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

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 quadruple...
(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 → σ, X ∈ N, σ ∈ (N ∪ T)*; and S a distinguished symbol of N. A binary relation ⇒ of derivation between strings over the vocabulary N ∪ T of the grammar is defined such that αXβ ⇒ ασβ iff X → σ is a production of P; now, ⇒* may be defined as the reflexive and transitive closure of ⇒. The language generated by the grammar, noted L(G), is the set of strings wε T*, such that S ⇒* w.

An attribute grammar is defined upon a context-free grammar G=(N, T, P, S), by associating each symbol Xε N∪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.a← f(X1.b, ..., Xk.c), associated with each production p=Xr→X1...Xn in the grammar. 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 Xj.b, ..., Xk.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(X1.a, ..., Xk.b), 0 ≤ i, k ≤ n, where B is a Boolean predicate on the values of attribute occurrences 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 quadruple AG=(G,A,R,B), where

1. G = (N, T, P, S) is a context-free grammar;
2. A = ∪Xε N∪T A(X) is a finite set of attributes;
3. R = ∪pε P R(p) is a finite set of attribution rules, as above; and
4. B = ∪pε 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 aε 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 x1,...,xn it builds a state set S1 at each position i of the string, 0 ≤ i ≤ n+1. Each state in S1 is of the form <A→αβ, f, δ>, where A→αβ 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 ≤ f ≤ i), and δ a string of k symbols of lookahead (k≥0). To begin, all state sets are initialized to empty and the initial state <ϕ→S⊥, 0, ⊥k> is put into S0; here ⊥ 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→αβ, f, δ, > is in Si iff the following derivations are valid:
S ⇒* σAσ; σ ⇒* x ...xf; and α ⇒* xf+1...xi.

Since the number of possible states is finite the algorithm terminates. The input string is accepted if Sn+1={<ϕ→S⊥, 0, ⊥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→αβ 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.a1, ..., A.an], where A is the category and A.ai, 1≤i≤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
\(<A\rightarrow S[a,\ldots, a_n] \perp \ast, 0, \perp >\), where \(S\) is the start symbol of the grammar and \(S[a_1,\ldots, a_n]\) 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:

\[
S_0 := \{ < \phi \rightarrow S[a_1,\ldots, a_n] \perp \ast, 0, \perp > \};
\]

**begin**

For each state \(s\) in \(S_1\), repeat until no more states may be added to \(S_1\) or \(S_{i+1}\)

1. **Predictor**
   If \(s = < A \rightarrow a, f, \delta >\) (i.e., \(s\) is not final and \(X\) non-terminal)
   \(S_1 := S_1 \cup \{ < X \rightarrow \phi, f, \delta >\}
   \)
   \(X \rightarrow a\) in \(P\) and \(f = \text{FIRST}_k(\delta)\)

2. **Completer**
   If \(s = < A \rightarrow a, f, \delta >\) (i.e., \(s\) is final)
   and \(X = X_i + X_{i+1}\) \(\ldots\)
   \(A = A \rightarrow a\) \(\rightarrow A\)
   \(\delta = x_{i+1} \ldots x_{i+k}\)
   \(A_e := \text{eval}_s(A, \alpha, A \rightarrow a)\)
   \(\leq \delta\)
   \(A_e = A \rightarrow a\) \(\rightarrow A\)
   \(\phi = \text{FIRST}_k(\delta)\)

3. **Scanner**
   If \(s = < A \rightarrow a, f, \delta >\) (i.e., \(s\) is not final and \(X\) the next input symbol)
   \(S_{i+1} := S_{i+1} \cup \{ < X \rightarrow A_e, f, \delta >\}
   \)
   \(X \rightarrow a\) \(\rightarrow A\)

**end**

If \(S_{i+1}\) is empty, reject and terminate

If \(< \phi \rightarrow S[a_1,\ldots, a_n] \perp \ast, 0, \perp >\) 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 \(<A \rightarrow a, f, \delta >\) input to the Completer could be shown more explicitly as \(A = A[a, a_1,\ldots, a_n]\). As the state enters the completer, the attribute occurrences \(A[a, a_1,\ldots, a_n]\) are univaluated; however, since the grammar is \(S\)-attributed it is easy to show that the attribute occurrences on the right-hand side \(A[a, a_1,\ldots, a_n]\) of the production have already been evaluated. Hence, the evaluation of the attribute occurrences of \(A\) reduces to the application of the attributed association \(A[a, a_1,\ldots, a_n]\) to the attribute values in \(\alpha\). This is done by the function \(\text{eval}_s(A, \alpha, A \rightarrow a)\), which returns the attributed symbol \(A_e\), identical to \(A[a, a_1,\ldots, a_n]\), except that its attribute occurrences have been evaluated, as required.

The last state set generated by the algorithm contains final states of the form \(<\phi \rightarrow S[\ldots] \perp \ast, 0, \perp >\), in which the attributed start symbol \(S[\ldots]\) 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:

\[
S \rightarrow A \quad S.v \leftarrow A.v
\]

\[
A \rightarrow A \quad A_{0,v} \leftarrow A_{1,v} + 1
\]

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

\[
S_0 = \{ < \phi \rightarrow S[1] \perp \ast, 0, \perp >, < S[1] \rightarrow A[1] \perp \ast, 0, \perp >, < A[1] \rightarrow A[v] \perp \ast, 0, \perp > \}
\]

\[
S_1 = \{ < A \rightarrow a, 0, \perp >
\]

\[
< S[v] \rightarrow A[1] \perp \ast, 0, \perp >, < A[v] \rightarrow A[v] \perp \ast, 0, \perp >,
\]

\[
< A[1] \rightarrow A[2] \perp \ast, 0, \perp >, < A[2] \rightarrow A[3] \perp \ast, 0, \perp >,
\]

\[
\text{ad infinitum ...}
\]

\[
S_2 = \{ < \phi \rightarrow S[1] \perp \ast, 0, \perp >
\]

\[
< \phi \rightarrow S[2] \perp \ast, 0, \perp >,
\]

\[
\text{ad infinitum ...}
\]

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 \( \langle A \rightarrow \alpha \beta, f, \delta \rangle \) 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.