Concepts Introduced in Chapter 5

- Syntax-Directed Definitions
- Translation Schemes
- Synthesized Attributes
- Inherited Attributes
- Dependency Graphs

Syntax-Directed Translation

- Uses a grammar to direct the translation. The grammar defines the syntax of the input language.
- Semantic rules or actions are associated with grammar productions.
- Attributes are values associated with grammar symbols representing programming language constructs. These values are computed by semantic rules associated with the grammar productions.

Notations for Associating Semantic Rules with Grammar Productions

- Syntax-Directed Definitions
- Translation Schemes

Syntax-Directed Definitions

- High-level specifications for translation.
- User does not have to explicitly specify the order in which the translation occurs.
- Each production in the grammar $A \rightarrow \alpha$ has associated with it a set of semantic rules of the form

$$b := f\ (c_1, c_2, ..., c_k)$$

where f is a function, $c_1, ..., c_k$ are attributes of the grammar symbols of the production, and b is an attribute associated with A or one of the symbols of the rhs of the production.

Attributes

- Synthesized Attribute
  - Value is determined from the attribute values of the children of the node.
  - Used in YACC.
- Inherited Attribute
  - Value at a node in the parse tree is defined in terms of the attributes of the parent and/or siblings of that node.

Syntax-Directed Definition Example

- A syntax-directed definition hides many implementation details.

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

Syntax-Directed Definitions (cont.)

Annotated Parse Tree

- Parse tree showing the values of the attributes at each node.

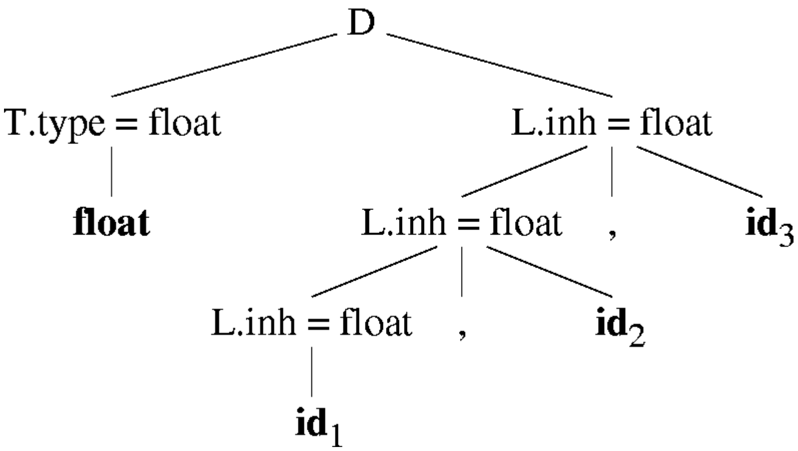
S-Attributed Definition

- A syntax-directed definition that uses synthesized attributes exclusively.

Example of Using Inherited Attributes

Production	Semantic Rules
D → T L	L.inh = T.type
T → **int**	T.type := integer
T → **float**	T.type := float
L → L_1 , **id**	L_1.inh := L.inh addtype(**id**.entry, L.inh)
L → **id**	addtype(**id**.entry, L.inh)

Parse Tree with Inherited Attribute *inh* at Each Node Labeled *L*



Dependency Graphs

- The interdependencies between the inherited and synthesized attributes at the nodes in a parse tree can be depicted by a directed graph called a dependency graph.

L-Attributed Definitions

- A syntax-directed definition is L-attributed if each inherited attribute of X_j, $1 \leq j \leq n$ on the rhs of $A \rightarrow X_1 X_2 \ldots X_n$ depends only on
  - the attributes of the symbols $X_1, X_2, \ldots, X_{j-1}$ to the left of X_j
  - the inherited attributes of A

A Non-L-Attributed Syntax-Directed Definition

PRODUCTION	SEMANTIC RULES
$A \rightarrow L\ M$	L.i = l(A.i) M.i = m(L.s) A.s = f(M.s)
$A \rightarrow Q\ R$	R.i = r(A.i) Q.i = q(R.s) A.s = f(Q.s)

Translation Schemes

- Indicate the order in which translation takes place.
- Are context-free grammars in which attributes are associated with grammar symbols and semantic actions are enclosed between { } and are inserted within the right sides of productions.

Example Translation Scheme

$E \rightarrow T\ R$

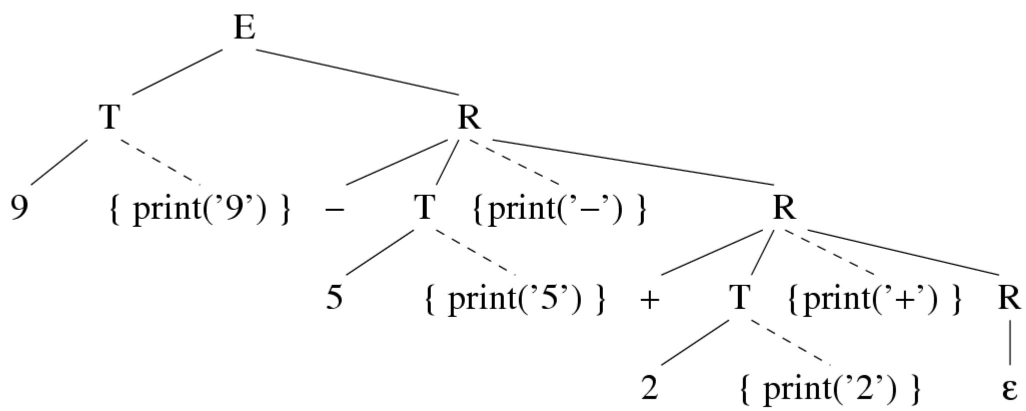
$R \rightarrow +\ T\ \{\ print('+')\ \}\ R_1$

$R \rightarrow -\ T\ \{\ print('-')\ \}\ R_1$

$R \rightarrow \varepsilon$

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

Parse Tree for 9-5+2 Showing Actions



Requirements for Evaluating Attributes in a Translation Scheme

- An inherited attribute for a symbol on the rhs of a production must be computed in an action before that symbol is parsed.
- A synthesized attribute for a symbol cannot be referenced in an action before that symbol is parsed.
- A synthesized attribute for the nonterminal on the left can only be computed after all attributes it references have been computed.

Syntax-Directed Construction of Syntax Trees

- Can use a syntax tree as an intermediate step to decouple parsing from intermediate code generation.
- Advantages
  - A grammar suitable for parsing may not reflect the natural hierarchical structure of the constructs.
  - A parsing method constrains the order in which nodes in a parse tree are considered.

Example of Syntax-Directed Construction of a Syntax Tree

Production	Semantic Rule
$E \rightarrow E_1 + T$	E.nptr := mknode('+', E_1.nptr, T.nptr)
$E \rightarrow E_1 - T$	E.nptr := mknode('−', E_1.nptr, T.nptr)
$E \rightarrow T$	E.nptr := T.nptr
$T \rightarrow (E)$	T.nptr := E.nptr
T→id	T.nptr := mkleaf(id, id.entry)
T→num	T.nptr := mkleaf(num, num.val)

Example of Syntax-Directed Construction of a Syntax Tree (cont.)

See Fig 5.11.

The syntax tree is constructed bottom-up.

```
1. p1 := new Leaf(id, entry-a);
2. p2 := new Leaf(num, 4);
3. p3 := new Node('−', p1, p2);
4. p4 := new Leaf(id, entry-c);
5. p5 := new Node('+', p3, p4);
```

Synthesized Attributes on the Parser Stack

- LR parser generator can easily support synthesized attributes.
- Extra fields in the parser stack can be used to hold the values of synthesized attributes.

Syntax-Directed Translation with YACC

Production	Semantic Action
1. S → E	{ printf("%d\n", $1); }
2. E → E + E	{ $$ = $1 + $3; }
3. E → E * E	{ $$ = $1 * $3; }
4. E → (E)	{ $$ = $2; }
5. E → I	{ $$ = $1; }
6. I → I digit	{ $$ = 10 * $1 + $2 – '0'; }
7. I → digit	{ $$ = $1 – '0'; }

Syntax-Directed Trans. with YACC (cont.)

	State	input	val	Production Used
1.	-	23*5+4$	-	
2.	2	3*5+4$	-	
3.	I	3*5+4$	2	I → digit
4.	I3	*5+4$	2	
5.	I	*5+4$	23	I → I digit
6.	E	*5+4$	23	E → I
7.	E*	5+4$	23	
8.	E*5	5+4$	23	
9.	E*I	+4$	23, 5	I → digit

Syntax-Directed Translation with YACC (cont.)

	State	input	val	Production Used
10.	E*E	+4$	23, 5	E → I
11.	E	+4$	115	E → E * E
12.	E+	4$	115	
13.	E+4	$	115	
14.	E+I	$	115, 4	I → digit
15.	E+E	$	115, 4	E → I
16.	E	$	119	E → E + E
17.	E$	-	119	
18.	S	-	-	S → E