Syntax Directed Translation Scheme

Prepared By:

Dr. D. P. Singh

Graphic Era Deemed to be University, Dehradun

- Once the syntactic structure is known, semantic analyzer computes more information related to the meaning of the program.
- e.g.
- in typed language like C, semantic analyzer adds information to symbol table and performs the type checking.
- The information which is computed is beyond the capabilities of parsing techniques, therefore it is not regarded as syntax.
- to interleave syntax with semantic analysis, syntax directed translation is used.

Syntax Directed Translation/Definition

- Attributes are associated with the grammar symbols to represent the language constructs.
- Semantic rules are associated with the productions to compute the values of attributes.
- An attribute may hold a string, a number, a memory location etc.
- Semantic rules:
  - May generate intermediate code
  - May put information into the symbol table
  - May perform type checking
  - May issue error message
- Syntax Directed Translation= Grammar+ Semantic rules

Notations for attaching Semantic Rules

1. Syntax Directed Translation:
   - More information about implementation details
   - Indicate the order of evaluation of Semantic rules
2. Syntax Directed Definition:
   - Give high level specifications
   - no order of evaluation of semantic rules (hide implementation details)

Types of attributes:

Value of an attribute of a grammar symbol at a given parse-tree node is defined by a semantic rule associated with the production used at that node.

1. Synthesized attribute

   The value of attribute of a grammar symbol at a given parse tree node is determined by the attributes values at the child nodes.

Annotated Parse Tree

Parse tree shows the attribute values at each node is called annotated parse tree or decorated parse tree

The process of computing the attribute values at each node is known as annotating or decorating.

Example of SDD

- e.g.

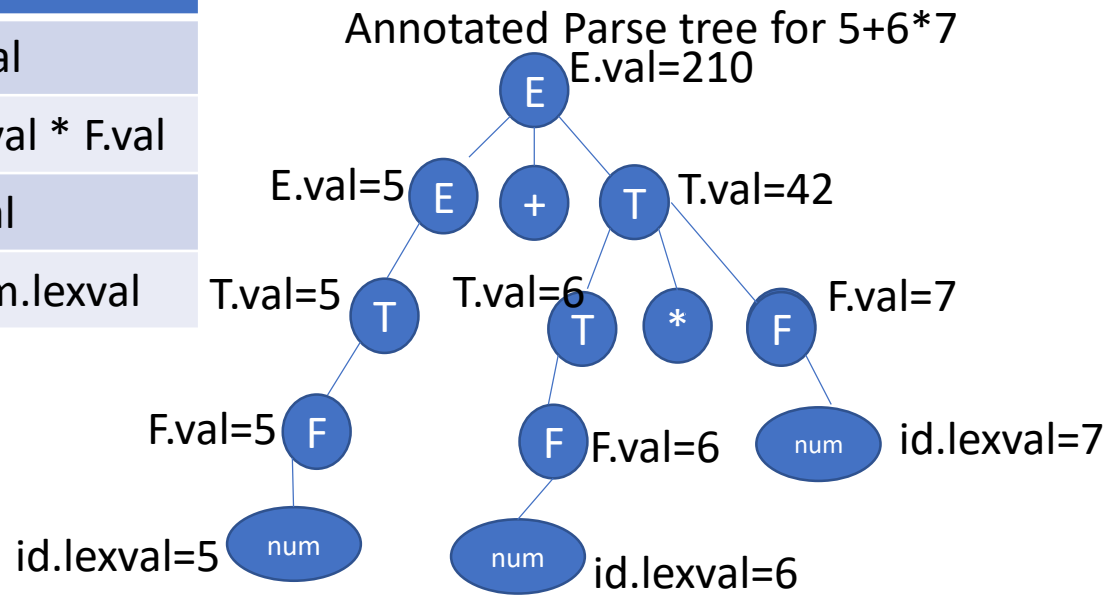
$E \rightarrow E^1+T$	$E.val=E^1.val+T.val$
$E \rightarrow T$	$E.val=T.val$
$T \rightarrow T^1*F$	$T.val=T^1.val * F.val$
$T \rightarrow F$	$T.val=F.val$
$F \rightarrow num$	$F.val=num.lexval$

S-Attributed Definition

- S-Attributed definition is a syntax directed definition

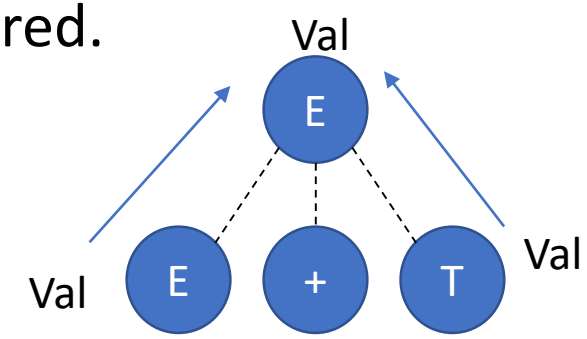
  that uses only synthesized attributes.
- Evaluation order: Bottom up or Post Order traversal of the parse tree
- e.g. SDD for evaluation of arithmetic expression

$E \rightarrow E^1+T$	$E.val=E^1.val+T.val$
$E \rightarrow T$	$E.val=T.val$
$T \rightarrow T^1*F$	$T.val=T^1.val * F.val$
$T \rightarrow F$	$T.val=F.val$
$F \rightarrow num$	$F.val=num.lexval$



Dependency Graph

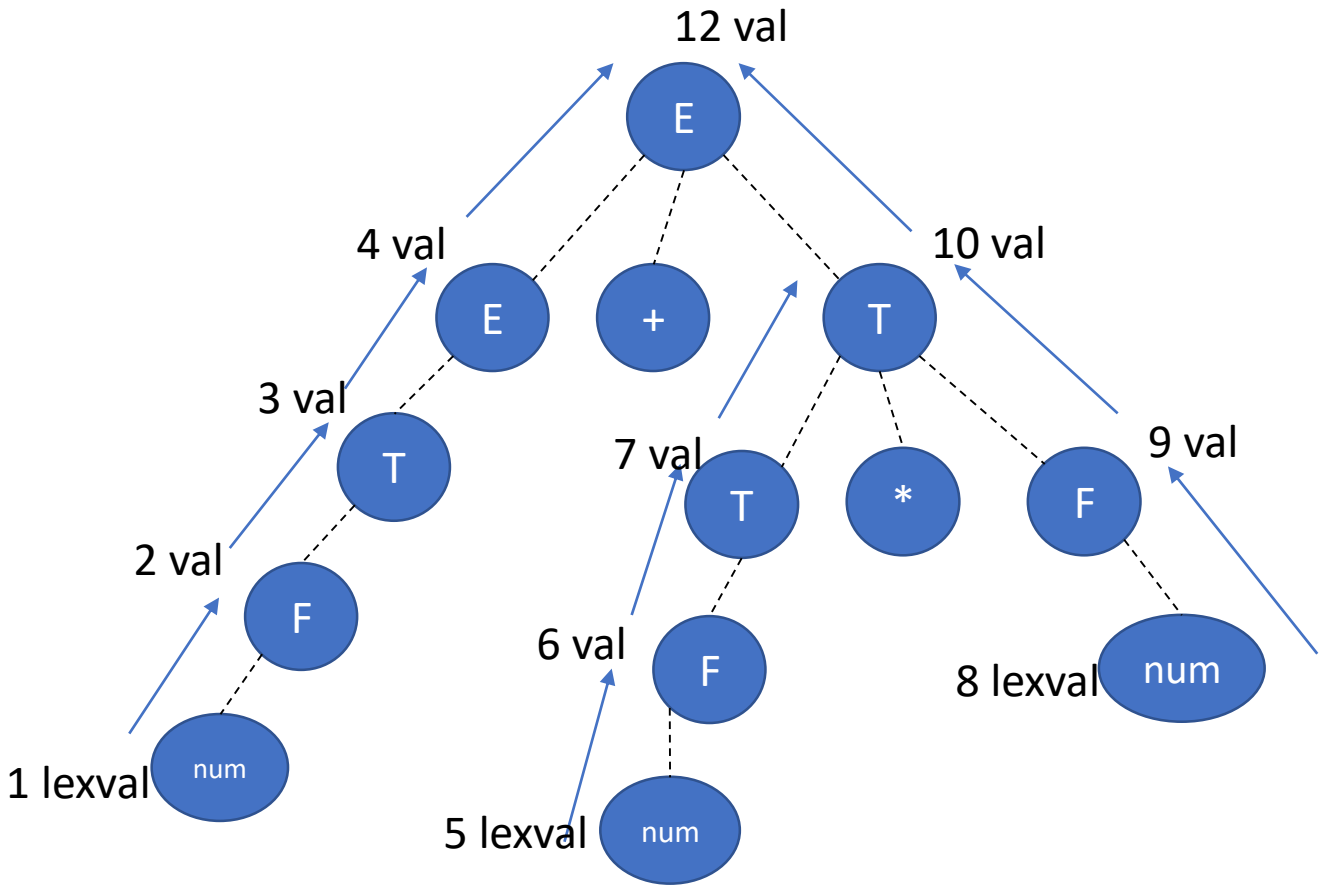
- General technique used to evaluate SDD
- with both Synthesized and inherited attributes
- Shows the interdependencies among the attributes of the various nodes of the parse tree
- If the value of attribute b depends upon the value of attribute c then there will be an edge from node for c to node for b (b←c).
- Dependency Rule:
  - If the attribute value of b depends upon the attribute value of c then the semantic rule to evaluate c will be processed first then semantic rule to evaluate b will be fired.



Construction of Dependency Graph

Construct the dependency graph for 5+6*7

$E \rightarrow E^1+T$	$E.val=E^1.val+T.val$
$E \rightarrow T$	$E.val=T.val$
$T \rightarrow T^1*F$	$T.val=T^1.val * F.val$
$T \rightarrow F$	$T.val=F.val$
$F \rightarrow num$	$F.val=num.lexval$

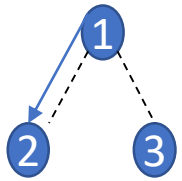


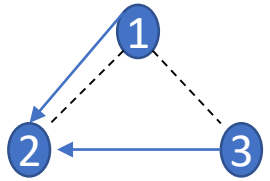
2 val means topological order is 2
and the attribute is val

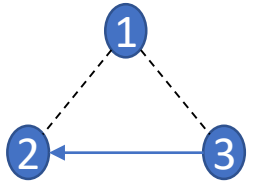
Types of attributes cntd…

2. Inherited attribute

   The value of attribute of a grammar symbol at a given parse tree node is determined by the attributes values at the parent and/or sibling nodes.







- e.g.

D→TL	L.in=T.type
T→int	T.type=inetger
L→L^1,id	L^1 .in=L.in, addtype(id.entry , L.in)
L→id	addtype(id.entry, L.in)

L-attributed Definition

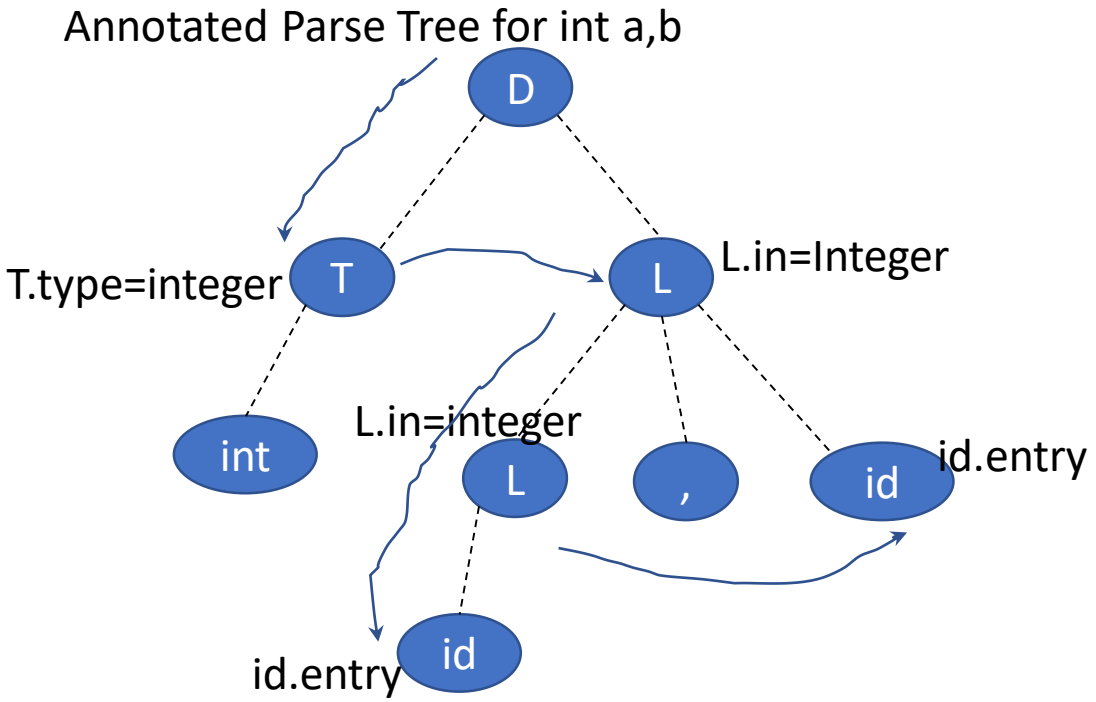
- For every production $A \rightarrow X_1X_2X_3X_4\ldots\ldots X_n$.

- Each inherited attribute of X_j for $1 \leq j \leq n$ will depends upon
  - The attributes of the symbols $X_1, X_2, X_3\ldots\ldots X_{j-1}$ or
  - The attribute of A.

Note: Uses the attribute from the above or from the left.
Order of evaluation: Depth First, Left to right

L-attributed definition cntd…

D→TL	L.in=T.type
T→int	T.type=integer
L→L^1,id	L^1 .in=L.in, addtype(id.entry , L.in)
L→id	addtype(id.entry, L.in)



Syntax Directed Translation Scheme

- SDT is a CFG in which attributes are associated
  with the grammar symbols and semantic
  actions are enclosed within braces ({})
  are written in the right side of the production.

- Semantic actions are like subroutines that are called at the appropriate times by the parser to enable the translation

- Position of the semantic action on right side of the production indicates time when it is called by the parser.

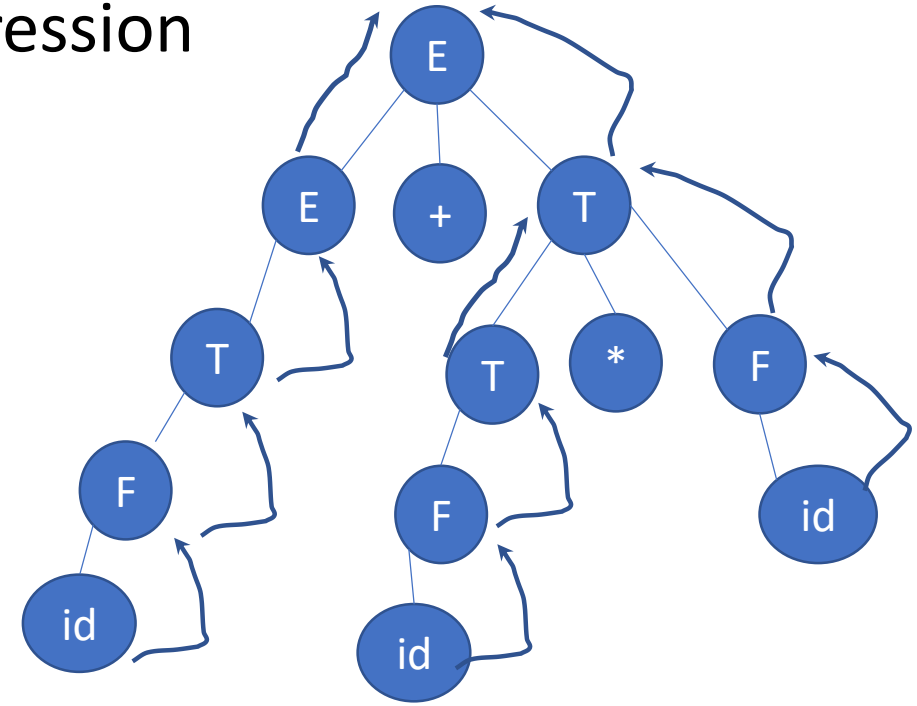
- Ensure the availability of attribute when the action refers to it.

Difference between S-attributed SDT and L-attributed SDT

S-attributed SDT	L-attributed SDT
Only synthesized attributes are used	Uses both synthesized and inherited attributes. Each inherited attribute is restricted to inherit either from parent or from left sibling e.g. A→XYZ { Y.attr=A.attr √ Y.attr=X.attr √ X.attr=Z.attr X }
Semantic actions are placed at right end of the production e.g. A→ XYZ {semantic action}	Semantic actions are placed anywhere on right hand side e.g. A→ { semantic action} XYZ A→ X{semantic action} YZ A→XY {semantic action}Z A→ XYZ {semantic action}
Attributes are evaluated by Bottom up parsing	Attributes are evaluated by traversing Depth first, left to right

Examples of SDT

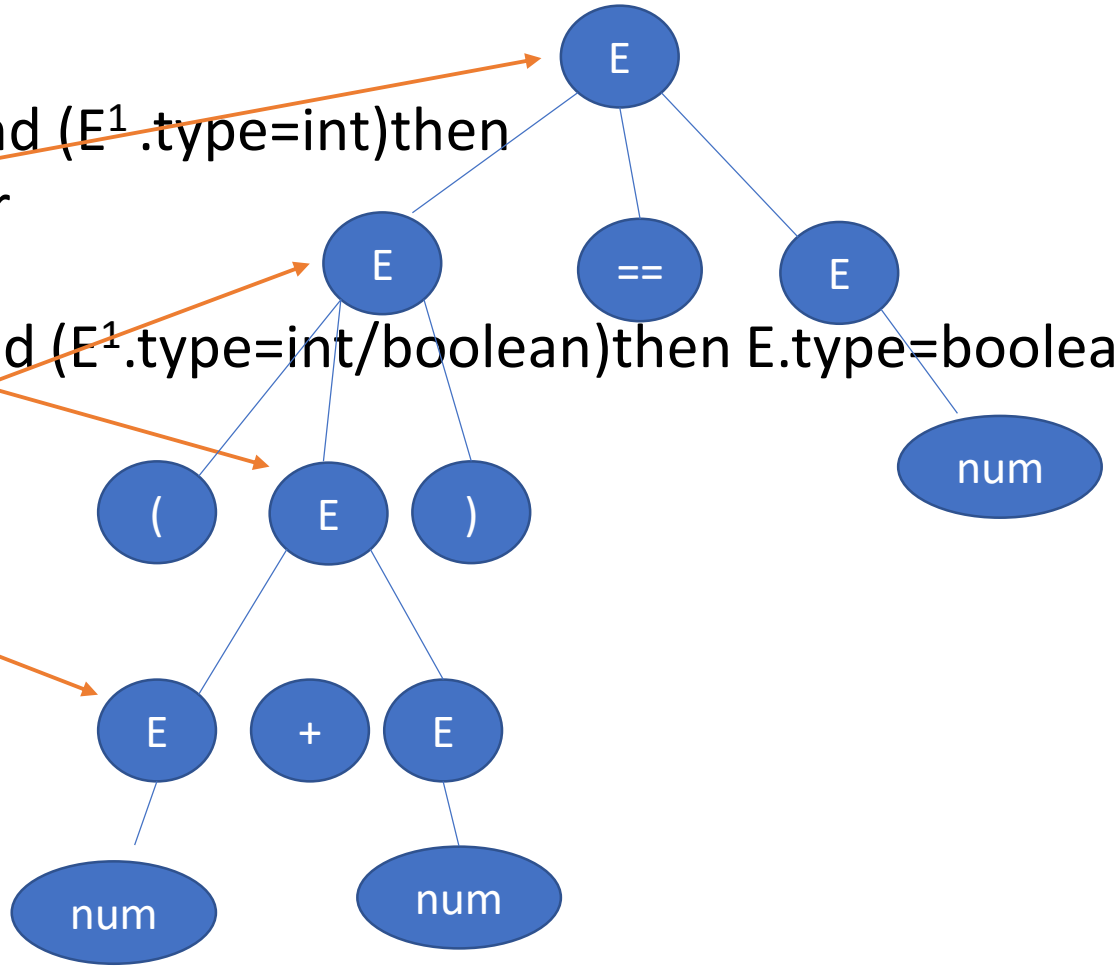
- Write the grammar first
- Construct the parse tree
- e.g. Write the SDT for arithmetic expression
- $E \rightarrow E^1+T$ {E.val= E^1 .val+T.val}
- $E \rightarrow T$ {E.val=T.val}
- $T \rightarrow T^1*F$ {T.val=T^1.val*F.val}
- $T \rightarrow F$ {T.val=F.val}
- $F \rightarrow id$ {F.val=id.lexval}



- Think how the bottom up evaluation can be done for id+id*id

SDT for Type Checking

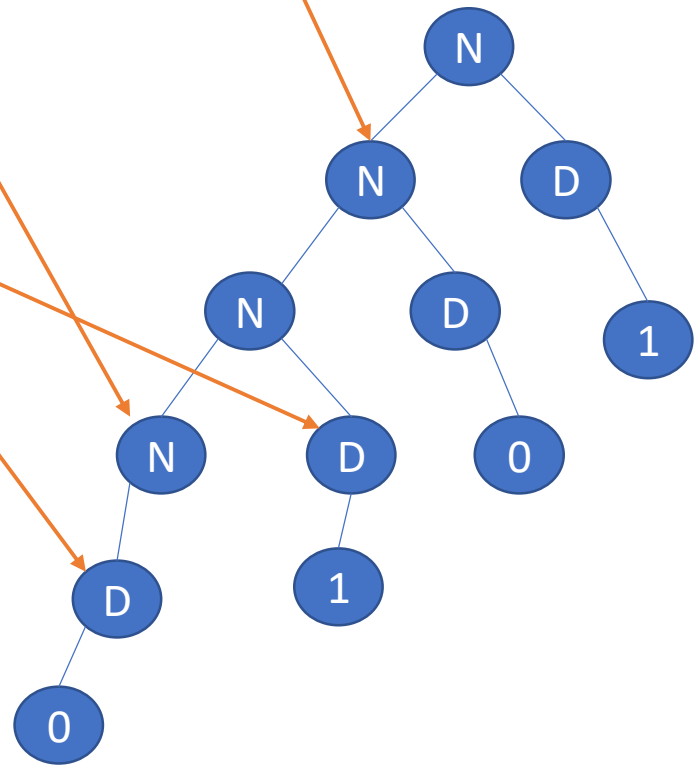
- $E \rightarrow E^1+E^2$ {if(E^1 .type==E^2 .type) and (E^1 .type=int)then
  E.type=int else error
  }
- $E \rightarrow E^1==E^2$ {if(E^1.type==E^2.type)and (E^1.type=int/boolean)then E.type=boolean else error}
- $E \rightarrow (E^1)$ {E.type=E^1.type}
- E→num {E.type=int}
- E→true {E.type=boolean}
- E→false {E.type=boolean}



- Think about the evaluation of expression (2+3)==5

SDT for counting 0 in Binary Number

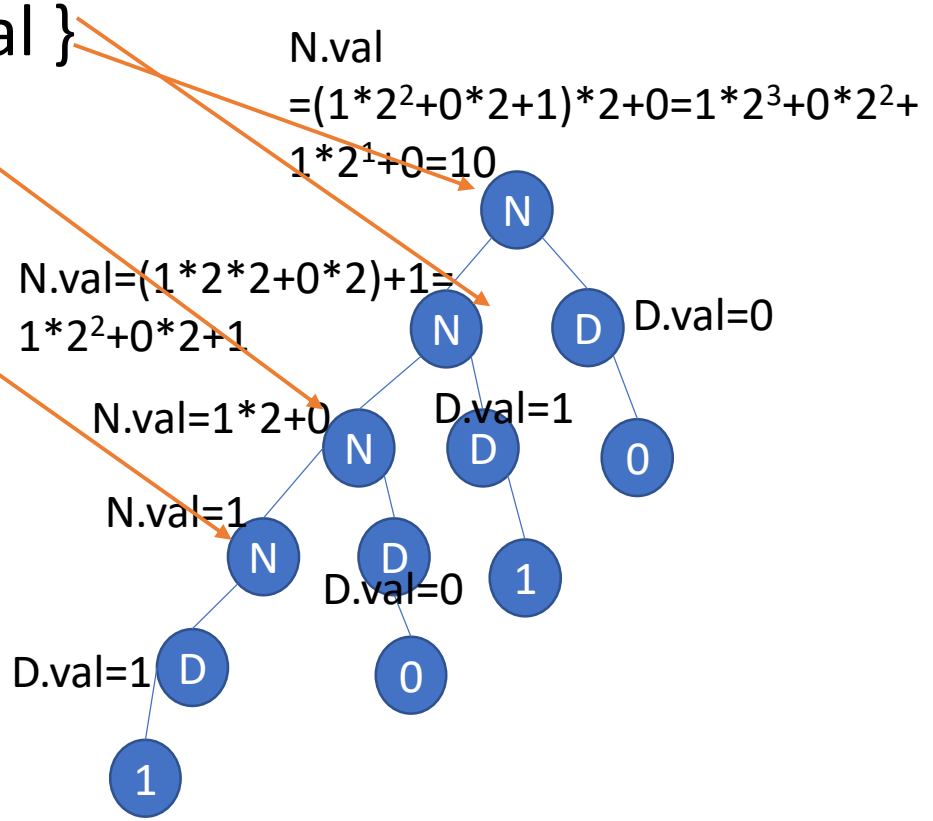
- $N \rightarrow N^1 D$ { $N.count = N^1.count + D.count$ }
- $N \rightarrow D$ {$N.count = D.count$ }
- $D \rightarrow 0$ {$D.count = 1$ }
- $D \rightarrow 1$ {$D.count = 0$ }



- Think about the binary number 0101

SDT for Binary to Decimal Conversion

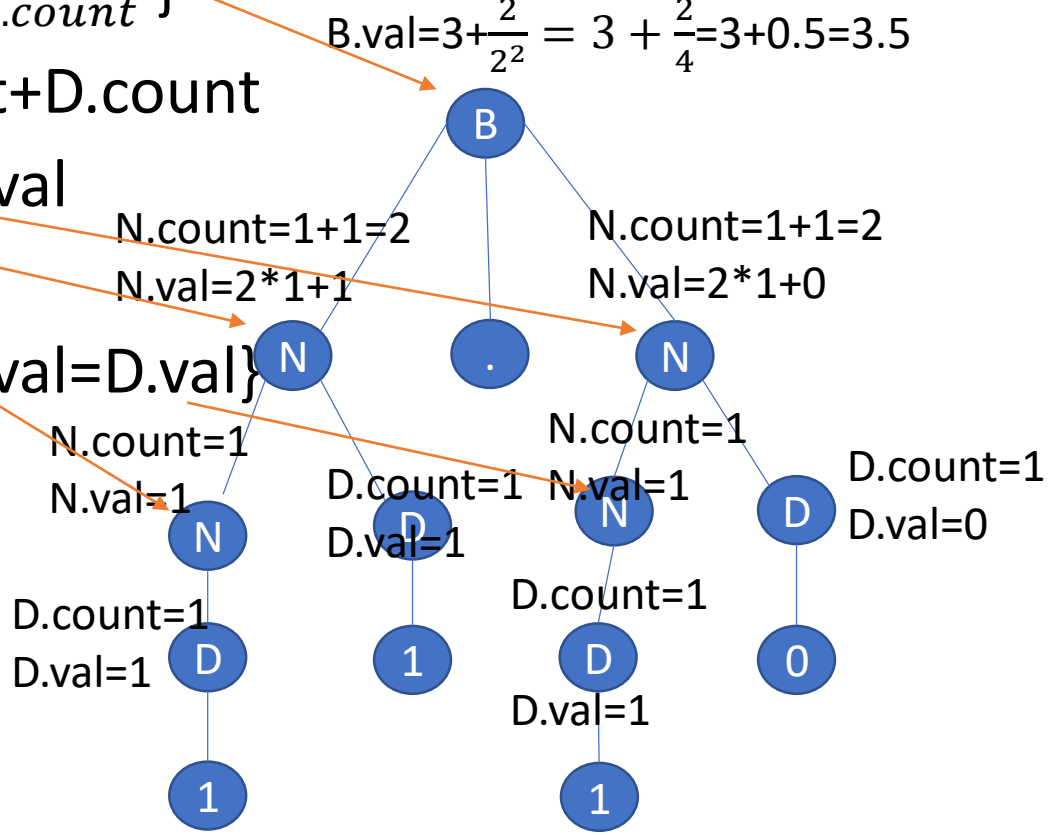
- $N \rightarrow N^1 D$ { $N.val = N^1.val * 2 + D.val$ }
- $N \rightarrow D$ {$N.val = D.val$ }
- $D \rightarrow 0$ {$D.val = 0$ }
- $D \rightarrow 1$ {$D.val = 1$ }



- Think about the binary number 1010

SDT for Decimal point Binary to Decimal

- B→$N^1.N^2$ { B.val=N^1.val+$\frac{N^2.val}{2^{N^2.count}}$ }
- N→N^1D {N.count=N^1.count+D.count
  N.val=2*N^1.val+D.val
  }
- N→D {N.count=D.count,N.val=D.val}
- D→1 {D.count=1, D.val=1}
- D→0 {D.count=1, D.val=0}



- Think about the binary number 11.10

- Thank You