Assignment - 3
Compiler Design

Name – Shagun Verma
Section – E1
Roll no – 64

Ques 1. Define a Syntax directed definition (SDD). Differentiate between S – attributed and L – attributed definations with suitable examples.

Ans An SDD is a context free grammar in which each grammar symbol is associated with attributes, and each production has rules that define how attribute values are computed. These rules specify how attributes are calculated from other attribute values.

Difference between S – attributed and L – attributed definations:

- S – attributed : use only synthesized attributes. Values are computed from the attributes of the children in the parse tree.
  ex:- Evaluating Airthmetic Expressions
- L – attributed : use both synthesized and inherited attributes. Values are computed using the attributes of the children / sibling.
  ex:- Type checking where inheritted attributes are passed down.

Ques 2. Discuss how bottom – up evaluation for S – attributed definations is performed. why is it not suitable for L – attributed definations?

Ans Bottom – up parsing computes synthesized attributes from the leaves to the root. Since S – attributed definations have only synthesized attributes, they suits bottom – up evaluation. Not suitable for L – attributed definations requires inherited attributes, which need to be passed down or across sibling, making bottom – up evaluation impractical.

Ques 3. Described L-attributed definations. Provide an example where Synthesized and inherited attributes are both used.

Ans 3. An L-attributed is a special kind of Syntax-directed definations where each inherited attribute can be evaluated from left to right using the inherited attributes of parent nodes or Synthesized attributed of left siblings.

ex:- In an expression grammar, Synthesized attributed calculate the expression value, while inherited attributes calculate the expression value, while inherited attributes manage variable types and scope.

Ques 4. What are the challenge involved in top-down translation of Syntax-directed definations? Explain with an example.

1. Circular Dependencies: Attributes may depend on each other, causing evaluation conflicts.
2. Inherited Attribute management: Passing attributes down the parse tree is complex.
3. Complexity in non-LL Grammars: Difficult for left-recursive or ambigous grammars.

ex:- In expression evaluation, managing inherited attributes for operator precedence can be problematic.

Ques 5. How does the direction of attribute evaluation affect the implementation of translation schemes in Compilers?

Ans 5. 1. Top-Down (L-attributed): Efficient for inherited attributes but struggles with the Synthesized ones.

2. Bottom-up (S-attributed): Efficient for Synthesized attributed but not for inherited one.

3. Hybrid approach: Combining both for Complex Cases.

4. Efficiency Consideration: Direct influence on parsing strategy (LL for top-down, LR for bottom-up)

5. Implementation Complexity: Direction affects how easily attributes are propagated.

Ques 6. 7. Consider grammar with the following translation rules and € as Start Symbol.

E → E#T { E.val = E.val * T.val }

E → T { E.val = T.val }

T → T&F { T.value = T.val + F.val }

T → F { T.val = F.val }

F → num { F.val = num.val }

Compute E.value for the root of the parse tree for the expression: 2# 3& 5# 6& 4.

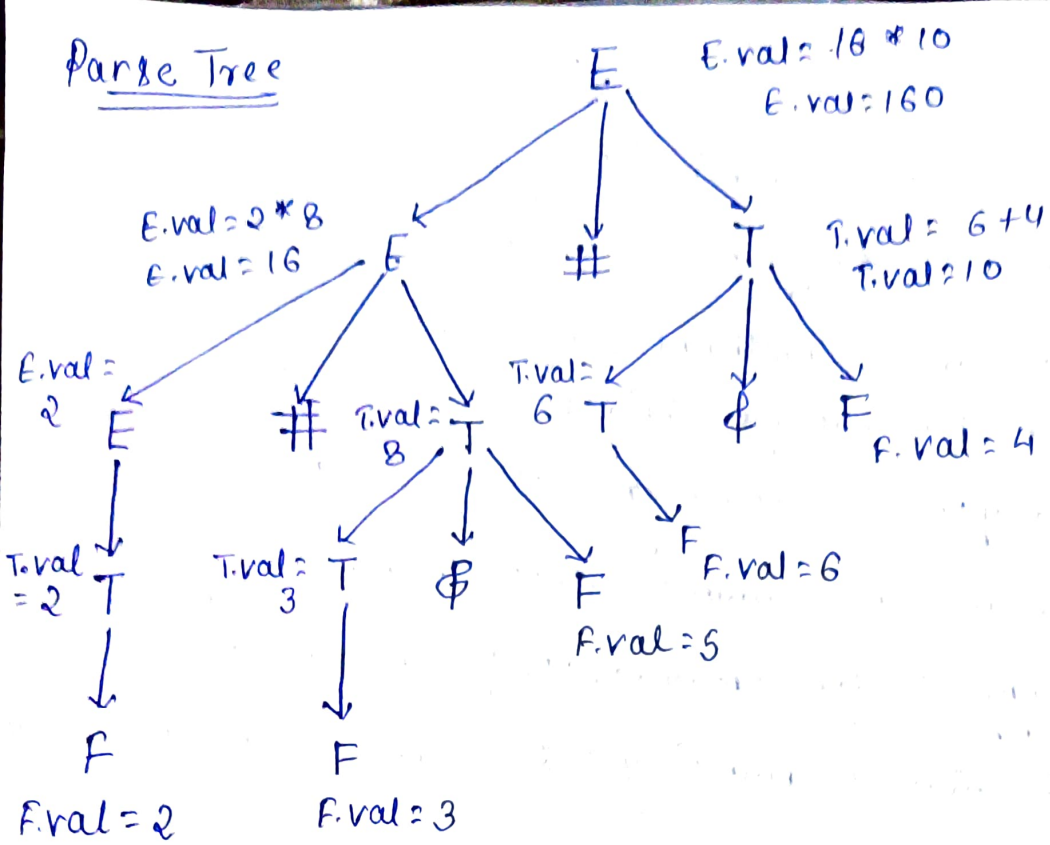
a) 200

b) 180

c) 160

d) 40

Parse Tree



So, the value of the expression is 160

Ques 6. Consider the translation scheme shown below:

$S \rightarrow TR$

$R \rightarrow +T \{ Print('+'); \} R \mid \epsilon$

$T \rightarrow num \{ print(num.val); \}$

Here num is a token and num.val is an integer value for input string 9+5+2, the translation will print.

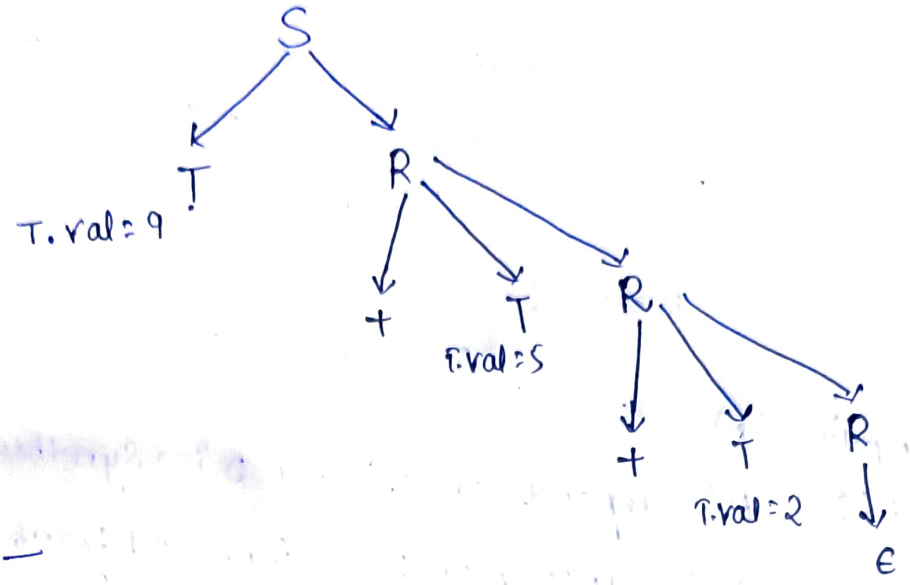
a) 9+5+2

b) 95+2+

c) 95 2++

d) ++9 5 2

Parse Tree



output :—

9 + 5 + 2

Ques 8. Consider the following translation scheme

S → ER

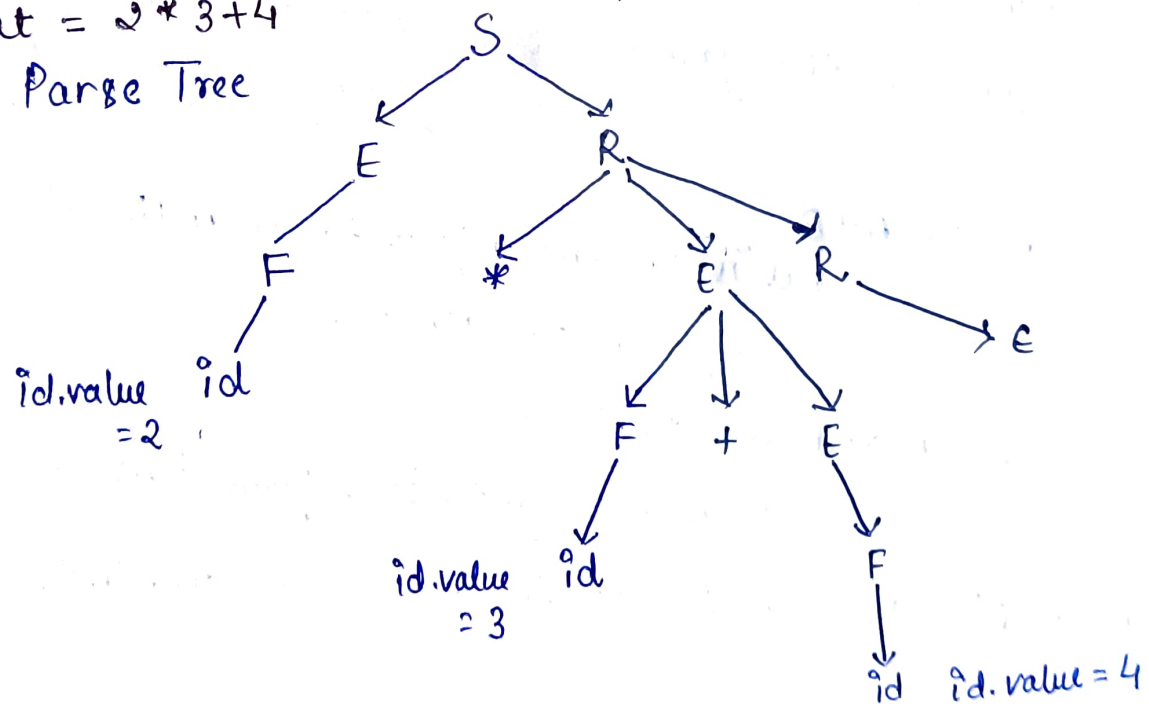
R → * E {print(*);} R | ϵ

E → F + E {print(+);} | F

F → (S) | id {print(id.value);}

Input = 2 * 3 + 4

Ans Parse Tree



Output :- 2 * + 3 4

Ques 9. Let the 2 production be $P \to CD$ and $P \to AB$

Consider rules: Rule 1: $P.i = C.i + D.i$ and

$C.i = D.i - 1$

Rule 2: $A.i = P.i * 2$ and

$B.i = A.i + P.i$

Ans 9. option (b)

Rule 1: P.i depends on C.i and D.i { Synthesized from child }

C.i depends on D.i { inherited from sibling }

Rule 1: is not L-attributed as C.i is defined using D.i which is not a parent child relation.

Rule 2: A.i depends on P.i { synthesized from ~~parent~~ parent. }

B.i depends on A.i and P.i { both synthesized from parent }

Rule 2: is L-attributed

Ques 10. Consider the grammar : $S \to A+B$ { S.val = A.val + B.val }

$A \to id$ { A.val = id.val }

$B \to id$ { B.val = id.val }

Ans 10. In $S \to A+B$.

S.val is Synthesized from A.val and B.val

Synthesized attribute

In $A \to id$ and $B \to id$

Both are synthesized from id.val

ans is (C) SDT is L-attributed and S-attributed.