Natural Language Processing CSCI 4152/6509 — Lecture 19 Other Types of Grammars: PCFG, DCG

Instructors: Vlado Keselj
Time and date: 14:35 – 15:55, 4-Dec-2025
Location: Studley LSC-Psychology P5260

Previous Lecture

- Natural language syntax:
  - phrase structure, clauses, sentences
  - Parsing, parse tree examples
- Context-Free Grammars review:
  - formal definition
  - inducing a grammar from parse trees
  - derivations, and other notions
- Bracket representation of a parse tree
- Typical phrase structure rules in English:
  - S, NP, VP, PP, ADJP, ADVP

Heads and Dependency

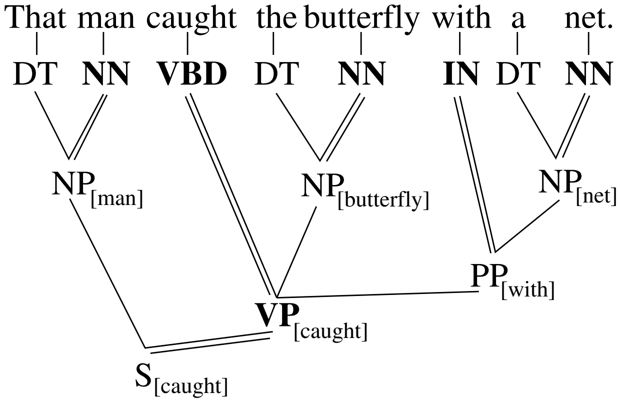
- a phrase typically has a central word called *head*, while other words are direct or indirect *dependents*
- a head is also called a *governor*, although sometimes these concepts are considered somewhat different
- phases are usually called by their head; e.g., the head of a noun phrase is a noun

Example with Heads and Dependencies

That man caught the butterfly with a net.

Example with Heads and Dependencies

- the parse tree of "That man caught the butterfly with a net."
- annotate dependencies, head words



Head-feature Principle

- Head Feature Principle:
  It is a principle that a set of characteristic features of a head word are transferred to the containing phrase.
- Examples of annotating head in a context-free rule:

$$NP \rightarrow DT\ NN_H$$

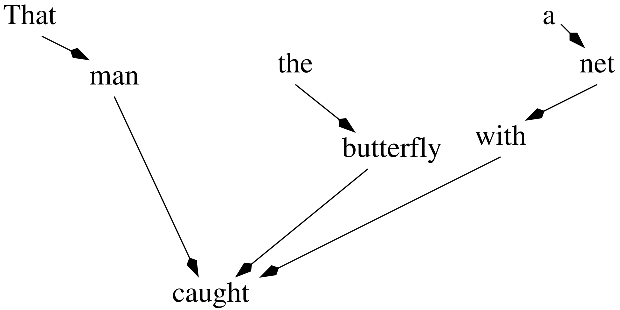
- or

$$[NP] \rightarrow [DT]\ H[NN]$$

- HPSG—Head-driven Phrase Structure Grammars

Dependency Tree

- dependency grammar
- example with “That man caught the butterfly with a net.”



Arguments and Adjuncts

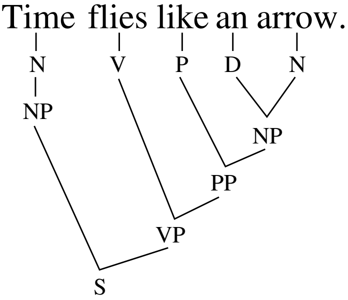
- There ar two kinds of dependents:
  1. **arguments,** which are required dependents, e.g.,
     We deprived him of food.
  2. **adjuncts,** which are not required;
     - ⋆ they have a “less tight” link to the head, and
     - ⋆ can be moved around more easily

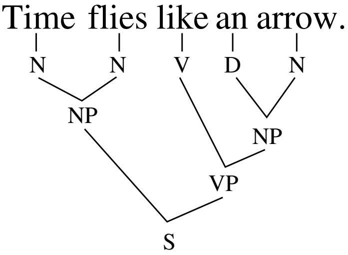
     Example:
     We deprived him of food yesterday in the restaurant.

Probabilistic Context-Free Grammar (PCFG)

- Reading: Chapters 13 and 14
- also known as Stochastic Context-Free Grammar (SCFG)
- Handles ambiguous trees using a probabilistic model

Ambiguity Example





S	$\rightarrow$	NP VP	VP	$\rightarrow$	V NP	N	$\rightarrow$	time	V	$\rightarrow$	like
NP	$\rightarrow$	N	VP	$\rightarrow$	V PP	N	$\rightarrow$	arrow	V	$\rightarrow$	flies
NP	$\rightarrow$	N N	PP	$\rightarrow$	P NP	N	$\rightarrow$	flies	P	$\rightarrow$	like
NP	$\rightarrow$	D N				D	$\rightarrow$	an			

PCFG as a Probabilistic Model

- A generative model based on probabilistic derivation, for example:

$$\mathrm{S} \Rightarrow \mathrm{NP\ VP} \Rightarrow \mathrm{D\ N\ VP} \Rightarrow \ldots$$

- Each step is probabilistic use of one production

Probabilistic Context-Free Grammar Example

$$\begin{array}{rcll} \text{S} & \rightarrow & \text{NP VP} & /1 \\ \text{NP} & \rightarrow & \text{N} & /.4 \\ \text{NP} & \rightarrow & \text{N N} & /.2 \\ \text{NP} & \rightarrow & \text{D N} & /.4 \\ \text{V} & \rightarrow & \text{like} & /.3 \\ \text{V} & \rightarrow & \text{flies} & /.7 \\ \text{P} & \rightarrow & \text{like} & /1 \end{array} \qquad \begin{array}{rcll} \text{VP} & \rightarrow & \text{V NP} & /.5 \\ \text{VP} & \rightarrow & \text{V PP} & /.5 \\ \text{PP} & \rightarrow & \text{P NP} & /1 \end{array} \qquad \begin{array}{rcll} \text{N} & \rightarrow & \text{time} & /.5 \\ \text{N} & \rightarrow & \text{arrow} & /.3 \\ \text{N} & \rightarrow & \text{flies} & /.2 \\ \text{D} & \rightarrow & \text{an} & /1 \end{array}$$

- The following condition must be satisfied for each nonterminal N:

$$\sum_{i=1}^{n} \mathrm{P}(N \rightarrow \alpha_i) = 1$$

Computational Tasks for PCFG Model

- Evaluation

$$P(\text{tree}) = ?$$

- Generation
- Learning
- Inference
  - Marginalization

$$\mathrm{P}(\text{sentence}) = ?$$

  - Conditioning

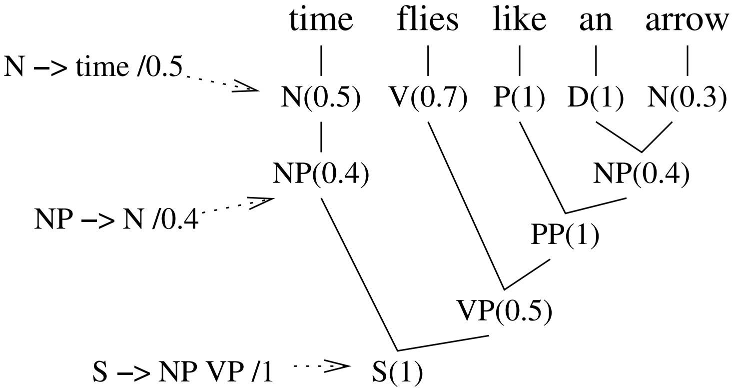
$$\mathrm{P}(\text{tree}|\text{sentence}) = ?$$

  - Completion

$$\arg\max_{\text{tree}} \mathrm{P}(\text{tree}|\text{sentence})$$

Evaluation example: time flies like an arrow (1st meaning)

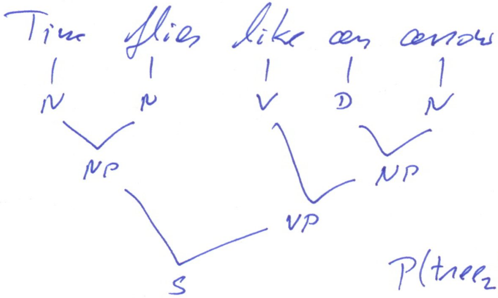
Evaluation



P(tree) = 0.5x0.7x1x1x0.3x0.4x0.4x1x0.5x1=0.0084

Evaluation example: time flies like an arrow (2nd meaning)

Similarly



$P(tree_2) = 0.00036$

Generation (sampling)

$\underline{S} \Rightarrow \underline{NP}\ VP \Rightarrow \underline{N}\ VP \Rightarrow \text{flies}\ \underline{VP} \Rightarrow \ldots$

$S \to NP\ VP / 1$

$NP \to N / 0.4$	$N \to \text{time} / 0.5$
$NP \to N\ N / 0.2$	$N \to \text{arrow} / 0.3$
$NP \to D\ N / 0.4$	$N \to \text{flies} / 0.2$

- choose rule randomly according to the given distribution

Question: Is the process going to stop?

A: Stops with probability 1 if the grammar is proper.

Good News: A grammar learned from a corpus is always proper.

Learning and Inference

Using Prolog to Parse NL

Example: Consider a simple CFG to parse the following two sentences: "the dog runs" and "the dogs run"
The grammar is:

```
S  -> NP VP          N  -> dog
NP -> D N            N  -> dogs
D  -> the            VP -> run
                     VP -> runs
```

`How to parse: the dog runs`

Using Difference Lists: Idea

Consider rule: `S -> NP VP` and sentence [the,dog,runs]

Using Difference Lists to Parse CFG

The problem of parsing using this grammar can be expressed in the following way in Prolog:

```
s(S,R)  :- np(S,I), vp(I, R).
np(S,R) :- d(S,I), n(I,R).
d([the|R], R).
n([dog|R], R).
n([dogs|R], R).
vp([run|R], R).
vp([runs|R], R).
```

Additional Material

Parsing using Difference Lists

Save this in file `parse.prolog`. On Prolog prompt we type: `?- ['parse.prolog'].`

```
% parse.prolog compiled 0.00 sec, 1,888 bytes

Yes
?- s([the,dog,runs],[]).

Yes
?- s([runs,the,dog],[]).

No
```

Basic Definite Clause Grammar (DCG)

- DCG — Prolog built-in mechanism for parsing

Example

```
s  --> np, vp.
np --> d, n.
d  --> [the].
n  --> [dog].
n  --> [dogs].
vp --> [run].
vp --> [runs].
```

Building a Parse Tree

A parse tree can be built in the following way:

```
s(s(Tn,Tv))     --> np(Tn), vp(Tv).
np(np(Td,Tn))   --> d(Td), n(Tn).
d(d(the))       --> [the].
n(n(dog))       --> [dog].
n(n(dogs))      --> [dogs].
vp(vp(run))     --> [run].
vp(vp(runs))    --> [runs].
```

At Prolog prompt we type and obtain:

```
?- s(X, [the, dog, runs], []).
 X = s(np(d(the),n(dog)),vp(runs));
```

Handling Agreement

```
s(s(Tn,Tv))        --> np(Tn,A), vp(Tv,A).
np(np(Td,Tn),A)    --> d(Td), n(Tn,A).
d(d(the))          --> [the].
n(n(dog),sg)       --> [dog].
n(n(dogs),pl)      --> [dogs].
vp(vp(run),pl)     --> [run].
vp(vp(runs),sg)    --> [runs].
```

This grammar will accept sentences "the dog runs" and "the dogs run" but not "the dog run" and "the dogs runs". Other phenomena can be modeled in a similar fashion.

Embedded Code

We can embed additional Prolog code using braces, e.g.:

```
s(T)    --> np(Tn), vp(Tv), {T = s(Tn,Tv)}.
```

and so on, is another way of building the parse tree.

Expressing PCFGs in DCGs

Let us consider the previous example of a PCFG:

S	$\rightarrow$	NP VP	/1	VP	$\rightarrow$	V NP	/.5	N	$\rightarrow$	time	/.5
NP	$\rightarrow$	N	/.4	VP	$\rightarrow$	V PP	/.5	N	$\rightarrow$	arrow	/.3
NP	$\rightarrow$	N N	/.2	PP	$\rightarrow$	P NP	/1	N	$\rightarrow$	flies	/.2
NP	$\rightarrow$	D N	/.4					D	$\rightarrow$	an	/1
V	$\rightarrow$	like	/.3								
V	$\rightarrow$	flies	/.7								
P	$\rightarrow$	like	/1								

The probabilities can be calculated as an additional argument:

```
s(T,P) --> np(T1,P1), vp(T2,P2),
           {T = s(T1,T2), P is P1 * P2 * 1}.
np(T,P) --> n(T1,P1), {T = n(T1), P is P1 * 0.4}.
```

and so on.

Full PCFG Expressed in DCG

```
s(s(Tn,Tv),P) --> np(Tn,P1), vp(Tv,P2), {P is P1 * P2}.
np(np(T),P) --> n(T,P1), {P is P1 * 0.4}.
np(np(T1,T2),P) --> n(T1,P1), n(T2,P2),
                              {P is P1 * P2 * 0.2}.
np(np(Td,Tn),P) --> d(Td,P1), n(Tn,P2),
                              {P is P1 * P2 * 0.4}.
v(v(like), 0.3) --> [like].
v(v(flies), 0.7) --> [flies].
p(p(like), 1.0) --> [like].
vp(vp(Tv,Tn), P) --> v(Tv, P1), np(Tn, P2),
                              {P is P1 * P2 * 0.5}.
vp(vp(Tv,Tp), P) --> v(Tv, P1), pp(Tp, P2),
                              {P is P1 * P2 * 0.5}.
pp(pp(Tp,Tn), P) --> p(Tp, P1), np(Tn, P2),
                                   {P is P1 * P2}.
n(n(time), 0.5) --> [time].
n(n(arrow), 0.3) --> [arrow].                ...
```

Example Run in Prolog Interpreter

?- `s(T,P,[time,flies,like,an,arrow],[]).`
the interpreter would reply with: `T = s(np(n(time)),`
```
      vp(v(flies), pp(p(like), np(d(an), n(arrow)))))
P = 0.0084
```
and after typing ; (semi-colon), we get: `T = s(np(n(time), n(flies)),`
```
      vp(v(like), np(d(an), n(arrow))))
P = 0.00036
```
After typing second ';', the interpreter reports 'No' since there are no more parse trees.

Efficient Inference in PCFG Model

- Using backtracking is not efficient approach
- Chart parsing is an efficient approach
- We will take a look at the CYK chart parsing algorithm

CYK Chart Parsing Algorithm

- When parsing NLP, there are generally two approaches:
  1. Backtracking to find all parse trees
  2. Chart parsing
- CYK algorithm: a simple chart parsing algorithm
- CYK: Cocke-Younger-Kasami algorithm
- CYK can be applied only to a CNF grammar

Chomsky Normal Form

- all rules are in one of the forms:
  1. $A \rightarrow B\,C$, where A, B, and C are nonterminals, or
  2. $A \rightarrow w$, where A is a nonterminal and w is a terminal
- If a grammar is not in CNF, it can be converted to it

Is the following grammar in CNF?

$$
\begin{array}{rcl}
\mathrm{S} & \rightarrow & \mathrm{NP\ VP} \\
\mathrm{NP} & \rightarrow & \mathrm{N} \\
\mathrm{NP} & \rightarrow & \mathrm{N\ N} \\
\mathrm{NP} & \rightarrow & \mathrm{D\ N}
\end{array}
\quad
\begin{array}{rcl}
\mathrm{VP} & \rightarrow & \mathrm{V\ NP} \\
\mathrm{VP} & \rightarrow & \mathrm{V\ PP} \\
\mathrm{PP} & \rightarrow & \mathrm{P\ NP}
\end{array}
\quad
\begin{array}{rcl}
\mathrm{N} & \rightarrow & \mathrm{time} \\
\mathrm{N} & \rightarrow & \mathrm{arrow} \\
\mathrm{N} & \rightarrow & \mathrm{flies} \\
\mathrm{D} & \rightarrow & \mathrm{an}
\end{array}
\quad
\begin{array}{rcl}
\mathrm{V} & \rightarrow & \mathrm{like} \\
\mathrm{V} & \rightarrow & \mathrm{flies} \\
\mathrm{P} & \rightarrow & \mathrm{like}
\end{array}
$$

How about this grammar? (Is it in CNF?)

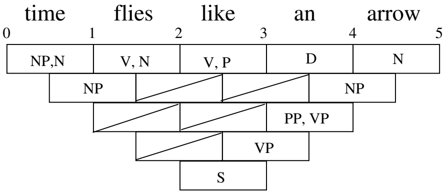
$$\begin{array}{lll} S & \rightarrow & NP\ VP \\ NP & \rightarrow & \text{time} \\ NP & \rightarrow & N\ N \\ NP & \rightarrow & D\ N \end{array} \quad \begin{array}{lll} VP & \rightarrow & V\ NP \\ VP & \rightarrow & V\ PP \\ PP & \rightarrow & P\ NP \\ & & \end{array} \quad \begin{array}{lll} N & \rightarrow & \text{time} \\ N & \rightarrow & \text{arrow} \\ N & \rightarrow & \text{flies} \\ D & \rightarrow & \text{an} \end{array} \quad \begin{array}{lll} V & \rightarrow & \text{like} \\ V & \rightarrow & \text{flies} \\ P & \rightarrow & \text{like} \\ & & \end{array}$$

CYK Example: time flies like an arrow

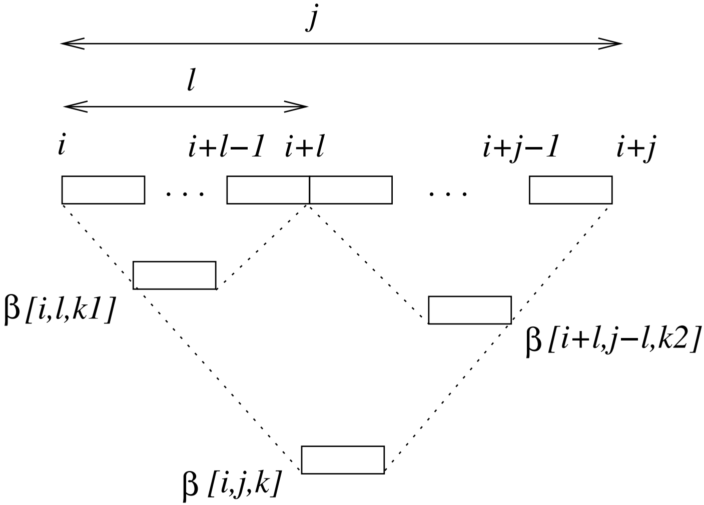
CYK Example

The following grammar in CNF is given:

S	$\rightarrow$	NP VP	VP	$\rightarrow$	V NP	N	$\rightarrow$	time	V	$\rightarrow$	like
NP	$\rightarrow$	time	VP	$\rightarrow$	V PP	N	$\rightarrow$	arrow	V	$\rightarrow$	flies
NP	$\rightarrow$	N N	PP	$\rightarrow$	P NP	N	$\rightarrow$	flies	P	$\rightarrow$	like
NP	$\rightarrow$	D N				D	$\rightarrow$	an			



Explanation of Index Use in CYK



CYK Algorithm

Require: sentence $= w_1 \dots w_n$, and a CFG in CNF with nonterminals $N^1 \dots N^m$,
N^1 is the start symbol

Ensure: parsed sentence

allocate matrix $\beta \in \{0,1\}^{n \times n \times m}$ and initialize all entries to 0
for $i \leftarrow 1$ to n **do**
    for all rules $N^k \rightarrow w_i$ **do**
        $\beta[i,1,k] \leftarrow 1$
for $j \leftarrow 2$ to n **do**
    for $i \leftarrow 1$ to $n-j+1$ **do**
        for $l \leftarrow 1$ to $j-1$ **do**
            for all rules $N^k \rightarrow N^{k_1} N^{k_2}$ **do**
                $\beta[i,j,k] \leftarrow \beta[i,j,k]$ OR $(\beta[i,l,k_1]$ AND $\beta[i+l,j-l,k_2])$
return $\beta[1,n,1]$

Efficient Inference in PCFG Model

- consider marginalization task:

$\mathrm{P}(\text{sentence}) =?$

- or: $\mathrm{P}(\text{sentence}) = \mathrm{P}(w_1 w_2 \ldots w_n | S)$
- One way to compute:

$$\mathrm{P}(\text{sentence}) = \sum_{t \in T} \mathrm{P}(t),$$

- Likely inefficient; need a parsing algorithm

Efficient PCFG Marginalization

- Idea: adapt CYK algorithm to store marginal probabilities
- Replace algorithm line:

  $$\beta[i,j,k] \leftarrow \beta[i,j,k] \text{ OR } (\beta[i,l,k_1] \text{ AND } \beta[i+l,j-l,k_2])$$

  with

  $$\beta[i,j,k] \leftarrow \beta[i,j,k] + \mathrm{P}(N^k \to N^{k_1} N^{k_2}) \cdot \beta[i,l,k_1] \cdot \beta[i+l,j-l,k_2]$$

- and the first-chart-row line:

  $$\beta[i,1,k] \leftarrow 1$$

  with

  $$\beta[i,1,k] \leftarrow \mathrm{P}(N^k \to w_i)$$

Probabilistic CYK for Marginalization

Require: sentence $= w_1 \dots w_n$, and a PCFG in CNF with nonterminals $N^1 \dots N^m$, N^1 is the start symbol

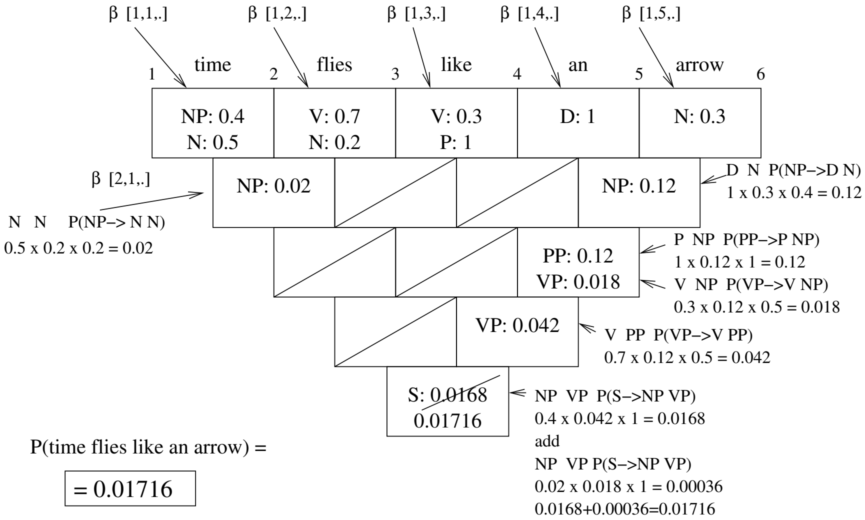
Ensure: P(sentence) is returned

```
allocate β ∈ ℝ^{n×n×m} and initialize all entries to 0
for i ← 1 to n do
    for all rules N^k → w_i do
        β[i,1,k] ← P(N^k → w_i)
for j ← 2 to n do
    for i ← 1 to n − j + 1 do
        for l ← 1 to j − 1 do
            for all rules N^k → N^{k1} N^{k2} do
                β[i,j,k] ← β[i,j,k]+
                    P(N^k → N^{k1} N^{k2}) · β[i,l,k1] · β[i+l, j−l, k2]
return β[1,n,1]
```

PCFG Marginalization Example (grammar)

$$
\begin{array}{rcll}
S & \to & NP\ VP & /1 \\
NP & \to & \text{time} & /.4 \\
NP & \to & N\ N & /.2 \\
NP & \to & D\ N & /.4 \\
V & \to & \text{like} & /.3 \\
V & \to & \text{flies} & /.7 \\
P & \to & \text{like} & /1
\end{array}
\quad
\begin{array}{rcll}
VP & \to & V\ NP & /.5 \\
VP & \to & V\ PP & /.5 \\
PP & \to & P\ NP & /1
\end{array}
\quad
\begin{array}{rcll}
N & \to & \text{time} & /.5 \\
N & \to & \text{arrow} & /.3 \\
N & \to & \text{flies} & /.2 \\
D & \to & \text{an} & /1
\end{array}
$$

PCFG Marginalization Example (chart)



Conditioning

- Conditioning in the PCFG model: $\mathrm{P}(\mathrm{tree}|\mathrm{sentence})$
- Use the formula:

$$\mathrm{P}(\mathrm{tree}|\mathrm{sentence}) = \frac{\mathrm{P}(\mathrm{tree}, \mathrm{sentence})}{\mathrm{P}(\mathrm{sentence})} = \frac{\mathrm{P}(\mathrm{tree})}{\mathrm{P}(\mathrm{sentence})}$$

- $\mathrm{P}(\mathrm{tree})$ — directly evaluated
- $\mathrm{P}(\mathrm{sentence})$ — marginalization

Completion

- Finding the most likely parse tree of a sentence:

$$\arg\max_{\text{tree}} \mathrm{P}(\text{tree}|\text{sentence})$$

- Use the CYK algorithm in which line 9 is replaced with:

  9: $\beta[i,j,k] \leftarrow \max(\beta[i,j,k], \mathrm{P}(N^k \rightarrow N^{k_1} N^{k_2}) \cdot \beta[i,l,k_1] \cdot \beta[i+l, j-l, k_2])$

- Return the most likely tree

CYK-based Completion Algorithm

Require: sentence $= w_1 \ldots w_n$, and a PCFG in CNF with nonterminals $N^1 \ldots N^m$, N^1 is the start symbol

Ensure: The most likely parse tree is returned

allocate $\beta \in \mathbb{R}^{n \times n \times m}$ and initialize all entries to 0

for $i \leftarrow 1$ to n **do**

  for all rules $N^k \rightarrow w_i$ **do**

    $\beta[i,1,k] \leftarrow \mathrm{P}(N^k \rightarrow w_i)$

for $j \leftarrow 2$ to n **do**

  for $i \leftarrow 1$ to $n - j + 1$ **do**

    for $l \leftarrow 1$ to $j - 1$ **do**

      for all rules $N^k \rightarrow N^{k_1} N^{k_2}$ **do**

        $\beta[i,j,k] \leftarrow \max(\beta[i,j,k], \mathrm{P}(N^k \rightarrow N^{k_1} N^{k_2}) \cdot \beta[i,l,k_1] \cdot \beta[i+l, j-l, k_2])$

return Reconstruct$(1, n, 1, \beta)$

Algorithm: Reconstruct(i, j, k, β)

Require: β — table from CYK, i — index of the first word, j — length of sub-string sentence, k — index of non-terminal

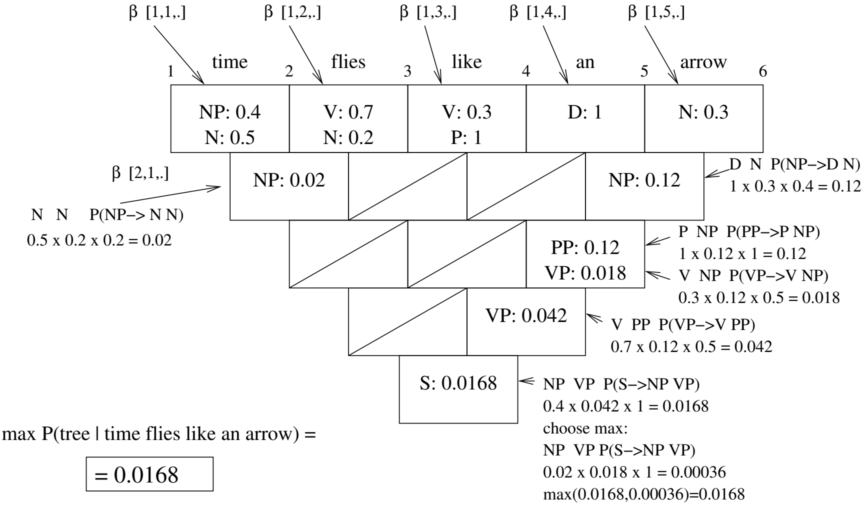
Ensure: a most probable tree with root N^k and leaves $w_i \dots w_{i+j-1}$ is returned

if $j = 1$ **then**
    return tree with root N^k and child w_i
for $l \leftarrow 1$ to $j - 1$ **do**
    for all rules $N^k \rightarrow N^{k_1} N^{k_2}$ **do**
        if $\beta[i, j, k] = \mathrm{P}(N^k \rightarrow N^{k_1} N^{k_2}) \cdot \beta[i, l, k_1] \cdot \beta[i + l, j - l, k_2]$ **then**
            create a tree t with root N^k
            t.*left_child* $\leftarrow \mathrm{Reconstruct}(i, l, k_1, \beta)$
            t.*right_child* $\leftarrow \mathrm{Reconstruct}(i + l, j - l, k_2, \beta)$
            return t

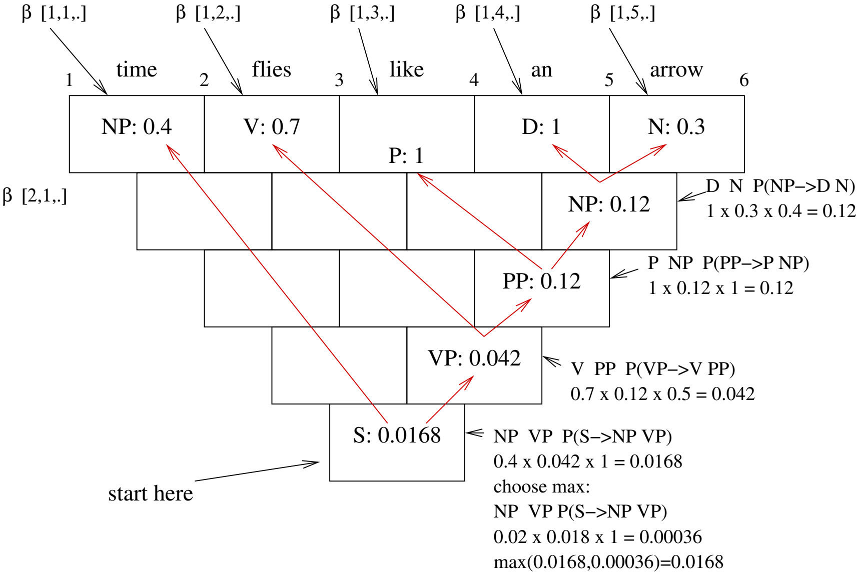
PCFG Completion Example (grammar)

$$
\begin{array}{rcll}
S & \to & NP\ VP & /1 \\
NP & \to & \text{time} & /.4 \\
NP & \to & N\ N & /.2 \\
NP & \to & D\ N & /.4 \\
V & \to & \text{like} & /.3 \\
V & \to & \text{flies} & /.7 \\
P & \to & \text{like} & /1
\end{array}
\qquad
\begin{array}{rcll}
VP & \to & V\ NP & /.5 \\
VP & \to & V\ PP & /.5 \\
PP & \to & P\ NP & /1
\end{array}
\qquad
\begin{array}{rcll}
N & \to & \text{time} & /.5 \\
N & \to & \text{arrow} & /.3 \\
N & \to & \text{flies} & /.2 \\
D & \to & \text{an} & /1
\end{array}
$$

PCFG Completion Example (chart)

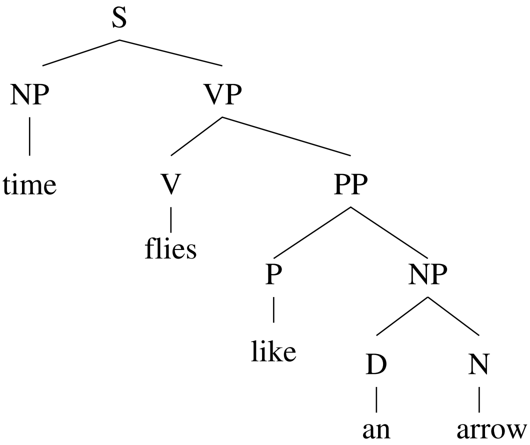


PCFG Completion Example (tree reconstruction)



PCFG Completion Example (final tree)

The most probable three:



Issues with PCFGs

1. Structural dependencies
   - Dependency on position in a tree
   - Example: consider rules NP $\rightarrow$ PRP and NP $\rightarrow$ DT NN
   - PRP is more likely as a subject than an object
   - NL parse trees are usually deeper on their right side
2. Lexical dependencies
   - Example: PP-attachment problem
   - In a PCFG, decided using probabilities for higher level rules; e.g., $\mathrm{NP} \rightarrow \mathrm{NP\ PP}$, $\mathrm{VP} \rightarrow \mathrm{VBD\ NP}$, and $\mathrm{VP} \rightarrow \mathrm{VBD\ NP\ PP}$
   - Actually, they frequently depend on the actual words

PP-Attachment Example

- Consider sentences:
  - "Workers dumped sacks into a bin." and
  - "Workers dumped sacks of fish."
- and rules:
  - $\mathrm{NP} \rightarrow \mathrm{NP\ PP}$
  - $\mathrm{VP} \rightarrow \mathrm{VBD\ NP}$
  - $\mathrm{VP} \rightarrow \mathrm{VBD\ NP\ PP}$

A Solution: Probabilistic Lexicalized CFGs

- use heads of phrases
- expanded set of rules, e.g.:
  VP(dumped) $\rightarrow$ VBD(dumped) NP(sacks) PP(into)
- large number of new rules
- sparse data problem
- solution: new independence assumptions
- proposed solutions by Charniak, Collins, etc. around 1999