An Extension of Earley's Algorithm for S-Attributed Grammars

Nelson Correa
Department of Electrical Engineering
Universidad de los Andes
Apartado Aéreo 4976, Bogotá, D.E., Colombia
bitnet : NCORREA at ANDESCOL

Abstract

Attribute grammars are an elegant formalization of the augmented context-free grammars characteristic of most current natural language systems. This paper presents an extension of Earley's algorithm to Knuth's attribute grammars, considering the case of S-attributed grammars. For this case, we study the conditions on the underlying base grammar under which the extended algorithm may be guaranteed to terminate. *Finite partitioning* of attribute domains is proposed to guarantee the termination of the algorithm, without the need for any restrictions on the context-free base.

1. Introduction

Earley's (1970) algorithm is a general algorithm for context-free languages, widely used in natural language processing (King, 1983; Shieber, 1985) and syntactic pattern recognition (Fu, 1982), where the full generative power of context-free grammar is required. The original algorithm and its common implementations, however, assume the atomic symbols of context-free grammars, thus limiting its applicability to systems with attributed symbols, or *attribute grammars* (Knuth, 1968).

Attribute grammar is an elegant formalization of the augmented context-free grammars characteristic of most current NLP systems. It is more general than members of the family of unification-based grammar formalisms (Kay, 1985; Shieber, 1986), mainly in that it allows and encourages the use of simpler attribution functions than unification for the definition of attribute values, and hence can lead to *computationally efficient* grammatical definitions, while maintaining the advantages of a well-understood declarative formalism. Attribute grammar has been used in the past by the author to define computational models of Chomsky's Government-binding theory, from which practical parsing programs were developed (Correa, 1987a).

Many systems based on Earley's algorithm have a clear division between the phases of syntactic and semantic analysis. Yellin (1988), for instance, uses a syntactic analysis phase in which Earley's algorithm builds a factored parse tree (FPT) for the given input, which is then followed by up to two phases of semantic analysis, during which the FPT is attributed and evaluated. Watt (1980) and Jones and Madsen (1980) propose a close interaction between syntactic and semantic analysis in the form of "attribute-directed" parsing. However, their particular realization of the technique is severely restricted for NLP applications, since it uses a deterministic one-path (LR) algorithm, applicable only to semantically unambiguous grammars.

Pereira and Warren (1983) and Shieber (1985) present versions of Earley's algorithm for unification grammars, in which unification is the sole operation responsible for attribute evaluation. However, given the high computational cost of unification, important differences between attribute and unification grammars in their respective attribution domains and functions (Correa, forthcoming), and the more general nature of attribute grammars in this regard, it is of interest to investigate the extension of Earley's algorithm directly to the main subclasses of attribute grammar.

The paper is organized as follows: Section 2 presents preliminary elements, including a definition of attribute grammar and Earley's algorithm. Section 3 presents the extension of the algorithm for S-attributed grammars. In Section 4, we consider the conditions on the underlying grammar under which the extended algorithm may be guaranteed to terminate for each input. For the S-attributed case we show that the algorithm terminates if the grammar has no cycles or, equivalently, if it is finitely ambiguous. However, *finite partitioning* of attribute domains may be used to guarantee the termination of the algorithm, without the need for restrictions on the context-free base. Finally, a conclusion and note on implementation are given.

2. Notation and Preliminaries

We follow the usual notation and terminology for grammars and languages. A *language* is a set of strings over a finite set T of symbols. A *grammar* is a formal device for specifying which strings are in the set. In particular, a *context-free grammar* is a cuadruple

(N, T, P, S), where N is a finite set of string categories; T a finite set of terminal symbols; P a finite set of productions or *rewriting rules* of the form $X \rightarrow \sigma$, $X \in N$, $\sigma \in (N \cup T)^*$; and S a distinguished symbol of N. A binary relation $\Rightarrow$ of *derivation* between strings over the vocalulary $N \cup T$ of the grammar is defined such that $\alpha X \beta \Rightarrow \alpha \sigma \beta$ iff $X \rightarrow \sigma$ is a production of P; now, $\Rightarrow^*$ may be defined as the reflexive and transitive closure of $\Rightarrow$. The *language generated* by the grammar, noted L(G), is the set of strings $\omega \in T^*$, such that $S \Rightarrow^* \omega$.

An *attribute grammar* is defined upon a context-free grammar G=(N, T, P, S), by associating each symbol $X \in N \cup T$ with a finite set A(X) of *attributes*, and a type or domain *dom(a)* for each attribute *a* thus defined (Knuth, 1968). Each attribute *a* of X, noted *X.a*, takes values over its domain and represents a specific, possibly context-sensitive property of the symbol.

Attribute values are defined by *attribution rules* of the form $X_i.a \leftarrow f(X_j.b, \ldots, X_k.c)$, associated with each production $p = X_0 \rightarrow X_1 \ldots X_n$ in the grammar, $0 \leq i,j,k \leq n$. Here, *f* is an applicative expression (function) whose value depends on the values of attribute occurrences associated with symbols in the production. Each time *p* applies in a derivation, the attribution rule defines the value of the attribute occurrence X.a as a function of the occurrences $X_j.b, \ldots, X_k.c$, associated with other symbols in *p* . We let R(p) denote the packet of attribution rules associated with *p* . The grammar may also define *attribute conditions* of the form $B(X_i.a, \ldots, X_k.b)$, $0 \leq i, k \leq n$, where *B* is a Boolean predicate on the values of attribute ocurrences in *p* . This condition must be satisfied in any derivation requiring the application of *p* , and thus contributes to the notion of grammaticality in the language generated by the grammar. We let B(p) denote the packet of attribute conditions associated with *p* .

The above remarks are summarized as follows: An *attribute grammar* is a cuadruple AG=(G,A,R,B), where

i. G = (N, T, P, S) is a context-free grammar;
ii. $A = \bigcup_{X \in N \cup T} A(X)$ is a finite set of attributes;
iii. $R = \bigcup_{p \in P} R(p)$ is a finite set of attribution rules, as above; and
iv. $B = \bigcup_{p \in P} B(p)$ is a finite set of attribute conditions, as above.

The base grammar G assigns a derivation tree τ to each sentence in L(G). The tree is annotated at each node labelled X with the set A(X) of attributes associated with X; each attribute $a \in A(X)$ defines an attribute occurrence X.a at node X. If the grammar is well defined (Knuth, 1968), it is possible to evaluate each attribute occurrence on the tree, and we say that τ is *correctly attributed* iff all attribute conditions yield 'true.' The *language* generated by the attribute grammar, L(AG), is now the subset of L(G) whose members have at least one correctly attributed tree.

It is possible to classify the attributes in AG according to the manner in which their values are defined. We say an attribute X.a is *synthesized* if its value depends only on attributes of daughters of X; it is *inherited* if its value depends on attributes associated with the parent or sisters of X. We say the grammar is *S-attributed* if it contains only synthesized attributes. A more general and practically important class of *L-attributed* grammars is obtained if we allow attributes of both kinds, but such that each inherited attribute depends only on inherited attributes of the parent, or attributes of the sisters to its left (Bochmann, 1976).

Earley's algorithm is a recognizer for CFGs which uses top-down prediction in combination with bottom-up parsing actions. Given an input string $x_1, \ldots, x_n$ it builds a state set S_i at each position *i* of the string, $0 \leq i \leq n+1$. Each *state* in S_i is of the form $<A \rightarrow \alpha \bullet \beta, f, \delta>$, where $A \rightarrow \alpha \bullet \beta$ is a dotted-production, *f* an index to the position in the input string where this instance of the production began to be recognized ($0 \leq f \leq i$), and δ a string of *k* symbols of lookahead ($k \geq 0$). To begin, all state sets are initialized to empty and the initial state $<\phi \rightarrow \bullet S \perp, 0, \perp^k>$ is put into S_0; here $\perp$ is the end-of-input marker. States are processed in order according to the position of their "dot" following three actions, Predictor, Completer, and Scanner, while maintaining the following invariant:

> State $< A \rightarrow \alpha \bullet \beta, f, \delta>$ is in S_i iff the following derivations are valid:
> $S \Rightarrow^* \sigma A \upsilon$; $\sigma \Rightarrow^* x \ldots x_f$; and $\alpha \Rightarrow^* x_{f+1} \ldots x_i$.

Since the number of possible states is finite the algorithm terminates. The input string is accepted if $S_{n+1} = \{<\phi \rightarrow S \perp \bullet, 0, \perp^k>\}$. The correctness of this acceptance condition is a consequence of the invariant.

3. Extension to S-attributed Grammars

The chief element of the extension of the algorithm is a change in the representation of the *states* in Earley's original algorithm to *attributed representations*. Now, each dotted production $A \rightarrow \alpha \bullet \beta$ in a state consists of symbols attributed according to the grammar. For each category symbol A in the base grammar, we define the attributed symbol $A[A.a_1, \ldots, A.a_n]$, where A is the category and $A.a_i$, $1 \leq i \leq n$, an attribute occurrence of A.

The extended algorithm, in addition to syntactically recognizing the input string, evaluates the attribution associated with each of its possible derivations. In particular, for each derivation of the attributed final state

$<\phi \to S[S.a_1,...,S.a_n] \perp\bullet, 0, \perp^k>$, where S is the start symbol of the grammar and $S.a_1,...,S.a_n$ the attribute occurrences of S associated with that state, the algorithm evaluates the corresponding attribute occurrences. For an S-attributed grammar, this is achieved by the following modification of Earley's algorithm, in its Completer step:

```
S0 := { $<\phi \to \bullet S[S.a_1,...,S.a_n]\perp, 0, \perp^k>$ };
for i := 1 to n do
begin
  For each state s in $S_i$, repeat until no more
   states may be added to $S_i$ or $S_{i+1}$
  begin
  1. *Predictor*
  If s = $< A \to \alpha \bullet X\beta, f, \delta>$
  (i.e., s is not final and X non-terminal)
     $S_i := S_i \cup \{ <X \to \bullet\sigma, i, \mu> \mid$
                   $X \to \sigma$ in P and $\mu$ in $FIRST_k(\beta\delta)\}$
  2. *Completer*
  If s = $< A \to \alpha\bullet, f, \delta>$ (i.e., s is final)
  and $\delta = x_{i+1} \dots x_{i+k}$
  a.  Ac := eval_s( A, $\alpha$, $A \to \alpha$)
  b.  $S_i := S_i \cup \{ < X \to \alpha Ac \bullet \beta, k, \mu> \mid$
                        $<X \to \alpha \bullet A\beta, k, \mu>$ in $S_f$ }
  3. *Scanner*
  If s = $< A \to \alpha \bullet x_{i+1}\beta, f, \delta>$
  (i.e., s not final and $x_{i+1}$ the next input symbol)
     $S_{i+1} := S_{i+1} \cup \{ < X \to \alpha x_{i+1} \bullet \beta, f, \delta> \}$
  end;
  If $S_{i+1}$ is empty, reject and terminate
end;
If $<\phi \to S[S.a_1,...,S.a_n]\perp\bullet, 0, \perp^k>$ in $S_{n+1}$, accept.
```

Extension of Earley's Algorithm for S-attributed Grammars

The states in the algorithm are attributed as indicated above. For example, the symbol A in the state $<A \to \alpha\bullet, f, \delta>$ input to the Completer could be shown more explicitly as $A=A[A.a_1, ..., A.a_n]$. As the state enters the completer, the attribute occurrences $A.a_i$ of A are unevaluated; however, since the grammar is S-attributed it is easy to show that the attribute occurrences on the right-hand side α of the production have already been evaluated. Hence, evaluation of the attribute occurrences of A reduces to application of the attribution associated with the production $A \to \alpha$, according to the attribute values in α. This is done by the function *eval_s(A, α, $A \to \alpha$)*, which returns the attributed symbol Ac, identical to A, except that its attribute occurrences have been evaluated, as required.
The last state set generated by the algorithm contains final states of the form $<\phi \to S[\ ...] \perp\bullet, 0, \perp^k>$, in which the attributed start symbol S[...] is already evaluated. Here the extended algorithm differs form Earley's; whereas the original algorithm generates at most one final state, regardless of the ambiguity of the underlying grammar, the extended algorithm may generate several instances of this state, if the grammar is ambiguous. Each instance of the final state corresponds to a different derivation of the initial symbol, leading to a different evaluation of the symbol's attributes.

4. Finite Partitioning of Attribute Domains

The last remark in the extension of section 3 shows a defect of the Extended Algorithm: It may not terminate in the general case. For the S-attributed case, however, this may happen only if the underlying grammar is infinitely ambiguous or, equivalently, if it has *cycles* or derivations of the form $A \Rightarrow^+ A$, for some $A \in N$.

Consider, for example, the following grammar, which "measures" the length of each derivation of the sole string 'a' it generates:

	<u>attribution</u>:
$S \to A$	$S.v \leftarrow A.v$
$A \to A$	$A_0.v \leftarrow A_1.v + 1$
$A \to a$	$A.v \leftarrow 1$

Given the input string 'a', the algorithm defines three attributted state sets:

S_0 = {$<\phi \to \bullet S[v]\perp, 0, \perp^k>$, $<S[v] \to \bullet A[\ v], 0, \perp^k>$,
$<A[v] \to \bullet a, 0, \perp^k>$, $<A[\ v] \to \bullet A[v], 0, \perp^k>$ }

S_1 = {$<A \to a\bullet, 0, \perp^k>$,
$<S[v] \to A[1]\bullet, 0, \perp^k>$, $<A[v] \to A[1]\bullet, 0, \perp^k>$,
$<\phi \to S[1]\bullet\perp, 0, \perp^k>$
$<S[v] \to A\bullet[2], 0, \perp^k>$, $<A[v] \to A[2]\bullet, 0, \perp^k>$,
ad infinitum ... }

S_2 = {$<\phi \to S[1]\perp\bullet, 0, \perp^k>$,
$<\phi \to S[2]\perp\bullet, 0, \perp^k>$,
ad infinitum ... }

Since S_1 is infinite, the algorithm does not terminate.

Cyclic grammars play an important role in most recent linguistic theories, including Government-binding (GB), Lexical-Functional Grammar (LFG) and GPSG (cf. Berwick, 1988; Correa, 1987b; Kornai and Pullum, 1990). These have in common that they have shifted from rule-based descriptions of language, to declarative or *principle-based* descriptions, in which the role of phrase structure rules or principles is relatively minor. Thus, to make the extension of the algorithm useful for natural language applications it becomes necessary to ensure its termination, in spite of cyclic bases.

The termination of the Extended Algorithm may be guaranteed while maintaining its full generality, through a *finite partition* on the attribute domains associated with each cyclic symbol in the grammar. For each such domain *dom* (*a*), the partition defines a finite collection of equivalence classes on attribute values. Now, before adding a new state $<A \rightarrow \alpha \bullet \beta, f, \delta>$ to a state set S_i, we test for *equivalence* (according to the defined partitions) rather than equality to some previously added state; if the new state is equivalent to some other, it is not added. It is easy to show that the number of attributed dotted items in the grammar, and hence the size of the state sets, is now finite. This number is in fact identical to that of Earley's algorithm, except for a constant multiplicative factor, dependent on the grammar and the size of the partitions selected for attribute domains. Since the size of the state sets possible with finite partitioning is now finite, the algorithm always terminates.

After establishing a correspondence between attribute and unification grammar (UG), we may see that the technique of "restriction" used by Shieber (1985) in his extended algorithm is related to finite partitioning on attribute domains, in fact a particular case which takes advantage of the more structured attribute domains of UG. For attribute grammar, given that the domains involved are more general (e.g., the integers), finite partitioning is the required device.

5. Conclusions and Implementation Status

This paper presented and extension of Earley's algorithm to S-attributed grammars. Combining on-line semantic evaluation with the execution of syntactic actions, the algorithm is an effective realization of attribute-directed parsing, as proposed by Watt (1980) and Jones and Madsen (1980). Although the algorithm is a recognizer, it computes the semantic values associated with each derivation of the input string, and hence need not be extended to compute tree representations. In attribute grammars with conditions on productions, the values of attributes already evaluated may be used to guide the parsing process, reducing the number of states that may be generated by the algorithm.

The extension of the algorithm has been written in "C", using an efficient "C" implementation of Earley's original algorithm (Chamorro and Correa, 1990), and is currently being tested on small grammars. The extended algorithm will be the kernel of ANDES-I, a programming environment for attribute grammars, intended for natural language applications.

References

Berwick, Robert. 1988. "Principle-based Parsing". MIT A.I. Memo 972, revised. MIT, Cambridge, MA.

Bochmann, Gregor. 1976. "Semantic Evaluation from Left to Right." *Communications of the ACM* , Vol. 19, No. 2, p. 55-62.

Chamorro, Miriam, and N. Correa. 1990. "An Efficient 'C' Implementation of Earley's Algorithm" - in Spanish. CIFI, Universidad de los Andes, Bogotá, Colombia.

Correa, Nelson. 1987a. "An Attribute Grammar Implementation of Government-binding Theory." *Proceedings of the 25th Annual Meeting of the ACL* , Stanford University, Stanford, CA.

Correa, Nelson. 1987b. "Empty Categories, Chain Binding, and Parsing." Parsing Seminar; *MIT Working Papers of the Lexicon Project*, MIT, Cambridge, MA.

Correa, Nelson. Forthcoming. Attribute and Unification Grammar: A Review of Formalisms and Comparision. CIFI, Universidad de los Andes.

Earley, Jay. 1970. "An Efficient Context-free Parsing Algorithm." *Communications of the ACM* , Vol. 13, No. 2, p. 94-102.

Fu, K.S. 1982. *Syntactic Pattern Recognition* . Academic Press, New York.

Jones, Neil D., and M. Madsen. 1980. "Attribute Influenced LR Parsing." In N. D. Jones, ed., *Semantics-directed Compiler Generation* , LNCS 94. Springer-Verlag, New York.

Kay, Martin. 1985. Parsing in Functional Unification Grammar. In D. Dowty, L. Karttunen, and A. Zwicky, eds., *Natural Language Parsing* , Cambridge University Press, Cambridge, England.

King, Margaret, ed. 1983. *Natural Language Parsing* . Academic Press, New York.

Knuth, Donald. 1968. "Semantics of Context-free Languages." Mathematical Systems Theory, Vol. 2, No. 2, p. 127-145.

Kornai, Andras, and G. Pullum. 1990. "The X-bar Theory of Phrase Structure." Language, Vol. 66.

Pereira, Fernando, and D. H. Warren. 1983. "Parsing as Deduction." *Proceedings of the 21st Annual Meeting of the ACL* , MIT, Cambridge, MA.

Shieber, Stuart. 1985. "Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms." *Proceedings of the 23rd Annual Meeting of the ACL* , University of Chicago, Chicago, IL.

Shieber, Stuart. 1986. *An Introduction to Unification Based Approaches to Grammar* . CSLI Lecture Notes No. 4, Stanford, CA.

Watt, David. 1980. "Rule Splitting and Attribute Directed Parsing." In N.D.Jones, ed., *Semantics-directed Compiler Generation* , LNCS 94. Springer-Verlag, NY.

Yellin, Daniel. 1988. "Generalized Attributed Parsing." Manuscript; IBM Research, Yorktown Heights, NY.