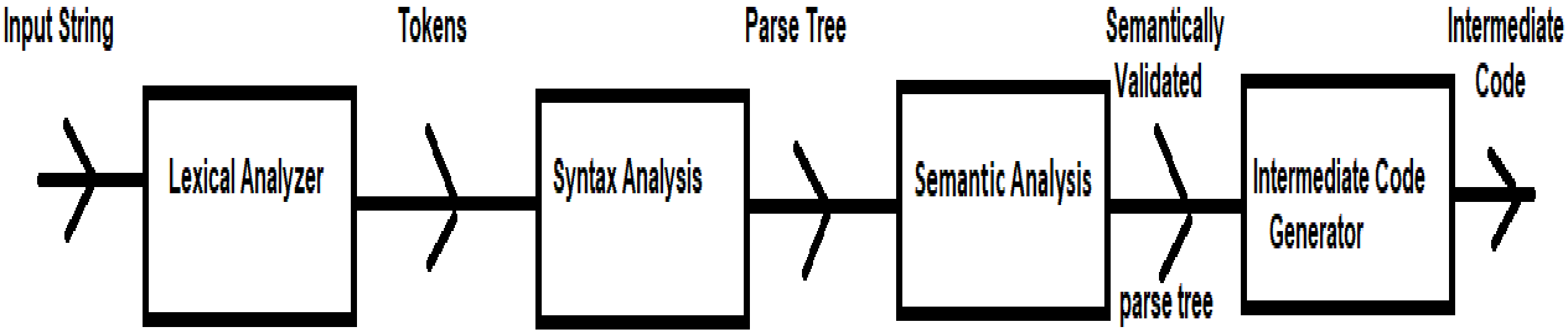
Syntax Directed Translation Scheme

Syntax Directed Translation



What is Semantic Action/Rules

- Alone CFG is not sufficient for conversion of parse tree to Intermediate code(IC). Some additional information is required for conversion of parse tree to IC.
- We attach some attribute with the variables (non terminals NT) of CFG and write some additional information corresponding to every production rule. This additional information tells the kind of action to be taken on execution of that production. **This extra added information is called semantic action (SA)**.
- SA is written in curly braces and attached with the production rule of the grammar. **SA is executed whenever parser recognizes the input string generated by CFG.**
- We can perform computation or printing of message through SA
  - E.g.: E→E+T {E.value=E.value+T.value}
  - here content written within parenthesis is called semantic action
  - value is the attribute of respective variable(non terminal)

Different ways to represent semantic action

There are two ways to represent the semantic rule/action associated with a grammar symbols:

- **Syntax Directed Definition (SDD)**: is a context-free grammar together with attributes and rules.
  - Attributes are associated with grammar symbol
  - Rules/action are associated with productions.

  SDD are highly readable and give high level specifications for translations. But they hide many implementation details. They don't specify the order of evaluation of semantic actions.
- **Syntax Directed Translation (SDT)**: SDT are more efficient than SDD as they indicate the order of evaluation of semantic actions associated within a production rule. It embeds semantic action within the production bodies. The process of syntax directed translation is two-fold:
  - Construction of syntax tree
  - Computing values of attributes at each node by visiting the nodes of syntax tree.

Types of Attributes

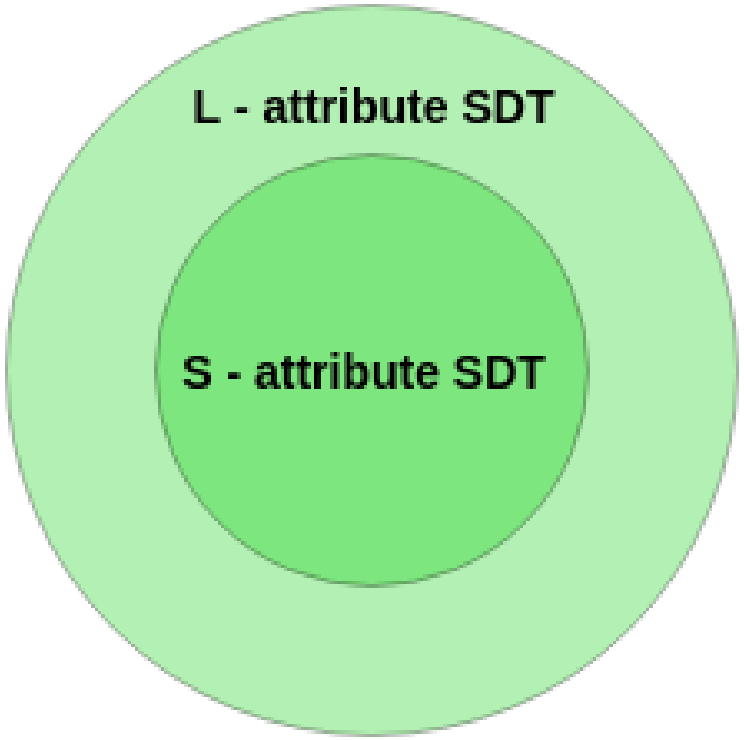
- There are two types of Attributes:
  - Synthesized attribute: These are those attributes which derive their values from their children nodes i.e. value of synthesized attribute at node is computed from the values of attributes at children nodes in parse tree.
  - Inherited Attributes: These are the attributes which derive their values from their parent or sibling nodes i.e. value of inherited attributes are computed by value of parent or sibling nodes.

Types of Syntax Directed Translation Scheme

- Based on the type of attributes used we can have two types of SDT scheme
  - S Attribute SDT
    - We use only synthesized attribute.
    - Semantic action are placed at right end of production. So it is also known as **Postfix SDT**.
    - Attributes are evaluated in bottom up parsing manner.
  - L Attribute SDT
    - It use both inherited and synthesized attributes. Each inherited attribute is restricted to inherit either from parent or left sibling (not right sibling) only.
    - Semantic action are placed anywhere on RHS in the production rule not necessarily after the symbols of the RHS of production rule.
    - Attributes are evaluated by traversing parse tree in **depth first left to right manner**.

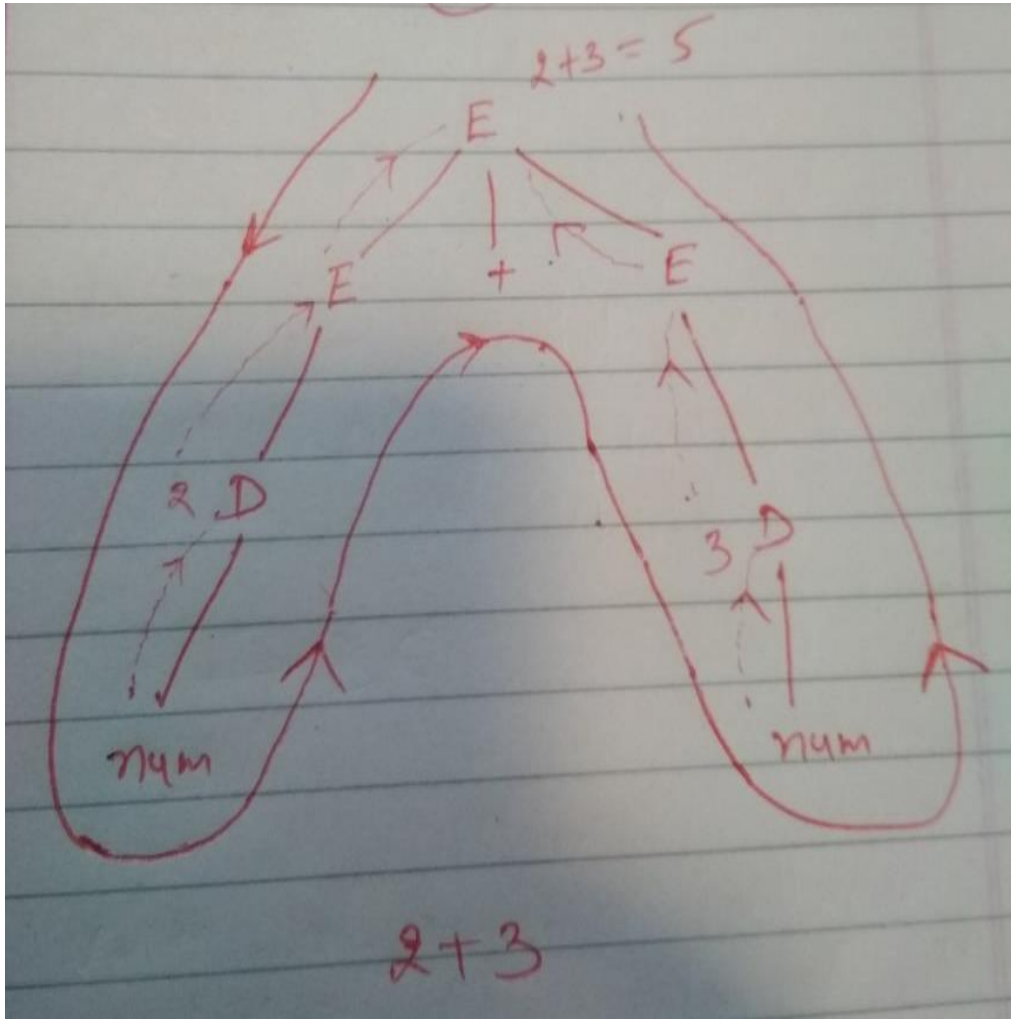
Types of Syntax Directed Translation Scheme

- If a definition is S-attributed, then it is also L-attributed but NOT vice-versa.



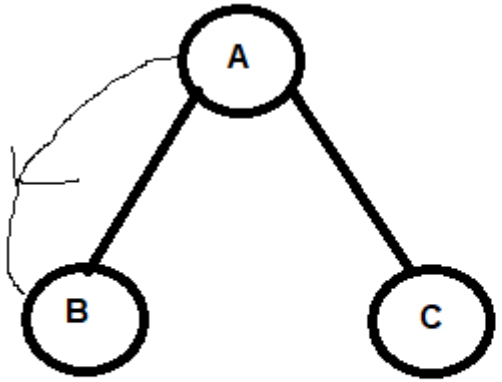
S-Attribute SDT scheme

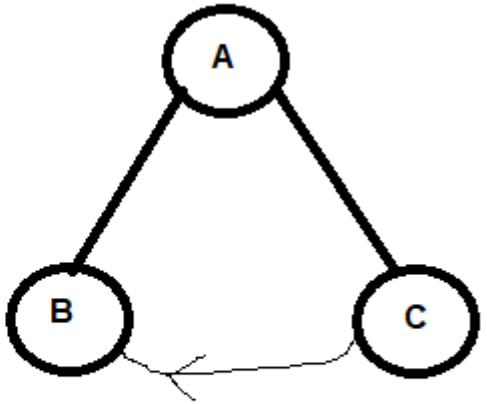
- In terms of **production rule**, value of variable on LHS of production rule depend on the value of variable on RHS of production rule only.
  - E→E+E {E.val= E.val+ E.val}
  - E→D {E.val= D.val}
  - D→num {D.val=num.val}
- In terms of **parse tree**, value of attribute at a node is computed from the value of attribute at the children of that node in the parse tree.
- The annotated parse tree is generated and attribute values are computed in **bottom up** manner.



L-Attribute SDT scheme

- In L attribute SDT, value of variable on RHS of production rule depends on translation of non terminal on **left side** of production rule or variable which is **left to it in the RHS**.
- Value of inherited attribute is computed from the value of attribute at **left sibling** or **parent node**.
- The annotated parse tree is generated and attribute values are computed in top down manner.
  - A→BC {B.val=5*A.val}
  - A→BC {C.val=B.val}





- A→BC {B.val=5*A.val}
- A→BC {C.val=B.val}

Types of Syntax Directed Translation Scheme

#1 A->LM {L.val=A.val, M.val=L.val, A.val=M.val}

A->QR {R.val=A.val, Q.val=R.val, A.val=Q.val}

(a) S-attribute SDT (b) L-attribute SDT (C) Both (d) None of the above

Types of Syntax Directed Translation Scheme

#1 A->LM {L.val=A.val, M.val=L.val, A.val=M.val}

A->QR {R.val=A.val, Q.val=R.val, A.val=Q.val}

(a) S-attribute SDT (b) L-attribute SDT (C) Both (d) None of the above

d is correct

Types of Syntax Directed Translation Scheme

#2 A->BC {B.val=A.val}

(a) S-attribute SDT (b) L-attribute SDT (C) Both (d) None of the above

Types of Syntax Directed Translation Scheme

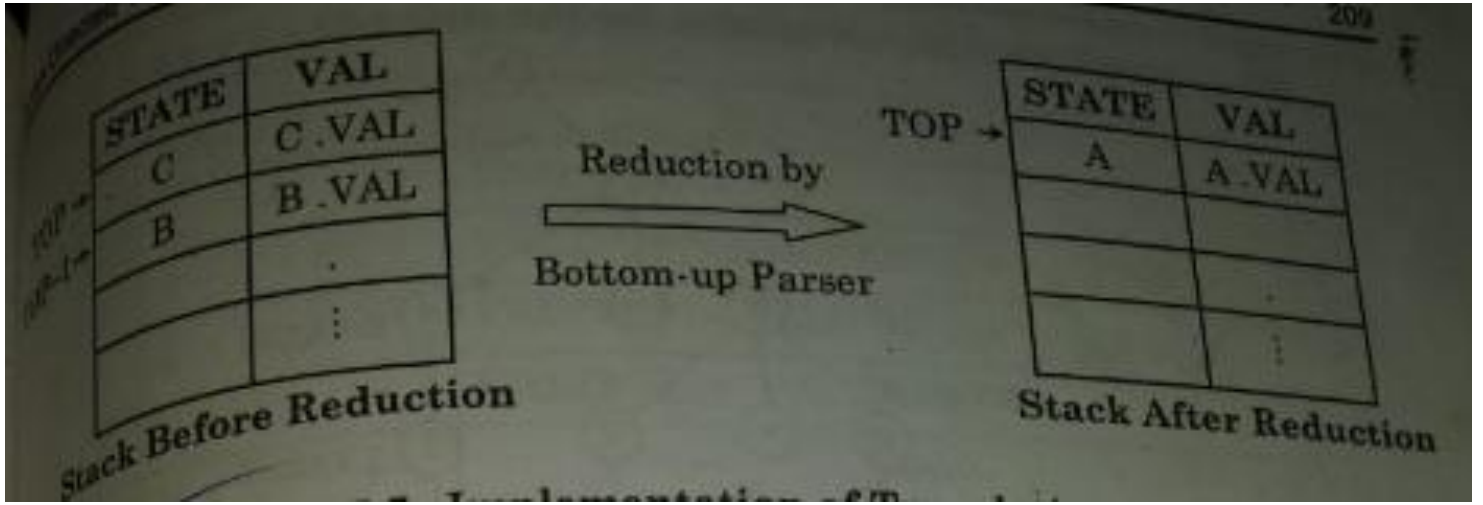
#2 A->BC {B.val=A.val}

(a) S-attribute SDT (b) L-attribute SDT (C) Both (d) None of the above

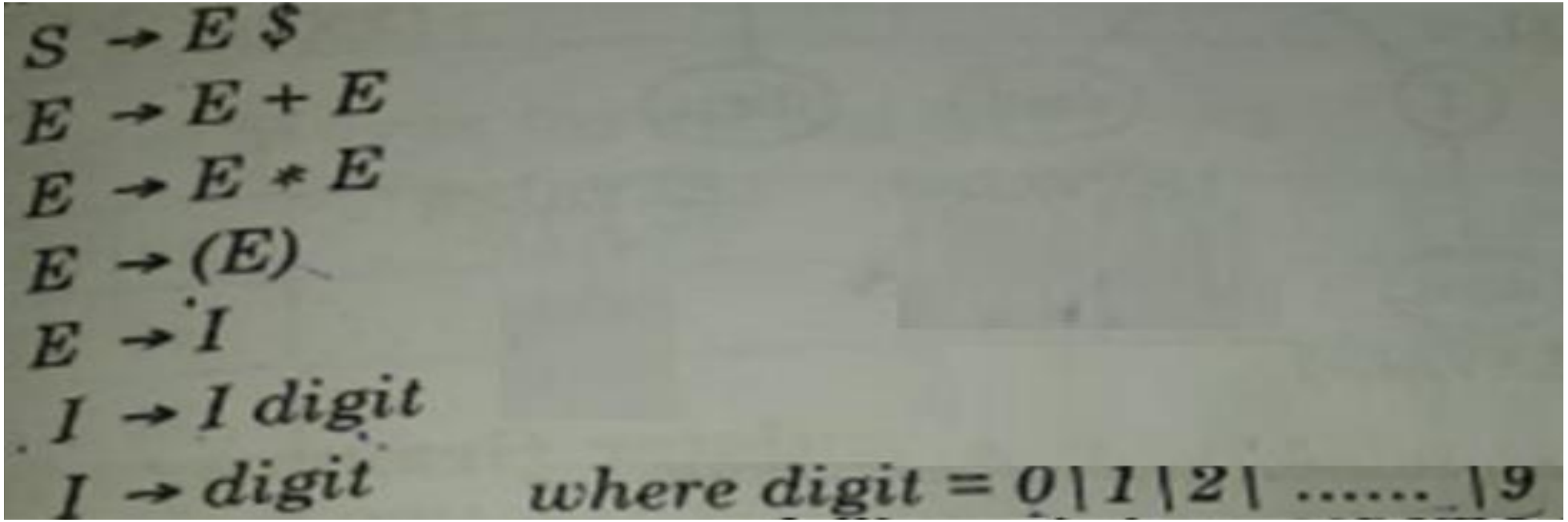
b is correct

Implementation of Syntax Directed Translators

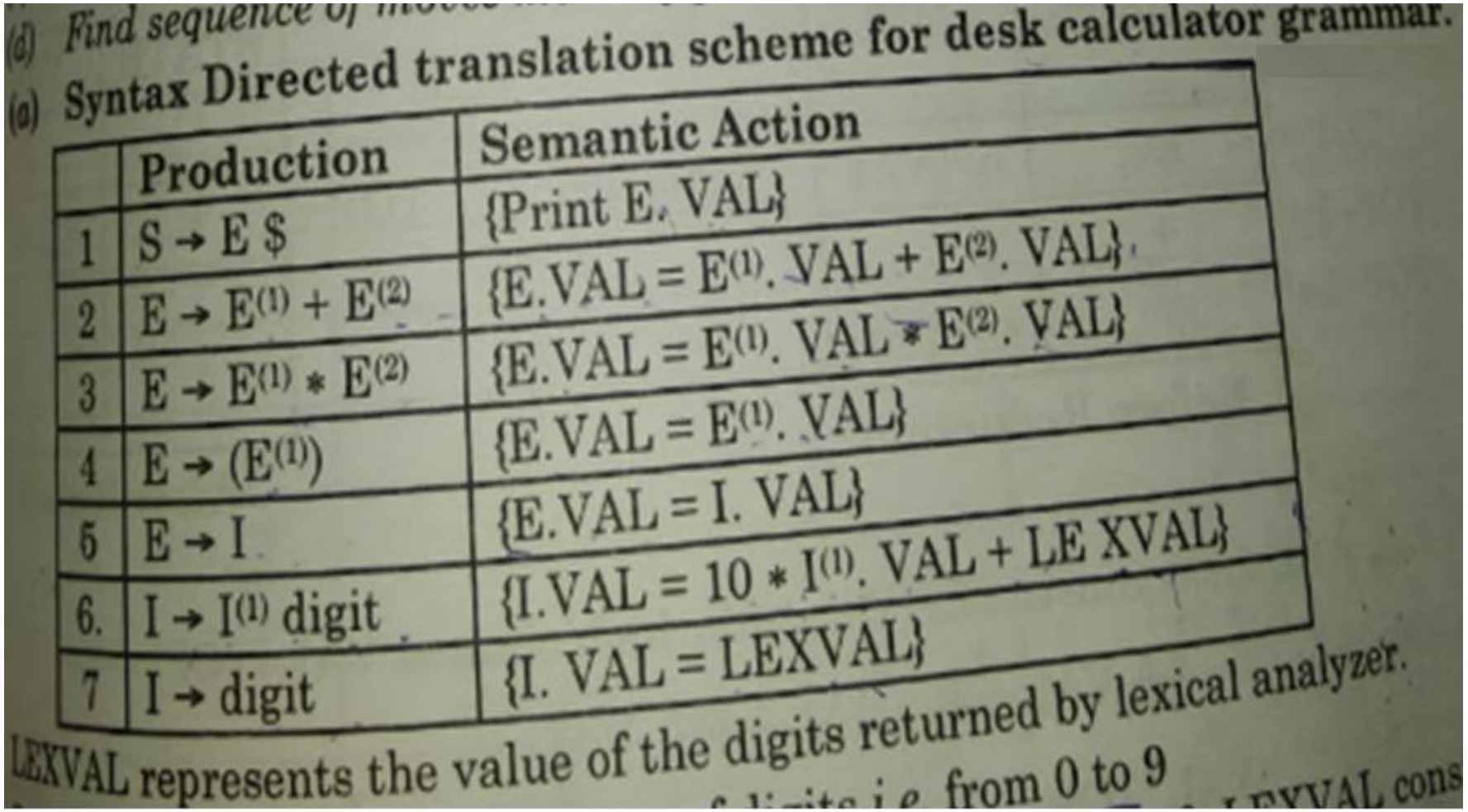
- SDT is implemented by constructing a parse tree and then parsing the input through parse tree in **top to down left to right manner**.
- SDT can be implemented using stack which consists of pairs of arrays “State” and “Val”.
- State entry shows a pointer to the parsing table. This array consists of all the non terminals appeared in CFG.
- Each Val represents the value associated with corresponding State symbol.
- E.g. A->B*C {A.val=B.val*C.Val} to compute the translation of variable A, first of all, we have to insert B and C into the stack with their corresponding values B.Val and C.Val respectively. The bottom up parser will perform the translation when BC will be reduced to A.



Implementation of Syntax Directed Translators



Implementation of Syntax Directed Translators



(d) Find sequence of

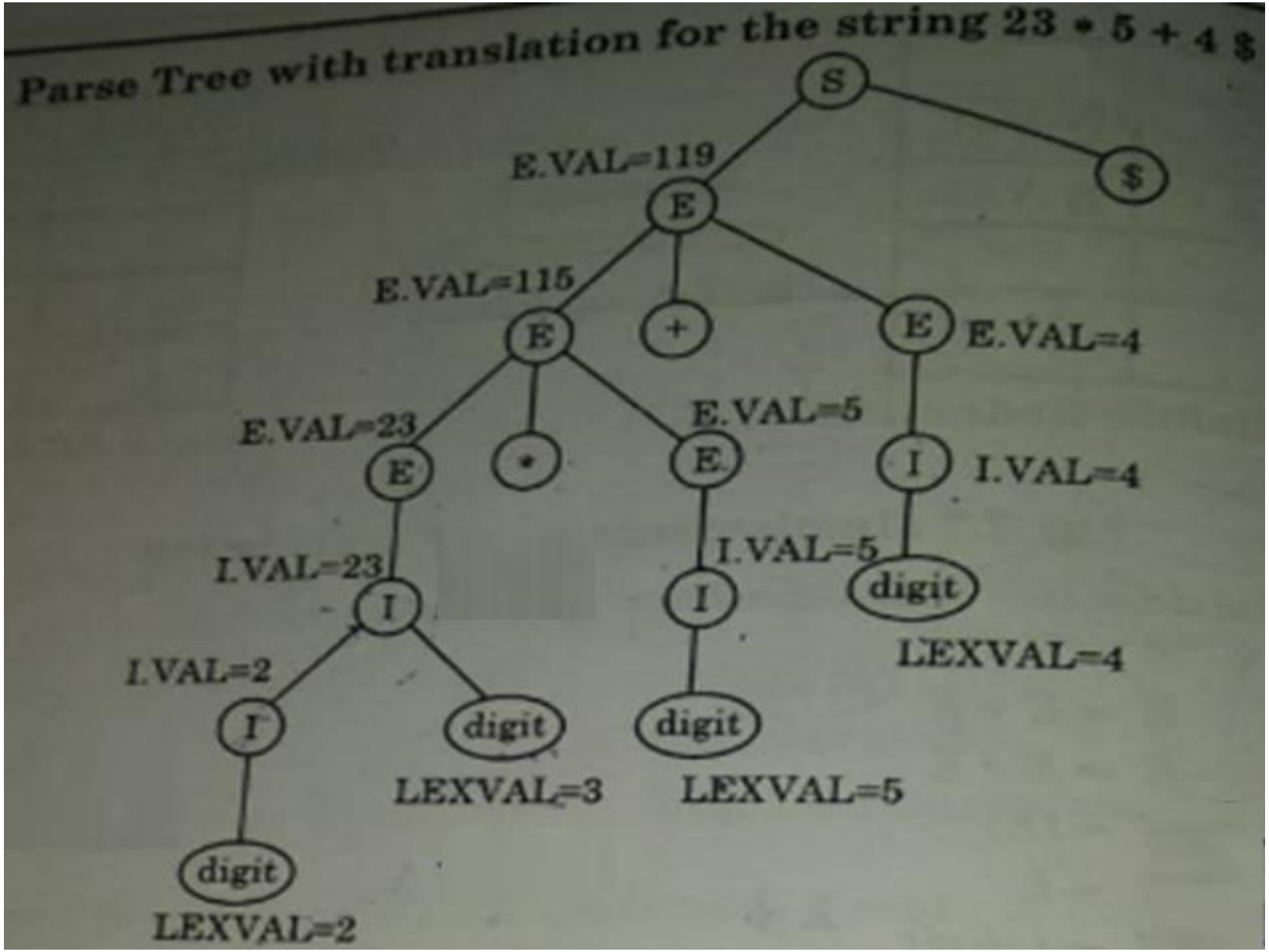
(a) Syntax Directed translation scheme for desk calculator grammar.

	Production	Semantic Action
1	$S \rightarrow E\ \$$	{Print E. VAL}
2	$E \rightarrow E^{(1)} + E^{(2)}$	{E.VAL = $E^{(1)}$. VAL + $E^{(2)}$. VAL}
3	$E \rightarrow E^{(1)} * E^{(2)}$	{E.VAL = $E^{(1)}$. VAL * $E^{(2)}$. VAL}
4	$E \rightarrow (E^{(1)})$	{E.VAL = $E^{(1)}$. VAL}
5	$E \rightarrow I$	{E.VAL = I. VAL}
6.	$I \rightarrow I^{(1)}$ digit	{I.VAL = 10 * $I^{(1)}$. VAL + LE XVAL}
7	$I \rightarrow$ digit	{I. VAL = LEXVAL}

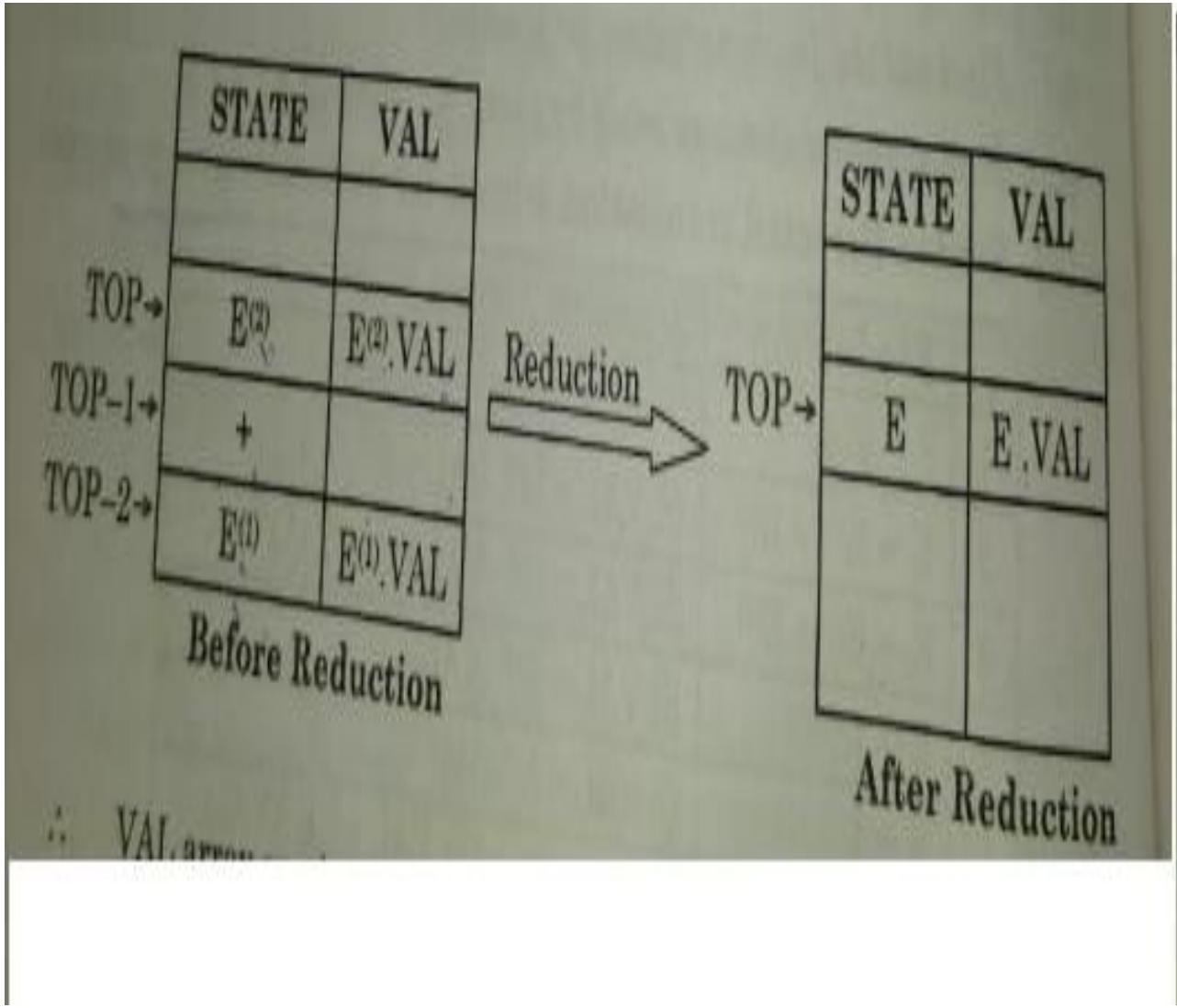
LEXVAL represents the value of the digits returned by lexical analyzer.

from 0 to 9

Implementation of Syntax Directed Translators



Implementation of Syntax Directed Translators



	Production	Semantic Action
1	S → E $	Print VAL [TOP]
2	E → E + E	VAL [TOP] = VAL[TOP-2] + VAL[TOP]
3	E → E * E	VAL [TOP] = VAL[TOP-2] * VAL[TOP]
4	E → (E)	VAL [TOP] = VAL [TOP-1]
5	E → I	None
6.	I → I digit	VAL [TOP] = 10 * VAL [TOP] + LEXVAL
7	I → digit	VAL [TOP] = LEXVAL