Compiler Design

Unit-4

Syntax-Directed Translation

Syntax Directed Definitions, Construction of syntax tree, S- attributed and L-attributed SDDs with example.

ERROR Handling during Syntax analysis

- When the stream of tokens coming from lexical analyzer disobeys the syntactic (grammatical) rules of a language, syntactic error occurs.
- Handling Syntactic Error:
  - Report Errors
  - Recover from discovered error
  - Aim at not slowing down the processing of the remaining program

ERROR Handling during Syntax analysis

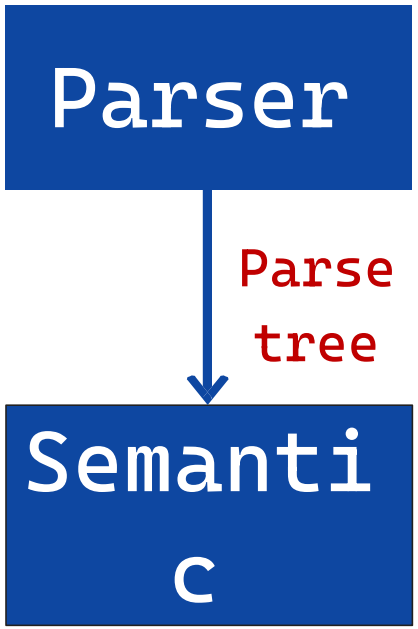
- **Error Recovery Strategies:**
- **Panic Mode Recovery:** The parser discards the input symbols one at a time until one of designated set of synchronizing tokens is found.
- Simple and it does not go into loop
- Adequate when the presence of multiple errors in same statement is rare
- **Phrase Level Recovery:** The parser performs local correction on remaining input when the error is discovered.
- The parser replaces the prefix of the remaining input by some string that allows the parser to carry on its execution
- Drawback: Error correction is difficult when actual error occurred before the point of detection

ERROR Handling during Syntax analysis

- **Error Productions:** If we know common errors that can be encountered, we can augment the grammar for the language with productions that generate erroneous constructs.
- Use the new grammar (with augmented productions) for the parser.
- In case an error production is used by parser, generate appropriate diagnostic to indicate the errors /erroneous constructs.
- **Global Correction:** The aim is to make as few changes as possible while converting an incorrect input string to a valid string.
- Given an incorrect input x, find a parse tree for a related string w(using the given grammar) such that the number of changes (insertion/deletion) required to transform x to w is minimum
- Too costly to implement

SDT(Syntax Directed Translation)

- Semantic Analysis

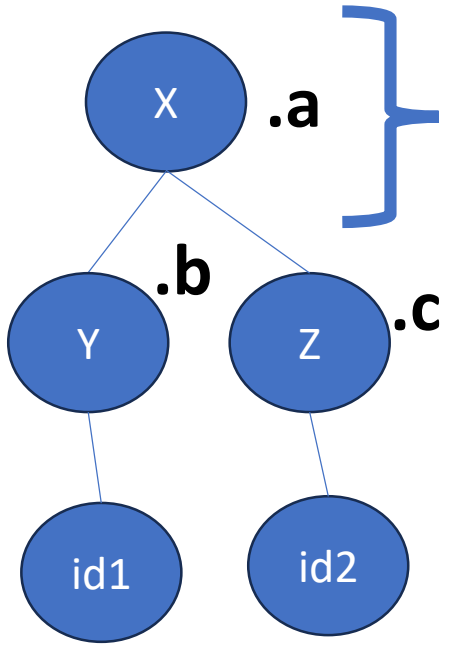


SDT

- Types of Semantic action performed by Semantic Analyzer:
- Scope resolution
- Type checking
- Array bound checking
- Flow Control Check

SDT(Syntax Directed Translation)

- Syntax directed translation is a generalization of context free grammar in which each grammar symbol has an associated set of attributes.
- The attributes can be a number, type, memory location, return type etc….
- Types of attributes are:
    1. Synthesized attributed
    2. inherited attributed
- Grammar + Semantic Rules = SDT

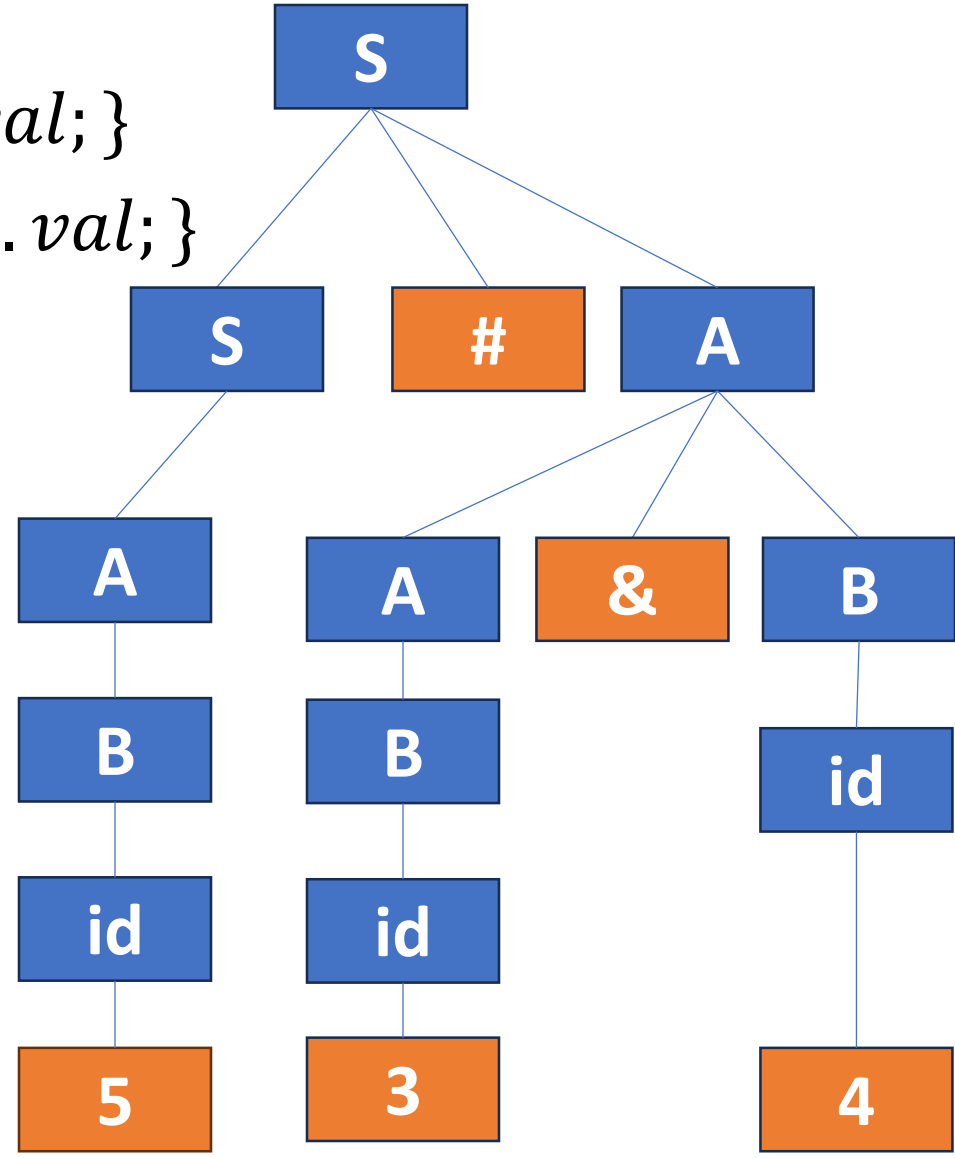


SDT(Syntax Directed Translation)

- $S \rightarrow S\#A|A \; \{S.val = S.val * A.val; S.val = A.val; \}$
- $A \rightarrow A\&B|B \; \{A.val = A.val + A.val; A.val = B.val; \}$
- $B \rightarrow id \; \{B.val = id.val; \}$
- Given String: 5#3&4

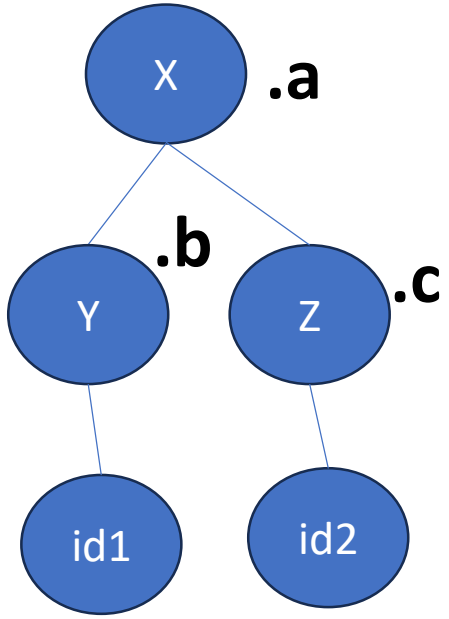
SDT(Syntax Directed Translation)

- $S \rightarrow S\#A|A \ \{S.val = S.val * A.val; S.val = A.val;\}$
- $A \rightarrow A\&B|B \ \{A.val = A.val + A.val; A.val = B.val;\}$
- $B \rightarrow id \ \{B.val = id.val;\}$
- Given String: 5#3&4



SDT(Syntax Directed Translation)

- **Annotated Parse Tree:** A Parse Tree with attribute values at each node, is annotated parse tree.
- **Synthesized Attributes:** An Attribute is said to be a synthesized attribute if its value at a Parent Node is determined from the attribute values of the children of the node.
- Synthesized Attributes can be evaluated during a single bottom up traversal
- **Inherited attribute:** An inherited value at a node in a parse tree is computed from the value of attributes at the parent and/or siblings of the node.



SDT(Syntax Directed Translation)

- Application:
  - Executing Arithmetic Expression
  - Conversion from infix to postfix
  - Conversion from infix to prefix
  - Conversion from binary to decimal
  - Counting number of reductions
  - Creating Syntax Tree
  - Generating Intermediate Code
  - Type Checking
  - Storing type information into symbol

Arithmetic Expression by SDT

- $E \rightarrow E\&T\{E.val = E.val * T.val\}$
- $E \rightarrow T\{E.val = T.val\}$
- $T \rightarrow T@F\{T.val = T.val - F.val\}$
- $T \rightarrow F\{T.val = F.val\}$
- $F \rightarrow num\{F.val = num\}$
- Input String: 4&8@5&7@3

Arithmetic Expression

- $E \rightarrow E_1 + T$ $\quad E.val \rightarrow E_1.val + T.val$
- $E \rightarrow T$ $\quad E.val \rightarrow T.val$
- $T \rightarrow T * F$ $\quad T.val \rightarrow T.val * F.val$
- $T \rightarrow F$ $\quad T.val \rightarrow F.val$
- $F \rightarrow (E)$ $\quad F.val \rightarrow E.val$
- $F \rightarrow digit$ $\quad F.val \rightarrow digit.lexval$
- Input String:2+9*8

Infix to postfix

- $E \rightarrow E_1 + T$
- $E \rightarrow E_1 - T$
- $E \rightarrow T$
- $T \rightarrow 0$
- $T \rightarrow 1$

:

- $T \rightarrow 9$
- Input String:1-2+3

$$E.x \rightarrow E_1.x|T.x| +$$
$$E.x \rightarrow E_1.x|T.x| -$$
$$E.x \rightarrow T.x$$
$$T \rightarrow 0$$
$$T \rightarrow 1$$
$$\vdots$$
$$T \rightarrow 9$$

Exercise

Production	Semantic rules
L → E_n	Print (E.val)
E → E_1+T	E.Val = E_1.val + T.val
E → T	E.Val = T.val
T → T_1*F	T.Val = T_1.val * F.val
T → F	T.Val = F.val
F → (E)	F.Val = E.val
F → digit	F.Val = digit . lexval

Annotated parse tree for 3*5+4n

Draw Annotated Parse tree for following:

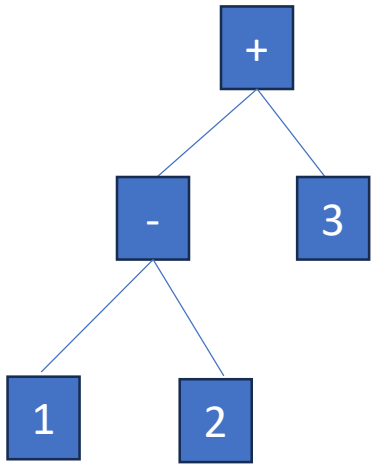
1. 7+3*2n
2. (3+4)*(5+6)n

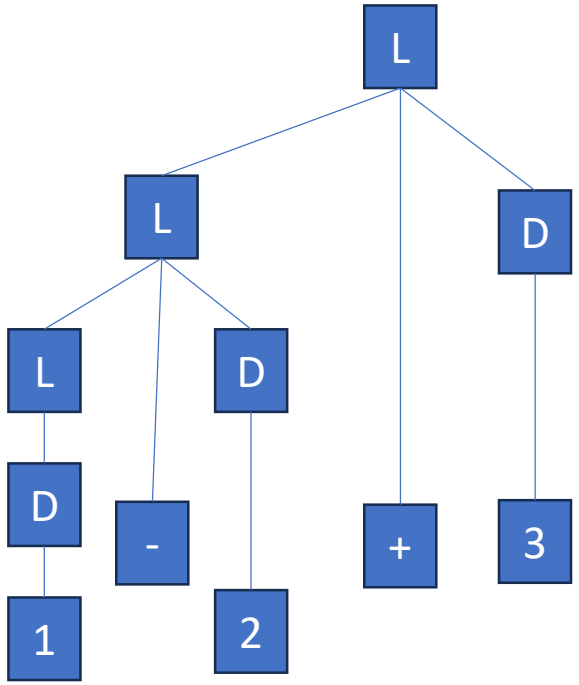
SDT

- **Abstract Syntax Tree (AST):** Each node in an AST represents an operator and the children of the operator are the operands (of this operation)
- e.g., 1 +2
- There is no unimportant details present.

SDT

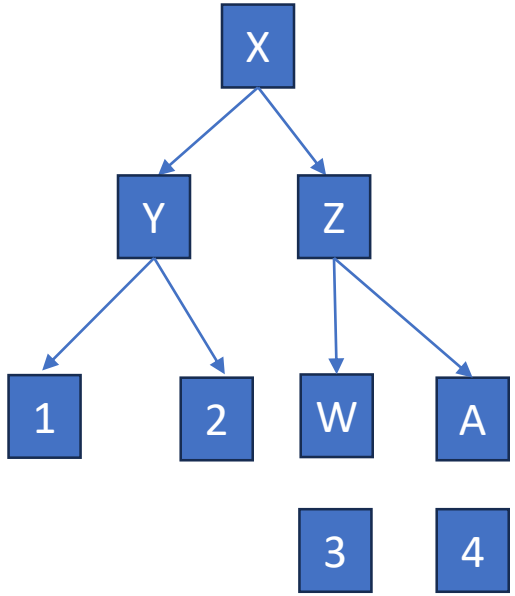
- **Concrete Syntax Tree:** It is a normal parse tree and underlying grammar is called concrete syntax for the language.
- e.g., 1 -2 + 3
- Left to right evaluation





Traversal of a Tree

- Starts at the root and visit each node of the tree in some particular order.
- Depth First Traversal: start with root => recursively visit the children of each node in left to right manner.



SDT

- Translation: Given an input string x, when we construct a parse tree for x and convert it into annotated parse tree (output), this mapping from input to output is called translation.
- Translation Scheme: It is a context free grammar in which program pieces/fragments are embedded in RHS of the productions (called semantic actions)
- Translation Scheme specifies explicitly the evaluation order of semantic actions.
- Translation scheme generates an output for each string x present in the language generated by the grammar by executing the actions in the order in which they appear in depth first traversal of parse tree for x.

SDT(Syntax Directed Translation)

- $S \rightarrow AS \ \{printf(3);\}$
- $A \rightarrow AB \ \{printf(5);\}$
- $A \rightarrow a \ \{printf(1);\}$
- $B \rightarrow bC \ \{printf(6);\}$
- $B \rightarrow dB \ \{printf(4);\}$
- $C \rightarrow c \ \{printf(2);\}$
- Given String: $aadbc$

Infix to postfix

- $E \rightarrow E_1 + T$
- $E \rightarrow E_1 - T$
- $E \rightarrow T$
- $T \rightarrow 0$
- $T \rightarrow 1$

:

- $T \rightarrow 9$
- Input String:1-2+3

$E.x \rightarrow E_1.x|T.x| +$

$E.x \rightarrow E_1.x|T.x| -$

$E.x \rightarrow T.x$

$T \rightarrow 0$

$T \rightarrow 1$

:

$T \rightarrow 9$

- $E \rightarrow E_1 + T$: print(“+”);
- $E \rightarrow E_1 - T$: print(“-”);
- $E \rightarrow T$: print(“”);
- $T \rightarrow 0$:print(“0”);
- $T \rightarrow 1$:print(“1”);

:

- $T \rightarrow 9$:print(“9”);
- Input String:1-2+3

Types of Attributes

- S-Attributed: S-attributed definition is one such class of syntax directed definition with synthesized attributes only.
- Synthesized attributes can be evaluated using bottom up parser only.
- L-Attributed: A syntax directed definition is L-attributed if each inherited attribute of $X_j, 1 \leq j \leq n$,on the right side of $A \rightarrow X_1, X_2, \ldots, X_n$ depends only on:
  - 1. The attributes of the symbols $X_1, X_2, \ldots, X_{j-1}$to the left of $X_j$in the production and
  - 2. The inherited attribute of A.

Types of SDT

- S-Attributed SDT
  - Based on synthesized Attribute
  - Use bottom up parsing
  - Semantic rules always written at right most position in RHS
- L-Attributed SDT:
  - Based on both synthesized and inherited attributes
  - Top down Parsing
  - Semantic rules anywhere in RHS

Questions

Which of the following statement is False?

1. S-attributed SDT uses synthesized attributes.
2. L-attributed SDT uses inherited attribute.
3. Every S-attributed SDT is L-attributed SDT.
4. None of the above

Questions

Which of the following statement is False?

1. S-attributed SDT uses synthesized attributes.
2. L-attributed SDT uses inherited attribute.
3. Every S-attributed SDT is L-attributed SDT.
4. None of the above

- Answer: 2.

Example

- $A \rightarrow LM \; \{L.i = l(A.i); M.i = m(L.s); A.s = f(M.s)\}$
- $A \rightarrow QR \; \{R.i = \; r(A.i); \; Q.i = q(R.s); A.s = f(Q.s)\}$

Example

Consider the productions $A \to PQ$ and $A \to XY$. Each of the five non-terminals $A, P, Q, X,$ and Y has two attributes: s is a synthesized attribute, and i is an inherited attribute. Consider the following rules.

- Rule 1 : $P.i = A.i + 2$, $Q.i = P.i + A.i$, and $A.s = P.s + Q.s$
- Rule 2 : $X.i = A.i + Y.s$ and $Y.i = X.s + A.i$

Which one of the following is TRUE?

A. Both Rule 1 and Rule 2 are L-attributed.
B. Only Rule 1 is L-attributed.
C. Only Rule 2 is L-attributed.
D. Neither Rule 1 nor Rule 2 is L-attributed.

Example

Consider the productions $A \rightarrow PQ$ and $A \rightarrow XY$. Each of the five non-terminals A, P, Q, X, and Y has two attributes: s is a synthesized attribute, and i is an inherited attribute. Consider the following rules.

- Rule 1 : $P.i = A.i + 2$, $Q.i = P.i + A.i$, and $A.s = P.s + Q.s$
- Rule 2 : $X.i = A.i + Y.s$ and $Y.i = X.s + A.i$

Which one of the following is TRUE?

A. Both Rule 1 and Rule 2 are L-attributed.
B. Only Rule 1 is L-attributed.
C. Only Rule 2 is L-attributed.
D. Neither Rule 1 nor Rule 2 is L-attributed.

- Answer: B.

Consider the following grammar and the semantic actions to support the inherited type declaration attributes. Let X_1, X_2, X_3, X_4, X_5, and X_6 be the placeholders for the non-terminals D, T, L or L_1 in the following table:

Production rule	Semantic action
$D \rightarrow TL$	$X_1.\text{type} = X_2.\text{type}$
$T \rightarrow \text{int}$	$T.\text{type} = \text{int}$
$T \rightarrow \text{float}$	$T.\text{type} = \text{float}$
$L \rightarrow L_1, id$	$X_3.\text{type} = X_4.\text{type}$ $\text{addType}(id.\text{entry}, X_5.\text{type})$
$L \rightarrow id$	$\text{addType}(id.\text{entry}, X_6.\text{type})$

Which one of the following are appropriate choices for X_1, X_2, X_3 and X_4?

A. $X_1 = L,\ X_2 = T,\ X_3 = L_1,\ X_4 = L$
B. $X_1 = T,\ X_2 = L,\ X_3 = L_1,\ X_4 = T$
C. $X_1 = L,\ X_2 = L,\ X_3 = L_1,\ X_4 = T$
D. $X_1 = T,\ X_2 = L,\ X_3 = T,\ X_4 = L_1$

Consider the following grammar and the semantic actions to support the inherited type declaration attributes. Let X_1, X_2, X_3, X_4, X_5, and X_6 be the placeholders for the non-terminals D, T, L or L_1 in the following table:

Production rule	Semantic action
$D \rightarrow TL$	$X_1.\text{type} = X_2.\text{type}$
$T \rightarrow \text{int}$	$T.\text{type} = \text{int}$
$T \rightarrow \text{float}$	$T.\text{type} = \text{float}$
$L \rightarrow L_1, id$	$X_3.\text{type} = X_4.\text{type}$ $\text{addType}(id.\text{entry}, X_5.\text{type})$
$L \rightarrow id$	$\text{addType}(id.\text{entry}, X_6.\text{type})$

Which one of the following are appropriate choices for X_1, X_2, X_3 and X_4?

A. $X_1 = L,\ X_2 = T,\ X_3 = L_1,\ X_4 = L$
B. $X_1 = T,\ X_2 = L,\ X_3 = L_1,\ X_4 = T$
C. $X_1 = L,\ X_2 = L,\ X_3 = L_1,\ X_4 = T$
D. $X_1 = T,\ X_2 = L,\ X_3 = T,\ X_4 = L_1$

- Answer: A.