Compiler Design

Today's Class Topics

7:30

- Syntax directed translation (SDT)
- Types of attributes & SDT
- 3 Address code
- Implementation of 3 address code
  - Quadrupole
  - Triple
  - Indirect Triple

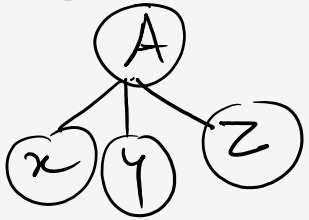
Compiler Design

Types of Attributes:

- Synthesized
- Inherited Attribute

Synthesized Attribute: value depends on children, it is an Attribute of a non terminal which on the Left Hand side (LHS) of the Array;

$A \rightarrow xyz$



$A.s = f(x.s, y.s, z.s)$

↑ Synthesized Attribute

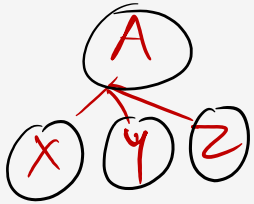
$A.val = x.val + y.val \times z.val$

Compiler Design

Inherited Attribute: value depends on parent or siblings

It is an Attribute of a non terminal which is on the RHS of the Array.

$A \rightarrow XYZ$



$Y.e = f(A.e | X.e | Z.e)$

Compiler Design

$E \rightarrow E + T \quad \{ E.val = E_1.val + T.val \}$

Types of SDT:

- S-Attributed SDT
- L-Attributed SDT

S-Attributed SDT: A SDT that uses only synthesized Attributes is called S-Attributed SDT

1) Attributes are evaluated in Bottom up approach as the parent value depends on its children
2) The semantic Action can be placed on the Rightmost end of RHS of the production.

$A \rightarrow xyz \{ \quad \}$

Ex:

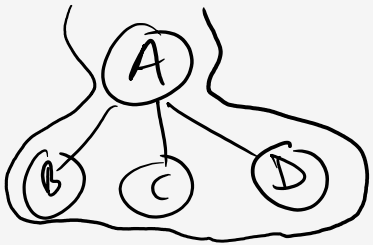
$A \rightarrow BCD$

~~$A.s = B.s$~~

$\{ A.val = B.val \}$

$\{ A.val = C.val \}$

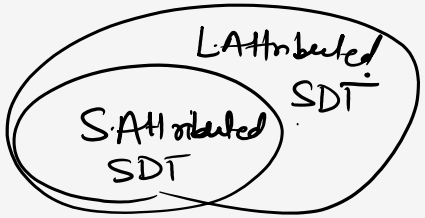
$\{ A.val = D.val \}$



Compiler Design

L-Attributed SDT: A SDT that uses both Synthesized and inherited attributes is called as L-Attributed SDT but each inherited attribute is restricted to Inherit from parent or left Sibling only

$A \rightarrow BCD$ | $C.val = A.val$ ✓ ; $C.val = B.val$ ✓



1) Attributes are evaluated in depth first and Left to Right

2) The semantic Action can be placed any where on the RHS of the production $A \rightarrow xyz$ ⇒

$A \rightarrow \{\}xyz$
$A \rightarrow x\{\}yz$
$A \rightarrow xy\{\}z$
$A \rightarrow xyz\{\}$

Compiler Design

Ex:- The following SDT is

$A \rightarrow BA_1 \quad \{A_1.x = B.y\}$

$B \rightarrow B_1 C \quad \{C.x = B.x - B_1.x\}$

a) S-Attributed SDT

b) L-Attributed SDT

c) Both S & L-Attributed

d) None

Ex:- $A \rightarrow SBX \quad \{X.i = S.i + A.i ; \; B.i = S.i\}$

$B \rightarrow ACD \quad \{C.i = A.i + B.i - D.i\}$ not L-Attributed

a) S-Attributed

b) L-Attributed

c) Both

d) None

Compiler Design

Ex:- $A \rightarrow BC \quad \{B.i = A.i ; C.i = B.i ; A.i = C.i\}$

Ex:- $A \rightarrow QR \{R.i = A.i ; \quad Q.i = R.i ; A.i = Q.i\}$

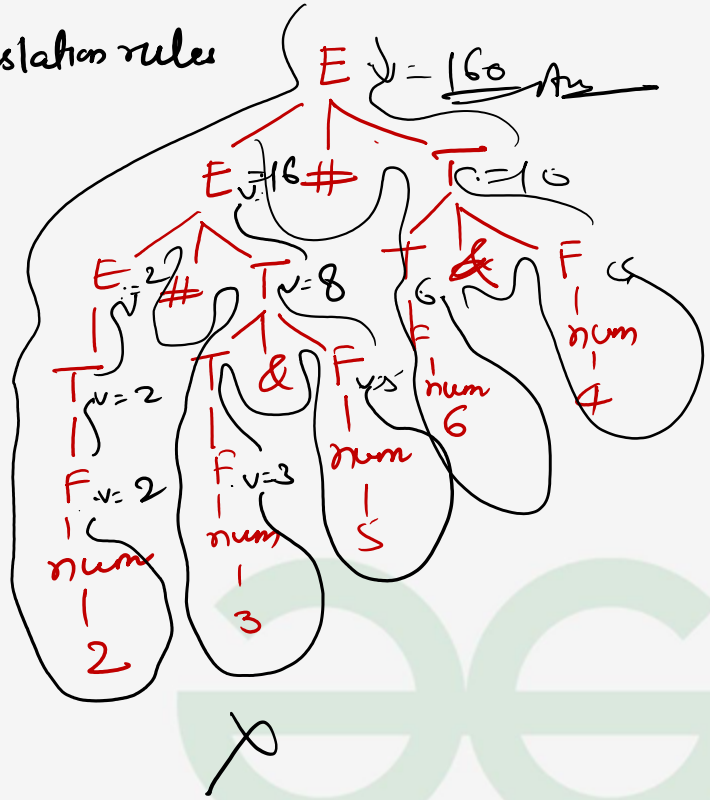
Ex:- $A \rightarrow P\,Q \{A.i = P.i , A.i = Q.i\}$

Compiler Design

Question:- Consider the grammar with the following translation rules

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

Compute Evaluate for the Root of the Parse tree for the expression 2#3 &5 #6 &4



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

=> 16*10=160

Compiler Design

GATE-2019 — 2024

⇒ Question:-

Production	Semantic Rules
$E \to E \# T$	$E.val = E_1.val * T.val$
$E \to T$	$E.val = T.val$
$T \to T \& F$	$T.val =$? $T.val = T.val - F.val$
$T \to F$	$T.val = F.val$
$F \to digit$	$F.val = digit.val$

⇒ 1) If the expression 8#12&4#16&12#4&2 is evaluated to 512 Then which of the following is Correct replacement of Blank

a) T.val = T.val * F.val c) T.val = T.val - F.val

b) T.val = T.val + F.val d) None

⇒ 2) 10#8&6#9&4#5&2 ?

$10 * (8-6) * (9-4) * (5-2)$

$10 * 2 * 5 * 3 = 300$

$- \to *$

$* \to *$

$\# \to *$

$8 * (12-4) * (16-12) * (4-2)$

$8 * 8 * 4 * 2$

$2^3 * 2^3 * 2^2 * 2^1$

$2^9 = 512$

Compiler Design

SDD to create 3-Address Code:- $x = a + b * c$

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

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

$E \rightarrow T \quad \{ E.place = T.place \}$

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

$T \rightarrow F \quad \{ T.place = F.place \}$

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

Compiler Design

2019 – 2024

H.W

Question: Consider the following SDT

$E \rightarrow E_1 * T$ { E.val = E_1.val + T.val }

/T { E.val = T.val + 1 }

$T \rightarrow T_1 + F$ { T.val = T_1.val * F.val }

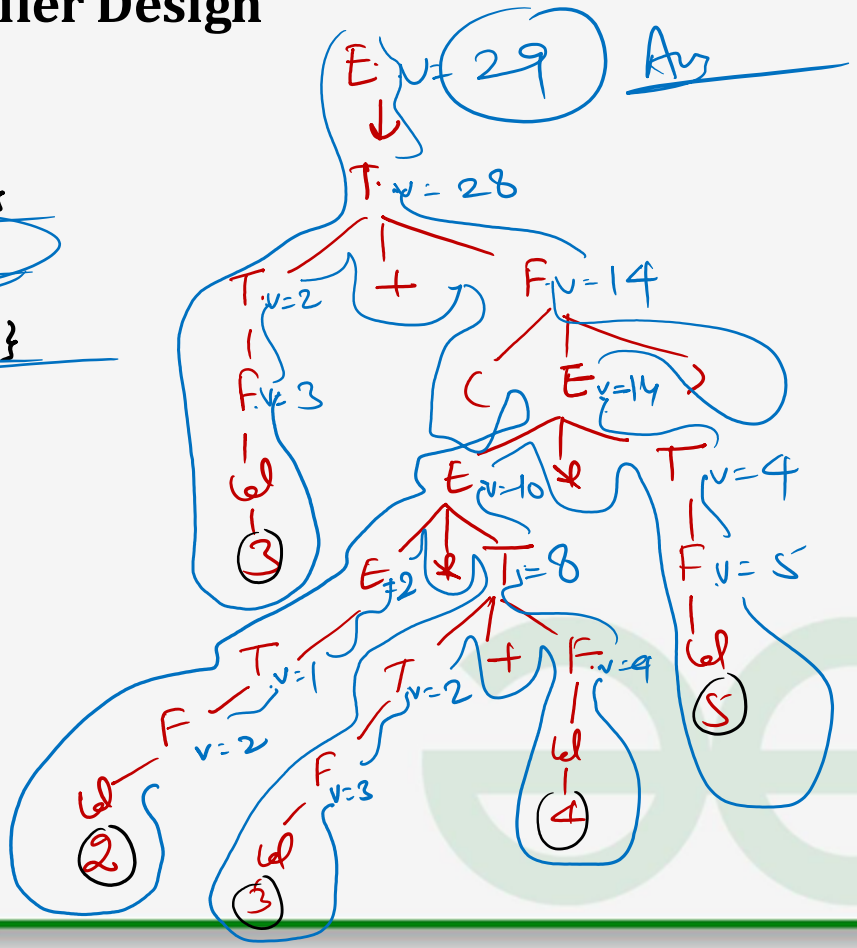
/F { T.val = F.val − 1 }

$F \rightarrow (E)$ { F.val = E.val }

/id { F.val = id.val }

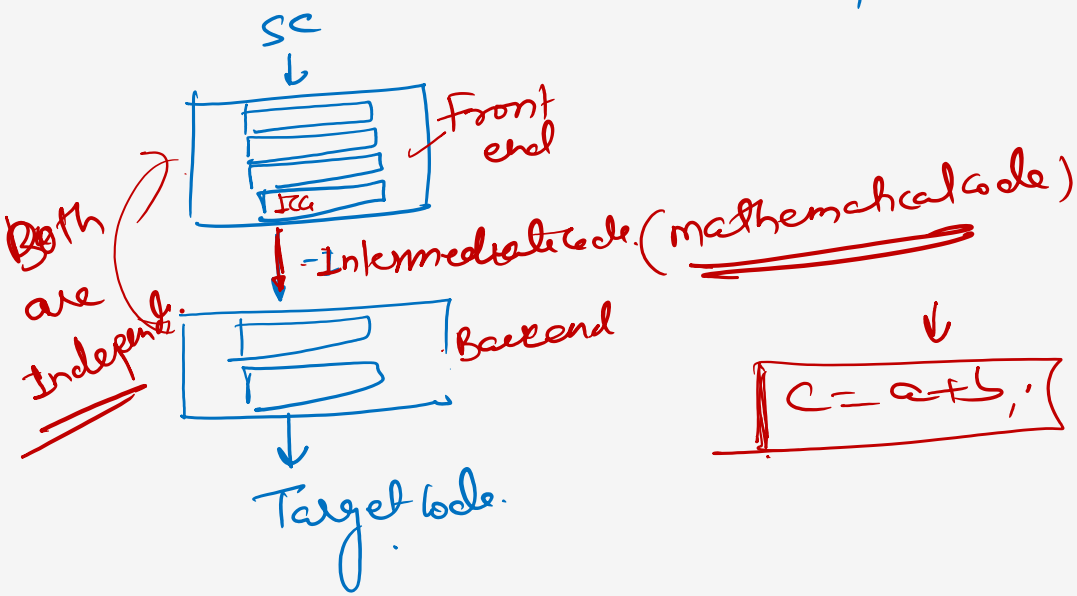
if input = 3 + (2 * 3 + 4 * 5)

Then o/p = ?



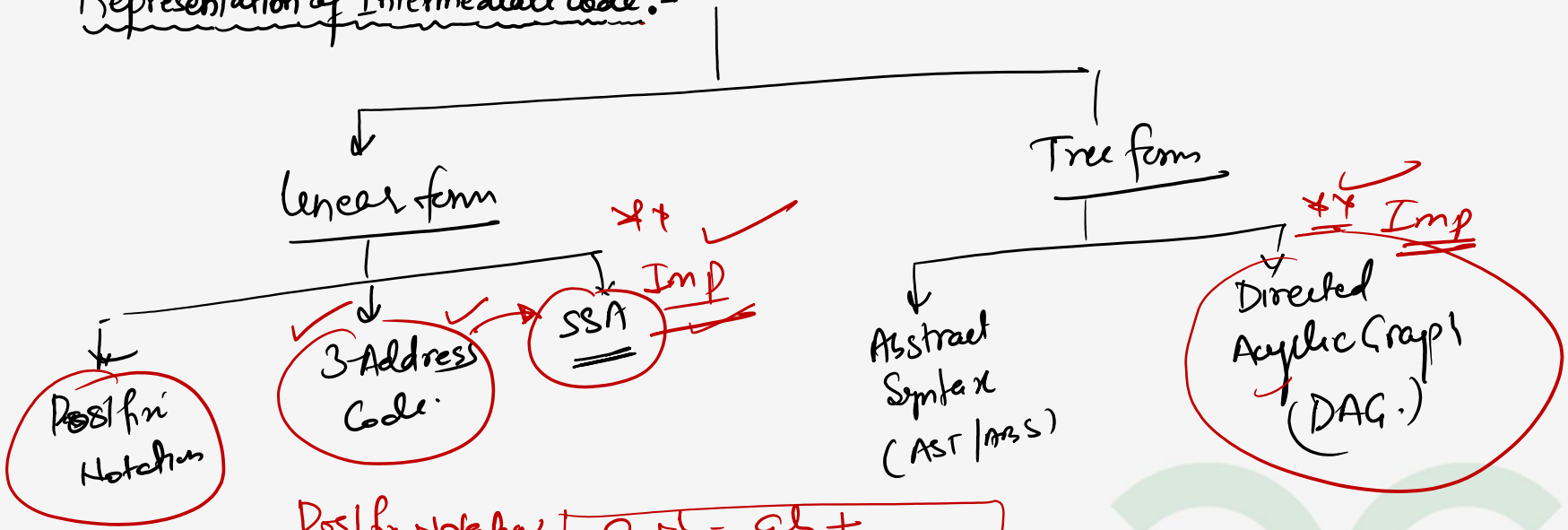
Compiler Design

Intermediate Code generation:- portability can enhance by intermediate code.



Compiler Design

Representation of Intermediate Code :-



Postfix Notation

a+b = ab+

b*c = bc*

a+b*c = abc*+

a*b-c => ab*c-

Compiler Design

3-Address Code :-

Compiler Design

Compiler Design

Question:- Write a 3-Address code for the following code

1) $x = ((a+b) - (c*d)) * (a+b-c)$

2) $x = a+b*c-d$, where x, a are real and b, c, d are integers

Compiler Design

3) if (a>b)
    a=a+1
else
    b=b-1

4) for(i=0; i<=n; i++)
{
    x=x+i;
    y=y-i;
}

Compiler Design

5) if((a>b) && (c>d))

```
    x = x+1
else
    y = y+1
```

Backpatching:- Leaving the labels and filling them after is called as Backpatching

Compiler Design

6)
```
for(i=1; i≤n; i++)
{ x = a+b*c;
}
```

Compiler Design

7) Switch (i)

```
{
Case 1:
    x1 = a1+b1*c1;
        break;

Case 2:
    x2 = a2+b2*c2;
        break;

default:
    x3 = a3+b3*c3;
        break;
}
```

Thank You !