Chapter 10

Context-free parsing

Parsing is the task of determining whether a string can be derived from a given context-free grammar, and if so, how. A parser's output is a tree, like the ones shown in Figure 9.13. Such trees can answer basic questions of who-did-what-to-whom, and have applications in downstream tasks like semantic analysis (chapter 12 and 13) and information extraction (chapter 17).

For a given input and grammar, how many parse trees are there? Consider a minimal context-free grammar with only one non-terminal, X, and the following productions:

$$\begin{aligned} &\text{X} \rightarrow \text{X X} \\ &\text{X} \rightarrow \textit{aardvark} \mid \textit{abacus} \mid \ldots \mid \textit{zyther} \end{aligned}$$

The second line indicates unary productions to every nonterminal in Σ. In this grammar, the number of possible derivations for a string $\boldsymbol{w}$ is equal to the number of binary bracketings, e.g.,

$$((((w_1\, w_2)\, w_3)\, w_4)\, w_5), \quad (((w_1\, (w_2\, w_3))\, w_4)\, w_5), \quad ((w_1\, (w_2 (w_3\, w_4)))\, w_5), \quad \ldots.$$

The number of such bracketings is a **Catalan number**, which grows super-exponentially in the length of the sentence, $C_n = \frac{(2n)!}{(n+1)!n!}$. As with sequence labeling, it is only possible to exhaustively search the space of parses by resorting to locality assumptions, which make it possible to search efficiently by reusing shared substructures with dynamic programming. This chapter focuses on a bottom-up dynamic programming algorithm, which enables exhaustive search of the space of possible parses, but imposes strict limitations on the form of scoring function. These limitations can be relaxed by abandoning exhaustive search. Non-exact search methods will be briefly discussed at the end of this chapter, and one of them — **transition-based parsing** — will be the focus of chapter 11.

S	$\rightarrow$ NP VP
NP	$\rightarrow$ NP PP | *we* | *sushi* | *chopsticks*
PP	$\rightarrow$ IN NP
IN	$\rightarrow$ *with*
VP	$\rightarrow$ V NP | VP PP
V	$\rightarrow$ *eat*

Table 10.1: A toy example context-free grammar

10.1 Deterministic bottom-up parsing

The **CKY algorithm**[1] is a bottom-up approach to parsing in a context-free grammar. It efficiently tests whether a string is in a language, without enumerating all possible parses. The algorithm first forms small constituents, and then tries to merge them into larger constituents.

To understand the algorithm, consider the input, *We eat sushi with chopsticks*. According to the toy grammar in Table 10.1, each terminal symbol can be generated by exactly one unary production, resulting in the sequence NP V NP IN NP. In real examples, there may be many unary productions for each individual token. In any case, the next step is to try to apply binary productions to merge adjacent symbols into larger constituents: for example, V NP can be merged into a verb phrase (VP), and IN NP can be merged into a prepositional phrase (PP). Bottom-up parsing searches for a series of mergers that ultimately results in the start symbol S covering the entire input.

The CKY algorithm systematizes this search by incrementally constructing a table t in which each cell $t[i, j]$ contains the set of nonterminals that can derive the span $\boldsymbol{w}_{i+1:j}$. The algorithm fills in the upper right triangle of the table; it begins with the diagonal, which corresponds to substrings of length 1, and then computes derivations for progressively larger substrings, until reaching the upper right corner $t[0, M]$, which corresponds to the entire input, $\boldsymbol{w}_{1:M}$. If the start symbol S is in $t[0, M]$, then the string $\boldsymbol{w}$ is in the language defined by the grammar. This process is detailed in Algorithm 13, and the resulting data structure is shown in Figure 10.1. Informally, here's how it works:

- Begin by filling in the diagonal: the cells $t[m - 1, m]$ for all $m \in \{1, 2, \ldots, M\}$. These cells are filled with terminal productions that yield the individual tokens; for the word $w_2 =$ *sushi*, we fill in $t[1, 2] = \{\text{NP}\}$, and so on.
- Then fill in the next diagonal, in which each cell corresponds to a subsequence of length two: $t[0, 2], t[1, 3], \ldots, t[M - 2, M]$. These cells are filled in by looking for

[1]The name is for Cocke-Kasami-Younger, the inventors of the algorithm. It is a special case of **chart parsing**, because its stores reusable computations in a chart-like data structure.

binary productions capable of producing at least one entry in each of the cells corresponding to left and right children. For example, VP can be placed in the cell $t[1, 3]$ because the grammar includes the production VP $\rightarrow$ V NP, and because the chart contains V $\in t[1, 2]$ and NP $\in t[2, 3]$.

- At the next diagonal, the entries correspond to spans of length three. At this level, there is an additional decision at each cell: where to split the left and right children. The cell $t[i, j]$ corresponds to the subsequence $\boldsymbol{w}_{i+1:j}$, and we must choose some *split point* $i < k < j$, so that the span $\boldsymbol{w}_{i+1:k}$ is the left child, and the span $\boldsymbol{w}_{k+1:j}$ is the right child. We consider all possible k, looking for productions that generate elements in $t[i, k]$ and $t[k, j]$; the left-hand side of all such productions can be added to $t[i, j]$. When it is time to compute $t[i, j]$, the cells $t[i, k]$ and $t[k, j]$ are guaranteed to be complete, since these cells correspond to shorter sub-strings of the input.
- The process continues until we reach $t[0, M]$.

Figure 10.1 shows the chart that arises from parsing the sentence *We eat sushi with chopsticks* using the grammar defined above.

10.1.1 Recovering the parse tree

As with the Viterbi algorithm, it is possible to identify a successful parse by storing and traversing an additional table of back-pointers. If we add an entry X to cell $t[i, j]$ by using the production $X \rightarrow Y Z$ and the split point k, then we store the back-pointer $b[i, j, X] = (Y, Z, k)$. Once the table is complete, we can recover a parse by tracing this pointers, starting at $b[0, M, \text{S}]$, and stopping when they ground out at terminal productions.

For ambiguous sentences, there will be multiple paths to reach S $\in t[0, M]$. For example, in Figure 10.1, the goal state S $\in t[0, M]$ is reached through the state VP $\in t[1, 5]$, and there are two different ways to generate this constituent: one with (*eat sushi*) and (*with chopsticks*) as children, and another with (*eat*) and (*sushi with chopsticks*) as children. The presence of multiple paths indicates that the input can be generated by the grammar in more than one way. In Algorithm 13, one of these derivations is selected arbitrarily. As discussed in § 10.3, **weighted context-free grammars** compute a score for all permissible derivations, and a minor modification of CKY allows it to identify the single derivation with the maximum score.

10.1.2 Non-binary productions

As presented above, the CKY algorithm assumes that all productions with non-terminals on the right-hand side (RHS) are binary. In real grammars, such as the one considered in chapter 9, there are other types of productions: some have more than two elements on the right-hand side, and others produce a single non-terminal.

Algorithm 13 The CKY algorithm for parsing a sequence $\boldsymbol{w} \in \Sigma^*$ in a context-free grammar $G = (N, \Sigma, R, S)$, with non-terminals N, production rules R, and start symbol S. The grammar is assumed to be in Chomsky normal form (§ 9.2.1). The function PICKFROM($b[i, j, X]$) selects an element of the set $b[i, j, X]$ arbitrarily. All values of t and b are initialized to $\varnothing$.

```
procedure CKY($\boldsymbol{w}, G = (N, \Sigma, R, S)$)
    for $m \in \{1 \ldots M\}$ do
        $t[m-1, m] \leftarrow \{X : (X \rightarrow w_m) \in R\}$
    for $\ell \in \{2, 3, \ldots, M\}$ do                                  ▷ Iterate over constituent lengths
        for $m \in \{0, 1, \ldots M - \ell\}$ do                            ▷ Iterate over left endpoints
            for $k \in \{m+1, m+2, \ldots, m+\ell-1\}$ do                  ▷ Iterate over split points
                for $(X \rightarrow Y\ Z) \in R$ do                          ▷ Iterate over rules
                    if $Y \in t[m, k] \wedge Z \in t[k, m+\ell]$ then
                        $t[m, m+\ell] \leftarrow t[m, m+\ell] \cup X$        ▷ Add non-terminal to table
                        $b[m, m+\ell, X] \leftarrow b[m, m+\ell, X] \cup (Y, Z, k)$   ▷ Add back-pointers
    if S $\in t[0, M]$ then
        return TRACEBACK($S, 0, M, b$)
    else
        return $\varnothing$
procedure TRACEBACK($X, i, j, b$)
    if $j = i + 1$ then
        return $X$
    else
        $(Y, Z, k) \leftarrow$ PICKFROM($b[i, j, X]$)
        return $X \rightarrow$ (TRACEBACK($Y, i, k, b$), TRACEBACK($Z, k, j, b$))
```

- Productions with more than two elements on the right-hand side can be **binarized** by creating additional non-terminals, as described in § 9.2.1. For example, the production VP $\rightarrow$ V NP NP (for ditransitive verbs) can be converted to VP $\rightarrow$ VP$_{ditrans}$/NP NP, by adding the non-terminal VP$_{ditrans}$/NP and the production VP$_{ditrans}$/NP $\rightarrow$ V NP.

- What about unary productions like VP $\rightarrow$ V? While such productions are not a part of Chomsky Normal Form — and can therefore be eliminated in preprocessing the grammar — in practice, a more typical solution is to modify the CKY algorithm. The algorithm makes a second pass on each diagonal in the table, augmenting each cell $t[i, j]$ with all possible unary productions capable of generating each item already in the cell: formally, $t[i, j]$ is extended to its **unary closure**. Suppose the example grammar in Table 10.1 were extended to include the production VP $\rightarrow$ V, enabling sentences with intransitive verb phrases, like *we eat*. Then the cell $t[1, 2]$ — corresponding to the word *eat* — would first include the set {V}, and would be augmented to the set {V, VP} during this second pass.

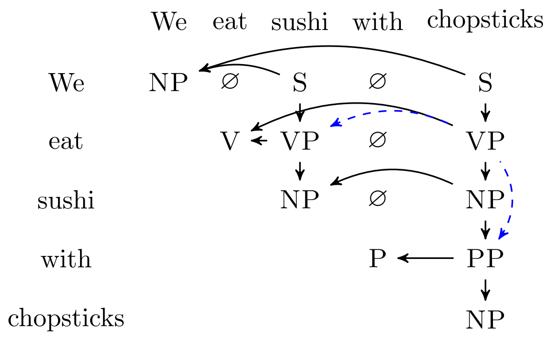


Figure 10.1: An example completed CKY chart. The solid and dashed lines show the back pointers resulting from the two different derivations of VP in position $t[1, 5]$.

10.1.3 Complexity

For an input of length M and a grammar with R productions and N non-terminals, the space complexity of the CKY algorithm is $\mathcal{O}(M^2N)$: the number of cells in the chart is $\mathcal{O}(M^2)$, and each cell must hold $\mathcal{O}(N)$ elements. The time complexity is $\mathcal{O}(M^3R)$: each cell is computed by searching over $\mathcal{O}(M)$ split points, with R possible productions for each split point. Both the time and space complexity are considerably worse than the Viterbi algorithm, which is linear in the length of the input.

10.2 Ambiguity

In natural language, there is rarely a single parse for a given sentence. The main culprit is ambiguity, which is endemic to natural language syntax. Here are a few broad categories:

- **Attachment ambiguity**: e.g., *We eat sushi with chopsticks*, *I shot an elephant in my pajamas.* In these examples, the prepositions (*with*, *in*) can attach to either the verb or the direct object.
- **Modifier scope**: e.g., *southern food store*, *plastic cup holder*. In these examples, the first word could be modifying the subsequent adjective, or the final noun.
- **Particle versus preposition**: e.g., *The puppy tore up the staircase.* Phrasal verbs like *tore up* often include particles which could also act as prepositions. This has structural implications: if *up* is a preposition, then *up the staircase* is a prepositional phrase; if *up* is a particle, then *the staircase* is the direct object to the verb.
- **Complement structure**: e.g., *The students complained to the professor that they didn't understand.* This is another form of attachment ambiguity, where the complement

that they didn't understand could attach to the main verb (*complained*), or to the indirect object (*the professor*).

- **Coordination scope**: e.g., *"I see," said the blind man, as he picked up the hammer and saw.* In this example, the lexical ambiguity for *saw* enables it to be coordinated either with the noun *hammer* or the verb *picked up*.

These forms of ambiguity can combine, so that seemingly simple headlines like *Fed raises interest rates* have dozens of possible analyses even in a minimal grammar. In a broad coverage grammar, typical sentences can have millions of parses. While careful grammar design can chip away at this ambiguity, a better strategy is combine broad coverage parsers with data-driven strategies for identifying the correct analysis.

10.2.1 Parser evaluation

Before continuing to parsing algorithms that are able to handle ambiguity, let us stop to consider how to measure parsing performance. Suppose we have a set of *reference parses* — the ground truth — and a set of *system parses* that we would like to score. A simple solution would be per-sentence accuracy: the parser is scored by the proportion of sentences on which the system and reference parses exactly match.[2] But as any student knows, it always nice to get *partial credit*, which we can assign to analyses that correctly match parts of the reference parse. The PARSEval metrics (Grishman et al., 1992) score each system parse via:

Precision: the fraction of constituents in the system parse that match a constituent in the reference parse.

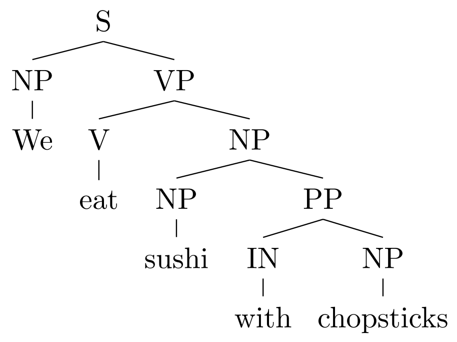
Recall: the fraction of constituents in the reference parse that match a constituent in the system parse.

In **labeled precision** and **recall**, the system must also match the phrase type for each constituent; in **unlabeled precision** and **recall**, it is only required to match the constituent structure. As described in chapter 4, the precision and recall can be combined into an F-MEASURE by their harmonic mean.

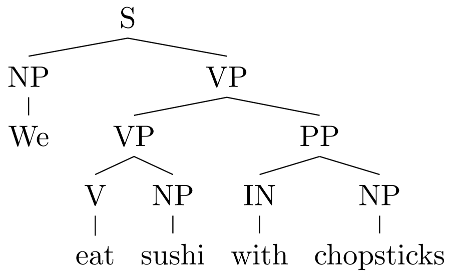
Suppose that the left tree of Figure 10.2 is the system parse, and that the right tree is the reference parse. Then:

- S $\rightarrow \boldsymbol{w}_{1:5}$ is *true positive*, because it appears in both trees.

[2]Most parsing papers do not report results on this metric, but Suzuki et al. (2018) find that a strong parser recovers the exact parse in roughly 50% of all sentences. Performance on short sentences is generally much higher.



(a) system output



(b) reference

Figure 10.2: Two possible analyses from the grammar in Table 10.1

- VP $\rightarrow \boldsymbol{w}_{2:5}$ is *true positive* as well.
- NP $\rightarrow \boldsymbol{w}_{3:5}$ is *false positive*, because it appears only in the system output.
- PP $\rightarrow \boldsymbol{w}_{4:5}$ is *true positive*, because it appears in both trees.
- VP $\rightarrow \boldsymbol{w}_{2:3}$ is *false negative*, because it appears only in the reference.

The labeled and unlabeled precision of this parse is $\frac{3}{4} = 0.75$, and the recall is $\frac{3}{4} = 0.75$, for an F-measure of 0.75. For an example in which precision and recall are not equal, suppose the reference parse instead included the production VP $\rightarrow$ V NP PP. In this parse, the reference does not contain the constituent $\boldsymbol{w}_{2:3}$, so the recall would be 1.[3]

10.2.2 Local solutions

Some ambiguity can be resolved locally. Consider the following examples,

(10.1) a. We met the President on Monday.
b. We met the President of Mexico.

Each case ends with a prepositional phrase, which can be attached to the verb *met* or the noun phrase *the president*. If given a labeled corpus, we can compare the likelihood of the observing the preposition alongside each candidate attachment point,

$$\mathrm{p}(\textit{on} \mid \textit{met}) \gtrless \mathrm{p}(\textit{on} \mid \textit{President}) \quad [10.1]$$

$$\mathrm{p}(\textit{of} \mid \textit{met}) \gtrless \mathrm{p}(\textit{of} \mid \textit{President}). \quad [10.2]$$

[3]While the grammar must be binarized before applying the CKY algorithm, evaluation is performed on the original parses. It is therefore necessary to "unbinarize" the output of a CKY-based parser, converting it back to the original grammar.

A comparison of these probabilities would successfully resolve this case (Hindle and Rooth, 1993). Other cases, such as the example *we eat sushi with chopsticks*, require considering the object of the preposition: consider the alternative *we eat sushi with soy sauce*. With sufficient labeled data, some instances of attachment ambiguity can be solved by supervised classification (Ratnaparkhi et al., 1994).

However, there are inherent limitations to local solutions. While toy examples may have just a few ambiguities to resolve, realistic sentences have thousands or millions of possible parses. Furthermore, attachment decisions are interdependent, as shown in the garden path example:

(10.2) Cats scratch people with claws with knives.

We may want to attach *with claws* to *scratch*, as would be correct in the shorter sentence in *cats scratch people with claws*. But this leaves nowhere to attach *with knives*. The correct interpretation can be identified only be considering the attachment decisions jointly. The huge number of potential parses may seem to make exhaustive search impossible. But as with sequence labeling, locality assumptions make it possible to search this space efficiently.

10.3 Weighted Context-Free Grammars

Let us define a derivation τ as a set of **anchored productions**,

$$\tau = \{X \to \alpha, (i, j, k)\}, \tag{10.3}$$

with X corresponding to the left-hand side non-terminal and α corresponding to the right-hand side. For grammars in Chomsky normal formal, α is either a pair of non-terminals or a terminal symbol. The indices i, j, k anchor the production in the input, with X deriving the span $\boldsymbol{w}_{i+1:j}$. For binary productions, $\boldsymbol{w}_{i+1:k}$ indicates the span of the left child, and $\boldsymbol{w}_{k+1:j}$ indicates the span of the right child; for unary productions, k is ignored. For an input $\boldsymbol{w}$, the optimal parse is,

$$\hat{\tau} = \underset{\tau \in \mathcal{T}(\boldsymbol{w})}{\operatorname{argmax}} \Psi(\tau), \tag{10.4}$$

where $\mathcal{T}(\boldsymbol{w})$ is the set of derivations that yield the input $\boldsymbol{w}$.

Define a scoring function Ψ that decomposes across anchored productions,

$$\Psi(\tau) = \sum_{(X \to \alpha, (i,j,k)) \in \tau} \psi(X \to \alpha, (i, j, k)). \tag{10.5}$$

This is a locality assumption, akin to the assumption in Viterbi sequence labeling. In this case, the assumption states that the overall score is a sum over scores of productions,

		$\psi(\cdot)$	$\exp \psi(\cdot)$
S	$\rightarrow$ NP VP	0	1
NP	$\rightarrow$ NP PP	-1	$\frac{1}{2}$
	$\rightarrow$ *we*	-2	$\frac{1}{4}$
	$\rightarrow$ *sushi*	-3	$\frac{1}{8}$
	$\rightarrow$ *chopsticks*	-3	$\frac{1}{8}$
PP	$\rightarrow$ IN NP	0	1
IN	$\rightarrow$ *with*	0	1
VP	$\rightarrow$ V NP	-1	$\frac{1}{2}$
	$\rightarrow$ VP PP	-2	$\frac{1}{4}$
	$\rightarrow$ MD V	-2	$\frac{1}{4}$
V	$\rightarrow$ *eat*	0	1

Table 10.2: An example weighted context-free grammar (WCFG). The weights are chosen so that $\exp \psi(\cdot)$ sums to one over right-hand sides for each non-terminal; this is required by probabilistic context-free grammars, but not by WCFGs in general.

which are computed independently. In a **weighted context-free grammar** (WCFG), the score of each anchored production $X \rightarrow (\alpha, (i, j, k))$ is simply $\psi(X \rightarrow \alpha)$, ignoring the anchor (i, j, k). In other parsing models, the anchors can be used to access features of the input, while still permitting efficient bottom-up parsing.

Example Consider the weighted grammar shown in Table 10.2, and the analysis in Figure 10.2b.

$$\begin{aligned}\Psi(\tau) =& \psi(\text{S} \rightarrow \text{NP VP}) + \psi(\text{VP} \rightarrow \text{VP PP}) + \psi(\text{VP} \rightarrow \text{V NP}) + \psi(\text{PP} \rightarrow \text{IN NP}) \\ &+ \psi(\text{NP} \rightarrow \textit{We}) + \psi(\text{V} \rightarrow \textit{eat}) + \psi(\text{NP} \rightarrow \textit{sushi}) + \psi(\text{IN} \rightarrow \textit{with}) + \psi(\text{NP} \rightarrow \textit{chopsticks}) \qquad [10.6] \\ =& 0 - 2 - 1 + 0 - 2 + 0 - 3 + 0 - 3 = -11. \qquad [10.7]\end{aligned}$$

In the alternative parse in Figure 10.2a, the production VP $\rightarrow$ VP PP (with score -2) is replaced with the production NP $\rightarrow$ NP PP (with score -1); all other productions are the same. As a result, the score for this parse is -10. This example hints at a problem with WCFG parsing on non-terminals such as NP, VP, and PP: a WCFG will *always* prefer either VP or NP attachment, regardless of what is being attached! Solutions to this issue are discussed in § 10.5.

Algorithm 14 CKY algorithm for parsing a string $\boldsymbol{w} \in \Sigma^*$ in a weighted context-free grammar (N, Σ, R, S), where N is the set of non-terminals and R is the set of weighted productions. The grammar is assumed to be in Chomsky normal form (§ 9.2.1). The function TRACEBACK is defined in Algorithm 13.

```
procedure WCKY($\boldsymbol{w}, G = (N, \Sigma, R, S)$)
    for all $i, j, X$ do                                    ▷ Initialization
        $t[i, j, X] \leftarrow 0$
        $b[i, j, X] \leftarrow \varnothing$
    for $m \in \{1, 2, \ldots, M\}$ do
        for all $X \in N$ do
            $t[m, m+1, X] \leftarrow \psi(X \rightarrow w_m, (m, m+1, m))$
    for $\ell \in \{2, 3, \ldots M\}$ do
        for $m \in \{0, 1, \ldots, M - \ell\}$ do
            for $k \in \{m+1, m+2, \ldots, m+\ell-1\}$ do
                $t[m, m+\ell, X] \leftarrow \max_{k,Y,Z} \psi(X \rightarrow Y\ Z, (m, m+\ell, k)) + t[m, k, Y] + t[k, m+\ell, Z]$
                $b[m, m+\ell, X] \leftarrow \operatorname{argmax}_{k,Y,Z} \psi(X \rightarrow Y\ Z, (m+\ell, k)) + t[m, k, Y] + t[k, m+\ell, Z]$
    return TRACEBACK($S, 0, M, b$)
```

10.3.1 Parsing with weighted context-free grammars

The optimization problem in Equation 10.4 can be solved by modifying the CKY algorithm. In the deterministic CKY algorithm, each cell $t[i, j]$ stored a set of non-terminals capable of deriving the span $\boldsymbol{w}_{i+1:j}$. We now augment the table so that the cell $t[i, j, X]$ is the *score of the best derivation* of $w_{i+1:j}$ from non-terminal X. This score is computed recursively: for the anchored binary production $(X \rightarrow Y\ Z, (i, j, k))$, we compute:

- the score of the anchored production, $\psi(X \rightarrow Y\ Z, (i, j, k))$;
- the score of the best derivation of the left child, $t[i, k, Y]$;
- the score of the best derivation of the right child, $t[k, j, Z]$.

These scores are combined by addition. As in the unweighted CKY algorithm, the table is constructed by considering spans of increasing length, so the scores for spans $t[i, k, Y]$ and $t[k, j, Z]$ are guaranteed to be available at the time we compute the score $t[i, j, X]$. The value $t[0, M, \mathrm{S}]$ is the score of the best derivation of $\boldsymbol{w}$ from the grammar. Algorithm 14 formalizes this procedure.

As in unweighted CKY, the parse is recovered from the table of back pointers b, where each $b[i, j, X]$ stores the argmax split point k and production $X \rightarrow Y\ Z$ in the derivation of $\boldsymbol{w}_{i+1:j}$ from X. The top scoring parse can be obtained by tracing these pointers backwards from $b[0, M, \mathrm{S}]$, all the way to the terminal symbols. This is analogous to the computation of the best sequence of labels in the Viterbi algorithm by tracing pointers backwards from the end of the trellis. Note that we need only store back-pointers for the *best* path to $t[i, j, X]$; this follows from the locality assumption that the global score for a parse is a combination of the local scores of each production in the parse.

Example Let's revisit the parsing table in Figure 10.1. In a weighted CFG, each cell would include a score for each non-terminal; non-terminals that cannot be generated are assumed to have a score of $-\infty$. The first diagonal contains the scores of unary productions: $t[0, 1, \text{NP}] = -2$, $t[1, 2, \text{V}] = 0$, and so on. The next diagonal contains the scores for spans of length 2: $t[1, 3, \text{VP}] = -1 + 0 - 3 = -4$, $t[3, 5, \text{PP}] = 0 + 0 - 3 = -3$, and so on. Things get interesting when we reach the cell $t[1, 5, \text{VP}]$, which contains the score for the derivation of the span $\boldsymbol{w}_{2:5}$ from the non-terminal VP. This score is computed as a max over two alternatives,

$$
\begin{aligned}
t[1, 5, \text{VP}] = \max(&\psi(\text{VP} \rightarrow \text{VP PP}, (1, 3, 5)) + t[1, 3, \text{VP}] + t[3, 5, \text{PP}],\\
&\psi(\text{VP} \rightarrow \text{V NP}, (1, 2, 5)) + t[1, 2, \text{V}] + t[2, 5, \text{NP}]) && [10.8]\\
= \max(&-2 - 4 - 3,\ -1 + 0 - 7) = -8. && [10.9]
\end{aligned}
$$

Since the second case is the argmax, we set the back-pointer $b[1, 5, \text{VP}] = (\text{V}, \text{NP}, 2)$, enabling the optimal derivation to be recovered.

10.3.2 Probabilistic context-free grammars

Probabilistic context-free grammars (PCFGs) are a special case of weighted context-free grammars that arises when the weights correspond to probabilities. Specifically, the weight $\psi(X \rightarrow \alpha, (i, j, k)) = \log \text{p}(\alpha \mid X)$, where the probability of the right-hand side α is conditioned on the non-terminal X, and the anchor (i, j, k) is ignored. These probabilities must be normalized over all possible right-hand sides, so that $\sum_\alpha \text{p}(\alpha \mid X) = 1$, for all X. For a given parse τ, the product of the probabilities of the productions is equal to $\text{p}(\tau)$, under the **generative model** $\tau \sim \text{DRAWSUBTREE}(S)$, where the function DRAWSUBTREE is defined in Algorithm 15.

The conditional probability of a parse given a string is,

$$
\text{p}(\tau \mid \boldsymbol{w}) = \frac{\text{p}(\tau)}{\sum_{\tau' \in \mathcal{T}(\boldsymbol{w})} \text{p}(\tau')} = \frac{\exp \Psi(\tau)}{\sum_{\tau' \in \mathcal{T}(\boldsymbol{w})} \exp \Psi(\tau')}, \qquad [10.10]
$$

where $\Psi(\tau) = \sum_{X \rightarrow \alpha, (i,j,k) \in \tau} \psi(X \rightarrow \alpha, (i, j, k))$. Because the probability is monotonic in the score $\Psi(\tau)$, the maximum likelihood parse can be identified by the CKY algorithm without modification. If a normalized probability $\text{p}(\tau \mid \boldsymbol{w})$ is required, the denominator of Equation 10.10 can be computed by the **inside recurrence**, described below.

Algorithm 15 Generative model for derivations from probabilistic context-free grammars in Chomsky Normal Form (CNF).

```
procedure DrawSubtree(X)
    sample $(X \to \alpha) \sim \mathrm{p}(\alpha \mid X)$
    if $\alpha = (Y\ Z)$ then
        return DrawSubtree($Y$) $\cup$ DrawSubtree($Z$)
    else
        return $(X \to \alpha)$        ▷ In CNF, all unary productions yield terminal symbols
```

Example The WCFG in Table 10.2 is designed so that the weights are log-probabilities, satisfying the constraint $\sum_\alpha \exp \psi(X \to \alpha) = 1$. As noted earlier, there are two parses in $\mathcal{T}$(*we eat sushi with chopsticks*), with scores $\Psi(\tau_1) = \log \mathrm{p}(\tau_1) = -10$ and $\Psi(\tau_2) = \log \mathrm{p}(\tau_2) = -11$. Therefore, the conditional probability $\mathrm{p}(\tau_1 \mid \boldsymbol{w})$ is equal to,

$$\mathrm{p}(\tau_1 \mid \boldsymbol{w}) = \frac{\mathrm{p}(\tau_1)}{\mathrm{p}(\tau_1) + \mathrm{p}(\tau_2)} = \frac{\exp \Psi(\tau_1)}{\exp \Psi(\tau_1) + \exp \Psi(\tau_2)} = \frac{2^{-10}}{2^{-10} + 2^{-11}} = \frac{2}{3}. \quad [10.11]$$

The inside recurrence The denominator of Equation 10.10 can be viewed as a language model, summing over all valid derivations of the string $\boldsymbol{w}$,

$$\mathrm{p}(\boldsymbol{w}) = \sum_{\tau' : \text{yield}(\tau') = \boldsymbol{w}} \mathrm{p}(\tau'). \quad [10.12]$$

Just as the CKY algorithm makes it possible to maximize over all such analyses, with a few modifications it can also compute their sum. Each cell $t[i, j, X]$ must store the log probability of deriving $\boldsymbol{w}_{i+1:j}$ from non-terminal X. To compute this, we replace the maximization over split points k and productions $X \to Y\ Z$ with a "log-sum-exp" operation, which exponentiates the log probabilities of the production and the children, sums them in probability space, and then converts back to the log domain:

$$t[i, j, X] = \log \sum_{k,Y,Z} \exp\left(\psi(X \to Y\ Z) + t[i, k, Y] + t[k, j, Z]\right) \quad [10.13]$$

$$= \log \sum_{k,Y,Z} \exp\left(\log \mathrm{p}(Y\ Z \mid X) + \log \mathrm{p}(Y \to \boldsymbol{w}_{i+1:k}) + \log \mathrm{p}(Z \to \boldsymbol{w}_{k+1:j})\right) \quad [10.14]$$

$$= \log \sum_{k,Y,Z} \mathrm{p}(Y\ Z \mid X) \times \mathrm{p}(Y \to \boldsymbol{w}_{i+1:k}) \times \mathrm{p}(Z \to \boldsymbol{w}_{k+1:j}) \quad [10.15]$$

$$= \log \sum_{k,Y,Z} \mathrm{p}(Y\ Z, \boldsymbol{w}_{i+1:k}, \boldsymbol{w}_{k+1:j} \mid X) \quad [10.16]$$

$$= \log \mathrm{p}(X \rightsquigarrow \boldsymbol{w}_{i+1:j}), \quad [10.17]$$

with $X \rightsquigarrow \boldsymbol{w}_{i+1:j}$ indicating the event that non-terminal X yields the span $w_{i+1}, w_{i+2}, \ldots, w_j$. The recursive computation of $t[i, j, X]$ is called the **inside recurrence**, because it computes the probability of each subtree as a combination of the probabilities of the smaller subtrees that are inside of it. The name implies a corresponding **outside recurrence**, which computes the probability of a non-terminal X spanning $\boldsymbol{w}_{i+1:j}$, joint with the outside context $(\boldsymbol{w}_{1:i}, \boldsymbol{w}_{j+1:M})$. This recurrence is described in § 10.4.3. The inside and outside recurrences are analogous to the forward and backward recurrences in probabilistic sequence labeling (see § 7.5.3). They can be used to compute the marginal probabilities of individual anchored productions, $\mathrm{p}(X \to \alpha, (i, j, k) \mid \boldsymbol{w})$, summing over all possible derivations of $\boldsymbol{w}$.

10.3.3 *Semiring weighted context-free grammars

The weighted and unweighted CKY algorithms can be unified with the inside recurrence using the same semiring notation described in § 7.7.3. The generalized recurrence is:

$$t[i, j, X] = \bigoplus_{k,Y,Z} \psi(X \to Y\ Z, (i, j, k)) \otimes t[i, k, Y] \otimes t[k, j, Z]. \qquad [10.18]$$

This recurrence subsumes all of the algorithms that have been discussed in this chapter to this point.

Unweighted CKY. When $\psi(X \to \alpha, (i, j, k))$ is a *Boolean truth value* $\{\top, \bot\}$, $\otimes$ is logical conjunction, and $\bigoplus$ is logical disjunction, then we derive CKY recurrence for unweighted context-free grammars, discussed in § 10.1 and Algorithm 13.

Weighted CKY. When $\psi(X \to \alpha, (i, j, k))$ is a scalar score, $\otimes$ is addition, and $\bigoplus$ is maximization, then we derive the CKY recurrence for weighted context-free grammars, discussed in § 10.3 and Algorithm 14. When $\psi(X \to \alpha, (i, j, k)) = \log \mathrm{p}(\alpha \mid X)$, this same setting derives the CKY recurrence for finding the maximum likelihood derivation in a probabilistic context-free grammar.

Inside recurrence. When $\psi(X \to \alpha, (i, j, k))$ is a log probability, $\otimes$ is addition, and $\bigoplus = \log \sum \exp$, then we derive the inside recurrence for probabilistic context-free grammars, discussed in § 10.3.2. It is also possible to set $\psi(X \to \alpha, (i, j, k))$ directly equal to the probability $\mathrm{p}(\alpha \mid X)$. In this case, $\otimes$ is multiplication, and $\bigoplus$ is addition. While this may seem more intuitive than working with $\log$ probabilities, there is the risk of underflow on long inputs.

Regardless of how the scores are combined, the key point is the locality assumption: the score for a derivation is the combination of the independent scores for each anchored production, and these scores do not depend on any other part of the derivation. For example, if two non-terminals are siblings, the scores of productions from these non-terminals are computed independently. This locality assumption is analogous to the first-order Markov assumption in sequence labeling, where the score for transitions between tags depends only on the previous tag and current tag, and not on the history. As with sequence labeling, this assumption makes it possible to find the optimal parse efficiently; its linguistic limitations are discussed in § 10.5.

10.4 Learning weighted context-free grammars

Like sequence labeling, context-free parsing is a form of structure prediction. As a result, WCFGs can be learned using the same set of algorithms: generative probabilistic models, structured perceptron, maximum conditional likelihood, and maximum margin learning. In all cases, learning requires a **treebank**, which is a dataset of sentences labeled with context-free parses. Parsing research was catalyzed by the **Penn Treebank** (Marcus et al., 1993), the first large-scale dataset of this type (see § 9.2.2). Phrase structure treebanks exist for roughly two dozen other languages, with coverage mainly restricted to European and East Asian languages, plus Arabic and Urdu.

10.4.1 Probabilistic context-free grammars

Probabilistic context-free grammars are similar to hidden Markov models, in that they are generative models of text. In this case, the parameters of interest correspond to probabilities of productions, conditional on the left-hand side. As with hidden Markov models, these parameters can be estimated by relative frequency:

$$\psi(X \to \alpha) = \log \mathrm{p}(X \to \alpha) \qquad [10.19]$$

$$\hat{\mathrm{p}}(X \to \alpha) = \frac{\mathrm{count}(X \to \alpha)}{\mathrm{count}(X)}. \qquad [10.20]$$

For example, the probability of the production NP $\to$ DET NN is the corpus count of this production, divided by the count of the non-terminal NP. This estimator applies to terminal productions as well: the probability of NN $\to$ *whale* is the count of how often *whale* appears in the corpus as generated from an NN tag, divided by the total count of the NN tag. Even with the largest treebanks — currently on the order of one million tokens — it is difficult to accurately compute probabilities of even moderately rare events, such as NN $\to$ *whale*. Therefore, smoothing is critical for making PCFGs effective.

10.4.2 Feature-based parsing

The scores for each production can be computed as an inner product of weights and features,

$$\psi(X \to \alpha, (i, j, k)) = \boldsymbol{\theta} \cdot \boldsymbol{f}(X, \alpha, (i, j, k), \boldsymbol{w}), \qquad [10.21]$$

where the feature vector $\boldsymbol{f}$ is a function of the left-hand side X, the right-hand side α, the anchor indices (i, j, k), and the input $\boldsymbol{w}$.

The basic feature $\boldsymbol{f}(X, \alpha, (i, j, k)) = \{(X, \alpha)\}$ encodes only the identity of the production itself. This gives rise to a discriminatively-trained model with the same expressiveness as a PCFG. Features on anchored productions can include the words that border the span w_i, w_{j+1}, the word at the split point w_{k+1}, the presence of a verb or noun in the left child span $w_{i+1:k}$, and so on (Durrett and Klein, 2015). Scores on anchored productions can be incorporated into CKY parsing without any modification to the algorithm, because it is still possible to compute each element of the table $t[i, j, X]$ recursively from its immediate children.

Other features can be obtained by grouping elements on either the left-hand or right-hand side: for example it can be particularly beneficial to compute additional features by clustering terminal symbols, with features corresponding to groups of words with similar syntactic properties. The clustering can be obtained from unlabeled datasets that are much larger than any treebank, improving coverage. Such methods are described in chapter 14.

Feature-based parsing models can be estimated using the usual array of discriminative learning techniques. For example, a structure perceptron update can be computed as (Carreras et al., 2008),

$$\boldsymbol{f}(\tau, \boldsymbol{w}^{(i)}) = \sum_{(X \to \alpha, (i,j,k)) \in \tau} \boldsymbol{f}(X, \alpha, (i, j, k), \boldsymbol{w}^{(i)}) \qquad [10.22]$$

$$\hat{\tau} = \underset{\tau \in \mathcal{T}(\boldsymbol{w})}{\operatorname{argmax}}\, \boldsymbol{\theta} \cdot \boldsymbol{f}(\tau, \boldsymbol{w}^{(i)}) \qquad [10.23]$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{f}(\tau^{(i)}, \boldsymbol{w}^{(i)}) - \boldsymbol{f}(\hat{\tau}, \boldsymbol{w}^{(i)}). \qquad [10.24]$$

A margin-based objective can be optimized by selecting $\hat{\tau}$ through cost-augmented decoding (§ 2.4.2), enforcing a margin of $\Delta(\hat{\tau}, \tau)$ between the hypothesis and the reference parse, where Δ is a non-negative cost function, such as the Hamming loss (Stern et al., 2017). It is also possible to train feature-based parsing models by conditional log-likelihood, as described in the next section.

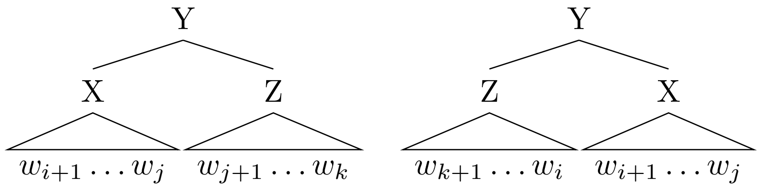


Figure 10.3: The two cases faced by the outside recurrence in the computation of $\beta(i, j, X)$

10.4.3 *Conditional random field parsing

The score of a derivation $\Psi(\tau)$ can be converted into a probability by normalizing over all possible derivations,

$$p(\tau \mid \boldsymbol{w}) = \frac{\exp \Psi(\tau)}{\sum_{\tau' \in \mathcal{T}(\boldsymbol{w})} \exp \Psi(\tau')}. \quad [10.25]$$

Using this probability, a WCFG can be trained by maximizing the conditional log-likelihood of a labeled corpus.

Just as in logistic regression and the conditional random field over sequences, the gradient of the conditional log-likelihood is the difference between the observed and expected counts of each feature. The expectation $E_{\tau|\boldsymbol{w}}[\boldsymbol{f}(\tau, \boldsymbol{w}^{(i)}); \boldsymbol{\theta}]$ requires summing over all possible parses, and computing the marginal probabilities of anchored productions, $p(X \to \alpha, (i, j, k) \mid \boldsymbol{w})$. In CRF sequence labeling, marginal probabilities over tag bigrams are computed by the two-pass **forward-backward algorithm** (§ 7.5.3). The analogue for context-free grammars is the **inside-outside algorithm**, in which marginal probabilities are computed from terms generated by an upward and downward pass over the parsing chart:

- The upward pass is performed by the inside recurrence, which is described in § 10.3.2. Each inside variable $\alpha(i, j, X)$ is the score of deriving $\boldsymbol{w}_{i+1:j}$ from the non-terminal X. In a PCFG, this corresponds to the log-probability $\log p(\boldsymbol{w}_{i+1:j} \mid X)$. This is computed by the recurrence,

$$\alpha(i, j, X) \triangleq \log \sum_{(X \to Y\ Z)} \sum_{k=i+1}^{j} \exp\left(\psi(X \to Y\ Z, (i, j, k)) + \alpha(i, k, Y) + \alpha(k, j, Z)\right). \quad [10.26]$$

  The initial condition of this recurrence is $\alpha(m-1, m, X) = \psi(X \to w_m)$. The denominator $\sum_{\tau \in \mathcal{T}(\boldsymbol{w})} \exp \Psi(\tau)$ is equal to $\exp \alpha(0, M, \text{S})$.

- The downward pass is performed by the **outside recurrence**, which recursively populates the same table structure, starting at the root of the tree. Each outside variable

$\beta(i, j, X)$ is the score of having a phrase of type X covering the span $(i+1 : j)$, joint with the exterior context $\boldsymbol{w}_{1:i}$ and $\boldsymbol{w}_{j+1:M}$. In a PCFG, this corresponds to the log probability $\log \mathrm{p}((X, i+1, j), \boldsymbol{w}_{1:i}, \boldsymbol{w}_{j+1:M})$. Each outside variable is computed by the recurrence,

$$\exp \beta(i, j, X) \triangleq \sum_{(Y \to X\ Z)} \sum_{k=j+1}^{M} \exp\left[\psi(Y \to X\ Z, (i, k, j)) + \alpha(j, k, Z) + \beta(i, k, Y)\right] \tag{10.27}$$

$$+ \sum_{(Y \to Z\ X)} \sum_{k=0}^{i-1} \exp\left[\psi(Y \to Z\ X, (k, i, j)) + \alpha(k, i, Z) + \beta(k, j, Y)\right]. \tag{10.28}$$

The first line of Equation 10.28 is the score under the condition that X is a left child of its parent, which spans $\boldsymbol{w}_{i+1:k}$, with $k > j$; the second line is the score under the condition that X is a right child of its parent Y, which spans $\boldsymbol{w}_{k+1:j}$, with $k < i$. The two cases are shown in Figure 10.3. In each case, we sum over all possible productions with X on the right-hand side. The parent Y is bounded on one side by either i or j, depending on whether X is a left or right child of Y; we must sum over all possible values for the other boundary. The initial conditions for the outside recurrence are $\beta(0, M, \mathrm{S}) = 0$ and $\beta(0, M, X \neq \mathrm{S}) = -\infty$.

The marginal probability of a non-terminal X over span $\boldsymbol{w}_{i+1:j}$ is written $\mathrm{p}(X \rightsquigarrow \boldsymbol{w}_{i+1:j} \mid \boldsymbol{w})$. This probability can be computed from the inside and outside scores,

$$\mathrm{p}(X \rightsquigarrow \boldsymbol{w}_{i+1:j} \mid \boldsymbol{w}) = \frac{\mathrm{p}(X \rightsquigarrow \boldsymbol{w}_{i+1:j}, \boldsymbol{w})}{\mathrm{p}(\boldsymbol{w})} \tag{10.29}$$

$$= \frac{\mathrm{p}(\boldsymbol{w}_{i+1:j} \mid X) \times \mathrm{p}(X, \boldsymbol{w}_{1:i}, \boldsymbol{x}_{j+1:M})}{\mathrm{p}(\boldsymbol{w})} \tag{10.30}$$

$$= \frac{\exp\left(\alpha(i, j, X) + \beta(i, j, X)\right)}{\exp \alpha(0, M, S)}. \tag{10.31}$$

Marginal probabilities of individual productions can be computed similarly (see exercise 2). These marginal probabilities can be used for training a conditional random field parser, and also for the task of unsupervised **grammar induction**, in which a PCFG is estimated from a dataset of unlabeled text (Lari and Young, 1990; Pereira and Schabes, 1992).

10.4.4 Neural context-free grammars

Neural networks and can be applied to parsing by representing each span with a dense numerical vector (Socher et al., 2013; Durrett and Klein, 2015; Cross and Huang, 2016).[4] For example, the anchor (i, j, k) and sentence $\boldsymbol{w}$ can be associated with a fixed-length column vector,

$$\boldsymbol{v}_{(i,j,k)} = [\boldsymbol{u}_{w_{i-1}}; \boldsymbol{u}_{w_i}; \boldsymbol{u}_{w_{j-1}}; \boldsymbol{u}_{w_j}; \boldsymbol{u}_{w_{k-1}}; \boldsymbol{u}_{w_k}], \qquad [10.32]$$

where $\boldsymbol{u}_{w_i}$ is a word embedding associated with the word w_i. The vector $\boldsymbol{v}_{i,j,k}$ can then be passed through a feedforward neural network, and used to compute the score of the anchored production. For example, this score can be computed as a bilinear product (Durrett and Klein, 2015),

$$\tilde{\boldsymbol{v}}_{(i,j,k)} = \text{FeedForward}(\boldsymbol{v}_{(i,j,k)}) \qquad [10.33]$$

$$\psi(X \to \alpha, (i, j, k)) = \tilde{\boldsymbol{v}}_{(i,j,k)}^{\top} \boldsymbol{\Theta} \boldsymbol{f}(X \to \alpha), \qquad [10.34]$$

where $\boldsymbol{f}(X \to \alpha)$ is a vector of features of the production, and $\boldsymbol{\Theta}$ is a parameter matrix. The matrix $\boldsymbol{\Theta}$ and the parameters of the feedforward network can be learned by backpropagating from an objective such as the margin loss or the negative conditional log-likelihood.

10.5 Grammar refinement

The locality assumptions underlying CFG parsing depend on the granularity of the non-terminals. For the Penn Treebank non-terminals, there are several reasons to believe that these assumptions are too strong (Johnson, 1998):

- The context-free assumption is too strict: for example, the probability of the production NP $\to$ NP PP is much higher (in the PTB) if the parent of the noun phrase is a verb phrase (indicating that the NP is a direct object) than if the parent is a sentence (indicating that the NP is the subject of the sentence).
- The Penn Treebank non-terminals are too coarse: there are many kinds of noun phrases and verb phrases, and accurate parsing sometimes requires knowing the difference. As we have already seen, when faced with prepositional phrase attachment ambiguity, a weighted CFG will either always choose NP attachment (if $\psi(\text{NP} \to \text{NP PP}) > \psi(\text{VP} \to \text{VP PP})$), or it will always choose VP attachment. To get more nuanced behavior, more fine-grained non-terminals are needed.
- More generally, accurate parsing requires some amount of **semantics** — understanding the meaning of the text to be parsed. Consider the example *cats scratch people*

[4]Earlier work on neural constituent parsing used transition-based parsing algorithms (§ 10.6.2) rather than CKY-style chart parsing (Henderson, 2004; Titov and Henderson, 2007).

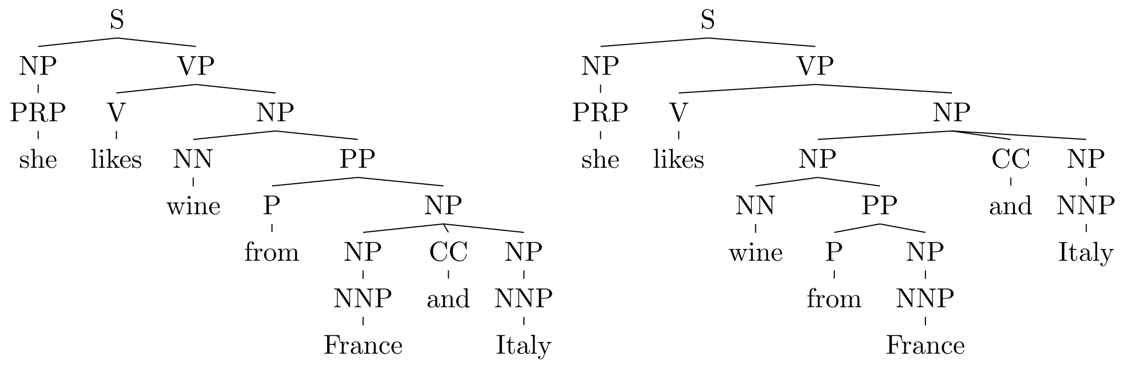


Figure 10.4: The left parse is preferable because of the conjunction of phrases headed by *France* and *Italy*, but these parses cannot be distinguished by a WCFG.

with claws: knowledge of about cats, claws, and scratching is necessary to correctly resolve the attachment ambiguity.

An extreme example is shown in Figure 10.4. The analysis on the left is preferred because of the conjunction of similar entities *France* and *Italy*. But given the non-terminals shown in the analyses, there is no way to differentiate these two parses, since they include exactly the same productions. What is needed seems to be more precise non-terminals. One possibility would be to rethink the linguistics behind the Penn Treebank, and ask the annotators to try again. But the original annotation effort took five years, and there is a little appetite for another annotation effort of this scope. Researchers have therefore turned to automated techniques.

10.5.1 Parent annotations and other tree transformations

The key assumption underlying context-free parsing is that productions depend only on the identity of the non-terminal on the left-hand side, and not on its ancestors or neighbors. The validity of this assumption is an empirical question, and it depends on the non-terminals themselves: ideally, every noun phrase (and verb phrase, etc) would be distributionally identical, so the assumption would hold. But in the Penn Treebank, the observed probability of productions often depends on the parent of the left-hand side. For example, noun phrases are more likely to be modified by prepositional phrases when they are in the object position (e.g., *they amused the students from Georgia*) than in the subject position (e.g., *the students from Georgia amused them*). This means that the NP $\rightarrow$ NP PP production is more likely if the entire constituent is the child of a VP than if it is the child

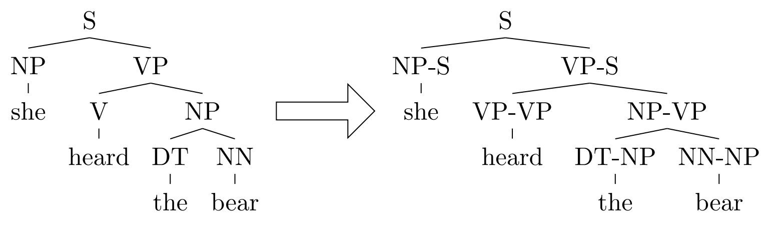


Figure 10.5: Parent annotation in a CFG derivation

of S. The observed statistics are (Johnson, 1998):

$$\Pr(\text{NP} \rightarrow \text{NP PP}) = 11\% \tag{10.35}$$

$$\Pr(\text{NP under S} \rightarrow \text{NP PP}) = 9\% \tag{10.36}$$

$$\Pr(\text{NP under VP} \rightarrow \text{NP PP}) = 23\%. \tag{10.37}$$

This phenomenon can be captured by **parent annotation** (Johnson, 1998), in which each non-terminal is augmented with the identity of its parent, as shown in Figure 10.5). This is sometimes called **vertical Markovization**, since a Markov dependency is introduced between each node and its parent (Klein and Manning, 2003). It is analogous to moving from a bigram to a trigram context in a hidden Markov model. In principle, parent annotation squares the size of the set of non-terminals, which could make parsing considerably less efficient. But in practice, the increase in the number of non-terminals that actually appear in the data is relatively modest (Johnson, 1998).

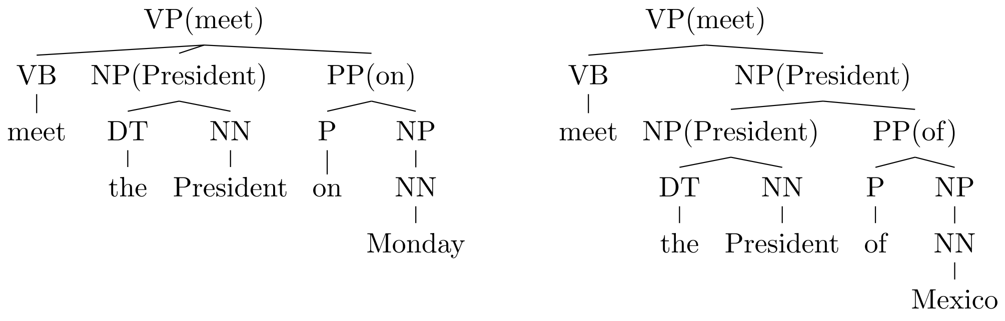
Parent annotation weakens the WCFG locality assumptions. This improves accuracy by enabling the parser to make more fine-grained distinctions, which better capture real linguistic phenomena. However, each production is more rare, and so careful smoothing or regularization is required to control the variance over production scores.

10.5.2 Lexicalized context-free grammars

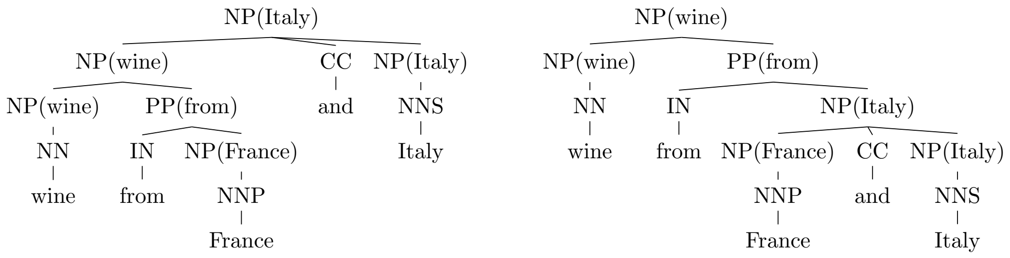
The examples in § 10.2.2 demonstrate the importance of individual words in resolving parsing ambiguity: the preposition *on* is more likely to attach to *met*, while the preposition *of* is more likely to attachment to *President*. But of all word pairs, which are relevant to attachment decisions? Consider the following variants on the original examples:

(10.3) a. We $\underline{\text{met}}$ the $\underline{\text{President}}$ of Mexico.
b. We $\underline{\text{met}}$ the first female $\underline{\text{President}}$ of Mexico.
c. They had supposedly $\underline{\text{met}}$ the $\underline{\text{President}}$ on Monday.

The underlined words are the **head words** of their respective phrases: *met* heads the verb phrase, and *President* heads the direct object noun phrase. These heads provide useful



(a) Lexicalization and attachment ambiguity



(b) Lexicalization and coordination scope ambiguity

Figure 10.6: Examples of lexicalization

semantic information. But they break the context-free assumption, which states that the score for a production depends only on the parent and its immediate children, and not the substructure under each child.

The incorporation of head words into context-free parsing is known as **lexicalization**, and is implemented in rules of the form,

$$\text{NP}(\textit{President}) \rightarrow \text{NP}(\textit{President})\ \text{PP}(\textit{of}) \quad [10.38]$$
$$\text{NP}(\textit{President}) \rightarrow \text{NP}(\textit{President})\ \text{PP}(\textit{on}). \quad [10.39]$$

Lexicalization was a major step towards accurate PCFG parsing in the 1990s and early 2000s. It requires solving three problems: identifying the heads of all constituents in a treebank; parsing efficiently while keeping track of the heads; and estimating the scores for lexicalized productions.

Non-terminal	Direction	Priority
S	right	VP SBAR ADJP UCP NP
VP	left	VBD VBN MD VBZ TO VB VP VBG VBP ADJP NP
NP	right	N* EX $ CD QP PRP ...
PP	left	IN TO FW

Table 10.3: A fragment of head percolation rules for English (Magerman, 1995; Collins, 1997)

Identifying head words

The head of a constituent is the word that is the most useful for determining how that constituent is integrated into the rest of the sentence.[5] The head word of a constituent is determined recursively: for any non-terminal production, the head of the left-hand side must be the head of one of the children. The head is typically selected according to a set of deterministic rules, sometimes called **head percolation rules**. In many cases, these rules are straightforward: the head of a noun phrase in a NP $\rightarrow$ DET NN production is the head of the noun; the head of a sentence in a S $\rightarrow$ NP VP production is the head of the verb phrase.

Table 10.3 shows a fragment of the head percolation rules used in many English parsing systems. The meaning of the first rule is that to find the head of an S constituent, first look for the rightmost VP child; if you don't find one, then look for the rightmost SBAR child, and so on down the list. Verb phrases are headed by left verbs (the head of *can plan on walking* is *planned*, since the modal verb *can* is tagged MD); noun phrases are headed by the rightmost noun-like non-terminal (so the head of *the red cat* is *cat*),[6] and prepositional phrases are headed by the preposition (the head of *at Georgia Tech* is *at*). Some of these rules are somewhat arbitrary — there's no particular reason why the head of *cats and dogs* should be *dogs* — but the point here is just to get some lexical information that can support parsing, not to make deep claims about syntax. Figure 10.6 shows the application of these rules to two of the running examples.

Parsing lexicalized context-free grammars

A naïve application of lexicalization would simply increase the set of non-terminals by taking the cross-product with the set of terminal symbols, so that the non-terminals now

[5]This is a pragmatic definition, befitting our goal of using head words to improve parsing; for a more formal definition, see (Bender, 2013, chapter 7).

[6]The noun phrase non-terminal is sometimes treated as a special case. Collins (1997) uses a heuristic that looks for the rightmost child which is a noun-like part-of-speech (e.g., NN, NNP), a possessive marker, or a superlative adjective (e.g., *the greatest*). If no such child is found, the heuristic then looks for the *leftmost* NP. If there is no child with tag NP, the heuristic then applies another priority list, this time from right to left.

include symbols like NP(*President*) and VP(*meet*). Under this approach, the CKY parsing algorithm could be applied directly to the lexicalized production rules. However, the complexity would be cubic in the size of the vocabulary of terminal symbols, which would clearly be intractable.

Another approach is to augment the CKY table with an additional index, keeping track of the head of each constituent. The cell $t[i, j, h, X]$ stores the score of the best derivation in which non-terminal X spans $\boldsymbol{w}_{i+1:j}$ with head word h, where $i < h \leq j$. To compute such a table recursively, we must consider the possibility that each phrase gets its head from either its left or right child. The scores of the best derivations in which the head comes from the left and right child are denoted t_ℓ and t_r respectively, leading to the following recurrence:

$$t_\ell[i, j, h, X] = \max_{(X \to Y Z)} \max_{k > h} \max_{k < h' \leq j} t[i, k, h, Y] + t[k, j, h', Z] + \psi(X(h) \to Y(h) Z(h')) \quad [10.40]$$

$$t_r[i, j, h, X] = \max_{(X \to Y Z)} \max_{k < h} \max_{i < h' \leq k} t[i, k, h', Y] + t[k, j, h, Z] + (\psi(X(h) \to Y(h') Z(h))) \quad [10.41]$$

$$t[i, j, h, X] = \max\left(t_\ell[i, j, h, X], t_r[i, j, h, X]\right). \quad [10.42]$$

To compute t_ℓ, we maximize over all split points $k > h$, since the head word must be in the left child. We then maximize again over possible head words h' for the right child. An analogous computation is performed for t_r. The size of the table is now $\mathcal{O}(M^3 N)$, where M is the length of the input and N is the number of non-terminals. Furthermore, each cell is computed by performing $\mathcal{O}(M^2)$ operations, since we maximize over both the split point k and the head h'. The time complexity of the algorithm is therefore $\mathcal{O}(RM^5 N)$, where R is the number of rules in the grammar. Fortunately, more efficient solutions are possible. In general, the complexity of parsing can be reduced to $\mathcal{O}(M^4)$ in the length of the input; for a broad class of lexicalized CFGs, the complexity can be made cubic in the length of the input, just as in unlexicalized CFGs (Eisner, 2000).

Estimating lexicalized context-free grammars

The final problem for lexicalized parsing is how to estimate weights for lexicalized productions $X(i) \to Y(j)\ Z(k)$. These productions are said to be bilexical, because they involve scores over pairs of words: in the example *meet the President of Mexico*, we hope to choose the correct attachment point by modeling the bilexical affinities of (*meet*, *of*) and (*President*, *of*). The number of such word pairs is quadratic in the size of the vocabulary, making it difficult to estimate the weights of lexicalized production rules directly from data. This is especially true for probabilistic context-free grammars, in which the weights are obtained from smoothed relative frequency. In a treebank with a million tokens, a vanishingly small fraction of the possible lexicalized productions will be observed more than once.[7] The Charniak (1997) and Collins (1997) parsers therefore focus on approximating the probabilities of lexicalized productions, using various smoothing techniques and independence assumptions.

In discriminatively-trained weighted context-free grammars, the scores for each production can be computed from a set of features, which can be made progressively more fine-grained (Finkel et al., 2008). For example, the score of the lexicalized production NP(*President*) → NP(*President*) PP(*of*) can be computed from the following features:

$$\begin{aligned}\boldsymbol{f}(\text{NP}(\textit{President}) \rightarrow \text{NP}(\textit{President})\ \text{PP}(\textit{of})) = \{&\text{NP}(*) \rightarrow \text{NP}(*)\ \text{PP}(*),\\ &\text{NP}(\textit{President}) \rightarrow \text{NP}(\textit{President})\ \text{PP}(*),\\ &\text{NP}(*) \rightarrow \text{NP}(*)\ \text{PP}(\textit{of}),\\ &\text{NP}(\textit{President}) \rightarrow \text{NP}(\textit{President})\ \text{PP}(\textit{of})\}\end{aligned}$$

The first feature scores the unlexicalized production NP → NP PP; the next two features lexicalize only one element of the production, thereby scoring the appropriateness of NP attachment for the individual words *President* and *of*; the final feature scores the specific bilexical affinity of *President* and *of*. For bilexical pairs that are encountered frequently in the treebank, this bilexical feature can play an important role in parsing; for pairs that are absent or rare, regularization will drive its weight to zero, forcing the parser to rely on the more coarse-grained features.

In chapter 14, we will encounter techniques for clustering words based on their **distributional** properties — the contexts in which they appear. Such a clustering would group rare and common words, such as *whale, shark, beluga, Leviathan*. Word clusters can be used as features in discriminative lexicalized parsing, striking a middle ground between full lexicalization and non-terminals (Finkel et al., 2008). In this way, labeled examples containing relatively common words like *whale* can help to improve parsing for rare words like *beluga*, as long as those two words are clustered together.

10.5.3 *Refinement grammars

Lexicalization improves on context-free parsing by adding detailed information in the form of lexical heads. However, estimating the scores of lexicalized productions is difficult. Klein and Manning (2003) argue that the right level of linguistic detail is somewhere between treebank categories and individual words. Some parts-of-speech and non-terminals are truly substitutable: for example, *cat*/N and *dog*/N. But others are not: for example, the preposition *of* exclusively attaches to nouns, while the preposition *as* is more

[7]The real situation is even more difficult, because non-binary context-free grammars can involve **trilexical** or higher-order dependencies, between the head of the constituent and multiple of its children (Carreras et al., 2008).

likely to modify verb phrases. Klein and Manning (2003) obtained a 2% improvement in F-MEASURE on a parent-annotated PCFG parser by making a single change: splitting the preposition category into six subtypes. They propose a series of linguistically-motivated refinements to the Penn Treebank annotations, which in total yielded a 40% error reduction.

Non-terminal refinement process can be automated by treating the refined categories as **latent variables**. For example, we might split the noun phrase non-terminal into NP1, NP2, NP3, . . . , without defining in advance what each refined non-terminal corresponds to. This can be treated as partially supervised learning, similar to the multi-component document classification model described in § 5.2.3. A latent variable PCFG can be estimated by expectation maximization (Matsuzaki et al., 2005):[8]

- In the E-step, estimate a marginal distribution q over the refinement type of each non-terminal in each derivation. These marginals are constrained by the original annotation: an NP can be reannotated as NP4, but not as VP3. Marginal probabilities over refined productions can be computed from the **inside-outside algorithm**, as described in § 10.4.3, where the E-step enforces the constraints imposed by the original annotations.
- In the M-step, recompute the parameters of the grammar, by summing over the probabilities of anchored productions that were computed in the E-step:

$$E[\text{count}(X \to Y\ Z)] = \sum_{i=0}^{M} \sum_{j=i}^{M} \sum_{k=i}^{j} \text{p}(X \to Y\ Z, (i,j,k) \mid \boldsymbol{w}). \qquad [10.43]$$

As usual, this process can be iterated to convergence. To determine the number of refinement types for each tag, Petrov et al. (2006) apply a split-merge heuristic; Liang et al. (2007) and Finkel et al. (2007) apply **Bayesian nonparametrics** (Cohen, 2016).

Some examples of refined non-terminals are shown in Table 10.4. The proper nouns differentiate months, first names, middle initials, last names, first names of places, and second names of places; each of these will tend to appear in different parts of grammatical productions. The personal pronouns differentiate grammatical role, with PRP-0 appearing in subject position at the beginning of the sentence (note the capitalization), PRP-1 appearing in subject position but not at the beginning of the sentence, and PRP-2 appearing in object position.

10.6 Beyond context-free parsing

In the context-free setting, the score for a parse is a combination of the scores of individual productions. As we have seen, these models can be improved by using finer-grained non-

[8]Spectral learning, described in § 5.5.2, has also been applied to refinement grammars (Cohen et al., 2014).

Proper nouns			
NNP-14	*Oct.*	*Nov.*	*Sept.*
NNP-12	*John*	*Robert*	*James*
NNP-2	*J.*	*E.*	*L.*
NNP-1	*Bush*	*Noriega*	*Peters*
NNP-15	*New*	*San*	*Wall*
NNP-3	*York*	*Francisco*	*Street*
Personal Pronouns			
PRP-0	*It*	*He*	*I*
PRP-1	*it*	*he*	*they*
PRP-2	*it*	*them*	*him*

Table 10.4: Examples of automatically refined non-terminals and some of the words that they generate (Petrov et al., 2006).

terminals, via parent-annotation, lexicalization, and automated refinement. However, the inherent limitations to the expressiveness of context-free parsing motivate the consideration of other search strategies. These strategies abandon the optimality guaranteed by bottom-up parsing, in exchange for the freedom to consider arbitrary properties of the proposed parses.

10.6.1 Reranking

A simple way to relax the restrictions of context-free parsing is to perform a two-stage process, in which a context-free parser generates a k-best list of candidates, and a **reranker** then selects the best parse from this list (Charniak and Johnson, 2005; Collins and Koo, 2005). The reranker can be trained from an objective that is similar to multi-class classification: the goal is to learn weights that assign a high score to the reference parse, or to the parse on the k-best list that has the lowest error. In either case, the reranker need only evaluate the K best parses, and so no context-free assumptions are necessary. This opens the door to more expressive scoring functions:

- It is possible to incorporate arbitrary non-local features, such as the structural parallelism and right-branching orientation of the parse (Charniak and Johnson, 2005).
- Reranking enables the use of **recursive neural networks**, in which each constituent span $\boldsymbol{w}_{i+1:j}$ receives a vector $\boldsymbol{u}_{i,j}$ which is computed from the vector representations of its children, using a composition function that is linked to the production

rule (Socher et al., 2013), e.g.,

$$\boldsymbol{u}_{i,j} = f\left(\Theta_{X \to Y\ Z} \begin{bmatrix} \boldsymbol{u}_{i,k} \\ \boldsymbol{u}_{k,j} \end{bmatrix}\right) \qquad [10.44]$$

The overall score of the parse can then be computed from the final vector, $\Psi(\tau) = \boldsymbol{\theta}\boldsymbol{u}_{0,M}$.

Reranking can yield substantial improvements in accuracy. The main limitation is that it can only find the best parse among the K-best offered by the generator, so it is inherently limited by the ability of the bottom-up parser to find high-quality candidates.

10.6.2 Transition-based parsing

Structure prediction can be viewed as a form of search. An alternative to bottom-up parsing is to read the input from left-to-right, gradually building up a parse structure through a series of **transitions**. Transition-based parsing is described in more detail in the next chapter, in the context of dependency parsing. However, it can also be applied to CFG parsing, as briefly described here.

For any context-free grammar, there is an equivalent **pushdown automaton**, a model of computation that accepts exactly those strings that can be derived from the grammar. This computational model consumes the input from left to right, while pushing and popping elements on a stack. This architecture provides a natural transition-based parsing framework for context-free grammars, known as **shift-reduce parsing**.

Shift-reduce parsing is a type of transition-based parsing, in which the parser can take the following actions:

- *shift* the next terminal symbol onto the stack;
- *unary-reduce* the top item on the stack, using a unary production rule in the grammar;
- *binary-reduce* the top two items onto the stack, using a binary production rule in the grammar.

The set of available actions is constrained by the situation: the parser can only shift if there are remaining terminal symbols in the input, and it can only reduce if an applicable production rule exists in the grammar. If the parser arrives at a state where the input has been completely consumed, and the stack contains only the element S, then the input is accepted. If the parser arrives at a non-accepting state where there are no possible actions, the input is rejected. A parse error occurs if there is some action sequence that would accept an input, but the parser does not find it.

Example Consider the input *we eat sushi* and the grammar in Table 10.1. The input can be parsed through the following sequence of actions:

1. **Shift** the first token *we* onto the stack.
2. **Reduce** the top item on the stack to NP, using the production NP $\rightarrow$ *we*.
3. **Shift** the next token *eat* onto the stack, and **reduce** it to V with the production V $\rightarrow$ *eat*.
4. **Shift** the final token *sushi* onto the stack, and **reduce** it to NP. The input has been completely consumed, and the stack contains $[\text{NP}, \text{V}, \text{NP}]$.
5. **Reduce** the top two items using the production VP $\rightarrow$ V NP. The stack now contains $[\text{VP}, \text{NP}]$.
6. **Reduce** the top two items using the production S $\rightarrow$ NP VP. The stack now contains [S]. Since the input is empty, this is an accepting state.

One thing to notice from this example is that the number of shift actions is equal to the length of the input. The number of reduce actions is equal to the number of non-terminals in the analysis, which grows linearly in the length of the input. Thus, the overall time complexity of shift-reduce parsing is linear in the length of the input (assuming the complexity of each individual classification decision is constant in the length of the input). This is far better than the cubic time complexity required by CKY parsing.

Transition-based parsing as inference In general, it is not possible to guarantee that a transition-based parser will find the optimal parse, $\text{argmax}_\tau \Psi(\tau; \boldsymbol{w})$, even under the usual CFG independence assumptions. We could assign a score to each anchored parsing action in each context, with $\psi(a, c)$ indicating the score of performing action a in context c. One might imagine that transition-based parsing could efficiently find the derivation that maximizes the sum of such scores. But this too would require backtracking and searching over an exponentially large number of possible action sequences: if a bad decision is made at the beginning of the derivation, then it may be impossible to recover the optimal action sequence without backtracking to that early mistake. This is known as a **search error**. Transition-based parsers can incorporate arbitrary features, without the restrictive independence assumptions required by chart parsing; search errors are the price that must be paid for this flexibility.

Learning transition-based parsing Transition-based parsing can be combined with machine learning by training a classifier to select the correct action in each situation. This classifier is free to choose any feature of the input, the state of the parser, and the parse history. However, there is no optimality guarantee: the parser may choose a suboptimal parse, due to a mistake at the beginning of the analysis. Nonetheless, some of the strongest

CFG parsers are based on the shift-reduce architecture, rather than CKY. A recent generation of models links shift-reduce parsing with recurrent neural networks, updating a hidden state vector while consuming the input (e.g., Cross and Huang, 2016; Dyer et al., 2016). Learning algorithms for transition-based parsing are discussed in more detail in § 11.3.

Exercises

1. Design a grammar that handles English subject-verb agreement. Specifically, your grammar should handle the examples below correctly:

   (10.4) a. She sings.
   b. We sing.

   (10.5) a. *She sing.
   b. *We sings.

2. Extend your grammar from the previous problem to include the auxiliary verb *can*, so that the following cases are handled:

   (10.6) a. She can sing.
   b. We can sing.

   (10.7) a. *She can sings.
   b. *We can sings.

3. French requires subjects and verbs to agree in person and number, and it requires determiners and nouns to agree in gender and number. Verbs and their objects need not agree. Assuming that French has two genders (feminine and masculine), three persons (first [*me*], second [*you*], third [*her*]), and two numbers (singular and plural), how many productions are required to extend the following simple grammar to handle agreement?

S	$\rightarrow$	NP VP
VP	$\rightarrow$	V | V NP | V NP NP
NP	$\rightarrow$	DET NN

4. Consider the grammar:

S	$\rightarrow$	NP VP
VP	$\rightarrow$	V NP
NP	$\rightarrow$	JJ NP
NP	$\rightarrow$	*fish* (the animal)
V	$\rightarrow$	*fish* (the action of fishing)
JJ	$\rightarrow$	*fish* (a modifier, as in *fish sauce* or *fish stew*)

Apply the CKY algorithm and identify all possible parses for the sentence *fish fish fish fish*.

5. Choose one of the possible parses for the previous problem, and show how it can be derived by a series of shift-reduce actions.

6. To handle VP coordination, a grammar includes the production VP $\rightarrow$ VP CC VP. To handle adverbs, it also includes the production VP $\rightarrow$ VP ADV. Assume all verbs are generated from a sequence of unary productions, e.g., VP $\rightarrow$ V $\rightarrow$ *eat*.

   a) Show how to binarize the production VP $\rightarrow$ VP CC VP.
   b) Use your binarized grammar to parse the sentence *They eat and drink together*, treating *together* as an adverb.
   c) Prove that a weighted CFG cannot distinguish the two possible derivations of this sentence. Your explanation should focus on the productions in the original, non-binary grammar.
   d) Explain what condition must hold for a parent-annotated WCFG to prefer the derivation in which *together* modifies the coordination *eat and drink*.

7. Consider the following PCFG:

$$p(X \rightarrow X\ X) = \frac{1}{2} \qquad [10.45]$$

$$p(X \rightarrow Y) = \frac{1}{2} \qquad [10.46]$$

$$p(Y \rightarrow \sigma) = \frac{1}{|\Sigma|}, \forall \sigma \in \Sigma \qquad [10.47]$$

   a) Compute the probability $p(\hat{\tau})$ of the maximum probability parse for a string $\boldsymbol{w} \in \Sigma^M$.
   b) Compute the conditional probability $p(\hat{\tau} \mid \boldsymbol{w})$.

8. Context-free grammars can be used to parse the internal structure of words. Using the weighted CKY algorithm and the following weighted context-free grammar, identify the best parse for the sequence of morphological segments *in+flame+able*.

S	$\rightarrow$	V	0
S	$\rightarrow$	N	0
S	$\rightarrow$	J	0
V	$\rightarrow$	VPref N	-1
J	$\rightarrow$	N JSuff	1
J	$\rightarrow$	V JSuff	0
J	$\rightarrow$	NegPref J	1
VPref	$\rightarrow$	*in+*	2
NegPref	$\rightarrow$	*in+*	1
N	$\rightarrow$	*flame*	0
JSuff	$\rightarrow$	*+able*	0

9. Use the inside and outside scores to compute the marginal probability $\text{p}(X_{i+1:j} \rightarrow Y_{i+1:k}\ Z_{k+1:j} \mid \boldsymbol{w})$, indicating that Y spans $\boldsymbol{w}_{i+1:k}$, Z spans $\boldsymbol{w}_{k+1:j}$, and X is the parent of Y and Z, spanning $\boldsymbol{w}_{i+1:j}$.

10. Suppose that the potentials $\Psi(X \rightarrow \alpha)$ are log-probabilities, so that $\sum_\alpha \exp \Psi(X \rightarrow \alpha) = 1$ for all X. Verify that the semiring inside recurrence from Equation 10.26 generates the log-probability $\log \text{p}(\boldsymbol{w}) = \log \sum_{\tau:\text{yield}(\tau)=\boldsymbol{w}} \text{p}(\tau)$.