Natural Language Processing CS 6320

Lecture 11

Statistical and Neural Parsing

Instructor: Sanda Harabagiu

Probabilistic Context Free Grammars

- The simplest **augmentation** of the Context-Free Grammar (CFG) is the Probabilistic Context-Free Grammar (PCFG)

❑ CFG is defined by (N, Σ, P, S)

1. N – a set of non-terminals
2. Σ - set of terminals ($N \cap \Sigma = \varnothing$)
3. P – set of productions of the form:

   $A \rightarrow \beta$ $, A \in N$, β-a string of symbols s $\in (\Sigma \cup N)$
4. S – a designated start symbol

❑ Each rule in P gets assigned a conditional probability

$A \rightarrow \beta$ [p]

$\Rightarrow$ A PCFG is a 5-tuple $G = (N, \Sigma, P, S, D)$

where **D** is a function assigning probabilities to each rule in P

$P(A \rightarrow \beta)$ or $P(A \rightarrow \beta | A)$

- Given a non-terminal A, and $r_1, r_2, \ldots, r_i$, all expanding A and

$$\sum_{j=1}^{i} P(r_j) = 1$$

Example

S → NP VP	[.80]	Det → that [.05] | the [.80] | a	[.15]
S → Aux NP VP	[.15]	Noun → book	[.10]
S → VP	[.05]	Noun → flights	[.50]
NP → Det Nom	[.20]	Noun → meal	[.40]
NP → Proper-Noun	[.35]	Verb → book	[.30]
NP → Nom	[.05]	Verb → include	[.30]
NP → Pronoun	[.40]	Verb → want	[.40]
Nom → Noun	[.75]	Aux → can	[.40]
Nom → Noun Nom	[.20]	Aux → does	[.30]
Nom → Proper-Noun Nom	[.05]	Aux → do	[.30]
VP → Verb	[.55]	Proper-Noun → TWA	[.40]
VP → Verb NP	[.40]	Proper-Noun → Denver	[.40]
VP → Verb NP NP	[.05]	Pronoun → you [.40] | I [.60]	

Using Probabilities

- to estimate probabilities concerning a sentence and its parse trees
  - ➢ useful in disambiguation

A PCFG is said to be consistent if the sum of the probabilities of all sentences in the language equals 1.

How can we define <u>the probability of a tree</u>?

$$P(T,S)=\prod_{u\in T}P(r(n))$$

The nodes in the parse tree

The grammar rules

The parse tree

The sentence

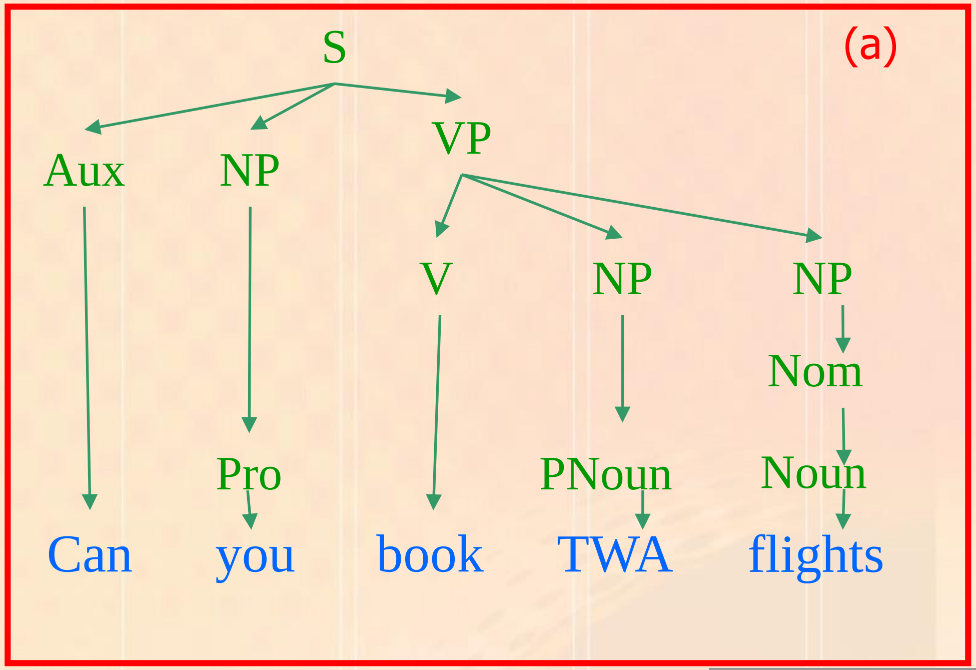
$$P(T,S) = P(T)\,P(S|T) = P(T)$$

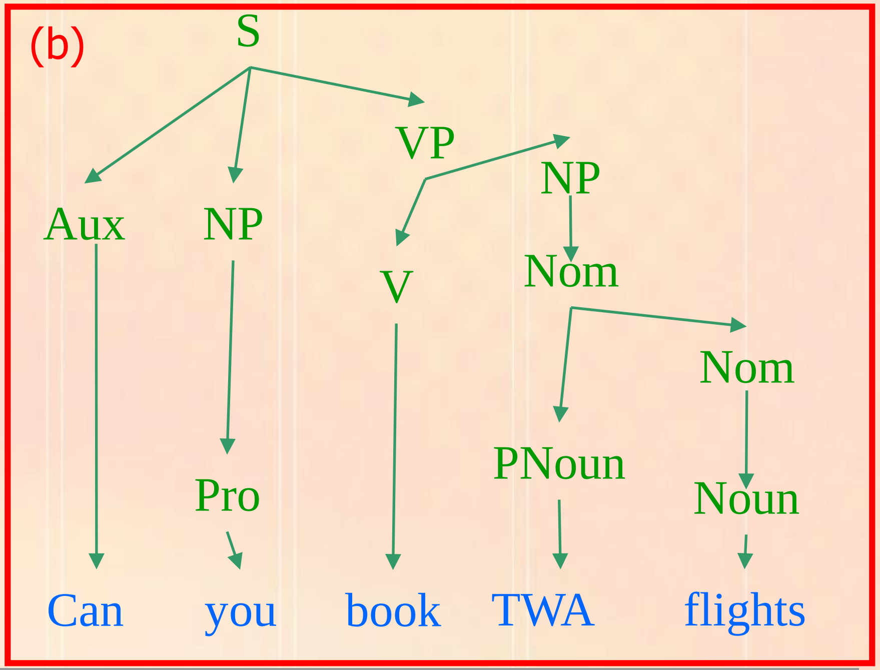
A join probability of the parse and the sentence

A probability of the parse tree

Join Probability

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





$P(T_a) =$
$.15 \times .40 \times .05 \times .05 \times .35 \times$
$\times .75 \times .40 \times .40 \times .40 \times$
$\times .30 \times .40 \times .50 =$ 1.5 x 10-6

$P(T_b) =$
$.15 \times .40 \times .40 \times .05 \times .05 \times$
$\times .75 \times .40 \times .40 \times .40 \times .30 \times$
$\times .40 \times .50 =$ 1.7 x 10-6

S → Aux NP VP	.15	*S → Aux NP VP*	.15
NP → Pro	.40	*NP → Pro*	.40
VP → V NP NP	.05	*VP → V NP*	.40
NP → Nom	.05	*NP → Nom*	.05
NP → PNoun	.35	*Nom → PNoun Nom*	.05
Nom → Noun	.75	*Nom → Noun*	.75
Aux → Can	.40	*Aux → Can*	.40
NP → Pro	.40	*NP → Pro*	.40
Pro → you	.40	*Pro → you*	.40
Verb → book	.30	*Verb → book*	.30
PNoun → TWA	.40	*PNoun → TWA*	.40
Noun → flights	.50	*Noun → flights*	.50

Formalization

- How can we formalize the intuition of selecting the parse with the highest probability?
- Given a sentence S, let us denote by τ(S) all the parse trees derived from S. To select the most likely tree for S:

$$\boxed{\hat{T}(S) = \underset{T \in \tau(S)}{\operatorname{argmax}} P(T \mid S)}$$

$$\downarrow$$

$$\boxed{\hat{T}(S) = \underset{T \in \tau(S)}{\operatorname{argmax}} \frac{P(T,S)}{P(S)}} \rightarrow \boxed{\hat{T}(S) = \underset{T \in \tau(S)}{\operatorname{argmax}} P(T,S)}$$

But, P(T, S) = P(T)

$$\Rightarrow \boxed{\hat{T}(S) = \underset{T \in \tau(S)}{\operatorname{argmax}} P(T)}$$

Probabilistic CYK Parsing of PCFGs

- ❑ The parsing problem for PCFG is to produce the most likely parse tree for a sentence by using:

$$\boxed{\hat{T}(S) = \underset{T \in \tau(S)}{\operatorname{argmax}} P(T)}$$

- Three possibilities: extend the CYK algorithm
- ❑ Extension of the CYK (Cocke – Younger – Kasami) algorithm

$\rightarrow$ bottom–up parser

- **Input:**
  - a Chomsky normal form PCFG, G=(N, Σ, P, S, D). Assume that S is the non-terminal with index 1.
  - n words $w_1 \ldots w_n$
- **Data structure:**
  - a dynamic programming array $\pi[i,j,a]$ holds the maximum probability for a constituent with non-terminal index a spanning words i..j
- **Output:**
  - the maximum probability parse $\pi[1,n,1]$

Remember????

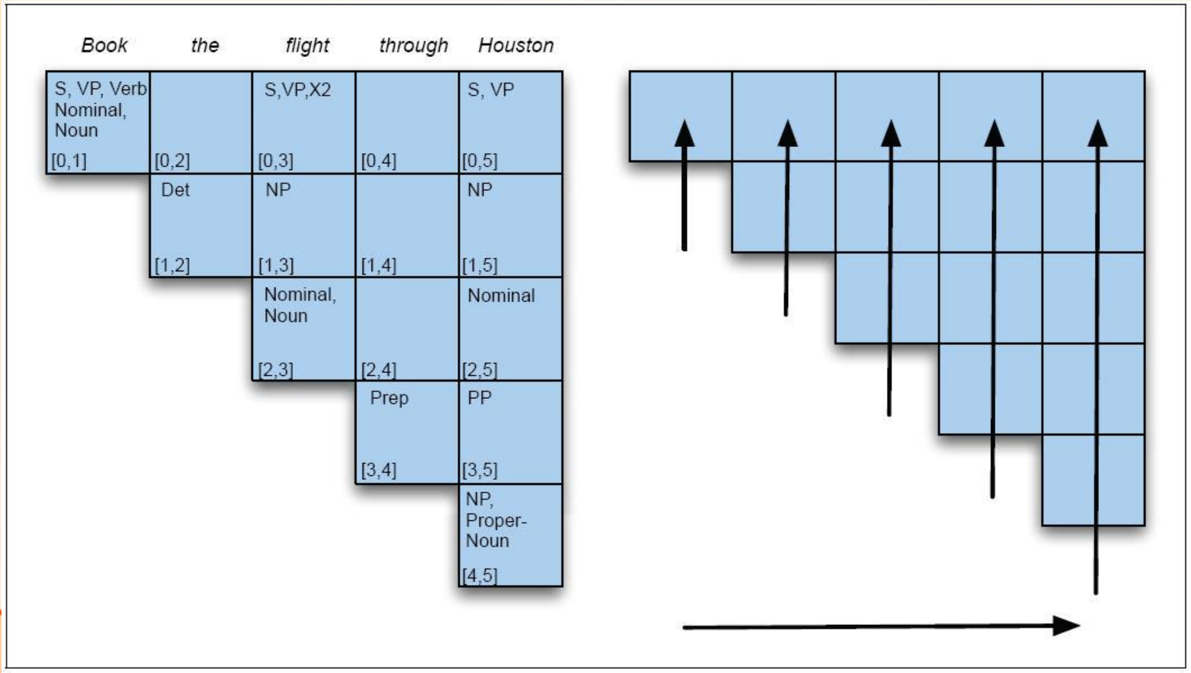
```
function CKY-PARSE (words, grammar) returns table

    for j ← 1 to LENGTH (words) do
        table[j-1,j] ← {A| A→ words [j] ∈ grammar}
        for i ← j-1 downto 0 do
            for k ← i+1 to j-1 do
                table[i,j] ← table[i,j] ∪
                    {A| A → B C ∈ grammar,
                        B ∈ table[i,k],
                        C ∈ table[k,j] }
```

The **outermost loop** of the algorithm iterates over the columns,

the **second loop** iterates over the rows, from the bottom up.

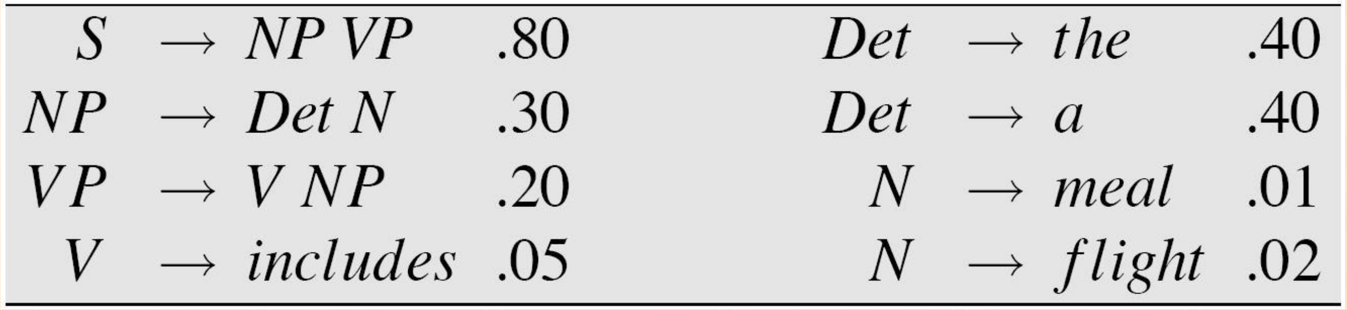
The purpose of the **inner-most loop** is to range over all the places where a substring spanning ***i*** to ***j*** in the input might be split in two.
As ***k*** ranges over the places where the string can be split, the pairs of cells we consider move, in lockstep, to <u>the right</u> along row ***i*** and <u>down</u> along column ***j***.



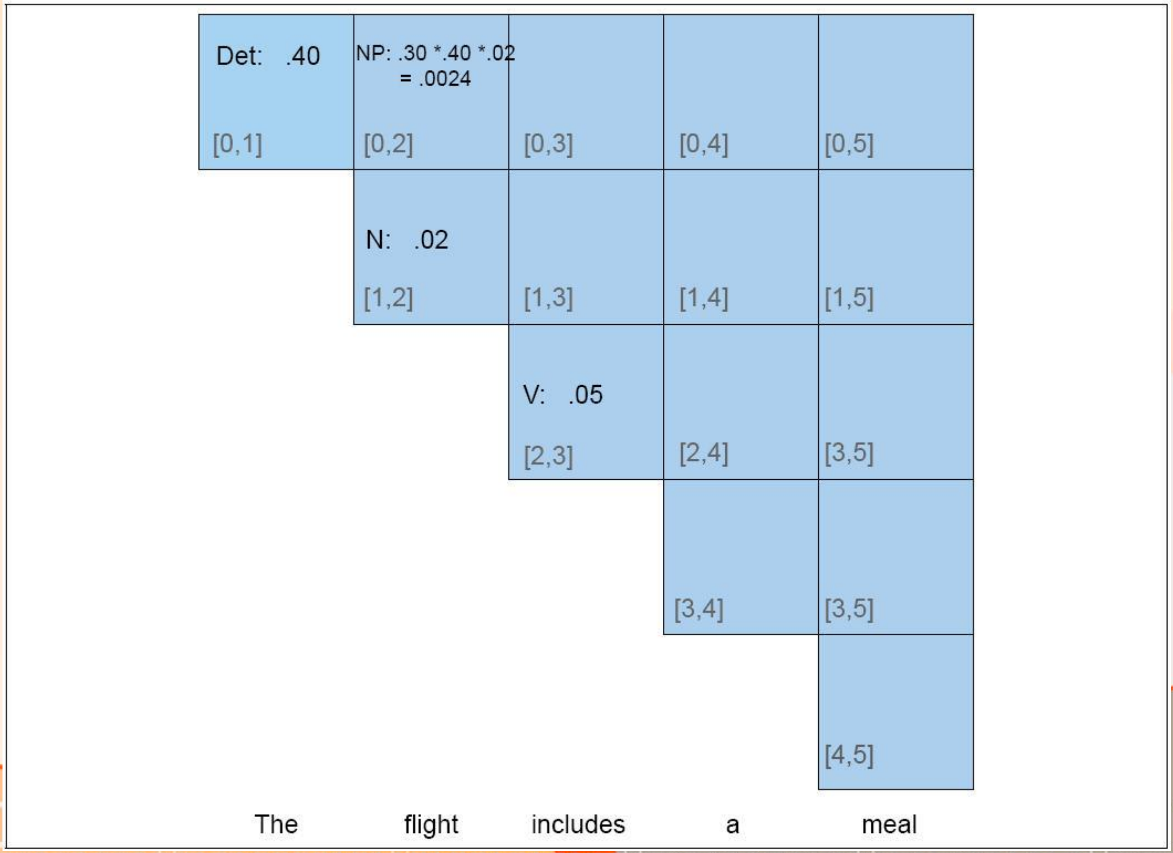
Book	the	flight	through	Houston
S, VP, Verb Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	S, VP [0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper-Noun [4,5]

And now...

- *Suppose you have:*



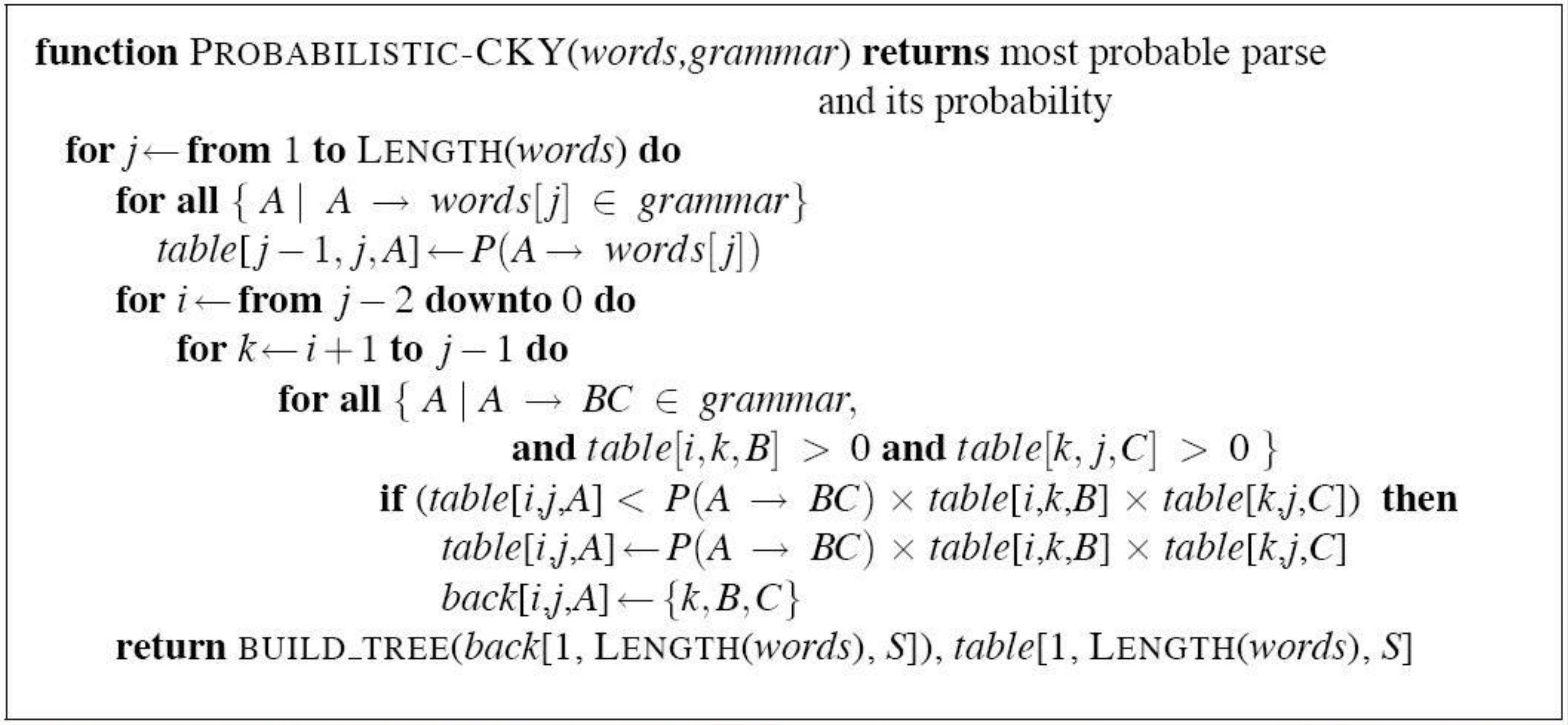
S	$\rightarrow$	$NP\ VP$	.80	Det	$\rightarrow$	*the*	.40
NP	$\rightarrow$	$Det\ N$	.30	Det	$\rightarrow$	*a*	.40
VP	$\rightarrow$	$V\ NP$	.20	N	$\rightarrow$	*meal*	.01
V	$\rightarrow$	*includes*	.05	N	$\rightarrow$	*flight*	.02



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

The filling of the table: Exercise!!!!

CYK Algorithm



```
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($words$), $S$]
```

Back is an array of back-pointers used

to recover the most probable parse !!!!

Learning PCFG Probabilities

- From a **treebank**

$$\boxed{P(\alpha \to \beta \mid \alpha) = \frac{Count(\alpha \to \beta)}{\sum_{\gamma} Count(\alpha \to \gamma)} = \frac{Count(\alpha \to \beta)}{Count(\alpha)}}$$

❑ **What happens if we do not have a Treebank, but we have a Kind-of Statistical Parser:**

- *Begin with a parser with equal rule probabilities, then parse the sentence, compute a probability for each parse;*
- *Use these probabilities to **weight the counts**, re-estimate the rule probabilities, and so on, until our probabilities converge!*

The standard algorithm for computing this solution to learning the grammar rule probabilities is called **the inside-outside algorithm**; Baker (1979), a generalization of the forward-backward algorithm for HMMs. It is a special case of the Expectation Maximization (EM) algorithm. (*Manning, C. D. and Schutze, H. (1999). ¨ Foundations of Statistical Natural Language Processing. MIT Press.)*

Problems with PCFGs

While probabilistic context-free grammars are a natural extension to context-free grammars, they have two main problems as probability estimators:

1. ***Poor independence assumptions****: CFG rules impose an independence assumption on probabilities, resulting in poor modeling of structural dependencies across the parse tree.*
2. ***Lack of lexical conditioning****: CFG rules don't model syntactic facts about specific words, leading to problems with subcategorization ambiguities, preposition attachment, and coordinate structure ambiguities.*

Because of these problems, most current probabilistic parsing models use some augmented version of PCFGs, or modify the Treebank-based grammar in some way.

Let us see how?

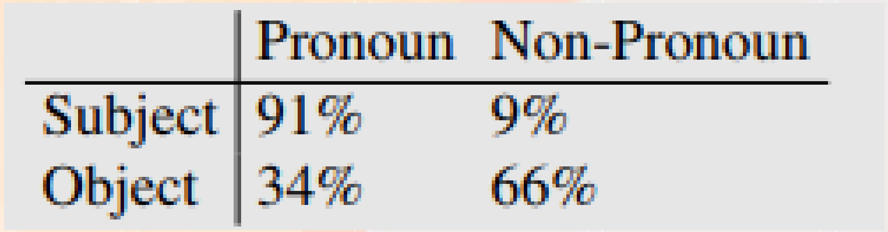
Independence Assumptions Miss Structural Dependencies Between Rules

❑ *In a CFG the expansion of a non-terminal* ***is independent of the context****, that is, of the other nearby non-terminals in the parse tree. Similarly, in a PCFG, the probability of a particular rule like NP → Det N is also* ***independent*** *of the rest of the tree!!!*

➢ *Unfortunately, this CFG independence assumption results in poor probability estimates!*

o *This is because in English the choice of how a node expands can depend on the location of the node in the parse tree.*

▪ *For example, in English it turns out that NPs that are syntactic subjects are far more likely to be pronouns, and NPs that are syntactic objects are far more likely to be non-pronominal (e.g., a proper noun or a determiner noun sequence), as shown by these statistics for NPs in the Switchboard corpus:*



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

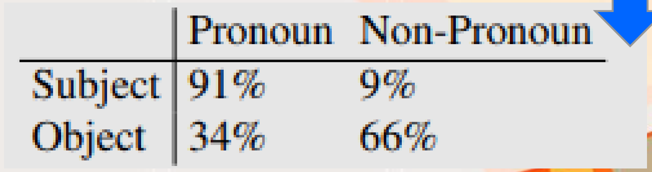
What can be done?

Unfortunately, there is no way to represent the contextual difference in the probabilities in a PCFG.

- Consider two expansions of the non-terminal **NP** as a *pronoun* or as a *determiner+noun*. How shall we set the probabilities of these two rules? If we set their probabilities to their overall probability in the Switchboard corpus, the two rules have about equal probability:

$$\begin{array}{lll} NP \rightarrow DT\ NN & .28 \\ NP \rightarrow PRP & .25 \end{array}$$

- Because PCFGs don't allow a rule probability to be conditioned on surrounding context, this equal probability is all we get; there is no way to capture the fact that in subject position, the probability for NP → PRP should go up to .91, while in object position, the probability for NP → DT NN should go up to .66.

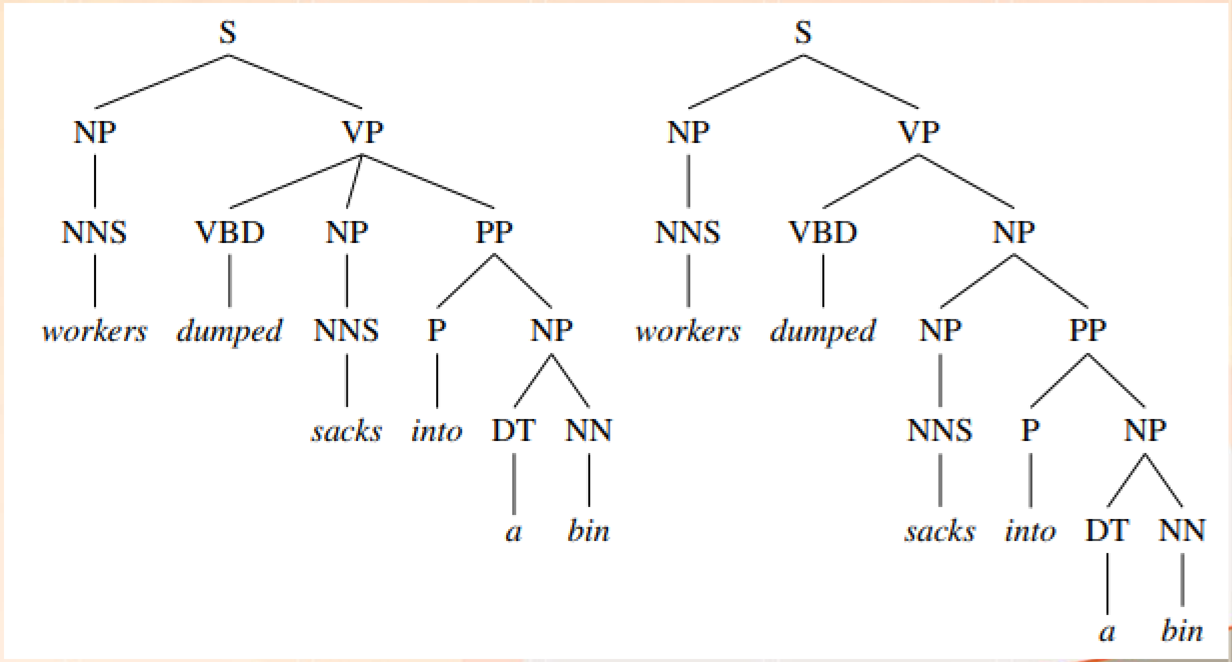


	Pronoun	Non-Pronoun
Subject	91%	9%
Object	34%	66%

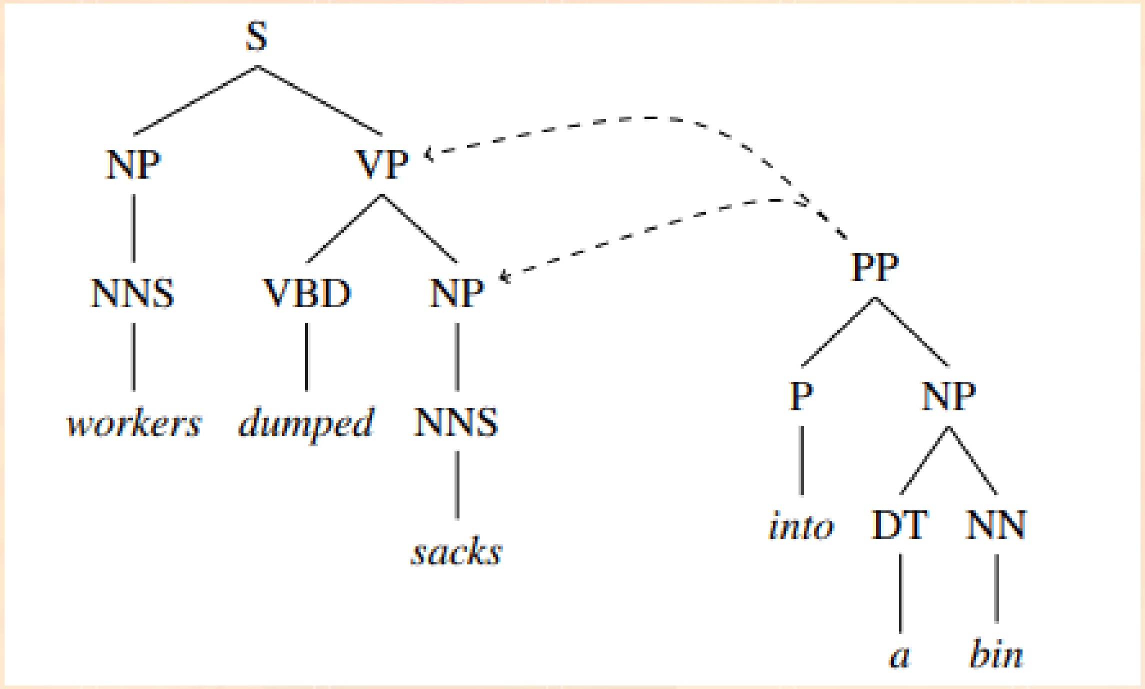
Lack of Sensitivity to Lexical Dependencies

- *PCFGs lack sensitivity to the words in the parse tree!*
- In addition to lexical rules, lexical information is useful in other places in the grammar, such as in resolving prepositional phrase (PP) attachment ambiguities. Since prepositional phrases in English can modify a noun phrase or a verb phrase, when a parser finds a prepositional phrase, it must decide where to attach it into the tree.

Example:



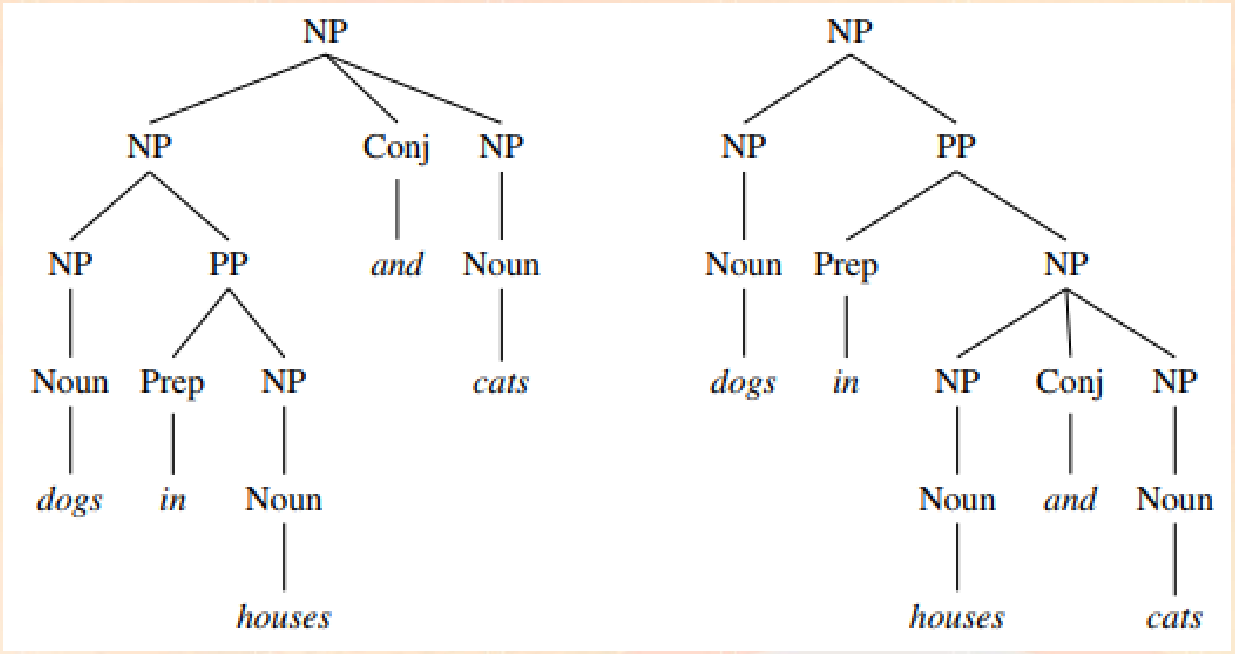
Another view of PP attachment



- *To get the correct parse, we need a model that augments the PCFG probabilities to deal with lexical dependency statistics for different verbs and prepositions*

Coordination Ambiguities

- *lexical dependencies are the key to choosing the proper parse when coordination is needed!*

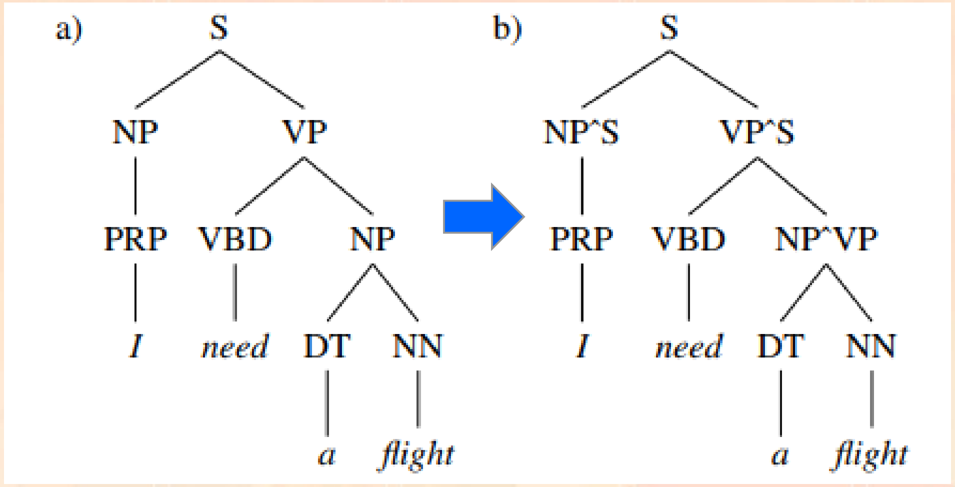


- *Conclusion: context-free grammars are incapable of modeling important structural and lexical dependencies!!! What can be done????*

Some solutions

Idea 1: *split the NP non-terminal into two versions: one for subjects, one for objects!*

- *Implement this intuition of splits by doing* ***parent annotation***
- *an NP node that is the subject of the sentence and hence has parent S would be annotated NPˆS, while a direct object NP whose parent is VP would be annotated NPˆVP*



Node-splitting is not without problems:

1. *it increases the size of the grammar*
2. *reduces the amount of training data available for each grammar rule, leading to overfitting.*

It is important to split to just the correct level of granularity

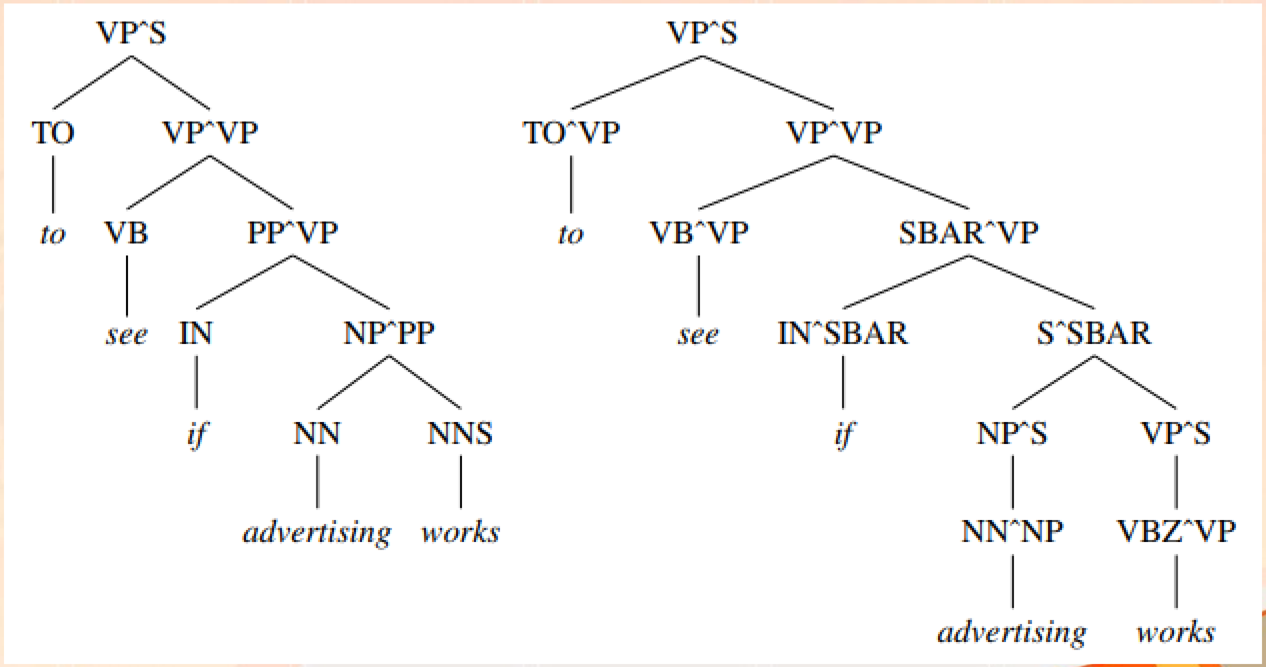
for a particular training set

Additional Splitting

- *In addition to splitting the phrasal nodes, we can also improve a PCFG by splitting the pre-terminal part-of-speech nodes (Klein and Manning, 2003b).*

How???:

Example: the Penn Treebank tag IN can mark a wide variety of parts-of-speech, including subordinating conjunctions (while, as, if), complementizers (that, for), and prepositions (of, in, from). Some of these differences can be captured by parent annotation (subordinating conjunctions occur under S, prepositions under PP), while others require specifically splitting the pre-terminal nodes.



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

- *Instead of modifying the grammar rules, we modify the probabilistic model of the parser to allow for lexicalized rules.*
- *Lexicalized parsers includes the well-known Collins parser (Collins, 1999) and Charniak parser (Charniak, 1997), Charniak both of which are publicly available and widely used throughout natural language processing.*

http://www.cs.columbia.edu/~mcollins/code.html

https://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/parser/charniak/CharniakParser.html

These parsers use a lexicalized grammar, in which each non-terminal in the lexicalized grammar tree is annotated with its lexical head, where a rule like
$VP \rightarrow VBD\ NP\ PP$ would be extended as:

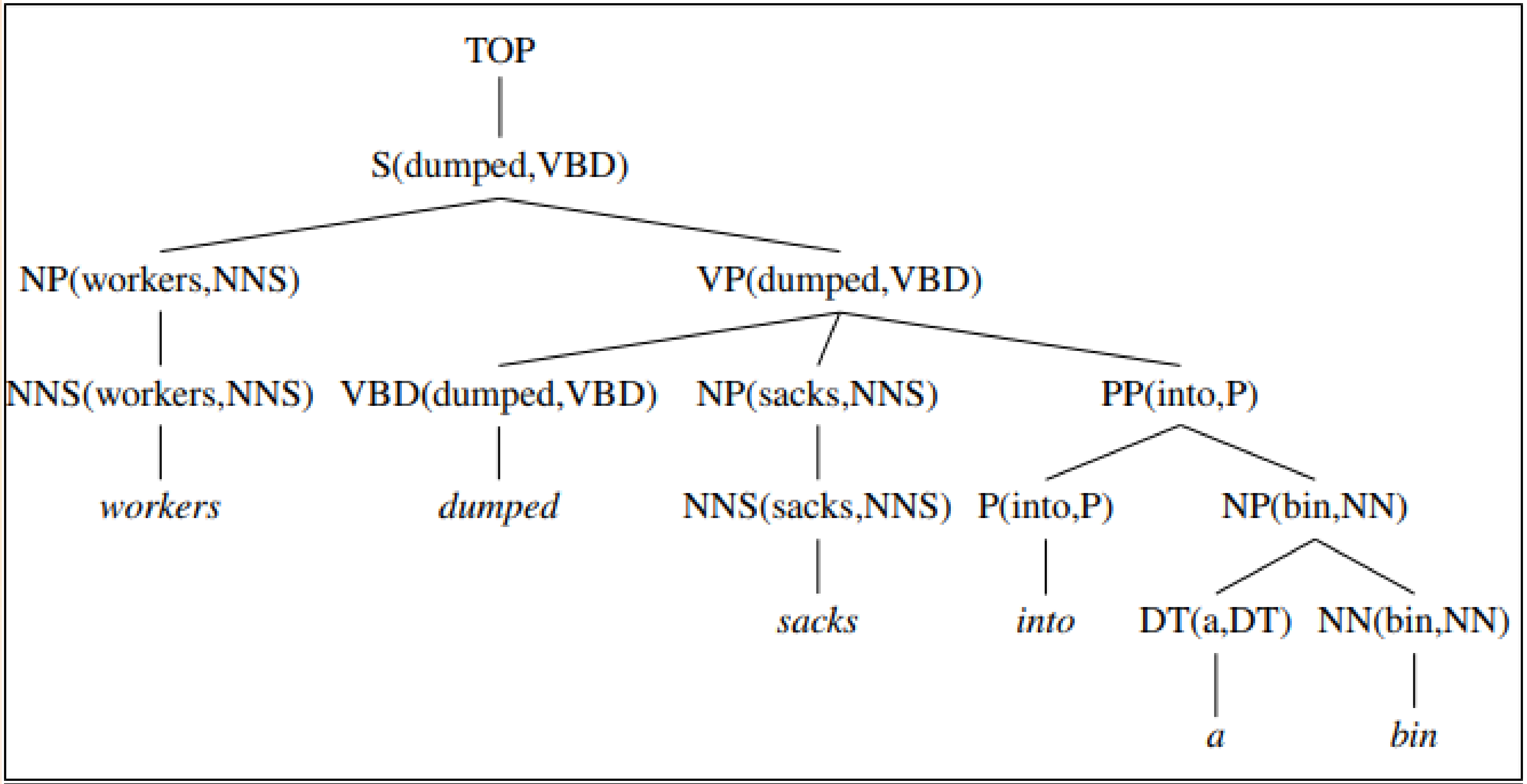
$$VP(dumped) \rightarrow VBD(dumped)\ NP(sacks)\ PP(into)$$

In the standard type of lexicalized grammar, we actually make a further extension, which is to associate the head tag, (the part-of-speech tags of the headwords), with the non-terminal symbols as well. Each rule is thus lexicalized by both the headword and the head tag of each constituent resulting in a format for lexicalized rules like

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

Lexicalized Tree + PCFG rules

- *a WSJ sentence, adapted from Collins (1999)*



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

Parsing as Language Modeling

- *A parsing model parses a sentence(**x**)into its phrasal structure(**y**)according to:*

$$\underset{y' \in \mathcal{Y}(x)}{\operatorname{argmax}} P(x, y'),$$

*given all possible structures of **x.** If we think of a tree(**x**,**y**)as a sequence(**z**) we can define a probability distribution over (**x**,**y**) as follows:*

$$\begin{aligned} P(x, y) = P(z) &= P(z_1, \cdots, z_m) \\ &= \prod_{t=1}^{m} P(z_t | z_1, \cdots, z_{t-1}) \end{aligned}$$

Then we reduce parsing to language modeling and can use language modeling techniques of estimating $P(z_t|z_1,\cdots,z_{t-1})$ for parsing (Choe and Charniak, 2016).

But How?????????????

Models:

1. *The LSTM-LM of Zaremba et al. (2014) turns($x_1,\cdots,x_{t-1}$)into* ***h_t****, a hidden state of an LSTM and uses* ***h_t*** *to guess x_t:*

- where W is a parameter matrix.

$$P(x_t|x_1,\cdots,x_{t-1}) = P(x_t|h_t) = \mathrm{softmax}(Wh_t)[x_t],$$

2. *The phrasal structure (**y**) can be expressed as a sequence and build a machine translation parser (MTP), which is a sequence-to-sequence model that translates **x** into **y** using a conditional probability:*

$$P(\boldsymbol{y}|\boldsymbol{x}) = P(y_1,\cdots,y_l|\boldsymbol{x}) = \prod_{t=1}^{l} P(y_t|\boldsymbol{x},y_1,\cdots,y_{t-1})$$

- where the conditioning event($x,y_1,\cdots,y_{t-1}$) is modeled by an LSTM encoder and an LSTM de-coder. HOW???

*The encoder maps **x** into:*

*h^e, a set of vectors that represents **x**, and the decoder obtains a summary vector(h'_t) which is the concatenation of the decoder's hidden state(h_t^d) and the weighted sum of the word representations ($\sum_{i=1}^{h} \alpha_i h_i^e$) with an alignment vector(α). The decoder predicts y_t given h'_t.*

More models

3. *Recurrent Neural Network Grammars (RNNG), a parsing model that defines a joint distribution over a tree in terms of **actions** the model takes to generate the tree (Dyer et al., 2016):*

$$P(x,y) = P(a) = \prod_{t=1}^{m} P(a_t | a_1, \cdots, a_{t-1}),$$

- *where **a** is a sequence of actions whose output precisely matches the sequence of symbols in z*

$$P(x,y) = P(z) = P(z_1, \cdots, z_m)$$

$$= \prod_{t=1}^{m} P(z_t | z_1, \cdots, z_{t-1})$$

Code available at:
https://github.com/clab/rnng

4. *Choe and Charniak, 2016:*

$$P(x,y) = P(z) = \prod_{t=1}^{m} P(z_t | z_1, \cdots, z_{t-1})$$

$$= \prod_{t=1}^{m} P(z_t | h_t)$$

$$= \prod_{t=1}^{m} \mathrm{softmax}(W h_t)[z_t],$$

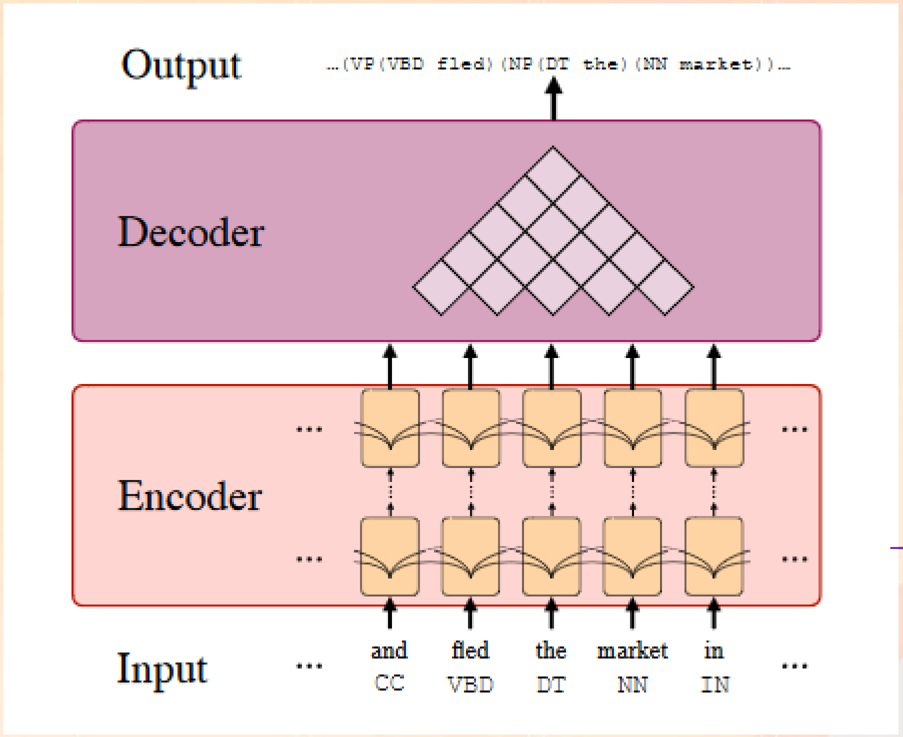
Where h_t is a hidden state of an LSTM.
Code available at:
https://github.com/cdg720/emnlp2016
https://github.com/clab/rnng

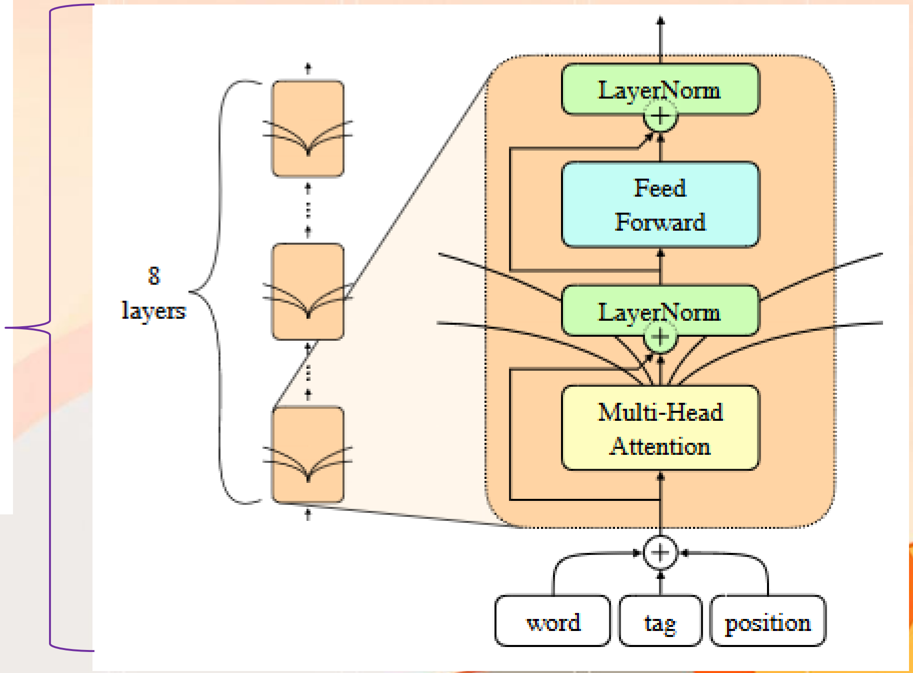
State-of-the-art

- *Constituency Parsing with a Self-Attentive Encoder* (Kitaev & Klein, 2018)

http://nlp.cs.berkeley.edu/pubs/Kitaev-Klein_2018_SelfAttentiveParser_paper.pdf

Code: https://github.com/nikitakit/self-attentive-parser

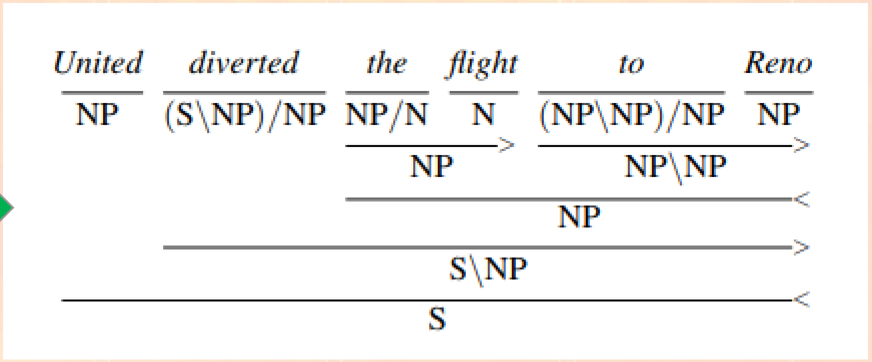




Probabilistic CCG Parsing

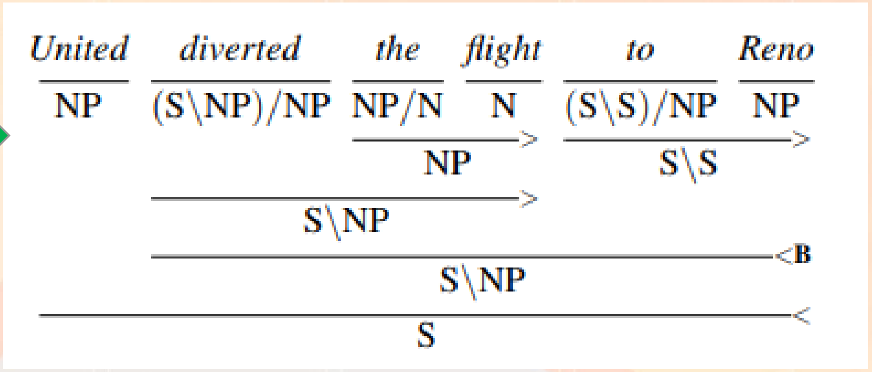
- *The difficulties with CCG parsing arise from the ambiguity caused by the large number of complex lexical categories combined with the very general nature of the grammatical rules.*

The category assigned to *to* expects to find two arguments: one to the right as with a traditional preposition, and one to the left that corresponds to the NP to be modified



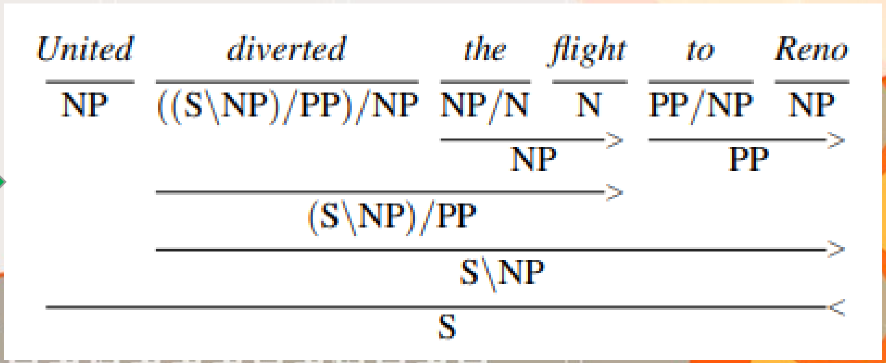
United diverted the flight to Reno
NP (S\NP)/NP NP/N N (NP\NP)/NP NP
NP
NP\NP
NP
S\NP
S

Assign to *to* the category (S\S)/NP, which permits the derivation where *to Reno* modifies the preceding verb phrase



United diverted the flight to Reno
NP (S\NP)/NP NP/N N (S\S)/NP NP
NP
S\S
S\NP
<B
S\NP
S

View *divert* as a ditransitive verb by assigning it to the category: ((S\NP)/PP)/NP, while treating *to Reno* as a simple prepositional phrase



United diverted the flight to Reno
NP ((S\NP)/PP)/NP NP/N N PP/NP NP
NP
PP
(S\NP)/PP
S\NP
S

Supertagging

- *the large number of lexical categories available for each word, combined with the promiscuity of CCG’s combinatoric rules, leads to an explosion in the number of (mostly useless) constituents – need for* ***supertags!****!!*

- *CCG supertaggers rely on treebanks such as CCGbank to provide both the overall set of lexical categories as well as the allowable category assignments for each word in the lexicon. CCGbank includes over <u>1000 lexical categories</u>, however, in practice, most supertaggers limit their tagsets to those tags that occur at least 10 times in the training corpus. This results in an overall total of around <u>425 lexical categories</u> available for use in the lexicon.*

$$
\begin{aligned}
\hat{T} &= \underset{T}{\operatorname{argmax}}\, P(T|W) \\
&= \underset{T}{\operatorname{argmax}} \prod_i P(t_i | w_{i-l}^{i+l}, t_{i-k}^{i-1}) \\
&= \underset{T}{\operatorname{argmax}} \prod_i \frac{\exp\left(\sum_i w_i f_i(t_i, w_{i-l}^{i+l}, t_{i-k}^{i-1})\right)}{\sum_{t' \in \text{tagset}} \exp\left(\sum_i w_i f_i(t', w_{i-l}^{i+l}, t_{i-k}^{i-1})\right)}
\end{aligned}
$$

CCG Parsing using the A* Algorithm

- ❑ *The **A* algorithm** is a heuristic search method that employs **an agenda** to find an optimal solution.*
- ➢ ***Search states** representing partial solutions are added to an agenda based on a cost function, with the least-cost option being selected for further exploration at each iteration.*
- ➢ *When a state representing a complete solution is first selected from the agenda, it is guaranteed to be optimal and the search terminates.*
- ➢ *The A* cost function, **f(n)**, is used to efficiently guide the search to a solution. The f-cost has two components:*

1. *g(n), the exact cost of the partial solution represented by the state n, and*
2. *h(n) a heuristic approximation of the cost of a solution that makes use of n.*

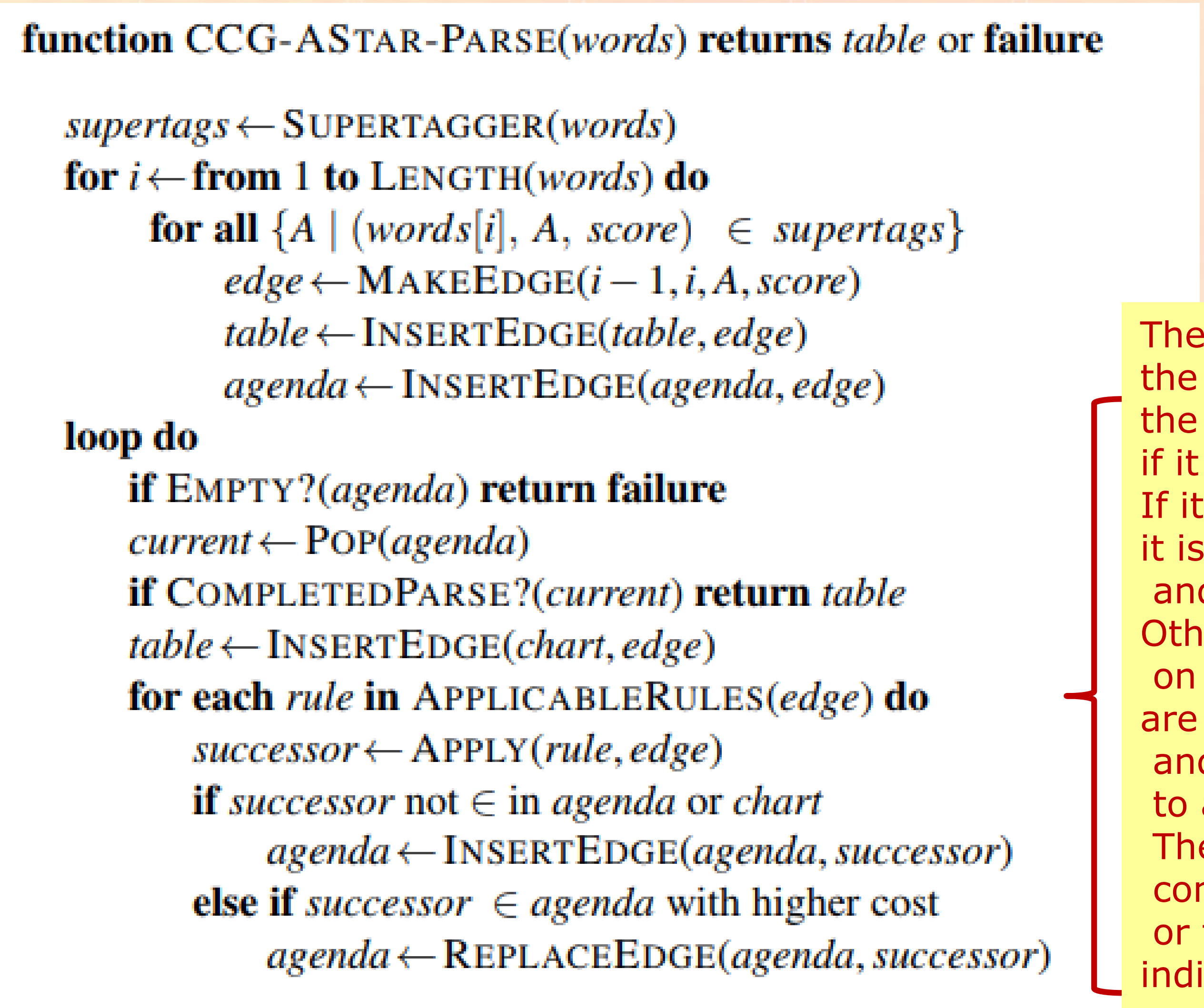
When h(n) satisfies the criteria of not overestimating the actual cost, A will find an optimal solution.*

- ❑ *4 parsing: search states correspond to edges representing completed constituents.*
- ➢ *Edges specify a constituent's :*
    - *start and end positions,*
    - *its grammatical category, and*
    - *its f-cost.*

g represents the current cost of an edge and the h represents an estimate of the cost to complete a derivation that makes use of that edge!!!

A*-based CCG parsing

- *Using information from a super-tagger, an agenda and* ***a parse table*** *are initialized with states representing all the possible lexical categories for each word in the input, along with their f-costs.*



```
function CCG-ASTAR-PARSE(words) returns table or failure

  supertags ← SUPERTAGGER(words)
  for i ← from 1 to LENGTH(words) do
      for all {A | (words[i], A, score) ∈ supertags}
          edge ← MAKEEDGE(i − 1, i, A, score)
          table ← INSERTEDGE(table, edge)
          agenda ← INSERTEDGE(agenda, edge)
  loop do
      if EMPTY?(agenda) return failure
      current ← POP(agenda)
      if COMPLETEDPARSE?(current) return table
      table ← INSERTEDGE(chart, edge)
      for each rule in APPLICABLERULES(edge) do
          successor ← APPLY(rule, edge)
          if successor not ∈ in agenda or chart
              agenda ← INSERTEDGE(agenda, successor)
          else if successor ∈ agenda with higher cost
              agenda ← REPLACEEDGE(agenda, successor)
```

The main loop removes the lowest cost edge from the agenda and tests to see if it is a complete derivation. If it reflects a complete derivation it is selected as the best solution and the loop terminates. Otherwise, new states based on the applicable CCG rules are generated, assigned costs, and entered into the agenda to await further processing. The loop continues until a complete derivation is discovered, or the agenda is exhausted, indicating a failed parse.

Cost Functions

- *For the generic PCFG model, we defined the probability of a tree as the product of the probability of the rules that made up the tree.*
- *Given CCG's lexical nature, we'll make the simplifying assumption that the probability of a CCG derivation is just the product of the probability of the super-tags assigned to the words in the derivation, ignoring the rules used in the derivation!!!*

More formally, given a sentence S and derivation D that contains supertag sequence T, we have:

$$\begin{aligned} P(D,S) &= P(T,S) \\ &= \prod_{i=1}^{n} P(t_i|s_i) \end{aligned}$$

- *we'd prefer to have states scored by a cost function where lower is better:*

$$\begin{aligned} P(D,S) &= P(T,S) \\ &= \sum_{i=1}^{n} -\log P(t_i|s_i) \end{aligned}$$

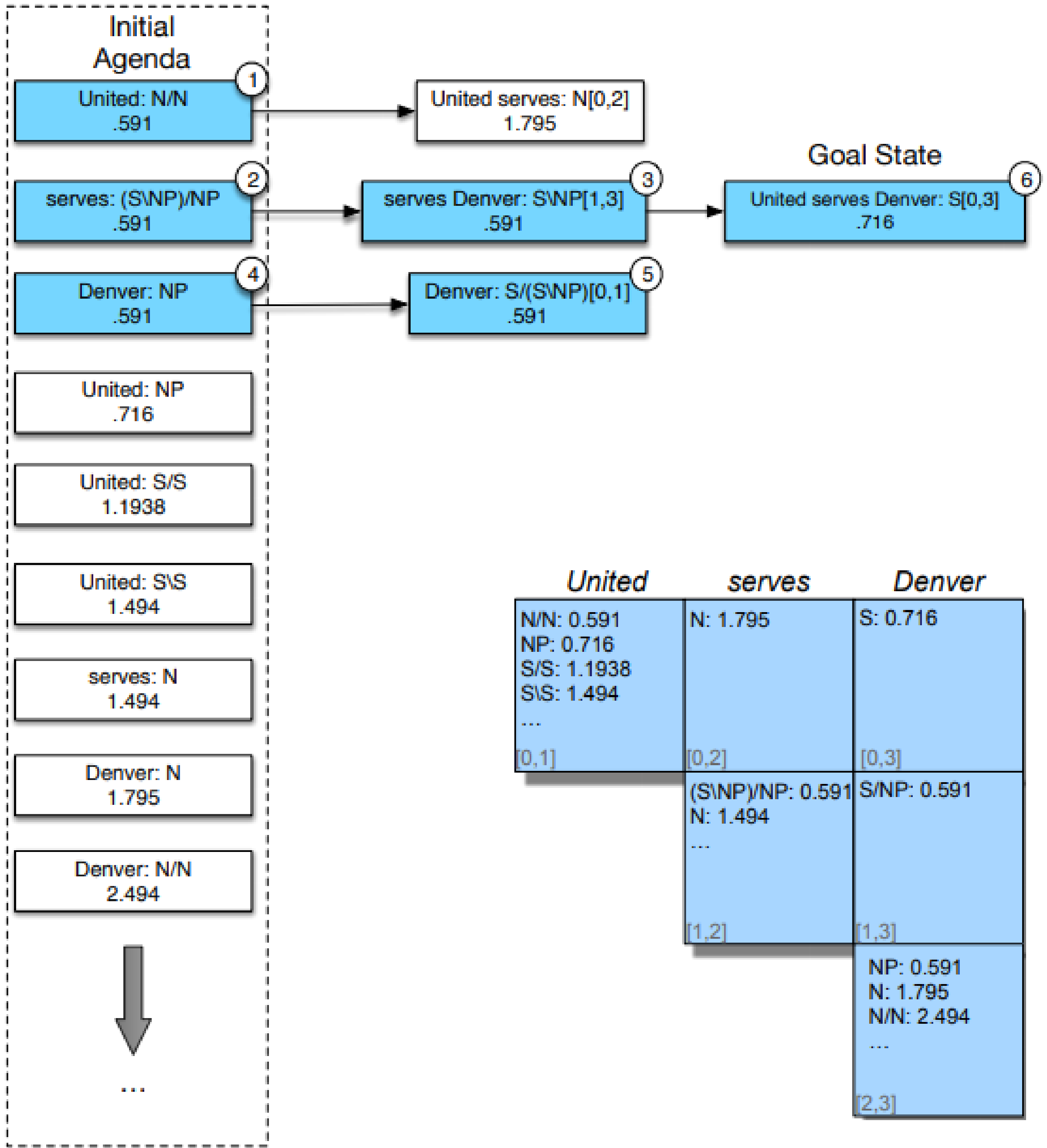
The heuristic

- *For h(n), we need a score that approximates but never overestimates the actual cost of the final derivation.*
- *A simple heuristic that meets this requirement assumes that each of the words in the outside span will be assigned its most probable super-tag. If these are the tags used in the final derivation, then its score will equal the heuristic. If any other tags are used in the final derivation the f-cost will be higher since the new tags must have higher costs, thus guaranteeing that we will not overestimate. Putting this all together, we arrive at the following definition of a suitable f-cost for an edge.*

$$
\begin{aligned}
f(w_{i,j}, t_{i,j}) &= g(w_{i,j}) + h(w_{i,j}) \\
&= \sum_{k=i}^{j} -\log P(t_k | w_k) + \\
&\sum_{k=1}^{i-1} \max_{t \in tags} (-\log P(t | w_k)) + \sum_{k=j+1}^{N} \max_{t \in tags} (-\log P(t | w_k))
\end{aligned}
$$

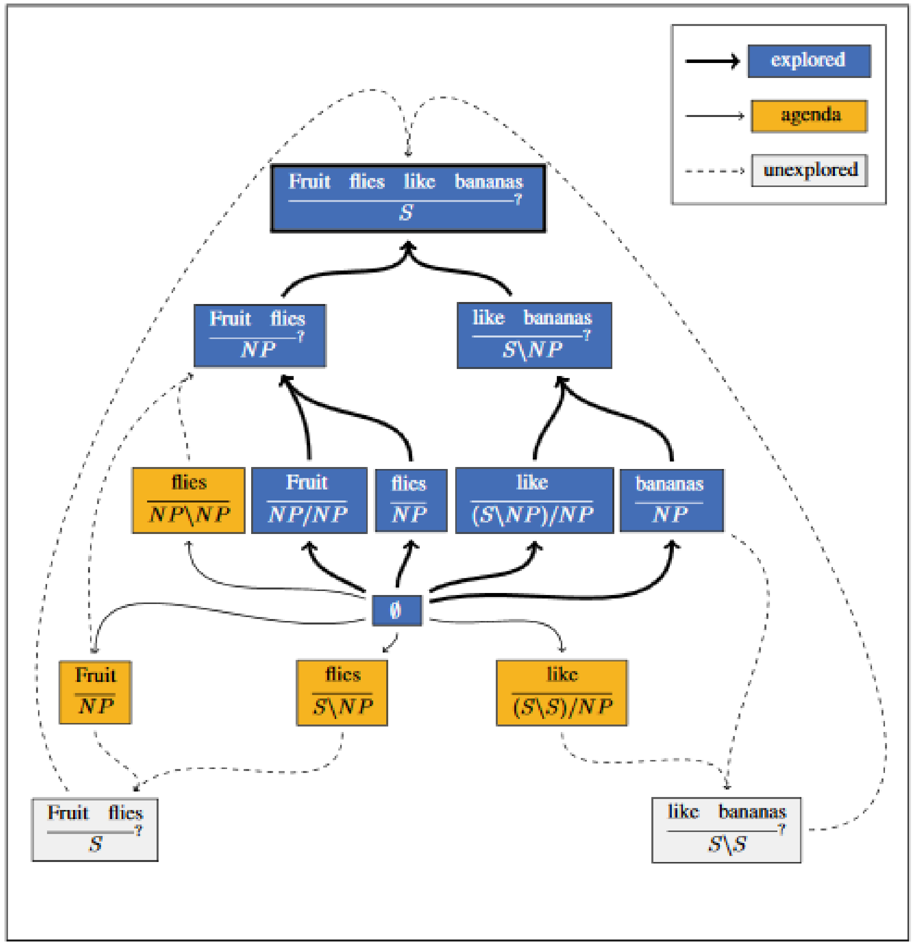
An example

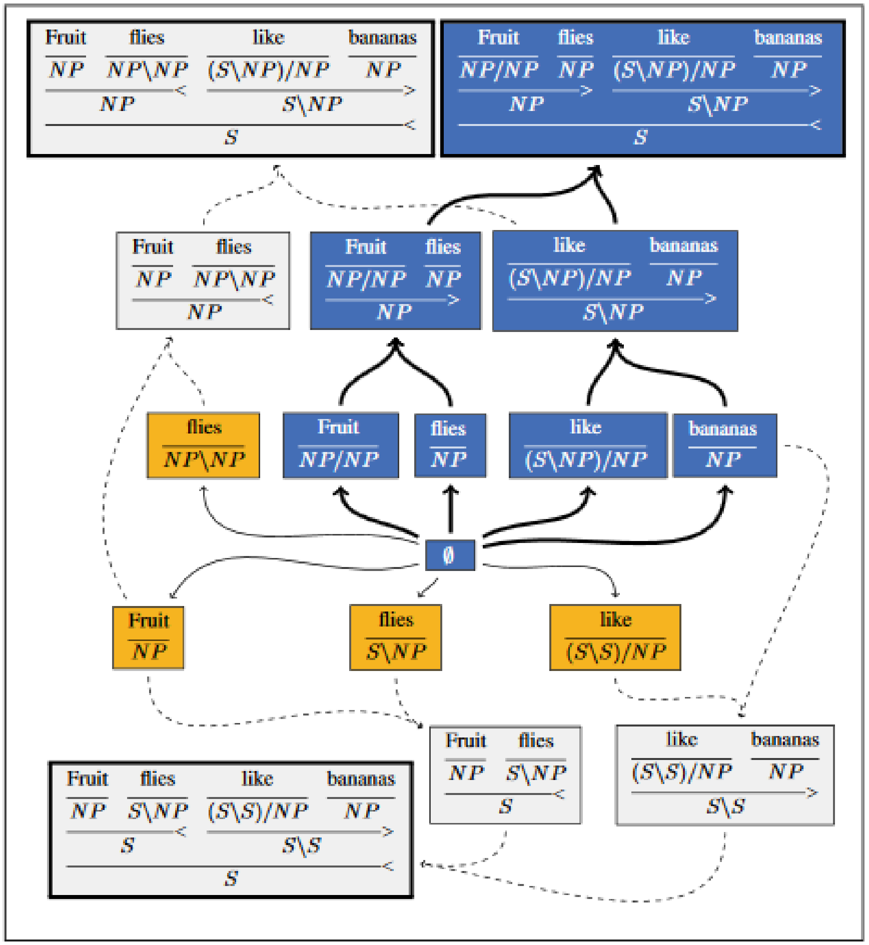
The circled numbers on the white boxes indicate the order in which the states are popped from the agenda



State-of-the-art

- *Global Neural CCG Parsing with Optimality Guarantees (Kenton Lee, Mike Lewis & Luke Zettlemoyer) – Best Paper EMNLP 2016*

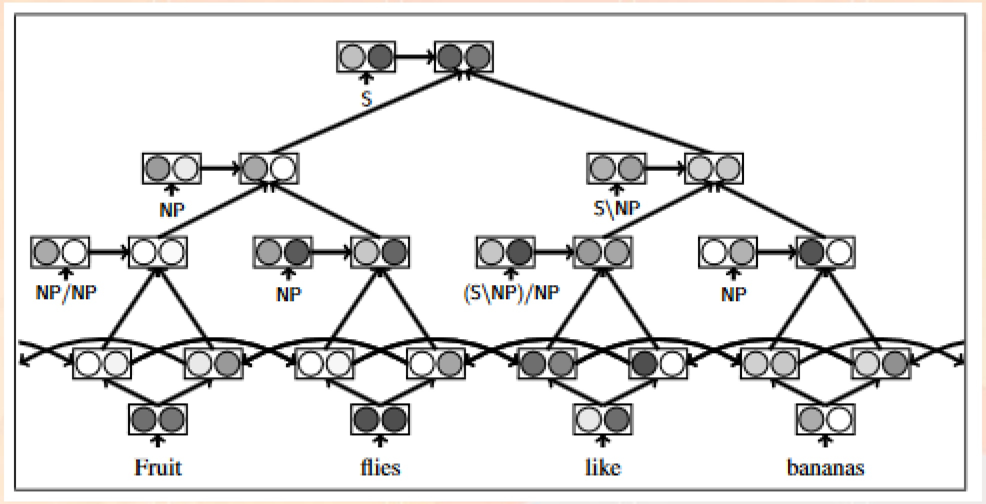




CCG parsing as hypergraph search, showing partial views of the search space. Weighted hyper-edges from child nodes to a parent node represent rule productions scored by a parsing model. A path starting at ∅, for example the set of bolded hyperedges, represents the derivation of a parse. During decoding, we find the highest scoring path to a complete parse.

The Idea:

Use a variant of the Tree-LSTM (*Tai et al., 2015*)connected to a bidirectional LSTM (*Hochreiter and Schmidhuber, 1997*) at the leaves. The combination of linear and tree LSTMs allows the hidden representation of partial parses to condition on both the partial structure and the full sentence.



Visualization of the Tree-LSTM which computes vector embeddings for each parse node. The leaves of the Tree-LSTM are connected to a bidirectional LSTM over words, encoding lexical information within and outside of the parse

Summary

We sketched the basics of probabilistic parsing, concentrating on probabilistic context-free grammars and probabilistic lexicalized context-free grammars.

• Probabilistic grammars assign a probability to a sentence or string of words

• A probabilistic context-free grammar (PCFG) is a context-free grammar in which every rule is annotated with the probability of that rule being chosen.

• The probabilistic CKY (Cocke-Kasami-Younger) algorithm is a probabilistic version of the CKY parsing algorithm.

• PCFG probabilities can be learned by counting in a parsed corpus or by parsing a corpus.

• Raw PCFGs suffer from poor independence assumptions among rules and lack of sensitivity to lexical dependencies.

• One way to deal with this problem is to split and merge non-terminals.

• Probabilistic lexicalized CFGs are another solution to this problem in which the basic PCFG model is augmented with a lexical head for each rule

• Parsers for lexicalized PCFGs (like the Charniak and Collins parsers) are based on extensions to probabilistic CKY parsing.

• Parsers based on probabilistic CCG use the A algorithm for optimal parses.*