A Modification of the Earley-Shieber Algorithm for Direct Parsing of ID/LP Grammars

James Kilbury
Technische Universität Berlin
Institut für Angewandte Informatik
Projektgruppe KIT, Sekr. FR 5-8
Franklinstr. 28/29, D-1000 Berlin 10

1. Introduction

Immediate Dominance/Linear Precedence (ID/LP) Grammar is a formalism that has recently been developed within the framework of Generalized Phrase Structure Grammar (GPSG) as presented by Gazdar/Pullum (1982). The carefully described formal properties and the restricted power of GPSG have made it a focal point of interest in the parsing of natural language. Of the various devices introduced in GPSG only the ID/LP formalism will be discussed in this paper. [1]

The basic idea of the ID/LP formalism is straightforward. A context-free phrase-structure rule A --> α, where A is a nonterminal symbol and α is a string of nonterminal and terminal symbols, contains information of two kinds: first, the members of α are identified as successors of A, and second, the linear order of the members of α is specified. An ID/LP grammar states this information separately in immediate dominance and linear precedence rules, respectively, but it retains the power of a context-free phrase-structure (i.e. type 2) grammar. The linguistic motivation for ID/LP grammar arises from its capacity to express generalizations about word order that are not statable with context-free rules. It offers solutions to problems involving so-called free word order, where the variable order of constituents is not reflected in the semantic representation of the corresponding phrase (cf Uszkoreit 1983).

The algorithm of Earley (1970) for parsing context-free (type 2) languages combines the techniques and advantages of top-down and bottom-up parsing and constitutes the most efficient practical parsing algorithm known [2] for this language class. Shieber (1984) adapts Earley's algorithm to the ID/LP formalism and retains its essential parsing strategy.

---

[1] My thanks go to Thomas Christaller, Roger Evans, Gerald Gazdar, Christopher Habel, Camilla Schwind, Hans Uszkoreit, Bernhard Zimmermann, and two anonymous referees for comments related to this paper. Kilbury (1984a) contains an extended discussion of the ideas presented here, while Kilbury (1984b) describes a system which uses the modified Earley-Shieber parser.

[2] The algorithm of Valiant (1975) is slightly more efficient than Earley's under worst-case conditions but does not appear to be practical for parsing natural languages.

The objective of this paper is to present a genuine modification of the Earley algorithm. Although the version given here is adjusted to the ID/LP formalism, the essential modification involves the so-called predictor and applies equally to Earley's algorithm itself; it promises to make the algorithm more efficient for parsing with large grammars (see below). To facilitate comparison, this presentation closely follows that of Shieber (1984) and Aho/Ullman (1972). The algorithm is kept as free as possible of complications in order to highlight its structure and the essential modification. Finally, an implementation of the algorithm in PROLOG is given.

2. ID/LP Grammars

For an ID/LP grammar $G = \langle N, T, ID, LP, S \rangle$

- N is a finite set of nonterminal symbols,
- T is a finite set of (pre)terminal symbols,
- ID is a finite set of numbered immediate dominance rules,
- LP is a finite set of linear precedence rules, and
- S is a designated start (root) symbol in N.

To simplify this presentation, no distinction is made between terminal and preterminal symbols. In grammars of natural languages the latter are understood as nonterminal symbols that are directly expanded to a terminal symbol (i.e. as lexical categories).

An ID rule is a triple $\langle k, A, \alpha \rangle$ with an integer k (the rule number), $A \in N$, and $\alpha \in 2^{(N \cup T)} \setminus \emptyset$ (i.e., α is a nonempty set of terminal and nonterminal symbols). In the special GPSG notation where $\alpha = \{a_1, \ldots, a_n\}$ we write

$$k: A \longrightarrow a_1, \ldots, a_n$$

where the integer and colon may be omitted so that the numbering is specified implicitly by the order of presentation.

LP is a transitive, asymmetric, and irreflexive relation $\{\langle a, b \rangle\} \subset (N \cup T) \times (N \cup T)$. In GPSG notation we write $a < b < \ldots < f$ where $\{\langle a, b \rangle, \langle b, \ldots \rangle, \ldots, \langle \ldots, f \rangle\} \subseteq LP$. The relation defines a partial ordering over the direct successors of a node.

We also define $a \leq b \iff \neg (b < a)$ and extend the relations to sets and strings: if δ is the set {b, c, ..., f} or the string bc...f, then $a < \delta$ iff $a < b \wedge a < c \wedge \ldots \wedge a < f$. We stipulate $\neg(\emptyset < b)$ and thus $b \leq \emptyset$.

The language L(G) generated by an IDLPG is defined using the relation $\leq$ and the following definitions (cf Shieber 1984:139). For $\alpha = a_1 \ldots a_n$ with $a_i \in (N \cup T)$ for all $1 \leq i \leq n$

(1) LP-acceptable(α) iff $a_i \leq a_j$ for all $1 \leq i \leq j \leq n$,

(2) permute(α) = {β | card(α) = n and $\beta = b_1 \ldots b_n$ with $b_i \in \alpha$ for all $1 \leq i \leq n$ },

(3) expand(α) = {β | $\beta \in$ permute(α) and LP-acceptable(β)}.

The remaining definitions parallel standard usage:

(4) w_1 ---> w_2 (w_1 directly derives w_2) iff
$w_1 \in (N \cup T)^*$, $w_1 = \alpha B \beta$, $w_2 = \alpha\beta\gamma$, <k, B, β'> $\in$ ID, and $\beta \in$ expand(β');

(5) $\xrightarrow{*}$ is the reflexive and transitive closure of ---> ;

(6) L(G) = {w | w $\in$ T* and S $\xrightarrow{*}$ w }.

3. The algorithm for recognition and parsing

We take the following as given:

(1) an ID/LP grammar G without
a) deletion (i.e., rules of the form A --> ε are not allowed) or
b) identical recursion such that for some A, B $\in$ N, A $\xrightarrow{*}$ B and B $\xrightarrow{*}$ A;

(2) a "first" relation F = {<B, k>}, where B $\in (N \cup T)$, k is an integer, <k, A, {B} $\cup$ β> $\in$ ID for some A and β, B $\notin \beta$, and B $\leq \beta$ (i.e., LP allows B to occur as the left-most successor of A); [3]

(3) a string of terminal symbols $w = a_1 \ldots a_n$ over T*.

The algorithm proceeds by the construction of a sequence of item lists $I_1, \ldots, I_n$ where each item in I_i is a quadruple [A, α, β, k] such that
- α is a string consisting of successors of A in the order in which they were identified,
- α' is the set whose members form α,
- β is the set of successors of A that remain to be identified,
- <n, A, $\alpha' \cup \beta$> $\in$ ID and $\alpha' \cap \beta = \emptyset$,
- k (with $k \leq i$) is a "back pointer" identifying the item list I_k in which the recognition of the production <n, A, $\alpha' \cup \beta$> was begun.

An item [A, α, β, k] corresponds to an item [A --> α.β, k] in the notation of Aho/Ullman (1972:320ff) for Earley's algorithm.

---

[3] The constraint B $\notin \beta$ here and below follows from the assumption that the successors of a node constitute a set, i.e., a node cannot have identical successors. The grammar definition and algorithm can easily be modified to cope with identical successors (cf Shieber 1984:143), but the constraint is reasonable for linguistic grammars in which nonterminal symbols bear parameters for features, semantic representations, or derivation trees, since the parametrized successors of a node will be distinct.

To construct item lists I_1, ..., I_n build I_i given a_i ($1 \leq i \leq n$) as follows:

(1) scanner:

If $i > 1$ then for each item $[A, \alpha, \{a\} \cup \beta, k] \in I_{i-1}$ such that $a = a_i$, $a \leq \beta$, and $a \notin \beta$, add the item $[A, \alpha a, \beta, k]$ to I_i.

(2) predictor for terminal symbols:

For all $\langle a, k\rangle \in F$ such that $a = a_i$, $\langle k, A, \{a\} \cup \beta\rangle \in ID$, and $a \notin \beta$, add the item $[A, a, \beta, (i-1)]$ to I_i.

After (1) and (2) repeat (3) and (4) until no new items can be added to I_i:

(3) completer:

For all $[B, \gamma, \emptyset, j] \in I_i$ and for all $[A, \alpha, \{B\} \cup \beta, k] \in I_j$ such that $B \leq \beta$ and $B \notin \beta$, add the item $[A, \alpha B, \beta, k]$ to I_i.

(4) predictor for nonterminal symbols:

For all $[B, \gamma, \emptyset, j] \in I_i$ and for all $\langle B, k\rangle \in F$ such that $\langle k, A, \{B\} \cup \beta\rangle \in ID$ and $B \notin \beta$, add the item $[A, B, \beta, j]$ to I_i.

A string w is recognized iff there exists at least one item $[S, \alpha, \emptyset, 0] \in I_n$ where S is the start symbol.

With slight modification the algorithm can be used for parsing. Let each $A \in (N \cup T)$ bear a parameter for its syntactic derivation tree, constant for terminal symbols and variable for nonterminals. In an item $[A(x), \alpha, \emptyset, k]$ the parameter x of A is instantiated iff the parameter of each symbol in α is instantiated. If x is instantiated in some $[S(x), \alpha, \emptyset, 0] \in I_n$ for w, then it is a derivation tree for w. If w is syntactically ambiguous to the degree k, then there are k distinct instantiations of x in k distinct items in I_n. [4] The same extension from recognition to parsing can be made with the Earley and Earley-Shieber algorithms.

The algorithm has been stated in this form in order to facilitate comparison with the presentations in Shieber(1984:141) and Aho/Ullman (1972:321); note the identity of step (1) with their step (4) and step (3) with their step (5). The principal modification lies in the predictor of their steps (1), (3), and (6). The predictor of Earley's algorithm searches the entire grammar for potentially applicable productions before each input symbol is read, while the predictor here uses the relation F to introduce items after a terminal or nonterminal symbol has been identified. The algorithm here thus amounts to a combination of Earley's algorithm with a left-corner parser (cf Ross 1982) made more efficient through the use of the relation F. [5]

---

[4] Alternatively, items can be augmented with a fifth term in which the derivation tree is constructed.

[5] The variant of Earley's algorithm presented in Pulman (1983:118) also incorporates a left-corner technique (although without a first relation) and occupies a position between Earley and the algorithm here, whose predictor

Earley's algorithm is costly precisely because it must anticipate each possible following phrase and (pre)terminal symbol throughout the analysis. For a natural language the number of possibilities can be enormous; consider e.g. the variety of constituents with which an English or German sentence may begin. The modified version first identifies a constituent and then -- rather than searching the entire grammar as in a normal left-corner parser -- uses the relation F to find exactly those productions by which the identified constituent may be introduced as left-most successor. An increase in practical efficiency is thus achieved in comparison both with the Earley algorithm and the left-corner parser. A formal analysis of the efficiency would also depend crucially on properties of the particular grammar.

The Earley-Shieber algorithm is more general, however, since it accepts grammars with deletion and identical recursion; these restrictions are linguistically motivated but cannot be discussed here.

The modified algorithm may be stated more elegantly by generalizing the predictor for terminal and nonterminal symbols as follows:

```
program PARSE a_1 ... a_n:
begin
for i := 1 to n do
   begin
   enter item [a_i, a_i, ∅, (i-1)] in I_i;
   CLOSURE for item [a_i, a_i, ∅, (i-1)] in step i
   end
end;

procedure CLOSURE for item [B, γ, ∅, j] in step i:
begin
COMPLETE for item [B, γ, ∅, j] in step i;
PREDICT for item [B, γ, ∅, j] in step i
end;

procedure COMPLETE for item [B, γ, ∅, j] in step i:
begin
if i > 1 then for each [A, α, {B} ∪ β, k] ∈ I_j such that B ≤ β and B ∉ β
   begin
   enter item [A, αB, β, k] in I_i;
   if β = ∅ then CLOSURE for item [A, αB, ∅, k] in step i
```

---

introduces fewer superfluous items than that of Pulman. His suggestions for further improvements of the algorithm (122ff) are misleading, however, since they amount to a radical restriction of the class of accepted grammars.

```
    end
end;

procedure PREDICT for item [B, γ, ∅, j] in step i:
begin
for each <B, k> ∈ F where <k, A, β> ∈ ID do
   begin
   enter item [A, B, β \ {B}, j] in I_i;
   if β \ {B} = ∅ then CLOSURE for item [A, B, ∅, j] in step i
   end
end.
```

The termination of PARSE is established in two steps: (1) In COMPLETE, $A \in N$ since A is in an item entered by CLOSURE. Therefore $\alpha \neq \emptyset$ and $/\alpha/ \geq 1$, so $k \leq (j-1)$, i.e., in each recursive cycle j is reduced by at least 1 since the new value of j is the old value of k. Since $k \geq 0$, COMPLETE terminates in at most j steps. (2) By assumption, G has no identical recursion (see above). Since N and ID are finite, PREDICT terminates after a finite number of items have been added to I_i:

$[N_1, B, \emptyset, j]$
$[N_2, N_1, \emptyset, j]$
...
$[N_{n-1}, N_{n-2}, \emptyset, j]$
$[N_n, N_{n-1}, \beta, j]$,

where $N_n = S$ and/or $\beta \neq \emptyset$.

4. Implementation in PROLOG

4.1 The modified Earley-Shieber algorithm

The algorithm has been implemented in Waterloo PROLOG, Version 1.4, and runs on an NAS AS/7031 (IBM 370) with IBM VM/SP operating system.

Two built-in predicates deserve comment. The predicate addax adds an argument having the form of an axiom to the set of axioms in the active workspace. The predicate axn is used here with the format

axn(<name>, <nargs>, <axiom>, <index>)

to unify <axiom> with an axiom from the database that has the predicate name <name> and the number of arguments <nargs>; if <axiom> is the nth such axiom, then <index> is the integer n.

Although they are formally defined as quadruples above, the items are represented with 5-place predicates whose first argument is the number of the corresponding item list. Input and output predicates have been omitted from the listing. Note that the string abc and the set {a, b, c} are both implemented as the PROLOG list a.b.c.nil here. Since the program closely parallels the final version of the algorithm given above, no further documentation is provided.

```
parse(*inputlist) <-
     parse1(1, *inputlist) &
     output.

parse1(*i, nil).
parse1(*i, *a.*rest) <-
     diff(*i, 1, *h) &
     addax( item(*i, *a, *a.nil, nil, *h) ) &
     closure(*i, item(*i, *a, *a.nil, nil, *h) ) &
     sum(*i, 1, *j) &
     parse1(*j, *rest).

closure(*i, *item) <-
     complete(*i, *n1, *item) &
     predict(*i, *n2, *item).

complete(1, *n, *item).
complete(*i, *n, *item) <-
     scan(*i, *n, *item) &
     sum(*n, 1, *m) &
     complete(*i, *m, *item).
complete(*i, *n, *item).

scan(*i, *n, item(*list, *b, *gamma, nil, *j) ) <-
     axn(item, 5, item(*j, *a, *alpha, *theta, *k), *n) &
     /* nth item in list j */
     complete_item(*i, *b, item(*j, *a, *alpha, *theta, *k) ).

complete_item(*i, *b, item(*list, *a, *alpha, *theta, *k) ) <-
     test_daughters(*theta, *b, *beta) &
     append(*alpha, *b.nil, *chi) &
     addax( item(*i, *a, *chi, *beta, *k) ) &
     completed( item(*i, *a, *chi, *beta, *k) ).
complete_item(*i, *b, *item).

predict(*i, *n, *item) <-
     predict_item(*i, *n, *item) &
     sum(*n, 1, *m) &
     predict(*i, *m, *item).
predict(*i, *n, *item).

predict_item(*i, *n, item(*list, *b, *gamma, nil, *j) ) <-
     axn(f, 2, f(*b, *k), *n) &
     rule(*k, *a, *theta) &
     minus(*theta, *b, *beta) &
     addax( item(*i, *a, *b.nil, *beta, *j) ) &
     completed( item(*i, *a, *b.nil, *beta, *j) ).

completed( item(*i, *a, *chi, nil, *k) ) <-
     closure(*i, item(*i, *a, *chi, nil, *k) ).
completed(*item).

minus(*b.*delta, *b, *delta).
minus(*head.*tail, *b, *head.*delta) <- minus(*tail, *b, *delta).

test_daughters(*b.*beta, *b, *beta) <- can_precede(*b, *beta).
test_daughters(*head.*tail, *b, *head.*beta) <-
     can_precede(*b, *head.nil) &
     test_daughters(*tail, *b, *beta).

can_precede(*b, nil).
can_precede(*b, *head.*tail) <-
     lp(*head, *b) &
     / & fail.
can_precede(*b, *head.*tail) <- can_precede(*b, *tail).
```

4.2 Test grammar

The following grammar, presented in GPSG notation, has been used for testing:

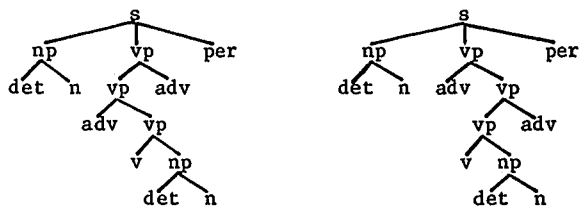
```
/*  ID rules                LP rules          */
1: s  --> np, vp, per.      np < vp < per.
2: np --> pn.
3: np --> det, n.           det < n.
4: vp --> vp, adv.
5: vp --> v, np.            v < np.
6: vp --> v.
```

A transducer has been implemented in PROLOG that translates the source grammar given above into the following object grammar that serves as input for the parser:

```
rule(1, s, np.vp.per.nil).  /* ID rules */
rule(2, np, pn.nil).
rule(3, np, det.n.nil).
rule(4, vp, vp.adv.nil).
rule(5, vp, v.np.nil).
rule(6, vp, v.nil).
lp(np, vp).  /* LP rules */
lp(vp, per).
lp(det, n).
lp(v, np).
start(s).  /* start symbol */
f(np, 1).  /* first relation */
f(pn, 2).
f(det, 3).
f(vp, 4).
f(adv, 4).
f(v, 5).
f(v, 6).
```

4.3 Test string and derivation trees

The string w = det.n.adv.v.det.n.adv.per.nil (corresponding to an English sentence like The girl smugly wrote a program today.) has been used for testing. It is syntactically ambiguous and has two derivation trees corresponding to the first expansion of vp with left or right recursion in rule 4:



(left recursion) (right recursion)

4.4 Test run

Given the test string and test grammar as input, the program of section 4.1 produces the following list of items. To their right the numbers of the source items and rules with which they are derived have been added.

1	item(1, det, det.nil, nil, 0)	-
2	item(1, np, det.nil, n.nil, 0)	1 & r_3
3	item(2, n, n.nil, nil, 1)	-
4	item(2, np, det.n.nil, nil, 0)	3
5	item(2, s, np.nil, vp.per.nil, 0)	4 & r_1
6	item(3, adv, adv.nil, nil, 2)	-
7	item(3, vp, adv.nil, vp.nil, 2)	6 & r_4
8	item(4, v, v.nil, nil, 3)	-
9	item(4, vp, v.nil, np.nil, 3)	8 & r_5
10	item(4, vp, v.nil, nil, 3)	8 & r_6
11	item(4, vp, adv.vp.nil, nil, 2)	10 & 7
12	item(4, s, np.vp.nil, per.nil, 0)	11 & 5
13	item(4, vp, vp.nil, adv.nil, 2)	11 & r_4
14	item(4, vp, vp.nil, adv.nil, 3)	10 & r_4
15	item(5, det, det.nil, nil, 4)	-
16	item(5, np, det.nil, n.nil, 4)	15 & r_3
17	item(6, n, n.nil, nil, 5)	-
18	item(6, np, det.n.nil, nil, 4)	17 & 16
19	item(6, vp, v.np.nil, nil, 3)	18 & 9
20	item(6, vp, adv.vp.nil, nil, 2)	19 & 7
21	item(6, s, np.vp.nil, per.nil, 0)	20 & 5
22	item(6, vp, vp.nil, adv.nil, 2)	20 & r_4
23	item(6, vp, vp.nil, adv.nil, 3)	19 & r_4
24	item(6, s, np.nil, vp.per.nil, 4)	18 & r_1
25	item(7, adv, adv.nil, nil, 6)	-
26	item(7, vp, vp.adv.nil, nil, 2)	25 & 22
27	item(7, s, np.vp.nil, per.nil, 0)	26 & 5
28	item(7, vp, vp.nil, adv.nil, 2)	26 & r_4
29	item(7, vp, vp.adv.nil, nil, 3)	25 & 23
30	item(7, vp, adv.vp.nil, nil, 2)	29 & 7
31	item(7, s, np.vp.nil, per.nil, 0)	30 & 5
32	item(7, vp, vp.nil, adv.nil, 2)	30 & r_4
33	item(7, vp, vp.nil, adv.nil, 3)	29 & r_4
34	item(7, vp, adv.nil, vp.nil, 6)	25 & r_4
35	item(8, per, per.nil, nil, 7)	-

```
36 item(8, s, np.vp.per.nil, nil, 0)    35 & 27  *** recognized ***
37 item(8, s, np.vp.per.nil, nil, 0)    35 & 31  *** recognized ***
```

5. References

Aho, Alfred V./Ullman, Jeffrey D. (1972): The Theory of Parsing, Translation, and Compiling: Volume 1, Parsing. Prentice-Hall: Englewood Cliffs, N.J.

Earley, Jay (1970): "An Efficient Context-Free Parsing Algorithm." Communications of the ACM 13.94-102.

Gazdar, Gerald/Pullum, Geoffrey K. (1982): Generalized Phrase Structure Grammar: A Theoretical Synopsis. Cognitive Science Research Paper 007, University of Sussex.

Kilbury, James (1984a): Earley-basierte Algorithmen fur direktes Parsen mit ID/LP-Grammatiken. KIT-Report 16, TU Berlin.

Kilbury, James (1984b): "GPSG-Based Parsing and Generation," in C.-R. Rollinger (ed.), Probleme des (Text-) Verstehens - Ansätze der Künstlichen Intelligenz, 67-76. Tübingen: Niemeyer.

Ross, Kennth M. (1982): "An Improved Left-Corner Parsing Algorithm," Proceedings of COLING 82, 333-338.

Shieber, Stuart M. (1984): "Direct Parsing of ID/LP Grammar." Linguistics and Philosophy 7.135-154.

Pulman, Stephen G. (1983): "Generalised Phrase Structure Grammar, Earley's Algorithm, and the Minimisation of Recursion," in K. Sparck Jones and Y. Wilks (eds.), Automatic Natural Language Processing, 117-131. Chichester: Harwood.

Uszkoreit, Hans (1983): "A Framework for Processing Partially Free Word Order," Proceedings of the 21st Annual Meeting of the ACL, 106-112. Cambridge, Mass.

Valiant, Leslie G. (1975): "General Context Free Recognition in Less Than Cubic Time." Journal of Computer and System Sciences 10.308-315.