Grammatical Disambiguation

The Linear Complexity Hypothesis for Natural Language

Roland Hausser

Universität Erlangen-Nürnberg (em.)

Abstract

By combining concatenation operations of a constant complexity degree with a strictly time-linear derivation order, the basic computational complexity of DBS is linear time. The only way to increase DBS complexity above linear is repeating ambiguity. In natural language, however, repeating ambiguity is prevented by grammatical disambiguation.

A classic example of a grammatical ambiguity is the 'garden path' sentence The horse raced by the barn fell. The continuation horse+raced introduces an ambiguity between horse raced and horse which was raced, leading to two parallel derivation strands up to The horse raced by the barn . Depending on whether the continuation is interpunctuation or a verb, they are grammatically disambiguated, resulting in unambiguous output.

A repeating ambiguity occurs in The man who loves the woman who feeds Lucy who Peter loves., with the subordinating conjunction who serving as subject or as object. These readings are grammatically disambiguated by continuing after who with either a verb or a noun.

keywords: repeating ambiguity, grammatical disambiguation, stand-alone algorithm, agent-based data-driven, sign-based substitution-driven, 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 elementary operations needed for adding a next input item and (ii) the increase in the number of operations with the increase of n. The basic complexity degrees are a linear, polynomial, exponential, or unbounded increase with the length of the input:[1]

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*
   $n \cdot \infty$

[1] The use of the letter n as a variable for (i) the length of an expression and (ii) within a complexity degree or a formula representing a formal language, e.g. $a^n b^n$, must be distinguished. For example,

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

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*[2] is adding the next word (i.e. the minimum). The respective application numbers are shown for length 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 class, polynomial, but in different complexity classes, linear vs. polynomial, in LAG. In PSG a^nb^n and $a^nb^nc^n$ are in different classes, polynomial vs. exponential, but in the same complexity class, linear, in LAG. The formal language L_{no} is linear in PSG, but exponential (C3) 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 *pairwise inverse* input, like abcd dcba. All formal languages which conform to this structure are computationally tractable, i.e. either linear or polynomial, while those which do not are computationally intractable.[3]

Data-driven LAG, in contrast, favors input which is *not recursively ambiguous*. All formal languages which are recursively ambiguous with a degree greater 2 are computationally intractable, such as 3SAT, SUBSET-SUM, L_{no}, and HCFL (Greibach 1973). Unambiguous languages, such as a^nb^n, $a^nb^nc^n$, $a^nb^nc^nd^n$, etc., a^{2^n}, $a^{n!}$, or single return languages, such as WW^r, WW, and WWW, are computationally tractable.

The two complexity hierarchies may be compared graphically as follows:

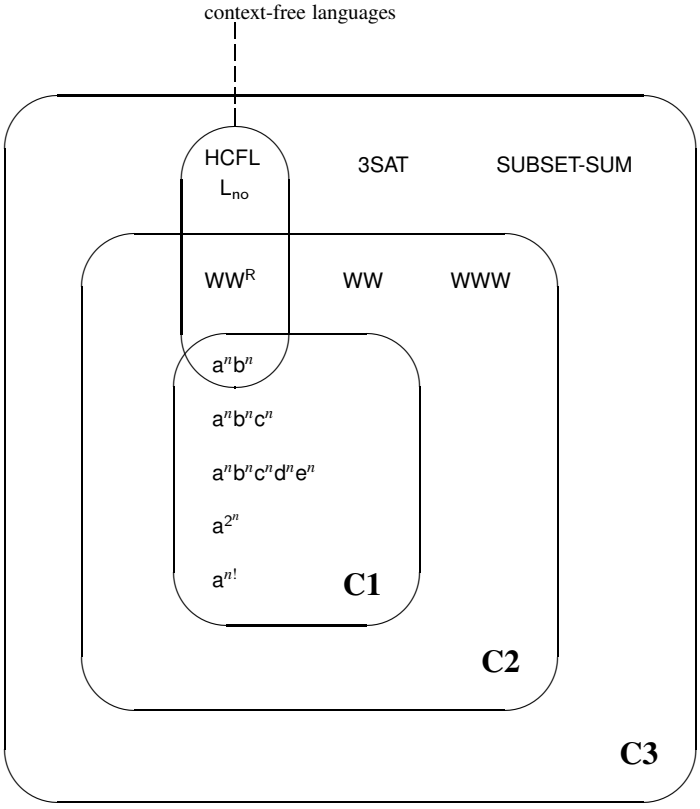
---

setting n in a^nb^n to 3 (n=3) results in aaabbb of length n=6. We distinguish these different but related uses by different fonts: n for formula internal use and n for length.

[2] 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.

[3] It is doubtful that context-free PSG approximates the syntax of programming languages (Harrison 1978, p.219f.; Ginsburg 1980, p.8).

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, polynomial vs. exponential, is not because $a^nb^nc^n$ is particularly complex, but because it is not pairwise.

3 Comparing Explicitly Defined Examples in PSG and DBS

Formal languages which distinguish between PSG and LAG are a^nb^n and $a^nb^nc^n$:

3.1 EXPLICIT PSGS FOR THE FORMAL LANGUAGES a^nb^n AND $a^nb^nc^n$

a^nb^n (*polynomial* n^3)

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

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

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

$a^nb^n\ c^n$ (*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 → B c	*rule 5*
c C → c c}	*rule 6*

The PSG for $a^nb^nc^n$ 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) nodes.

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^nb^n and $a^nb^nc^n$

a^nb^n (*linear*)	$a^nb^nc^n$ (*linear*)
LX =$_{def}$ {[a (a)], [b (b)]}	LX =$_{def}$ {[a (a)], [b (b)], [c (c)]}
ST$_S$ =$_{def}$ {[(a) {r$_1$, r$_2$}]}	ST$_s$ =$_{def}$ {[(a) {r$_1$, r$_2$}]}
r$_1$: (X) (a) ⇒ (aX) {r$_1$, r$_2$}	r$_1$: (X) (a) ⇒ (aX) {r$_1$, r$_2$}
r$_2$: (aX) (b) ⇒ (X) {r$_2$}	r$_2$: (aX) (b) ⇒ (Xb) {r$_2$, r$_3$}
ST$_F$ =$_{def}$ {[ε rp$_2$]}.	r$_3$: (bX) (c) ⇒ (X) {r$_3$}
	ST$_F$ =$_{def}$ {[ε 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)	WW (*exponential*)
V =$_{def}$ {S, a, b, c, d}	Similar to the PSG for $a^nb^nc^n$
V$_T$=$_{def}$ {a, b, c, d}	(3.1), the derivation generates
P =$_{def}$ {S → a S a,	intermediate expressions like aSA,
S → b S b,	abSBA, abcSCBA, etc., and
S → c S c,	then reorders in lower case.
S → d S d,	
S → a a,	
S → b b,	
S → c c,	
S → d d}	

In LAG, in contrast, WW^r and WW are both n^2 polynomial. Rule r$_1$ adds the counterpart letter at the beginning of the category in WW^r, but at the end in WW:

3.4 Explicit LAGs for WW^r and WW

WW^r (*polynomial* n^2)
LX =$_{def}$ {[a (a)], [b (b)], [c (c)], [d (d)] . . . }
ST$_S$ =$_{def}$ {[(seg$_c$) {r$_1$, r$_2$}]}, where seg$_c$ ε {a, b, c, d, . . . }

r_1: (X) (seg_c) $\Rightarrow$ (seg_c X) {r_1, r_2}
r_2: (seg_c X) (seg_c) $\Rightarrow$ (X) { r_2 }
$ST_F =_{def}$ {[ε rp_2] }

WW (*polynomial* n^2)
$LX =_{def}$ {[a (a)], [b (b)], [c (c)], [d (d)] . . . }
$ST_S =_{def}$ {[(seg_c) {r_1, r_2}]}, where seg_c ε {a, b, c, d, . . . }
r_1: (X) (seg_c) $\Rightarrow$ (X seg_c) {r_1, r_2}
r_2: (seg_c X) (seg_c) $\Rightarrow$ (X) { r_2}
$ST_F =_{def}$ {[ε rp_2]}

The recursive ambiguity of WW^r and WW is of the kind *single return*: 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 PSG and LAG complexity with the noise language L_{no}, which is linear in PSG, but exponential in LAG. Devised by D. Applegate, it begins with an arbitrary sequence of 0 and 1, followed by #, 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 till 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 (linear)	**L_{no}** in LAG (exponential)
S → 1S1	LX $=_{def}$ {[0 (0)], [1 (1)], [# (#)]}
S → 1S	$ST_S =_{def}$ {[(seg_c) {r_1, r_2, r_3, r_4, r_5}] },
S → 0S0	where seg ε {0, 1}.
S → 0S	r_1: (seg_c)(seg_d) ⇒ ε {r_1, r_2, r_3, r_4, r_5}
S → #	r_2: (seg_c)(seg_d) ⇒ (seg_d){r_1, r_2, r_3, r_4, r_5}
	r_3: (X)(seg_c) ⇒ (X){r_1 r_2, r_3, r_4, r_5}
	r_4: (X)(seg_c) ⇒ (seg_c X){r_1 r_2, r_3, r_4, r_5}
	r_5: (X)(#) ⇒ (X){r_6}
	r_6: (seg_c X)(seg_c) ⇒ (X) {r_6}
	$ST_F =_{def}$ {[ε 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	—
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 bpractically 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 $a^{2^i} =_{def} \{a^i \mid i \text{ is a positive power of } 2\}$ (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). This language is called *inherently ambiguous* because there is no unambiguous PSG for it.[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.

[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] Hopcroft and Ullman (1979), pp. 99–103.

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]

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

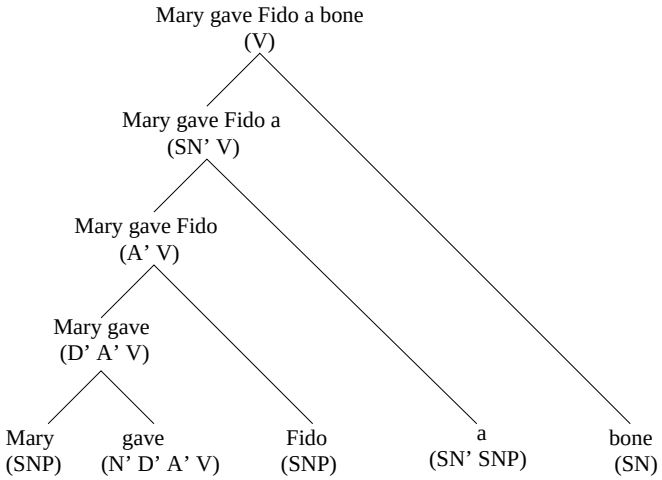
In the application of LAG to natural language, the possible divergence between surfaces and categories is used for defining grammatically motivated categories such as [gave (N′ D′ A′ V)]. Also, the abstract rule names are replaced by grammatically meaningful ones like DET+CN or NP+VERB.

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

[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.

5.1 TIME-LINEAR LAG ANALYSIS OF AN ENGLISH SENTENCE



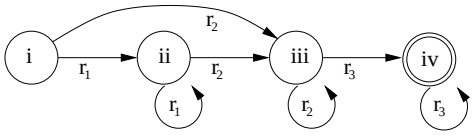
The first word [Mary (SNP)] cancels[8] the valency position N′ in the second word [gave (N′ D′ A′ V)], resulting in the new sentence start [Mary gave (D′ A′ V)] one level above (bottom up). Next the current sentence start [Mary gave (D′ A′ V)] combines with the current next word [Fido (SNP)], resulting 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)].

In NEWCAT'86, this manner of analysis was successfully applied to 221 constructions of German and 114 constructions of English. The goal was to show 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 $a^n b^n c^n$

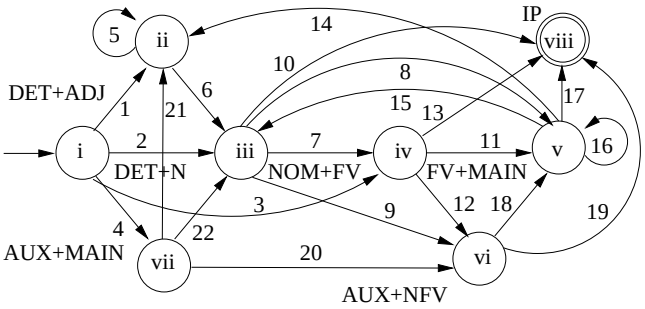


[8] At this stage of the theory, valency positions are canceled by deletion, as in Categorial Grammar. This is not a problem as long as the theory is sign-based. In agent-based DBS, however, the valency information must be preserved for the speak mode by using #-marking for canceling. For example, [Mary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

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 a declarative to a yes-no interrogative, 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 for an extended fragment of English:

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.

It turned out that an ATN like 5.3 does not provide the hoped for contribution to linguistic 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 sign-based system, i.e. a system without the distinction between the speak and the hear mode, but is impractical for long 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 represented as the variable (X) and the category of the next word as the constant (a). In 3.2, the 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 $\{r_1, r_2\}$. Next, the rules of the rule package are applied to the resulting sentence start and the new next word in the loaded input sequence.

Incremental loading, in contrast, models the word form by word form processing of the human hear mode. Lookup of the next word activates all operations matching it with their second input pattern. Activated operations look at the now front for proplets matching their first input pattern. Those which find one apply.

For example, the derivation of the dog barked begins with automatic word form recognition and storage of the first word, here to an empty now front.

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

Because there is no next word, no operation is activated and the derivation continues with another automatic word form recognition:

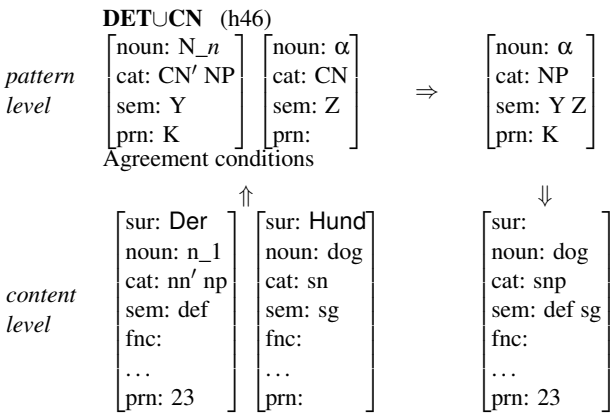
6.3 STORING NEXT WORD AT NOW FRONT

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

The 'next word,' here a noun, activates all recognition operations which match it with their second input pattern (operations 28–53 in TExer3). The 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



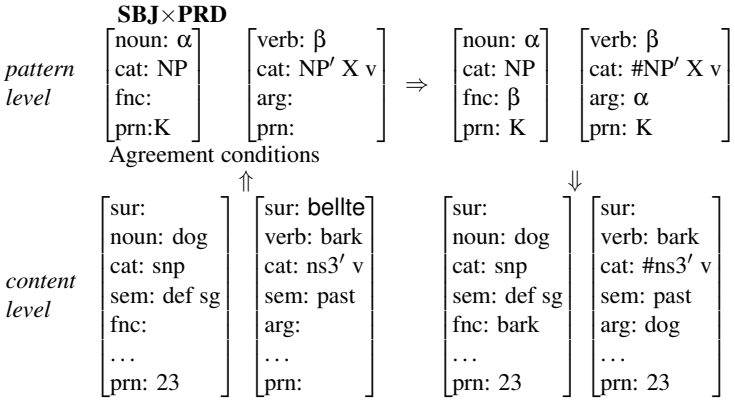
The successful application of a hear mode operation triggers the lookup and storage of another 'next word,' here bark, by automatic word form recognition:

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 TExer3).

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



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

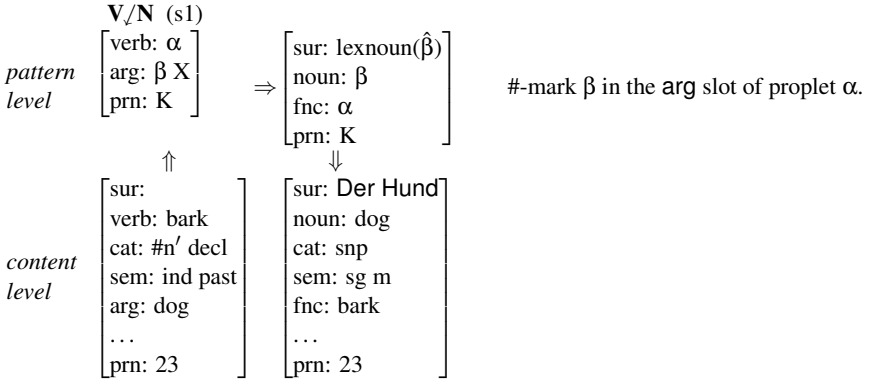
7 From the DBS Hear Mode to the DBS Speak Mode

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 Sect. 6:

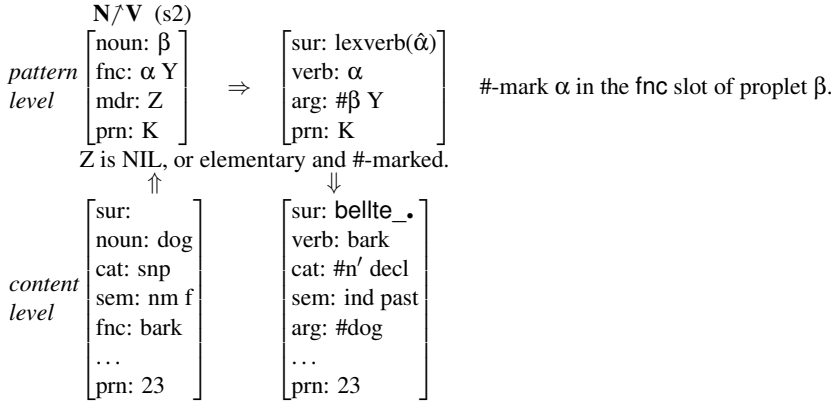
7.1 NAVIGATING WITH V↙N FROM *bark* TO *dog*



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

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

7.2 NAVIGATING WITH N↗V FROM dog BACK TO *bark*

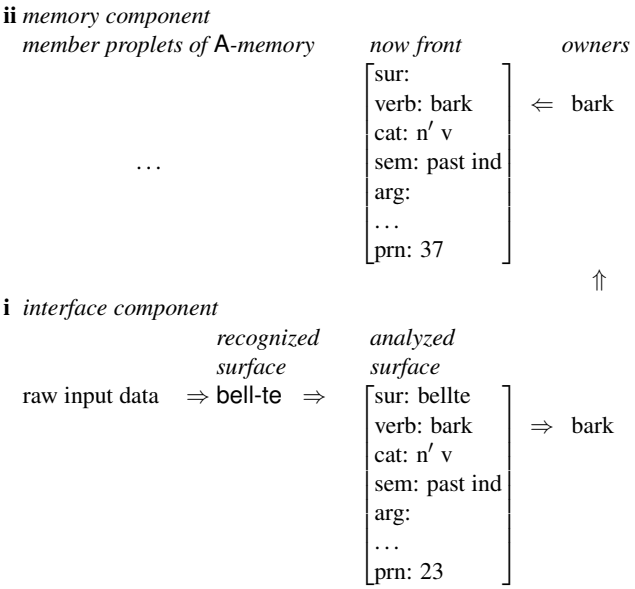


Lexverb uses bark, ind past, and decl to produce the surface bell-te_.. In LAG, the rules to be tried are called by the rule package of the current rule. In DBS, in contrast, the operations to be tried are activated by the next word (data-driven).

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:

8.1 OWNER-BASED STORAGE OF LANGUAGE PROPLET AT NOW FRONT



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, but without the sur 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 (subclause) is completed, indicated by incrementing the prn value. After processing, now front clearance leaves the proplets behind as a 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 in natural language seems to be prevented by grammatical disambiguation. This makes sense because an exponential increase of readings would be a serious obstacle to successful communication.

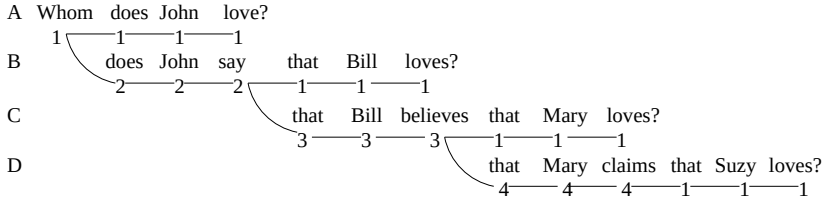
9 Incremental Surface Production in the DBS Speak mode

The input to the agent's speak mode is content resulting from current observation, activated in memory, or derived by inferencing. If there was an ambiguity in the hear mode (laboratory setup or repeated hear say), different readings are treated as different contents and thus as different inputs to the speak mode. In short, there is no ambiguity in the speak mode. It follows that DBS speak mode derivations are of linear complexity.

10 Ambiguity in Natural Language

There is repeating ambiguity in natural language, but to increase complexity, the readings would have to be global (FoCL Sect. 11.3). In natural language, [global recursive] ambiguities do not seem to exist. Consider, for example, the following derivation structure (TExer3 Sect. 5.5.):

10.1 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 Sect. 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 10.1, 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 analysis. As long as no natural language can be shown to have repeating global ambiguity, the Linear Complexity Hypothesis for natural language is without a counterexample.

11 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.

12 Conclusion

This paper compares three methods of determining computational complexity: (i) sign-based substitution-driven stand-alone algorithms of classic computer science,

(ii) sign-based data-driven stand-alone algorithms in LAG, and (iii) agent-based data-driven hear mode grammars in DBS. It is shown that the separate hear and speak modes in agent-based data-driven DBS allow to reduce the computational complexity of natural language from computationally intractible (i.e. undecidable or exponential, as assumed in sign-based substitution-driven ST, EST, REST, GB, LFG, GPSG, and HPSG) to linear. This result is important for the computational reconstruction of communication as a talking robot.

Bibliography

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

CC = Hausser, R. (2019) *Computational Cognition, Integrated DBS Software Design for Data-Driven Cognitive Processing*, pp. i–xii, 1–237, `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* 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*, 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 Journal of Computing Vol. 6.2:323–350

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