Attributed Translation Grammar

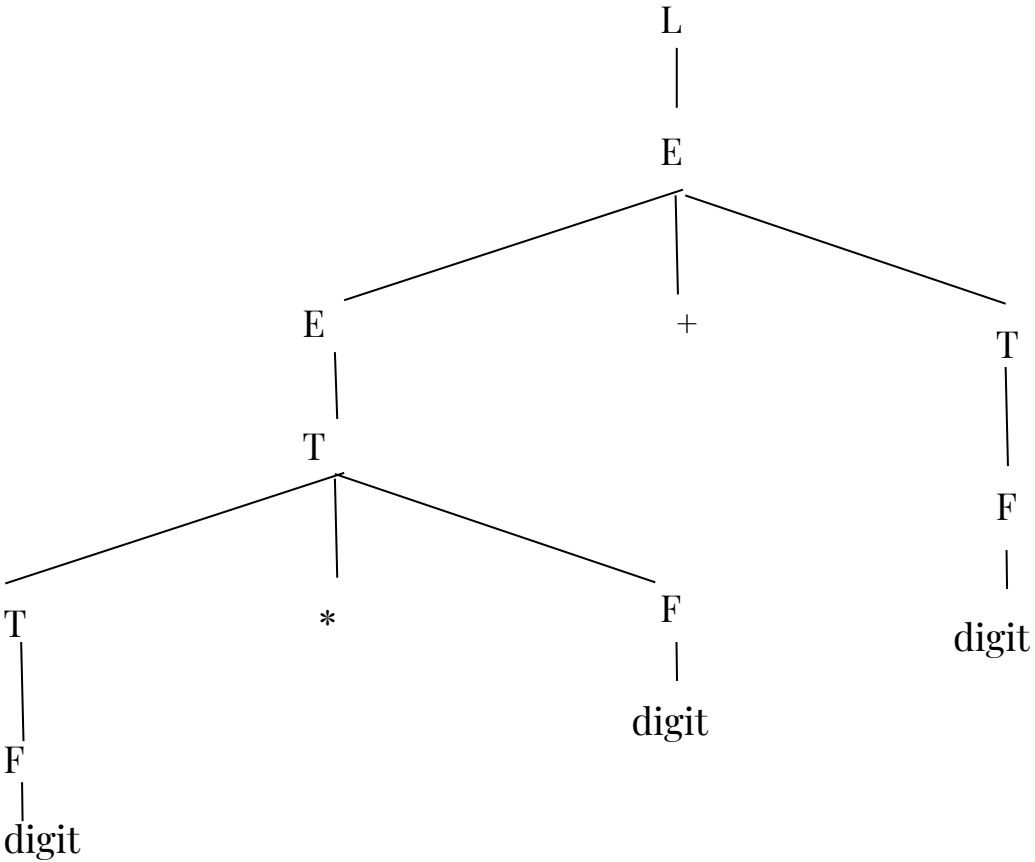
Sudakshina Dutta

Consider the following grammar

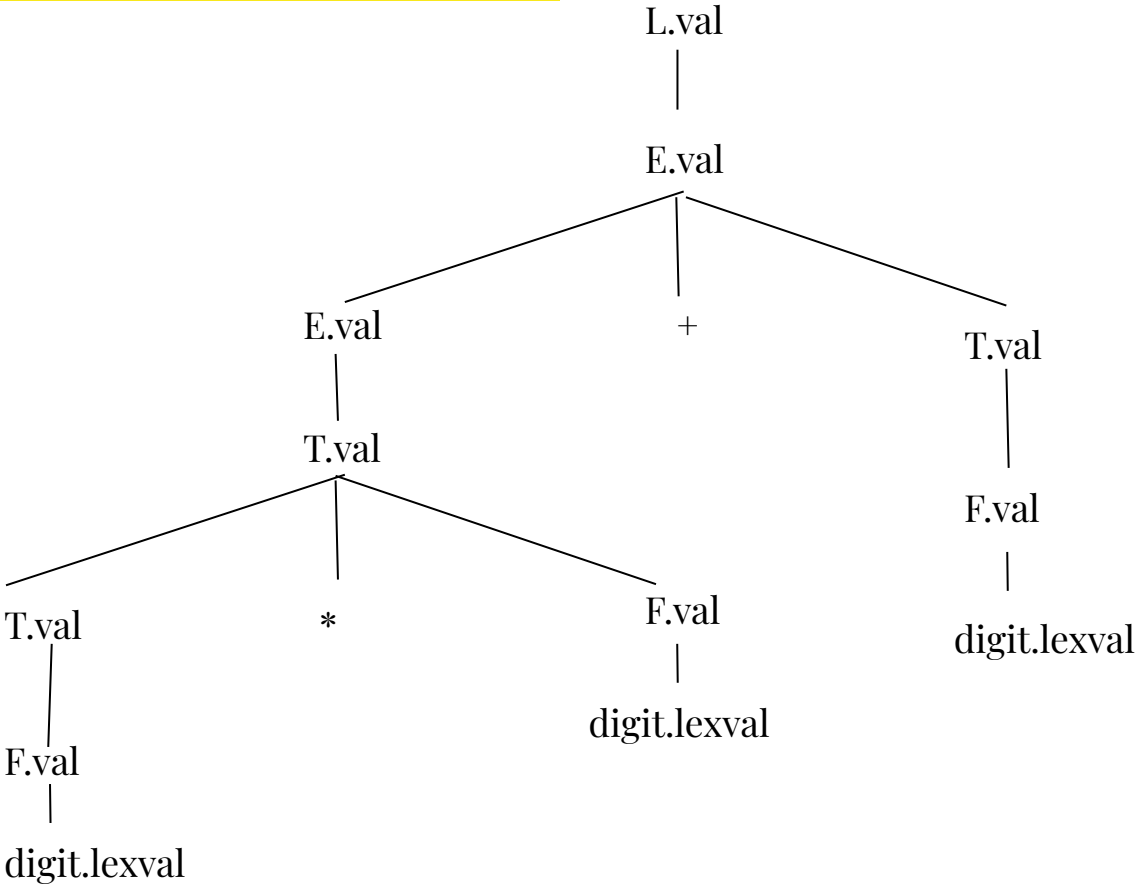
- L -> E n
- E -> E + T
- E -> T
- T -> T * F
- T -> F
- F -> (E)
- F -> digit

Production	Semantic Rules
L -> E n	L.val = E.val
E -> E_1 + T	E.val = E_1.val + T.val
E -> T	E.val = T.val
T -> T_1 * F	T.val = T_1.val * F.val
T -> F	T.val = F.val
F -> (E)	F.val = E.val
F -> digit	F.val = digit.lexval

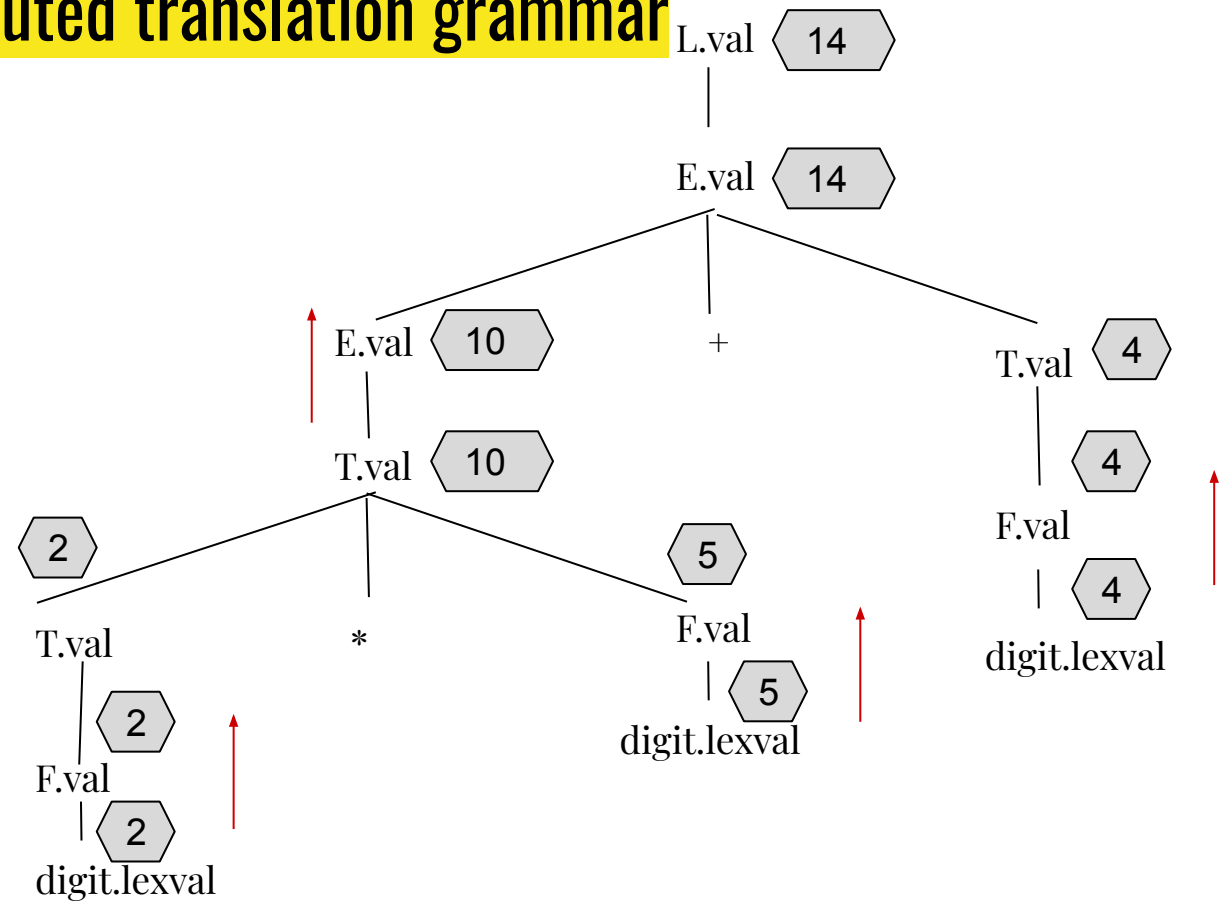
Attributed translation grammar



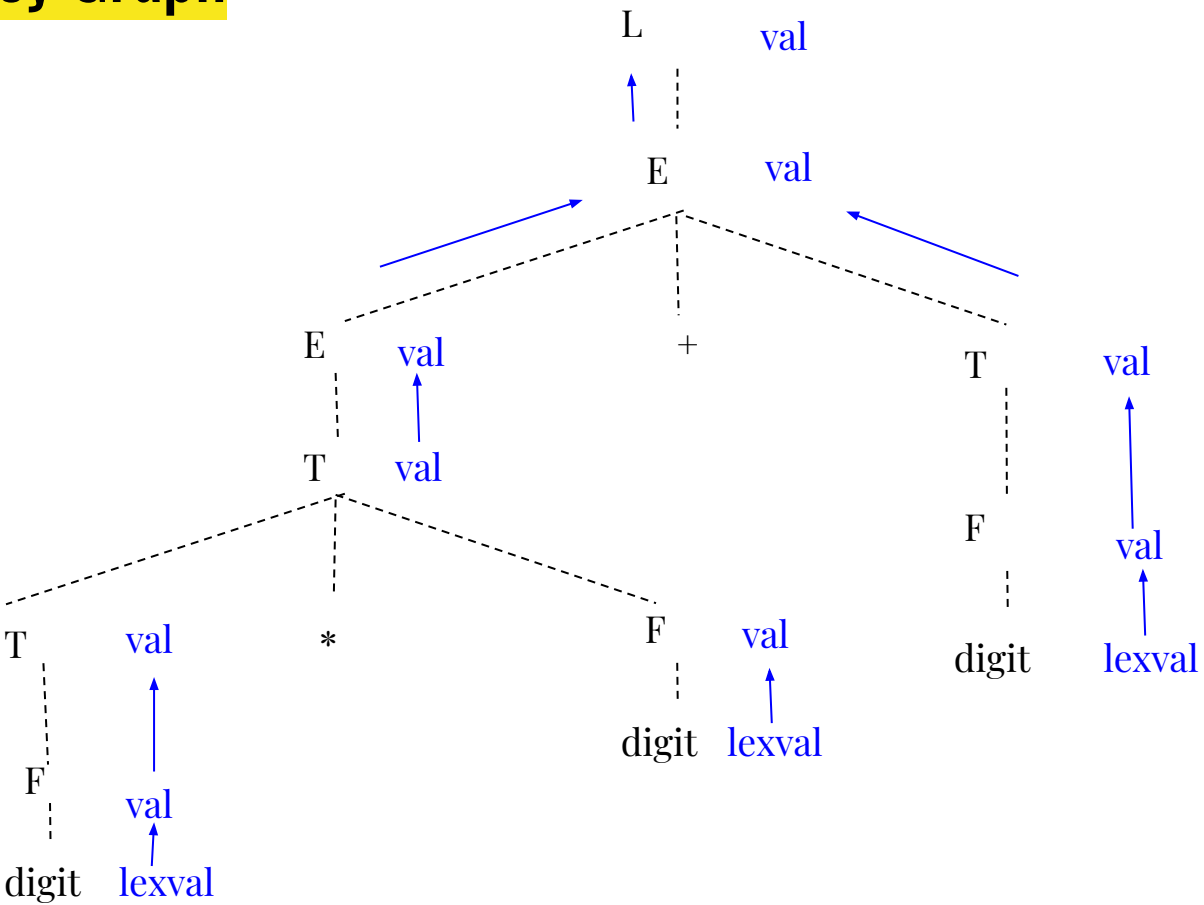
Attributed translation grammar



Attributed translation grammar



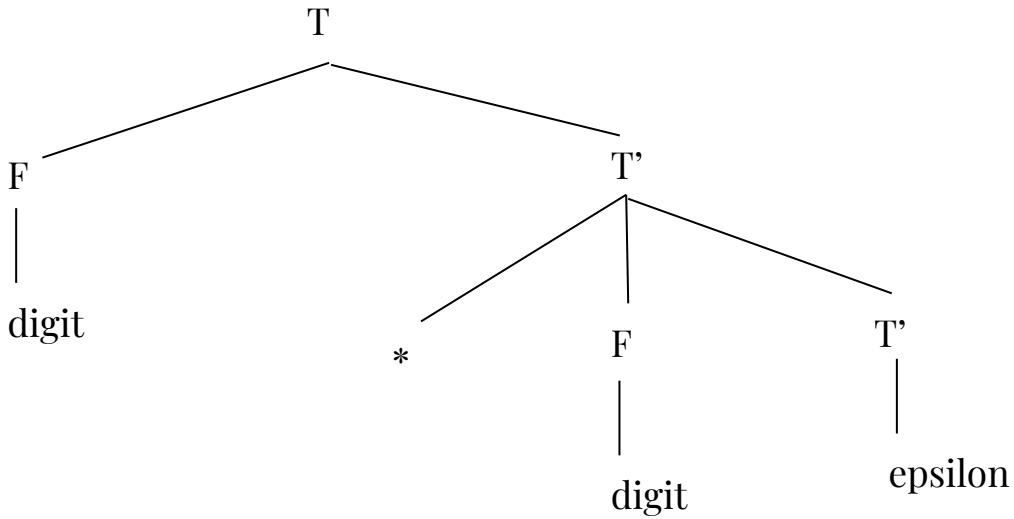
Dependency Graph



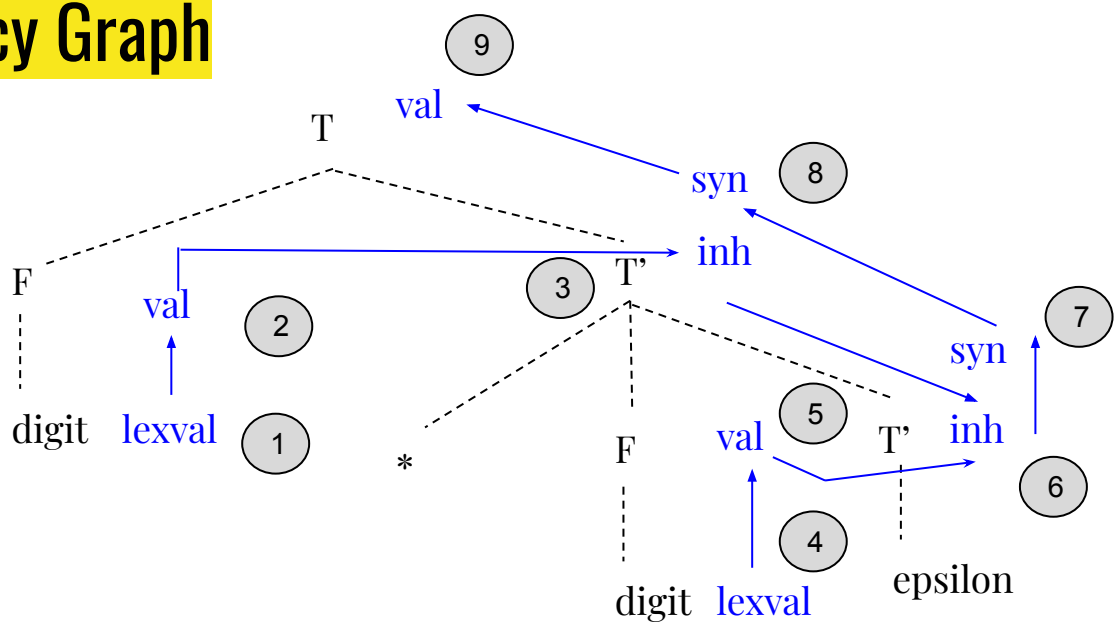
Consider the following grammar

- T -> FT’
- T’ -> *FT’
- T’ -> epsilon
- F -> digit

Productions	Semantic Rules
T -> FT’	T’.inh = F.val
	T.val = T’.syn
T’ -> *FT_1’	T_1’.inh = T’.inh * F.val
	T’.syn = T_1’.syn
T’ -> epsilon	T’.syn = T’.inh
F -> digit	F.val = digit.lexval

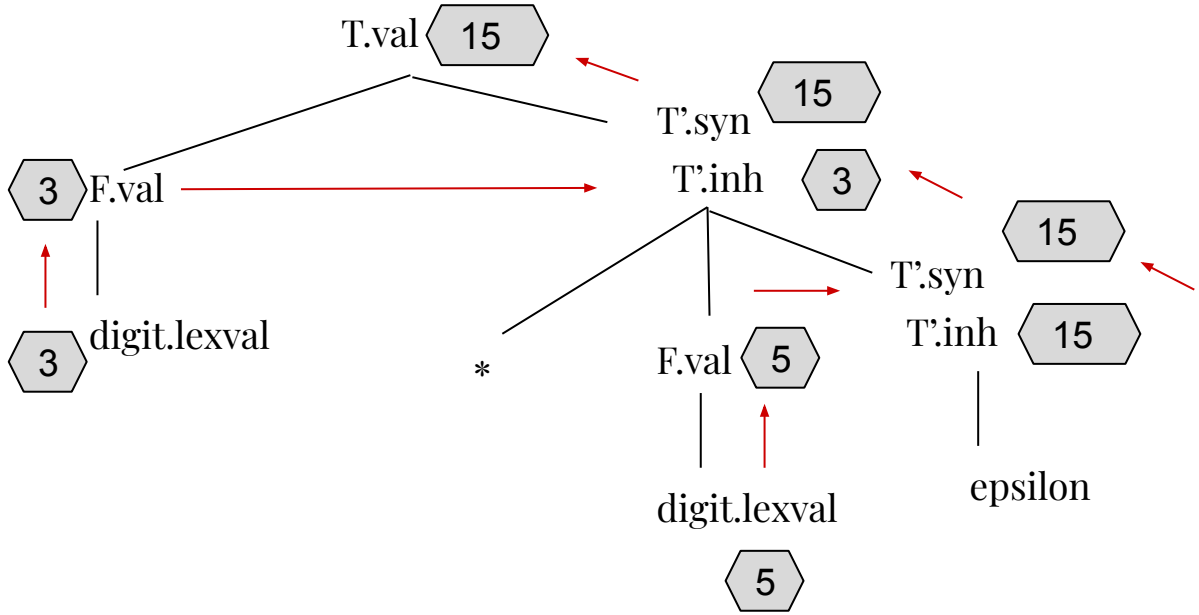

T
F
T'
digit
*
F
T'
digit
epsilon

Dependency Graph



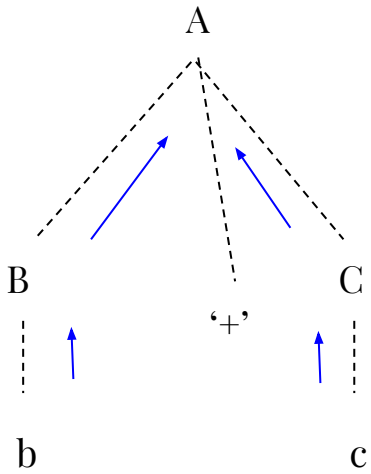
Productions	Semantic Rules
T -> FT'	T'.inh = F.val T.val = T'.syn
T' -> *FT$_1$'	T$_1$'.inh = T'.inh * F.val T'.syn = T$_1$'.syn
T' -> epsilon	T'.syn = T'.inh
F -> digit	F.val = digit.lexval

Topological sort is followed to derive the ordering



Productions	Semantic Rules
T -> FT'	T'.inh = F.val T.val = T'.syn
T' -> *FT$_1$'	T$_1$'.inh = T'.inh * F.val T'.syn = T$_1$'.syn
T' -> epsilon	T'.syn = T'.inh
F -> digit	F.val = digit.lexval

Synthesized Attributes



A ↠ B '+' C { A.val = B.val + C.val; print("a")}

B ↠ b { B.val = b.lexval; print("b")}

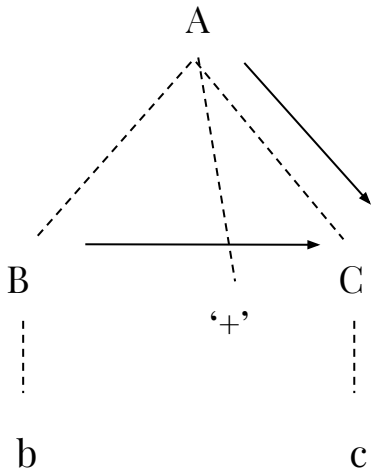
C ↠ c { C.val = c.lexval; print("c")}

Printed string : "b" "c" "a"

Post-order traversal

Semantic rules are placed at the end

Inherited Attributes

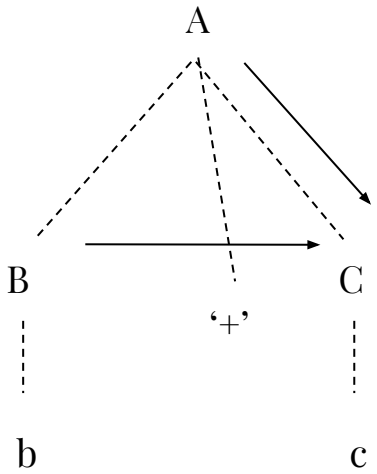


A ↣ B '+' C

B ↣ b

C ↣ c

Inherited Attributes



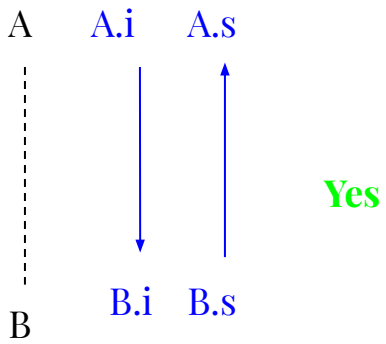
$A \twoheadrightarrow B$ '+' $\{C.i = f(Att_{A1}, ...Att_{Am}, Att_{B1}, ..Att_{Bn})\}$ C

$B \twoheadrightarrow b$

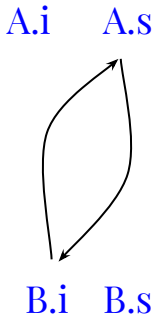
$C \twoheadrightarrow c$

Semantic rules are placed anywhere

Circularity in the dependency graph



Yes

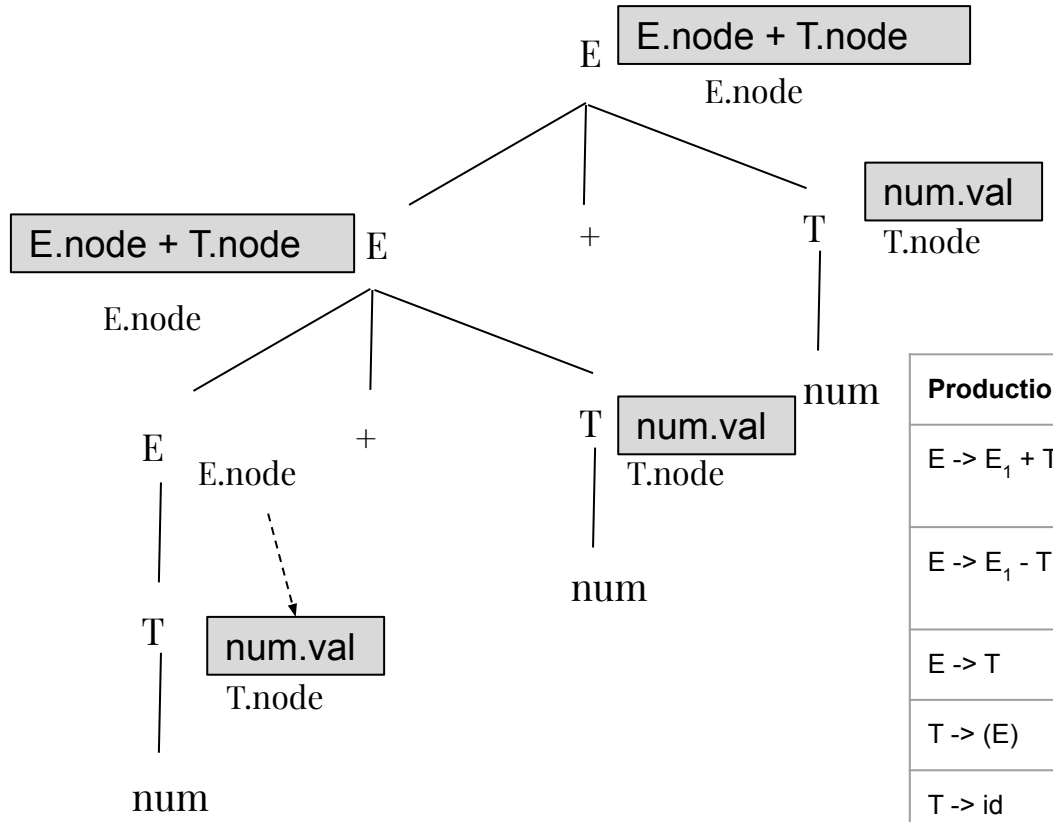


No

Consider the following grammar

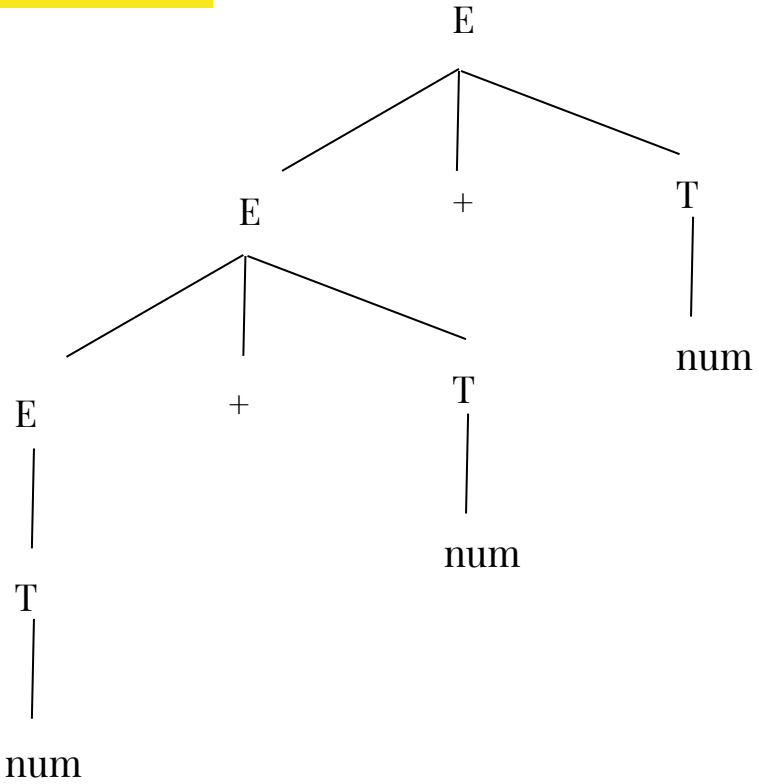
- E -> E + T
- E -> E - T
- E -> T
- T -> (E)
- T -> id
- T -> num

Production	**Semantic rules**
E -> E_1 + T	E.node = new Node('+', E_1.node, T.node)
E -> E_1 - T	E.node = new Node('-', E_1.node, T.node)
E -> T	E.node = T.node
T -> (E)	T.node = E.node
T -> id	T.node = new Leaf(id, id.entry)
T -> num	T.node = new Leaf(num, num.val)

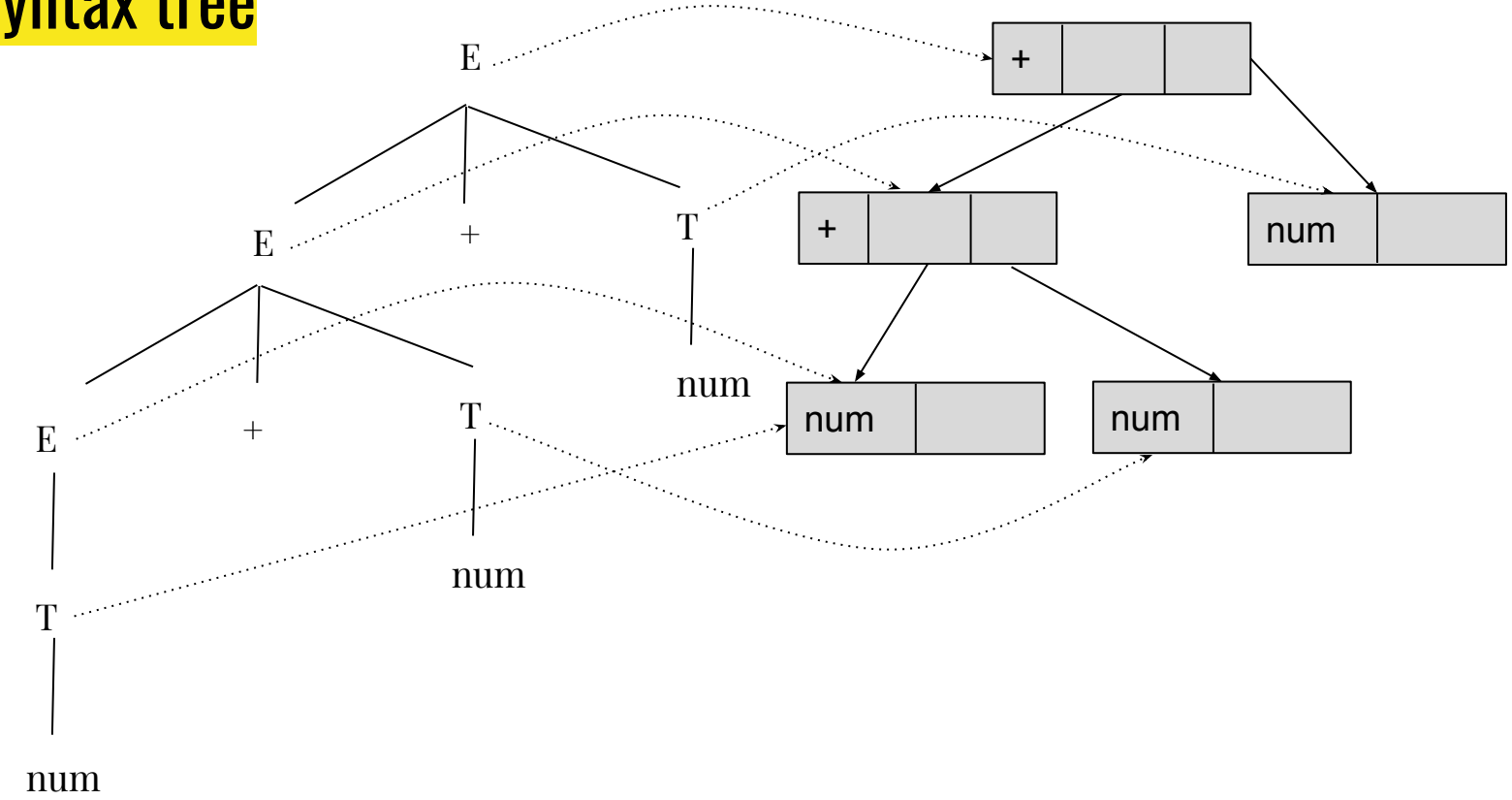


Production	Semantic rules
E -> E_1 + T	E.node = new Node('+', E_1.node, T.node)
E -> E_1 - T	E.node = new Node('-', E_1.node, T.node)
E -> T	E.node = T.node
T -> (E)	T.node = E.node
T -> id	T.node = new Leaf(id, id.entry)
T -> num	T.node = new Leaf(num, num.val)

Parse tree



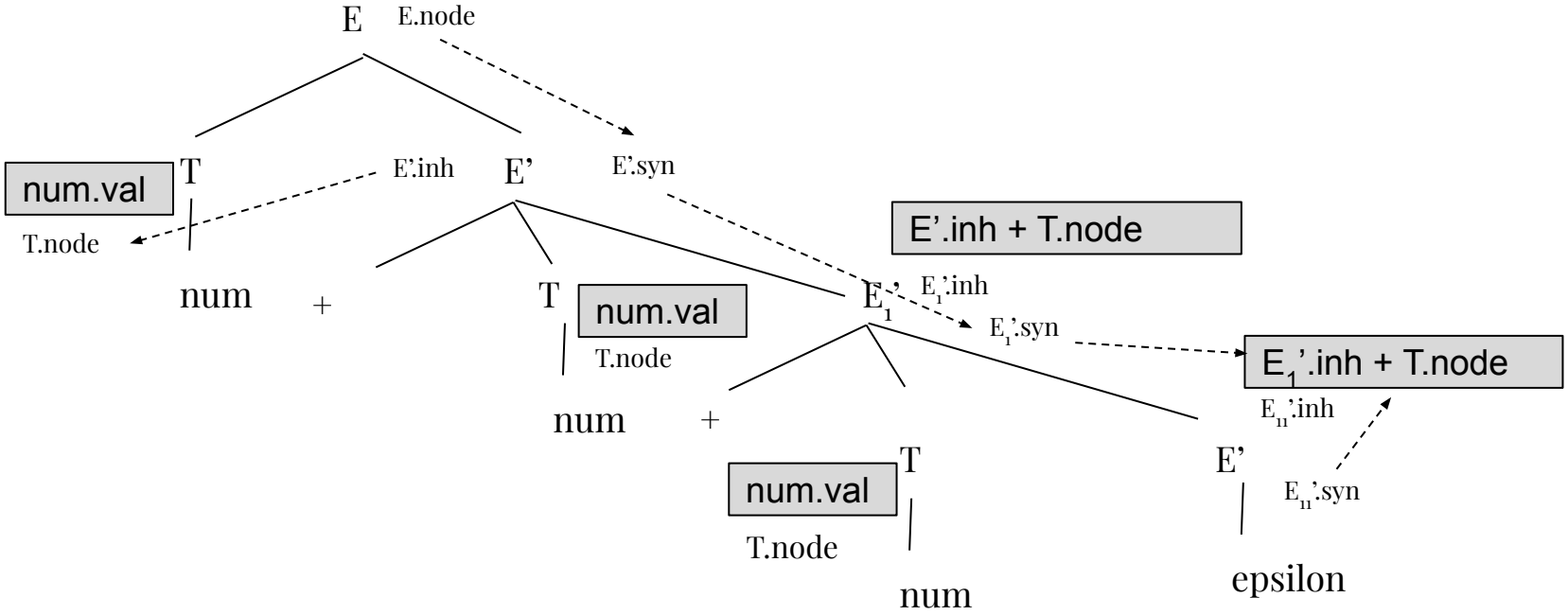
Syntax tree



Consider the following grammar

- E -> TE’
- E’ -> $+TE_1$’
- E’ -> $-TE_1$’
- E’ -> epsilon
- T -> (E)
- T -> id
- T -> num

Productions	**Semantic rules**
E -> TE’	E.node = E’.syn
	E’.inh = T.node
E’ -> +TE_1’	E_1’.inh = new Node(‘+’, E’.inh, T.node)
	E’.syn = E_1’.syn
E’ -> -TE_1’	E_1’.inh = new Node(‘-’, E’.inh, T.node)
	E’.syn = E_1’.syn
E’ -> epsilon	E’.syn = E’.inh
T -> (E)	T.node = E.node
T -> id	T.node = new Leaf(id, id.entry)
T -> num	T.node = new Leaf(num, num.val)


E
E.node
num.val
T
E'.inh
E'
E'.syn
T.node
num
+
T
num.val
T.node
E'.inh + T.node
E1'
E1'.inh
E1'.syn
num
+
T
num.val
T.node
num
E1'.inh + T.node
E1n'.inh
E'
E1n'.syn
epsilon