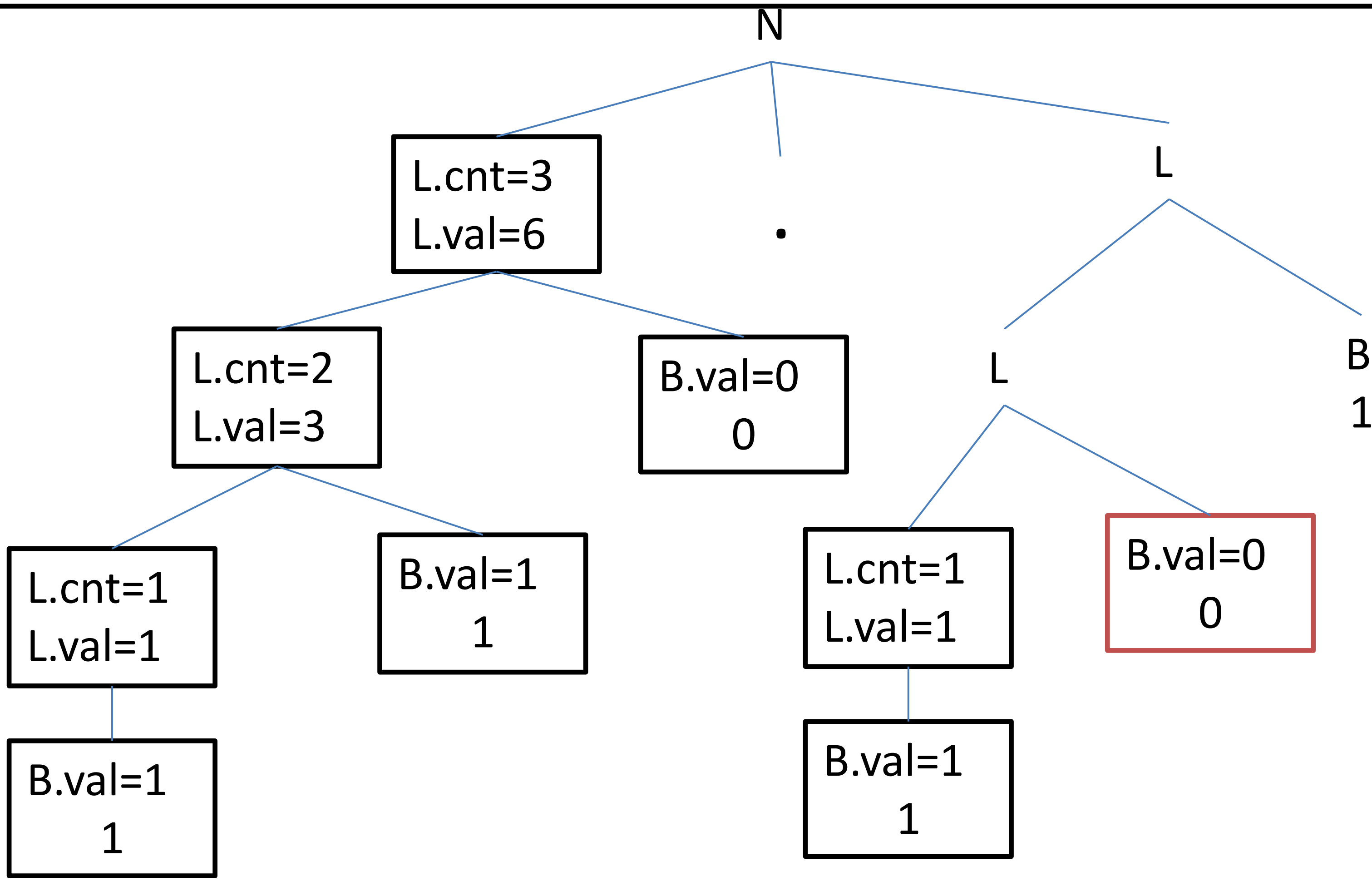
1. $N \rightarrow L_1.L_2$ {N.val = L_1.val + (L_2.val / 2^L_2.cnt)}
2. $L \rightarrow L_1B$ {L.cnt=L_1.cnt+1; L.val=L_1.val*2 + B.val}
3. $L \rightarrow B$ {L.cnt = 1 ; L.val = B.val}
4. $B \rightarrow 0$ {B.val = 0}
5. $B \rightarrow 1$ {B.val = 1}



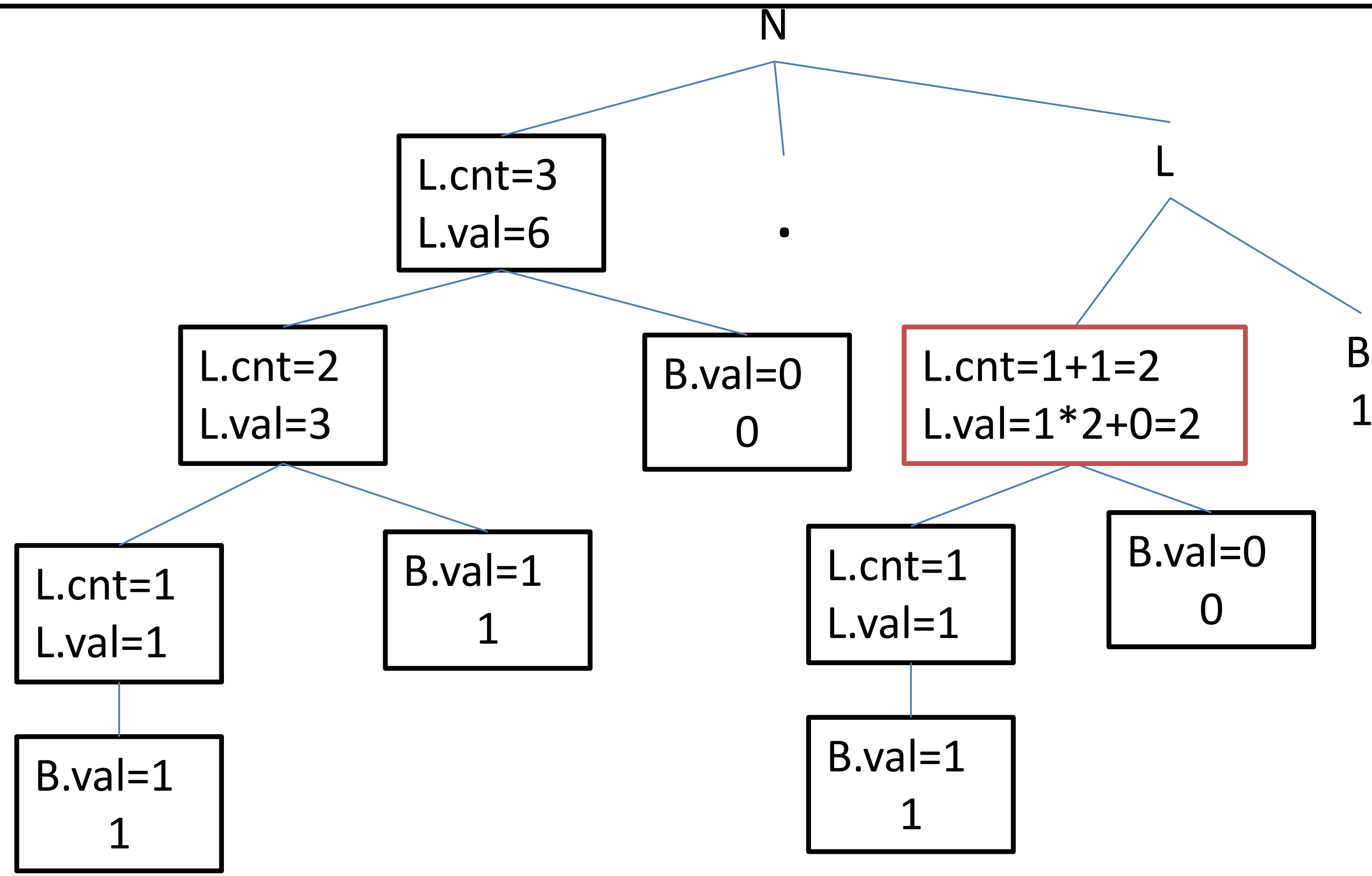
1. $N \rightarrow L_1.L_2$ $\{N.val = L_1.val + (L_2.val / 2\text{^}L_2.cnt)\}$
2. $L \rightarrow L_1B$ $\{L.cnt=L_1.cnt+1; L.val=L_1.val*2 + B.val\}$
3. $L \rightarrow B$ $\{L.cnt = 1 ; L.val = B.val\}$
4. $B \rightarrow 0$ $\{B.val = 0\}$
5. $B \rightarrow 1$ $\{B.val = 1\}$



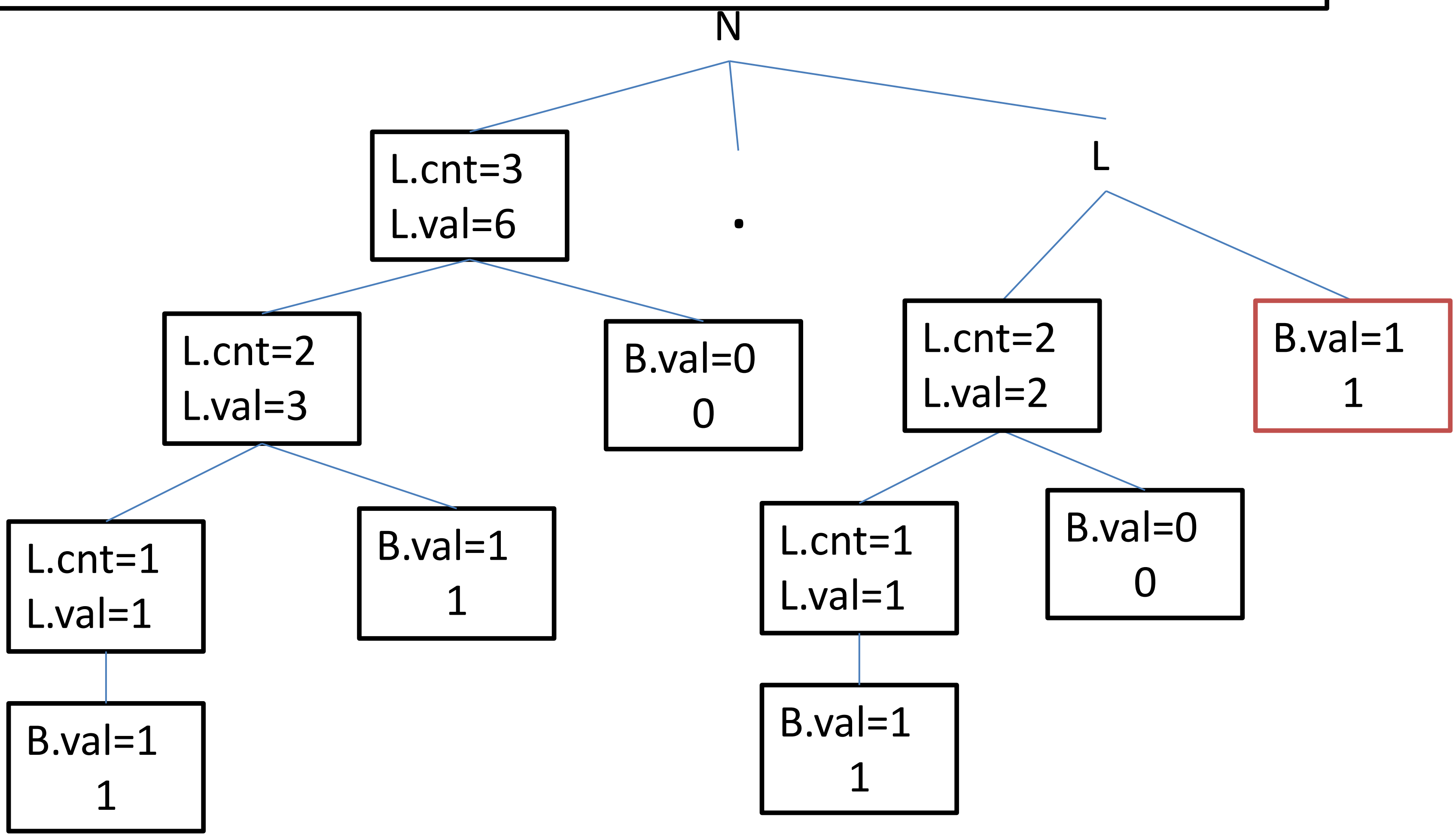
1. $N \rightarrow L_1.L_2$ $\{N.val = L_1.val + (L_2.val / 2^{\wedge}L_2.cnt)\}$
2. $L \rightarrow L_1B$ $\{L.cnt=L_1.cnt+1; L.val=L_1.val*2 + B.val\}$
3. $L \rightarrow B$ $\{L.cnt = 1 ; L.val = B.val\}$
4. $B \rightarrow 0$ $\{B.val = 0\}$
5. $B \rightarrow 1$ $\{B.val = 1\}$



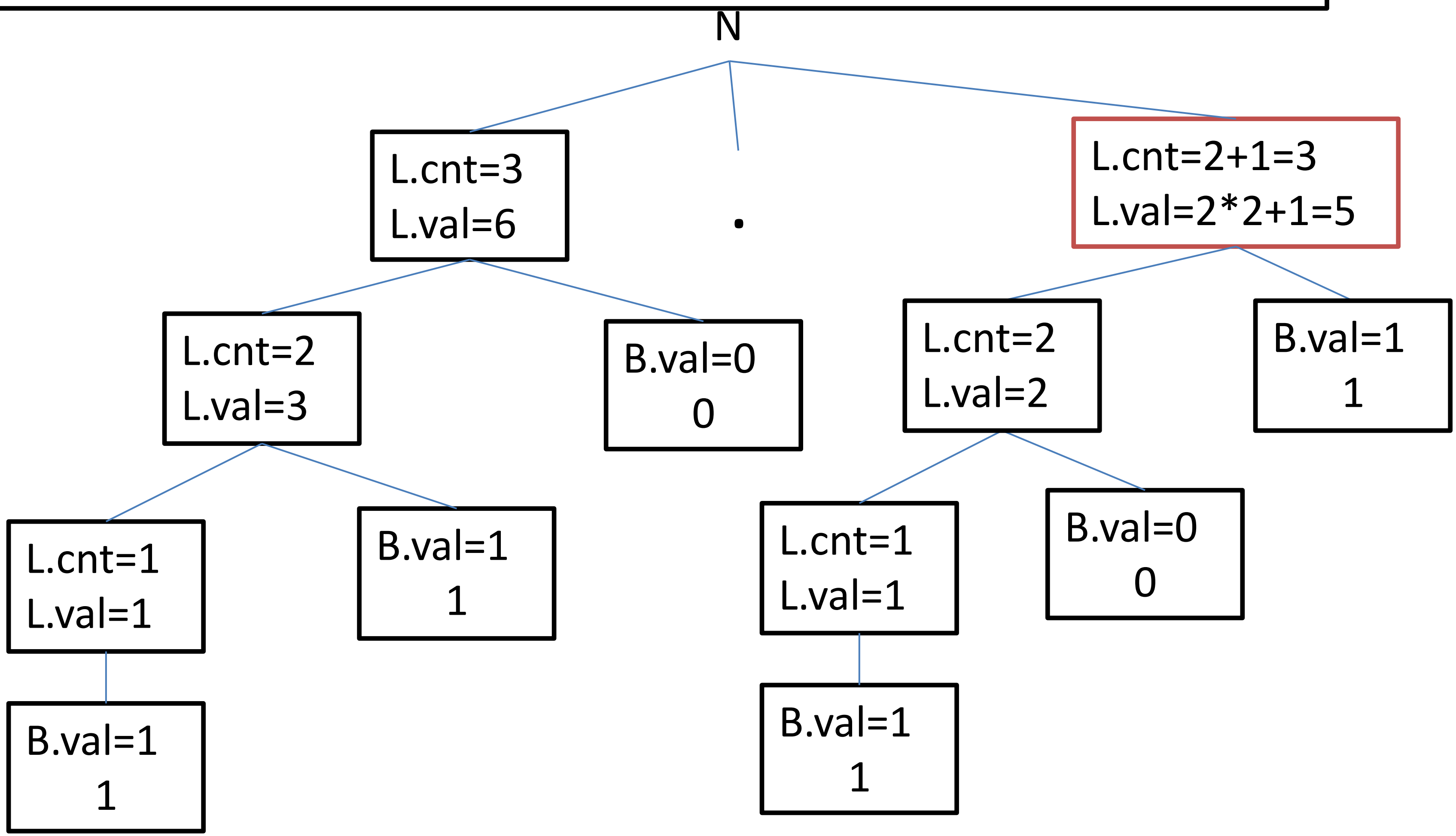
1.	$N \rightarrow L_1.L_2$	$\{N.val = L_1.val + (L_2.val / 2\text{^}L_2.cnt)\}$
2.	$L \rightarrow L_1B$	$\{L.cnt=L_1.cnt+1;\ L.val=L_1.val*2 + B.val\}$
3.	$L \rightarrow B$	$\{L.cnt = 1 ;\ L.val = B.val\}$
4.	$B \rightarrow 0$	$\{B.val = 0\}$
5.	$B \rightarrow 1$	$\{B.val = 1\}$



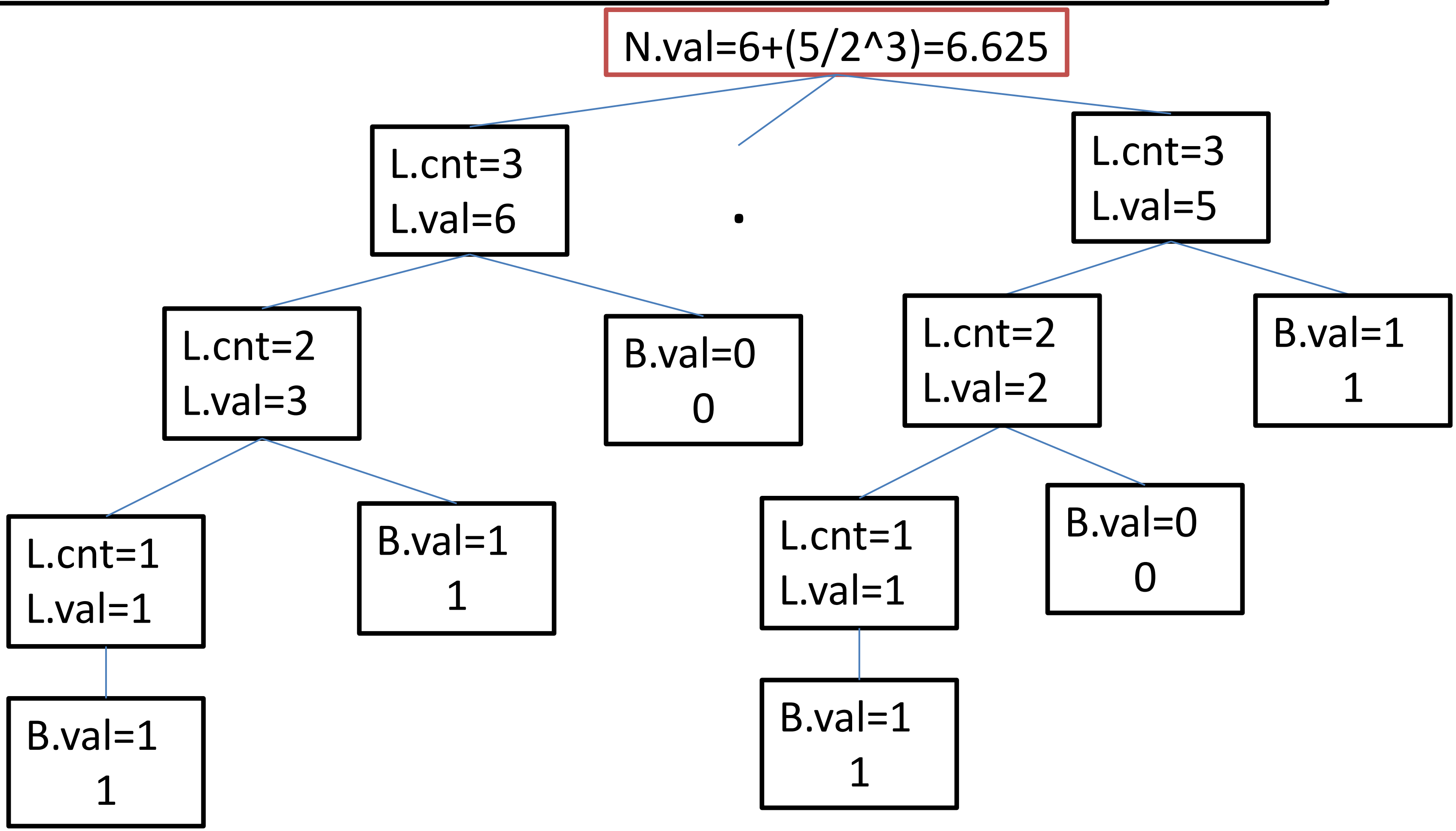
1. $N \rightarrow L_1.L_2$ $\{N.val = L_1.val + (L_2.val / 2\text{^}L_2.cnt)\}$
2. $L \rightarrow L_1B$ $\{L.cnt=L_1.cnt+1; L.val=L_1.val*2 + B.val\}$
3. $L \rightarrow B$ $\{L.cnt = 1 ; L.val = B.val\}$
4. $B \rightarrow 0$ $\{B.val = 0\}$
5. $B \rightarrow 1$ $\{B.val = 1\}$



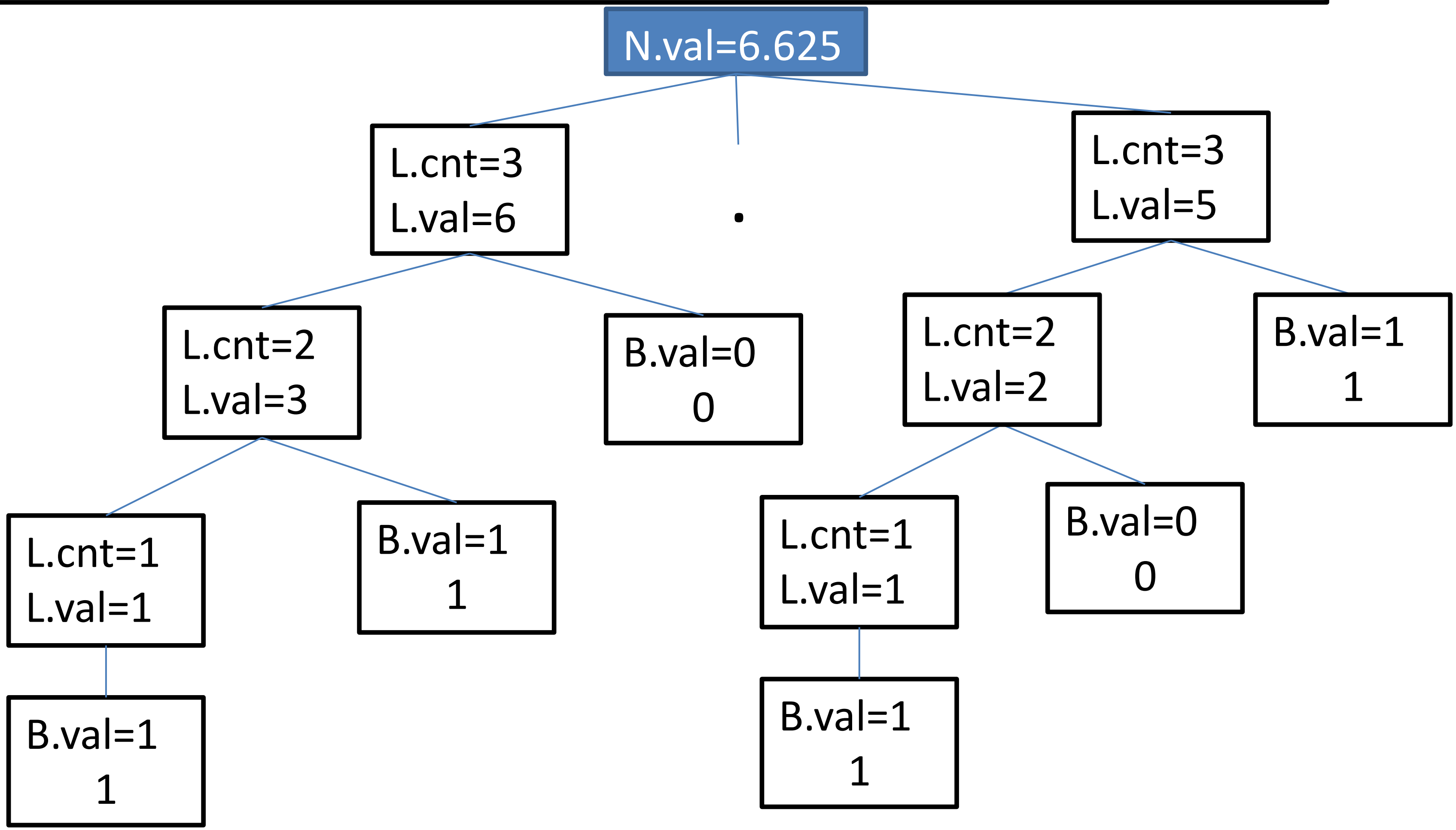
1.	$N \rightarrow L_1.L_2$	{N.val = L_1.val + (L_2.val / 2^L_2.cnt)}
2.	$L \rightarrow L_1B$	{L.cnt=L_1.cnt+1; L.val=L_1.val*2 + B.val}
3.	$L \rightarrow B$	{L.cnt = 1 ; L.val = B.val}
4.	$B \rightarrow 0$	{B.val = 0}
5.	$B \rightarrow 1$	{B.val = 1}



1.	$N \rightarrow L_1.L_2$	$\{N.val = L_1.val + (L_2.val / 2\text{^}L_2.cnt)\}$
2.	$L \rightarrow L_1B$	$\{L.cnt=L_1.cnt+1;\ L.val=L_1.val*2 + B.val\}$
3.	$L \rightarrow B$	$\{L.cnt = 1 ;\ L.val = B.val\}$
4.	$B \rightarrow 0$	$\{B.val = 0\}$
5.	$B \rightarrow 1$	$\{B.val = 1\}$



1. $N \rightarrow L_1.L_2$ {N.val = L_1.val + (L_2.val / 2^L_2.cnt)}
2. $L \rightarrow L_1 B$ {L.cnt=L_1.cnt+1; L.val=L_1.val*2 + B.val}
3. $L \rightarrow B$ {L.cnt = 1 ; L.val = B.val}
4. $B \rightarrow 0$ {B.val = 0}
5. $B \rightarrow 1$ {B.val = 1}



Example 3

- Given a grammar:

  E → E + T | T

  T → T * F | F

  F → num

- What are the semantic rules(informal notations) for this grammar?
  - There can be a ‘value’ attribute for E, T and F. (non-terminals)
  - There can be a ‘lexvalue’ attribute for num (terminal)

Example 3

- Given a grammar:

  $E \rightarrow E + T \mid T$

  $T \rightarrow T * F \mid F$

  $F \rightarrow num$

1. $E \rightarrow E + T$ { E.value = E.value + T.value}
2. $E \rightarrow T$ {E.value = T.value}
3. $T \rightarrow T * F$ {T.value = T.value * F.value}
4. $T \rightarrow F$ {T.value = F.value}
5. $F \rightarrow num$ {F.value = num.lexvalue}

Example 3

1. $E \rightarrow E + T$ { E.value = E.value + T.value}
2. $E \rightarrow T$ {E.value = T.value}
3. $T \rightarrow T * F$ {T.value = T.value * F.value}
4. $T \rightarrow F$ {T.value = F.value}
5. $F \rightarrow num$ {F.value = num.lexvalue}

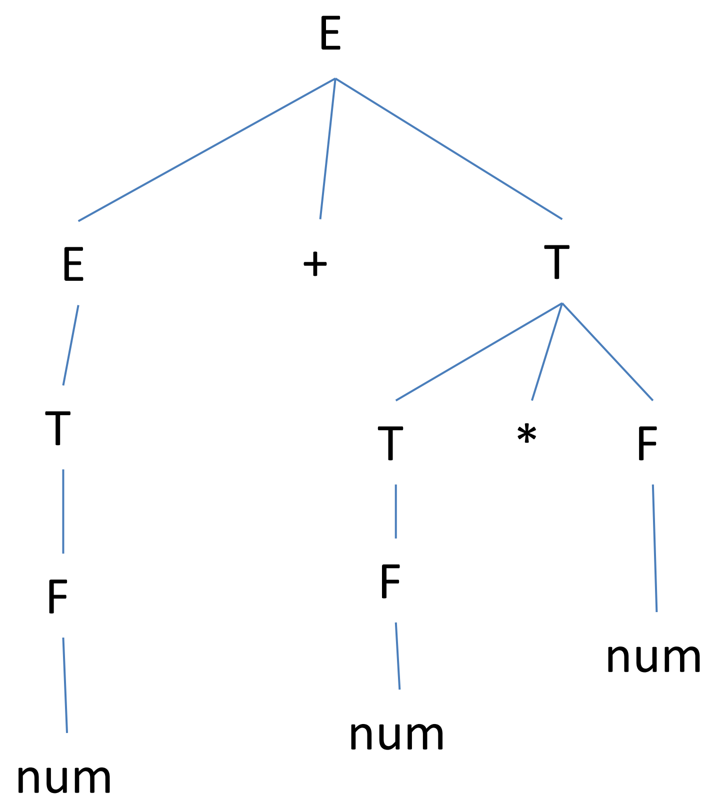
For input, 2 + 3 * 4

- Parse Tree??

Example 3

1. E → E + T { E.value = E.value + T.value}
2. E → T {E.value = T.value}
3. T → T * F {T.value = T.value * F.value}
4. T → F {T.value = F.value}
5. F → num {F.value = num.lexvalue}

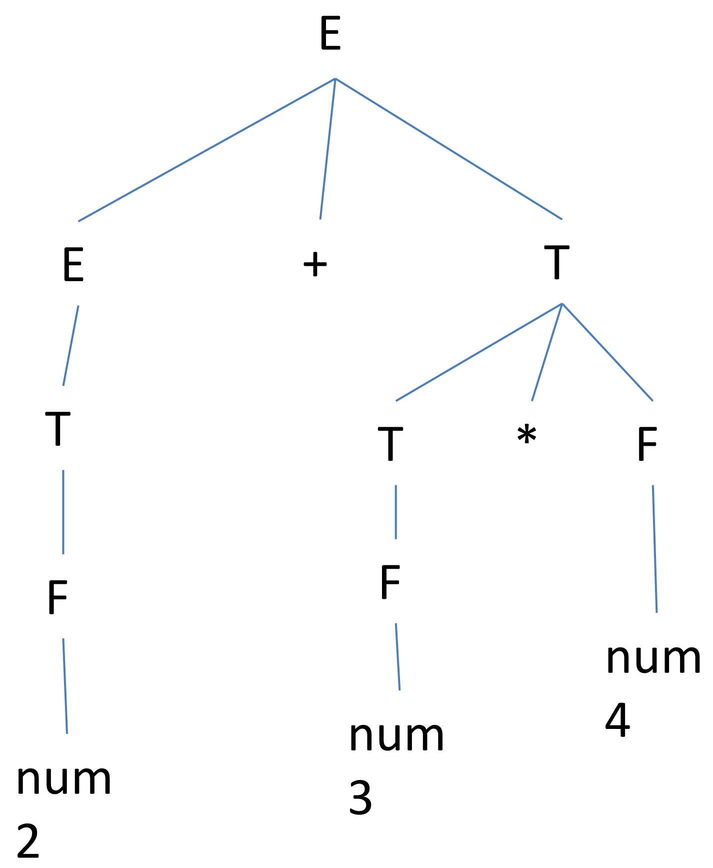
For input, 2 + 3 * 4



Example 3

1. E → E + T { E.value = E.value + T.value}
2. E → T {E.value = T.value}
3. T → T * F {T.value = T.value * F.value}
4. T → F {T.value = F.value}
5. F → num {F.value = num.lexvalue}

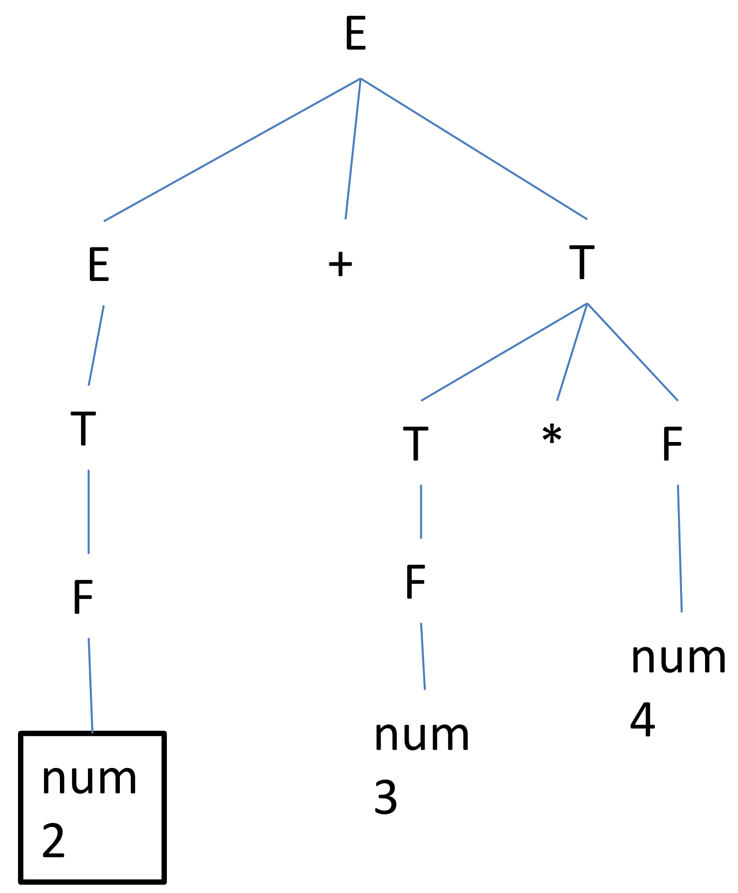
For input, 2 + 3 * 4



Example 3

1. $E \rightarrow E + T$ { E.value = E.value + T.value}
2. $E \rightarrow T$ {E.value = T.value}
3. $T \rightarrow T * F$ {T.value = T.value * F.value}
4. $T \rightarrow F$ {T.value = F.value}
5. $F \rightarrow num$ {F.value = num.lexvalue}

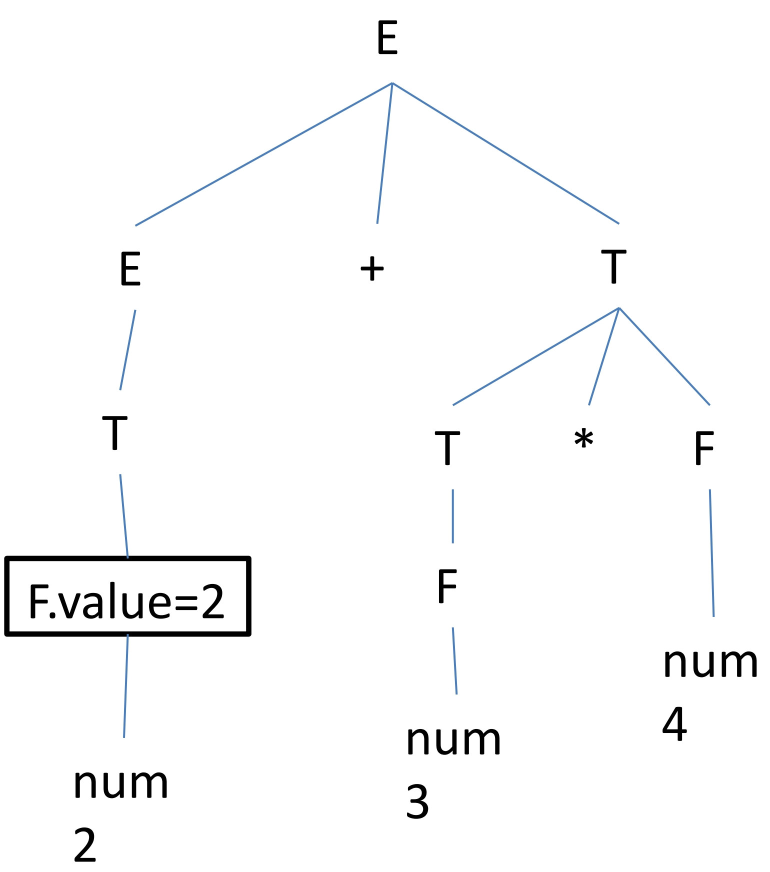
For input, 2 + 3 * 4



Example 3

1. $E \rightarrow E + T$ { E.value = E.value + T.value}
2. $E \rightarrow T$ {E.value = T.value}
3. $T \rightarrow T * F$ {T.value = T.value * F.value}
4. $T \rightarrow F$ {T.value = F.value}
5. $F \rightarrow num$ {F.value = num.lexvalue}

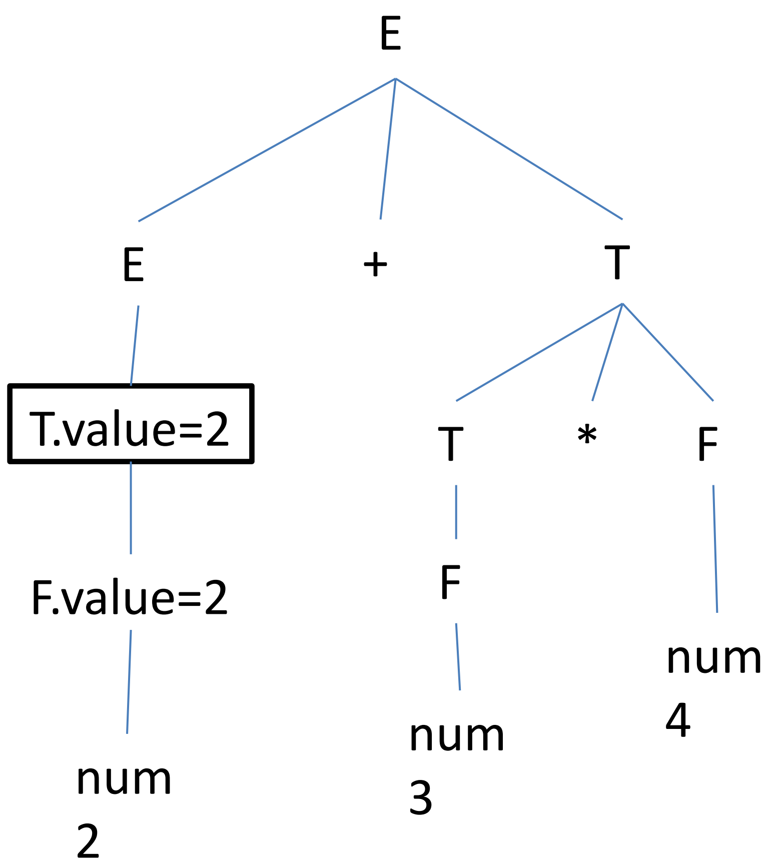
For input, 2 + 3 * 4



Example 3

1. E $\rightarrow$ E + T { E.value = E.value + T.value}
2. E $\rightarrow$ T {E.value = T.value}
3. T $\rightarrow$ T * F {T.value = T.value * F.value}
4. T $\rightarrow$ F {T.value = F.value}
5. F $\rightarrow$ num {F.value = num.lexvalue}

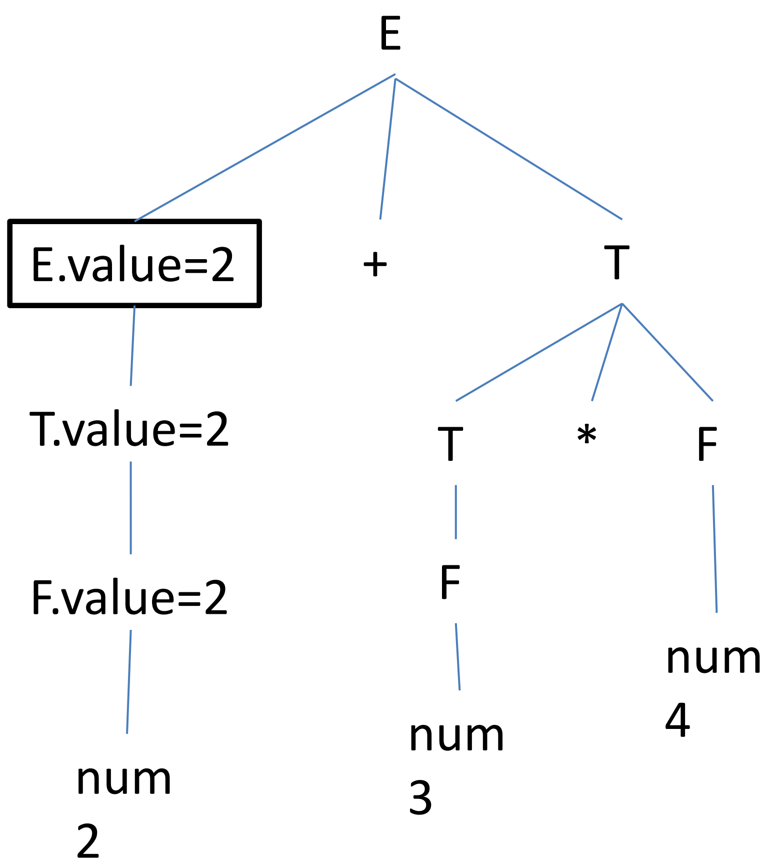
For input, 2 + 3 * 4



Example 3

1. E → E + T { E.value = E.value + T.value}
2. E → T {E.value = T.value}
3. T → T * F {T.value = T.value * F.value}
4. T → F {T.value = F.value}
5. F → num {F.value = num.lexvalue}

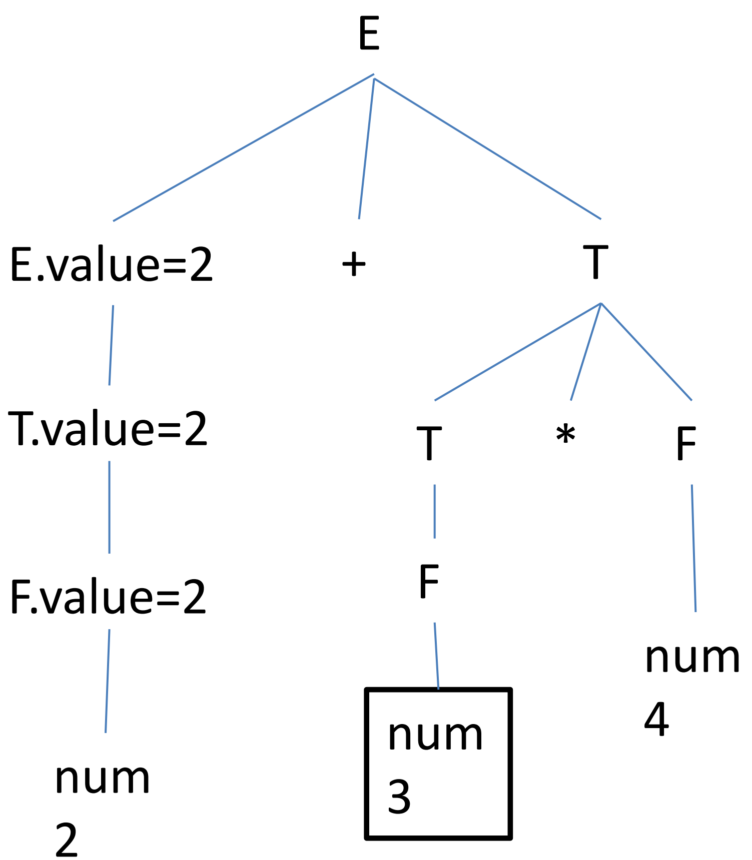
For input, 2 + 3 * 4



Example 3

1. E → E + T { E.value = E.value + T.value}
2. E → T {E.value = T.value}
3. T → T * F {T.value = T.value * F.value}
4. T → F {T.value = F.value}
5. F → num {F.value = num.lexvalue}

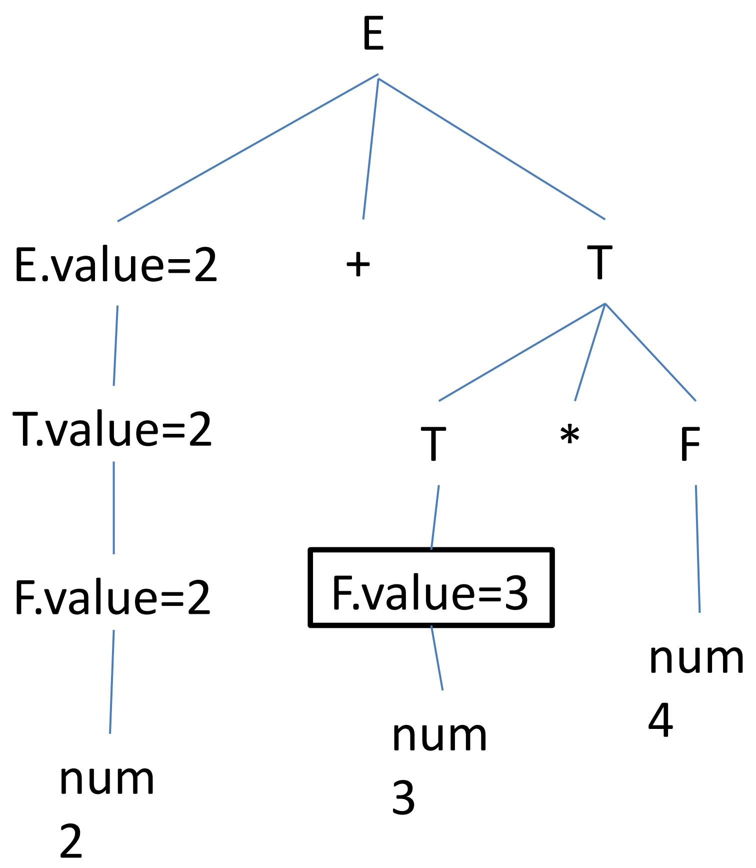
For input, 2 + 3 * 4



Example 3

1. $E \rightarrow E + T$ { E.value = E.value + T.value}
2. $E \rightarrow T$ {E.value = T.value}
3. $T \rightarrow T * F$ {T.value = T.value * F.value}
4. $T \rightarrow F$ {T.value = F.value}
5. $F \rightarrow num$ {F.value = num.lexvalue}

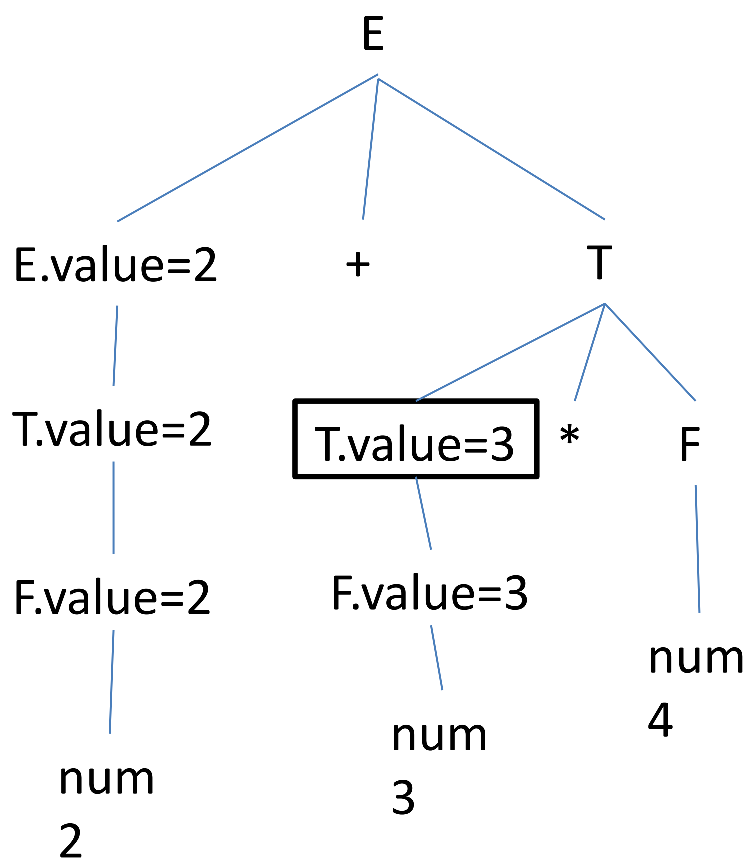
For input, 2 + 3 * 4



Example 3

1. E → E + T { E.value = E.value + T.value}
2. E → T {E.value = T.value}
3. T → T * F {T.value = T.value * F.value}
4. T → F {T.value = F.value}
5. F → num {F.value = num.lexvalue}

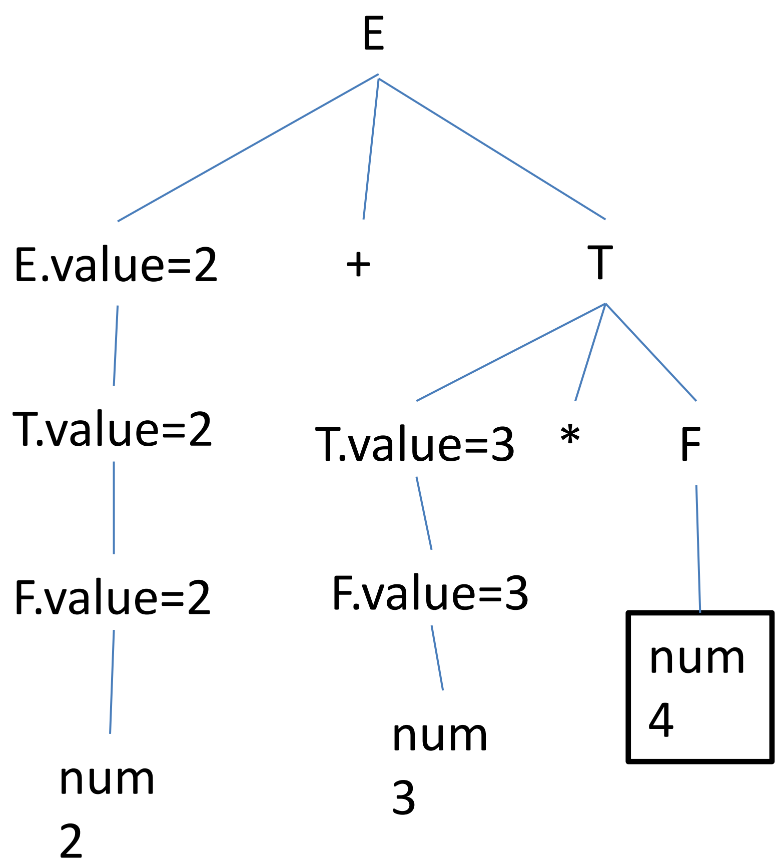
For input, 2 + 3 * 4



Example 3

1. E → E + T { E.value = E.value + T.value}
2. E → T {E.value = T.value}
3. T → T * F {T.value = T.value * F.value}
4. T → F {T.value = F.value}
5. F → num {F.value = num.lexvalue}

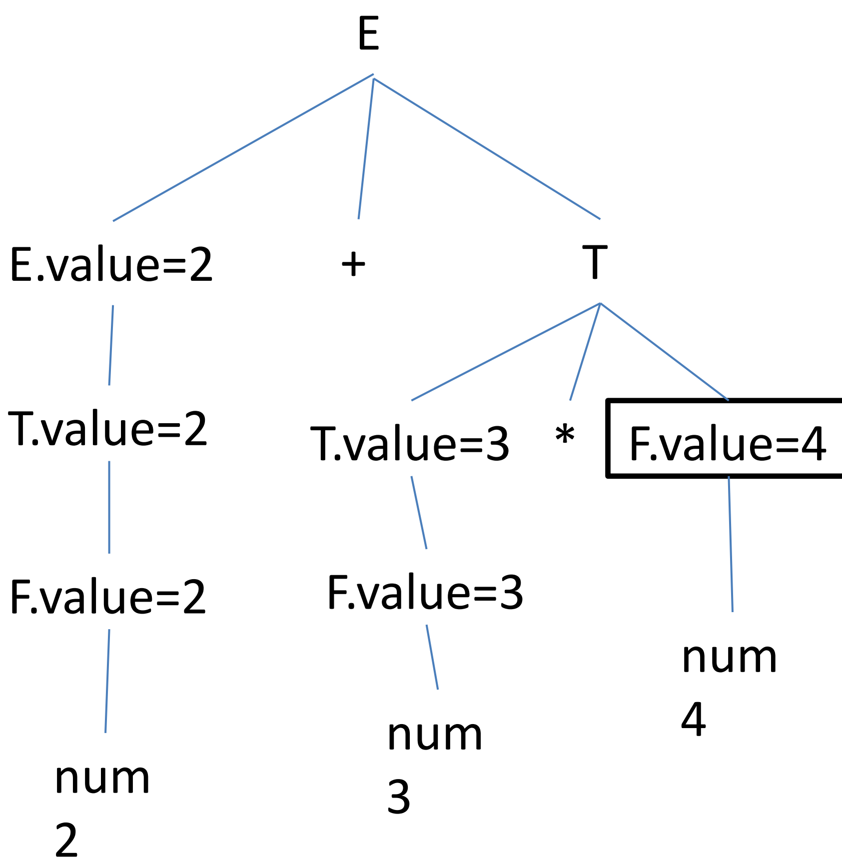
For input, 2 + 3 * 4



Example 3

1. E $\rightarrow$ E + T { E.value = E.value + T.value}
2. E $\rightarrow$ T {E.value = T.value}
3. T $\rightarrow$ T * F {T.value = T.value * F.value}
4. T $\rightarrow$ F {T.value = F.value}
5. F $\rightarrow$ num {F.value = num.lexvalue}

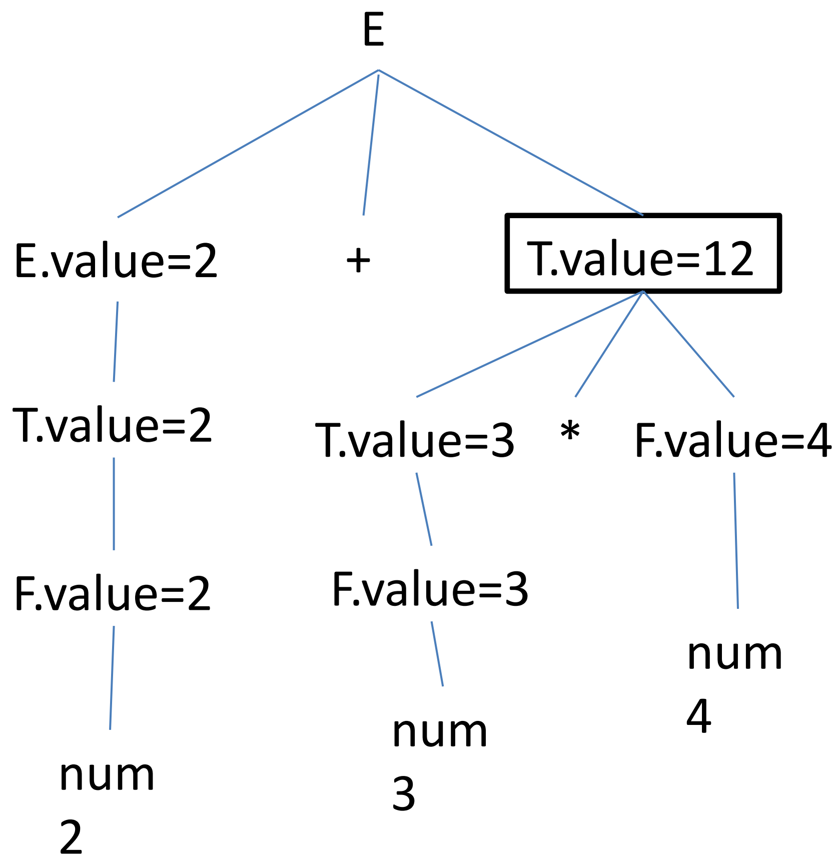
For input, 2 + 3 * 4



Example 3

1. E → E + T { E.value = E.value + T.value}
2. E → T {E.value = T.value}
3. T → T * F {T.value = T.value * F.value}
4. T → F {T.value = F.value}
5. F → num {F.value = num.lexvalue}

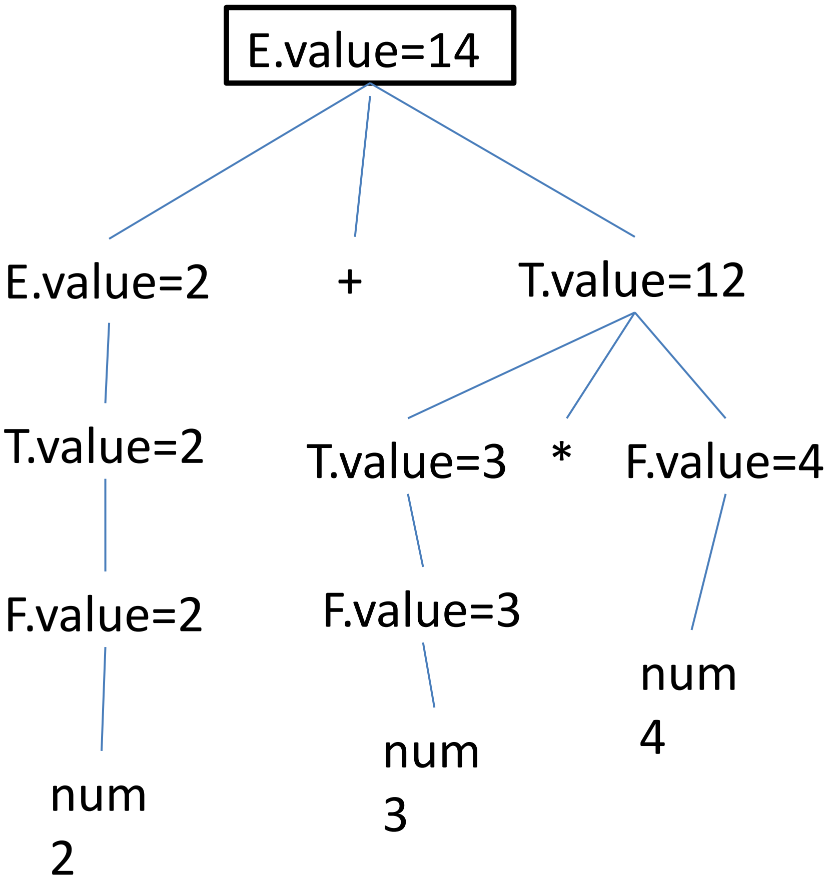
For input, 2 + 3 * 4



Example 3

1. E → E + T { E.value = E.value + T.value}
2. E → T {E.value = T.value}
3. T → T * F {T.value = T.value * F.value}
4. T → F {T.value = F.value}
5. F → num {F.value = num.lexvalue}

For input, 2 + 3 * 4



Example 4

- Write SDT to convert infix to postfix

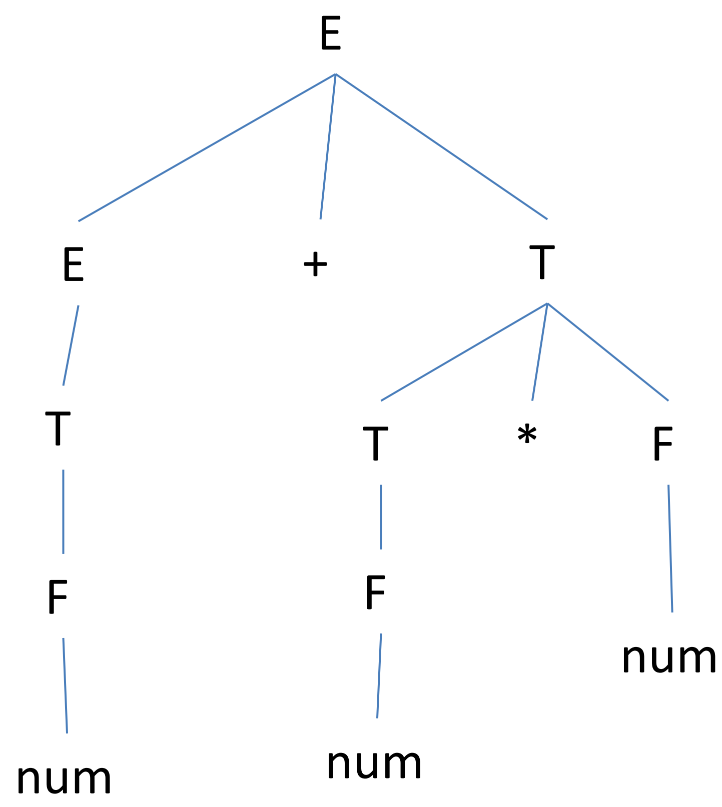
For input, 2 + 3 * 4
Output: 2 3 4 * +

Example 4

- SDT to convert infix to postfix

E→ E + T	{printf("+");}
E → T	{}
T → T * F	{printf("*");}
T→F	{}
F→ num	{printf(num.lval);}

For input, 2 + 3 * 4

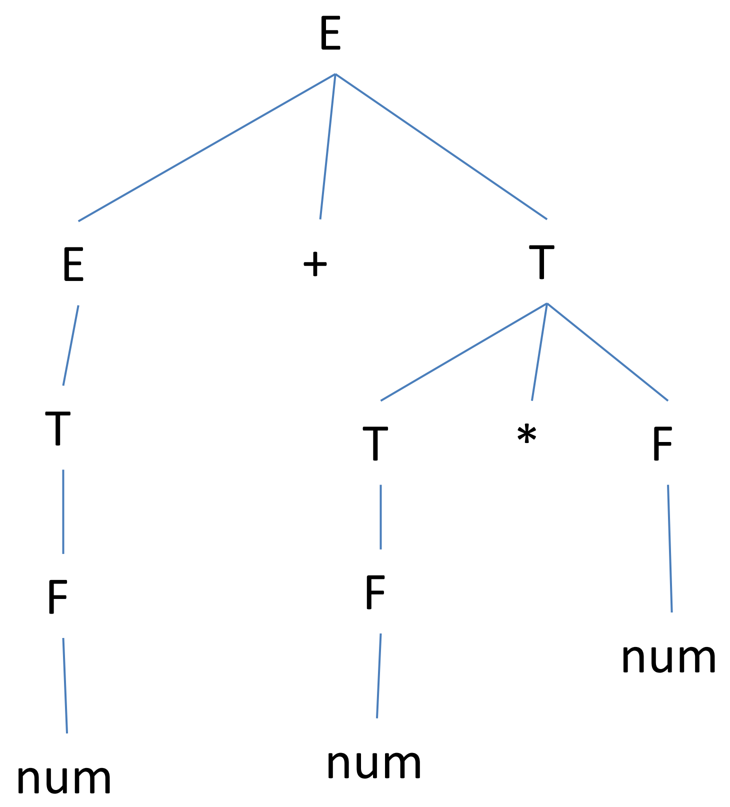


Example 4

- SDT to convert infix to postfix

E→ E + T	{printf("+");}	1
E → T	{}	2
T → T * F	{printf("*");}	3
T→F	{}	4
F→ num	{printf(num.lval);}	5

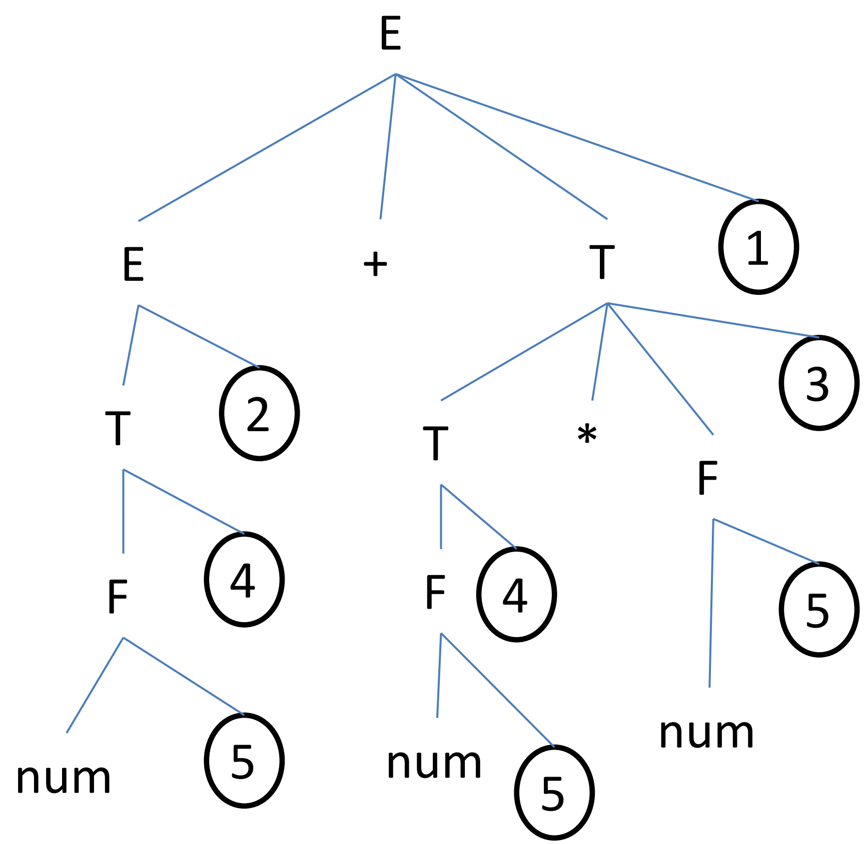
For input, 2 + 3 * 4



Example 4 (taking top-down approach)

- SDT to convert infix to postfix

E → E + T	{printf("+");}	1
E → T	{}	2
T → T * F	{printf("*");}	3
T → F	{}	4
F → num	{printf(num.lval);}	5



For input, 2 + 3 * 4

Example 4 (taking top-down approach)

- SDT to convert infix to postfix

E→ E + T {printf("+");} 1

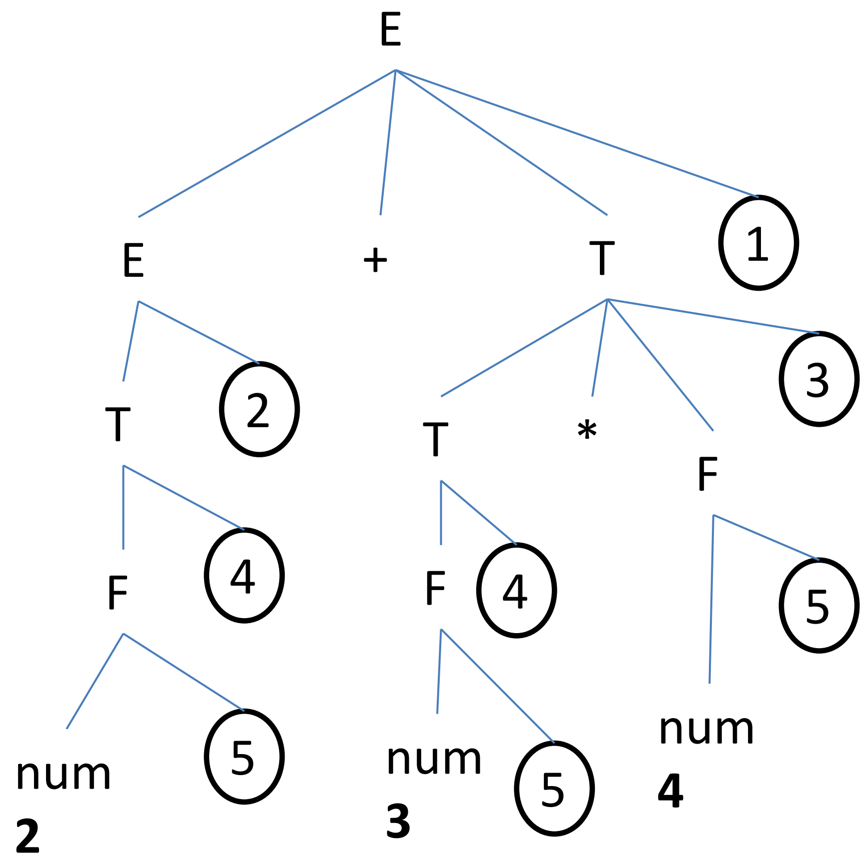
E → T {} 2

T → T * F {printf("*");} 3

T→F {} 4

F→ num {printf(num.lval);} 5

For input, 2 + 3 * 4



Example 4 (taking top-down approach)

- SDT to convert infix to postfix

E→ E + T {printf("+");} 1

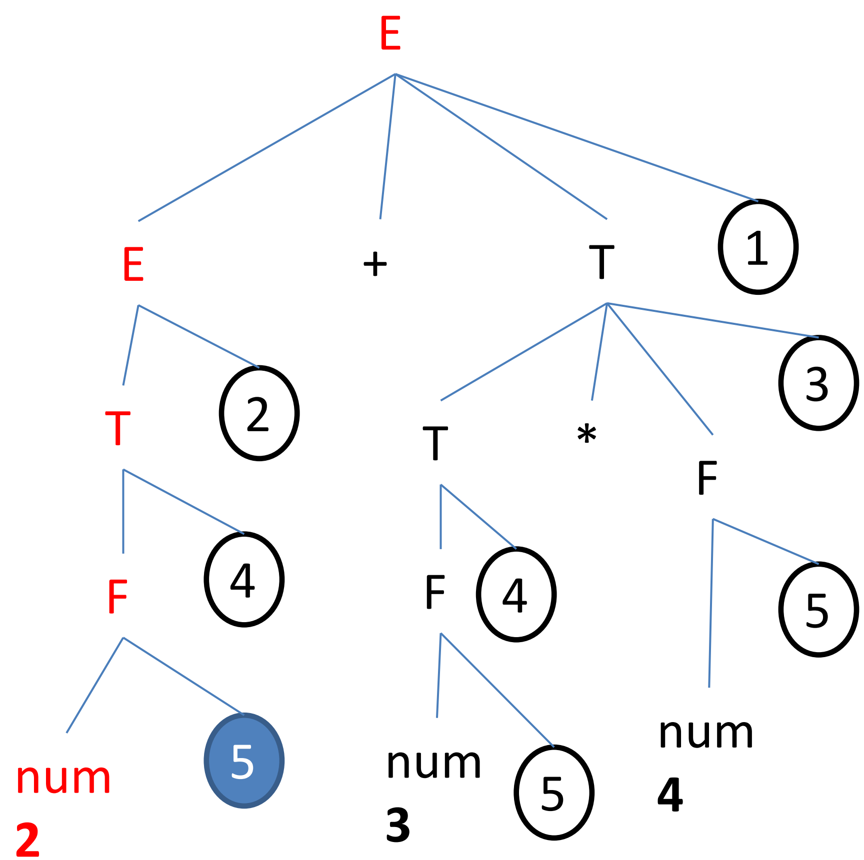
E → T {} 2

T → T * F {printf("*");} 3

T→F {} 4

F→ num {printf(num.lval);}

5



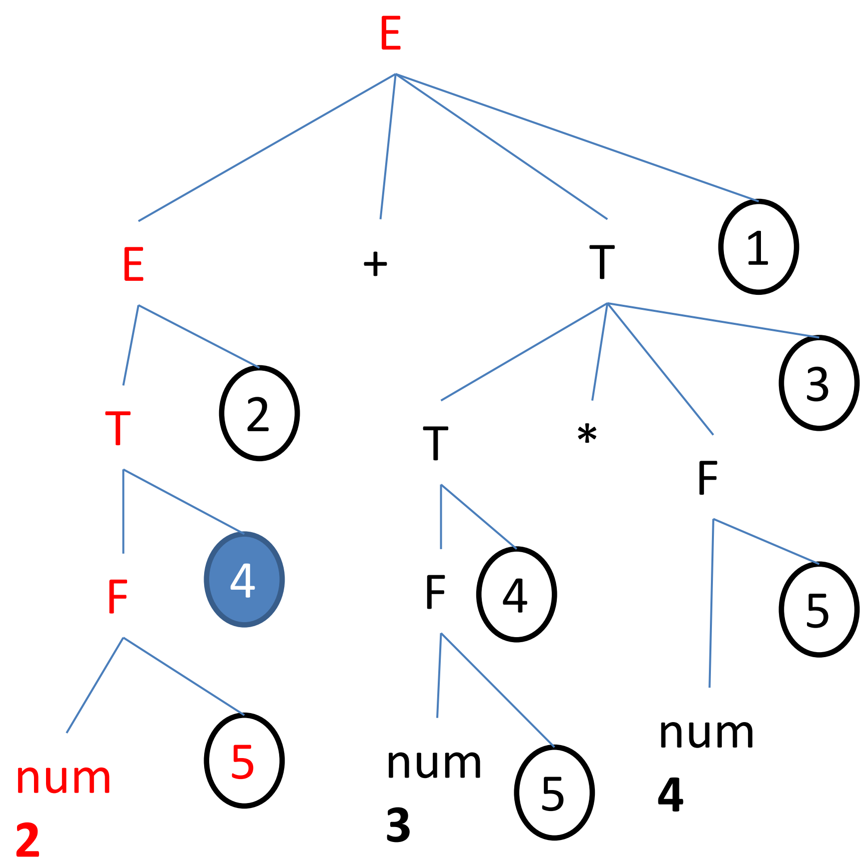
For input, 2 + 3 * 4

Output: 2

Example 4 (taking top-down approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}	1
E → T	{}	2
T → T * F	{printf("*");}	3
T→F	{}	4
F→ num	{printf(num.lval);}	5



For input, 2 + 3 * 4

Output: 2

Example 4 (taking top-down approach)

- SDT to convert infix to postfix

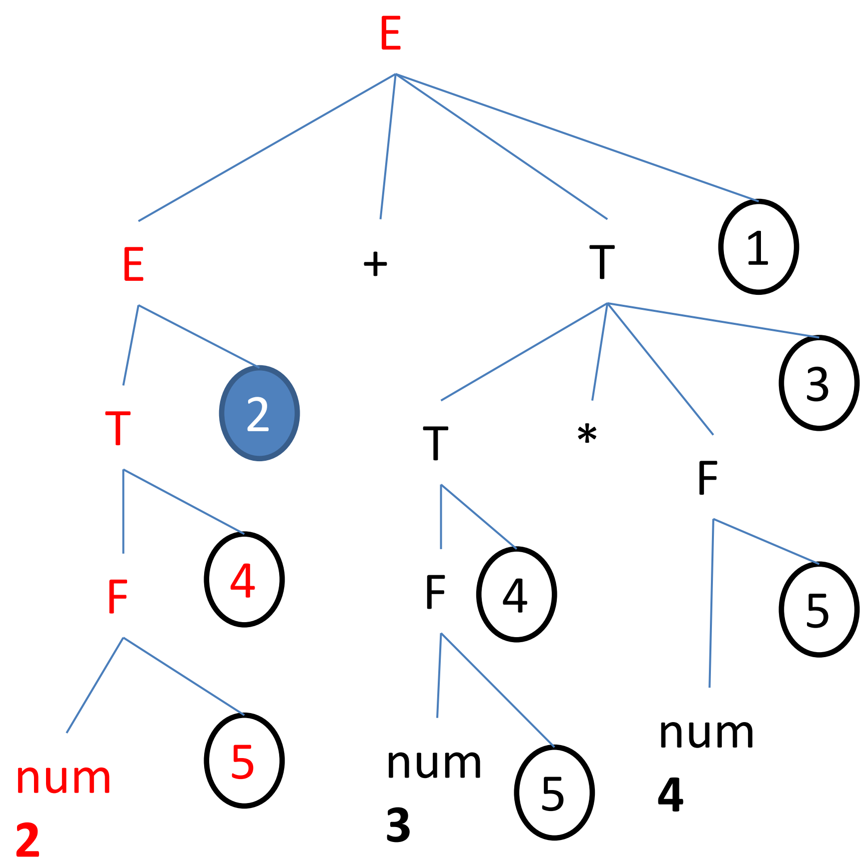
E→ E + T {printf("+");} 1

E → T {} 2

T → T * F {printf("*");} 3

T→F {} 4

F→ num {printf(num.lval);} 5



For input, 2 + 3 * 4

Output: 2

Example 4 (taking top-down approach)

- SDT to convert infix to postfix

E→ E + T {printf("+");} 1

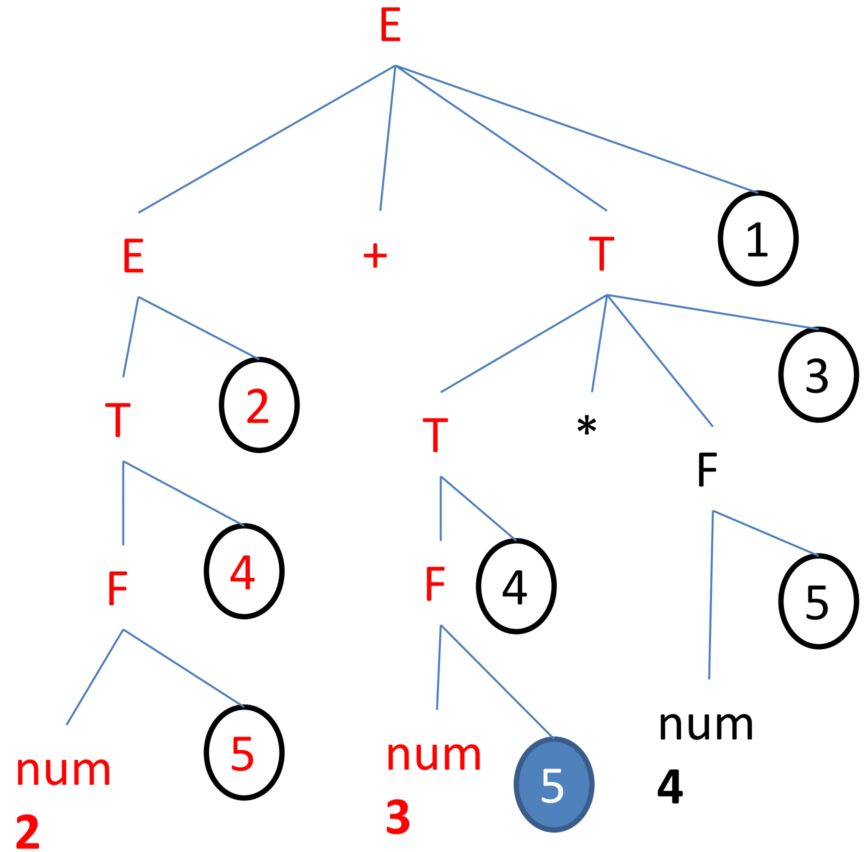
E → T {} 2

T → T * F {printf("*");} 3

T→F {} 4

F→ num {printf(num.lval);}

5



For input, 2 + 3 * 4

Output: 2 3

Example 4 (taking top-down approach)

- SDT to convert infix to postfix

E→ E + T {printf("+");} 1

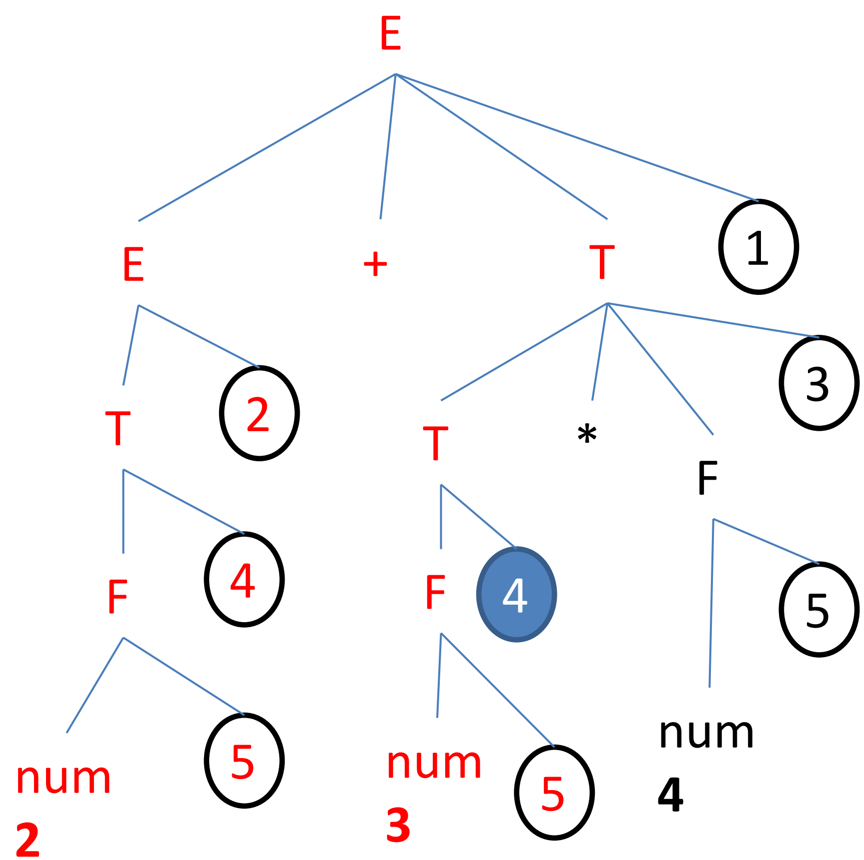
E → T {} 2

T → T * F {printf("*");} 3

T→F {} 4

F→ num {printf(num.lval);}

5



For input, 2 + 3 * 4

Output: 2 3

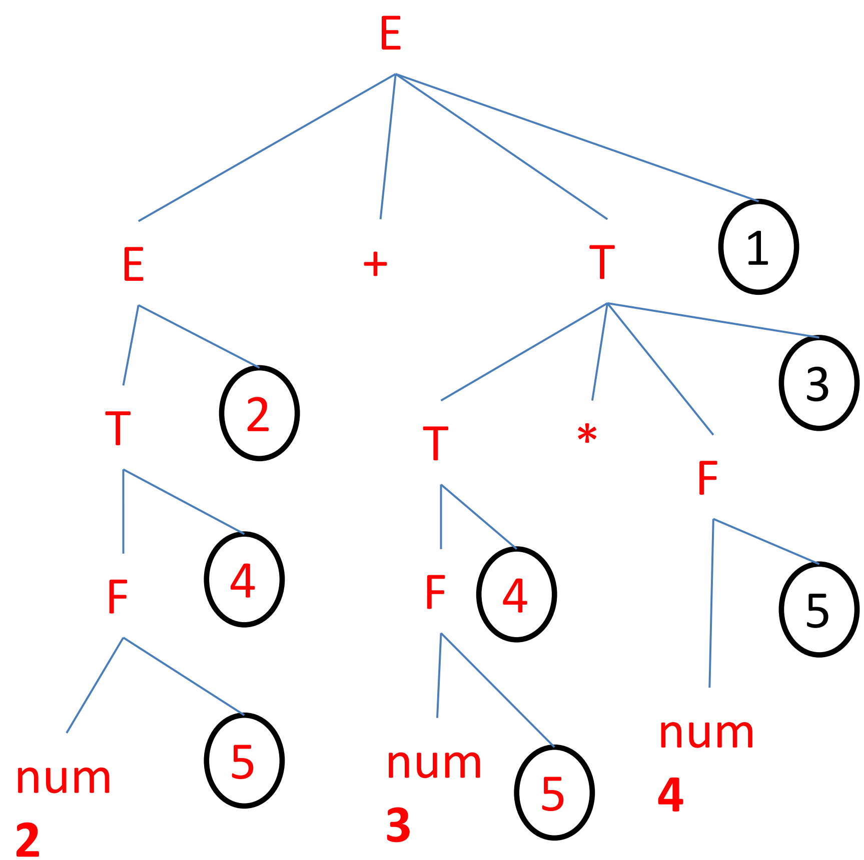
Example 4 (taking top-down approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}	1
E → T	{}	2
T → T * F	{printf("*");}	3
T→F	{}	4
F→ num	{printf(num.lval);}	5

For input, 2 + 3 * 4

Output: 2 3



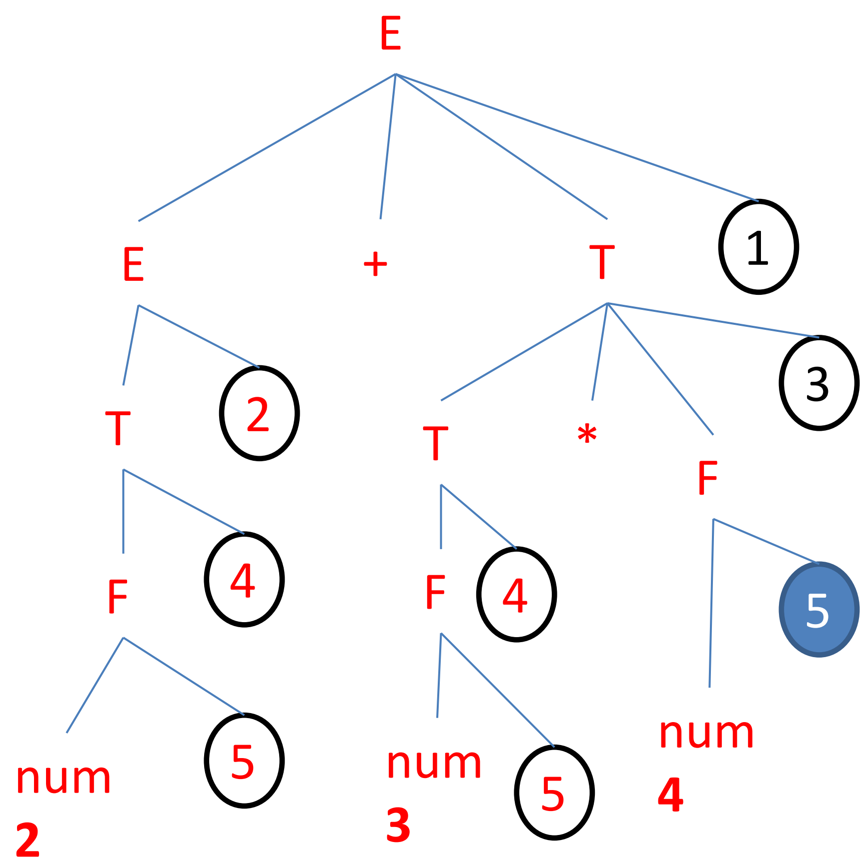
Example 4 (taking top-down approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}	1
E → T	{}	2
T → T * F	{printf("*");}	3
T→F	{}	4
F→ num	{printf(num.lval);}	5

For input, 2 + 3 * 4

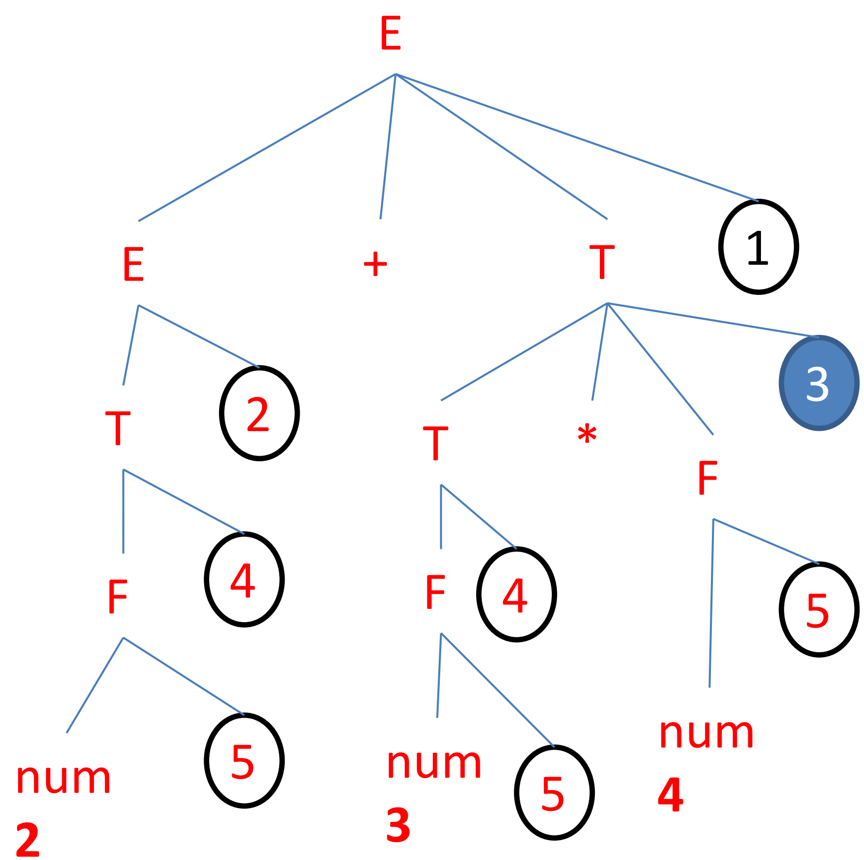
Output: 2 3 4



Example 4 (taking top-down approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}	1
E → T	{}	2
T → T * F	{printf("*");}	3
T→F	{}	4
F→ num	{printf(num.lval);}	5



For input, 2 + 3 * 4

Output: 2 3 4 *

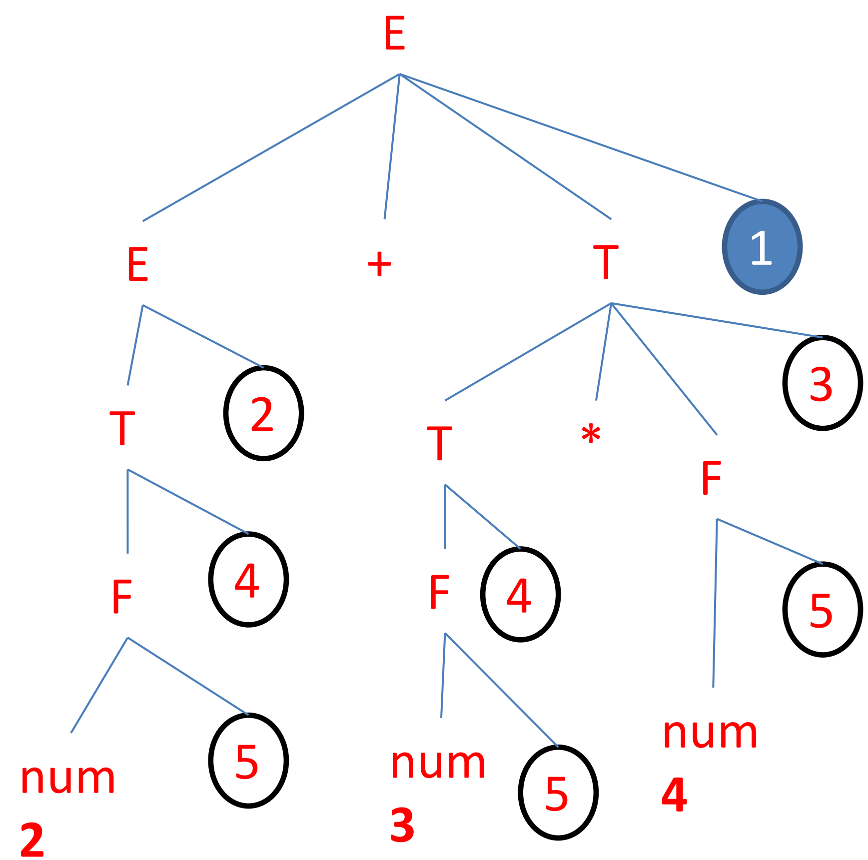
Example 4 (taking top-down approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}	1
E → T	{}	2
T → T * F	{printf("*");}	3
T→F	{}	4
F→ num	{printf(num.lval);}	5

For input, 2 + 3 * 4

Output: 2 3 4 * +



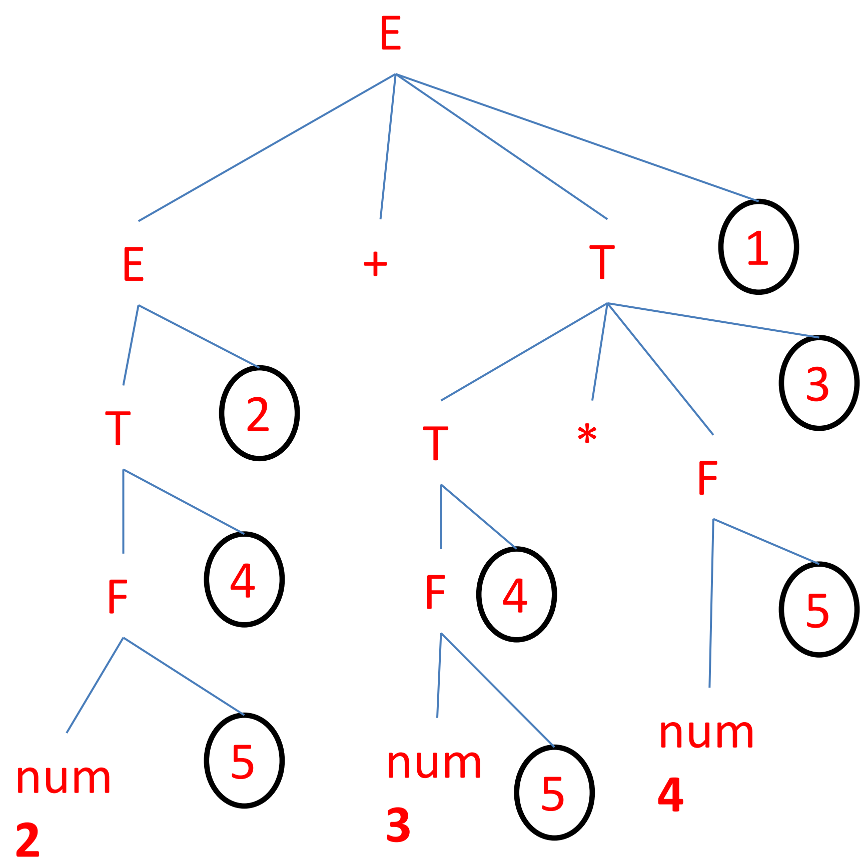
Example 4 (taking top-down approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}	1
E → T	{}	2
T → T * F	{printf("*");}	3
T→F	{}	4
F→ num	{printf(num.lval);}	5

For input, 2 + 3 * 4

Output: 2 3 4 * +

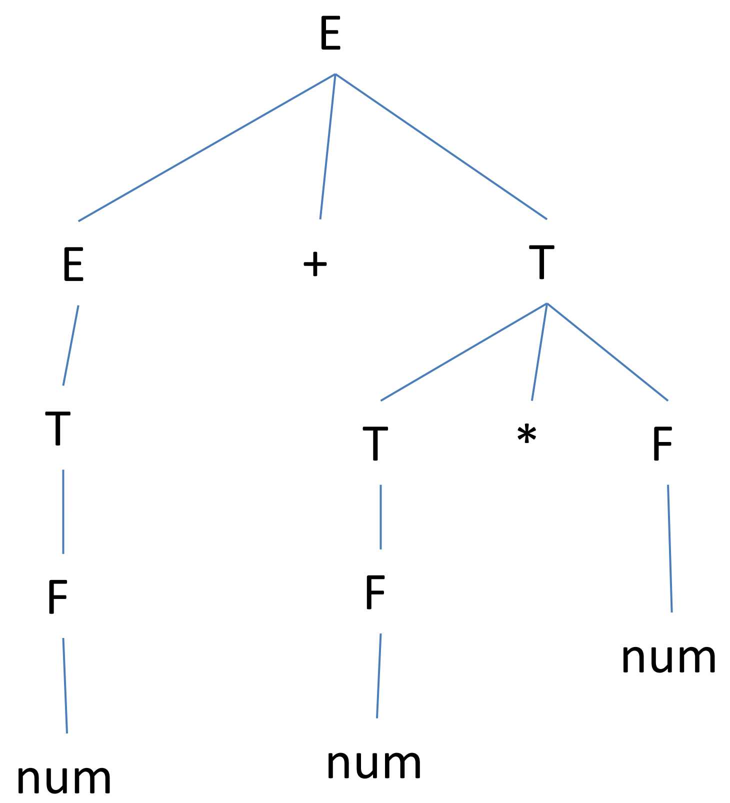


Example 4 (bottom-up approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}
E → T	{}
T → T * F	{printf("*");}
T→F	{}
F→ num	{printf(num.lval);}

For input, 2 + 3 * 4

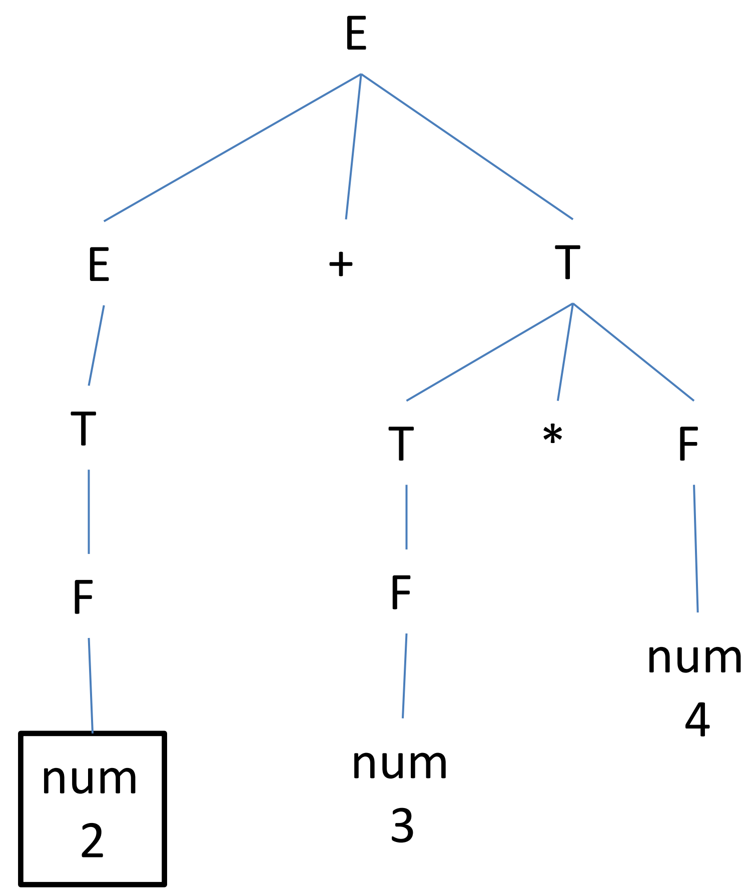


Example 4 (bottom-up approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}
E → T	{}
T → T * F	{printf("*");}
T→F	{}
F→ num	{printf(num.lval);}

For input, 2 + 3 * 4

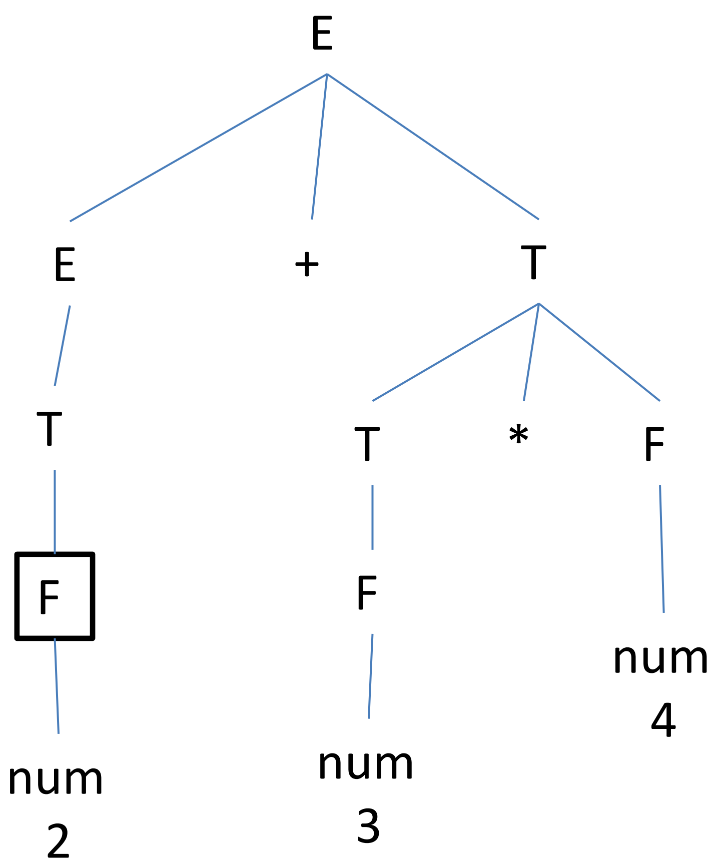


Example 4 (bottom-up approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}
E → T	{}
T → T * F	{printf("*");}
T→F	{}
F→ num	{printf(num.lval);}

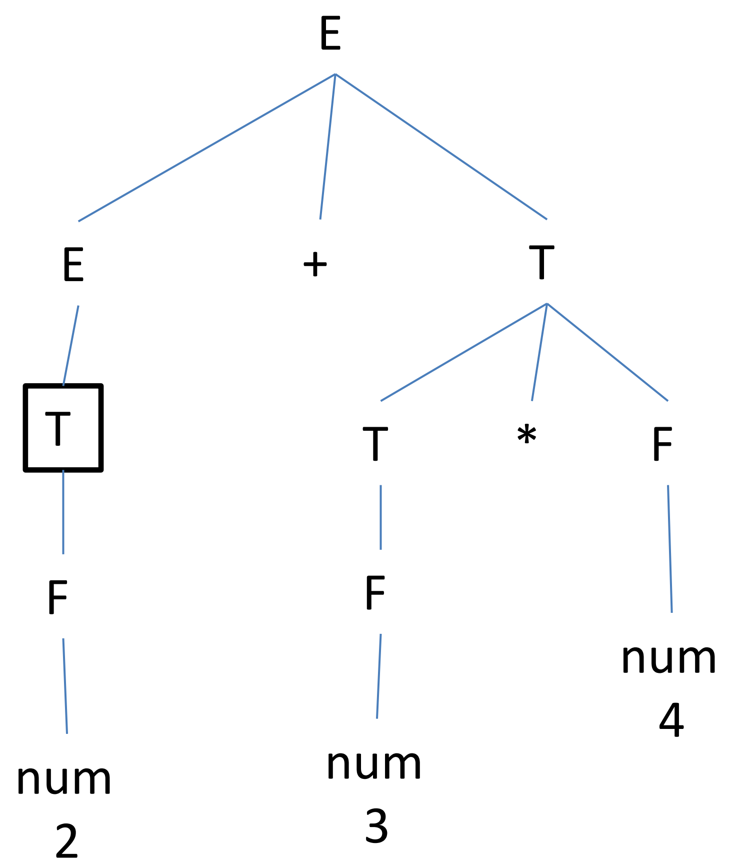
For input, 2 + 3 * 4
Output: 2



Example 4 (bottom-up approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}
E → T	{}
T → T * F	{printf("*");}
T→F	{}
F→ num	{printf(num.lval);}



For input, 2 + 3 * 4

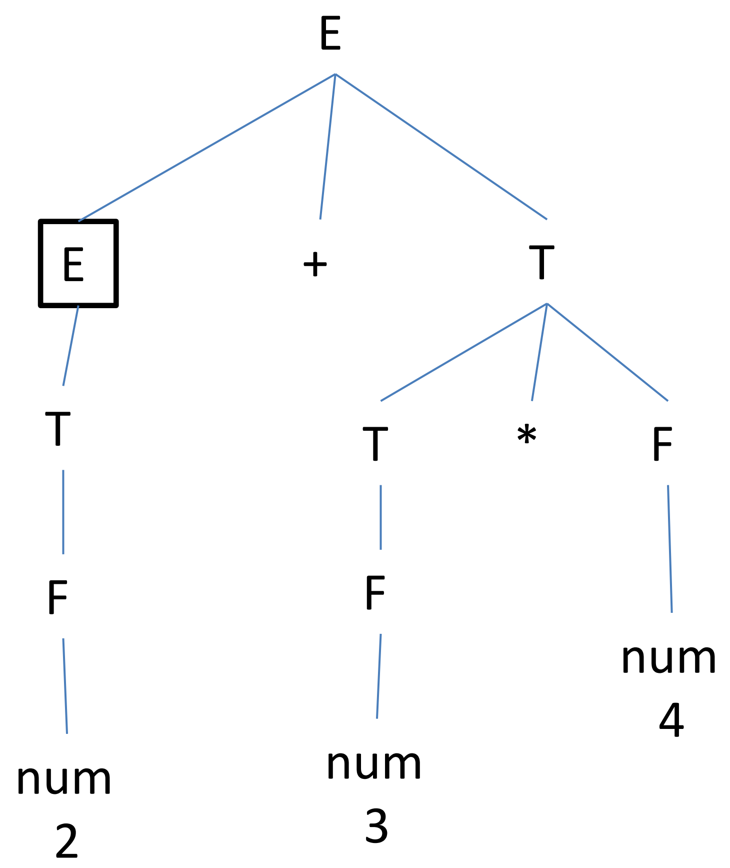
Output: 2

Example 4 (bottom-up approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}
E → T	{}
T → T * F	{printf("*");}
T→F	{}
F→ num	{printf(num.lval);}

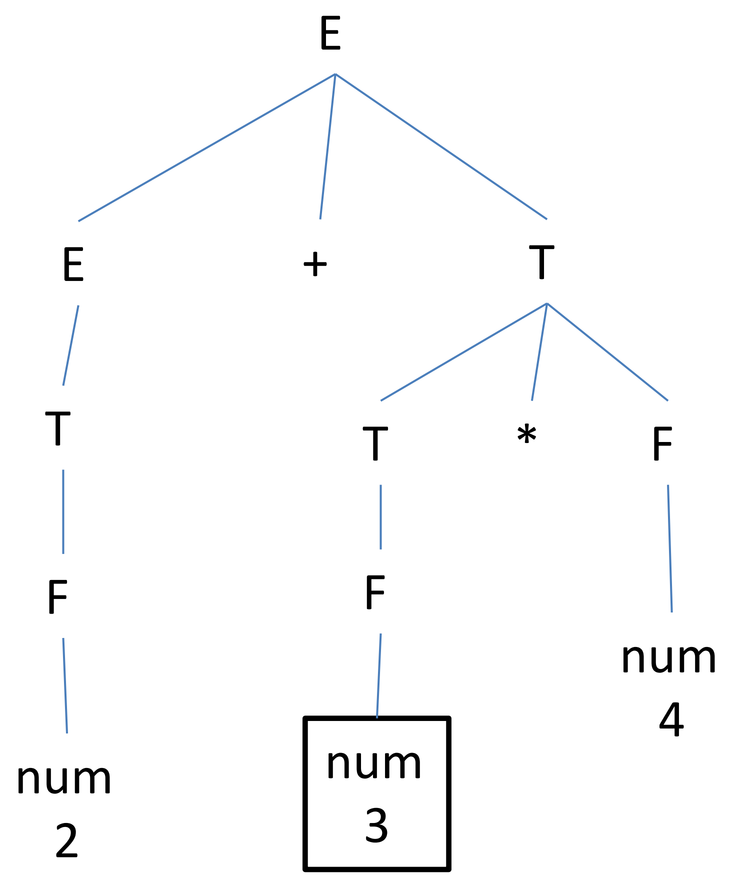
For input, 2 + 3 * 4
Output: 2



Example 4 (bottom-up approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}
E → T	{}
T → T * F	{printf("*");}
T→F	{}
F→ num	{printf(num.lval);}



For input, 2 + 3 * 4

Output: 2

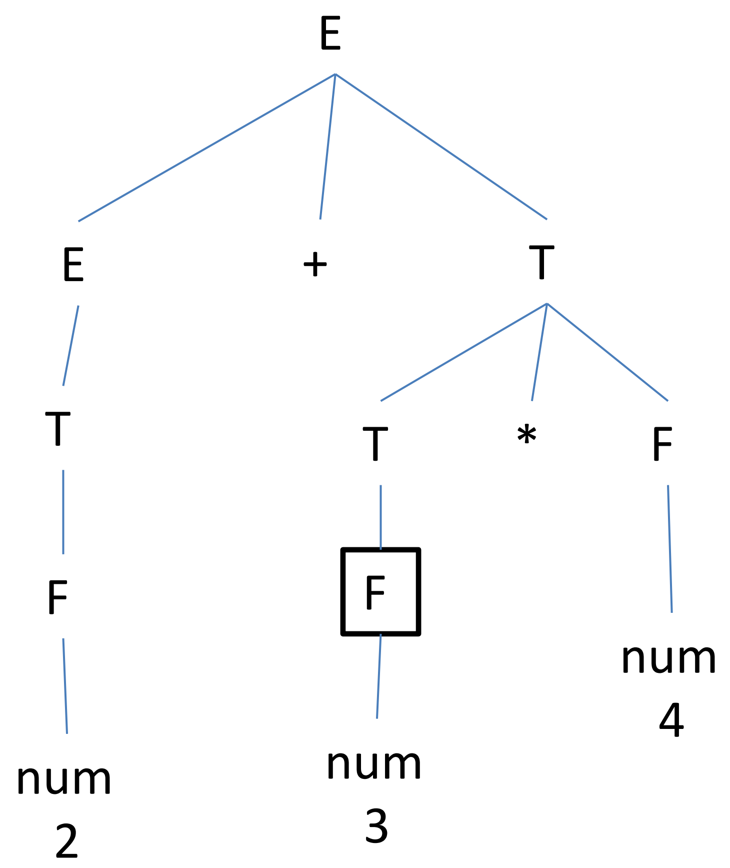
Example 4 (bottom-up approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}
E → T	{}
T → T * F	{printf("*");}
T→F	{}
F→ num	{printf(num.lval);}

For input, 2 + 3 * 4

Output: 2 3



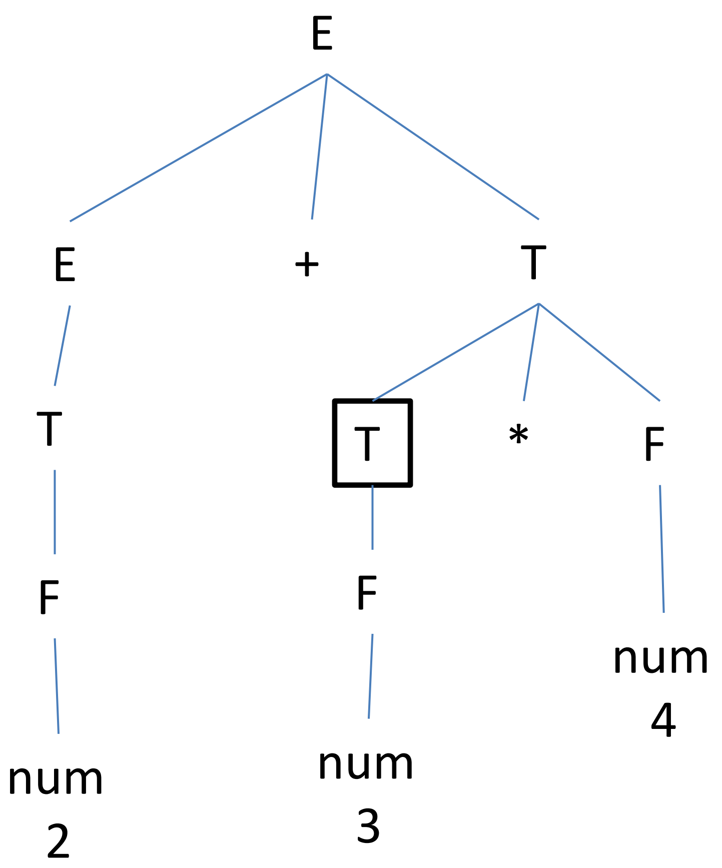
Example 4 (bottom-up approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}
E → T	{}
T → T * F	{printf("*");}
T→F	{}
F→ num	{printf(num.lval);}

For input, 2 + 3 * 4

Output: 2 3



Example 4 (bottom-up approach)

- SDT to convert infix to postfix

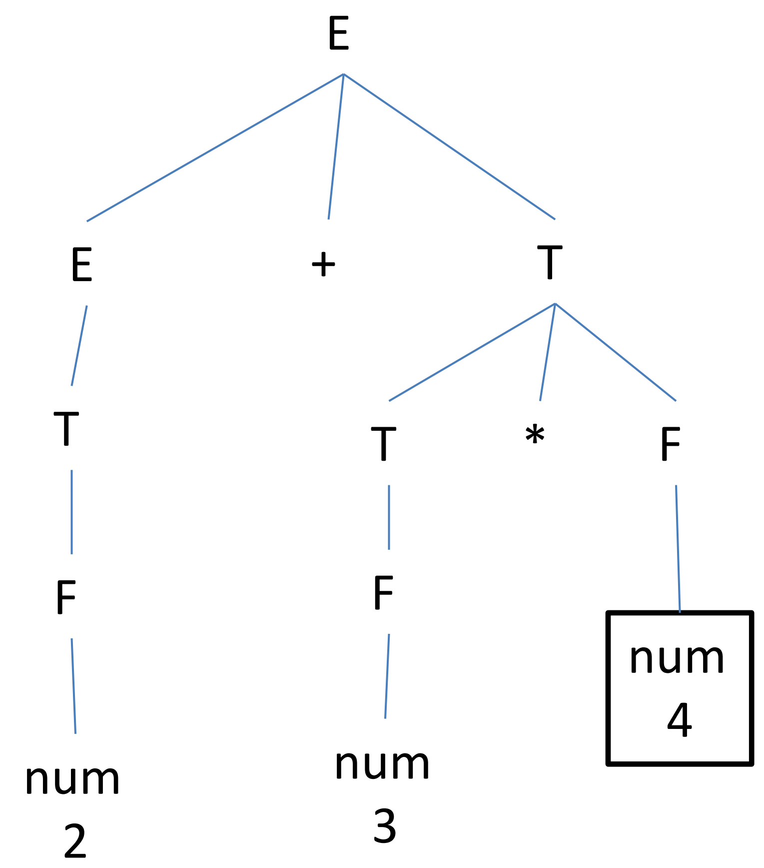
E→ E + T　　　{printf("+");}

E → T　　　{}

T → T * F　　　{printf("*");}

T→F　　　{}

F→ num　　　{printf(num.lval);}



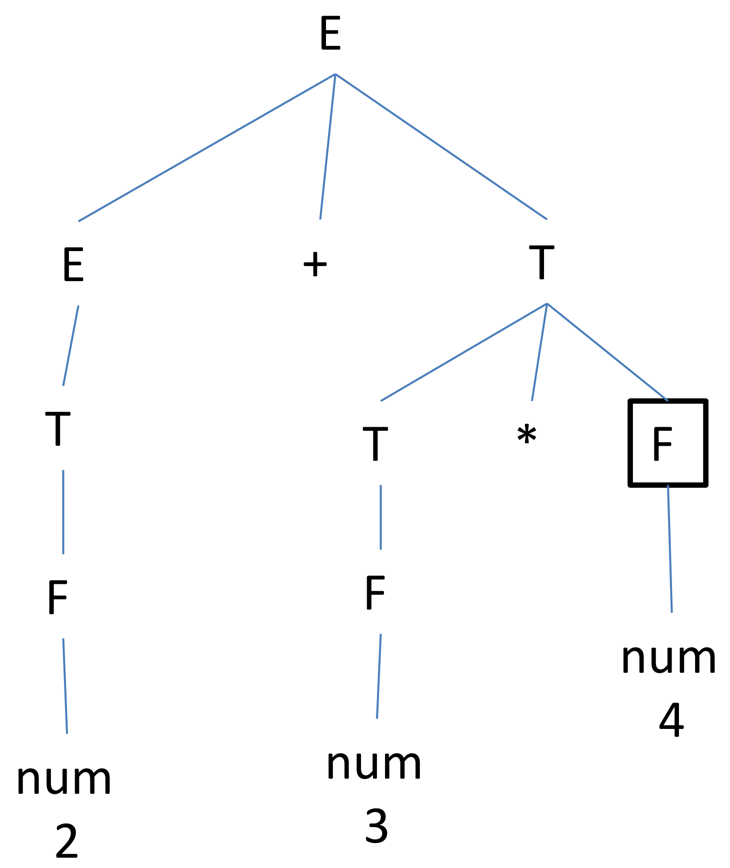
For input, 2 + 3 * 4

Output: 2 3

Example 4 (bottom-up approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}
E → T	{}
T → T * F	{printf("*");}
T→F	{}
F→ num	{printf(num.lval);}

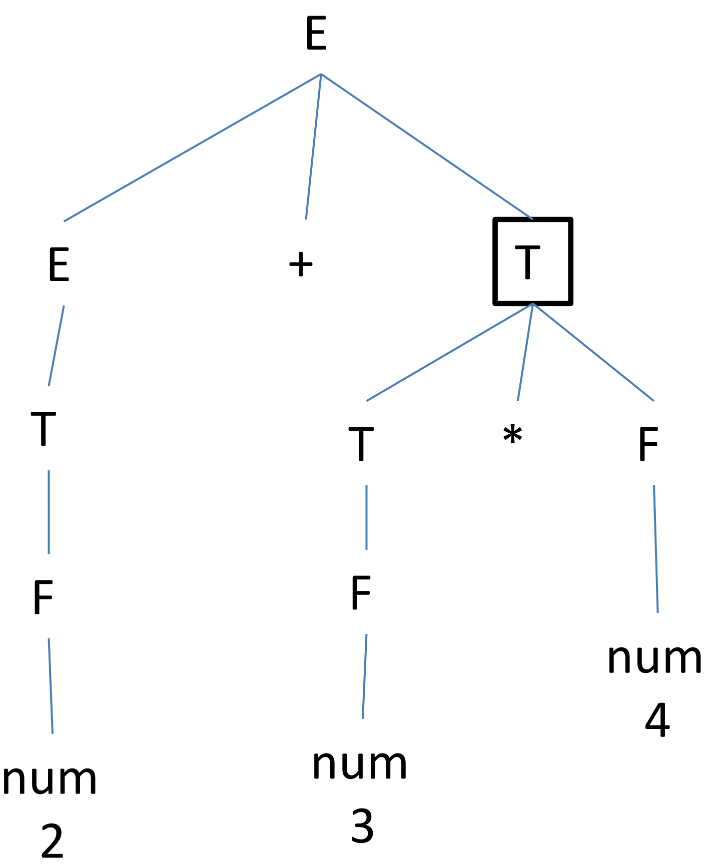


For input, 2 + 3 * 4
Output: 2 3 4

Example 4 (bottom-up approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}
E → T	{}
T → T * F	{printf("*");}
T→F	{}
F→ num	{printf(num.lval);}

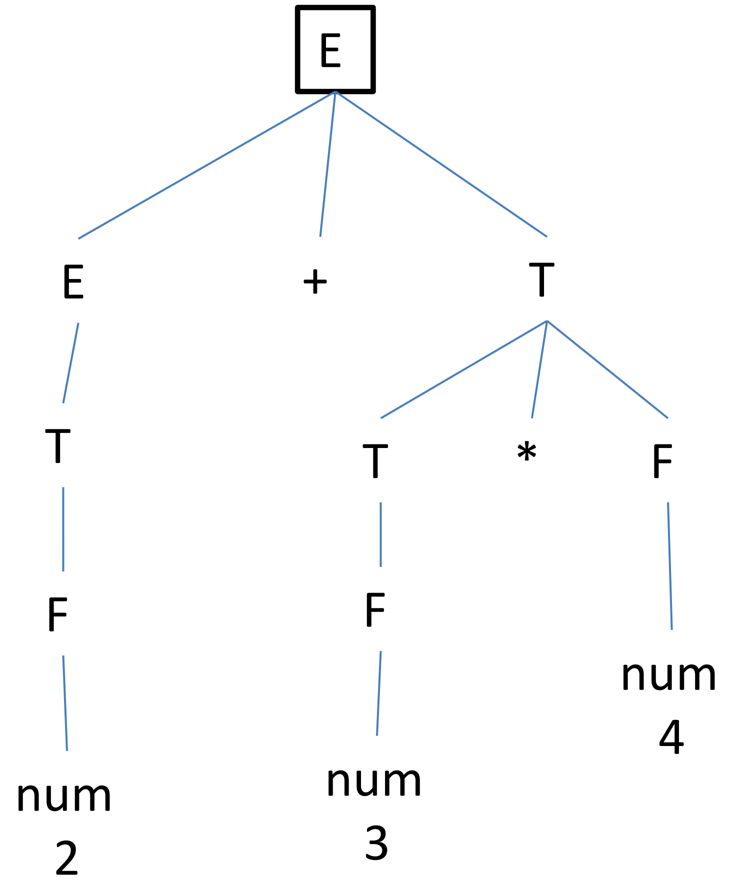


For input, 2 + 3 * 4
Output: 2 3 4 *

Example 4 (bottom-up approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}
E → T	{}
T → T * F	{printf("*");}
T→F	{}
F→ num	{printf(num.lval);}



For input, 2 + 3 * 4

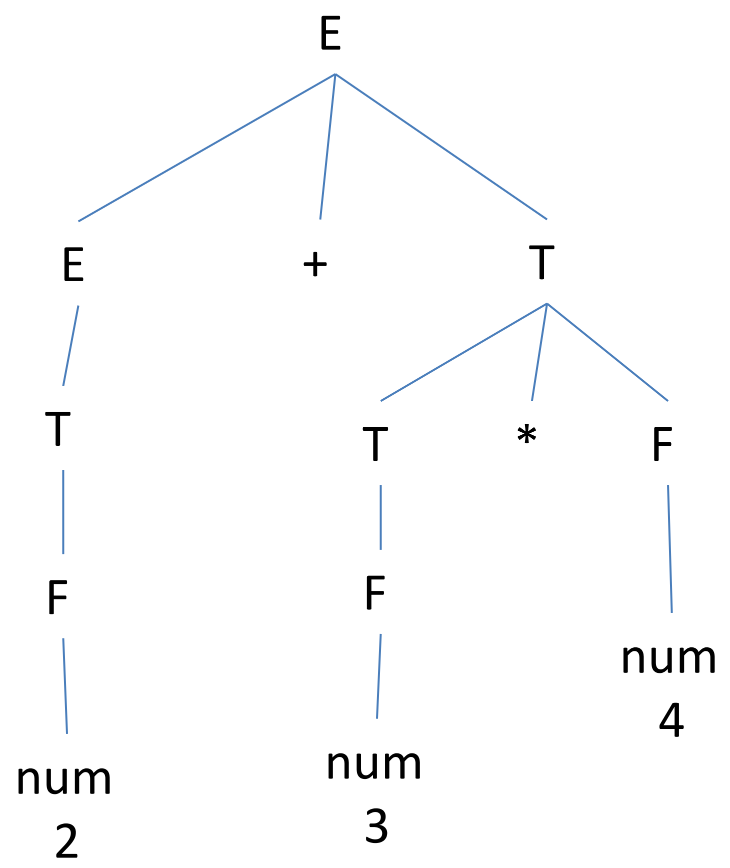
Output: 2 3 4 * +

Example 4 (bottom-up approach)

- SDT to convert infix to postfix

E→ E + T	{printf("+");}
E → T	{}
T → T * F	{printf("*");}
T→F	{}
F→ num	{printf(num.lval);}

For input, 2 + 3 * 4
Output: 2 3 4 * +



Example 5

- SDT to build a syntax tree

For input, 2 + 3 * 4

Example 5

- SDT to build a syntax tree

For input, 2 + 3 * 4