SYNTAX DIRECTED TRANSLATION (SDT).

All the operations after the parsing will be carried out with Syntax directed Translation.

A syntax directed Translation is a extension of the CFG, in which for each production, they attached Semantic rules (or semantic actions).

By evaluation of Semantic Rules, it may produce

1. intermediate code
2. It may generate target code.
3. It may issue error messages.
4. It may save the information in symbol table.
5. It may perform type checking.
6. In fact it peforms every thing.

There are 2 notations for associating, semantic rules with the production. —

1. ~~Semantic~~ Syntax directed definition (SDD)
2. Translations Scheme

SDD (~~Semantic~~ Syntax directed definition)

SDD is a high level specification for translation. They hide many information details and free the user from having the specific, explicitly the order in which translation takes place.

Translation scheme indicates the order in which semantic rules are to be evaluated, so they allows some implementation details to be shown.

Detailed version of SDD.

A syntax directed definition is generalization of the context free grammar in which each grammar symbol associated with set of attributes, partition into 2 subsets called Synthesized and inherited of that grammar symbol.

An attribute can represent anything we choose namely a string, a no, a type, a memory location or whatever.

The value of the attributes at a parse tree nodes is defined by semantic rules associated with each production use at that node.

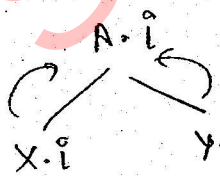
Synthesized attributes are evaluated in bottom-up order, i.e. A attribute is said to be synthesized if its value at a parse tree node is determined by attribute values at the child nodes.

An Inherited attribute is a one whose value at a node in a parse tree is defined in terms of attribute at a parent and are sibling of that node.

eg. Synthesized

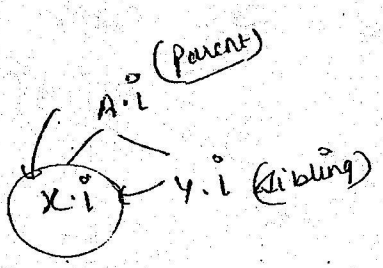
$A \to XY \{ A.i = X.i * Y.i \}$

A.i
X.i
Y.i

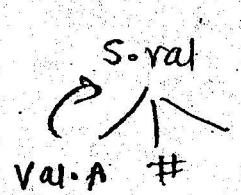


Inherited

$A \to XY \{ X.i = 2 * A.i + Y.i \}$



eg. $S \to A \# B$ $\{ S.val = 2 + A.val \}$



So, Synthesized.

eg. $S = A \# B$ $\{ B.val = A.val * 2 \}$

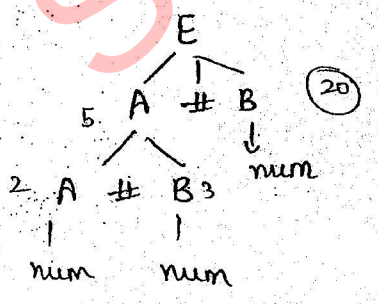
↳ Inherited (sibling).

Q consider the following SDD. in which each variable associated with the attribute val. left side variable attribute determine using right side variable attributes.

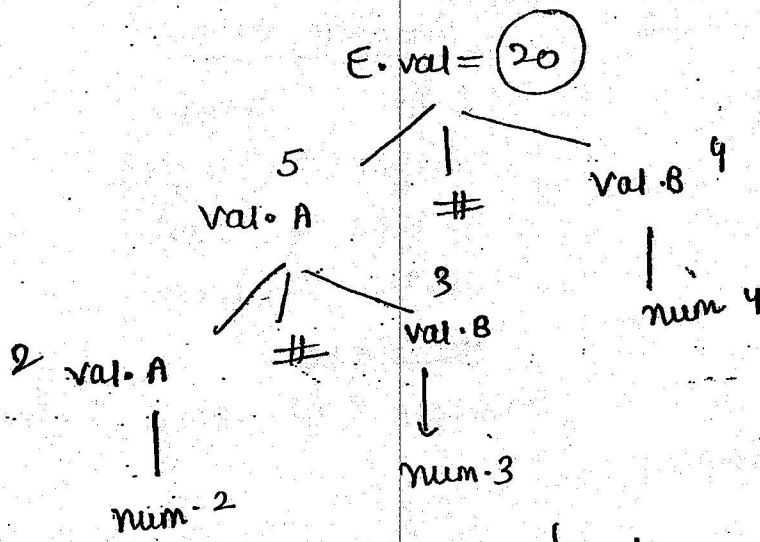
$E \to A \# B$ $\{ E.val = A.val * B.val \}$
$A \to A \# B$ $\{ A.val = A.val + B.val \}$
$A \to num$ $\{ A.val = num.lex \}$
$B \to num$ $\{ B.val = num.lex \}$

Determine the output produced by the SDD for the given input string $w = 2 \# 3 \# 4$. Where num is a token, lex is a value associated with num.

⇒



o/p :- (20).



Abstract syntax tree.

↳ Known as · Annotated parse tree

Decorated parse tree.

Q. Consider the following SDD. means previous value.

$E \rightarrow E+T$ { E.val = E1.val || T.val || + }

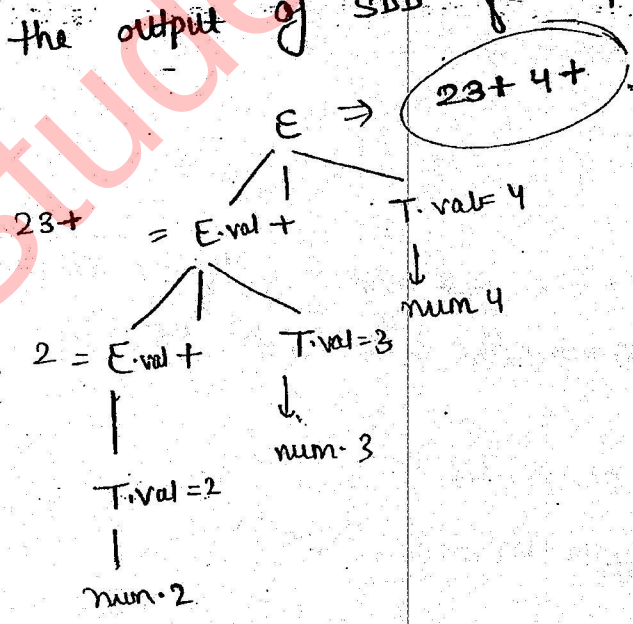
$E \rightarrow T$ { E.val = T.val }

$T \rightarrow Num$ { T.val = num.lex }

Where || represents concatenation operation.

num is token associated with some lexem

find the output of SDD for input string w = 2+3+4.

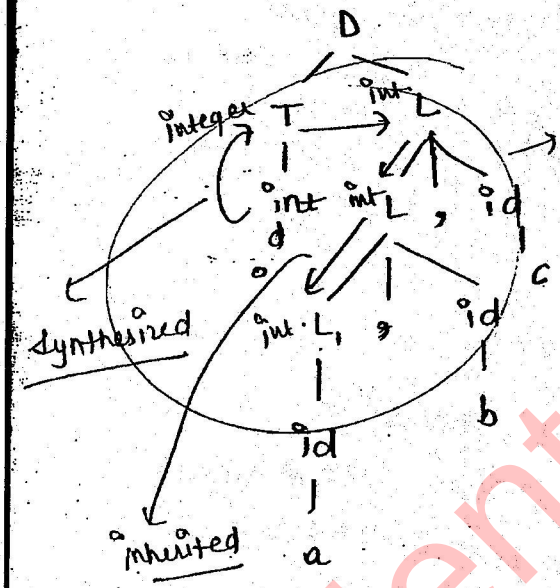


Q. Consider the following SDD.

Semantic rules
$D \to TL$	
$T \to int$	$L.in = T.type$
$T \to real$	$T.type = integer$
$L \to L_1, id$	$T.Type = real$
$L \to id$	$L_1.in = L.in$
	~~addtype~~
	addtype (id.entry, L.in)

Input string is suppose
int a, b, c.

SDT for declaration of variable.



→ combination of synthesized and inherited attribute.

There are 2 types of SDD :-

1. S-attributed definition.
2. L-attributed definition.

A definition is said to be S-attributed, if it contains only synthesized attributes otherwise, it can be L-attributes.

Synth S-attribute :- Synthesized
L-attribute :- Synthesized + Inherited.

S-attributed definitions are evaluated in Bottom up order

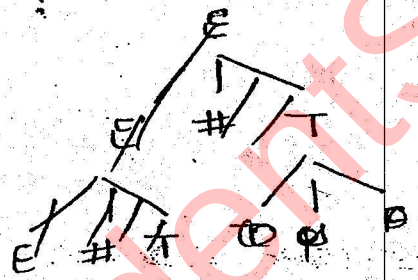
where L-attributed definitions are evaluated in Top down order.

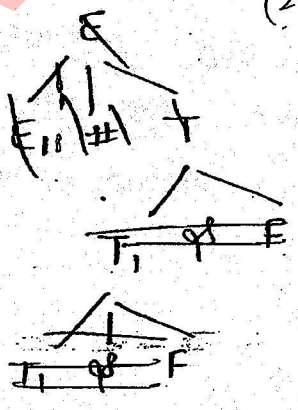
Q. Consider the following SDD and E as a start symbol
Q. Each variable associated with attribute val.

$E \rightarrow E_1 \# T$		$\{E.val = E_1.val * T.val\}$
$E \rightarrow T$		$\{E.val = T.val\}$
$T \rightarrow T_1 \$ F$		$\{T.val = T_1.val + F.val\}$
$T \rightarrow F$		$\{T.val = F.val\}$
$F \rightarrow num$		$\{F.val = num.val\}$

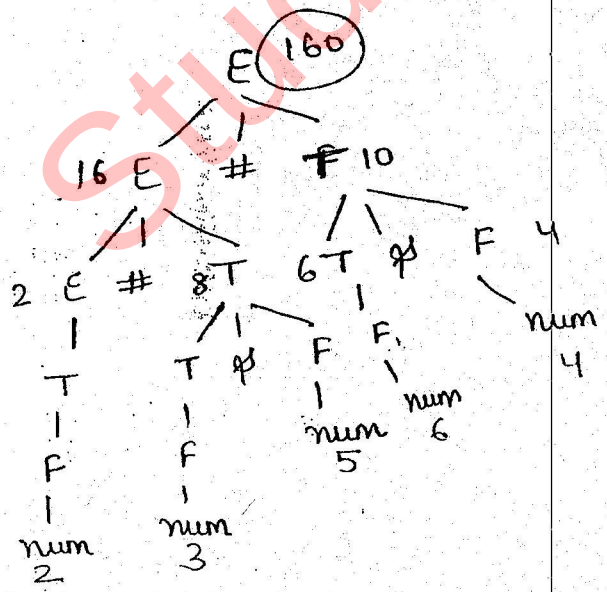
Compute E.val for root node of the parse tree for exp. $2 \# 3 \$ 5 \# 6 \$ 4$.

(a). 40
(b). 160
(c). 180
(d). 200





$(2*3)+(5*6)+4$
$6+30+4$
40



TRANSLATION SCHEME

A Translation scheme is a ~~Syntax~~ context free grammar in which semantic actions are enclosed between braces and they are inserted in right side of the production anywhere.

The translation schemes are evaluated by travelling in depth first order.

Q. Consider the following Translation scheme

$E \rightarrow TR$ #1

$R \rightarrow +T\{ print\{'+'\} \}\ R_1 \mid \varepsilon$ #2

$T \rightarrow num\ \{ print\{num.val\} \}$

Here num is a token that represents an integer and num.val represent the corresponding integer value. for the input string 9+5+2 this translation scheme will print that :-

(a) 9+5+2

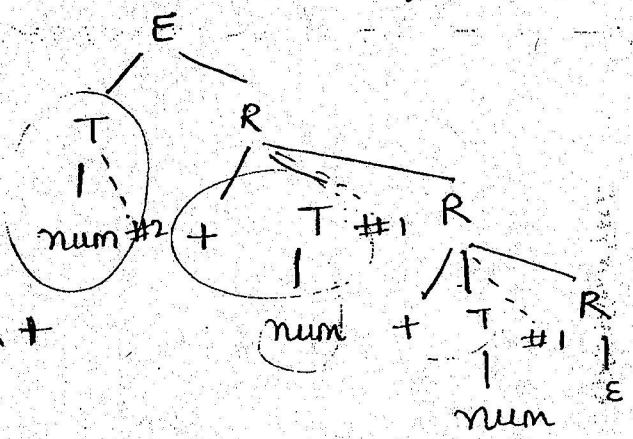
(b) 95+2+

(c) 952++

(d) ++92

↳ also done postfix

Move in depth first order



95+2+

Q. Consider the following SDTS. (syntax directed Translation scheme) with the non-terminals ~~S, A~~ S, A and terminals a, b.

$S \rightarrow aA$ { print 1 }

$S \rightarrow a$ { print 2 }

$A \rightarrow Sb$ { print 3 }

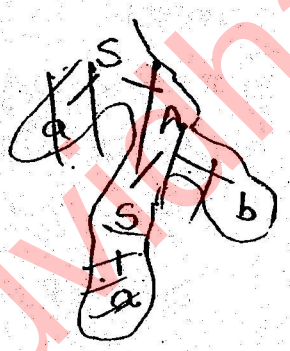
Using the above SDTS the output printed by bottom up parser method for the input string w = aab.

(a). 132

(b). 223

(c). 231

(d). Syntax error



When bottom-up parser is used, don't construct tree.

w = aab — print ②

= aSb — ③

= aA — ①

⇒ S.

Q. Consider the following grammar :-

$S \rightarrow F \mid H$

$F \rightarrow P \mid C$

$H \rightarrow d \mid C$

stm. ① ~~LL(R)~~. LL(1) can parse all string that are generating using Grammar G.

stm. ② LR(1) can parse all strings that are generated by using grammar G.

(a) only S_1 is correct

(b) only S_2 is correct

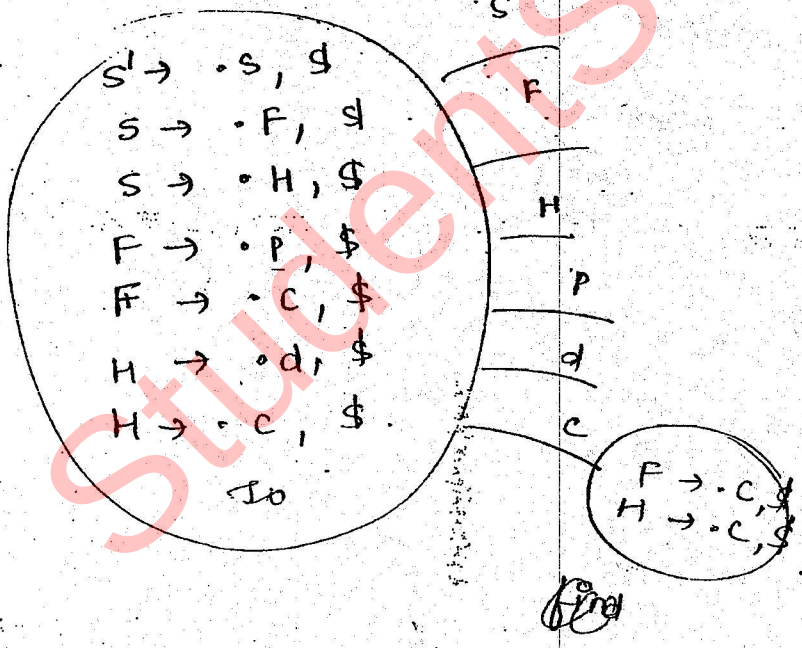
(c) Both S_1 and S_2 are correct

(d) Neither S_1 and S_2 is correct.

Sol. Not LL(1) ∵ $FT(F) \cap FT(H)$

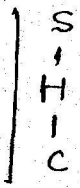
$\Rightarrow \{P, C\} \cap \{d, C\}$

$\neq \phi$



and also, it is ambiguous.

for string c we can draw 2 trees



So not LR(1).