CS & IT ENGINEERING

Compiler Design

Syntax Directed Translation

Lecture No. 1

By- DEVA Sir

Review

SDT ?

Attributes ?

Definitions ?

compiler phases

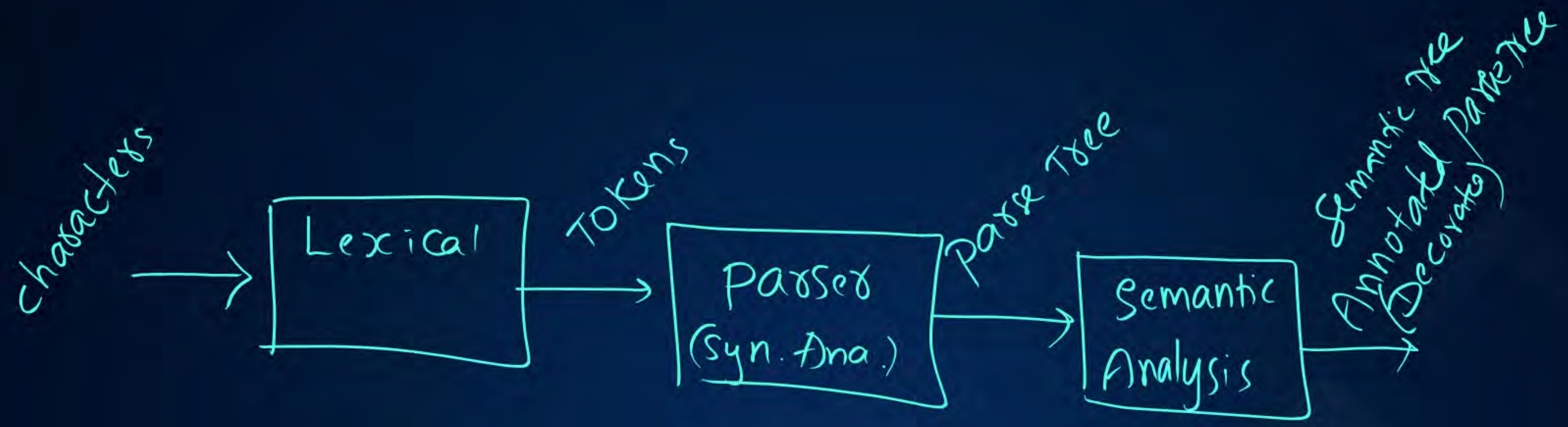
Language Translation

Lexical Analysis

Symbol Table,
Tokens
Lexical Errors
Functionality

Syntax Analysis :

Ambiguous & unamb CFG
Elimination of Left Recursion
Left factoring, FIRST & FOLLOW SET
LL(1) CFG, LL(1) Table, ~~LMD Algo~~
LR CFG, LR Tables, ~~Reverse of RMD Algo~~
Operator precedence


characters
Lexical
Tokens
Parser
(Syn. Ana.)
Parse Tree
Semantic
Analysis
Semantic Tree
Annotated
(Decorated) parse tree

Syntax Directed Translation :

- It can be used anywhere in Translation
- It can be used to perform semantic Analysis
- It can be used to produce Intermediate code
- It can be used to evaluate expressions
- It can be used to convert one form of exp to other form (infix/pre/postfix) (in/pre/post)
- It can be used to translate one number system to other number system

SDT Vs Semantic Analysis

- all functionalities of semantic analysis
- any other functionality

- Type checking
- Declaration of variables
- Function compatibility

①

```
1 void main( )
2 {
3     int x=10;
4     y=x;
5 }
```

⇒ Semantic error
y is not declared

y → type entry is blank in symbol table

x → integer

		line	type
x	id1	3,4	integer
y	id2	4	

(2)

```
void main()
{
    f();
}
```

⇒ semantic error

f is not declared

3

```
#include<stdio.h>
void main()
{
    f();
}
```

⟹ Linker Error

f is not resolved

④

```
void main()
{
    char ch='A';
    float f=20.34f;

    ch=f;
}
```

ch → 1 byte

f → 4 bytes

⇒ Semantic Error

But some compilers can also do implicit conversions

(5)

```
void main()
{
    char ch;
    float f = 20.34f;

    ch = (char)f;
}
```

⇒ No error

(6)

```
void f()
{
    int x;
    x=10;
}

void main()
{
    f(2);
}
```

```
void f(void);
```

Semantic Error

(Function compatibility issue)

- Function name
- Return type
- no. of parameters
- each parameter type

(7)

```
void main()
{
    Integer x;
}
```

⇒ Semantic Error

(8)

```
typedef int Integer;
void main()
{
    Integer x;
}
```

⇒ No error

Write ⟹ character ⟹ Lexical

See ⟹ structure ⟹ Syntax

think ⟹ Semantic

2+3

23

Words
Syntax } eyes

Semantic } Mind (Interpretation)

Syntax Directed Translation

Syntax + Translation

(Action)

(program)

Semantic

o/p

Tree generate

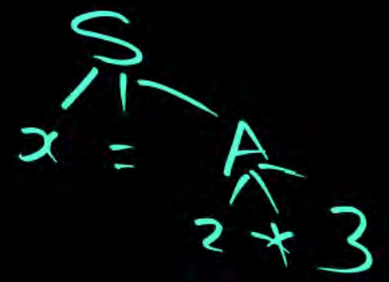
$\boxed{x = 2*3;}$

Syntax

Translation → $x = 2 * 3;$

$\boxed{2.type == 3.type == x.type}$

Translation →



Translation → $x = 6;$

SDT :

Syntax + Translation

$+ \rightarrow$ Addition

$+ \rightarrow$ Concatenation

CFG + Translation for every Rule

$E \rightarrow E_1 + E_2 \; \{ E.x = E_1.x * E_2.x \}$

$E \rightarrow E_1 * E_2 \; \{ E.x = E_1.x - E_2.x \}$

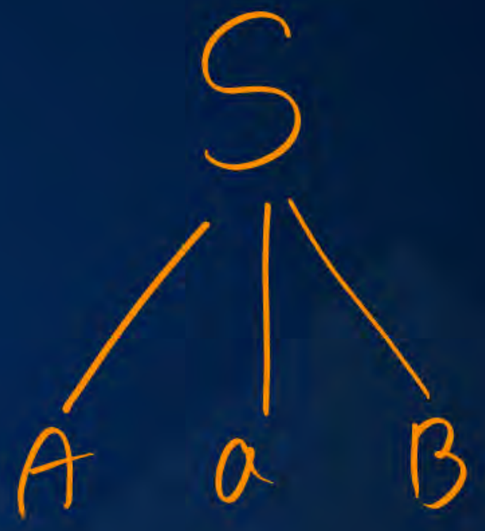
$E \rightarrow id \; \{ E.x = id \}$

Rule	Translation
$E \rightarrow id$	$\{ E.x = id.val \}$
⇓	⇓
Syntax production	Translation
	$\{ \quad \}$
	x, val ⇒ Attributes

⇓

Definition ?

$$S \longrightarrow AaB$$



S is parent for A, a, B

A is child of S, Left sibling of a and B

a is child of S, Right sibling of A, Left sibling of B

B is child of S, Right sibling of A & a.

Parent

Child

Sibling → Left, Right

Attributes

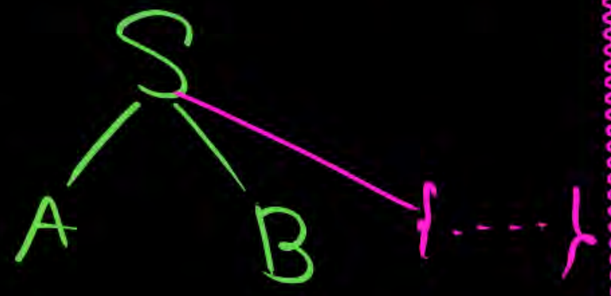
1. Inherited Attribute
2. Synthesited Attribute

① Inherited Attribute

Attribute value at any node $\boxed{\text{computed}}$ based on parent or/and siblings.

$$S \rightarrow A\boxed{B} \quad \{B.x = f(S.y \text{ or } A.z)\}$$

x is inherited



② Synthesized Attribute

Attribute value computed based on children.

$$\boxed{S} \rightarrow AB \quad \{S.x = f(A.y \text{ or } B.z)\}$$

Inherited Attribute

LHS $\longrightarrow$ RHS

Computation happens on RHS side symbols

Synthesized Attribute

LHS $\longrightarrow$ RHS

Computation happens on LHS side symbol

1

1. $D \rightarrow T\ L$ $\{L.type = T.type\}$

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

3. $T \rightarrow float$ $\{T.type = float\}$

4. $L \rightarrow L_1, id$ $\{L_1.type = L.type;$ ~~AddType(id.entry, L.type)~~ $\}$

5. $L \rightarrow id$ $\{$~~AddType(id.entry, L.type)~~$\}$

type is _____

not inherited

not synthesized

type entry

In production 3, type is synthesized Attribute.

In SDT, type is neither Inherited nor synthesized Attribute.

L value

$$\boxed{a} = b + c;$$

Alloc	Use	computation
a	b	a
b	c	
c		

a is defined

a is computed

a is assigned

(2)

$\boxed{E} \rightarrow E_1 + E_2 \quad \{ E_1.x = E_2.y \; ; \; E.\overset{synt}{y} = E_1.x + 2 \}$

$\boxed{E} \rightarrow E_1 * \boxed{E_2} \quad \{ \underbrace{E_2.\overset{inherite}{y} = E_1.x + 3}_{} \; ; \; \underbrace{E.\overset{synt}{y} = E_2.y * 2}_{} \}$

$E \rightarrow id \quad \{ E.x = id.val + 1 \; ; \; E.\overset{synte}{y} = id.val * 3 \}$

Attributes

x

y

Val

x is neither inherited nor synthesized Attribute

y is neither inherited nor synthesized Attribute

(3)

$E \rightarrow \boxed{A}B \quad \{\boxed{A.x} = B.y \; ; \; \cancel{\text{print}(A.x)}\}$

$\boxed{A} \rightarrow a \quad \{\boxed{A.x} = a.num \; ; \; \cancel{\text{print}(a.num)}\}$

$\boxed{B} \rightarrow b \quad \{\boxed{B.y} = 100\}$

x
y
num

x is neither inherited nor synthesized

y is your choice

(4)

$E \longrightarrow \boxed{A}B \quad \{\boxed{A.x} = B.y \; ; \; \overbrace{E.y = A.x}^{y \text{ is synthesized}} \}$?

$\boxed{A} \longrightarrow a \quad \{\boxed{A.x} = a.num \; ; \; \cancel{Print(a.num)}\}$

$\boxed{B} \longrightarrow b \quad \{\boxed{B.y} = 100\}$

Independent Computation

x is <u>neither inherited nor synthesized</u>

y is <u>synthesized</u>

$E \rightarrow E+T \mid a$

$T \rightarrow F*T \mid b$

$F \rightarrow c$

$\underbrace{+ \text{ and } *}$

	+	*
+		<
*	>	

Summary

- SDT
- Attributes
- Next: Definitions of SDT
  - L-attributed SDT
  - S-attributed SDT


THANK YOU
GW
SOLDIERS !