Chapter 16
Language and Complexity

This is the dog, that worried the cat, that killed the rat, that ate the malt, that lay in the house that Jack built.

Mother Goose, *The House that Jack Built*

This is the malt that the rat that the cat that the dog worried killed ate.

Victor H. Yngve (1960)

Much of the humor in musical comedy and comic operetta comes from entwining the main characters in fabulously complicated plot twists. Casilda, the daughter of the Duke of Plaza-Toro in Gilbert and Sullivan's *The Gondoliers*, is in love with her father's attendant Luiz. Unfortunately, Casilda discovers she has already been married (by proxy) as a babe of six months to "the infant son and heir of His Majesty the immeasurably wealthy King of Barataria". It is revealed that this infant son was spirited away by the Grand Inquisitor and raised by a "highly respectable gondolier" in Venice as a gondolier. The gondolier had a baby of the same age and could never remember which child was which, and so Casilda was in the unenviable position, as she puts it, of "being married to one of two gondoliers, but it is impossible to say which". By way of consolation, the Grand Inquisitor informs her that "such complications frequently occur".

Luckily, such complications don't frequently occur in natural language. Or do they? In fact there are sentences that are so complex that they are hard to understand, such as Yngve's sentence above, or the sentence:

"*The Republicans who the senator who she voted for chastised were trying to cut all benefits for veterans*".

Studying such sentences, and more generally understanding what level of complexity tends to occur in natural language, is an important area of language processing. Complexity plays an important role, for example, in deciding when we need to use a particular formal mechanism. Formal mechanisms like finite automata, Markov models, transducers, phonological rewrite rules, and context-free grammars, can be described in terms of their **power**, or equivalently in terms of the **complexity** of the phenomena that they can describe. This chapter introduces the Chomsky hierarchy, a theoretical tool that allows us to compare the expressive power or complexity of these different formal mechanisms. With this tool in hand, we summarize arguments about the correct formal power of the syntax of natural languages, in particular English but also including a famous Swiss dialect of German that has the interesting syntactic property called **cross-serial dependencies**. This property has been used to argue that context-free

power
complexity

grammars are insufficiently powerful to model the morphology and syntax of natural language.

In addition to using complexity as a metric for understanding the relation between natural language and formal models, the field of complexity is also concerned with what makes individual constructions or sentences hard to understand. For example we saw above that certain **nested** or **center-embedded** sentences are difficult for people to process. Understanding what makes some sentences difficult for people to process is an important part of understanding human parsing.

16.1 The Chomsky Hierarchy

How are automata, context-free grammars, and phonological rewrite rules related? What they have in common is that each describes a **formal language**, which we have seen is a set of strings over a finite alphabet. But the kind of grammars we can write with each of these formalism are of different **generative power**. One grammar is of greater generative power or **complexity** than another if it can define a language that the other cannot define. We will show, for example, that a context-free grammar can be used to describe formal languages that cannot be described with a finite-state automaton.

generative power

It is possible to construct a hierarchy of grammars, where the set of languages describable by grammars of greater power subsumes the set of languages describable by grammars of lesser power. There are many possible such hierarchies; the one that is most commonly used in computational linguistics is the **Chomsky hierarchy** (Chomsky, 1959a), which includes four kinds of grammars: Fig. 16.1 shows the four grammars in the Chomsky hierarchy as well as a useful fifth type, the *mildly context-sensitive* languages.

Chomsky hierarchy

This decrease in the generative power of languages from the most powerful to the weakest can in general be accomplished by placing constraints on the way the grammar rules are allowed to be written. Fig. 16.2 shows the five types of grammars in the extended Chomsky hierarchy, defined by the constraints on the form that rules must take. In these examples, A is a single non-terminal, and α, β, and γ are arbitrary strings of terminal and non-terminal symbols. They may be empty unless this is specifically disallowed below. x is an arbitrary string of terminal symbols.

Turing-equivalent, **Type 0** or **unrestricted** grammars have no restrictions on the form of their rules, except that the left-hand side cannot be the empty string ϵ. Any (non-null) string can be written as any other string (or as ϵ). Type 0 grammars characterize the **recursively enumerable** languages, that is, those whose strings can be listed (enumerated) by a Turing Machine.

recursively enumerable

Context-sensitive grammars have rules that rewrite a non-terminal symbol A in the context $\alpha A\beta$ as any non-empty string of symbols. They can be either written in the form $\alpha A\beta \rightarrow \alpha\gamma\beta$ or in the form $A \rightarrow \gamma / \alpha\_\_\beta$. We have seen this latter version in the Chomsky-Halle representation of phonological rules (Chomsky and Halle, 1968) like this flapping rule:

Context-sensitive

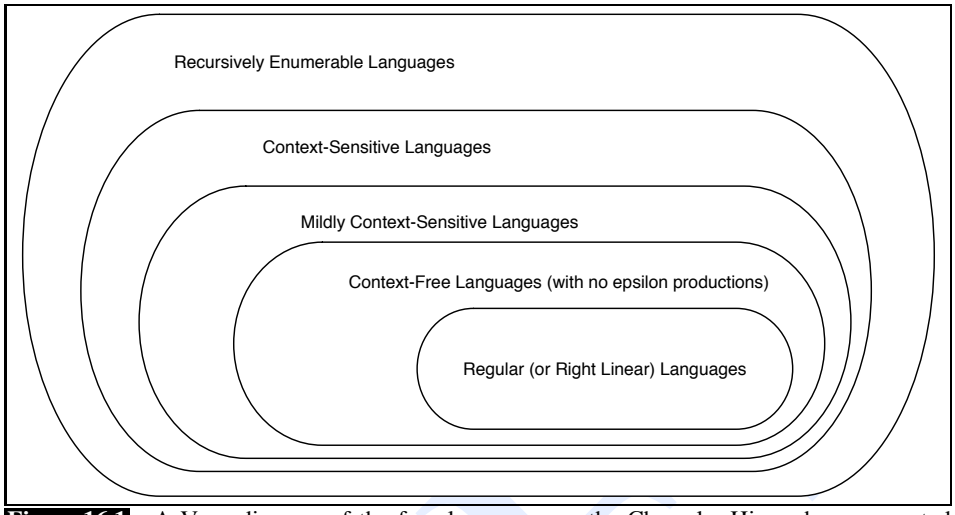

Figure 16.1 A Venn diagram of the four languages on the Chomsky Hierarchy, augmented with a fifth class, the mildly context-sensitive languages.

Type	Common Name	Rule Skeleton	Linguistic Example
0	Turing Equivalent	$\alpha \rightarrow \beta$, s.t. $\alpha \neq \epsilon$	HPSG, LFG, Minimalism
1	Context Sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$, s.t. $\gamma \neq \epsilon$	
-	Mildly Context Sensitive		TAG, CCG
2	Context Free	$A \rightarrow \gamma$	Phrase Structure Grammars
3	Regular	$A \rightarrow xB$ or $A \rightarrow x$	Finite State Automata

Figure 16.2 The Chomsky Hierarchy, augumented by the mildly context-sensitive grammars.

/t/ → [dx] / V́ ___ V

While the form of these rules seems context-sensitive, Ch. 7 showed that phonological rule systems that do not have recursion are actually equivalent in power to the regular grammars.

Another way of conceptualizing a rule in a context-sensitive grammar is as rewriting a string of symbols δ as another string of symbols ϕ in a "non-decreasing" way; such that ϕ has at least as many symbols as δ.

context-free

We studied **context-free** grammars in Ch. 12. Context-free rules allow any single non-terminal to be rewritten as any string of terminals and non-terminals. A non-terminal may also be rewritten as ϵ, although we didn't make use of this option in Ch. 12.

Regular grammars are equivalent to regular expressions. That is, a given regular language can be characterized either by a regular expression of the type we discussed in Chapter 2, or by a regular grammar. Regular grammars can either be **right-linear** or **left-linear**. A rule in a right-linear grammar has a single non-terminal on the left, and at most one non-terminal on the right-hand side. If there is a non-terminal on the right-hand side, it must be the last symbol in the string. The right-hand-side of left-linear grammars is reversed (the right-hand-side must start with (at most) a single non-terminal). All regular languages have both a left-linear and a right-linear grammar. For the rest of our discussion, we will consider only the right-linear grammars.

right-linear
left-linear

For example, consider the following regular (right-linear) grammar:

$$\begin{aligned} S &\rightarrow aA \\ S &\rightarrow bB \\ A &\rightarrow aS \\ B &\rightarrow bbS \\ S &\rightarrow \epsilon \end{aligned}$$

It is regular, since the left-hand-side of each rule is a single non-terminal and each right-hand side has at most one (rightmost) non-terminal. Here is a sample derivation in the language:

$$\begin{aligned} S \Rightarrow aA \Rightarrow aaS \Rightarrow aabB \Rightarrow aabbbS \Rightarrow aabbbaA \\ \Rightarrow aabbbaaS \Rightarrow aabbbaa \end{aligned}$$

We can see that each time S expands, it produces either *aaS* or *bbbS*; thus the reader should convince themself that this language corresponds to the regular expression $(aa \cup bbb)*$.

We will not present the proof that a language is regular if and only if it is generated by a regular grammar; it was first proved by Chomsky and Miller (1958) and can be found in textbooks like Hopcroft and Ullman (1979) and Lewis and Papadimitriou (1988). The intuition is that since the non-terminals are always at the right or left edge of a rule, they can be processed iteratively rather than recursively.

mildly context-sensitive

The fifth class of languages and grammars that is useful to consider is the **mildly context-sensitive grammars** and the **mildly context-sensitive languages**. Mildly context-sensitive languages are a proper subset of the context-sensitive languages, and a proper superset of the context-free languages. The rules for mildly context-sensitive languages can be described in a number of ways; indeed it turns out that various grammar formalisms, including Tree-Adjoining Grammars (Joshi, 1985), Head Grammars Pollard (1984), Combinatory Categorial Grammars (CCG), (Steedman, 1996, 2000) and also a specific version of Minimalist Grammars (Stabler, 1997), are all weakly equivalent (Joshi et al., 1991).

16.2 How to Tell if a Language Isn't Regular

How do we know which type of rules to use for a given problem? Could we use regular expressions to write a grammar for English? Or do we need to use context-free rules or even context-sensitive rules? It turns out that for formal languages there are methods for deciding this. That is, we can say for a given formal language whether it is representable by a regular expression, or whether it instead requires a context-free grammar, and so on.

So if we want to know if some part of natural language (the phonology of English, let's say, or perhaps the morphology of Turkish) is representable by a certain class of grammars, we need to find a formal language that models the relevant phenomena and figure out which class of grammars is appropriate for this formal language.

Why should we care whether (say) the syntax of English is representable by a regular language? One main reason is that we'd like to know which type of rule to use in writing computational grammars for English. If English is regular, we would write regular expressions, and use efficient automata to process the rules. If English is context-free, we would write context-free rules and use the CKY algorithm to parse sentences, and so on.

Another reason to care is that it tells us something about the formal properties of different aspects of natural language; it would be nice to know where a language "keeps" its complexity; whether the phonological system of a language is simpler than the syntactic system, or whether a certain kind of morphological system is inherently simpler than another kind. It would be a strong and exciting claim, for example, if we could show that the phonology of English was capturable by a finite-state machine rather than the context-sensitive rules that are traditionally used; it would mean that English phonology has quite simple formal properties. Indeed, this fact was shown by Johnson (1972), and helped lead to the modern work in finite-state methods shown in Chapters 3 and 4.

16.2.1 The Pumping Lemma

The most common way to prove that a language is regular is to actually build a regular expression for the language. In doing this we can rely on the fact that the regular languages are closed under union, concatenation, Kleene star, complementation, and intersection. We saw examples of union, concatenation, and Kleene star in Ch. 2. So if we can independently build a regular expression for two distinct parts of a language, we can use the union operator to build a regular expression for the whole language, proving that the language is regular.

Pumping Lemma

Sometimes we want to prove that a given language is *not* regular. An extremely useful tool for doing this is the **Pumping Lemma**. There are two intuitions behind this lemma. (Our description of the pumping lemma draws from Lewis and Papadimitriou (1988) and Hopcroft and Ullman (1979).) First, if a language can be modeled by a finite automaton with a finite number of states, we must be able to decide with a bounded amount of memory whether any string was in the language or not. This amount of memory can be different for different automata, but for a given automaton it can't grow larger for different strings (since a given automaton has a fixed number of states). Thus the memory needs must not be proportional to the length of the input. This means for example that languages like $a^n b^n$ are not likely to be regular, since we would need some way to remember what n was in order to make sure that there were an equal number of a's and b's. The second intuition relies on the fact that if a regular language has any long strings (longer than the number of states in the automaton), there must be some sort of loop in the automaton for the language. We can use this fact by showing that if a language *doesn't* have such a loop, then it can't be regular.

Let's consider a language L and the corresponding deterministic FSA M, which has

N states. Consider an input string also of length N. The machine starts out in state q_0; after seeing 1 symbol it will be in state q_1; after N symbols it will be in state q_n. In other words, a string of length N will go through $N+1$ states (from q_0 to q_N). But there are only N states in the machine. This means that at least two of the states along the accepting path (call them q_i and q_j) must be the same. In other words, somewhere on an accepting path from the initial to final state, there must be a loop. Fig. 16.3 shows an illustration of this point. Let x be the string of symbols that the machine reads on going from the initial state q_0 to the beginning of the loop q_i. y is the string of symbols that the machine reads in going through the loop. z is the string of symbols from the end of the loop (q_j) to the final accepting state (q_N).

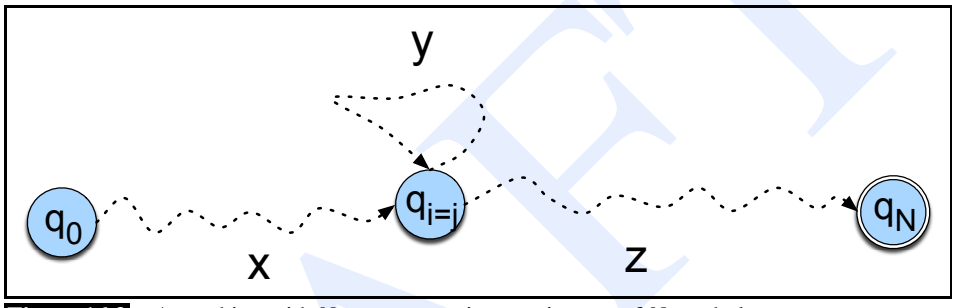

Figure 16.3 A machine with N states accepting a string xyz of N symbols

The machine accepts the concatenation of these three strings of symbols, that is, xyz. But if the machine accepts xyz it must accept xz! This is because the machine could just skip the loop in processing xz. Furthermore, the machine could also go around the loop any number of times; thus it must also accept $xyyz$, $xyyyz$, $xyyyyz$, and so on. In fact, it must accept any string of the form xy^nz for $n \geq 0$.

The version of the pumping lemma we give is a simplified one for infinite regular languages; stronger versions can be stated that also apply to finite languages, but this one gives the flavor of this class of lemmas:

> **Pumping Lemma**. Let L be an infinite regular language. Then there are strings x, y, and z, such that $y \neq \epsilon$ and $xy^nz \in L$ for $n \geq 0$.

The pumping lemma states that if a language is regular, then there is some string y that can be "pumped" appropriately. But this doesn't mean that if we can pump some string y, the language must be regular. Non-regular languages may also have strings that can be pumped. Thus the lemma is not used for showing that a language *is* regular. Rather it is used for showing that a language *isn't* regular, by showing that in some language there is no possible string that can be pumped in the appropriate way.

Let's use the pumping lemma to show that the language a^nb^n (i.e., the language consisting of strings of as followed by an equal number of bs) is not regular. We must show that any possible string s that we pick cannot be divided up into three parts x, y, and z such that y can be pumped. Given a random string s from a^nb^n, we can distinguish three ways of breaking s up, and show that no matter which way we pick, we cannot find some y that can be pumped:

1. y is composed only of as. (This implies that x is all as too, and z contains all the

*b*s, perhaps preceded by some *a*s.) But if y is all *a*s, that means xy^nz has more *a*s than xyz. But this means it has more *a*s than *b*s, and so cannot be a member of the language a^nb^n!

2. y is composed only of *b*s. The problem here is similar to case 1; If y is all *b*s, that means xy^nz has more *b*s than xyz, and hence has more *b*s than *a*s.
3. y is composed of both *a*s and *b*s (this implies that x is only *a*s, while z is only *b*s). This means that xy^nz must have some *b*s before *a*s, and again cannot be a member of the language a^nb^n!

Thus there is no string in a^nb^n that can be divided into x, y, z in such a way that y can be pumped, and hence a^nb^n is not a regular language.

But while a^nb^n is not a regular language, it is a context-free language. In fact, the context-free grammar that models a^nb^n only takes two rules! Here they are:

$$S \rightarrow a\ S\ b$$
$$S \rightarrow \epsilon$$

Here's a sample parse tree using this grammar to derive the sentence *aabb*:

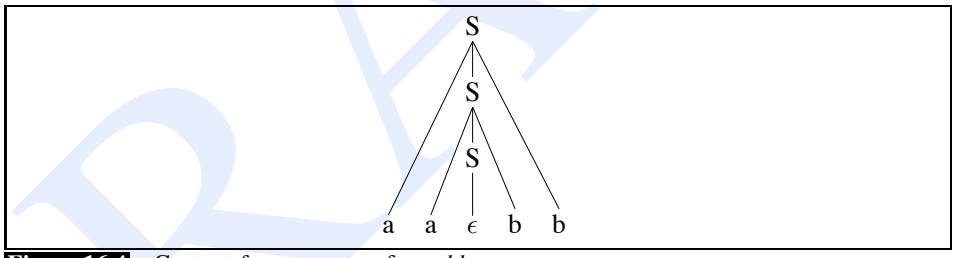

Figure 16.4 Context-free parse tree for *aabb*.

There is also a pumping lemma for context-free languages, that can be used whether or not a language is context-free; complete discussions can be found in Hopcroft and Ullman (1979) and Partee et al. (1990).

16.2.2 Are English and Other Natural Languages Regular Languages?

> *"How's business?" I asked.*
> *"Lousy and terrible." Fritz grinned richly. "Or I pull off a new deal in the next month or I go as a gigolo,"*
> "Either *...or ...," I corrected, from force of professional habit.*
> *"I'm speaking a lousy English just now," drawled Fritz, with great self-satisfaction. "Sally says maybe she'll give me a few lessons."*
>
> Christopher Isherwood, "Sally Bowles", from *Goodbye to Berlin*. 1935

Consider a formal version of the English language modeled as a set of strings of words. Is this language a regular language? It is generally agreed that natural languages like English, viewed in this way, are not regular, although most attempted proofs of this are well-known to be incorrect.

One kind of argument that is often made informally is that English number agreement cannot be captured by a regular grammar, because of the potentially unbounded distance between the subject and the verb in sentences like these:

(16.1) Which *problem* did your professor say she thought *was* unsolvable?

(16.2) Which *problems* did your professor say she thought *were* unsolvable?

In fact, a simple regular grammar *can* model number agreement, as Pullum and Gazdar (1982) show. Here's their regular (right-linear) grammar that models these sentences:

$$
\begin{aligned}
S &\rightarrow \text{Which problem did your professor say T} \\
S &\rightarrow \text{Which problems did your professor say U} \\
T &\rightarrow \text{she thought T} \mid \text{you thought T} \mid \text{was unsolvable} \\
U &\rightarrow \text{she thought U} \mid \text{you thought U} \mid \text{were unsolvable}
\end{aligned}
$$

So a regular grammar could model English agreement. This grammar isn't elegant, and would have a huge explosion in the number of grammar rules, but that's not relevant to the question of the regularity or non-regularity of English.

Another common flaw with previously attempted proofs, pointed out by Mohri and Sproat (1998), is that the fact that a language L contains a subset L' at position P' in the Chomsky hierarchy does not imply that the language L is also at position P'. For example, a regular language can contain as a proper subset a context-free language. Thus the following two languages are context-free

$$L_1 = \{a^n b^n \ : n \in N\} \tag{16.3}$$

$$L_2 = \{ww^R \ : w \in \Sigma^*\} \tag{16.4}$$

and yet both L_1 and L_2 are contained in the regular language L:

$$L = \{a^p b^q \ : p,q \in N\} \tag{16.5}$$

Thus, the fact that a language L contains a sublanguage that is very complex says nothing about the overall complexity of language L.

There are correct proofs that English (or rather "the set of strings of English words considered as a formal language") is not a regular language, based on the pumping lemma. A proof by Partee et al. (1990), for example, is based on a famous class of sentences with **center-embedded** structures (Yngve, 1960); here is a variant of these sentences:

center-embedded

The cat likes tuna fish.
The cat the dog chased likes tuna fish.
The cat the dog the rat bit chased likes tuna fish.

The cat the dog the rat the elephant admired bit chased likes tuna fish.

These sentences get harder to understand as they get more complex. For now, let's assume that the grammar of English allows an indefinite number of embeddings. Then in order to show that English is not regular, we need to show that languages with sentences like these are isomorphic to some non-regular language. Since every fronted *NP* must have its associated verb, these sentences are of the form:

(the + noun)^n (transitive verb)$^{n-1}$ likes tuna fish.

The idea of the proof will be to show that sentences of these structures can be produced by intersecting English with a regular expression. We will then use the pumping lemma to prove that the resulting language isn't regular.

In order to build a simple regular expression that we can intersect with English to produce these sentences, we define regular expressions for the noun groups (A) and the verbs (B):

A = { the cat, the dog, the rat, the elephant, the kangaroo,... }
B = { chased, bit, admired, ate, befriended, ... }

Now if we take the regular expression `/A* B* likes tuna fish/` and intersect it with English (considered as a set of strings), the resulting language is:

$$L = x^n y^{n-1} \textit{ likes tuna fish}, \;\; x \in A, y \in B$$

This language L can be shown to be non-regular via the pumping lemma (see Exercise 2). Since the intersection of English with a regular language is not a regular language, English cannot be a regular language either (since the regular languages are closed under intersection).

There is a well-known flaw, or at least an overly strong assumption with this proof, which is the assumption that these structures can be nested indefinitely. Sentences of English are clearly bounded by some finite length; perhaps we can safely say that all sentences of English are less than a billion words long. If the set of sentences is finite, then all natural languages are clearly finite-state. This is a flaw with all such proofs about the formal complexity of natural language. We will ignore this objection for now, since conveniently imagining that English has an infinite number of sentences can prove enlightening in understanding the properties of finite English.

A more worrisome potential flaw with this proof is that it depends on the assumption that these double relativizations of objects are strictly grammatical (even if hard to process). The research of Karlsson (2007) suggests that, while some kinds of center-embeddings are grammatical, these double relativizations of objects are in fact ungrammatical. In any case, sentences like this get hard much faster than a billion words, and are difficult to understand after a couple nestings. We will return to this issue in Sec. 16.4.

16.3 Is Natural Language Context-Free?

The previous section argued that English (considered as a set of strings) doesn't seem like a regular language. The natural next question to ask is whether English is a context-free language. This question was first asked by Chomsky (1956), and has an interesting history; a number of well-known attempts to prove English and other languages non-context-free have been published, and all except two have been disproved after publication. One of these two correct (or at least not-yet disproved) arguments derives from the syntax of a dialect of Swiss German; the other from the morphology of Bambara, a Northwestern Mande language spoken in Mali and neighboring countries (Culy, 1985). The interested reader should see Pullum (1991, pp. 131–146) for an extremely witty history of both the incorrect and correct proofs; this section will merely summarize one of the correct proofs, the one based on Swiss German.

Both of the correct arguments, and most of the incorrect ones, make use of the fact that the following languages, and ones that have similar properties, are not context-free:

$$\{xx \mid x \in \{a,b\}^*\} \tag{16.6}$$

This language consists of sentences containing two identical strings concatenated. The following related language is also not context-free:

$$a^n b^m c^n d^m \tag{16.7}$$

The non-context-free nature of such languages can be shown using the pumping lemma for context-free languages.

cross-serial dependencies

The attempts to prove that the natural languages are not a subset of the context-free languages do this by showing that natural languages have a property of these xx languages called **cross-serial dependencies**. In a cross-serial dependency, words or larger structures are related in left-to-right order as shown in Fig. 16.5. A language that has arbitrarily long cross-serial dependencies can be mapped to the xx languages.

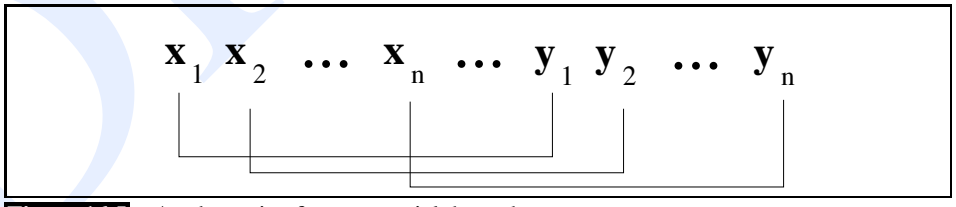

Figure 16.5 A schematic of a cross-serial dependency.

The successful proof, independently proposed by Huybregts (1984) and Shieber (1985a) (as we might expect from the prevalence of multiple discovery in science; see page 13) shows that a dialect of Swiss German spoken in Zürich has cross-serial constraints which make certain parts of that language equivalent to the non-context-free language $a^n b^m c^n d^m$. The intuition is that Swiss German allows a sentence to have a string of dative nouns followed by a string of accusative nouns, followed by a string of dative-taking verbs, followed by a string of accusative-taking verbs.

We will follow the version of the proof presented in Shieber (1985a). First, he notes that Swiss German allows verbs and their arguments to be ordered cross-serially. Assume that all the example clauses we present below are preceded by the string *"Jan säit das"* ("Jan says that"):

(16.8) *...mer em Hans es huus hälfed aastriiche.*
...we Hans/DAT the house/ACC helped paint.
"...we helped Hans paint the house."

Notice the cross-serial nature of the semantic dependency: both nouns precede both verbs, and *em Hans* (Hans) is the argument of *hälfed* (helped) while *es huus* (the house) is the argument of *aastriiche* (paint). Furthermore, there is a cross-serial case dependency between the nouns and verbs; *hälfed* (helped) requires the dative, and *em Hans* is dative, while *aastriiche* (paint) takes the accusative, and *es huus* (the house) is accusative.

Shieber points out that this case marking can occur even across triply embedded cross-serial clauses like the following:

(16.9) *...mer d'chind em Hans es huus haend wele laa*
...we the children/ACC Hans/DAT the house/ACC have wanted to let
hälfe aastriiche.
help paint.
"...we have wanted to let the children help Hans paint the house."

Shieber notes that among such sentences, those with all dative NPs preceding all accusative NPs, and all dative-subcategorizing V's preceding all accusative-subcategorizing V's are acceptable.

(16.10) Jan säit das mer (d'chind)* (em Hans)* es huus haend wele laa* hälfe*
aastriche.

Let's call the regular expression above R. Since it's a regular expression (you see it only has concatenation and Kleene stars) it must define a regular language, and so we can intersect R with Swiss German, and if the result is context-free, so is Swiss German.

But it turns out that Swiss German requires that the number of verbs requiring dative objects (*hälfe*) must equal the number of dative NPs (*em Hans*) and similarly for accusatives. Furthermore, an arbitrary number of verbs can occur in a subordinate clause of this type (subject to performance constraints). This means that the result of intersecting this regular language with Swiss German is the following language:

(16.11) L = Jan säit das mer (d'chind)^n(em Hans)^m es huus haend wele (laa)^n (hälfe)^m
aastriiche.

But this language is of the form $wa^nb^mxc^nd^my$, which is not context-free!
So we can conclude that Swiss German is not context-free.

16.4 Complexity and Human Processing

We noted in passing earlier that many of the sentences that were used to argue for the non-finite-state nature of English (like the "center-embedded" sentences) are quite difficult to understand. If you are a speaker of Swiss German (or if you have a friend who is), you will notice that the long cross-serial sentences in Swiss German are also rather difficult to follow. Indeed, as Pullum and Gazdar (1982) point out,

> precisely those construction-types that figure in the various proofs that English is not context-free appear to cause massive difficulty in the human processing system...

This brings us to a second use of the term **complexity**. In the previous section we talked about the complexity of a language. Here we turn to a question that is as much psychological as computational: the complexity of an individual sentence. Why are certain sentences hard to comprehend? Can this tell us anything about computational processes?

Many things can make a sentence hard to understand. For example we saw in Ch. 14 that a word is read more slowly if it is unpredictable; i.e., has a low N-gram probability or a low parse probability. We also saw in Ch. 14 **garden-path sentences** where ambiguity can cause difficulty; if there are multiple possible parses, a human reader (or listener) sometimes chooses the incorrect parse, leading to a double-take when switching back to the other parse. Other factors that affect sentence difficulty include implausible meanings and bad handwriting.

Another kind of difficulty seems to be related to human memory limitations, and it is this particular kind of complexity (often called "linguistic complexity" or "syntactic complexity") that bears an interesting relation to the formal-language complexity from the previous section.

Consider these sentences from Gibson (1998) that cause difficulties when people try to read them (we will use the # to mean that a sentence causes extreme processing difficulty). In each case the (ii) example is significantly more complex than the (i) example:

(16.12) (i) The cat likes tuna fish.
(ii) #The cat the dog the rat the goat licked bit chased likes tuna fish.

(16.13) (i) The child damaged the pictures which were taken by the photographer who the professor met at the party.
(ii) #The pictures which the photographer who the professor met at the party took were damaged by the child.

(16.14) (i) The fact that the employee who the manager hired stole office supplies worried the executive.
(ii) #The executive who the fact that the employee stole office supplies worried hired the manager.

The earliest work on sentences of this type noticed that they all exhibit *nesting* or *center-embedding* (Chomsky, 1957; Yngve, 1960; Chomsky and Miller, 1963; Miller and Chomsky, 1963). That is, they all contain examples where a syntactic category A is nested within another category B, and surrounded by other words (X and Y):

$[_B$ X $[_A]$ Y]

In each of the examples above, part (i) has zero or one embedding, while part (ii) has two or more embeddings. For example in (16.12ii) above, there are three reduced relative clauses embedded inside each other:

(16.15) # $[_S$ The cat $[_{S'}$ the dog $[_{S'}$ the rat $[_{S'}$ the elephant admired] bit] chased] likes tuna fish].

In (16.13ii), the relative clause *who the professor met at the party* is nested in between *the photographer* and *took*. The relative clause *which the photographer . . . took* is then nested between *The pictures* and *were damaged by the child*.

(16.16) #The pictures [which the photographer [who the professor met at the party] took] were damaged by the child.

The difficulty with these nested structures is not caused by ungrammaticality, since the structures that are used in the complex sentences in (16.12ii)–(16.14ii) are the same ones used in the easier sentences (16.12i)–(16.14i). The difference between the easy and complex sentences seems to relate to the number of embeddings. But there is no natural way to write a grammar that allows N embeddings but not $N+1$ embeddings. Rather, the complexity of these sentences seems to be a processing phenomenon; some fact about the human parsing mechanism is unable to deal with these kinds of multiple nestings, in English and in other languages (Cowper, 1976; Babyonyshev and Gibson, 1999).

The difficulty of these sentences seems to have something to do with *memory limitations*. Early formal grammarians suggested that this might have something to do with how the parser processed embeddings. For example Yngve (1960) suggested that the human parser is based on a limited-size stack, and that the more incomplete phrase-structure rules the parser needs to store on the stack, the more complex the sentence.

self-embedded

Miller and Chomsky (1963) hypothesized that **self-embedded** structures are particularly difficult. A self-embedded structure contains a syntactic category A nested within another example of A, and surrounded by other words (x and y below); such structures might be difficult because a stack-based parser might confuse two copies of the rule on the stack.

The intuitions of these early models are important, although we no longer believe that the complexity problems have to do with an actual stack. For example, we now know that there are complexity differences between sentences that have the same number of embeddings, such as the well-known difference between subject-extracted relative clauses (16.17ii) and object-extracted relative clauses (16.17i):

(16.17) (i) $[_S$ The reporter $[_{S'}$ who $[_S$ the senator attacked]] admitted the error].
(ii) $[_S$ The reporter $[_{S'}$ who $[_S$ attacked the senator]] admitted the error].

The object-extracted relative clauses are more difficult to process, as measured for example by the amount of time it takes to read them, and other factors (MacWhinney, 1977, 1982; MacWhinney and Csaba Pléh, 1988; Ford, 1983; Wanner and Maratsos, 1978; King and Just, 1991; Gibson, 1998). Indeed, Karlsson (2007) has shown in a

study of seven languages that the grammaticality of center embeddings depends a lot on the particular syntactic structure (e.g., relative clauses versus double relativization of objects) being embedded. Another problem for the old-fashioned stack-based models is the fact that discourse factors can make some doubly nested relative clauses easier to process, such as the following double nested example:

(16.18) The pictures [that the photographer [who I met at the party] took] turned out very well.

What seems to make this structure less complex is that one of the embedded NPs is the word *I*; pronouns like *I* and *you* seem to be easier to process, perhaps because they do not introduce a new entity to the discourse.

One human parsing model that accounts for all of this data is the Dependency Locality Theory (Gibson, 1998, 2003). The intuition of the DLT is that object relatives are difficult because they have two nouns that appear before any verb. The reader must hold on to these two nouns without knowing how they will fit into the sentences.

More specifically, the DLT proposes that the processing cost of integrating a new word *w* is proportional to the distance between *w* and the syntactic item with which *w* is being integrated. Distance is measured not just in words, but in how many new phrases or discourse referents have to be held in memory at the same time. Thus the memory load for a word is higher if there have been many intervening *new discourse referents* since the word has been predicted. Thus the DLT predicts that a sequence of NPs can be made easier to process if one of them is a pronoun that is already active in the discourse, explaining (16.18).

In summary, the complexity of these 'center-embedded' and other examples does seem to be related to memory, although not in as direct a link to parsing stack size as was first thought 40 years ago. Understanding the relationship between these memory factors and the statistical parsing factors mentioned in Ch. 14 is an exciting research area that is just beginning to be investigated.

16.5 Summary

This chapter introduced two different ideas of **complexity**: the complexity of a formal language, and the complexity of a human sentence.

- Grammars can be characterized by their **generative power**. One grammar is of greater generative power or **complexity** than another if it can define a language that the other cannot define. The **Chomsky hierarchy** is a hierarchy of grammars based on their generative power. It includes **Turing equivalent**, **context-sensitive**, **context-free**, and **regular** grammars.
- The **pumping lemma** can be used to prove that a given language is **not regular**. English is not a regular language, although the kinds of sentences that make English non-regular are exactly those that are hard for people to parse. Despite many decades of attempts to prove the contrary, English does, however, seem to be a context-free language. The syntax of Swiss-German and the morphology of

Bambara, by contrast, are not context-free and seem to require mildly context-sensitive grammars.

- Certain **center-embedded** sentences are hard for people to parse. Many theories agree that this difficulty is somehow caused by **memory limitations** of the human parser.

Bibliographical and Historical Notes

Chomsky (1956) first asked whether finite-state automata or context-free grammars were sufficient to capture the syntax of English. His suggestion in that paper that English syntax contained "examples that are not easily explained in terms of phrase structure" was a motivation for his development of syntactic transformations.

Choksky's proof was based on the language $\{xx^R : x \in \{a,b\}^*\}$. x^R means "the reverse of x", so each sentence of this language consists of a string of as and bs followed by the reverse or "mirror image" of the string. This language is not regular; Partee et al. (1990) shows this by intersecting it with the regular language aa^*bbaa^*. The resulting language is $a^n b^2 a^n$; it is left as an exercise for the reader (Exercise 3) to show that this is not regular by the pumping lemma.

Chomsky proof shows that English had mirror-like properties, relying on multiple embeddings of the following English syntactic structures, where $S_1, S_2, \ldots, S_n$ are declarative sentences in English,

- If S_1, then S_2
- Either S_3, or S_4
- The man who said S_5 is arriving today

See Chomsky (1956) for details.

Pullum (1991, pp. 131–146) is the definitive historical study of research on the non-context-free-ness of natural language. The early history of attempts to prove natural languages non-context-free is summarized in Pullum and Gazdar (1982). The pumping lemma was originally presented by Bar-Hillel et al. (1961), who also offer a number of important proofs about the closure and decidability properties of finite-state and context-free languages. Further details, including the pumping lemma for context-free languages (also due to Bar-Hillel et al. (1961)) can be found in a textbook in automata theory such as Hopcroft and Ullman (1979).

Yngve's idea that the difficulty of center-embedded sentences could be explained if the human parser was finite-state was taken up by Church (1980) in his master's thesis. He showed that a finite-state parser that implements this idea could also explain a number of other grammatical and psycholinguistic phenomena. While the cognitive modeling field has turned toward more sophisticated models of complexity, Church's work can be seen as the beginning of the return to finite-state models in speech and language processing that characterized the 1980s and 1990s.

There are a number of other ways of looking at complexity that we didn't have space to go into here. One is whether language processing is NP-complete. **NP-**

NP-complete

complete is the name of a class of problems which are suspected to be particularly difficult to process. Barton et al. (1987) prove a number of complexity results about the NP-completeness of natural language recognition and parsing. Among other things, they showed that

1. Maintaining lexical and agreement feature ambiguities over a potentially infinite-length sentence causes the problem of recognizing sentences in some unification-based formalisms like Lexical-Functional Grammar to be NP-complete.
2. Two-level morphological parsing (or even just mapping between lexical and surface form) is also NP-complete.

Recent work has also begun to link processing complexity with information-theoretic measures like Kolmogorov complexity (Juola, 1999).

Finally, recent work has looked at the expressive power of different kinds of probabilistic grammars, showing for example that weighted context-free grammars (in which each rule has a weight) and probabilistic context-free grammars (in which the weights of the rules for a non-terminal must sum to 1) are equally expressive (Smith and Johnson, 2007; Abney et al., 1999a; Chi, 1999).

Exercises

16.1 Is the language $a^n b^2 a^n$ context-free?

16.2 Use the pumping lemma to show this language is not regular:

$$L = x^n y^{n-1} \textit{likes tuna fish}, x \in A, y \in B$$

16.3 Partee et al. (1990) showed that the language $xx^R, x \in a, b*$ is not regular, by intersecting it with the regular language aa^*bbaa^*. The resulting language is $a^n b^2 a^n$. Use the pumping lemma to show that this language is not regular, completing the proof that $xx^R, x \in a, b*$ is not regular.

16.4 Build a context-free grammar for the language

$$L = \{xx^R | x \in a, b*\}$$