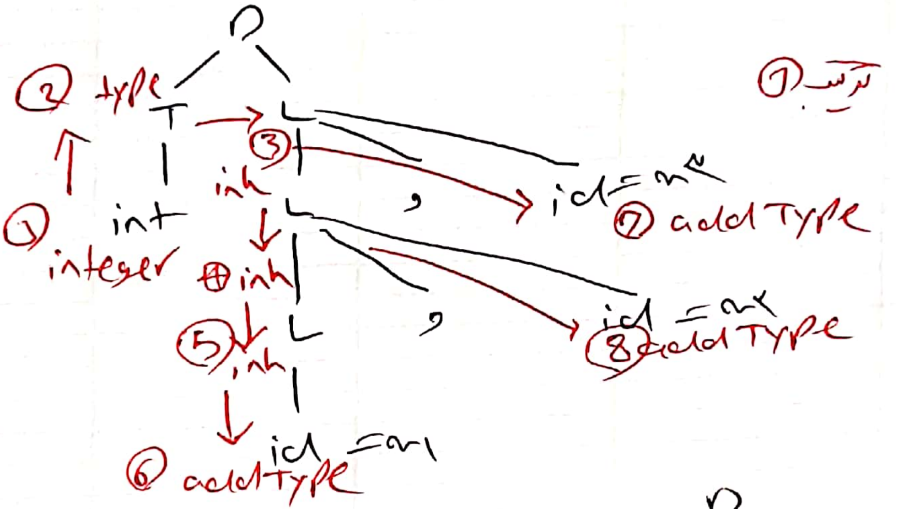
(اول

D→TL	L.a = T.type	inherited
T→int	T.type = integer	syn integer [illegible] ورودی
T→float	T.type = float	

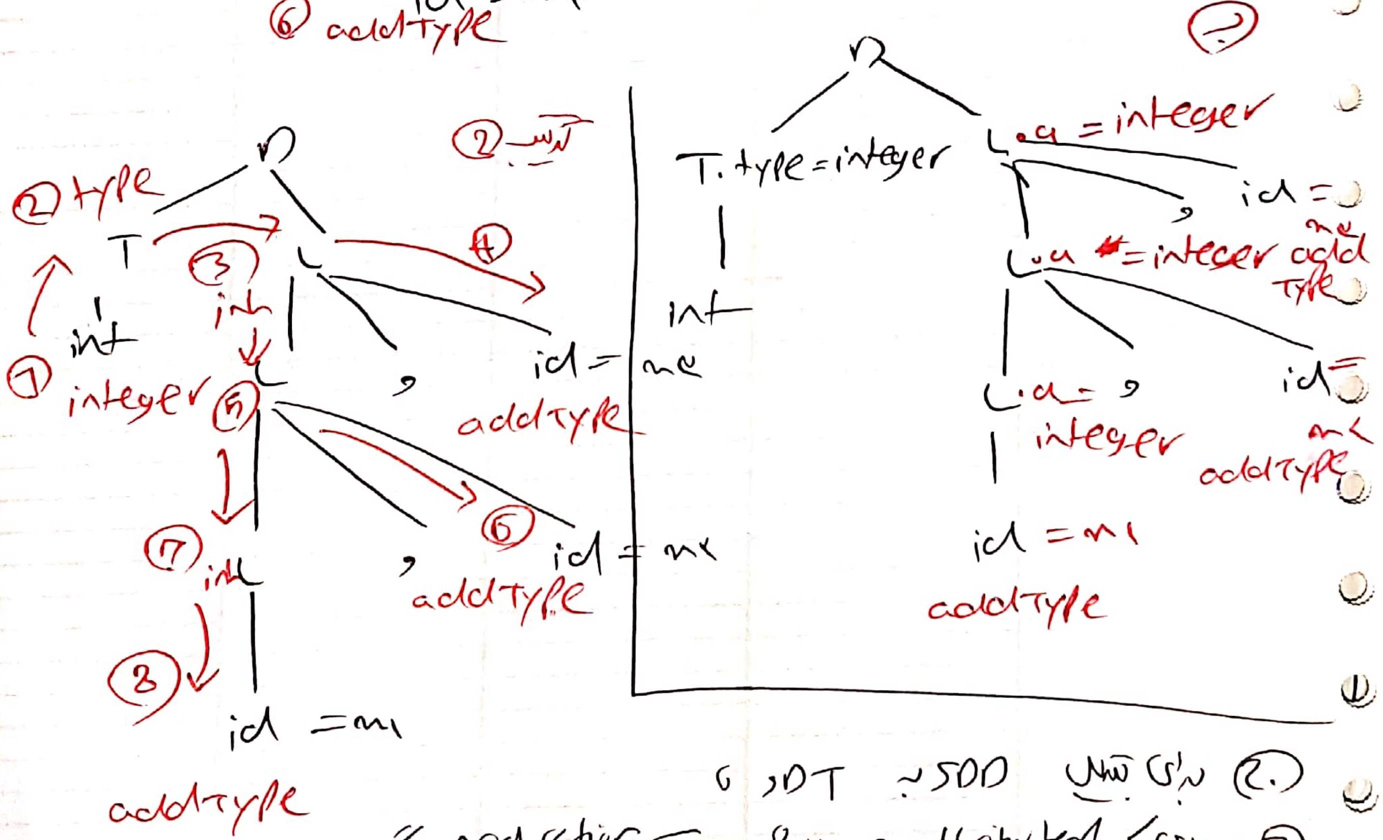
L→L1,id	L1.a = L.a
	addType(id.entry, L.a)
		inherited

L→id	addType(id.entry, L.a)	

(الف



ب



2.) برای تبدیل SDD به SDT ،

1) معانی که s_attributed باشه ← آخر production می‌گذاریم.

2) ~ inherited attr باشه ← قبل از سمبلش می‌گذاریم.

1) D→T{L.a = T.type}L

2) T→int {T.type = integer}

3) T→float {T.type = float}

4) L→{L1.a = L.a}, {addType(id.entry, L.a)} id;

5) L→{addType(id.entry, L.a)} id

تبدیل SDD به SDT　　　مثال ۲

T → F { T'.inh = F.val } T' { T.val = T'.syn }

T' → *F { T_1'.inh = T'.inh * F.val } T_1' { T'.syn = T_1'.syn }

T' → ε { T'.syn = T'.inh }

F → digit { F.val = digit.lexval }

پیاده‌سازی با روش top down ،

recursive descent ← به ازای هر non_terminal یک تابع داریم که ورودی آن

inherited attr ها و خروجی آن synthesized attr ها هستند.

```
val T( ){                    ①
    val  val_F , val_T'_inh , val_T'_syn , val_T

      val_F = F();
      val_T'_inh = val_F
        val_T'_syn  = T'(val_T'_inh);
        val_T  = val_T'_syn ;   return val_T}
```

```
②
  val  T'(val T'_inh){
 val val_F, val_T1'_inh, val T'_inh,
val_T1'_syn,   if ( current  == '*'){
val_T'_syn;          match('*');
                     current ++;
                  val_F = F();
 val_T1'_inh = val_T'_inh  * val_F;
 val_T1'_syn = T1'(val_T1'_inh);
    val_T'_syn = val_T1'_syn ;
    return  val_T'_syn; } else
     { return val_T'_inh;}
```

```
val F( ){                  ③
val   val_F;
val_F = find(digit.
             lexval);
return val_F;}
```

سوال ۲_ رویکرد bottom up

به ازای actionهایی که attr های inherited درسمت راست production دارند یک مارکر تعریف میکنیم و این actionها به actionهای مارکر منتقل میشوند.

$T \rightarrow F M T' \{ T.val = T'.syn \}$

$T' \rightarrow * F N T_1' \{ T'.syn = T_1'.syn \}$

$T' \rightarrow \varepsilon \{ T'.syn = T'.inh \}$

$F \rightarrow digit \{ F.val = digit.lexval \}$

$M \rightarrow \varepsilon \{ T'.inh = F.val \}$

$N \rightarrow \varepsilon \{ T_1'.inh = T'.inh \times F.val \}$

S → L.L | L
L → LB | B
B → 0 | 1

جواب ۵

isleft یه attr
inherited است
که نشان میدهد آیا نود سمت
چپ dot قرار دارد یا نه

len ← طول رشته بیت ها
باشد

production	قوانین

S → L1.L2	L1.isleft = true
	L2.isLeft = false
	S.val = L1.val + L2.val

S → L	L.isleft = true
	S.val = L.val

L → L1B	L1.isleft = L.isleft
	L.len = L1.len + 1
	L.val = L.isleft ?
	L1.val * 2 + B.val :
	L1.val + B.val * ۲ ^ (-L.len)

L → B	L.len = 1
	L.val = L.isleft ?
	B.val : B.val/2

B → 0	B.val = 0
B → 1	B.val = 1

سوال ۳

سوال ۴ نوشتن یک L_attributed SDD

ⓐ S → if (C) S1 else S2

* کدهای C و S1 و S2 را از قبل داریم

```
L1 = new()
C.false = L1
S1.next = S.next
S.code = C.code || S1.code || label || L1 || S2.code
```

ⓑ S → do S1 while (C)

```
L1 = new()
C.true = L1
S.code = label || L1 || S1.code || C.code
```

ⓒ S → '۲' L 'y' ; L → LS | ε

سوال 5

S → if (B) S1

B → B1 || B2

B → B1 && B2

B → true

B → false

① S → if(B) M S1 { backpatch (B.truelist, M.instr);
S.nextlist = merge (B.falselist , S1.nextlist); }

② B → B1 || M B2 { backpatch (B1.falselist, M.instr);
B.truelist = merge (B1.truelist , B2.truelist);
B.falselist = B2.falselist ; }

③ B → B1 && M B2
{ backpatch (B1.truelist, M.instr);
B.truelist = B2.truelist;
B.falselist = merge(B1.falselist, B2.falselist); }

④ B → true { B.truelist = makelist(nextinstr) ;
gen('goto _'); }

⑤ B → false { B.falselist = makelist(nextinstr);
gen('goto _'); }

⑥ M → ε { M.instr = nextinstr; }

برای ② اگر B1 ، true باشد ← B هم true است
اگر B1 ، false باشد ← باید B2 هم چک شود

ادامه ی

S → ER

R → *E {print("*");} R | ε

E → F+E{ print("+");} | F

F → (S) | id {print(id.valve);}

رشته ورودی :

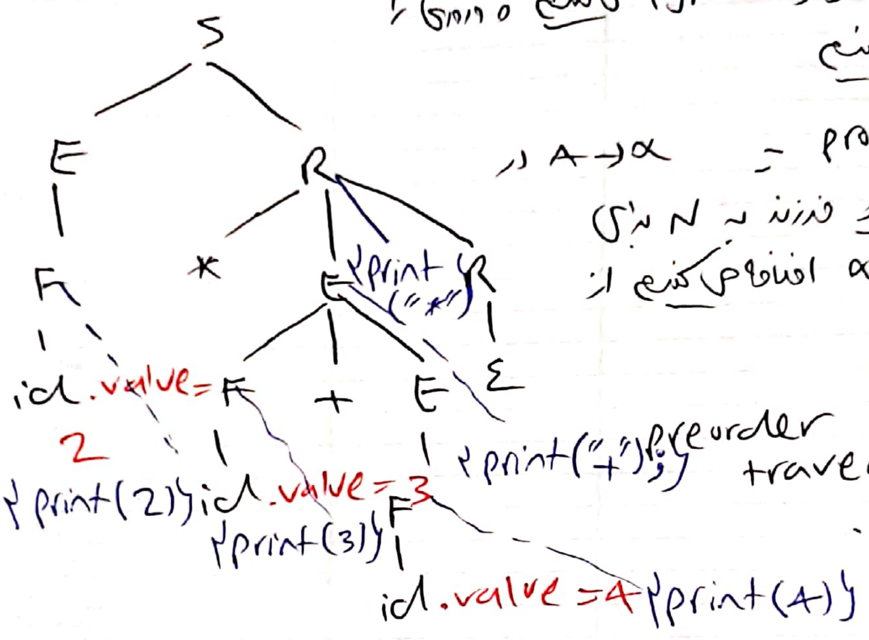
2*3+4

مراحل :

① action ها را ignore می‌کنیم و ورودی را پارس می‌کنیم

② برای production ها = A → α در نود N یک فرزند به N برای action های α اضافه می‌کنیم از چپ به راست

③ یک preorder traversal انجام می‌دهیم.



2*3+4

جواب 6

سوال

E → E+E {print("+")}

E → E*E {print "•"}

E → id

E → (E)

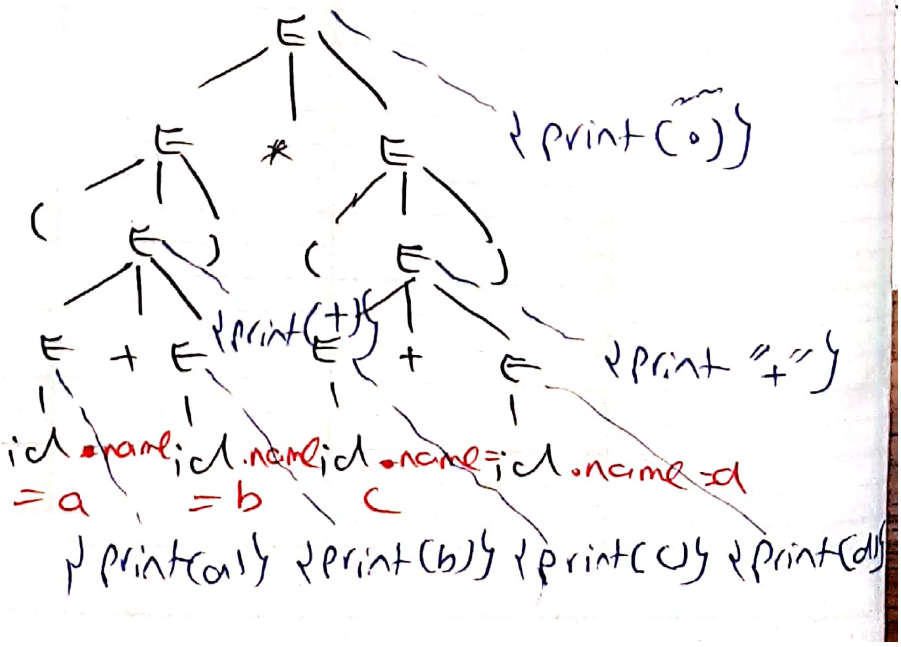
exp

(a+b)*(c+d)

خروجی که تولید traverse کند

دهید :

ab+cd+•


E
E
*
E
{print(•)}
(
E
)
(
E
)
E
+
E
{print(+)}
E
+
E
{print "+"}
id.name=a
id.name=b
id.name=c
id.name=d
{print(a)}
{print(b)}
{print(c)}
{print(d)}