Compiler Construction (CS-636)

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

(Week 13) Lecture 25 & 26

Objectives: Learning objectives of these lectures are

Students will able to understand:

- What is Syntax Directed Translation (SDT)?
- What are Semantic Rules?
- What are attributed used in SDT?
  - Synthesized Attributes
  - Inherited Attributes

Text Book & Resources:

Compilers Principles Techniques and Tools (2nd Edition) by Alfread V. Aho, Ravi Sethi.

Videos Links:

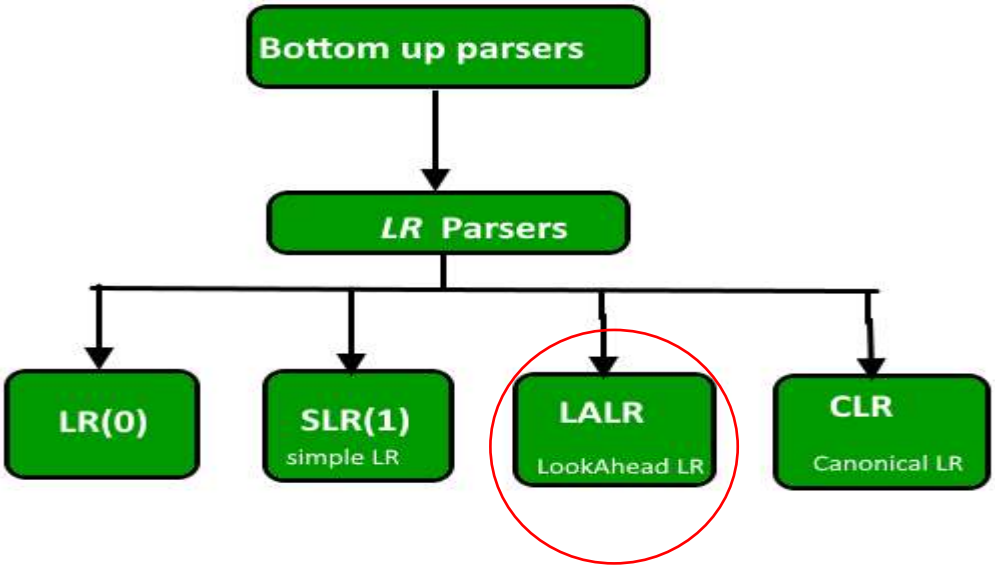
https://youtu.be/PSCRPkdHCaw (Part 1)
https://youtu.be/Kf9CYlBYwWs (Part 2)
https://youtu.be/p4Gina2glE8 (Part 3)
https://youtu.be/jUvhJXg_vmM (Part 4)
https://youtu.be/uOjd5t1m-84 (Part 5)

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

Summery Week#12 Lectures

In the last week, we had discussed the Bottom Up Parsers and also discussed the following types of parsers. From these parsers we had discussed LALR(1) Parser and its steps with different examples.

Classification (Types) of bottom up parsers



In this week, we will discuss the Syntax Directed Translation (SDT). The detail description of SDT is given below:

Syntax Directed Translation (SDT)

Background

Parser uses a CFG (Context-free-Grammar) to validate the input string and produce output for next phase of the compiler. Output could be either a parse tree or abstract syntax tree. Now to interleave semantic analysis with syntax analysis phase of the compiler, we use Syntax Directed Translation.

Definition

Syntax Directed Translation are augmented rules to the grammar that facilitate semantic analysis.
Grammar + Semantic Rules = SDT
SDT involves passing information bottom-up and/or top-down the parse tree in form of attributes attached to the nodes. Syntax directed translation rules use

1) lexical values of nodes,
2) constants &
3) attributes associated to the non-terminals in their definitions.

The general approach to Syntax-Directed Translation is to construct a parse tree or syntax tree and compute the values of attributes at the nodes of the tree by visiting them in some order. In many cases, translation can be done during parsing without building an explicit tree.

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

Example# 1

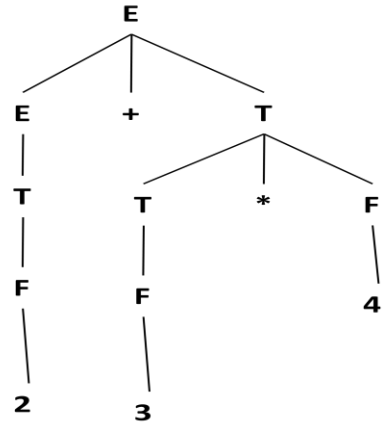
E→E+T | T
T→T*F | F
F→digit

This is a grammar to syntactically validate an expression having additions and multiplications in it. Now, to carry out semantic analysis we will augment SDT rules to this grammar, in order to pass some information up the parse tree and check for semantic errors, if any. In this example we will focus on evaluation of the given expression, as we don't have any semantic assertions to check in this very basic example.

S#	Production	Semantic Rules
1	E→E+T	{ E.Val = E.Val + T.Val }
2	E→ T	{ E.Val = T.Val }
3	T→T*F	{ T.Val = T.Val * F.Val }
4	T→F	{ T.Val = F.Val }
5	F→digit	{ F.Val = Lexval }

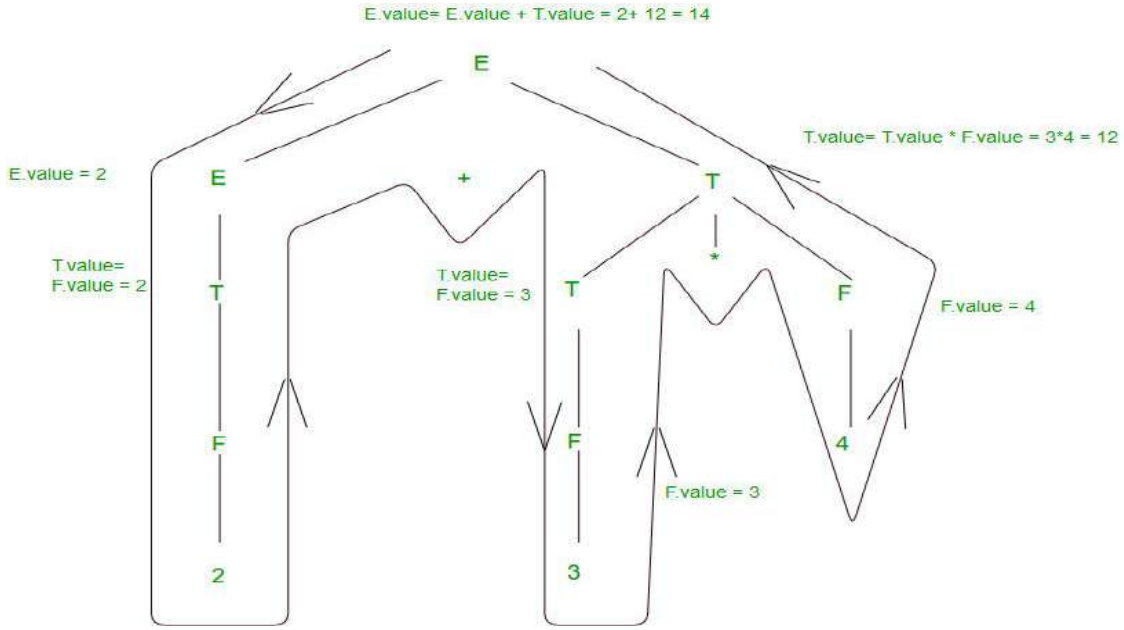
For understanding translation rules further, we take the first SDT augmented to [E -> E+T] production rule. The translation rule in consideration has val as attribute for both the non-terminals E & T. Right hand side of the translation rule corresponds to attribute values of right side nodes of the production rule and vice-versa. Generalizing, SDT are augmented rules to a CFG that associate 1) set of attributes to every node of the grammar and 2) set of translation rules to every production rule using attributes, constants and lexical values.

Let's take a string to see how semantic analysis happens – S = 2+3*4. Parse tree corresponding to S would be



To evaluate translation rules, we can employ one depth first search traversal on the parse tree. This is possible only because SDT rules don't impose any specific order on evaluation until children attributes are computed before parents for a grammar having all synthesized attributes.

Otherwise, we would have to figure out the best suited plan to traverse through the parse tree and evaluate all the attributes in one or more traversals. For better understanding, we will move bottom up in left to right fashion for computing translation rules of our example.



Above diagram shows how semantic analysis could happen. The flow of information happens bottom-up and all the children attributes are computed before parents, as discussed above. Right hand side nodes are sometimes annotated with subscript 1 to distinguish between children and parent.

Additional Information

Synthesized Attributes are such attributes that depend only on the attribute values of children nodes.

Thus [E -> E+T { E.val = E.val + T.val }] has a synthesized attribute val corresponding to node E. If all the semantic attributes in an augmented grammar are synthesized, one depth first search traversal in any order is sufficient for semantic analysis phase.

Inherited Attributes are such attributes that depend on parent and/or siblings' attributes. Thus [Ep -> E+T { Ep.val = E.val + T.val, T.val = Ep.val }], where E & Ep are same production symbols annotated to differentiate between parent and child, has an inherited attribute val corresponding to node T.

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

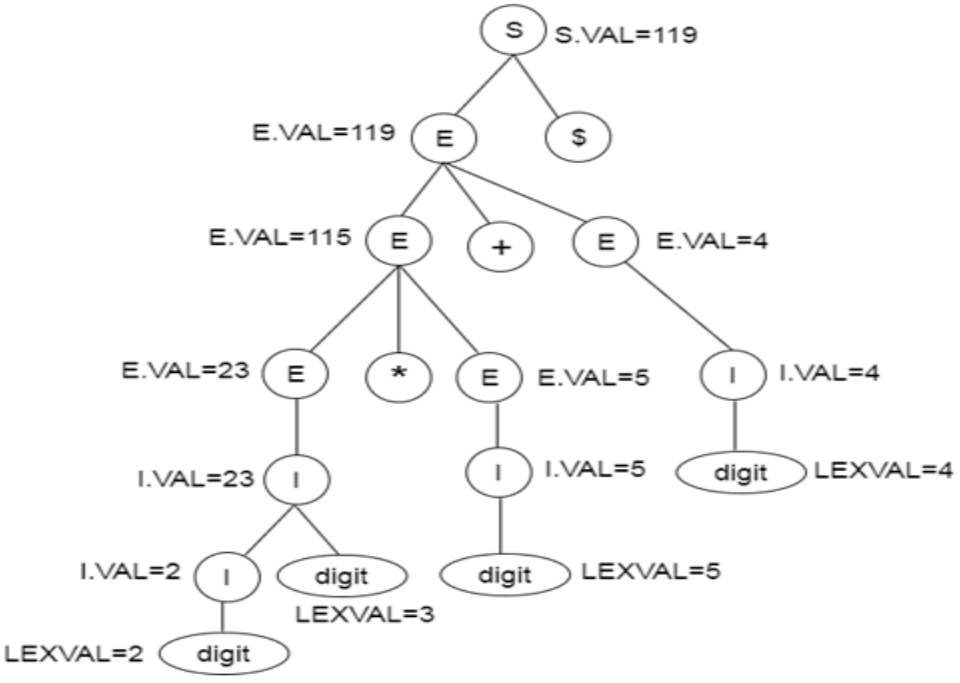
Example# 2

Given CFG

S→E
E→E+E | E*E | (E) | I
I→I digit
I→ digit

S#	Production	Semantic Rules
1	S→E $	{ Print E.Val }
2	E→E + E	{ E.Val = E.Val + E.Val }
3	E→E * E	{ E.Val = E.Val * E.Val }
4	E→(E)	{ E.Val = E.Val }
5	E→ I	{ E.Val = I.Val }
6	I→ I digit	{ I.Val = 10 * I.Val + LexVal }
7	I→ digit	{ I.Val = LexVal }

Parse tree for SDT: 23*5+4



Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

Example # 3

Write semantic rules which convert binary number to decimal number for the following CFG.

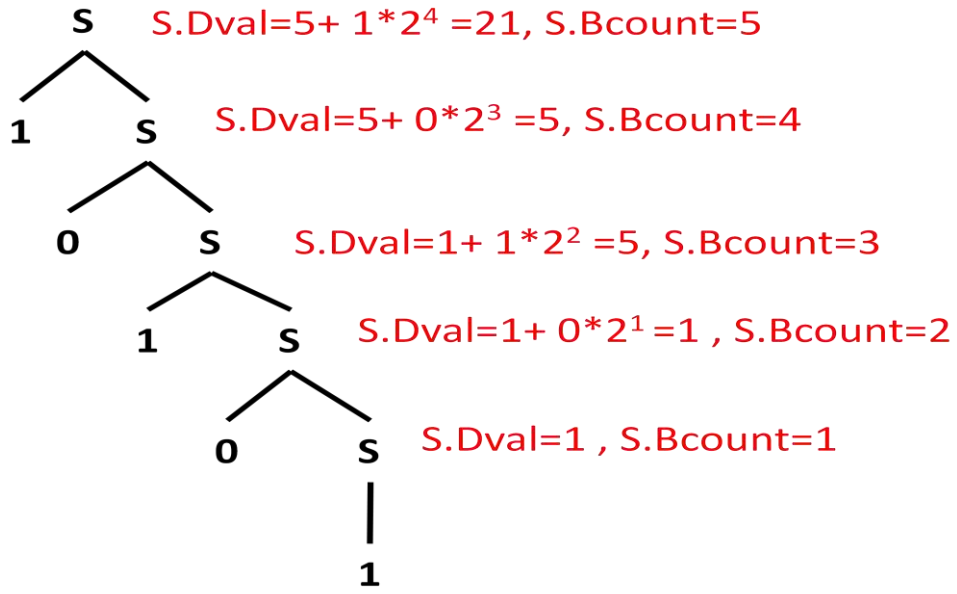
Given CFG

S→0S | 1S | 0 | 1

S#	Production	Semantic Rules
1	S→0S	S.DVal= S.Dval + 0*$2^{S.bcount}$, S.Bcount= S.bcount +1
2	S→1S	S.DVal= S.Dval + 1*$2^{S.bcount}$, S.Bcount= S.bcount +1
3	S→0	S.DVal=0 , S.Bcount=1
4	S→1	S.DVal=1 , S.Bcount=1

Parse tree for SDT: $(10101)_2=(21)_{10}$

$(10101)_2 = 1*2^4+0*2^3+1*2^2+0*2^1+1*2^0 = 16+0+4+0+1=(21)_{10}$



Example # 4

Construct a syntax-directed translation (SDT) scheme that translates arithmetic expressions from infix notation into postfix notation.

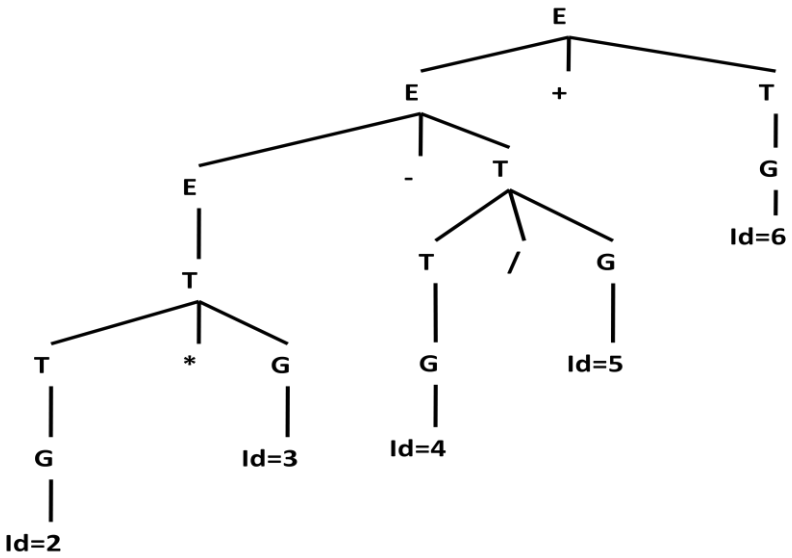
Given CFG

E→E-T | E+T |T
T→T*G | T/G |G
G→id

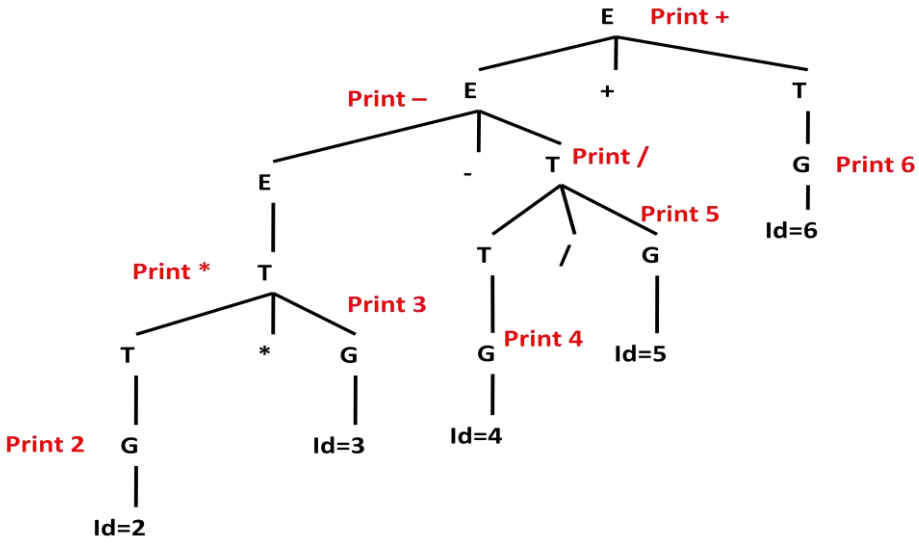
Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

S#	Production	Semantic Rules
1	E→E-T	Print -
2	E→E+T	Print +
3	E→T	{ }
4	T→T*G	Print *
5	T→T/G	Print /
6	T→G	{ }
7	G→id	Print id

Parse Tree (Infix Notation = 2*3 – 4 / 5 +6)



Parse Tree: (Postfix Notation = 2 3 * 4 5 / – 6 +)



Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

Example # 5

Construct a syntax-directed translation (SDT) scheme that translates arithmetic expressions from infix notation into postfix notation.

Given CFG

Expr→ Expr – Term | Expr +Term | Term
Term→ID

S#	Production Rules	Semantic Rules
1	Expr→ Expr –Term	Print -
2	Expr → Expr +Term	Print +
3	E→Term	{ }
4	Term→ID	Print ID.sval