Chapter 2

Syntax Directed Translation

LEARNING OBJECTIVES

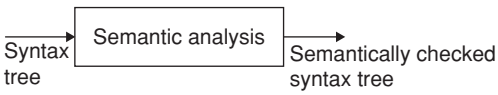
- *Syntax directed translation*
- *Syntax directed definition*
- *Dependency graph*
- *Constructing syntax trees for expressions*
- *Types of SDD's*
- *S-attributed definition*
- *L-attributed definitions*
- *Synthesized attributes on the parser*
- *Syntax directed translation schemes*
- *Bottom up evaluation of inherited attributes*

Syntax Directed Translation

To translate a programming language construct, a compiler may need to know the type of construct, the location of the first instruction, and the number of instructions generated…etc. So, we have to use the term 'attributes' associated with constructs.

An attribute may represent type, number of arguments, memory location, compatibility of variables used in a statement which cannot be represented by CFG alone.

So, we need to have one more phase to do this, i.e., 'semantic analysis' phase.



In this phase, for each production CFG, we will give some semantic rule.

Syntax directed translation scheme

A CFG in which a program fragment called output action (semantic action or semantic rule) is associated with each production is known as Syntax Directed Translation Scheme.

These semantic rules are used to

1. Generate intermediate code.
2. Put information into symbol table.
3. Perform type checking.
4. Issues error messages.

Notes:

1. Grammar symbols are associated with attributes.
2. Values of the attributes are evaluated by the semantic rules associated with production rules.

Notations for Associating Semantic Rules

There are two techniques to associate semantic rules:

Syntax directed definition (SDD) It is high level specification for translation. They hide the implementation details, i.e., the order in which translation takes place.

Attributes + CFG + Semantic rules = Syntax directed definition (SDD).

Translation schemes These schemes indicate the order in which semantic rules are to be evaluated. This is an input and output mapping.

Syntax Directed Definitions

A SDD is a generalization of a CFG in which each grammar symbol is associated with a set of attributes.

There are two types of set of attributes for a grammar symbol.

1. Synthesized attributes
2. Inherited attributes

Each production rule is associated with a set of semantic rules.

Semantic rules setup dependencies between attributes which can be represented by a dependency graph.

The dependency graph determines the evaluation order of these semantic rules.

Evaluation of a semantic rule defines the value of an attribute. But a semantic rule may also have some side effects such as printing a value.

Attribute grammar: An attribute grammar is a syntax directed definition in which the functions in semantic rules 'cannot have side effects'.

Annotated parse tree: A parse tree showing the values of attributes at each node is called an annotated parse tree.

The process of computing the attribute values at the nodes is called annotating (or decorating) of the parse tree.

In a SDD, each production $A \rightarrow \propto$ is associated with a set of semantic rules of the form:

$b = f(c_1, c_2, \ldots c_n)$ where

$f : A$ function

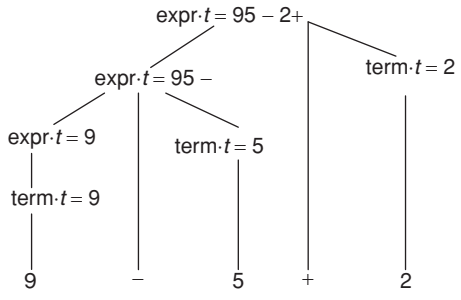
b can be one of the following:

b is a 'synthesized attribute' of A and $c_1, c_2, \ldots c_n$ are attributes of the grammar symbols in $A \rightarrow \propto$.

The value of a 'synthesized attribute' at a node is computed from the value of attributes at the children of that node in the parse tree.

Example:

Production	Semantic Rule
expr → expr1 + term	expr.t: = expr1.t||term.t||'+'
expr → expr1 – term	expr.t: = expr1.t||term.t||'–'
expr → term	expr.t: = term.t
term → 0	term.t: = '0'
term → 1	term.t: = '1'
⋮	⋮
term → 9	term.t: = '9'



b is an 'inherited attribute' of one of the grammar symbols on the right side of the production.

An 'inherited attribute' is one whose value at a node is defined in terms of attributes at the parent and/or siblings of that node. It is used for finding the context in which it appears.

Example: An inherited attribute distributes type information to the various identifiers in a declaration.

For the grammar

$D \rightarrow$ TL

$T \rightarrow$ int

$T \rightarrow$ real

$L \rightarrow L_1$, id

$L \rightarrow$ id

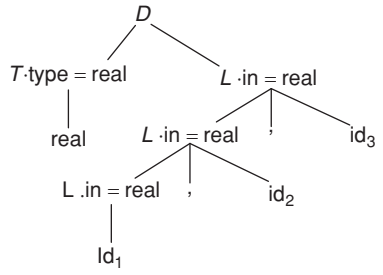
That is, The keyword int or real followed by a list of identifiers.

In this T has synthesized attribute type: T.type. L has an inherited attribute in L.in

Rules associated with L call for procedure add type to the type of each identifier to its entry in the symbol table.

Production	Semantic Rule
$D \rightarrow$ TL	L.in = T.type
$T \rightarrow$ int	T.type = integer
$T \rightarrow$ real	T.type = real
$L \rightarrow L_1$, id	addtype L_1.in = L.in(id.entry, L.in)
$L \rightarrow$ id	addtype (id.entry, L.in)

The annotated parse tree for the sentence real id_1, id_2, id_3 is shown below:



Synthesized Attribute

The value of a synthesized attribute at a node is computed from the value of attributes at the children of that node in a parse tree. Consider the following grammar:

$L \rightarrow E_n$

$E \rightarrow E_1 + T$

$E \rightarrow T$

$T \rightarrow T_1 * F$

$T \rightarrow F$

$F \rightarrow (E)$

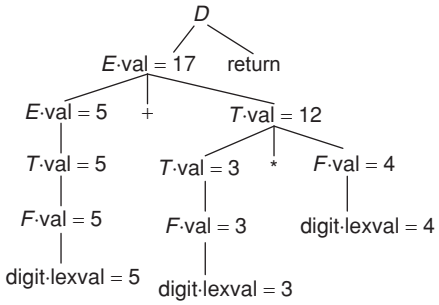
$F \rightarrow$ digit.

Let us consider synthesized attribute value with each of the non-terminals E, T and F.

Token digit has a synthesized attribute lexical supplied by lexical analyzer.

Production	Semantic Rule
$L \rightarrow E_n$	print (E.val)
$E \rightarrow E_1 + T$	E.val: = E_1.val + T.val
$E \rightarrow T$	E.val: = T_1.val
$T \rightarrow T_1 * F$	T.val: = T_1.val*F.val
$T \rightarrow F$	T.val: = F.val
$F \rightarrow (E)$	F.val: = E.val
$F \rightarrow$ digit	F.val: = digit.lexval

The Annotated parse tree for the expression 5 + 3 * 4 is shown below:



Example 1: Consider an example, which shows semantic rules for Infix to posfix translation:

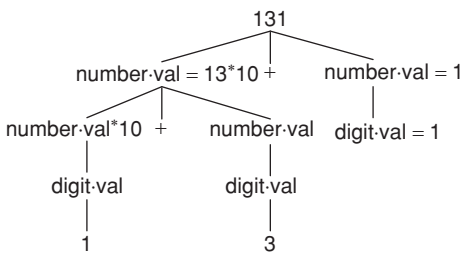
Production	Semantic Rules
expr → expr1 + term	expr.t: = expr1.t||term.t||'+'
expr → expr1 – term	expr.t: = expr1.t||term.t ||'–'
expr → term	expr.t: = term.t
term → 0	term.t: = '0'
⋮	⋮
term→ 9	term.t := '9'

Example 2: Write a SDD for the following grammar to determine number.val.

number → number digit
digit→ 0|1|…9

$$\begin{cases} \text{digit.val} := \text{'0'} \\ \text{digit.val} := \text{'1'} \\ \vdots \\ \text{digit.val} = \text{'9'} \end{cases}$$

number.val:=number.val * 10 + digit.val
Annotated tree for 131 is

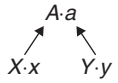


Dependency Graph

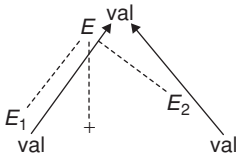
The interdependencies among the attributes at the nodes in a parse tree can be depicted by a directed graph called dependency graph.

- Synthesized attributes have edges pointing upwards.
- Inherited attributes have edges pointing downwards and/or sidewise.

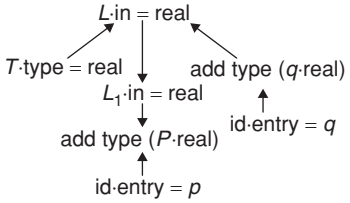
Example 1: $A.a := f(X.x, Y.y)$ is a semantic rule for $A \rightarrow XY$. For each semantic rule that consists of a procedure call:



Example 2:



Example 3: real p, q;



Evaluation order

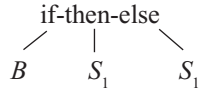
A topological sort of directed acyclic graph is an ordering $m_1, m_2, \ldots m_k$ of nodes of the graph S. t edges go from nodes earlier in the ordering to later nodes.

$m_i \rightarrow m_j$ means m_i appears before m_j in the ordering.

If $b := f(c_1, c_2, \ldots, c_k)$, the dependent attributes $c_1, c_2, \ldots c_k$ are available at node before f is evaluated.

Abstract syntax tree

It is a condensed form of parse tree useful for representing language constructs.
Example



Constructing Syntax Trees for Expressions

Each node in a syntax tree can be implemented as a record with several fields.

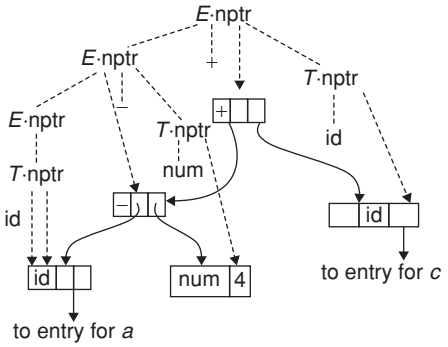
In the node for an operator, one field identifies the operator and the remaining fields contain pointers to the nodes for the operands.

1. mknode (op, left, right)
2. mkleaf (id, entry). Entry is a pointer to symbol table.
3. mkleaf (num, val)

Example:

Production	Semantic Rules
$E \to E_1 + T$	E.nptr := mknode ('+', E_1.nptr, T.nptr)
$E \to E_1 - T$	E.nptr := mknode ('–', E_1.nptr, T.nptr)
$E \to T$	E.nptr := T.nptr
$T \to (E)$	T.nptr := E.nptr
$T \to$ id	T.nptr := mkleaf(id, id.entry)
$T \to$ num	T.nptr := mkleaf(num, num.val)

Construction of a syntax tree for $a - 4 + c$



Types of SDD's

Syntax Directed definitions (SDD) are used to specify syntax directed translations. There are two types of SDD.

1. S-Attributed Definitions
2. L-Attributed Definitions.

S-attributed definitions

- Only synthesized attributes used in syntax direct definition.
- S-attributed grammars interact well with *LR* (*K*) parsers since the evaluation of attributes is bottom-up. They do not permit dependency graphs with cycles.

L-attributed definitions

- Both inherited and synthesized attribute are used.
- L-attributed grammar support the evaluation of attributes associated with a production body, dependency–graph edges can go from left to right only.
- Each S-attributed grammar is also a L-attributed grammar.
- L-attributed grammars can be incorporated conveniently in top down parsing.
- These grammars interact well with *LL* (*K*) parsers (both table driven and recursive descent).

Synthesized Attributes on the Parser Stack

A translator for an S-attributed definition often be implemented with *LR* parser generator. Here the stack is implemented by a pair of array state and val.

- Each state entry is pointed to a *LR* (1) parsing table.
- Each val[i] holds the value of the attributes associated with the node. For $A \to xyz$, the stack will be:

	State	Val
Top →	Z	Z.z
	Y	Y.y
	X	X.x

Example: Consider the following grammar:

$S \to E\$$	{print(E.val)}
$E \to E + E$	{E.val := E.val + E.val}
$E \to E^*E$	{E.val := E.val * E.val}
$E \to (E)$	{E.val := E.val}
$E \to I$	{I.val := I.val * 10 + digit}
$I \to I$ digit	
$I \to$ digit	{I.val := digit}

Implementation

$S \to E\$$	print (val [top])
$E \to E + E$	val[ntop] := val[top] + val[top-2]
$E \to E^*E$	val[ntop] := val[top] * val[top-2]
$E \to (E)$	val[ntop] := val[top-1]
$E \to I$	val[ntop] := val[top]
$I \to I$ digit	val[ntop] := 10*val[top] + digit
$I \to$ digit	val[ntop] := digit

L-attributed Definitions

A syntax directed definition is *L*-attributed if each inherited attribute of X_j, $1 \le j \le n$, on the right side of $A \to X_1 X_2 \ldots X_n$, depends only on

1. The attributes of symbols $X_1, X_2, \ldots, Xj_{-1}$ to the left of X_j in the production.
2. The inherited attributes of A.

Every S-attributed definition is L-attributed, because the above two rules apply only to the inherited attributes.

Syntax Directed Translation Schemes

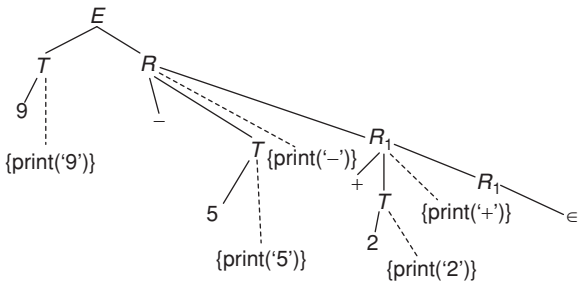
A translation scheme is a CFG in which attributes are associated with grammar symbols and semantic actions are enclosed between braces { } are inserted within the right sides of productions.

Example: $E \to TR$

$R \to$ op T {print (op.lexeme)} $R_1 | \in$

$T \to$ num {print (num.val)}

Using this, the parse tree for 9 – 5 + 2 is



If we have both inherited and synthesized attributes then we have to follow the following rules:

1. An inherited attribute for a symbol on the right side of a production must be computed in an action before that symbol.
2. An action must not refer to a synthesized attribute of a symbol on the right side of the action.
3. A synthesized attribute for the non–terminal on the left can only be computed after all attributes it references, have been computed.

Note: In the implementation of L-attributed definitions during predictive parsing, instead of syntax directed translations, we will work with translation schemes.

Eliminating left recursion from translation scheme

Consider following grammar, which has left recursion

$E \rightarrow E + T$ {print ('+') ;}

When transforming the grammar, treat the actions as if they were terminal symbols. After eliminating recursion from the above grammar.

$E \rightarrow TR$
$R \rightarrow +T$ {print ('+');} R
$R \rightarrow \in$

Bottom-up Evaluation of Inherited Attributes

- Using a bottom up translation scheme, we can implement any L-attributed definition based on LL (1) grammar.
- We can also implement some of L-attributed definitions based on LR (1) using bottom up translations scheme.
  - The semantic actions are evaluated during the reductions.
  - During the bottom up evaluation of S-attributed definitions, we have a parallel stack to hold synthesized attributes.

Where are we going to hold inherited attributes?

We will convert our grammar to an equivalent grammar to guarantee the following:

- All embedding semantic actions in our translation scheme will be moved to the end of the production rules.
- All inherited attributes will be copied into the synthesized attributes (may be new non-terminals).

Thus we will evaluate all semantic actions during reductions, and we find a place to store an inherited attribute. The steps are

1. Remove an embedding semantic action S_i, put new non-terminal M_i instead of that semantic action.
2. Put S_i into the end of a new production rule $M_i \rightarrow \in$.
3. Semantic action S_i will be evaluated when this new production rule is reduced.
4. Evaluation order of semantic rules is not changed. i.e., if

$$A \rightarrow \{S_1\}\ X_1\{S_2\}X_2 \ldots \{S_n\}X_n$$

After removing embedding semantic actions:

$A \rightarrow M_1X_1M_2X_2 \ldots M_nX_n$
$M_1 \rightarrow \in \{S_1\}$
$M_2 \rightarrow \in \{S_2\}$
$\vdots$
$M_n \rightarrow \in \{S_n\}$

For example,

$E \rightarrow TR$
$R \rightarrow +T$ {print ('+')} R_1
$R \rightarrow \in$
$T \rightarrow$ id {print (id.name)}
$\Downarrow$ remove embedding semantic actions
$E \rightarrow TR$
$R \rightarrow +TMR_1$
$R \rightarrow \in$
$T \rightarrow$ id {print (id.name)}
$M \rightarrow \in$ {print ('+')}

Translation with inherited attributes

Let us assume that every non-terminal A has an inherited attribute $A.i$ and every symbol X has a synthesized attribute $X.s$ in our grammar.

For every production rule $A \rightarrow X_1, X_2 \ldots X_n$, introduce new marker non-terminals

$M_1, M_2, \ldots M_n$ and replace this production rule with $A \rightarrow M_1X_1M_2X_2 \ldots M_nX_n$

The synthesized attribute of X_i will not be changed.

The inherited attribute of X_i will be copied into the synthesized attribute of M_i by the new semantic action added at the end of the new production rule

$$M_i \rightarrow \in$$

Now, the inherited attribute of X_i can be found in the synthesized attribute of M_i.

$A \rightarrow \{B.i = f_1(\ldots)\ B\ \{\ c.i = f_2(\ldots)\}\ c\ \{A.s = f_3(\ldots)\}$

$\Downarrow$

$A \rightarrow \{M_1.i = f_1(\ldots)\}\ M_1\ \{B.i = M_1.s\}\ B\ \{M_2.i = f_2(\ldots)\}M_2$
$\{c.i = M_2.S\}\ c\ \{A.s = f_3(\ldots)\}$
$M_1 \rightarrow \in\ \{M_1.s = M_1.i\}$
$M_2 \rightarrow \in\ \{M_2.s = M_2.i\}$

EXERCISES

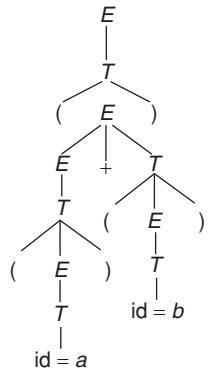
Practice Problems I

Directions for questions 1 to 13: Select the correct alternative from the given choices.

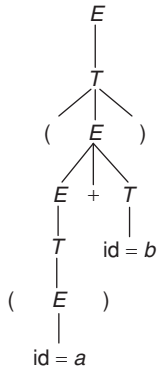
1. The annotated tree for input ((*a*) + (*b*)), for the rules given below is

Production	Semantic Rule
$E \to E + T$	$ $ = mknode ('+', $1, $3)
$E \to E - T$	$ $ = mknode ('–', $1, $3)
$E \to T$	$ $ = $1;
$T \to (E)$	$ $ = $2;
$T \to$ id	$ $ = mkleaf (id, $1)
$T \to$ num	$ $ = mkleaf (num, $1)

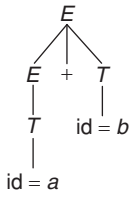
(A)



(B)



(C)



(D) None of these

2. Let synthesized attribute val give the value of the binary number generated by S in the following grammar.

$S \to L\ L$
$S \to L$
$L \to LB$
$L \to B$
$B \to 0$
$B \to 1$

Input 101.101, S.val = 5.625
use synthesized attributes to determine S.val
Which of the following are true?

(A) $S \to L_1.L_2$ {S.val = L_1.val + L_2.val/ (2**L_2.bits)
$|L$ {S.val = L.val; S.bits = L.bits}

(B) $L \to L_1\ B$ {L.val = L_1.val*2 + B.val;
L.bits = L_1.bits + 1}
$|B$ {L.val = B.val; L.bits = 1}

(C) $B \to 0$ {B.val = 0}
$|1$ {B.val = 1}

(D) All of these

3. Which of the following productions with translation rules converts binary number representation into decimal.

(A)

Production	Semantic Rule
$B \to 0$	B.trans = 0
$B \to 1$	B.trans = 1
$B \to B_0$	B_1.trans = B_2.trans*2
$B \to B_1$	B_1.trans = B_2.trans * 2 + 1

(B)

Production	Semantic Rule
$B \to 0$	B.trans = 0
$B \to B_0$	B_1.trans = B_2.trans*4

(C)

Production	Semantic Rule
$B \to 1$	B.trans = 1
$B \to B_1$	B_1.trans = B_2.trans*2

(D) None of these

4. The grammar given below is

Production	Semantic Rule
$A \to$ LM	L.i := l(A. i)
	M.i := m(L.s)
	A.s := f(M.s)
A → QR	R.i := r(A.i)
	Q.i := q(R.s)
	A.s := f(Q.s)

(A) A L-attributed grammar
(B) Non-L-attributed grammar
(C) Data insufficient
(D) None of these

5. Consider the following syntax directed translation:

$S \to aS$ {$m := m + 3$; print (m);}
$|bS$ {m: = m*2; print (m) ;}
$|\in$ {m: = 0 ;}

A shift reduce parser evaluate semantic action of a production whenever the production is reduced.

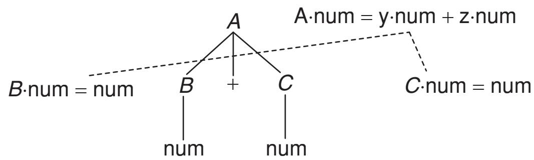
If the string is = *a a b a b b* then which of the following is printed?

(A) 0 0 3 6 9 12 (B) 0 0 0 3 6 9 12
(C) 0 0 0 3 6 9 12 15 (D) 0 0 3 9 6 12

6. Which attribute can be evaluated by shift reduce parser that execute semantic actions only at reduce moves but never at shift moves?

(A) Synthesized attribute (B) Inherited attribute
(C) Both (a) and (b) (D) None of these

7. Consider the following annotated parse tree:



Which of the following is true for the given annotated tree?

(A) There is a specific order for evaluation of attribute on the parse tree.
(B) Any evaluation order that computes an attribute '*A*' after all other attributes which '*A*' depends on, is acceptable.
(C) Both (A) and (B)
(D) None of these.

Common data for questions 8 and 9: Consider the following grammar and syntax directed translation.

$E \to E + T$	$E_1.\text{val} = E_2.\text{val} + T.\text{val}$
$E \to T$	$E.\text{val} = T.\text{val}$
$T \to T*P$	$T_1.\text{val} = T_2.\text{val} * P.\text{val} * P.\text{num}$
$T \to P$	$T.\text{val} = P.\text{val} * P.\text{num}$
$P \to (E)$	$P.\text{val} = E.\text{val}$
$P \to 0$	$P.\text{num} = 1$
	$P.\text{val} = 2$
$P \to 1$	$P.\text{num} = 2$
	$P.\text{val} = 1$

8. What is E.val for string 1*0?

(A) 8 (B) 6
(C) 4 (D) 12

9. What is the E.val for string 0 * 0 + 1?

(A) 8 (B) 6
(C) 4 (D) 12

10. Consider the following syntax directed definition:

Production	**Semantic Rule**
$S \to b$	$S.x = 0$ $S.y = 0$
$S \to S_1\ I$	$S.x = S_1.x + I.dx$ $S.y = S_1.y + I.dy$
$I \to$ east	$I.dx = 1$ $I.dy = 0$
$I \to$ north	$I.dx = 0$ $I.dy = 1$
$I \to$ west	$I.dx = -1$ $I.dy = 0$
$I \to$ south	$I.dx = 0$ $I.dy = -1$

If Input = begin east south west north, after evaluating this sequence what will be the value of $S.x$ and $S.y$?

(A) (1, 0) (B) (2, 0)
(C) (−1, −1) (D) (0, 0)

11. What will be the values $s.x$, $s.y$ if input is 'begin west south west'?

(A) (–2, –1)
(B) (2, 1)
(C) (2, 2)
(D) (3, 1)

12. Consider the following grammar:

$S \to E$	$S.\text{val} = E.\text{val}$
	$E.\text{num} = 1$
$E \to E*T$	$E_1.\text{val} = 2 * E_2.\text{val} + 2 * T.\text{val}$
	$E_2.\text{num} = E_1.\text{num} + 1$
	$T.\text{num} = E_1.\text{num} + 1$
$E \to T$	$E.\text{val} = T.\text{val}$
	$T.\text{num} = E.\text{num} + 1$
$T \to T + P$	$T_1.\text{val} = T_2.\text{val} + P.\text{val}$
	$T_2.\text{num} = T_1.\text{num} + 1$
	$P.\text{num} = T_1.\text{num} + 1$
$T \to P$	$T.\text{val} = P.\text{val}$
	$P.\text{num} = T.\text{num} + 1$
$P \to (E)$	$P.\text{val} = E.\text{val}$
$P \to i$	$\begin{Bmatrix} E.\text{num} = P.\text{num} \\ P.\text{val} = I\ \lvert P.\text{num} \end{Bmatrix}$

Which attributes are inherited and which are synthesized in the above grammar?

(A) Num attribute is inherited attribute. Val attribute is synthesized attribute.
(B) Num is synthesized attribute. Val is inherited attribute.
(C) Num and val are inherited attributes.
(D) Num and value are synthesized attributes.

13. Consider the grammar with the following translation rules and E as the start symbol.

$E \to E_1@T\ \{E.\text{value} = E_1.\text{value}*T.\text{value}\}$

$\quad\quad |T\ \{E.\text{value} = T.\text{value}\}$

$T \to T_1$ and $F\ \{T.\text{value} = T_1.\text{value} + F.\text{value}\}$

$\quad\quad |F\ \{T.\text{value} = F.\text{value}\}$

$F \to$ num $\quad \{F.\text{value} = \text{num.value}\}$

Compute E.value for the root of the parse tree for the expression: 2 @ 3 and 5 @ 6 and 4

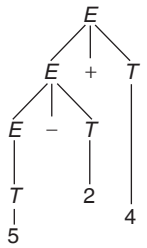
(A) 200 (B) 180
(C) 160 (D) 40

Practice Problems 2

Directions for questions 1 to 10: select the correct alternative from the given choices.

1. Consider the following Tree:

Production	Meaning
$E \rightarrow E_1 + T$	$E.t = E_1.t * T.t$
$E \rightarrow E_1 - T$	$E.t = E_1.t + T.t$
$E \rightarrow T$	$E.t = T.t$
$t \rightarrow 0$	$T.t =$ '0'
$t \rightarrow 5$	$T.t =$ '5'
$t \rightarrow 2$	$T.t =$ '2'
$t \rightarrow 4$	$T.t =$ '4'



After evaluation of the tree the value at the root will be:
(A) 28 (B) 32
(C) 14 (D) 7

2. The value of an inherited attribute is computed from the values of attributes at the _____
(A) Sibling nodes (B) Parent of the node
(C) Children node (D) Both (A) and (B)

3. Consider an action translating expression:

expr → expr + term {print ('+')}
expr → expr – term {print ('–')}
expr → → term
term → 1 {print ('1')}
term → 2 {print ('2')}
term → 3 {print ('3')}

Which of the following is true regarding the above translation expression?
(A) Action translating expression represents infix notation.
(B) Action translating expression represents prefix notation.
(C) Action translating expression represents postfix notation.
(D) None of these

4. In the given problem, what will be the result after evaluating 9 – 5 + 2?
(A) + – 9 5 2 (B) 9 – 5 + 2
(C) 9 5 – 2+ (D) None of these

5. In a syntax directed translation, if the value of an attribute node is a function of the values of attributes of children, then it is called:
(A) Synthesized attribute (B) Inherited attribute
(C) Canonical attributes (D) None of these

6. Inherited attribute is a natural choice in:
(A) Keeping track of variable declaration
(B) Checking for the correct use of L-values and R-values.
(C) Both (A) and (B)
(D) None of these

7. Syntax directed translation scheme is desirable because
(A) It is based on the syntax
(B) Its description is independent of any implementation.
(C) It is easy to modify
(D) All of these

8. A context free grammar in which program fragments, called semantic actions are embedded within right side of the production is called,
(A) Syntax directed translation
(B) Translation schema
(C) Annotated parse tree
(D) None of these

9. A syntax directed definition specifies translation of construct in terms of:
(A) Memory associated with its syntactic component
(B) Execution time associated with its syntactic component
(C) Attributes associated with its syntactic component
(D) None of these

10. If an error is detected within a statement, the type assigned to the Statement is:
(A) Error type (B) Type expression
(C) Type error (D) Type constructor

Previous Years' Questions

Common data for questions 1 (A) and 1 (B): Consider the following expression grammar. The semantic rules for expression evaluation are stated next to each grammar production: **[2005]**

$E \rightarrow$ number	E.val = number.val
$\mid E$ '+' E	$E^{(1)}$.val = $E^{(2)}$.val + $E^{(3)}$.val
$\mid E \rightarrow E$	$E^{(1)}$.val = $E^{(2)}$.val × $E^{(3)}$.val

1. (A) The above grammar and the semantic rules are fed to a yacc tool (which is an LALR (1) parser generator) for parsing and evaluating arithmetic expressions. Which one of the following is true about the action of yacc for the given grammar?
(A) It detects recursion and eliminates recursion
(B) It detects reduce-reduce conflict, and resolves

(C) It detects shift-reduce conflict, and resolves the conflict in favor of a shift over a reduce action.
(D) It detects shift-reduce conflict, and resolves the conflict in favor of a reduce over a shift action.

(B) Assume the conflicts in Part (A) of this question are resolved and an LALR (1) parser is generated for parsing arithmetic expressions as per the given grammar. Consider an expression 3 × 2 + 1. What precedence and associativity properties does the generated parser realize?
(A) Equal precedence and left associativity; expression is evaluated to 7
(B) Equal precedence and right associativity; expression is evaluated to 9
(C) Precedence of '×' is higher than that of '+', and both operators are left associative; expression is evaluated to 7
(D) Precedence of '+' is higher than that of '×', and both operators are left associative; expression is evaluated to 9

2. In the context of abstract-syntax-tree (AST) and control-flow-graph (CFG), which one of the following is TRUE? **[2015]**
(A) In both AST and CFG, let node N_2 be the successor of node N_1. In the input program, the code corresponding to N_2 is present after the code corresponding to N_1.
(B) For any input program, neither AST nor CFG will contain a cycle.
(C) The maximum number of successors of a node in an AST and a CFG depends on the input program.
(D) Each node in AST and CFG corresponds to at most one statement in the input program.

3. Consider the following Syntax Directed Translation Scheme (SDTS), with non-terminals {S, A} and terminals {**a, b**}. **[2016]**

$S \rightarrow aA$ { print 1 }
$S \rightarrow a$ { print 2 }
$A \rightarrow Sb$ { print 3 }

Using the above SDTS, the output printed by a bottom-up parser, for the input **aab** is:
(A) 1 3 2 (B) 2 2 3
(C) 2 3 1 (D) syntax error

4. Which one of the following grammars is free from *left recursion*? **[2016]**
(A) $S \rightarrow AB$
$A \rightarrow Aa/b$
$B \rightarrow c$
(B) $S \rightarrow Ab/Bb/c$
$A \rightarrow Bd/\varepsilon$
$B \rightarrow e$
(C) $S \rightarrow Aa/B$
$A \rightarrow Bb/Sc/\varepsilon$
$B \rightarrow d$
(D) $S \rightarrow Aa/Bb/c$
$A \rightarrow Bd/\varepsilon$
$B \rightarrow Ae/\varepsilon$

Answer Keys

Exercises

Practice Problems 1

1. A **2.** D **3.** A **4.** B **5.** A **6.** A **7.** B **8.** C **9.** B **10.** D
11. A **12.** A **13.** C

Practice Problems 2

1. A **2.** D **3.** C **4.** C **5.** A **6.** C **7.** D **8.** B **9.** C **10.** C

Previous Years' Questions

1. (a) C (b) B **2.** C **3.** C **4.** A