COMPLEXITY AND DECIDABILITY IN LEFT-ASSOCIATIVE GRAMMAR[1]

ROLAND HAUSSER

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 systems of Post (1936). Rewriting rules have the following form:

(1.1) The Schema of a Phrase-Structure Rewriting Rule

$$A \rightarrow B\ C$$

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 a Categorial Canceling Rule

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

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.

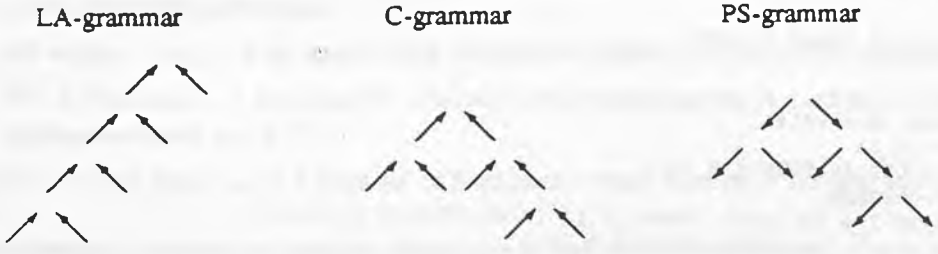
(1.3) The Schema of a Left-Associative Rule

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

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 different rule schemata result in 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),

[1] The results reported in this paper are published in Hausser, R. (1989) *Computation of Language*, Springer-Verlag Berlin-New York (Symbolic Computation – *Artificial Intelligence*), June 1989.

and (iii) use the same rules in the same derivation order. In other words, LA-grammar achieves "absolute type transparency"[2] 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 computional 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 tree structures generated by PS-grammar and C-grammar are semantically motivated *constituent structures*. Constituent structures are based on *substitution* and *movement tests* which are intended to reveal which parts of the sentence belong most closely together. The completely regular tree structure of LA-grammar, on the other hand, is based on the notion of *possible continuations* and reflects the time-linear nature of language.

As an example of a left-associative parse consider (2.1).

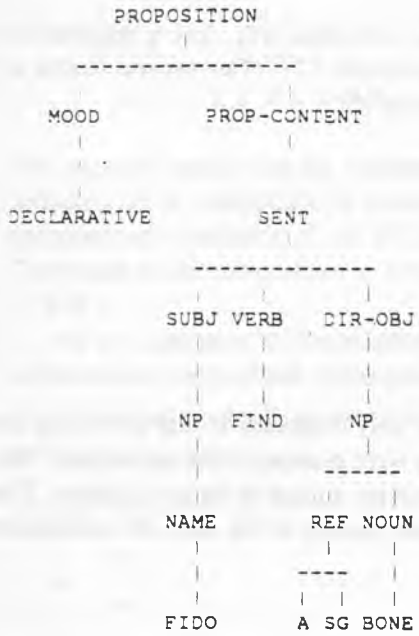
(2.1) A Sample Derivation

```
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)))))))))))
```

[2]Berwick & Weinberg (1984), p. 41.



The algorithm of left-associative grammar (LA-grammar) always combines a sentence start and a next word into a new sentence start. In a semantically interpreted LA-grammar, a homomorphic semantic hierarchy is constructed simultaneously with the linear syntactic parse. The semantic hierarchy expresses many of the intuitions central to constituent structure and may be displayed as a structured list or, equivalently, as a tree structure. The following discussion of LA-grammar is limited to the formal properties of the linear syntax.

3. The Mathematical Definition

(3.1) Formal Definition of Left-Associative Grammar[3]

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. $LX \subset (W \times C^+)$ is a finite set comprising the *lexicon.*
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. $ST_S = \{(rp_s\ cat_s), \ldots\}$ is a finite set of *initial states*, where each rp_s is a subset of n called a start rule package and each $cat_s \in C^+$.
7. $ST_F = \{(rp_f\ cat_f), \ldots\}$ is a finite set of *final states*, where each $rp_f \in RP$ and each $cat_f \in C^*$.

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

[3]Let us recall some notation from set theory needed in this 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 $(X \times Y)$ 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., $n = \{i \mid 0 \leq i < n\}$.

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

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

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

LA-grammar is equally suitable for parsing and generation. 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. The grammatical analysis provided by LA-parsers and LA-generators is simply a trace of the computation. The declarative linguistic analysis and the computation are merely different aspects of the same left-associative structure.

(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
       (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
```

The parse is called with the function "(z ...)". Note that categories precede the surfaces in (3.3). Each left-associative composition is characterized by a word number (e.g., 4), a sentence start consisting of a category 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).

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 $a^k b^k c^k$ defined in (3.2).

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

```
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
   1 2   (B C C)
 A A A
   1 1   (B B B 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 used for parsing, the function 'gram-gen' is called with two arguments: the "recursion factor" of the grammar (cf. Section 6), and a list of the words to be used.[4] The output is a 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).

[4] In another version, 'gram-gen' is called with the maximal surface length rather than the recursion factor.

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

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

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 element of the set of complete well-formed expressions of the LA-grammar for $a^k b^k c^k$ defined in (3.2).

4. Generative Capacity and the Chomsky Hierarchy

The most basic formal result in LA-grammar is that it generates all—and only—the recursive languages. That LA-grammar generates **all** recursive languages follows from the fact that a categorial operation can be any total recursive function.[5] That LA-grammar generates **only** the recursive languages, on the other hand, is due to the linear structure of the derivation: at each left-associative composition there is only a finite number of sentence starts, each with a finite rule package, and a finite number of next word readings, resulting in a finite number of new sentence starts.[6]

Furthermore, we may show that the automata-theoretic hierarchy of regular, context-free, and context-sensitive languages is clearly reflected in the formalism of LA-grammar. Specifically, **regular languages** are generated by LA-Grammars with rules using only empty categorial operations, e.g.,

(4.1) LA-Rule with Empty Categorial Operation

r_i: [ϵ CAT-2] $\Rightarrow$ [rp_i ϵ]

The proof is based on a systematic translation procedure from Finite State Automata into LA-grammars with rules like (4.1).[7]

Context-free languages are generated by LA-grammars with categorial operations which work only on the first segment of CAT-1 or CAT-3, e.g.,

r_i: [(a X)(a)] $\Rightarrow$ [rp_i (X)]

or

r_i: [(X)(a)] $\Rightarrow$ [rp_i (a X)]

where X is a variable for sequences of category segments. The proof is based on the corresponding restrictions on pushdown automata. In particular, the automaton may look only at the top of the stack, represented in the rule by CAT-1, and the automaton may only push or pop one element at a time from the stack (with the corresponding result represented by CAT-3).

Context-sensitive languages are generated by LA-grammars where the length of the categories is bounded by $C \cdot n$, where C is a finite constant and n is the length of a complete well-formed input expression. The proof is based on the corresponding restrictions on linearly bounded automata.

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

A more natural way of dividing possible languages in LA-grammar than the Chomsky hierarchy is the hierarchy of A-LAGs, B-LAGs, and C-LAGs. This new hierarchy is based on the properties of the categorial operations of the rules of LA-grammar. The crucial formal property of a categorial operation—from a complexity point of view—is whether or not it has to search through indefinitely-long sentence-start categories.

---

[5] For a detailed proof see Hausser (1989), Theorem 2, p. 135.

[6] For a detailed proof see Hausser (1989), Theorem 1, p. 134.

[7] For a detailed discussion see Hausser (1989), Section 8.2. A more general characterization of the regular languages is given in Theorem 3, p. 138.

(5.1) 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.[8] A language is called a C-language iff it is recognized by a C-LAG.

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, 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. But the LA-grammars for many context-sensitive languages, e.g., $a^kb^kc^k$ (cf. (3.2)), $a^kb^kc^kd^ke^k$, WW, and WWW, are also C-LAGs.

Generally speaking, an 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).

Non-constant LA-grammars are divided into the *B-LAGs* and *A-LAGs*.

(5.2) 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.

(5.3) 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:

(5.4) 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.

(6) Decidability

For arbitrary context-free grammars it is undecidable whether the languages generated are ambiguous, in an inclusion relation, or equivalent. In LA-grammar, on the other hand, questions of ambiguity, emptiness, inclusion, and equivalence are decidable for a large subset of the C-LAGs which includes context-sensitive languages. These results are based on the fact that the derivational structure of LA-grammar clearly exhibits the occurrence of grammatical recursions.

The following definition is based on the notion of "abstract derivations". Two derivations are represented by the same abstract derivation if they differ only in the choice of words, but exhibit the same sequence of rules and the same sequence of categories. In an abstract derivation different words of the same category, e.g., (table (sn)) and (chair (sn)), are represented by one abstract word, e.g., (A (sn)).

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

(6.1) Definition of a Grammatical Recursion

An abstract derivation exhibits a grammatical recursion if and only if

1. the surface exhibits two or more identical subsequences which are directly adjacent,
2. the rule sequence exhibits two or more identical subsequences which correspond to the surface, and
3. each instance of the recursion affects the sentence-start category in a regular way.

How sentence-start categories are affected by a recursion depends on the type of the recursion. LA-grammar distinguishes between (i) constant, (ii) increasing, (iii) decreasing, and (iv) simultaneously increasing and decreasing grammatical recursions. A recursion is constant if the sentence start categories at the beginning of two adjacent loops are identical. A recursion is increasing if the sentence start category at the beginning of the second loop is longer than the sentence start category at the beginning of the first loop. And correspondingly for the other cases.

Here is how the algorithm recognizes and types recursions: Assume the generator has derived a string of length n, and is in the process of adding the n+1st word—e.g., A—by means of a certain rule, e.g. 1.

(6.2) Example of a Grammatical Recursion

```
.......ABCABC A
.......123123 1 (cat)
```

The algorithm for recognizing recursions checks whether the current rule, i.e., rule 1, has two predecessors. If so, it checks whether the (abstract) surfaces added by the occurrences of rule 1 are all the same. If so, it checks whether the rule sequences and the surface sequences between the occurrences of rule 1 are identical. If all these conditions are satisfied, a recursion has been recognized. Finally, the recursion is typed by comparing the categories of the expression in question with its shorter predecessors ending in surface A and rule 1.

The crucial problem for proving decidability in LA-grammar is to determine how often grammatical recursions have to be applied in order for the set of completions to be a "representative sample". In the class of C-LAGs, the grammatical structure provides a "recursion factor" which specifies how often the increasing recursions of the grammar have to be applied in order to arrive at a representative sample. During the generation of longer and longer expressions, the system keeps track of increasing recursions and stops the recursion as soon as the number specified by the recursion factor has been reached.

In most C-LAGs this procedure results in a finite set of derivations which is representative in the sense that all sentence types generated by the grammar are exemplified in it. Such a representative sample provides the basis for deciding ambiguity, inclusion, equivalence, and (non-)emptiness of C-LAGs.

Not all C-LAGs are decidable, however. In C-LAGs with *simultaneously increasing and decreasing recursions such that the increase is greater than the decrease* the recursion factor does not guarantee the derivation of a representative sample. Those C-LAGs where the process of a systematic derivation, controlled by a grammar-dependent recursion factor, results in finite sets of representative samples are called D-LAGs ("Decidable C-LAGs"). An example of a D-LAG is $a^k b^k c^k$, for which a representative sample is derived in (3.4).

Can the technique of proving the subset and the equality relationship, as well as ambiguity and (non-)-emptiness for a large class of context-free and context-sensitive languages be used for PS-grammars as well? Because PS-grammars have a different derivational structure, the method of deriving longer and longer sentence starts cannot be applied directly in PS-grammar. The only possibility would be a systematic translation of PS-grammars into C-LAGs, and proving the properties in question indirectly by way of the weakly equivalent C-LAGs.

However, this approach requires that there is a general algorithm for translating PS-grammars into LA-grammars. No such algorithm has been found. Furthermore, experience writing LA-grammars for

languages described originally as PS-grammars has shown that the construction of the LA-grammar is never based on the PS-grammar for the language, but proceeds from the language directly. Thus, it is unlikely that such an algorithm exists.

(7) The Complexity of Sound C-LAGs

Earley (1970) showed that the Earley algorithm recognizes **unambiguous** context-free grammars in $|G|^2 \cdot n^2$, but **ambiguous** context-free grammars in $|G|^2 \cdot n^3$ (where $|G|$ is the size of the grammar and n the length of the input string). Thus, computational complexity in PS-grammar depends not only on the class of the grammar, e.g., regular, context-free, or context-sensitive, but also on whether or not the grammar is ambiguous.

It is similar in LA-grammar: computational complexity depends not only on whether the grammar is a C-LAG, B-LAG, or A-LAG, but also on whether or not the grammar is ambiguous. LA-grammar distinguishes three levels of ambiguity:

(7.1) Three Levels of Ambiguity in LA-Grammar

1. unambiguous grammars
2. syntactically-ambiguous grammars
3. lexically-ambiguous grammars

Syntactic ambiguity is defined in terms of the input-compatibility of rules.

(7.2) 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.

(7.3) 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)].

(7.4) 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.

(7.5) Definition of Lexically-Ambiguous LA-Grammars

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

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. Unambiguous C-LAGs are proven to parse in linear time. This result follows from definition (7.3), and is significant insofar as it applies not only to the (non-deterministic) context-free but also to many context-sensitive languages (e.g., $a^k b^k c^k$ as defined in (3.2)).

In the case of syntactically ambiguous LA-grammars, the crucial source of computational complexity are *recursive ambiguities*. In *sound* LA-grammars recursive ambiguities are restricted by the *single return principle*.

(7.6) The Single Return Principle (SRP)

> If a syntactic ambiguity arises inside a recursion, then only one of the branches resulting from the ambiguity may feed back into the recursion.

As a consequence of the SRP, sound LA-grammars have—at most—$(C \cdot n)$ readings.[9] Furthermore, the SRP does not decrease the generative capacity of an LA-grammar.[10] Because for any LA-grammar there exists a weakly equivalent sound LA-grammar, any syntactically ambiguous C-language can be parsed in n^2.

In the case of systematic lexical ambiguity, finally, there are two choices. One is to eliminate the lexical ambiguities by means of neutral categories. The other is to "pack" the readings, which may be exponential in number, into a single representation. It may be shown that these strategies are always possible within the class of C-LAGs.[11] In summary, if a language can be generated or recognized by a C-LAG then there exists a C-LAG which will parse it in n^2.

References

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

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

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

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

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

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

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

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

[10] See Hausser (1989), Theorem 11, p. 224.

[11] Note that the problem of Boolean satisfiability (cf. Hopcroft & Ullman (1979), p. 325) exceeds not only the power of context-free grammars, but also of C-LAGs. An LA-grammar would not only have to build longer and longer categories in order to keep track of the different value assignments, but it would also have to check through the category each time it encounters another propositional variable. It is this second requirement which violates the definition of C-LAGs.