Faculty of Computer Science, Dalhousie University *26-Nov-2026*

CSCI 4152/6509 — Natural Language Processing

Lecture 22: CYK Algorithm

Location: Goldberg Computer Science Building rm143 Instructor: Vlado Keselj
Time: 13:05 – 14:25

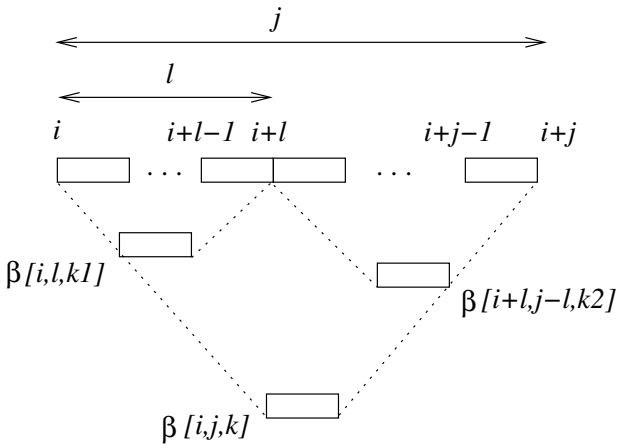
Previous Lecture

- Arguments and Adjuncts
- Some Parse Tree Examples
- Probabilistic Context-Free Grammar
  - PCFG as a probabilistic model
- **CYK Chart Parsing Algorithm**
- Chomsky Normal Form (CNF)
- CYK algorithm example

CYK Algorithm (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., $\beta_{ij} = \{N | N \Rightarrow^* w_i \dots w_{i+j-1}\}$.

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 $w_i \dots w_{i+j-1}$, and $\beta_{ijk} = 0$ that it cannot.

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 $\beta[1, n, 1]$

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

27 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 nonterminal is N^k. Algorithm 2 is the probabilistic CYK algorithm for calculating $\mathrm{P}(\text{sentence})$.

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

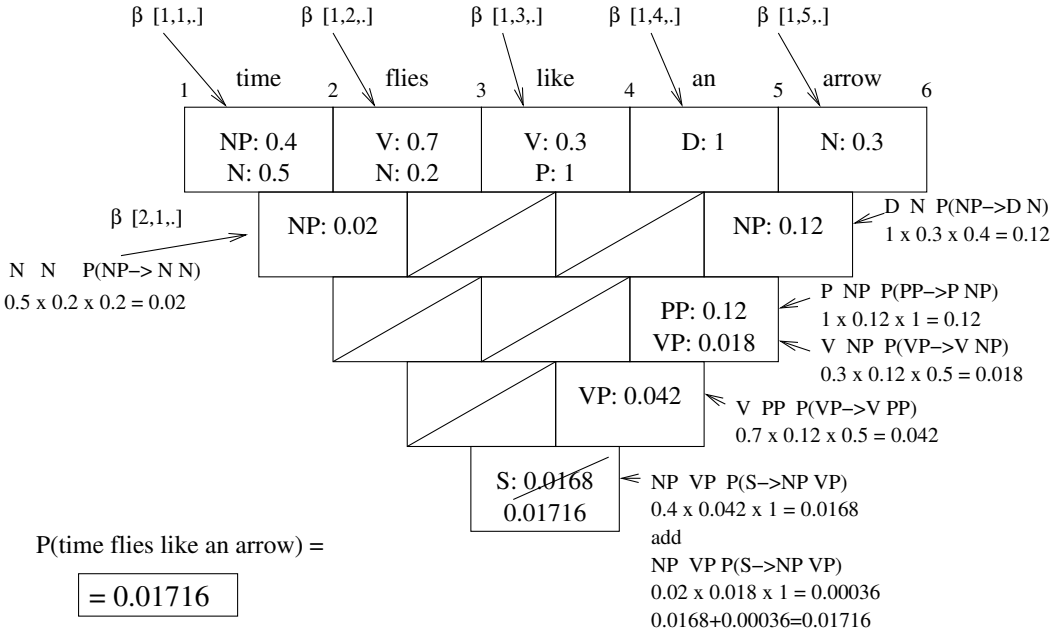
Require: sentence $= w_1 \dots w_n$, and a PCFG in CNF with nonterminals $N^1 \dots N^m$, N^1 is the start symbol
Ensure: P(sentence) is returned

```
allocate 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)

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.

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

1: allocate $\beta \in \mathbb{R}^{n \times n \times m}$ and initialize all entries to 0
2: **for** $i \leftarrow 1$ to n **do**
3: **for all** rules $N^k \rightarrow w_i$ **do**
4: $\beta[i,1,k] \leftarrow \mathrm{P}(N^k \rightarrow w_i)$
5: **for** $j \leftarrow 2$ to n **do**
6: **for** $i \leftarrow 1$ to $n-j+1$ **do**
7: **for** $l \leftarrow 1$ to $j-1$ **do**
8: **for all** rules $N^k \rightarrow N^{k_1}N^{k_2}$ **do**
9: $\beta[i,j,k] \leftarrow \max(\beta[i,j,k], \mathrm{P}(N^k \rightarrow N^{k_1}N^{k_2}) \cdot \beta[i,l,k_1] \cdot \beta[i+l, j-l, k_2])$
10: **return** Reconstruct$(1, n, 1, \beta)$

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

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

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								

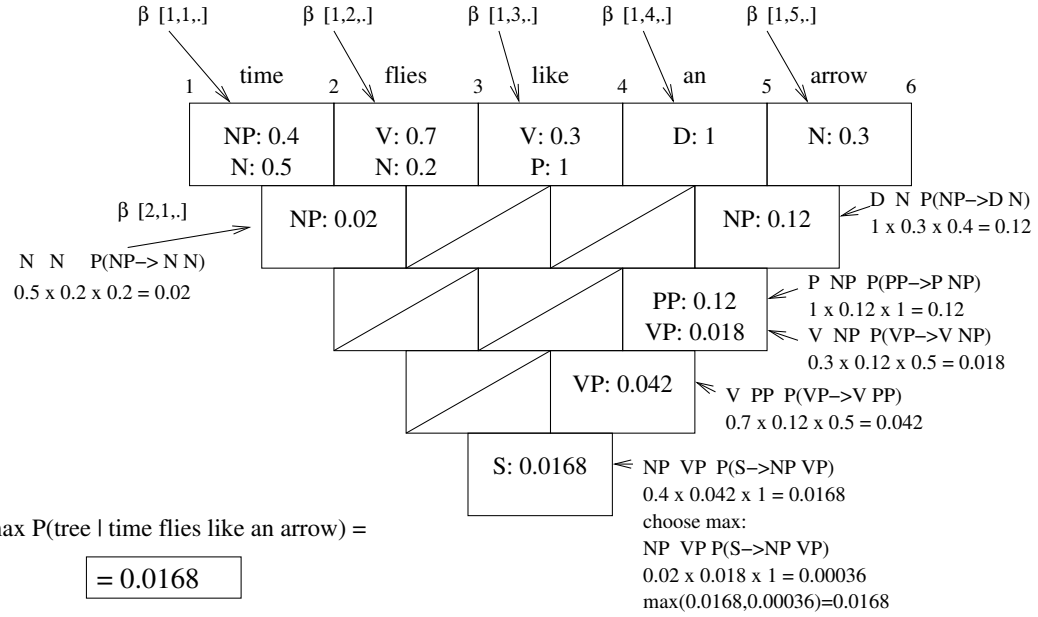
Algorithm 4 Reconstruct(i, j, k, β)

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

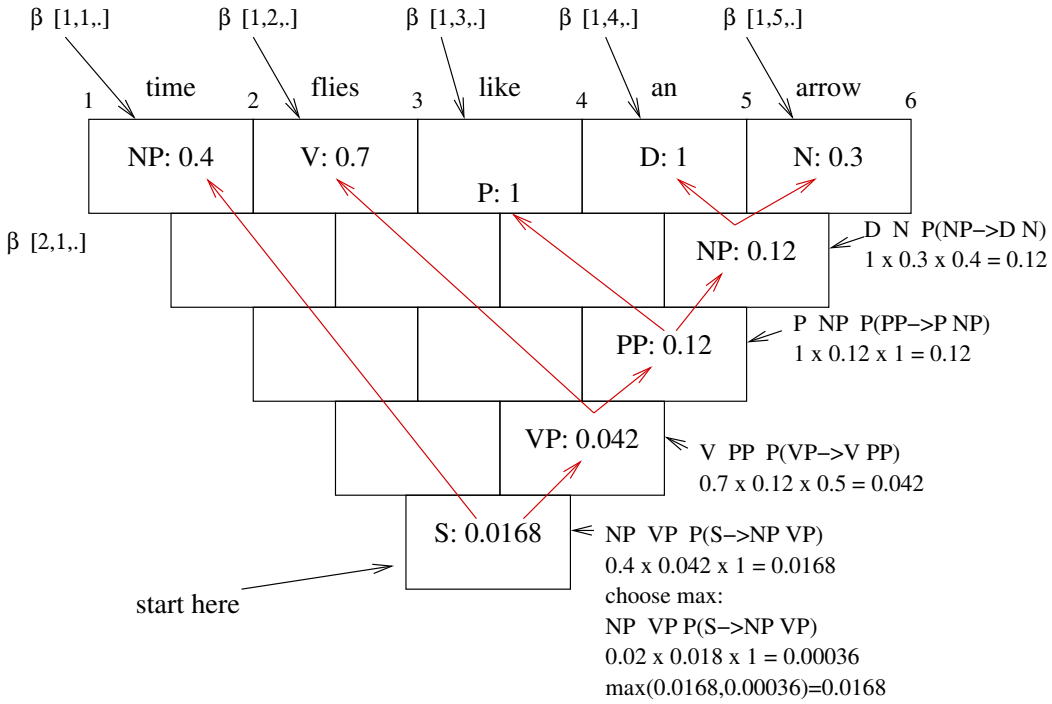
Ensure: a most probable tree with root N^k and leaves $w_i \dots w_{i+j-1}$ is returned

```
if j = 1 then
    return tree with root N^k and child w_i
for l ← 1 to j − 1 do
    for all rules N^k → N^{k1} N^{k2} do
        if β[i, j, k] = P(N^k → N^{k1} N^{k2}) · β[i, l, k1] · β[i + l, j − l, k2] then
            create a tree t with root N^k
            t.left_child ← Reconstruct(i, l, k1, β)
            t.right_child ← Reconstruct(i + l, j − l, k2, β)
            return t
```

PCFG Completion Example (chart)

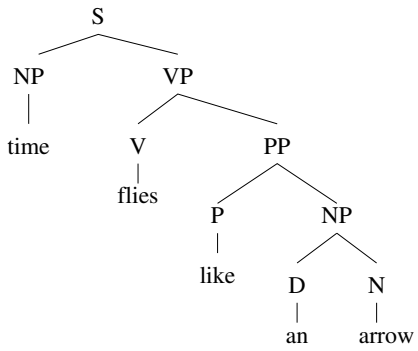


PCFG Completion Example (tree reconstruction)



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

28 Non-context-free Natural Language Phenomena

28.1 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.

28.2 Natural Language Phenomena

Natural Language Phenomena

Three well-known phenomena:

- Agreement
- Movement
- Subcategorization

28.2.1 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.

28.2.2 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) ) . )
```

28.2.3 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.

29 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 $= \frac{17}{22} \approx 0.772727272727273$.