SYNTAX DIRECTED TRANSLATION

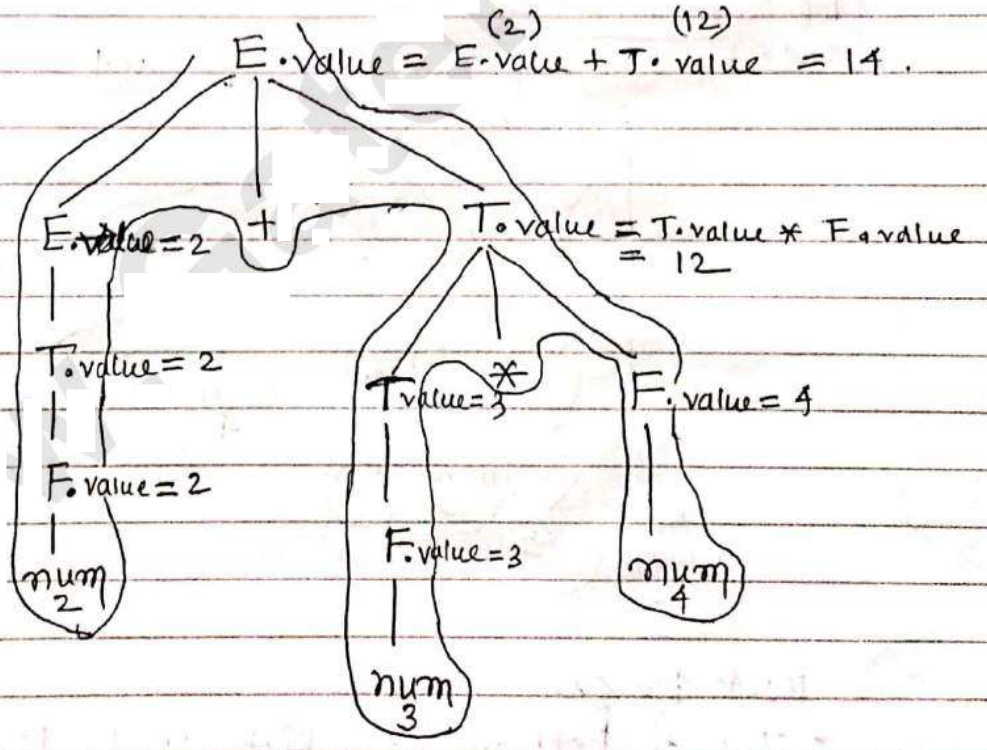
Grammar + Semantic rules = SDT (Syntax directed Translation)

STD for evaluation of expression —

	grammar	rules
(1)	$E \rightarrow E+T$	$\{E.value = E.value + T.value\}$
	$/T$	$\{E.value = T.value\}$
	$T \rightarrow T*F$	$\{T.value = T.value * F.value\}$
	$/F$	$\{T.value = F.value\}$
	$F \rightarrow num$	$\{F.value = num.lvalue\}$

ex —

2+3*4



→ When reduction will occure, then take production and take action.

(2)

Symantic action (1)

$E \rightarrow E + T$ { print ("+"); }

/ T { } (2)

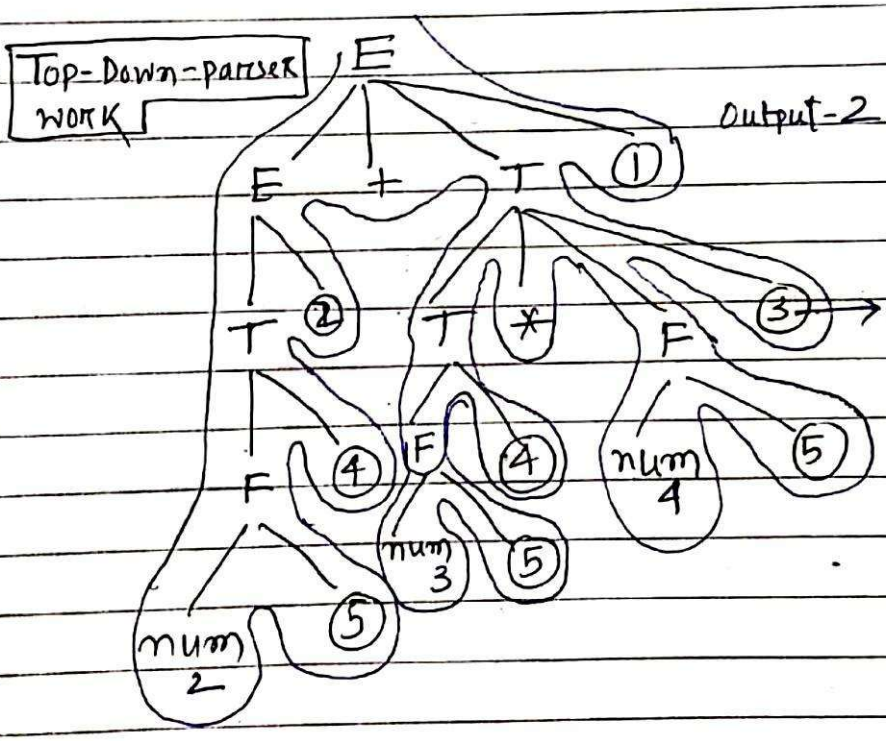
$T \rightarrow T * F$ {print("*"); } (3)

/ F { } (4)

$F \rightarrow num$ { print("num.lval); } (5)

; Top down parser and Bottom-up-parser work with SDT =

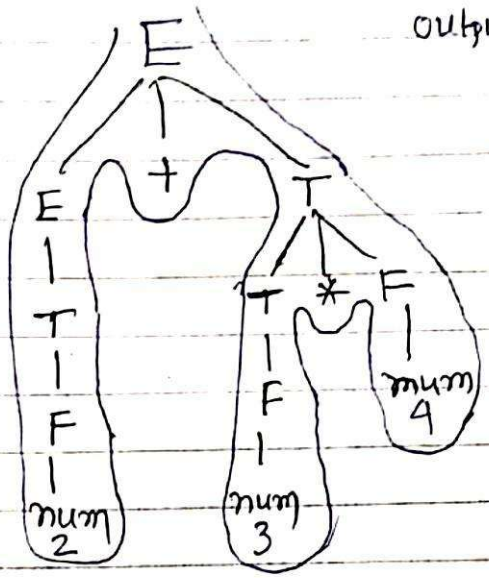
ex = 2+3*4



→ Top to Bottom and Right left to Right. When see any sematic number then take action.

Bottom - Down - parser work

output - 2 3 4 * +



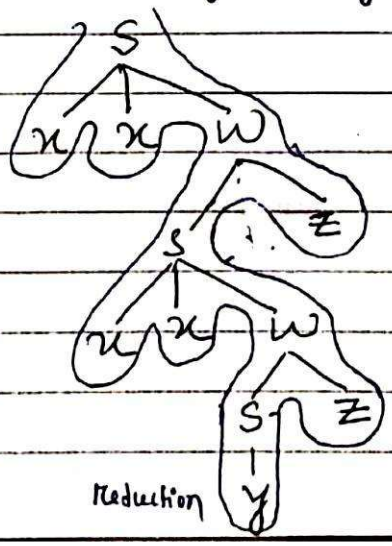
→ using TDP and BUP we get same output.

(3)

S → xxW {printf(1);}
/ y {printf(2);}
W → Sz {printf(3);}

String xxxxyzz.

generate given string using Button Down parser —



output - 2 3 1 3 1

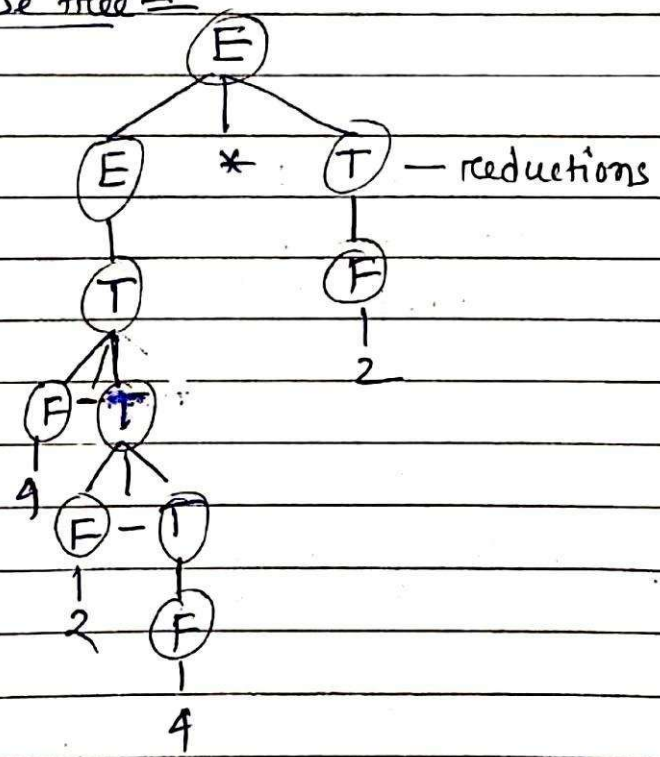
(4) $W = 4-2-4*2$

$E \rightarrow E*T \quad \{E.val = E.val * T.val;\}$
$\quad /T \quad \{E.val = T.val;\}$
$T \rightarrow F-T \quad \{T.val = F.val - T.val;\}$
$\quad /F \quad \{T.val = F.val;\}$
$F \rightarrow 2 \quad \{F.val = 2;\}$
$\quad /4 \quad \{F.val = 4;\}$

↳ In this grammar $- > *$.
'*' is left associative.
'-' is Right associative.

↳ (shortcut rule) by not drawing parse tree. $W = (4-(2-4)*2) = 12$

parse tree =



— reductions

→ Total number of reduction is - 10.

⑤

	Semantic action.
$E \rightarrow E \# T$	$\{ E.val = E.val \# * T.val ; \}$
$/ T$	$\{ E.val = T.val ; \}$
$T \rightarrow T \& F$	$\{ T.val = T.val + F.val ; \}$
$/ F$	$\{ T.val = F.val ; \}$
$F \rightarrow num$	$\{ F.val = num.lval ; \}$

$w =$ 2 # 3 & 5 # 6 & 4 → input.

→ In given grammar (+ > *). '+' has higher precedence.

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

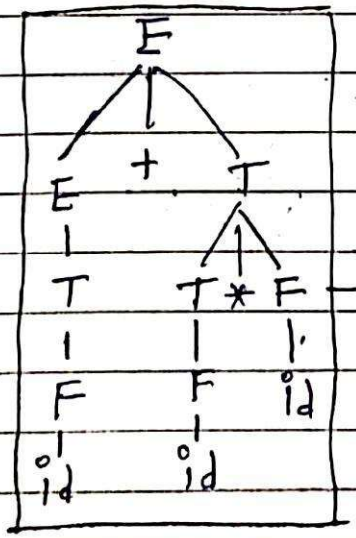
$= ((2 * 8) * 10)$

$= 160$ (output) (quick way)

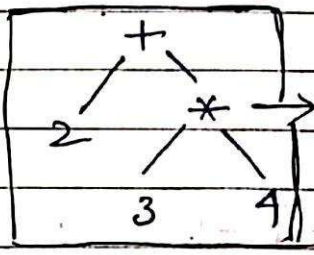
→ Other wise draw parse tree, then find output.

- Abstract and concrete parse tree –

2+3*4



→ Concrete parse tree.



Abstract parse tree.

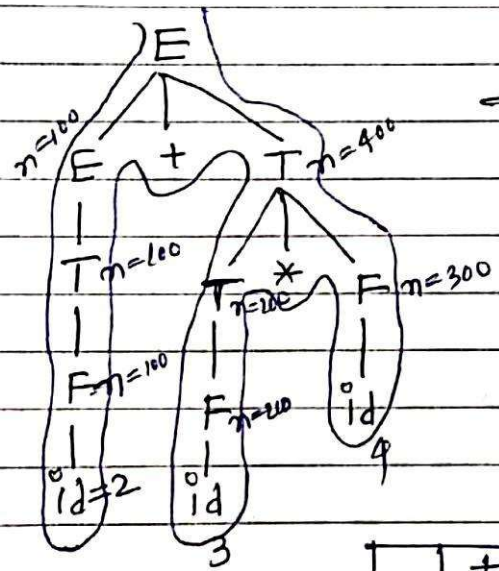
(Variables are not shown)

nptr → node pointer

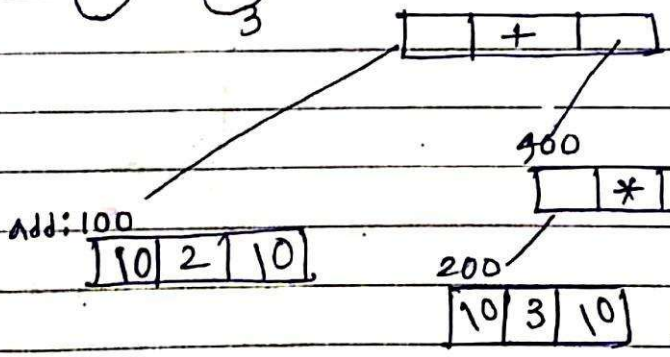
- **STD to build syntax tree =**

①	Rule
$E \to E+T$	{ E.nptr = mknode(E1.nptr, '+', T.nptr); }
/ T	{ E.nptr = ... T.nptr ; }
$T \to T*F$	{ T.nptr = mknode(T.nptr, '*', F.nptr); }
/ F	{ T.nptr = ... F.nptr ; }
$F \to id$	{ F.nptr = mknode(null, id name, null); }

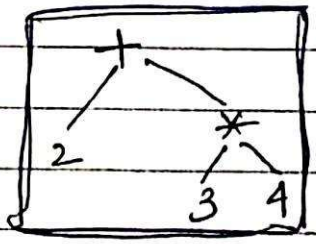
ex- 2 * 3 * 4



→ annotated parse tree.
(parse tree showing the value of attributes at each node).



→ Datastructure of abstract parse tree



→ abstract syntax tree.

- **STD for type checking** =

① $E \rightarrow E_1 + E_2$ { if ((E_1.type == E_2.type) && (E_1.type = int)) then E.type = int else error; }.

 / E_1 == E_2 { if ((E_1.type == E_2.type) && (E_1.type = int/boolean)) then E.type = boolean else error; }

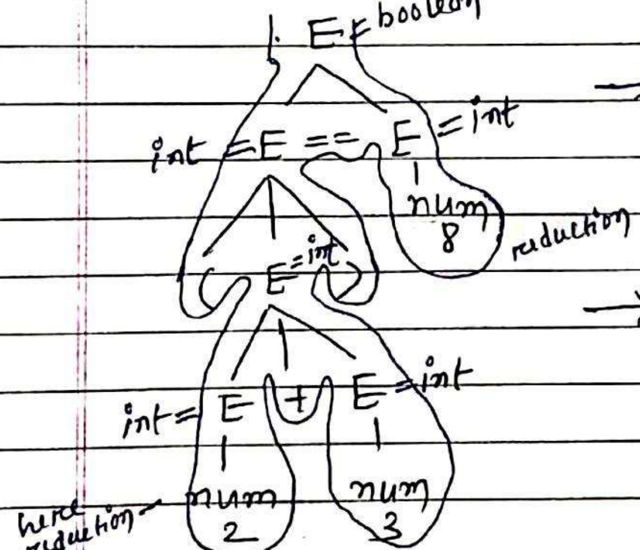
 / (E_1) { E.type = E_1.type; }

 / num { E.type = int; }

 / True { E.type = bool; }

 / False { E.type = bool; }

ex — (2+3) == 8 (find the type of expression)



→ according to give rules.

→ So the given expression is Boolean expression.

- Given a string, count no. of 1's in string using this grammar (using SDT).

	count 1's	count 0's	countbit
N → L	{N.C = L.C}		
L → LB	{L.C = L.C + B.C}		
/B	{L.C = B.C}		
B → 0	{B.C = 0}	B.C = 1	B.C = 1
/1	{B.C = 1}	B.C = 0	B.C = 1

here, N → Number, L → List of bit, B → bit

B.C → Bit count

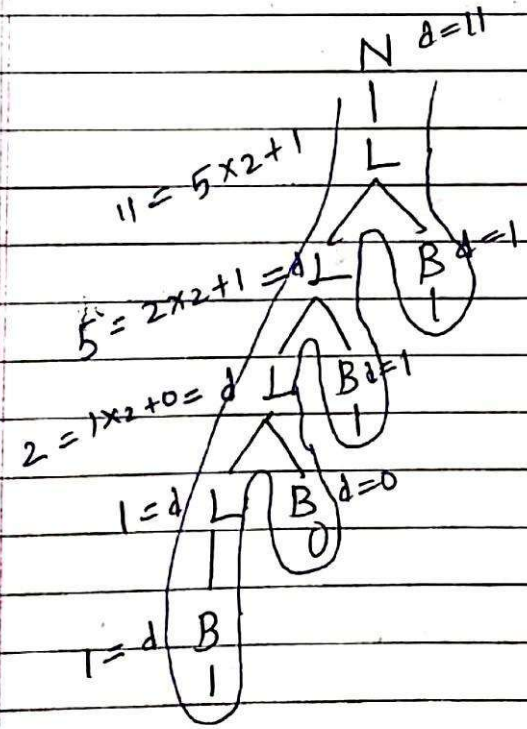
• Convert binary to Decimal using the grammar →

10110	1011 1 (b)
11×2+0	11×2+1
=22	=23

N → L	{N.dval = L.dval}
L → L₁B	{L.dval = L₁.dval * 2 + B.dval}
/B	{L.dval = B.dval}
B → 0	{B.dval = 0}
/1	{B.dval = 1}

(d.val → decimal value)

Ex = find out the decimal value of given string –
(SDTS) 1011 → decimal value (11).



(d → decimal)

11.01 → you should chang the ~~the~~ rule to
3.25 Convert 11.01 to ~~Bol~~ Decimal

- Convert binary number with Decimal point to Decimal grammar =

11.01	11.11
$.01 = \frac{1}{2^2} = 0.25$	$.11 = \frac{3}{2^2} = \frac{3}{4} = 0.75$
$11.01 = 3.025$	$11.11 = 3.75$

S-rule

Convert

$N \rightarrow L_1 . L_2 \quad \{ N.dval = L_1.dval + \frac{L_2.dval}{2^{L_2.c}} \}$

$L \rightarrow L_1 B \quad \{ L.c = L_1.c + B.c ; L.dval = d_1.dval + B.dval ; \}$

$/B \quad \{ L.c = B.c ; L.dval = B.dval ; \}$

$B \rightarrow 0 \quad \{ B.c = 1 ; B.dval = 0 \}$

$/1 \quad \{ B.count = 1 ; B.dval = 1 \}$

- SDT to generate three address code = (Semantic rule)

T → term
F → Factor

$S \rightarrow id = E \quad \{ gen(id.name = E.place) ; \}$

$E \rightarrow E_1 + T \quad \{ E.place = newTemp() ; gen(E.place = E_1.place + T.place) ; \}$

$/T \quad \{ E.place = T.place ; \}$

$T \rightarrow T * F \quad \{ T.place = newtemp() ; gen(T.place = T_1.place * E.place) ; \}$

$/F \quad \{ T.place = F.place ; \}$

$F \rightarrow id \quad \{ F.place = id.name ; \}$.

ex =

$x = a + a * c$

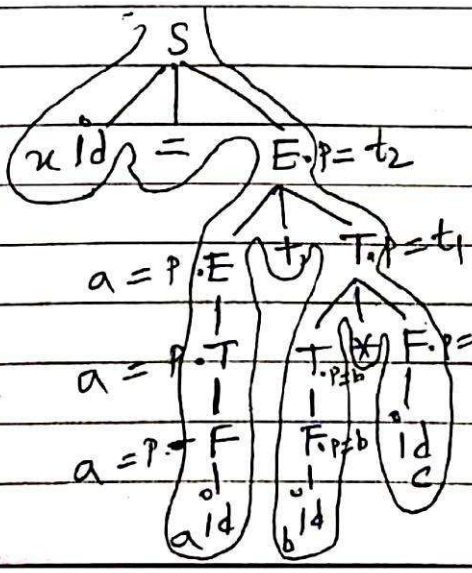
output :

$t_1 = b * c$

$t_2 = a + t_1$

$x = t_2$

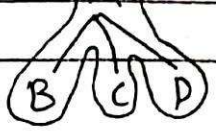
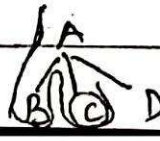
Bottom of parsing



- SDT are two types =
  (i) S-attribute SDT.
  (ii) L-attribute SDT.

A → BCD
A.s → f(B.s, C.s, D.s)
('A' is taking value from its children) → synthesized attributes.

A → BCD
C.i → A.i
C.i → B.i
C.i → D.i
→ if 'C' taking attribute from its parents and siblings, it called inherited attribute

S-attributed SDT	L-attributed SDT
1) Uses only synthesized attributes	1) Uses both inherited and synthesized attributes. Each inherited attribute is restricted to inherit either from ~~part~~ parent or left siblings only. Ex: A → XYZ { Y.S = A.S ✓, Y.S = X.S ✓, Y.S = Z.S }
2) semantic ~~atribut~~actions are placed at right end of production. A → BBCC{}	2) Semantic actions are place anywhere on R.H.S.
3) Attributes are evaluated during Bottom-Down-parsing. 	3) Attributes are evaluated by traversing parse tree depth first, left to right. 

Attribute = value

L-attri
S-attri

classmate
Date
Page 79

[example] — findout SDT are S-attributed or L-attributed =

Inherited Not S-att

① $A \rightarrow LM$ { L.i = f(A.i); M.i = f(L.s); A.s = f(M.s); }

$A \rightarrow QR$ { R.i = f(A.i); Q.i = f(R.i); A.s = f(Q.s); }

not L-att

ⓐ S-attributed ⓒ both

ⓑ L-attributed ⓓ none.

② $A \rightarrow BC$ { B.s = A.s; } not s-attri

ⓐ S-attri ⓒ both

ⓑ L-attri ⓓ none

- SDT to store type information into Symbol table =

① grammar + rule = SDT

$D \rightarrow TL$ { L.in = T.type } [L-attribute]

$T \rightarrow$ int { T.type = int; }

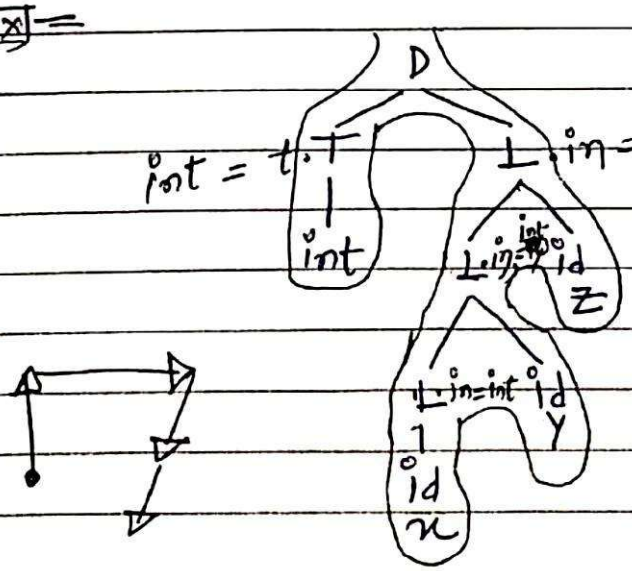
/ char { T.type = char }

$L \rightarrow L_1, id$ { $L.in = L_1.in$, addtype(id.name, $L_1.in$); }

/ id { addtype(id.name, L.in) }

→ (this SDT are L-attributed defination.) int x, y, z

ex =



T → type

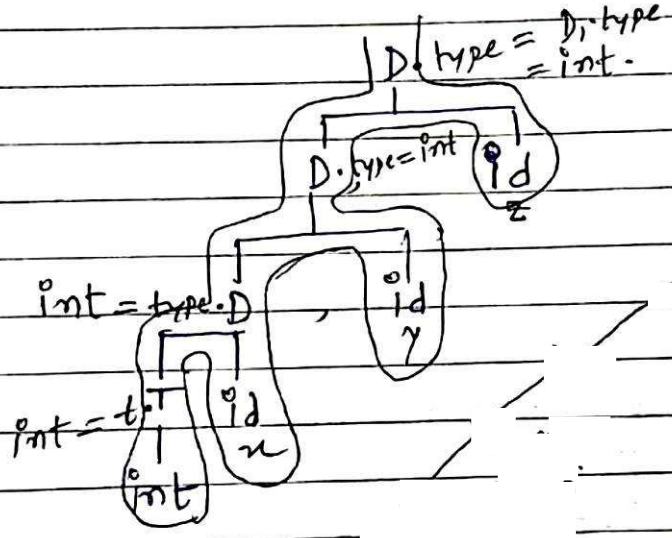
id name | int
x | int
y | int
z | int

(2) **S-attributed**

$D \rightarrow D_1, id$ { add-type (id.name, D_1.type) }; D.type = D_1.type }
/ T id { add-type (id.name, T.type), D.type = T.type; }

$T \rightarrow$ int { T.type = int; }
/ char { T.type = char; }

Ex: int x, y, z



x	int
y	int
z	int