Lecture 5

Bottom-Up Parsing

The natural counterpart to top-down parsing is bottom-up parsing, where trees are built starting at the leaves and moving towards the root. Like top-down parsing, bottom-up parsing is fairly simple and acts as the foundation of a variety of parsing algorithms. But just like top-down parsing, it has certain shortcomings regarding psycholinguistic adequacy.

1 Intuition

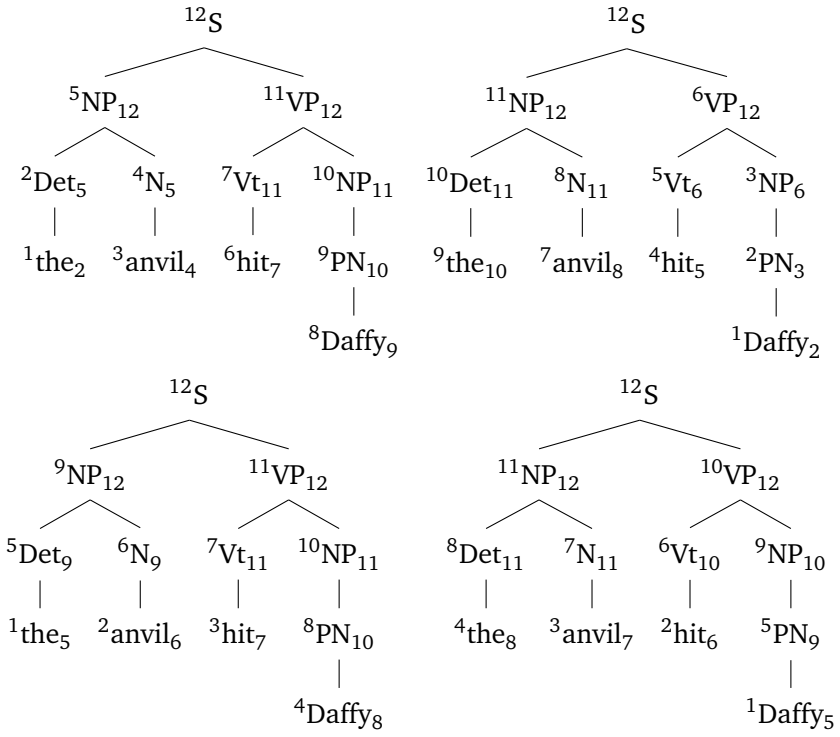
For illustration we use the same grammar as for the top-down parser in Lecture 3.

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

A bottom-up parser essentially applies the rewrite rules in reverse. If the current input i appears on the right-hand side of a rewrite rule for N, replace i by N.

string	**rule**
the	read input
det	Det →the
det anvil	read input
det N	N →anvil
NP	NP →Det N
NP hit	read input
NP Vt	Vt →hit
NP Vt Daffy	read input
NP Vt PN	PN →Daffy
NP Vt NP	NP →PN
NP VP	VP →Vt NP
S	S →NP VP

The order in which rules are applied once again gives rise to at least four different types of parsers, a helpful first approximation of which can be gleaned from the examples below. The four trees below correspond to (in clockwise order) left-to-right depth first, right-to-left depth first, right-to-left breadth first, and left-to-right breadth first.



2 Formal Specification

2.1 Parsing Schema

Since bottom-up parsers are essentially the dual of top-down parsers, the former's deductive definition closely resembles that of the latter. Whereas a top-down parser starts with $[0, S, n]$ and seeks to derive $[n, , n]$, the bottom-up parser has axiom $[0, , 0]$ and goal $[0, S, n]$. So axioms and goals are simply switched (and $[n, , n]$ is replaced by $[0, , 0]$). Similarly, the top-down scan and predict rules have bottom-up counterparts Shift and Reduce. The reduce rule is exactly the predict rule of a top-down parser with top and bottom switched. The shift rule is the scan rule of a right-to-left top-down parser with top and bottom switched.

$$\textbf{Shift} \qquad \frac{[i, \beta, j]}{[i, \beta a, j+1]}\, a = w_j$$

$$\textbf{Reduce} \qquad \frac{[i, \alpha\gamma\beta, j]}{[i, \alpha N\beta, j]}\, N \to \gamma \in R$$

Example 5.1 Bottom-up parse of *The anvil hit Daffy*

Here's a bottom-up parse table for our standard example sentence *The anvil hit Daffy* using the parsing schema above. Note that rules are applied in arbitrary order to reflect the fact that the parsing schema still lacks a control structure and thus imposes no specific rule order.

parse item	inference rule
[0,,0]	axiom
[0,the,1]	shift
[0,the anvil,2]	shift
[0,the N,2]	reduce(7)
[0,Det N,2]	reduce(6)
[0,Det N hit,3]	shift
[0,NP hit,3]	reduce(3)
[0,NP hit Daffy,4]	shift
[0,NP hit PN,4]	reduce(8)
[0,NP Vt PN,4]	reduce(10)
[0,NP Vt NP,4]	reduce(2)
[0,NP VP,4]	reduce(5)
[0,S,4]	reduce(1)

2.2 Control Structure

The control structure can be (partially) encoded by adding the familiar • to the rules.

Left-to-right, depth-first The only axiom is $[0, \bullet, 0]$, and the only goal is $[0, S\bullet, n]$.

$$\textbf{Shift} \qquad \frac{[i, \beta\bullet, j]}{[i, \beta a\bullet, j+1]}\, a = w_j$$

$$\textbf{Reduce} \qquad \frac{[i, \alpha\gamma\bullet, j]}{[i, \alpha N\bullet, j]}\, N \rightarrow \gamma \in R$$

This kind of parser is also called a *shift reduce parser*. Notice that even though the parser reads the input from left-to-right, the structure building process is partially right-to-left since the reduction rule reduces elements to the left of • from right to left.

Left-to-right, breadth first It is surprisingly difficult to specifier a breadth first bottom-up parser in a deductive fashion (more on that below), and consequently the rules are a lot more complicated. The axioms and goals are the same as for the depth-first parser, though.

$$\textbf{Shift} \qquad \frac{[i, \beta\bullet, j]}{[i, \beta a\bullet, j+1]}\, a = w_j$$

$$\textbf{Reduce} \qquad \frac{[i, \alpha \bullet \beta\gamma\delta, j]}{[i, \alpha\beta N \bullet \delta, j]}\, N \rightarrow \gamma \in R$$

$$\textbf{Return} \qquad \frac{[i, \alpha \bullet \beta, j]}{[i, \bullet\alpha\beta, j]}\, \alpha \in (N \cup T)^+,\ \neg\exists N[N \rightarrow \beta \in R]$$

Example 5.2 Depth-first parse of *The anvil hit Daffy*

parse item	inference rule
[0,•,0]	axiom
[0,the •,1]	shift
[0,Det •,1]	reduce(6)
[0,Det anvil •,2]	shift
[0,Det N •,2]	reduce(7)
[0,NP •,2]	reduce(3)
[0,NP hit •,3]	shift
[0,NP Vt •,3]	reduce(10)
[0,NP Vt Daffy •,4]	shift
[0,NP Vt PN •,4]	reduce(8)
[0,NP Vt NP •,4]	reduce(2)
[0,NP VP •,4]	reduce(5)
[0,S •,4]	reduce(1)

Example 5.3 Breadth-first parse of *The anvil hit Daffy*

parse item	inference rule
[0,•,0]	axiom
[0,the •,1]	shift
[0,the anvil •,2]	shift
[0,the anvil hit •,3]	shift
[0,the anvil hit Daffy •,4]	shift
[0,•the anvil hit Daffy,4]	return
[0,Det •anvil hit Daffy,4]	reduce(6)
[0,Det N •hit Daffy,4]	reduce(7)
[0,Det N Vt •Daffy,4]	reduce(10)
[0,Det N Vt PN •,4]	reduce(8)
[0,•Det N Vt PN,4]	return
[0,NP •Vt PN,4]	reduce(3)
[0,NP Vt NP •,4]	reduce(2)
[0,•NP Vt NP,4]	return
[0,NP VP •,4]	reduce(5)
[0,•NP VP,4]	return
[0,S •,4]	reduce(1)

Exercise 5.1. Mirroring the redundancy of the index j for top-down parsers, index i can safely be eliminated from the parse items of a bottom-up parser. Explain why. ⊙

Exercise 5.2. Our breadth-first parser is actually too permissive. Show that the rules as given can also be used to construct a depth-first parse of *The anvil hit Daffy*. ⊙

Exercise 5.3. Is there a way to restrict the breadth-first parser so that it can no longer construct depth-first parses? ⊙

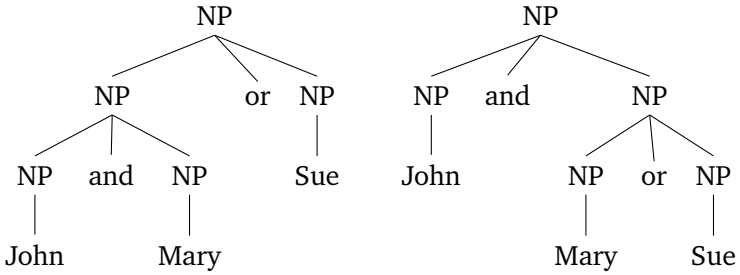
Notice that both parser have a certain overlap between the rules. In the case of the depth-first parser, the domains of the shift and reduce rules are not disjoint. That is to say, for some parse items both shift and reduce are valid continuations of the parse, which is called a *shift reduce conflict*. Shift reduce conflicts are an unavoidable consequence of non-determinism in the grammar. A top-down parser faces non-determinism with respect to which rewrite rule to apply to a given non-terminal N. For a bottom-up parser, this step is usually deterministic because distinct non-terminals can be assumed to also have distinct right-hand sides in rewrite rules (rules assigning parts of speech to words are one notable exception). But the bottom-up parser must decide whether to reduce right away or read another input symbol first, which might allow for a different reduction step.

Example 5.4 A depth-first parse with delayed shift

Consider the grammar below, which generates conjunctions and disjunctions of proper names.

1) NP → NP and NP
2) NP → NP or NP
3) NP → John | Mary | Sue

Now consider the phrase *John and Mary or Sue*, which has two semantically distinct structures.



If the bottom-up parser reduces as early as possible, we get the tree to the left. This also shows that reduction proceeds from the right edge of the parse item.

parse item	**inference rule**
[0,•,0]	axiom
[0,John •,1]	shift
[0,NP •,1]	reduce(3)
[0,NP and •,2]	shift
[0,NP and Mary •,3]	shift
[0,NP and NP •,3]	reduce(3)
[0,NP •,3]	reduce(1)
[0,NP or •,4]	shift
[0,NP or Sue •,5]	shift
[0,NP or NP •,5]	reduce(3)
[0,NP •,5]	reduce(2)

In order to obtain the other tree, the parser must delay the application of reduce(1)

until reduce(2).

parse item	inference rule
[0,•,0]	axiom
[0,John •,1]	shift
[0,NP •,1]	reduce(3)
[0,NP and •,2]	shift
[0,NP and Mary •,3]	shift
[0,NP and NP •,3]	reduce(3)
[0,NP and NP or •,4]	shift
[0,NP and NP or Sue •,5]	shift
[0,NP and NP or NP •,5]	reduce(3)
[0,NP and NP •,5]	reduce(2)
[0,NP •,5]	reduce(1)

The breadth-first parser has a comparable overlap between shift and return. Once again this corresponds to the distinction between waiting for more input or building structure on top of the input read so far. The focal point of non-determinism, however, lies in the definition of the reduction rule, which allows the parser to non-deterministically partition the string to the right of • into three segments β, γ and δ to reduce β. This rule is indispensable in cases where the symbol immediately after • cannot be reduced, or where reduction would be possible but would prevent the parser from assigning an alternative structure.

Example 5.5 A breadth-first parse with structural ambiguity

Consider now the two available breadth-first parses for the NP *John and Mary or Sue*. The first few steps are the same.

parse item	inference rule
[0,•,0]	axiom
[0,John •,1]	shift
[0,John and •,2]	shift
[0,John and Mary •,3]	shift
[0,John and Mary or •,4]	shift
[0,•John and Mary or Sue,5]	return
[0,NP •and Mary or Sue,5]	reduce(3)
[0,NP and NP •or Sue,5]	reduce(3)
[0,NP and NP or NP •,5]	reduce(3)
[0,•NP and NP or NP,5]	return

The structural difference now depends on which rule the parser applies first, reduce(1) or reduce(2). Keep in mind that both are valid. For reduce(1), we have $\beta = \varepsilon$, $\gamma =$ NP and NP, and $\delta =$ or NP. For reduce(2), we have $\beta =$ NP and, $\gamma =$ NP or NP, and $\delta = \varepsilon$. After each rule, the parser has to use the return rule to move the dot into a position from where it can apply the other reduce rule.

parse item	inference rule
[0,NP •or NP,5]	reduce(1)
[0,•NP or NP,5]	return
[0,NP,5]	reduce(2)

parse item	inference rule
[0,NP and NP •,5]	reduce(2)
[0,•NP and NP,5]	return
[0,NP,5]	reduce(1)

3 Psycholinguistic Adequacy of Shift Reduce Parser

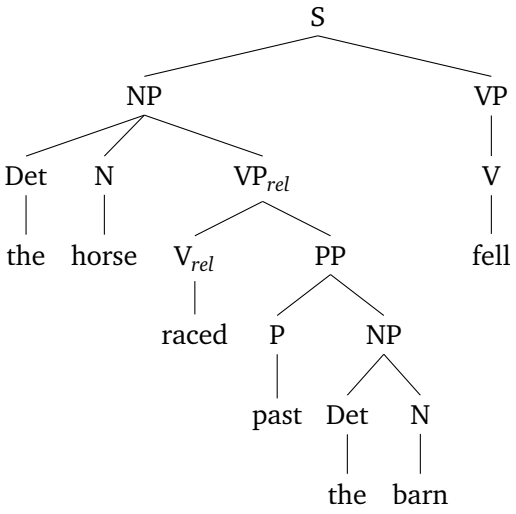
3.1 Garden Paths

If the parser prioritizes reduction over shifting (which is up to the control structure), a serial shift reduce parser with backtracking makes similar predictions to a recursive descent parser. Garden path effects cannot arise if shifting is prioritized to such an extent that *fell* is read in by the parser before it attempts to reduce *raced* to a V.

Example 5.6 Shift-reduce parse for *The horse raced past the barn fell*

We operate with the same grammar as in 2.3.

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



The parse history is given in Fig. 5.1.

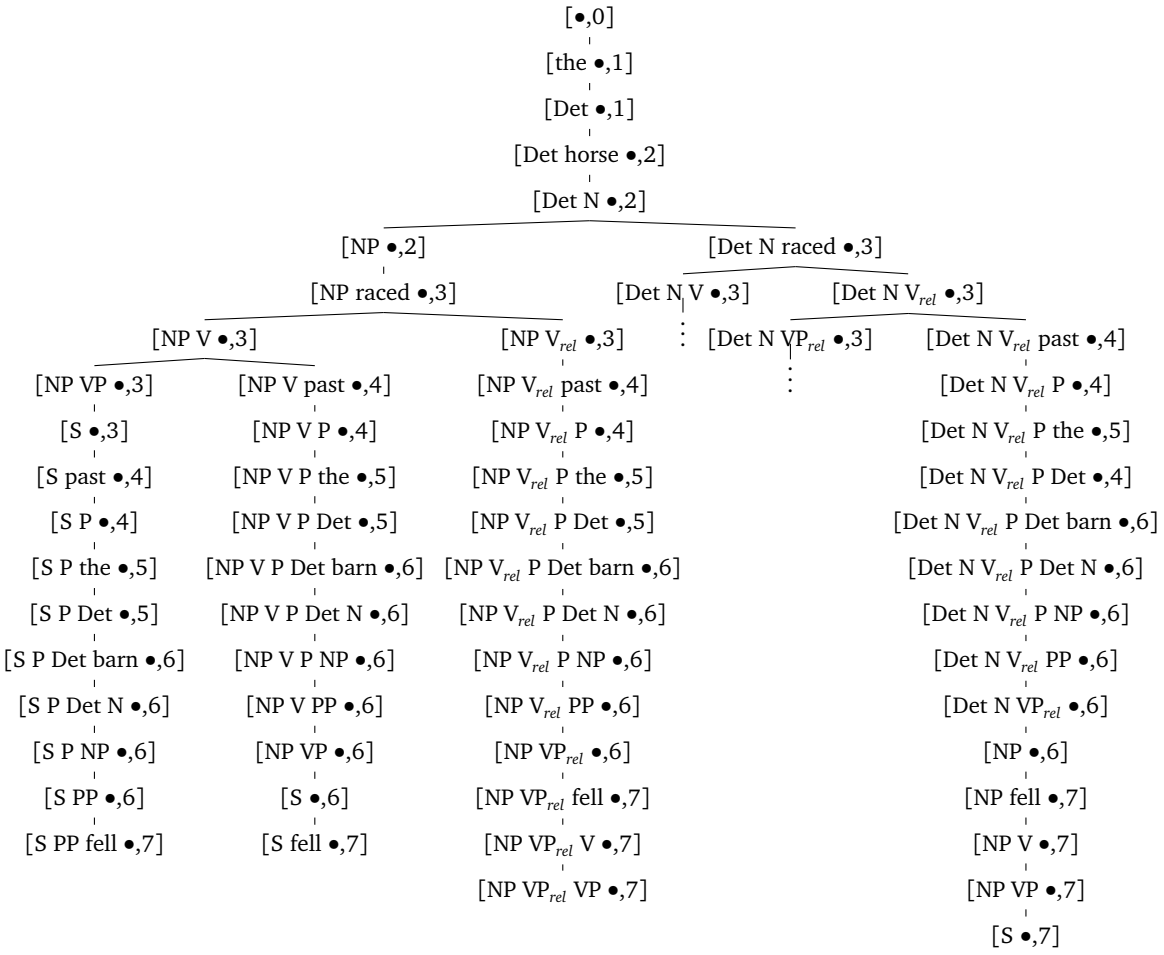


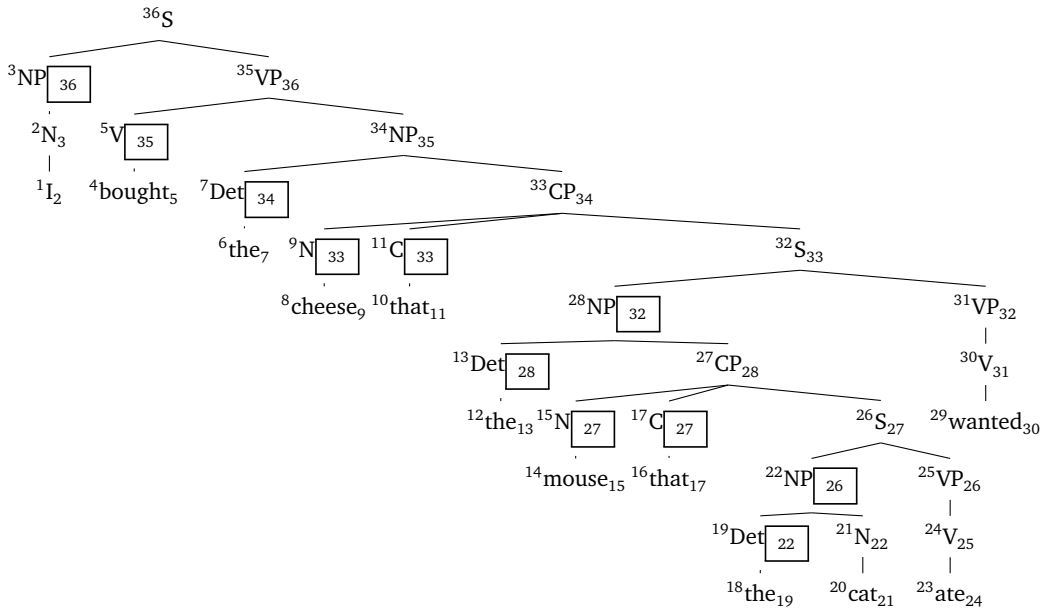
Figure 5.1: Parse history for serial shift reduce parse of *the horse raced past the barn fell*; the parser moves through the history in a recursive descent fashion

3.2 Embeddings

Just like the top-down parser, the bottom-up parser predicts center embedding constructions to be fairly difficult.

Example 5.7 Run of shift reduce parser over center embedding sentence

We use the same promotion-style analysis of relative clauses as in Sec. 3.



The parse shows a payload of 11, just as in the case of the recursive descent parser. The maximum tenure is 33, which is a lot more than the recursive descent parser's 17. The sum tenure is 184 (over three times the recursive descent parser's 55).

In contrast to the top-down parser, however, the bottom-up parser doesn't fare any better with the right embedding construction. So just like the top-down parser predicts center embedding to be difficult simply because center embedding involves left embedding dependencies and those are problematic for top-down parsers, the bottom-up parser apparently struggles with center embedding because it also struggles with right embedding. The reason is simple:

1. Reduction of A and B to C is possible only if both A and B have already been recognized.

2. The shift reduce parser fully assembles A before getting started on B. Hence the time A must be stored in memory is directly proportional to the size of B.

3. Right embedding increases the size of B.

Metric	Center	Right
Payload	11	11
MaxTen	17	9
SumTen	55	48

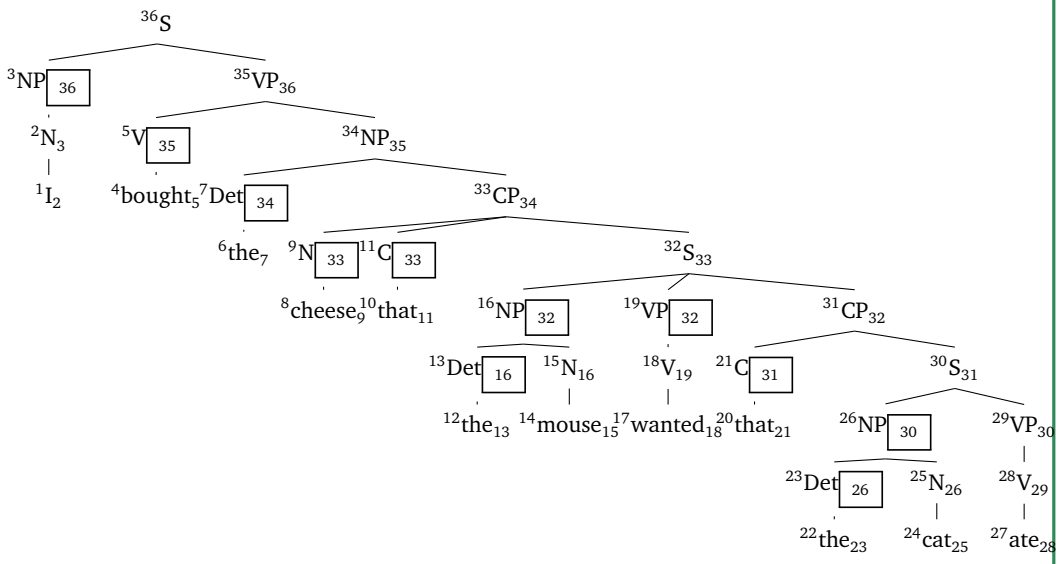
(a) Recursive Descent

Metric	Center	Right
Payload	11	11
MaxTen	33	33
SumTen	184	185

(b) Shift Reduce

Table 5.1: Overview of parser performance for right and center embedding constructions

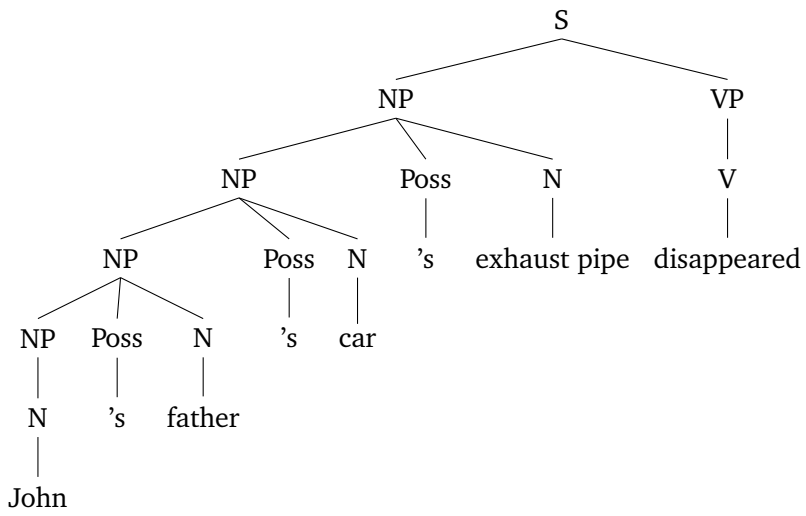
Example 5.8 Run of shift reduce parser over right embedding sentence



The payload is 11, exactly the same as for the center embedding sentence. Maximum tenure is also unchanged at 33. Summed tenure even increased from 184 to 185.

Table 5.1 compares the performance of the recursive descent parser and the shift reduce parser over the two examples sentences.

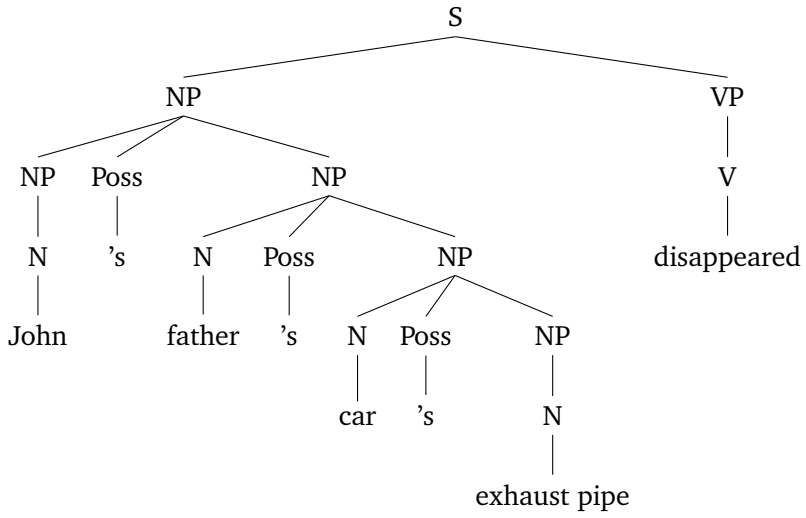
Exercise 5.4. Assume that the sentence *John's father's car's exhaust pipe disappeared* has the structure below.



⊙

Annotate the tree with indices according to the order in which it would be built by I) a recursive descent parser and II) a shift reduce parser. For each parser, calculate payload, maximum tenure and summed tenure. Do you see a difference between the two parsers regarding these difficulty metrics? If so, explain in intuitive terms what causes the performance gap.

Exercise 5.5. Repeat the previous exercise, but now assume the following structure instead:



⊙

Exercise 5.6. Last time we saw that left-to-right top-down parsers cannot account for merely local syntactic coherence effects, irrespective of whether one proceeds depth-first or breadth-first. Can a bottom-up parser account for this phenomenon? Does it have to use a specific search method (depth-first VS breadth-first)? Do shift and reduce need to be prioritized in a specific fashion? ⊙

3.3 General Remarks

A bottom-up parser that always prefers shift isn't truly incremental and thus a bad model of human sentence processing. A shift reduce parser, on the other hand, is

incremental but not *predictive*. While a specific sequence of shift and reduce steps can block certain analysis (see the coordination example above), a bottom-up parser does not actively restrict its hypothesis. Given an input string $\alpha\beta$ where α has been fully analyzed — i.e. the parser has assigned a single connected subtree to α — the conjectured structure of α has no effect on which structures the parser may entertain for β. In particular, the parser will entertain analyses of β that are incompatible with its analysis of α.

This noncommittal attitude does not seem to be shared by the human parser. For one thing, α can prime the parser towards certain structures. Consider the following contrast.

(1) a. The grave robber buried in the sand all the treasures he had stolen.
b. The mine buried in the sand exploded.

Even though *buried* can be a finite verb as well as a participle, the former is strongly preferred in example (1a). In (1b), on the other hand, the participle interpretation is favored.

Similarly, processing can be shown in self-paced reading experiments to slow down in ungrammatical sentences as soon as it becomes evident that the sentence cannot be salvaged anymore.

(2) * The grave robber in the sand all the treasures he had stolen.

This sentence can be recognized as ungrammatical once *all* is encountered. This inference is not made by the shift-reduce parser, however, which will continue to parse this sentence until all symbols have been read and all possible reductions have been carried out. A top-down parser, on the other hand, will always crash at *all* since none of its conjectured structures are compatible with *all* at this position in the sentence.

Exercise 5.7. Draw the parse history (i.e. the prefix tree of parse tables) for the unsuccessful shift-reduce parse of the sentence *the grave robber in the sand all the treasures*, given the grammar below.

1)	S	→	NP VP	6)	Det	→	the | all
2)	NP	→	N	7)	N	→	grave | grave robber | sand | treasures
3)	NP	→	Det N	8)	P	→	in
4)	NP	→	NP PP	9)	V	→	stole | sand
5)	VP	→	V				

How many steps does the parser spend on parses that a more predictive parser could have already identified as unsalvageable? ⊙

Lecture 6

Left-Corner Parsing

Top-down parsers and bottom-up parsers each turned out to have their advantages as well as their disadvantages. Top-down parsers are purely predictive. The input string is only checked against fully built branches — those that end in a terminal symbol — but does not guide the prediction process itself. Bottom-up parsers are purely driven by the input string and lack any kind of predictiveness. In particular, a bottom-up parser may entertain analyses for the substring spanning from position i to j that are incompatible with the analysis for the substring from 0 to $i-1$. Neither behavior seems to be followed by the human parser all the time.

Merely local syntactic coherence effects suggest that the human parser sometimes entertains incompatible parses, just like bottom-up parsers. But these effects are very rare and very minor compared to, say, the obvious difficulties with garden path sentences. The human parser is also predictive since ungrammatical sentences are recognized as such as soon as the structure becomes unsalvageable. At the same time, though, the prediction process differs (at least naively) from pure top-down parsing as it seems to be actively guided by the input. What we should look at, then, is a formal parsing model that integrates top-down prediction and bottom-up reduction. Left-corner parsing does exactly that.

1 Intuition

The ingenious idea of left-corner (LC) parsing is to restrict the top-down prediction step such that the parser conjectures X only if there is already some bottom-up evidence for the existence of X. More precisely, the parser conjectures an XP only if a *possible left corner of X* has already been identified. The *left corner of a rewrite rule* is the leftmost symbol on the righthand side of the rewrite arrow. For instance, the left corner of NP → Det N is Det. Thus Y is a possible left corner of X only if the grammar contains a rewrite rule $X \rightarrow Y\ \gamma$. In this case, the parser may conjecture the existence of X and γ once it has reached Y in a bottom-up fashion.

Consider our familiar toy grammar.

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