PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY B

Phil. Trans. R. Soc. B (2012) **367**, 1956–1970
doi:10.1098/rstb.2012.0077

Research

Formal language theory: refining the Chomsky hierarchy

Gerhard Jäger[1,*] **and James Rogers**[2]

[1]*Department of Linguistics, University of Tuebingen, Wilhelmstrasse 19, Tuebingen 72074, Germany*
[2]*Department of Computer Science, Earlham College, Richmond, IN, USA*

The first part of this article gives a brief overview of the four levels of the Chomsky hierarchy, with a special emphasis on context-free and regular languages. It then recapitulates the arguments why neither regular nor context-free grammar is sufficiently expressive to capture all phenomena in the natural language syntax. In the second part, two refinements of the Chomsky hierarchy are reviewed, which are both relevant to the extant research in cognitive science: the mildly context-sensitive languages (which are located between context-free and context-sensitive languages), and the sub-regular hierarchy (which distinguishes several levels of complexity within the class of regular languages).

Keywords: formal language theory; complexity; artificial grammar learning

1. INTRODUCTION

The field of formal language theory (FLT)—initiated by Noam Chomsky in the 1950s, building on earlier work by Axel Thue, Alan Turing and Emil Post—provides a measuring stick for linguistic theories that sets a minimal limit of descriptive adequacy. Chomsky suggested a series of massive simplifications and abstractions to the empirical domain of natural language. (In particular, this approach ignores meaning entirely. Also, all issues regarding the usage of expressions such as their frequency, context dependence and processing complexity are left out of consideration. Finally, it is assumed that patterns that are productive for short strings apply to strings of arbitrary length in an unrestricted way. The immense success of this framework—influencing not only linguistics to this day but also theoretical computer science and, more recently, molecular biology—suggests that these abstractions were well chosen, preserving the essential aspects of the structure of natural languages.[1]

An *expression* in the sense of FLT is simply a finite string of symbols, and a (*formal*) *language* is a set of such strings. The theory explores the mathematical and computational properties of such sets. To begin with, formal languages are organized into a nested hierarchy of increasing complexity.

In its classical formulation [3], this so-called *Chomsky hierarchy* has four levels of increasing complexity: regular, context-free, context-sensitive and computably enumerable languages. Subsequent work in formal linguistics showed that this fourfold distinction is too coarse-grained to pin down the level of complexity of natural languages along this domain. Therefore, several refinements have been proposed. Of particular importance here are the levels that extend the class of context-free languages (CFLs)—the so-called *mildly context-sensitive languages*—and those that further delimit the regular languages—the *sub-regular hierarchy*.

In this article, we will briefly recapitulate the characteristic properties of the four classical levels of the Chomsky hierarchy and their (ir)relevance to the analysis for natural languages. We will do this in a semi-formal style that does not assume any specific knowledge of discrete mathematics beyond elementary set theory. On this basis, we will explain the motivation and characteristics of the mildly context-sensitive and the sub-regular hierarchies. In this way, we hope to give researchers working in artificial grammar learning (AGL) an iron ration of FLT that helps them to relate experimental work to formal notions of complexity.

2. THE CHOMSKY HIERARCHY

A formal language in the sense of FLT is a set of sequences, or strings over some finite vocabulary Σ. When applied to natural languages, the vocabulary is usually identified with words, morphemes or sounds.[2] FLT is a collection of mathematical and algorithmic tools about how to define formal languages with finite means, and how to process them computationally. It is important to bear in mind that FLT is neither concerned with the meanings of strings, nor with the quantitative/statistical aspects such as the frequency or probability of strings. This in no way suggests that these aspects are not important for the analysis of sets of strings in the real world—this is just not what FLT traditionally is about (even though it is of course possible to extend FLT accordingly—see §7).

To be more specific, FLT deals with formal languages (= sets of strings) that can be defined by finite means,

* Author for correspondence (gerhard.jaeger@uni-tuebingen.de).

One contribution of 13 to a Theme Issue 'Pattern perception and computational complexity'.

even if the language itself is infinite. The standard way to give such a finite description is with a grammar. Four things must be specified to define a grammar: a finite vocabulary of symbols (referred to as *terminals*) that appear in the strings of the language; a second finite vocabulary of extra symbols called *non-terminals*; a special designated non-terminal called the *start symbol*; and a finite set of rules.

From now on, we will assume that when we refer to a grammar $\mathcal{G}$ we refer to a quadruple $\langle \Sigma, NT, S, R \rangle$, where Σ is the set of terminals, NT is the set of non-terminals, S is the start symbol and R is the set of rules. Rules have the form $\alpha \rightarrow \beta$, understood as 'α may be replaced by β', where α and β are strings of symbols from Σ and/or NT. Application of the rule '$\alpha \rightarrow \beta$' to a string means finding a substring in it that is identical to α and replacing that substring by β, keeping the rest the same. Thus, applying '$\alpha \rightarrow \beta$' to $x\alpha y$ produces $x\beta y$.

$\mathcal{G}$ will be said to *generate* a string w consisting of symbols from Σ if and only if it is possible to start with S and produce w through some finite sequence of rule applications. The sequence of modified strings that proceeds from S to w is called a *derivation* of w. The set of all strings that $\mathcal{G}$ can generate is called the *language* of $\mathcal{G}$, and is notated $L(\mathcal{G})$.

The question whether a given string w is generated by a given grammar $\mathcal{G}$ is called the *membership problem*. It is *decidable* if there is a Turing machine (or an equivalent device, i.e. a computer program running on a machine with unlimited memory and time resources) that answers this question with 'yes' or 'no' in finite time. A grammar $\mathcal{G}$ is called *decidable* if the membership problem is decidable for every string of terminals of that grammar. In a slight abuse of terminology, a language is called decidable if it has a decidable grammar. A class of grammars/languages is called decidable if and only if all its members are decidable.

(a) *Computably enumerable languages*

The class of all languages that can be defined by some formal grammar is called *computably enumerable*. It can be shown that any kind of formal, algorithmic procedure that can be precisely defined can also be expressed by some grammar—be it the rules of chess, the derivations of logic or the memory manipulations of a computer program. In fact, any language that can be defined by a Turing machine (or an equivalent device) is computably enumerable, and vice versa.

All computably enumerable languages are *semi-decidable*. This means that there is a Turing machine that takes a string w as input and outputs the answer 'yes' if and only if w is generated by $\mathcal{G}$. If w is not generated by $\mathcal{G}$, then the machine either outputs a different answer or it runs forever.

Examples of languages with this property are the set of computer programs that halt after a finite number of steps (simply compile the program into a Turing machine and let it run, and then output 'yes' if the program terminates), or the set of provable statements of first-order logic. (A Turing machine can systematically list all proofs of theorems one after the other; if the last line of the proof equals the string in question: output 'yes'; otherwise, move on to the next proof.)

(b) *Context-sensitive languages*

Context-sensitive grammars[3] are those grammars where the left-hand side of each rule (α) is never longer than the right-hand side (β). Context-sensitive languages are then the languages that can be defined by some context-sensitive grammar. The definition of this class of grammars immediately ensures a decision procedure for the membership problem. Starting from a string in question w, there are finitely many ways in which rules can be applied backward to it. None of the resulting strings is longer than w. Repeating this procedure either leads to shorter strings or to a loop that need not be further considered. In this way, it can be decided in finite time whether w is derivable from S.

Even though the question whether or not a given string w is generated by a given context-sensitive grammar $\mathcal{G}$ is in principle decidable, computing this answer may be algorithmically so complex that it is, for practical purposes, intractable.[4]

It should be noted that there are decidable languages that are not context-sensitive (even though they do not have any practical relevance in connection with natural languages).

Examples of context-sensitive languages (that are not context-free) are as follows (we follow the common notation where x^i denotes a consecutive string of symbols that contains exactly i repetitions of the string x):

- the set of all prime numbers (where each number n is represented by a string of length n);
- the set of all square numbers;
- the *copy language*, i.e. the set of all strings over Σ that consist of two identical halfs;
- $a^n b^m c^n d^m$;
- $a^n b^n c^n$; and
- $a^n b^n c^n e^n f^n$.

(c) *Context-free languages*

In a context-free grammar, all rules take the form

$$A \rightarrow \beta,$$

where A is a single non-terminal symbol and β is a string of symbols.[5] CFLs are those languages that can be defined by a context-free grammar.

Here, the non-terminals can be interpreted as names of syntactic categories, and the arrow '$\rightarrow$' can be interpreted as 'consists of'. Therefore, the derivation of a string x in such a grammar implicitly imposes a hierarchical structure of x into ever larger sub-phrases. For this reason, context-free grammars/languages are sometimes referred to as phrase structure grammars/languages, and it is assumed that such languages have an intrinsic hierarchical structure.

As hierarchical structure is inherent in many sequential phenomena in biology and culture—from problem solving to musical structure—context-free grammars are a very versatile analytical tool.

It is important to keep in mind though that a CFL (i.e. a set of strings) does not automatically come equipped with an intrinsic hierarchical structure. There may be several grammars for the same language that impose entirely different phrase structures.

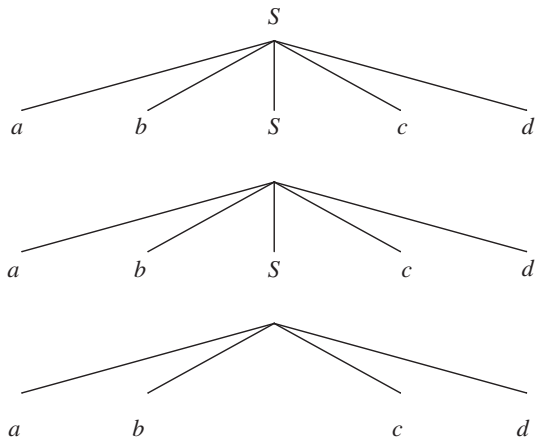


Figure 1. Phrase structure tree.

This point can be illustrated with the language $(ab)^n(cd)^n$. A simple grammar for it has only two rules:

— $S \rightarrow abScd$; and
— $S \rightarrow abcd$.

The derivation for the string *abababcdcdcd* can succinctly be represented by the *phrase structure tree* given in figure 1. In such a tree diagram, each local tree (i.e. each node together with the nodes below it that are connected to it by a direct line) represents one rule application, with the node on top being the left-hand side and the node on the bottom, the right-hand side. The sequence that is derived can be read off the *leaves* (the nodes from which no line extends downwards) of the tree.

The same language can also be described by a somewhat more complex grammar, using the rules:

— $S \rightarrow aTd$;
— $T \rightarrow bSc$; and
— $T \rightarrow bc$.

According to this grammar, the phrase structure tree for *abababcdcdcd* comes out as given in figure 2.

So both grammars impose a hierarchical structure on the string in question, but these structures differ considerably. It is thus important to keep in mind that phrase structures are tied to particular grammars and need not be intrinsic to the language as such.

Natural languages often provide clues about the hierarchical structure of their sentences beyond the plain linear structure. (Intonation, semantic coherence, morphological agreement and relative syntactic independence are frequently used criteria for a sub-string to be considered a coherent hierarchical unit.) Therefore most linguists require a grammar not just to generate the correct set of strings to be adequate; rather, it must also assign a plausible phrase structure.

The membership problem for CFLs is solvable in cubic time, i.e. the maximum time that is needed to decide whether a given string x belongs to $L(\mathcal{G})$ for some context-free grammar $\mathcal{G}$ grows with the third power of the length of x. This means that there are efficient algorithms to solve this problem.

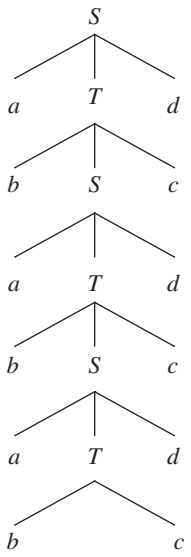


Figure 2. Different phrase structure tree for the same string as in figure 1.

Examples of (non-regular) CFLs are given in the left column of table 1. Where appropriate, minimally differing examples for a non-context-free language (that are all context-sensitive) are given in the right column for contrast.

(d) *Regular languages*

Regular languages are those languages that are defined by regular grammars. In such a grammar, all rules take one of the following two forms:

$A \rightarrow a$,
$A \rightarrow aB$.

Here, A and B denote non-terminal symbols and a a terminal symbol.[6]

As regular grammars are also context-free, the non-terminals can be seen as category symbols and the arrow as 'consists of'. According to another natural interpretation, non-terminals are the names of the *states of an automaton*. The arrow '$\rightarrow$' symbolizes possible state transitions, and the terminal on the right-hand side is a symbol that is emitted as a side effect of this transition. The start symbol S designates the initial state, and rules without a non-terminal on the right-hand side represent transitions into the final state. As there are finitely many non-terminals, a regular grammar thus describes a *finite state automaton* (FSA). In fact, it can be shown that each FSA can be transformed into one that is described by a regular grammar without altering the language that is being described. Therefore, regular grammars/languages are frequently referred to as *finite state* grammars/languages.

The membership problem for regular languages can be solved in linear time, i.e. the recognition time grows at most proportionally to the length of the string in question. Regular languages can thus be processed computationally in a very efficient way.

Table 2 gives some examples of regular languages in the left column. They are contrasted to similar non-regular (context-free) languages in the right column (figure 3).

Table 1. Context-free and non-context-free languages

context-free languages	non-context-free languages
mirror language (i.e. the set of strings xy over a given Σ such that y is the mirror image of x)	*copy language* (i.e. the set of strings xx over a given Σ such that x is an arbitrary string of symbols from Σ)
palindrome language (i.e. the set of strings x that are identical to their mirror image)	
$a^n b^n$	$a^n b^n c^n$
$a^n b^m c^m d^n$	$a^n b^m c^n d^m$
well-formed programs of Python (or any other high-level programming language)	
Dyck language (the set of well-nested parentheses)	
well-formed arithmetic expression	

Table 2. Regular and non-regular languages.

regular languages	non-regular languages
$a^n b^m$	$a^n b^n$
the set of strings x such that the number of 'a's in x is a multiple of 4	the set of strings x such that the number of 'a's and the number of 'b's in x are equal
the set of natural numbers that leave a remainder of 3 when divided by 5	

As the examples illustrate, regular grammars are able to count up to a certain number. This number may be arbitrarily large, but for each regular grammar there is an upper limit for counting. No regular grammar is able to count two sets of symbols and compare their size if this size is potentially unlimited. As a consequence, $a^n b^n$ is not regular.

The full proof of this fact goes beyond the scope of this overview article, and the interested reader is referred to the literature cited. The crucial insight underlying this proof is quite intuitive though, and we will provide a brief sketch.

For each regular grammar $\mathcal{G}$, it is possible to construct an algorithm (a FSA) that reads a string from left to right, and then outputs 'yes' if the string belongs to $L(\mathcal{G})$, and 'no' otherwise. At each point in time, this algorithm is in one of $k+1$ different states, where k is the number of non-terminals in $\mathcal{G}$. Suppose, for a *reductio ad absurdum*, that $L = a^n b^n$ is a regular language, and let $\mathcal{G}^*$ be a regular grammar that recognizes L and that has k^* non-terminals. Then the corresponding recognition algorithm has k^*+1 different states. Now let i be some number more than k^*+1. According to the assumption, $a^i b^i$ belongs to $L(\mathcal{G})$. When the recognition algorithm reads in the sequence of 'a's at the beginning of the string, it will visit the same state for the second time after at most k^*+1 steps. So a sequence of i consecutive 'a's will be indistinguishable for the algorithm from a sequence of $i-k'$ consecutive 'a's, for some positive $k' \leq k^*+1$. Hence, if the algorithm accepts the string $a^i b^i$, it will also accept the string $a^{i-k'} b^i$. As this string does not belong to $a^n b^n$, the algorithm does not accept all and only the elements of $a^n b^n$, contra assumption. Therefore $a^n b^n$ cannot be a regular language.

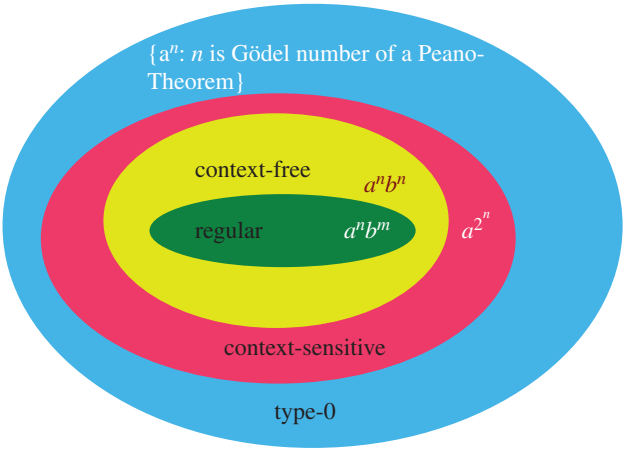


Figure 3. Chomsky hierarchy.

As mentioned above, each regular language corresponds to some FSA, i.e. an algorithm that consumes one symbol at a time and changes its state according to the symbol consumed. As the name suggests, such an automaton has finitely many states. Conversely, each FSA can be transformed into a regular grammar $\mathcal{G}$ such that the automaton accepts all and only the strings in $L(\mathcal{G})$.

The other levels of the Chomsky hierarchy likewise each correspond to a specific class of automata. Context-free grammars correspond to FSAs that are additionally equipped with a *pushdown stack*. When reading an input symbol, such a machine can—next to changing its state—add an item on top of a stack or remove an item from the top of the stack.

Context-sensitive grammars correspond to *linearly bounded automata*. These are essentially Turing machines, i.e. FSAs with a memory tape that can perform arbitrary operations (writing and erasing symbols on the tape and moving the tape in either direction) during state transitions. The length of the available tape is not infinite, though, but bounded by a number that is a linear function of the length of the input string.

Finally, type-0 grammars correspond to unrestricted Turing machines.

3. WHERE ARE NATURAL LANGUAGES LOCATED?

The issue of where natural languages are located within this hierarchy has been an open problem over decades. Chomksky [4] pointed out already in the 1950s that English is not a regular language, and this argument probably carries over to all other natural languages. The crucial insight here is that English has *centre embedding* constructions. These are constructions

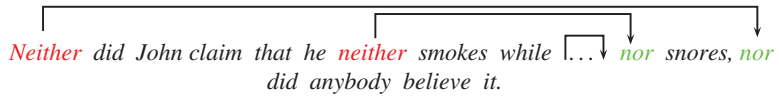


Figure 4. Nested dependencies in English.

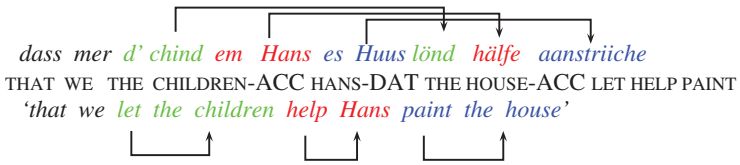


Figure 5. Cross-serial dependencies in Swiss German.

involving two dependent elements *a* and *b* that are not adjacent, and that may contain another instance of the same construction between the two parts. An example is a *neither–nor* construction, as illustrated in figure 4. The pair-wise dependencies between *neither* and *nor* are nested. As far as the grammar of English goes, there is no fixed upper bound on the number of levels of embedding.[7] Consequently, English grammar allows for a potentially *unlimited number* of nested dependencies of *unlimited size*. Regular grammars are unable to recognize this kind of unlimited dependencies because this involves counting and comparing. As mentioned at the end of the previous section, regular languages cannot do this.

The issue of whether all natural languages are context-free proved to be more tricky.[8] It was finally settled only in the mid-1980s, independently by the scholars Riny Huybregts [5], Stuart Shieber [6] and Christopher Culy [7]. Huybregts and Shieber use essentially the same argument. They notice that the dependencies between verbs and their objects in Swiss German are unbounded in length. However, they are not nested, but rather interleaved so that they cross each other. An example (taken from [6]) is given in figure 5.

Here, the first in a series of three article–noun phrases (*d'chind* 'the child') is the object of the first verb, *lönd* 'let' (*lönd* requires its object to be in accusative case and *d'chind* is in accusative); the second article–noun phrase (*em Hans*, 'Hans', carrying dative case) is the object of the second verb (*hälfe* 'help', which requires its object to be in dative case) and the third article–noun phrase (*es Huus* 'the house', accusative case) is the object of the third verb (*aanstriiche* 'paint', which requires an accusative object). In English, as shown in the glosses, each verb is directly adjacent to its object, which could be captured even by a regular grammar. Swiss German, however, has *crossing dependencies* between objects and verbs, and the number of these interlocked dependencies is potentially unbounded. Context-free grammars can only handle an unbounded number of interlocked dependencies *if they are nested*. Therefore Swiss-German cannot be context-free. Culy makes a case that the rules of word formation in the West-African language Bambara conspire to create unbounded crossing dependencies as well, thus establishing the non-context-freeness of this language of well-formed words.

Simple toy languages displaying the same structural properties are the copy language—where each grammatical string has the form *www* for some arbitrary string *w*, and this creates dependencies of the corresponding symbols in the first and the second half of the string—and $a^n b^m c^n d^m$, where the dependencies between the '*a*'s and the '*c*'s include an unlimited number of open dependencies reaching from the '*b*'s to the '*d*'s. Therefore, both languages are not context-free.

4. MILDLY CONTEXT-SENSITIVE LANGUAGES

After this brief recapitulation of the 'classical' Chomsky hierarchy, the rest of the paper will review two extensions that have proved useful in linguistics and cognitive science. The first one—dealt with in this section—considers levels between context-free and context-sensitive languages, so-called *mildly context-sensitive* languages. The following section is devoted to the *subregular hierarchy*, a collection of language classes that are strictly included in the regular languages.

Since the 1980s, several attempts have been made to design grammar formalisms that are more suitable for doing linguistics than the rewrite grammars from the Chomsky hierarchy, while at the same time approximating the computational tractability of context-free grammars. The most prominent examples are Joshi's [8] *tree adjoining grammar* (TAG) and Mark Steedman's *combinatory categorial grammar* [9,10]. In 1991, Joshi *et al.* [11] proved that four of these formalisms (the two already mentioned ones and Gerald Gazdar's [12] *linear indexed grammars* and Carl Pollard's [13] *head grammars*) are equivalent, i.e. they describe the same class of languages. A series of related attempts to further extend the empirical coverage of such formalisms and to gain a deeper understanding of their mathematical properties converged to another class of mutually equivalent formalisms (including David Weir's [14] *linear context-free rewrite systems* and *set-local multi-component TAGs*, and Ed Stabler's [15] formalization of Noam Chomsky's [16] *minimalist grammars* (MGs)) that describe an even larger class of formal languages. As there are no common terms for these classes, we will refer to the smaller class as TAG languages and the larger one as MG languages.

The membership problem for TAG languages is $\mathcal{O}(n^6)$, i.e. the time that the algorithm takes grows with the 6th power of the length of the string in question. Non-CFLs that belong to the TAG languages are, for instance:

— $a^n b^m c^n d^m$;
— the copy language;
— $a^n b^n c^n$; and
— $a^n b^n c^n d^n$.

The descriptive power of TAG languages is sufficient to capture the kind of crossing dependencies that are observed in Swiss German and Bambara.[9]

MGs (and equivalent formalisms) are still more powerful than that. While TAG languages may only contain up to four different types of interlocked unlimited (crossing or nesting) dependencies, there is no such upper bound for MG languages. To be more precise, each MG language has a finite upper bound for the number of different types of dependencies, but within the class of MG languages this bound may be arbitrarily large. This leads to a higher computational complexity of the membership problem. It is still always polynomial, but the highest exponent of the term may be arbitrarily large.

Becker *et al.* [18] argue that this added complexity is actually needed to capture all aspects of word order variation in German.

Non-TAG languages that are included in the MG languages are, for instance,

— $a_1^n \dots a_k^n$ for arbitrary k and
— w^k for arbitrary k, i.e. the k-copy language for any k.

Joshi [8] described a list of properties that an extension of the CFLs should have if it is to be of practical use for linguistics:

— It contains all CFLs.
— It can describe a limited number of types of cross-serial dependencies.
— Its membership problem has polynomial complexity.
— All languages in it have *constant growth property.*

With regard to the last property, let L be some formal language, and let l_1, l_2, $l_3, \dots$ be the strings in L, ordered according to length. L has the *constant growth property* if there is an upper limit for the difference in length between two consecutive elements of this list. The motivation for this postulate is that in each natural language, each sentence can be extended to another grammatical sentence by adding a single word (like an adjective or an adverb) or another short conjunct. Hence there cannot be arbitrarily large gaps in the list of possible lengths the sentences of a language can have.

This postulate excludes many context-sensitive languages, such as the set of square numbers, the set of prime numbers or the set of powers of 2, etc.

Joshi calls classes of languages with these properties *mildly context-sensitive* because they extend the CFLs, but only slightly, preserving many of the 'nice' features of the CFLs. Both TAG languages and MG languages are mildly context-sensitive classes in this sense.

The refinement of the Chomsky hierarchy that emerges from this line of research is displayed in figure 6.

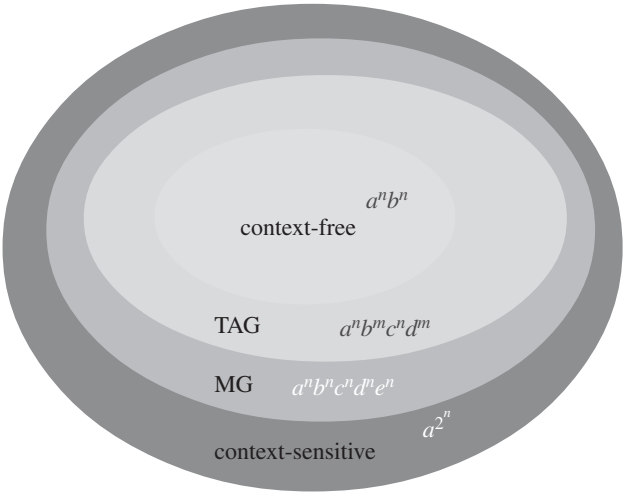


Figure 6. The mildly context-sensitive sub-hierarchy.

It should be noted that Michaelis & Kracht [19] present an argument that Old Georgian is not an MG language. This conclusion only follows, though, if a certain pattern of complex case marking of this language is applicable recursively without limit. Of course, this issue cannot be settled for a dead language, and so far the investigation of living languages with similar properties has remained inconclusive. Most experts therefore assume at this time that all natural languages are MG languages.

5. COGNITIVE COMPLEXITY

Classes of the Chomsky hierarchy provide a measure of the complexity of patterns based on the structure of the mechanisms (grammars, automata) that can distinguish them. But, as we observed in §2*c*, these mechanisms make judgements about strings in terms of specific analyses of their components. When dealing with an unknown mechanism, such as a cognitive mechanism of an experimental subject, we know nothing about the analyses they employ in making their judgements, we know only that they can or cannot make these judgements about strings correctly.

The question for AGL, then, is what characteristics of the formal mechanisms are shared by the physical mechanisms employed by an organism when it is learning a pattern. What valid inferences may one make about the nature of an unknown mechanism that can distinguish the same sorts of patterns?

Here, the grammar- and automata-theoretic characterizations of the Chomsky hierarchy are much less useful. As we saw in §§2*d* and 4, mechanisms with widely differing natures often turn out to be equivalent in the sense that they are capable of describing exactly the same class of languages.[10]

Mechanisms that can recognize arbitrary CFLs are not limited to mechanisms that analyse the string in a way analogous to a context-free grammar. Dependency grammars [20], for example, analyse a string in terms of a binary relation over its elements, and there is well-defined class of these grammars that can distinguish all and only the CFLs. In learning a CFL, it is not necessary to analyse it in terms

of immediate constituency as it is formalized by context-free grammars.

Similarly, within each of the levels of the Chomsky hierarchy there are classes of languages that do not require the full power of the grammars associated with that level. The language $a^n b^n$, for example, while properly context-free, can be recognized by a FSA that is augmented with a simple counter. The languages $a^n b^m c^m d^n$ and $a^n b^n$ with explicit nested dependencies cannot. On the other hand, these can still be recognized by mechanisms that are simpler than those that are required to recognize CFLs in general.

So what can one say about a mechanism that can learn a properly context-free pattern? For one thing, it is not finite-state: that is, there is no bound, independent of the length of the string, on the quantity of information that it must infer in making a judgement about whether the string fits the pattern. Beyond that, there is very little if anything that we can determine about the nature of that information and how it is used simply from the evidence that an organism can learn the pattern.[11]

The situation is not hopeless, however. No matter how complicated the information inferred by a mechanism in analysing a string, it must be based on recognizing simple patterns that occur in the string itself. One can, for example, identify the class of patterns that can be recognized simply on the basis of the adjacent pairs of symbols that occur in the string. Any mechanism that is based, in part, on that sort of information about the string will need at least to be able to distinguish patterns of this sort.

In §6, we introduce a hierarchy of language-theoretic complexity classes that are based on this sort of distinction: what relationships between the symbols in the string a mechanism must be sensitive to (to attend to) in order to distinguish patterns in that class. Since they are based solely on the relationships that are explicit in the strings themselves, these classes are fundamental: *every* mechanism that can recognize a pattern that is properly in one of these classes must *necessarily* be sensitive to the kinds of relationships that characterize the class.

On the other hand, the fact that they are defined in terms of explicit relationships in the string itself also implies that they are all finite-state. But they stratify the finite-state languages in a way that provides a measure of complexity that is independent of the details that may vary between mechanisms that can recognize a given pattern, one that does not depend on *a priori* assumptions about the nature of the mechanism under study. Because this is a notion of complexity that is necessarily relevant to cognitive mechanisms, and because the relative complexity of patterns is invariant over the range of equivalent mechanisms (a property not shared by measures like, for example, minimum description length), it provides a useful notion of cognitive complexity.

This is a notion of complexity that is particularly useful for AGL: the patterns are relatively simple, and are therefore relatively practical to test and they provide information about the capabilities of the organisms that is relevant, regardless of what additional capabilities it may have that enable it to learn more complicated patterns.

There are analogous hierarchies of classes that are based on relationships that are explicit in trees and in more complicated tree-like structures that stratify the CFLs and a range of mildly context-sensitive languages [21]. These, do, however, apply only to mechanisms that analyse strings in terms of tree-like partial orders.

In §6, we survey a hierarchy of complexity classes that is based on adjacency within a string, the so-called local hierarchy [22]. There is a parallel hierarchy that is based on precedence (over arbitrary distances) that distinguishes long-distance relationships within the string, including many that are relevant to a broad range of aspects of human languages—including some, but clearly not all, long-distance relationships in syntax. More details of this hierarchy can be found in [23].

6. SUBREGULAR LANGUAGES

A subregular language is a set of strings that can be described without employing the full power of FSAs. Perhaps a better way of thinking about this is that the patterns that distinguish the strings that are included in the language from those that are not can be identified by mechanisms that are simpler than FSAs, often much simpler.

Many aspects of human language are manifestly subregular, including most 'local' dependencies and many 'non-local' dependencies as well. While these phenomena have usually been studied as regular languages, there are good reasons to ask just how much processing power is actually needed to recognize them. In comparative neurobiology, for example, there is no reason to expect non-human animals to share the full range of capabilities of the human language faculty. Even within human cognition, if one expects to find modularity in language processing, then one may well expect to find differences in the capabilities of the cognitive mechanisms responsible for processing the various modules. Similarly, in cognitive evolution one would not generally expect the antecedents of the human language faculty to share its full range of cognitive capabilities; we expect complicated structures to emerge, in one way or another, from simpler ones.

The hierarchy of language classes we are exploring here are characterized *both* by computational mechanisms (classes of automata and grammars) and by model-theoretic characterizations: characterizations in terms of logical definability by classes of logical expressions. The computational characterizations provide us with the means of designing experiments: developing practical sets of stimuli that allow us to probe the ability of subjects to distinguish strings in a language in a given class from strings that are not in that language and which suffice to resolve the boundaries of that class. The model-theoretic characterizations, because of their abstract nature, allow us to draw conclusions that are valid for all mechanisms which are capable of making those distinctions. It is the interplay between these two ways of characterizing a class of languages that provides a sound foundation for designing AGL experiments and

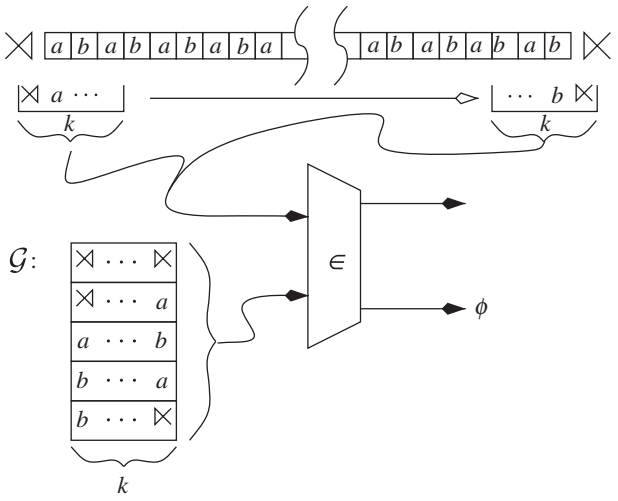


Figure 7. Scanners have a sliding window of width k, a parameter, which moves across the string stepping one symbol at a time, picking out the k-factors of the string. For SL languages the string is accepted if and only if each of these k-factors is included in a look-up table.

interpreting their results. *Both* types of characterizations are essential to this enterprise.

(a) *Strictly local languages*

We begin our tour of these language classes at the lowest level of complexity that is not limited to languages of finitely bounded size, patterns which depend solely on the blocks of symbols which occur consecutively in the string, with each block being considered independently of the others. Such patterns are called *strictly local* (SL).[12]

An SL_k definition is just a set of blocks of k adjacent symbols (called *k-factors*) drawn from the vocabulary augmented with two symbols, '⋊' and '⋉', denoting the beginning and end of the string, respectively. A string satisfies such a description if and only if every k-factor that occurs in the string is licensed by the definition. The SL_2 description $\mathcal{G}_{(AB)^n} = \{\rtimes A, AB, BA, B\ltimes\}$, for example, licenses the set of strings of the form $(AB)^n$.[13]

Abstract processing models for local languages are called *scanners* (figure 7). For strictly k-local languages, the scanner includes a look-up table of k-factors. A string is accepted if and only if every k-factor which occurs in the string is included in the look-up table. The look-up table is formally identical to an SL_k description. These automata have no internal state. Their behaviour, at each point in the computation, depends only on the symbols which fall within the window at that point. This implies that every SL_k language will be closed under substitution of suffixes in the sense that, if the same k-factor occurs somewhere in two strings that are in the language, then the result of substituting the suffix, starting at that shared k-factor, of one for the suffix of the other must still be in the language.

Both the SL_k descriptions and the strictly k-local scanners are defined solely in terms of the length k blocks of consecutive symbols that occur in the string, taken in isolation. This has a number of implications for cognitive mechanisms that can recognize SL languages:

— Any cognitive mechanism that can distinguish member strings from non-members of an SL_k language must be sensitive to, at least, the length k blocks of consecutive symbols that occur in the presentation of the string.
— If the strings are presented as sequences of symbols in time, then this corresponds to being sensitive, at each point in the string, to the immediately prior sequence of $k - 1$ symbols.
— Any cognitive mechanism that is sensitive *only* to the length k blocks of consecutive symbols in the presentation of a string will be able to recognize *only* SL_k languages.

Note that these mechanisms are *not* sensitive to the k-factors which *do not* occur in the string.

(b) *Probing the SL boundary*

In order to design experiments testing an organism's ability to recognize SL languages, one needs a way of generating sets of stimuli that sample languages that are SL and sets that sample languages that are minimally non-SL. (We return to these issues in §9.) This is another place in which computational characterizations of language classes are particularly useful. The language of strings of alternating symbols (e.g. 'A's and 'B's: $(AB)^n$), for example, is SL_2. Mechanisms that are sensitive to the occurrence of length 2 blocks of consecutive symbols are capable, in principle, of distinguishing strings that fit such a constraint (e.g. $(AB)^{i+j+1}$, for some i and j) from those that do not (e.g. $(AB)^iAA(AB)^j$). The ability to do so can be tested using sets of strings that match these patterns.[14]

Conversely, the language of strings in which some symbol (e.g. 'B') is required to occur at least once is not SL_k for any k. (We refer to this language as some-B.) Mechanisms that are sensitive only to the occurrence of fixed size blocks of consecutive symbols are incapable of distinguishing strings that meet such a constraint from those that do not. Thus these organisms would not, all other things being equal, recognize that stimuli of the form A^{i+j+1} do not belong to a language correctly generalized from sets of stimuli of the form A^iBA^j.

(c) *Locally* k-*testable languages*

Notice that if the B were forbidden rather than required, the second pattern would be a SL (even SL_1) property. So we could define a property requiring *some* B as the *complement*—the language that contains all and only the strings that do not occur in that language—of an SL_1 language. In this case, we take the complement of the set of strings in which B does not occur. If we add complement to our descriptions, it turns out that our descriptions will be able to express all Boolean operators: *conjunction* (and), *disjunction* (or) and *negation* (not) in any combination.

In order to do this, we will interpret k-factors as atomic (unanalysed) properties of strings; a string *satisfies* a k-factor if and only if that factor occurs

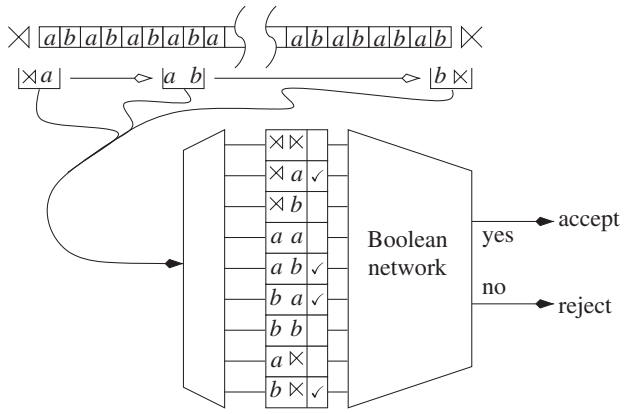


Figure 8. LT automata keep a record of which k-factors occur in a string and feed this information into a Boolean network. The automaton accepts if, once the entire string has been scanned, the output of the network is 'yes', and rejects otherwise.

somewhere in the string. We can then build descriptions as expressions in a *propositional logic* over these atoms. We refer to formulae in this logic as k-expressions. A k-expression defines the set of all (and only) strings that satisfy it. A language that is definable in this way is a *locally k-testable* (LT_k) language. The class of languages that are definable by k-expressions for any finite k is denoted LT.

By way of example, we can define the set of strings which do not start with A and contain at least one B with the 2-expression: $(\neg{\rtimes}A)\wedge B$.

Note that any SL_k definition $\mathcal{G}$ can be translated into a k-expression which is the conjunction $\neg f_1 \wedge \neg f_2 \wedge \cdots$ in which the f_i are the k-factors which are *not* included in $\mathcal{G}$. SL_k definable constraints are, in essence, conjunctions of negative atomic constraints and every such constraint is LT_k definable: SL is a proper subclass of LT.

A scanner for an LT_k language contains, instead of just a look-up table of k-factors, a table in which it records, for every k-factor over the vocabulary, whether or not that k-factor has occurred somewhere in the string. It then feeds this information into a combinatorial (Boolean) network which implements some k-expression. When the end of the string is reached, the automaton accepts or rejects the string depending on the output of the network. (See figure 8.)

Since an LT_k scanner records only which k-factors occur in the string, it has no way of distinguishing strings which are built from the same set of k-factors. Hence, a language is LT_k if and only if there is no pair of strings, each comprising the same set of k-factors, one of which is included in the language and the other excluded.

From a cognitive perspective, then:

— Any cognitive mechanism that can distinguish member strings from non-members of an LT_k language must be sensitive, at least, to the *set* of length k contiguous blocks of symbols that occur in the presentation of the string—both those that do occur and those that do not.
— If the strings are presented as sequences of symbols in time, then this corresponds to being sensitive, at each point in the string, to the set of length k blocks of symbols that occurred at any prior point.
— Any cognitive mechanism that is sensitive *only* to the occurrence or non-occurrence of length k contiguous blocks of symbols in the presentation of a string will be able to recognize *only* LT_k languages.

One of the consequences of the inability of k-expressions to distinguish strings which comprise the same set of k-factors is that LT languages cannot, in general, distinguish strings in which there is a single occurrence of some symbol from those in which there are multiple occurrences: the strings $\rtimes\underbrace{A\cdots A}_{k-1}B\underbrace{A\cdots A}_{k-1}\ltimes$ and $\rtimes\underbrace{A\cdots A}_{k-1}B\underbrace{A\cdots A}_{k-1}B\underbrace{A\cdots A}_{k-1}\ltimes$ comprise exactly the same set of k-factors. Consequently, no mechanism that is sensitive *only* to the set of fixed size blocks symbols that occur in a string will be able, in general, to distinguish strings with a single instance of a symbol from those with more than one.

(d) *Probing the LT boundary*

The language of strings in which some block of k contiguous symbols is required to occur at least once (e.g. some-B of §6*b*, for which any $k \geq 1$ will do) is LT_k. Mechanisms which are sensitive to the set of fixed length blocks of consecutive symbols which have occurred are capable, in principle, of distinguishing strings that meet such a constraint (e.g. A^iBA^j) from those that do not (e.g. A^{i+j+1}). Again, these patterns form a basis for developing sets of stimuli that provide evidence of an organism's ability to make these distinctions.

Conversely, the language of strings in which some block of k contiguous symbols is required to occur *exactly* once (e.g. one-B, in which exactly one 'B' occurs in every string) is not LT_k for any k. Mechanisms that are sensitive only to the set of fixed length blocks of consecutive symbols which have occurred are incapable of distinguishing strings that meet such a constraint from those that do not. Thus sets of stimuli generated by the patterns A^iBA^{j+k+1} (in the set one-B) and $A^iBA^jBA^k$ (not in that set) can be used to probe whether an organism is limited to distinguishing sets of strings on this basis.

(e) *FO(+1) definability: LTT*

This weakness of LT is, simply put, an insensitivity to quantity as opposed to simple occurrence. We can overcome this by adding *quantification* to our logical descriptions, that is, by moving from propositional logic to a first-order logic which we call FO(+1). Logical formulae of this sort make (Boolean combinations of) assertions about which symbols occur at which positions ($\sigma(x)$, where σ is a symbol in the vocabulary and x is a variable ranging over positions in a string), about the adjacency of positions ($x \lhd y$, which asserts that the position represented by y is the successor of that represented by x) and about the identity of positions ($x \approx y$), with the positions being quantified existentially ($\exists$) or universally ($\forall$). This

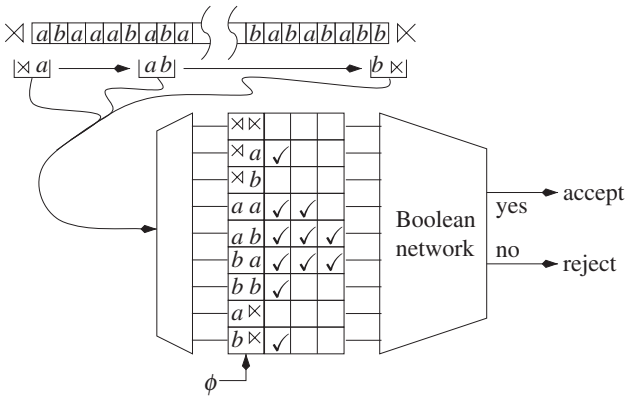


Figure 9. LTT automata count the number of k-factors that occur in a string up to some bound and feed this information into a Boolean network. The automaton accepts if, once the entire string has been scanned, the output of the network is 'Yes', and rejects otherwise.

allows us to distinguish, for example, one occurrence of a B from another:

$$\varphi_{\text{One-B}} = (\exists x)[B(x) \wedge (\neg\exists y)[B(y) \wedge \neg x \approx y]]$$

This FO(+1) formula requires that there is some position in the string (call it x) at which a B occurs and there is no position (y) at which a B occurs that is distinct from that ($\neg x \approx y$).

In this example, we have defined a property expressed in terms of the occurrence of a single symbol B, but, using the successor relation, we could just as well be restricting the number of occurrences of any k-factor, for some fixed k. Moreover, using multiple levels of quantification, we can distinguish arbitrarily many distinct positions, but, since a single formula can only contain a fixed number of quantifiers, there is a fixed finite bound on the number of positions a given formula can distinguish. Hence FO(+1) formulae can, in essence, count, but only up to some fixed threshold. Note that the fixed threshold is compatible with subitization as well as actual counting.

The class of FO(+1) definable languages is characterized by what is known as *local threshold testability* (LTT). LTT automata extend LT automata by counting the number of occurrences of each k-factor, with the counters counting up to a fixed maximum and then simply staying at that value if there are additional occurrences. (See figure 9.)

This gives us a cognitive interpretation of LTT:

— Any cognitive mechanism that can distinguish member strings from non-members of an LTT language must be sensitive, at least, to the multiplicity of the length k blocks of symbols, for some fixed k, that occur in the presentation of the string, distinguishing multiplicities only up to some fixed threshold t.
— If the strings are presented as sequences of symbols in time, then this corresponds to being able count up to some fixed threshold.
— Any cognitive mechanism that is sensitive *only* to the multiplicity, up to some fixed threshold (and, in particular, not to the order) of the length k blocks of symbols in the presentation of a string will be able to recognize *only* LTT languages.

(f) *Probing the LTT boundary*

The language of strings in which some block of k contiguous symbols is required to occur exactly once (e.g. one-B, for which any k and $t \geq 1$ will do) is $\text{LTT}_{k,t}$. Mechanisms which are sensitive to the multiplicity, up to some fixed threshold, of fixed length blocks of consecutive symbols which have occurred are capable, in principle, of distinguishing strings that meet such a constraint (e.g. A^iBA^{j+k+1}) from those that do not (e.g. $A^iBA^jBA^k$).

Conversely, the language of strings in which some block of k contiguous symbols is required to occur *prior* to the occurrence of another (e.g. no-B-after-C, in which no string has an occurrence of 'C' that precedes an occurrence of 'B', with 'A's freely distributed) is not $\text{LTT}_{k,t}$ for any k or t. Mechanisms that are sensitive only to the multiplicity, up to some fixed boundary, of the occurrences of fixed length blocks of consecutive symbols are incapable of distinguishing strings that meet such a constraint from those that do not. Sets of stimuli that test this ability can be based on the patterns $A^iBA^jCA^k$, $A^iBA^jBA^k$ and $A^iCA^jCA^k$, all of which satisfy the no-B-after-C constraint, and $A^iCA^jBA^k$, which violates it.

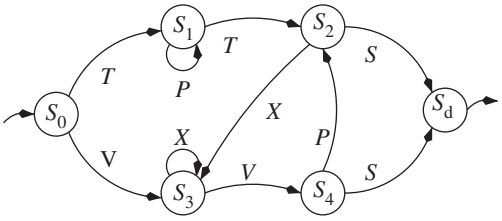
(g) *FO(<) definability: SF*

If we extend the logic of FO(+1) to express relationships in terms of precedence (<) as well as successor, then we can define constraints in terms of both the multiplicity of factors and their order.[15] The class of FO(<) definable languages is properly known as LTO (*locally testable with order*), but this turns out to be equivalent to the better known class of *star-free* (SF) languages. These are the class of languages that are definable by regular expressions without Kleene-closure—in which the '*' operator does not occur—but with complement—in which the '$\overline{(\cdot)}$' operator may occur [22].

This is, in terms of model-theoretic definability, the strongest class that is a proper subclass of the regular languages. The regular languages are the sets of strings that are definable using +1 (and/or <) and monadic second-order quantification—quantifications over subsets of positions as well as over individual positions. It is not clear that this increase in definitional power is actually useful from a linguistic point of view. There seems to be very few, if any, natural linguistic constraints that are regular but not star-free.

(i) *Cognitive interpretation of SF*

— Any cognitive mechanism that can distinguish member strings from non-members of an SF language must be sensitive, at least, to the order of the length k blocks of symbols, for some fixed k and some fixed maximum length of the sequences of blocks, that occur in the presentation of the string.
— If the strings are presented as sequences of symbols in time, then this corresponds to being sensitive to the set of sequences, up to that maximum length,



3-factors

$\rtimes TP$	$\rtimes TT$	$\rtimes VX$	$\rtimes VV$
TPP	TPT	TTX	TTS
TXX	TXV	$TS\ltimes$	
VXX	VXV	VVP	VVS
VPX	VPS	$VS\ltimes$	
PPP	PPT	PTX	PTS
PXX	PXV	$PS\ltimes$	
XXX	XXV	XVP	XVS

Figure 10. Reber's [30] grammar.

of the length k blocks that have occurred at any prior point.

— Any cognitive mechanism that is sensitive *only* to the set of fixed length sequences of length k blocks of symbols in the presentation of a string will be able to recognize *only* SF languages.

7. STATISTICAL MODELS OF LANGUAGE

Many studies of AGL have focused on statistical learning [25–27]. Language models which are based on the probability of a given symbol following another are Markov processes [28]. These can be interpreted as FSAs with transition probabilities where the underlying FSA recognizes an SL_2 language. 'n-gram' and 'n-factor' are equivalent concepts; in general, an n-gram model is a weighted version of a FSA that recognizes an SL_n language; an n-gram model of a language (an $(n-1)$st-order Markov model) *is* a strictly n-local distribution.

Statistical models of language are not directly comparable to the sets of strings of traditional FLT, but there is a clear distinction between SL languages and n-gram models in which probabilities are not preserved under substitution of suffixes. Nevertheless, a language learner that infers probabilities of n-grams must be able to distinguish n-grams. In other words, it must attend to the n-factors of the input. Thus, the notion of cognitive complexity that we have developed here is still relevant.

Each of the levels of the hierarchy corresponds to a class of statistical distributions. The number of parameters, though, rises rapidly—the number of parameters of an LT_k distribution is exponential in k. In application, the higher complexity models are likely to be infeasible. On the other hand, there is a complexity hierarchy that parallels the local hierarchy but which is based on precedence—order of symbols independent of the intervening material [23]—which also provides a basis for statistical models. The *strictly piecewise* distributions, those analogous to n-gram models, are both feasible and are capable of discriminating many long-distance dependencies [29]. The question of whether a learner can attend to subsequences (with arbitrary intervening material) in addition to or rather than substrings (factors) is significant.

8. SOME CLASSIC ARTIFICIAL GRAMMARS

Some of the earliest AGL experiments were conducted by Reber [30]. These were based on the grammar represented by the FSA in figure 10. This automaton recognizes an SL_3 language, licensed by the set of 3-factors also given in the figure—to learn a language of this form, the subject need only attend to the blocks of three consecutive symbols occurring in the strings, recognizing an exception when one of the forbidden blocks occurs.

Saffran *et al.* [26] presented their test subjects with continuous streams of words in which word boundaries were indicated only by the transitional probabilities between syllables. In general, this would give an arbitrary SL_2 distribution, but in this case the probabilities of the transitions internal to the words is 1.0 and all transitions between words are equiprobable.[16] Under these circumstances, the language is the same as that of the SL_2 automaton on which the distribution is based—i.e. this language is simply a strictly 2-local language. It should be noted, though, that from the perspective of cognitive complexity we have presented here this is a distinction without a difference. Whether the language is a non-trivial statistical model or not, to learn it the subjects need only attend to the pairs of adjacent syllables occurring in the stimulus.

Marcus *et al.* [31] specifically avoided prediction by transitional probabilities by testing their subjects with strings generated according to the training pattern, but over a novel vocabulary. Gomez & Gerken [32] used a similar design. In the latter case, the grammars they used are similar to that of Reber and also license SL_3 languages. Marcus *et al.* limited their stimuli to exactly three syllables in order to eliminate word length as a possible cue. In general, every language of three syllable words is trivially SL_4. The focus of the experiments, though, were strings distinguished by where and if syllables were repeated (i.e. *ABA* versus *AAB* versus *ABB*). Languages in which no syllable is repeated are simply SL_2; those in which either the first pair of syllables, the last pair of syllables and/or the first and last syllable are repeated are SL_3. In all of these cases, the language can be distinguished by simply attending to blocks of consecutive syllables in isolation.

Finally, Saffran [27] used a language generated by a simple context-free grammar in which there is no self-embedding—no category includes a constituent which is of the same category. Hence, this language is also finite and trivially SL. Again, though, the experiments focused on the ability of the subjects to learn patterns within these finite bounds. In this case, there were two types of patterns. In the first type, a word in some category occurs only in the presence of

a word in another specific category. In the second type a word in some category occurs only when a word of a specific category occurs somewhere prior to it in the word. These are both non-strictly local patterns. The first is LT_1—learning this pattern requires attention to the set of syllables that occur in the word. The second is strictly 2-piecewise testable, at the lowest level of the precedence-based hierarchy. Learning it requires attention to the set of pairs of syllables in which one occurs prior to the other in the word, with arbitrary intervening material.

Many recent AGL experiments have employed patterns of the types $(AB)^n$ (repeated 'AB' pairs) and A^nB^n (repeated 'A's followed by exactly the same number of 'B's, sometimes with explicit pairing of 'A's and 'B's either in nested order or in 'cross-serial' order: first 'A' with first 'B', etc.) As noted in §6*a*, $(AB)^n$ is strictly 2-local, the simplest of the complexity classes we have discussed here. A^nB^n is properly context-free, with or without explicit dependencies. All automata that can recognize non-finite-state languages can be modelled as FSAs with some sort of additional unbounded memory (one or more counters, a stack, a tape, etc.) The question of what capabilities are required to recognize properly CFLs, then, is a question of how that storage is organized.

As noted in §5, A^nB^n without explicit dependencies can be recognized by counting 'A's and 'B's (with a single counter), a simpler mechanism than that required to recognize CFLs in general. A^nB^n with explicit nested dependencies cannot be recognized using a single counter, but it is a *linear* CFL,[17] also simpler than the class of CFLs in general. The question of whether there are dependencies between the 'A's and 'B's is another issue that generally depends on knowing the way that the strings are being analysed. But it is possible to make the distinction between languages that can be recognized with a single counter and those that are properly linear CFLs without appealing to explicit dependencies by using the language $A^nB^mC^mD^n$.[18] If the dependencies between the 'A's and 'B's are cross-serial, then in A^nB^n is properly non-context-free. A language that makes the same distinction without explicit dependencies is $A^nB^mC^nD^m$.

The difficulty of identifying which type of structure is being used by a subject to recognize a given non-regular pattern in natural languages delayed confirmation that there were human languages that employed cross-serial dependencies for decades [5–7, 34,35]. In AGL experiments, one has the advantage of choosing the pattern, but the disadvantage of not having *a priori* knowledge of which attributes of the symbols are being distinguished by the subjects. The fact that a particular 'A' is paired with a particular 'B' means that those instances must have a different character than other instances of the same category. Indeed, these patterns can be represented as A^nA^n with explicit dependencies of some sort between the 'A's in the first half of the string and those in the second half. Hence, most artificial grammars of this sort are actually more complicated versions of $A^nB^mC^mD^n$ or $A^nB^mC^nD^m$. There seems to be little advantage to using patterns in which the dependencies are less explicit than these.

9. DESIGNING AND INTERPRETING ARTIFICIAL GRAMMAR LEARNING EXPERIMENTS

All of this gives us a foundation for exploring AGL experiments from a formal perspective. We will consider familiarization/discrimination experiments. We will use the term *generalize* rather than 'learn' or 'become familiarized with' and will refer to a response that indicates that a string is recognized as an exception as *surprise*.

Let us call the set generated by the artificial grammar we are interested in **I**, the *intended* set. The subject is exposed to some finite subset of this, which we will call **F**, the *familiarization* set. It then generalizes to some set (possibly the same set—the generalization may be trivial). *Which* set they generalize to gives evidence of the features of **F** the subject attended to. An error-free learner would not necessarily generalize to **I**, any superset of **F** is consistent with their experience. We will assume that the set the subject generalizes to is not arbitrary—it is not restricted in ways that are not exhibited by **F**—and that the inference mechanism exhibits some sort of minimality—it infers judgements about the stimuli that are not in **F** as well as those that are.[19]

We then present the subject with a set which we will call **D**, the *discrimination* set, which includes some stimuli that are in **I** and some which are not, and observe which of these are treated by the subject as familiar and which are surprising. One can draw conclusions from these observations only to the extent that **D** distinguishes **I** from other potential generalizations. That is, **D** needs to distinguish all supersets and subsets of **I** that are consistent with (i.e. supersets of) **F**.

Figure 11 represents a situation in which we are testing the subject's ability to recognize a set that is generated by the pattern $\mathbf{I} = A^nB^n$, in which a block of 'A's is followed by block of 'B's of the same length, and we have exposed the subject to stimuli of the form $\mathbf{F} = AABB$. The feature that is of primary interest is the fact that the number of 'A's and 'B's is exactly the same, a constraint that is properly context-free.

The innermost circle encompasses **F**. The bold circle delimits the intended set **I**. The other circles represent sets that are consistent with **F** but which generalize on other features. For example, a subject that identifies only the fact that all of the 'A's precede all of the 'B's would generalize to the set we have called A^nB^m. This set is represented by the middle circle on the right-hand side of the figure and encompasses **I**. A subject that identifies, in addition, the fact that all stimuli in **F** are of even length might generalize to the set we label $A^nB^m_{\text{even}}$. This is represented by the inner circle on the right-hand side.

It is also possible that the subject has learned that there are at least as many 'A's as there are 'B's (A^nB^{n+m}) or *v.v.* These sets include **I** and that portion of A^nB^m above (resp., below) the dotted line. Alternatively, if the subject has generalized from the fact that the number of 'A's is equal to the number of 'B's but not the fact that all of the 'A's precede all of the 'B's, they might generalize to the set we label AB_{equal}. Included in this set, as a proper subset, is the language $(AB)^n$, which is not consistent with **F** but does share the feature of the 'A's and 'B's being equinumerous.

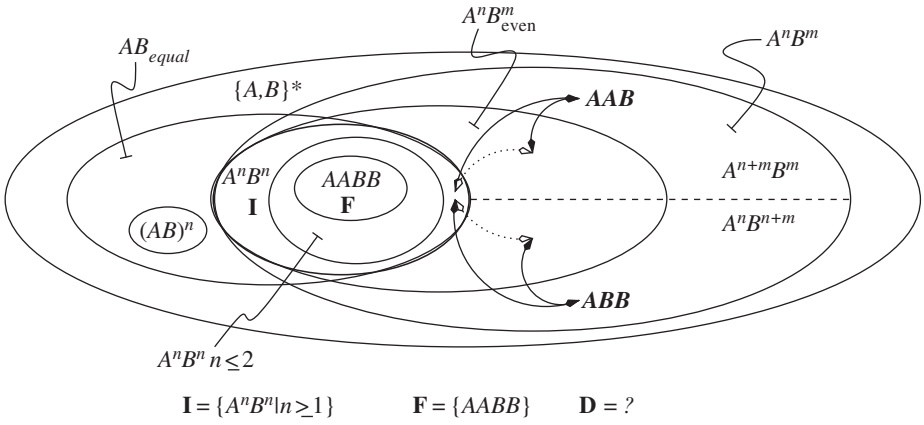


Figure 11. Language inference experiments.

It is also possible that the subject makes the minimal generalization that the number of '*A*'s is finitely bounded. The smallest of these sets consistent with **F** is the set we have labelled A^nB^n, $n \leq 2$.

These, clearly, are not all of the sets the subject might consistently infer from **F** but they are a reasonable sample of principled generalizations that we need to distinguish from **I**. Note that, in this case, the potential generalizations span the entire range of classes from SL_2 (A^nB^m), through CF (A^nB^{n+m} and AB_{equal}). If we are to distinguish, say, the boundary between finite state and context-free our probes will need to distinguish these. To do that, **D** must include stimuli in the symmetric differences of the potential generalizations (including **I**), that is to say stimuli that are in one or the other but not both.

For example, suppose we test the subject with strings of the form *AAB*. These are not in A^nB^n (properly CF) but are in the set A^nB^m (SL), so subjects that have generalized to the former will generally find them surprising while those that have generalized to the latter will not. On the other hand, they are also not in the set $A^nB^m_{\text{even}}$, which is finite state, but are in $A^{n+m}B^m$, which is properly context-free. Thus these stimuli do not resolve the finite state boundary.

Suppose, then, that we test with strings of the form *AAAB*. These, again, are not in (CF) but are in A^nB^m (SL). But they are also not in both of $A^nB^m_{\text{even}}$ (finite state) and $A^{n+m}B^m$ (properly context-free). Again, they do not resolve the finite state boundary.

One can distinguish the finite state from the non-finite state languages in this particular set of potential generalizations if one includes strings of both of these forms in **D**, but that still will not distinguish A^nB^n from A^nB^{n+m}, which is presumably not significant here but may well be in testing other hypotheses. These can be distinguished by including strings of the forms that correspond to these but include more '*B*'s than '*A*'s.

None of these, though, distinguish a learner that has generalized to a finite set (A^nB^n, $n \leq 2$). To get evidence that the learner has done this, one needs to include strings of length greater than four.

One ends up with a discrimination set that includes at least five variants of the pattern A^nB^m for different values of n and m between two and six. This seems to be very near the boundary of practicality for most experiments involving living organisms. There are two ways that one might resolve this limitation: one can find experiments which can distinguish performance on stimuli of this size, perhaps not being able to draw any conclusions for some types of subject, or one can refine one's hypothesis so that it is practically testable. In any case, the issue is one that requires a good deal of careful analysis and it is an issue that cannot be ignored.

10. CONCLUSION

The notion of language theoretic complexity, both with respect to the Chomsky hierarchy and with respect to the sub-regular hierarchies, is an essential tool in AGL experiments. In the design of experiments they provide a way of formulating meaningful, testable hypotheses, of identifying relevant classes of patterns, of finding minimal pairs of languages that distinguish those classes and of constructing sets of stimuli that resolve the boundaries of those languages. In the interpretation of the results of the experiments the properties of the complexity classes provide a means of identifying the pattern a subject has generalized to, the class of patterns the subject population is capable of generalizing to and, ultimately, a means of identifying those features of the stimulus that the cognitive mechanisms being used are sensitive to.

In this paper, we have presented a scale for informing these issues that is both finer than and broader than the finite state/context-free contrast that has been the focus of much of the AGL work to date. While some of the distinctions between classes are subtle, and some of the analyses delicate, there are effective methods for distinguishing them that are generally not hard to apply and the range of characterizations of the classes provides a variety of tools that can be employed in doing so. More importantly, the capabilities distinguished by these classes are very likely to be significant in resolving the issues that much of this research is intended to explore.

Finally, fully abstract characterizations of language classes, like many of those we have presented here, provide information about characteristics of the processing mechanism that are necessarily shared by all mechanisms that are capable of recognizing languages in these classes. This provides a foundation for

unambiguous and strongly supported results about cognitive mechanisms for pattern recognition.

We thank William Tecumseh Fitch and the anonymous reviewers for immensely helpful comments on the draft version of this article.

ENDNOTES

[1]Authoritative textbooks on this field are [1,2].

[2]This points to another simplification that is needed when applying FLT to natural languages: in each language with productive word formation rules, the set of possible words is unbounded. Likewise, the set of morphemes is in principle unbounded if loans from other languages, acronym formation and similar processes are taken into consideration. It is commonly assumed here that the object of investigation is an idealized language that does not undergo change. When the vocabulary items are identified with words, it is tacitly taken for granted that the words of a language form a finite number of grammatical categories, and that it is thus sufficient to consider only a finite number of instances of each class.

[3]The term 'context-sensitive' has only historical significance. It has noting to do with context-dependency in a non-technical sense in any way. The same applies to the term 'context-free'.

[4]In the terminology of computational complexity theory, the problem is PSPACE hard.

[5]In context-free grammars, the right-hand side of a rule may be the empty string, while in context-sensitive grammars this is not licit. Therefore, strictly speaking, not every context-free grammar is context-sensitive. This is a minor technical point though that can be ignored in the present context.

[6]Equivalently, we may demand that the rules take the form '$A \to a$' or '$A \to Ba$', with the non-terminal, if present, preceding the terminal. It is crucial though that within a given grammar, all rules start with a terminal on the right-hand side, or all rules end with a terminal.

[7]Note that here one of the idealizations mentioned above come into play here: It is taken for granted that a productive pattern—forming a *neither–nor* construction out of two grammatical sentences—can be applied to arbitrarily large sentences to form an even larger sentence.

[8]Here, we strictly refer to the problem whether the set of strings of grammatical English sentences is a CFL, disregarding all further criteria for the linguistic adequacy of a grammatical description.

[9]A thorough discussion of types of dependencies in natural languages and mild context-sensitivity can be found in [17].

[10]Even more strongly, it is generally possible to convert descriptions from one class of models to descriptions in an equivalent class fully automatically.

[11]This does not imply that mechanisms that are physically finitely bounded—the brain of an organism, for example—are restricted to recognizing only finite-state languages. The organism may well employ a mechanism that, in general, requires unbounded memory which would simply fail when it encounters a string that is too complex, if it ever did encounter such a string.

[12]More details on this and the other local classes of languages can be found in [24].

[13]We use capital letters here to represent arbitrary symbols drawn from mutually distinct categories of the symbols of the vocabulary. Although none of our mechanisms involve the sort of string rewriting employed by the grammars of the first part of this paper and we distinguish no formal set of non-terminals, there is a rough parallel between this use of capital letters to represent categories of terminals and the interpretation of non-terminals as representing grammatical categories in phrase-structure grammars.

[14]The patterns are both defined in terms of the parameters i and j so that the length of the strings do not vary between them.

[15]The successor relationship is definable using only $<$ and quantification, so one no longer explicitly needs the successor relation. Similarly, multiplicity of factors can be defined in terms of their order, so one does not actually need to count to a threshold greater than 1.

[16]Technically, the final probabilities of this distribution were all 0, i.e. the distribution included no finite strings.

[17]In which only a single non-terminal occurs at any point in the derivation.

[18]Note that two counters would suffice to recognize $A^nB^mC^mD^n$ but, as Minsky showed [33], two counters suffice to recognize *any* computable language.

[19]The assumption that the generalization is not arbitrary implies, *inter alia*, that if it includes strings that are longer than those in **F** it will include strings of arbitrary length. This allows one to verify that a subject has not simply made some finite generalization of the (necessarily finite) set **F**.

REFERENCES

1 Hopcroft, J. E., Motwani, R. & Ullman, J. D. 2000 *Introduction to automata theory, languages and computation*. Reading, MA: Addison-Wesley.

2 Sipser, M. 1997 *Introduction to the theory of computation*. Boston, MA: PWS Publishing.

3 Chomsky, N. 1956 Three models for the description of language. *IRE Trans. Inform. Theory* **2**, 113–124. (doi:10.1109/TIT.1956.1056813)

4 Chomsky, N. 1957 *Syntactic structures*. The Hague, The Netherlands: Mouton.

5 Huybregts, R. 1984 The weak inadequacy of context-free phrase structure grammars. In *Van periferie naar kern* (eds. G. J. de Haan, M. Trommelen & W. Zonneveld), ppp. 81–99. Dordrecht, The Netherlands: Foris.

6 Shieber, S. 1985 Evidence against the context-freeness of natural language. *Linguist. Phil.* **8**, 333–343. (doi:10.1007/BF00630917)

7 Culy, C. 1985 The complexity of the vocabulary of Bambara. *Linguist. Philos.* **8**, 345–351. (doi:10.1007/BF00630918)

8 Joshi, A. 1985 How much context-sensitivity is necessary for characterizing structural descriptions—tree adjoining grammars. In *Natural language processing. Theoretical, computational and psychological perspectives* (eds. D. Dowty, L. Karttunen & A. Zwicky). Cambridge, UK: Cambridge University Press.

9 Ades, A. E. & Steedman, M. J. 1982 On the order of words. *Linguist. Philos.* **4**, 517–558. (doi:10.1007/BF00360804)

10 Steedman, M. 2000 *The syntactic process*. Cambridge, MA: MIT Press.

11 Joshi, A., Vijay-Shanker, K. & Weir, D. 1991 The convergence of mildly context-sensitive grammar formalisms. In *Processing of linguistic structure* (eds. P. Sells, S. Shieber & T. Wasow), pp. 31–81. Cambridge, MA: MIT Press.

12 Gazdar, G. 1988 Applicability of indexed grammars to natural languages. Technical Report 85–34, Center for the Study of Language and Information, Stanford, CA.

13 Pollard, C. J. 1984 Generalized phrase structure grammars, head grammars and natural language. PhD thesis, Stanford, CA.

14 Weir, D. J. 1988 Characterizing mildly context-sensitive grammar formalisms. PhD thesis, University of Pennsylvania.

15 Stabler, E. P. 1997 Derivational minimalism. In *Logical aspects of computational linguistics* (ed. C. Retoré), pp. 68–95. Berlin, Germany: Springer.

16 Chomsky, N. 1995 *The minimalist program*. Cambridge, MA: MIT Press.

17 Stabler, E. P. 2004 Varieties of crossing dependencies: structure dependence and mild context sensitivity. *Cogn. Sci.* **28**, 699–720. (doi:10.1207/s15516709cog2805_4)

18 Becker, T., Joshi, A. & Rambow, O. 1991 Long-distance scrambling and tree adjoining grammars. In *Proc. 5th Conf. of European Chapter of the Association for Computational Linguistics, Berlin, Germany, 9–11 April 1991*, pp. 21–26. Association for Computational Linguistics.

19 Michaelis, J. & Kracht, M. 1997 Semilinearity as a syntactic invariant. In *Logical aspects of computational linguistics* (ed. C. Retoré), pp. 329–345. Berlin, Germany: Springer.
20 Tesnière, L. 1959 *Elèments de syntaxe structurale*. Paris, France: Klincksiek.
21 Rogers, J. 2003 wMSO theories as grammar formalisms. *Theoret. Comput. Sci.* **293**, 291–320. (doi:10.1016/S0304-3975(01)00349-8)
22 McNaughton, R. & Papert, S. A. 1971 *Counter-free automata*. Cambridge, MA: MIT Press.
23 Rogers, J., Heinz, J., Bailey, G., Edlefsen, M., Visscher, M., Wellcome, D. & Wibel, S. 2010 On languages piecewise testable in the strict sense. In *The mathematics of language: revised selected papers from the 10th and 11th Biennial Conference on the Mathematics of Language* (eds. C. Ebert, G. Jäger & J. Michaelis), Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence, vol. 6149, pp. 255–265. Berlin, Germany: FoLLI/Springer.
24 Rogers, J. & Pullum, G. 2007 Aural pattern recognition experiments and the subregular hierarchy. In *Proc. 10th Mathematics of Language Conference* (ed. M. Kracht), pp. 1–7. Los Angeles, CA: University of California.
25 Christiansen, M. H. & Chater, N. (ed.) 2001 *Connectionist psycholinguistics*. New York, NY: Ablex.
26 Saffran, J. R., Aslin, R. N. & Newport, E. L. 1996 Statistial learning by 8-month-old infants. *Science* **274**, 1926–1928. (doi:10.1126/science.274.5294.1926)
27 Saffran, J. R. 2001 The use of predictive dependencies in language learning. *J. Mem. Lang.* **44**, 493–515. (doi:10.1006/jmla.2000.2759)
28 Manning, C. & Schütze, H. 1999 *Foundations of statistical natural language processing*. Cambridge, MA: MIT Press.
29 Heinz, J. & Rogers, J. 2010 Estimating strictly piecewise distributions. In *Proc. 48th Annual Meeting of the Association for Computational Linguistics, Uppsala, Sweden, 11–16 July 2010*, pp. 886–896. Association for Computational Linguistics.
30 Reber, A. S. 1967 Implicit learning of artificial grammars. *J. Verb. Learn. Verb. Behav.* **6**, 855–863. (doi:10.1016/S0022-5371(67)80149-X)
31 Marcus, G. G., Vijayan, S., Bandi Rao, S. & Vishton, P. M. 1999 Rule learning by seven-month-old infants. *Science* **283**, 77–79. (doi:10.1126/science.283.5398.77)
32 Gomez, R. L. & Gerken, L. 1999 Artificial grammar learning by 1-year-olds leads to specific and abstract knowledge. *Cognition* **70**, 109–135. (doi:10.1016/S0010-0277(99)00003-7)
33 Minsky, M. L. 1961 Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing machines. *Ann. Math.* **74**, 437–455. (doi:10.2307/1970290)
34 Pullum, G. K. & Gazdar, G. 1982 Natural languages and context-free languages. *Linguist. Phil.* **4**, 471–504. (doi:10.1007/BF00360802)
35 Pullum, G. K. 1985 On two recent attempts to show that English is not a CFL. *Comput. Linguist.* **10**, 182–186.