Statistical Parsing

Hyopil Shin

SNU 4th Industrial Revolution Academy: Artificial Intelligence Agent

기본 아이디어
cfg(context free grammer)의 rule도 확률로 assign해보자.
이걸 바탕으로 문장의 parse tree를 완성해보자.

Introduction

- Why Probabilistic Parsing?
  - Disambiguation – coordination ambiguity/attachment ambiguity
    - Compute the probability of each interpretation and choose the most-probable interpretation
  - Language modeling for speech recognition
    - N-grams are used in speech recognizers to predict upcoming words
    - Probabilistic versions of more sophisticated grammars can provide additional predictive power to a speech recognizer
- Probabilistic Context-free Grammar – a probabilistic augmentation of context-free grammars in which each rule is associated with a probability.

Probabilistic Context-Free Grammars

- Also known as the Stochastic Context-Free Grammar

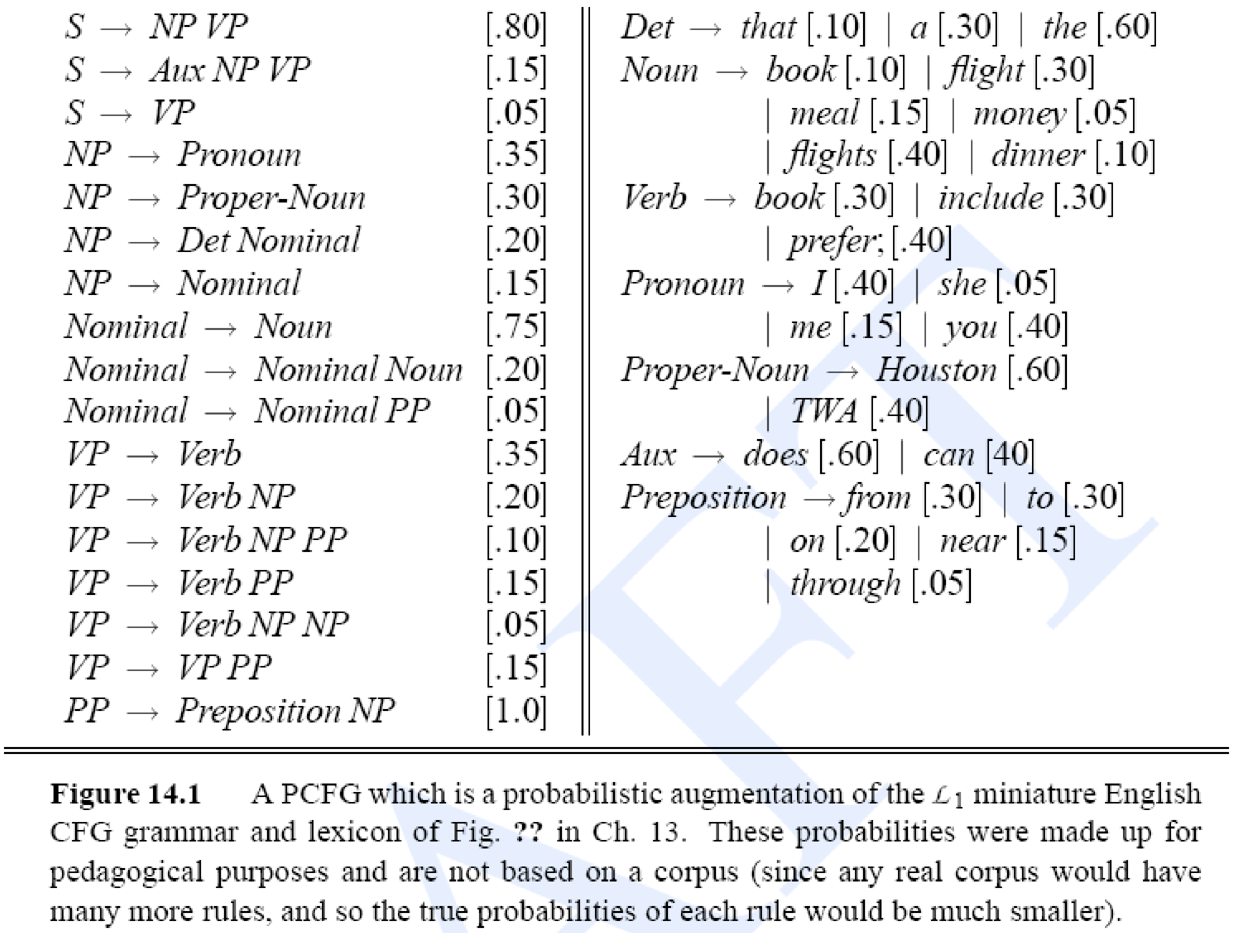
- A PCFG *G* has five parameters
  1. A set of non-terminal symbols (or "variables") *N* syntactic categories lexical categories
  2. A set of terminal symbols $\sum$ (disjoint from *N*) words
  3. A set of productions *P*, each of the form $A \rightarrow \beta$, where *A* is a non-terminal symbol and β is a string of symbols from the infinite set of strings $(\sum \cup N)^*$
  4. A designated start symbol *S* (or N^1)
  5. Each rule in *P* is augmented with a conditional probability assigned by a function *D*

     $A \rightarrow \beta$ [*prob.*]

     $P(A \rightarrow \beta)$ or $P(A \rightarrow \beta | A)$ $\Longrightarrow$ $\forall A \quad \sum_{\beta} P(A \rightarrow \beta) = 1$

- A PCFG $G = (N, \sum, P, S, D)$

Probabilistic Context-Free Grammars



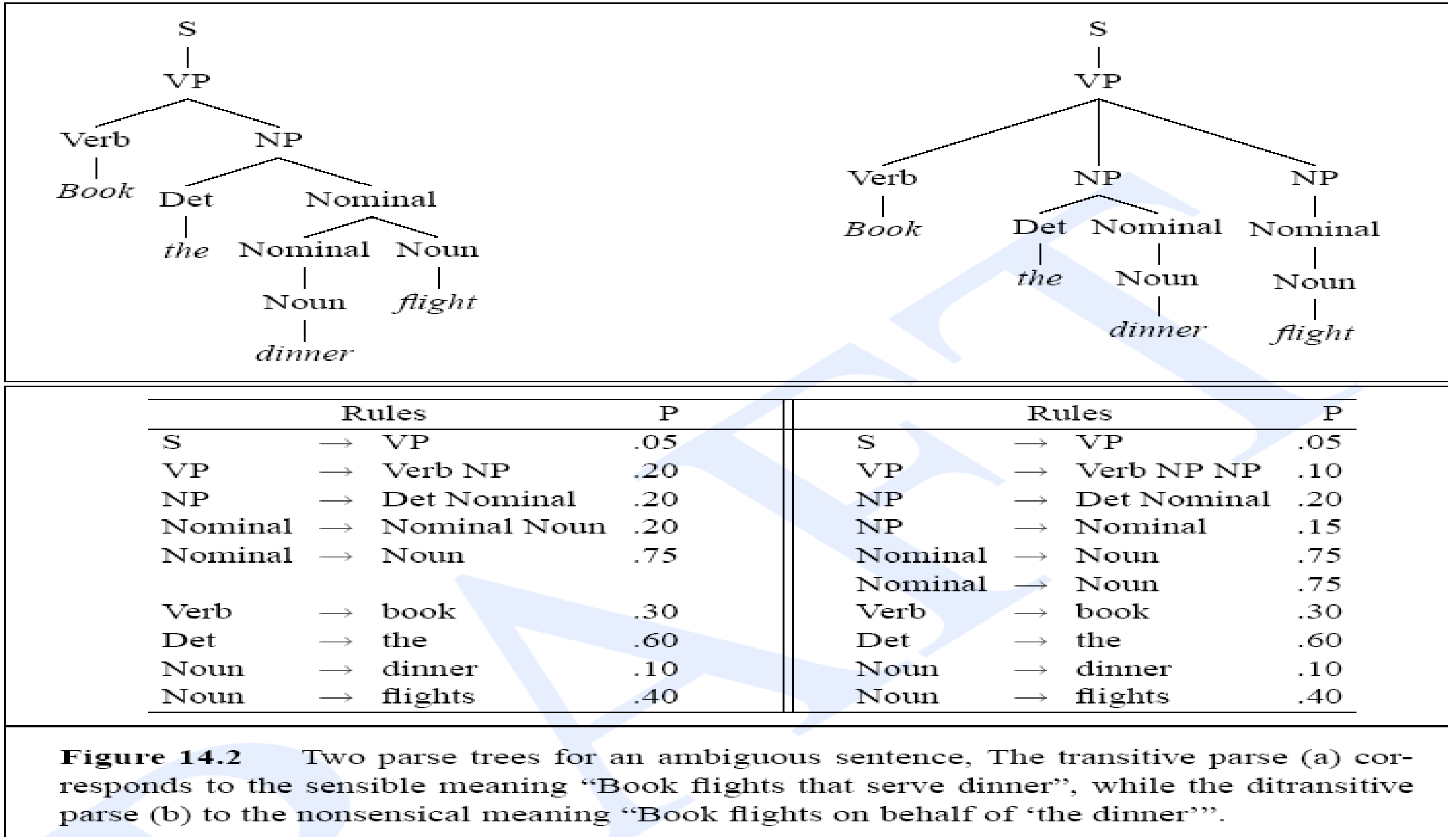
S → *NP VP*	[.80]
S → *Aux NP VP*	[.15]
S → *VP*	[.05]
NP → *Pronoun*	[.35]
NP → *Proper-Noun*	[.30]
NP → *Det Nominal*	[.20]
NP → *Nominal*	[.15]
Nominal → *Noun*	[.75]
Nominal → *Nominal Noun*	[.20]
Nominal → *Nominal PP*	[.05]
VP → *Verb*	[.35]
VP → *Verb NP*	[.20]
VP → *Verb NP PP*	[.10]
VP → *Verb PP*	[.15]
VP → *Verb NP NP*	[.05]
VP → *VP PP*	[.15]
PP → *Preposition NP*	[1.0]

Det → *that* [.10] | *a* [.30] | *the* [.60]
Noun → *book* [.10] | *flight* [.30]
| *meal* [.15] | *money* [.05]
| *flights* [.40] | *dinner* [.10]
Verb → *book* [.30] | *include* [.30]
| *prefer;* [.40]
Pronoun → *I* [.40] | *she* [.05]
| *me* [.15] | *you* [.40]
Proper-Noun → *Houston* [.60]
| *TWA* [.40]
Aux → *does* [.60] | *can* [40]
Preposition → *from* [.30] | *to* [.30]
| *on* [.20] | *near* [.15]
| *through* [.05]

Figure 14.1 A PCFG which is a probabilistic augmentation of the $\mathcal{L}_1$ miniature English CFG grammar and lexicon of Fig. ?? in Ch. 13. These probabilities were made up for pedagogical purposes and are not based on a corpus (since any real corpus would have many more rules, and so the true probabilities of each rule would be much smaller).

PCFGs for Disambiguation

⌘ “Book the dinner flight”



Rules			P	Rules			P
S	→	VP	.05	S	→	VP	.05
VP	→	Verb NP	.20	VP	→	Verb NP NP	.10
NP	→	Det Nominal	.20	NP	→	Det Nominal	.20
Nominal	→	Nominal Noun	.20	NP	→	Nominal	.15
Nominal	→	Noun	.75	Nominal	→	Noun	.75
				Nominal	→	Noun	.75
Verb	→	book	.30	Verb	→	book	.30
Det	→	the	.60	Det	→	the	.60
Noun	→	dinner	.10	Noun	→	dinner	.10
Noun	→	flights	.40	Noun	→	flights	.40

Figure 14.2 Two parse trees for an ambiguous sentence, The transitive parse (a) corresponds to the sensible meaning “Book flights that serve dinner”, while the ditransitive parse (b) to the nonsensical meaning “Book flights on behalf of ‘the dinner’”.

PCFGs for Disambiguation

- The probability of a particular parse T is defined as the product of the probabilities of all the *n* rules used to expand each of the *n* non-terminal nodes in the parse tree T
- P(T,S) = $\prod_{i=1}^{n} P(RHS_i \mid LHS_i)$

$$P(T,S) = P(T)P(S \mid T)$$

$$P(T,S) = P(T)P(S \mid T) = P(T)$$

$$\begin{aligned} P(T_{left}) &= .05 * .20 * .20 * .20 * .75 * .30 * .60 * .10 * .40 = \mathbf{2.2} \times \mathbf{10^{-6}} \\ P(T_{right}) &= .05 * .10 * .20 * .15 * .75 * .75 * .30 * .60 * .10 * .40 = \mathbf{6.1} \times \mathbf{10^{-7}} \end{aligned}$$

PCFGs for Disambiguation

- The string of words S is called the yield of any parse tree over S

$$\hat{T}(S) = \operatorname*{argmax}_{T s.t. S=\text{yield}(T)} P(T|S)$$

- By definition

$$\hat{T}(S) = \operatorname*{argmax}_{T s.t. S=\text{yield}(T)} \frac{P(T,S)}{P(S)}$$

- P(S) will be constant for each tree

$$\hat{T}(S) = \operatorname*{argmax}_{T s.t. S=\text{yield}(T)} P(T,S)$$

- P(T,S) = P(T) thus

$$\hat{T}(S) = \operatorname*{argmax}_{T s.t. S=\text{yield}(T)} P(T)$$

PCFGs for Language Modeling

- PCFG assigns a probability to the string of words constituting a sentence.
- The probability of an unambiguous sentence is P(T,S) = P(T)
- The probability of an ambiguous sentence is the sum of the probabilities of all the parse trees for the sentence
- Also PCFG can assign a probability to substring of a sentence

$$\begin{aligned} P(S) &= \sum_{T s.t. S=\text{yield}(T)} P(T,S) \\ &= \sum_{T s.t. S=\text{yield}(T)} P(T) \end{aligned}$$

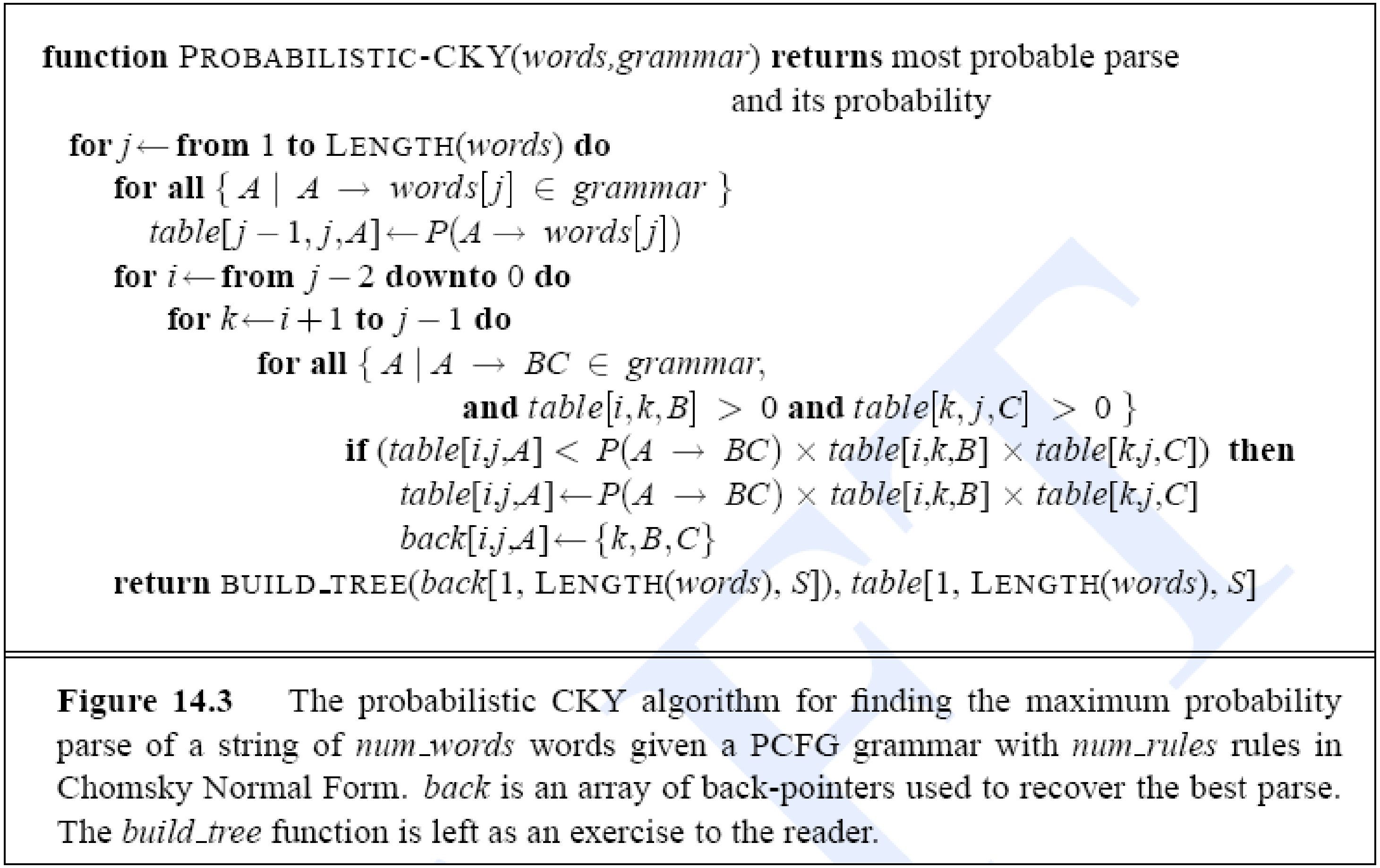
PCFGs for Language Modeling

- PCFG is also useful for language modeling.
  - Assign probability to substrings of a sentence
  - N-gram model is ignoring potentially useful prediction cues
  - "the contract ended with a loss of 7 cents after trading as low as 9 cents"

$$P(w_i|w_1,w_2,\ldots,w_{i-1}) = \frac{P(w_1,w_2,\ldots,w_{i-1},w_i,\ldots)}{P(w_1,w_2,\ldots,w_{i-1},\ldots)}$$

$$P(w_i|w_1,w_2,\ldots,w_{i-1}) \approx \frac{P(w_{i-1},w_i)}{P(w_{i-1})}$$

Probabilistic CKY Parsing of PCFGs

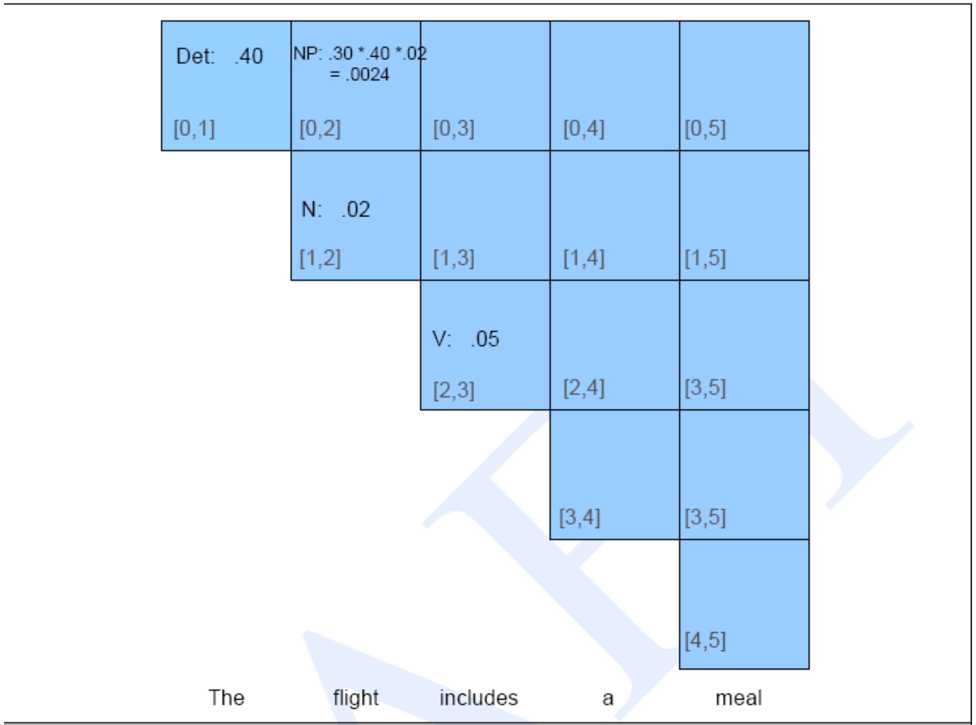


```
function PROBABILISTIC-CKY($words$,$grammar$) returns most probable parse
                                                  and its probability
  for $j \leftarrow$ from 1 to LENGTH($words$) do
     for all $\{ A \mid A \rightarrow words[j] \in grammar \}$
        $table[j-1, j, A] \leftarrow P(A \rightarrow words[j])$
     for $i \leftarrow$ from $j-2$ downto 0 do
        for $k \leftarrow i+1$ to $j-1$ do
           for all $\{ A \mid A \rightarrow BC \in grammar,$
                    and $table[i,k,B] > 0$ and $table[k,j,C] > 0 \}$
              if ($table[i,j,A] < P(A \rightarrow BC) \times table[i,k,B] \times table[k,j,C]$) then
                 $table[i,j,A] \leftarrow P(A \rightarrow BC) \times table[i,k,B] \times table[k,j,C]$
                 $back[i,j,A] \leftarrow \{k,B,C\}$
     return BUILD_TREE($back[1,$ LENGTH($words$)$, S]$), $table[1,$ LENGTH($words$)$, S]$
```

Figure 14.3 The probabilistic CKY algorithm for finding the maximum probability parse of a string of *num_words* words given a PCFG grammar with *num_rules* rules in Chomsky Normal Form. *back* is an array of back-pointers used to recover the best parse. The *build_tree* function is left as an exercise to the reader.

Probabilistic CKY Parsing of PCFGs

S	→ *NP VP*	.80		*Det*	→ *the*	.50
NP	→ *Det N*	.30		*Det*	→ *a*	.40
VP	→ *V NP*	.20		*N*	→ *meal*	.01
V	→ *includes*	.05		*N*	→ *flight*	.02



Det: .40 [0,1]	NP: .30 *.40 *.02 = .0024 [0,2]	[0,3]	[0,4]	[0,5]
	N: .02 [1,2]	[1,3]	[1,4]	[1,5]
		V: .05 [2,3]	[2,4]	[3,5]
			[3,4]	[3,5]
				[4,5]
The	flight	includes	a	meal

Figure 14.4 The beginning of the probabilistic CKY matrix. Filling out the rest of the chart is left as Exercise 14.4 for the reader.

Learning PCFG Rule Probabilities

- ⌘ Where do PCFG rule probabilities come from?
  - Use Treebank, a corpus of already -parsed sentences
    - Penn Treebank – a collection of parse trees in English, Chinese, and Korean
    - Given a treebank,

$$P(\alpha \to \beta|\alpha) = \frac{\mathrm{Count}(\alpha \to \beta)}{\sum_\gamma \mathrm{Count}(\alpha \to \gamma)} = \frac{\mathrm{Count}(\alpha \to \beta)}{\mathrm{Count}(\alpha)}$$

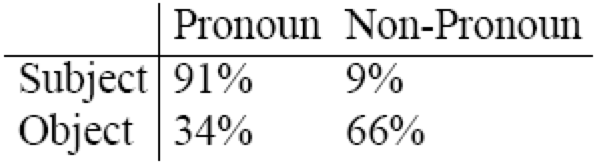
  - With a non-probabilistic parser, we can generate the counts we need for computing PCFG rule probabilities by first parsing a corpus of sentences with the parser
    - If sentences were ambiguous? - parse the corpus, increment a counter for every rule in the parse, and then normalize to get probabilities

Learning PCFG Rule Probabilities

- But, since most sentences are ambiguous
  - Can have multiple parses, we don’t know which parse to count the rules in
  - Instead, we need to keep a separate count for each parse of a sentence and weight each of these partial counts by the probability of the parse it appears in
  - But, these parse probabilities to weight the rules we need to already have a probabilistic parser
  - → chicken-and-egg problem
  - Inside-outside algorithm, EM(Expectation-Maximization) algorithm

Problems with PCFGs - Poor independence assumptions

- In a CFG the expansion of a non-terminal is independent of the context → resulting in poor modeling of structural dependencies across the parse tree.
  - In English the choice of how a node expands can after all be dependent on the location of the node in the parse tree.
  - In English NPs that are syntactic subjects are far more likely to be pronouns, while NPs that are syntactic objects are far more likely to be non-pronominal
  - PCFGs don't allow a rule probability to be conditioned on surround context. Equal probability is all we get
  - Parent annotation is introduced



	Pronoun	Non-Pronoun
Subject	91%	9%
Object	34%	66%

Problems with PCFGs -Lack of Sensitivity to Lexical Dependencies

- CFG rules don’t model syntactic facts about specific words, leading to problems with subcategorization ambiguities, preposition attachment, and coordinate structure ambiguities
  - “workers dumped sacks into a bin”
  - NP attachment is slightly more common in English. If we trained these rule probabilities on a corpus, we might always prefer NP attachment, causing us to misparse this sentence

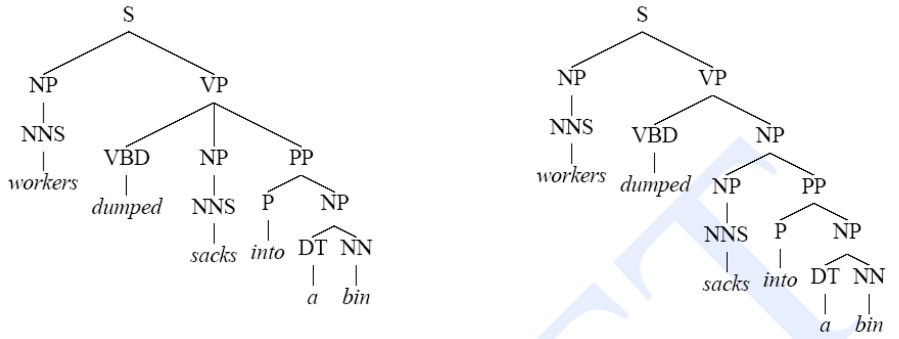


Figure 14.5 Two possible parse trees for a **prepositional phrase attachment ambiguity**. The left parse is the sensible one, in which ‘into a bin’ describes the resulting location of the sacks. In the right incorrect parse, the sacks to be dumped are the ones which are already ‘into a bin’, whatever that could mean.

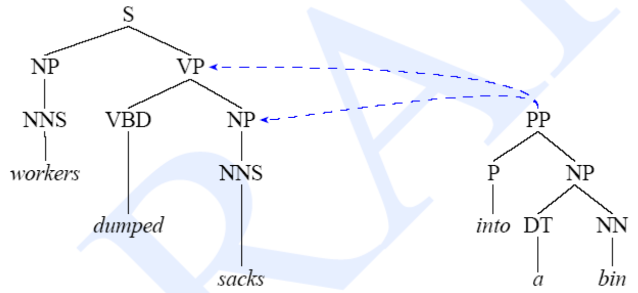
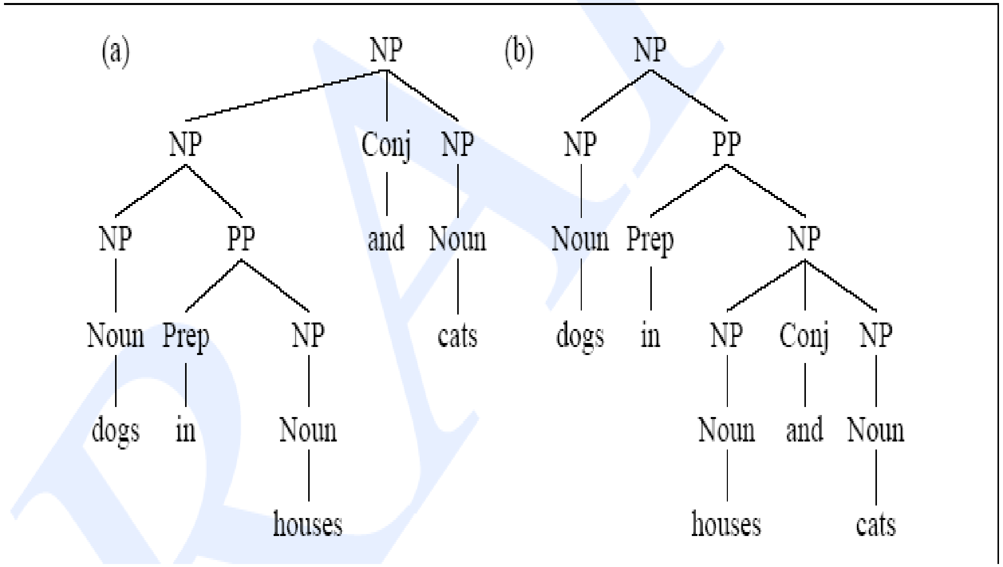


Figure 14.6 Another view of the preposition attachment problem; should the PP on the right attach to the VP or NP nodes of the partial parse tree on the left?

Problems with PCFGs -Lack of Sensitivity to Lexical Dependencies

- Lexical dependency statistics required
  - The affinity between the verb “dumped” and the preposition “into” is greater than the affinity between the noun “sacks” and the preposition “into”, thus leading to VP attachment
- Coordination ambiguity
  - “dogs in houses and cats”



Improving PCFGs By Splitting and Merging NonTerminals

- To model structural dependencies
  - Split the NP non-terminal into two versions: one for subjects, one for objects(NPsubj,NPobject)
  - Parent annotation(Johnson 1998) – annotate each node with its parent in the parse tree.

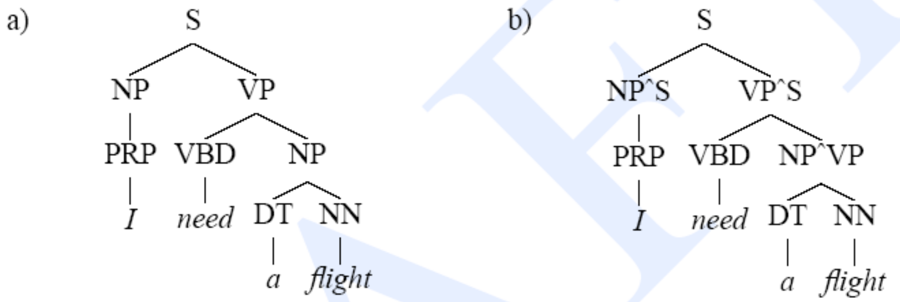


Figure 14.8 A standard PCFG parse tree (a) and one which has **parent annotation** on the nodes which aren't preterminal (b). All the non-terminal nodes (except the preterminal part-of-speech nodes) in parse (b) have been annotated with the identity of their parent.

Improving PCFGs By Splitting and Merging NonTerminals

- In cases where parent annotation is insufficient, we can also hand-write rules that specify a particular node split based on other features of the tree
- Node splitting problems – increases the size of the grammar, and hence reduces the amount of training data available for each grammar rule, leading to overfitting

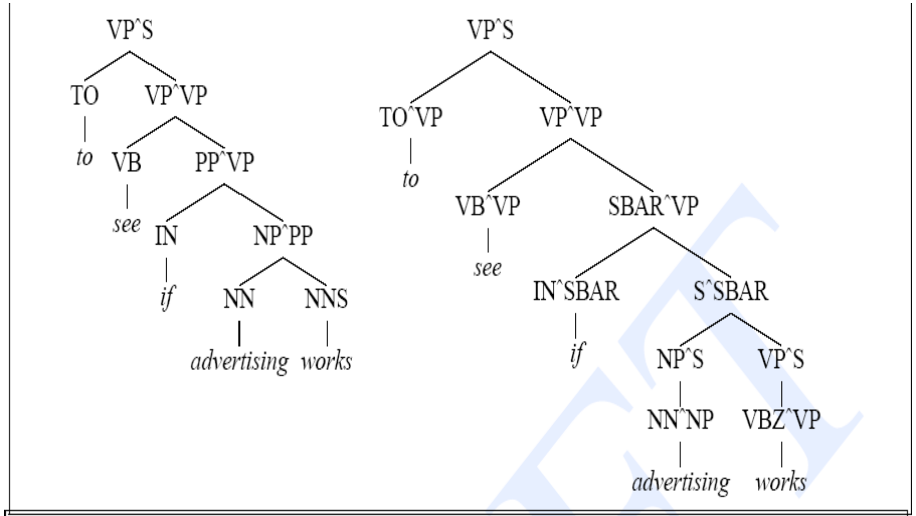
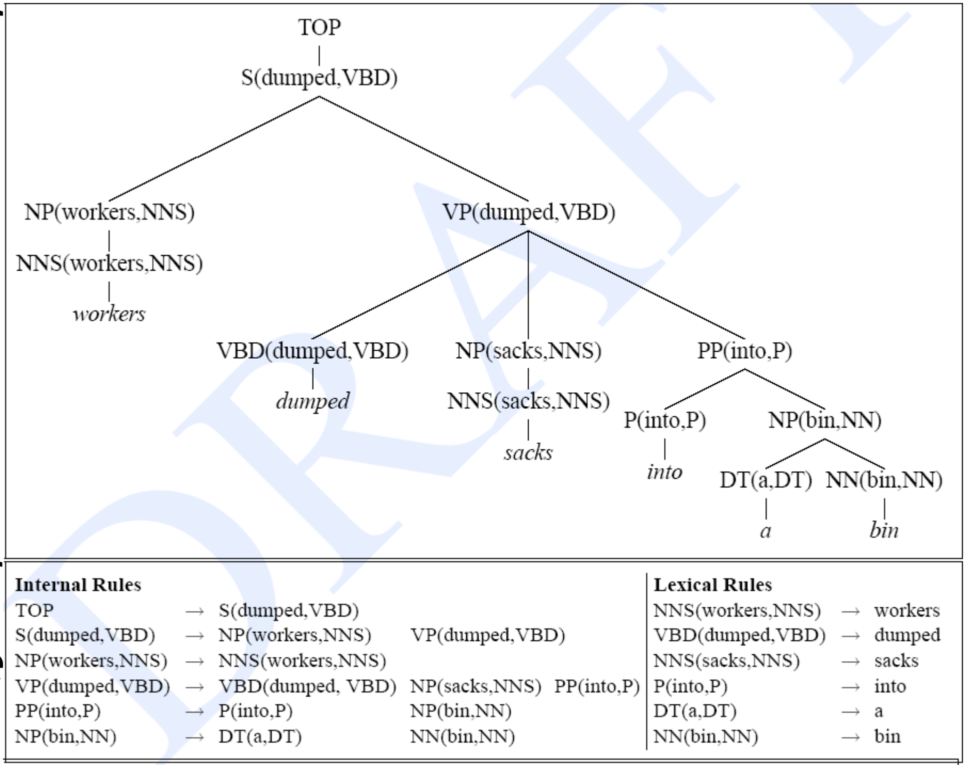


Figure 14.9 An incorrect parse even with a parent annotated parse (left). The correct parse (right), was produced by a grammar in which the pre-terminal nodes have been split, allowing the probabilistic grammar to capture the fact that *if* prefers sentential complements; adapted from Klein and Manning (2003b).

Probabilistic Lexicalized CFGs

- Modify the probabilistic model of the parser to allow for lexicalized rules → Collins Parser, Charniak parser
  - Head tag with nonterminal symbols
  - VP(dumped, VBD) → VBD(dumped, VBD) NP(sacks, NNS) PP(into IN)
  - Lexical rules – express the expansion of a preterminal to a word(deterministic)
  - Internal rules – express the other rule expansions



Internal Rules					**Lexical Rules**		
TOP	→	S(dumped,VBD)			NNS(workers,NNS)	→	workers
S(dumped,VBD)	→	NP(workers,NNS)	VP(dumped,VBD)		VBD(dumped,VBD)	→	dumped
NP(workers,NNS)	→	NNS(workers,NNS)			NNS(sacks,NNS)	→	sacks
VP(dumped,VBD)	→	VBD(dumped, VBD)	NP(sacks,NNS)	PP(into,P)	P(into,P)	→	into
PP(into,P)	→	P(into,P)	NP(bin,NN)		DT(a,DT)	→	a
NP(bin,NN)	→	DT(a,DT)	NN(bin,NN)		NN(bin,NN)	→	bin

Figure 14.10 A lexicalized tree, including head tags, for a WSJ sentence, adapted from Collins (1999). Below we show the PCFG rules that would be needed for this parse tree, internal rules on the left, and lexical rules on the right.

Probabilistic Lexicalized CFGs

- Probability?
  - Too specific
  - Counts of fully lexicalized PCFG rules like this will be far too sparse and most rule probabilities will come out zero
  - Instead, breaking down each rule, so that we would estimate the probability as the product of smaller independent probability estimates

$$P(VP(dumped,VBD) \rightarrow VBD(dumped,VBD)NP(sacks,NNS)PP(into,P))$$
$$= \frac{Count(VP(dumped,VBD) \rightarrow VBD(dumped,VBD)NP(sacks,NNS)PP(into,P))}{Count(VP(dumped,VBD))}$$

The Collins parser

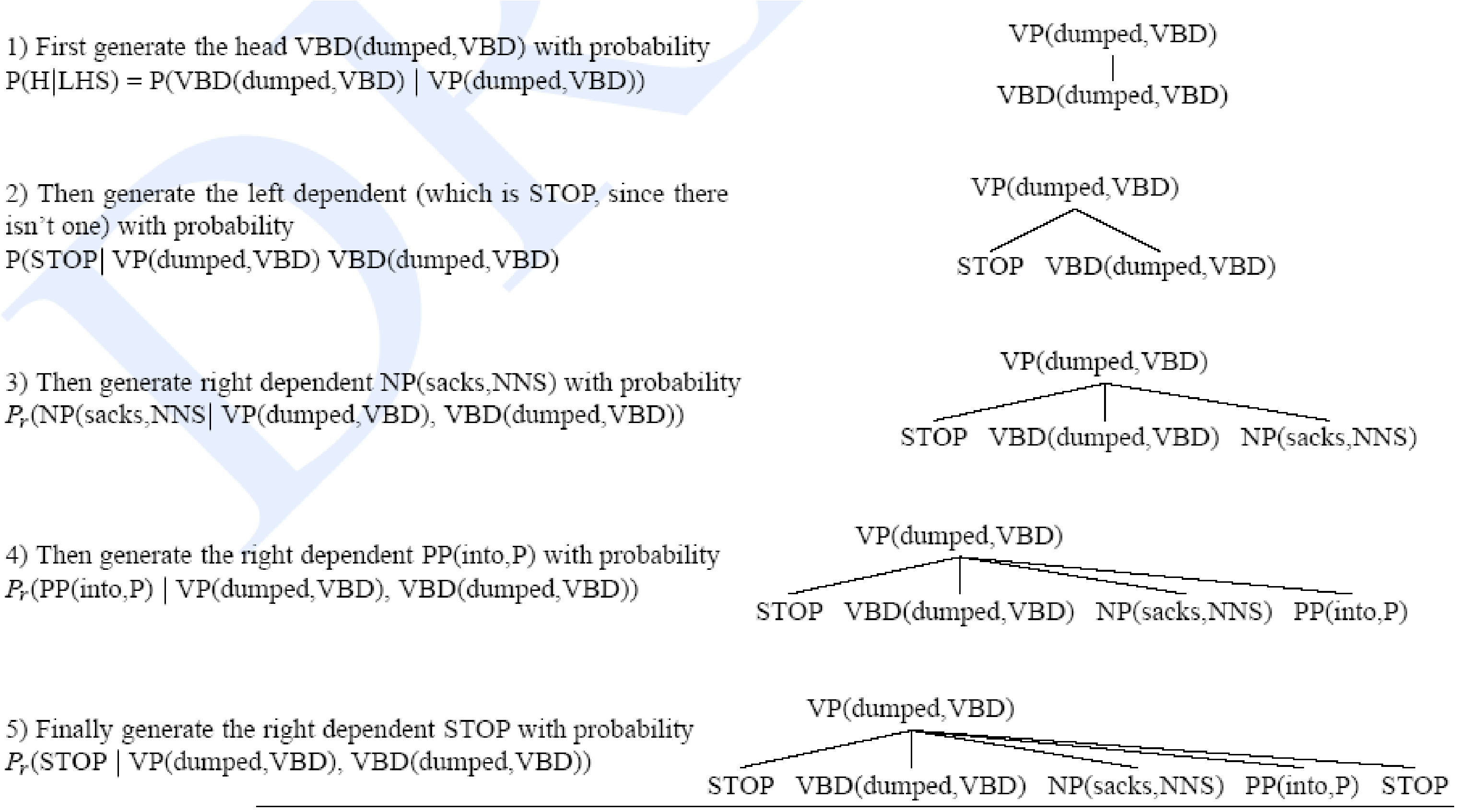
- Intuition – think of the right-hand side of every (internal) CFG rule as consisting of a head non-terminal, together with the non-terminals to the left of the head, and the non-terminals to the right of the head

$$LHS \rightarrow L_n \, L_{n-1} \ldots L_1 \, H \, R_1 \ldots R_{n-1} \, R_n$$

- Collins Model1 – given the left-hand side, first generate the head of the rule, and then generate the dependents of the head, one by one, from inside out.
  - STOP – to stop generating dependents on a given side

$$P(\; VP(dumped,VBD) \rightarrow \mathrm{STOP}\;\; VBD(dumped,VBD)\;\; NP(sacks,NNS)\;\; PP(into,P)\;\; \mathrm{STOP}$$

The Collins parser



1) First generate the head VBD(dumped,VBD) with probability
P(H|LHS) = P(VBD(dumped,VBD) | VP(dumped,VBD))

VP(dumped,VBD)
VBD(dumped,VBD)

2) Then generate the left dependent (which is STOP, since there isn't one) with probability
P(STOP| VP(dumped,VBD) VBD(dumped,VBD)

VP(dumped,VBD)
STOP
VBD(dumped,VBD)

3) Then generate right dependent NP(sacks,NNS) with probability
P_r(NP(sacks,NNS| VP(dumped,VBD), VBD(dumped,VBD))

VP(dumped,VBD)
STOP
VBD(dumped,VBD)
NP(sacks,NNS)

4) Then generate the right dependent PP(into,P) with probability
P_r(PP(into,P) | VP(dumped,VBD), VBD(dumped,VBD))

5) Finally generate the right dependent STOP with probability
P_r(STOP | VP(dumped,VBD), VBD(dumped,VBD))

The Collins Parser

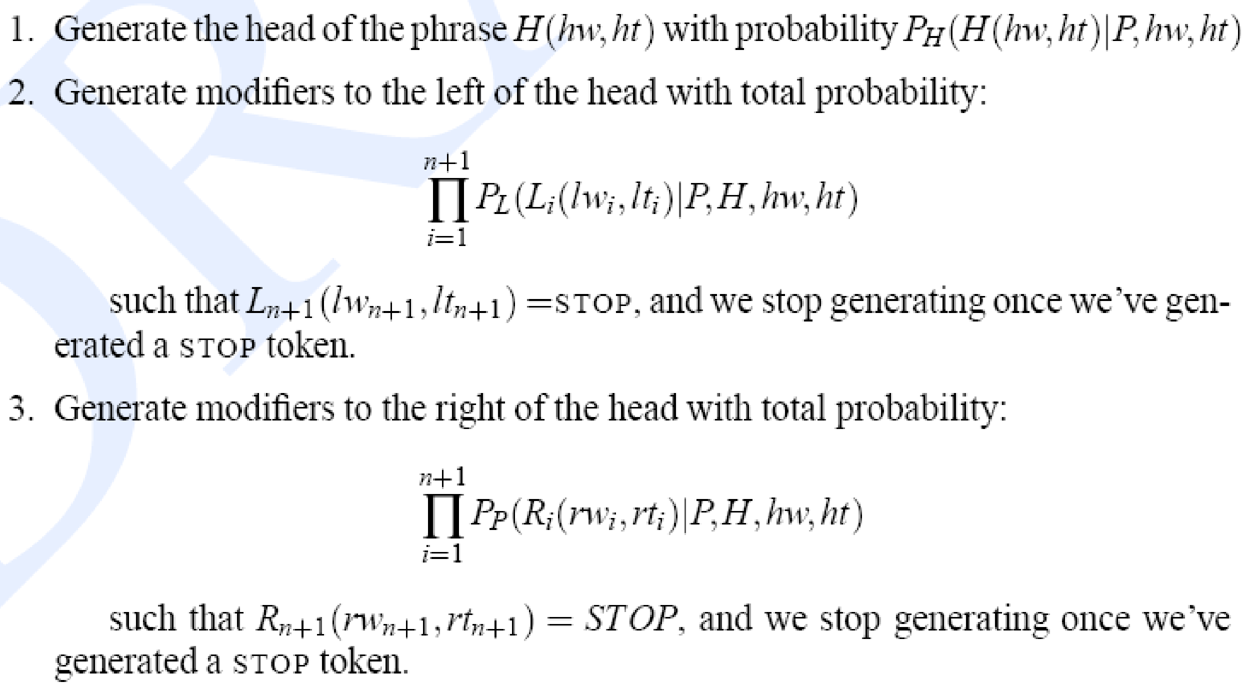
$$P(\ VP(dumped, VBD) \rightarrow VBD(dumped, VBD)\ \ NP(sacks, NNS) PP(into, P)\ )$$

is estimated as:

$$\begin{aligned} P_H(VBD|VP, dumped) &\times P_L(STOP|VP, VBD, dumped) \\ &\times P_R(NP(sacks, NNS)|VP, VBD, dumped) \\ &\times P_R(PP(into, P)|VP, VBD, dumped) \\ &\times P_R(STOP|VP, VBD, dumped) \end{aligned}$$

$$P_R(NP(sacks, NNS)|VP, VBD, dumped) = \frac{\text{Count}(\ VP(dumped, VBD)\ \ \text{with}\ \ NNS(sacks)\text{as a daughter somewhere on the right}\ )}{\text{Count}(\ VP(dumped, VBD)\ )}$$

The Collins Parser



1. Generate the head of the phrase $H(hw, ht)$ with probability $P_H(H(hw, ht)|P, hw, ht)$
2. Generate modifiers to the left of the head with total probability:

$$\prod_{i=1}^{n+1} P_L(L_i(lw_i, lt_i)|P, H, hw, ht)$$

such that $L_{n+1}(lw_{n+1}, lt_{n+1})$ =STOP, and we stop generating once we've generated a STOP token.

3. Generate modifiers to the right of the head with total probability:

$$\prod_{i=1}^{n+1} P_P(R_i(rw_i, rt_i)|P, H, hw, ht)$$

such that $R_{n+1}(rw_{n+1}, rt_{n+1}) = STOP$, and we stop generating once we've generated a STOP token.