Theoretical Computer Science
Elsevier

COMPLEXITY IN LEFT-ASSOCIATIVE GRAMMAR

ROLAND HAUSSER
Friedrich-Alexander Universität
D-8520 Erlangen, Bismarckstr. 12, Germany
email: rrh@linguistik.uni-erlangen.de

Communicated by M.A. Harrison

This paper presents a mathematical definition of Left-Associative Grammar, and describes its formal properties.[1] Conceptually, LA-grammar is based on the notion of possible *continuations*, in contrast to more traditional systems such as Phrase Structure Grammar and Categorial Grammar, which are linguistically motivated in terms of possible *substitutions*. It is shown that LA-grammar generates all and only the recursive languages. The Chomsky hierarchy of regular, context-free, and context-sensitive languages is reconstructed in LA-grammar by simulating finite state automata, push-down automata, and linearly bounded automata, respectively. Using alternative restrictions on LA-grammars, the new language hierarchy of A-LAGs, B-LAGs, C-LAGs is proposed.

The class of C-LAGs is divided into three subclasses representing different degrees of ambiguity and associated computational complexity. The class of C-LAGs without recursive ambiguities (called the C1-LAGs) parses in linear time, and includes all deterministic CF-languages, plus CF-languages with non-recursive ambiguities, e.g., $a^n b^n c^m d^m \cup a^n b^m c^m d^n$, plus many context-sensitive languages, such as $a^n b^n c^n$, $a^n b^n c^n d^n e^n$, $\{a^n b^n c^n\}^*$, a^{2^i}, $a^k b^m c^{k \cdot m}$ and $a^{i!}$. The class of C-LAGs with recursive "single return" ambiguities (called C2-LAGs) parses in n^2, and includes certain non-deterministic CF-languages such as WW^R, plus context-sensitive languages like WW, WWW, WWWWW, and $\{\mathrm{WWW}\}^*$. Finally, the class of unrestricted C-LAGs (called C3-LAGs) parses in exponential time and contains CF-languages like L_{no} and the "hardest context-free language" HCFL, plus context-sensitive languages like $\mathcal{NP}$-complete Subset Sum and SAT.

[1] Some of the results reported in this paper are published in [12]. Discussions with David Applegate, Gary Miller, Daniel Sleator, and Shanghua Teng at Carnegie Mellon University in October of 1989 resulted in a revised complexity result for ambiguous C-LAGs. I am especially indebted to David Applegate, who provided the crucial grammars for L_{no}, $a^{i!}$, Subset Sum, and SAT.

1. Formal Rule Schemata of Generative Grammar

Left-associative grammar (LA-grammar) is a comparatively new formalism. By way of introduction, let us compare it with more widely known systems, namely phrase structure grammar (PS-grammar) and categorial grammar (C-grammar).

The formalism of PS-grammar is based on the rewriting system of Post (1936). Rewriting rules have the following form:

(1.1) The Schema of Phrase-Structure Rewriting Rules

$$A \rightarrow B\ C$$

By replacing (rewriting) the symbol A with B and C, this rule generates a tree structure with A dominating B and C. Conceptually, the derivation order of rewriting rules is top-down.

The formalism of C-grammar is based on the categorial-canceling rules of Leśnieswki (1929) and Ajdukiewicz (1935). Categorial-canceling rules have the following form:

(1.2) The Schema of Categorial Canceling Rules

$$\alpha_{(Y|X)} \bullet \beta_{(Y)} \Rightarrow \alpha\beta_{(X)}$$

This rule schema combines α and β into $\alpha\beta$ by canceling the Y in the category of α with the corresponding category of β. The result is a tree structure with $\alpha\beta$ of category X dominating α and β. Conceptually, the derivation order of categorial-canceling rules is bottom-up.

Finally consider the rule schema of LA-grammar.

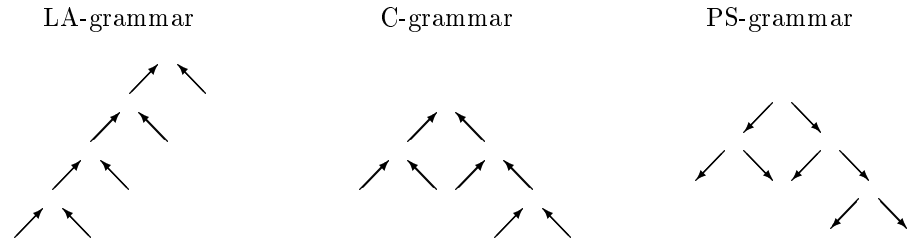
(1.3) The Schema of Left-Associative Concatenation Rules

$$r_i\text{: [CAT-1 CAT-2]} \Rightarrow [rp_i\ \text{CAT-3}]$$

A left-associative rule r_i maps a sentence start (represented by its category CAT-1) and a next word (represented by its category CAT-2) into the rule package rp_i and a new sentence start (represented by its category CAT-3). A *state* in LA-grammar is defined as an ordered pair, consisting of a rule package and a sentence start. In the next composition, the rules in the rule package are applied to the sentence start resulting from the last composition and a new next word.

The three different rule schemata represent three different conceptual derivation orders:

(1.4) Three Grammatical Derivation Orders



LA-grammars are input-output equivalent to their parsers and generators in that (i) they take the same input (i.e., an unanalyzed surface string), (ii) generate the same output (a left-associative syntactic analysis), and (iii) use the same rules in the same derivation order. In other words, LA-grammar achieves "absolute type transparency" [Berwick & Weinberg (1984), p. 41] because it is strongly input-output equivalent to its parsers and generators.

PS-grammar and C-grammar, on the other hand, are unsuitable for direct parsing. Parsers for context-free PS-grammars, for example, cannot possibly apply the rules of the grammar directly because the rules rewrite an initial start symbol, while the parser takes sentences as input. The standard solution to this dilemma consists in computational routines which reconstruct the grammatical analysis in an indirect way by building large intermediate structures (e.g., "state sets", "charts", "tables") which are not part of the grammar.

2. Syntax and Semantics

The surface structures of natural language are linear. When we utter or understand a sentence, we process it word by word, starting at the beginning. This fundamental structural property is formalized in LA-grammar, which computes possible continuations. The left-assocative derivation order of LA-grammar reflects the time-linear nature of language.

The linear syntactic derivations in LA-grammar may be extended to generate homomorphic semantic hierarchies, defined as minimal databases which are suitable for pragmatic interpretation. As an example of a left-associative parse in natural language consider (2.1).

(2.1) A Sample Derivation

```
NEWCAT> (z Fido found a bone.)
Elapsed real time = 779 milliseconds
User cpu time = 660 milliseconds
System cpu time = 20 milliseconds
Total cpu time = 680 milliseconds
    Linear Analysis:

    *START_0
    1
       (NA) FIDO
       (N SC V) FOUND
    *NOM+FVERB_3
    2
       (SC V) FIDO FOUND
       (SQ) A
    *FVERB+MAIN_4
    3
       (SQ V) FIDO FOUND A
       (SN) BONE
    *DET+NOUN_2
    4
       (V) FIDO FOUND A BONE
       (V DECL) .
    *CMPLT_13
```

```
    5
       (DECL) FIDO FOUND A BONE .


    Hierarchical Analysis:
(PROPOSITION-5_6_13
        (MOOD (DECLARATIVE-5_6_13))
        (PROP-CONTENT
          ((SENT-2_6_13 (SUBJ ((NP-1_6_13 (NAME (FIDO-1_6_13)))))
                        (VERB (FIND-2_6_13))
                        (DIR-OBJ
                        ((NP-3_6_13 (REF (A-3_6_13 SG-4_6_13))
                                    (NOUN ((BONE-4_6_13)))))))))


        PROPOSITION
             |
     -----------------
     |               |
   MOOD        PROP-CONTENT
     |               |
     |               |
     |               |
DECLARATIVE        SENT
                     |
                --------------
                |    |       |
            SUBJ VERB   DIR-OBJ
              |    |       |
              |    |       |
              |    |       |
             NP  FIND     NP
              |             |
              |         ------
              |         |    |
            NAME       REF NOUN
              |         |    |
              |       ----   |
              |       |  |   |
            FIDO      A SG BONE
```

The linear syntactic analysis in 2.1 is followed by a semantic hierarchy which is constructed incrementally and simultaneously with the syntactic derivation. The semantic hierarchy expresses many of the intuitions central to constituent structure.[2] It is displayed as a structured list and (optionally) as the equivalent tree structure. The following discussion of LA-grammar is limited to the formal properties of the linear syntax.

3. The Mathematical Definition

Let us recall some notation from set theory needed in the following definition. 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 × 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., $n = \{i \mid 0 \leq i < n\}$.

(3.1) Formal Definition of Left-Associative Grammar

An LA-grammar is defined as a 7-tuple <W, C, LX, CO, RP, ST_S, ST_F >, where

1. W is a finite set of *word surfaces*.
2. C is a finite set of *category segments*.
3. LX ⊂ (W × C^+) is a finite set comprising the *lexicon*.
4. CO = (co_0 ... co_{n-1}) is a finite sequence of total recursive functions from (C^* × C^+) into $C^* \cup \{\perp\}$, called *categorial operations*.
5. RP = (rp_0 ... rp_{n-1}) is an equally long sequence of subsets of n called *rule packages*.
6. ST_S = {(rp_s cat_s), ...} is a finite set of *initial states*, where each rp_s is a subset of n called a start rule package and each cat_s ε C^+.
7. ST_F = {(rp_f cat_f), ...} is a finite set of *final states*, where each rp_f ε RP and each cat_f ε C^*.

For theoretical reasons, the categorial operations are defined as total functions. In practice, the categorial operations are defined on easily-recognizable subsets of (C^* × 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 specified by (i) a lexicon LX, (ii) a set of start states ST_S, (iii) a sequence of rules, each defined as an ordered pair (co_i rp_i), and (iv) a set of final states ST_F. A left-associative rule r_i takes a sentence start ss and a

[2]Constituent structures are a special kind of semantic hierarchy. They are based methodologically on *substitution* and *movement tests* which are intended to reveal which parts of the sentence belong most closely together. The rules of PS-grammar and C-grammar, which generate constituent structures, impose a partial derivation order which reflects the structure of the hierarchy, and not the time-linear nature of language. LA-grammar differs from constituent structure grammars in that it generates semantic hierarchies in a time-linear order.

next word nw as input, and applies the categorial operation co_i to the sentence category cat-1 and the next word category cat-2.

If the input condition of the categorial operation is satisfied by (cat-1 cat-2), the application of r_i is successful and an output is derived. The output consists of the pair [rp_i ss'], where rp_i is a *rule package* and ss' is a *resulting sentence start*. If the input condition of the categorial operation is not satisfied by (cat-1 cat-2), the application of rule r_i is not successful and no output is derived.

The rule package rp_i provided by the rule r_i contains all rules which may apply after rule r_i was successful. A rule package is defined as a set of rule names, where the name of a rule r_g is the place number g of its categorial operation co_g in the sequence CO.

The general format of LA-grammars is illustrated below with the context-sensitive language $a^n b^n c^n$.

(3.2) The Definition of $a^n b^n c^n$

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

The lexicon LX defines three words, 'a', 'b', and 'c'. Each word is an ordered pair consisting of a surface, e.g., 'a', and a category defined as a list of category segments, e.g., (bc). The start state in ST_S specifies that the first word must be of category (bc), i.e., an 'a'. Furthermore, the only rules to be tried at the beginning are r-1 and r-2. The rules specify the categorial operations in terms of the sequence variable X, standing for zero or more category segments, and the category segments b and c. Rule r-1 takes a sentence start of any category (represented by '(X)'), and a next word of category (bc)—that is, an 'a'—and derives the output category by adding a b to the beginning and a c to the end of the sentence start category. Rule r-2 takes a sentence start category which starts with an b and ends with a c, a next word of category (b)—that is, a 'b'—and derives an output category by deleting the first segment. And similarly for r-3. Note that the three rules of $a^n b^n c^n$ have *incompatible* input conditions (they all take different next words). Thus the grammar is an example of an *unambiguous* C-LAG and parses in linear time.[3]

In LA-grammar the parsing of a language consists in applying the grammar rules directly to the input string.[4] Thus, the declarative linguistic analysis and the computation are merely different aspects of the same left-associative structure. The grammatical analysis provided by LA-parsers and LA-generators is simply a trace of the computation.

(3.3) Parsing aaabbbccc with Active Rule Counter

```
NEWCAT> (z a a a b b b c c c)
;  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
```

[3] See Section 8 for further discussion.

[4] In contrast to PS-grammar which requires the construction of state sets, charts, or tables to mediate between the parser input, i.e., a string of words, and the grammar rules.

```
   (B B C C) A A
   (B C) A
*RULE-1
3
   (B B B C C C) A A A
   (B) B
*RULE-2
4
   (B B C C C) A A A B
   (B) B
*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
```

Each left-associative composition is characterized by a word number (e.g., 4), a sentence start consisting of a category[5] and a surface (e.g., (B B C C) A A A B), a next word (e.g., (B) B), and a rule (e.g., *RULE-2). The result of the composition is shown in the first line of the next "history section" (e.g., (B C C C) A A A B B).

LA-grammar is as suitable for direct generation as it is for direct parsing. The only difference is that in parsing, the next word is provided by the input string, while in generation the next word is chosen from the lexicon. As an illustration of the relation between an LA-grammar and its generator, consider the following derivation of well-formed expressions up to length 12 using the grammar for $a^n b^n c^n$ defined in (3.2).

(3.4) Generating the Representative Sample of $a^n b^n c^n$

```
NEWCAT> (gram-gen 3 '(a b c))

Parses of length 2:
 A B
   2   (C)
 A A
   1   (B B C C)

Parses of length 3:
 A B C
   2 3   (NIL)
 A A B
```

[5]Note that categories precede the surfaces in (3.3). In the current implementation, the parse is called with the function "(z ...)".

```
   1 2   (B C C)
 A A A
   1 1   (B B B C C C)

Parses of length 4:
 A A B B
   1 2 2   (C C)
 A A A B
   1 1 2   (B B C C C)
 A A A A
   1 1 1   (B B B B C C C C)

Parses of length 5:
 A A B B C
   1 2 2 3   (C)
 A A A B B
   1 1 2 2   (B C C C)
 A A A A B
   1 1 1 2   (B B B C C C C)

Parses of length 6:
 A A B B C C
   1 2 2 3 3   (NIL)
 A A A B B B
   1 1 2 2 2   (C C C)
 A A A A B B
   1 1 1 2 2   (B B C C C C)

Parses of length 7:
 A A A B B B C
   1 1 2 2 2 3   (C C)
 A A A A B B B
   1 1 1 2 2 2   (B C C C C)

Parses of length 8:
 A A A B B B C C
   1 1 2 2 2 3 3   (C)
 A A A A B B B B
   1 1 1 2 2 2 2   (C C C C)

Parses of length 9:
 A A A B B B C C C
   1 1 2 2 2 3 3 3   (NIL)
 A A A A B B B B C
   1 1 1 2 2 2 2 3   (C C C)

Parses of length 10:
 A A A A B B B B C C
   1 1 1 2 2 2 2 3 3   (C C)

Parses of length 11:
 A A A A B B B B C C C
   1 1 1 2 2 2 2 3 3 3   (C)

Parses of length 12:
 A A A A B B B B C C C C
   1 1 1 2 2 2 2 3 3 3 3   (NIL)
```

After loading the same grammar as is used for parsing, the function 'gram-gen' is called with two arguments: the "recursion factor" of the grammar [Hausser (1989), pp. 193 ff], and a list of the words to be used.[6] The output is a

[6] In another version, 'gram-gen' is called with the maximal surface length rather than the

systematic generation, starting with well-formed expressions of length 2. Each derivation consists of a surface, a sequence of rules, and a result category. As an example of a single derivation, consider (3.5):

(3.5) A Complete Well-Formed Expression in $a^n b^n c^n$

```
A A A B B C C C
  1 1 2 2 2 3 3 3   (NIL)
```

In (3.5) the surface and the rule sequence are lined up so that it is apparent which word was added by which rule. Derivation (3.5) characterizes a complete well-formed expression because it represents the rule state (rp-3 ϵ), which is an element of the set of complete well-formed expressions of the LA-grammar for $a^n b^n c^n$ defined in (3.2).

4. The Hierarchy of A-LAGs, B-LAGs, and C-Lags

Intuitively, LA-grammar may be viewed as a mathematical generalization of (one-way) pushdown automata. Consider a PDA move δ, defined as a function from

$$Q \times (\sum \cup \epsilon) \times , \text{ into } Q \times , ^*$$

where Q is the set of states, $\sum$ is the surface alphabet of the language, and , is the stack alphabet. Corresponding to the mapping from Q into Q, a rule in LA-grammar has the specification of a rule package rp-i for each rule r-i:

$$r_i\text{: [CAT-1 CAT-2]} \Rightarrow [rp_i \text{ CAT-3}]$$

And corresponding to the mapping from $(\sum \cup \epsilon) \times ,$ into , *, a rule in LA-grammar has a categorial operation, defined as a mapping from (CAT-1 CAT-2) into CAT-3. It follows that for any PDA there exists an equivalent LA-grammar.

The converse does not hold, however, because LA-grammars are much more general than PDAs. LA-grammar distinguishes systematically between the surface and the category of expressions. Thus a single alphabet is used to specify both the sentence start category (corresponding to the stack ,) and the next word category (corresponding to the surface alphabet $\sum$). Furthermore, the categorial operations of LA-grammar are defined as total recursive functions from $(C^* \times C^+)$ into $C^* \cup \{\perp\}$.

It follows that LA-grammar recognizes and generates **all** recursive languages [Hausser (1989), Theorem 2, p. 135]. Furthermore, due to the linear structure of the derivation LA-grammar generates **only** the recursive languages [Hausser (1989), Theorem 1, p. 134]. Thus all possible analyses are derived in finitely many steps for any given finite input, and there is no halting problem in LA-grammar and associated parsers.

Given the powerful categorial operations of LA-grammar, we are interested in languages where the categorial operations may be specified in terms of simple

recursion factor.

patterns. Consider the following three rule schemas, whereby CAT-3 contains at most one sequence variable, e.g., X.

r_i: [(seg-1...seg-k X) CAT-2] $\Rightarrow$ [rp_i CAT-3]

r_i: [(X seg-1...seg-k) CAT-2] $\Rightarrow$ [rp_i CAT-3]

r_i: [(seg-1...seg-i X seg-i+1...seg-k) CAT-2] $\Rightarrow$ [rp_i CAT-3]

These schemas have in common that the pattern matching has to check exactly k segments in the sentence start category. LA-grammars conforming to this simple type of pattern matching are called constant LA-grammars or C-LAGs.

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

r_i: [(X seg-1...seg-k Y) CAT-2] $\Rightarrow$ [rp_i CAT-3]

the pattern matching has to search through an indefinite number of category segments (represented by X or Y, depending on where the search begins). In such non-constant LA-grammars CAT-3 may contain more than one sequence variable (e.g., X and Y).

(4.1) 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.

The C-languages contain many context-sensitive languages (e.g., 3.2) and all context-free languages. Context-free languages in LA-grammar are characterized by categorial operations which may only look at a finite number of segments at the beginning of the sentence-start category [Hausser (1989), Theorem 4, p. 138]. The regular languages, furthermore, are generated by C-LAGs where the length of the sentence start category is restricted by a finite constant [Hausser (1989), Theorem 4, p. 138, and Section 8.2].

Because of their restricted categorial operations, C-LAGs resemble PDAs most closely. This similarity between C-LAGs and PDAs explains why C-LAGs are much easier to write than corresponding PS-grammars.

> Whenever one is faced with some new context-free set, it is generally much easier to describe a pushdown automaton to accept it than to try to produce a grammar.
>
> M.A. Harrison (1978), p. 135.

C-LAGs differ from PDAs, however, in that they are more general, accepting a much larger class of languages. Furthermore, C-LAGs provide a simpler, more transparent, mathematical notation. The use of *pattern matching* in specifying the categorial operations of C-LAGs constitutes a major contrast to PS-grammars, which use the *rewriting* of variables (non-terminals) instead. PDAs

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

make very limited use of pattern matching by looking only at the first element of the stack.

LA-grammars which are not C-LAGs are called non-constant LA-grammars. In non-constant LA-grammars a categorial operation is viewed as a sequence of Turing machine moves, using the category as the tape. Non-constant LA-grammars are divided into the *B-LAGs* and *A-LAGs*.

(4.2) 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 $C \cdot n$, 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.

The class of languages generated by B-LAGs is the class of the context-sensitive languages. The proof [Hausser (1989), Theorem 5, p. 142] is based on the corresponding restrictions on linearly bounded automata.

(4.3) Definition of the Class of A-LAGs

> The class of A-LAGs consists of *all* LA-grammars because there is no limit on the length of the categories, or on 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:

(4.4) The Hierarchy of A-LAGs, B-LAGs, and C-LAGs

> 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 many context-sensitive languages, all context-free languages, and all regular languages.

The remainder of this paper will deal mostly with the formal properties of C-LAGs.

5. Complexity and Ambiguity

Because the categorial operations of C-LAGs look at no more than k sentence-start category segments, for some constant k, the application of a rule may be taken as the "primitive operation" for purposes of complexity analysis. Thus, the complexity of a C-LAG corresponds to the maximal number of rule applications per input. For example, a C-LAG parses in linear time if the maximal number of rule applications per input of length n is $C \cdot n$, for some constant C. A C-LAG parses in square time if the maximal number of rule applications per input of length n is $C \cdot n^2$, etc.

The class of C-LAGs is divided into three subclasses according to (i) types of ambiguity, and (ii) corresponding degrees of computational efficiency.[8] These subclasses are the C1-LAGs, which parse in n (linear time), the C2-LAGs, which parse in n^2 (square time), and the C3-LAGs, which parse in 2^n (exponential time).

In order to describe the subclassses of C-LAGs in more detail, we must first explain the different types of ambiguity in LA-grammar. There are (i) unambiguous LA-grammars, (ii) syntactically-ambiguous LA-grammars, and (iii) lexically-ambiguous LA-grammars. Syntactic ambiguity is defined in terms of the *input-compatibility* of rules.

(5.1) Three Types of Input Conditions

1. **Incompatible** input conditions: Two rules have incompatible input conditions if there exist no input pairs which are accepted by both rules.
2. **Compatible** input conditions: Two rules have compatible input conditions if there exists at least one input pair accepted by both rules, and there exists at least one input pair accepted by one rule, but not the other.
3. **Identical** input conditions: Two rules have identical input conditions if it holds for all input pairs that they are either accepted by both rules, or rejected by both rules.

(5.2) Definition of Unambiguous LA-Grammars

> An LA-grammar is unambiguous if and only if (i) it holds for all rule packages that their rules have *incompatible* input conditions, and (ii) there are no lexical ambiguities.

Examples of incompatible input conditions are [(a X)(b)] and [(c X)(b)], as well as [(a X)(b)] and [(a X)(c)]. Since unambiguous LA-grammars permit–in any given state–at most one continuation, they represent the class of deterministic LA-grammars. Examples of unambiguous C-LAGs are 3.2 for $a^n b^n c^n$ and 6.1 for a^{2^i}. Unambiguous C-LAGs parse in linear time.

(5.3) Definition of Syntactically-Ambiguous LA-Grammars

> An LA-grammar is syntactically ambiguous if and only if (i) it has at least one rule package containing at least two rules with *compatible* input conditions, and (ii) there are no lexical ambiguities.

For example, [(a X)(b)] and [(X a)(b)] represent compatible input conditions.

The computational complexity of a syntactically ambiguous LA-grammar, and the number of possible readings in the associated language, depend on

[8] A similar situation with regard to ambiguity arises also in PS-grammar. For example, the Earley algorithm (Earley 1970) recognizes **unambiguous** context-free PS-grammars in $|G|^2 \cdot n^2$, but **ambiguous** context-free PS-grammars in $|G|^2 \cdot n^3$ (where $|G|$ is the size of the grammar and n the length of the input string).

four types of syntactic ambiguity, based on the binary features $\pm$*local* and $\pm$*recursive*. In a local ambiguity only one branch (resulting from an ambiguity split at a certain state) reaches a final state, whereas in a non-local ambiguity two or more branches reach a final state (for some given input).

An ambiguity is called recursive if it arises inside a recursive loop. That is, a certain state (rp-i, CAT) has more than one continuation (for a given next word), and one or more of these continuations lead back into this state at a later point in the derivation, allowing for a repetition of the ambiguity split.

An ambiguity is called non-recursive, on the other hand, if none of the branches resulting from the ambiguity can ever feed back into the state which caused the ambiguous continuation. C-LAGs with non-recursive ambiguities parse in linear time, just like unambiguous C-LAGs. An example of an LA-grammar with a non-local, non-recursive ambiguity is 6.2 for $\mathrm{a}^n\mathrm{b}^n\mathrm{c}^m\mathrm{d}^m \cup \mathrm{a}^n\mathrm{b}^m\mathrm{c}^m\mathrm{d}^n$.

C-LAGs with recursive ambiguities can be of exponential complexity: there may be, e.g., a doubling of readings each time the derivation returns to the ambiguous state (examples are the C-LAGs 8.4 for L_{no} and 8.5 for Subset Sum). However, if only a single branch of a recursive ambiguity split may return to the ambiguous state, then the complexity of the grammar is n^2. An example of a C-LAG with a recursive "single return" ambiguity is 7.1 for WW^R.[9]

[9]By definition 5.3, LA-grammars like 7.1 for WW^R are ambiguous even though all complete expressions correspond to unique final states.

In other words, if the LA-grammar for a language does not produce ambiguously analyzed sentence starts, then the grammar is deterministic in the sense that for any state there is at most one continuation–because all rule packages contain only incompatible rules. Conversely, if the LA-grammar for a language produces ambiguous sentence starts, then the grammar is non-deterministic in the sense that there are states with more than one possible continuation for a given next word. Thus the notions 'unambiguous' and 'deterministic,' as well as 'ambiguous' and 'non-deterministic,' coincide in LA-grammar.

(5.4) Definition of Lexically-Ambiguous LA-Grammars

> An LA-grammar is lexically ambiguous if its lexicon contains at least two analyzed words with identical surfaces.

A non-linguistic example of lexical ambiguity is propositional calculus, e.g., (x v y v z) & (..., where the propositional variable x is analyzed as (x (1)) and (x (0)), y is analyzed as (y (1)) and (y (0)), etc.

6. The C1-LAGs: Parsing in Linear Time

The most efficient type of constant LA-grammars are the C1-LAGs, which parse in linear time. A grammar G is a C1-LAG if G is a C-LAG that does not generate any recursive ambiguities. The class of C1-LAGs analyzes all deterministic context-free languages[10] and all context-free languages with non-recursive ambiguities. In addition the C1-LAGs analyze context-sensitive languages which are unambiguous or non-recursively ambiguous.

Examples of unambiguous context-sensitive C-LAGs are $a^n b^n c^n$ defined in 3.2 and $a^{2^i} =_{def}$ {a^i | i is a positive power of 2} defined in 6.1.

(6.1) The Unambiguous C1-LAG for Context-Sensitive a^{2^i}

LX $=_{def}$ {[a (a)]}
ST$_S$ $=_{def}$ {({r-1} (a))}
r-1: [(a) (a)] ⇒ [{r-2 } (aa)],
r-2: [(aX) (a)] ⇒ [{r-2, r-3} (Xbb)],
r-3: [(bX) (a)] ⇒ [{r-2, r-3} (Xaa)]
ST$_F$ $=_{def}$ {[rp-1 (aa)], [rp-2 (bXb], [rp-3 (aXa)]}.

6.1 is a C1-LAG because r-2 and r-3 have incompatible input conditions. A comparison of 6.1 with corresponding phrase structure grammars [Hopcroft & Ullman (1979)][11] for a^{2^i} illustrates the formal and conceptual simplicity of LA-grammar.

A C1-LAG with a non-recursive ambiguity is given in 6.2 for the language a^{n}b^{n}c^{m}d^m ∪ a^{n}b^{m}c^{m}d^n. This language has been called an *inherently ambiguous language* [cf. Hopcroft and Ullman (1979), pp. 99 – 103] because there exist no unambiguous PS-grammars for it.

(6.2) The Ambiguous C1-LAG for Context-Free $\mathbf{a}^n\mathbf{b}^n\mathbf{c}^m\mathbf{d}^m \cup \mathbf{a}^n\mathbf{b}^m\mathbf{c}^m\mathbf{d}^n$

LX $=_{def}$ {(a (a)), (b (b)), (c (c)), (d (d))}
ST$_S$ $=_{def}$ {[{r-1, r-2, r-5} (a)]}
r-1: [(X) (a)] ⇒ [{r-1, r-2, r-5} (a X)]
r-2: [(a X) (b)] ⇒ [{r-2, r-3} (X)]
r-3: [(X) (c)] ⇒ [{r-3, r-4} (c X)]

[10] Deterministic CF-languages are defined in terms of deterministic PDAs [Harrison 1978, p. 129].

[11] See op. cit., p. 224 for the canonical context-sensitive PS-grammar of a^{2^i}, and p. 220 for a version as an unrestricted PS-grammar.

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

The grammar defined in 6.2 exhibits a syntactic ambiguity in the sense of definition 5.4: the rule package rp-1 calls the input-compatible rules r-2 and r-5, and the rule package rp-3 calls the input-compatible rules r-3 and r-4. Nevertheless, the grammar parses in linear time because the ambiguity splits are not part of a recursion: rp-2 and rp-5 do not call r-1, and rp-4 does not call r-3. In the worst case, the grammar generates two analyses, based on two parallel linear branches [Hausser (1989), pp. 154 ff].

The class of C1-LAGs cuts across other language classes which have been proposed as extensions of the context-free languages. These are the tree adjoining languages, or TALs, accepted by tree adjoining grammars, or TAGs [cf. Joshi (1987)], and the indexed languages [cf. Hopcroft and Ullman (1979), pp. 389ff]. The C1-LAGs accept TALs such as $a^n b^n c^n$, indexed languages which are not TALs such as $a^n b^n c^n d^n e^n$, $\{a^n b^n c^n\}^*$, $a^k b^m c^{k \cdot m}$,[12] and a^{2^i}, as well as $a^{i!}$ which is not an indexed language [Hayashi (1973)].[13]

We may regard the C1-LAGs as an extension of the deterministic CF-grammars (or LR(k)-grammars),[14] providing linear parsing for a much larger

[12] Hausser 1989 presented $a^k b^m c^{k \cdot m}$ as a B-LAG. A C1-LAG for context-sensitive $a^k b^m c^{k \cdot m}$ is presented below:

LX $=_{def}$ {(a (a)), (b (b)), (c (c))}
$ST_S =_{def}$ {[{r-1} (a)]}
r-1:[(X)(a)] ⇒ [{r-1, r-2}(Xa)]
r-2:[(X)(b)] ⇒ [{r-2, r-3}(Xb)]
r-3:[(aaXb)(c)] ⇒ [{r-4}(baX)]
r-4:[(Xb)(c)] ⇒ [{r-4, r-5, r-7} (bX)]
r-5:[(bXaa)(c)] ⇒ [{r-6}(Xab)]
r-6:[(bX)(c)] ⇒ [{r-6, r-3, r-9}(Xb)]
r-7:[(Xba)(c)] ⇒ [{r-9}(X)]
r-8:[(abX)(c)] ⇒ [{r-9}(X)]
r-9:[(bX)(c)] ⇒ [{r-9}(X)]
$ST_F =_{def}$ {[rp-9 ϵ]}

[13] Applegate has constructed a C1-LAG for $a^{i!}$ as follows:
"The category after reading $a^{x!}$ consists of:
1. #
2. x written as an x bit binary number
3. #
4. x! written as an x^2 bit binary number
5. #
Then, while reading the next (x+1)!-x! a's, we adjust the category to be correct after reading $a^{(x+1)!}$. The basic idea is that we expand x and x! to be x+1 and $(x+1)^2$ bit numbers, increment x, and multiply x! by the result to get (x+1)!. Then, we compute how many extra a's are needed to get from where we are to $a^{(x+1)!}$, and then match those a's."

[14] Any deterministic CF-language has an LR(k)-grammar. Every LR(k) language is a de-

class of languages in a simple, unified framework.

7. The C2-LAGs: Parsing in n^2

The second most efficient type of C-LAGs are the C2-LAGs, which parse in n^2. A grammar G is a C2-LAG if G generates recursive ambiguities which are restricted by the "Single Return Principle" (SRP):

> A syntactic ambiguity arising inside a recursion constitutes a "single return" if exactly one of the branches resulting from the ambiguity may feed back into the recursion.

An LA-grammar satisfies the SRP if all its recursive ambiguities are "single return." As a consequence of the SRP, C2-LAGs have—at most—$(C \cdot n)$ readings.[15] Consequently, any C2-LAG can be parsed in n^2.

As a case in point consider the non-deterministic context-free language WW^R, defined in 7.1.

(7.1) The Nondeterministic Context-Free Language WW^R

LX $=_{def}$ {(a (a)), (b (b)), (c (c)), (d (d)), ...}
ST_S $=_{def}$ {[{r-1, r-2}(seg1)]}
r-1: [(X)(seg1)] ⇒ [{r-1, r-2}(seg1 X)]
r-2: [(seg1 X)(seg1)] ⇒ [{r-2}(X)]
ST_F $=_{def}$ {[rp-2 ϵ]}

This language consists of an arbitrarily long sequence of arbitrary letters (called W) followed by an inverse sequence. The worst case for parsing WW^R is input strings consisting of even numbers of the same letter. Consider 7.2:

(7.2) The Derivational Structure of the Worst Case in WW^R (for Input *aaaaaa*)

```
rules:          analyses:

  2             a $ a
  1 2 2         a a $ a a
  1 1 2 2 2     a a a $ a a a
  1 1 1 2 2     a a a a $ a a
  1 1 1 1 2     a a a a a $ a
  1 1 1 1 1     a a a a a a $
```

7.2 exhibits a recursive ambiguity: each time r-1 has applied, the system tries both r-1 and r-2 at the next composition. But once a derivation has entered r-2 it cannot go back to r-1. Thus, only one branch of the ambiguity (represented by the application of r-1) can reenter the recursion. In other words, the LA-grammar 7.1 satisfies the SRP.

terministic context-free language. Cf. Harrison (1978), p. 501, and 554.

[15] C is some finite, grammar dependent constant reflecting the number of rules introducing recursive ambiguities and n is the length of the input.

An example of a C2-LAG for a context-sensitive language is the grammar for WW. This grammar is just like 7.1, except for the output category of r-1: in context-sensitive WW, r-1 produces (X seg1), while in context-free WW^R, r-1 produces (seg1 X). Furthermore, the C2-LAGs for context-free WW^R and context-sensitive WW use exactly the same number of rules for corresponding input (i.e., they have the same complexity).

In summary, the class of C2-LAGs, defined in terms of recursive "single return" ambiguities, parses a subset of the nondeterministic context-free languages in n^2, e.g., WW^R. In addition, the C2-LAGs include context-sensitive languages like WW (a TAL), plus WWW, WWWWW, and {WWW}*, which are Index Languages but not TALs.

8. The C3-LAGs: Parsing in Exponential Time

The C3-LAGs contain C-LAGs with unrestricted recursive ambiguities and parse in 2^n (exponential time). An example of a non-deterministic context-free language accepted by a C3-LAG is the language L_{no} (or "noise"-language). L_{no} has a simple context-free PS-grammar.

8.1 PS-Definition of L_{no}

$S \rightarrow 1S1$
$S \rightarrow 0S0$
$S \rightarrow 1S$
$S \rightarrow 0S$
$S \rightarrow \#$

L_{no} generates expressions with the structure W'#W^R. The symbol # separates W' and W^R. W^R is the inverse of W, and W' differs from W in that it may contain an arbitrary number of additional 0's and 1's. These additional symbols are interspersed with and indistinguishable from those which have a counterpart in W^R. In other words, the additional symbols in W' function as noise.

Traditional parsers based on context-free PS-grammars like Earley and CYK can handle L_{no} in n^3 because the PS-grammar rules reflect the basic structure of CF-languages, which is "inverse pairs," e.g., 'abc...cba'.[16] Example 8.2 presents a sample derivation within the PS-grammar 8.1 as well as the corresponding states created by the Earley algorithm.

8.2 A PS-Grammar Derivation of 10010#101 in L_{no}

```
derivation tree:       derived strings:       states:

        S
      / | \                                   1.S1     1S1.
    1   S  1              1S1                 1.S
      / | \                                   0.S0     0S0.
```

[16] Any context-free language is homomorphic with the intersection of a regular set and a semi Dyck set (Chomsky-Schützenberger theorem). Cf. Harrison (1978), pp. 317ff.

```
0  S  0            10S01              0.S
 / |                                   0.S0
0  S               100S01             0.S     0S.
 / | \                                 1.S1    1S1.
1  S  1           1001S101           1.S
 / |                                   0.S0
0  S              10010S101          0.S     0S.
   |
   #              10010#101          #.
```

In L_{no} an Earley parser creates only two states for each terminal preceding #, e.g., '1.S1' and '1.S'. Thus, if # is preceded by n terminals, the number of states is $2n$ upon reaching #.

C-LAGs, on the other hand, use the different structure of a double ended queue. This structure is well suited for repetitions of any number, whereby the repetitions may be modified, e.g., inverted, doubled, halved, etc. C-LAGs are highly efficient for deterministic languages of any kind (not just context-free ones). They are also efficient for languages that have nonrecursive ambiguities and languages that have recursive "single return" ambiguities.

Parsers in general, and C-LAGs in particular, are computationally highly complex, however, if the string to be repeated contains an unspecified number of symbols with alternative interpretations such that these interpretations are relevant in later repetitions. This is the characteristic property of $\mathcal{NP}$-hard languages, i.e., languages which take $\mathcal{N}$ondeterministic $\mathcal{P}$olynomial time for verification, and exponential time for recognition.

An example of an $\mathcal{NP}$-hard language is 3SAT, an instance of Boolean satisfiability [cf. Hopcroft & Ullman (1979), pp. 324ff]. Consider an arbitrary Boolean formula like 8.3:

8.3 A Formula in 3SAT

(x v $\bar{y}$ v $\bar{z}$) & (y v z v u) & (x v z v $\bar{u}$) & ($\bar{x}$ v y v u)

The sign 'v' stands for logical *or* (disjunction), the sign '&' stands for logical *and* (conjunction), and the bar above some of the variables, e.g., $\bar{z}$, stands for negation. 3SAT is a specialized Boolean language, restricted to conjunctions such that each conjunct consists of a disjunction with three disjuncts.

The problem of *satisfying* formulas like 8.3 consists in finding a value assignment which makes the formula true (if such an assignment exists). This problem is computationally complex because the analysis has to remember potentially 2^n many variable assignments. For example, at the first variable x, two hypotheses must be pursued, namely that x is 1 (true) and that x is 0 (false). At the next variable y, the analysis has to remember four possible value assignments, namely (x=1 y=0), (x=1 y=1), (x=0 y=1), and (x=0 y=0). Thus, each time a new variable is encountered, the number of possible assignments is doubled.

The point is that in LA-grammar nondeterministic context-free L_{no} and

$\mathcal{NP}$-complete 3SAT are alike. Because C-LAGs are not based on the "inverse pair" structure of context-free languages, the C-LAG complexity of L_{no} is the same as that of, e.g., context-sensitive L^3_{no}, defined as W'#W#W" (where W' and W" are noisy versions of W).

The only way a C-LAG can handle L_{no} is by giving two interpretations to each symbol preceding #, one as a genuine symbol and one as a noise symbol. This results in an exponential number of readings (for the input preceding #), each characterized by a different category. Assuming input 10010#..., for example, one reading has the category (10010), representing the hypothesis that the pre-# symbols are all 'genuine'. Another reading has the category (1001), representing the hypothesis that the last pre-# symbol is a noise symbol, etc.

The syntactically ambiguous C3-LAG in 8.4 generates these different hypotheses systematically from left to right.

8.4 Syntactically Ambiguous C3-LAG for L_{no}

LX $=_{def}$ {(0(0)) (1(1)) (#(#))}
ST_S $=_{def}$ {[{r-1, r-2, r-3, r-4}(0)] [{r-1, r-2, r-3, r-4, r-5}(1)]}
Let seg1 ε {0, 1} *and* seg2 ϵ {0, 1}.
r-1: [(seg1)(seg2)] $\Rightarrow$ [{r-1, r-2, r-3, r-4, r-5} ϵ]
r-2: [(seg1)(seg2)] $\Rightarrow$ [{r-1, r-2, r-3, r-4, r-5} (seg2)]
r-3: [(X)(seg1)] $\Rightarrow$ [{r-1 r-2, r-3, r-4, r-5}(X)]
r-4: [(X)(seg1)] $\Rightarrow$ [{r-1 r-2, r-3, r-4, r-5}(seg1 X)]
r-5: [(X)(#)] $\Rightarrow$ [{r-4}(X)]
r-6: [(seg1 X)(seg1)] $\Rightarrow$ [{r-4}(X)]
ST_F $=_{def}$ {[rp-6 ϵ]}

The grammar 8.4 is clearly a context-free C-LAG: no input pattern looks at more than one initial ss-category segment. It is not a C2-LAG, however, because it doesn't satisfy the single return principle.

Consider the rules r-3 and r-4, which have identical input conditions. Rule r-3 ignores the category of the next word in the output category (i.e., treats the next word as a noise symbol), while r-4 adds the next word category at the beginning of the output category. Rule r-3 causes an ambiguity split by calling input compatible r-3 and r-4. Rule r-4 causes a similar ambiguity split. Furthermore, these ambiguities are recursive because both branches of the respective splits feed back into the rules causing the ambiguity.

The nature of the language L_{no} is such that the multiple recursive ambiguity cannot be eliminated without changing the generative properties of the grammar. Thus, L_{no} is a counterexample to the claim [Hausser (1989)] that for any C-LAG there exists an equivalent C-LAG which parses in n^2.

Another nondeterministic context-free C3-LAG is HCFL, the "hardest context-free language" [Greibach (1973)].[17] HCFL is like L_{no} in that it provides for "noise" symbols which may precede and follow the "genuine" symbol sequences.

[17] See also Harrison (1978), pp. 326ff.

It remains to show that the class of C3-LAGs is $\mathcal{NP}$-complete. C3-LAGs clearly are within $\mathcal{NP}$ because C-LAGs are defined to verify in linear time—and *a fortiori* in polynomial time. To show that C3-LAGs are $\mathcal{NP}$-complete we define a C3-LAG for $\mathcal{NP}$-complete Subset Sum.

Subset Sum is defined as the language y#a_1#a_2#a_3#...#a_n# such that y, a_1, a_2, ..., a_n are all binary strings containing exactly the same number of digits. Furthermore, when viewed as binary numbers presented least significant digit first, y is equal to the sum of a subset of the a_i.

8.5 Syntactically Ambiguous C3-LAG for Subset Sum

LX $=_{def}$ {(0(0)), (1(1)), (#(#))}
ST_S $=_{def}$ {[{r-1 r-2}(0)], [{r-1 r-2}(1)]}
Let seg1 ε {0, 1}
r-1: [(X)(seg1)] ⇒ [{r-1, r-2}(seg1 X)]
r-2: [(X)(#)] ⇒ [{r-3, r-4, r-6, r-7, r-12, r-14}(# X)]
r-3: [(X seg1)(seg1)] ⇒ [{r-3, r-4, r-6, r-7}(0 X)]
r-4: [(X #)(#)] ⇒ [{r-3, r-4, r-6, r-7, r-12, r-14}(# X)]
r-5: [(X seg1)(seg1)] ⇒ [{r-5, r-6, r-7, r-11}(0 X)]
r-6: [(X 1)(0)] ⇒ [{r-5, r-6, r-7, r-11}(1 X)]
r-7: [(X 0)(1)] ⇒ [{r-8, r-9, r-10}(1 X)]
r-8: [(X seg1)(seg1)] ⇒ [{r-8, r-9, r-10}(1 X)]
r-9: [(X 1)(0)] ⇒ [{r-5, r-6, r-7, r-11}(0 X)]
r-10: [(X 0)(1)] ⇒ [{r-8, r-9, r-10}(0 X)]
r-11: [(X #)(#)] ⇒ [{r-3, r-4, r-6, r-7, r-12, r-14}(# X)]
r-12: [(X 0)(seg1)] ⇒ [{r-4, r-12, r-14}(0 X)]
r-13: [(X 0)(seg1)] ⇒ [{r-11, r-13, r-14}(0 X)]
r-14: [(X 1)(seg1)] ⇒ [{r-11, r-13 r-14}(1 X)]
ST_F $=_{def}$ {[rp-4 (X)]}

The above C3-LAG copies y into the category, and then non-deterministically either does or does not subtract each a_i from y. If the result of the subtraction is zero, it enters an accepting state, otherwise not.[18]

The C3-LAG for Subset Sum resembles that for L_{no} in that each a_i may function either as noise or as a genuine subset. Note, however, that 8.5 is a context-sensitive C-LAG: rule r-4, for example, works both at the end and the beginning of the sentence start category. The C-LAG 8.4, on the other hand, is context-free because the categorial operations affect only the first segment of sentence start categories.[19]

[18] Like all other LA-grammars defined in this paper, 8.5 has been tested as a parser. The rule system may be further simplified by collapsing rules.

[19] The C3-LAG for SAT is based on input strings where each disjunction is encoded as a vector, indicating for each variable if it is absent (a), present (p), or present negated (n). For example, if the input contains the variable w, x, y, and z, then (w v x) & (z v $\bar{y}$ v x) is represented as (ppaa)(apnp). These vectors are preceded by k letters, where k represents the number of distinct variables in the formula.

The C3-LAG begins by assigning all possible value assignments to the initial k-letter se-

9. Open Questions

At present C3-LAGs raise the following open question:.

- C3-LAGs generate $\mathcal{NP}$-hard context-sensitive languages such as Subset Sum and SAT, and are therefore $\mathcal{NP}$-complete. CS-recognition is known to be PSPACE complete [Hopcroft and Ullman (1979), p. 346,7]. Because a PSPACE-complete problem is not likely to be in $\mathcal{NP}$,[20] the set of C-languages is likely to be a proper subset of the B-languages (CS-languages). What is an example of a context-sensitive language which is not a C-language?

- LA-grammars are closed under union, concatenation, and Kleene Closure [Hausser (1989), Theorem 6, p. 145]. It remains to be shown whether different types of LA-grammars constitute *abstract families of languages* in the sense of Ginsberg and Greibach (1969).[21]

- There are syntactically ambiguous C-LAGs which can be translated into lexically ambiguous C-LAGs, and there are lexical ambiguities which can be simulated syntactically. Are syntactic and lexical ambiguities always intertranslatable? Do C-LAGs formally reflect the conceptual difference between lexical and syntactic ambiguity?[22]

Regarding the last point, consider the following lexically ambiguous variant of the syntactically ambiguous C3-LAG 8.4 for L_{no}.

9.1 A Lexically Ambiguous C3-LAG for L_{no}

LX $=_{def}$ {(0(0)), (1(1)), (0(3)), (1(3)), (#(#))}
ST_S $=_{def}$ {[{r-1, r-2}(0)] [{r-1, r-2}(1)]}
Let seg1 ε {0,1}. X' = nil *if* X = 3, *and* X' = X *otherwise.*
r-1: [(X)(seg1)] ⇒ [{r-1, r-2, r-3}(seg1 X')],
r-2: [(X)(3)] ⇒ [{r-1, r-2, r-3}(X')],
r-3: [(X)(#)] ⇒ [{r-4}(X)]
r-4: [(seg1 X)(seg1)] ⇒ [{r-4}(X)]

---

quence, resulting in 2^k different readings, e.g., 1111, 1110, 1101, 1100, etc. Then each assignment is tested left to right against each vector. For example, the assignment 1010 makes the first clause (disjunction) in (ppaa)(apnp) true, but the second false. As the grammar tests the variable values against the vector values, the vector values are discarded while the variable values are recycled and applied to the next disjunction.

[20] "Not only is a PSPACE-complete problem not likely to be in $\mathcal{P}$, it is also not likely to be in $\mathcal{NP}$. Hence the property whose existence is PSPACE-complete probably cannot even be *verified* in polynomial time using a polynomial length 'guess'." (Gary and Johnson 1979:171).

[21] See also Hopcroft & Ullman (1979), pp. 270ff.

[22] If the ambiguity of a language is intuitively lexical, then a syntactically ambiguous C-LAG will often require more rules than the corresponding lexically ambiguous C-LAG. On the other hand, if the ambiguity of a language is syntactic, then a lexically ambiguous C-LAG will often require more rules than the corresponding syntactically ambiguous C-LAG (as witnessed by 7.2 and 9.2).

$ST_F =_{def} \{[rp\text{-}4\ \epsilon]\}$

Note that 9.1 is not syntactically ambiguous because r-1, r-2, and r-3 have incompatible input conditions.

Given the view of L_{no} as a noise language, where certain words create confusion because of their homonymy with "genuine" words, the lexically ambiguous version 9.1 seems intuitively more natural than the syntactically ambiguous C3-LAG presented in 8.4. On the other hand, the nondeterministic context-free language WW^R represents an instance of syntactic ambiguity (see the C2-LAG defined in 7.2), though 9.2 demonstrates the possibility of a lexically ambiguous C-LAG for this language:[23]

9.2 A Lexically Ambiguous C-LAG for WW^R, W ε {a, b}$^+$

$LX =_{def} \{(a(0)), (b(1)), (a(a)), (b(b))\}$
$ST_S =_{def} \{[\{r\text{-}1, r\text{-}2\}\ (0)], [\{r\text{-}1, r\text{-}2\}\ (1)]\}$
r-1 $[(X)(0)] \Rightarrow [\{r\text{-}1, r\text{-}2, r\text{-}3, r\text{-}4\}(0\ X)]$
r-2 $[(X)(1)] \Rightarrow [\{r\text{-}1, r\text{-}2, r\text{-}3, r\text{-}4\}(1\ X)]$
r-3 $[(0\ X)(a)] \Rightarrow [\{r\text{-}3, r\text{-}4\}(X)]$
r-4 $[(1\ X)(b)] \Rightarrow [\{r\text{-}3, r\text{-}4\}(X)]$
$ST_F =_{def} \{[rp\text{-}3\ \epsilon], [rp\text{-}4\ \epsilon]\}$

9.2 uses the rules r-1, r-2 and the category segments in {0, 1} for W, and rules r-3, r-4 and the category segments in {a, b} for W^R. As soon as r-3 or r-4 fire in a derivation, r-1 and r-2 cannot be reapplied. Furthermore, r-3 and r-4 ensure the proper derivation of W^R by cancelling '0' with 'a' and '1' with 'b'. The C-LAG 9.2 is not syntactically ambiguous because all rules have incompatible input conditions.[24]

10. Comparing the CFGs and the C-LAGs

In computer science, the class of context-free languages has been thoroughly studied in the last 30 years, yielding many interesting formal and practical results. However, "It is no secret that context-free grammars are only a first order approximation to the various mechanisms used for specifying the syntax of modern programming languages" [Ginsberg 1980, p.7].[25]

Similar considerations hold for applications to natural languages analysis. Because the generative power of context-free grammars is too weak for the

[23] The grammar 9.2 was also provided by David Applegate.

[24] In the analysis of natural language, the distinction between syntactic and lexical ambiguities seems to be fairly clear intuitively. But within the formalism of C-LAGs the relation between syntactic and lexical ambiguities is more complicated than previously thought. Reliance on linguistically motivated concepts resulted in the claim that for any C-LAG there exists an equivalent one which parses in n^2 (Hausser 1989).

The language L_{no} demonstrated the necessity to distinguish between C-LAGs in general and C-LAGs which parse efficiently. At the same time it was shown by David Applegate that the class of C-LAGs is much larger than previously thought: the C1-LAGs include the indexed language $a^{i!}$ and the C3-LAGs are $\mathcal{NP}$-complete, including Subset Sum and 3SAT.

[25] See also Harrison 1978, pp. 219ff.

description of natural languages, context-free "base"-grammars have been augmented with powerful mechanisms which increase the generative capacity far beyond that of the context-free grammars. For example, standard transformational grammar is recursively enumerable [Peters & Ritchie (1973)] and LFG and GPSG are $\mathcal{NP}$-complete [Barton et al. (1987)].

For these reasons an alternative grammar formalism better suited for specifying the syntax of programming as well as natural languages is of considerable theoretical and practical interest. However, up to now alternative formalisms have either turned out to be weakly equivalent to the context-free phrase structure grammars (e.g., categorial grammar), or the new systems have been defined as *extensions* of context-free phrase structure grammars, based on the use of certain additional formal devices (e.g., the tree adjoining grammars and the index grammars). Thereby the class of confree-languages constitutes a proper subset of the class of tree adjoing languages (TALs), and the class of TALs constitutes a proper subset of the class of index languages.[26]

This raises the question, how "natural" is the class of context-free languages? In formal language theory, classes of languages are defined in terms of formal grammar types, and formal grammar types are distinguished in terms of different properties of the underlying mathematical formalism. Within phrase structure grammar, the class of context-free languages is defined in terms of restrictions ("left-hand sides of the rewriting rules must consist of single non-terminals") which are natural only within this particular formalism.

If we change the formalism, we get other types of natural restrictions which in turn yield other classes of languages. In LA-grammar, for example, the language classes defined by the C-1, C-2, and C-3 LAGs are defined in terms of whether or not ambiguities are recursive, and whether recursive ambiguities are "single return" or not. The resulting classification of languages is orthogonal to the context-free languages and their extensions, e.g., the tree-adjoining languages and the index languages.

This alternative classification of languages provides a new perspective on formal language theory. At the same time LA-grammar provides for a straightforward formal reconstruction of the PS-grammar hierarchy (as shown in Section 4). Thus, the many results proved on the basis of PS-grammars remain directly available in LA-grammar.

How does the LA-grammar hierarchy compare intuitively and computationally with the Chomsky hierarchy? To answer this question we must look at the languages which are being classified. Formal languages, like natural languages, exist independently of particular formal grammars, or grammar formalisms.

For example, $a^n b^n$, $a^n b^n c^n$, WW^R, and WW are names for formal languages. The notation of these names provides an informal description of the languages which is independent of the PS-grammar or the LA-grammar formalism. ¿From

[26] Also, the parsing efficiency of these extensions of context-free PS-grammar is too low to be of practical interest.

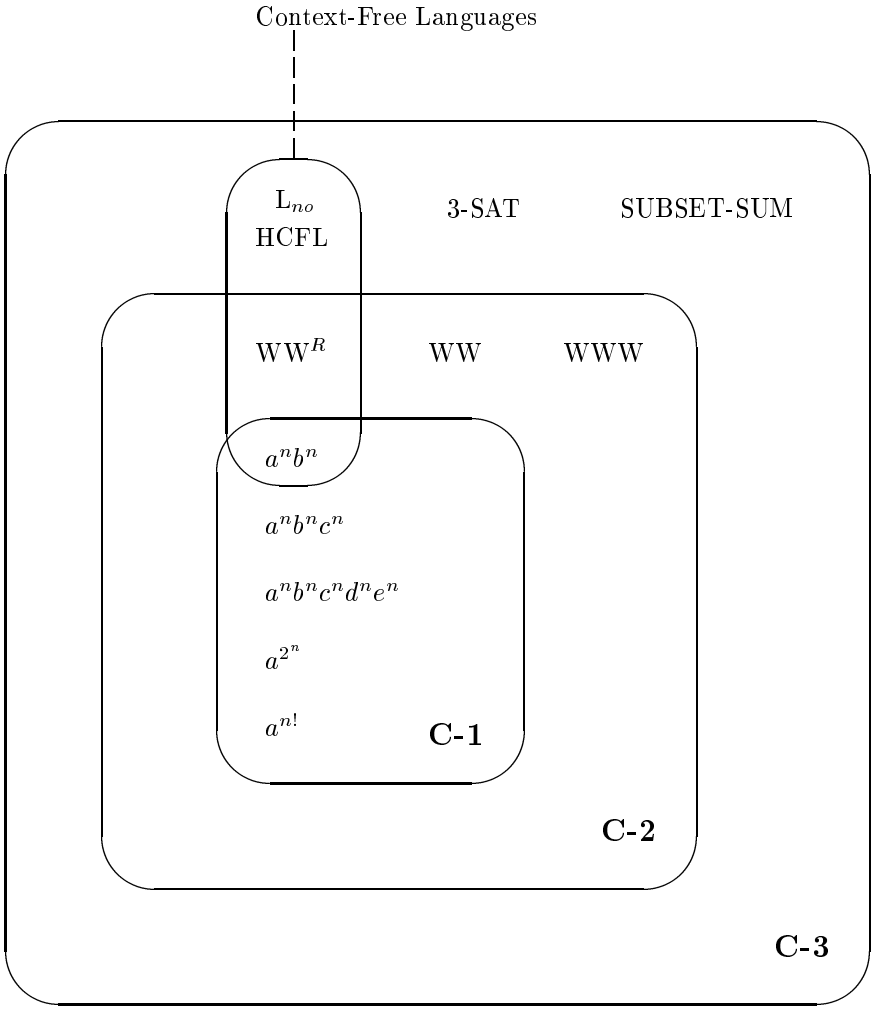
a pretheoretical point of view, one would be inclined to class $a^n b^n$ with $a^n b^n c^n$ (as well as $a^n b^n c^n d^n$, etc.), and WW^R with WW (as well as WWW, etc.).

It is always surprising to the beginning student that the Chomsky hierarchy puts $a^n b^n$ with WW^R into one class (context-free) that parses relatively efficiently, but puts $a^n b^n c^n$ with WW into another class (context-sensitive) that is PSPACE-complete. The LA-grammar hierarchy is intuitively more natural because it classifies $a^n b^n$ with $a^n b^n c^n$ (and $a^n b^n c^n d^n$, etc.) as C1-languages, and WW^R with WW (and WWW, etc.) as C2-languages.

Furthermore, we have argued that L_{no} and HCFL share structural properties with SAT and Subset Sum. From the viewpoint of LA-grammar, the parsing of L_{no} and HCFL in n^3—using a conventional PS-grammar based parser like Earley's or CYK—is just as much an artifact of the PS-grammar formalism as the exclusion of $a^n b^n c^n$, a^{2^i}, $a^k b^m c^{k \cdot m}$, $a^{i!}$, etc., from linear parsing, and the exclusion of WW, WWW, etc., from n^2 parsing.

The correlation of the context-free languages and the C-1, C-2, and C-3 languages is summarized in the following diagram.

10.1 Relating the Context-Free Languages to the C-Languages



The formal system of C-LAGs greatly expands the set of deterministic CF-languages (C1-LAGs, linear time), and the set of "reasonable" non-deterministic CF-Languages (C2-LAGs, square time). It also results in grouping context-free L_{no} and HCFL with SAT and Subset Sum in the class of C3-LAGs, which is

$\mathcal{NP}$-complete and parses in exponential time.

Concluding Remark

Analyzing the basic mathematical and computational properties of LA-grammar and comparing it with PS-grammar is of theoretical interest within formal language theory. LA-grammar provides a classification of formal languages that is orthogonal to the familiar distinction between regular, context-free and context-sensitive languages. For the first time, there is a true alternative to the Chomsky hierarchy.

The formal simplicity, computational efficiency, and mathematical power of LA-grammar will be relevant in a much larger context, however, if the formalism in question is well-suited for practical applications. For the wider public, interest in a new grammatical algorithm will be proportional to the usefulness of the grammar in a variety of different tasks, such as linguistic description, computational processing, and psychological explanation.

The suitability of LA-grammar for the description of natural language has been shown in extensive analyses of English and German syntax [Hausser (1986)], as well as in smaller systems of French, classical Latin, Polish, and Japanese. These applications demonstrate that LA-grammar handles the different types of structures characteristic of natural language (word order, agreement, long distance dependencies, center embedding, etc.) in a simple and linguistically well-motivated manner.

Regarding computational processing, LA-grammars are easy to parse because the rules of the grammar are used directly ("absolute type transparency"). Consequently, analysis of the processing provides heuristics for simplifying the grammar, and improvements in the grammar translate into faster processing. LA-grammars for natural (as well as formal) languages are easy to write, to extend, and to debug because the derivation of sentences is based on the principle of possible continuations.

¿From a psychological viewpoint, finally, the most basic property of LA-grammar is its strictly time-linear approach to language, providing a formalism that is input-output equivalent with the speaker-hearer. Thus, the LAG algorithm of possible continuations is linguistically and psychologically well-motivated. That LA-grammar is also computationally simple and efficient may provide additional incentive to further explore this new approach to natural and formal languages.

References

Ajdukiewicz, K. (1935) "Die syntaktische Konnexität," *Studia Philosophica*, 1:1-27.

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

Berwick, R.C., and A.S. Weinberg (1984) *The Grammatical Basis of Linguistic Performance: Language Use and Acquisition*. The MIT-Press, Cambridge, Massachusetts.

Book, R.V. (ed.) (1980) *Formal Language Theory: Perspectives and Open Problems*. Academic Press, New York.

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

Gary, M. and D. Johnson (1979) *Computers and Intractibility*. San Francisco: W.H. Freeman and Co.

Ginsberg, S. (1980) "Formal Language Theory: Methods for Specifying Families of Formal Languages—Past-Present-Future," in Book (1980), pp. 1-22..

Ginsberg, S. and S. A. Greibach (1969) "Abstract Families of Languages," *Memoirs of the American Mathematical Society*, 87.

Greibach, S.A. (1973) "The Hardest Context-Free Language," *SIAM Journal of Computing*, 2, pp. 304-310.

Harrison, M.A. (1978) *Introduction to Formal Language Theory*. Addison-Wesley Publishing Company, Reading, Massachusetts.

Hayashi, T. (1973) "On derivation trees of indexed grammars—an extension of the uvwxy theorem," *Publications of the Research Institute for Mathematical Sciences 9:1*, pp. 61-92.

Hausser, R. (1986) *NEWCAT: Parsing Natural Language Using Left-Associative Grammar*. Springer-Verlag Berlin–New York (Lecture Notes in Computer Science 231).

Hausser, R. (1989) *Computation of Language*. Springer-Verlag Berlin–New York (Symbolic Computation – *Artificial Intelligence*).

Hopcroft, J.E., and Ullman, J.D. (1969) *Formal Languages and their Relation to Automata*. Addison-Wesley Publishing Company, Reading, Massachusetts.

Hopcroft, J.E., and Ullman, J.D. (1979) *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, Reading, Massachusetts.

Joshi, A. (1987) "Intro to TAG" in A. Manaster-Ramer (ed.) *Mathematics of Language*. 1987, John Benjamins, Amsterdam.

Leśniewski, S. (1929) "Grundzüge eines neuen Systems der Grundlage der Mathematik," *Fundamenta Mathematicae*, Warsaw.

Peters, S., and Ritchie, R. (1973) "On the Generative Power of Transformational Grammar," *Information and Control*, 18:483-501.

Post, E. (1936) "Finite Combinatory Processes—Formulation I," *Journal of Symbolic Logic*, I.