Lecture 4

Top-Down Parsing: Psycholinguistic Adequacy

1 Two Basic Advantages

1.1 Incrementality

The most basic requirement for a psychologically plausible parser is that it works in an incremental fashion, that is to say, parsing can take place as soon as the first word of the sentence has been uttered, rather than delaying parsing until the full sentence has been heard. You showed in an earlier exercise that the top-down parser only needs to keep track of the starting position of an item, which entails that it can be run in an incremental fashion.

1.2 Simplicity

Top-down parsers are conceptually simple. The top-down orientation of the parser mirrors the view of CFGs as top-down generators. In addition, the inference rules are extremely simple (compared to, say, *left-corner parsing*, which we will discuss at a later point), seeing how the inference rule of the parser directly mirrors the rewrite rules of the grammar. This is particularly appealing for proponents of a *transparent parser*. A transparent parser must use the grammar in as direct a fashion as possible. That precludes processing-based explanations of syntactic phenomena — e.g. island effects — and prioritizes a maximally simple parser.

2 Garden Path Effects

2.1 Garden Paths as Backtracking

One of the best known phenomena in syntactic processing are *garden path effects*, which arise with sentences that are grammatically well-formed but nonetheless difficult to parse (Frazier 1979; Frazier and Rayner 1982). More precisely, a garden path sentence is a sentence w that up to some w_i has a strongly preferred analysis that must be discarded at w_{i+1}. Here's a list of examples that includes more than the usual suspects:

(1) *Structural ambiguity*

a. The horse raced past the barn fell.

b. The raft floated down the river sank.
c. The player tossed a Frisbee smiled.
d. The doctor sent for the patient arrived.
e. The cotton clothing is made of grows in Mississippi.
f. Fat people eat accumulates.
g. I convinced her children are noisy.

(2) *Lexical ambiguity*

a. The old train the young.
b. The old man the boat.
c. Until the police arrest the drug dealers control the street.
d. The dog that I had really loved bones.
e. The man who hunts ducks out on weekends.

Frazier (1979) proposes to treat garden path effects as a result of reanalysis: the parser is forced to abandon its current set of hypotheses, backtrack to an earlier point, and build a new structure from there. If for some reason this process proves too difficult, the parser gets stuck and assigns no structure at all.

This kind of analysis is compatible with top-down parsing, depending on the data structure we use. Remember that the data structure keeps track of the items in the current parse, but also of alternative parses. The control structure includes a set of instructions for which parses should be prioritized. Let's try to make Frazier's account precise using a top-down parser with priority queues and a serial strategy for expanding multiple parses.

2.2 Priority Queues and Prefix Trees

Let's assume that our control structure operates on a *priority queue*, i.e. a list of parse tables that can be reordered, to which we can add new entries, and from which we can delete old entries (note: even though I call it a list, that doesn't mean queues are best implemented as lists; in Python, for example, you're better off using arrays/dictionaries). Priority queues provide an easy way of prioritizing specific parses. Suppose that our priority queue is $[i_1, \ldots, i_n]$, $n \geq 1$, where each i_j is some parse item. Since i_1 is the first item in the queue it has the highest priority and should be acted on first. The parser removes i_1 from the queue, computes all parse items than can be inferred from i_1, and reinserts them into the queue at some position corresponding to their relative priority with respect to the other items in the queue.

Example 4.1 Priority Queues for Serial and Parallel Parsing

Consider once again our toy grammar from Cha. 3.

1)	S	→	NP VP	6)	Det	→	a | the
2)	NP	→	PN	7)	N	→	car | truck | anvil
3)	NP	→	Det N	8)	PN	→	Bugs | Daffy
4)	VP	→	Vi	9)	Vi	→	fell over
5)	VP	→	Vt NP	10)	Vt	→	hit

Suppose that we have a recursive descent parser that explores the space of possible parses with a priority queue such that

- only the first item of the queue can be worked on,
- new items can only be added to the top of the queue.

Such a queue is also called a *stack*. Then the queue in the parse for *The anvil hit Daffy* grows as indicated below. We assume that the parser has some mechanism for preferring specific rewrite rules whenever more than one can apply.

queue	**parser operation**
[0,S]	axiom
[0.NP VP]	predict(1)
[0,PN VP] [0,Det N VP]	predict(2), predict(3)
[0,Bugs VP] [0,Daffy VP] [0, Det N VP]	predict(8)
[0,Daffy VP] [0,Det N VP]	failed parse
[0,Det N VP]	failed parse
[0,a N VP] [0,the N VP]	predict(6)
[0,the N VP]	failed parse
[1,N VP]	scan
[1,car VP] [1,anvil VP] [1,truck VP]	predict(7)
[1,anvil VP] [1,truck VP]	failed parse
[2,VP] [1,truck VP]	scan
[2,Vi] [2,Vt NP] [1,truck VP]	predict(4), predict(5)
[2,fell over] [2,Vt NP] [1,truck VP]	predict(9)
[2,Vt NP] [1,truck VP]	failed parse
[2,hit NP] [1,truck VP]	predict(11)
[3,NP] [1,truck VP]	scan
[3,PN] [3,Det N] [1,truck VP]	predict(2), predict(3)
[3,Bugs] [3,Daffy] [3,Det N] [1,truck VP]	predict(8)
[3,Daffy] [3,Det N] [1,truck VP]	failed parse
[4,] [3,Det N] [1,truck VP]	scan, parse found!

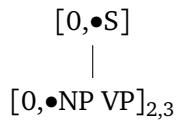
Now compare this to a recursive descent parser that operates almost exactly the same except that new items are added to the end of the queue. While the previous parser fully explored one parse before moving on to another one, this one alternates between parses after every step. In psycholinguistic parlance, then, the parser above is *serial* whereas the one below is *parallel*.

queue	parser operation
[0,S]	axiom
[0.NP VP]	predict(1)
[0,PN VP] [0,Det N VP]	predict(2), predict(3)
[0, Det N VP] [0,Bugs VP] [0,Daffy VP]	predict(8)
[0,Bugs VP] [0,Daffy VP] [0,a N VP] [0, the N VP]	predict(6)
[0,Daffy VP] [0,a N VP] [0, the N VP]	failed parse
[0,a N VP] [0, the N VP]	failed parse
[0,the N VP]	failed parse
[1,N VP]	scan
[1,car VP] [1,anvil VP] [1,truck VP]	predict(7)
[1,anvil VP] [1,truck VP]	failed parse
[1,truck VP] [2, VP]	scan
[2,VP]	failed parse
[2,Vi] [2,Vt NP]	predict(4), predict(5)
[2,Vt NP] [2,fell over]	predict(9)
[2,fell over] [2,hit NP]	predict(10)
[2,hit NP]	failed parse
[3,NP]	scan
[3,PN] [3,Det N]	predict(2), predict(3)
[3,Det N] [3,Bugs] [3,Daffy]	predict(8)
[3,Bugs] [3,Daffy] [3,a N] [3,the N]	predict(6)
[3,Daffy] [3,a N] [3, the N]	failed parse
[3,a N] [3, the N] [4,]	scan, parse found!

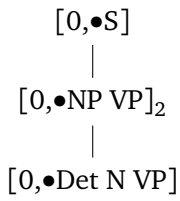
As so often, a more abstract perspective is helpful here to truly appreciate what is going on. Instead of a table listing the modifications of the priority queue, we can use a *prefix tree* as a unified way of representing multiple parses at once. The root of the prefix tree is always our start axiom [0,S] (since the final position is redundant for top-down parsing, it can be safely omitted). If parse item q is obtained from item p via a prediction or scan rule, q is added to the tree as a daughter of p. Furthermore, each node that is not a leaf is also subscripted with the list of unused predict rules. We call a node in this tree a *parse leaf* iff it is a leaf or has at least one subscript, and a (strictly descending) path spanning from the root to a leaf or a parse leaf is called a *parse path*. The set of parse tables, then, corresponds exactly to the set of parse paths, and a priority queue defines a specific traversal of the prefix tree.

Example 4.2 Parse Tables as Trees

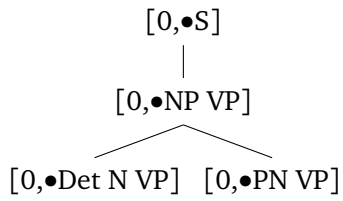
Suppose we are using the grammar from the beginning of Lecture 3 to parse *the anvil hit Daffy* with a recursive descent parser. Our parse table starts with [0,•S] as usual, from which we can only predict [0,•NP VP]. This can be represented as the tree below.



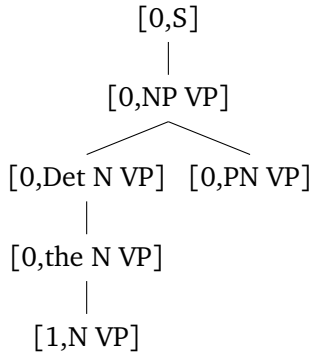
Note that [0,•NP VP] has subscripts 2 and 3 to indicate that we can use rules 2 and 3 of our CFG to predict new parse items. Suppose the parser uses predict(3) to create the item [0,•Det N VP]. Then this item is added as a daughter of [0,•NP VP] to the previous tree, and the subscript 3 is also removed from [0,•NP VP]. We do not need to add a subscript to [0,•Det N VP] since it is a leaf.



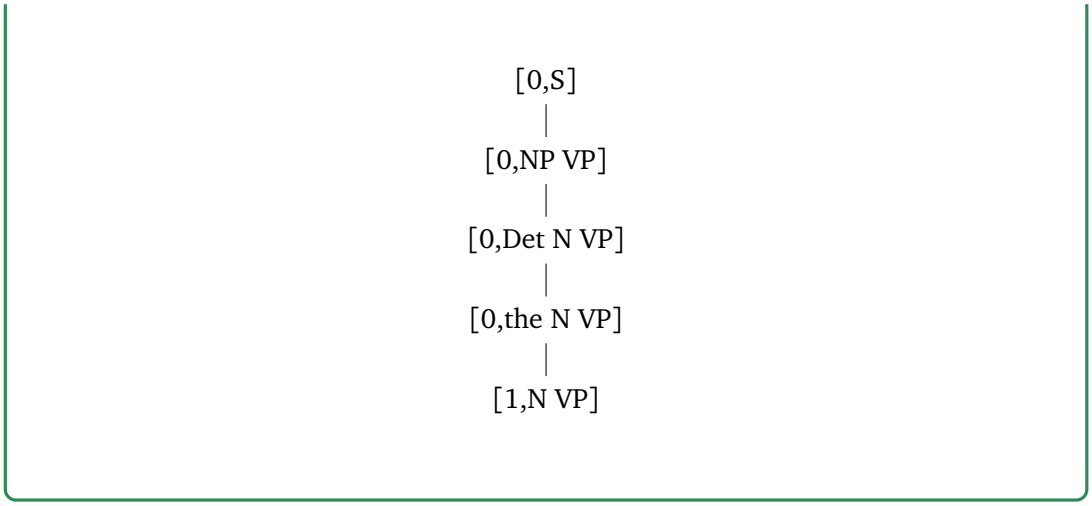
The next step of the parser now depends on how the control structure prioritizes distinct parses. We can either expand the table that ends in [0,•Det N VP], or go back to the one ending in [0,•NP VP] and apply predict(2). Suppose we do the latter, so that [0,•PN VP] is added as the second daughter of [0,•NP VP], which thus loses its last subscript.



Now let's expand [0,•Det N VP] again to obtain [0, •the N VP] and then apply a scan step, yielding [1, •N VP].



If our parser is really smart, it will be able to infer at this point that the scanned word *the* can never be obtained from [0,PN VP] (technically this is achieved by associating every parse item p with a regular expression that describes the possible left edges of the strings that can be derived from p). So the parser can remove [0,PN VP] from the tree, which is tantamount to discarding the parse table where NP was rewritten as PN.



What makes the tree representation of the parse space appealing is that the construction and prioritization of parse tables can be reduced to strategies for tree building. In particular, the familiar notions of depth-first and breadth-first carry over in a natural fashion.

depth first/serial expand a parse item p that was introduced during the previous parse step; if the parse item p cannot be expanded, expand the lowest parse leaf l that dominates p

breadth first/parallel before expanding a parse item introduced during parse step j, all parse items that were introduced at parsing step i must have been rewritten, for every $i < j$

The depth first strategy leads to a parser that always builds a single complete parse history rather than multiple partial ones. If the parse history cannot be expanded anymore, the parser either stops (successful parse) or backtracks to the last choice point in the parse history and tries a different choice instead. This corresponds exactly to the notion of serial parsing in the psycholinguistic literature, and as we saw in example 4.1 this can be implemented as a stack, i.e. a priority queue where new parse items are always added at the start of the queue.

The intuitive counterpart of serial parsing is *fully parallel parsing*, where all parse tables are built up at the same time. This is exactly the behavior of the breadth-first strategy or a priority queue where new parse items are added at the end However, even proponents of parallel parsing usually do not assume that the human parser computes and stores all options all the time. Instead, only a subset of very likely parses is claimed to be worked on in parallel, with all others either discarded or at least not expanded on. On a technical level we may formalize this in terms of a probabilistic procedure for which parse items may be expanded, and in which order. The details are not important here, the basic insights is simply that just as we can add probabilities to the control schema to regulate which inference rules the parser may apply, we can also use probabilities to prioritize the expansion of certain parse tables over others.

2.3 Formal Account of Garden Paths

A recursive descent parser with a depth-first expansion strategy for prefix trees, coupled with a preference ranking over prediction steps, can easily account for garden path

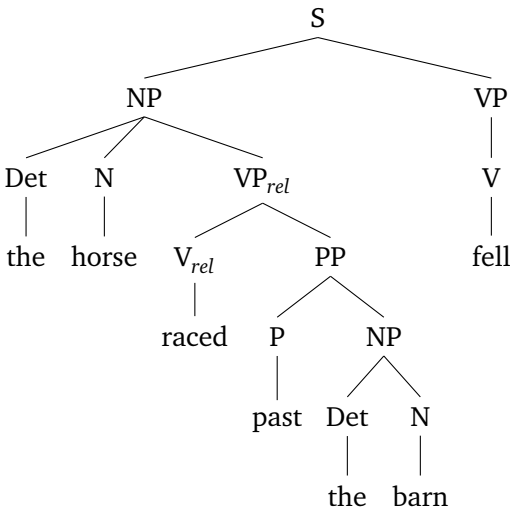
effects. In the case of *the horse raced past the barn fell*, for example, the parse builds a single parse table, and due to how certain rewrite rules are preferred over others, this parse table encodes the structure for *the horse raced past the barn*. When encountering *fell*, the parser has to backtrack. A successful parse requires backtracking to the point where *raced* is analyzed as part of the VP rather than the NP — that's quite a distance.

Example 4.3 Backtracking in *the horse raced past the barn fell*

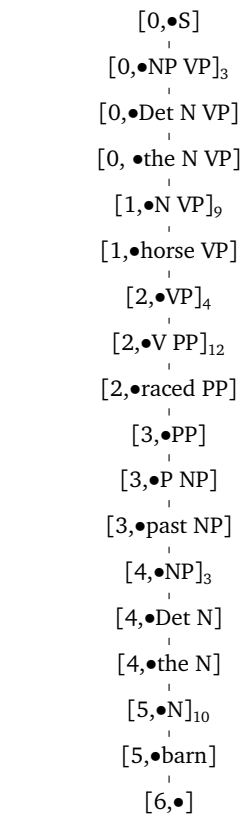
Assume we have the following (massively simplified) grammar:

1)	S	→NP VP	8)	Det	→the
2)	NP	→Det N	9)	N	→barn
3)	NP	→Det N VP_{rel}	10)	N	→horse
4)	VP	→V	11)	P	→past
5)	VP	→V PP	12)	V	→fell
6)	VP_{rel}	→V_{rel} PP	13)	V	→raced
7)	PP	→P NP	14)	V_{rel}	→raced

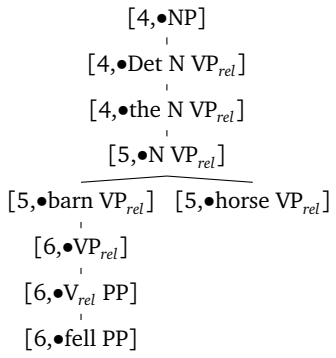
Here's the resulting tree for our garden path sentence.



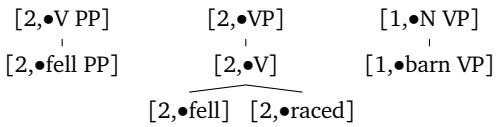
If the parser prefers NP → Det N over NP →Det N CP and operates in a recursive descent fashion, the construction of the first parse table results in the prefix tree below.



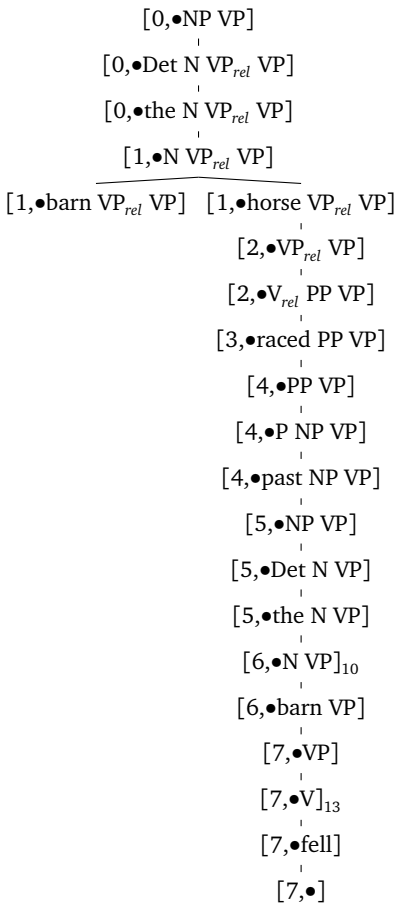
Since this parse does not succeed, the parser needs to backtrack. The closest choice point is $[5,\bullet N]_{10}$, which obviously does not fix the problem of integrating *fell* into the structure, as can be verified after a single scan step. The next choice point is $[4,\bullet NP]_3$. Here the parser still has the option of replacing NP by Det N CP, which won't help much either, but it takes quite a while to realize this because the tree for the parse table obtained by following this option involves a choice point, too.



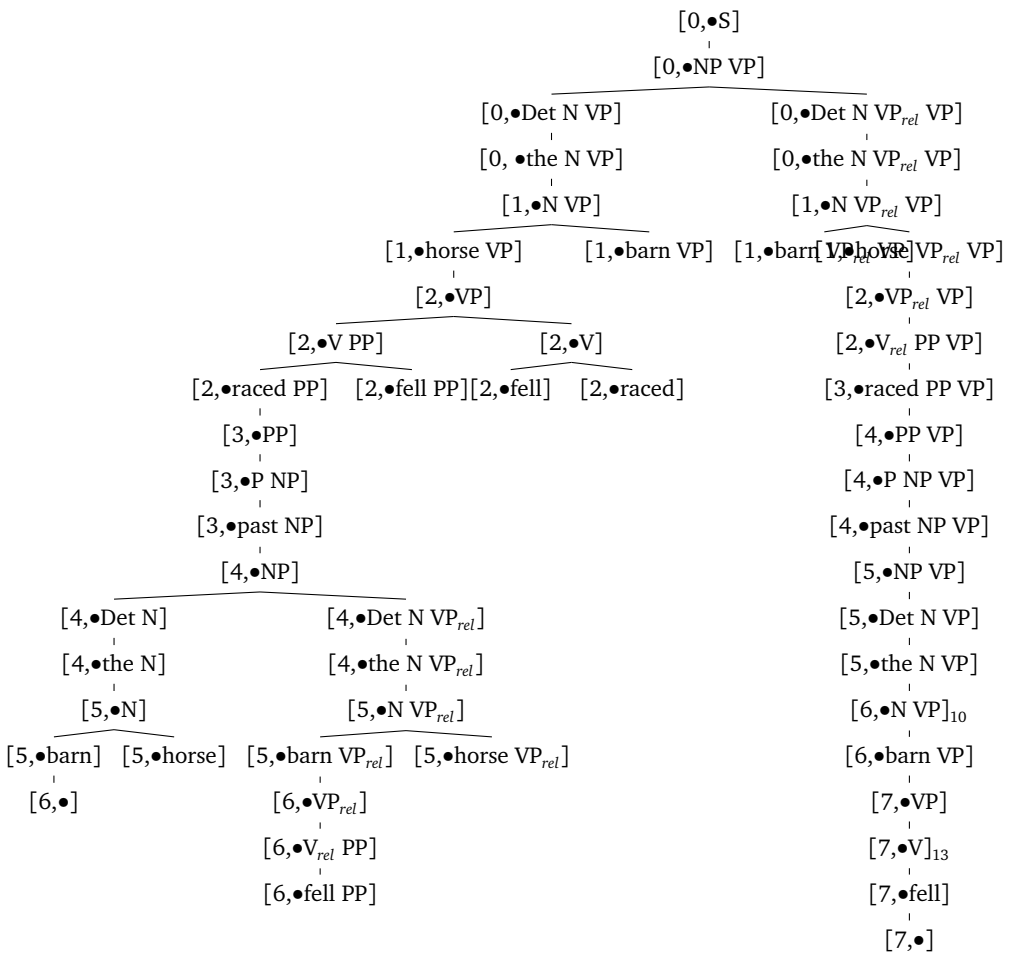
Since this venue didn't yield a successful parse either, the parse backtracks to $[2,\bullet V\ PP]_{12}$, and after this fails, to $[2,\bullet VP]_4$. Once again it is not successful, and the same holds once it expands [1,•N VP].



Only if the parser backtracks all the way to $[0,\bullet\text{NP VP}]_3$, essentially undoing all its work so far, can it find a working parse.



The prefix tree for all the parse histories built by the parser before it encounters a successful parse it much bigger than the simple phrase structure tree for the sentence.



In sum, the Frazier model can be defined more rigorously as a recursive descent parser with a stack as its priority queue. As the Frazier model is designed to account for garden paths, it is no surprise that its formal counterparts makes the right processing predictions in this case.

3 Center Embedding

It is a well-known fact that center embedding constructions are harder to parse than right embedding constructions, even if one controls for sentence length, discourse and vocabulary.

(3) a. I bought the cheese (that) the mouse (that) the cat (that) the dog chased ate wanted.

b. I bought the cheese (that) the mouse wanted (that) the cat ate (that) the dog wanted.

A common idea is that the difficulty of center embedding has to do with an increased memory burden, either because more items have to be kept in memory for a non-trivial

amount of time, or because specific items have to be stored for a longer time, or a combination of the two. The specifics vary between accounts.

Kimball (1973), for instance, proposes that the parser cannot work on more than two sentences at once, i.e. it is possible to keep up to two CPs in memory, but not more than that. Right embedding of CPs is fine because opening a new CP is the last step needed to close the containing CP, which can subsequently be removed from memory. Center embedding, on the other hand, requires that both CPs be stored in memory, so it is limited to one level of embedding.

Gibson's (1998) *Syntactic Prediction Locality Theory* (SPLT) contends that memory burden increases the more dependencies need to be stored at the same time, and since center embedding of the type NP-S-VP necessarily involves starting a new dependency before the old one between NP and VP has been finished, center embedding is difficult.

This argument can be ported into our more formal setting via a *linking hypothesis* between control structure and processing difficulty. Remember that we can annotate phrase structure trees to indicate the order in which nodes are introduced and rewritten by the parser. Obviously any item that is not rewritten immediately after its introduction — i.e. any item whose prefix and suffix differ by more than 1 — needs to be stored in memory. Let's put these items in boxes so that they are easy to pick out at a glance. Parsing difficulty, then, can be measured in a variety of ways.

Tenure the *tenure of node n* is the difference between its subscript and its superscript

Payload the *payload of tree t* is the number of nodes whose tenure is strictly greater than 1

These two measures can be combined to create a variety of difficulty metrics such as the two below (cf. Kobele et al. 2013, Graf and Marcinek 2014, and Graf et al. 2015).

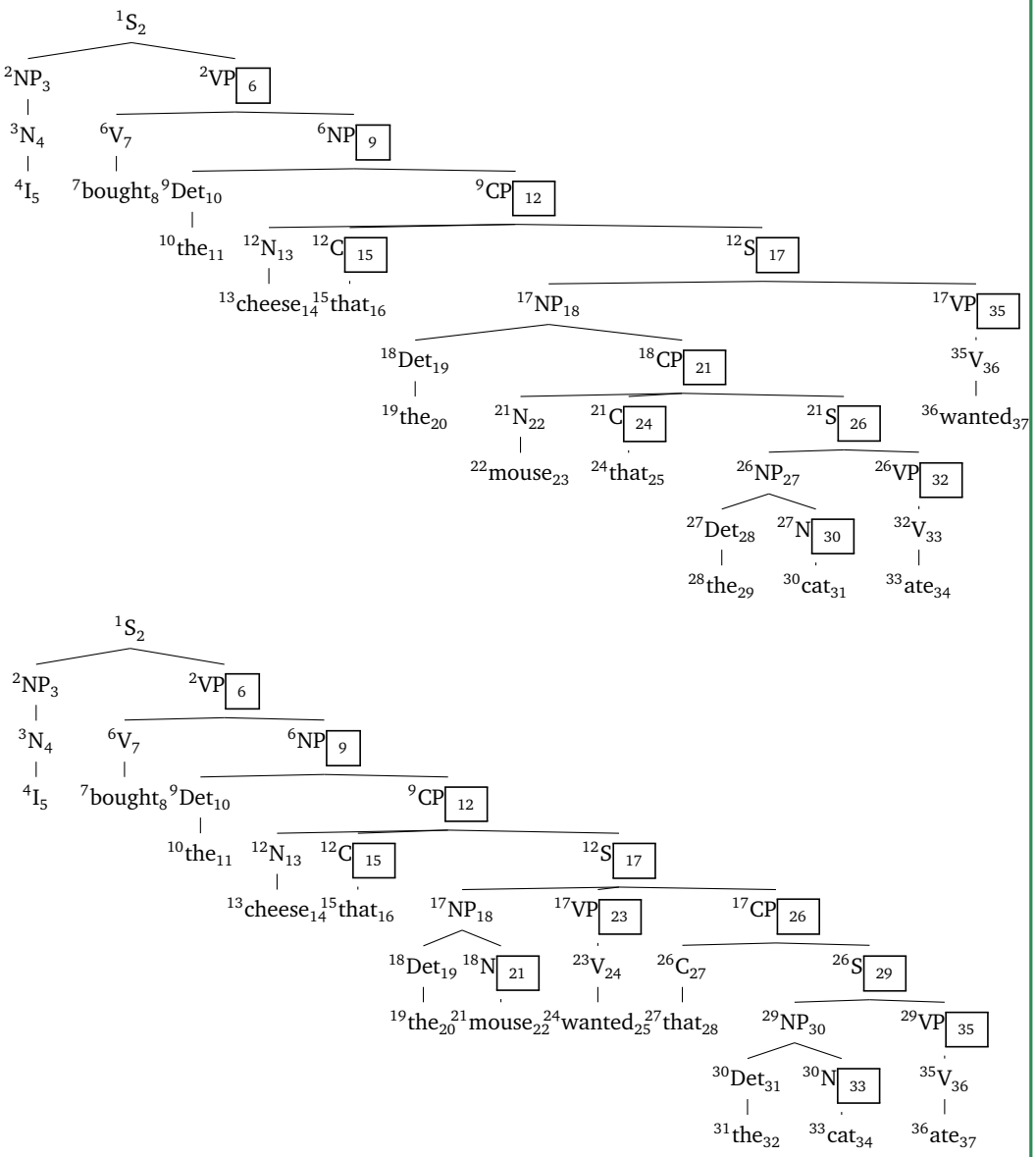
MaxTen greatest tenure among all the nodes; $\max(\{\text{tenure}(n)\})$

SumTen total number of steps that items must be memorized; $\sum(\{\text{tenure}(n) > 1\})$

All of these metrics capture the fact that right embedding isn't harder than center embedding. **MaxTen** and **SumTen** also predict right embedding to be easier.

Example 4.4 Center embedding and maximum tenure

Consider the center embedding and right embedding trees below, which have been annotated according to the behavior of a recursive descent parser. For center embedding we are using a promotion-style analysis of relative clauses (Vergnaud 1974; Kayne 1994), which posits that the relative clause is an argument of the determiner, with the head noun residing in a CP specifier. For right embedding the extraposed relative clause is considered a daughter of S.



The difficult center embedding sentence has a payload of 11, a maximum tenure of 17, and a sum tenure of 55. The easier right embedding sentence has a payload of 11, a maximum tenure of 9, and a sum tenure of 48. Irrespective of how we weigh or rank these three metrics, then, right embedding is never predicted to be harder than center embedding. As long as we do not take payload as the only difficulty metric, right embedding is correctly predicted to be easier than center embedding.

Notice that our explanation for the difficulty of center embedding is very different from the one we used for garden path effects. Garden path effects were explained in terms of the difficulty of finding a right parse among the many possible ones. We used prefix trees as a way of representing the parser's route through this search space, and since a prefix tree encodes many parsing tables at ones, our account is ultimately based

on the processing challenges of coordinating multiple parse tables — if all parse tables could be easily stored in memory and expanded in a breadth-first manner, garden path effects would not arise.

Center embedding, on the other hand, is explained purely in terms of how the parser constructs the correct parse. How the parser actually finds this parse does not factor into the explanation. The claim is that even if the parser had a perfect *oracle*, a machine that could tell it at every step which inference step must be taken to get the correct structure, the difference between center embedding and right embedding would not disappear because the former still puts a higher load on working memory than the latter.

4 Problems

4.1 Memory Usage in Left Embedding

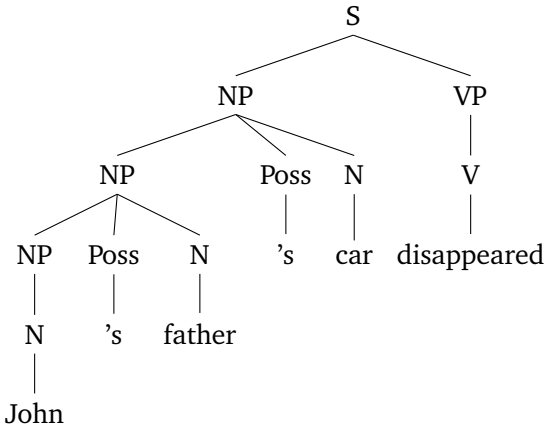
A careful examination of the top-down parsers behavior over center embedding constructions reveals that these configurations aren't the only ones that should cause an increased memory burden. Coupled with our metrics of processing difficulty, a recursive descent parser predicts that left embedding should be hard to parse, too — which is not borne out empirically.

(4) a. The exhaust pipe of the car of the father of the mechanic is broken.
b. The mechanic's father's car's exhaust pipe is broken.

The cause for the predicted difficulty spike with left embedding is straight-forward. Suppose the parser conjectures at step i that node n in the tree has daughters d_1 and d_2. Steps $i+1$ to j are spent by the parser expanding the subtree rooted in d_1. The bigger the subtree, the higher the value of $j+1$, the point at which the parser can move on to expanding d_2. Since d_2 must be kept in memory from step i until step $j+1$, and the value of j increases with the size of the subtree rooted in d_1, every construction that increases the size of a left sibling — including left embedding — should increase parsing difficulty.

4.2 Looping in Left Recursion

Left embedding constructions are problematic for top-down parsers in more respects than just psycholinguistic adequacy. Consider a grammar that licenses possessive structures such as the one below.



In order to arrive at this structure, the parser must first infer [0,•NP VP], and then expand the NP to get [0,•NP Poss N VP]. But now the parser once again has to expand the NP, possibly creating [0,•NP Poss N Poss N VP]. It is easy to see that the parser can keep rewriting NP *ad infinitum*, essentially looping the NP rewriting step and producing longer and longer parse items. Without a smart control structure that detects this kind of looping, the parser will soon run out of working memory, or if there is no limit on the amount of memory, keep looping forever without ever constructing a parse.

A standard solution is to use a probabilistic control structure that prunes away all parse tables that fall below a certain probability threshold. As the parser keeps conjecturing bigger and bigger structures, the probability of these parses decreases until they are eventually pruned away. This is also called a *beam parser*. Beam parsers require a method for automatically inferring the necessary probabilities from a corpus, and their overall behavior can be difficult to figure out. So the appealing simplicity of top-down parsers is lost to some extent. For more details on top-down beam parsing, see Roark (2001a,b, 2004).

4.3 Merely Local Syntactic Coherence Effects

Merely local syntactic coherence effects is a term coined by Tabor et al. (2004) to refer to cases where an analysis that is locally well-formed but incompatible with the structure built so far nonetheless induces a temporary increase in parsing difficulty (see also Konieczny 2005, Konieczny et al. 2009, and Bicknell et al. 2009). This is exemplified by the contrast in (5). During self-paced reading experiments, subjects' reading speed of (5a) decreases at *tossed*, indicating an increase in processing difficulty. The effect disappears, however, when *tossed* is replaced by *thrown*.

(5) a. The coach smiled at the player tossed a frisbee.
b. The coach smiled at the player thrown a frisbee.

It seems as if the fact that *the player* can be locally analyzed as the subject of *tossed* confuses the parser. Since this interpretation is not available with *thrown* due to the unambiguous past participle morphology, no slowdown occurs.

Merely local syntactic coherence effects are completely unexpected with a top-down parser because there is no way the parser could infer an item that treats *tossed* as a finite verb with *the player* as its subject. The preceding parts of the sentence have already disambiguated the structure of the sentence to an extent where this is no longer a viable hypothesis. These effects are expected, however, if the parser proceeds bottom-up instead of top-down. More on that in chapter 5.

References and Further Reading

Bicknell, Klinton, Roger Levy, and Vera Demberg. 2009. Correcting the incorrect: Local coherence effects modeled with prior belief update. In *Proceedings of the 35th Annual Meeting of the Berkeley Linguistics Society*, 13–24.

Frazier, Lyn. 1979. *On comprehending sentences: Syntactic parsing strategies*. Doctoral Dissertation, University of Connecticut.

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

Gibson, Edward. 1998. Linguistic complexity: Locality of syntactic dependencies. *Cognition* 68:1–76.

Graf, Thomas, Brigitta Fodor, James Monette, Gianpaul Rachiele, Aunika Warren, and Chong Zhang. 2015. A refined notion of memory usage for minimalist parsing. In *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*, 1–14. Chicago, USA: Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W15-2301.

Graf, Thomas, and Bradley Marcinek. 2014. Evaluating evaluation metrics for minimalist parsing. In *Proceedings of the 2014 ACL Workshop on Cognitive Modeling and Computational Linguistics*, 28–36.

Kayne, Richard S. 1994. *The antisymmetry of syntax*. Cambridge, MA: MIT Press.

Kimball, John. 1973. Seven principles of surface structure parsing in natural language. *Cognition* 2:15–47.

Kobele, Gregory M., Sabrina Gerth, and John T. Hale. 2013. Memory resource allocation in top-down Minimalist parsing. In *Formal Grammar: 17th and 18th International Conferences, FG 2012, Opole, Poland, August 2012, Revised Selected Papers, FG 2013, Düsseldorf, Germany, August 2013*, ed. Glyn Morrill and Mark-Jan Nederhof, 32–51. Berlin, Heidelberg: Springer. URL https://doi.org/10.1007/978-3-642-39998-5_3.

Konieczny, Lars. 2005. The psychological reality of local coherences in sentence processing. In *Proceedings of the 27th Annual Conference of the Cognitive Science Society*.

Konieczny, Lars, Daniel Müller, Wibke Hachmann, Sarah Schwarzkopf, and Sascha Wolfer. 2009. Local syntactic coherence interpretation. Evidence from a visual world study. In *Proceedings of the 31st Annual Conference of the Cognitive Science Society*, 1133–1138.

Roark, Brian. 2001a. Probabilistic top-down parsing and language modeling. *Computational Linguistics* 27:249–276.

Roark, Brian. 2001b. *Robust probabilistic predictive syntactic processing: Motivations, models, and applications*. Doctoral Dissertation, Brown University.

Roark, Brian. 2004. Robust garden path parsing. *Natural Language Engineering* 10:1–24.

Tabor, Whitney, Bruno Galantucci, and Daniel Richardson. 2004. Effects of merely local syntactic coherence on sentence processing. *Journal of Memory and Language* 50:355–370.

Vergnaud, Jean-Roger. 1974. *French relative clauses*. Doctoral Dissertation, MIT.