Grammatical Disambiguation

The Natural Language Linear Complexity Hypothesis

Roland Hausser
Universität Erlangen-Nürnberg (em.)

Abstract

By combining concatenations of constant complexity with a strictly time-linear derivation order, the computational complexity degree of DBS (AIJ'01) is linear time (TCS'92). The only way to increase DBS complexity above linear would be a recursive ambiguity in the hear mode. In natural language, however, recursive ambiguity is prevented by grammatical disambiguation.

An example of grammatically disambiguating a nonrecursive ambiguity is the 'garden path' sentence The horse raced by the barn fell (Bever 1970). The continuation horse+raced introduces a local ambiguity between *horse raced* (active) and *horse which was raced* (passive), leading to two parallel derivation strands up to and including barn. Depending on continuing after barn with an interpunctuation or a verb, one of the [-global] readings (FoCL 11.3) is grammatically eliminated.

An example of grammatically disambiguating a recursive ambiguity is The man who loves the woman who loves Tom who Lucy loves, with the subordinating conjunction who. Depending on whether the continuation after who is a verb or a noun, one of the two [-global] readings is grammatically eliminated (momentary choice between who being subject or object).

keywords: recursive ambiguity, grammatical disambiguation, agent-based data-driven versus sign-based substitution-driven ontology, holistic vs. incremental loading of input

1 Degrees of Computational Complexity

Given an algorithm taking an input of length n (n>1), its time complexity is commonly estimated (i) by counting the number of primitive operations needed for adding a next input item and (ii) the increase in the number of operations with the increase of the length n. The basic complexity degrees are a linear, polynomial, exponential, or unbounded increase with the length of the input:

1.1 Basic Degrees of Complexity

1. *Linear complexity*
   $1 \cdot n$, $2 \cdot n$, $3 \cdot n$, $4 \cdot n$, etc. (e.g., 2, 4, 6, 8, .. for n=2)
2. *Polynomial complexity*
   n^1, n^2, n^3, n^4, etc. (e.g., 2, 4, 9, 16, 25, 36, 49, 64, ... for n=2))
3. *Exponential complexity*
   1^n, 2^n, 3^n, 4^n, etc. (e.g., 1, 4, 8, 16, 32, 64, 128, 256,... for n=2))
4. *Undecidable*
   $n \cdot \infty$

In praxi, the most important distinction is between the computationally *tractable* and *intractable* complexity degrees. As shown by Garey and Johnson (1979), the boundary is between the (2) polynomial and the (3) exponential algorithms:

1.2 TIMING OF POLYNOMIAL VS. EXPONENTIAL ALGORITHMS

	problem size n		
time complexity	10	50	100
n^3	0.001 seconds	0.125 seconds	1.0 seconds
2^n	0.001 seconds	35.7 years	10^{15} centuries

The *primitive operation*[1] used is adding the next word (i.e., the minimum). The respective application numbers are shown for lengths 10, 50, and 100.

2 The Orthogonal LAG and PSG Complexity Hierarchies

Two complexity hierarchies are orthogonal if they classify certain formal languages differently. For example, in PSG the formal languages a^nb^n and WW^r are in the same complexity class, polynomial, but in different classes, C1 (linear) vs. C2 (polynomial), in LAG. In PSG, a^nb^n and $a^nb^nc^n$ are in different complexity classes, polynomial vs. exponential, but in the same class, C1 (linear), in LAG. The formal language L_{no} is polynomial in PSG, but exponential in LAG. The reason for these differences is (i) the substitution-driven derivation of PSG and (ii) the data-driven derivation of LAG.

Substitution-driven PSG favors input which is *pairwise inverse*, like abcd dcba. Formal languages which require no more than this correspondence, are called context-free and of polynomial complexity, but formal languages which exceed the pairwise inverse correspondence are computationally intractable in PSG.[2]

Data-driven LAG favors input which is *not recursively ambiguous*. Unambiguous languages, such as a^nb^n, $a^nb^nc^n$, $a^nb^nc^nd^n$, etc., a^{2^n}, $a^{n!}$, and single return languages, such as WW^r, WW, and WWW, are computationally tractable, but languages which are recursively ambiguous with a degree greater 2, such as 3SAT,

[1] Earley (1970) characterizes a primitive operation as "in some sense the most complex operation performed by the algorithm whose complexity is independent of the size of the grammar and the input string." The nature of the primitive operation varies from one grammar formalism to the next.

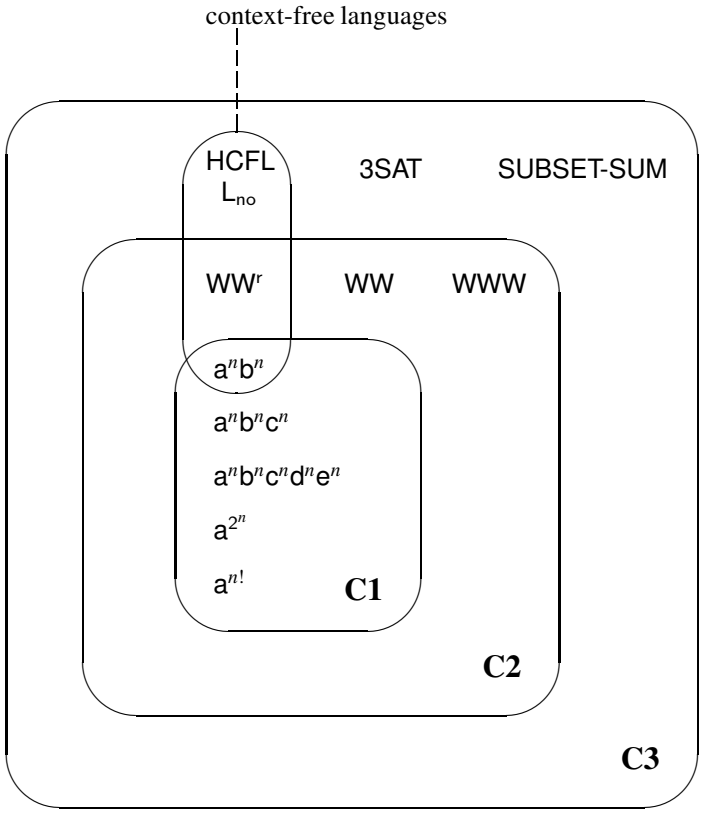
For example, Earley chose the operation of *adding a state to a state set* as the primitive operation of his famous algorithm for context-free grammars (FoCL 9.3). In LA Grammar, the subclass of C-LAGs uses a *rule application* as its primitive operation.

[2] According to Harrison (1978, p.219f.) and Ginsburg (1980, p.8), it is 'doubtful' that the structure of context-free PSG approximates the syntax of the programming languages. In other words, the programming languages must be computationally tractable but certainly not pairwise inverse.

SUBSET-SUM, L_{no}, and HCFL (Greibach 1973) are computationally intractable in LAG/DBS,

The two complexity hierarchies may be compared graphically as follows:

2.1 ORTHOGONAL RELATION BETWEEN C AND CF LANGUAGES



That LAG classifies L_{no} with inherently complex 3SAT and SUBSET-SUM is not because L_{no} is particularly complex, but because it is recursively ambiguous. That PSG classifies a^nb^n and $a^nb^nc^n$ in different classes, context-free (polynomial) vs. context-sensitive (exponential), is not because $a^nb^nc^n$ is particularly complex as compared to a^nb^n, but because it is not pairwise. That PSG classifies WW^r and WW in different classes, context-free vs. context-sensitive, is not because WW is particularly complex as compared to WW^r, but because it is not inverse.

3 Comparing Explicitly Defined Examples in PSG and DBS

Formal languages critical for distinguishing the complexity hierarchies of PSG and LAG are a^nb^n and $a^nb^nc^n$:

3.1 EXPLICIT PSGS FOR THE FORMAL LANGUAGES $a^n b^n$ AND $a^n b^n c^n$

$a^n b^n$ (*polynomial* n^3)

$V =_{def} \{S, a, b\}$

$V_T =_{def} \{a, b\}$

$P =_{def} \{S \rightarrow a\ S\ B$
$S \rightarrow a\ b\}$

$a^n b^n c^n$ (*exponential*)

$V =_{def} \{S, B, C, D_1, D_2, a, b, c\}$

$V_T =_{def} \{a, b, c\}$

$P =_{def} \{S \rightarrow a\ S\ B\ C,$ *rule 1*
$S \rightarrow a\ b\ C,$ *rule 2*
$C\ B \rightarrow D_1\ B$ *rule 3a*
$D_1\ B \rightarrow D_1\ B$ *rule 3b*
$D_1\ D_2 \rightarrow B\ D_2$ *rule 3c*
$B\ D_2 \rightarrow B\ C$ *rule 3d*
$B\ b \rightarrow b\ b$ *rule 4*
$B\ C \rightarrow B\ c$ *rule 5*
$c\ C \rightarrow c\ c\}$ *rule 6*

The PSG for $a^n b^n c^n$ generates pairwise inverse aSBC, aaSBCBC, aaaSBCBCBC, etc. with rule 1 and concludes with rule 2. Then the BCBCBC... sequence is changed into lower case and reordered step by step into bbb...ccc... with rules 3a–6. The rules compute *possible substitutions* and distinguish between nonterminal (e.g., B) and terminal (e.g., b) symbols.

The operations of a LAG, in contrast, compute *possible continuations* and distinguish between the surface and the category of an input, e.g., [aaabb (abb)] (FoCL 10.4.1), running the derivation via the category, used as counter.

3.2 EXPLICIT LAGS FOR $a^n b^n$ AND $a^n b^n c^n$

$a^n b^n$ (*linear*)

$LX =_{def} \{[a\ (a)], [b\ (b)]\}$
$ST_S =_{def} \{[(a)\ \{r_1, r_2\}]\}$
r_1: (X) (a) $\Rightarrow$ (aX) $\{r_1, r_2\}$
r_2: (aX) (b) $\Rightarrow$ (X) $\{r_2\}$
$ST_F =_{def} \{[\varepsilon\ rp_2]\}$.

$a^n b^n c^n$ (*linear*)

$LX =_{def} \{[a\ (a)], [b\ (b)], [c\ (c)]\}$
$ST_s =_{def} \{[(a)\ \{r_1, r_2\}]\}$
r_1: (X) (a) $\Rightarrow$ (aX) $\{r_1, r_2\}$
r_2: (aX) (b) $\Rightarrow$ (Xb) $\{r_2, r_3\}$
r_3: (bX) (c) $\Rightarrow$ (X) $\{r_3\}$
$ST_F =_{def} \{[\varepsilon\ rp_3]\}$.

Another language pair critical for the distinction between PSG and LAG are inverse WW^r and repeating WW. Both are pairwise, but inverse WW^r is context-free (n^3 polynomial), while repeating WW is context-sensitive (exponential) in PSG:

3.3 EXPLICIT PSG FOR WW^r AND INFORMAL FOR WW

WW^r (*polynomial* n^3)

$V =_{def} \{S, a, b, c, d\}$
$V_T =_{def} \{a, b, c, d\}$
$P =_{def} \{S \rightarrow a\ S\ a,$
$S \rightarrow b\ S\ b,$
$S \rightarrow c\ S\ c,$
$S \rightarrow d\ S\ d,$
$S \rightarrow a\ a,$
$S \rightarrow b\ b,$
$S \rightarrow c\ c,$
$S \rightarrow d\ d\}$

WW (*exponential*)

Similar to the PSG for $a^n b^n c^n$ (3.1), the derivation generates intermediate expressions like aSA, abSBA, abcSCBA, etc., and then reorders in lower case.

In LAG, WW^r and WW are in the same complexity class, namely n^2 polynomial:

3.4 EXPLICIT LAGS FOR WW^r AND WW

WW^r (*polynomial* n^2)

$LX =_{def} \{[a\ (a)], [b\ (b)], [c\ (c)], [d\ (d)] \dots\}$
$ST_S =_{def} \{[(seg_c)\ \{r_1, r_2\}]\}$, where $seg_c \in \{a, b, c, d, \dots\}$
r_1: $(X)\ (seg_c) \Rightarrow (seg_c\ X)\ \{r_1, r_2\}$
r_2: $(seg_c\ X)\ (seg_c) \Rightarrow (X)\ \{r_2\}$
$ST_F =_{def} \{[\varepsilon\ rp_2]\}$

WW (*polynomial* n^2)

$LX =_{def} \{[a\ (a)], [b\ (b)], [c\ (c)], [d\ (d)] \dots\}$
$ST_S =_{def} \{[(seg_c)\ \{r_1, r_2\}]\}$, where $seg_c \in \{a, b, c, d, \dots\}$
r_1: $(X)\ (seg_c) \Rightarrow (X\ seg_c)\ \{r_1, r_2\}$
r_2: $(seg_c\ X)\ (seg_c) \Rightarrow (X)\ \{r_2\}$
$ST_F =_{def} \{[\varepsilon\ rp_2]\}$

In WW^r, rule r_1 adds the counterpart letter (seg_c) at the beginning of the category, but at the end in WW.

The recursive ambiguity of WW^r and WW is of the kind *single return* (FoCL 11.5.3): in each derivation step, the rule package $\{r_1, r_2\}$ of r_1 calls two input-compatible rules (ambiguity), but the continuation split is disambiguated by the following input.

We complete the orthogonal relation between the PSG and the LAG complexity hierarchies with the noise language L_{no}, which is context-free (polynomial) in PSG, but C3 (exponential) in LAG. Devised by D. Applegate, its expressions consist of an arbitrary sequence of 0 and 1, followed by the separation symbol #, followed by an inverse copy with arbitrarily missing symbols of the initial sequence. Thus, when the initial sequence is read in by a LAG, it is not known until the end which pre-separation digits turn out to be genuine and which are noise:

3.5 EXPLICIT PSG AND LAG FOR L_{no}

L_{no} in PSG (context-free, polynomial)

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

L_{no} in LAG (C3, exponential)

$LX =_{def} \{[0\ (0)], [1\ (1)], [\#\ (\#)]\}$
$ST_S =_{def} \{[(seg_c)\ \{r_1, r_2, r_3, r_4, r_5\}]\}$,
where $seg \in \{0, 1\}$.
r_1: $(seg_c)(seg_d) \Rightarrow \varepsilon\ \{r_1, r_2, r_3, r_4, r_5\}$
r_2: $(seg_c)(seg_d) \Rightarrow (seg_d)\{r_1, r_2, r_3, r_4, r_5\}$
r_3: $(X)(seg_c) \Rightarrow (X)\{r_1\ r_2, r_3, r_4, r_5\}$
r_4: $(X)(seg_c) \Rightarrow (seg_c\ X)\{r_1\ r_2, r_3, r_4, r_5\}$
r_5: $(X)(\#) \Rightarrow (X)\{r_6\}$
r_6: $(seg_c\ X)(seg_c) \Rightarrow (X)\ \{r_6\}$
$ST_F =_{def} \{[\varepsilon\ rp_6]\}$

The complexity hierarchies of PSG and LAG may be summarized as follows:

3.6 COMPLEXITY DEGREES OF THE LAG AND PSG HIERARCHIES

	LA Grammar	*PS Grammar*
undecidable	—	recursively enumerable languages
decidable	A languages[3]	—
exponential	B languages	context-sensitive languages
exponential	C3 languages	
polynomial	C2 languages	context-free languages
linear	C1 languages	regular languages

The LAG hierarchy does not have a class of undecidable languages, while the PSG hierarchy does not have a class of decidable languages. The statement 'there is a parser for the context-free languages' means that there is a PSG parser which can handle all languages in the class, e.g., the Earley parser. 'There is no parser for the context-sensitive languages' means that there is no parser for all languages in the class. Thus, there may exist a computationally tractable parser specifically for context-sensitive $a^n b^n c^n$, but not for inherently complex 3SAT or SUBSET-SUM, which are also context-sensitive.

4 Sub-Hierarchy of C1, C2, and C3 LAGs

Compared to the A and B LAGs, the C LAGs constitute the most restricted class of LAGs, parsing the smallest LAG class of languages. However, compared to the context-free languages (which are properly contained in the C languages), the class of C languages is quite large (2.1). It is therefore theoretically interesting and practically useful to differentiate the C LAGs further into subclasses by defining a sub-hierarchy.

In the C LAGs, the complexity of a rule application is constant (TCS'92, Definition 4.1). Therefore, the number of rule applications in a C LAG derivation depends solely on the ambiguity degree. Different ambiguity degrees naturally define the sub-hierarchy of the C LAGs: in the subclasses of C1, C2, and C3 LAGs, increasing degrees of ambiguity result in increasing degrees of complexity.

The subclass with the lowest complexity and the lowest generative capacity is the C1 LAGs. A C LAG is a C1 LAG if it is not recursively ambiguous. The class of C1 languages parses in linear time and contains all deterministic context-free languages which are recognized by a DPDA without ε-moves, plus context-free languages with –recursive ambiguities, e.g., $a^k b^k c^m d^m \cup a^k b^m c^m d^k$, as well as many context-sensitive languages, e.g., $a^k b^k c^k$, $a^k b^k c^k d^k e^k$, $\{a^k b^k c^k\}^*$, L_{square}, L^k_{hast}, a^{2^i}, $a^k b^m c^{k \cdot m}$, and $a^{i!}$, whereby the last one is not even an index language.[4] Examples of unambiguous context-sensitive C1 LAGs are $a^k b^k c^k$ defined in 10.3.3 and

[3]The algebraic definition of LA-grammar (FoCL 10.2) benefited greatly from help by Professor Dana Scott, who also provided the proof that the class of A-languages comprises *all* recursive languages (FoCL. 11.1.3).

$a^{2^i} =_{def} \{a^i \mid i \text{ is a positive power of 2}\}$ (TCS'92 Definition 5.) A non-recursively ambiguous C1 LAG is $a^k b^k c^m d^m \cup a^k b^m c^m d^k$ (TCS'92)[5].

A C LAG is a C2 LAG if it generates recursive ambiguities which are restricted by the single return principle.

4.1 The Single Return Principle (SRP)

> A recursive ambiguity is single return if exactly one of the parallel paths returns to the state resulting in the ambiguity in question.

The class of C2 languages parses in polynomial time and contains certain non-deterministic context-free languages like WW^R and L^{∞}_{hast}, plus context-sensitive languages like WW, $W^{k\geq 3}$, $\{WWW\}^*$, and $W_1W_2W_1^RW_2^R$.[6]

For example, the worst case in parsing WW^R is inputs consisting of an even number of the same word (letter). Consider the derivation structure for the input a a a a a a, with `1` for r-1 and `2` for r-2.

4.2 Derivation structure of the worst case in WW^R

```
rules:          applications:

2               a$a
122             aa$aa
11222           aaa$aaa
11122           aaaa$aa
11112           aaaaa$a
11111           aaaaaa$
```

The unmarked middle of the intermediate strings generated in the course of the derivation is indicated by `$`. Of the six hypotheses, the first two are invalidated by the fact that the input string continues, the third hypothesis correctly corresponds to the input a a a a a a, and the remaining three hypotheses are invalidated by the fact that the input does not provide any more words (grammatical disambiguation in a formal language).

A C LAG is a C3 LAG if it generates unrestricted recursive ambiguities. The class of C3 languages parses in exponential time and contains the deterministic context-free language L_{no}, the hardest context-free language HCFL, plus context-sensitive languages like SubsetSum and SAT, which are $\mathcal{NP}$-complete.[7]

---

[4] A C1 LAG for $a^k b^k c^m d^m \cup a^k b^m c^m d^k$ is defined in FoCL 11.5.2; for L_{square} and L^k_{hast} in Stubert (1993), pp. 16 and 12; for $a^k b^k c^k d^k e^k$ in CoL, p. 233; for $a^k b^m c^{k \cdot m}$ in TCS'92, p. 296 and for a^{2^i} in FoCL 11.5.1. A C1 LAG for $a^{i!}$ is sketched in TCS'92, p. 296, footnote 13.

[5] This language has been called *inherently ambiguous* because there is no unambiguous PSG for it (Hopcroft and Ullman 1979, pp. 99–103).

[6] The C2 LAGs for WW^R and WW are defined in 3.3; for L^{∞}_{hast} in Stubert 1993, p. 16; for WWW in CoL, p. 215; for $W^{k\geq 3}$ in CoL, p. 216; and for $W_1W_2W_1^RW_2^R$ in FoCL 11.5.7.

[7] A C3 LAG for L_{no} is defined in 3.5; for HCFL in Stubert (1993), p. 16; for SubsetSum in FoCL 11.5.8; and for SAT in TCS'92, p. 302, footnote 19.

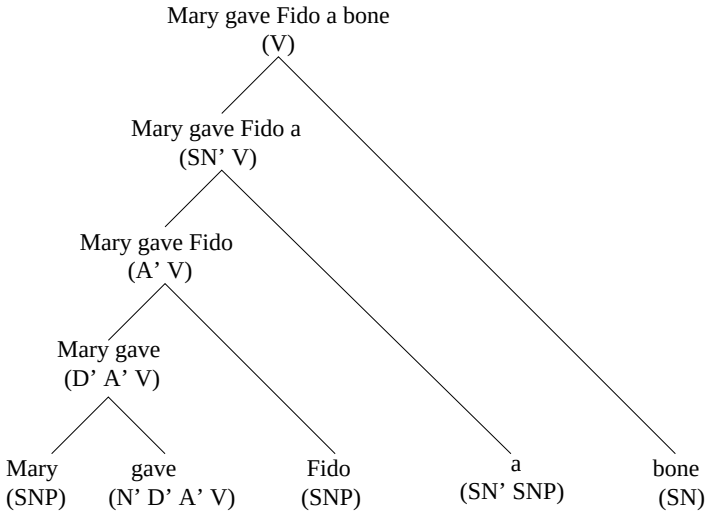
5 Applying LAG to Natural Language

In a LAG, the lexical entries have a two-level structure, consisting of a surface and a category, e.g., [a (a)] in linear notation. During a derivation, the surface and the category may diverge, e.g., [aaab (aab)] (FoCL 10.4.1, segment 4). In a LAG for a formal language, the rules have abstract names like r-1 or r-2.

In the application of LAG to natural language, the possible divergence between surfaces and categories is used for defining grammatically motivated categories such as [gave (N′ D′ A′ V)]. Also, the abstract rule names are replaced by grammatically meaningful ones like DET+CN or NP+VERB.

The following analysis is based on the LAG *LA E2* defined in FoCL 17.4.1 for a small fragment of English:

5.1 TIME-LINEAR LAG ANALYSIS OF AN ENGLISH SENTENCE



The first word [Mary (SNP)] cancels[8] the valency position N′ in the second word [gave (N′ D′ A′ V)], resulting in the new sentence start [Mary gave (D′ A′ V)] one level above (bottom up). Next the current sentence start [Mary gave (D′ A′ V)] combines with the current next word [Fido (SNP)], resulting in the new sentence start [Mary gave Fido (A′ V)] with the canceled valency position D′. This time-linear procedure continues until all valency positions are canceled, resulting in [Mary gave Fido a bone (V)].

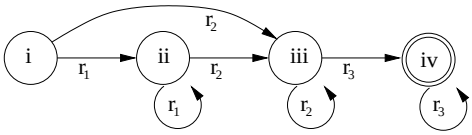
The application of this method to 221 constructions of German and 114 constructions of English in NEWCAT'86 showed that a strictly time-linear derivation order for natural language was empirically feasible. Also, the NEWCAT program was shown to be extremely efficient computationally as compared to competing efforts at the time, such as phrase structure-based LFG.

Nevertheless, NEWCAT is still a stand-alone algorithm in the style of classic complexity analysis. Instead of being data-driven, the system of rule packages

in a LAG constitutes a finite state transition network (FSN). By annotating the transitions with the associated rule name, the FSN is turned into an ATN.

Consider the ATN of the LAG grammar for context-sensitive $a^n b^n c^n$ (3.2):

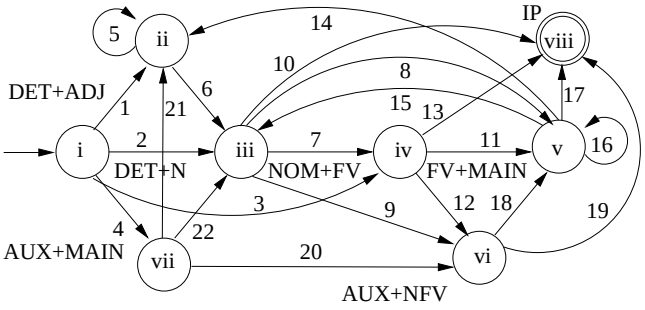
5.2 ANNOTATED TRANSITION NETWORK OF THE LAG FOR $a^n b^n c^n$



This ATN consists of four states, represented as the circles i – iv. Each state is defined as an ordered pair consisting of a rule name and a rule package. State i corresponds to the start state ST_S, while the states ii, iii, and iv correspond to the output of rules r_1, r_2, and r_3. State iv has a double circle, indicating a possible final state (definition of ST_F in 3.2).

However, when the LAG system was applied to basic structures of natural language, such as extending declaratives to yes-no interrogatives, it turned out that the use of rule packages became prohibitively complex. Consider the finite state backbone of *LA E3* defined in FoCL 17.5.6.

5.3 ANNOTATED TRANSITION NETWORK FOR *LA E3*



(ii)	1, 5, 14, 21	DET+ADJ	(v)	8, 11, 16, 18	FVERB+MAIN
(iii)	2, 6, 15, 22	DET+N	(vi)	9, 12, 20	AUX+NFV
(iv)	3, 7	NOM+FV	(vii)	4	AUX+MAIN
			(viii)	10, 13, 17, 19	IP

Including the start state, there are eight states. As in 5.2, the transition from one state to the next is annotated with a rule name, thus restricting the transition to a specific categorial operation. The problem was not in writing the rule packages for controlling the parsing of a particular natural language construction, but writing the ATN for the complete system.

[8] At this stage of the theory, valency positions are canceled by deletion, as in CG. This is not a problem as long as the theory is sign-based. In DBS, however, the valency information must be preserved for repeated hear say in the speak mode by using #-marking for canceling. For example, [Mary gave (D' A' V)] cancels by deletion in LAG, but in DBS by #-marking, as in [Mary gave (#N' D' A' V)].

It turned out that such ATNs do not provide the hoped for contribution to heuristics. This coincided with a more general problem inherited from classic complexity theory, namely the stand-alone nature of LAG as an algorithm.

6 From LAG to the DBS Hear Mode

LAG and DBS share the time-linear derivation order, but differ in their method of loading the input, which is *holistic* in LAG and *incremental* in DBS. Holistic loading takes a complete sequence as input, e.g., aaabbbccc, and then processes it word by word from left to right. This works for parsing individual sentences in a collection of linguistic examples, but is impractical for texts like a Tolstoy novel.

Processing in a holistic loading system may be illustrated as follows:

6.1 CONNECTING A SENTENCE START TO ITS SUCCESSOR

surface level	aa	abbbbccc⇒	aaa	bbbccc
rule level	r_1:(X)	(a)	⇒ (aX)	$\{r_1, r_2\}$

At the surface level, the next word is added at the end of the current sentence start, regardless of the categorial operation. At the rule level, the category of the sentence start is (X) and the category of the next word is (a). In 3.2, rule r_1 attaches the next word category a at the front end of the sentence start variable (X) as (aX) and calls the rule package $\{r_1, r_2\}$. The rules of the rule package are applied to the resulting sentence start and the new next word in the loaded input sequence.

In the hear mode, the start of parsing is a special case because an initial operation activated by a proplet matching its second input pattern can not find a proplet matching its first input proplet at an empty now front. In a text, the initial composition is unique, but in the linguistic analysis of isolated examples each has one.

For example, the derivation of The dog barked. as an isolated linguistic example begins with the recognition of the surface The. Based on matching a type on raw data provided by the input component, it serves as input to automatic word form recognition. The output is stored at the now front, which happens to be empty:

6.2 STORING SENTENCE-INITIAL WORD AT EMPTY NOW FRONT

member proplets	*now front*	owners
	sur: Der noun: N_*n* cat: CN′ NP sem: sg fnc: ... prn: 23	the

Without a next word yet, no operation is activated and the derivation continues with another automatic word form recognition, resulting in the following constellation:

6.3 STORING NEXT WORD AT NOW FRONT

member proplets	*now front*	owners
	sur: Hund noun: dog cat: sn sem: sg fnc: mdr: nc: pc: prn: 23	dog
	...	
	sur: Der noun: N_*n* cat: CN′ NP sem: sg fnc: mdr: nc: pc: prn: 23	the

The token line for storing a next word at the now front is determined alphabetically by the core value, here dog. This supports efficient computational string search (Knuth et al. 1977) for storage and retrieval.

From here on out, the derivation continues in standard fashion. The 'next word,' here a noun, activates all recognition operations which match it with their second input pattern (operations 28–53 in TExer). Activated operations look at the now front for input matching their first input pattern. Those which find one apply.

6.4 ABSORBING *dog* INTO *the* WITH DET∪CN

DET∪CN (h46)

pattern level	noun: N_*n* cat: CN′ NP sem: Y prn: K	noun: α cat: CN sem: Z prn:	⇒	noun: α cat: NP sem: Y Z prn: K
	Agreement conditions	⇑		⇓	
content level	sur: Der noun: n_1 cat: nn′ np sem: def fnc: mdr: nc: pc: prn: 23	sur: Hund noun: dog cat: sn sem: sg fnc: mdr: nc: pc: prn:		sur: noun: dog cat: snp sem: def sg fnc: mdr: nc: pc: prn: 23

The successful application of a hear mode operation triggers the lookup and storage of another 'next word,' here bark, by automatic word form recognition:

6.5 STORING NEXT WORD AT CURRENT NOW FRONT

member proplets	*now front*	owners
	sur: bellte verb: bark cat: ns13′ v sem: ind past arg: … prn: 23	bark
	…	
	sur: noun: dog cat: snp sem: def sg fnc: … prn: 23	dog

The storage of a next word automatically activates all operations which match the next word with their second input pattern (operations 1–27 in TExer).

6.6 CROSS-COPYING *dog* AND bark WITH SBJ×PRD

SBJ×PRD

pattern level	noun: α cat: NP fnc: prn:K	verb: β cat: NP′ X v arg: prn:	⇒	noun: α cat: NP fnc: β prn: K	verb: β cat: #NP′ X v arg: α prn: K

Agreement conditions

⇑ ⇓

content level	sur: noun: dog cat: snp sem: def sg fnc: … prn: 23	sur: bellte verb: bark cat: ns3′ v sem: past arg: … prn:	sur: noun: dog cat: snp sem: def sg fnc: bark … prn: 23	sur: verb: bark cat: #ns3′ v sem: past arg: dog … prn: 23

Depending on the input, the derivation may continue indefinitely, for example by adding interpunctuation and going on to the next sentence, resulting in a content defined as a set (order-free) of proplets connected and ordered by address, e.g., (bark 23). With the start of a new sentence, the accumulation of parallel readings, if any, starts from scratch.

7 From the DBS Hear Mode to the DBS Speak Mode

The counterpart of the DBS hear mode is the speak mode. It rides piggyback on the think mode which activates content by navigating along the semantic relations of structure coded by address. A speak mode derivation is a think mode navigation with the optional production of language-dependent surfaces.

A think-speak mode operation has one input and one output pattern. The operators are $\swarrow$ and $\nearrow$ for the subject/predicate, $\searrow$ and $\nwarrow$ for the object\predicate, $\downarrow$ and $\uparrow$ for the modifier|modified, and $\rightarrow$ and $\leftarrow$ for the conjunct−conjunct relation. Language-dependent surfaces are produced from the goal proplet.

The following speak mode production uses the content derived in 6 as input:

7.1 NAVIGATING WITH V$\swarrow$N FROM *bark* TO *dog*

V$\swarrow$N (s1)

pattern level

$$\begin{bmatrix} \text{verb: } \alpha \\ \text{arg: } \beta\ \text{X} \\ \text{prn: K} \end{bmatrix} \Rightarrow \begin{bmatrix} \text{sur: lexnoun}(\hat{\beta}) \\ \text{noun: } \beta \\ \text{fnc: } \alpha \\ \text{prn: K} \end{bmatrix}$$

#-mark β in the arg slot of proplet α.

$\Uparrow$ $\Downarrow$

content level

$$\begin{bmatrix} \text{sur:} \\ \text{verb: bark} \\ \text{cat: \#n}' \text{ decl} \\ \text{sem: ind past} \\ \text{arg: dog} \\ \text{mdr:} \\ \text{nc:} \\ \text{pc:} \\ \text{prn: 23} \end{bmatrix} \quad \begin{bmatrix} \text{sur: Der Hund} \\ \text{noun: dog} \\ \text{cat: snp} \\ \text{sem: sg m} \\ \text{fnc: bark} \\ \text{mdr:} \\ \text{nc:} \\ \text{pc:} \\ \text{prn: 23} \end{bmatrix}$$

The language-dependent surface is realized from a list which connects relatively language-independent core values to their language-dependent counterpart, here dog $\Rightarrow$ Hund, and interprets the cat value def and the sem value sg as the German definite article form der. The resulting surface is Der Hund.

The next navigation step returns from the subject to the verb:

7.2 NAVIGATING WITH N$\nearrow$V FROM dog BACK TO *bark*

N$\nearrow$V (s2)

pattern level

$$\begin{bmatrix} \text{noun: } \beta \\ \text{fnc: } \alpha\ \text{Y} \\ \text{mdr: Z} \\ \text{prn: K} \end{bmatrix} \Rightarrow \begin{bmatrix} \text{sur: lexverb}(\hat{\alpha}) \\ \text{verb: } \alpha \\ \text{arg: \#}\beta\ \text{Y} \\ \text{prn: K} \end{bmatrix}$$

#-mark α in the fnc slot of proplet β.

Z is NIL, or elementary and #-marked.

$\Uparrow$ $\Downarrow$

content level

$$\begin{bmatrix} \text{sur:} \\ \text{noun: dog} \\ \text{cat: snp} \\ \text{sem: sg m} \\ \text{fnc: bark} \\ \ldots \\ \text{prn: 23} \end{bmatrix} \quad \begin{bmatrix} \text{sur: bellte\_.} \\ \text{verb: bark} \\ \text{cat: \#n}' \text{ decl} \\ \text{sem: ind past} \\ \text{arg: \#dog} \\ \ldots \\ \text{prn: 23} \end{bmatrix}$$

Lexverb uses bark, ind past, and decl to produce the surface bell-te_.. In LAG, the rules to be tried are called by the rule package of the current rule. In DBS, in contrast, the operations to be tried are activated by the next word (data-driven).

8 Incremental Lexical Lookup in the DBS Hear mode

The next word originates as raw data input to a sensor of the agent's interface component and is recognized as a language-dependent surface. The surface is used for lexical lookup, the result of which is stored at the current now front (CC 12.4.4):

8.1 OWNER-BASED STORAGE OF LANGUAGE PROPLET AT NOW FRONT

ii *memory component*

member proplets of A*-memory*	*now front*	*owners*
	sur:	
	verb: bark	⇐ bark
	cat: n′ v	
...	sem: past ind	
	arg:	
	...	
	prn: 37	

⇑

i *interface component*

	recognized surface	*analyzed surface*	
raw input data	⇒ bell-te ⇒	sur: bellte	
		verb: bark	⇒ bark
		cat: n′ v	
		sem: past ind	
		arg:	
		...	
		prn:	

At level (i), a language-dependent word form type matching the raw data results in the recognized surface (⇒) bell-te (here letter sequence). It is used for lexical lookup of the analyzed surface, i.e., the complete proplet, from the allomorph trie structure (CC 12.5.3). Using string search, the core value bark serves (a) to access (⇑) the token line of bark and (b) to store (⇐) the proplet retrieved from the trie structure, without the sur value but with an automatically assigned prn value, at the now front (ii).

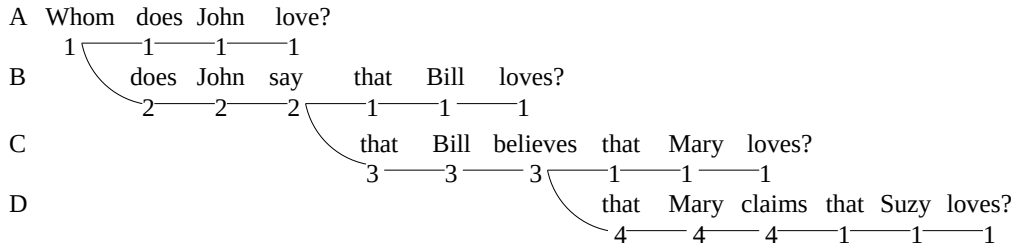
Each activated operation looks at the now front for a proplet matching its first input pattern. In natural language parsing, the number of proplets at the current now front is usually no more than four or five because the now front is cleared whenever a proposition (clause) is completed, indicated by in- or decrementing the prn value (CLaTR$_2$ 11.4.10). After processing, now front clearance leaves the proplets behind as member proplets (loomlike clearance). The now front is cleared by moving it and the owner values one step to the right into fresh memory space.

Content stored as member proplets cannot be changed. The only way to correct is adding new content, like a diary entry referring by address to the content to be corrected. Proplets without an open continuation slot are not tried as input to a first input pattern. The only way to increase the DBS hear mode complexity above linear is a *recursive* ambiguity, which is prevented in natural language by grammatical disambiguation.

9 Ambiguity in Natural Language

There is repeating ambiguity in natural language, but to increase complexity, the readings would have to be [+global] (FoCL 11.3). In natural language, systematically repeating[9] [+global] ambiguity does not seem to exist. Consider, for example, the following derivation structure:[10]

9.1 AMBIGUITY STRUCTURE OF AN UNBOUNDED SUSPENSION



In line A, who(m) is the object of an elementary proposition with a transitive verb which does not take a clausal object. Thus all proplets in line A share the prn value 1. In line B, in contrast, the matrix verb takes a clausal object which who(m) belongs to. Thus, does John say X has the prn value 2, whereby X is Bill loves who(m) with the prn value 1. The construction in line B terminates because the verb *love* does not take a clausal object.

Line C branches off line B because the first object clause uses a verb which takes a second object clause as its oblique argument. Thus, does John say X continues to have the prn value 2, but the new object clause Bill believes Y has the new prn value 3, whereby Y is that Mary loves who(m) with the prn value 1. The construction terminates in branch C.

Line D branches off C because the second object clause uses a verb which takes a third object clause as its argument. Thus, does John say X continues to have the prn value 2, Bill believes Y continues to have the prn value 3, but the new object clause that Mary claims Z has the prn value 4, whereby Z is that Suzy loves who(m) with the prn value 1.

Even though an unbounded suspension (i) may be continued indefinitely and (ii) causes a systematic syntactic ambiguity, it does not increase the computational complexity of natural language (FoCL 11.5). This is because one of the two branches always terminates before the next ambiguity is complete. In other words, there is no global ambiguity in 9.1, in the same sense as there is no global ambiguity in the 'garden path' sentence (FoCL 11.3.6).

Another construction with a repeating local ambiguity is adnominal (aka relative) clauses. See TExer3 Sects. 3.3, 3.4, and 5.6 for complete declarative analyses. As

[9] Systematically repeating (i.e. recursive or iterative) [+global] readings would be a serious impediment to successful communication.

[10] For the complete declarative DBS analysis of this example see TExer 5.5.

long as no natural language can be shown to have repeating global ambiguity, the Linear Complexity Hypothesis for natural language is without counterexample.

10 Language Dependence of Grammatical Disambiguation

Ambiguities are language dependent. For example, the translation of *flying airplanes* into German disambiguates the English readings grammatically into fliegende Flugzeuge and Flugzeuge fliegen. The translation of *horse raced by the barn* into German disambiguates the English readings into Pferd jagte ... vorbei and Pferd das ... vorbeigejagt wurde. The translation of *man who* into German disambiguates the English readings into Man der and Man den. English *Who does John say that ...* doesn't even have a literal translation into German. With English as an isolating and German as an inflectional language, local ambiguities and their grammatical resolution seem to be a typological phenomenon.

11 The Bach-Peters Sentence

The computational undecidability of natural language[11] alleged by Phrase Structure Grammar (PSG) is based on a transformational analysis of the following example:

THE MAN WHO DESERVES *it* WILL GET THE PRIZE *he* WANTS.

The formal proof by Peters and Ritchie (1973)[12] relies on two reciprocal recursions, one deriving the pronoun *it* transformationally from the 'full' noun phrase the prize he wants, the other deriving the pronoun *he* transformationally from the 'full' noun phrase man who deserves it.

The alternative DBS hear mode analysis ($CLaTR_2$ 11.4.9–11.4.12) is of linear time complexity because the coreference between the prize he wants and it is defined by address instead of a transformation, and similarly for the coreference between man who deserves it and he. As in natural language communication, ambiguity in DBS is limited to the hear mode. The speak mode counterpart to hear mode ambiguity is paraphrase. While the hearer's ambiguity may result in multiple simultaneous readings, paraphrase is a matter of choice which depends on the speaker's rhetorical purpose. Each paraphrase is of linear complexity.

Hear mode ambiguity is of two kinds, [+global] and [-global] (FoCL 11.3). Relevant for complexity are only the [+global] ambiguities. Because each reading of length n requires exactly n-1 derivation steps and each derivation step of the DBS C-LAGs are below a grammar-dependent constant C, the computational complexity of the hear mode depends solely on the number of readings.

[11] Classifying natural language as computationally undecidable has been noted to be unlikely by Harman (1963), Gazdar (1981), McCawley (1982), Ross (1986), and many others.

[12] In direct consultation with Prof. Chomsky (personal communication by Bob Ritchie, Stanford 1983).

12 Conclusion

This paper compares the computational complexity of three algorithms for analyzing natural language: (i) the sign-based substitution-driven algorithm of PSG, (ii) the sign-based data-driven algorithm of LAG, and (iii) the agent-based data-driven speak and hear mode algorithms of DBS.

Sign-based PSG and LAG have in common that they do not distinguish between the speak and the hear mode, but differ in their derivation principle, which is substitution-driven in PSG, but data-driven in LAG. More specifically, the input to a derivation in PSG, i.e. ST, EST, REST, GB, GPSG, HPSG, etc., is always the same S node (for start or sentence) and the output is different phrase structures. The input to a LAG derivation, in contrast, is a time-linear sequence of lexical proplets provided by automatic word form recognition, and the output a content in which the lexical proplets are connected by the classical semantic relations of structure, i.e. subject/predicate, object\predicate, modifier|modified, and conjunct−conjunct.

Data-driven LAG and DBS have in common that they compute possible continuations, but differ in that DBS distinguishes between the speak and the hear mode, while LAG does not. In agent-based DBS, the input to the speak mode is a content, defined as a set of proplets connected by the classical semantic relations of structure, and the output a language-dependent surface. The input to the hear mode is a language-dependent surface and the output a content.

Summary:
Ambiguity in natural language communication is limited to hear mode interpretation and may result in multiple simultaneous readings. The speak mode counterpart to ambiguity is paraphrase. For speak mode production, alternative paraphrases require a choice, guided by speaker's rhetorical purpose. Each paraphrase is by principle unambiguous and therefore of linear complexity.

Hear mode ambiguity is of two kinds, [+global] and [-global] (FoCL 11.3). Relevant for computational complexity are the [+global] ambiguities. Because each reading of length n requires exactly n-1 derivation steps and each derivation step of the C-LAGs is by definition below a grammar-dependent constant C, the computational complexity of the DBS hear mode depends solely on the number of readings.

Because recursive ambiguity is prevented by grammatical disambiguation (9.1), the hear mode is of linear complexity, like the speak mode. Therefore the overall computational complexity degree of natural language is linear.

Bibliography

Aho, A. V., and J. D. Ullman (1977) *Principles of Compiler Design*, Reading, MA: Addison-Wesley

AIJ'01 = Hausser, R. (2001) "Database Semantics for Natural Language," *Artificial Intelligence Journal*, Vol. 130.1:283–305, Amsterdam, Elsevier

Bever, T.G (1970) "The cognitive basis for linguistic structures". In: J.R. Hayes (ed.) *Cognition and the development of language*, pp. 279-362, New York: Wiley

CC = Hausser, R. (2019) *Computational Cognition, Integrated DBS Software Design for Data-Driven Cognitive Processing*, pp. 237, `lagrammar.net`

CLaTR = Hausser R. (2011) *Computational Linguistics and Talking Robots; Processing Content in DBS*, pp. 286. Springer (preprint CLaTR$_2$ `lagrammar.net`)

CoL = Hausser, R. (1989) *Computation of Language, An Essay on Syntax, Semantics, and Pragmatics in Natural Man-Machine Communication*, Symbolic Computation: Artificial Intelligence, pp. 425, Springer

Earley, J. (1970) "An Efficient Context-Free Parsing Algorithm," *Commun. ACM*, Vol. 13.2:94–102

FoCL = Hausser, R. (1999) *Foundations of Computational Linguistics, Human-Computer Communication in Natural Language, 3rd ed. 2014*, Springer

Garey, M.R., and D.S. Johnson (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*, San Francisco: W. H. Freeman

Ginsburg, S. (1980) "Formal Language Theory: Methods for Specifying Formal Languages – Past, Present, Future," in R.V. Book (ed.), 1–22

Greibach, S. (1973) "The hardest context-free language," *SIAM J. Comput.* Vol. 2:304–310

Harrison, M. (1978) *Introduction to Formal Language Theory*, pp. 608, Reading, Mass.: Addison-Wesley

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

Knuth, D.E., J.H. Morris, and V.R. Pratt (1977) "Fast Pattern Matching in Strings," *SIAM J. Comput.* Vol. 6.2:323–350

Levelt, W.J.M. (1981) "The speaker's linearization problem," Transactions of the Royal Society, 295.1077:305-315

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

Stubert, B. (1993) *"Einordnung der Familie der C-Sprachen zwischen die kontextfreien und die kontextsensitiven Sprachen,"* CLUE-betreute Studienarbeit der Informatik, Friedrich Alexander Universität Erlangen Nürnberg

TCS'92 = Hausser, R. (1992) "Complexity in left-associative grammar," *Theoretical Computer Science*, Vol. 106.2:283–308, Elsevier