COMPUTATIONAL COMPLEXITY OF LEFT-ASSOCIATIVE GRAMMAR

ROLAND HAUSSER

Institut für Deutsche Philologie
Ludwig-Maximilians Universität München
München, West Germany

1. CHOICE OF A PRIMITIVE OPERATION

The first step in the analysis of computational efficiency of a new algorithm is the choice of the **primitive operation.** Consider how Earley specified and motivated his choice of a primitive operation for the Earley algorithm:

> "Griffiths and Petrick... have expressed their algorithms as sets of nondeterministic rewriting rules for a Turing-machine-like device. Each application of one of these is a primitive operation. We have chosen as our primitive operation the act of adding a state to a state set (or attempting to add one which is already there). We feel that this is comparable to their primitive operation because both are in some sense the most complex (! RRH.) operation performed by the algorithm whose complexity is independent of the size of the grammar or the input string."
>
> Earley 1970, p. 100.

The reason why Earley is able to use the act of adding, or attempting to add, a state to a sate set as his primitive operation is the following. Even though "for some grammars the number of states in a state set can grow indefinitely with the length of the string being recognized"[1], the operation of testing whether or not a state has already been added to the state set is handled in such a way that the size of the search space is independent of the length of the string under analysis.[2] This is possible because of a structured build up of the state set The efficiency of the Earley algorithm derives directly from the fact that the presence of a state in a state set can be checked in an amount of time which is independent of the size of the state set

[1] Earley 1970, p. 98.

[2] See Earley 1970, p. 97, (3).

2. THE FORMAL DEFINITION OF LA-GRAMMAR

What is the most natural definition of a primitive operation in LA-grammar? In order to answer this question we need a formal definition of LA-grammar.[3]

1 *Formal definition of Left-associative Grammar:*

A left-associative grammar (or LA-grammar) is defined as a 6-tuple $<W, C, LX, CO, RP, rp_S>$, where

1. W is a finite set of *word surfaces*;
2. C is a finite set of *category segments*;
3. $LX \subset (W \times C^+)$ is a finite set comprising the *lexicon*;[4]
4. $CO = (co_0 \ldots co_{n-1})$ is a finite sequence of total recursive functions from $(C^* \times C^+)$ into $C^* \cup \{\perp\}$ called *categorial operations*.
5. $RP = (rp_0 \ldots rp_{n-1})$ is an equally long sequence of subsets of n called *rule packages*.
6. rp_S is a subset of n called the *start rule package*.

For theoretical reasons, the categorial operations are defined as total functions. In practice, the categorial operations are defined on easily recognizable subsets of $(C^* \times C^+)$, where anything outside these subsets is mapped into the arbitrary "don't care" value $\{\perp\}$, making the categorial operations total.

An LA-grammar is usually specified in terms of (i) a lexicon LX, (ii) a set of start states ST_S, (iii) a sequence of rules, and (iv) a set of final states ST_F. Let us illustrate this general format of LA-grammars with a simple example of a formal language, namely the context-sensitive language $a^kb^kc^k$.

2 *The definition of* $a^kb^kc^k$:[5]

$LX =_{def} \{[a\ (bc)], [b\ (b)], [c\ (c)]\}$
$ST_S =_{def} \{(\{r\text{-}1, r\text{-}2\}\ (bc))\}$
r-1: $[(X)\ (bc)] \Rightarrow [\{r\text{-}1, r\text{-}2\}\ (bXc)]$,
r-2: $[(bXc)\ (b)] \Rightarrow [\{r\text{-}2, r\text{-}3\}\ (Xc)]$,
r-3: $[(cX)\ (c)] \Rightarrow [\{r\text{-}3\}\ (X)]$
$ST_F =_{def} \{[rp\text{-}3\ \epsilon]\}$.

[3]Let us recall some notation from set theory needed for this purpose. If X is a set, then X^+ is the "positive closure", i.e., the set of all concatenations of elements of X. X^* is the Kleene closure of X, defined as $X^+ \cup \epsilon$, where ϵ is the "empty sequence". The power set of X is denoted by 2^X. If X and Y are sets, then $(X \times Y)$ is the Cartesian product of X and Y, i.e., the set of ordered pairs consisting of an element of X and an element of Y. For convenience we also identify integers with sets, i.e., $n = \{i \mid 0 \leq i < n\}$.

[4]Since LX is finite, each $w \in W$ is related by LX to a non-empty finite set of elements of C^+.

Given example 2, let us consider the relation between the definition of LA-grammar as a 6-tuple <W, C, LX, CO, RP, rp_s>, and the specification of an LA-grammar in terms of LX, ST_s, RL, and ST_F. The sets of word surfaces W and category segments C are implicitly characterized in the definition of LX: W $=_{def}$ {a, b, c} and C $=_{def}$ {b, c}. The sequences CO and RP, furthermore, are implicitly characterized in the definition of the rules r-1, r-2, and r-3. rps, finally, is specified in terms of ST_S.

As an illustration of the relation between a LA-grammar and its parser consider the following NEWCAT derivation of "aaabbbccc" using the grammar for $a^k b^k c^k$ defined in 2.

3 *Sample derivation of* aaabbbccc *with active rule counter:*[6]

```
* (z aaabbbccc)
  1: Applying rules (RULE-1 RULE-2)
  2: Applying rules (RULE-1 RULE-2)
  3: Applying rules (RULE-1 RULE-2)
  4: Applying rules (RULE-2 RULE-3)
  5: Applying rules (RULE-2 RULE-3)
  6: Applying rules (RULE-2 RULE-3)
  7: Applying rules (RULE-3)
  8: Applying rules (RULE-3)
  Number of rule applications: 14.

    *START-0
    1
        (B C) A
        (B C) A
    *RULE-1
    2
        (B B C C) A A
        (B C) A
    *RULE-1
    3
        (BBBCCC) AAA
        (B) B
    *RULE-2
    4
        (BBCCC) AAAB
        (B) B
```

[5] (X) is a variable for sequences of category segments (cf. r-1, r-2, and r-3 in 2). For example, if (X) is (cc), then (bX) = (bcc). Strictly speaking, the surfaces of well-formed expressions of $a^k b^k c^k$ should be represented as, e.g., (aaabbbccc) rather than "aaabbbccc". The parentheses surrounding sentence start surfaces are omitted for simplicity. The parentheses are present in the representation of the categories, however, in order to maintain the distinction between categories, e.g., (b), and category segments, e.g., "b". In LA-grammars of natural language a sequence consisting of "a", "b", "c" is written as (a b c) rather than (abc) in order to permit use of category segments like "S3" consisting of more than one letter.

6 The rule counter is part of the testing environment of LA-grammar, and was written with the help of Mr. Todd Kaufmann (Carnegie-Mellon University).

```
*RULE-2
5
    (B C C C) A A A B B
    (B) B
*RULE-2
6
    (C C C) A A A B B B
    (C) C
*RULE-3
7
    (C C) A A A B B B C
    (C) C
*RULE-3
8
    (C) A A A B B B C C
    (C) C
*RULE-3
9
    (NIL) A A A B B B C C C
```

The rule applications specify (i) the number of the combination step, e.g. "; 3:", and the rule package(s) active at this combination step, e.g., "(RULE-1 RULE-2)". The number of rules fired in a combination step is the sum of all rules in the rule packages associated with this combination step. Since $a^kb^kc^k$ is an unambiguous language, each combination step has only one rule package. In ambiguous derivations at least one combination step number occurs more than once, which means that more than one rule package is fired in the combination. The rule applications in 3 show that the first 2k combination steps involve two applications each, whereas the remaining k-1 combination steps involve only one rule application. The LA-grammar defined in 2 parses well-formed strings of length *n* in exactly *(4/3n + l/3(n - 1))* rule applications. That is, $a^kb^kc^k$ parsed in linear time. Furthermore, a parallel implementation of the LA-grammar for $a^kb^kc^k$ with two processors would parse with time complexity of *(n-1).*

3. THE HIERARCHY OF LA-GRAMMARS

For purposes of complexity analysis, the crucial formal property of a categorial operation is whether or not it has to search through indefinitely long sentence start categories.

4 *Definition of the class of* **C-LAGs**:

> The class of *constant* LA-grammars or C-LAGs consists of grammars where no categorial operation co i looks at more than K segments in the sentence start categories, for a finite constant K.[7] A language is called a C-language iff it is recognized by a C-LAG.

[7] This finite constant will vary between different grammars.

LA-grammars for regular and context-free languages are all C-LAGs because in regular languages the length of the sentence start category is restricted by a finite constant (cf. Theorem 3, 4), and in context-free languages the categorial operation may only look at a finite number of segments at the beginning of the sentence start category (cf. Theorem 4, 5). But the LA-grammars for many context-sensitive languages, e.g., $a^kb^kc^k$, $a^kb^kc^kd^ke^k$, WW, and WWW, are also C-LAGs.

Generally speaking, a LA-grammar is a C-LAG if its rules conform to the following schemas:

r_i: [(seg-1 ... seg-k X) CAT-2] => [RP_i CAT-3]

r_i: [(X seg-1 ... seg-k) CAT-2] => [RP_i CAT-3]

r_i: [(seg-1 ... seg-i X seg-i+1 ... seg-k) CAT-2] => [RP_i CAT-3]

Thereby CAT-3 may contain at most one sequence variable (e.g. X).

On the other hand, if an LA-grammar has rules of the form

r_i: [(X seg-1 ... seg-k Y) CAT-2] => [RP_i CAT-3]

the grammar is not a constant LA-grammar. In non-constant LA-grammars CAT-3 may contain more than one sequence variable (e.g. X and Y).[8]
Non-constant LA-grammars are divided into the *B-LAGs* and *A-LAGs.*

5 *Definition of the class of* **B-LAGs**:

> The class of *bounded* LA-grammars or *B-LAGs* consists of grammars where for any complete well-formed expression E the length of intermediate sentence start categories is bounded by C • n, where n is the length of E and C is a constant. A language is called a B-language if it is recognized by a B-LAG, but not by a C-LAG.

6 *Definition of the class of* **A-LAGs**:

> The class of **A-LAGs** consists of all LA-grammars because there is no bound on the length of the categories, or the number of category segments read by the categorial operations. A language is called an A-language if it is recognized by an A-LAG, but not by a B-LAG.

The three classes of LA-grammars defined above are related in the following hierarchy.

7 *The* **hierarchy** *of A-LAGs, B-LAGs, and C-LAGs:*

[8] The exact definition of C-LAGs and B-LAGs benefitted from a discussion with Professor Helmut Schwichtenberg.

The class of A-LAGs recognizes all recursive languages, the class of B-LAGs recognizes all context-sensitive languages, and the class of C-LAGs recognizes most context-sensitive languages, all context-free languages, and all regular languages.

Let cs_c represent the context-sensitive languages recognized by C-LAGs and CSb the context-sensitive languages recognized by B-LAGs. Then the conventional classes of regular (r), context-free (cf), context-sensitive (cs), recursive (rec), and recursively enumerable languages (r.e.) relate to the A-, B-, and C-languages as follows:

$$\mathbf{r} \subset \mathbf{cf} \subset \mathbf{cs}_c \subseteq \mathbf{C} \subset \mathbf{cs}_b \subseteq \mathbf{B} \subset \mathbf{A} = \mathbf{rec} \subset \mathbf{r.e.}$$

4. THE PRIMITIVE OPERATION OF C-LAGS

The most complex operation whose complexity is independent of the size of the grammar or the input string is the application of a rule to a given ss-nw pair. However, rule applications may be taken as the primitive operations of LA-grammar only if categorial operations do not have to search through indefinitely long sentence start categories. This condition is satisfied by the class of C-LAGs (cf. 4).

8 *The* **primitive operation** *of C-LAGs:*

In C-LAGs the *primitive operation* is defined as the application of a rule to a given ss-nw pair.

B-LAGs are not necessarily slower than C-LAGs. It is just that their complexity analysis cannot use rule applications as their primitive operations because the categorial operations may have to look at an indefinite number of CAT-1 segments. Since C-LAGs cover all context-free languages as well as many context-sensitive languages, our discussion of C-LAG complexity is considerably more general than the traditional discussion of context-free PS-grammar complexity.

Whether or not a given LA-grammar is a C-LAG is obvious from the structure of the rules. Furthermore, the exact complexity of a given input string is provided automatically by the rule counter during a parse. In addition, based on the grammar and the complexity measures of inputs, it is often possible to find a "closed form expression" which characterizes the complexity of the grammar for arbitrary n. Thus, the C-LAG 2 for $a^kb^kc^k$ was determined to parse in *(4/3n + 1/3(n -1)).*

5. THE COMPLEXITY OF UNAMBIGUOUS C-LAGS

Beyond the complexity analysis of individual grammars, however, we would like to arrive at general results for whole classes of languages. The first such general result is presented in Theorem 7.

9 *Theorem 7:*

> Unambiguous C-LAGs are parsed in $C \bullet n$, where C is a small constant representing the maximal number rules in a rule package, and n is the length of the input.
>
> Proof:
>
> An LA-grammar is unambiguous iff (i) it holds for all rule packages that their rules have incompatible input conditions, and (ii) there are no lexical ambiguities. Therefore, each combination step results in at most one continuation. Thus the number of elementary operations at any transition is equal to the number of rules in the current rule package.
>
> *Q.E.D.*

This result is considerably better than that of Earley (1970). Earley's algorithm parses unambiguous context-free languages in $|G|^2 \bullet n^2$, where |G| is the size of the context-free grammar, and n is the length of the input string.

First, the complexity of the Earley's algorithm, as well as any other conventional parsing algorithm, depends heavily on the size |G| of the grammar,[9] whereas LA-grammar complexity is independent of the size of the grammar. Second, regarding the length of the input n, LA-grammar parses C-LAGs in linear time n, whereas the Earley algorithm parses context-free grammars in cubic time n^2. And third, C-LAGs cover not only all context-free languages but also a large portion of the context-sensitive languages, while the Earley algorithm, as well as all other conventional general purpose parsers such as CYK, parse only the context-free languages, or a mere subset of the context-free languages (e.g. LR-parsers).

6. COMPLEXITY COMPARISONS

To get a feeling for the relation between PS-grammars and equivalent LA-grammars, and their respective behavior in terms of efficiency, let us consider

[9] Barton, Berwick, and Ristad 1986 say on p. 250: "Crucially, grammar size affects recognition time in all known CFG recognition algorithms. For GPSG, this corresponds to the set of admissible local trees, and this set is astronomical...".

the formal languages described in Earley 1970, namely ab^k, a^kb, a^kb^k, ab^kcd^m, Propositional Calculus, GRE, and NSE. We describe each language[10] in terms of (i) an LA-grammar, (ii) the PS-grammar provided by Earley 1970, and (iii) the complexity results for LA-grammar, the Earley algorithm, and - if available - the BU (bottom-up), SBU (specialized bottom-up), TD (top-down), and STD (specialized top-down) algorithms as evaluated by Griffiths and Petrick 1965. The LA-grammars are presented in canonical form, consisting of (i) a lexicon LX, (ii) a set of start states ST_s, (iii) a set of rules, called r-0, r-1, etc., and (iv) a set of final states ST_F. Note that Earley formulated the PS-grammars for ab^k, a^kb, etc., such that exponents like k must usually be interpreted as > 0.

10 *The context-free language* a^kb^k:

1. **Formulation in LA-grammar:**

   $\mathbf{LX} =_{def} \{(a\ (a)),\ (b\ (b))\}$
   $\mathbf{ST}_S =_{def} \{[\{r\text{-}1,\ r\text{-}2\}(a)]\}$
   $\mathbf{r\text{-}1}$: $[(X)(a)] \Rightarrow [\{r\text{-}1,\ r\text{-}2\}(a\ X)]$
   $\mathbf{r\text{-}2}$: $[(a\ X)(b)] \Rightarrow [\{r\text{-}2\}(X)]$
   $\mathbf{ST}_F =_{def} \{[rp\text{-}2,\ \epsilon]\}$

2. Formulation in PS-grammar:

   (1) $S \rightarrow aSb$
   (2) $S \rightarrow ab$

3. Complexity (Number of operations per input of length n):

   Early: [6n + 4]
   TD: [5n - 1]
   STD: [5n - 1]
   BU: [11 • 2n-1]
   SBU: [6n]
   LAG: [(n-1) + l/2n]

For the languages Propositional Calculus, NSE, and GRE neither Petrick and Griffiths 1965 nor Earley 1970 provide "closed-form expressions" for their complexity results. Instead, Earley 1970 gives the number of operations for specific sentences of the languages. In the case of propositional calculus grammar and NSE, Earley 1970 provides data for the PA[11], SBU, and the Earley

[10] ab^k, a^kb, and ab^kcd^m are omitted for reasons of space.

[11] The 'predictive analyzer', a modified top-down algorithm described in Griffiths and Petrick 1965.

algorithm. In the case of GRE, Earley 1970 provides data for the SBU and the Earley algorithm only.

11 *The context-free language* **Propositional Calculus:**

1. Formulation in LA-grammar:

LX = {(p (T)), (q (T)), (r (T)), (p' (T)), (q' (T)), (r' (T)), (and (BIN)), (or (BIN)), (impl (BIN)), (not (NEG)), ([(L)), (] (R))}
$\mathrm{ST}_S =_{def}$ {[{r-1, r-2}(seg1)]}, where seg1 ε {L, T, NEG}
r-1: [(X NEG)(seg2)] ⇒ [{r-1, r-2, r-3} (X seg2)],
where seg2 ε {L, T}
r-2: [(X L)(seg1)] ⇒ [{r-1, r-2, r-3}(X L seg1)],
where seg1 ε {L, T, NEG}
r-3: [(X T)(BIN)] ⇒ [{r-4}(X BIN)]
r-4: [(X BIN)(seg1)] ⇒ [{r-1, r-2, r-3, r-5}(X seg1)],
where seg ε {L, T, NEG}
r-5: [(X L T)(R)] ⇒ [{r-3 r-5}(X T)]
$\mathrm{ST}_F =_{def}$ {[rp-0 (T)], [rp-4 (T)], [rp-5 (T)]}

2. Formulation in PS-grammar:

(1) F → C	(9) L → L'
(2) F → S	(10) L → p
(3) F → P	(11) L → q
(4) F → U	(12) L → r
(5) C → U impl U	(13) S → U or S
(6) U → (F)	(14) S → U or U
(7) U → not U	(15) P → U and P
(8) U → L	(16) P → U and U

3. Complexity (Number of operations per input of length n):

Sentence	Length	PA	SBU	Earley	LAG
p	1	14	18	28	1
(p and q)	5	89	56	68	11
(p' and q) or r or p or q'	13	232	185	148	24
p impl ((q impl not (r' or (p and q))) impl (q' or r))	26	712	277	277	57
not (not p' and (q or r) and p')	17	1955	223	141	32
((p and q) or (q and r) or (r and p')) impl not ((p' or q') and (r' or p))	38	2040	562	399	84

12 *The regular language GRE:*

1. **Formulation in LA-grammar:**

LX $=_{def}$ {(a (a)), (b (b)), (e (e)), (d (d))}
$ST_S =_{def}$ {[{r-1, r-2}(seg1)]}
r-1: [(seg1)(seg2)] $\Rightarrow$ [∅ (a)],
where seg1 = a and seg2 = b, or seg1 = e and seg2 = a.
r-2: [(e)(d)] $\Rightarrow$ [{r-3}(d)]
r-3: [(d)(e)] $\Rightarrow$ [{r-2, r-4}(e)]
r-4: [(e)(a)] $\Rightarrow$ [{r-5}(b)]
r-5: [(b)(b)] $\Rightarrow$ [{r-5}(b)]
$ST_F =_{def}$ {[rp-1 (a)], [rp-4 (b)], [rp-5 (b)]}

2. Formulation in PS-grammar:

(1) X $\rightarrow$ a	(4) Y $\rightarrow$ e
(2) X $\rightarrow$ Xb	(5) Y $\rightarrow$ YdY
(3) X $\rightarrow$ Ya	

3. Complexity (Number of operations per input of length n):

Sentence	Length	PA	SBU	Earley	LAG
ededea	6	35	52	33	8
ededeab^{4}	10	75	92	45	12
ededeab^{10}	16	99	152	63	18
ededeab^{200}	206	859	2052	663	208
(ed)^{4}eabb	12	617	526	79	16
(ed)^{7}eabb	18	24352	16336	194	25
(ed)^{8}eabb	20	86139	54660	251	28

The worst case for this LAG is the sequence 'eded...', which requires *3/2n* steps.

13 *The regular language NSE:*

1. **Formulation in LA-grammar:**

LX $=_{def}$ {(a (a)), (b (b)), (c (c)), (d (d))}
$ST_S =_{def}$ {[{r-1, r-2, r-3}(a)]}
r-1: [(a)(d)] $\Rightarrow$ [{r-3, r-4} (d)]
r-2: [(a)(b)] $\Rightarrow$ [{r-5, r-6}(b)]

r-3: [(d)(d)] ⇒ [{r-3, r-4}(d)]
r-4: [(d)(b)] ⇒ [{r-5}(b)]
r-5: [(b)(c)] ⇒ [{r-6, r-7}(c)]
r-6: [(c)(d)] ⇒ [{r-3, r-4}(d)]
r-7: [(c)(b)] ⇒ [{r-5}(b)]
$ST_F =_{def}$ {[rp-2 (b)], [rp-4 (b)], [rp-7 (b)]}

2. Formulation in PS-grammar:

(1) $S \rightarrow AB$	(5) $B \rightarrow DB$
(2) $A \rightarrow a$	(6) $C \rightarrow c$
(3) $A \rightarrow SC$	(7) $D \rightarrow d$
(4) $B \rightarrow b$	

3. Complexity (Number of operations per input of length n):

Sentence	Length	SBU	Earley	LAG
adbcddb	7	43	44	13
$ad^3bcbcd^3bcd^4b$	18	111	108	34
$adbcd^2bcd^sbcd^3b$	19	117	114	37
$ad^{18}b$	20	120	123	39
$a(bd)^3d^2(bcd)^2dbcd^4b$	24	150	141	46
$a(bcd)^2dbcd^3bcb$	16	100	95	32

Since none of the LA-rules have rule packages containing more than two rules, this LAG parses in less than *2(n-l)* steps (linear time).

The analysis of the language NSE completes the comparison of grammars for the languages described in Earley 1970. In each of the above comparisons the LAG-algorithm turned out to be by far the fastest Furthermore, the LAG-algorithm parses a much larger class of languages than the Earley algorithm, or any other general purpose parser.

7. THE COMPLEXITY OF AMBIGUOUS C-LAGS

Space does not permit presentation of the complexity results for ambiguous C-LAGs. It turns out, however, that the excellent complexity behavior of unambiguous C-LAGs extends also to ambiguous C-LAGs. For a detailed discussion and proof of complexity see Hausser 1988, forthcoming.

Aho, A.V., and J.D. Ullman 1972. *The Theory of Parsing, Translation, and Compiling. Vol.1: Parsing,* Prentice Hall, Englewood Cliffs, New Jersey.

Aho, A.V., and J.D. Ullman 1979. *Principles of Compiler Design,* Addison-Wesley, Reading, Massachusetts.

Barton, G.E., R.C. Berwick, and E.S. Ristad 1987. *Computational Complexity and Natural Language,* MIT-Press, Cambridge, Massachusetts.

Earley, J. 1970. *"An Efficient Context-Free Parsing Algorithm",* CACM 13(2): 94-102.

Griffiths, T., and Petrick, S. 1965 *"On the Relative Efficiencies of Context-Free Grammar Recognizers",* CACM 8, p. 289-300.

Hausser, R. 1985. *"Left-associative Grammar and the Parser NEWCAT",* Center for the Study of Language and Information, Stanford University, IN-CSLI-85-5.

Hausser, R. 1986. *NEWCAT: Parsing Natural Language Using Left-associative Grammar,* Lecture Notes in Computer Science, Springer-Verlag, Berlin.

Hausser, R. 1987. *"Left-Associative Grammar: Theory and Implementation",* Center for Machine Translation, Carnegie-Mellon University, CMU-CMT-87-104.

Hausser, R. 1988a. *"Left-Associative Grammar, an Informal Outline",* Computers and Translation, Vol. 3.1:23-67, Kluwer Academic Publishers, Dordrecht.

Hausser, R. 1988b. *"Algebraic Definitions of Left-Associative Grammar",* Computers and Translation, Vol. 3.2 (1988), Kluwer Academic Publishers, Dordrecht

Hausser, R. forthcoming. *Computation of Language,* Springer-Verlag, Heidelberg, 1988.

Shieber, S., S. Stucky, H. Uszkoreit, and J. Robinson 1983. *"Formal Constraints on Metarules",* in *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics,* Cambridge, Massachusetts.

Tomita, M. 1986. *Efficient Parsing for Natural Languages,* Kluwer Academic Publishers, Boston-Dordrecht.

Younger, D.H. 1966 *"Context-Free Language Processing in n^3",* General Electric R & D Center, Schenctady, N.Y.