Faculty of Computer Science, Dalhousie University *30-Nov-2023*

CSCI 4152/6509 — Natural Language Processing

Lecture 24: Typical Phrase Structure of English

Location: Rowe 1011 Instructor: Vlado Keselj
Time: 16:05 – 17:25

Previous Lecture

- Definite Clause Grammar (DCG)
  - Basic DCG example
  - Building a parse tree in DCG
  - Agreement example in DCG
  - Embedded code in DCG
- Probabilistic Context-Free Grammars (PCFG)
- PCFG definition
- PCFG as a probabilistic model

23 Context-Free Grammars for Natural Language

23.1 Typical Phrase Structure Rules in English

Typical Phrase Structure Rules in English

- We will cover some typical phrase structure rules
- Specific to English but also generalizable to other languages
- **Not** *all rules are covered,* but the general principles should be adopted

Typical Sentence Rules (S)

```
S -> NP VP          Declarative sentences, e.g.:
              I want a flight from Halifax to Chicago.
              The flight should be eleven a.m. tomorrow.

S -> VP              Imperative sentences, e.g.:
                     Show the lowest fare.
                     Delete the file.

S -> Aux NP VP       Yes-no questions, e.g.:
                   Do any of these flights have stops?
                   Can you give me some information for United?
                                          (pragmatic meaning)

S -> Wh-NP VP      Wh-subject questions, e.g.:
                   What airlines fly from Halifax?
                   Whose flights serve breakfast.
```

```
S -> Wh-NP Aux NP VP   Wh-non-subject questions, e.g.:
                       What flights do you have on Tuesday?
```

Noun Phrase (NP)

- typically: pronouns, proper nouns, or determiner-nominal construction
- some typical rules

```
NP -> PRP                      e.g.: you
NP -> NNP | NNPS               e.g.: Halifax
NP -> PDT? DT JJ* NN PP*
NP -> NN NN                    e.g.: computer science
```

- in the last rule, we use regular expression notation to describe a set of different rules
- example: all the various flights from Halifax to Toronto
- example 2: all the thick red books on the shelves in the library
- determiners and nominals
- modifiers before head noun and after head noun
- postmodifier phrases

```
NP -> DT JJ* NN RelC
```

Examples of determiners: *a* stop, *the* flights, *this* flight

The determiners can be more complex; e.g., they can consist of a noun phrase and posessive ending *'s*, such as "United's flight" and "Denver's mayor's mother's canceled flight".

However, we will not label those structures as determiners in a context-free parse trees.

Relative Clauses

- RelC — relative clause
- clause (sentence-like phrase) following a noun phrase
- example: gerundive relative clause: flights arriving after 5pm
- example: infinitive relative clause: flights to arrive tomorrow
- example: restrictive relative clause: flight that was canceled yesterday

Verb Phrase (VP)

- organizes arguments around the verb
- typical rules

```
VP -> Verb             intransitive verbs;
                       e.g.: disappear
VP -> Verb NP          transitive verbs:
                       e.g.: prefer a morning flight
VP -> Verb NP NP       ditransitive verbs:
                       e.g.: send me an email
VP -> Verb PP*         sentential complements
VP -> Verb NP PP*
VP -> Verb NP NP PP*
```

- sentential complements, e.g.: You said these were two flights that were the cheapest.

Prepositional Phrase (PP)

- Preposition (IN) relates a noun phrase to other word or phrase
- Prepositional Phrase (PP) consists of a preposition and the noun phrase which is an object of that preposition
- There is typically only one rule for the prepositional phrase:

```
PP -> IN NP
```

- examples: from Halifax, before tomorrow, in the city
- PP-attachment ambiguity

Adjective Phrase (ADJP)

- less common
- examples:
  - She is very sure of herself.
  - . . . the least expensive fare . . .

Adverbial Phrase (ADVP)

- Example:

```
(S (NP preliminary findings)
   (VP were reported
       (ADVP (NP a year) ago)))
```

- another example: years ago

Overview of Syntactic Tags

Tag	Description
S	Sentence
NP	Noun Phrase
VP	Verb Phrase
PP	Prepositional Phrase
ADJP	Adjective Phrase
ADVP	Adverbial Phrase

About Typical Rules

- Only some typical rules are presented
- For example: We see the cat, and you see a dog.
- The sentence could be described with: `S -> S CC S`
- Relative clauses are labeled in Penn treebank using SBAR ($\bar{S}$) non-terminal; e.g.:

```
(S (NP (NP Lorillard Inc.)
       ,
       (NP (NP the unit)
           (PP of (NP (ADJP New York-based)
                      Loews Corp.)))
       (SBAR that
             (S (NP *gap*)
```

```
                (VP makes (NP Kent cigarettes))))
    ,)
(VP stopped (VP using (NP crocidolite))))
```

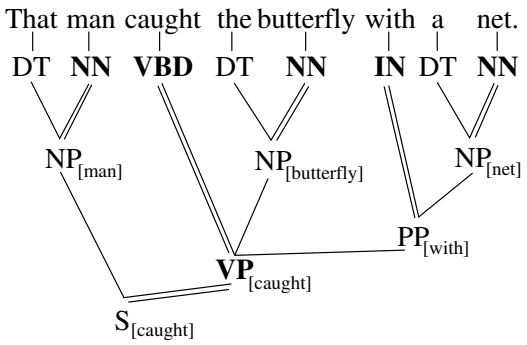
24 Heads and Dependency

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

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



Head Feature Principle

The **Head Feature Principle** is the principle that a constituent in a parse tree will have the same set of some characteristic features as its head child. The features that are shared in this way between parent and head child are called **head features**.

The head feature principle is particularly emphasized in some grammar formalisms, such as the Head-driven Phrase Structure Grammar (HPSG).

If we use a context-free grammar, or a similar grammar formalism, to parse natural language, an interesting question is how to detect head-dependent relations. We can achieve it by annotating head child in each context-free rule. Sometimes in literature, these heads are labeled on the right-hand side of each rule using a subscript H, as in:

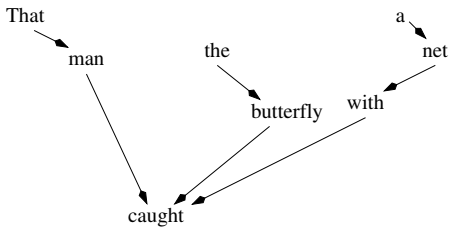
$$NP \rightarrow DT\ NN_H$$

or, using a Head-driven Phrase Structure Grammar (HPSG) style, they are labeled using a letter H before the child constituent, as in:

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

Dependency Tree

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



A dependency tree consists of direct dependence relations between words in a sentences, and the words are the nodes of the tree. A typed dependency tree also includes labels of the edges, describing the types of dependencies. A dependency tree can be easily produced from a context-free parse tree in which the heads of the phrases are labelled.

A dependency grammar is a grammar that defines rules that are used to form a dependency tree.

Arguments and Adjuncts

- There ar two kinds of dependents:
  1. **arguments,** which are required dependents, e.g.,
     We deprived him <u>of food.</u>
  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 <u>yesterday</u> <u>in the restaurant</u>.

25 CYK Chart Parsing Algorithm

Slide notes:

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

The CYK algorithm (Cocke-Younger-Kasami) is a well-known efficient parsing algorithm. The algorithm has a running-time complexity of $O(n^3)$ for a sentence of length n.

CYK can be applied only to a CNF (Chomsky Normal Form) grammar, so if the grammar is not already in CNF, we would have to convert it to CNF. A Context-Free Grammar is in CNF if all its rules are either of the form $A \rightarrow B\,C$, where A, B, and C are nonterminals, or $A \rightarrow w$, where A is a nonterminal and w is a terminal. If a CFG is not in CNF, it can be converted into CNF.

Chomsky Normal Form

- all rules are in one of the forms:
  1. $A \to B\,C$, where A, B, and C are nonterminals, or
  2. $A \to w$, 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?

S	→	NP VP	VP	→	V NP	N	→	time	V	→	like
NP	→	N	VP	→	V PP	N	→	arrow	V	→	flies
NP	→	N N	PP	→	P NP	N	→	flies	P	→	like
NP	→	D N				D	→	an			

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

S	→	NP VP	VP	→	V NP	N	→	time	V	→	like
NP	→	time	VP	→	V PP	N	→	arrow	V	→	flies
NP	→	N N	PP	→	P NP	N	→	flies	P	→	like
NP	→	D N				D	→	an			

Note: What if the grammar is not in CNF

There is a standard algorithm for converting arbitrary CFG into CNF. The problem is: How to calculate probabilities of the rules in the new grammar? One way is to sample from the old grammar, and to estimate probabilities in the new grammar by parsing the sample sentences and counting. The probabilities can also be calculated directly, but it is not a straightforward task.

Here are the steps needed for conversion of an arbitrary CFG into CNF. This is just a partial sketch of the algorithm: calculating probabilities of the new rules in the first two steps is not trivial and it is not given.

Eliminate empty rules $N \to \epsilon$

Find all "nullable" nonterminals, i.e., terminals N such that $N \Rightarrow^* \epsilon$.

From each rule $A \to X_1 \ldots X_n$ create new rules by striking out some nullable nonterminals. This is done for all combination of nonterminals in the rule, except for striking out all $X_1 \ldots X_n$ if they are all nullable.

Remove empty rules.

If the start symbol is nullable, add $S \to \epsilon$, and treat that as a special case.

Eliminate unit rules $N \to M$

For any two variables A and B, such that $A \Rightarrow^* B$, for all non-unit rules $B \to \zeta$, we add $A \to \zeta$. Remove unit rules.

All possible derivations $A \Rightarrow^* B$ are easy to find since the empty rules are already eliminated.

Eliminate terminals in rules, except $A \to w$

For each terminal w that appears on the right hand side of some rule with some other symbols, we introduce a new nonterminal N_w, and a rule $N_w \to w$ with probability 1. Then we replace w in all other rules with N_w.

Eliminate rules $A \to B_1 B_2 \ldots B_n$ **(**$n > 2$**)**

For each rule $A \to B_1 B_2 \ldots B_n$ ($n > 2$), we introduce $n-2$ new nonterminals $X_1, \ldots, X_{n-2}$, and replace this rule with the following rules: $A \to B_1 X_1$, $X_1 \to B_2 X_2, \ldots X_{n-2} \to B_{n-1} B_n$, and assign the following probabilities to them: $\mathrm{P}(A \to B_1 X_1) = \mathrm{P}(A \to B_1 B_2 \ldots B_n)$, $\mathrm{P}(X_1 \to B_2 X_2) = 1, \ldots \mathrm{P}(X_{n-2} \to B_{n-1} B_n) = 1$.

CYK Example

Let us first show the CYK algorithm on an example, before presenting it in detail. We assume that the following grammar in CNF is given:

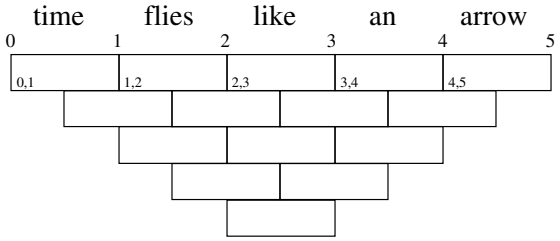
S	→	NP VP	VP	→	V NP	N	→	time	V	→	like
NP	→	time	VP	→	V PP	N	→	arrow	V	→	flies
NP	→	N N	PP	→	P NP	N	→	flies	P	→	like
NP	→	D N				D	→	an			

CYK Example (continued): time flies like an arrow

Using this grammar, we want to parse the sentence 'time flies like an arrow'. We first tokenize the sentence by breaking into the words, and we will enumerate all boundaries between words, starting with the start of the first word in the following way:

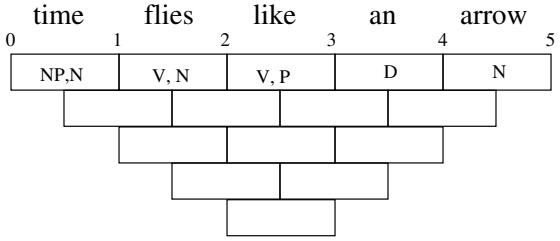
$_0$ time $_1$ flies $_2$ like $_3$ an $_4$ arrow $_5$

We organize parsing around a table, called **parsing chart** or simply **chart**, as follows:

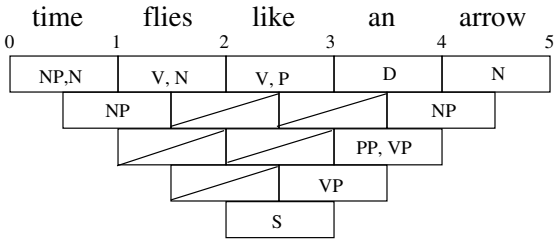


Each entry of the parsing chart is called a **chart entry**. In the empty chart above, we added numbers in the first row of chart entries, which denote spans that are associated with the entries. The first row of chart entries have all spans of lenght 1.

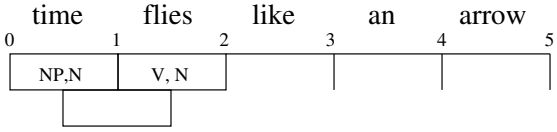
To fill the first row of the table we look at the words and all rules of the form $N \to w$, where w is the appropriate word, and we add the corresponding non-terminal N to the appropriate chart entry. This is how we fill the first row of the chart and obtain the following:



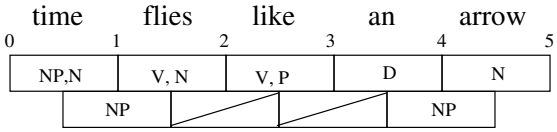
Completely filled chart looks as follows:



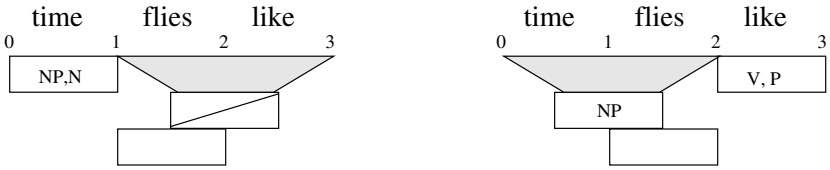
To fill the second row, we look at the span covered by the chart entry. For example, the first chart entry in the second row, covers the span 0–2, as shown below:



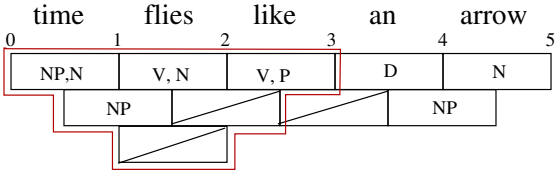
By finding all pairs of non-terminals from the two chart entries that cover sub-spans 0–1 and 1–2, we find pairs:
NP V
NP N
N V
N N
and then we look for the rules, for which one of these sequences is on the right-hand side. We can find the following rule:
NP $\rightarrow$ N N
and we add the non-terminal NP to the entry for the span 0–2. In some cases, we will not find any matching rules and we obtain empty chart entries, which are marked by one crossing diagonal:



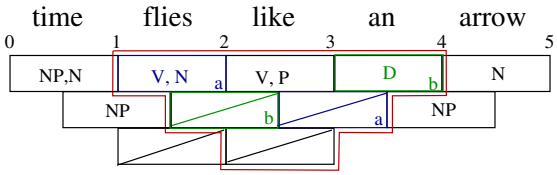
We fill out similarly rows after 2, but it gets a bit more complicated since we will have more than one pair of sub-spans that correspond to a chart entry. For example, the first entry of the third row covers the span 0–3, and then we need to look at the pairs of entries for span 0–1 and 1–3, and 0–2 and 2–3, as shown:



In the first case, when we look at entries ^{0}[NP, N]^{1} and ^{1}[ ]^{3} we do not get any pairs of non terminals. In the second case, we look at the entries ^{0}[NP]^{2} and ^{2}[V, P]^{3} and get the following pairs of non-terminals:
NP V
NP P
for which we cannot find any appropriate rules, whose right-hand sides correspond to these pairs. This is why the entry ^{0}[ ]^{3} remains empty, as follows:



Similarly, we fill the next entry covering the span 1–4 by looking at the pairs of entries covering spans 1–2 and 2–4 (labeled with 'a' in the figure below), and the pair 1–3 and 3–4 (labeled with 'b'):

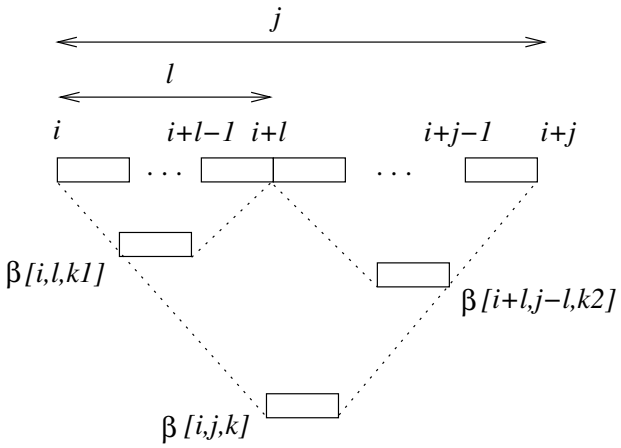


And we continue the process in this way.

Implementation

The example implies that we need to use a two-dimensional table to store chart entries. Using a two-dimensional table is a possible solution, but in that case the table entries would be quite complex since each of them needs to store a set of non-terminals. To make the solution simpler, we can use a three-dimensional table, such that the third dimension corresponds to all different non-terminals.

Explanation of Index Use in CYK



CYK Algorithm

Let all nonterminals be: $N^1, \ldots N^m$.

In the standard CYK algorithm, we have a two dimensional table β in which only the entries $\beta_{ij}, 1 \leq i \leq i+j-1 \leq n$, are used. Each entry β_{ij} contains a set of nonterminals that can produce substring $w_i \ldots w_{i+j-1}$ using the grammar rules, i.e., $\beta_{ij} = \{N | N \Rightarrow^* w_i \ldots w_{i+j-1}\}$.

If we enumerate all nonterminals: N^1, N^2, …, N^m, then each set of nonterminals β_{ij} can be represented by extending β to be a 3-dimensional table β_{ijk}, in which $\beta_{ijk} = 1$ means that N^k can produce substring $w_i \ldots w_{i+j-1}$, and $\beta_{ijk} = 0$ that it cannot.

The line $\beta[i,j,k] \leftarrow \beta[i,j,k]$ OR $(\beta[i,l,k_1]$ AND $\beta[i+l, j-l, k_2])$ in the algorithm is essentially is a shorthand expression for:

if $\beta[i,l,k_1]$ AND $\beta[i+l, j-l, k_2]$ **then**
  $\beta[i,j,k] \leftarrow 1$

Algorithm 1 CYK Parsing 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 $\beta[1,n,1]$

26 Efficient Inference in PCFG Model

Efficient Inference in PCFG Model

Let us consider the marginalization task:

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

If 'sentence' is the following sequence of words: $w_1 w_2 \dots w_n$, then $\mathrm{P}(\text{sentence})$ is the following conditional probability:

$$\mathrm{P}(\text{sentence}) = \mathrm{P}(w_1 w_2 \dots w_n | S)$$

i.e., it is the probability of generating the sentence given that we start from S, i.e, it is $\mathrm{P}(S \Rightarrow^* w_1 \dots w_n)$.

An obvious way to calculate this marginal probability is to find all parse trees of a sentence and sum their probabilities, i.e:

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

where T is the set of all parse trees of the sentence 'sentence'. However, this may be very inefficient. We also need a way to find all parse trees.

As an example illustrating that the above direct approach may lead to an exponential algorithm, consider a CFG with only two rules $S \Rightarrow S\ S$ and $S \Rightarrow a$. The sentences a^n have as many parse trees as there are binary trees with n leaves, which is a well-known Catalan number, $\approx \frac{4^n}{n^{3/2}\sqrt{\pi}}$ as $n \rightarrow \infty$.

An algorithm for efficient marginalization can be derived from the CYK algorithm.

PCFG Marginalization

The CKY algorithms is adapted to solve the problem of efficient PCFG marginalization, but replacing entries of the table β with numbers between 0 and 1. These numbers are called inside probabilities, and they represent the following probabilities:

$$\beta[i,j,k] = \mathrm{P}(w_i \dots w_{i+j-1} | N^k)$$

So, $\beta[i,j,k]$ is the probability that the string $w_i \dots w_{i+j-1}$ is generated in a derivation where the starting non-terminal is N^k. Algorithm 2 is the probabilistic CYK algorithm for calculating $\mathrm{P}(\text{sentence})$.

Algorithm 2 Probabilistic CYK for $\mathrm{P}(\text{sentence})$

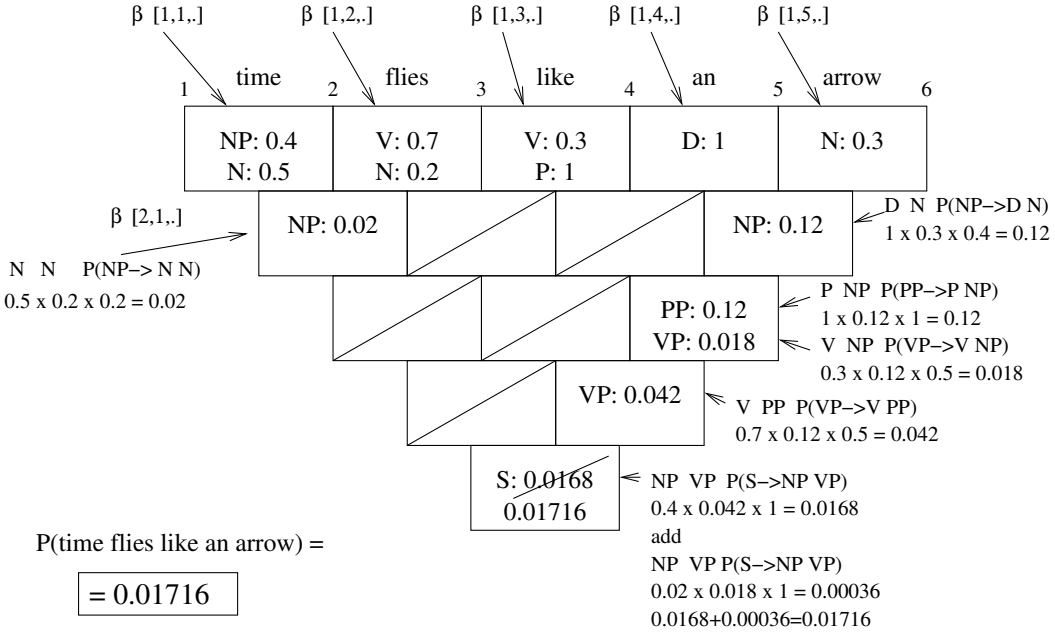
Require: sentence $= w_1 \dots w_n$, and a PCFG in CNF with nonterminals $N^1 \dots N^m$, N^1 is the start symbol
Ensure: P(sentence) is returned

allocate $\beta \in \mathbb{R}^{n \times n \times m}$ 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**
        $\beta[i, j, k] \leftarrow \beta[i, j, k] + \mathrm{P}(N^k \rightarrow N^{k_1} N^{k_2}) \cdot \beta[i, l, k_1] \cdot \beta[i + l, j - l, k_2]$
return $\beta[1, n, 1]$

PCFG Marginalization Example (grammar)

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

PCFG Marginalization Example (chart)



Conditioning

The conditioning computational problem in the PCFG model becomes the task of finding the conditional probability $\mathrm{P}(\text{tree}|\text{sentence})$, for a particular sentence and a particular parse three of the given sentence. Using the definition of the conditional probability, we have:

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

and since the sentence is a part of the parse tree, we can further write:

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

Slide notes:

Conditioning

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

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

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

$\mathrm{P}(\text{tree})$ is calculated by multiplying probabilities of all rules in the tree, and $\mathrm{P}(\text{sentence})$ is calculated by the Algorithm 2 used for marginalization.

Completion

The completion task becomes the parsing problem; i.e., the problem of finding the most probably parse tree give the sentence, which can be expressed as:

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

Slide notes:

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

The most probable completion is computed in a similar way to the marginalization algorithm (Algorithm 2). The difference is that the line 9 is replaced by the line

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

Additionally in step 10, we are not just interested in $\beta[1,n,1]$, which is the probability of the most probable tree, but we also want to obtain the actual tree.

The tree can be reconstructed from the table using algorithm 4:

10: **return** Reconstruct$(1, n, 1, \beta)$

Algorithm 3 CYK-based Completion Algorithm for arg $\max_t \mathrm{P}(t|\text{sentence})$

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: 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 4 Reconstruct(i, j, k, β)

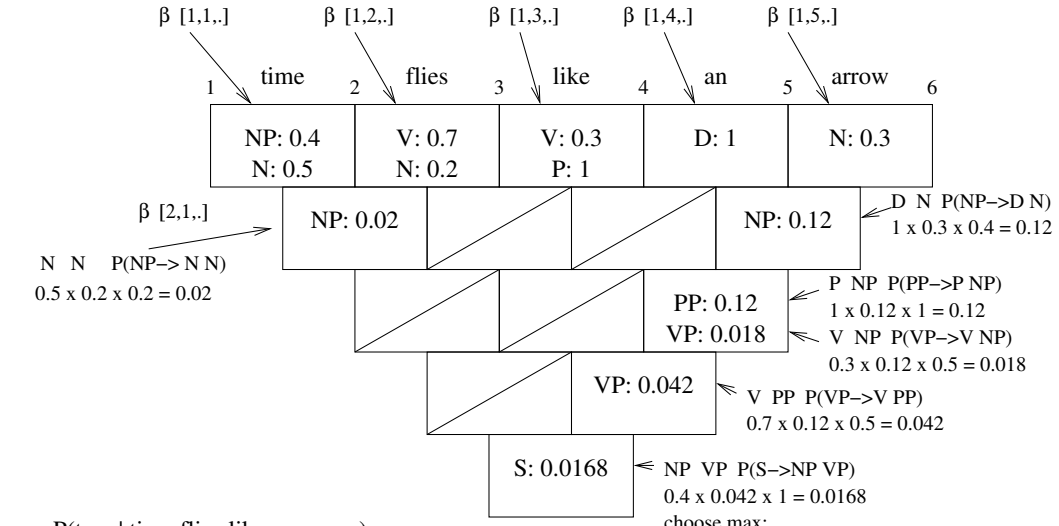
Require: β — table from CYK, i — index of the first word, j — length of sub-string sentence, k — index of non-terminal
Ensure: a most probable tree with root N^k and leaves $w_i \dots w_{i+j-1}$ is returned

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
      $t.\textit{left\_child} \leftarrow$ Reconstruct(i, l, k_1, β)
      $t.\textit{right\_child} \leftarrow$ Reconstruct$(i + l, j - l, k_2, \beta)$
      return t

PCFG Completion Example (grammar)

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

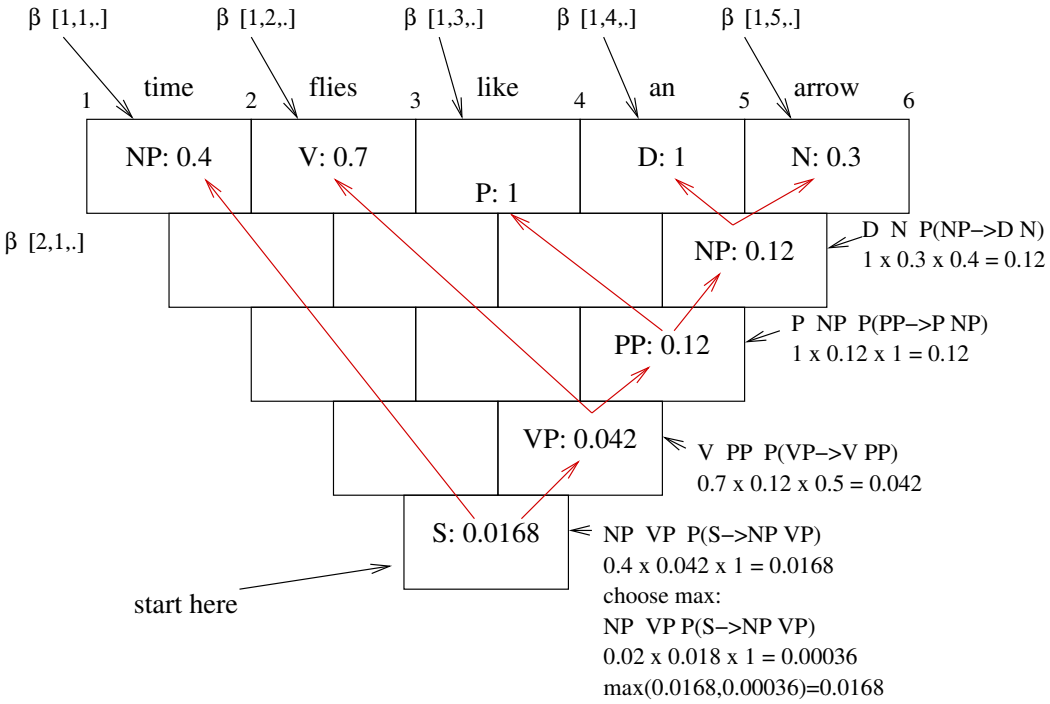
PCFG Completion Example (chart)



max P(tree | time flies like an arrow) =

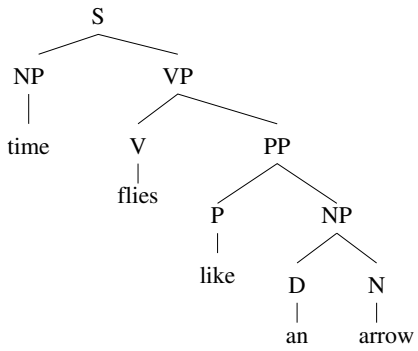
= 0.0168

PCFG Completion Example (tree reconstruction)



PCFG Completion Example (final tree)

The most probable three:



Topics related to PCFGs

– An interesting open problem is whether the inference in PCFGs can be reduced to a message-passing-style algorithm as used in Bayesian Networks?

Issues with PCFGs

The Probabilistic Context-Free Grammars were shown to perform quite well in parsing English, but usually with some additional mechanisms to address certain issues. Two most prominent issues in using PCFGs to parse nature languages are the inability of PCFGs to capture structural and lexical dependencies.

Structural dependencies are rule dependencies on the position in a parse tree. For example, pronouns occur more frequently as subjects than objects in sentences, so the rule choice between NP $\rightarrow$ PRP and NP $\rightarrow$ DT NN should depend on the position of a noun phrase in a tree. Generally, NL parse trees are usually deeper at their right side than the left side, and this propertly is typically not modeled well with PCFGs.

Lexical dependencies are rule dependencies on the words that are eventually derived from those rules, particularly phrase head words. As an example, the PP-attachment problem is resolved based on the rule probabilities of the rules applied higher in the parse tree, such as NP $\rightarrow$ NP PP and VP $\rightarrow$ VB NP PP, while they truly frequently depend on the verb being used and other word, particularly head words.

Slide notes:

PP-Attachment Example

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

As an example, let us consider simple sentences:

- "Workers dumped sacks into a bin." (from [JM]), and
- "Workers dumped sacks of fish."

At some level of parsing, we can see both of these sentences as:

- NP VBD NP PP

and now the question is whether the "NP PP" should be combined to make an NP, or the sequence "VBD NP PP" should be combined to make a VP. In a PCFG this will depend only on the probability of the rules: NP $\rightarrow$ NP PP, VP $\rightarrow$ VBD NP, and VP $\rightarrow$ VBD NP PP. However, we can see that the probailities should actually depend on affinity of the verb 'dump' and preposition 'into' on one side, and the noun 'sacks' and the preposition 'of' on other side.

A Solution: Probabilistic Lexicalized CFGs

- use heads of phrases
- expanded set of rules, e.g.:

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

- large number of new rules
- sparse data problem
- solution: new independence assumptions
- proposed solutions by Charniak, Collins, etc. around 1999

27 Non-context-free Natural Language Phenomena

27.1 Are Natural Languages Context-Free?

Are Natural Languages Context-Free?

- Can we use CFG directly to model the syntax?

- Surprisingly effective in many cases
- However, not considered sufficient
- Some NL are provably not context-free due to $ww = w^2$ forms
- Additionally, NL Phenomena

CFGs are usually not sufficient for NL parsing because of additional syntactic restrictions, referred to as Natural Language Phenomena.

27.2 Natural Language Phenomena

Natural Language Phenomena

Three well-known phenomena:

- Agreement
- Movement
- Subcategorization

27.2.1 Phenomenon 1: Agreement

Agreement

- Phenomenon which requires that constituents must agree on some features before being combined to larger constituents
- Example: "This book" vs. "These book"*, or "He works" vs. "He work"*
- The relevant features are propagated from child nodes to parent nodes; e.g., consider examples:
  These problems usually persist.
  This problem usually persists.

Agreement is a natural language phenomenon, which is a grammatical constraint requiring that constituents combined in large constituents must agree on some features. These features are associated with words at the bottom layer, and they are propagated in a bottom-up fashion to higher levels of a parse tree to larger constituents.

Agreement Examples

- subject-verb agreement
  For example, "I work." and "He works." vs. *"I works." and *"He work."
- specifier-head agreement
  For example, "This book." and "These books." vs. *"This books." and "These book."

Agreement can be a non-local dependency, e.g:

The **women** who found the wallet **were** given a reward.

27.2.2 Phenomenon 2: Movement

Movement

- movement: an natural language phenomenon, in which a constituent in a grammatically valid sentence, can sometimes be moved to another position and the new sentence remains grammatically valid
- example: "Are you well?" from "You are well."

Movement Examples

E.g, wh-movement

Which book should Peter buy ?
filler gap

Another example:

```
(S (NP (NP Air Canada) ,
       (NP (NP-*filler* one of many airline companies)
           (SBAR that (S (NP-*gap*)
                         (VP flies from Halifax
                                  to Toronto))
           )) ,
   (VP canceled the flights yesterday) ) . )
```

27.2.3 Phenomenon 3: Subcategorization

Subcategorization

- **Subcategorization** phenomenon: tendency of verbs to prefer or require certain types of arguments
- Example, correct sentences:
  The defendant **disappeared**.
  The defendant **denied** the accusation.
- but the following sentences are not correct:
  The defendant **denied**.
  The defendant **disappeared** the accusation.
- The verbs 'deny' and 'disappear' belong to different subcategories.
- For example, some verbs do not take a noun-phrase object, and some do (direct and indirect objects)

Subcategorization is a natural language phenomenon that denotes tendency of verbs to have certain preferences regarding the arguments that they can take. Some verbs may allow or disallow certain arguments to be combined with them in a verb phrase or a sentence. In this way, the verbs can be divided into syntactic subcategories of the "verb" category.

Subcategorization Example

Let us consider an subcategorization example, based on Sag and Wasow, 1999.

The following are two grammatical sentences:

> The problem disappeared. and
> The defendant denied the accusation.

However, the following two are grammatically incorrect:

> *The problem disappeared the accusation. and
> *The defendant denied.

Explanation:

– "disappear" does not take an object (verb valence)
– "deny" requires an object

If we wanted to capture the syntactic structures of the above sentences in a CFG, using typical linguistic nonterminal names, we could define a CFG with the following rules:

S $\rightarrow$ NP VP	NP $\rightarrow$ D N	D $\rightarrow$ the
N $\rightarrow$ problem	VP $\rightarrow$ V	V $\rightarrow$ disappeared

for the first sentence. In order to handle the second sentence as well, we define some additional rules:

N $\rightarrow$ defendant	VP $\rightarrow$ V NP	V $\rightarrow$ denied
N $\rightarrow$ accusation		

The resulting grammar accepts both sentences; however, some additional, grammatically incorrect, sentences would be accepted as well, including:

**The problem disappeared the accusation.*

and

**The defendant denied.*

The problem is that the two verbs 'disappear' and 'deny' behave differently: 'disappear' does not allow an object, while 'deny' requires an object. This phenomenon is called *subcategorization.* We could solve this problem by splitting the category V, of all verbs, into two categories, IV and TV—intransitive verbs and transitive verbs—and still remain within the CFG formalism. However, there are two reasons why this solution may not be satisfactory:

First, this method does not scale well with the number of NL phenomena: We have just suggested replacing the category V with two categories IV and TV in order to handle subcategorization. In order to handle number agreement (e.g., '*He works.*' vs. '**He work.*'), the two categories would be replaced by IV-sg (for singular) and IV-pl (for plural), and TV-sg and TV-pl. Furthermore, in order to handle person agreement (e.g., '*I am.*' vs. '**I is.*') these 4 categories would be replaced by $4 \cdot 3 = 12$ categories, and so on.

Not only the number of nonterminals would exponentially grow in this way, but the number of rules would also exponentially grow. For example, the method for handling number agreement directly leads to replacing three rules

S $\rightarrow$ NP VP	NP $\rightarrow$ D N	VP $\rightarrow$ V

with the following 6 rules

S $\rightarrow$ NP-sg VP-sg	NP-sg $\rightarrow$ D N-sg	VP-sg $\rightarrow$ V-sg
S $\rightarrow$ NP-pl VP-pl	NP-pl $\rightarrow$ D N-pl	VP-pl $\rightarrow$ V-pl

This approach would lead to a CFG which is too large to be constructed and maintained in practice. The problem could be solved by designing additional tools for maintaining such grammars, but it implies that the model is not good enough, and those tools should be a part of it. A large CFG raises also efficiency issues regarding the parsing algorithm.

A similar approach was taken in the Generalized Phrase Structure Grammar (GPSG) model (Gazdar et al. 1985), and it did lead to a large number of rules.

Second, this approach fails at a task called capturing linguistically significant generalizations. In the above example, some structure is obviously hidden behind the new nonterminal names—e.g, NP-sg, NP-pl, and VP-sg—and it is not a coincidence that the suffixes '-sg' and '-pl' match in all nonterminals of a rule. This regularity is not captured by the suggested approach because a CFG is not concerned with its names for nonterminals.

28 Parser Evaluation

Parser evaluation is still a matter of active discussion and research. **PARSEVAL measures** are commonly used for evaluating context-free parsing. You can read more about these measure in the [JM] textbook (Section 14.7, page 479).

To compute the PARSEVAL measures, the parse trees are first decomposed into **labeled constituents (LC),** which are triples consisting of the starting and ending point of a constituent's span in a sentence, and the constituent's label. For each sentence, the sets of constituents obtained from the parse tree obtained from a parser (PT), and from the given, "gold standard" parse tree (GT) are compared. The following measures are usually calculated:

$$\begin{aligned} \text{labelled recall} &= \frac{\text{number of correct LC in PT}}{\text{number of LC in GT}} \\ \text{labelled precision} &= \frac{\text{number of correct LC in PT}}{\text{number of LC in PT}} \\ \text{F-measure} &= \frac{2 \cdot (\text{labelled precision}) \cdot (\text{labelled recall})}{(\text{labelled precision}) + (\text{labelled recall})} \end{aligned}$$

The fourth measure, number of cross-brackets, is also used.

The state-of-the-art parsers obtain up to 90% precision and recall on the Penn Treebank data.

Example

Let us consider the following two sentences:

Time flies like an arrow.
and
He ate the cake with a spoon.

These could easily be ambiguous sentences for a parser, and let us assume that our "gold standard" parse trees; i.e., preferred or correct parse trees are as follows:

Gold standard

```
(S (NP (NN time) (NN flies))
   (VP (VB like)
       (NP (DT an) (NN arrow))))

(S (NP (PRP he))
   (VP (VBD ate) (NP (DT the)
                     (NN cake))
       (PP (IN with)
           (NP (DT a) (NN spoon)))))
```

while the parse trees returned by a parser are:

Parser result

```
(S (NP (NN time))
   (VP (VB flies)
       (PP (IN like)
           (NP (DT an) (NN arrow)))))

(S (NP (PRP he))
   (VP (VBD ate)
       (NP (DT the) (NN cake)
```

```
            (PP (IN with)
                (NP (DT a) (NN spoon))))))
```

First, we list the labeled edges of the parse trees:

```
  time flies like an arrow
0     1     2    3  4     5
```

```
Gold standard:
S  0,5 time flies like an arrow
NP 0,2 time flies
NN 0,1 time
NN 1,2 flies
VP 2,5 like an arrow
VB 2,3 like
NP 3,5 an arrow
DT 3,4 an
NN 4,5 arrow
```

```
Parser result:
S  0,5 time flies like an arrow
NP 0,1 time
NN 0,1 time
VP 1,5 flies like an arrow
VB 1,2 flies
PP 2,5 like an arrow
IN 2,3 like
NP 3,5 an arrow
DT 3,4 an
NN 4,5 arrow
```

```
 he ate the cake with a spoon
0  1   2   3    4    5 6     7
```

```
Gold standard:
S   0,7 he ate ...ke with a spoon
NP  0,1 he
PRP 0,1 he
VP  1,7 ate the cake with a spoon
VBD 1,2 ate
NP  2,4 the cake
DT  2,3 the
NN  3,4 cake
PP  4,7 with a spoon
IN  4,5 with
NP  5,7 a spoon
DT  5,6 a
NN  6,7 spoon
```

```
Parser result:
S   0,7 he ate the cake with a spoon
NP  0,1 he
PRP 0,1 he
VP  1,7 ate the cake with a spoon
VBD 1,2 ate
NP  2,7 the cake with a spoon
DT  2,3 the
NN  3,4 cake
PP  4,7 with a spoon
IN  4,5 with
NP  5,7 a spoon
DT  5,6 a
NN  6,7 spoon
```

After counting the number of correctly identified edges (true positives), non-identified edges (false negatives), incorrectly identified edges (false positives), and the total number of edges, we can calculate precision and recall:

Precision$= \frac{17}{23} \approx 0.739130434782609$ and Recall $= \frac{17}{22} \approx 0.772727272727273$.