Attributes

- Information associated with a grammar symbol
- Computed using semantic rules associated with grammar rules

Section 5.1

Example:

PRODUCTION	SEMANTIC RULES
$L \rightarrow E \setminus n$	`print E.val`
$E \rightarrow E_1 + T$	`E.val = E1.val + T.val`
$E \rightarrow T$	`E.val = T.val`
$T \rightarrow T_1 * F$	`T.val = T1.val * F.val`
$T \rightarrow F$	`T.val = F.val`
$F \rightarrow (E)$	`F.val = E.val`
$F \rightarrow$ **num**	`F.val = num.val`

Attributes

Synthesized attribute: attribute of a node (non-terminal) that depends on the value of attributes of children nodes in the parse tree

Inherited attribute: attribute of a node (non-terminal) that depends on the value of attributes of siblings and parent node in the parse tree

Example:

PRODUCTION	SEMANTIC RULES
$D \rightarrow TL$	`L.in = T.type`
$T \rightarrow \textbf{int}$	`T.type = INT`
$T \rightarrow \textbf{float}$	`T.type = FLOAT`
$L \rightarrow L_1, \textbf{id}$	`L1.in = L.in`
	`addtype(L.in, id.entry)`
$L \rightarrow \textbf{id}$	`addtype(L.in, id.entry)`

Syntax-directed definitions

Definition: a CFG where each grammar production $A \rightarrow \alpha$ is associated with a set of semantic rules of the form

```
b = f(c1, c2, ..., ck);
```

where:

`b` is a synthesized attribute of A or an inherited attribute of one of the grammar symbols in α

`c1, c2, ...` are attributes of the symbols used in the production

Translation scheme: CFG along with semantic rules inserted at appropriate positions in the RHS of each grammar production

Dependency graph

Directed graph showing the dependencies between attributes at various nodes in the parse tree

Algorithm:

```
for each node $n$ in the parse tree
    for each attribute $a$ of the grammar symbol at $n$
        construct a node in the dependency graph for $a$
for each node $n$ in the parse tree
    for each semantic rule b=f(c1,...,ck) associated with
        the production used at $n$
            construct an edge from each $c_i$ to $b$
```

Topological sort: order the nodes of the graph as $m_1, m_2, \ldots, m_n$ such that no edge goes from m_{i+k} to m_i for any i, k

Evaluation of SDDs

General scheme:

1. Parse the input program and construct the parse tree.
2. Draw the dependency graph for the parse tree.
3. Do a topological sort for the dependency graph.
4. Traverse nodes in topologically sorted order, and evaluate attributes at each node.

S-attributed definitions

Section 5.3

Definition: SDD with only synthesized attributes

Scheme:

1. Extend parser stack to have an extra field that stores the value of attributes.
   ALT. have a parsing stack and a parallel, value stack.

top →	X	$X.x$
	Y	$Y.y$
	$\vdots$	$\vdots$

2. When pushing a terminal symbol on parsing stack, push corresponding attribute value on value stack

S-attributed definitions

3. For the rule

$$\boxed{A \rightarrow X_1 X_2 \ldots X_r \qquad A.a = f(X_1.x_1, X_2.x_2, \ldots, X_r.x_r)}$$

modify the value stack as follows:

```
ntop = top - r + 1;
val[ntop] = f(val[top-r+1], ..., val[top]);
top = ntop;
```

Example:

PRODUCTION	SEMANTIC RULES
$L \rightarrow E \backslash n$	`print val[top]`
$E \rightarrow E_1 + T$	`val[ntop] = val[top-2] + val[top]`
$E \rightarrow T$	
$T \rightarrow T * F$	`val[ntop] = val[top-2] * val[top]`
$T \rightarrow F$	
$F \rightarrow (E)$	`val[ntop] = val[top-1]`
$F \rightarrow$ **num**	

L-attributed definitions

Definition: A SDD istb *L-attributed* if each inherited attribute of X_i in the RHS of $A \rightarrow X_1 \ldots X_n$ depends only on

1. attributes of $X_1, X_2, \ldots, X_{i-1}$ (symbols to the left of X_i in the RHS);
2. inherited attributes of A.

Section 5.4

Restrictions for translation schemes:

1. Inherited attribute of X_i must be computed by an action before X_i.
2. An action must not refer to synthesized attribute of any symbol to the right of that action.
3. Synthesized attribute for A can only be computed after all attributes it references have been completed (usually at end of RHS).

Bottom-up translation

Removing embedded actions:
for each embedded action
    replace action by a distinct **marker** non-terminal M
    add production $M \longrightarrow \epsilon$ to the grammar
    attach the action to the end of this production

NOTE: Original grammar and modified grammar accept the same language; actions are performed in the same order during parsing.

Example:

$$S \longrightarrow a\ A \quad \{C.i = f(A.s)\} \quad C$$
$$S \longrightarrow b\ A\ B \quad \{C.i = A.s\} \quad C$$
$$C \longrightarrow c \quad \{C.s = g(C.i)\}$$

$\Longrightarrow$

PRODUCTION	SEMANTIC RULES
$S \longrightarrow aANC$	$(N.i = A.s,\ C.i = N.s)$
$N \longrightarrow \epsilon$	$N.s = f(A.s)$
$S \longrightarrow bABMC$	$(M.i = A.s,\ C.i = M.s)$
$M \longrightarrow \epsilon$	$M.s = A.s$
$C \longrightarrow c$	$C.s = g(C.i)$

Bottom-up translation

Assumption: Each symbol X has one synthesized ($X.s$) and one inherited ($X.i$) attribute.

1. Replace each $A \longrightarrow X_1 \ldots X_n$ by
   $$\boxed{A \longrightarrow M_1 X_1 \ldots M_n X_n, \quad M_i \longrightarrow \epsilon \;\; \{X_i.i = f(\ldots)\}}$$
   where each M_i is a new marker non-terminal
2. When reducing by $M_i \longrightarrow \epsilon$:

$top \longrightarrow$	X_{i-1}	$X_{i-1}.s$
$top - 1 \longrightarrow$	M_{i-1}	$X_{i-1}.i$
	$\vdots$	$\vdots$
$top - 2i + 4 \longrightarrow$	X_1	$X_1.s$
$top - 2i + 3 \longrightarrow$	M_1	$X_1.i$
$top - 2i + 2 \longrightarrow$	M_A	$A.i$
$\ldots$	$\ldots$	

Compute $X_i.i$ and push on stack; $top \longleftarrow top + 1$

Bottom-up translation

3. When reducing by $A \rightarrow M_1 X_1 \ldots M_n X_n$:

   ```
   A.s = f(val[top-2n+2],...,val[top]);
   val[top-2n+1] = A.s;
   top = top-2n+1;
   ```

4. Simplifications:

   If X_j has no inherited attributes or is computed by a copy rule $X_j.i = X_{j-1}.s$ discard M_j; adjust indices of val array suitably.

   If $X_1.i$ exists and $X_1.i = A.i$, omit M_1.

   (avoids parsing conflicts in left recursive grammars)

NOTES:

i) $LL(1)$ grammar + markers is $LL(1)$ $\Rightarrow$ no conflicts

ii) $LR(1)$ grammar + markers may not be $LR(1)$ $\Rightarrow$ conflicts may occur

Bottom-up translation

Example:

PRODUCTION	SEMANTIC RULES	STACK OPS
$S \rightarrow aANC$	$(N.i = A.s,\ C.i = N.s)$	
$N \rightarrow \epsilon$	$N.s = f(A.s)$	`val[ntop] = f(val[top])`
$S \rightarrow bABMC$	$(M.i = A.s,\ C.i = M.s)$	
$M \rightarrow \epsilon$	$M.s = A.s$	`val[ntop] = val[top-1]`
$C \rightarrow c$	$C.s = g(C.i)$	`val[ntop] = g(val[top-1])`

Miscellaneous

non-L-attributed definitions:

$$\begin{array}{lll} D & \rightarrow & L : T \\ T & \rightarrow & \textbf{integer} \mid \textbf{char} \\ L & \rightarrow & L, \textbf{id} \mid \textbf{id} \end{array} \quad \Rightarrow \quad \begin{array}{lll} D & \rightarrow & \textbf{id}\ L \\ T & \rightarrow & \textbf{integer} \mid \textbf{char} \\ L & \rightarrow & , \textbf{id}\ L \mid\ : T \end{array}$$

"Hard" L-attributed definitions:

PRODUCTION	SEMANTIC RULES
$S \rightarrow L$	$L.count = 0$
$L \rightarrow L_1 1$	$L_1.count = L.count + 1$
$L \rightarrow \epsilon$	print$(L.count)$

Top-down translation

Section 5.5

Left-recursion elimination:

Input:

$$\begin{array}{ll} A \rightarrow A_1 Y & \{\ A.a = g(A_1.a, Y.y)\ \} \\ A \rightarrow X & \{\ A.a = f(X.x)\ \} \end{array}$$

Output:

$$\begin{array}{l} A \rightarrow X\ \ \{\ R.i = f(X.x)\ \}\ \ R\ \ \{\ A.a = R.s\ \} \\ R \rightarrow Y\ \ \{\ R_1.i = g(R.i, Y.y)\ \}\ \ R_1\ \ \{\ R.s = R_1.s\ \} \\ R \rightarrow \epsilon\ \ \{\ R.s = R.i\ \} \end{array}$$

Example:

$$\begin{array}{ll} E \rightarrow E_1 + T & \{\ E.val = E_1.val + T.val\ \} \\ E \rightarrow E_1 - T & \{\ E.val = E_1.val - T.val\ \} \\ E \rightarrow T & \{\ E.val = T.val\ \} \\ T \rightarrow (E) & \{\ T.val = E.val\ \} \\ T \rightarrow \textbf{num} & \{\ T.val = num.val\ \} \end{array}$$

$$\begin{array}{ll} E \rightarrow & T\ \ \{\ R.i = T.val\ \}\ \ R \\ & \{\ E.val = R.s\ \} \\ R \rightarrow & +\,T\ \ \{\ R_1.i = R.i + T.val\ \}\ \ R_1 \\ & \{\ R.s = R_1.s\ \} \\ R \rightarrow & -\,T\ \ \{\ R_1.i = R.i - T.val\ \}\ \ R_1 \\ & \{\ R.s = R_1.s\ \} \\ R \rightarrow & \epsilon\ \ \{\ R.s = R.i\ \} \end{array}$$

Predictive translation

Input: translation scheme based on a grammar suitable for predictive parsing

Output: Code for a syntax-directed translator

Method:

1. For each nonterminal A, construct a function with
   Input parameters: one for each inherited attribute of A;
   Return value: synthesized attributes of A;
   Local variables: one for each attribute of each grammar symbol that appears in a production for A.
2. Code for non-terminal A decides what production to use based on the current input symbol (switch statement). Code for each production forms one case of a switch statement.

Predictive translation

3. In the code for a production, tokens, nonterminals, actions in the RHS are considered left to right.

   (i) For token X: save $X.s$ in the variable created for X; generate a call to match X and advance input.

   (ii) For nonterminal B: generate an assignment

   ```
   c = B(b1, b2, ..., bk);
   ```

   where:

   `b1, b2, ...` are variables corresponding to inherited attributes of B,

   `c` is the variable for synthesized attribute of B,

   `B` is the function created for B.

   (iii) For an action, copy the code into the function, replacing each reference to an attribute by the variable created for that attribute.