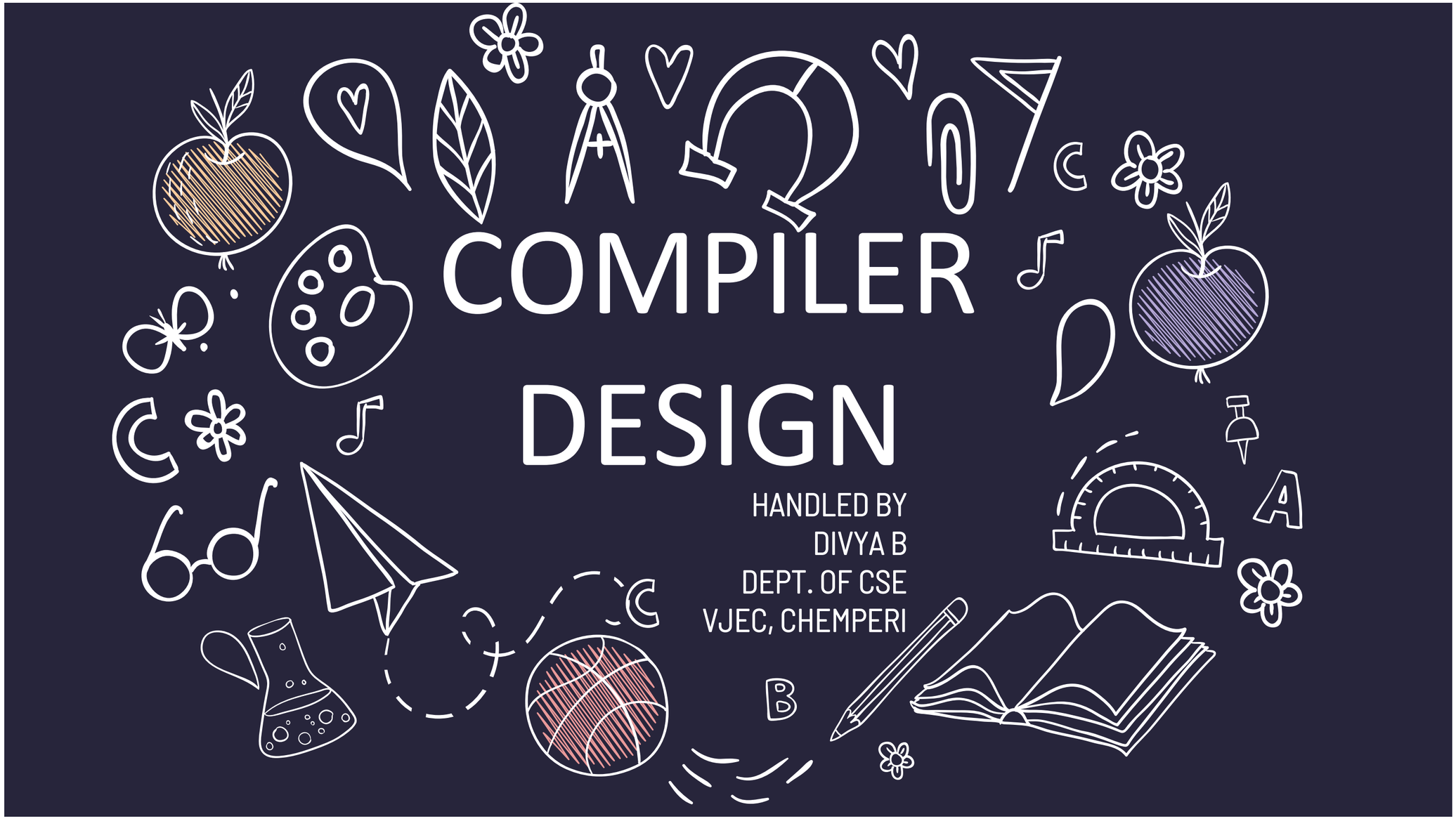

COMPILER
DESIGN
HANDLED BY
DIVYA B
DEPT. OF CSE
VJEC, CHEMPERI

SEMANTIC ANALYSIS

- In the semantic analysis phase, semantic information is added to the parse tree and certain checks are performed based on this information.
- It logically follows the parsing phase, in which the parse tree is generated, and logically precedes the code generation phase, in which (intermediate/target) code is generated.
- Examples of semantic information that is added and checked is type checking and the binding of variables and function names to their definitions (object binding).

SEMANTIC ANALYSIS

- Sometimes also some early code optimization is done in this phase.
- For this phase the compiler usually maintains symbol tables in which it stores what each symbol (variable names, function names, etc.) refers to.

SEMANTIC ANALYSIS

- Following things are done during semantic analysis
  1. **Disambiguate overloaded operators**
  2. **Type checking**
  3. **Uniqueness checking**
  4. **Type coercion**
  5. **Name Checks**

SEMANTIC ANALYSIS

1. **Disambiguate Overloaded operators**
   - ✓ If an operator is overloaded, the meaning of that particular operator must be specified properly because the next phase is code generation.

SEMANTIC ANALYSIS

2. Type checking

- ✓ The process of verifying and enforcing the constraints of types is called type checking.
- ✓ This may occur either at compile-time (static check) or run-time (dynamic check).
- ✓ Static type checking is a primary task of the semantic analysis carried out by a compiler.
- ✓ If type rules are enforced strongly (that is, generally allowing only those automatic type conversions which do not lose information), the process is called strongly typed, if not, weakly typed.

SEMANTIC ANALYSIS

3. Uniqueness checking

- ✓ Checking whether a variable name is unique or not, in its scope.

4. Type coercion (Implicit conversion)

- ✓ This can be done dynamically as well as statically.

5. Name checks

- ✓ Check whether any variable has a name which is not allowed. E.g. Name is same as a keyword (Ex. int in java).

SEMANTIC ERRORS

1. Type mismatch
2. Undeclared variable
3. Reserved keywords misuse
4. Multiple declaration of variable in a scope
5. Accessing an out of scope variable
6. Actual and formal parameter mismatch

SEMANTIC ANALYSIS

- Evaluation of semantic rules may
  - ✓ Generate code
  - ✓ Insert information into the symbol table
  - ✓ Perform semantic check
  - ✓ Issue error messages etc.

- There are two ways to represent the semantic rules associated with grammar symbols
  - ✓Syntax-Directed Definitions (SDD)
  - ✓Syntax-Directed Translation Schemes (SDT)

SYNTAX DIRECTED DEFINITIONS

- Syntax Directed Definitions are a generalization of context-free grammars in which
  - ✓ Grammar symbols have an associated set of attributes.
  - ✓ Productions are associated with semantic rules for computing the values of attributes.
  - ✓ Such formalism generates **annotated parse trees** where each node of the tree is a record with a field for each attribute (e.g., **X.a** indicates the attribute **a** of the grammar symbol **X**).

SYNTAX DIRECTED DEFINITIONS

- The parse tree containing the values of attributes at each node for given input string is called **annotated or decorated parse tree**.

ATTRIBUTE GRAMMAR

- Attributes are properties associated with grammar symbols.
- Attributes can be numbers, strings, memory locations, datatypes, etc.
- Attribute grammar is a special form of Context-Free Grammar where some additional information (attributes) are appended to one or more of its non-terminals in order to provide context-sensitive information.

ATTRIBUTE GRAMMAR

- Attribute grammar is a medium to provide semantics to the context-free grammar and it can help specify the syntax and semantics of a programming language.
- Attribute grammar (when viewed as a parse tree) can pass values or information among the nodes of a tree.

ATTRIBUTE GRAMMAR

- $E \rightarrow E_1 + T$ { E.value = E_1.value + T.value }
  - ✓ The right part of the CFG contains the semantic rules that specify how the grammar should be interpreted.
  - ✓ Here, the values of non-terminals E_1 and T are added together and the result is copied to the non-terminal E.
- Semantic attributes may be assigned to their values from their domain at the time of parsing and evaluated at the time of assignment or conditions.

ATTRIBUTE GRAMMAR

- Based on the way the attributes get their values, they can be broadly divided into two categories
  - ✓ Synthesized attributes
  - ✓ Inherited attributes

ATTRIBUTE GRAMMAR

- Synthesized attributes
  - ✓ These are those attributes which get their values from their children nodes i.e. value of synthesized attribute at node is computed from the values of attributes at children nodes in a parse tree.

ATTRIBUTE GRAMMAR

- Synthesized attributes
  - ✓ E.g. S → ABC
  - ✓ S.a = A.a, B.a, C.a
  - ✓ If S is taking values from its child nodes (A,B,C), then it is said to be a synthesized attribute, as the values of ABC are synthesized to S.

Computation of Synthesized Attributes

1. Write the SDD using appropriate semantic rules for each production in given grammar.
2. The annotated parse tree is generated and attribute values are computed in bottom up manner.
3. The value obtained at root node is the final output.

Computation of Synthesized Attributes

- Consider the following grammar

$S \rightarrow E$

$E \rightarrow E_1 + T$

$E \rightarrow T$

$T \rightarrow T_1 * F$

$T \rightarrow F$

$F \rightarrow digit$

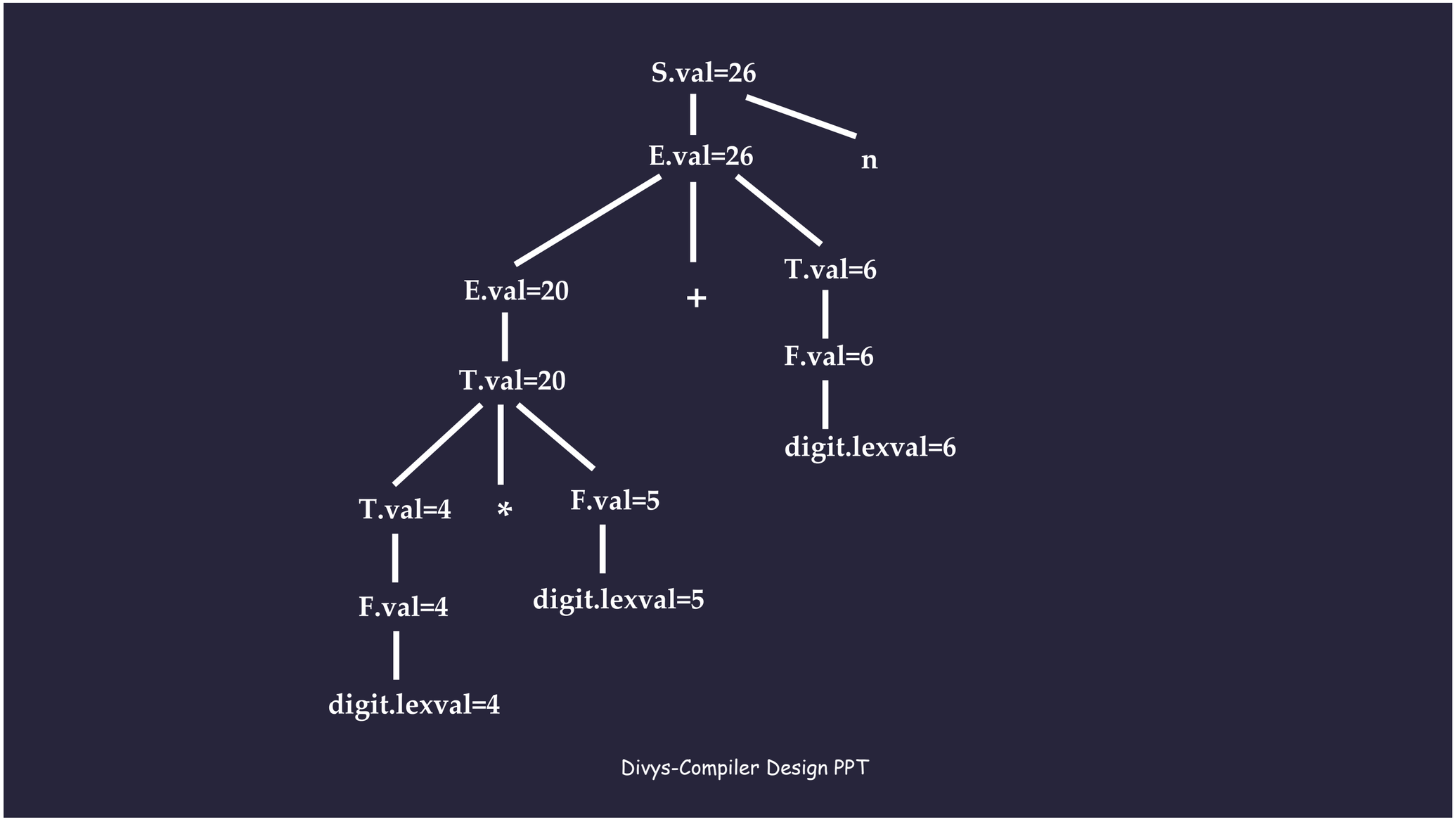
Computation of Synthesized Attributes

- The SDD for the grammar can be written as follows.
- It evaluates expressions terminated by an end marker n.

Production	Semantic Rules
$S \rightarrow E\ n$	Print (E.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
$F \rightarrow digit$	F.val = digit.lexval

Computation of Synthesized Attributes

- Let us assume an input string **4 * 5 + 6 n** for computing synthesized attributes.
- Compute the annotated parse tree for the above string.


S.val=26
E.val=26
n
E.val=20
+
T.val=6
T.val=20
F.val=6
digit.lexval=6
T.val=4
*
F.val=5
F.val=4
digit.lexval=5
digit.lexval=4
Divys-Compiler Design PPT

ATTRIBUTE GRAMMAR

- Inherited attributes
  - ✓ These are the attributes which inherit their values from their parent or sibling nodes.

ATTRIBUTE GRAMMAR

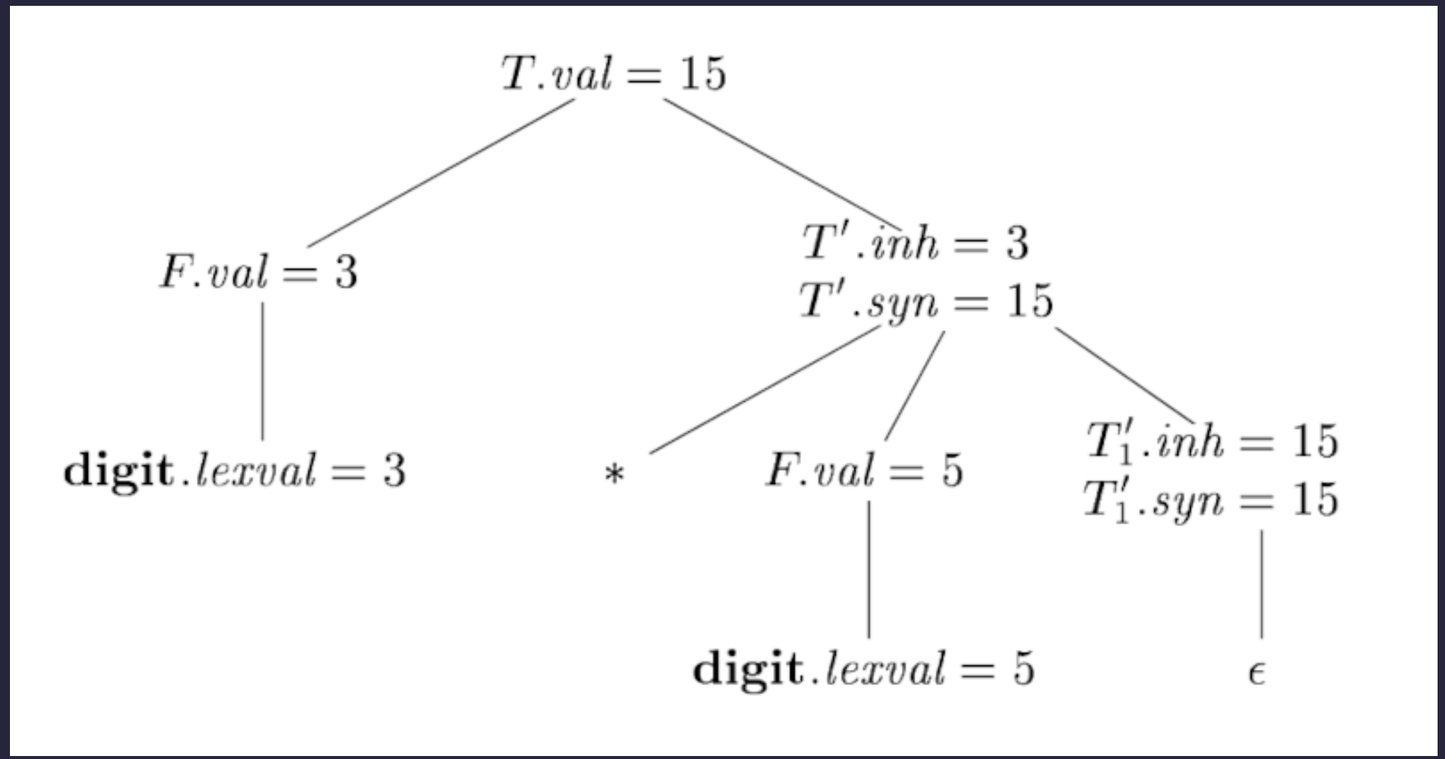
- Computation of inherited attributes
  1. Construct the SDD using semantic actions.
  2. The annotated parse tree is generated and attribute values are computed in top down manner.

Annotated Parse Trees

Production	Semantic Rules
$T \rightarrow FT'$	$T'.inh = F.val$ $T.val = T'.syn$
$T' \rightarrow *FT_1'$	$T_1'.inh = T'.inh \times F.val$ $T'.syn = T_1'.syn$
$T' \rightarrow \varepsilon$	$T_1'.syn = T'.inh$
$F \rightarrow digit$	$F.val = digit.lexval$

Annotated Parse Trees

Let us assume an input string **3 x 5**



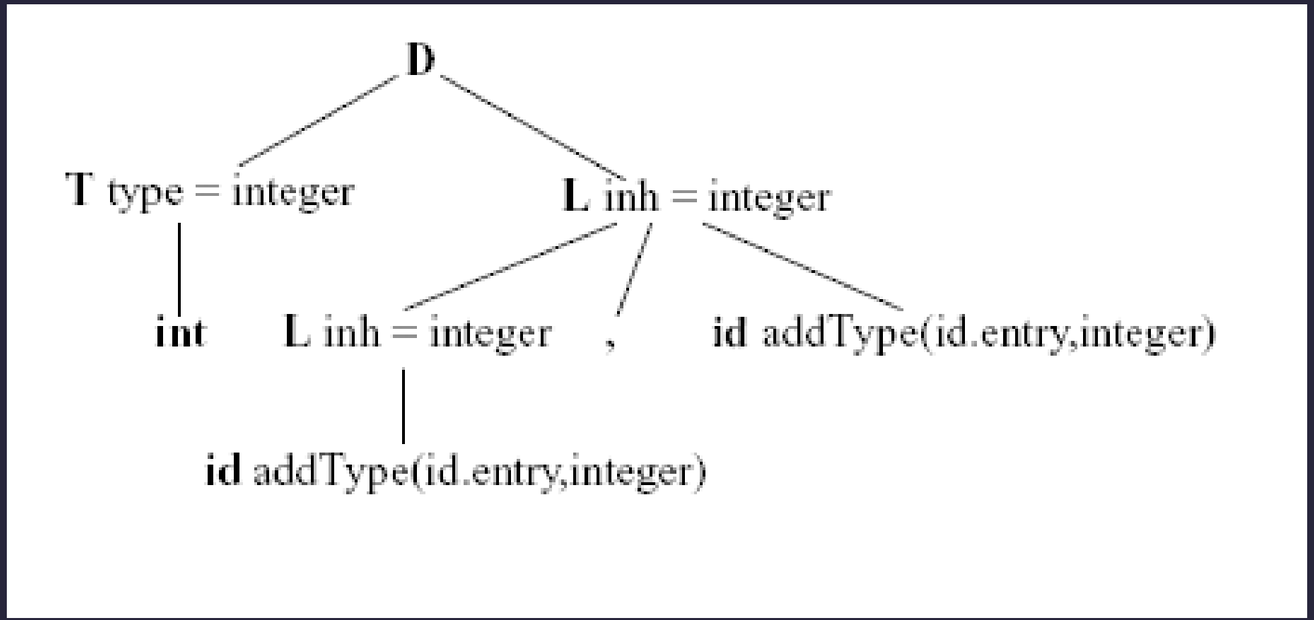
Annotated Parse trees

- Using the syntax-directed definition (SDD), construct the annotated parse tree for the input expression: "int *a, b*".

D -> T L	L.inh = T.type
T -> int	T.type = integer
T -> float	T.type = float
L -> L1, id	L1.inh = L.inh
	addType(id.entry,L.inh)
L -> id	addType(id.entry,L.inh)

Annotated Parse trees

- Using the syntax-directed definition (SDD), construct the annotated parse tree for the input expression: "int *a, b*".



IMPLEMENTING SYNTAX DIRECTED DEFINITIONS

- Dependency Graphs
  - Implementing a Syntax Directed Definition consists primarily in finding an order for the evaluation of attributes.
    - ✓ Each attribute value must be available when a computation is performed.
    - ✓ Dependency graphs are the most general technique used to evaluate syntax directed definitions with both synthesized and inherited attributes.
  - Annotated parse tree shows the values of attributes, dependency graph helps to determine how those values are computed.

IMPLEMENTING SYNTAX DIRECTED DEFINITIONS

- Dependency Graphs
  - The interdependencies among the attributes of the various nodes of a parse tree can be depicted by a directed graph called a dependency graph.
  - For each parse tree node, say a node labelled by grammar symbol X, the dependency graph has a node for each attribute associated with X.

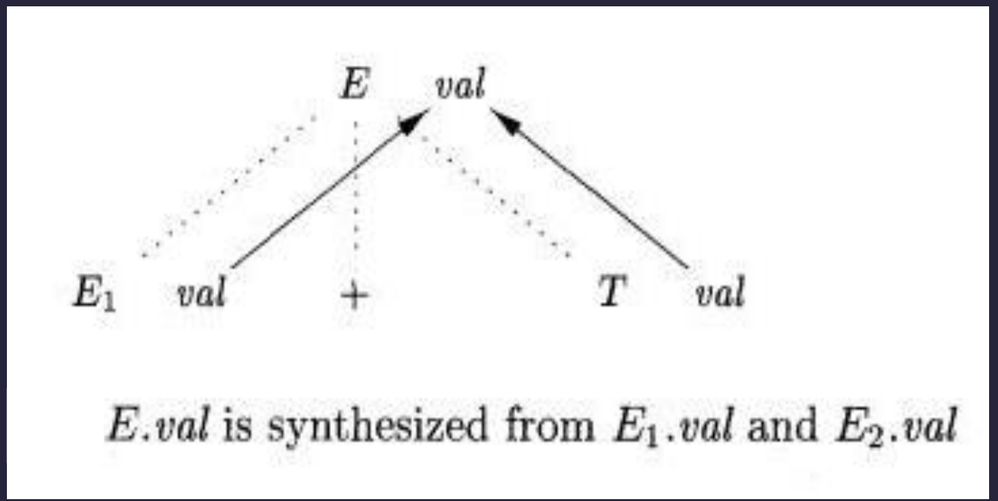
IMPLEMENTING SYNTAX DIRECTED DEFINITIONS

- Dependency Graphs
  - Suppose that a semantic rule associated with a production p defines the value of synthesized attribute A.b in terms of the value X.c then the dependency graph has an edge from X.c to A.b.

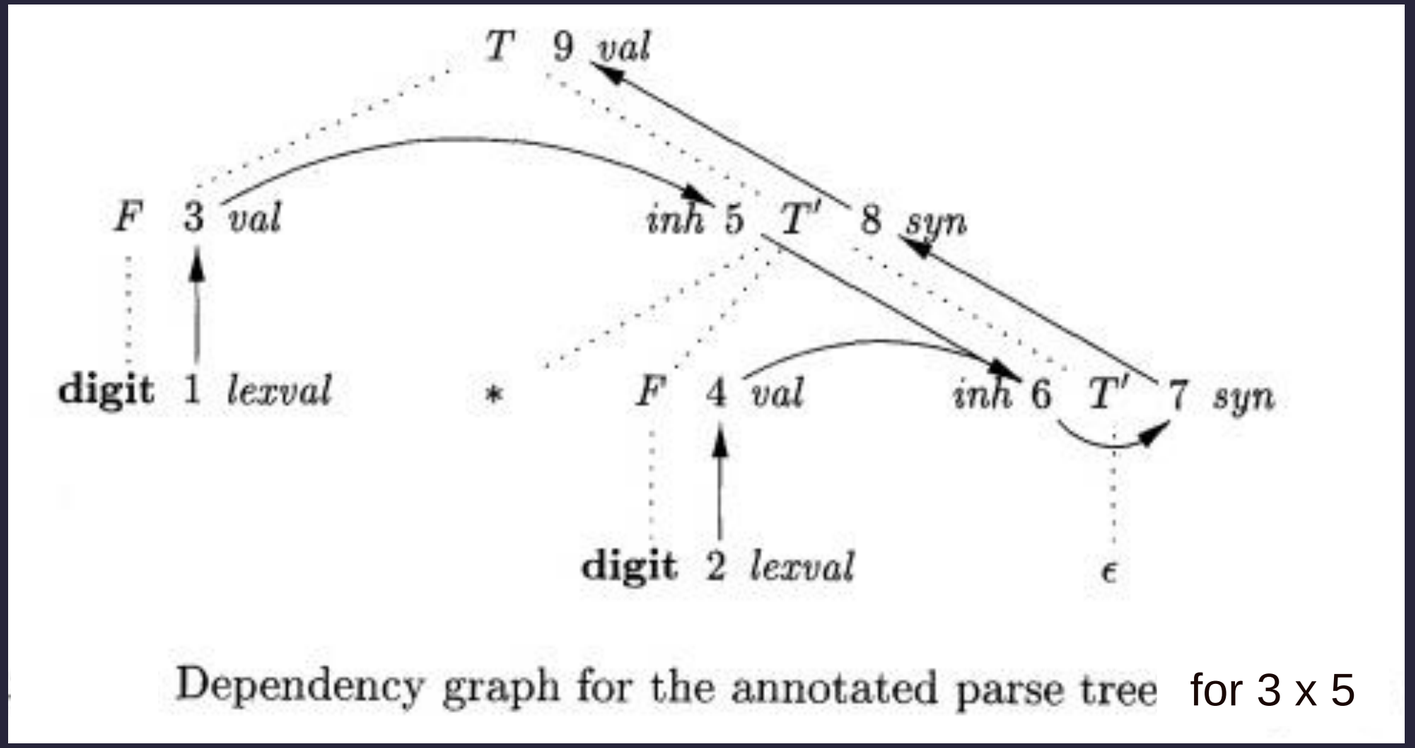
IMPLEMENTING SYNTAX DIRECTED DEFINITIONS

- Dependency Graphs
  - Suppose that a semantic rule associated with a production p defines the value of inherited attribute B.c in terms of the value X.a then the dependency graph has an edge from X.a to B.c.

IMPLEMENTING SYNTAX DIRECTED DEFINITIONS



IMPLEMENTING SYNTAX DIRECTED DEFINITIONS



Dependency graph for the annotated parse tree for 3 x 5

IMPLEMENTING SYNTAX DIRECTED DEFINITIONS

- Evaluation order of Dependency Graphs
  1. If there is an edge from node M to N, then the attribute corresponding to M must be evaluated first before evaluating N.
  2. Such an ordering embeds a directed graph into a linear order, and is called a **topological sort** of the graph.
  3. If there is any cycle in the graph ,then there is no topological sort.ie, there is no way to evaluate SDD on this parse tree.

TYPES OF SYNTAX DIRECTED DEFINITIONS

- S-attribute definitions
- L-attribute definitions

TYPES OF SYNTAX DIRECTED DEFINITIONS

- S-attribute definitions
  - A SDD is S-attributed if every attribute is synthesized.
  - When a SDD is S-attributed, we can evaluate its attributes in bottom-up order of the nodes of the parse tree.
  - S-attributed definitions can be implemented during bottom-up parsing, since a bottom-up parse corresponds to a postorder traversal.
  - Postorder corresponds exactly to the order in which an LR parser reduces a production body to its head.

TYPES OF SYNTAX DIRECTED DEFINITIONS

- S-attribute definitions

```
postorder(N)    {

for ( each child C of N, from the left ) postorder(C);

evaluate the attributes associated with node N;

}
```

TYPES OF SYNTAX DIRECTED DEFINITIONS

- L-attribute definitions
  - If an SDT uses both synthesized attributes and inherited attributes with a restriction that inherited attribute can inherit values from parent and left siblings only, it is called as L-attributed SDT.
  - E.g. A→BCD {B.a=A.a, C.a=B.a}
    C.a=D.a This is not possible.

TYPES OF SYNTAX DIRECTED DEFINITIONS

- L-attribute definitions
  - Attributes in L-attributed SDTs are evaluated by depth-first and left-to-right parsing manner.
  - Semantic actions are placed anywhere in RHS.

$$A \rightarrow \{\ \}BC$$
$$B\{\ \}C$$
$$BC\{\ \}$$

If an attribute is S attributed , it is also L attributed.

TYPES OF SYNTAX DIRECTED DEFINITIONS

PRODUCTION	SEMANTIC RULE
$T \rightarrow F\ T'$	$T'.inh = F.val$
$T' \rightarrow *\ F\ T_1'$	$T_1'.inh = T'.inh \times F.val$

This SDD is L-attributed.

TYPES OF SYNTAX DIRECTED DEFINITIONS

PRODUCTION	SEMANTIC RULES
$A \rightarrow B\ C$	$A.s = B.b;$
	$B.i = f(C.c, A.s)$

This SDD is not L-attributed because B.i takes values from C.c.

TYPES OF SYNTAX DIRECTED DEFINITIONS

- Check whether the following are L-attributed definitions.

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

It is L-attributed definition

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

It is not L-attributed definition because Q.i takes value from R.s

BOTTOM UP EVALUATION OF S-ATTRIBUTED DEFINITIONS

- Consider a SDD as S → ABC {S.at = f(A.at, B.at, C.at)}

Index	State stack	Value stack
top	C	C.at
top-1	B	B.at
top-2	A	A.at
	.	
	.	
	.	
bottom		

Index	State stack	
ntop	S	f(A.at, B.at, C.at)}
	.	
	.	
	.	
bottom		

ntop = top – r + 1 where r is the length of RHS

BOTTOM UP EVALUATION OF S-ATTRIBUTED DEFINITIONS

- Consider the example of desk calculator.

Index	Value stack implementation
L → E n	Print (E.val)
E → E + T	val[ntop]= val[top-2] + val[top]
E → T	No action
T → T * F	val[ntop]= val[top-2] * val[top]
T → F	No action
F → (E)	val[ntop]= val[top-1]
F → digit	No action

PARSING & EVALUATION OF 4 * 5 + 3

STACK		INPUT	REDUCE ACTION
STATE	VALUE		
		4 * 5 + 3 $	
digit	4	*5 + 3 $	
F	4	*5 + 3 $	F → digit

STACK		INPUT	REDUCE ACTION
STATE	VALUE		
		4 * 5 + 3 $	
digit	4	* 5 + 3 $	
F	4	* 5 + 3 $	F → digit
T	4	* 5 + 3 $	T → F
T *	4 -	5 + 3 $	
T * digit	4 - 5	+ 3 $	
T * F	4 - 5	+ 3 $	F → digit
T	20	+ 3 $	T → T * F
E	20	+ 3 $	E → T
E +	20 -	3 $	
E + digit	20 - 3	$	
E + F	20 - 3	$	F → digit
E + T	20 - 3	$	T → F
E	23	$	E → E + T