On Kilbury's Modification of Earley's Algorithm

HANS LEISS
Siemens AG

We improve on J. Kilbury's proposal to interchange "predictor" and "scanner" in Earley's parser. This modification of Earley's parser can trivially be combined with those suggested by S. Graham, M. Harrison, and W. Ruzzo, leading to smaller parse tables and almost the power of lookahead 1. Along these lines we can also obtain Earley-parsers having partial lookahead $r > 1$, without storing right contexts. Parse trees with shared structure can be stored in the parse tables directly, rather than constructing the trees from "dotted rules."

Categories and Subject Descriptors: D.3.4 [**Programming Languages**]: Processors—*parsing*; F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems—*parsing*; I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*language parsing and understanding*

General Terms: Algorithms, Languages

Additional Key Words and Phrases: Context-free grammars, Earley's algorithm, Graham–Harrison–Ruzzo's algorithm, inductive definitions, lookahead, representation of parse trees, structure sharing, Tomita's packed forest representation

1. INTRODUCTION

Using context-free grammars for the description of natural languages has regained much attention in theoretical linguistics since 1980 [4]. In particular, Earley's [3] algorithm for parsing with context-free grammars has been reconsidered and modified by several authors [7–9]. We give a common refinement of both Kilbury's [7, 8] modification and Graham, Harrison, and Ruzzo's [5] improvement of Earley's parser, which can be combined with a compact representation of partial parse trees in the parse tables, similar to Tomita's [11] "packed forest."

Kilbury [7] observed that Earley's algorithm suffers from an inefficiency hardly tolerable in natural language parsing. Working through the input from left to right, Earley's algorithm considers many useless alternatives when "predicting" how the parse might continue. For a large grammar, this may slow down the

Author's address: Hans Leiss, Zentralbereich Forschung und Entwicklung, ZFE IS INF 21, Siemens AG, Otto-Hahn-Ring 6, D-8000 München 83, FRG.

parsing process considerably. As an example, note that any particular sentence realizes only one of many alternative ways to open a sentence in English.

Although his observation is correct, Kilbury's modified algorithm [8] has its drawbacks, too. A comparison of both algorithms [12] shows that the new parser is better than Earley's only for a restricted class of context-free grammars.

We point out here that a proper statement of Kilbury's observation improves Earley's algorithm in the general case, too. A similar but weaker correction has independently been given by Doerre and Momma [2].

The parser thus obtained is essentially an Earley-parser with lookahead 1. The point is that only a small and natural modification of the Graham–Harrison–Ruzzo version of Earley's parser is needed here, with some special treatment of deletions and chain derivations.

It is well known how to equip Earley's parser with lookahead >1, but the additional power has to be paid for by storing right contexts and calculating deletions and chain derivations during the parse [1]. We show how to avoid this when lookahead >1 is restricted to the critical phase of "predicting."

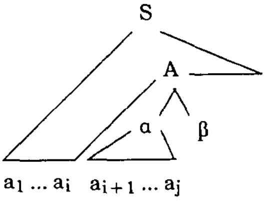
The paper is organized as follows: In Section 2 we give a sketch of Earley's parser. Sections 3 and 4 present modifications of this parser proposed by Kilbury [7, 8] and Graham, Harrison, and Ruzzo [4, 5]. We state both the parser and its modifications in terms of tree-constructing operations; for a given input, the set of trees generated by the parser is the smallest set of derivation trees closed under these operations. In Section 5 we introduce our modification of the parser as a refined family of such "closure operations." The set of trees generated by these is characterized, showing, in particular, that we almost got the power of lookahead 1. Section 6 contains examples and an iterative on-line algorithm implementing our modification. In Section 7 we compare our method with Harrison's modification and discuss the explicit use of parse trees, using a representation with structure sharing between partial parse trees. Some further improvements concerning lookahead >1, error handling, dealing with unknown input words, and unification of features attached to grammar symbols are given in Section 8. The Appendix contains the proof of the characterization theorem of Section 5.

2. A SKETCH OF EARLEY'S PARSER

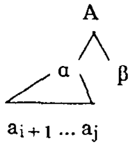
Let $G = (V, \Sigma, P, S)$ be a context-free grammar, and $w = a_1 \cdots a_n$, a word over Σ, fixed throughout this paper. (There is no need to distinguish between terminal symbols Σ and nonterminal symbols V, as far as parsing is concerned. Hence, assume that $\Sigma = V$ throughout; in practice, this has the advantage of allowing parsing and testing with a sentential form input.) We use $A, A_1, \ldots$ to denote elements of V, and $\alpha, \alpha_1, \ldots$ to denote elements of V^*, that is, words over V, where ε is the empty word. We call $(A \longrightarrow \alpha \cdot \beta)$ a *dotted rule* when $(A \longrightarrow \alpha\beta) \in P$. Earley's parser is usually described as building a table that contains sets of dotted rules in its fields. We prefer to use sets of derivation trees; that is, instead of putting $(A \longrightarrow \alpha \cdot \beta)$ into field (i, j), we build a set $\text{tree}(i, j, A \longrightarrow \alpha \cdot \beta)$ of derivation trees.

The output of Earley's parser can be characterized as follows:

(1) For all $0 \le i \le j \le n$ and each rule $(A \to \alpha\beta) \in P$, $t \in \text{tree}(i, j, A \to \alpha \cdot \beta)$ if and only if (iff) there is a derivation tree of form



with $t =$



that is, iff for some $\gamma \in V^*$, $S \Rightarrow^* a_1 \cdots a_i A \gamma$, and t is a derivation tree for $A \Rightarrow \alpha\beta \Rightarrow^* a_{i+1} \cdots a_j \beta$.

Then, clearly,

$$parse(a_1 \cdots a_n, P, S) = \cup \{\text{tree } (0, n, S \to \alpha \cdot) \mid (S \to \alpha) \in P\}$$

is the set of all parse trees for $a_1 \cdots a_n$ with root S.

Of course, when G has ε-rules or admits chain derivations such as $C \Rightarrow^* B \Rightarrow^* C$, $\text{tree}(i, j, A \to \alpha \cdot \beta)$ may be infinite. In order to get a terminating parser, we have to make these sets finite by using a suitable, normal form for trees, which is done in Section 5. Note that the table of dotted rules familiar from [1] or [6] stores $(A \to \alpha \cdot \beta)$ in field (i, j) iff $\text{tree}(i, j, A \to \alpha \cdot \beta) \neq \emptyset$. In particular, this table is always finite since P is.

Now, Earley's algorithm is based on the observation that the family of these sets, $\text{tree}(i, j, A \to \alpha \cdot \beta)$, as defined by (1), is the smallest family satisfying the following closure properties:

(2) Inductive definition of the set of parse trees for $a_1 \cdots a_n$:

(a) Initialize

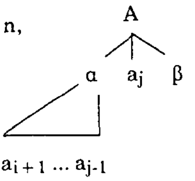
$$(S \to \alpha) \in P \Rightarrow \begin{array}{c} S \\ | \\ \alpha \end{array} \in \text{tree}(0, 0, S \to \cdot\ \alpha).$$

(b) Predict

$$\begin{array}{l} 0 \le i \le j < n, \\ (A \to \alpha) \in P, \\ \text{tree}(i, j, C \to \beta \cdot A\gamma) \neq \emptyset, \end{array} \quad \Rightarrow \begin{array}{c} A \\ | \\ \alpha \end{array} \in \text{tree}(j, j, A \to \cdot\, \alpha).$$

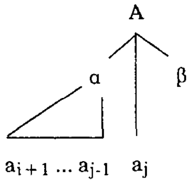
(c) Scan

$0 \le i \le j \le n,$



$\in \mathrm{tree}(i, j\text{-}1, A \rightarrow \alpha \cdot a_j \beta)$

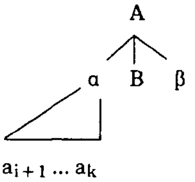
⇒



$\in \mathrm{tree}(i, j, A \rightarrow \alpha\, a_j \cdot \beta)$.

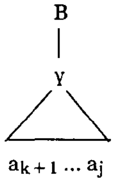
(d) Complete

$0 \le i \le k \le j \le n,$



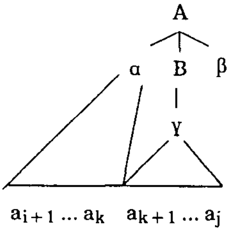
$\in \mathrm{tree}(i, k, A \rightarrow \alpha \cdot B \beta)$

, and



$\in \mathrm{tree}(k, j, B \rightarrow \gamma \cdot)$

⇒



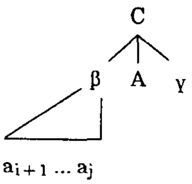
$\in \mathrm{tree}(i, j, A \rightarrow \alpha B \cdot \beta)$

Clearly, the closure operations Predict, Scan, and Complete can be used to build up the sets tree$(i, j, A \rightarrow \alpha \cdot \beta)$ from below, starting with the sets generated by Initialize. In the case of trees, this Predict–Scan–Complete loop may produce new trees at each finite stage. If we cut off the trees at the top, at some finite stage no new tops, that is, dotted rules, will appear; this is how Earley's algorithm avoids an infinite loop.

The reader who wants a more detailed statement of the algorithm may consult Aho and Ullman [1] or Harrison [6].

3. KILBURY'S MODIFICATION

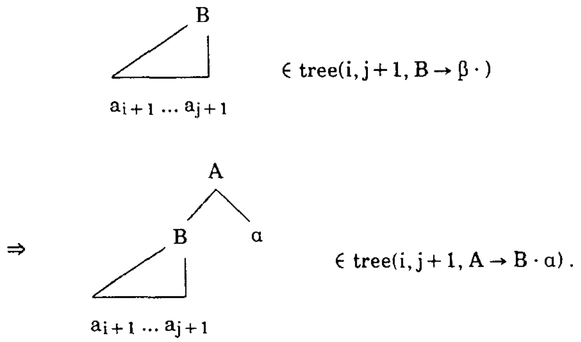
Suppose in parsing $a_1 \cdots a_n$ we have constructed the tree



and are now "expecting" a string of category A to come next. Earley's predictor (2b) works top–down and attaches α to node A whenever $(A \rightarrow \alpha) \in P$. Since (in unfortunate cases) most of these extensions will not be consistent with $a_{j+1} \cdots a_n$, Kilbury [6, 7] claims that we could save a lot of work by using essentially the following predictor:

Predict-K

(3) (i) $(A \longrightarrow a\alpha) \in P$, $a = a_{j+1} \Longrightarrow .a_{j+1} \in \text{tree}\ (j, j+1, A \longrightarrow a \cdot \alpha)$.
(ii) $(A \longrightarrow B\alpha) \in P$,



The idea is that before predicting which rules might be useful in analyzing $a_{j+1} \cdots a_n$ we should scan the next input symbol and exploit the information about a_{j+1} in Earley's predictor. But, as can be seen from (3ii), the predictor suggested by Kilbury works bottom–up. Hence, in general, it will construct trees that do not fit into the "expected" category (see [2] or [12] for examples). By adopting Predict-K, to put it formally, we give up the "only if" part of property (1). For example, in (3ii) we may put t into tree$(i, j+1, A \longrightarrow B \cdot \alpha)$ even if there is no derivation of form $S \Longrightarrow^* a_1 \cdots a_i A\eta$. Thus, we tend to build trees whose roots are not reachable from S. (We found that this effect of changing the order of scanning and predicting in Earley's algorithm had been known to Pulman [9].)

4. PRECOMPUTING DELETIONS AND CHAIN DERIVATIONS

In order to combine the advantages of both Earley's and Kilbury's predictor, we follow Harrison [6] in precomputing the following sets, which do not depend on the input string:

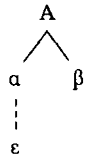
(4) (a) DEL := $\{A \mid A \in V, A \Longrightarrow^* \varepsilon\}$.
(b) SUB := $\{(A, B) \mid A, B \in V, A \Longrightarrow^* B\}$.
(c) FIRST := $\{(A, B) \mid A, B \in V$, and for some $\eta \in V^*, A \Longrightarrow^* B\eta\}$.

Sets (a) and (b) have to be calculated to determine (c), but they will also be used to prevent the parser from building chain derivations and deletions explicitly. Set (c) will be used mainly to make Earley's and Kilbury's predictors transitively closed, in order to link them in a way that will be explained below.

We note that Harrison [6] uses (4a, c) in a predictor, which can be stated in our terminology as follows:

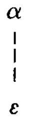
Predict-H

$$\begin{array}{l} 0 \le i < j < n, \\ tree(i, j, B \to \gamma \cdot C\,\delta) \neq \emptyset, \\ C \Rightarrow^{*} A\,\eta, \\ (A \to \alpha\,\beta) \in P, \\ \alpha \Rightarrow^{*} \varepsilon \end{array} \quad \Rightarrow$$



$\in tree(j, j, A \to \alpha \cdot \beta)$.

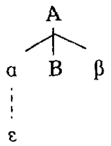
Here we let



denote some derivation tree or sequence of trees verifying $\alpha \Rightarrow^{*} \varepsilon$. Existence of such trees can be tested using (4a). Note that (5) combines several applications of (2b) and (2d).

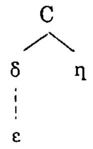
Predict-H works top–down and amounts to a transitive version of Earley's predictor in the sense that, without recursion, it leads to

(5)



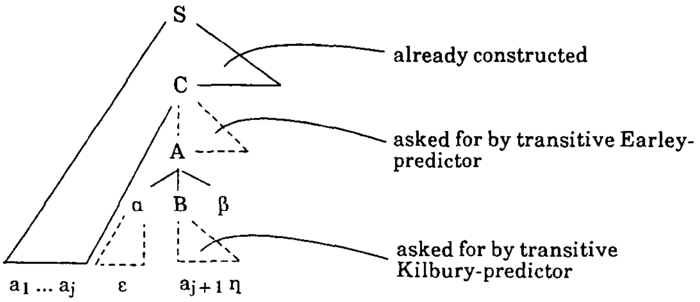
$\in tree(j, j, A \to \alpha \cdot B\, b)$,

$$\begin{array}{l} B \Rightarrow^{*} C\,\gamma, \\ (C \to \delta\,\eta) \in P, \\ \delta \Rightarrow^{*} \varepsilon \end{array} \quad \Rightarrow$$



$\in tree(j, j, C \to \delta \cdot \eta)$.

In a similar way we can use (4c) to make Kilbury's predictor transitive. The essential point is that we can then link what is constructed top–down by Earley with what is built bottom–up by Kilbury, thereby filtering out useless rules in an early step of the parsing process. That is, our predictor below will consider rule $(A \to \alpha B \beta)$ to be useful in parsing $a_{j+1} \cdots a_n$ whenever there is a derivation tree as shown:


S
C
A
α
B
β
already constructed
asked for by transitive Earley-predictor
asked for by transitive Kilbury-predictor
a1 ... aj
ε
aj+1 η

Note that by using (4) we can check whether the dotted trees exist. At the time of predicting the rule $(A \longrightarrow \alpha B \beta)$ to be used, these trees are not yet constructed, and η needs not be consistent with $a_{j+2} \cdots a_n$.

5. USING REFINED CLOSURE CONDITIONS TO IMPROVE EARLEY'S PARSER

The predictor just sketched is part of our modification of Earley's algorithm, which we now state in terms of closure conditions. For convenience, we define sets Expect(j) of categories expected to fit to prefixes of $a_j \cdots a_n$ and treat Scan as a special case of Complete, using "formal" productions $(a_j \longrightarrow a_j)$.

As we mentioned earlier, deletions or chain derivations may cause the set of parse trees for $a_1 \cdots a_n$ to be infinite. Hence, in a finite amount of time the best we can do is to produce all parse trees up to a suitable equivalence. To this end, we want to identify all deletions of category A and all chain derivations from category B to category C. Therefore, let

$$\begin{array}{c}A\\ \|\\ B\end{array} \qquad \text{and} \qquad \begin{array}{c}\alpha\\ \|\\ \varepsilon\end{array}$$

denote some derivation tree or sequence of trees (say, of minimal size) verifying $A \Longrightarrow^* B$ and $\alpha \Longrightarrow^* \varepsilon$, respectively, which we choose in advance and call *canonical.*

Our closure operations for constructing parse trees only use canonical deletions and chain derivations and, thus, neglect differences in various ways to delete α or to turn from A to its subcategory B. But, whenever there is need for a finer analysis of deletions or chain derivations (as may be the case in natural language parsing), we can as well choose any finite set of derivation trees for $A \Longrightarrow^* B$ and $A \Longrightarrow^* \varepsilon$ to provide us with canonical deletions and chain derivations.

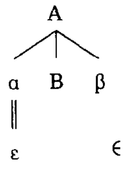
(6) Inductive definition of the modified set of parse trees for $a_1 \cdots a_n$:

(a) Initialize: Expect(1) = $\{S\}$.

(b) Scan$'$: $0 < j \leq n \Longrightarrow a_j \in \text{tree}(j-1, j, a_j \longrightarrow a_j \cdot)$.

(c) Predict$'$:

$$\begin{array}{l} 0 \leq j < n, \\ C \in \text{Expect}(j+1), \\ C \Rightarrow^* A\,\eta, \\ (A \rightarrow \alpha\, B\, \beta) \in P, \\ \alpha \Rightarrow^* \varepsilon, \\ B \Rightarrow^* a_{j+1}\, \theta, \\ \beta \neq \varepsilon \end{array} \quad \Rightarrow$$

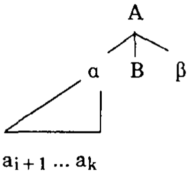


$\in \text{tree}(j, j, A \rightarrow \alpha \cdot B\, \beta)$.

(d) Complete′:

(1)

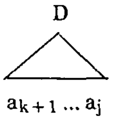
$0 \leq i < k < j \leq n,$



$\in \text{tree}(i, k, A \to \alpha \cdot B\,\beta)$,

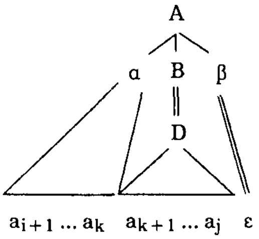
$B \Rightarrow^* D,$

$\beta \Rightarrow^* \varepsilon,$



$\in \text{tree}(k, j, D \to \delta \cdot)$

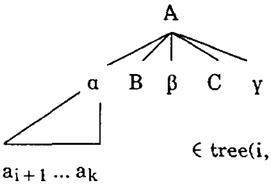
$\Rightarrow$



$\in \text{tree}(i, j, A \to \alpha\, B\, \beta \cdot)$.

(2)

$$0 \leq i \leq k < j < n,$$

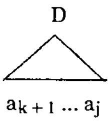


$\in \text{tree}(i, k, A \to \alpha \cdot B\,\beta\,C\,\gamma)$,

$B \Rightarrow^* D,$

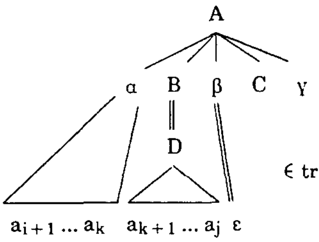
$\beta \Rightarrow^* \varepsilon,$

$C \Rightarrow^* a_{j+1}\,\eta,$



$\in \text{tree}(k, j, D \to \delta \cdot)$,

$\Rightarrow$



$\in \text{tree}(i, j, A \to \alpha\, B\, \beta \cdot C\, \gamma)$.

(e) Expect: $0 \leq i < j < n$, $\text{tree}(i, j, A \to \alpha \cdot B\beta) \neq \emptyset \Rightarrow B \in \text{Expect}(j + 1)$.

The idea behind Predict′ has been explained in Section 4: The precomputed first-relation (4c) is used twice rather than once, as in Harrison's predictor. However, using FIRST to decide whether $B \Rightarrow^* a_{k+1}\eta$ in building tree$(k, k, A \rightarrow \alpha \cdot B\beta)$ does *not* imply that we look ahead one symbol. To see this, note that the only way $t \in$ tree$(k, k, A \rightarrow \alpha \cdot B\beta)$ is used is when Complete′ (in part 2 with $i = k$) combines t with some $s \in$ tree$(k, j, D \rightarrow \delta \cdot)$ for some $k < j$, that is, when $a_{k+1} \cdots a_j$ has been scanned. Thus, we can delay predicting rules (to be used in parsing prefixes of $a_{k+1} \cdots a_n$) until we have done Scan′ for a_{k+1}.

The precomputed chain derivations and deletions are used in Complete′, too. But here, using (4c) to check whether $C \Rightarrow^* a_{j+1}\eta$ in Complete′ (2) cannot be delayed; hence, we have to look ahead one symbol to ensure consistency of the stored partial parse trees with the next input symbol. (See Section 6 for a remark on how to remove this looking ahead without losing too much of its effect.) However, we do not have the full advantage of lookahead 1, since in part (1) of Complete′ we did not check whether the tree constructed can be combined with a_{j+1} later on.

Remark 5.1. It seems a bit complicated to check the conditions in Predict′. Therefore, we point out that the parse-time complexity of Predict′ can be reduced further as follows: For each $C \in V$ and $B \in \Sigma \cup V$, we can compute in advance

$$\begin{aligned}\text{Pred}(C, B) := \{(t, A \rightarrow \alpha \cdot B\beta) \mid\ & C \Rightarrow^* A\eta,\ (A \rightarrow \alpha B\beta) \in P,\ \alpha \Rightarrow^* \varepsilon,\ \beta \neq \epsilon, \\ & \text{and } t \text{ the tree combining an application of } (A \rightarrow \alpha B\beta) \text{ with} \\ & \text{the canonical derivation of } \alpha \Rightarrow^* \varepsilon\}.\end{aligned}$$

At parse time, having scanned a_{j+1}, predicting can be done by selecting from Pred(C, B), that is, by defining, for all $C \in$ Expect$(j + 1)$ and $B \Rightarrow^* a_{j+1}\eta$ (even $B \in$ Expect$(j + 1)$ suffices, with the lookahead built into Complete′),

$$\text{tree}(j, j, A \rightarrow \alpha \cdot B\beta) := \{t\} \qquad \text{iff} \quad (t, A \rightarrow \alpha \cdot B\beta) \in \text{Pred}(C, B).$$

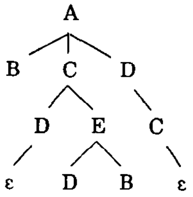
This means that we do some searching through the precomputed relations and set of productions at parser construction time rather than at parse time.

Finally, we have to characterize the smallest set of derivation trees closed under the operations in (6). Since these obviously are more restrictive than the original ones in (2), it is not clear that we still get "all" parse trees for $a_1 \cdots a_n$. To be more specific, we have to introduce some terminology. The following notion of normal form derivation trees depends on our choice of canonical deletions and chain derivations.

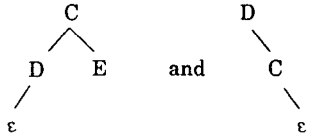
Definition 5.2. Let s and t be derivation trees. We call s a *cone of* t if s is not a singleton and there are subtrees t_1 of t and t_2 of t_1 such that s is obtained from t_1 by removing all but the root of t_2. A tree is called *thin* if at most one of its leaves has label $\neq \varepsilon$. We say a tree is *in normal form* if all of its maximal thin cones are canonical. The *normal form of* t is obtained from t by replacing its maximal thin cones by their canonical counterparts.

Note that different maximal thin cones of t are disjoint; hence, the normal form of t is well defined. (If we work with several canonical trees for $A \Rightarrow^* B$, etc., a tree can have several normal forms.) Intuitively, deletions and subclassifications in a normal form tree are canonical, that is, the most "simple" among all possible ones.

Example 5.3. The tree

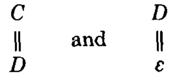


is in normal form iff its maximal thin cones, that is,

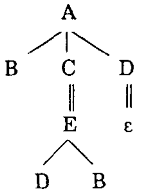


are canonical.

If



denote the canonical trees verifying $C \Rightarrow^* E$ and $D \Rightarrow^* \varepsilon$, then



is the normal form of the above tree.

We are now in a position to state precisely what is generated by the operations in (6). As one can see from statement (i) of the following characterization theorem, most of the entries are consistent with the next input symbol. In order not to store contexts (beyond right-hand sides of the rules), we had to allow in statement (ii) some entries that may not be consistent with the next input symbol.

CHARACTERIZATION THEOREM 5.4. *For any context-free grammar* (V, Σ, P, S) *and word* $a_1 \cdots a_n$ *over* V, *let*

$$\{tree(i, j, A \rightarrow \alpha \cdot \beta) \mid 0 \leq i \leq j \leq n, (A \rightarrow \alpha\beta) \in P\}$$

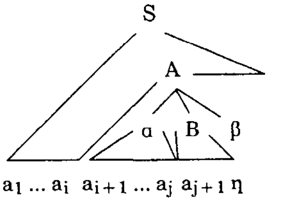
be the smallest family of sets of derivation trees satisfying closure conditions (6) *above. Then for any derivation tree t, we have*

(*i*) *$t \in tree(i, j, A \rightarrow \alpha \cdot B\beta)$ iff $i < j$ or $\beta \neq \varepsilon$, and there are derivations $S \Rightarrow^* a_1 \cdots a_i A\theta$, $\alpha \Rightarrow^* a_{i+1} \cdots a_j$, and $B \Rightarrow^* a_{j+1}\eta$, such that t is the normal form of the combined derivation $A \Rightarrow \alpha B\beta \Rightarrow^* a_{i+1} \cdots a_j B\beta$.*

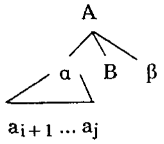
(*ii*) *$t \in tree(i, j, D \rightarrow \delta \cdot)$ iff either $i = j - 1$, $(D \rightarrow \delta) = (a_j \rightarrow a_j)$, and t is the singleton tree $.a_j$, or there are k, α, and β with $i < k < j$, $\delta = \alpha\beta$, and derivations $S \Rightarrow^* a_1 \cdots a_i D\theta$, $\alpha \Rightarrow^* a_{i+1} \cdots a_k$, and $\beta \Rightarrow^* a_{k+1} \cdots a_j$ such that t is the normal form tree of the combined derivation $D \Rightarrow \alpha\beta \Rightarrow^* a_{i+1} \cdots a_k\beta \Rightarrow^* a_{i+1} \cdots a_k\, a_{k+1} \cdots a_j$.*

PROOF. The proof of the Characterization theorem is given in the Appendix. The nontrivial cases look as follows:

Case (i) ($i < j$ or $\beta \neq \varepsilon$). There is a derivation tree s such that t is the normal form tree of t_0:

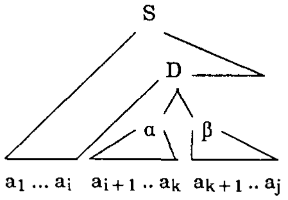


tree s

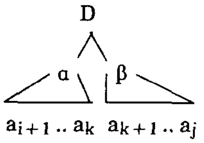


tree t_0

Case (ii) ($i < k < j$ and $(D \rightarrow \delta) = (D \rightarrow \alpha\beta)$). There is a derivation tree s such that t is the normal form tree of t_0:



tree s

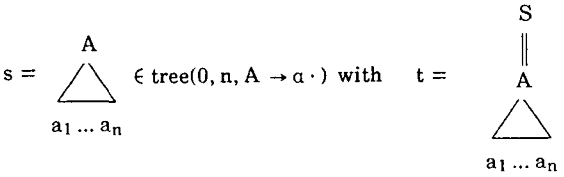


tree t_0

COROLLARY 5.5. *For $a_1 \cdots a_n \neq \varepsilon$, the following conditions are equivalent:*

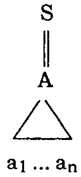
(*i*) *t is the normal form of some derivation tree verifying $S \Rightarrow^* a_1 \cdots a_n$.*

(*ii*) *Either $n = 1$ and t is the normal form tree for (the chain derivation) $S \Rightarrow^* a_1$, or $n > 1$ and there is $(A \rightarrow \alpha) \in P$ and a tree*



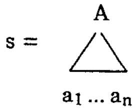
PROOF. (ii) ⟹ (i): In case $n > 1$, note that by (ii) of the theorem the tree s is in normal form and splits at its root into at least two subtrees with yield $\neq \varepsilon$. Putting on top of it a canonical tree for $S \Rightarrow^* A$ gives a tree t in normal form.

(i) ⟹ (ii): We know that t is of form

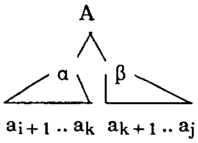


,

where either



is the singleton tree $.a_1$, or is of form



for some rule $(A \rightarrow \alpha\beta)$ and some k with $1 < k < n$. To see this, recall that a maximal chain derivation beginning with S must be canonical. But now from (ii) of the theorem, we get $s \in \text{tree}(0, n, A \rightarrow \alpha\beta\,\cdot\,)$. □

Definition 5.6

$$\text{tree}(i, j) := \cup\ \{\text{tree}(i, j, A \rightarrow \alpha \cdot \beta) \mid (A \rightarrow \alpha\beta) \in P\}.$$

By the corollary, all parse trees (in normal form) can be obtained from the table $\{\text{tree}(i, j) \mid 0 \leq i \leq j \leq n\}$. Hence, we can talk of *the parser defined by closure conditions* (6).

We remark that the use of a normal form for derivation trees is not only necessary to keep tree(i, j) finite, but it also keeps each tree entered into tree(i, j) small (for a good choice of canonical trees).

Comparing the characterization of Earley's parser given in (1) with the above characterization of our modification, we find that the sets tree(i, j) constructed by the new parser in general are smaller than those constructed by Earley's. The amount of savings depends partially on whether the grammar admits deletions and chain derivations, but is mainly due to the lookahead implicit in Predict′ and Complete′.

6. AN ITERATIVE ON-LINE ALGORITHM FOR THE MODIFIED PARSER

We now give a program that implements the parser defined by (6) as an on-line algorithm. In fact, it becomes an on-line algorithm by putting an end marker at the end of the input, thereby avoiding the explicit use of n below. The overall structure is similar to that of Harrison [6] and will be justified afterward.

A bit of notation is needed. We assume that the production rules are enumerated, and we identify a derivation tree with the sequence of rule numbers of the corresponding leftmost derivation. For sequences $t = (n_1, \ldots, n_k)$ and $s = (m_1, \ldots, m_l)$, we define $s\hat{}t := (n_1, \ldots, n_k, m_1, \ldots, m_l)$. Since we allow nonterminal symbols at leaves of our trees, we have to indicate the end of a path; so we admit the corresponding leaf's label as a "special" number.

We use $t\colon A \Rightarrow^* B$ to say that t is the canonical tree verifying $A \Rightarrow^* B$, assuming t to be the empty list if $B = A$. Similarly, $t\colon A \Rightarrow^* \varepsilon$ and $t\colon \alpha \Rightarrow^* \varepsilon$ are meant to show trees of canonical deletions.

(7) *The modified algorithm of Earley e.a.*

Input: a string $a_1 \cdots a_n$, a finite set P of production rules, and a start symbol S.

Output: the family of all sets tree$(i, j, A \rightarrow \alpha \cdot \beta)$ as characterized above.

Program:

```
  begin
    expect (1) := $\{S\}$;
    for $j = 1$ to $n$
                                                        /* scan $a_j$   scan ($j$) */
    tree ($j - 1, j, a_j \rightarrow a_j \cdot$ ) := $\{(a_j)\}$;
     /* predict tops of trees for prefixes of $a_j \cdots a_n$   predict ($j - 1$) */
    for all $C \in$ expect ($j$)
      and $m\colon (A \rightarrow \alpha B\beta) \in P$
        with $C \Rightarrow^* A\theta$ for some $\theta \in V^*$,
          $l\colon \alpha \Rightarrow^* \varepsilon$,
          $\beta \neq \varepsilon$,
        and $B \Rightarrow^* a_j\eta$ for some $\eta \in V^*$
      put tree ($j - 1, j - 1, A \rightarrow \alpha \cdot B\beta$) := $\{(m)\hat{}l\}$;
    for $i = j - 1$ down to 0
                                              /* complete all sets tree ($i, j, \ldots$ ) */
      for $k = j - 1$ down to $i$                           /* complete ($i, k, j$) */
        for all $(A \rightarrow \alpha B\beta\gamma) \in P$,
             $(D \rightarrow \delta) \in P \cup \{a_{j-1} \rightarrow a_{j-1}\}$,
             $t \in$ tree ($i, k, A \rightarrow \alpha \cdot B\beta\gamma$),
           and $s \in$ tree ($k, j, D \rightarrow \delta \cdot$ )
             with $m\colon B \Rightarrow^* D$,
                  $l\colon \beta \Rightarrow^* \varepsilon$,
             and either $\gamma = \varepsilon$ and $i < k$
                  or for some $C \in V$ and $\theta, \eta \in V^*$
                    $\gamma = C\theta$ and $C \Rightarrow^* a_{j+1}\eta$
           put $t\hat{}m\hat{}s\hat{}l$ into tree ($i, j, A \rightarrow \alpha B\beta \cdot \gamma$);
           /* collect all expected categories for prefixes of $a_{j+1} \cdots a_n$ */
    if $j < n$ then put expect ($j + 1$) :=
      $\{$B | tree ($i, j, A \rightarrow \alpha \cdot B\beta) \neq \emptyset$ for some $i < j$ and $(A \rightarrow \alpha B\beta) \in P\}$
  end
```

Let us indicate briefly why this program is a faithful implementation of the parser defined by (6).

PROPOSITION 6.1. *For $0 \le i \le j \le n$, let tree*(i, j) be the set of trees contributing to $a_{i+1} \cdots a_j$ generated by the program above, while tree(i, j) is the corresponding set of trees defined by the parser* (6). *Then tree*(i, j) = tree(i, j) for $0 \le i \le j \le n$.*

PROOF (Sketch). Clearly, tree*$(i, j) \subseteq$ tree(i, j), since entering a tree into tree*(i, j) is only allowed according to the closure conditions (6). To see tree$(i, j) \subseteq$ tree*(i, j), we have to show that the loop structure in the program is strong enough to ensure closure of tree*(i, j) under the operations given in (6). Since these operations preserve set inclusion, we may proceed by induction on j.

It is easy to derive from (6) the following properties:

(8) (a) tree(j, j) can be computed from $\{$tree$(i, j) \mid 0 \le i < j\}$ (by (6c, e)).

(b) For $i < j$, tree(i, j) can be computed from $\{$tree$(i, i)\} \cup \{$tree(i, k), tree$(k, j) \mid i < k < j\}$ (by (6b, d)), and we can ensure each application of ($6d_1$) to precede all applications of ($6d_2$).

Using (8a) gives tree$(j-1, j-1) \subseteq$ tree*$(j-1, j-1)$ once we have tree$(i, j-1) \subseteq$ tree*$(i, j-1)$ for all $i < j-1$ at the end of the outer loop with value $j-1$. To get tree$(i, j) \subseteq$ tree*(i, j) for all $i < j$ after executing the outer loop with value j, note that by (8b) we only need tree$(k, j) \subseteq$ tree*(k, j) for all $i < k < j$, before applying Complete′. This is ensured by using Complete′ for $k = j-1$ down to i and only afterward looking for tree$(i-1, j)$. □

Note that we may define Expect$(j+1)$ by inserting at the end of the completer "**and if** $\gamma = C\theta$, **then put** C into Expect$(j+1)$," rather than by using a separate loop. Also, if we do not want the completer to look ahead explicitly in order to be able to check $C \Rightarrow^* a_{j+1}\eta$, we can transfer this test into the predictor step, where we can disregard those $C \in$ Expect$(j+1)$ that do not satisfy $C \Rightarrow^* a_{j+1}\eta$ for some η. This keeps the sets built up by the predictor without change, but the completer will make some useless entries. Also, it seems more complicated to give a precise characterization of the trees generated by this version.

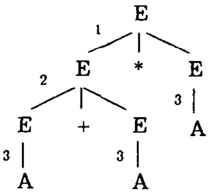
We now demonstrate our algorithm by three examples. Entries in the parse tables are ordered according to the above iterative version of the algorithm. Parse trees are coded by sequences of rule numbers and symbols at their leaves. SUB- and FIRST- are the precomputed relations SUB and FIRST without the trivial tuples (X, X).

Example 6.2. This is the standard textbook example of arithmetic terms:

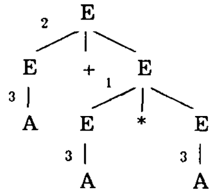
Productions	Input	Start symbol
1 $E \to E * E$	$A + A * A$	E
2 $E \to E + E$		
3 $E \to A$		
SUB-	DEL	FIRST-
3 $E \to A$	None	(EA)

Parse trees (with an indication how the coding is related to the tree)

1. (1 2 3 A + 3 A * 3 A)



2. (2 3 A + 1 3 A * 3 A)



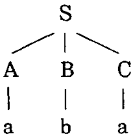
Parse table:

(⟨i j⟩	⟨dotted rule⟩	⟨coded tree⟩	⟨program step⟩)
0 1	$A \longrightarrow A \cdot$	(A)	scan (1)
0 0	$E \longrightarrow \cdot E * E$	(1)	predict (0)
0 0	$E \longrightarrow \cdot E + E$	(2)	predict (0)
0 1	$E \longrightarrow E \cdot + E$	$(23A)$	complete (0, 0, 1)
1 2	$+ \longrightarrow + \cdot$	(+)	scan (2)
0 2	$E \longrightarrow E + \cdot E$	$(23A\ +)$	complete (0, 1, 2)
2 3	$A \longrightarrow A \cdot$	(A)	scan (3)
2 2	$E \longrightarrow \cdot E * E$	(1)	predict (2)
2 2	$E \longrightarrow \cdot E + E$	(2)	predict (2)
2 3	$E \longrightarrow E \cdot * E$	$(13A)$	complete (2, 2, 3)
0 3	$E \longrightarrow E + E \cdot$	$(23A + 3A)$	complete (0, 2, 3)
0 3	$E \longrightarrow E \cdot * E$	$(123A + 3A)$	complete (0, 0, 3)
3 4	$* \longrightarrow * \cdot$	$(*)$	scan (4)
2 4	$E \longrightarrow E * \cdot E$	$(13A\ *)$	complete (2, 3, 4)
0 4	$E \longrightarrow E * \cdot E$	$(123A + 3A\ *)$	complete (0, 3, 4)
4 5	$A \longrightarrow A \cdot$	(A)	scan (5)
4 4	$E \longrightarrow \cdot E * E$	(1)	predict (4)
4 4	$E \longrightarrow \cdot E + E$	(2)	predict (4)
2 5	$E \longrightarrow E * E \cdot$	$(13A * 3A)$	complete (2, 4, 5)
0 5	$E \longrightarrow E * E \cdot$	$(123A + 3A * 3A)$	complete (0, 4, 5)
0 5	$E \longrightarrow E + E \cdot$	$(23A + 13A * 3A)$	complete (0, 2, 5)

Example 6.3. This example shows that our modification of Earley's parser gives smaller tables than the one proposed by Doerre and Momma [2], a different combination of Earley's with Kilbury's algorithm.

Productions	SUB-	FIRST-
1 $S \longrightarrow ABC$	5 $B \longrightarrow b$	(S, a)
2 $A \longrightarrow a$	4 $C \longrightarrow a$	(B, b)
3 $A \longrightarrow Aa$	2 $A \longrightarrow a$	(C, a)
4 $C \longrightarrow a$		(A, a)
5 $B \longrightarrow b$		(S, A)

Start symbol	Deletable categories	Input	Parse tree
S	None	aba	$(12a5b4a)$



Parse table

⟨⟨i j⟩	⟨dotted rule⟩	⟨coded tree⟩	⟨program step⟩⟩
0 1	$a \rightarrow a\cdot$	(a)	scan (1)
0 0	$S \rightarrow \cdot ABC$	(1)	predict (0)
0 0	$A \rightarrow \cdot Aa$	(3)	predict (0)
0 1	$S \rightarrow A \cdot BC$	$(12a)$	complete (0, 0, 1)
1 2	$b \rightarrow b\cdot$	(b)	scan (2)
0 2	$S \rightarrow AB \cdot C$	$(12a5b)$	complete (0, 1, 2)
2 3	$a \rightarrow a\cdot$	(a)	scan (3)
0 3	$S \rightarrow ABC\cdot$	$(12a5b4a)$	complete (0, 2, 3)

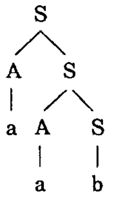
The parse table of [2] does not store the scanning, but contains the following useless dotted rules, due to a less restrictive completer:

01	$A \rightarrow a\cdot$	12	$B \rightarrow b\cdot$
01	$A \rightarrow A \cdot a$	23	$C \rightarrow a\cdot$

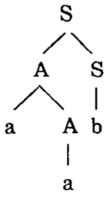
Example 6.4. This example is given for comparison with Graham, Harrison, and Ruzzo's improved version of Earley's algorithm (see [5, p. 428]). We introduce new rule numbers R_i to denote canonical derivations for subcategorizations and deletions, if necessary.

Productions	SUB-	FIRST-
1 $S \rightarrow AS$	R_1 $A \rightarrow b$	(S, a)
2 $S \rightarrow b$	R_2 $A \rightarrow a$	(A, b)
3 $A \rightarrow aA$	2 $S \rightarrow b$	(A, a)
4 $A \rightarrow bA$		(S, b)
5 $A \rightarrow \varepsilon$		(S, A)

Start symbol	Deletable categories	Input	Parse tree
S	A	aab	



(1 R2 a 1 R2 a 2 b)



(1 3 a R2 a 2 b)

Parse table

⟨⟨i j⟩	⟨dotted rule⟩	⟨coded trees⟩	⟨program step⟩⟩
0 1	$a \rightarrow a\cdot$	(a)	scan (1)
0 0	$S \rightarrow \cdot AS$	(1)	predict (0)
0 0	$A \rightarrow \cdot aA$	(3)	predict (0)
0 1	$S \rightarrow A \cdot S$	$(1R_2a)$	complete (0, 0, 1)
0 1	$A \rightarrow a \cdot A$	$(3a)$	complete (0, 0, 1)
1 2	$a \rightarrow a\cdot$	(a)	scan (2)
1 1	$S \rightarrow \cdot AS$	(1)	predict (1)
1 1	$A \rightarrow \cdot aA$	(3)	predict (1)
1 2	$S \rightarrow A \cdot S$	$(1R_2a)$	complete (1, 1, 2)
1 2	$A \rightarrow a \cdot A$	$(3a)$	complete (1, 1, 2)
0 2	$A \rightarrow aA\cdot$	$(3aR_2a)$	complete (0, 1, 2)

0 2	$S \rightarrow A \cdot S$	$(1 3 a R_2 a)$	complete (0, 0, 2)
2 3	$b \rightarrow b \cdot$	(b)	scan (3)
2 2	$S \rightarrow \cdot A S$	(1)	predict (2)
2 2	$A \rightarrow \cdot b A$	(4)	predict (2)
1 3	$S \rightarrow A S \cdot$	$(1 R_2 a 2 b)$	complete (1, 2, 3)
1 3	$A \rightarrow a A \cdot$	$(3 a R_1 b)$	complete (1, 2, 3)
0 3	$S \rightarrow A S \cdot$	$(1 3 a R_2 a 2 b)$, $(1 R_2 a 1 R_2 a 2 b)$	complete (0, 2, 3), complete (0, 1, 3)
0 3	$A \rightarrow a A \cdot$	$(3 a 3 a R_1 b)$	complete (0, 1, 3)

The parse table in [5] has 39 entries (and, again, the scanning is not stored in their table, as it is above).

7. TIME AND SPACE COMPLEXITY

In this section we consider the worst-case time and space complexities of our algorithm on a random access machine. Recall that the difference between our and other versions of Earley's algorithm comes from two facts:

(i) The use of precomputations is extended to get smaller parse tables, and
(ii) partial parse trees in a suitable, normal form are constructed and stored during the parse, rather than just storing dotted rules.

Since previous versions do not build up parse trees explicitly, we ignore the building of trees and consider our use of precomputations with respect to dotted rules only. We point out that we do not change the overall structure of the algorithm, and hence its well-known $O(n^3)$ upper bound for time complexity is, *in general*, not improved.

7.1 The Effect of Improved Predictor and Completer with Respect to Dotted Rules

Using precomputed sets and relations as parse-time tests to avoid unnecessary computation steps can only yield a gain in computation time when these tests are *informative*, that is, when they separate their possible inputs into two nonempty subclasses. This will be satisfied for programming or natural language grammars, but it sometimes is not for small example grammars. Note that this proviso applies to any method that uses relations such as Sub or First. Therefore, we have to compare the use of these relations in our and other methods.

The most extensive use of precomputations so far seems to be by Graham, Harrison, and Ruzzo [5], also described by Harrison [6]. From the parse table built up by the parser of Section 6, we derive an *item table* of dotted rules by

$$\text{Items}(i, j) := \{(A \rightarrow \alpha \cdot \beta) \mid \text{tree } (i, j, A \rightarrow \alpha \cdot \beta) \neq \emptyset\}.$$

Let ItemsH(i, j) be the corresponding set of dotted rules obtained by algorithm 12.6.4 of [6].

(a) Concerning our more restrictive predictor (cf., Section 4 for Predict′ and Predict-H), we find the following difference (for $i = j$):

$$\begin{aligned} \text{ItemsH}(j,j) &= \{(A \to \alpha \cdot \beta) \mid (A \to \alpha\beta) \in P, \\ &\qquad S \Rightarrow^* a_1 \cdots a_j A\eta \text{ for some } \eta \in V^*, \alpha \Rightarrow^* \varepsilon\}, \\ (*) \quad \text{Items}(j,j) &= \text{ItemsH}(j,j) - \\ &\qquad \{(A \to \alpha \cdot B\beta) \mid \beta \\ &\qquad = \varepsilon \text{ or } (a_{j+1}, B) \notin \text{First}\}. \end{aligned}$$

(Note that our use of normal forms does not reduce the set of expected categories.)

Thus, the average space saved in storing the diagonal elements of the item table is roughly $d \cdot \sum_{j=1,\ldots,n} |\text{ItemsH}(j,j) - \text{Items}(j,j)|$, where d is the average size of dotted rules. We conclude from $(*)$ that, in general, the smaller the grammar's first relation, the more we gain in space in comparison to Harrison's predictor. Clearly, the space saved in this way is bounded by $n \cdot m$ times the number of dotted rules, where m is the maximum size of a dotted rule.

Keeping field (j, j) of the item table small is important because each missing entry reduces the *time* to compute fields (j, k) for $j < k \leq n$: For each $(A \to \alpha \cdot B\beta) \in \text{ItemsH}(j, j)$ such that $(B\ a_{j+1}) \notin$ FIRST, we save

$$c_{\text{First}} \cdot \sum_{j<k\leq n} |\{(C \to \gamma \cdot) \mid (C \to \gamma \cdot) \in \text{ItemsH}(j, k)\}|$$

many steps, where c_{First} is the time needed for a lookup in the first relation. The reason is that a completed dotted rule $(C \to \gamma \cdot) \in \text{ItemsH}(j, k)$ spans $a_{j+1} \cdots a_k$, and hence trying to link B to C must fail, as $(B, a_{j+1}) \notin$ FIRST. We omit estimating the gain obtained from missing elements of the form $(A \to \alpha \cdot B\beta)$ with $\beta = \varepsilon$.

Note that our predictor will not improve Harrison's in case the grammar has only one terminal letter and no useless nonterminals, as here the first relation extends $\Sigma \times V$.

(b) Concerning our more restrictive completer, the built-in lookahead (cf., Complete′ (2) of Section 4) leads to a similar effect in space and time reductions. Namely, for $i < j < n$ we get

$$\text{Items}(i, j) = \text{ItemsH}(i, j) - \{(A \to \alpha B\beta \cdot C\gamma) \mid (a_{j+1}, C) \notin \text{FIRST}\}.$$

Again, each $(A \to \alpha B\beta \cdot C\gamma) \in \text{ItemsH}(i, j) - \text{Items}(i, j)$ reduces the time to compute fields $(i, k), j < k \leq n$, by

$$c_{\text{Sub}} \cdot \sum_{j<k\leq n} |\{(D \to \delta \cdot) \mid (D \to \delta \cdot) \in \text{ItemsH}(j, k)\}|,$$

where c_{Sub} is the time needed for a lookup in the precomputed relation SUB.

In our algorithm, there is a certain amount of additional time spent on lookups in the precomputed relations, which we did not yet take into account. If the predictor is precomputed according to Remark 5.1, the additional lookups are only those in the completer steps that check for compatibility with the next input symbol. The number of these lookups that are not made in Harrison's completer is at most n times the number of dotted rules of the form $(A \longrightarrow \alpha \cdot BC\gamma)$.

7.2 The Space Complexity of Using Partial Parse Trees in Normal Form

We first look at the size of a single normal form parse tree and then consider the size of the parse tables storing such trees, rather than just dotted rules.

In order to measure the space needed to store a tree t, we use the coding of parse trees introduced in Section 6. Thus, if $A \in V$ and k: $(A \longrightarrow B_1 \cdots B_n) \in P$,

$$\mathrm{code}(.A) := (A),$$

and if t is a tree with root labeled A and sons $t_1, \ldots, t_n$ with roots labeled $B_1, \ldots, B_n$, respectively, then

$$\mathrm{code}(t) := (k)\,\hat{}\,\mathrm{code}(t_1)\ \hat{}\ \cdots\ \hat{}\ \mathrm{code}(t_n).$$

Definition 7.1. Let t be a derivation tree and $w \in V^*$. Define the size of t and the degree of amgibuity of w with respect to usual and normal form trees by

$$\begin{aligned}
\mathrm{size}(t) &:= \mathrm{length}(\mathrm{code}(t)),\\
\deg(w) &:= |\{t \mid t \text{ is a derivation tree for } w\}|,\\
\deg_{\mathrm{nf}}(w) &:= |\{t \mid t \text{ is a derivation tree for } w,\ t \text{ is in normal form}\}|.
\end{aligned}$$

If P has ε- or chain rules, $\{\mathrm{size}(t) \mid t \text{ is a derivation tree for } w\}$ may be unbounded, and hence $\deg(w)$ may be infinite. Thus, we cannot construct all parse trees for w in finite time and space. This is overcome by using normal forms:

PROPOSITION 7.2. *Let t be a normal form derivation tree for $w \in V^*$.*

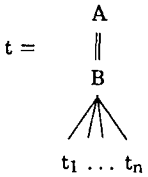
(1) *size(t) is linearly bounded length(w).*
(2) *$\deg_{nf}(w) \leq \deg(w)$, and $\deg_{nf}(w)$ is finite.*

PROOF

(1) Let R be the maximum number of symbols in the right-hand sides of rules in P. For each tree t, let $y(t)$ be the number of leaves of t not marked with ε. By induction on the structure of normal form trees, we show

$$(*)\ \mathrm{size}(t) \leq (R + 2) \cdot (y(t) - 1) + 2.$$

For $y(t) \leq 1$, $(*)$ is clear by the normal forms for deletions and chain derivations. So, let t be of form



Then $y(t) = \sum_{i=1,\ldots,n} y(t_i)$, and $y(t_i) < y(t)$ for each i.
Then with $c_0 := |\{i \mid y(t_i) = 0\}|$ and $c_1 := |\{i \mid y(t_i) \neq 0\}|$, we get

$$\begin{aligned}
\text{size}(t) &= 2 + \textstyle\sum_{i=1,\ldots,n} \text{size}(t_i) \\
&\leq 2 + \textstyle\sum_{i=1,\ldots,n} ((R+2)(y(t_i) - 1) + 2) \\
&\quad + 2 + c_0 + (R+2)(y(t) - c_1) + 2c_1 \\
&= 2 + (R+2)(y(t) - 1) + c_0 + 2c_1 - (R+2)(c_1 - 1) \\
&= 2 + (R+2)(y(t) - 1) + c_0 + 2 - R(c_1 - 1) \\
&\leq 2 + (R+2)(y(t) - 1),
\end{aligned}$$

using $c_0 \leq R - 2$ and $2 \leq c_1$ in the last step.

(2) This follows directly from (1). □

Now consider the size of our parse tables storing normal form trees. By the proposition, the size of each $t \in \text{tree}(i, j, A \longrightarrow \alpha \cdot \beta)$ is $O(j - i)$. Hence, if the grammar is unambiguous and reduced, we can store normal form trees in the parse table with little overhead, as compared to the dotted-rule versions. As we use ambiguity with respect to normal form trees, some grammars with a high, or even an infinite, degree of ambiguity in the usual sense are covered here, too.

However, we have to be careful in dealing with highly ambiguous grammars. The way trees are constructed by our algorithm was motivated by the closure conditions (6) on the set of normal form parse trees. No effort had been made to get a compact representation of sets of parse trees.

So we now have to combine some common substructure sharing in the storage of trees with the overall structure of Earley's algorithm. The idea is to turn from the *set* $\text{tree}(i, j, A \longrightarrow \alpha \cdot \beta)$ of all normal form parse trees t satisfying the condition stated in the characterization theorem, to one single object that identifies a common substructure between trees. At the very least, the top $(A \longrightarrow \alpha \cdot \beta)$ and the leaves $a_{i+1} \cdots a_j$ can be shared.

With this in mind, it is instructive to reconsider the completer part (i.e., step complete (i, k, j) in the terminology of Section 6). For some condition $\text{cond}(i, j, k, \gamma)$, Complete′ takes the form

$$\begin{aligned}
&t \in \text{tree}(i, k, A \longrightarrow \alpha \cdot B\beta\gamma), \\
&s \in \text{tree}(k, j, D \longrightarrow \delta \cdot\,),\ m\colon B \Longrightarrow^* D,\ l\colon \beta \Longrightarrow^* \varepsilon,\ \text{cond}(i, j, k, \gamma) \\
&\quad \Longrightarrow t\hat{}m\hat{}s\hat{}l \in \text{tree}(i, j, A \longrightarrow \alpha B\beta \cdot \gamma).
\end{aligned}$$

We thus construct $|\text{tree}(i, k, A \longrightarrow \alpha \cdot B\beta\gamma)| \cdot |\text{tree}(k, j, D \longrightarrow \delta \cdot\,)|$ many trees. However, as the construction is uniform in s and t, a compact representation of $\text{tree}(i, j, A \longrightarrow \alpha B\beta \cdot \gamma)$ would instead use a schema $(\uparrow t)\hat{}m\hat{}(\uparrow s)\hat{}l$ with *pointers* $\uparrow t$ to $\text{tree}(i, k, A \longrightarrow \alpha \cdot B\beta\gamma)$ and $\uparrow s$ to $\text{tree}(k, j, D \longrightarrow \delta \cdot\,)$. Note that steps later in the parsing process only need i, j, A, and γ, so the schema is sufficient.

This way of representing trees schematically can be improved further. Observe that the construction of tree $t\hat{}m\hat{}s\hat{}l$ is not only uniform in t and s, but also in δ: We only need to know that s is a tree with root D, spanning $a_{k+1} \cdots a_j$. Therefore, we should avoid splitting the set of these trees into disjoint subsets, $\text{tree}(k, j, D \longrightarrow \delta \cdot\,)$ according to different branchings $(D \longrightarrow \delta)$ at the top, but rather we should use the union tree$(k, j, D) = \cup\{\text{tree}(k, j, D \rightarrow \delta\,\cdot\,) \mid (D \rightarrow \delta) \in P\}$. In this way we reduce several schemata $(\uparrow t)\hat{}m\hat{}(\uparrow s_\delta)\hat{}l$, $(D \rightarrow \delta) \in P$, to a single schema $(\uparrow t)\hat{}m\hat{}(\uparrow s)\hat{}l$ with $\uparrow s$ pointing to tree(k, j, D). Consequently, fewer schemata are put into tree$(i, j, A \rightarrow \alpha B\beta \cdot \gamma)$, and the size to store this set is reduced.

In terms of parse trees, the pointers in a schema point to a set of alternative subtrees, while the rule numbers represent a structure shared between trees. The use of tree(k, j, D) can be seen as using a structure combining several trees by at least identifying the leaves $a_{k+1} \cdots a_j$ as well as the root node D.

Note that the use of tree(k, j, D) seems to be the same as Tomita's [11] packed shared forest representation. Tomita also discusses a defect in Earley's representation of parse trees by pointers between items.

So far, completed parse trees for substrings $a_{k+1 \cdots a_j}$ have been considered. In principle, it is also possible to combine partial parse trees further. Namely, the above construction of trees is also uniform in i: $t_1 \in$ tree$(i_1, k, A \rightarrow \alpha \cdot B\beta\gamma)$, and $t_2 \in$ tree$(i_2, k, A \rightarrow \alpha \cdot B\beta\gamma)$ share at least leaves $a_{i_2} \cdots a_k$ (if $i_1 \leq i_2$) and the top branching coming from $(A \rightarrow \alpha B\beta\gamma)$. A common structure could be built, coding the positions i_r into its alternative substructures. This combined structure can well be extended when trees are linked to B. However, having worked through $\beta\gamma$, we need to extract the alternatives in order to put the completed trees into different tree(i_r, k, A)'s. This seems to be an expensive operation and not worth the apparently small gain we could obtain from the kind of structure sharing we discussed.

We sum up our modification into the following algorithm, separating between (schemata for) partial trees *ptree*$(i, j, A \rightarrow \alpha \cdot \beta)$ and completed trees *ctree* (i, j, A):

(9) *The modified Earley's algorithm with compact tree representation*

Input: a string $a_1 \cdots a_n$, a finite set P of production rules, and a start symbol S.

Output: the family of all sets ctree(i, j, A) and ptree$(i, j, A \rightarrow \alpha \cdot \beta)$ as described above.

```
Program:
  begin
    expect (1) := {$S$};
    for $j = 1$ to $n$                                   /* scan ($j$) */
      ctree ($j - 1, j, a_j$) := {($a_j$)};

      for all $C \in$ expect ($j$)                      /* predict ($j - 1$) */
        and $m$: ($A \rightarrow \alpha B\beta$) $\in P$
          with $C \Rightarrow^* A\theta$ for some $\theta \in V^*$,
            $l$: $\alpha \Rightarrow^* \varepsilon$,
            $\beta \neq \varepsilon$,
          and $B \Rightarrow^* a_j\eta$ for some $\eta \in V^*$
        put ptree ($j - 1, j - 1, A \rightarrow \alpha \cdot B\beta$) := {$(m)\hat{}l$};
      for $i = j - 1$ down to 0
        for $k = j - 1$ down to $i$                              /* complete $(i, k, j)$ */
          for all $p$: $(A \rightarrow \alpha B \beta \gamma) \in P$,
              tree $(i, k, A \rightarrow \alpha \cdot B \beta \gamma) \neq \emptyset$,
              $D \in V$
            and ctree $(k, j, D) \neq \emptyset$
              with $m$: $B \Rightarrow^* D$,
                $l$: $\beta \Rightarrow^* \varepsilon$,
            do: if $\gamma = \varepsilon$ and $i < k$,
                put($\uparrow$ptree $(i, k, A \rightarrow \alpha \cdot B \beta \gamma)$)^$m$^($\uparrow$ctree $(k, j, D)$)^$l$
                into ctree $(i, j, A)$;
              if $\gamma = C\theta$ and $C \Rightarrow^* a_{j+1}\eta$ for some $C \in V$ and $\theta, \eta \in V^*$
                put ($\uparrow$ptree $(i, k, A \rightarrow \alpha \cdot B \beta \gamma)$)^$m$^($\uparrow$ctree $(k, j, D)$)^$l$
                into ptree $(i, j, A \rightarrow \alpha B \beta \cdot \gamma)$
                and $C$ into expect $(j + 1)$;
end
```

The parse trees for $a_1 \cdots a_n$ can easily be enumerated from ctree $(0, n, S)$ by recursively substituting pointers by trees in the set of alternatives pointed at. The enumeration function is assumed to distinguish between rule numbers and pointers.

8. EXTENSIONS

Let us mention some possibly useful extensions and modifications.

(a) *Earley parser with lookahead* $r > 1$. We have obtained (almost) the advantage of a parser with lookahead 1 by "cheap" modifications of the well-known algorithm. If we are willing to store along with a dotted rule $(A \rightarrow \alpha \cdot \beta)$ a right context of A of length r (or derive this information from the coded tree), precompute whether $A_1 \cdots A_k \Rightarrow^* a_{j+1} \cdots a_{j+r}\eta$ for some $\eta \in V^*$ and do some further computations concerning deletions during the parse, then we may get the power of lookahead r (see [1, *4.2.18]).

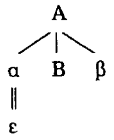
As the main problems are apparently due to many useless rules inserted by Predict (see [5]), we propose a less ambitious solution by strengthening Predict′ only, while leaving Complete′ unchanged:

Predict′ with lookahead r

(i)

$j + r \leq n$,
$1 \leq k < r$,
$C \in \text{expect}(j + 1)$,
$C \Rightarrow^* A\, B_1 \ldots B_k\, \theta$,
$(A \rightarrow \alpha\, B\, \beta) \in P$,
$\alpha \Rightarrow^* \varepsilon$,
$B\, \beta\, B_1 \ldots B_k \Rightarrow^* a_{j+1} \ldots a_{j+r}\, \eta$,
$\beta \neq \varepsilon$, and
$B \Rightarrow^* a_{j+1} \ldots a_{j+l}$ for some $1 \leq l < r$ or
$B \Rightarrow^* a_{j+1} \ldots a_{j+r}$ gamma

$\Rightarrow$

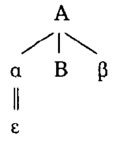


$\in$ tree$(j, j, A \rightarrow \alpha\, .\, B\, \beta)$.

(ii)

$j<n$,
$1 \le k \le m \le \min(r, n\text{-}j)$,
$C \in \mathrm{expect}(j+1)$,
$C \Rightarrow^* A\, B_1 \ldots B_k$,
$(A \to \alpha\, B\, \beta) \in P$,
$\alpha \Rightarrow^* \varepsilon$,
$B\, \beta\, B_1 \ldots B_k \Rightarrow^* a_{j+1} \ldots a_{j+m}$,
$\beta \neq \varepsilon$, and
$B \Rightarrow^* a_{j+1} \ldots a_{j+k}$ for some $1 \le k \le m$

$\Rightarrow$



$\in \mathrm{tree}(j, j, A \to \alpha \cdot B\, \beta)$.

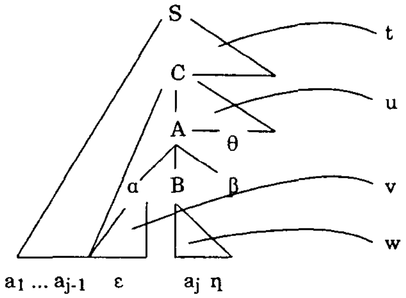
Note that we have to make some precomputations to be able to check whether $\delta \Rightarrow^* a_{j+1} \cdots a_{j+r}\theta$ for some θ, but need not store context or compute deletions during the parse. (The length of δ is bounded.) The conditions on B ensure that B is consistent with $a_{j+1} \cdots a_{j+r}$ and will not be deleted in all succeeding parses.

Clearly, it depends heavily on grammar and average input whether the advantage of small tables and few useless trees affords the additional work needed for the lengthy checks in this predictor. But it seems reasonable to expect that we obtain most of the advantages with considerably less work of known Earley parsers with full lookahead r [1–10].

(b) *Separating terminals in a lexicon.* In applications to natural languages, we separate terminals in a lexicon to avoid searching through lots of useless rules $(A \to a)$ when looking for rules of a different kind. Since the lexicon may be very large, we should not precompute $A \Rightarrow^* a\eta$ for each terminal a, but rather let Predict′ and Complete′ do this for the few terminals found in the input string. To handle lexical ambiguities, a modification of Scan′ might insert into tree$(j-1, j)$ various trees for a single terminal a_j.

(c) *Robustness and error handling.* A parser ought to have some flexibility in reaction to simple input errors such as misprints or unbalanced parentheses in programming languages. Moreover, in parsing natural language an input sentence often will be grammatical although some input word is not stored in the lexicon or some of its syntactical classifications are missing there. How can we avoid refutation in these cases?

The proper place for further modification is Predict′. When some partial parse tree t has been built, Predict′ checks whether there are trees u, v, w and a rule $(A \to \alpha B \beta)$ as shown:



If Predict′ does not succeed, we might call a predictor that is less restrictive:

(i) Instead of checking whether $B \Rightarrow^* a_j\eta$, we make a test if, with $a = a_j$,

$$B \Rightarrow^* b\eta \qquad \text{for some } \eta \in V^* \text{ and } b \in \Sigma \text{ with } aRb,$$

where R is a relation on Σ that may depend on B. For example, aRb could say "a can be viewed as misprint for b" or "a and b are finite forms of the same verb, probably differing in tempus."

That is, when expecting something of type B and reading a next input symbol not consistent with your expectation, see whether it erroneously stands for some B-plausible alternative. If so, try to find a parse giving the alternative reading. Of course, Complete′ has to be modified to be able to substitute b for a_j when linking partial parse trees.

(ii) Suppose no useful lexical entry for a_j can be found. If C is a lexical category, assume a_j has category C (and perhaps make an entry in the lexicon). Else use further lookahead and make a test whether

$$\beta \Rightarrow^* a_{j+1}\eta \qquad \text{or} \qquad B \Rightarrow^* Xa_{j+1}\eta \text{ for some } X \in V$$

instead of $B \Rightarrow^* a_j\eta$. If you succeed, assume a_j has (open lexical) category B or X respectively, and continue.

(iii) Existence of tree u above can be weakened in a manner generalizing how we handled existence of w in (i). For simplicity, suppose we are working in a grammar with parameterized categories $A(p)$. For each expectation $C(p)$ where Predict′ did not succeed, make a further call of Predict′ with expectations $C(q)$ for certain "wrong" parameters q. Suppose you find a derivation tree $s(q)$ verifying $C(q) \Rightarrow^* a_j \cdots a_m$ for some $j \leq m \leq n$. Then the completer has to transform $s(q)$ into a tree $s(p)$ for some derivation $C(p) \Rightarrow^* b_j \cdots b_r$ before linking it to tree t.

What is needed here is a transformational component that is able to turn a phrase α of type A into a phrase α' of type A', working recursively on derivation trees. For example, there might be a case parameter in noun phrases and a procedure turning a complex dative noun phrase into the corresponding nominative noun phrase.

Finding $s(q)$ could also indicate an error in tree t. Then we would need a transformation of t, first substituting the node marked $C(p)$ by $C(q)$ and then reorganizing this tree to eliminate p in favor of q. The tree $t(q)$ thus obtained can then be linked to $s(q)$.

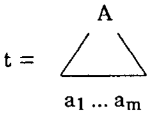
Clearly, this last method of error handling would be more difficult and expensive than the other ones. Note that there are connections to error-generating rules in compiling.

(d) *Extensions to ID/LP-grammars and unification-based grammars.* There seems to be no problem in generalizing what is discussed here to Immediate-Dominance/Linear-Precedence grammars and to Shieber's parser [10].

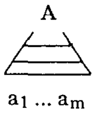
Also, it is easy to extend our method to grammars with a context-free skeleton and features attached to its symbols. Roughly, one can view features as algebraic terms over some set of function symbols, which may contain free variables. The only modification to our algorithm is that when trees are glued together the features at the nodes becoming identified have to be unified. However, there is a problem, in that for such grammars the precomputation of the first relation need not terminate. For example, a rule $(A(x) \longrightarrow A(fx)B(gx))$ will lead to an infinite first relation, containing $(A(x^n), A(f^n_x))$ for each n. There is a remedy to this problem that computes a finitely representable relation $\mathrm{FIRST}^+ \supseteq \mathrm{FIRST}$ from the productions. Using FIRST^+ in the algorithm will exclude fewer unnecessary partial parse trees than using FIRST, but does not admit incorrect ones. The computation of FIRST^+ is too complicated to be included here; in the example, FIRST^+ would contain $(A(yx), A(f_y))$, instead of $(A(x), A(f^n_x))$ for each n. As unification is an expensive operation in the natural language grammars—in fact, it is more complicated than term unification—it is useful to keep the parsing table small: This reduces the number of comparisons between symbols, and hence the number of unifications.

APPENDIX: Proof of the Characterization Theorem

If

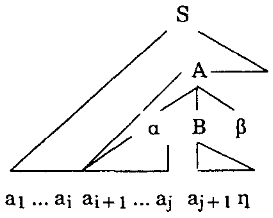


is a derivation tree, we denote its normal form by

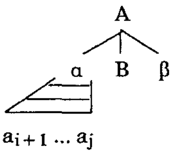


By induction on j, we simultaneously prove claims (i) and (ii) below:

CLAIM (i). *If there is a tree*



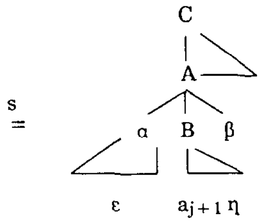
with $i < j$ or $\beta \neq \varepsilon$, then



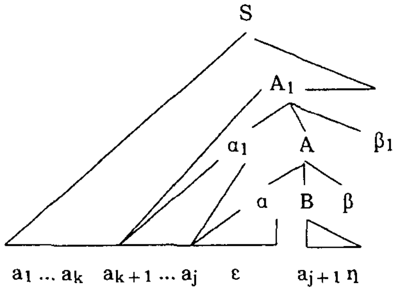
$\in$ tree$(i, j, A \rightarrow \alpha \cdot B\,\beta)$

PROOF

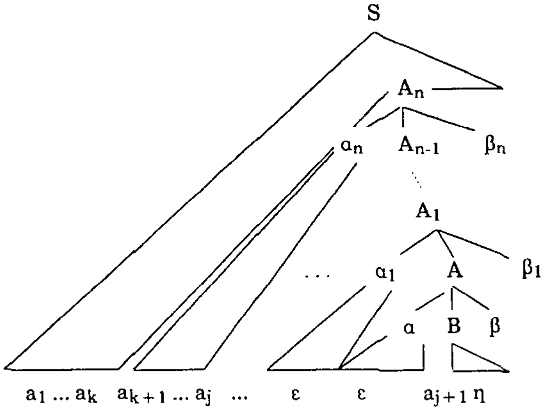
Case 1: $i = j$. It is sufficient to show that there is a tree



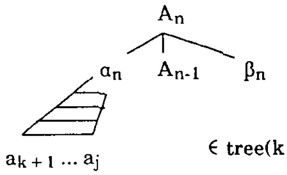
with $C \in \text{Expect}(j + 1)$, since the claim then follows using (6a). Now, if $j = 0$, take $C = S \in \text{Expect}(1)$ and note that $S \Rightarrow^* A\theta$ by assumption. If $j > 0$, we can split the given tree as follows:



If $k = j$, we have $A_1 \Rightarrow^* A\beta_1$ and split in the same way with A_1 instead of A, until finally we come to

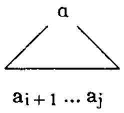


with $k < j$. By induction on Claim (i), Case 2, $k < j$, we get

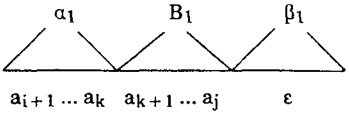


$\in \text{tree}(k, j, A_n \rightarrow \alpha_n \cdot A_{n-1}\, b_n)$, hence, $A_{n-1} \in \text{Expect}(j + 1)$. Noting that $A_{n-1} \Rightarrow^* A\beta_1 \cdots \beta_{n-1}$, we can choose A_{n-1} for C.

Case 2: $i < j$. Split α into $\alpha_1 B_1 \beta_1$ and

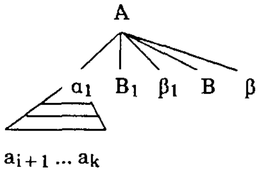


into



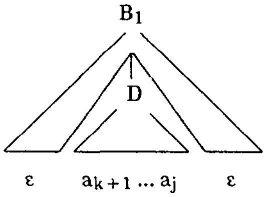
with $a_{k+1} \cdots a_j \neq \varepsilon$.

By induction on Claim (i), $k < j$, we have

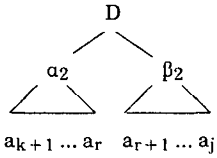


$\in \text{tree}(i, k, A \rightarrow \alpha_1 \cdot B_1\, \beta_1\, B\, \beta)$.

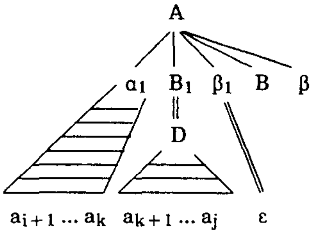
Now using Claim (ii) for $k < j$, we can split the tree for $B_1 \Rightarrow^* a_{k+1} \cdots a_j$ into



such that the subtree with root D either is the singleton tree $.a_j \in \text{tree}(k, j, a_j \rightarrow a_j \cdot\,)$ or splits into

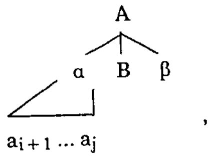


for some $k < r < j$, and its normal form belongs to $\text{tree}(k, j, D \rightarrow \alpha_2\beta_2 \cdot\,)$. Using $B_1 \Rightarrow^* D$ and (6d$_2$), we get



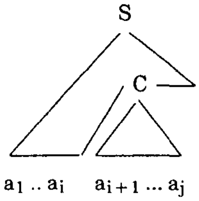
$\in \text{tree}(i, j, A \rightarrow \alpha \cdot B\, \beta)$.

Since the subtree with root D either is the singleton $.a_j$ or has a true branching at its root, this tree is in normal form. Clearly, it is the normal form of

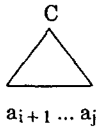


and we are done with Case 2. □

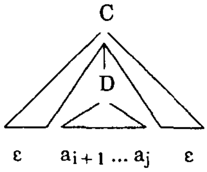
CLAIM (ii). *If there is a derivation tree*



with $i < j$, then we can split

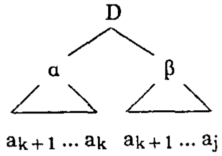


into

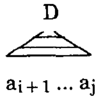


such that

(*a*) *either $i + 1 = j$ and the tree with root D is the singleton tree $.a_j \in tree(i, j, a_j \rightarrow a_j \cdot)$, or*
(*b*) *for some k with $i < k < j$ this tree splits into*



with

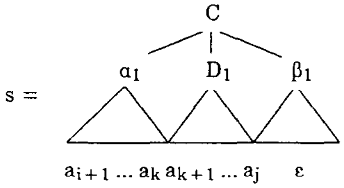


$\in tree(i, j, D \rightarrow \alpha\,\beta \cdot)$.

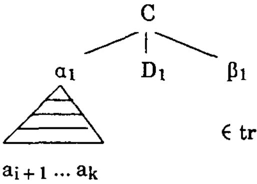
PROOF. By induction on the depth of the subtree s with root C.

Case 3: $s = .a_j$. Then the claim is clear by (6b).

Case 4:

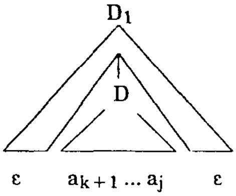


for some $i < k < j$. Then by Claim (i), $k < j$, we know that

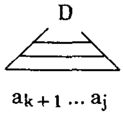


$\in$ tree(i, k, $C \to \alpha_1 \cdot D_1 \beta_1$),

and by induction on Case 4, we can split the subtree with root D_1 into

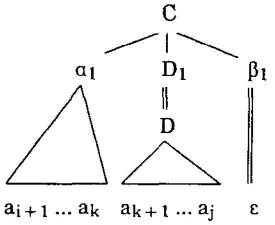


such that



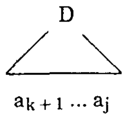
$\in$ tree(k, j, $D \to \delta \cdot$).

Hence, by ($6d_1$),



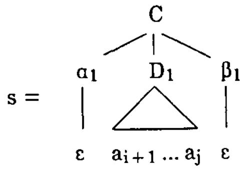
$\in$ tree(i, j, $C \to \alpha \cdot$).

This tree is in normal form, since by Claim (ii) we know that

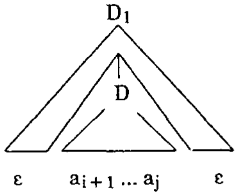


is either a singleton tree $.a_j$ or branches at the root to two nonempty subwords of $a_{k+1} \cdots a_j$. (Hence, there is no linking of chain derivations at node D.) As this tree is the normal form of s, the proof of Case 4 is finished.

Case 5:



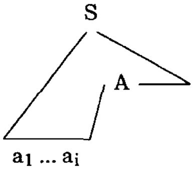
Note that the subtree with root D_1 and yield $a_{i+1} \cdots a_j$ is smaller than s, whence by induction on Claim (ii) we can split it into



such that the subtree with root D satisfies the conditions in Claim (ii). Since $C \Rightarrow^* D_1 \Rightarrow^* D$, we are done with Case 5.

Hence, Claim (ii) is shown. □

PROOF OF THE CHARACTERIZATION THEOREM FROM CLAIMS (I) AND (II). Direction $\Leftarrow$ for (i) and (ii) of the theorem can easily be obtained from the claims. The reverse implication $\Rightarrow$ is done by induction and a careful look on how a tree is entered into tree$(i, j, A \rightarrow \alpha \cdot Bb)$ by the rules in (6): First note that if $A \in$ Expect$(i + 1)$ or tree$(i, j, A \rightarrow \alpha \cdot B\beta) \neq \varnothing$, there is a derivation tree



This follows from Expect(1) = $\{S\}$ by induction via (6c) and (6d$_2$). Moreover, the rules in (6) only allow one to build trees in normal form. Now suppose $t \in$ tree$(i, j, A \rightarrow \alpha \cdot Bb)$. This can happen only as the result of applying (6c) or (6d$_2$). The claim of (i) is clear by the above remarks, using induction in the completer case. Similarly, if $t \in$ tree$(i, j, D \rightarrow \delta \cdot)$, we must have used (6b) or (6d$_1$). The claim of (ii) is again clear, using induction and Claim (i) in the completer case. □

ACKNOWLEDGMENTS

I want to thank Bernhard Thull for his enthusiasm in writing a LISP implementation of the parser. It motivated me to clarify some of the subtleties involved in the modification reported here. I also thank the referees for helpful criticism.

REFERENCES

1. AHO, A., AND ULLMAN, J. *The Theory of Parsing, Translation, and Compiling. Parsing, vol. 1.* Prentice-Hall, Englewood Cliffs, N.J., 1972.
2. DOERRE, J., AND MOMMA, S. Modifikationen des Earley-Algorithmus und ihre Verwendung für ID/LP-Grammatiken. Stuttgarter Arbeiten zur Computerlinguistik, Universität Stuttgart, FRG, Nov. 1985.
3. EARLEY, J. An efficient context-free parsing algorithm. *Commun. ACM 13*, 2 (Feb. 1970) 94–102.
4. GAZDAR, G., KLEIN, E., PULLUM, G. D., AND SAG, I. *Generalized Phrase Structure Grammar.* Basil Blackwell, Oxford, U.K., 1985.
5. GRAHAM, S., HARRISON, M., AND RUZZO, W. An improved context-free recognizer. *ACM Trans. Program. Lang. Syst. 2*, 3 (July 1980), 415–462.
6. HARRISON, M. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, Mass., 1978.
7. KILBURY, J. Earley-basierte Algorithmen für direktes Parsen mit ID/LP-Grammatiken. KIT-Rep. 16, Institut für angewandte Informatik, TU Berlin, Berlin, June 1984.
8. KILBURY, J. A modification of the Earley–Shieber algorithm for direct parsing with ID/LP-grammars. In *GWAI-84. Proceedings of the 8th German Workshop on Artificial Intelligence* (Wingst/Stade, Oct. 1984). *Informatik-Fachberichte* 103, J. Laubsch, Ed. Springer-Verlag, Berlin, 1985, pp. 39–48.
9. PULMAN, S. G. Generalized phrase structure grammar, Earley's algorithm, and the minimization of recursion. In *Automatic Natural Language Parsing*, K. Sparck-Jones and Y. Wilks, Eds. Ellis Horwood, Chichester, U.K., 1983, pp. 117–131.
10. SHIEBER, S. Direct parsing of ID/LP grammars. *Linguist. Philos. 7* (1984), 135–154.
11. TOMITA, M. An efficient context-free parsing algorithm for natural languages. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence* (Los Angeles, Ca., Aug. 18–23, 1985), pp. 756–764.
12. WANG, W. The comparison of Kilbury's algorithm with the Earley's for parsing context-free grammars. BUCS Tech. Rep. 85/013, Dept. of Computer Science, Boston Univ., Boston, Mass., Nov. 1985.

Received August 1986; revised August 1988; accepted December 1989