Attribute-Directed Top-Down Parsing

Karel Müller

Czech Technical University, Department of Computers
Prague, Czechoslovakia

Abstract. This paper deals with a method how an effective attribute-directed top-down parser and attribute evaluator can be constructed from a conditional L-attributed grammar (CLAG). The method is based on exploitation of an attribute stack in attribute evaluation and on definition of a translation scheme for CLAG.

1 Basic Concepts and Notations

Our definition of attribute grammars is based on [4] and [5]. An *attribute grammar* (AG) G over a *semantic domain* D is a context-free grammar $G_0 = (N, \Sigma, P, S)$, the *underlying context-free grammar* of G, augmented with attributes and semantic rules. A production $p \in P$ is denoted by $p : X_0 \rightarrow X_1 X_2 ... X_n$ where $X_0 \in N$, $X_i \in (N \cup \Sigma)$ for all i, $0 < i \leq n$, $(n \geq 0)$. The semantic domain D is a pair (Ω, Φ), where Ω is a set of sets, the sets of attribute values, and the set $\{true,false\}$ of boolean values, and Φ is a collection of mappings (called *semantic functions* of the form $f : V_1 \times ... \times V_m \rightarrow V_0$, where $m \geq 0$ and $V_i \in \Omega, 0 \leq i \leq m$. The set of *attribute symbols* denoted by Att is partitioned into Inh (*inherited* attribute symbols) and Syn (*synthesized* attribute symbols). For each attribute symbol $b \in Att$, a set $V(b) \in \Omega$ contains all possible values of the attributes corresponding to b.

For $X \in N, Att(X)$ denotes the set of attribute symbols of X. An *attribute* is denoted $X.a$, where $X \in N$ and $a \in Att(X)$. $Inh(X)$ $(Syn(X))$ denotes the set of inherited (synthesized) attribute symbols of X. We assume that the start symbol has no inherited attributes and terminals have no attributes at all.

For $X \in N$, let Ord define a linear ordering of the attributes of X with the inherited attributes preceding the synthesized attributes. Thus, for all $X \in N$, $Ord(X)$ is an ordering of $Att(X)$ and for $b \in Att(X)$, $Ord(X)(b)$ is the index of b with respect to this ordering.

A production $p : X_0 \rightarrow X_1 X_2 ... X_n$ has an *attribute occurrence* $k.b$, $0 \leq k \leq n$, if $X_k.b$ is an attribute. An attribute occurrence $k.b$ of p is called an *input occurrence*, if either $b \in Inh$ and $k = 0$, or $b \in Syn$ and $k > 0$. Otherwise $k.b$ is said to be an *output occurrence*. For each output occurrence $k.b$ of p, there is exactly one *semantic rule* of the form $k.b := f(j_1.a_1, ..., j_m.a_m)$, where every $j_i.a_i$ is an input occurrence of p and f is a semantic function in Φ of the type $f : V_1 \times ... \times V_m \rightarrow V_0$, where $V_0 = V(b)$ and for $1 \leq i \leq m$: $V_i = V(a_i)$. Notice that attribute grammars are in Bochmann normal form. An attribute grammar is *L-attributed*, if for every semantic rule $k.b := f(j_1.a_1, ..., j_m.a_m)$ such that b is an inherited attribute holds $j_i < k$ for each $i = 1, ..., m$.

A finite set $C(p)$ of *semantic conditions* is associated with each production $p \in P$. A semantic condition is an expression of the form $q(j_1.a_1, ..., j_m.a_m)$, where every

$j_i.a_i$ is an input occurrence of p and q is a boolean-valued function of the type $q : V_1 \times ... \times V_m \rightarrow \{true, false\}$, where $V_i = V(a_i)$, $1 \leq i \leq m$. An attribute grammar in which for all productions p the set $C(p)$ of semantic conditions is empty, is called an *unconditional* attribute grammar. Otherwise AG is called *conditional.*

Let t be a complete derivation tree of underlying CFG G_0 of G, u its node labelled with X. Then for all $b \in Att(X)$, $u.b$ is an *attribute instance* attached to a node u. If a node u has n sons $u_1, ..., u_n$ which are labelled according to a production $p : X_0 \rightarrow X_1 X_2 ... X_n$, then each semantic rule $k.b := f(j_1.a_1, ..., j_m.a_m)$ associated with p is interpreted as an *evaluation instruction* $u_k.b := f(u_{j_1}.a_1, ..., u_{j_m}.a_m)$ associated with attribute instance $u_k.b$, and each semantic condition $q(j_1.a_1, ..., j_m.a_m)$ from $C(p)$ is interpreted as a *test instruction* $q(u_{j_1}.a_1, ..., u_{j_m}.a_m)$ associated with p.

A derivation tree t is *well evaluated* if all attribute instances have values according to the associated evaluation instructions, and all test instructions associated with the productions used in the tree yield *true*. $TREES(G)$ denotes the set of all well evaluated derivation trees of G. The *language generated* (or *defined*) by an AG G is defined by $L(G) = \{w \mid w = yield(t)$, for some $t \in TREES(G)\}$. Notice that $L(G) \subseteq L(G_0)$. For an unconditional AG G, $L(G) = L(G_0)$.

Let the start symbol of the underlying CFG of an AG G have a distinguished synthesized attribute symbol r. The *translation* (more precisely *string-to-value translation*) $T(G)$ *generated* (or *defined*) by AG G is the mapping from $L(G)$ to subsets of the set $V(r)$ defined by $T(G)(w) = \{x \mid x = u.r$, u is the root of a well evaluated tree t, r is its distinguished attribute and $w = yield(t)\}$. This set may contain more than one element. In this case G is called *semantically ambiguous*, otherwise G is *semantically unambiguous.*

Throughout this paper, conditional L-attributed grammars (CLAG) are treated. It is well known that any derivation tree in CLAG can be evaluted using the one-pass evaluation strategy [2].

2 Attribute Stack

In order to obtain a translation defined by a L-attribute grammar for an input string, we can simulate the one-pass evaluation of a derivation tree and allocate memory for attribute instances using a stack of registers, which can hold attribute values. For an interior node u labelled with X, and its sons $u_1, ..., u_n$ labelled with $X_1, ..., X_n$, the stack of attribute registers (attribute stack) will be used in the following way:

- Before entering a subtree with the root u, the top of the attribute stack consists of registers with evaluated attributes from $Inh(X)$ and registers with undefined values of attributes from $Syn(X)$.
- After leaving this subtree, the top of the attribute stack consists of registers with attributes from $Syn(X)$.
- Before evaluation of inherited attributes of X_i, the attribute stack contains registers with evaluated attributes from $Syn(X_{i-1}), ..., Syn(X_1), Inh(X)$ and registers with undefined values of attributes from $Syn(X)$. Registers for all attributes from $Att(X_i)$ are then added to the stack, attributes from $Inh(X_i)$ are evaluated and a subtree with the root u_i is entered.

- After evaluation of attributes from $Syn(X)$, the attribute stack contains registers with evaluated attributes from $Syn(X_n), ..., Syn(X_1), Inh(X), Syn(X)$. These registers except $Syn(X)$ are then removed.

Definition 1. Attribute stack.
Let $D = (\Omega, \Phi)$ be the semantic domain of a conditional L-attribute grammar G. Let Nat be the set of natural numbers, Pos the set of positive integers, $Val = V_1 \cup ... \cup V_n \cup \{undef\}$ for all V_i from Ω the set containing all possible attribute values including undefined value $undef$. An *attribute stack* over the domain D is a data structure of the type $Astack$ for which the following operations are defined:

$empty : \; \rightarrow Astack$
$push : \; Astack, Val \rightarrow Astack$
$add : \; Astack, Nat \rightarrow Astack$
$remove : Astack, Nat \rightarrow Astack$
$read : \; Astack, Pos \rightarrow Val$
$write : \; Astack, Pos, Val \rightarrow Astack$
$length : Astack \rightarrow Nat$

These operation should satisfy the following equations:

$add(s, 0) = s$
$add(s, n) = push(add(s, n-1), undef)$ for $n > 0$
$remove(s, 0) = s$
$remove(push(s, x), n) = remove(s, n-1)$ for $n > 0$
$read(push(s, x), 1) = x$
$read(push(s, x), n) = read(s, n-1)$ for $n > 1$
$write(push(s, x), 1, y) = push(s, y)$
$write(push(s, x), n, y) = push(write(s, n-1, y), x)$ for $n > 1$
$length(empty) = 0$
$length(push(s, x)) = length(s) + 1$

3 Translation scheme for CLAG

In order to formally describe an attribute evaluation using the attribute stack for a given L-attribute grammar, each semantic rule will be transformed to an operation of the type $Astack \rightarrow Astack$ and each semantic condition to an operation of the type $Astack \rightarrow \{true, false\}$. Adding new registers and removing old registers will be done in the same way. These operations will be called *semantic operations* and *semantic predicates*.

For any production $p : X_0 \rightarrow X_1 X_2 ... X_n$, we will define the following semantic operations and predicates:

- $A_{p,i}$ adding registers for attributes of X_i, $1 \leq i \leq n$, to the attribute stack,
- $E_{p,i}$ evaluation of the inherited attributes of X_i, $1 \leq i \leq n$,
- $E_{p,0}$ evaluation of the synthesized attributes of X_0,
- R_p removing registers with synthesized attributes of the right-hand side of p and inherited attributes of the left-hand side of p from the attribute stack,
- $P_{p,0}$ a predicate which is evaluated and tested before entering a subtree with the root X_1
- $P_{p,i}$ a predicate which is evaluated and tested after leaving a subtree with the root X_i, $1 \leq i \leq n$.

Semantic operations and predicates can be constructed by the Algorithm 1.

Algorithm 1: Construction of semantic operations and predicates.
Input: A conditional L-attributed grammar G.
Output: OP, the set of semantic operations, and PR, the set of semantic predicates.
Method: For each production $p: X_0 \to X_1X_2...X_n$, let F_k be the set of all semantic rules $k.b := f(j_1.a_1, ..., j_m.a_m)$, $0 \le k \le n$, and C_k the set of all semantic conditions $q(j_1.a_1, ..., j_m.a_m)$, for which $k = max(j_1, ..., j_m)$. For $k = 0, 1, ..., n$ construct the semantic operations and the semantic predicates according to the following rules:

(1) For each attribute occurrence $i.a$ in F_k define the attribute stack selector $sel_k(i.a)$ as follows:

$$sel_k(i.a) = \begin{cases} Ord(X_k)(a) & \text{if } k > 0, i = k, \\ |Att(X_k)| + \sum_{j=i+1}^{k-1} |Syn(X_j)| + Ord(X_i)(a) - |Inh(X_i)| & \text{if } k > 0, 0 < i < k, \\ |Att(X_k)| + \sum_{j=1}^{k-1} |Syn(X_j)| + Ord(X_0)(a) & \text{if } k > 0, i = 0, \\ \sum_{j=i+1}^{n} |Syn(X_j)| + Ord(X_i)(a) - |Inh(X_i)| & \text{if } k = 0, i > 0, \\ \sum_{j=1}^{n} |Syn(X_j)| + Ord(X_i)(a) & \text{if } k = 0, i = 0. \end{cases}$$

(2) For each attribute occurrence $i.a$ in C_k define the attribute stack selector $selc_k(i.a)$ as follows:

$$selc_k(i.a) = \begin{cases} Ord(X_k)(a) - |Inh(X_k)(a)| & \text{if } k > 0, i = k, \\ \sum_{j=i+1}^{k} |Syn(X_j)| + Ord(X_i)(a) - |Inh(X_i)| & \text{if } k > 0, 0 < i < k, \\ \sum_{j=1}^{k} |Syn(X_j)| + Ord(X_0)(a) & \text{if } k > 0, i = 0, \\ Ord(X_0)(a) & \text{if k=0,i=0.} \end{cases}$$

(3) For each semantic rule $k.b := f(j_1.a_1, ..., j_m.a_m)$ define the semantic operation $sop_{k,b}$ as

$$sop_{k,b}(s) = write(s, sel_k(k.b), f(read(s, sel_k(j_1.a_1)), ..., read(s, sel_k(j_m.a_m)))).$$

Construct the semantic operation $E_{p,k}$ as a composition of the operations $sop_{k,b}$:

$$E_{p,k}(s) = sop_{k,b_1}(sop_{k,b_2}(...(sop_{k,b_m}(s))...)).$$

Add $E_{p,k}$ to OP.

(4) For each semantic condition $q(j_1.a_1, ..., j_m.a_m)$ from C_k define the semantic predicate $spr_{k,q}$ as

$$spr_{k,q}(s) = q(read(s, selc_k(j_1.a_1)), ..., read(s, selc_k(j_m.a_m))).$$

Construct the semantic predicate $P_{p,k}$ as a conjuction of the predicates $spr_{k,q}$:

$$P_{p,k}(s) = spr_{k,q_1}(s) \textbf{ and } ... \textbf{ and } spr_{k,q_m}(s).$$

Add $P_{p,k}$ to PR.

(5) If $k > 0$ and $sz = |Att(X_k)|$ is greater then 0, then add to OP the semantic operation $A_{p,k}$ defined as $A_{p,k}(s) = add(s, sz)$.

(6) If $k = 0$ and $sz = |Inh(X_0)| + \sum_{j=1}^{n} |Syn(X_j)|$ is greater then 0, then add to OP the semantic operation R_p defined as $R_p(s) = remove(s, sz)$.

Definition 2. Translation scheme for CLAG.
Let G be a conditional L-attributed grammar over a semantic domain D with underlying CFG $G_0 = (N, \Sigma, P, S)$, OP the set of semantic operations, and PR the set of semantic predicates constructed by the Algorithm 1. A *translation scheme* for G is the translation grammar $Q = (N, \Sigma, \Gamma, R, S)$, where $\Gamma = OP \cup PR$ and each production $r \in R$ corresponds to one and only one production $p \in P$ in the following

way:

$p : X_0 \rightarrow X_1X_2...X_n$

$r : X_0 \rightarrow P_{p,0}A_{p,1}E_{p,1}X_1P_{p,1}...A_{p,n}E_{p,n}X_nP_{p,n}E_{p,0}R_p$

If any of the symbols $P_{p,i}$, $A_{p,i}$, $E_{p,i}$ or R_p does not exist then empty string is used instead of the symbol in production r. Any symbol from the set Γ will be called an *action symbol.*

Definition 3. Attributed derivation.
Let $Q = (N, \Sigma, OP \cup PR, R, S)$ be the translation scheme of a CLAG. An *attributed form* (A-form) is a pair (α, s) where $\alpha \in \Sigma^*\{.\}(N \cup \Sigma \cup OP \cup PR)^*$, $s \in Astack$. A *direct attributed derivation* is the relation between attributed forms denoted by $\Rightarrow$ and defined as follows:

1. $(\alpha X\beta, s) \Rightarrow (\alpha\delta\beta, s)$ if $X \in N$, $X \rightarrow \delta$ is a rule in R,
2. $(\alpha.a\beta, s) \Rightarrow (\alpha a.\beta, s)$ if $a \in \Sigma$,
3. $(u.E\beta, s) \Rightarrow (u.\beta, E(s))$ if $E \in OP$,
4. $(u.C\beta, s) \Rightarrow (u.\beta, s)$ if $C \in PR$, $C(s) = true$.

The direct attributed derivation according to the first rule is called a *syntax derivation,* the others are called *semantic derivations.* Notation $f \Rightarrow^* g$ expresses that an A-form g is derived from an A-form f, i.e. that there is a sequence of attributed forms $f = f_0, f_2, ..., f_n = g$, where $f_i \Rightarrow f_{i+1}$, $0 \leq i < n$. This sequence is called an *attributed derivation* of the length n of the A-form g from the A-form f.

Definition 4. Let $Q = (N, \Sigma, OP \cup PR, R, S)$ be the translation scheme of a CLAG. The *language generated* by Q is defined by

$L(Q) = \{u \mid (.S,\ add(empty, |Syn(S)|)) \Rightarrow^* (u., s),\ u \in \Sigma^*\}.$

The *translation generated* by Q is the mapping from $L(Q)$ to subsets of the set $V(r)$, r is the distinguished attribute of S, defined by

$T(Q)(u) = \{v \mid (.S,\ add(empty, |Syn(S)|)) \Rightarrow^* (u., s),\ v = read(s, Ord(S)(r))\}.$

Theorem 5. *Let G be a conditional L-attributed grammar, Q be the translation scheme for G. Then $L(G) = L(Q)$ and $T(G) = T(Q)$.*

Proof. Can be found in [6].

4 Nondeterministic Machine for CLAG

The translation defined by a CLAG can be performed by a pushdown automaton with an infinite set of states. We define a pushdown automaton M as a system $M = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ in the same way as in [1] with the only exception that the set of states K may be infinite.

Theorem 6. *Let G be a CLAG, r the distinguished synthesized attribute. There exists a pushdown automaton M with potentially infinite set of states K, and a mapping f of the type $K \rightarrow V(r)$, such that the language accepted by M equals $L(G)$ and for $w \in L(G)$, $v = T(G)(w)$ if and only if $(q_0, w, Z_0) \vdash_M^* (q, e, e)$ and $v = f(q)$.*

Proof. Let $G_0 = (N, \Sigma, P, S)$ be the underlying CFG of G and $Q = (N, \Sigma, OP \cup PR, R, S)$ the translation scheme for G. Then $M = (K, \Sigma, \Gamma, \delta, q_0, S, \emptyset)$ where

K is the set of all possible values of the type Astack,
$\Gamma = \Sigma \cup N \cup OP \cup PR \cup \{E\}$, E is a new symbol,
q_0 is value of the operation $add(empty, |Syn(S)|)$
$\delta(q, a, a) = \{(q, e)\}$ for all $a \in \Sigma$,
$\delta(q, e, X)$ contains (e, α) for all production $X \rightarrow \alpha \in R$,
$\delta(q, e, Op) = \{(Op(q), e)\}$ for all $Op \in OP$
$\delta(q, e, Pr) = \{(q, \text{ if } Pr(q) \text{ then } e \text{ else } E)\}$ for all $Pr \in PR$.
The mapping f is defined as $f(s) = read(s, Ord(S)(r))$. The rest of the proof can be found in [6].

5 Deterministic Top-down Machine for CLAG

A deterministic top-down parser for CLAG can be driven not only by a lookahead symbol but also by conditions over attributes. Such parser is said to be *attribute-driven.* The following definition determines a class of translation schemes for which a deterministic top-down attribute-driven parser can be constructed.

Definition 7. A translation scheme $Q = (N, \Sigma, OP \cup PR, R, S)$ of a CLAG G is a ALL(1) translation scheme if for all $X \in N$ the following holds: if there are distinct productions $p_1 : X \rightarrow \alpha_1$ and $p_2 : X \rightarrow \alpha_2$, such that:
$FIRST_1(\alpha_1.FOLLOW_1(X)) \cap FIRST_1(\alpha_2.FOLLOW_1(X)) \neq \emptyset$,
then $\alpha_1 = P_1\beta_1$, $\alpha_2 = P_2\beta_2$, P_1 and $P_2 \in PR$, and for any value s of the type *Astack*, for which both $P_1(s)$ and $P_2(s)$ are defined, expression $(P_1(s)$ **and** $P_2(s))$ yields false.

Definition 8. A parse table for an ALL(1) translation scheme Q is a mapping M of the type $N \times (\Sigma \cup \{e\}) \rightarrow ACT$, in which ACT is a set of actions containing elements **expand**(p), **select**$(p_1, p_2, ..., p_n)$ and **error**, where $p, p_1, ..., p_n$ are productions of T.
- $M(X, u) =$ **expand**(p) if $p : X \rightarrow \alpha$, $u \in FIRST_1(\alpha.FOLLOW_1(X))$ and either the first symbol of the string α is a predicate symbol or for any other production $X \rightarrow \beta$ holds $u \in FIRST_1(\beta.FOLLOW_1(X))$.
- $M(X, u) =$ **select**$(p_1, ..., p_n)$ if $p_1 : X \rightarrow P_1\alpha_1 \; ... \; p_n : X \rightarrow P_n\alpha_n$ are all X-production for which P_i is in PR and $u \in FIRST_1(\alpha_i.FOLLOW1(X))$.
- Otherwise $M(X, u) =$ error.

Algorithm 2: ALL(1) parser for translation scheme.
Input: An ALL(1) translation scheme Q for CLAG G with the distinguished attribute r, an input string w.
Output: if $w \in L(G)$, then $T(G)(w)$; otherwise, an error indication.
Method: Let M be the parse table for T. A configuration of the parser is a triple (v, α, s), where $v \in \Sigma^*$ is an unread part of the input, $\alpha \in (N \cup \Sigma \cup OP \cup PR)^*$ is a current content of the parsing stack and s is a current value of the attribute stack. A move of the parser is the relation between configurations denoted by $\vdash$ and defined as follows:
1. $(av, Z\alpha, s) \vdash (v, \alpha, s)$ if $Z \in \Sigma$, $Z = a$,
2. $(av, Z\alpha, s) \vdash (av, \beta\alpha, s)$ if $Z \in N$, $M(Z, a) =$ **expand**$(Z \rightarrow \beta)$,

3. $(av, Z\alpha, s) \vdash (av, \beta\alpha, s)$ if $Z \in N,\ M(Z,a) = \textbf{select}(..., Z \to P\beta, ...),$
$P \in PR,\ P(s) = true,$
4. $(av, Z\alpha, s) \vdash (av, \alpha, s)$ if $Z \in PR,\ Z(s) = true,$
5. $(av, Z\alpha, s) \vdash (av, \alpha, Z(s))$ if $Z \in OP.$

The execution of the algorithm is as follows:

(1) Starting in the initial configuration $C_0 = (w, S, add(empty, |Syn(S)|))$, compute successive next configurations $C_0 \vdash C_1 \vdash ... \vdash ...$ until no further configurations can be computed.

(2) If the last computed configuration is (e, e, s) then result is $read(s, Ord(s)(r))$. Otherwise, report an error.

Translation schemes can be transformed by transformations known for translation grammars. Therefore an ALL(1) parser can be constructed also in case the underlying CF grammar of a CLAG G is not LL(1) but a transformation of the translation scheme for G into an ALL(1) form succeeds. Moreover, special transformations for translation schemes can be developed. These transformations respect the semantics of action symbols. For more details see [6].

6 Implementation

The method described in the previous sections has been fully implemented in the compiler constructor ATRAG 4.0 [6]. This system was used several times as a tool supporting development and implementation of a commercial compiler. For instance, the front-end part of the Pascal compiler for processor Intel 8096 family was specified by a conditional L-attribute grammar with non LL(1) syntax. The recent practical expoitation of ATRAG is the front-end part of a translator from Hewlett-Packard Basic 5.5 into ANSI-C language.

References

[1] Aho,A.V., Ullman,J.D.: The theory of parsing, translation and compiling. Vol.1 and Vol.2, Prentice Hall, Engelwood Cliffs, N.J., 1972.

[2] op den Akker,R.: Parsing attribute grammar. Doct. Diss., Dept. Comput. Sci., University of Twente, The Netherlands, 1988.

[3] op den Akker,R., Melichar,B. and Tarhio,J.: Attribute evaluation and parsing. In: Proc. of International Summer School SAGA (ed. H.Alblas and B.Melichar), Lect. Notes Comput. Sci. 545, Springer-Verlag, Berlin, 1991, pp. 187-214.

[4] Filè,G.: The theory of attribute grammars. Doct. Diss., Twente University of Technology, Enschede, The Netherlands, 1983.

[5] Knuth,D.E.: Semantics of context-free languages. Math. System Theory 2 (1968), pp. 127-145.

[6] Müller,K.: Attribute-directed top-down parsing. Research Rep. DC-92-05, Dept. of Comp., Czech Univ. of Techn., Prague, 1992.

[7] Watt,D.A.: Rule splitting and attribute-directed parsing. In: Proc. of Workshop Aarhus (ed. N.D.Jones), Lect. Notes Comput. Sci. 94, Springer-Verlag, Berlin, 1980, pp. 363 - 392.