CO3 : SDD, evaluation orders for SDD's, Applications of SDT.
Symbol table : Structure, Operations, Implementation and Management.

KESHAV MEMORIAL ENGINEERING COLLEGE

Expt. No._____
Date_____

CO3

Syntax Directed Definitions (SDD)

CFG with attributes and rules.
- attributes ← associated with ← grammar symbols
- rules ← productions.

X : a → to denote the value of 'a' at a particular parse-tree node labeled X.
- X → symbol
- a → attributes (types) → strings, table references
  - numbers, types.

* describe how the attributes are computed at those nodes of the parse tree.

For each Nonterminal of a production rule.

Synthesized Attributes
* defined by a semantic rule associated with the production at N. (Tree node)
* must have non terminal as its head.
* N is only defined in terms of (attribute values) children of N and N itself

Inherited Attributes
* defined by a semantic rule associated with the production at the parent of N
* Nonterminal must be in its body as a symbol.
* N's parent N itself N's siblings.

Terminals
can have → Synthesized Attributes. (but not for inherited)
* Att for terminals have lexical values supplied by lexical analyzer
* No semantic rules in SDD. for computing value of an attribute for a terminal.

Evaluation orders for SDD's

S-Attributed SDD
* only involves Synthesized Attributes.
* each rule computes an attribute for the NT at the head of a production from attributes taken from the body of the production.
* implemented naturally in conjunction with an LR parser.
* SDD without side effects called an attribute grammar.
* Annotated parse trees are used.

Ex:- (.val) → single synthesized attribute

$E \rightarrow E_1 + T$

E.val = E_1.val + T.val.

$F \rightarrow$ (digit ← Terminal)

F.val = digit.lexval.

~~L-inherited~~ well suited for
* Bottom-up parsing techniques.

L-Attributed SDD
can inherit values from left siblings only.
* uses both synthesized and inherited attributes.
* evaluated by depth-first and left-to-right parsing manner.
* semantic actions are placed anywhere in RHS.
* Top-Down.

Ex:-
① $T \rightarrow FT'$
T'.inh = F.val
② $T' \rightarrow *FT_1'$
T_1'.inh = T'.inh * F.val

Dependency graphs.
⇒ the flow of information among the attribute instances in a particular parse tree.

* In SDD, terminal digit has a synthesized attribute lexval (Integer)

Ex:-

Production	Semantic Rules.
$A \rightarrow B$	A.s := B.i ;
	B.i := A.s + 1 ;

NOTE:- An inherited attribute at a Node N cannot be defined in terms of attribute values of the children of node N.
But a synthesized attribute at Node N can be defined in terms of inherited attribute.

Syntax Directed Translation: SDT

CFG with program fragments embedded within production bodies.
↓
are called Semantic actions.

* { }
  - must be placed in
  - they can occur at any position within production body.
* implemented by 1st building a parse tree and then performing actions left to right (DFS).
* SDT's are implemented during parsing without building a parse tree.

Concepts: 1. Attributes. 2. Translation schemes.
- Attributes → used in SDD
- Translation schemes → written in SDT

Attributes: It is any quantity associated with a programming construct.
Ex:- Data types, no. of instructions, location, non-terminals / terminals symbols.

Translation scheme: It is a notation for attaching program fragments to the productions of a grammar.

SDD → more readable.
SDT → more efficient.
} differences

3.4 SDT.
unit-3 ⇒ p11
this book last

Applications:

1. Construct Syntax Trees.
2. Execute Arithmetic expression.
3. Infix to postfix
4. Infix to prefix.
5. Binary to decimal conversion.
6. Used to generate intermediate code.
7. Storing information into symbol table.
8. type checking also.

9. In counting number of reduction.

D → TL	L.inh = T.type
T → int	T.type = integer
T → float	T.type = float
L → L1, id	L1.inh = L.inh; add type (id.entry, L.inh)
L → id	add type (id.entry, L.inh).

Symbol Table

* DS to hold information about source program constructs.
* used by the synthesis phases to generate the target code.
* Information:
  - → identifier (lexeme) (string)
  - → its type → metadata.
  - → its position
* It associates attributes with identifiers used in a program.
* Used by analysis and synthesis phase:
  - ⇒ To verify the used identifiers have been declared.
  - ⇒ To verify that expressions and assignments are semantically correct (type checking.)
  - ⇒ To generate intermediate (or) target code.

Symbol Table Structure

lang: Pascal; scoping unit: procedures
C; blocks, functions

* Scope.
* Visibility → how much of the rest of the program can access that variable.
* Life times
* global variables.
* Automatic → stack storage.
* static variables. * Storage class

Contents in Symbol Table

How names are stored

* Name; a string.
* Attribute;
  - → Reserved word.
  - → variable name.
  - → Type name.
  - → procedure name.
  - → constant name.

* Fixed-length name.
* Variable-length name.
* Symbol table entries.

* Data type
* Storage allocation, size, ...
* Scope information: where and when it can be used.
* The scanner can enter an identifier into a symbol table if it is not already there

Expt. No.
Date

Operations on Symbol Table.

* allocate → new empty symbol table.
* free → remove all entries.
* insert
* lookup
* set_attribute
* get_attribute.
* delete

Implementation techniques

⇒ First consideration is how to insert and lookup names variety of implementation techniques.

1. List ← ordered / unordered } O(n)
2. Self-organized list } O(n)
3. Binary search Tree } O(log2n) O(n)
4. Hash tables. O(1)

Management:

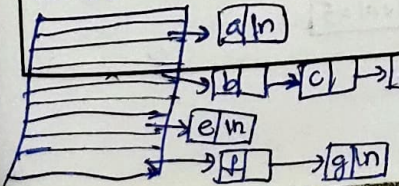
- A global Symbol table.
- A scope symbol table.

Intermediate code Generation

Source code → frontend translate → Intermediate code → backend generates → Target code.

parser → Static type checker → Intermediate code generator → code generator Backend.

Structure of hash table look like as



Abstract syntax treee

→ DAG
⇒ postfix notation : linear equation of tree.
⇒ 3-address code : a = b(op) c.
↓
abstract form of intermediate code

Quad ruples (four fields)

op
arg1
arg2
res.

Triples (three fields)

op
arg1
arg2

Indirect triples.
listing pointers to triplets.

index	Triplets
0	T0
1	T1
2	T2
3	T3

Declarations of Data types

D → T id ; D | E
T → B C | record '{' D '}'
B → int | float
C → ∈ | [num] C

D → Sequence of declarations.
T → basic, array / record types
B → basic types int and float
C → generates strings of zero or more integers, each integer surrounded by brackets.

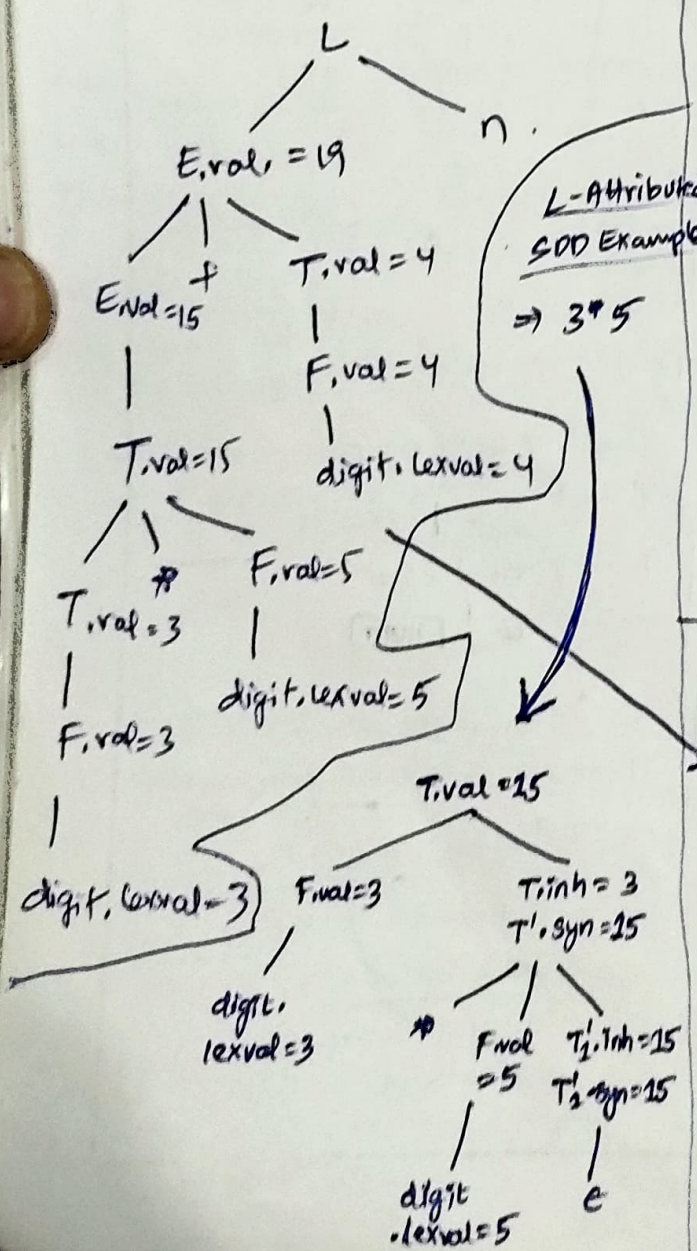
Page No.

SDD of a simple Desk Calculator.

SDD of evaluation of expression

Annotated parse tree for 3*5+4

Production	Semantic Rules
L → En	L.val = E.val
E → E1+T	E.val = E1.val + T.val
E → T	E.val = T.val
T → T1 * F	T.val = T1.val * F.val
T → F	T.val = F.val
F → digit	F.val = digit.lexval



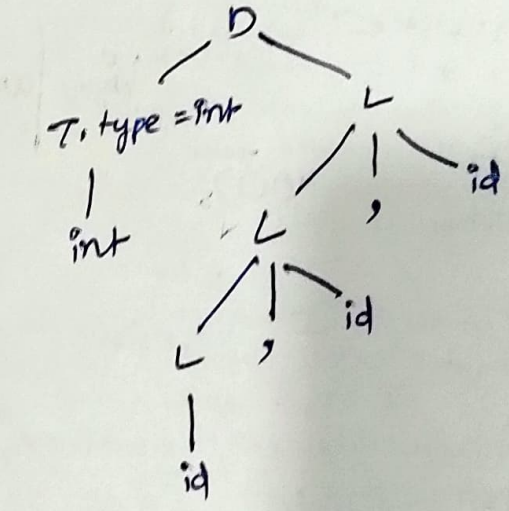
L-Attributed SDD Example.

⇒ 3*5

SDD for simple type declaration

Productions	Semantic Rules
1. D → TL	L.inh = T.type
2. T → int	T.type = int
3. T → float	T.type = float
4. L → L1 , id	L1.inh = L.inh
5. L → id	addType (id.entry, L.inh)
	addType (id.entry, L.inh).

for "int a,b,c"



evaluating an SDD at the Nodes of a parse Tree: