Lecture 3

Top-Down Parsing

Top-down parsing is arguably the simplest parsing model. In fact, it is so intuitive that syntax students, for example, have a tendency to automatically use this algorithm when asked to determine if a grammar generates a given sentence. That's because top-down parsing is very close to the idea of phrase structure grammars as top-down generators.

1 Intuition

Suppose you are given the following CFG.

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

When asked to show that this grammar generates the sentence *The anvil hit Daffy*, you might draw a tree. But a different method is to provide a tabular depiction of the rewrite process.

string	**rule**
S	start
NP VP	S → NP VP
Det N VP	NP → Det N
the N VP	Det → the
the anvil VP	N → anvil
the anvil Vt NP	VP → Vt NP
the anvil hit NP	Vt → hit
the anvil hit PN	NP → PN
the anvil hit Daffy	PN →Daffy

Of course the rewrite rules could also be applied in other orders.

string	rule
S	start
NP VP	S → NP VP
NP Vt NP	VP → Vt NP
NP Vt PN	NP → PN
NP Vt Daffy	PN →Daffy
NP hit Daffy	Vt → hit
Det N hit Daffy	NP → Det N
Det anvil hit Daffy	N → anvil
the anvil hit Daffy	Det → the

string	rule
S	start
NP VP	S → NP VP
Det N VP	NP → Det N
Det N Vt NP	VP → Vt NP
the N Vt NP	Det → the
the anvil Vt NP	N → anvil
the anvil hit NP	Vt → hit
the anvil hit PN	NP → PN
the anvil hit Daffy	PN →Daffy

In all three cases we proceed top-down: non-terminals are replaced by a string of terminals and/or non-terminals. From the perspective of phrase structure trees, the trees are growing from the root towards the leafs. The difference between the three tables is the order in which non-terminals are rewritten.

- Table 1: depth-first, left-to-right
- Table 2: depth-first, right-to-left
- Table 3: breadth-first, left-to-right

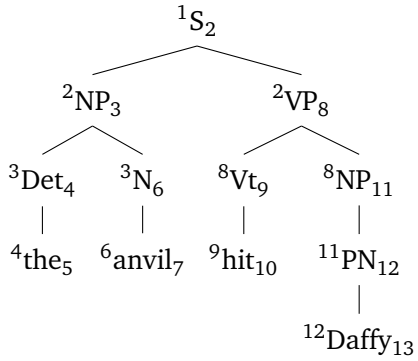
depth-first rewrite some symbol that was introduced during the previous rewriting step

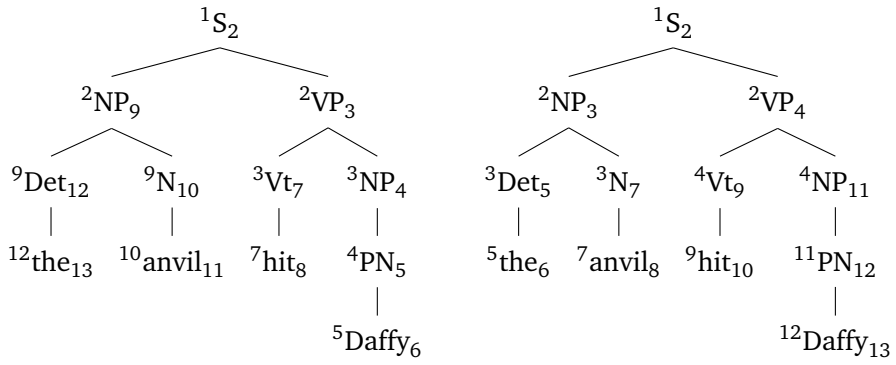
breadth-first before rewriting a symbol introduced during rewriting step j, all symbols that were introduced at rewriting step i must have been rewritten, for every $i < j$

left-to-right if several symbols are eligible to be rewritten, rewrite the leftmost one

right-to-left if several symbols are eligible to be rewritten, rewrite the rightmost one

We can visualize the differences between these strategies by annotating phrase structure trees with indices to indicate when a symbol is first introduced (prefix) and when it is rewritten (suffix). For terminal symbols, which are never rewritten, we stipulate that the suffix is one higher than the prefix.





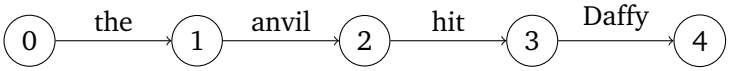
Exercise 3.1. Draw the table for a breadth-first, right-to-left parser, and the corresponding annotated phrase structure tree. ⊙

Exercise 3.2. Consider the sentence *Bugs fell over*. Draw the tables for a left-to-right depth-first strategy and left-to-right breadth-first strategy, and draw the corresponding annotated phrase structure trees. Do the two strategies differ at all over such a short sentence? If so, how, and at which nodes? ⊙

2 Formal Specification

2.1 Sentences as Indexed Strings

The input to the parser is just a sequence of words, i.e. a string. We adopt the standard convention for numbering positions in a string, i.e. the i-th word in the string occurs between positions $i-1$ and i.



Given a string w that consists of n words, each word can be referenced by the number to its left. If $w = \textit{the anvil hit Daffy}$, $w_0 = \textit{the}$ and $w_3 = \textit{Daffy}$, for instance. Notice that if the grammar allows for empty heads, an input sentence can be mapped to infinitely many such strings depending on where one posits empty heads.

2.2 Parsing Schema

The parsing schema can be easily stated in terms of logical inference rules. This idea was originally called *parsing as deduction* (Pereira and Warren 1983; Shieber et al. 1995) and was later refined by Sikkel (1997). Parsing as deduction makes it very easy to see what distinguishes different parsing schema and put various control structures on top of them. It's also the foundation for *semiring parsing* (Goodman 1999), which provides an abstract perspective that unifies recognizers and parsers, among other things.

Given a CFG $G := \langle N, T, S, R\rangle$ and input string $w = w_1 \cdots w_n$, our *items* take the form $[i, \beta, j]$, where

- $\beta \in (N \cup T)^*$ is a string of terminal and/or non-terminal symbols, and
- i and j are positions in the string such that $0 \leq i < j \leq n+1$.

An item encodes the conjecture that the substring spanning from i to j can be generated from β via G. Our only *initial item* (also called *axiom*) is $[0, S, n]$, which denotes that the grammar generates the string spanning from 0 to n. As our *final item* (also called *goal*) we require $[n, , n]$.

The parser uses two inference rules; one for scanning, the other one for top-down prediction.

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

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

The scanning rule tells us that the starting position of an item can be moved one to the right if the item starts with a terminal symbol that matches the word in the input string at the corresponding position. The predict rule produces a new item from an old by rewriting one the one non-terminal symbols using a production of G.

Strictly speaking the parsing schema does not give us the actual system of axioms and inference rules. Rather, these are obtained once the variables in the parsing schema are instantiated according to the rewrite grammar. For example, the predict rule would the instantations below (and several more) given the grammar presented earlier.

$$\frac{[i, \text{S}, j]}{[i, \text{NP VP}, j]}$$

$$\frac{[i, \text{NP VP}, j]}{[i, \text{Det N VP}, j]}$$

$$\frac{[i, \text{NP VP}, j]}{[i, \text{aN VP}, j]}$$

The instantiation of a parsing schema according to a grammar is called a *parsing system*. In contrast to the parsing schema, a parsing system may have an infinite number of rules. That's not the case here because our grammar does not have recursion and thus can only create a finite number of sentences.

Exercise 3.3. (Psycho-)Linguists do not distinguish between parsing schema and parsing system (nor do they factor out control structure and data structure). Is their notion of parser closer to a parsing schema or a parsing system? Or do they unwittingly switch between the two depending on context? ⊙

Example 3.1 Top-down parse of *The anvil hit Daffy*

Let G be the grammar at the beginning of the handout. Then a depth-first, left-to-right parse of *The anvil hit Daffy* will proceed as follows, where predict(n) denotes a prediction step using rule n.

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

Note that the table above only shows the prediction steps leading to a successful parse. The parser, on the other hand, could in principle apply every possible prediction step (depending on its control structure). So from [0,NP VP,4], for instance, the parser may not only predict [0,Det N VP,4] via rule 3 but also [0,PN VP,4] via rule 2 since both can be applied to NP.

As was discussed last time, the parsing schema only describes the starting point of the parser, the desired outcome, and which inference steps the parser may take to get from the former to the latter. It does not specify in which order these steps should be taken, nor how the parser handles multiple parses in parallel. This is left to the control structure and the data structures.

2.3 Control Structure

We can add a control structure to the parser to describe both the parsers directionality (left-to-right VS right-to-left) and its search method (depth-first VS breadth-first). There's many ways of specifying the control structure, but one simple option is to encode it in the parse schema. More precisely, we add a dot • to the items to highlight which symbol should be expanded. The different strategies then correspond to different ways of moving the dot through the parse items.

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

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

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

This type of top-down parser is also called a *recursive descent parser*.

Left-to-right, breadth-first As before the axiom and goal are $[0, \bullet S, n]$ and $[n, \bullet, n]$, respectively.

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

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

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

Example 3.2 Breadth-first parse of *The anvil hit Daffy*

Once again we assume that G is the grammar at the beginning of the handout.

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

And as before the parser actually predicts a lot more items, the table only lists those that are used in the successful parse.

Exercise 3.4. Specify the right-to-left versions of the parsers above. ⊙

Exercise 3.5. Parse the sentence *The truck fell over* will all four different types of top-down parsers. Use tables such as the ones in the previous two examples. ⊙

Exercise 3.6. The left-to-right breadth-first parser differs slightly from the intuitive one we saw in Sec. 1. In what respect? And which one of the two versions would be more efficient for syntactic processing? (Hint: draw a phrase structure tree as before and annotate the nodes according to when they are introduced by the parser, and when they are scanned or rewritten. Then compare this tree to the one in Sec. 1.) ⊙

Exercise 3.7. The parse items for all the parsers above include both the beginning and the end of the substring they span. This seems a bad fit for incremental processing, where the length of the input string isn't known in advance. Can we remove the index of the end position from the parse items and still have a functional parser? What changes must be made to the inference rules, axioms, and goals? ⊙

2.4 Data Structure

There's a myriad of different data structures a top-down parser may employ. We won't discuss them in great detail until later, but let's quickly summarize what the parser needs to be able to do.

- **Store history of successful parses**
  Recall that a parser differs from a recognizer in that the latter only groups sentences into grammatical and ungrammatical whereas the former also assigns tree structures to well-formed sentences. This structure can be easily inferred from the parse history, but this means that the parser needs to keep track of this history.

- **Dealing with non-determinism**
  We have implicitly assumed in our examples that the parser always makes the right decision with respect to which rewrite rule should be applied to which non-terminal symbol. But this is of course not the case in practice. Instead, the parser needs a way to deal with multiple parses, which means being able to store multiple parse histories.

A single parse history can be stored as a table, for example. Multiple parses, then, are a table of such parse history tables.

Exercise 3.8. Formulate a strategy for converting a given parse history, stored as a table, into the corresponding phrase structure tree. ⊙

Smarter data structures can greatly improve the parser's efficiency. For instance, identical items can be shared across parse histories. That way, if the parser currently has three histories containing the item [4,•NP,20], the possible inferences from this item only need to be computed once rather than three times, which would be a waste of resources. Similarly, some of the inferences from this item can also be reused for a parse with item [2, •NP, 18]. These techniques for sharing structure and memorizing the results of computation so that they can be reused somewhere else later on fall under the umbrella of *dynamic programming* and *tabulation*. These are very powerful ideas, but their addition does not alter the parsing schema or the control structure.

The control structure also needs to include some strategy for how distinct parses should be prioritized. For instance, one may finish one parse before moving on to the next, or interleave them. Often parses are put in a *priority queue*. At each step, the first parse in the queue is opened and one inference step is applied. Depending on the outcome of this inference, the parse may be removed from the top spot and inserted somewhere else in the queue. Similarly, the inference step may have created new parse tables that also need to be inserted somewhere in the queue. As we will see next time, the choice of control structure plays a great role for what psycholinguistic predictions are being made.

References and Further Reading

Goodman, Joshua. 1999. Semiring parsing. *Computational Linguistics* 25:573–605.

Pereira, Fernando C. N., and David Warren. 1983. Parsing as deduction. In *21st Annual Meeting of the Association for Computational Linguistics*, 137–144.

Shieber, Stuart M., Yves Schabes, and Fernando C. Pereira. 1995. Principles and implementations of deductive parsing. *Journal of Logic Programming* 24:3–36.

Sikkel, Klaas. 1997. *Parsing schemata*. Texts in Theoretical Computer Science. Berlin: Springer.