Probabilistic Context-Free Grammars (PCFGs)

Michael Collins

1 Context-Free Grammars

1.1 Basic Definition

A context-free grammar (CFG) is a 4-tuple $G = (N, \Sigma, R, S)$ where:

- N is a finite set of non-terminal symbols.
- Σ is a finite set of terminal symbols.
- R is a finite set of rules of the form $X \rightarrow Y_1 Y_2 \ldots Y_n$, where $X \in N$, $n \geq 0$, and $Y_i \in (N \cup \Sigma)$ for $i = 1 \ldots n$.
- $S \in N$ is a distinguished start symbol.

Figure 1 shows a very simple context-free grammar, for a fragment of English. In this case the set of non-terminals N specifies some basic syntactic categories: for example S stands for "sentence", NP for "noun phrase", VP for "verb phrase", and so on. The set Σ contains the set of words in the vocabulary. The start symbol in this grammar is S: as we will see, this specifies that every parse tree has S as its root. Finally, we have context-free rules such as

$$\texttt{S} \rightarrow \texttt{NP VP}$$

or

$$\texttt{NN} \rightarrow \texttt{man}$$

The first rule specifies that an S (sentence) can be composed of an NP followed by a VP. The second rule specifies that an NN (a singular noun) can be composed of the word man.

Note that the set of allowable rules, as defined above, is quite broad: we can have any rule $X \rightarrow Y_1 \ldots Y_n$ as long as X is a member of N, and each Y_i for

$i = 1 \ldots n$ is a member of either N or Σ. We can for example have "unary rules", where $n = 1$, such as the following:

$$\begin{array}{rcl} \texttt{NN} & \rightarrow & \texttt{man} \\ \texttt{S} & \rightarrow & \texttt{VP} \end{array}$$

We can also have rules that have a mixture of terminal and non-terminal symbols on the right-hand-side of the rule, for example

$$\begin{array}{rcl} \texttt{VP} & \rightarrow & \texttt{John Vt Mary} \\ \texttt{NP} & \rightarrow & \texttt{the NN} \end{array}$$

We can even have rules where $n = 0$, so that there are no symbols on the right-hand-side of the rule. Examples are

$$\begin{array}{rcl} \texttt{VP} & \rightarrow & \epsilon \\ \texttt{NP} & \rightarrow & \epsilon \end{array}$$

Here we use ϵ to refer to the empty string. Intuitively, these latter rules specify that a particular non-terminal (e.g., VP), is allowed to have no words below it in a parse tree.

1.2 (Left-most) Derivations

Given a context-free grammar G, a left-most derivation is a sequence of strings $s_1 \ldots s_n$ where

- $s_1 = \texttt{S}$. i.e., s_1 consists of a single element, the start symbol.
- $s_n \in \Sigma^*$, i.e. s_n is made up of terminal symbols only (we write Σ^* to denote the set of all possible strings made up of sequences of words taken from Σ.)
- Each s_i for $i = 2 \ldots n$ is derived from s_{i-1} by picking the left-most non-terminal X in s_{i-1} and replacing it by some β where $X \rightarrow \beta$ is a rule in R.

As one example, one left-most derivation under the grammar in figure 1 is the following:

- $s_1 = \texttt{S}$.
- $s_2 = \texttt{NP VP}$. (We have taken the left-most non-terminal in s_1, namely S, and chosen the rule $\texttt{S} \rightarrow \texttt{NP VP}$, thereby replacing S by NP followed by VP.)

N = {S, NP, VP, PP, DT, Vi, Vt, NN, IN}
S = S
Σ = {sleeps, saw, man, woman, dog, telescope, the, with, in}

$R =$

S	→	NP	VP
VP	→	Vi	
VP	→	Vt	NP
VP	→	VP	PP
NP	→	DT	NN
NP	→	NP	PP
PP	→	IN	NP

Vi	→	sleeps
Vt	→	saw
NN	→	man
NN	→	woman
NN	→	telescope
NN	→	dog
DT	→	the
IN	→	with
IN	→	in

Figure 1: A simple context-free grammar. Note that the set of non-terminals N contains a basic set of syntactic categories: S=sentence, VP=verb phrase, NP=noun phrase, PP=prepositional phrase, DT=determiner, Vi=intransitive verb, Vt=transitive verb, NN=noun, IN=preposition. The set Σ is the set of possible words in the language.

- $s_3 =$ `DT NN VP`. (We have used the rule `NP` → `DT NN` to expand the left-most non-terminal, namely `NP`.)
- $s_4 =$ `the NN VP`. (We have used the rule `DT` → `the`.)
- $s_5 =$ `the man VP`. (We have used the rule `NN` → `man`.)
- $s_6 =$ `the man Vi`. (We have used the rule `VP` → `Vi`.)
- $s_7 =$ `the man sleeps`. (We have used the rule `Vi` → `sleeps`.)

It is very convenient to represent derivations as *parse trees*. For example, the above derivation would be represented as the parse tree shown in figure 2. This parse tree has `S` as its root, reflecting the fact that $s_1 =$ `S`. We see the sequence `NP VP` directly below `S`, reflecting the fact that the `S` was expanded using the rule `S` → `NP VP`; we see the sequence `DT NN` directly below the `NP`, reflecting the fact that the `NP` was expanded using the rule `NP` → `DT NN`; and so on.

A context-free grammar G will in general specify a set of possible left-most derivations. Each left-most derivation will end in a string $s_n \in \Sigma^*$: we say that s_n

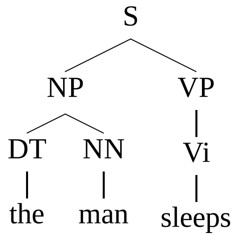


Figure 2: A derivation can be represented as a parse tree.

is the *yield* of the derivation. The set of possible derivations may be a finite or an infinite set (in fact the set of derivations for the grammar in figure 1 is infinite).

The following definition is crucial:

- A string $s \in \Sigma^*$ is said to be in the *language* defined by the CFG, if there is at least one derivation whose yield is s.

2 Ambiguity

Note that some strings s may have more than one underlying derivation (i.e., more than one derivation with s as the yield). In this case we say that the string is *ambiguous* under the CFG.

As one example, see figure 3, which gives two parse trees for the string *the man saw the dog with the telescope*, both of which are valid under the CFG given in figure 1. This example is a case of prepositional phrase attachment ambiguity: the prepositional phrase (PP) *with the telescope* can modify either *the dog*, or *saw the dog*. In the first parse tree shown in the figure, the PP modifies *the dog*, leading to an NP *the dog with the telescope*: this parse tree corresponds to an interpretation where the dog is holding the telescope. In the second parse tree, the PP modifies the entire VP *saw the dog*: this parse tree corresponds to an interpretation where the man is using the telescope to see the dog.

Ambiguity is an astonishingly severe problem for natural languages. When researchers first started building reasonably large grammars for languages such as English, they were surprised to see that sentences often had a very large number of possible parse trees: it is not uncommon for a moderate-length sentence (say 20 or 30 words in length) to have hundreds, thousands, or even tens of thousands of possible parses.

As one example, in lecture we argued that the following sentence has a surprisingly large number of parse trees (I've found 14 in total):

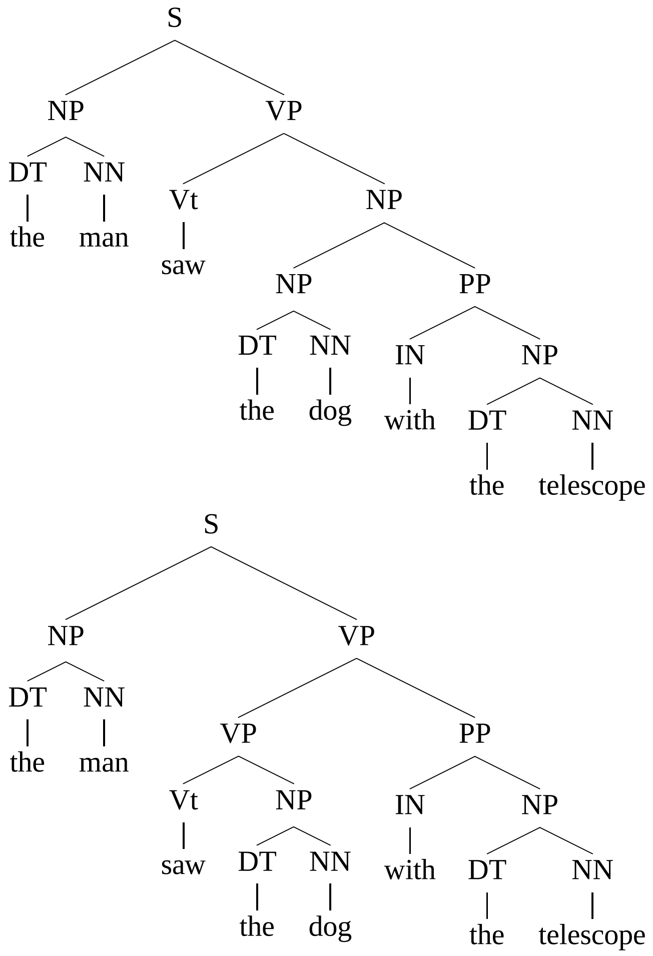


Figure 3: Two parse trees (derivations) for the sentence *the man saw the dog with the telescope*, under the CFG in figure 1.

She announced a program to promote safety in trucks and vans

Can you find the different parse trees for this example?

3 Probabilistic Context-Free Grammars (PCFGs)

3.1 Basic Definitions

Given a context-free grammar G, we will use the following definitions:

- $\mathcal{T}_G$ is the set of all possible left-most derivations (parse trees) under the grammar G. When the grammar G is clear from context we will often write this as simply $\mathcal{T}$.

- For any derivation $t \in \mathcal{T}_G$, we write $\textsf{yield}(t)$ to denote the string $s \in \Sigma^*$ that is the yield of t (i.e., $\textsf{yield}(t)$ is the sequence of words in t).

- For a given sentence $s \in \Sigma^*$, we write $\mathcal{T}_G(s)$ to refer to the set

  $$\{t : t \in \mathcal{T}_G, \textsf{yield}(t) = s\}$$

  That is, $\mathcal{T}_G(s)$ is the set of possible parse trees for s.

- We say that a sentence s is *ambiguous* if it has more than one parse tree, i.e., $|\mathcal{T}_G(s)| > 1$.

- We say that a sentence s is *grammatical* if it has at least one parse tree, i.e., $|\mathcal{T}_G(s)| > 0$.

The key idea in probabilistic context-free grammars is to extend our definition to give a *probability distribution over possible derivations*. That is, we will find a way to define a distribution over parse trees, $p(t)$, such that for any $t \in \mathcal{T}_G$,

$$p(t) \geq 0$$

and in addition such that

$$\sum_{t \in \mathcal{T}_G} p(t) = 1$$

At first glance this seems difficult: each parse-tree t is a complex structure, and the set $\mathcal{T}_G$ will most likely be infinite. However, we will see that there is a very simple extension to context-free grammars that allows us to define a function $p(t)$.

Why is this a useful problem? A crucial idea is that once we have a function $p(t)$, we have a ranking over possible parses for any sentence in order of probability. In particular, given a sentence s, we can return

$$\arg\max_{t \in \mathcal{T}_G(s)} p(t)$$

as the output from our parser—this is the most likely parse tree for s under the model. Thus if our distribution $p(t)$ is a good model for the probability of different parse trees in our language, we will have an effective way of dealing with ambiguity.

This leaves us with the following questions:

- How do we define the function $p(t)$?
- How do we learn the parameters of our model of $p(t)$ from training examples?
- For a given sentence s, how do we find the most likely tree, namely

  $$\arg\max_{t \in \mathcal{T}_G(s)} p(t)?$$

  This last problem will be referred to as the *decoding* or *parsing* problem.

In the following sections we answer these questions through defining *probabilistic context-free grammars* (PCFGs), a natural generalization of context-free grammars.

3.2 Definition of PCFGs

Probabilistic context-free grammars (PCFGs) are defined as follows:

Definition 1 (PCFGs) *A PCFG consists of:*

1. *A context-free grammar $G = (N, \Sigma, S, R)$.*
2. *A parameter*

   $$q(\alpha \rightarrow \beta)$$

   for each rule $\alpha \rightarrow \beta \in R$. The parameter $q(\alpha \rightarrow \beta)$ can be interpreted as the conditional probabilty of choosing rule $\alpha \rightarrow \beta$ in a left-most derivation, given that the non-terminal being expanded is α. For any $X \in N$, we have the constraint

   $$\sum_{\alpha \rightarrow \beta \in R : \alpha = X} q(\alpha \rightarrow \beta) = 1$$

   In addition we have $q(\alpha \rightarrow \beta) \geq 0$ for any $\alpha \rightarrow \beta \in R$.

Given a parse-tree $t \in \mathcal{T}_G$ containing rules $\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \ldots, \alpha_n \rightarrow \beta_n$, the probability of t under the PCFG is

$$p(t) = \prod_{i=1}^{n} q(\alpha_i \rightarrow \beta_i)$$

□

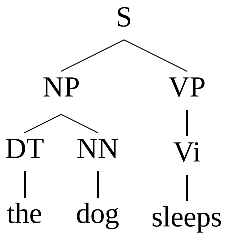
Figure 4 shows an example PCFG, which has the same underlying context-free grammar as that shown in figure 1. The only addition to the original context-free grammar is a parameter $q(\alpha \rightarrow \beta)$ for each rule $\alpha \rightarrow \beta \in R$. Each of these parameters is constrained to be non-negative, and in addition we have the constraint that for any non-terminal $X \in N$,

$$\sum_{\alpha \rightarrow \beta \in R : \alpha = X} q(\alpha \rightarrow \beta) = 1$$

This simply states that for any non-terminal X, the parameter values for all rules with that non-terminal on the left-hand-side of the rule must sum to one. We can verify that this property holds for the PCFG in figure 4. For example, we can verify that this constraint holds for $X = \texttt{VP}$, because

$$\begin{aligned}\sum_{\alpha \rightarrow \beta \in R : \alpha = \texttt{VP}} q(\alpha \rightarrow \beta) &= q(\texttt{VP} \rightarrow \texttt{Vi}) + q(\texttt{VP} \rightarrow \texttt{Vt NP}) + q(\texttt{VP} \rightarrow \texttt{VP PP}) \\ &= 0.3 + 0.5 + 0.2 \\ &= 1.0\end{aligned}$$

To calculate the probability of any parse tree t, we simply multiply together the q values for the context-free rules that it contains. For example, if our parse tree t is



then we have

$$\begin{aligned}p(t) \;=\; & q(\texttt{S} \rightarrow \texttt{NP VP}) \times q(\texttt{NP} \rightarrow \texttt{DT NN}) \times q(\texttt{DT} \rightarrow \texttt{the}) \times q(\texttt{NN} \rightarrow \texttt{dog}) \times \\ & q(\texttt{VP} \rightarrow \texttt{Vi}) \times q(\texttt{Vi} \rightarrow \texttt{sleeps})\end{aligned}$$

Intuitively, PCFGs make the assumption that parse trees are generated stochastically, according to the following process:

N = {S, NP, VP, PP, DT, Vi, Vt, NN, IN}
S = S
Σ = {sleeps, saw, man, woman, dog, telescope, the, with, in}

$R, q =$

S	→	NP	VP	1.0
VP	→	Vi		0.3
VP	→	Vt	NP	0.5
VP	→	VP	PP	0.2
NP	→	DT	NN	0.8
NP	→	NP	PP	0.2
PP	→	IN	NP	1.0

Vi	→	sleeps	1.0
Vt	→	saw	1.0
NN	→	man	0.1
NN	→	woman	0.1
NN	→	telescope	0.3
NN	→	dog	0.5
DT	→	the	1.0
IN	→	with	0.6
IN	→	in	0.4

Figure 4: A simple probabilistic context-free grammar (PCFG). In addition to the set of rules R, we show the parameter value for each rule. For example, $q(\texttt{VP} \rightarrow \texttt{Vt NP}) = 0.5$ in this PCFG.

- Define $s_1 = S, i = 1$.
- While s_i contains at least one non-terminal:
  - Find the left-most non-terminal in s_i, call this X.
  - Choose one of the rules of the form $X \rightarrow \beta$ from the distribution $q(X \rightarrow \beta)$.
  - Create s_{i+1} by replacing the left-most X in s_i by β.
  - Set $i = i + 1$.

So we have simply added probabilities to each step in left-most derivations. The probability of an entire tree is the product of probabilities for these individual choices.

3.3 Deriving a PCFG from a Corpus

Having defined PCFGs, the next question is the following: how do we derive a PCFG from a corpus? We will assume a set of training data, which is simply a set of parse trees $t_1, t_2, \ldots, t_m$. As before, we will write $\mathsf{yield}(t_i)$ to be the yield for the i'th parse tree in the sentence, i.e., $\mathsf{yield}(t_i)$ is the i'th sentence in the corpus.

Each parse tree t_i is a sequence of context-free rules: we assume that every parse tree in our corpus has the same symbol, S, at its root. We can then define a PCFG (N, Σ, S, R, q) as follows:

- N is the set of all non-terminals seen in the trees $t_1 \ldots t_m$.
- Σ is the set of all words seen in the trees $t_1 \ldots t_m$.
- The start symbol S is taken to be S.
- The set of rules R is taken to be the set of all rules $\alpha \rightarrow \beta$ seen in the trees $t_1 \ldots t_m$.
- The maximum-likelihood parameter estimates are

  $$q_{ML}(\alpha \rightarrow \beta) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

  where Count$(\alpha \rightarrow \beta)$ is the number of times that the rule $\alpha \rightarrow \beta$ is seen in the trees $t_1 \ldots t_m$, and Count(α) is the number of times the non-terminal α is seen in the trees $t_1 \ldots t_m$.

For example, if the rule $\texttt{VP} \rightarrow \texttt{Vt NP}$ is seen 105 times in our corpus, and the non-terminal $\texttt{VP}$ is seen 1000 times, then

$$q(\texttt{VP} \rightarrow \texttt{Vt NP}) = \frac{105}{1000}$$

3.4 Parsing with PCFGs

A crucial question is the following: given a sentence s, how do we find the highest scoring parse tree for s, or more explicitly, how do we find

$$\arg\max_{t \in \mathcal{T}(s)} p(t) \quad ?$$

This section describes a dynamic programming algorithm, *the CKY algorithm*, for this problem.

The CKY algorithm we present applies to a restricted type of PCFG: a PCFG where which is in Chomsky normal form (CNF). While the restriction to grammars in CNF might at first seem to be restrictive, it turns out not to be a strong assumption. It is possible to convert any PCFG into an equivalent grammar in CNF: we will look at this question more in the homeworks.

In the next sections we first describe the idea of grammars in CNF, then describe the CKY algorithm.

N = {S, NP, VP, PP, DT, Vi, Vt, NN, IN}
S = S
Σ = {sleeps, saw, man, woman, dog, telescope, the, with, in}

$R, q =$

S	$\rightarrow$	NP	VP	1.0
VP	$\rightarrow$	Vt	NP	0.8
VP	$\rightarrow$	VP	PP	0.2
NP	$\rightarrow$	DT	NN	0.8
NP	$\rightarrow$	NP	PP	0.2
PP	$\rightarrow$	IN	NP	1.0

Vi	$\rightarrow$	sleeps	1.0
Vt	$\rightarrow$	saw	1.0
NN	$\rightarrow$	man	0.1
NN	$\rightarrow$	woman	0.1
NN	$\rightarrow$	telescope	0.3
NN	$\rightarrow$	dog	0.5
DT	$\rightarrow$	the	1.0
IN	$\rightarrow$	with	0.6
IN	$\rightarrow$	in	0.4

Figure 5: A simple probabilistic context-free grammar (PCFG) in Chomsky normal form. Note that each rule in the grammar takes one of two forms: $X \rightarrow Y_1 \quad Y_2$ where $X \in N, Y_1 \in N, Y_2 \in N$; or $X \rightarrow Y$ where $X \in N$, $Y \in \Sigma$.

3.4.1 Chomsky Normal Form

Definition 2 (Chomsky Normal Form) *A context-free grammar $G = (N, \Sigma, R, S)$ is in Chomsky form if each rule $\alpha \rightarrow \beta \in R$ takes one of the two following forms:*

- *$X \rightarrow Y_1 Y_2$ where $X \in N, Y_1 \in N, Y_2 \in N$.*
- *$X \rightarrow Y$ where $X \in N$, $Y \in \Sigma$.*

Hence each rule in the grammar either consists of a non-terminal X rewriting as exactly two non-terminal symbols, $Y_1 Y_2$; or a non-terminal X rewriting as exactly one terminal symbol Y. □

Figure 5 shows an example of a PCFG in Chomsky normal form.

3.4.2 Parsing using the CKY Algorithm

We now describe an algorithm for parsing with a PCFG in CNF. The input to the algorithm is a PCFG $G = (N, \Sigma, S, R, q)$ in Chomsky normal form, and a sentence

$s = x_1 \ldots x_n$, where x_i is the i'th word in the sentence. The output of the algorithm is

$$\arg \max_{t \in \mathcal{T}_G(s)} p(t)$$

The CKY algorithm is a dynamic-programming algorithm. Key definitions in the algorithm are as follows:

- For a given sentence $x_1 \ldots x_n$, define $\mathcal{T}(i, j, X)$ for any $X \in N$, for any (i, j) such that $1 \leq i \leq j \leq n$, to be the set of all parse trees for words $x_i \ldots x_j$ such that non-terminal X is at the root of the tree.
- Define
  $$\pi(i, j, X) = \max_{t \in \mathcal{T}(i,j,X)} p(t)$$
  (we define $\pi(i, j, X) = 0$ if $\mathcal{T}(i, j, X)$ is the empty set).

Thus $\pi(i, j, X)$ is the highest score for any parse tree that dominates words $x_i \ldots x_j$, and has non-terminal X as its root. The score for a tree t is again taken to be the product of scores for the rules that it contains (i.e. if the tree t contains rules $\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \ldots, \alpha_m \rightarrow \beta_m$, then $p(t) = \prod_{i=1}^m q(\alpha_i \rightarrow \beta_i)$).

Note in particular, that

$$\pi(1, n, S) = \arg \max_{t \in \mathcal{T}_G(s)}$$

because by definition $\pi(1, n, S)$ is the score for the highest probability parse tree spanning words $x_1 \ldots x_n$, with S as its root.

The key observation in the CKY algorithm is that we can use a recursive definition of the π values, which allows a simple bottom-up dynamic programming algorithm. The algorithm is "bottom-up", in the sense that it will first fill in $\pi(i, j, X)$ values for the cases where $j = i$, then the cases where $j = i + 1$, and so on.

The base case in the recursive definition is as follows: for all $i = 1 \ldots n$, for all $X \in N$,

$$\pi(i, i, X) = \begin{cases} q(X \rightarrow x_i) & \text{if } X \rightarrow x_i \in R \\ 0 & \text{otherwise} \end{cases}$$

This is a natural definition: the only way that we can have a tree rooted in node X spanning word x_i is if the rule $X \rightarrow x_i$ is in the grammar, in which case the tree has score $q(X \rightarrow x_i)$; otherwise, we set $\pi(i, i, X) = 0$, reflecting the fact that there are no trees rooted in X spanning word x_i.

The recursive definition is as follows: for all (i, j) such that $1 \leq i < j \leq n$, for all $X \in N$,

$$\pi(i, j, X) = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \ldots (j-1)\}}} \left(q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z) \right) \quad (1)$$

The next section of this note gives justification for this recursive definition.

Figure 6 shows the final algorithm, based on these recursive definitions. The algorithm fills in the π values bottom-up: first the $\pi(i, i, X)$ values, using the base case in the recursion; then the values for $\pi(i, j, X)$ such that $j = i + 1$; then the values for $\pi(i, j, X)$ such that $j = i + 2$; and so on.

Note that the algorithm also stores *backpointer* values $bp(i, j, X)$ for all values of (i, j, X). These values record the rule $X \rightarrow YZ$ and the split-point s leading to the highest scoring parse tree. The backpointer values allow recovery of the highest scoring parse tree for the sentence.

3.4.3 Justification for the Algorithm

As an example of how the recursive rule in Eq. 2 is applied, consider parsing the sentence

$x_1 \ldots x_8 =$ *the dog saw the man with the telescope*

and consider the calculation of $\pi(3, 8, \texttt{VP})$. This will be the highest score for any tree with root VP, spanning words $x_3 \ldots x_8 =$ *saw the man with the telescope*. Eq. 2 specifies that to calculate this value we take the max over two choices: first, a choice of a rule $\texttt{VP} \rightarrow YZ$ which is in the set of rules R—note that there are two such rules, VP → Vt NP and VP → VP PP. Second, a choice of $s \in \{3, 4, \ldots 7\}$. Thus we will take the maximum value of the following terms:

$$\begin{array}{r}
q(\texttt{VP} \rightarrow \texttt{Vt NP}) \times \pi(3, 3, \texttt{Vt}) \times \pi(4, 8, \texttt{NP}) \\
q(\texttt{VP} \rightarrow \texttt{VP PP}) \times \pi(3, 3, \texttt{VP}) \times \pi(4, 8, \texttt{PP}) \\
q(\texttt{VP} \rightarrow \texttt{Vt NP}) \times \pi(3, 4, \texttt{Vt}) \times \pi(5, 8, \texttt{NP}) \\
q(\texttt{VP} \rightarrow \texttt{VP PP}) \times \pi(3, 4, \texttt{VP}) \times \pi(5, 8, \texttt{PP}) \\
q(\texttt{VP} \rightarrow \texttt{Vt NP}) \times \pi(3, 5, \texttt{Vt}) \times \pi(6, 8, \texttt{NP}) \\
q(\texttt{VP} \rightarrow \texttt{VP PP}) \times \pi(3, 5, \texttt{VP}) \times \pi(6, 8, \texttt{PP}) \\
\ldots \\
q(\texttt{VP} \rightarrow \texttt{Vt NP}) \times \pi(3, 7, \texttt{Vt}) \times \pi(8, 8, \texttt{NP}) \\
q(\texttt{VP} \rightarrow \texttt{VP PP}) \times \pi(3, 7, \texttt{VP}) \times \pi(8, 8, \texttt{PP})
\end{array}$$

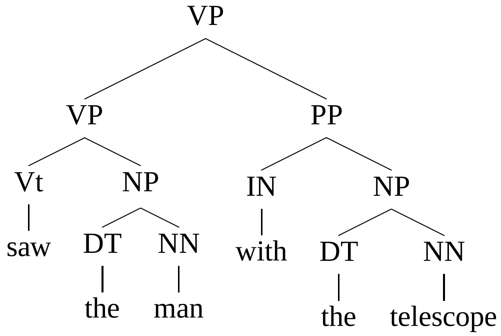
```
Input: a sentence $s = x_1 \ldots x_n$, a PCFG $G = (N, \Sigma, S, R, q)$.
Initialization:
For all $i \in \{1 \ldots n\}$, for all $X \in N$,
    $\pi(i, i, X) = \begin{cases} q(X \rightarrow x_i) & \text{if } X \rightarrow x_i \in R \\ 0 & \text{otherwise} \end{cases}$
Algorithm:
  • For $l = 1 \ldots (n - 1)$
      – For $i = 1 \ldots (n - l)$
          ∗ Set $j = i + l$
          ∗ For all $X \in N$, calculate
              $\pi(i, j, X) = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \ldots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$
            and
              $bp(i, j, X) = \arg \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \ldots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$
Output: Return $\pi(1, n, S) = \max_{t \in \mathcal{T}(s)} p(t)$, and backpointers $bp$ which allow recovery of $\arg\max_{t \in \mathcal{T}(s)} p(t)$.
```

Figure 6: The CKY parsing algorithm.

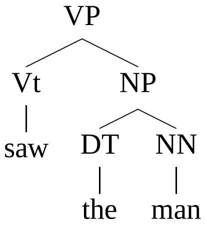
How do we justify this recursive definition? The key observation is that any tree t rooted in X, spanning words $x_i \ldots x_j$, must consist of the following:

- A choice of some rule $X \rightarrow Y\ Z \in R$, at the top of the tree.
- A choice of some value $s \in \{i \ldots j-1\}$, which we will refer to as the "split point" of the rule.
- A choice of a tree rooted in Y, spanning words $x_i \ldots x_s$, call this tree t_1
- A choice of a tree rooted in Z, spanning words $x_{s+1} \ldots x_j$, call this tree t_2.
- We then have
$$p(t) = q(X \rightarrow Y\ Z) \times p(t_1) \times p(t_2)$$
I.e., the probability for the tree t is the product of three terms: the rule probability for the rule at the top of the tree, and probabilities for the sub-trees t_1 and t_2.

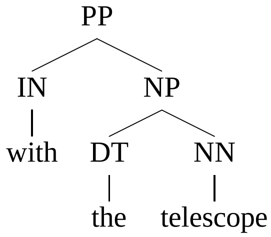
For example, consider the following tree, rooted in `VP`, spanning words $x_3 \ldots x_8$ in our previous example:



In this case we have the rule `VP` → `VP PP` at the top of the tree; the choice of split-point is $s = 5$; the tree dominating words $x_3 \ldots x_s$, rooted in `VP`, is



and the tree dominating words $x_{s+1} \ldots x_8$, rooted in `PP`, is



The second key observation is the following:

- *If the highest scoring tree rooted in non-terminal X, and spanning words $x_i \dots x_j$, uses rule $X \rightarrow Y \;\; Z$ and split point s, then its two subtrees must be: 1) the highest scoring tree rooted in Y that spanns words $x_i \dots x_s$; 2) the highest scoring tree rooted in Z that spans words $x_{s+1} \dots x_j$.*

The proof is by contradiction. If either condition (1) or condition (2) was not true, we could always find a higher scoring tree rooted in X, spanning words $x_i \dots x_j$, by choosing a higher scoring subtree spanning words $x_i \dots x_s$ or $x_{s+1} \dots x_j$.

Now let's look back at our recursive definition:

$$\pi(i, j, X) = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$$

We see that it involves a search over rules possible rules $X \rightarrow YZ \in R$, and possible split points s. For each choice of rule and split point, we calculate

$$q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z)$$

which is the highest scoring tree rooted in X, spanning words $x_i \dots x_j$, with this choice of rule and split point. The definition uses the values $\pi(i, s, Y)$ and $\pi(s + 1, j, Z)$, corresponding to the two highest scoring subtrees. We take the $\max$ over all possible choices of rules and split points.

3.4.4 The Inside Algorithm for Summing over Trees

We now describe a second, very similar algorithm, which sums the probabilities for all parse trees for a given sentence, thereby calculating the probability of the sentence under the PCFG. The algorithm is called *the inside algorithm*.

The input to the algorithm is again a PCFG $G = (N, \Sigma, S, R, q)$ in Chomsky normal form, and a sentence $s = x_1 \dots x_n$, where x_i is the i'th word in the sentence. The output of the algorithm is

$$p(s) = \sum_{t \in \mathcal{T}_G(s)} p(t)$$

Here $p(s)$ is the probability of the PCFG generating string s.

We define the following:

- As before, for a given sentence $x_1 \ldots x_n$, define $\mathcal{T}(i, j, X)$ for any $X \in N$, for any (i, j) such that $1 \leq i \leq j \leq n$, to be the set of all parse trees for words $x_i \ldots x_j$ such that non-terminal X is at the root of the tree.
- Define
$$\pi(i, j, X) = \sum_{t \in \mathcal{T}(i,j,X)} p(t)$$
(we define $\pi(i, j, X) = 0$ if $\mathcal{T}(i, j, X)$ is the empty set).

Note that we have simply replaced the $\max$ in the previous definition of π, with a sum.

In particular, we have

$$\pi(1, n, S) = \sum_{t \in \mathcal{T}_G(s)} p(t) = p(s)$$

Thus by calculating $\pi(1, n, S)$, we have calculated the probability $p(s)$.

We use a very similar recursive definition to before. First, the base case is as follows: for all $i = 1 \ldots n$, for all $X \in N$,

$$\pi(i, i, X) \quad = \quad \begin{cases} q(X \rightarrow x_i) & \text{if } X \rightarrow x_i \in R \\ 0 & \text{otherwise} \end{cases}$$

The recursive definition is as follows: for all (i, j) such that $1 \leq i < j \leq n$, for all $X \in N$,

$$\pi(i, j, X) = \sum_{\substack{X \rightarrow Y Z \in R, \\ s \in \{i \ldots (j-1)\}}} (q(X \rightarrow Y Z) \times \pi(i, s, Y) \times \pi(s + 1, j, Z)) \qquad (2)$$

Figure 7 shows the algorithm based on these recursive definitions. The algorithm is essentially identical to the CKY algorithm, but with $\max$ replaced by a sum in the recursive definition. The π values are again calculated bottom-up.

```
Input: a sentence $s = x_1 \ldots x_n$, a PCFG $G = (N, \Sigma, S, R, q)$.
Initialization:
For all $i \in \{1 \ldots n\}$, for all $X \in N$,

    $\pi(i, i, X) = \begin{cases} q(X \rightarrow x_i) & \text{if } X \rightarrow x_i \in R \\ 0 & \text{otherwise} \end{cases}$

Algorithm:

  • For $l = 1 \ldots (n-1)$
      – For $i = 1 \ldots (n-l)$
          ∗ Set $j = i + l$
          ∗ For all $X \in N$, calculate

              $\pi(i, j, X) = \sum_{\substack{X \rightarrow Y Z \in R, \\ s \in \{i \ldots (j-1)\}}} (q(X \rightarrow Y Z) \times \pi(i, s, Y) \times \pi(s+1, j, Z))$

Output: Return $\pi(1, n, S) = \sum_{t \in \mathcal{T}(s)} p(t)$
```

Figure 7: The inside algorithm.