Formal Methods in Experimental Syntax

Tim Hunter
timhunter@ucla.edu

May 10, 2018

Draft — Please do not cite
To appear in J. Sprouse (ed.), *The Oxford Handbook of Experimental Syntax*

Taking the theme of this handbook to be expanding the evidentiary basis on which theories of natural language syntax can be empirically evaluated, the goal of this chapter is to outline ways in which formal methods can facilitate this expansion. Since bringing any kind of evidence to bear on a theory naturally requires relevant linking hypotheses, expanding the range of useful evidence amounts to expanding the range of linking hypotheses that can be used to expose syntactic hypotheses to the empirical spotlight. This requires both formulating new linking hypotheses and ensuring that the underlying theory to be tested takes a form that these linking hypotheses can engage with. The topic of this chapter, then, is the use of linking hypothesis that require that the underlying theory takes the form of an explicit and self-contained formal grammar. Put differently, it is an overview of the benefits to be had, in the form of greater empirical testability, by formulating one's syntactic theories in this manner. And in keeping with the section of the handbook in which this chapter appears, the linking hypotheses considered here will serve to link syntactic theories to sentence-processing observations.[1]

I will discuss two classes of linking hypotheses, those based on **information-theoretic** complexity metrics (Section 2) and those based on **automata-theoretic** models of parsing (Section 3). Although both require that the underlying grammatical theory being tested takes the form of some formal grammar, they differ in what further details they require that the grammar furnishes. The use of information-theoretic complexity metrics (for example, surprisal or entropy reduction) requires some probability distribution to be defined over the expressions generated by the grammar, but little more, and broadly speaking this tends to amount to a lower "price of entry" than the automata-theoretic approaches impose. The automata-theoretic approaches work at a lower level of abstraction, and require supplementing the grammar with an explicit parsing algorithm (for example, top-down or bottom-up context-free parsing); the payoff for the extra effort involved is arguably that the candidate explanations provided by this approach have an additional causal, mechanistic character.

I will mostly restrict attention to simple, linguistically-inadequate kinds of grammars (e.g. finite state machines and context-free grammars) to illustrate the different kinds of linking hypotheses, and what a grammar must be like in order to mesh with these linking hypotheses. In Section 4 I will briefly outline how something like contemporary minimalist syntax can be put into a form that allows these linking hypotheses to engage with it.

In recent years, it is the information-theoretic approaches that have most frequently and fruitfully been combined with sophisticated grammars of the sort that syntacticians would find the most familiar. This is not due to any fundamental differences in compatibility, but rather simply due to the higher price of entry for the automata-theoretic methods and the extent of our current knowledge. As our understanding improves it is likely that automata-theoretic methods will more frequently be combined with more linguistically-realistic grammars.

Given the focus on foundational (and hopefully widely-applicable) concepts, there will be little or no discussion of either the finer points of contemporary syntax or any empirical details of recent experimental work in psycholinguistics. Of course the broader enterprise that this chapter aims to contribute to will likely get nowhere if these specifics are *always* left aside as they are here. The "back to basics" approach that I adopt is based on a hunch that these foundational ideas, as opposed to specifics, are the ones that are under-represented and under-utilized in current discussions.[2]

[1] See Levy (2013) and Hale (2017) for reviews that provide a broader historical context for many of the topics discussed here, with less focus on the goal of testing hypothesized grammars.

[2] I hope it goes without saying that this hunch could be wrong (but I include this footnote just to be sure).

1 Linking hypotheses, complexity metrics, and the general form of grammar-testing arguments

Most frequently, a grammar is empirically tested by considering the sound-meaning pairings that it generates. The usual way in which a grammar "pairs up" sounds with meanings is by generating a collection of more abstract syntactic objects — in neutral terms, a structural description, but typically something like a tree — each of which has a certain sound/pronunciation and a certain meaning. One way to go about bringing new kinds of observations/evidence to bear on hypothesized grammars is to follow this pattern and seek ways for structural descriptions to be connected to the relevant new kinds of observables, in something like the way they are usually connected to sounds and meanings. For example, a particular structural description can "have" a certain degree of complexity in much the same way that it "has" a sound and a meaning.

To be concrete, let us say that for any structural description t, its sound is $PHON(t)$ and its meaning is $SEM(t)$. Then the usual "linking hypotheses" (although it is unusual to call them that) which we use to test a particular posited mental grammar are something roughly like the following (ignoring many subtleties, such as ambiguities):

(1) A speaker encountering the string $PHON(t)$ will latch onto the meaning $SEM(t)$.

(2) A speaker wishing to express the meaning $SEM(t)$ will produce the string $PHON(t)$.

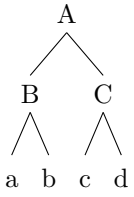
To expand the range of relevant evidence beyond sound-meaning pairings, then, we can consider adding other "lenses onto" the structural description t beyond the existing $PHON$ and SEM.[3] If we introduce some additional function F, such that $F(t)$ is some measure of the complexity of t, and accompany it with a linking hypothesis such as

(3) A speaker encountering the string $PHON(t)$ will experience perceptual difficulty proportional to $F(t)$.

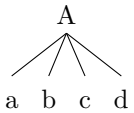
then we have new ways to find evidence for or against the structural descriptions generated by any particular grammar. In particular, it may be that two grammars are indistinguishable when viewed through the lenses of $PHON$ and SEM (via the two initial, conventional linking hypotheses above), but make different predictions about the cases where speakers will experience greater perceptual difficulty (as measure by reading times, or eye-tracking patterns, or whatever).

To take one exceedingly simple example, Miller and Chomsky (1963) suggest that "One rough measure of structural complexity that we might use ... is the node-to-terminal node ratio. ... This number measures roughly the amount of computation per input symbol that must be performed by the listener." They illustrate with the two trees in (4).

(4) a.



b.



Both have four terminal nodes, but (4a) has three non-terminal nodes whereas (4b) has only one. The ratio of nodes to terminal nodes is therefore $\frac{7}{4}$ for (4a), but $\frac{5}{4}$ for (4b). We can therefore imagine hypothesizing that a grammar that posits (4a) as the structural description underlying the string 'a b c d' will predict greater processing load than a grammar that posits (4b) instead.[4] This illustrates the general strategy that we can pursue in order to increase the "empirical payload" of a grammatical theory.

[3] Arguably we already make use of others, such as functions which measure the degree of syntactic well-formedness, or degree of semantic anomaly, etc. And there's no reason the logic here should be restricted to measures of "complexity", it's just been a useful place to start.

[4] To operationalize this, we would need either an additional assumption about the exact relationship between node ratio and some observable measure (e.g. reading time is 100ms times node ratio), or a separate sentence which has a lower node ratio than 'a b c d' does according to one grammar but has a higher node ratio than 'a b c d' according to the other grammar.

A metric such as node ratio, which produces a single number for a complete sentence (actually, for a complete structural description), is a good fit for certain relatively coarse-grained experimental measures of processing load that were used in the early days, such as whole-sentence reading times or accuracy in repeating back a given sentence. But a richer and finer-grained goal would be to identify a metric whose predictions can line up with the observations from more modern experimental paradigms such as self-paced reading, eye-tracking and electrophysiological techniques, where we obtain measures of difficulty at particular points in a sentence. Concretely, rather than $F(t)$ being a single number, $F(t)$ should be some sequence of numbers, one for each word (or other appropriate region), which are taken as measures of the complexity of the work triggered by encountering that particular word of the sentence. I will call a measure like this an *incremental complexity metric*. (Yngve (1960) had one early example; see Hale (2016) for others.)

2 Information-theoretic complexity metrics

One well-known class of incremental complexity metrics are those based on formal concepts from information theory. Broadly speaking, these ideas relate to degrees of uncertainty about future events, and to the way such uncertainty changes in the face of new information. The common intuition behind these complexity metrics is that the amount of work involved in "taking on board" some new information will be some function of the effect that this new information has on uncertainty — for example, the effect that coming to know that the third word of some sentence is 'cat' has on a comprehender's uncertainty about what the sentence is.

As this initial description might suggest, these complexity metrics are very general and abstract. In a sense they don't say things about what a comprehender *does with* this information, only how this new information relates to the information the comprehender already had before. In particular, there is nothing said about *how* information is represented (either the new information or the previously existing information) or how those representations are transformed; but the common assumption that sentences are processed incrementally corresponds roughly to an assumption about *which information* is represented and which points in time.[5]

This high level of abstraction has its benefits and its drawbacks. A drawback is the way they work at a distance from the specifics of the representations and algorithms, as just mentioned. This has the consequence that even when we find a particularly good fit between some collection of data and a theory based on these linking hypotheses, this does not constitute evidence for or against any particular set of underlying nuts-and-bolts mechanisms (of the sort that we will turn to in Section 3). Having said that, a main aim of the discussion that follows is to demonstrate that it still *can* constitute evidence for or against particular grammars. The flip side of this drawback is the advantage that, since a theory incorporating one of these complexity metrics only narrows down our live possibilities by a relatively modest degree, the price of entry for submitting a grammar to testing via these metrics — in terms of technical work and theoretical commitments — is accordingly also modest. A second advantage is that there is the possibility that we might find empirical support for such metrics that is genuinely independent of their use as probes into linguistic questions. At least to the extent that we suppose that there are domain-general facts of the matter about how the mind relates new information to old, the plausibility of these metrics can be independently assessed to a degree that more concrete, language-bound linking hypotheses cannot.

The most notable examples of these complexity metrics are surprisal (e.g. Hale, 2001; Levy, 2008) and entropy reduction (e.g. Hale, 2003, 2006; Yun et al., 2015). Other variants are also easily imaginable. These metrics differ in exactly how they use information-theoretic concepts about uncertainty to formulate a measure of when it is that new information takes a lot of work to take on board; put differently, they differ in exactly what kind of change in uncertainty they take to be indicative of "high workload".

For the purposes of the current chapter, however, these differences are not particularly consequential. The reason for this is that the requirements they impose on what a grammar must look like remain essentially the same: it must be possible to define a probability distribution over the grammar's possible derivations, and to update this distribution according to new partial information about a sentence being observed. The focus here will be on these requirements. Satisfying these is what allows a grammar to be "plugged into" these complexity metrics.

For illustration, I will demonstrate this "plugging in" by taking surprisal as a representative example of an information-theoretic complexity metric. This choice is purely one of convenience: it requires slightly less mathematical groundwork than the alternatives. Hale (2016) gives a thorough evaluation of the empirical and conceptual differences

[5] One might say that these metrics refer only to what Marr (1982) called the "computational level" of description, the most abstract of the three.

between surprisal and entropy reduction. The focus here will be on the way either one of these metrics can be used to bring a hypothesized mental grammar into empirical contact with incremental comprehension measures.

2.1 Surprisal

One well-known example of an incremental complexity metric is surprisal. Generally speaking, the surprisal at the occurrence of a particular event is large if the event was unexpected (had low probability), and is small if the event was expected (had high probability); in the extreme case where there was no alternative but for that particular event to happen (its probability is one), the surprisal is zero. To adopt surprisal as an incremental complexity metric for sentence comprehension, the idea is to take each of the words encountered in a sentence as a separate event, each of which will have its own surprisal value, and the crucial linking step is to hypothesize that we will see evidence of greater processing difficulty/complexity (e.g. reading slowdowns) when speakers encounter words with high surpisal values. We assume that the probability of a word, or the degree to which that word is expected, may differ depending on what other information about the sentence the comprehender already has[6]; accordingly, the relevant probabilities are conditional upon the preceding words of the sentence. The crucial probabilities therefore have the general form:

$$P(W_i = w_i \mid W_1 = w_1, W_2 = w_2, \ldots W_{i-1} = w_{i-1})$$

where W_i is the random variable whose corresponding to the event of encountering the ith word, and w_i is the particular word encountered in the ith position. For example, the probability relevant to calculating surprisal at the word 'chased' in 'the dog chased the cat' is

$$P(W_3 = \text{chased} \mid W_1 = \text{the}, W_2 = \text{dog})$$

The particular function that is applied to a probability of this sort in order to convert it to a surprisal value is the negative logarithm:

$$\text{surprisal at } w_i = -\log P(W_i = w_i \mid W_1 = w_1, W_2 = w_2, \ldots, W_{i-1} = w_{i-1})$$

For the purposes relevant to this article, the important thing to note about this is simply that is has the effect of turning higher probabilities into lower surprisal values, and turning lower probabilities into higher surprisal values: if $P(X) > P(Y)$, then $-\log P(X) < -\log P(Y)$. The particular choice of the negative logarithm function ensures that surprisal works in accord with certain intuitions about how a *measure of information* should behave. One example is the extreme case mentioned above, that if $P(X) = 1$, then $-\log P(X) = 0$, in accord with the intuition that no information has been obtained by observing an event that was certain to happen. A second is that surprisal values can sensibly be added. If we roll a fair four-sided die and a fair eight-sided die, then the surprisal at seeing the four-sided die come up on any particular side is $-\log \frac{1}{4} = 2$ and the surprisal at seeing the eight-sided die come up on any particular side is $-\log \frac{1}{8} = 3$. Summing these two surprisal values ($2 + 3 = 5$) gives the same result as calculating the surprisal from the perspective of the 32 different joint outcomes that were possible upon rolling the two dice together: $-\log \frac{1}{32} = 5$. This accords with our intuition that the amount of information obtained by finding out which side came up on the four-sided die and which side came up on the eight-sided die should be the same as that obtained by finding out which of the 32 joint events occurred. But since the relevant linking hypothesis is typically just the qualitative idea that higher surprisal values correlate with greater comprehension difficulty/load, these numerical details do not (yet) play a large role in the work discussed here and so the significant guiding idea is simply that lower (conditional) probabilities correlate with greater comprehension difficulty/load.

To take a concrete and complete example, suppose we are interested in calculating word-by-word surprisal values for the sentence 'John saw it'. To illustrate the division of labor between the specification of a probability distribution (which will shortly be part of a grammar's contribution) and the calculation of surprisal values from those probabilities, for this first example I will suppose that the comprehenders's expectations about the sentences he or she might encounter are simply specified by a finite lookup table, shown in (5).

(5)

0.4	John ran
0.15	John saw it
0.05	John saw them
0.25	Mary ran
0.1	Mary saw it
0.05	Mary saw them

[6] One could easily calculate surprisal simply based on expectations about individual words without reference to context, but this would imply, for example, the same surprisal value at the word 'dog' in 'The man bit the dog' as at the word 'dog' in 'The dog bit the man'.

At the first word, 'John', there is no other information about the sentence to condition upon, so the relevant probability is simply the sum of all the probabilities of sentences that have 'John' as their first word.

$$\begin{aligned}\text{surprisal at 'John'} &= -\log P(W_1 = \text{John}) \\ &= -\log(0.4 + 0.15 + 0.05) \\ &= -\log 0.6 \\ &= 0.74\end{aligned}$$

At the second word, 'saw', we need to consider probabilities conditioned upon the fact that the first word of the sentence is already known to be 'John'; concretely, this means that we restrict attention to the first three lines of the table in (5), and ask how much of the 0.6 probability mass there lies with sentences that furthermore have 'saw' as their second word. The resulting probability, 0.33, is lower than the relevant probability at the first word, 0.6; accordingly, the surprisal value is higher here (1.58 vs. 0.74), and greater comprehension difficulty at this word is predicted.

0.4	John ran
0.15	John saw it
0.05	John saw them

$$\begin{aligned}\text{surprisal at 'saw'} &= -\log P(W_2 = \text{saw} \mid W_1 = \text{John}) \\ &= -\log \frac{0.15 + 0.05}{0.4 + 0.15 + 0.05} \\ &= -\log 0.33 \\ &= 1.58\end{aligned}$$

Similarly, at the third word we restrict attention to the two lines of the table that are still "in play". The probability of 0.75 is higher than both of the two previous probabilities, and so the surprisal value here is lower than both of the two previous surprisal values.

0.15	John saw it
0.05	John saw them

$$\begin{aligned}\text{surprisal at 'it'} &= -\log P(W_3 = \text{it} \mid W_1 = \text{John}, W_2 = \text{saw}) \\ &= -\log \frac{0.15}{0.15 + 0.05} \\ &= -\log 0.75 \\ &= 0.42\end{aligned}$$

So the adoption of surprisal as a linking hypothesis has taken us from the hypothesized probability distribution in (5) to (what I will take to be[7]) testable predictions concerning the sentence 'John saw it': comprehension difficulty will be greatest at the word 'saw' (surprisal 1.58), lowest at the word 'it' (surprisal 0.42), and in between at the word 'John' (surprisal 0.74). The general idea is illustrated schematically in Figure 1. So surprisal amounts to a way to test the fit of a hypothesized probability distribution with some observations or data (though see footnote 7).

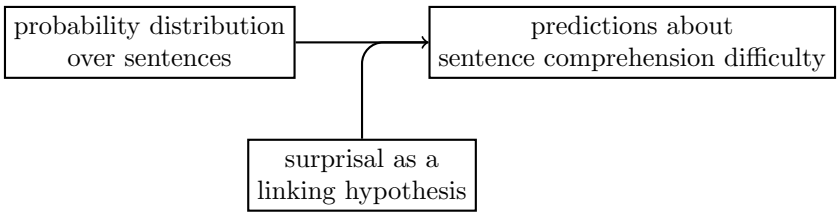


Figure 1: *Surprisal is a linking hypothesis which, taken together with a probability distribution over sentences, produces empirical predictions about sentence comprehension difficulty.*

But if there are unboundedly many relevant sentence-probabilities, then a finite mind cannot directly encode all the probabilities in a table like (5). The probabilities, like the sentences they are associated with, will have to instead be encoded in some finite system of rules. This system of rules would specify some collection of "primitive" probabilities (e.g. that the probability of the first word of a sentence being 'John' is 0.1, or that the probability of

[7]It is perhaps an unreasonable simplification to call these "testable predictions": there are of course still decisions to be made about how comprehension difficulty will be measured (e.g. reading times, electrophysiological responses); how surprisal values are assumed to relate to these measures (e.g. does a twice-as-large surprisal value predict, all else being equal, a twice-as-large reading time?); whether these surprisal values are assumed to be the *sole* predictor of these measures or one of a collection of interacting factors, etc.

a verb being 'run' is 0.2) and recipes for deriving other "composite" probabilities from those. Independent of the particular *numbers* entering into the calculation of sentence's probability, the *structure* of such a system of primitives and recipes constrains the range of probability distributions that can be defined. A natural way to take grammars to be part of what something like surprisal is testing is to realize that they amount to a hypothesis about the structure of this system.

2.2 The role of grammars

The aim in what follows is to illustrate how a grammar can be taken to play a role in defining a probability distribution; as a result, a grammar will play a role in determining surprisal predictions, which means in turn that empirical tests of surprisal predictions can provide evidence for or against hypothesized grammars. The machinery that we have introduced so far is entirely agnostic about the source of the probabilities that enter into these calculations (as illustrated in Figure 1), and this is what gives metrics like surprisal the high degree of versatility mentioned in the introduction: surprisal can be calculated on the basis of probabilities drawn from a finite lookup table as above, or probabilites defined by a relatively simple grammar (e.g. a collection of allowed bigrams, or a finite state grammar, as we will see shortly), or probabilities defined by a more linguistically sophisticated grammar (e.g. a context free grammar or a minimalist grammar). This formalism-neutrality is what makes it possible for information-theoretic metrics like surprisal to act as the playing field on which any two grammars can be pit against each other, no matter how different the two grammars are in their internals; see Figure 2.[8]

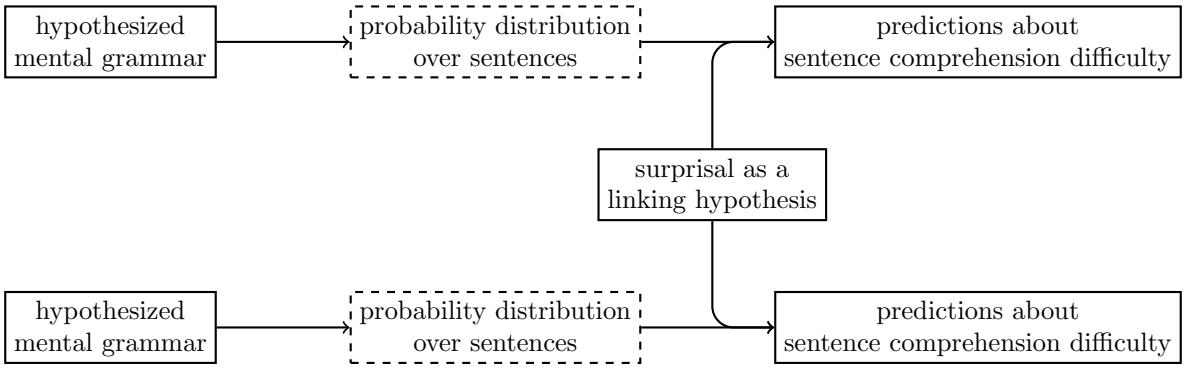


Figure 2: *Since surprisal can act as a test of probability distributions and probability distributions can be seen as consequences of hypothesized grammars, surprisal can act as a test of hypothesized grammars.*

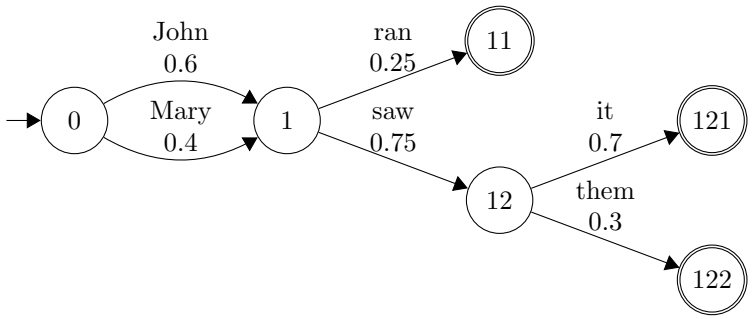
For this kind of argument to work, of course, it must be the case that different grammars have different effects on the probability distributions that go into calculating surprisal. Obviously choosing a particular grammar (in the traditional, non-probabilistic sense) does not pick out a particular probability distribution; but adopting a particular grammar — breaking down the specification of a set of expressions with accompanying probabilities into a particular system of interlocking rules — does *constrain* the range of probability distributions that might arise. For example, if our grammar expresses the assumption that generating a sentence amounts to generating a subject and generating a predicate independently (roughly, think of something like 'S $\rightarrow$ NP VP'), then there will be no way to attach probabilities to a this grammar in a way that assigns the six sentences in (5) the particular probabilities that they have there.[9] To see this, notice that in order to generate 'John saw it' and 'John saw them' with the probabilities shown (0.05 and 0.15, respectively), our grammar would need to generate the predicate 'saw it' with a probability exactly three times greater than the probability of the predicate 'saw them', because these two sentences differ only in the chosen predicate. (We don't know whether these two probabilities should be, for example, 0.3 and 0.1, in which case the probability of the subject 'John' would need to be 0.5; or whether they should be 0.6 and 0.2, with 'John' having probability 0.25, etc. But we do know that they need to stand in this 3:1 ratio.) But also, in order to generate

[8]Note in particular that even though the calculation of surprisal values appeals to the notion of transitioning from one word to the next linearly adjacent word, there is no assumption that the grammatical knowledge that the comprehender brings to the task takes the form of statements about which words can and cannot, or what is likely or unlikely to, linearly follow certain other words (in the manner familiar from n-gram models, for example). The information in the table in (5) does not take this form, but still provides everything we need to know in order to calculate surprisal values. Similarly, the information in grammars that work with hierarchical structures rather than linear structures also provides what we need in order to make surprisal calculations. The linear nature of the metric reflects the linear nature of the external sentence-comprehension task, not any assumptions about the internal knowledge being recruited by the comprehender.

'Mary saw it' and 'Mary saw them' with the probabilities shown, our grammar would need to generate 'saw it' with a probability exactly twice the probability of 'saw them'. So this grammatical assumption about the structure of the sentences commits us to working within a restricted range of probability distributions over that set of six sentences, which excludes the particular probability distribution shown in (5). All else being equal, then, discovering that the surprisal values computed from the probabilities in (5) make correct predictions would constitute evidence *against* analyzing the subject and predicate as independent subparts of a sentence.

To begin to look at concrete examples involving grammars, let us first consider the finite-state automaton in (6). It's useful to begin by looking at finite-state machines, rather than more sophisticated kinds of grammars, for a couple of reasons. The first reason is that it allows us to see clearly that the crucial property of a grammar is the way it factors out the *generative work* into a structured system of interlocking pieces, rather than any particular representations that are built; these two notions are often intertwined in grammars that manipulate tree structures. The second reason is that having seen surprisal calculated on the basis of two distinct grammatical systems — first FSAs, and then PCFGs below — it is easier to get a clear grasp on the idea of surprisal itself, as opposed to its incarnation in relation to any specific kind of grammar, and this in turn makes it easier to understand what it will take to operationalize the idea in whatever particular linguistically sophisticated grammatical system one might wish to (e.g. some version of minimalist syntax).

(6)



This FSA encodes the idea that a sentence is generated by independently generating a subject (corresponding to the choice of how to transition out of the start state) and generating a predicate (corresponding to the choice of how to get from the next state to some end state). It is this decomposition of the generative mechanisms into independent sub-parts that we can think of as attributing *grammatical structure* to the generated expressions; this is what the table in (5) does not do. There are a range of probability distributions that can be defined by putting probabilities on the transitions in (6), and the particular probabilities shown in the diagram pick out one of these. But as mentioned above, the probability distribution in (5) is *not* in this range, since any probability distribution defined on the basis of the FSA in (6) will have the property that

$$\frac{P(\text{John saw it})}{P(\text{John saw them})} = \frac{P(\text{Mary saw it})}{P(\text{Mary saw them})}$$

and this is not true of the distribution in (5). So, leaving aside the *particular* probabilities shown in (6), the example distribution is inconsistent with the hypothesized *grammatical structure* that is encoded in the "shape" of the FSA in (6), and accordingly observations about sentence comprehension difficulty that match up with surprisal values calculated from this distributions constitute (all else being equal) evidence against that hypothesized grammatical structure. (Similarly, observations that are in accord with surprisal predictions based on probability distribution defined by (6) will, all else being equal, constitute evidence in favour of it.) The FSA in (6)'s relative inflexibility with regard to definable probability distributions is a direct consequence of the way it assigns more grammatical structure than the table in (5).

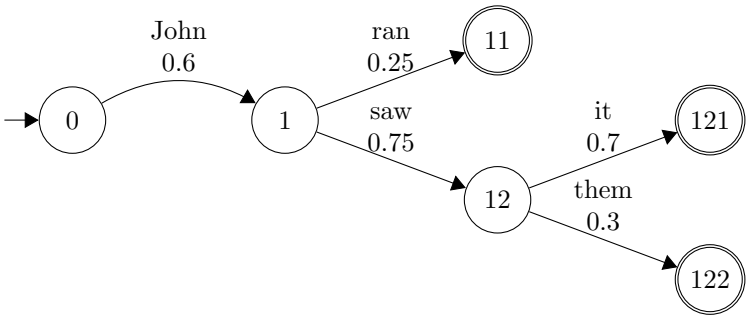
[9]The independence is crucial. I am simplifying things somewhat by taking this *probabilistic independence* assumption to be part and parcel of what is meant by hypothesizing a grammatical rule such as 'S → NP VP'. In principle, the *well-formedness independence* that is expressed by this rule — the fact that the ability of an NP and a VP to combine to form an S is independent of the particular NP and VP that we are considering (think of $1 \times 1 = 1$) — can be taken in isolation from any probabilistic independence assumptions. My (imperfect) justification for running these two things together is that if one takes a hypothesized grammar to be a mental generative mechanism rather than a predictor of well-formedness, then proposing a probabilistic generative mechanism that is independent of the one we call the grammar creates a redundancy and therefore is a move that at the very least requires its own justification. To the extent that we would like to leave this option open, when I say that adopting a particular grammar commits us to a restricted range of probability distributions, we should replace this with the claim that it makes that restricted set of distributions more parsimonious hypotheses than the alternatives.

It is perhaps unusual to invoke any notion of grammatical structure when dealing with FSAs, given their linear nature. If it helps, one can imagine the FSA in (6) being replaced with a series of rules like 'S → John X_1', 'S → Mary X_1', etc. But the significant point, notation aside, is that this grammar makes the meaningful claim that certain pairs of distinct sentences — such as the pair 'John saw it' and 'Mary saw it' — have their predicate part in common. In the familiar tree-structure notation, this surfaces as the claim that those two sentences' tree structures have a certain subtree in common; in the FSA setting, it surfaces as the claim that the state sequences used to generate the two sentences have a certain subsequence in common. The table in (5) does not make any such claim about any two sentences, and it is in this sense that the FSA ascribes more grammatical structure. The bread-and-butter of grammatical analysis is adding claims of this sort, although typically, of course, in more complex grammatical frameworks.

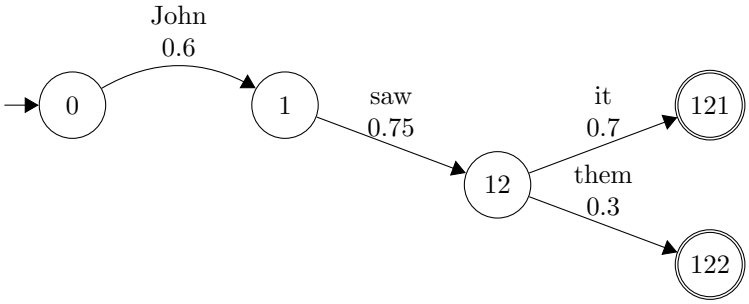
Although this important general idea about generative structure underlies the way any kind of grammar might be used to define a probability distribution, the way in which surprisal values are determined from (the distribution defined by) a grammar will vary from one kind of grammar to the next. It turns out to be rather simple if the grammar takes the form of an FSA.[10] For example, using the FSA in (6) and given the sentence 'John saw it', the surprisal at the word 'it' can be simply "read off" the corresponding arc in the diagram: 0.7.

But for the purposes of understanding how surprisal predictions can be calculated for other kinds of grammars,[11] a different way of thinking about this is more useful. (While the linear nature of FSAs does not disqualify them from serving as a useful illustration of the general notion of "grammatical structure" discussed above, it *does* make the task of extracting surprisal values from them deceptively simple. The goal of working with FSAs first was the make the first issue clear.) Instead of thinking about an ant walking along the arcs in the diagram, such that the surprisal at a particular word is simply the probability labeling that arc that the ant must walk along, a useful more general notion is to think of the grammatical possibilities narrowing down as more information about the sentence being encountered is revealed. In particular, this can take the form of considering a "narrowing down" of the grammar itself: if we take the grammar above and "prune out" parts of the grammar that are not consistent with the first word being 'John', for example, we get the FSA in (7a). Then going one step further, further "pruning" the grammar to generate only sentences that are, in addition, consistent with the second word being 'saw', produces the FSA in (7b). And similarly for the third word, which leaves us with (7c), an FSA which generates only the sentence being processed.

(7) a.



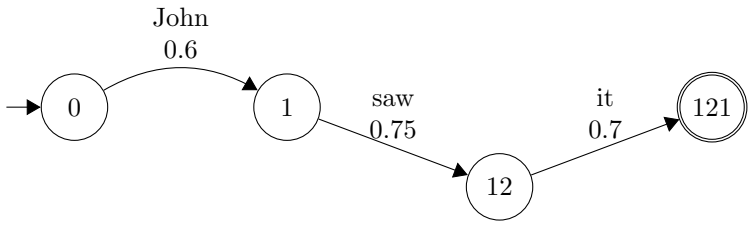
b.



[10] I am restricting attention here to deterministic FSAs, for simplicity. The situation for nondeterministic FSAs is similar in many important respects.

[11] And also for the purposes of understanding how we can calculate the predictions of other similar metrics besides surprisal, such as entropy reduction.

c.



What this provides is a sequence of grammars, each of which (in a certain relatively externalistic, but nonetheless useful sense) characterizes the comprehender's knowledge state at a particular point in the sentence: (7a), for example, represents combining the static "background" knowledge encoded in the original grammar in (6) (grammatical knowledge in the familiar sense), with the knowledge that the first word of the sentence currently being processed is 'John'. More precisely, the set of expressions generated by (7a) is the intersection of (a) the set of expressions generated by the original grammar in (6), and (b) the set of expressions whose pronunciation begins with the word 'John'. For this reason, we can refer to (7a) as an "intersection grammar". Similarly, (7b) is the intersection grammar that brings together the background knowledge in (6) and the information that the first two words are 'John saw'.

Notice now that if we sum the probabilities assigned to all the sentences generated by (6) the total is 1.0, but if we do the same for the subsequent intersection grammars the total is less than 1 — 0.6 in (7a) and $0.6 \times 0.75 = 0.45$ in (7b).[12] The fact that, out of the 1.0 probability mass in (6), there is 0.6 remaining in (7a), is precisely the fact that the probability of a sentence generated by (6) beginning with the word 'John' is 0.6 — which is to say, the fact that the surprisal at the first word is $-\log 0.6$.

$$\text{surprisal at 'John'} = -\log\left(\frac{\text{total probability mass in (7a)}}{\text{total probability mass in (6)}}\right) = -\log\left(\frac{0.6}{1.0}\right) = -\log 0.6 = 0.74$$

And similarly:

$$\text{surprisal at 'saw'} = -\log\left(\frac{\text{total probability mass in (7b)}}{\text{total probability mass in (7a)}}\right) = -\log\left(\frac{0.6 \times 0.75}{0.6}\right) = -\log 0.75 = 0.42$$

$$\text{surprisal at 'it'} = -\log\left(\frac{\text{total probability mass in (7c)}}{\text{total probability mass in (7b)}}\right) = -\log\left(\frac{0.6 \times 0.75 \times 0.7}{0.6 \times 0.75}\right) = -\log 0.7 = 0.52$$

So generally:

$$\text{surprisal at word } i = -\log\frac{\text{total probability mass in } \mathrm{G}_i}{\text{total probability mass in } \mathrm{G}_{i-1}}$$

where G_i is the intersection grammar that combines the comprehender's "static" mental grammar with the first i observed words of the relevant sentence.

The important thing to note about this formulation of surprisal is that it provides us with a recipe for calculating incremental surprisal values on the basis of a probabilistic grammar which does *not* have the "which word comes next?" structure that FSAs have. For any type of grammar one might be interested in (i.e. any system a syntactician might have devised for breaking down the generative work that gives rise to a sentence), as long as we are able to

- intersect a grammar with a given initial portion of a sentence (i.e. the first n words), and
- calculate the total probability mass remaining in such a grammar,

then we will be able to calculate incremental surprisal values from that kind of grammar. And of course, the particular structured arrangement of interlocking pieces that a grammar posits will impose constraints on the definable probability distributions that carry on through each of these intersection grammars, and therefore have an impact on the calculated surprisal values.

It turns out that both of these things can be done with other kinds of grammars that are more powerful than FSAs, such as (probabilistic) context-free grammars (PCFGs). The details for PCFGs are less intuitive, but a couple of

[12] So strictly speaking these intersection grammars do not define their own probability distributions over the generated sentences, they merely encode certain subsets of the distribution defined by (6). To be used with entropy reduction as opposed to surprisal, each of these grammars must be renormalized.

illustrative examples may serve to convey the basic idea. To begin, it is not difficult to see how to intersect the context-free grammar in (8) with the prefix 'John' to produce the grammar in (9), since these two grammars are equivalent to the FSAs in (6) and (7a) respectively. We simply remove the rule that generates 'Mary'. (Although note that things would be more complicated in an FSA where it was possible to return to the start state.)

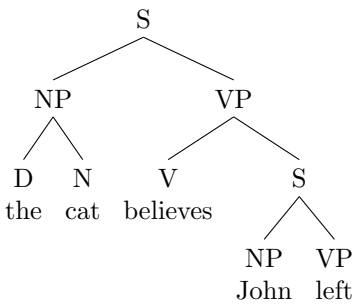
(8)
0.4	X_0	$\rightarrow$	Mary X_1
0.6	X_0	$\rightarrow$	John X_1
0.25	X_1	$\rightarrow$	ran
0.75	X_1	$\rightarrow$	saw X_{12}
0.7	X_{12}	$\rightarrow$	it
0.3	X_{12}	$\rightarrow$	them

(9)
0.6	X_0	$\rightarrow$	John X_1
0.25	X_1	$\rightarrow$	ran
0.75	X_1	$\rightarrow$	saw X_{12}
0.7	X_{12}	$\rightarrow$	it
0.3	X_{12}	$\rightarrow$	them

What distinguishes a CFG from an FSA, however, is its ability to depart from the strictly "right-branching" kind of structure that (8) generates. Consider for example the more interesting grammar in (10). An example of a structure that it generates is shown in (11).

(10)
1.0	S	$\rightarrow$	NP VP
0.6	NP	$\rightarrow$	John
0.4	NP	$\rightarrow$	D N
0.8	D	$\rightarrow$	the
0.2	D	$\rightarrow$	a
0.7	N	$\rightarrow$	cat
0.3	N	$\rightarrow$	dog
0.5	VP	$\rightarrow$	V NP
0.3	VP	$\rightarrow$	V S
0.2	VP	$\rightarrow$	left
1.0	V	$\rightarrow$	believes

(11)



Intersecting this grammar with the one-word prefix 'the' is less straightforward than the first example above, because the remaining portion of the sentence is not required to be a single constituent. (In fact, in this grammar, it will never be a single constituent.) Furthermore, although it is clear from the appearance of 'the' as the first word that some NP node must be expanded according to the rule 'NP $\rightarrow$ D N', we cannot assume that *all* NP nodes will be expanded according to this rule and ignore the other NP rules, in the way that we ignored "roads not taken" in the simpler example.

A mechanical procedure exists for solving these problems, however.[13] The result, for the prefix 'the', is shown in (12). It contains, in addition to all of the "original" rules from (10) (shown on the right), three new rules (shown on the left) which describe how the new nonterminals S′, NP′ and D′ are used. Note also that the start symbol of this new grammar is S′ (not S).

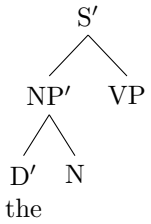
[13] See e.g. Bar-Hillel et al. (1961), Nederhof and Satta (2003).

(12)

1.0	S′	→	NP′ VP	1.0	S	→	NP VP
				0.6	NP	→	John
0.4	NP′	→	D′ N	0.4	NP	→	D N
0.8	D′	→	the	0.8	D	→	the
				0.2	D	→	a
				0.7	N	→	cat
				0.3	N	→	dog
				0.5	VP	→	V NP
				0.3	VP	→	V S
				0.2	VP	→	left
				1.0	V	→	believes

Each of the three new rules can be thought of as a specialized instance of a rule from the original grammar (and they are shown alongside the corresponding original rules in (12)). Since the start symbol of the new grammar is S′, these new rules amount to instances of the original rules that are forced to apply; for example, the NP′ that is the left daughter of the root S′ can *only* be expanded as D′ and N (not as 'John'). The upshot is that any derivation beginning with the new start symbol S′ must produce at least this partial structure:

(13)



The "primed" nonterminals represent nonterminals whose expansion is somehow restricted by the input we have seen so far (in this case, the word 'the'). Notice that since all the primed nonterminals have now been eliminated, and all that remains is to somehow expand an N node and a VP node, the rest of the derivation can proceed with all of the freedom that it would have if we were using the original grammar in (10), i.e. using the rules shown on the right in (12).

Besides constructing these intersection grammars, the second important prerequisite for deriving surprisal values is being able to calculate the total probability mass that is "left" in an intersection grammar such as the one in (12). The three new rules shown on the left have the same probabilities as the original rules they are based on, so the total probability assigned to all sentences generated by this grammar is less than 1 just as it was for the intersection grammars in (7). In the particular case of (12) it is perhaps not difficult to see that total probability assigned by this intersection grammar is $1.0 \times 0.4 \times 0.8$, since these are the probabilities of the three rules that we have been forced to use in order to generate 'the' as the first word. See Nederhof and Satta (2008) for the full details of how this probability can be computed in general.[14]

3 Automata-theoretic parsing models

In this section I will introduce three distinct parsing methods for context-free grammars: bottom-up, top-down and left-corner. The first two, bottom-up and top-down, are very simplistic and not particularly plausible as cognitive models, but they are useful stepping-stones for understanding the more complicated left-corner method, which better corresponds to (at least certain aspects of) human parsing.[15]

All three of the methods discussed here will be expressed as *transition systems*: given the task of parsing a particular sentence using a particular grammar, each method will specify a particular *starting configuration* and a particular *goal*

[14]Nederhof and Satta (2008) frame the problem in a manner that directly matches my presentation here, i.e. finding the total probability mass remaining in an intersection grammar, but other solutions to the general problem of computing prefix probabilities from a PCFG were given by Jelinek and Lafferty (1991) and Stolcke (1995). I adopt the presentation based on intersection grammars here because it generalizes well to other metrics besides surprisal (e.g. entropy reduction), and to other grammar formalisms besides CFGs.

[15]The content of this section is very similar to that of (at least) Abney and Johnson (1991), Resnik (1992), Wolf and Gibson (2006) and Kanazawa (2016, lecture 2). The core underlying ideas go back to Chomsky and Miller (1963), Chomsky (1963) and Miller and Chomsky (1963).

configuration; and in addition, will specify the allowable *transitions* by which we can progress from one configuration to another. If there is a sequence of transitions that leads from the starting configuration to the goal configuration, then the sentence is grammatical (and the sequence of transitions determines its structural description); otherwise, it is ungrammatical (i.e. has no structural description).

Note that what I am describing here as *parsing method* takes both a sentence and a grammar as "inputs". The parsing methods themselves make no reference to particular words or particular grammatical categories (such as NP or VP); rather, they are able to operate with whatever context-free grammar one provides. This contrasts with the way the term "parser" is sometimes used, namely to refer to a mechanism that takes only a sentence as an input and attempts to find an analysis for that sentence according to some grammar that is implicitly specified within the workings of that mechanism.

What these parsing methods provide is something like a "Marr (1982) level two" description of a mechanism for processing sentences. These algorithmic details provide plenty of hooks on which we can hang various linking hypotheses. The simple one that I will end up appealing to in this section is based on the idea that comprehension difficulty arises when the amount of information that needs to be simultaneously stored in order to grammatically analyze a sentence is large.

Throughout this section I will use the grammar in (14) for illustration. In presenting the general schemas for the various parsing methods, I will assume that the right hand side of each grammar rule is either (a) a single terminal symbol, or (b) a sequence of nonterminal symbols.[16] I will also assume, as is common, that the start symbol is always S.

(14)

S → NP VP	N → dog, cat, rat, wife, brother
S → WHILE S S	NP → John, Mary
NP → NP POSS N	V → barked, chased, bit, ate, fled
NP → (D) N (PP) (SRC) (ORC)	D → the
VP → V (NP) (PP)	P → on, in, with
PP → P NP	C → that
SRC → C VP	POSS → 's
ORC → NP V	WHILE → while

As a sort of motivating running example, and as an aid to understanding the differences between the three parsing methods being discussed, we'll consider ways to account for the pattern of human comprehension difficulty on left-embedded, right-embedded and center-embedded structures: while left-embedding and right-embedding structures can increase in depth apparently without bound and still remain easily comprehensible, center-embedding structures quickly become incomprehensible. I'll use the sentences in (15), (16) and (17) as illustrative examples for this point. (These particular sentences don't make a well-designed set of controlled experimental stimuli, but will allow for an easy illustration of the key ideas using the simple grammar in (14) above.)

(15) Left-branching structures
   a. John fled
   b. John 's dog fled
   c. John 's wife 's dog fled

(16) Right-branching structures
   a. Mary chased the cat
   b. Mary chased the cat that bit the rat
   c. Mary chased the cat that bit the rat that ate the cheese

(17) Center-embedding structures
   a. the rat fled
   b. the rat the cat chased fled
   c. the rat the cat the dog bit chased fled

[16] Any context-free grammar can be straightforwardly converted into this form, via the introduction of "singleton" nonterminals like WHILE in (14).

Specifically, I will aim to illustrate account of the fact that there is an increase in comprehension difficulty in (17c) relative to (17b), but no such increase in (15b)/(15c) or (16b)/(16c). (I leave aside any facts about the 'a.' sentences, which are shown just to demonstrate the sense in which these three pair-wise comparisons are analogous.) In concrete terms, the aim will be to find some function F such that

$$F(\text{tree for (15b)}) = F(\text{tree for (15c)})$$
$$F(\text{tree for (16b)}) = F(\text{tree for (16c)})$$
$$F(\text{tree for (17b)}) < F(\text{tree for (17c)})$$

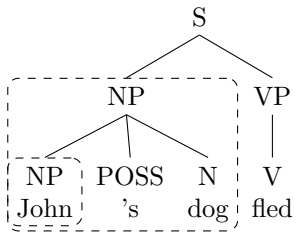
such that in combination with the linking hypothesis that

(18) A speaker processing the string $PHON(t)$ will experience greater perceptual difficulty the greater $F(t)$ is.

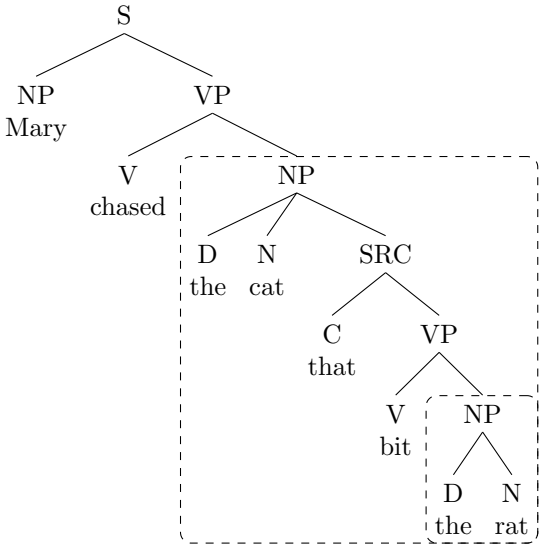
this simplified[17] pattern of facts has an explanation. The form of the explanation is exactly the one illustrated with the node-ratio example from Miller and Chomsky (1963) mentioned in Section 1. Of course since F is a function of the tree structures that a grammar associates with the relevant strings, the predictions we end up making will be sensitive to our choice of grammar.

The grammar in (14) generates all the sentences in (15), (16) and (17), with structures like the following. These are of course extremely simplistic, particularly as regards the treatment of relative clauses (SRC stands for "subject relative clause", ORC for "object relative clause"), but they capture the basic structural configurations well enough for our purposes. Specifically, these trees show structures for the 'b.' sentences in (15), (16) and (17); structures for the a.' sentences would have just one of the highlighted constituents each, and structures for the 'c.' sentences would have three of the highlighted constituents each. Nothing in particular hinges on the fact that the highlighted self-embedded constituents are NPs in all three cases; in (20) we could just as well highlight the relationship between the two VP nodes. The important point is that in (19) there are constituents appearing as the left portion of a larger constituent of the same type, in (20) there are constituents appearing as the right portion, and in (21) there are constituents appearing as medial portions of a larger constituent of the same type.

(19)

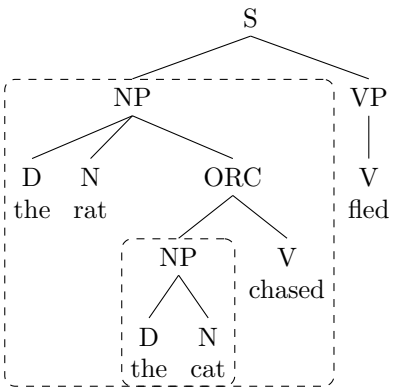


(20)



[17] In particular, it's not at all clear that there's much of a difference between, on the one hand, the (15a)/(15b) and (16a)/(16b) comparisons, and on the other, the (17a)/(17b) comparison. If there's not, then the distinctive status of (17c) might be better explained as some kind of "overflow" effect based on a function that has similar properties to the hypothetical F introduced above. Note also that I am leaving aside all comparisons between sentence of different embedding-types, so for example the relationship between F(tree for (16b)) and F(tree for (17b)) will not play any role in any predictions.

(21)



3.1 Bottom-up parsing

In bottom-up parsing, as in all three of the methods we will discuss here, a configuration has two parts, a *buffer* and a *stack*: the buffer is a record of our left-to-right progress through the sentence, and the stack is a record of what we have worked out on the basis of that part of the sentence that we have consumed so far. The stack takes the form of a sequence of nonterminal symbols. I will write configurations as an ordered pair, with the stack first and the buffer second. I will represent buffers with a '|' symbol separating the input consumed so far from that which remains.

The general idea behind bottom-up parsing is that we read in input from left-to-right, and when we find two elements adjacent to each other that match with the right-hand side of some grammar rule, we replace those two elements with the nonterminal symbol that appears on the left-hand side of that rule. For example, if the relevant grammar contains a rule 'VP → V NP' and we find occurrences of V and NP adjacent to each other (in that order), then they can be replaced on the stack with VP; this occurrence of VP may itself subsequently be replaced by some other symbol on the basis of a rule which has VP somewhere on its right-hand side. The "goal" is to reach a configuration where the stack contains only the start symbol, S.

The bottom-up parsing method is defined in (22). Since these transitions only manipulate the right edge of the sequence of nonterminal symbols that is the first component of each configuration, I will refer to that edge as the "top" of the stack. A SHIFT transition consumes a word (w_i) of input (moving the marker one place to the right in the buffer), and adds that word's category (X) to the top of the stack; a REDUCE transition leaves our position in the input unchanged but operates on the stack, replacing the symbols that appear on the right hand side of some rule ($Y_1 \dots Y_m$) with the symbol that appears on the left hand side of that rule (X).

(22) Given a sentence $w_1 \dots w_n$ to be parsed and a grammar:
- starting configuration: $(\epsilon, \mid w_1 \dots w_n)$
- goal configuration: $(\mathrm{S}, w_1 \dots w_n \mid)$
- SHIFT transitions: $(\mathbf{\Sigma}, w_1 \dots \mid w_i \dots w_n) \Longrightarrow (\mathbf{\Sigma}X, w_1 \dots w_i \mid \dots w_n)$
  if there is a rule $X \rightarrow w_i$ in the grammar
- REDUCE transitions: $(\mathbf{\Sigma}Y_1 \dots Y_m, w_1 \dots \mid \dots w_n) \Longrightarrow (\mathbf{\Sigma}X, w_1 \dots \mid \dots w_n)$
  if there is a rule $X \rightarrow Y_1 \dots Y_m$ in the grammar

(where $\mathbf{\Sigma}$ is a placeholder for sequences of nonterminal symbols)

The workings of these transitions are most easily understood via an example. The actions of a bottom-up parser on the sentence 'the dog chased the cat' are shown in Table 1. We begin in the starting configuration, with the stack empty and with the full sentence remaining to be read, as indicated by the marker at the beginning of the buffer. The first SHIFT step consumes the first word, 'the' (moving the marker one step to the right in the buffer), and puts its category, D, onto the stack. The second step does likewise for the word 'dog', and its category N. We then find ourselves with the sequence D N on the stack, which matches the right hand side of one of the grammar's rules, specifically the rule NP → D N. We can therefore take a REDUCE transition at step 3, replacing the current stack contents with NP and leaving the buffer's progress marker unchanged in its position after 'dog'.

In the next two steps, two more SHIFT transitions consume 'chased' and 'the'. No REDUCE transition is possible after Step 5, when the stack contains NP V D, because neither V D nor NP V D is the right hand side of any grammar rule.[18] A REDUCE transition next becomes possible only after the last word of the input ('dog') is consumed in Step 6, at which point the D N sequence that is on the top of the stack can be replaced with NP (Step 7). This then feeds

	Type of transition	Rule used	Configuration
0	—	—	(ϵ, | the dog chased the cat)
1	SHIFT	D → the	(D, the | dog chased the cat)
2	SHIFT	N → dog	(D N, the dog | chased the cat)
3	REDUCE	NP → D N	(NP, the dog | chased the cat)
4	SHIFT	V → chased	(NP V, the dog chased | the cat)
5	SHIFT	D → the	(NP V D, the dog chased the | cat)
6	SHIFT	N → cat	(NP V D N, the dog chased the cat |)
7	REDUCE	NP → D N	(NP V NP, the dog chased the cat |)
8	REDUCE	VP → V NP	(NP VP, the dog chased the cat |)
9	REDUCE	S → NP VP	(S, the dog chased the cat |)

Table 1: *A first illustration of **bottom-up** parsing.*

the final two REDUCE steps in Step 8 and Step 9, after which we have reached a configuration where all the input has been consumed (the progress marker is at the far right of the buffer) and the stack contains just the single symbol S, i.e. we have reached a goal configuration.

Ambiguity resolution

The only transitions shown in Table 1 are the "correct" transitions that form a path from the starting configuration to the goal configuration. But of course there are other transitions that we could have taken instead at certain points. For example, instead of applying REDUCE in Step 3 we could have applied SHIFT, which would have led to the configuration (D N V, the dog chased | the cat); and from that point, we could have carried out the same sequence of steps as appear in Table 1 to process the VP constituent, with D N at the bottom of the stack throughout instead of NP. This path hits a dead end, however, after the REDUCE step that forms the VP constituent, because we end up with D N VP in the stack (instead of NP VP) and there is now no way to "get at" the D and N that should be combined to form an NP: in order to be acted on by REDUCE a sequence of nonterminal symbols must be at the top of the stack, which means that the D and the N "missed their chance" after Step 2. It turns out that there is exactly one sequence of "correct" transitions for each derivation licensed by the grammar: if a word-sequence has two grammatical derivations (i.e. is structurally ambiguous) then there will be two distinct transition-sequences for that word-sequence that both end at the goal configuration.[19]

In a simple example such as Table 1, it is relatively straightforward to identify which transitions will turn out to be "wrong turns". But the general idea of deciding which transition to take from a particular configuration corresponds to decisions about (local or global) ambiguity. In this case choosing not to apply REDUCE after Step 2 perhaps seems "obviously wrong", but in general these kinds of choices correspond to decisions about ambiguity, either local or global. In the case of a globally ambiguous sentence, it will be possible to identify the particular point at which the paths to the two possible structural descriptions diverge. To illustrate a case of local ambiguity, consider the sentence in (23). In keeping with the "late closure" heuristic (Frazier and Rayner, 1982; Frazier and Clifton, 1996), comprehenders often analyze 'the dog that barked' as the object of the embedded verb 'chased'; this leads to a mild garden-path effect when it becomes apparent, at 'fled', that this phrase is in fact the matrix subject.

(23) While John chased the dog that barked fled.

When the bottom-up parsing system is applied to this sentence, a choice point arises between shift and reduce after 'chased', corresponding to the decision about whether to analyze this verb as transitive or intransitive or not. The two tables in (24) show the point where these two paths diverge, namely after the first two lines which are the same in each. The late closure heuristic says exactly that shift transitions, such as the one taken in the third line of the table in (24a), should be preferred over reduce transitions whenever a choice between them arises.

[18] For now I am glossing over the fact that a REDUCE transition, using the VP → V rule, is possible after Step 4.

[19] In particular, the sequence of configurations visited by a bottom-up parser corresponds exactly to a reverse rightmost context-free derivation: moving upwards in Table 1 corresponds to rewriting the rightmost nonterminal symbol at each step.

(24) a.

...	...	(WHILE NP, while John | chased the dog that barked fled)
SHIFT	V → chased	(WHILE NP V, while John chased | the dog that barked fled)
SHIFT	D → the	(WHILE NP V D, while John chased the | dog that barked fled)
SHIFT	N → dog	(WHILE NP V D N, while John chased the dog | that barked fled)
...	...	...
...	...	(WHILE NP V S, while John chased the dog that barked fled |)

b.

...	...	(WHILE NP, while John | chased the dog that barked fled)
SHIFT	V → chased	(WHILE NP V, while John chased | the dog that barked fled)
REDUCE	VP → V	(WHILE NP VP, while John chased | the dog that barked fled)
REDUCE	S → NP VP	(WHILE S, while John chased | the dog that barked fled)
SHIFT	D → the	(WHILE S D, while John chased | the dog that barked fled)
...	...	...
...	...	(WHILE S S, while John chased the dog that barked fled |)
REDUCE	S → WHILE S S	(S, while John chased the dog that barked fled |)

A simple view of the reanalysis that is required in this example would be to imagine a device that continues down the path taken in (24a) until it reaches a dead end after consuming 'fled' (there is nowhere to go from the transition which has 'WHILE NP V S' on the stack), and then returns to the divergence point to pursue the alternative path taken in (24b).

Embedding structures

Let us turn to the pattern noted in (15), (16) and (17) above. The workings of the bottom-up parsing system on the two-clause and three-clause center-embedded sentences, (17b) and (17c), are shown in Table 2. Notice that in order to analyze these sorts of structures, we must shift all the initial determiners and nouns before any REDUCE transitions can be taken, since the first chance we get to combine things into a completed constituent is only after the last determiner-noun pair has been consumed ('the cat' in (17b), 'the dog' in (17c)). A natural idea to consider is that what makes (17c) noticeably more difficult than (17b) is that processing (17c) requires an ability to maintain six symbols at a time in a stack-based memory system (see the configuration after shifting 'dog'), whereas processing (17b) only requires being able to maintain four (see the configuration after shifting 'cat').[20]

Recall however that it is only center-embedding structures that cause this drastic increase in comprehension difficulty. Next consider the workings of a bottom-up parsing system on the two-clause and three-clause left-embedded sentences, (15b) and (15c), shown in Table 3. In these scenarios it is possible to perform a REDUCE transition after consuming a beginning portion matching the pattern NP POSS N, as shown in the fourth step on both sides of Table 3. Note, however, that this is possible no matter how deeply the NP constituent thus formed needs to eventually be embedded, i.e. whether the NP constituent thus formed eventually turns out to be the matrix subject, as it is in (15b), or a subconstituent of the matrix subject, as it is in (15c). Either way we arrive at a configuration where one NP symbol on the stack suffices to track our progress through the structure. And, importantly, in the more deeply-embedded case of (15c), the resulting NP can serve as the starting point for another iteration of the "loop" indicated by the large braces in Table 3. These left-branching sentences differ *only* in how many times we go around that loop: processing the larger sentence in (15c) just involves revisiting some of the stack arrangements that also appear in the course of processing (15b), and the same would be true if we extended the pattern to create a yet longer sentence.[21] This means that the number of symbols a device must be able to maintain simultaneously will not grow as the size of the left-branching structure to be processed grows. Specifically, we can observe that the maximum number of symbols stored at a time is three on both sides of Table 3. This gives us the basis of an explanation for why large center-embedding structures become difficult to comprehend but large left-branching structures do not.

Concretely, what we can propose amounts to a new function $F(t)$ defined on structural descriptions of the sort mentioned in Section 1. Given a context-free parse tree, there is a uniquely determined sequence of configurations

[20] The point would not be significantly affected if we adopted a structure where 'the rat' was a subconstituent of 'the rat the cat chased'. If the structure were [NP [NBAR the rat] [ORC the cat chased]] then reduce steps forming the NBAR subconstituents could be performed early, but we would still find ourselves accumulating a number of NBARs on the stack that increases with the depth of embedding.

[21] i.e. 'John's brother's wife's dog fled'.

that a bottom-up parser must move through to process it, and therefore also a uniquely determined "maximum stack size". Let us write $MaxStack_{BU}(t)$ for this maximum stack size (BU for "bottom-up"; we will consider stack sizes for other parsing systems below). Then for the examples considered so far we have the following pattern:

$$MaxStack_{BU}(\text{tree for (15b)}) = MaxStack_{BU}(\text{tree for (15c)}) \qquad \text{(left-branching)}$$
$$MaxStack_{BU}(\text{tree for (17b)}) < MaxStack_{BU}(\text{tree for (17c)}) \qquad \text{(center-embedding)}$$

which in combination with the linking hypothesis that

(25) A speaker processing the string $PHON(t)$ will experience greater perceptual difficulty the greater $MaxStack_{BU}(t)$ is.

correctly predicts the relevant facts for these four sentences.

For the other pair however, i.e. (16b) and (16c), this theory makes the *wrong* predictions. The workings of the bottom-up parser on these right-branching sentences are shown in Table 4. The completely right-branching structure means that the bottom-up parsing must shift the entire sentence before performing any reductions at all; the first complete constituent that it can construct is the NP consisting of the last two words of the sentence. So for a bottom-up parser, right-branching structures share with center-embedding structures the property that the memory load increases with the depth of embedding.

$$MaxStack_{BU}(\text{tree for (16b)}) < MaxStack_{BU}(\text{tree for (16c)}) \qquad \text{(right-branching)}$$

The linking hypothesis in (25) therefore incorrectly predicts that (16c) should show greater comprehension difficulty than (16b).

This incorrect prediction of course also arises from having chosen the grammar in (14). The linking hypothesis serves to expose this grammar to evidence concerning comprehension difficulty. So of course one possible response to finding this incorrect prediction is to maintain (25) and reject the grammar in (14), and seek some other grammar that assigns different tree structures for (16b) and (16c) (e.g. perhaps left-branching structures) so that the correct predictions arise. Let us suppose though that we are more confident in our hypothesis that (14) is the correct mental grammar for the relevant speakers, than we are in the linking hypothesis in (25); then a reasonable next step is to seek some alternative to bottom-up parsing, rejecting (25) but leaving our hypothesized grammar in (14) in place.

3.2 Top-down parsing

As an alternative to bottom-up parsing, we can consider top-down parsing. As the names suggest, these two parsing methods can be thought of as inverses of each other: whereas bottom-up parsing works from the words up to the start symbol, top-down parsing works from the start symbol down to the words.

The top-down parsing method is defined in (26). Since the top-down parsing transitions only manipulate the left edge of the sequence of nonterminal symbols, I will refer to that edge as the "top" of the stack. The starting configuration has the start symbol on the stack, and the goal configuration has an empty stack. A PREDICT transition eliminates a symbol (X) currently at the top of the stack, by guessing a particular grammar rule that can be used to expand it and replacing it with that rule's right-hand side ($Y_1 \dots Y_m$). A MATCH transition eliminates a symbol (X) currently at the top of the stack by consuming a word (w_i) of the corresponding category.

(26) Given a sentence $w_1 \dots w_n$ to be parsed and a grammar:
- starting configuration: $(\text{S}, \mid w_1 \dots w_n)$
- goal configuration: $(\epsilon, w_1 \dots w_n \mid)$
- PREDICT transitions: $(X\boldsymbol{\Sigma}, w_1 \dots \mid \dots w_n) \Longrightarrow (Y_1 \dots Y_m\boldsymbol{\Sigma}, w_1 \dots \mid \dots w_n)$ if there is a rule $X \rightarrow Y_1 \dots Y_m$ in the grammar
- MATCH transitions: $(X\boldsymbol{\Sigma}, w_1 \dots \mid w_i \dots w_n) \Longrightarrow (\boldsymbol{\Sigma}, w_1 \dots w_i \mid \dots w_n)$ if there is a rule $X \rightarrow w_i$ in the grammar

(where $\boldsymbol{\Sigma}$ is a placeholder for sequences of nonterminal symbols)

Again, the mechanics of these transition rules are most easily understood via an example. The progress of a top-down parsing on the sentence 'the dog chased the cat' are shown in Table 5. We begin with the start symbol S on the

(17b): $_0$ the $_1$ rat $_2$ the $_3$ cat $_4$ chased $_5$ fled $_6$

Transition	Rule used	Configuration
—	—	(ϵ, 0)
SHIFT	D → the	(D, 1)
SHIFT	N → rat	(D N, 2)
SHIFT	D → the	(D N D, 3)
SHIFT	N → cat	(D N D N, 4)
REDUCE	NP → D N	(D N NP, 4)
SHIFT	V → chased	(D N NP V, 5)
REDUCE	ORC → NP V	(D N ORC, 5)
REDUCE	NP → D N	(NP, 5)
SHIFT	V → fled	(NP V, 6)
REDUCE	VP → V	(NP VP, 6)
REDUCE	S → NP VP	(S, 6)

(17c): $_0$ the $_1$ rat $_2$ the $_3$ cat $_4$ the $_5$ dog $_6$ bit $_7$ chased $_8$ fled $_9$

Transition	Rule used	Configuration
—	—	(ϵ, 0)
SHIFT	D → the	(D, 1)
SHIFT	N → rat	(D N, 2)
SHIFT	D → the	(D N D, 3)
SHIFT	N → cat	(D N D N, 4)
SHIFT	D → the	(D N D N D, 5)
SHIFT	N → dog	(D N D N D N, 6)
REDUCE	NP → D N	(D N D N NP, 6)
SHIFT	V → bit	(D N D N NP V, 7)
REDUCE	ORC → NP V	(D N D N ORC, 7)
REDUCE	NP → D N ORC	(D N NP, 7)
SHIFT	V → chased	(D N NP V, 8)
REDUCE	ORC → NP V	(D N ORC, 8)
REDUCE	NP → D N ORC	(NP, 8)
SHIFT	V → fled	(NP V, 9)
REDUCE	VP → V	(NP VP, 9)
REDUCE	S → NP VP	(S, 9)

Table 2: *The effect of* ***center-embedding*** *on* ***bottom-up*** *parsing. Memory load increases as embedding depth increases: maximum of 4 symbols for* (17b) *but 6 symbols for* (17c).

(15b): $_0$ John $_1$'s $_2$ dog $_3$ fled $_4$

Transition	Rule used	Configuration
—	—	(ϵ, 0)
SHIFT	NP → John	(NP, 1)
SHIFT	POSS → 's	(NP POSS, 2)
SHIFT	N → dog	(NP POSS N, 3)
REDUCE	NP → NP POSS N	(NP, 3)
SHIFT	V → fled	(NP V, 4)
REDUCE	VP → V	(NP VP, 4)
REDUCE	S → NP VP	(S, 4)

(15c): $_0$ John $_1$'s $_2$ wife $_3$'s $_4$ dog $_5$ fled $_6$

Transition	Rule used	Configuration
—	—	(ϵ, 0)
SHIFT	NP → John	(NP, 1)
SHIFT	POSS → 's	(NP POSS, 2)
SHIFT	N → wife	(NP POSS N, 3)
REDUCE	NP → NP POSS N	(NP, 3)
SHIFT	POSS → 's	(NP POSS, 4)
SHIFT	N → dog	(NP POSS N, 5)
REDUCE	NP → NP POSS N	(NP, 5)
SHIFT	V → fled	(NP V, 6)
REDUCE	VP → V	(NP VP, 6)
REDUCE	S → NP VP	(S, 6)

Table 3: *The effect of* ***left-embedding*** *on* ***bottom-up*** *parsing. No increase in memory load as embedding depth increases: maximum of 3 symbols in both cases.*

(16b): $_0$ Mary $_1$ chased $_2$ the $_3$ cat $_4$ that $_5$ bit $_6$ the $_7$ rat $_8$

Transition	Rule used	Configuration
—	—	(ϵ, 0)
SHIFT	NP → Mary	(NP, 1)
SHIFT	V → chased	(NP V, 2)
SHIFT	D → the	(NP V D, 3)
SHIFT	N → cat	(NP V D N, 4)
SHIFT	C → that	(NP V D N C, 5)
SHIFT	V → bit	(NP V D N C V, 6)
SHIFT	D → the	(NP V D N C V D, 7)
SHIFT	N → rat	(NP V D N C V D N, 8)
REDUCE	NP → D N	(NP V D N C V NP, 8)
REDUCE	VP → V NP	(NP V D N C VP, 8)
REDUCE	SRC → C VP	(NP V D N SRC, 8)
REDUCE	NP → D N SRC	(NP V NP, 8)
REDUCE	VP → V	(NP VP, 8)
REDUCE	S → NP VP	(S, 8)

(16c): $_0$ Mary $_1$ chased $_2$ the $_3$ cat $_4$ that $_5$ bit $_6$ the $_7$ rat $_8$ that $_9$ ate $_{10}$ the $_{11}$ cheese $_{12}$

Transition	Rule used	Configuration
—	—	(ϵ, 0)
SHIFT	NP → Mary	(NP, 1)
SHIFT	V → chased	(NP V, 2)
SHIFT	D → the	(NP V D, 3)
SHIFT	N → cat	(NP V D N, 4)
SHIFT	C → that	(NP V D N C, 5)
SHIFT	V → bit	(NP V D N C V, 6)
SHIFT	D → the	(NP V D N C V D, 7)
SHIFT	N → rat	(NP V D N C V D N, 8)
SHIFT	C → that	(NP V D N C V D N C, 9)
SHIFT	V → are	(NP V D N C V D N C V, 10)
SHIFT	D → the	(NP V D N C V D N C V D, 11)
SHIFT	N → cheese	(NP V D N C V D N C V D N, 12)
REDUCE	NP → D N	(NP V D N C V D N C V NP, 12)
REDUCE	VP → V NP	(NP V D N C V D N C VP, 12)
REDUCE	SRC → C VP	(NP V D N C V D N SRC, 12)
REDUCE	NP → D N SRC	(NP V D N C V NP, 12)
REDUCE	VP → V NP	(NP V D N C VP, 12)
REDUCE	SRC → C VP	(NP V D N SRC, 12)
REDUCE	NP → D N SRC	(NP V NP, 12)
REDUCE	VP → V	(NP VP, 12)
REDUCE	S → NP VP	(S, 12)

Table 4: *The effect of* ***right-embedding*** *on* ***bottom-up*** *parsing. Memory load increases as embedding depth increases: maximum of 8 symbols for* (16b) *but 12 symbols for* (16c).

stack, and the full sentence remaining to be read. The first two steps expand this occurrence of S according to the rule 'S → NP VP', and then expand the introduced occurence of NP according to the rule 'NP → D N'. The upshot of these first two steps is that we have replaced the task of fulfilling a prediction that 'the dog chased the cat' is a sentence (S) with the task of fulfilling a prediction that 'the dog chased the cat' is of the form 'D N VP'. In the third step we make some progress towards fulfilling this prediction: since the first word to be consumed is 'the' and this matches the category D that is currently on the top of the stack, a MATCH transition "cancels them out". The fourth step similarly consumes the word 'dog' and eliminates the corresponding symbol N. After this step we are left with the task of fulfilling the prediction that 'chased the cat' is a VP.

	Type of transition	Rule used	Configuration
0	—	—	(S, | the dog chased the cat)
1	PREDICT	S → NP VP	(NP VP, | the dog chased the cat)
2	PREDICT	NP → D N	(D N VP, | the dog chased the cat)
3	MATCH	D → the	(N VP, the | dog chased the cat)
4	MATCH	N → dog	(VP, the dog | chased the cat)
5	PREDICT	VP → V NP	(V NP, the dog | chased the cat)
6	MATCH	V → chased	(NP, the dog chased | the cat)
7	PREDICT	NP → D N	(D N, the dog chased | the cat)
8	MATCH	D → the	(N, the dog chased the | cat)
9	MATCH	N → cat	(ϵ, the dog chased the cat |)

Table 5: *A first illustration of* ***top-down*** *parsing.*

The fifth step replaces this VP prediction with the prediction of a V NP sequence. (Of course a "wrong turn" that could also be taken at this point would be to replace it with a prediction of simply a V instead; the particular wrong turns that present themselves differ according to the parsing method adopted.) Next the V is matched (Step 6) and then the NP is expanded (Step 7). Note that expanding the NP before the V is matched is not an option, because the PREDICT transition only manipulates the top of the stack. After MATCH transitions consume the last two words in Step 8 and Step 9, we have reached the end of the sentence with no predictions remaining to be fulfilled (the stack is empty), so the goal configuration has been reached.

A key point in understanding the relationship between bottom-up and top-down parsing — and also the left-corner parsing method, discussed in the next subsection, which can be seen as combining the advantages of both — is the fact that symbols on the stack on bottom-up parsing represent the already-processed part of the sentence, whereas symbols on the stack on top-down parsing represent (predictions about) the yet-to-come part of the sentence. In each bottom-up parsing configuration shown in Table 1, the sequence of nonterminal symbols corresponds to the portion of the sentence to the left of the '|' symbol: for example, 'D N' after Step 2 is a description of 'the dog', and 'NP V' after Step 4 is a description of 'the dog chased'. In the top-down parsing configurations shown in Table 5, however, the sequence of nonterminal symbols corresponds to the words to the right of the '|' symbol: for example, 'N VP' after Step 3 represents a prediction that can be fulfilled by 'dog chased the cat', and 'D N' after Step 7 represents a prediction that can be fulfilled by 'the cat'. A consequence of this difference is that the two methods differ in which points in the sentence allow for a *compact* representation of the parser's current internal state. The bottom-up parser's internal state after consuming the words 'the dog' can be represented compactly in one symbol, namely NP, as shown after Step 3 in Table 1. The top-down parser's state at this point in the sentence can also be represented compactly in one symbol, namely VP, as shown after Step 4 in Table 5. But notice that only with the top-down method can a parser's state after the word 'chased' be represented in a single symbol: since what is predicted at this point is a single NP constituent, this single symbol suffices for the top-down parser, as shown after Step 6 in Table 5, but the bottom-up parser has no one-symbol representation of its progress at this point since 'the dog chased' is not a constituent.

The effects of left-branching, right-branching and center-embedding structures on the stack requirements with top-down parsing can be clearly understood through these observations about the interpretations of stack symbols. Like bottom-up, the top-down method correctly predicts increasing maximum stack size as center-embedding depth increases, as shown in Table 6. But the top-down method shows the reverse pattern from the bottom-up method with regard to left-branching and right-branching structures, i.e. the maximum stack size increases for left-branching

but not for right-branching, as shown in Table 7 and Table 8.

$$MaxStack_{TD}(\text{tree for (15b)}) < MaxStack_{TD}(\text{tree for (15c)}) \quad \text{(left-branching)}$$
$$MaxStack_{TD}(\text{tree for (16b)}) = MaxStack_{TD}(\text{tree for (16c)}) \quad \text{(right-branching)}$$
$$MaxStack_{TD}(\text{tree for (17b)}) < MaxStack_{TD}(\text{tree for (17c)}) \quad \text{(center-embedding)}$$

The absence of any increase in maximum stack size in Table 8 has the same kind of explanation as we saw in Table 3: the difference between (16b) and (16c) is simply the difference between going around a "loop" once or twice, where the loop begins and ends at configurations where the prediction about the remaining portion of the sentence can be compactly represented as a single NP symbol (i.e. after 'chased' and 'bit' and 'ate'). So the efficient processing of right-branching structures by the top-down parser stems from the fact that these are structures where the remaining portions of a sentence have compact descriptions in terms of predicted constituents — just as the efficient processing of left-branching structures when working bottom-up stems from the fact that such structures allow compact representations in terms of already-processing constituents. When the top-down parser is confronted with left-branching structures, the required stack size increases with the depth of embedding (Table 7), because the remaining predicted structure takes the form of a large number of distinct constituents (and is at its largest at the beginning of the sentence); right-branching structures pose analogous difficulties for bottom-up parsing, since the already-consumed portion of a sentence can only be described with a large number of symbols (and is at its largest at the end of the sentence). Center-embedding structures impose requirements on the stack that increase with depth of embedding because near the middle of such sentences there is neither a compact representation in terms of already-processed constituents (see Table 2 nor one in terms of predicted constituents (see Table 6).

These facts about the workings of the top-down parser therefore mean that under the new linking hypothesis in (27), we make correct predictions regarding right-branching structures and center-embedding structures, but an incorrect prediction regarding left-branching structures.

(27) A speaker processing the string $PHON(t)$ will experience greater perceptual difficulty the greater $MaxStack_{TD}(t)$ is.

Again there is of course the option of attributing the blame to the grammar, rather than this linking hypothesis: given independent strong evidence that (27) were correct, we might prefer to modify our hypothesized grammar (e.g. perhaps attributing *right*-branching structures to the sentences in (15)). But in the next subsection we will see that there is a different linking hypothesis we can combine with this existing grammar that correctly accounts for all of the embedding facts we have discussed.

3.3 Left-corner parsing

As foreshadowed above, the left-corner parsing method can be seen as a "best of both worlds" mixture of bottom-up and top-down processing, incorporating the advantages of both: the efficient handling of left-branching structures via compact memory representations of already-consumed constituents, and the efficient handling of right-branching structures via compact memory representations of predicted constituents. Configurations still take the form of a stack paired with a buffer, but the stack can store two different kinds of symbols: "barred" versions of nonterminal symbols (e.g. $\overline{\text{NP}}$, $\overline{\text{VP}}$) which, like all symbols stored by a top-down parser, represent constituents predicted in the remaining portion of a sentence, and "plain"/"unbarred" nonterminal symbols (e.g. NP, VP) which, like all symbols stored by a bottom-up parser, represent constituents formed from the already-consumed portion.

The left-corner parsing method is defined in (28). The starting configuration has $\bar{\text{S}}$ on the stack, analogous to the starting configuration for top-down parsing; the goal is to fulfill this prediction. The top of the stack is on the left. The SHIFT and MATCH transitions are essentially identical to the transitions of the same names in the previous systems: note that SHIFT consumes a word and adds corresponding a bottom-up (i.e. "unbarred") nonterminal symbol to the stack, and MATCH consumes a word and removes a corresponding top-down ("barred") nonterminal symbol from the stack. These two simple types of transitions deal with rules that have only terminal symbols on the right-hand side (i.e. rules of the form '$X \rightarrow w$'). The other two types of transitions, LC-PREDICT and LC-CONNECT, deal with rules that have nonterminal symbols on the right-hand side (i.e. rules of the form '$X \rightarrow Y_1 \ldots Y_m$'); these transitions are more complex because they deal with the interplay between constituents recognized bottom-up (plain symbols) and constituents predicted top-down (barred symbols).

(17b): $_0$ the $_1$ rat $_2$ the $_3$ cat $_4$ chased $_5$ fled $_6$

Transition	Rule used	Configuration
—	—	(S, 0)
PREDICT	S → NP VP	(NP VP, 0)
PREDICT	NP → D N ORC	(D N ORC VP, 0)
MATCH	D → the	(N ORC VP, 1)
MATCH	N → rat	(ORC VP, 2)
PREDICT	ORC → NP V	(NP V VP, 2)
PREDICT	NP → D N	(D N V VP, 2)
MATCH	D → the	(N V VP, 3)
MATCH	N → cat	(V VP, 4)
MATCH	V → chased	(VP, 5)
PREDICT	VP → V	(V, 5)
MATCH	V → fled	(ϵ, 6)

(17c): $_0$ the $_1$ rat $_2$ the $_3$ cat $_4$ the $_5$ dog $_6$ bit $_7$ chased $_8$ fled $_9$

Transition	Rule used	Configuration
—	—	(S, 0)
PREDICT	S → NP VP	(NP VP, 0)
PREDICT	NP → D N ORC	(D N ORC VP, 0)
MATCH	D → the	(N ORC VP, 1)
MATCH	N → rat	(ORC VP, 2)
PREDICT	ORC → NP V	(NP V VP, 2)
PREDICT	NP → D N ORC	(D N ORC V VP, 2)
MATCH	D → the	(N ORC V VP, 3)
MATCH	N → cat	(ORC V VP, 4)
PREDICT	ORC → NP V	(NP V V VP, 4)
PREDICT	NP → D N	(D N V V VP, 4)
MATCH	D → the	(N V V VP, 5)
MATCH	N → dog	(V V VP, 6)
MATCH	V → bit	(V VP, 7)
MATCH	V → chased	(VP, 8)
PREDICT	VP → V	(V, 8)
MATCH	V → fled	(ϵ, 9)

Table 6: *The effect of **center-embedding** on **top-down** parsing. Memory load increases as embedding depth increases: maximum of 4 symbols for* (17b) *but 5 symbols for* (17c).

(15b): $_0$ John $_1$'s $_2$ dog $_3$ fled $_4$

Transition	Rule used	Configuration
—	—	(S, 0)
PREDICT	S → NP VP	(NP VP, 0)
PREDICT	NP → NP POSS N	(NP POSS N VP, 0)
MATCH	NP → John	(POSS N VP, 1)
MATCH	POSS → 's	(N VP, 2)
MATCH	N → dog	(VP, 3)
PREDICT	VP → V	(V, 3)
MATCH	V → fled	(ϵ, 4)

(15c): $_0$ John $_1$'s $_2$ wife $_3$'s $_4$ dog $_5$ fled $_6$

Transition	Rule used	Configuration
—	—	(S, 0)
PREDICT	S → NP VP	(NP VP, 0)
PREDICT	NP → NP POSS N	(NP POSS N VP, 0)
PREDICT	NP → NP POSS N	(NP POSS N POSS N VP, 0)
MATCH	NP → John	(POSS N POSS N VP, 1)
MATCH	POSS → 's	(N POSS N VP, 2)
MATCH	N → wife	(POSS N VP, 3)
MATCH	POSS → 's	(N VP, 4)
MATCH	N → dog	(VP, 5)
PREDICT	VP → V	(V, 5)
MATCH	V → fled	(ϵ, 6)

Table 7: *The effect of **left-embedding** on **top-down** parsing. Memory load increases as embedding depth increases: maximum of 4 symbols for* (15b) *but 6 symbols for* (15c).

(16b): $_0$ Mary $_1$ chased $_2$ the $_3$ cat $_4$ that $_5$ bit $_6$ the $_7$ rat $_8$

Transition	Rule used	Configuration
—	—	(S, 0)
PREDICT	S → NP VP	(NP VP, 0)
MATCH	NP → Mary	(VP, 1)
PREDICT	VP → V NP	(V NP, 1)
MATCH	V → chased	(NP, 2)
PREDICT	NP → D N SRC	(D N SRC, 2)
MATCH	D → the	(N SRC, 3)
MATCH	N → cat	(SRC, 4)
PREDICT	SRC → C VP	(C VP, 4)
MATCH	C → that	(VP, 5)
PREDICT	VP → V NP	(V NP, 5)
MATCH	V → bit	(NP, 6)
PREDICT	NP → D N	(D N, 6)
MATCH	D → the	(N, 7)
MATCH	N → rat	(ϵ, 8)

(16c): $_0$ Mary $_1$ chased $_2$ the $_3$ cat $_4$ that $_5$ bit $_6$ the $_7$ rat $_8$ that $_9$ ate $_{10}$ the $_{11}$ cheese $_{12}$

Transition	Rule used	Configuration
—	—	(S, 0)
PREDICT	S → NP VP	(NP VP, 0)
MATCH	NP → Mary	(VP, 1)
PREDICT	VP → V NP	(V NP, 1)
MATCH	V → chased	(NP, 2)
PREDICT	NP → D N SRC	(D N SRC, 2)
MATCH	D → the	(N SRC, 3)
MATCH	N → cat	(SRC, 4)
PREDICT	SRC → C VP	(C VP, 4)
MATCH	C → that	(VP, 5)
PREDICT	VP → V NP	(V NP, 5)
MATCH	V → bit	(NP, 6)
PREDICT	NP → D N SRC	(D N SRC, 6)
MATCH	D → the	(N SRC, 7)
MATCH	N → rat	(SRC, 8)
PREDICT	SRC → C VP	(C VP, 8)
MATCH	C → that	(VP, 9)
PREDICT	VP → V NP	(V NP, 9)
MATCH	V → ate	(NP, 10)
PREDICT	NP → D N	(D N, 10)
MATCH	D → the	(N, 11)
MATCH	N → cheese	(ϵ, 12)

Table 8: *The effect of **right-embedding** on **top-down** parsing. No increase in memory load as embedding depth increases: maximum of 3 symbols in both cases.*

(28) Given a sentence $w_1 \dots w_n$ to be parsed and a grammar:
- starting configuration: $(\bar{\text{S}}, \mid w_1 \dots w_n)$
- goal configuration: $(\epsilon, w_1 \dots w_n \mid)$
- SHIFT transitions: $(\mathbf{\Sigma}, w_1 \dots \mid w_i \dots w_n) \Longrightarrow (X\mathbf{\Sigma}, w_1 \dots w_i \mid \dots w_n)$ if there is a rule $X \rightarrow w_i$ in the grammar
- MATCH transitions: $(\overline{X}\mathbf{\Sigma}, w_1 \dots \mid w_i \dots w_n) \Longrightarrow (\mathbf{\Sigma}, w_1 \dots w_i \mid \dots w_n)$ if there is a rule $X \rightarrow w_i$ in the grammar
- LC-PREDICT transitions: $(Y_1\mathbf{\Sigma}, w_1 \dots \mid \dots w_n) \Longrightarrow (\overline{Y_2} \dots \overline{Y_m} X\mathbf{\Sigma}, w_1 \dots \mid \dots w_n)$ if there is a rule $X \rightarrow Y_1 \dots Y_m$ in the grammar
- LC-CONNECT transitions: $(Y_1\overline{X}\mathbf{\Sigma}, w_1 \dots \mid \dots w_n) \Longrightarrow (\overline{Y_2} \dots \overline{Y_m}\mathbf{\Sigma}, w_1 \dots \mid \dots w_n)$ if there is a rule $X \rightarrow Y_1 \dots Y_m$ in the grammar

(where $\mathbf{\Sigma}$ is a placeholder for sequences of nonterminal symbols)

An example to illustrate is shown in Table 9.

	Type of step	Rule used	Configuration
0	—	—	($\bar{\text{S}}$, | the dog chased the cat)
1	SHIFT	D → the	(D $\bar{\text{S}}$, the | dog chased the cat)
2	LC-PREDICT	NP → D N	($\bar{\text{N}}$ NP $\bar{\text{S}}$, the | dog chased the cat)
3	MATCH	N → dog	(NP $\bar{\text{S}}$, the dog | chased the cat)
4	LC-CONNECT	S → NP VP	($\overline{\text{VP}}$, the dog | chased the cat)
5	SHIFT	V → chased	(V $\overline{\text{VP}}$, the dog chased | the cat)
6	LC-CONNECT	VP → V NP	($\overline{\text{NP}}$, the dog chased | the cat)
7	SHIFT	D → the	(D $\overline{\text{NP}}$, the dog chased the | cat)
8	LC-CONNECT	NP → D N	($\bar{\text{N}}$, the dog chased the | cat)
9	MATCH	N → cat	(ϵ, the dog chased the cat |)

Table 9

The SHIFT transitions do bottom-up work, and only manipulate "plain" symbols (e.g. D at Step 1, V at Step 5); the MATCH transitions do top-down work, and only manipulate "barred" symbols (e.g. $\bar{\text{N}}$ at Step 3). The LC-PREDICT transitions and LC-CONNECT transitions both "trade in" a bottom-up recognized nonterminal (no bar) for some number of predictions (with bars) corresponding to that nonterminal's hypothesized sisters, according to some chosen grammar rule; the *left-corner* of a context-free rule is the first symbol on the right hand side, and it is the chosen grammar rule's left-corner that is "traded in" by these transitions. When LC-PREDICT applies in Step 2, for example, the already-recognized D is hypothesized to be the left-corner of an NP constituent expanded according to the rule 'NP → D N'; it is therefore traded in for a predicted N, accompanied by a bottom-up NP symbol that will be available to work with if that prediction of an N is fulfilled. When LC-CONNECT applies in Step 4, the recognized NP (left-corner of the rule 'S → NP VP') is similarly traded in for a predicted VP; what makes LC-CONNECT different is that we put this recognized NP towards the satisfaction of an *already predicted* instance of the parent nonterminal (here, $\bar{\text{S}}$), so we remove this symbol from the stack rather than adding a bottom-up instance of this parent nonterminal.

A graphical illustration of the relationship between LC-PREDICT and LC-CONNECT is shown in Figure 3. The latter gets its name from the fact that it *connects* the part of the tree that's growing upwards from the words with the part that's growing downwards.

The left-corner parsing system thus has the ability to mix together bottom-up representations of consituents in the already-seen part of the sentence and top-down representations of constituents predicted in the unseen part of the sentence. Note that in Table 9, there are compact one-symbol representations of the parser's state after consuming 'the dog' (the single $\overline{\text{VP}}$ symbol after Step 4) and after consuming 'the dog chased' (the single $\overline{\text{NP}}$ symbol after Step 6), as was the case for the top-down parser; but *in addition*, it also has a compact representation of the state after consuming 'the dog' in terms of the recognized NP after Step 3, as was the case for the bottom-up parser. (The fact that there is an additional $\bar{\text{S}}$ symbol on the stack at that point is an unfortunate distraction: this is a simply "constant-sized" indicator that we have not connected with the overall predicted S node yet, and does not grow with the depth of left-embedding as; see Table 11.[22])

[22] One can flip things around so that there is one extra symbol on the stack *after* the root S node has been created rather than before,

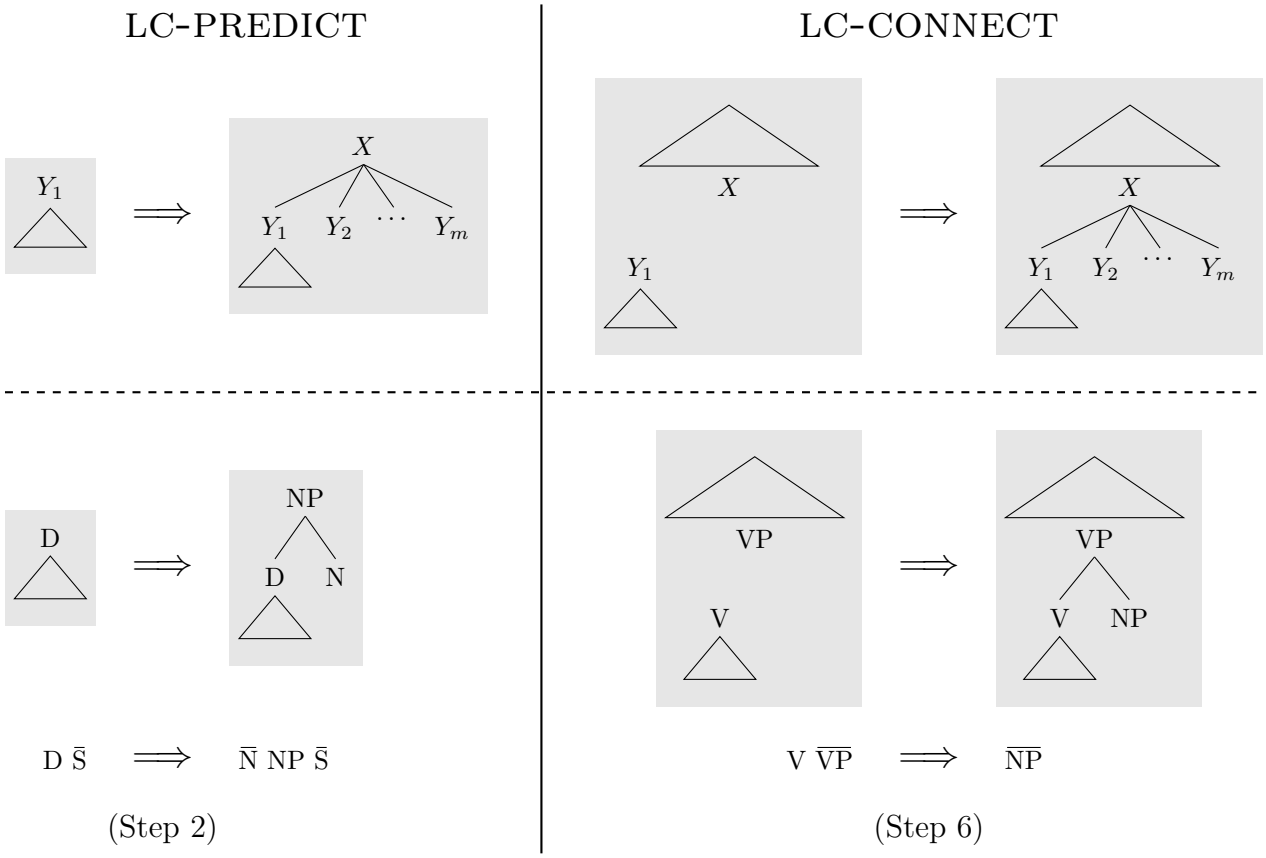


Figure 3: *Graphical illustration of* LC-PREDICT *and* LC-CONNECT. *The general form of each, corresponding to the appropriate parts of the definition in* (28)*, is shown at the top. The bottom shows two instantiations of this general form that appear in Table 9.*

This ability to use a single symbol both in situations where the seen portion of the sentence forms a constituent *and* where the unseen portion of the sentence forms a constituent, allows the left-corner parser to mimic the bottom-up parser's efficient treatment of left-branching structures *and* mimic the top-down parser's efficient treatment of right-branching structures. Table 11 shows how the stack contents are the same after 'John' and after 'John 's dog' (on the left) and after 'John' and 'John 's wife' and 'John 's wife 's dog' (on the right); a single bottom-up NP symbol can do the same work in all of these cases (in addition to the waiting prediction $\overline{\text{S}}$), creating the familiar looping pattern. And Table 12 shows a similar looping pattern for right-branching structures: at the points after 'chased' and 'bit' and 'ate', a single top-down $\overline{\text{NP}}$ symbol encapsulates all of the parser's internal state. But just as neither the bottom-up nor top-down method produced this kind of looping pattern on center-embedding structures, maximum stack depth for the left-corner parser increases with the depth of embedding for these sentences, as shown in Table 10.[23]

The stack size requirements for left-corner parsing therefore show the following pattern:

$$\begin{aligned} &\mathit{MaxStack}_{LC}(\text{tree for (15b)}) = \mathit{MaxStack}_{LC}(\text{tree for (15c)}) && \text{(left-branching)} \\ &\mathit{MaxStack}_{LC}(\text{tree for (16b)}) = \mathit{MaxStack}_{LC}(\text{tree for (16c)}) && \text{(right-branching)} \\ &\mathit{MaxStack}_{LC}(\text{tree for (17b)}) < \mathit{MaxStack}_{LC}(\text{tree for (17c)}) && \text{(center-embedding)} \end{aligned}$$

which in combination with the linking hypothesis that

(29) A speaker processing the string $\mathit{PHON}(t)$ will experience greater perceptual difficulty the greater $\mathit{MaxStack}_{LC}(t)$ is.

produces the full range of predictions that we set out to achieve.

3.4 Take-home messages

Having looked at the specifics of these context-free parsing methods in some depth, what have we learnt? There are a few intended take-home messages.

- The main goal is to convey a sense of what it is that one needs to add to a grammar to produce a theory of sentence comprehension: what needs to be added is what the bottom-up, top-down and left-corner methods all manage to add to (14). The grammar "partially determines" what the resulting combined system does: it should be clear that the grammar does not fully determine it, since we have seen three distinct options, each with different empirical pros and cons (while leaving the grammar unchanged). There are of course alternatives to breaking down the sentence comprehension system into two components with the shapes that I have outlined — including, for example, the option of not breaking down it into any two components thought of as a "grammar" and a "parser" at all, as proposed by Phillips (1996). But to the extent that one would like to take as a starting point a grammar in the sense that has (for better or for worse) become conventional, things have already been carved up in a certain way; the bottom-up, top-down and left-corner parsing methods are examples of things that have the shape of what got carved off to leave behind a grammar like (14).
- The question of what sequence of steps a parser goes through to arrive at a particular structural description can be separated from the question of ambiguity resolution. The first question is whether, for example, a parser arrives at a structural description for 'the dog chased the cat' via the sequence of steps shown in Table 1, Table 5 or Table 9. The second question concerns any "wrong turns" away from these paths through the search space are taken. At least below a certain level of abstraction, the second question presupposes an answer to the first question.
- The case study of center-embedding provides a concrete illustration of what it can look like for a theory to say that a certain sentence is grammatical and yet elicits judgements of unacceptability due to a precisely characterized form of "processing difficulty". A linking hypothesis such as (29) is the kind of proposal that has the explanatory force to make predictions about sentences other than those that originally motivated it.

by setting the start configuration to have the empty stack and the goal configuration to have a bottom-up S. The effect is just that the relevant instance of LC-CONNECT "in the middle", eliminating $\overline{\text{S}}$, is replaced by a corresponding instance of LC-PREDICT, introducing S.

[23] The underlying point here on which the overall argument relies is that only center-embedding structures yield the kind of nesting patterns in strings that are beyond the reach of a finite-state machine (Chomsky, 1963, pp.394–395). The finitely many configurations that are visited in the course of parsing an arbitrarily large left-branching or right-branching structure are in effect the states of a finite-state machine.

(17b): $_0$ the $_1$ rat $_2$ the $_3$ cat $_4$ chased $_5$ fled $_6$

Transition	Rule used	Configuration
—	—	($\bar{S}$, 0)
SHIFT	D → the	(D $\bar{S}$, 1)
LC-PREDICT	NP → D N ORC	($\bar{N}$ $\overline{ORC}$ NP $\bar{S}$, 1)
MATCH	N → rat	($\overline{ORC}$ NP $\bar{S}$, 2)
SHIFT	D → the	(D $\overline{ORC}$ NP $\bar{S}$, 3)
LC-PREDICT	NP → D N	($\bar{N}$ NP $\overline{ORC}$ NP $\bar{S}$, 3)
MATCH	N → cat	(NP $\overline{ORC}$ NP $\bar{S}$, 4)
LC-CONNECT	ORC → NP V	($\bar{V}$ NP $\bar{S}$, 4)
MATCH	V → chased	(NP $\bar{S}$, 5)
LC-CONNECT	S → NP VP	($\overline{VP}$, 5)
SHIFT	V → fled	(V $\overline{VP}$, 6)
LC-CONNECT	VP → V	(ϵ, 6)

(17c): $_0$ the $_1$ rat $_2$ the $_3$ cat $_4$ the $_5$ dog $_6$ bit $_7$ chased $_8$ fled $_9$

Transition	Rule used	Configuration
—	—	($\bar{S}$, 0)
SHIFT	D → the	(D $\bar{S}$, 1)
LC-PREDICT	NP → D N ORC	($\bar{N}$ $\overline{ORC}$ NP $\bar{S}$, 1)
MATCH	N → rat	($\overline{ORC}$ NP $\bar{S}$, 2)
SHIFT	D → the	(D $\overline{ORC}$ NP $\bar{S}$, 3)
LC-PREDICT	NP → D N ORC	($\bar{N}$ $\overline{ORC}$ NP $\overline{ORC}$ NP $\bar{S}$, 3)
MATCH	N → cat	($\overline{ORC}$ NP $\overline{ORC}$ NP $\bar{S}$, 4)
SHIFT	D → the	(D $\overline{ORC}$ NP $\overline{ORC}$ NP $\bar{S}$, 5)
LC-PREDICT	NP → D N	($\bar{N}$ NP $\overline{ORC}$ NP $\overline{ORC}$ NP $\bar{S}$, 5)
MATCH	N → dog	(NP $\overline{ORC}$ NP $\overline{ORC}$ NP $\bar{S}$, 6)
LC-CONNECT	ORC → NP V	($\bar{V}$ NP $\overline{ORC}$ NP $\bar{S}$, 6)
MATCH	V → bit	(NP $\overline{ORC}$ NP $\bar{S}$, 7)
LC-CONNECT	ORC → NP V	($\bar{V}$ NP $\bar{S}$, 7)
MATCH	V → chased	(NP $\bar{S}$, 8)
LC-CONNECT	S → NP VP	($\overline{VP}$, 8)
SHIFT	V → fled	(V $\overline{VP}$, 9)
LC-CONNECT	VP → V	(ϵ, 9)

Table 10: *The effect of* ***center-embedding*** *on* ***left-corner*** *parsing. Memory load increases as embedding depth increases: maximum of 5 symbols for* (17b) *but 7 symbols for* (17c).

(15b): $_0$ John $_1$'s $_2$ dog $_3$ fled $_4$

Transition	Rule used	Configuration
—	—	($\bar{S}$, 0)
SHIFT	NP → John	(NP $\bar{S}$, 1)
LC-PREDICT	NP → NP POSS N	($\overline{POSS}$ $\bar{N}$ NP $\bar{S}$, 1)
MATCH	POSS → 's	($\bar{N}$ NP $\bar{S}$, 2)
MATCH	N → dog	(NP $\bar{S}$, 3)
LC-CONNECT	S → NP VP	($\overline{VP}$, 3)
SHIFT	V → fled	(V $\overline{VP}$, 4)
LC-CONNECT	VP → V	(ϵ, 4)

(15c): $_0$ John $_1$'s $_2$ wife $_3$'s $_4$ dog $_5$ fled $_6$

Transition	Rule used	Configuration
—	—	($\bar{S}$, 0)
SHIFT	NP → John	(NP $\bar{S}$, 1)
LC-PREDICT	NP → NP POSS N	($\overline{POSS}$ $\bar{N}$ NP $\bar{S}$, 1)
MATCH	POSS → 's	($\bar{N}$ NP $\bar{S}$, 2)
MATCH	N → wife	(NP $\bar{S}$, 3)
LC-PREDICT	NP → NP POSS N	($\overline{POSS}$ $\bar{N}$ NP $\bar{S}$, 3)
MATCH	POSS → 's	($\bar{N}$ NP $\bar{S}$, 4)
MATCH	N → dog	(NP $\bar{S}$, 5)
LC-CONNECT	S → NP VP	($\overline{VP}$, 5)
SHIFT	V → fled	(V $\overline{VP}$, 6)
LC-CONNECT	VP → V	(ϵ, 6)

Table 11: *The effect of* ***left-embedding*** *on* ***left-corner*** *parsing. No increase in memory load as embedding depth increases: maximum of 4 symbols in both cases.*

(16b): $_0$ Mary $_1$ chased $_2$ the $_3$ cat $_4$ that $_5$ bit $_6$ the $_7$ rat $_8$

Transition	Rule used	Configuration
—	—	($\bar{S}$, 0)
SHIFT	NP → Mary	(NP $\bar{S}$, 1)
LC-CONNECT	S → NP VP	($\overline{VP}$, 1)
SHIFT	V → chased	(V $\overline{VP}$, 2)
LC-CONNECT	VP → V NP	($\overline{NP}$, 2)
SHIFT	D → the	(D $\overline{NP}$, 3)
LC-CONNECT	NP → D N SRC	($\bar{N}$ $\overline{SRC}$, 3)
MATCH	N → cat	($\overline{SRC}$, 4)
SHIFT	C → that	(C $\overline{SRC}$, 5)
LC-CONNECT	SRC → C VP	($\overline{VP}$, 5)
SHIFT	V → bit	(V $\overline{VP}$, 6)
LC-CONNECT	VP → V NP	($\overline{NP}$, 6)
SHIFT	D → the	(D $\overline{NP}$, 7)
LC-CONNECT	NP → D N	($\bar{N}$, 7)
MATCH	N → rat	(ϵ, 8)

(16c): $_0$ Mary $_1$ chased $_2$ the $_3$ cat $_4$ that $_5$ bit $_6$ the $_7$ rat $_8$ that $_9$ ate $_{10}$ the $_{11}$ cheese $_{12}$

Transition	Rule used	Configuration
—	—	($\bar{S}$, 0)
SHIFT	NP → Mary	(NP $\bar{S}$, 1)
LC-CONNECT	S → NP VP	($\overline{VP}$, 1)
SHIFT	V → chased	(V $\overline{VP}$, 2)
LC-CONNECT	VP → V NP	($\overline{NP}$, 2)
SHIFT	D → the	(D $\overline{NP}$, 3)
LC-CONNECT	NP → D N SRC	($\bar{N}$ $\overline{SRC}$, 3)
MATCH	N → cat	($\overline{SRC}$, 4)
SHIFT	C → that	(C $\overline{SRC}$, 5)
LC-CONNECT	SRC → C VP	($\overline{VP}$, 5)
SHIFT	V → bit	(V $\overline{VP}$, 6)
LC-CONNECT	VP → V NP	($\overline{NP}$, 6)
SHIFT	D → the	(D $\overline{NP}$, 7)
LC-CONNECT	NP → D N SRC	($\bar{N}$ $\overline{SRC}$, 7)
MATCH	N → rat	($\overline{SRC}$, 8)
SHIFT	C → that	(C $\overline{SRC}$, 9)
LC-CONNECT	SRC → C VP	($\overline{VP}$, 9)
SHIFT	V → ate	(V $\overline{VP}$, 10)
LC-CONNECT	VP → V NP	($\overline{NP}$, 10)
SHIFT	D → the	(D $\overline{NP}$, 11)
LC-CONNECT	NP → D N	($\bar{N}$, 11)
MATCH	N → cheese	(ϵ, 12)

Table 12: *The effect of* ***right-embedding*** *on* ***left-corner*** *parsing. No increase in memory load as embedding depth increases: maximum of 3 symbols in both cases.*

- In comparison with Section 2, considering specific parsing methods demonstrates the lengths to which one *does not* need to go in order for information-theoretic complexity metrics to get off the ground. In at least one reasonable sense, the linking hypotheses in Section 2 concern not how parsing happens, but rather what parsing achieves; accordingly, they can be described as pertaining to Marr's (1982) "computational level", in contrast to the linking hypotheses in this section which pertain to the "algorithmic level".

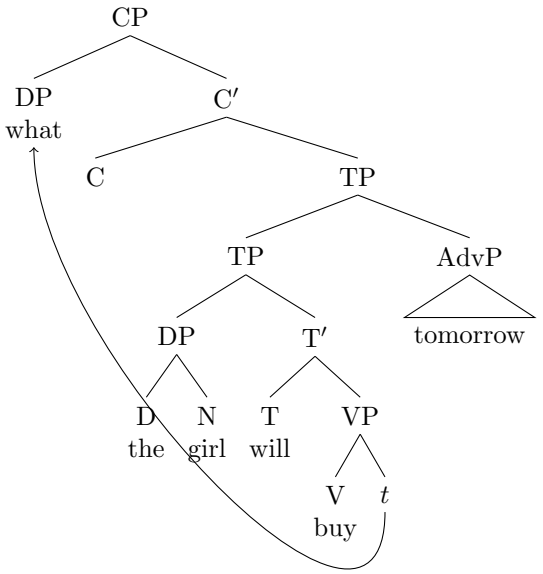
4 Connecting to contemporary syntax

In both of the preceding sections, I have used simple finite-state or context-free grammars for illustrative purposes. Of course modern theories of natural language syntax do not generally take this form, so it is important to ask how the various information-theoretic and automata-theoretic concepts outlined in the previous sections can be connected to more expressive kinds of grammars along the lines of what contemporary syntacticians are working with. A line of work descending from Stabler (1997) has shed light on this question as it relates to modern theories in the minimalist tradition, and this is what I will focus on here; but to a degree that is perhaps surprising, a similar story can be told for other systems such as TAGs and CCGs.

The crucial underlying issue is to identify exactly how minimalist grammars differ from CFGs, or exactly what minimalist grammars add to CFGs. An understanding of this provides a clear picture of what needs to be finessed in order for minimalist grammars to be plugged in to linking hypotheses of the sorts outlined in the previous sections. Michaelis (2001) showed that the minimalist grammars formulated in Stabler (1997) are in fact very similar to CFGs at a certain significant level of abstraction (see also Kobele et al. (2007)). Since the ways in which the ideas from previous sections relate to CFGs are generally well-understood, this paved the way for many of these ideas to be adapted to minimalist grammars.

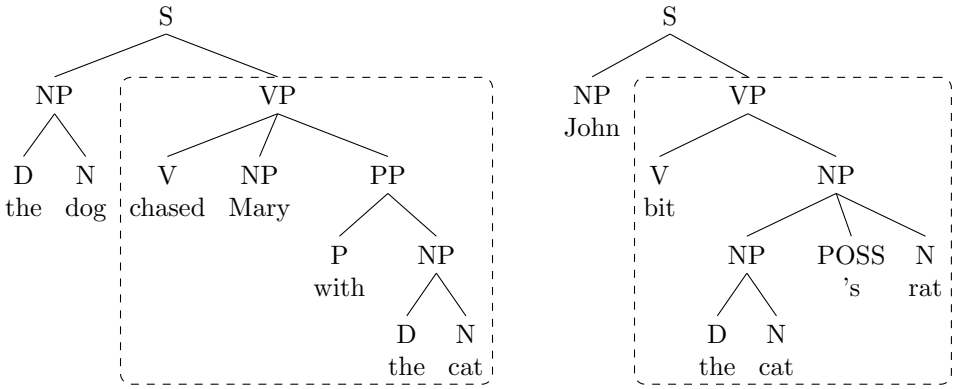
This means that a certain amount (not all) of the work involved in understanding how to formulate interesting linking hypotheses for minimalist grammars just *is* the work of understanding how to do so for CFGs. The same ideas play important roles. But understanding exactly how those familiar important ideas fit into the ecosystem of minimalist grammars involves a certain adjustment of perspective, because of the abstract level at which the crucial similarities identified by Michaelis reside. In other words, while there are objects in the minimalist grammar ecosystem that one can reason about using the thought patterns that are familiar from CFGs (that is the good news, which makes the easy part of the task easy), they are not objects that linguists are generally in the habit of writing down (this is the bad news, which makes the hard part of the task hard). In particular, there are objects in the minimalist grammar ecosystem that license the same simple, familiar patterns of reasoning that we naturally apply to CFG trees like the ones in (19), (20) and (21) above (the good news); but the relevant objects are not, despite surface similarities, the trees conventionally used to illustrate transformational derivations like the one in (30) (the bad news).

(30)



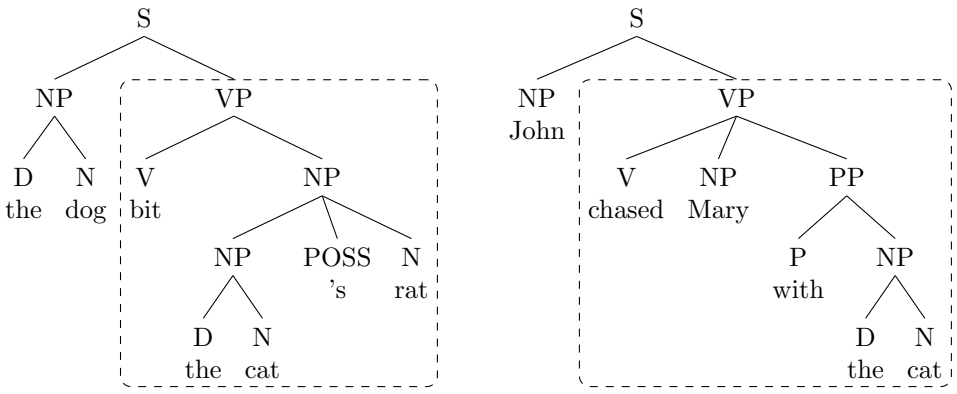
The pivotal property that is familiar from CFGs (and present in an obscured underlying sense in minimalist grammars) is a certain kind of *interchangeability* of subexpressions. Consider the two trees shown in (31), both generated by the grammar used in Section 3.

(31)



Both contain a node labeled VP. A fundamental consequence of this is that the two subtrees dominated by those two VP nodes are interchangeable, in the sense that if we swap one for the other we are guaranteed to get more trees that are well-formed according to the same grammar. Specifically, we can swap the VP subtrees around to get the two trees in (32), which are also both generated by the same grammar.

(32)



This is a consequence of the way the "goodness as a VP" of a particular subexpression is independent of the environment in which that subexpression might appear. When we describe something of the form 'VP → V NP' as a context-free rule, what we are saying is precisely that the right-hand side of the rule is a valid way for a VP to be constituted no matter what context this VP might be appearing in. Since the phrase 'bit the cat's rat' is good enough to fit under the node labeled VP in the second tree in (31), it can't fail to be good enough to fit under the node labeled VP in the first tree in (32) — it could only fail if there were conditions on "VP-hood" that depended on the environment into which a putative VP is to be put, but by assumption there are none.

This kind of modularity of tree structures is exactly what makes CFGs easy to work with, and plays an important role in operationalizing the linking hypotheses discussed above in combination with CFGs. Recall from Section 2 the idea of generating a sentence by generating independently chosen subparts, and the way this is the key point of contact between a hypothesized grammatical structure and the corresponding range of probability distributions. The relationship between (31) and (32) could be restated in probabilistic terms[24] as follows: since

$$P(\text{S} \rightarrow \text{the dog chased Mary with the cat}) = P(\text{S} \rightarrow \text{NP VP}) \times P(\text{NP} \rightarrow^* \text{the dog}) \times P(\text{VP} \rightarrow^* \text{chased Mary with the cat})$$

and

$$P(\text{S} \rightarrow \text{John bit the cat's rat}) = P(\text{S} \rightarrow \text{NP VP}) \times P(\text{NP} \rightarrow^* \text{John}) \times P(\text{VP} \rightarrow^* \text{bit the cat's rat})$$

we can conclude from the fact that these two sentences have non-zero probabilities that the six multiplied probabilities on the right-hand sides of these equations are also all non-zero; and by glueing the pieces together differently we can conclude that

$$P(\text{S} \rightarrow \text{the dog bit the cat's rat}) = P(\text{S} \rightarrow \text{NP VP}) \times P(\text{NP} \rightarrow^* \text{the dog}) \times P(\text{VP} \rightarrow^* \text{bit the cat's rat}) > 0$$

and

$$P(\text{S} \rightarrow \text{John chased Mary with the cat}) = P(\text{S} \rightarrow \text{NP VP}) \times P(\text{NP} \rightarrow^* \text{John}) \times P(\text{VP} \rightarrow^* \text{chased Mary with the cat}) > 0$$

mirroring the conclusion about categorial grammaticality above.[25]

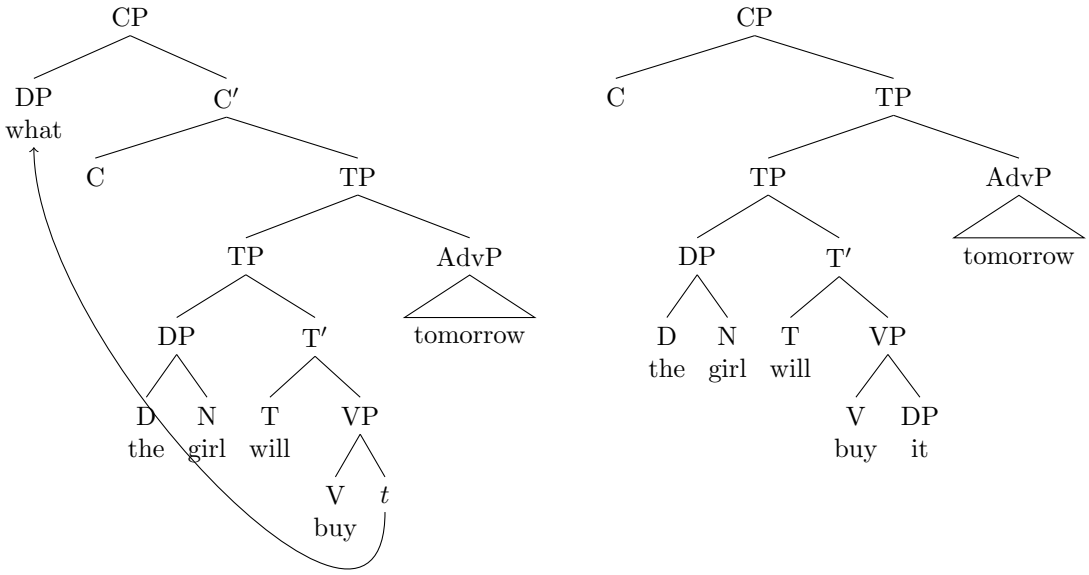
The role that this interchangeability plays in the automata-theoretic models from Section 3 is perhaps slightly less obvious but still significant. The way in which a parsing system can process unboundedly large structures of a certain sort involved the presence of a loop in the contents of the stack. Take, for example, the looping shown in Table 12, where we return to configurations where the stack contains just a single NP prediction. The reason that this single stack symbol suffices *both* after processing only the first two words 'Mary chased' of (16b) *and* after processing the first six words 'Mary chased the cat that bit' is exactly that the two corresponding remaining portions — 'the cat that bit the rat' and 'the rat' — are interchangeable by virtue of both being NPs. Similarly, the looping shown in Table 3 is a consequence of the way the difference between having consumed 'John' and having consumed 'John's dog' is irrelevant to what may come next, precisely because 'John' and 'John's dog' can go in all the same places.

For comparison, consider now two trees of the sort usually used to represent minimalist-style derivations.

[24] And lazily blurring the distinction between trees and strings for a moment.

[25] Put differently: we have done the same calculation twice, once in the boolean semiring and once in the probability semiring.

(33)

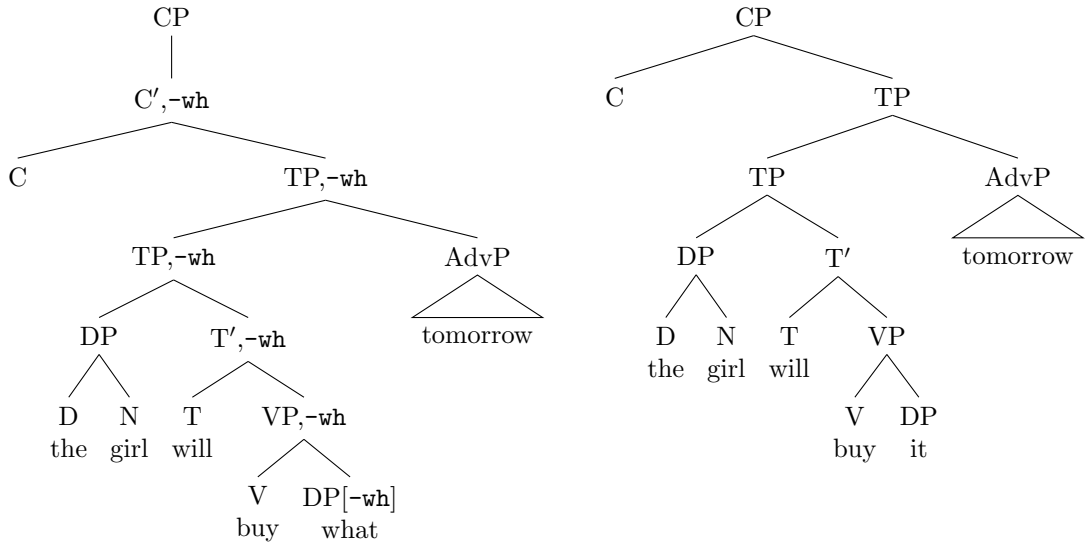


Notice that both of the trees in (33) contain a node labeled VP, just as the two trees in (31) did. But here, this does *not* license a conclusion that a certain subpart of one tree can be swapped with the other. The simple VP 'buy it' in the tree on the right cannot be substituted for any corresponding constituent in the left tree. And there is no "VP constituent" of the tree on the left that can be substituted into the corresponding position on the right. (It doesn't matter whether we imagine that the VP constituent in the left tree contains both the moved phrase 'what' and its trace, or only the trace: if we take the relevant constituent to include both, then the head of this chain will have no appropriate slot to fit into since the tree on the right has a non-interrogative C head; if we take the relevant constituent to include only the trace, then we will have something equivalent to an unbound trace.) While both trees have a node labeled VP, they do not have any corresponding interchangeable subparts in the sense illustrated above for CFGs.

So what it means for a node to be labeled VP (or anything else) in trees of the sort in (33) is simply not the same as what it means for a node to labeled VP (or anything else) in trees of the sort in (31). The presence of movement arrows does not only distort the surface word order — it also puts tangles into the otherwise modular workings of the grammar, and this modularity was the key to operationalizing the various linking hypotheses discussed above. Although transformational grammars can in a sense be thought of as the result of "adding movement to a CFG", this does not mean that the parts of the trees in (33) that are not movement arrows can be understood exactly as all of the tree structure in (31) can.

This is, as I mentioned above, the bad news; a shift in perspective is required. The good news is that there *does exist* a different way of saying what the parts of 'what the girl will buy tomorrow' are and giving labels to those parts, such that parts with the same labels can be interchanged just as they could in CFGs. The tree that says what those parts are, what labels they have, and how they are put together, is shown on the left in (34). The perspective that we are switching to does not require any changes to how we think of movement-free derivations, so the second tree in (34) is the same as the second tree in (33).

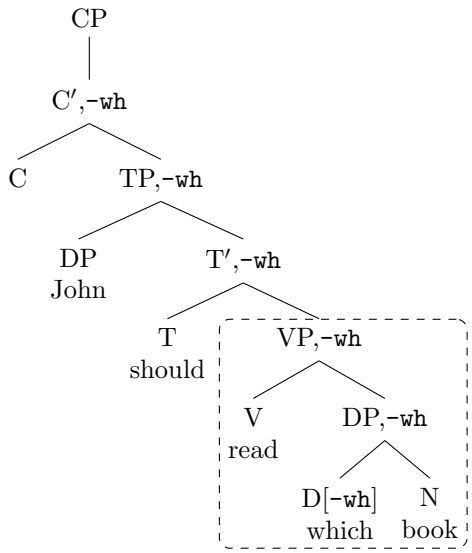
(34)



The fact that the derivation on the left has no subpart that can be interchanged with the VP 'buy it' is now accurately reflected in the fact that no node in the left tree shares this label. Writing 'DP[-wh]' rather than 'DP' as the label for 'what' simply says that it is the kind of thing that undergoes wh-movement. When we write things like 'VP,-wh' and 'TP,-wh', we are indicating the parts of the tree that, due to the tangles introduced by movement, are not interchangeable with subparts that simply bear the labels 'VP' and 'TP'. By encoding the fact that the tangling extends up as high as the C′,-wh node but not the CP node immediately above it, these annotations also encode the fact that the CP expression was constructed out of the C′,-wh expression in a way that involved resolving the tangle — in other words, by satisfying the requirement that 'what' undergoes wh-movement — so there is no need to also draw a line connecting the CP node to 'what'.

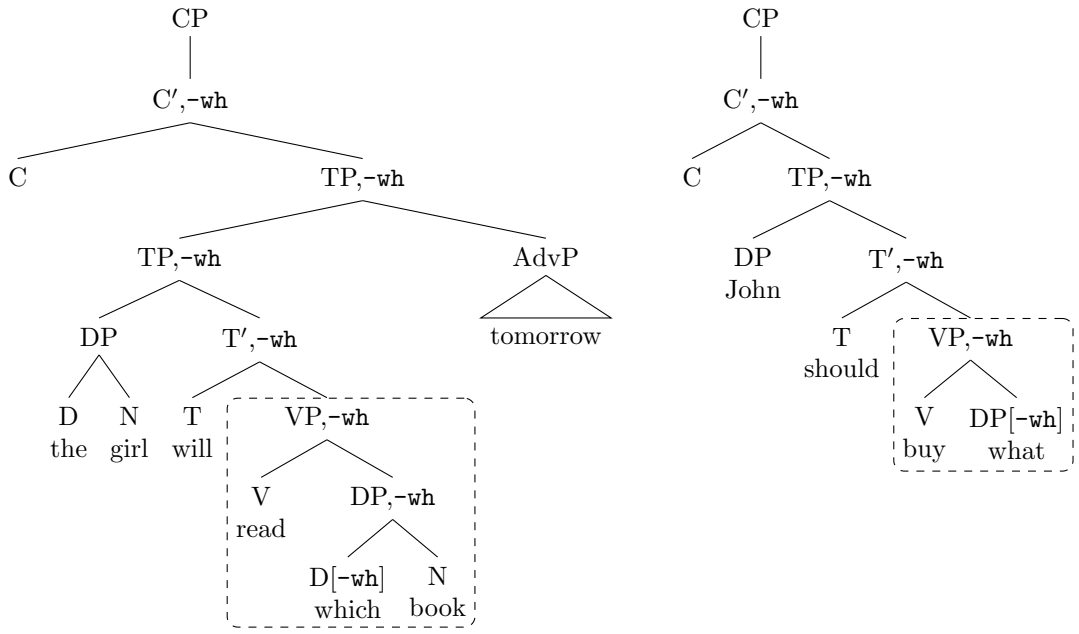
Such trees not only bring to the surface the fact that nothing in the left tree can be interchanged with 'buy it', they also make transparent the interchangeability relations that the expression constructed out of 'buy' and 'what' *does* participate in. The corresponding tree for 'which book John should read' is shown in (35).

(35)



This tree *does* have a node labeled 'VP,-wh', and this now does license the conclusion that certain other expressions will inevitably be grammatical. Specifically, we can swap around this subtree with the one bearing the same label in the tree on the left in (34), to produce these new grammatical trees:

(36)



What is happening here is that we are re-arranging the pieces of the two expressions

(37) a. what the girl will buy tomorrow (i.e. left tree in (34))
b. which book John should read (i.e. tree in (35))

to yield the pair

(38) a. which book the girl will read tomorrow (i.e. left tree in (36))
b. what John should buy (i.e. right tree in (36))

in just the same way we did for the CFG trees above. The pieces being swapped, indicated by boxes in the trees above, do not correspond to contiguous portions of the eventual linearized strings as they do with CFGs — but to get too distracted by this detail would be to focus unduly on these strings rather than the structure-building machinery of the grammar.

The trees in (34), (35) and (36) can be described as derivation trees. By adopting this representation, we focus attention on the way independently chosen pieces were snapped together to form a larger whole. The trees in (33), in contrast, give priority to surface constituency. In the case of a CFG one need not choose between which of these two properties to focus on, because the two are conflated — or perhaps we should say *confounded*. These derivation trees bear a significant resemblance to T-markers in Chomsky (1965), including the way binary-branching nodes represent operations that combine two expressions (i.e. generalized transformations) and unary-branching nodes represent operations that adjust the surface word order of an existing expression (i.e. singulary transformations). They focus attention on what the grammar generates and how it generates those things, rather than how those things are pronounced. The perspective they provide is therefore in line with the recent trend in minimalist syntax towards thinking of externalization as a relatively incidental aspect of the human language system. Abandoning the simple and familiar relationship between externalization and structure exhibited by CFG trees is arguably an overdue step that needs to be taken in order to bring our thinking fully into line with the fact that natural language grammars are not CFGs; perhaps adding movement arrows like in (33) was a temporarily useful cheap quick-fix.

In short, derivation trees allow us to think about the *range of possibilities* allowed by a minimalist grammar in exactly the same modular, tractable way that we think about the range of possibilities allowed by a CFG.[26]

For work that has combined minimalist grammars with the information-theoretic linking hypotheses from Section 2, this firm grasp on the range of possibilities essentially provides an immediate solution to the question of how to formulate probability distributions over derivations: just do to the trees in (34) and (35) what is standardly done

[26]What has been made more complicated by the shift away from trees like (33) is essentially the issue of linearization, and so this is where the finessing and adapting mentioned at the beginning of this section remains to be done.

to CFG trees; see e.g. section 4 of Yun et al. (2015).[27] Various more elaborate approaches to defining probability distributions over a CFG can also be imported to the minimalist grammar case (Hunter and Dyer, 2013), but the underlying CFG-like structure is what makes all of this possible.

As regards adapting the parsing methods from Section 3 to minimalist grammars, Stabler (2013) presented a "top-down minimalist parser" that in a sense does to the trees in (34) and (35) what the top-down method described in Section 3 does to standard CFG trees. This system has been used as the basis for for formulating and testing linking hypotheses along the lines of the stack-depth idea (e.g. Kobele et al., 2013; Graf et al., 2015, 2017). Given the complicated relationship between minimalist derivation trees and surface word order, Stabler's relatively direct application of the top-down method yields a parser that lacks a certain kind of "incrementality" in its treatment of movement dependencies; see Hunter (forthcoming) for discussion and a different proposal that tries to adapt the left-corner method to minimalist grammars instead. All of this work frames the parsing question as one of snapping together the modular, interchangeable parts of trees like those in (34) and (35), in a manner analogous to the way the parsing methods in Section 3 compose the modular parts of conventional CFG trees.

References

Abney, S. P. and Johnson, M. (1991). Memory requirements and local ambiguities of parsing strategies. *Journal of Psycholinguistic Research*, 20(3):233–250.

Bar-Hillel, Y., Perles, M., and Shamir, E. (1961). On formal properties of simple phrase-structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14:143–172.

Chomsky, N. (1963). Formal properties of grammars. In Luce, R. D., Bush, R. R., and Galanter, E., editors, *Handbook of Mathematical Psychology*, volume 2, pages 323–418. John Wiley and Sons, Inc., New York and London.

Chomsky, N. (1965). *Aspects of the Theory of Syntax*. MIT Press, Cambridge, MA.

Chomsky, N. and Miller, G. (1963). Introduction to the formal analysis of natural languages. In Luce, R. D., Bush, R. R., and Galanter, E., editors, *Handbook of Mathematical Psychology*, volume 2, pages 269–321. John Wiley and Sons, Inc., New York and London.

Frazier, L. and Clifton, C. (1996). *Construal*. MIT Press, Cambridge, MA.

Frazier, L. and Rayner, K. (1982). Making and correcting errors during sentence comprehension: Eye movements in the analysis of structurally ambiguous sentences. *Cognitive Psychology*, 14:178–210.

Graf, T., Fodor, B., Monette, J., Rachiele, G., Warren, A., and Zhang, C. (2015). A refined notion of memory usage for minimalist parsing. In *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*, pages 1–14. Association for Computational Linguistics.

Graf, T., Monette, J., and Zhang, C. (2017). Relative clauses as a benchmark for Minimalist parsing. *Journal of Language Modelling*, 5:57–106.

Hale, J. T. (2001). A probabilistic earley parser as a psycholinguistic model. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics*.

Hale, J. T. (2003). *Grammar, Uncertainty and Sentence Processing*. PhD thesis, Johns Hopkins University.

Hale, J. T. (2006). Uncertainty about the rest of the sentence. *Cognitive Science*, 30:643–672.

Hale, J. T. (2016). Information-theoretical complexity metrics. *Language and Linguistics Compass*, 10(9):397–412.

Hale, J. T. (2017). Models of human sentence comprehension in computational psycholinguistics. In *Oxford Research Encyclopedia of Linguistics*.

Hunter, T. (forthcoming). Left-corner parsing of minimalist grammars. In Berwick, B. and Stabler, E., editors, *Minimalist Parsing*. Oxford University Press.

Hunter, T. and Dyer, C. (2013). Distributions on minimalist grammar derivations. In *Proceedings of the 13th Meeting on the Mathematics of Language*.

[27] The remaining necessary finessing, as a result of shifting complications to linearization, involved the "intersection" step: deriving an intersection grammar representing the comprehender's knowledge at each intermediate point of the sentence.

Jelinek, F. and Lafferty, J. D. (1991). Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315–323.

Kanazawa, M. (2016). Formal grammar: An introduction. Online lecture notes: http://research.nii.ac.jp/ kanazawa/FormalGrammar/index.html.

Kobele, G. M., Gerth, S., and Hale, J. (2013). Memory resource allocation in top-down minimalist parsing. In Morrill, G. and Nederhof, M.-J., editors, *Formal Grammar 2012/2013*, volume 8036 of *Lecture Notes in Computer Science*, pages 32–51. Springer.

Kobele, G. M., Retoré, C., and Salvati, S. (2007). An automata theoretic approach to minimalism. In Rogers, J. and Kepser, S., editors, *Proceedings of the Workshop: Model-theoretic syntax at 10*. Dublin.

Levy, R. (2008). Expectation-based syntactic comprehension. *Cognition*, 106(3):1126–1177.

Levy, R. (2013). Memory and surprisal in human sentence comprehension. In van Gompel, R. P. G., editor, *Sentence Processing*, page 78114. Hove: Psychology Press.

Marr, D. (1982). *Vision: A computational investigation into the human representation and processing of visual information*. W.H. Freeman and Company, New York.

Michaelis, J. (2001). *On Formal Properties of Minimalist Grammars*. PhD thesis, Universität Potsdam.

Miller, G. A. and Chomsky, N. (1963). Finitary models of language users. In Luce, R. D., Bush, R. R., and Galanter, E., editors, *Handbook of Mathematical Psychology*, volume 2. Wiley and Sons, New York.

Nederhof, M. J. and Satta, G. (2003). Probabilistic parsing as intersection. In *8th International Workshop on Parsing Technologies*, pages 137–148. LORIA, Nancy, France.

Nederhof, M. J. and Satta, G. (2008). Computing partition functions of pcfgs. *Research on Language and Computation*, 6(2):139–162.

Phillips, C. (1996). *Order and Structure*. PhD thesis, MIT.

Resnik, P. (1992). Left-corner parsing and psychological plausibility. In *Proceedings of the Fourteenth International Conference on Computational Linguistics (COLING '92)*.

Stabler, E. P. (1997). Derivational minimalism. In Retoré, C., editor, *Logical Aspects of Computational Linguistics*, volume 1328 of *LNCS*, pages 68–95, Berlin Heidelberg. Springer.

Stabler, E. P. (2013). Two models of minimalist, incremental syntactic analysis. *Topics in Cognitive Science*, 5(3):611–633.

Stolcke, A. (1995). An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):167–201.

Wolf, F. and Gibson, E. (2006). Parsing: Overview. In *Encyclopedia of Cognitive Science*. John Wiley & Sons.

Yngve, V. H. (1960). A model and an hypothesis for language structure. In *Proceedings of the American Philosophical Society*, volume 104, pages 444–466.

Yun, J., Chen, Z., Hunter, T., Whitman, J., and Hale, J. (2015). Uncertainty in the processing of relative clauses in East Asian languages. *Journal of East Asian Linguistics*, 24(2).