Chapter 3

Semantic Analysis

3.1 Attribute Grammars

A parser which is just a recognizer is not sufficient for a compiler—its output conveys nothing about the nature of the parsed input, just whether it belongs to the language defined by the underlying grammar. In a compiler, a parser needs to communicate the parse tree it has internally generated. Hence, this section introduces an extended notion of a context-free grammar: the attribute grammar. An attribute grammar associates additional values (the *attribute instances*) with the nodes of a parse tree, and rules that relate them to each other. A parser in a compiler computes the values of the attribute instances and return them to the caller.

Attribute grammars carry a considerable amount of notational clutter. Some examples illustrate the central ideas.

Consider again the grammar for constant arithmetic expressions. For technical reasons, only one number has a literal: 42. An attribute grammar can describe how to actually compute the value of such an expression.

$$
\begin{array}{lll}
\langle\text{exp}\rangle & ::= \langle\text{term}\rangle & \langle\text{exp}\rangle.v = \langle\text{term}\rangle.v \\
 & \mid \langle\text{term}\rangle\ \texttt{+}\ \langle\text{exp}\rangle & \langle\text{exp}_1\rangle.v = \langle\text{term}\rangle.v + \langle\text{exp}_2\rangle.v \\
 & \mid \langle\text{term}\rangle\ \texttt{-}\ \langle\text{exp}\rangle & \langle\text{exp}_1\rangle.v = \langle\text{term}\rangle.v - \langle\text{exp}_2\rangle.v \\
\langle\text{term}\rangle & ::= \langle\text{prod}\rangle & \langle\text{term}\rangle.v = \langle\text{prod}\rangle.v \\
 & \mid \langle\text{prod}\rangle\ \texttt{*}\ \langle\text{term}\rangle & \langle\text{term}_1\rangle.v = \langle\text{prod}\rangle.v \cdot \langle\text{term}_2\rangle.v \\
 & \mid \langle\text{prod}\rangle\ \texttt{/}\ \langle\text{term}\rangle & \langle\text{term}_1\rangle.v = \langle\text{prod}\rangle.v / \langle\text{term}_2\rangle.v \\
\langle\text{prod}\rangle & ::= \texttt{42} & \langle\text{prod}\rangle.v = 42 \\
 & \mid \texttt{(}\ \langle\text{exp}\rangle\ \texttt{)} & \langle\text{prod}\rangle.v = \langle\text{exp}\rangle.v
\end{array}
$$

$$
\begin{array}{lll}
\langle\text{bit}\rangle & ::= \texttt{0} & \langle\text{bit}\rangle.v = 0 \\
 & \mid \texttt{1} & \langle\text{bit}\rangle.v = 2^{\langle\text{bit}\rangle.s} \\
\langle\text{bits}\rangle & ::= \langle\text{bit}\rangle & \langle\text{bits}\rangle.v = \langle\text{bit}\rangle.v, \langle\text{bit}\rangle.s = \langle\text{bits}\rangle.s, \langle\text{bits}\rangle.l = 1 \\
 & \mid \langle\text{bits}\rangle\ \langle\text{bit}\rangle & \langle\text{bits}_1\rangle.v = \langle\text{bits}_2\rangle.v + \langle\text{bit}\rangle.v, \langle\text{bit}\rangle.s = \langle\text{bits}_1\rangle.s, \\
 & & \langle\text{bits}_2\rangle.s = \langle\text{bits}_1\rangle.s + 1, \langle\text{bits}_1\rangle.l = \langle\text{bits}_2\rangle.l + 1 \\
\langle\text{num}\rangle & ::= \langle\text{bits}\rangle & \langle\text{num}\rangle.v = \langle\text{bits}\rangle.v, \langle\text{bits}\rangle.s = 0 \\
 & \mid \langle\text{bits}\rangle\ \texttt{.}\ \langle\text{bits}\rangle & \langle\text{num}\rangle.v = \langle\text{bits}_1\rangle.v + \langle\text{bits}_2\rangle.v, \langle\text{bits}_1\rangle.s = 0, \\
 & & \langle\text{bits}_2\rangle.s = -\langle\text{bits}_2\rangle.l
\end{array}
$$

3.1.1 Notation

3.1 Definition (Position)
A position *identifies an occurrence of a grammar symbol within a grammar production. The symbol is identified by a* $\circ$ *directly in front of it. Thus,* $\langle \circ A \to \alpha \rangle$ *identifies the left-hand side* A*, whereas* $\langle A \to \alpha \circ X \beta \rangle$ *identifies the* X*.*

3.2 Definition (Attribute grammar)
Let Att *be a set of* attributes.
An attribute grammar *is a tuple* $(G, \mathrm{Syn}, \mathrm{Inh}, \mathcal{R}, D)$ *where*

- $G = (N, T, P, S)$ *is a context-free grammar,*
- $\mathrm{Syn} : N \to \mathcal{P}(\mathrm{Att})$ *specifies for each nonterminal* A *the set* $\mathrm{Syn}(A)$ *of* synthesized attributes,
- $\mathrm{Inh} : N \to \mathcal{P}(\mathrm{Att})$ *defines for each nonterminal* A *the set* $\mathrm{Inh}(A)$ *of* inherited attributes,
- $\mathcal{R}$ *is a family of* attribution rules, *and*
- $D = (D^a)_{a \in \mathrm{Att}}$ *is a family of* attribute domains *such that* D^a *is the set of possible values for an attribute named* a*.*

For each A*, it holds that* $\mathrm{Syn}(A) \cap \mathrm{Inh}(A) = \varnothing$ *and* $\mathrm{Att}(A) := \mathrm{Inh}(A) \cup \mathrm{Syn}(A)$ *is the set of* attributes *of* A*.*

An attribute occurrence *is a pair* $p.a$ *consisting of a position in a production and an attribute. An attribute occurrence must either have the form* $\langle \circ A \to \alpha \rangle .a$ *with* $a \in \mathrm{Att}(A)$ *or* $\langle A \to \gamma \circ B \delta \rangle .a$ *with* $a \in \mathrm{Att}(B)$*.*

The occurrences of a production fall into two classes: the defining *occurrences* $\mathrm{Def}(A \to \alpha)$ *and the* applied *occurrences* $\mathrm{App}(A \to \alpha)$*.*

$$\begin{aligned}
\mathrm{Def}(A \to \alpha) :=& \{\langle \circ A \to \alpha \rangle .a \mid a \in \mathrm{Syn}(A)\} \\
&\cup \{\langle A \to \gamma \circ B \delta \rangle .a \mid \alpha = \gamma B \delta \ \wedge \ a \in \mathrm{Inh}(B)\} \\
\mathrm{App}(A \to \alpha) :=& \{\langle \circ A \to \alpha \rangle .a \mid a \in \mathrm{Inh}(A)\} \\
&\cup \{\langle A \to \gamma \circ B \delta \rangle .a \mid \alpha = \gamma B \delta \ \wedge \ a \in \mathrm{Syn}(B)\}
\end{aligned}$$

Each defining occurrence $p.a \in \mathrm{Def}(A \to \alpha)$ *has an associated* attribution rule

$$p.a = f_{p.a}(p_1.a_1, \ldots, p_m.a_m)$$

where m *is some natural number associated with* $p.a$*, and all* $p_j.a_j \in \mathrm{App}(A \to \alpha)$ *are applied attribute occurrences for the same production.* $f_{p.a}$ *must be a function* $D^{a_1} \times \cdots \times D^{a_m} \to D^a$*.* $\mathcal{R}$ *is the family of all such attribution rules, indexed by the defining attribute occurrences.*

□

The attribute dependencies of the grammar according to the above definition are in *Bochmann normal form* [Boc76]: Inherited attributes depend only on attributes “above” them in the parse tree, synthesized attributes on those “below”. An attribute grammar assigns a meaning to a derivation tree by prescribing how to label its nodes.

3.3 Definition (Derivation tree)
A derivation tree *for a context-free grammar* (N, T, P, S) *is a finite, ordered, directed tree with node set* V *such that each node is labeled with a production. Furthermore*

- *if node* v_0 *is labeled with* $A_0 \to A_1 \ldots A_n$*, then* v_0 *must have* n *successors* $v_1, \ldots, v_n$ *such that* v_i *is labeled with* $A_i \to \alpha_i$*,*

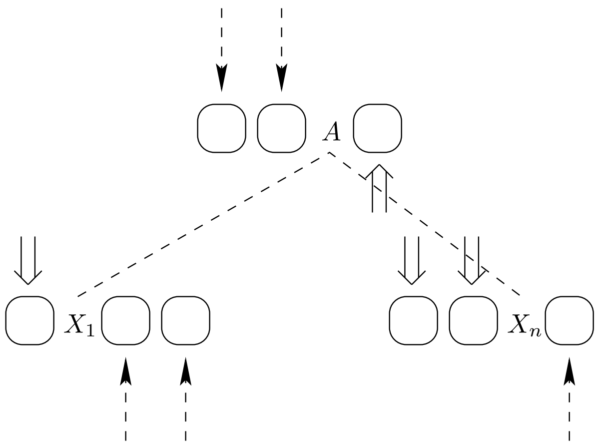

Figure 3.1: A grammar attribution

- *in a* full *derivation tree, the root node is labeled with a start production* $S \to \alpha$.

3.4 Definition (Attribute decoration)
An attribute decoration *associates with each node v in a derivation tree, which is labeled with production $A \to \alpha$, an* attribute instance $v.a \in D^a$ *for each* $a \in \mathrm{Att}(A)$.

An attribute decoration is valid *if for each node v_0, which is labeled with $A_0 \to A_1 \dots A_n$ and has successors $v_1, \dots, v_n$, and for each defining occurrence $p_0.a \in \mathrm{Def}(A_0 \to A_1 \dots A_n)$ with associated attribution rule*

$$p_0.a = f_{p_0.a}(p_1.a_1, \dots, p_m.a_m)$$

it holds that

$$v_{i_0}.a = f_{p_0.a}(v_{i_1}.a_1, \dots, v_{i_m}.a_m)$$

where position p_j identifies nonterminal occurrence A_{i_j} and thus node v_{i_j}.

□

Hence, an attribute grammar G with start production $S \to A$ defines a meaning function $\mathcal{M} : T^* \times D^{\mathrm{Inh}(A)} \to D^{\mathrm{Syn}(S)}$. For $\xi \in L(G)$, $\mathcal{M}(\xi, v_1, \dots, v_{|\,\mathrm{Inh}(A)|})$ seeds the derivation tree with attribute instances for the inherited attributes of A and yields instances of the synthesized attributes of S that result from the decoration of the derivation tree.

3.1.2 Computing a Decoration

There are many techniques for actually producing such a decoration of a derivation tree, some of them quite involved. It is, however, obviously desirable to generate the decoration during parsing. After all, the meaning function is only interested in the attribute instances of the start production; the derivation tree itself is not part of it. Hence, computing the decoration during parsing could avoid having to store the tree. Unfortunately, general attribute grammars may *require* the whole derivation tree to be present for performing attribute evaluation; the attribute rules may, after all, generate circularities.

Therefore, it is necessary to restrict the class of attribute grammars such that they become amenable to “on-the-fly” attribute evaluation. Two particular ways of formulating suitable restrictions on attribute grammars are *L-attributed* and *S-attributed* grammars.

3.5 Definition (L-attributed grammar)
An L-attributed *grammar is an attribute grammar where, for each rule*

$$d = f_d(a_1, \dots, a_{i_d})$$

with d of the form $\langle A \to \alpha \circ B\beta \rangle .a$ (with $a \in \mathrm{Inh}(B)$), each a_j either has the form $\langle \circ A \to \alpha B\beta \rangle .a$ (with $a \in \mathrm{Inh}(A)$) or $\langle A \to \gamma \circ C\delta B\beta \rangle .a$ (with $\gamma C\delta = \alpha$ and $a \in \mathrm{Syn}(C)$). (Rules with d of the form $\langle \circ A \to \alpha \rangle .a$ have no restrictions.)

In an L-attributed grammar, any attribute occurrence may only depend on occurrences to its *left* (hence *L*-attributed). Since a recursive-descent parser proceeds from left to right in grammar rules, L-attributed grammars lend themselves to on-the-fly attribute evaluation by recursive-descent parsers.

14 Example
Consider a fragment of a recursive descent parser for the grammar rule $A \to A_1 \dots A_m$. The parse function for A takes the input string and the inherited attributes of A as parameters and returns remaining string and the synthesized attributes of A as a result. For simplicity, we assume that there is only one synthesized attribute, s, and one inherited attribute, i.

$$\begin{array}{lcl}
[A] & : & T^* \times \mathrm{Inh}(A) \to T^* \times \mathrm{Syn}(A) \\
[A](\xi, \mathit{inh}) & = & \textbf{let } \mathit{inh}_1 = f_{\langle A \to \circ A_1 \dots A_m \rangle .i}(\mathit{inh}) \textbf{ in} \\
 & & \textbf{let } (\xi_1, \mathit{syn}_1) = [A_1](\xi, \mathit{inh}_1) \textbf{ in} \\
 & & \textbf{let } \mathit{inh}_2 = f_{\langle A \to A_1 \circ A_2 \dots A_m \rangle .i}(\mathit{inh}, \mathit{syn}_1) \textbf{ in} \\
 & & \vdots \\
 & & \textbf{let } (\xi_m, \mathit{syn}_m) = [A_m](\xi_{m-1}, \mathit{inh}_m) \textbf{ in} \\
 & & (\xi_m, f_{\langle \circ A \to A_1 \dots A_m \rangle .s}(\mathit{inh}, \mathit{syn}_1, \dots, \mathit{syn}_m))
\end{array}$$

□

However, L-attributed grammars present problems for recursive-ascent parsers: As a recursive-ascent parser proceeds, it always has to keep several different productions "in mind," each of which may have completely different attribute rules. It therefore would have to evaluate all of them in parallel—and thus do much superfluous work. Therefore, recursive-ascent parsers usually restrict the class of attribute grammars they accept even further: they simply do not allow inherited attributes.

3.6 Definition (S-attributed grammar)
An S-attributed grammar *is an attribute grammar* $(G, \mathcal{S}, \mathcal{I}, \mathcal{R}, D)$ *with* $\mathrm{Inh}(A) = \varnothing$ *for all* $A \in N$.

For an S-attributed grammar, attribute evaluation always flows upwards in the derivation tree: no circularities may occur, all dependencies point in the same direction. Because of this, it is actually sufficient to have just one synthesized attribute per nonterminal—multiple attributes are easily simulated by using aggregate values such as records. Therefore, the rest of this section assumes $\mathrm{Syn}(A) = \{v\}$ for all $A \in N$. Also, for simplicity of the presentation, $f_{\langle A \to \alpha \rangle .v}$ always has $|\alpha|$ arguments. Terminals simply yield some reserved value $\perp$ as their attribute which may not be used.