Chapter NLP:V

V. Syntax

- Introduction
- Phrase Structure Grammars
- Dependency Grammars
- Features and Unification

Phrase Structure Grammars

Formal Grammars

A formal grammar is defined by a set of rules with terminal and non-terminal symbols.

- Rules transform non-terminal symbols into other terminal or non-terminal symbols.
- Terminal symbols ($\approx$ words) cannot be transformed any further.
- Non-terminals express clusters or generalizations of terminals.

Grammar (Σ, N, S, R)

- Σ An alphabet (i.e., a finite set of terminal symbols).
- N A finite set of non-terminal symbols.
- S A start non-terminal symbol, $S \in N$.
- R A finite set of production rules, $R \subseteq (\Sigma \cup N)^+ \setminus \Sigma^* \times (\Sigma \cup N)^*$.

Phrase Structure Grammars

Chomsky Hierarchy

Formal grammars can be ordered in four types:

- Chomsky-0 (recursively enumerable). Any (Σ, N, S, R) as defined.
- Chomsky-1 (context-sensitive). Only rules $U \to V$ with $|U| \leq |V|$.
- Chomsky-2 (context-free). Only rules $U \to V$ with $U \in N$.
- Chomsky-3 (regular). Only rules $U \to V$ with $U \in N$ and $V \in \{\varepsilon, v, vW\}$, $v \in \Sigma$, $W \in N$.

In NLP most commonly used are regular and context-free grammars.

Remarks:

- Context-sensitive grammars allow multiple symbols on the left side (but at least one non-terminal) and multiple symbold on the right side without contraints.
  $S \rightarrow abc/aAbc$, $Ab \rightarrow bA$, $Ac \rightarrow Bbcc$, $bB \rightarrow Bb$, $aB \rightarrow aa/aaA$
- Context-free grammars require a single non-teminal symbol on the left side. For example:
  $N = \{S, X\}, \Sigma = \{a, b\}$, $S \rightarrow ab$, $S \rightarrow aXb$, $X \rightarrow ab$, $X \rightarrow aXb$
- Regular grammars are particularly useful in inferring information when language follows clear sequential patterns (i.e. pattern parsing). Consider our lecture on regular expressions for details.

Phrase Structure Grammars

Context-free grammars (CFG)

A phrase structure grammar is a syntactic structure based on the constituency relation between words.

Phrase structure grammars can be modeled as context-free grammars:

$$(\Sigma,\ S,\ N_{phr} \cup N_{pos},\ R_{phr} \cup R_{pos})$$

- Σ The alphabet.
- S The start symbol.
- N_{phr} A finite set of structural non-terminal symbols. NP, VP, . . .
- N_{pos} A finite set of lexical pre-terminal symbols. NN, VB, PRP, . . .

  $N_{phr} \cap N_{pos} = \emptyset$
- R_{phr} A finite set of structure production rules. S $\rightarrow$ NP VP, . . .

  $U \rightarrow V, U \in N_{phr} \quad V \in (N_{phr} \cup N_{pos})^*$
- R_{pos} A finite set of lexicon production rules. NP $\rightarrow$ DET NN NN, . . .

  $U \rightarrow v, U \in N_{pos} \quad v \in \Sigma$

Phrase Structure Grammars

Context-free grammars (CFG)

Some typical (English) phrase structures:

	Structural rule	Example
Clause Structures		
Declarative Clause	S $\rightarrow$ NP VP	I take the flight tomorrow
Imperative Clause	S $\rightarrow$ VP	Show me the next train
Yes-no Question	S $\rightarrow$ *Aux* NP VP	*Do* you get off there?
Noun Phrase Structures		
Determiners	NP $\rightarrow$ DET NP	the flight
Adjective Phrases	NP $\rightarrow$ JJ NP	the earliest flight
Gerundive	NP $\rightarrow$ NP VP	Show me the flights leaving today
Verb Phrase Structures		
Verb Phrase	VP $\rightarrow$ Verb NP	take the train
Sentential Complement	VP $\rightarrow$ Verb S	I think I want to take the train
Two Verb Phrases	VP $\rightarrow$ Verb VP	I want to arrange three flights
Coordinations		
Coordination	NP $\rightarrow$ NP and NP	the flights and the cost

Phrase Structure Grammars

Context-free grammars (CFG)

Some typical (English) phrase structures:

	Structural rule	Example
Clause Structures		
Declarative Clause	S $\rightarrow$ NP VP	I take the flight tomorrow
Imperative Clause	S $\rightarrow$ VP	Show me the next train
Yes-no Question	S $\rightarrow$ Aux NP VP	Do you get off there?
Noun Phrase Structures		
Determiners	NP $\rightarrow$DET NP	the flight
Adjective Phrases	NP $\rightarrow$ JJ NP	the earliest flight
Gerundive	NP $\rightarrow$ NP VP	Show me the flights leaving today
Verb Phrase Structures		
Verb Phrase	VP $\rightarrow$ Verb NP	take the train
Sentential Complement	VP $\rightarrow$ Verb S	I think I want to take the train
Two Verb Phrases	VP $\rightarrow$ Verb VP	I want to arrange three flights
Coordinations		
Coordination	NP $\rightarrow$ NP and NP	the flights and the cost

Phrase Structure Grammars

Context-free grammars (CFG)

Some typical (English) phrase structures:

	Structural rule	Example
Clause Structures		
Declarative Clause	S $\rightarrow$ NP VP	I take the flight tomorrow
Imperative Clause	S $\rightarrow$ VP	Show me the next train
Yes-no Question	S $\rightarrow$ Aux NP VP	Do you get off there?
Noun Phrase Structures		
Determiners	NP $\rightarrow$ DET NP	the flight
Adjective Phrases	NP $\rightarrow$ JJ NP	the earliest flight
Gerundive	NP $\rightarrow$ NP VP	Show me the flights leaving today
Verb Phrase Structures		
Verb Phrase	VP $\rightarrow$ *Verb* NP	*take* the train
Sentential Complement	VP $\rightarrow$ *Verb* S	I *think* I want to take the train
Two Verb Phrases	VP $\rightarrow$ *Verb* VP	I *want* to arrange three flights
Coordinations		
Coordination	NP $\rightarrow$ NP and NP	the flights and the cost

Phrase Structure Grammars

Context-free grammars (CFG)

Some typical (English) phrase structures:

	Structural rule	Example
Clause Structures		
Declarative Clause	S $\rightarrow$ NP VP	I take the flight tomorrow
Imperative Clause	S $\rightarrow$ VP	Show me the next train
Yes-no Question	S $\rightarrow$ Aux NP VP	Do you get off there?
Noun Phrase Structures		
Determiners	NP $\rightarrow$ DET NP	the flight
Adjective Phrases	NP $\rightarrow$ JJ NP	the earliest flight
Gerundive	NP $\rightarrow$ NP VP	Show me the flights leaving today
Verb Phrase Structures		
Verb Phrase	VP $\rightarrow$ Verb NP	take the train
Sentential Complement	VP $\rightarrow$ Verb S	I think I want to take the train
Two Verb Phrases	VP $\rightarrow$ Verb VP	I want to arrange three flights
Coordinations		
Coordination	NP $\rightarrow$ NP *and* NP	the flights *and* the cost

Phrase Structure Grammars

CFG: Example Grammar

	Structural rules		**Lexical rules**
s1	S $\rightarrow$ NP VP	l1	N $\rightarrow$ people
s2	VP $\rightarrow$ V NP	l2	| $\rightarrow$ fish
s3	| $\rightarrow$ V NP PP	l3	| $\rightarrow$ tanks
s4	NP $\rightarrow$ NP NP	l4	| $\rightarrow$ rods
s5	| $\rightarrow$ NP PP // binary	l5	V $\rightarrow$ people
s6	| $\rightarrow$ N // unary	l6	| $\rightarrow$ fish
s7	| $\rightarrow$ ε // empty	l7	| $\rightarrow$ tanks
s8	PP $\rightarrow$ P NP	l8	P $\rightarrow$ with

Alternative:

Structural rules	**Lexical rules**
S $\rightarrow$ NP VP	
NP $\rightarrow$ NP NP | NP PP | N | ε	N $\rightarrow$ people | fish | tanks | rods
VP $\rightarrow$ V NP | V NP PP	V $\rightarrow$ people | fish | tanks
PP $\rightarrow$ P NP	P $\rightarrow$ with

Phrase Structure Grammars

CFG Construction: Treebanks

- A phrase structure grammar consists of many (10k) rules.
- These rules are extracted from corpora with tree-structured expert annotations: treebanks. The most popular Treebanks are:
  1. The *Penn Treebank* (PTB) for constituency trees. [Marcus et al., 1993]
  2. The Universal Dependencies treebank for dependency structures.

Example from the Brown Corpus:

```
((S
   (NP-SBJ (DT That)
     (JJ cold) (, ,)
     (JJ empty) (NN sky) )
   (VP (VBD was)
     (ADJP-PRD (JJ full)
       (PP (IN of)
         (NP (NN fire)
           (CC and)
           (NN light) ))))
   (. .) ))
```

Structural rules		
S	→	NP-SBJ VP
NP-SBJ	→	DT JJ , JJ NN
VP	→	VBD ADJP-PRD
ADJP-PRD	→	JJ PP
PP	→	IN NP
NP	→	NN CC NN

Lexical rules		
NN	→	sky | fire | light
VBD	→	was
JJ	→	cold | empty | full
DT	→	That
IN	→	of
CC	→	and

Phrase Structure Grammars

Constituency Parsing

Classical parsing

- Hand-crafted grammar (CFG or more complex), along with a lexicon.
- Usage of grammar-based systems to prove parses from words.
- This scales badly and fails to give high coverage of language.

Example: “Fed raises interest rates 0.5% in effort to control inflation”

- Minimal grammar: 36 parses
- Real-size broad-coverage grammar: Millions of parses

Phrase Structure Grammars

CFG Modifications for Parsing

Parsing with a CFG from a Treebank often yields long, specific, and rare rules:

- Parsing is inefficient.
- Parsing generalizes poorly.
- Syntactic disambiguation is difficult.

Some rules from Penn:

```
NP → DT JJ NN
NP → DT JJ NN NN
NP → DT JJ JJ NN
NP → RB DT JJ NN NN
NP → RB DT JJ JJ NNS
NP → DT NNP NNP NNP NNP JJ NN
NP → DT VBG JJ NNP NNP CC NNP
NP → DT JJ NNS , NNS CC NN NNS NN
NP → NP JJ , JJ “ SBAR ” NNS
```

CFGs are often modified for parsing:

Probabilistic CFG	Extract the likelihood of each rule. Use the most likely rule when parsing.
Chomsky Normal Form	Normalize rules into (equivalent) binary ones.
Lexicalization	Add prior knowledge from a lexicon.
Linearization	Transform trees to sequences.

Phrase Structure Grammars

Probabilistic CFG

A probabilistic context-free grammar (PCFG) is a CFG where each production rule is assigned a probablility.

PCFG (Σ, N, S, R, P)

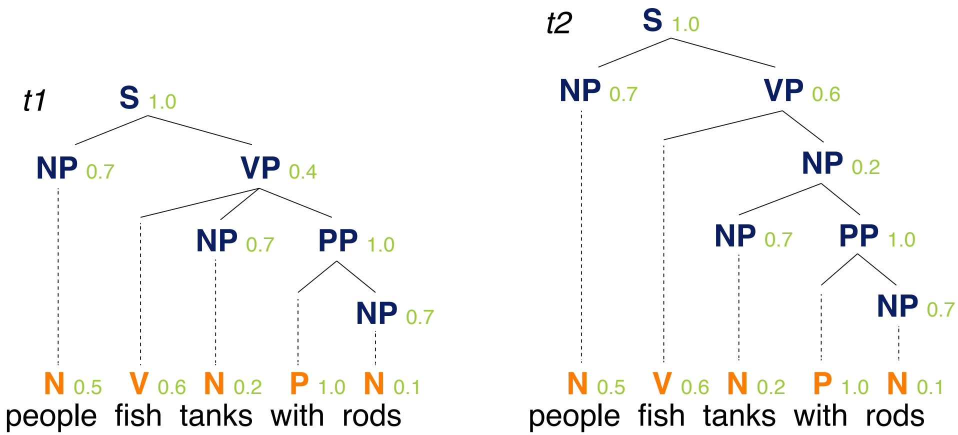
P A probability function $R \rightarrow [0, 1]$ from production rules to probabilities, such that

$$\forall U \in N : \sum_{(U \rightarrow V) \in R} P(U \rightarrow V) = 1$$

- The probability $P(t)$ of a parse tree t is the product of the probabilities of the rules used to generate it.
- The probability $P(s)$ of a clause s is the sum of the probabilities of the parses which yield s.

Phrase Structure Grammars

Probabilistic CFG



Structural rules	P
S $\rightarrow$ NP VP	1.0
VP $\rightarrow$ V NP	0.6
VP $\rightarrow$ V NP PP	0.4
NP $\rightarrow$ NP NP	0.1
NP $\rightarrow$ NP PP	0.2
NP $\rightarrow$ N	0.7
PP $\rightarrow$ P NP	1.0

Lexical rules	P
N $\rightarrow$ people	0.5
N $\rightarrow$ fish	0.2
N $\rightarrow$ tanks	0.2
N $\rightarrow$ rods	0.1
V $\rightarrow$ people	0.1
V $\rightarrow$ fish	0.6
V $\rightarrow$ tanks	0.3
P $\rightarrow$ with	1.0

$$
\begin{aligned}
P(t_1) &= 1.0 \cdot 0.7 \cdot 0.4 \cdot 0.5 \cdot 0.6 \cdot 0.7 \cdot 1.0 \cdot 0.2 \cdot 1.0 \cdot 0.7 \cdot 0.1 \\
&= 0.0008232 \\
P(t_2) &= 1.0 \cdot 0.7 \cdot 0.6 \cdot 0.5 \cdot 0.6 \cdot 0.2 \cdot 0.7 \cdot 1.0 \cdot 0.2 \cdot 1.0 \cdot 0.7 \cdot 0.1 \\
&= 0.00024696 \\
P(s) &= P(t_1) + P(t_2) = 0.0008232 + 0.00024696 \\
&= 0.00107016
\end{aligned}
$$

Phrase Structure Grammars

Chomsky Normal Form

A CFG is in Chomsky Normal Form if all rules in R are in either of the forms:

$U \rightarrow VW$	Two non-terminals on the right side.
$U \rightarrow v$	One terminal symbol on the right side.
$S \rightarrow e$	The empty clause without symbols.

Two grammars are (weakly) equivalent if they generate the same set of strings.

- Grammars can be equivalenty transformed by changing production rules.
- We can transform a CFG to parse it more efficiently.
- The Chomsky Normal Form is a binary branching normalization that is highly useful for parsing.

Phrase Structure Grammars

Chomsky Normal Form

A CFG is in Chomsky Normal Form if all rules in R are in either of the forms:

$U \to VW$	Two non-terminals on the right side.
$U \to v$	One terminal symbol on the right side.
$S \to e$	The empty clause without symbols.

Transformation Idea:

1. Split n-ary production rules with helper non-terminals.

   VP $\to$ NP VP NP $\quad\to\quad$ VP $\to$ NP α $\cup$ α $\to$ VP NP

2. Remove empty rules.

   NP $\to$ ϵ $\cup$ VP $\to$ V NP $\quad\to\quad$ VP $\to$ V NP $\cup$ VP $\to$ V)

3. Replace unary *structure* rules with direct rules.

   NP $\to$ PP $\cup$ PP $\to$ P N $\quad\to\quad$ PP $\to$ P N $\cup$ NP $\to$ P N)

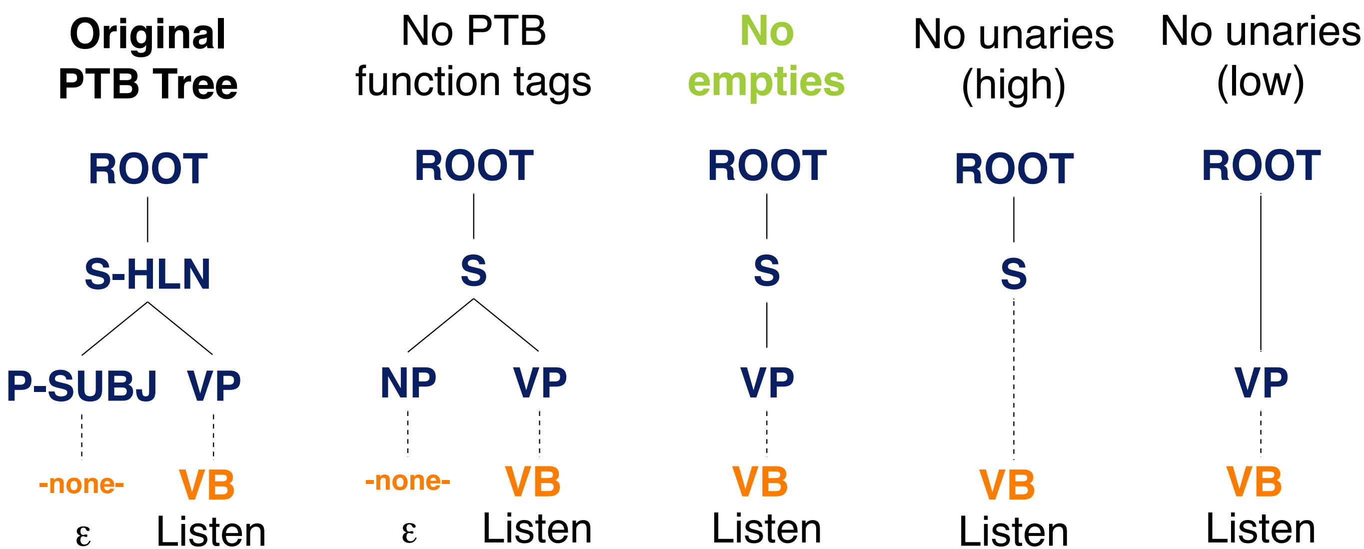
Phrase Structure Grammars

Chomsky Normal Form

A CFG is in Chomsky Normal Form if all rules in R are in either of the forms:

$U \to VW$	Two non-terminals on the right side.
$U \to v$	One terminal symbol on the right side.
$S \to e$	The empty clause without symbols.

Transformation of Penn (PTB) trees to Chomsky Normal Form (CNF):



Phrase Structure Grammars

CNF Transformation

Signature

- Input. The production rules $R = R_{phr} \cup R_{pos}$ of a CFG.
- Output. The production rules R^* of the normalized version of the CFG.

toChomskyNormalForm(Production rules R)

while an empty $(U \to \varepsilon) \in R$ **do**

    // Replace empty rules

while a unary $(U \to V) \in R$ **do**

    // Replace unary rules

while an n-ary $(U \to V_1 \ldots V_n) \in R$ **do**

    // Split n-ary rules with $n \geq 3$

return R

Phrase Structure Grammars

CNF Transformation: Replace Empty Rules

1. **while** an empty $(U \to \varepsilon) \in R$ **do**
2. $\quad R \leftarrow R \setminus \{U \to \varepsilon\}$
3. $\quad$ **for each** rule $(\mathbf{V} \to \mathbf{V_1 \ldots V_k}\ \mathbf{U}\ \mathbf{W_1 \ldots W_l}) \in R$ **do** // $k, l \geq 0$
4. $\quad\quad R \leftarrow R \cup \{V \to V_1 \ldots V_k\ W_1 \ldots W_l\}$

Structural rules R^{old}	**Lexical rules**
S → NP VP	
VP → V NP | V NP PP	V → people | fish | tanks
NP → NP NP | NP PP | N | ~~NP → ε~~	N → people | fish | tanks | rods
PP → P NP	P → with

Structural rules R$^{\text{no empties}}$	**Lexical rules**
S → **NP** **VP** | VP	
VP → **V** **NP** | V | **V** **NP** **PP** | V PP	V → people | fish | tanks
NP → **NP** **NP** | NP | **NP** **PP** | PP | N	N → people | fish | tanks | rods
PP → **P** **NP** | P	P → with

Phrase Structure Grammars

CNF Transformation: Replace Unary Rules (1)

while a unary $(U \rightarrow \mathbf{V}) \in R$ **do**
  $R \leftarrow R \setminus \{U \rightarrow \mathbf{V}\}$
  if $U \neq V$ **then**
    for each $(\mathbf{V} \rightarrow \mathbf{V_1} \ldots \mathbf{V_k}) \in R$ **do** $R \leftarrow R \cup \{U \rightarrow V_1 \ldots V_k\}$
    if not $(W \rightarrow V_1 \ldots V_k \; V \, W_1 \ldots W_l) \in R$ **then**
      for each $(V \rightarrow V_1 \ldots V_k) \in R$ **do** $R \leftarrow R \setminus \{V \rightarrow V_1 \ldots V_k\}$

Structural rules $R^{\text{no empties}}$	Lexical rules
S → NP VP | ~~VP~~	
VP → V NP | V | V NP PP | V PP	V → people | fish | tanks
NP → NP NP | NP | NP PP | PP | N	N → people | fish | tanks | rods
PP → P NP | P	P → with

Structural rules $R^{\text{no unaries 1}}$	Lexical rules
S → NP VP | V NP | V | V NP PP | V PP	
VP → V NP | V | V NP PP | V PP	V → people | fish | tanks
NP → NP NP | NP | NP PP | PP | N	N → people | fish | tanks | rods
PP → P NP | P	P → with

Phrase Structure Grammars

CNF Transformation: Replace Unary Rules (2)

while a unary $(U \rightarrow \mathbf{V}) \in R$ **do**
  $R \leftarrow R \setminus \{U \rightarrow \mathbf{V}\}$
  if $U \neq V$ **then**
    for each $(\mathbf{V} \rightarrow \mathbf{V_1 \ldots V_k}) \in R$ **do** $R \leftarrow R \cup \{U \rightarrow V_1 \ldots V_k\}$
    if not $(W \rightarrow V_1 \ldots V_k \; V \, W_1 \ldots W_l) \in R$ **then**
      for each $(V \rightarrow V_1 \ldots V_k) \in R$ **do** $R \leftarrow R \setminus \{V \rightarrow V_1 \ldots V_k\}$

Structural rules $R^{\text{no unaries 1}}$	Lexical rules
S → NP VP | V NP | V | V NP PP | V PP	
VP → V NP | ~~V~~ | V NP PP | V PP	V → people | fish | tanks
NP → NP NP | NP | NP PP | PP | N	N → people | fish | tanks | rods
PP → P NP | P	P → with

Structural rules $R^{\text{no unaries 2}}$	Lexical rules
S → NP VP | V NP | V | V NP PP | V PP	
VP → V NP | V NP PP | V PP	**V → people | fish | tanks**
	VP → people | fish | tanks
NP → NP NP | NP | NP PP | PP | N	N → people | fish | tanks | rods
PP → P NP | P	P → with

Phrase Structure Grammars

CNF Transformation: Replace Unary Rules (3-7)

while a unary $(U \rightarrow \mathbf{V}) \in R$ **do**
  $R \leftarrow R \setminus \{U \rightarrow \mathbf{V}\}$
  if $U \neq V$ **then**
    for each $(\mathbf{V} \rightarrow \mathbf{V_1} \ldots \mathbf{V_k}) \in R$ **do** $R \leftarrow R \cup \{U \rightarrow V_1 \ldots V_k\}$
    if not $(W \rightarrow V_1 \ldots V_k \; V \, W_1 \ldots W_l) \in R$ **then**
      for each $(V \rightarrow V_1 \ldots V_k) \in R$ **do** $R \leftarrow R \setminus \{V \rightarrow V_1 \ldots V_k\}$

Structural rules $R^{\text{no unaries 2}}$	Lexical rules
S → NP VP | V NP | ~~V~~ | V NP PP | V PP	
VP → V NP | V NP PP | V PP	V → people | fish | tanks
	VP → people | fish | tanks
NP → NP NP | ~~NP~~ | NP PP | ~~PP~~ | ~~N~~	N → people | fish | tanks | rods
PP → P NP | ~~P~~	P → with

Structural rules $R^{\text{no unaries}}$	Lexical rules
S → NP VP | V NP | V NP PP | V PP	S → people | fish | tanks
VP → V NP | V NP PP | V PP	V → people | fish | tanks
	VP → people | fish | tanks
NP → NP NP | NP PP | P NP	N → people | fish | tanks | rods
PP → P NP	P → with
	NP → people | fish | tanks | rods | with
	PP → with

Phrase Structure Grammars

CNF Transformation: Split n-ary rules with $n \geq 3$

11. **while** an n-ary $(U \to V_1 \dots V_n) \in R$ **do** // $n \geq 3$

12. $R \leftarrow (R \setminus \{U \to \mathbf{V_1}, V_2, \dots V_n\}) \cup \{U \to \mathbf{V_1}\,\mathbf{U\_V_1},\ \mathbf{U\_V_1} \to V_2 \dots V_n\}$

Structural rules R$^{\text{no unaries}}$	**Lexical rules**
S → NP VP | V NP | ~~V NP PP~~ | V PP	S → people | fish | tanks
VP → V NP | ~~V NP PP~~ | V PP	V → people | fish | tanks
	VP → people | fish | tanks
NP → NP NP | NP PP | P NP	N → people | fish | tanks | rods
PP → P NP	P → with
	NP → people | fish | tanks | rods | with
	PP → with

Structural rules R$^{\text{CNF}}$	**Lexical rules**
S → NP VP | V NP | **V** **S_V** | V PP	S → people | fish | tanks
S_V → NP PP	
VP → V NP | **V** **VP_V** | V PP	V → people | fish | tanks
VP_V → NP PP	VP → people | fish | tanks
NP → NP NP | NP PP | P NP	N → people | fish | tanks | rods
PP → P NP	P → with
	NP → people | fish | tanks | rods | with
	PP → with

Phrase Structure Grammars

Chomsky Normal Form Transformation: Pseudocode

Signature

- Input. The production rules $R = R_{phr} \cup R_{pos}$ of a CFG.
- Output. The production rules R^* of the normalized version of the CFG.

toChomskyNormalForm(Production rules R)

while an empty $(U \to \varepsilon) \in R$ **do**
$R \leftarrow R \setminus \{U \to \varepsilon\}$
for each rule $(V \to V_1 \ldots V_k \, U \, W_1 \ldots W_l) \in R$ **do** // $k, l \geq 0$
$R \leftarrow R \cup \{V \to V_1 \ldots V_k \;\; W_1 \ldots W_l\}$
while a unary $(U \to V) \in R$ **do**
$R \leftarrow R \setminus \{U \to V\}$
if $U \neq V$ **then**
for each $(V \to V_1 \ldots V_k) \in R$ **do** $R \leftarrow R \cup \{U \to V_1 \ldots V_k\}$
if not $(W \to V_1 \ldots V_k \;\; V \, W_1 \ldots W_l) \in R$ **then**
for each $(V \to V_1 \ldots V_k) \in R$ **do** $R \leftarrow R \setminus \{V \to V_1 \ldots V_k\}$
while an n-ary $(U \to V_1 \ldots V_n) \in R$ **do** // $n \geq 3$
$R \leftarrow (R \setminus \{U \to V_1 \ldots V_n\}) \cup \{U \to V_1 \, U\_V_1, \; U\_V_1 \to V_2 \ldots V_n\}$
return R

Remarks:

- The original algorithm presented by Chomsky has 5 steps: START, TERM, BIN, DEL, and UNIT.
- `toChomskyNormalForm` only uses DEL, UNIT, and BIN in that order. The order can be changed, but DEI must come before UNIT.
- START eliminated `S` on the RHS, which should not occur in natural language.
- TERM splits rules with mixed terminals and non-terminals. We assume that the initial Grammar was extraced without such constructs.

Phrase Structure Grammars

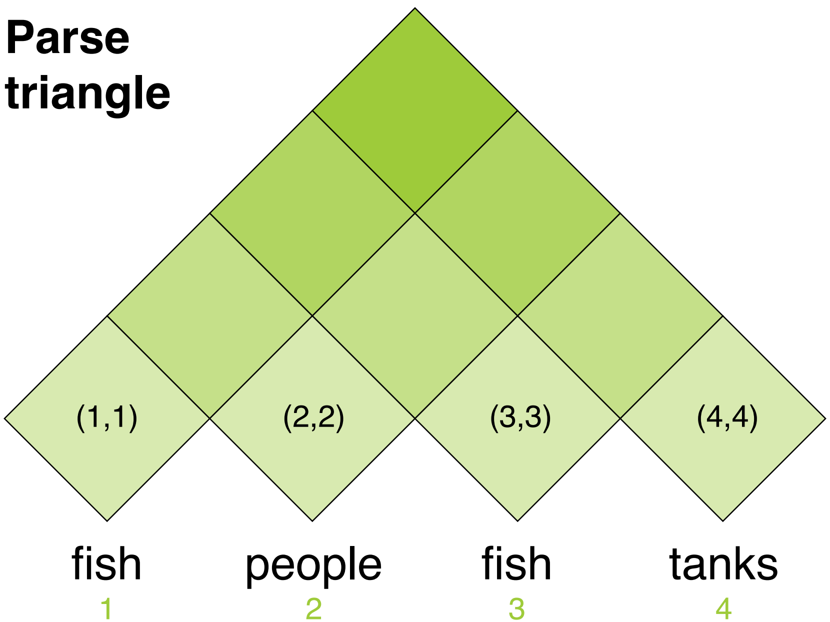
Cocke-Kasami-Younger (CKY) Parsing

PCFGs in CNF can be parsed via dynamic programming:

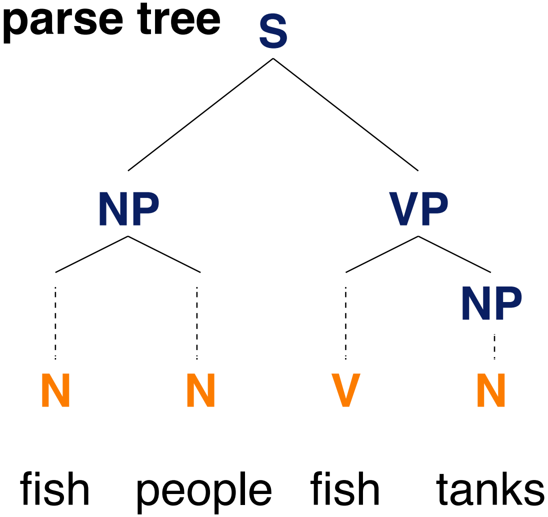
- CKY finds the most likely parse tree from the PCFGs probabilities.
- With a CFG in CNF, CKY parses in cubic time and quadratic space.

  With respect to the length of the sentence and the number of non-terminals.

Parse triangle



Most likely parse tree



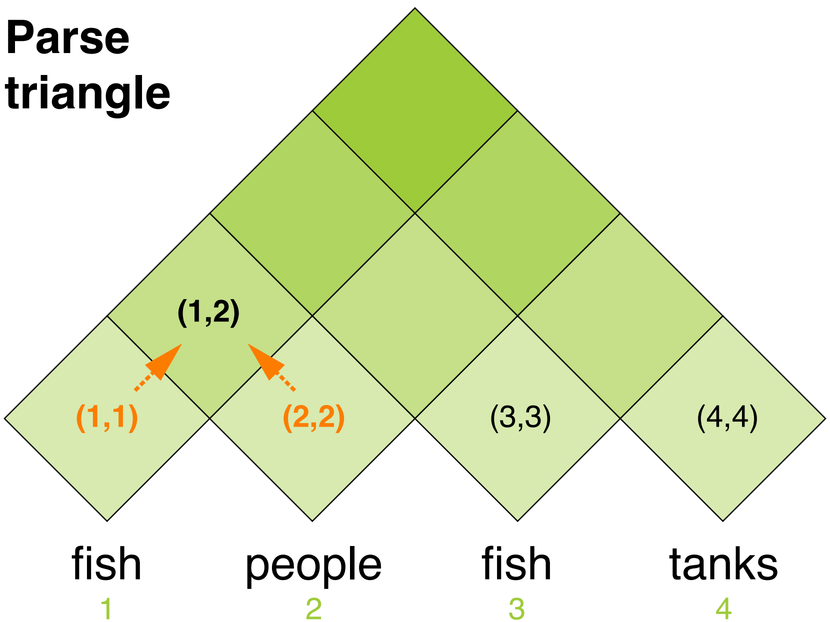
Parsing based on a PCFG

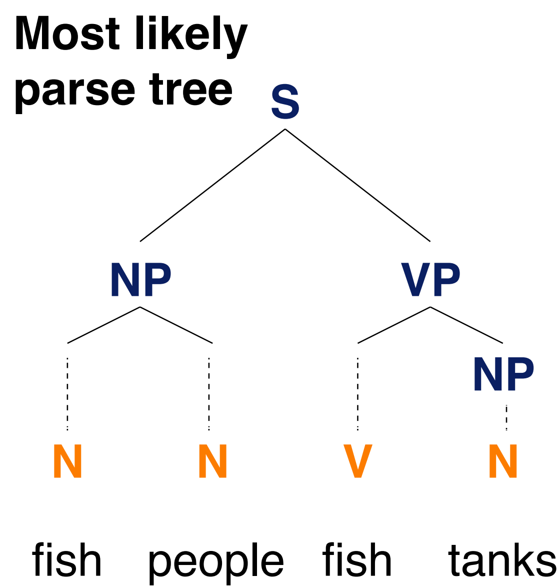
Cocke-Kasami-Younger (CKY) Parsing

PCFGs in CNF can be parsed via dynamic programming:

- CKY finds the most likely parse tree from the PCFGs probabilities.
- With a CFG in CNF, CKY parses in cubic time and quadratic space.

  With respect to the length of the sentence and the number of non-terminals.





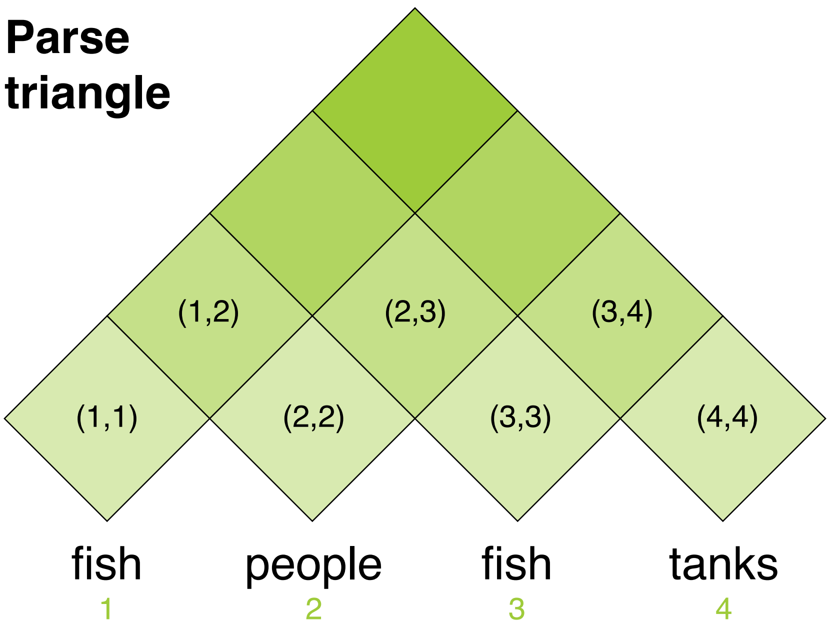
Parsing based on a PCFG

Cocke-Kasami-Younger (CKY) Parsing

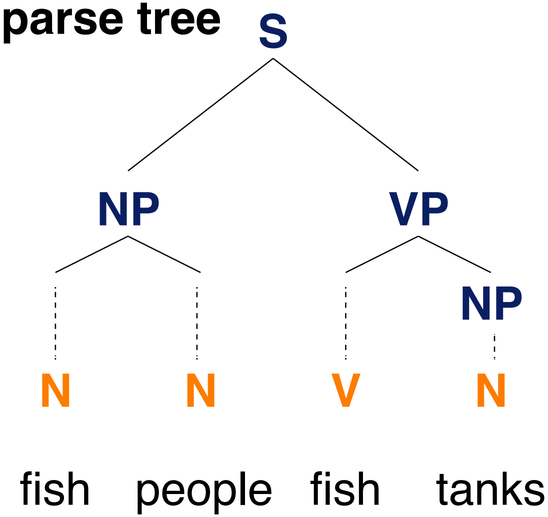
PCFGs in CNF can be parsed via dynamic programming:

- CKY finds the most likely parse tree from the PCFGs probabilities.
- With a CFG in CNF, CKY parses in cubic time and quadratic space.
  With respect to the length of the sentence and the number of non-terminals.

Parse triangle



Most likely parse tree

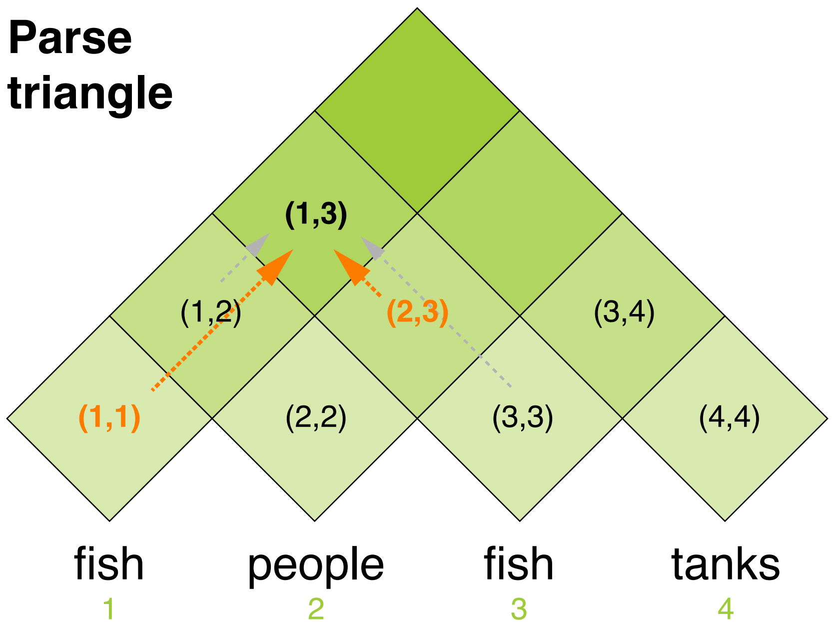


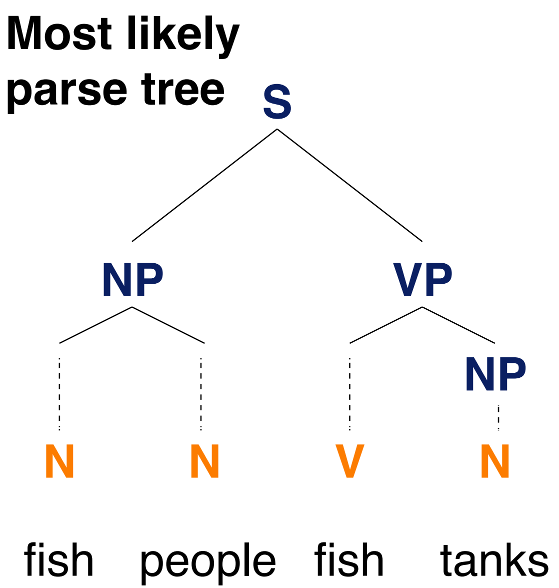
Parsing based on a PCFG

Cocke-Kasami-Younger (CKY) Parsing

PCFGs in CNF can be parsed via dynamic programming:

- CKY finds the most likely parse tree from the PCFGs probabilities.
- With a CFG in CNF, CKY parses in cubic time and quadratic space.
  With respect to the length of the sentence and the number of non-terminals.





Parsing based on a PCFG

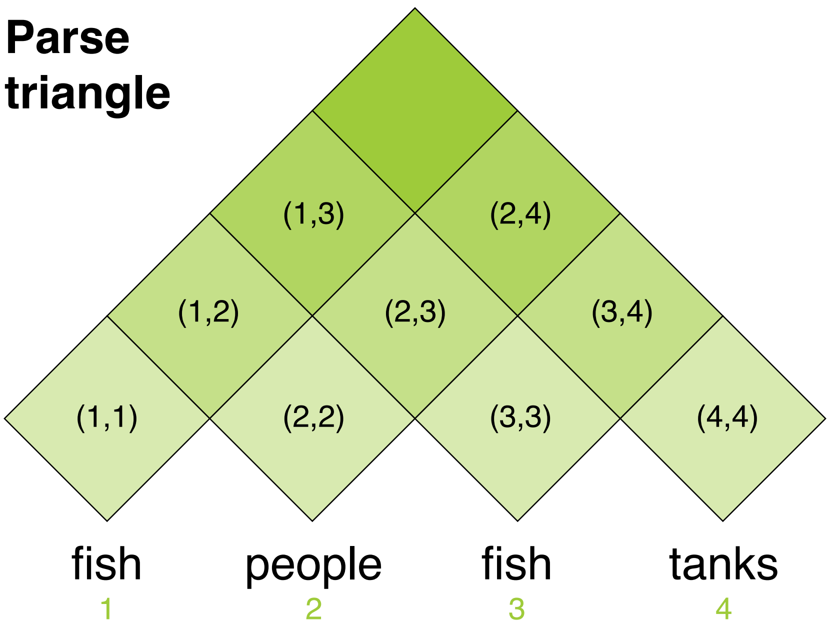
Cocke-Kasami-Younger (CKY) Parsing

PCFGs in CNF can be parsed via dynamic programming:

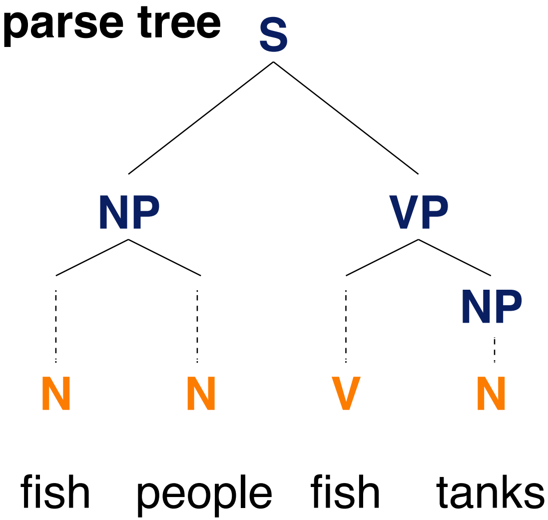
- CKY finds the most likely parse tree from the PCFGs probabilities.
- With a CFG in CNF, CKY parses in cubic time and quadratic space.

  With respect to the length of the sentence and the number of non-terminals.

Parse triangle



Most likely parse tree



Parsing based on a PCFG

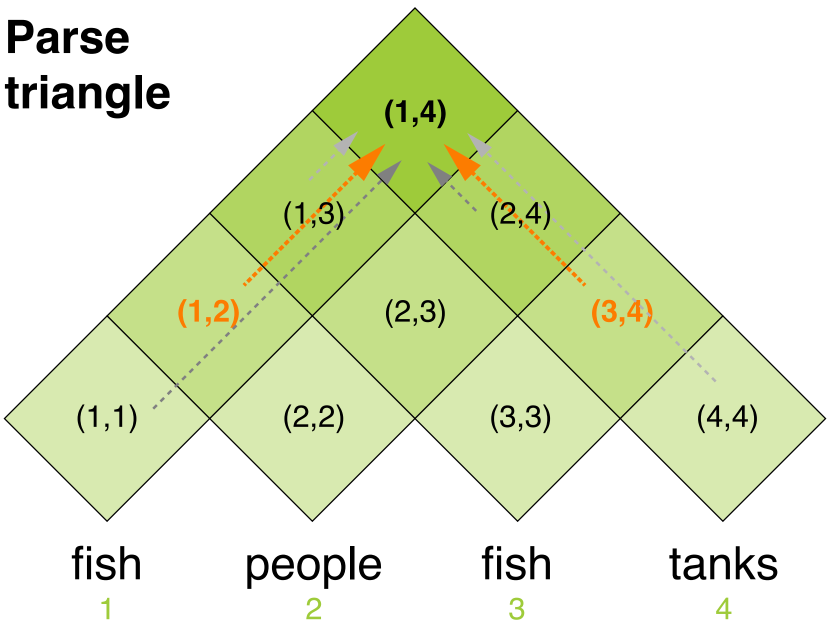
Cocke-Kasami-Younger (CKY) Parsing

PCFGs in CNF can be parsed via dynamic programming:

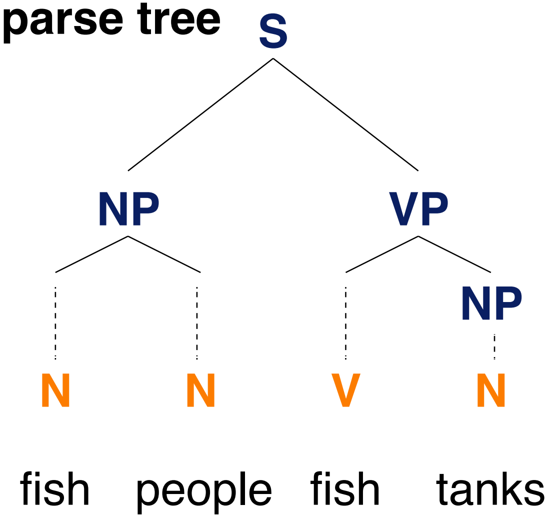
- CKY finds the most likely parse tree from the PCFGs probabilities.
- With a CFG in CNF, CKY parses in cubic time and quadratic space.

  With respect to the length of the sentence and the number of non-terminals.

Parse triangle

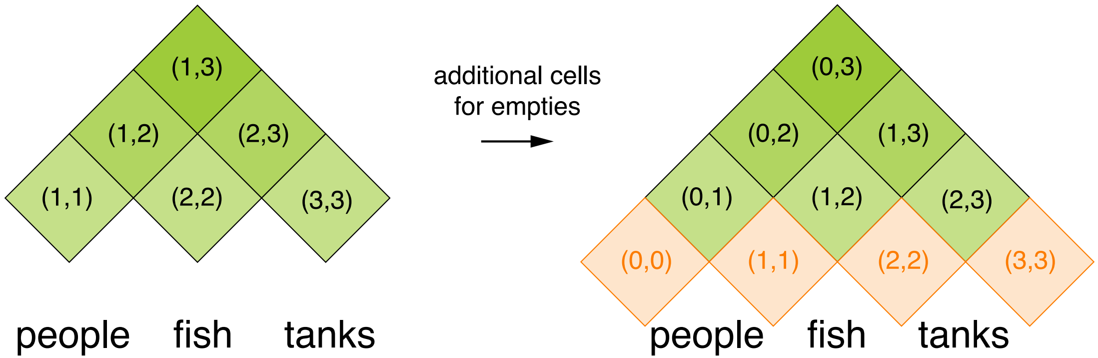


Most likely parse tree



Remarks:

- The binarization from the CNF is crucial for cubic time.
- CKY can be extended to include Unaries and Empties without increasing time complexity. This just makes the algorithm more messy:



Phrase Structure Grammars

CKY Parsing: Pseudo Code 1/2

Signature

- Input. A sentence (represented by a list of tokens), a binarized PCFG.
- Output. The most likely parse tree of the sentence.

```
extendedCKYParsing(List<Token> tokens, PCFG (Σ, N, S, R, P))

    double [][][] probs ← new double[#tokens][#tokens][#N]
    for int i ← 1 to #tokens do // Lexical rules (and unaries)
        for each U ∈ N do
            if (U→tokens[i]) ∈ P then
                probs[i][i][U] ← P(U→tokens[i])

        // ... handle unaries...

    // ... continued on next slide...
```

Parsing based on a PCFG

CKY Parsing: Pseudo Code 1/2

Signature

- Input. A sentence (represented by a list of tokens), a binarized PCFG.
- Output. The most likely parse tree of the sentence.

```
extendedCKYParsing(List<Token> tokens, PCFG (Σ, N, S, R, P))
    double [][][] probs ← new double[#tokens][#tokens][#N]
    for int i ← 1 to #tokens do // Lexical rules (and unaries)
        for each U ∈ N do
            if (U→tokens[i]) ∈ P then
                probs[i][i][U] ← P(U→tokens[i])
        boolean added ← 'true' // As of here: Handle unaries
        while added = 'true' do
            added ← 'false'
            for each U,V ∈ N do
                if probs[i][i][V]>0 and (U → V) ∈ P then
                    double prob ← P(U → V) · probs[i][i][V]
                    if prob > probs[i][i][U] then
                        probs[i][i][U] ← prob
                        added ← 'true'
    // ... continued on next slide...
```

Phrase Structure Grammars

CKY Parsing: Pseudo Code 2/2

```
        // ...  lines 1-14 on previous slide...
15.     for int length ← 2 to #tokens do // Structural rules
16.        for int beg ← 1 to #tokens - length + 1 do
17.           int end ← beg + length - 1
18.           for int split ← beg to end-1 do
19.
20.
                 // ...
21.
22.
23.
24.
25.
26.           // ...  handle unaries...
27.
28.
29.
30.
31.     return buildTree(probs) // Reconstruct tree from triangle
```

Phrase Structure Grammars

CKY Parsing: Pseudo Code 2/2

```
          // ...  lines 1-14 on previous slide...
15.       for int length ← 2 to #tokens do // Structural rules
16.           for int beg ← 1 to #tokens - length + 1 do
17.               int end ← beg + length - 1
18.               for int split ← beg to end-1 do
19.                   for int U,V,W ∈ N do
20.                       int prob ← probs[beg][split][V] ·
                                     probs[split+1][end][W] · P(U → V W)
21.                       if prob > probs[beg][end][U] then
22.                           probs[beg][end][U] ← prob
23.
24.
25.
26.               // ...  handle unaries...
27.
28.
29.
30.
31.       return buildTree(probs) // Reconstruct tree from triangle
```

Phrase Structure Grammars

CKY Parsing: Pseudo Code 2/2

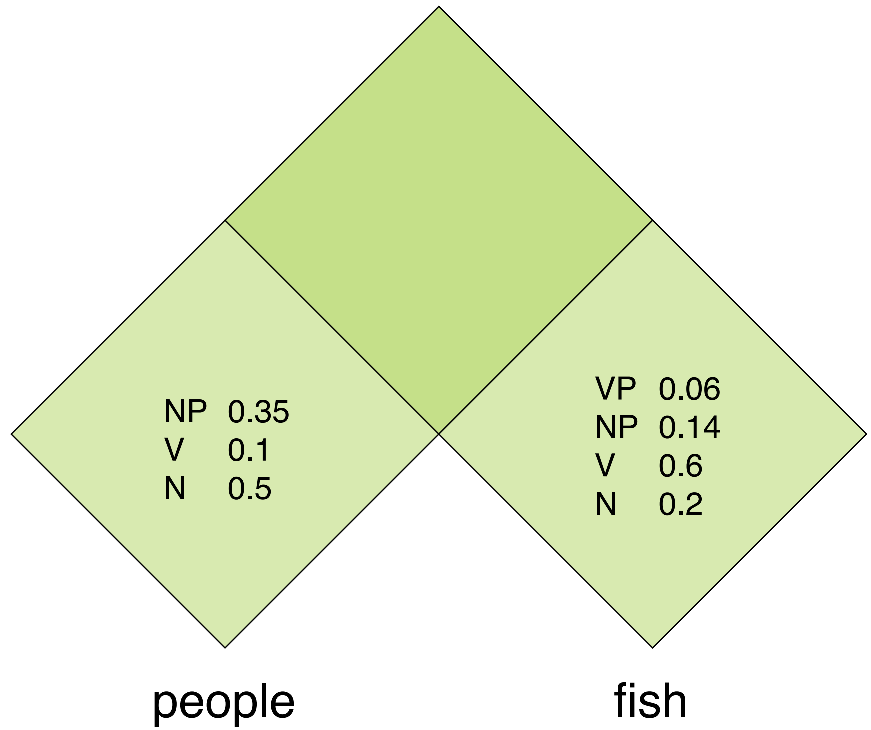
```
        // ...  lines 1-14 on previous slide...
15.     for int length ← 2 to #tokens do // Structural rules
16.         for int beg ← 1 to #tokens - length + 1 do
17.             int end ← beg + length - 1
18.             for int split ← beg to end-1 do
19.                 for int U,V,W ∈ N do
20.                     int prob ← probs[beg][split][V] ·
                                   probs[split+1][end][W] · P(U → V W)
21.                     if prob > probs[beg][end][U] then
22.                         probs[beg][end][U] ← prob
23.             boolean added ← 'true' // As of here:  Handle unaries
24.             while added do
25.                 added ← 'false'
26.                 for U,V ∈ N do
27.                     prob = P(U → V) · probs[beg][end][V];
28.                     if prob > probs[beg][end][U] then
29.                         probs[beg][end][U] ← prob
30.                         added ← 'true'
31.     return buildTree(probs) // Reconstruct tree from triangle
```

Phrase Structure Grammars

CKY Parsing: Example

A binarized PCFG

Structural rules		
s1	S → NP VP	0.9
s1’	S → VP	0.1
s2	VP → V NP	0.5
s2’	VP → V	0.1
s3’	VP → V VP_V	0.3
s3”	VP → V PP	0.1
s3”’	VP_V → NP PP	1.0
s4	NP → NP NP	0.1
s5	NP → NP PP	0.2
s6	NP → N	0.7
s7	PP → P NP	1.0



Filling cells

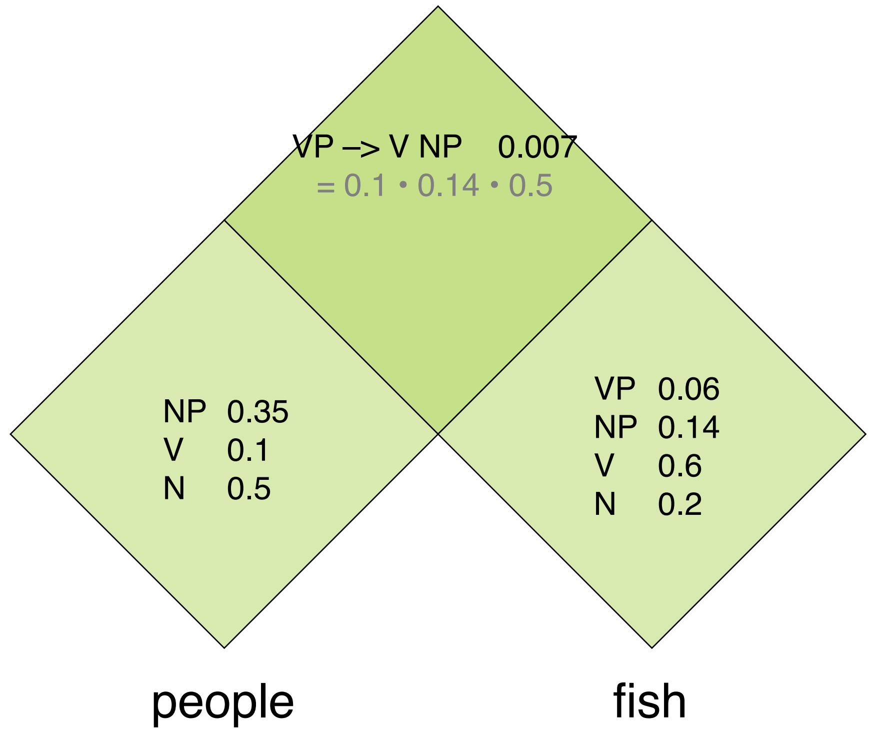
- Compute probabilities for each cell.
- Keep only highest for each left side.

Phrase Structure Grammars

CKY Parsing: Example

A binarized PCFG

Structural rules		
s1	S → NP VP	0.9
s1’	S → VP	0.1
s2	VP → V NP	0.5
s2’	VP → V	0.1
s3’	VP → V VP_V	0.3
s3”	VP → V PP	0.1
s3”’	VP_V → NP PP	1.0
s4	NP → NP NP	0.1
s5	NP → NP PP	0.2
s6	NP → N	0.7
s7	PP → P NP	1.0



Filling cells

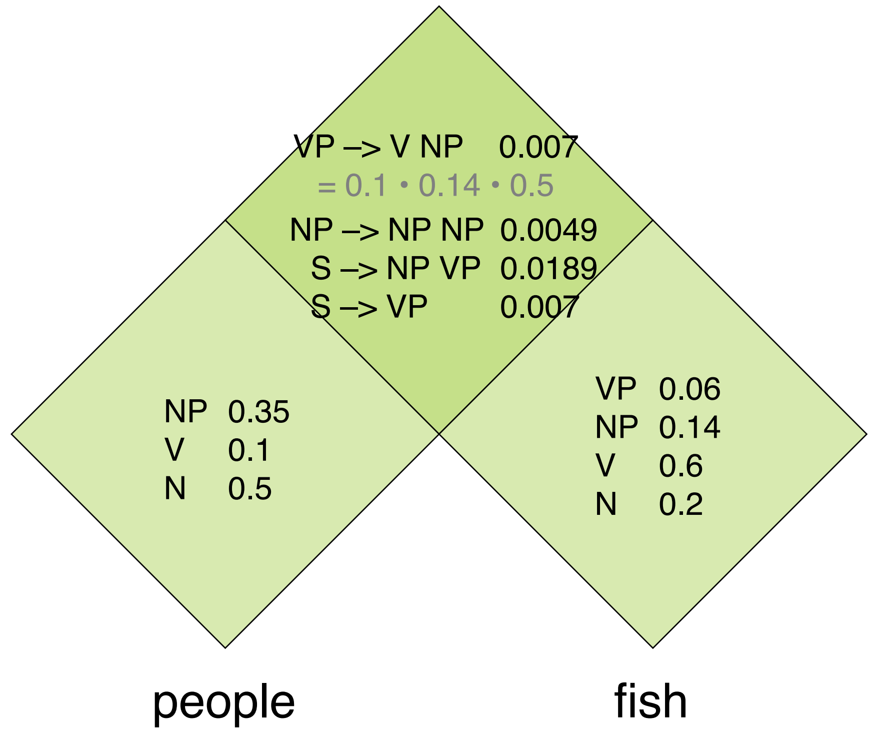
- Compute probabilities for each cell.
- Keep only highest for each left side.

Phrase Structure Grammars

CKY Parsing: Example

A binarized PCFG

Structural rules		
s1	$S \rightarrow NP\ VP$	0.9
s1’	$S \rightarrow VP$	0.1
s2	$VP \rightarrow V\ NP$	0.5
s2’	$VP \rightarrow V$	0.1
s3’	$VP \rightarrow V\ VP\_V$	0.3
s3”	$VP \rightarrow V\ PP$	0.1
s3”’	$VP\_V \rightarrow NP\ PP$	1.0
s4	$NP \rightarrow NP\ NP$	0.1
s5	$NP \rightarrow NP\ PP$	0.2
s6	$NP \rightarrow N$	0.7
s7	$PP \rightarrow P\ NP$	1.0

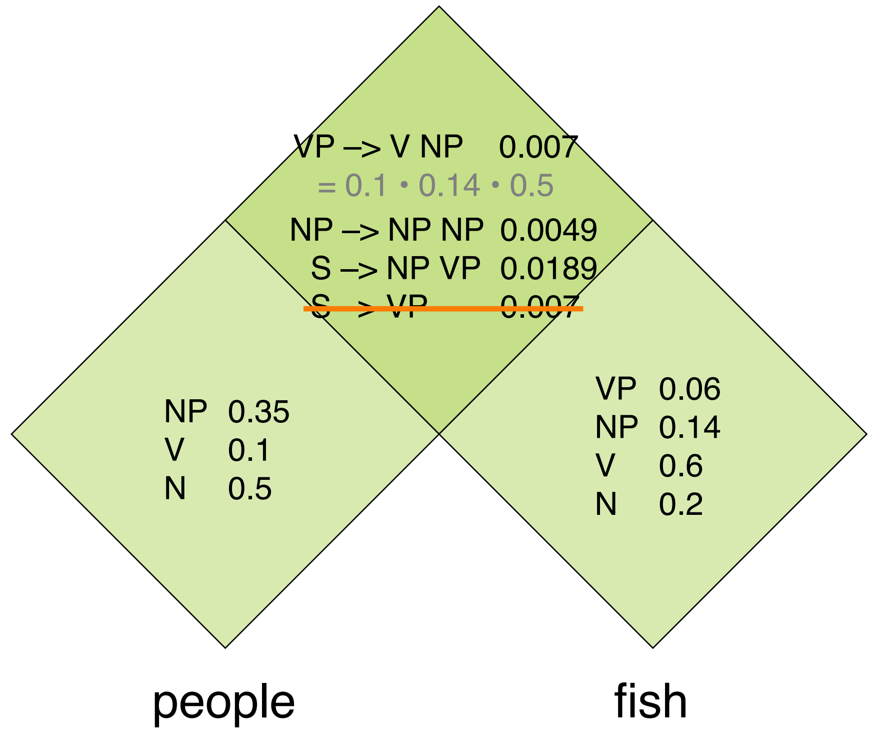


Filling cells

- Compute probabilities for each cell.
- Keep only highest for each left side.

A binarized PCFG

Structural rules		
s1	S → NP VP	0.9
s1’	S → VP	0.1
s2	VP → V NP	0.5
s2’	VP → V	0.1
s3’	VP → V VP_V	0.3
s3”	VP → V PP	0.1
s3”’	VP_V → NP PP	1.0
s4	NP → NP NP	0.1
s5	NP → NP PP	0.2
s6	NP → N	0.7
s7	PP → P NP	1.0



Filling cells

- Compute probabilities for each cell.
- Keep only highest for each left side.

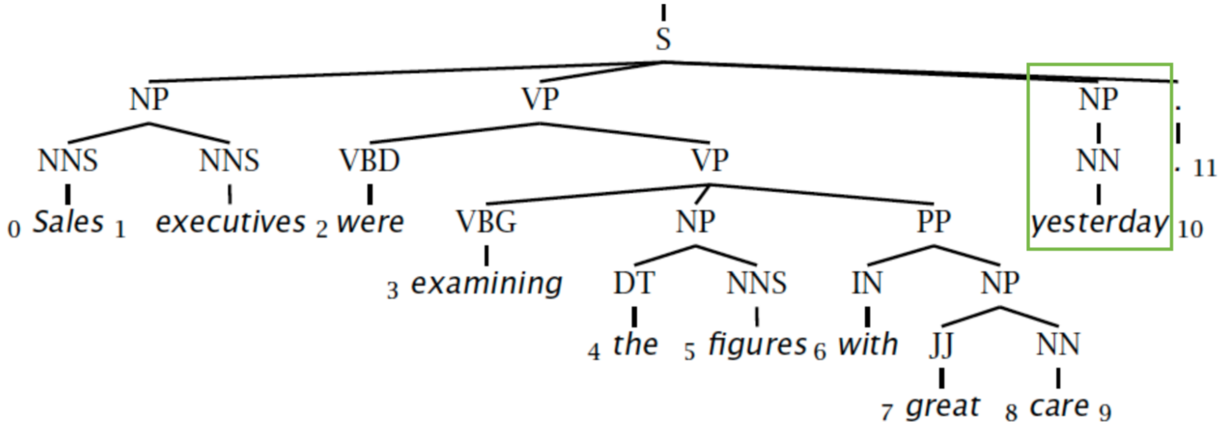
Remarks:

- CKY complexity of pseudo code part 1 is $\mathcal{O}(n \cdot |N|^2)$
  - $\mathcal{O}(n)$ times for-loop in lines 1–14, $n = \#$ tokens.
  - $\mathcal{O}(|N|)$ times for-loop in lines 3–5.
  - $\mathcal{O}(|N|^2)$ times while-loop in lines 7–14.
- CKY complexity of pseudo code part 2 is $\mathcal{O}(n^3 \cdot |N|^3)$
  - $\mathcal{O}(n)$ times for-loop in lines 15–30.
  - $\mathcal{O}(n)$ times for-loop in lines 16–30.
  - $\mathcal{O}(n)$ times for-loop in lines 18–22.
  - $\mathcal{O}(|N|^3)$ times for-loop in lines 19–22.
  - $\mathcal{O}(|N|^2)$ times while-loop in lines 24–30.
  - $\mathcal{O}(n^2)$ for building the tree in line 31.
- Extended CKY parsing has a runtime of $\mathcal{O}(n^3 \cdot |N|^3)$.

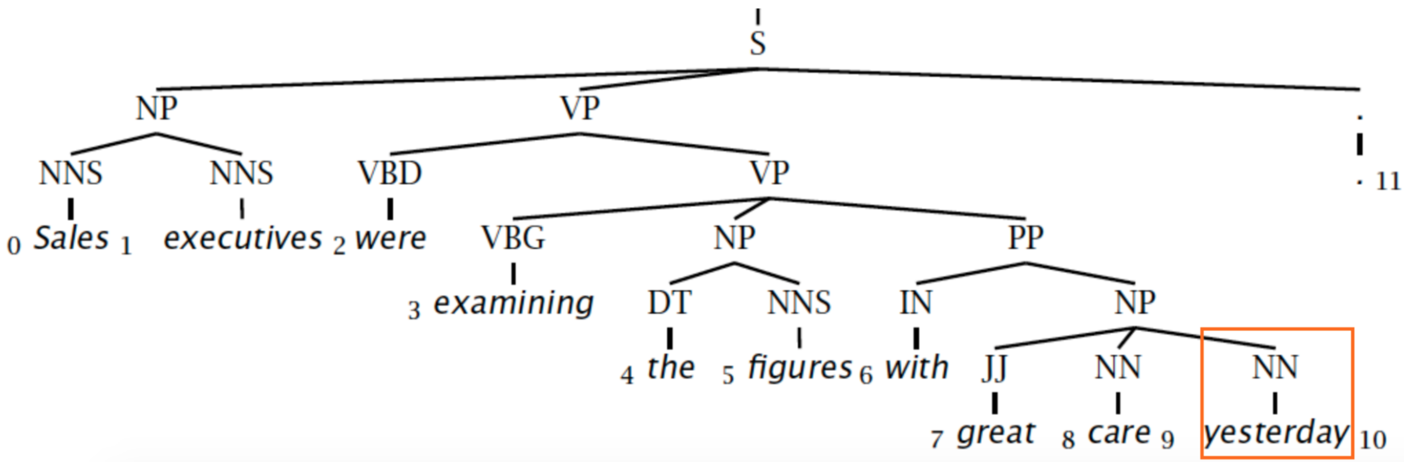
Remarks:

- CKY Parsing: Evaluation of Effectiveness

Gold standard brackets: **S-(0:11), NP-(0:2)**, VP-(2:9), VP-(3:9), **NP-(4:6)**, PP-(6-9), NP-(7,9), NP-(9:10)



Candidate brackets: **S-(0:11), NP-(0:2)**, VP-(2:10), VP-(3:10), **NP-(4:6)**, PP-(6-10), NP-(7,10)



Remarks:

- CKY Parsing: Evaluation of Effectiveness (continued)

  8 gold standard brackets
  S-(0:11), NP-(0:2), VP-(2:9), VP-(3:9), NP-(4:6), PP-(6:9), NP-(7,9), NP-(9:10)

  7 candidate brackets
  S-(0:11), NP-(0:2), VP-(2:10), VP-(3:10), NP-(4:6), PP-(6:10), NP-(7,10)

  Effectiveness in the example

  – Labeled precision (LP). 0.429 $=$ 3 / 7
  – Labeled recall (LR). 0.375 $=$ 3 / 8
  – Labeled F_1-score. 0.400 $=$ 2 · LP · LR / (LP + LR)
  – POS tagging accuracy. 1.000 $=$ 11 / 11

  Effectiveness of CKY in general [Charniak, 1997]

  – Labeled F_1 $\sim$0.73 when trained and tested on Penn Treebank.
  – CKY is robust (i.e., usually parses everything, but returns tiny probabilities).

Phrase Structure Grammars

Lexicalization

Problem: Probabilistic CFGs assume that the syntax is independent from the terminal symbols.

- PCFGs use production rules for parsing and parse tree probabilities for syntactic disambiguation.
- Information from the words is lost.
- Extending PCFGs by adding contraints from a lexicon is called lexicalization.

There are several PSG formalisms with varying degree of lexicalization:

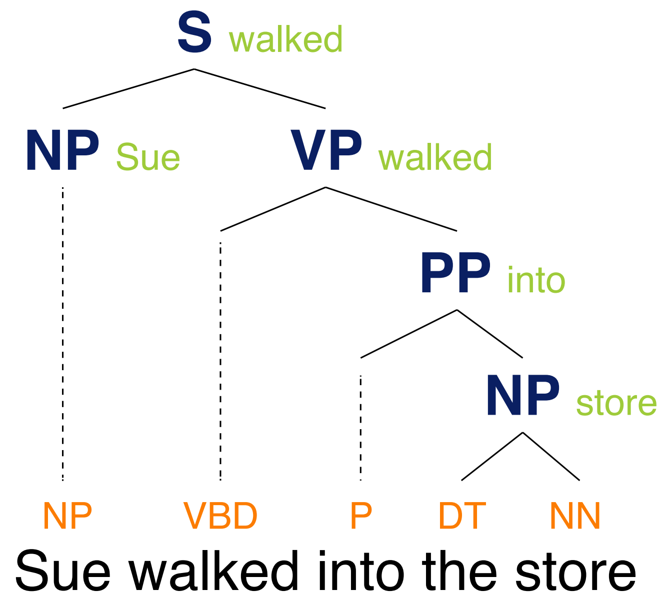
- Lexical-Function Grammar [Bresnan, 1982]
- Head-driven Phrase Structure Grammar [Pollard and Sag, 1994]
- Tree-Adjoining Grammar [Joshi, 1985]
- Combinatory Categorical Grammar
- ...

Phrase Structure Grammars

Lexicalized PCFG parsing[Collins, 1999]

Idea: The head word of a phrase gives a good representation of the phrase's structure and meaning.

$$P(\texttt{VP} \rightarrow \texttt{VBD PP}) = 0.00151$$
$$P(\texttt{VP} \rightarrow \texttt{VBD PP} \mid \text{said}) = 0.00001$$
$$P(\texttt{VP} \rightarrow \texttt{VBD PP} \mid \text{gave}) = 0.01980$$
$$P(\texttt{VP} \rightarrow \texttt{VBD PP} \mid \text{walked}) = 0.02730$$



Phrase Structure Grammars

Unlexicalization[Klein and Manning, 2003]

Idea: Lexicality is less important than grammatical features like verb form, presence of a verb auxiliary, . . .

- Rules are not systematically specified down to the level of lexical items.
- No semantic lexicalization for nouns, such as “$\text{NP}_{\text{stocks}}$”.
- Instead: Structural “lexicalization”, such as “$\text{NP}^{\text{S}}_{\text{CC}}$”.
  Meaning: Parent node is “S” and noun phrase is coordinating.
- Keep functional lexicalization of closed-class words, such as “VB-have”.
- Extension: learn the information that is stored for each non-terminal from the annotations. [Petrov and Knight, 2007]

Phrase Structure Grammars

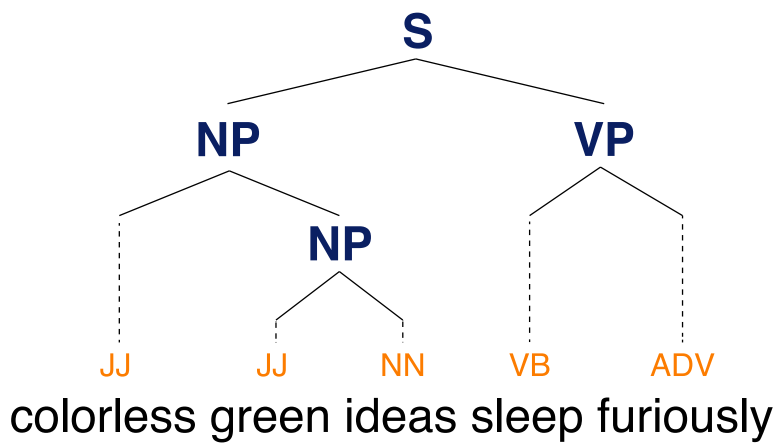
Linearized parsing[Vinyals, Kaiser, et al., 2015]

Idea: Linearize the parse tree and use sequence processing. i.e. conditional language modeling. The sentence is the input, the linearized parse tree the target.

Linearize with a depth-first traversal of the parse tree:

(S	`At the start of the traversal`
(D	`If descending to a non-terminal` **D**
)$_A$	`If ascending from a non-terminal` **A**
t	`If descending to a terminal` **t**
)$_S$	`At the end of the traversal`

Example with total outputs after each state:



1. (S
2. (S (NP
3. (S (NP JJ
4. (S (NP JJ (NP
5. (S (NP JJ (NP JJ
6. (S (NP JJ (NP JJ NN
7. (S (NP JJ (NP JJ NN)$_{NP}$

   ...

14. (S (NP JJ (NP JJ NN)$_{NP}$)$_{NP}$ (VP VB ADV)$_{VP}$)$_S$

Remarks:

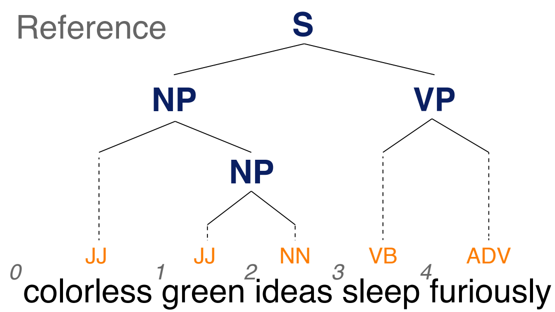
- Vinyals, Kaiser, et al. present linearaization as “Grammar as a Foreign Language”.
- They use a standard (in 2015 SoTA) machine translation neural network: an Encoder produces a representation of the text and a Decoder predicts the linearized parse tree.

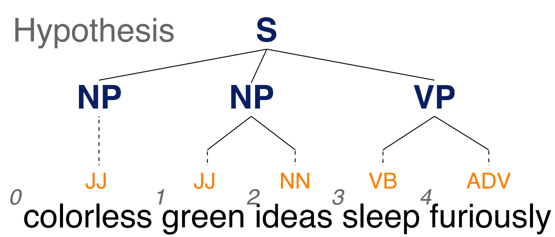
Phrase Structure Grammars

Evaluation[Sekine and Collins, evalb]

Given two different hypothesis parses, determine which is most similar to the reference parse by comparing common constituents.

- Each constituent spans a continuous range of text and has a label.
- Define each constituent as a triplet
  `(label, start, end).`
- Precision: how many triplets in the hypothesis parse are also in the reference. correctness
- Recall: how many triplets in the reference are also in the hypothesis. sensitivity, completeness





Phrase Structure Grammars

Evaluation [Sekine and Collins, evalb]

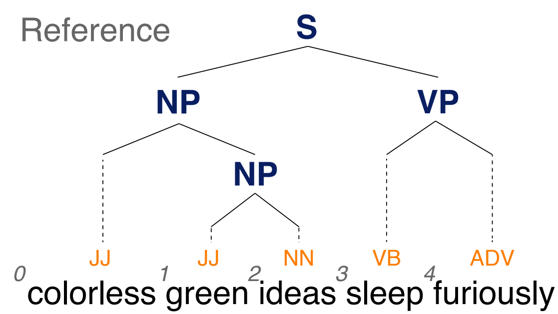
Given two different hypothesis parses, determine which is most similar to the reference parse by comparing common constituents.

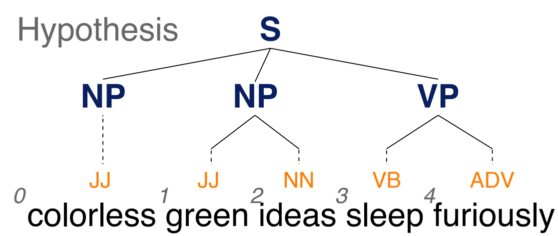
Parsers are evaluated with the harmonic mean (F_1) of the (averaged) labeled precision (LP) and labeled recall (LR):

$$\text{LP} = \frac{|\text{Triplets in hypothesis that are also in reference}|}{|\text{Triplets in hypothesis parse}|}$$

$$\text{LR} = \frac{|\text{Triplets in hypothesis that are also in reference}|}{|\text{Triplets in reference parse}|}$$

$$F_1 = \frac{2 \cdot LP \cdot LR}{LP + LR}$$





Phrase Structure Grammars

Evaluation[Sekine and Collins, evalb]

Given two different hypothesis parses, determine which is most similar to the reference parse by comparing common constituents.

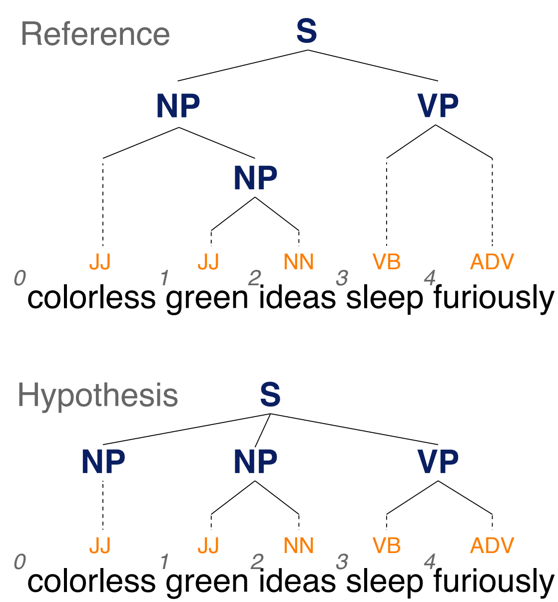
Parsers are evaluated with the harmonic mean (F_1) of the (averaged) labeled precision (LP) and labeled recall (LR):

$$\text{LP} = \frac{|\text{Triplets in hypothesis that are also in reference}|}{|\text{Triplets in hypothesis parse}|} = \frac{3}{4}$$

$$\text{LR} = \frac{|\text{Triplets in hypothesis that are also in reference}|}{|\text{Triplets in reference parse}|} = \frac{3}{4}$$

$$F_1 = \frac{2 \cdot LP \cdot LR}{LP + LR} = \frac{2 \cdot 0.75 \cdot 0.75}{0.75 + 0.75} = 0.75$$

Constituent Triples				
Reference parse	S(0,4)	NP(0,2)	NP(1,2)	VP(3,4)
Hypothesis parse	S(0,4)	NP(0,0)	NP(1,2)	VP(3,4)



Remarks:

- Those evaluation measures were developed at the `PARSEVAL` Workshop in 1998 and are often refered with this name.
- Evalb is the reference implementation of the `PARSEVAL` measures.
- Evalb also includes the `cross-bracket` and `unlabeled P/R` metrtics.

Phrase Structure Grammars
Evaluation: Comparison of Methods

- All in exactly the same setting on the Penn Treebank.

Approach	**Source**	**Labeled F_1**
Extended CKY parsing	[Charniak, 1997]	0.73
Lexicalized parsing	[Collins, 1999]	0.89
Unlexicalized parsing	[Klein and Manning, 2003]	0.86
Learned unlexicalized parsing	[Petrov and Klein, 2007]	0.90
Combining parsers (Ensemble)	[Fossum and Knight, 2009]	0.92
Linearized parsing (Learning)	[Vinyals, Kaiser, et al., 2015]	0.92
CKY + learned disambiguation	[Zhang et al., 2020]	**0.96**

- Besides F_1 score, the time to parse 1,000 sentences is often considered too.
- Linearized methods are usually very fast. Ensemble methodes perform well but are slow.
- CKY profits a lot from batching and parallelization.

