Parsing

Kevin Duh

Intro to NLP, Fall 2019

Refresher

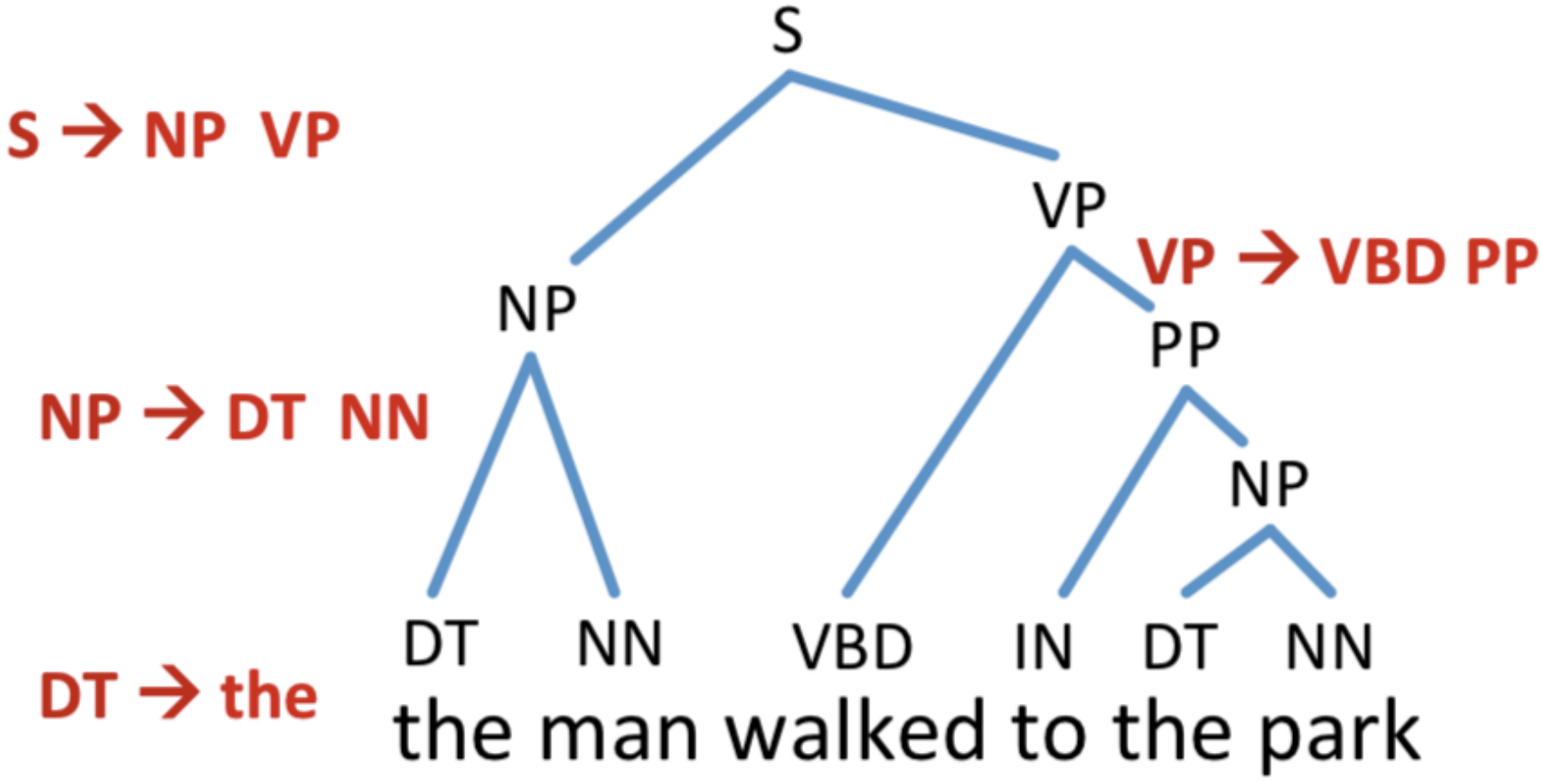
- Recall the Random Sentence Generator the CFG lectures and homework

S symbol

CFG rule application

Sentence

Example Sentence and Parse Tree



From: Kevin Gimpel, https://ttic.uchicago.edu/~kgimpel/teaching/31190/lectures/12.pdf

What is parsing?

- Parsing: given Sentence, find a legal (or most probable) parse tree (with S as root)

S symbol

CFG rule application

Sentence

Top-Down Parsing (v1)

- Run Random Sentence Generator until we get the sentence we want

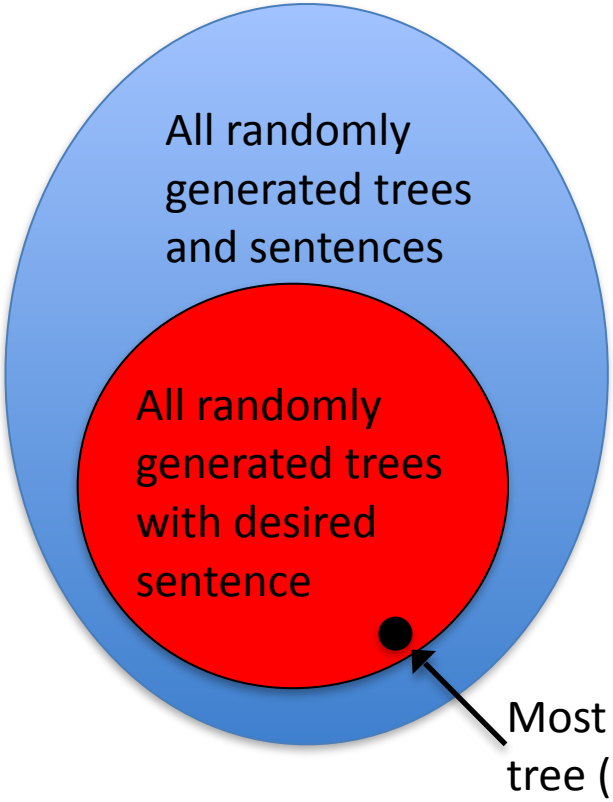
S symbol

CFG rule application

Sentence

Top-Down Parsing (v1)

- Run Random Sentence Generator until we get the sentence we want

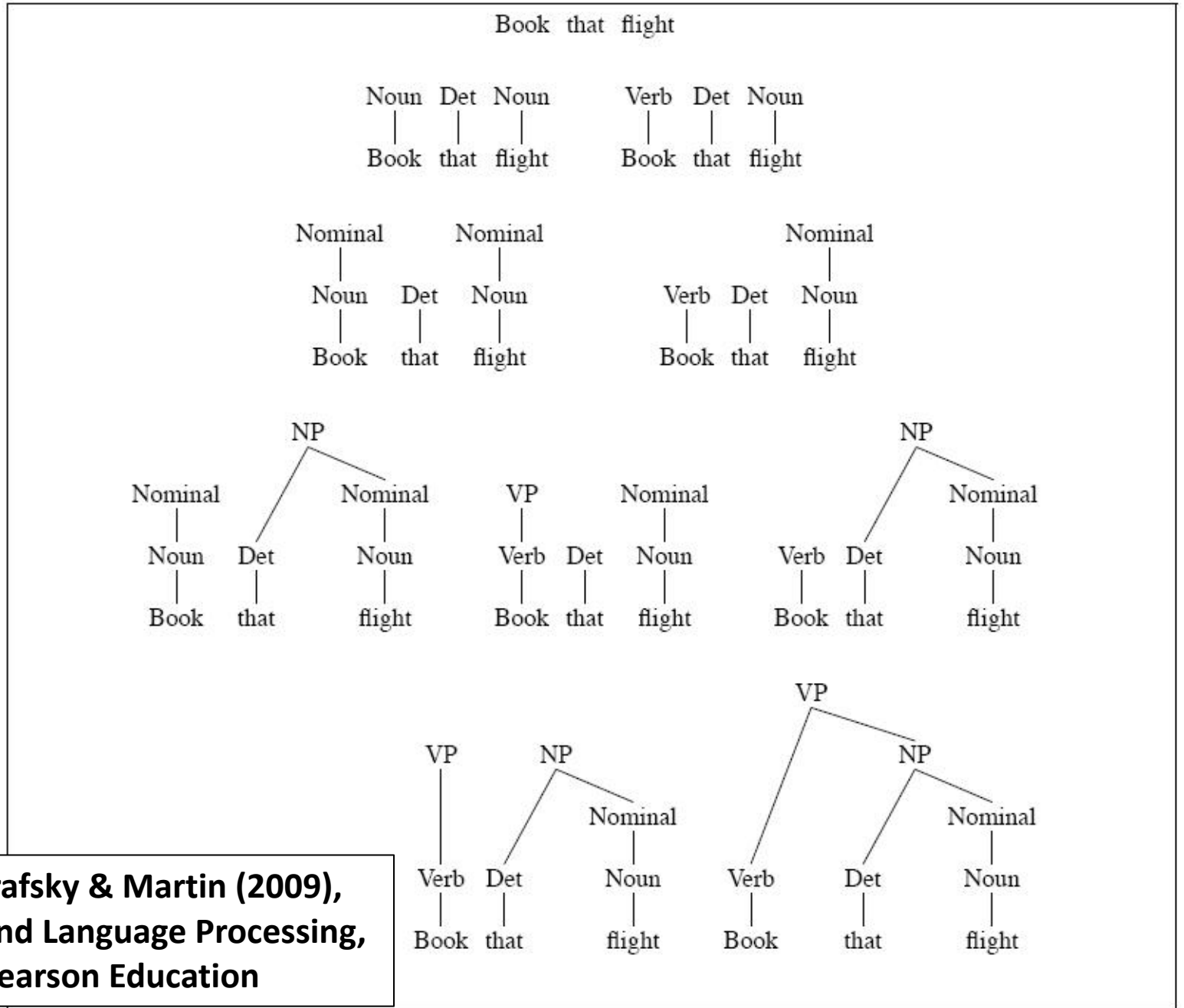


S symbol

CFG rule application

Sentence

Bottom-Up Parsing (v1)



From: Jurafsky & Martin (2009), Speech and Language Processing, 2nd ed., Pearson Education

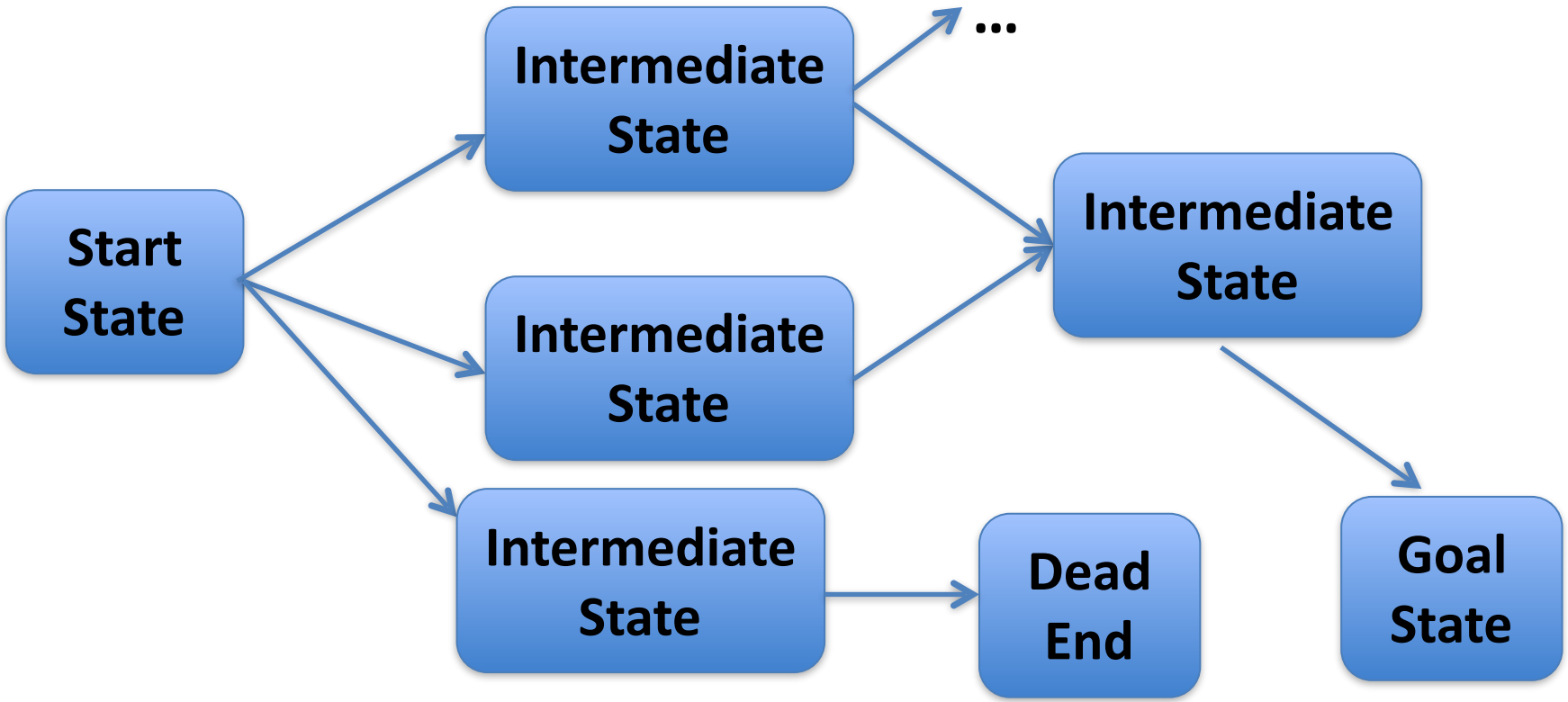
Overview

- Constituency Parsing
  - Parsing as Search; Dynamic Programming
  - CKY Algorithm
- Treebanks and Evaluation
- Dependency Parsing
  - Definitions
  - Eisner Algorithm

Top-Down vs. Bottom-Up

- Bottom-Up:
  - Doesn't waste time generating trees that don't get our desired sentence
- Top-Down:
  - Doesn't waste time generating trees that don't get to S symbol at the end

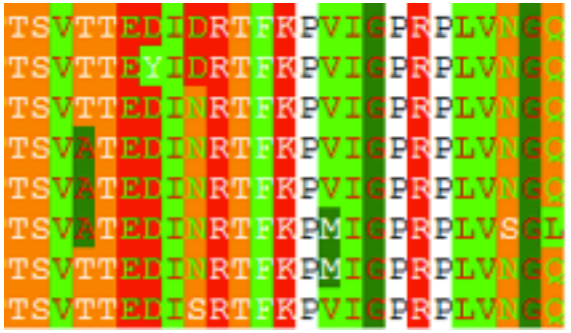
Parsing as a Search Problem



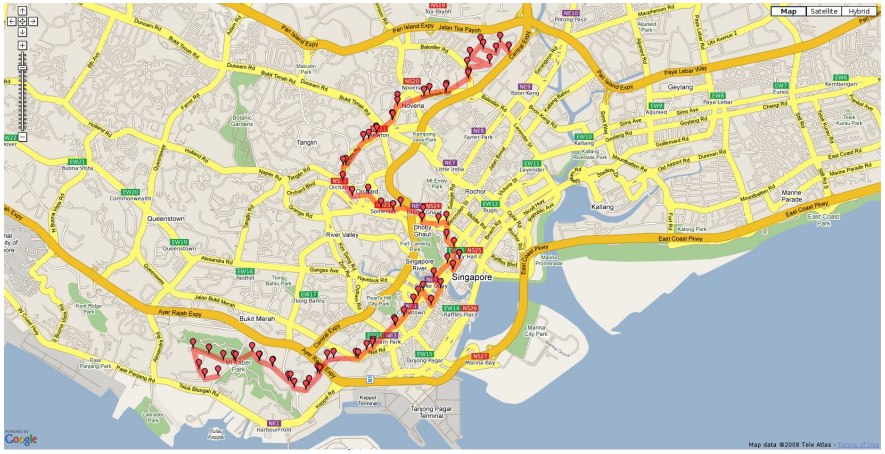
Many methods! Take an AI class. We'll focus on class of algorithms based on Dynamic Programming

Successful algorithms based on Dynamic Programming

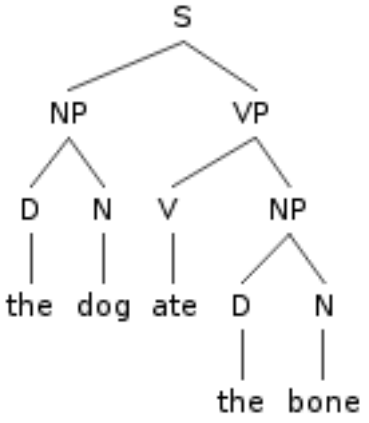
Viterbi Algorithm
for Digital Communications



Needleman-Wunsch Algorithm
for Biological Sequence Alignment



Bellman-Ford Algorithm
for Shortest Path on Graphs



Cocke-Younger-Kasami Algorithm
for parsing of language

What is Dynamic Programming

- A strategy for solving problems by
  - dividing into overlapping sub-problems, and
  - re-using sub-problem solutions

What is Dynamic Programming

- A strategy for solving problems by
  - dividing into overlapping sub-problems, and
  - re-using sub-problem solutions

- Can be thought as intelligent “brute-force”:
  - Try all possible solutions

    without actually trying

    all possible solutions

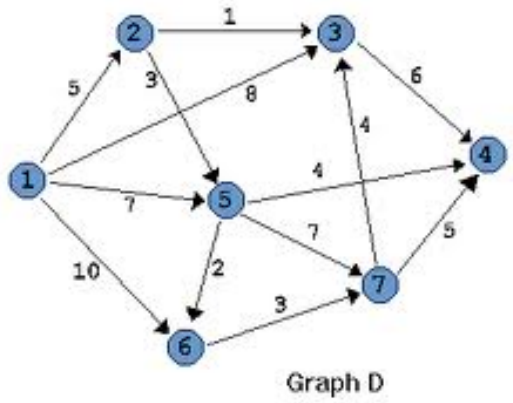
What is Dynamic Programming

- A strategy for solving problems by
  - dividing into overlapping sub-problems, and
  - re-using sub-problem solutions

- Can be thought as intelligent “brute-force”:
  - Try all possible solutions

    without actually trying

    all possible solutions



Graph D

Overview

- Constituency Parsing
  - Parsing as Search; Dynamic Programming
  - CKY Algorithm
- Treebanks and Evaluation
- Dependency Parsing
  - Definitions
  - Eisner Algorithm

CKY Algorithm for Parsing Trees

CKY Algorithm for Parsing Trees

- Assume Chomsky Normal Form (CNF)

CKY Algorithm for Parsing Trees

- Assume Chomsky Normal Form (CNF)
- Rules must be
  - RHS has 2 non-terminals: A → B C
  - RHS has 1 terminal: A → w

CKY Algorithm for Parsing Trees

- Assume Chomsky Normal Form (CNF)
- Rules must be
  - RHS has 2 non-terminals: A → B C
  - RHS has 1 terminal: A → w
- Bottom-up parsing

CKY Algorithm for Parsing Trees

- Assume Chomsky Normal Form (CNF)
- Rules must be
  - RHS has 2 non-terminals: A → B C
  - RHS has 1 terminal: A → w
- Bottom-up parsing
- Use an efficient data structure to store partial parsing sub-problem solutions

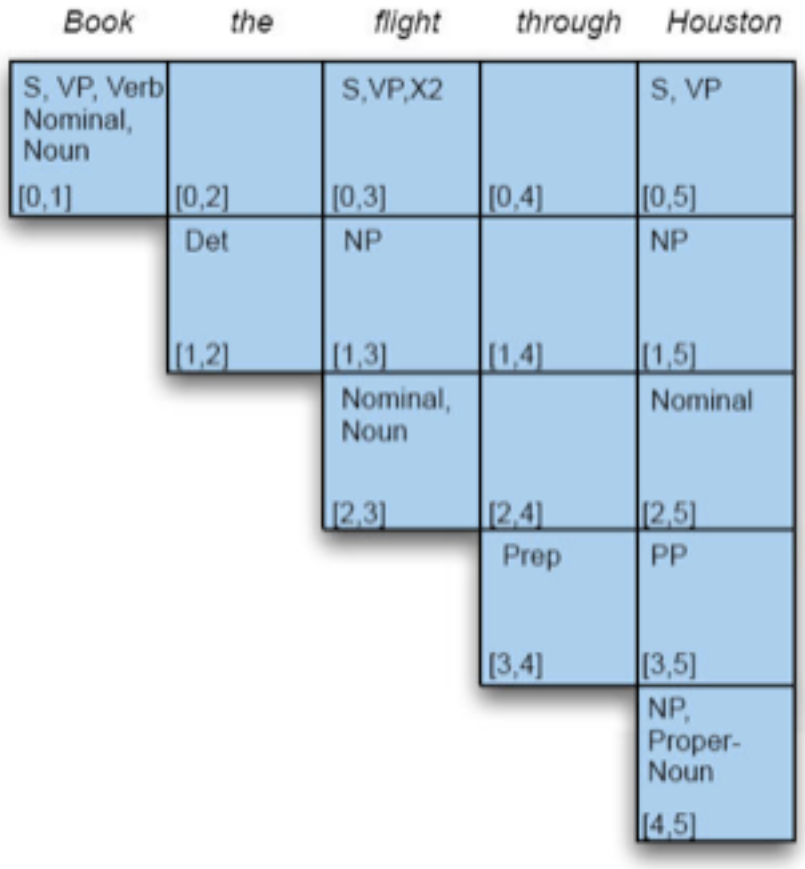
Conversion of any grammar to CNF

$\mathscr{L}_1$ Grammar	$\mathscr{L}_1$ in CNF
$S \rightarrow NP\ VP$	$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$	$S \rightarrow X1\ VP$
	$X1 \rightarrow Aux\ NP$
$S \rightarrow VP$	$S \rightarrow book \mid include \mid prefer$
	$S \rightarrow Verb\ NP$
	$S \rightarrow X2\ PP$
	$S \rightarrow Verb\ PP$
	$S \rightarrow VP\ PP$
$NP \rightarrow Pronoun$	$NP \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$NP \rightarrow TWA \mid Houston$
$NP \rightarrow Det\ Nominal$	$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow Noun$	$Nominal \rightarrow book \mid flight \mid meal \mid money$
$Nominal \rightarrow Nominal\ Noun$	$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$	$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow Verb$	$VP \rightarrow book \mid include \mid prefer$
$VP \rightarrow Verb\ NP$	$VP \rightarrow Verb\ NP$
$VP \rightarrow Verb\ NP\ PP$	$VP \rightarrow X2\ PP$
	$X2 \rightarrow Verb\ NP$
$VP \rightarrow Verb\ PP$	$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$	$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$	$PP \rightarrow Preposition\ NP$

From: Jurafsky & Martin (2009), Speech and Language Processing, 2nd ed., Pearson Education

Intuition

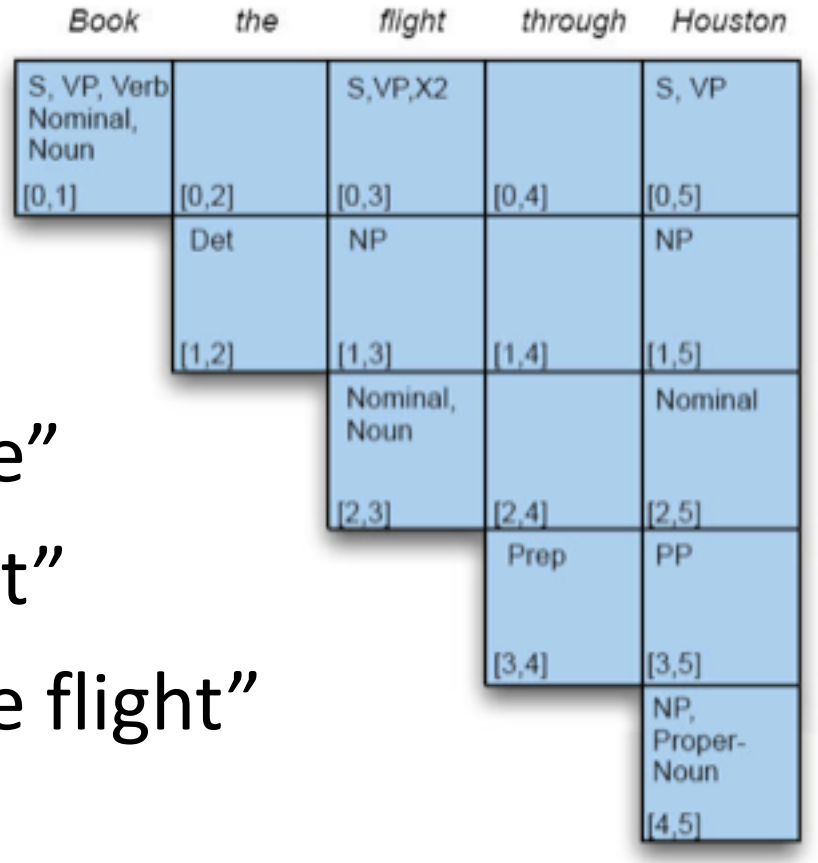
- 2-dim matrix encoding of solutions
- Each non-terminal above POS has exactly 2 children
- For each [i,j] span, there must be a position i<k<j that splits it into two constituents



From: Jurafsky & Martin (2009), Speech and Language Processing, 2nd ed., Pearson Education

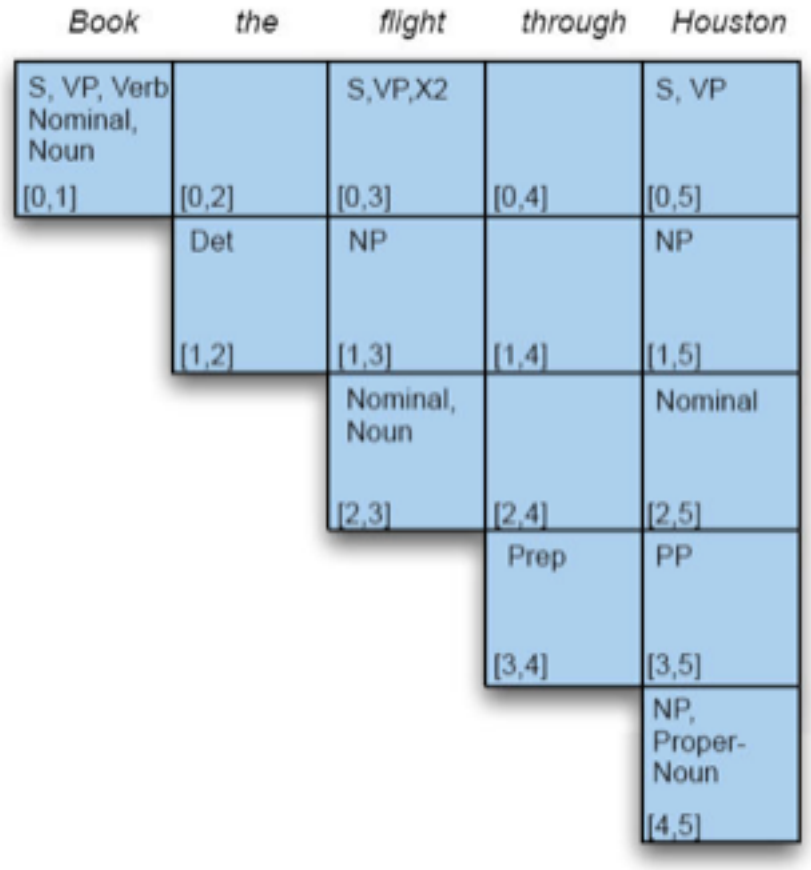
Example: Parsing
"Book the flight through Houston"

- $_0$ Book $_1$ the $_2$ flight $_3$ through $_4$ Houston $_5$
- [0,1] = parse for: "Book"
- [1,2] = parse for: "the"
- [2,3] = parse for: "flight"
- [0,2] = parse for: "Book the"
- [1,3] = parse for: "the flight"
- [0,3] = parse for: "Book the flight"
- [0,5] = parse for sentence



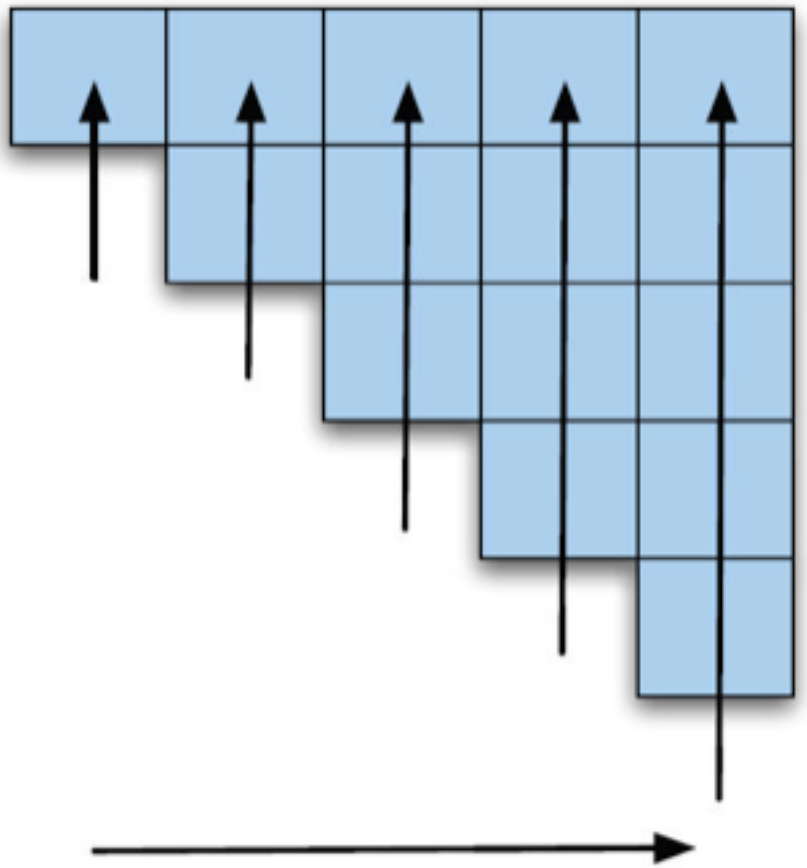
From: Jurafsky & Martin (2009), Speech and Language Processing, 2nd ed., Pearson Education

Example: Parsing
"Book the flight through Houston"



From: Jurafsky & Martin (2009), Speech and Language Processing, 2nd ed., Pearson Education

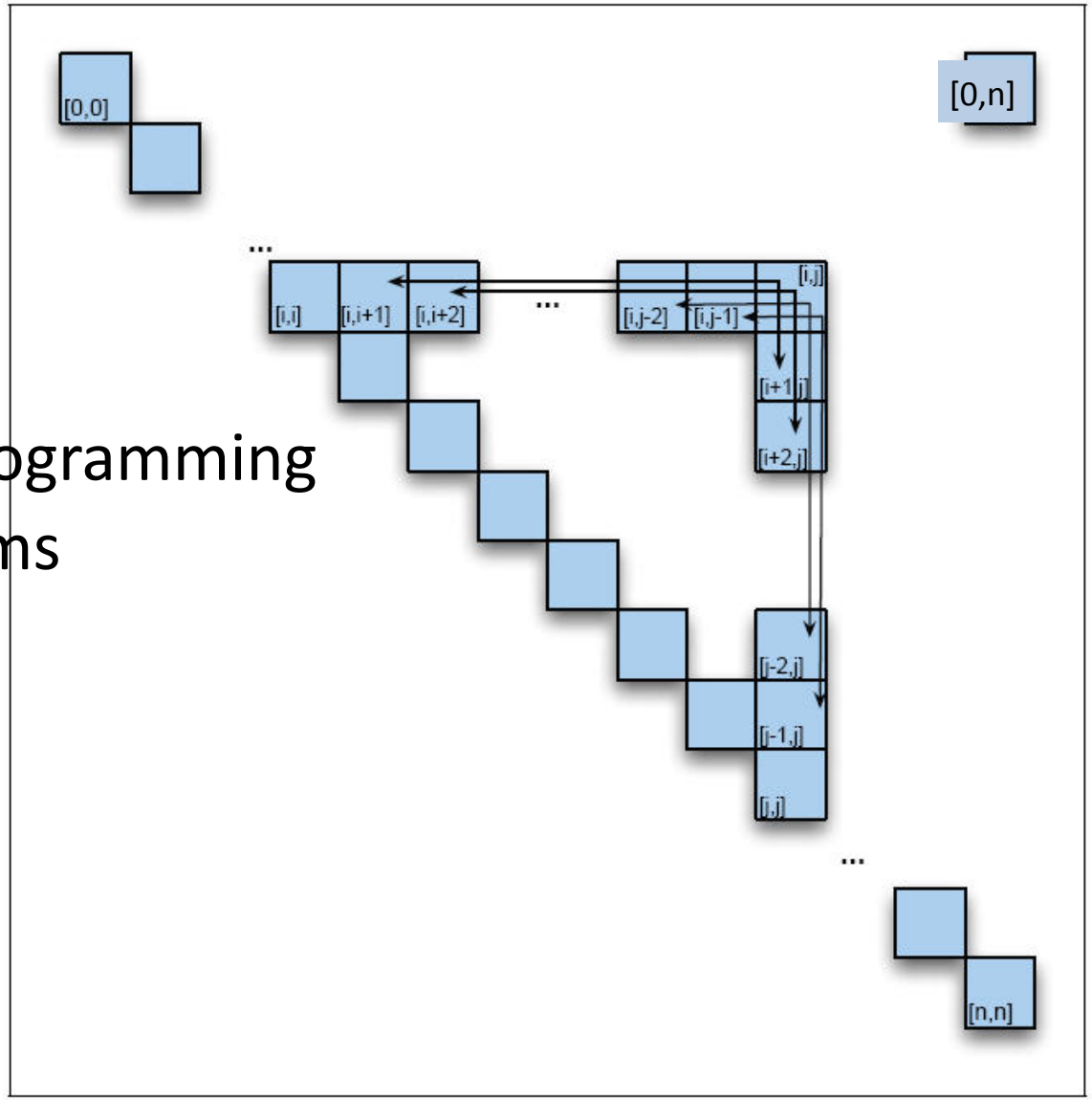
```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
      table[j − 1, j] ← {A | A → words[j] ∈ grammar}
      for i ← from j − 2 downto 0 do
          for k ← i + 1 to j − 1 do
              table[i,j] ← table[i,j] ∪
                           {A | A → BC ∈ grammar,
                                B ∈ table[i,k],
                                C ∈ table[k, j]}
```



Note: this finds all valid trees.
Think about how to modify this to find the most probable tree

From: Jurafsky & Martin (2009), Speech and Language Processing, 2nd ed., Pearson Education

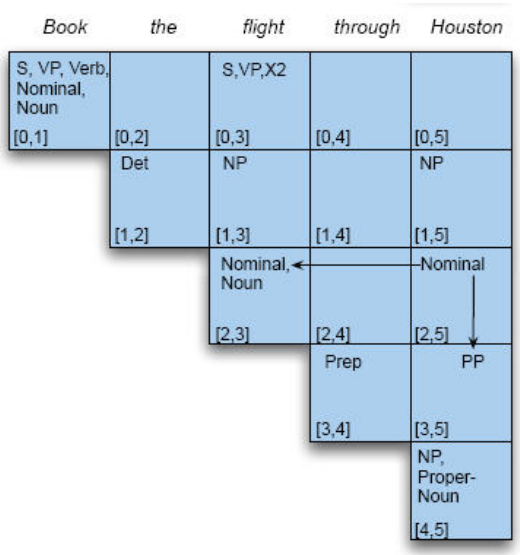
Dynamic Programming Sub-Problems

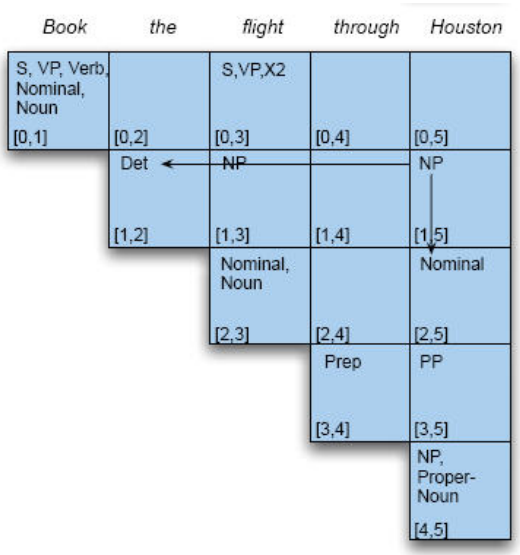


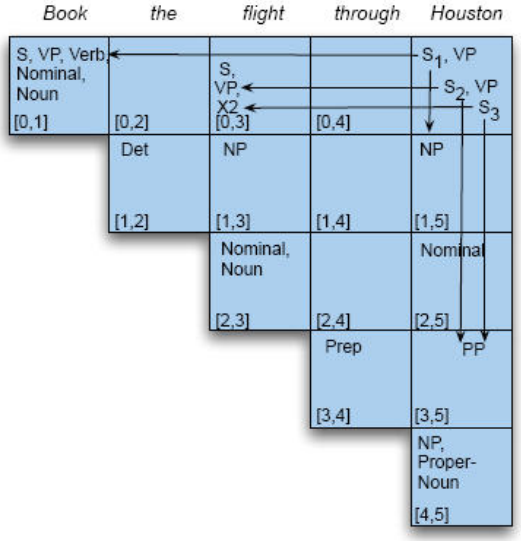
From: Jurafsky & Martin (2009), Speech and Language Processing, 2nd ed., Pearson Education

$\mathscr{L}_1$ in CNF

$S \rightarrow NP\ VP$
$S \rightarrow X1\ VP$
$X1 \rightarrow Aux\ NP$
$S \rightarrow book \mid include \mid prefer$
$S \rightarrow Verb\ NP$
$S \rightarrow X2\ PP$
$S \rightarrow Verb\ PP$
$S \rightarrow VP\ PP$
$NP \rightarrow I \mid she \mid me$
$NP \rightarrow TWA \mid Houston$
$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow book \mid flight \mid meal \mid money$
$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow book \mid include \mid prefer$
$VP \rightarrow Verb\ NP$
$VP \rightarrow X2\ PP$
$X2 \rightarrow Verb\ NP$
$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$







From: Jurafsky & Martin (2009), Speech and Language Processing, 2nd ed., Pearson Education

- CKY Exercise- Parse this sentence: **fruit flies like a banana**
- Grammar:
  - S → NP VP
  - NP → Adj Noun
  - NP → Det Noun
  - VP → Verb NP
  - VP → Verb PP
  - PP → Prep NP
  - Adj → fruit
  - Noun → fruit
  - NP → fruit
  - Noun → flies
  - Verb → flies
  - Verb → like
  - Prep → like
  - Det → a
  - Noun → banana

Overview

- Constituency Parsing
  - Parsing as Search; Dynamic Programming
  - CKY Algorithm
- Treebanks and Evaluation
- Dependency Parsing
  - Definitions
  - Eisner Algorithm

Treebank

- Treebank: A corpus of parsed sentences
- Penn Treebank
  - 40k sentences, mostly from Wall Street Journal

```
(S
  (SBAR-PRP
    (IN Because)
    (S
      (S
        (NP-SBJ (DT the) (NNP CD))
        (VP
          (VBD had)
          (NP
            (NP (DT an) (JJ effective) (NN yield))
            (PP (IN of) (NP (CD 13.4) (NN %))))
          (SBAR-TMP
            (WHADVP-4 (WRB when))
            (S
              (NP-SBJ-1 (PRP it))
              (VP
                (VBD was)
                (VP
                  (VBN issued)
                  (NP (-NONE- *-1))
                  (PP-TMP (IN in) (NP (CD 1984)))
                  (ADVP-TMP (-NONE- *T*-4)))))))))
                     ...
```

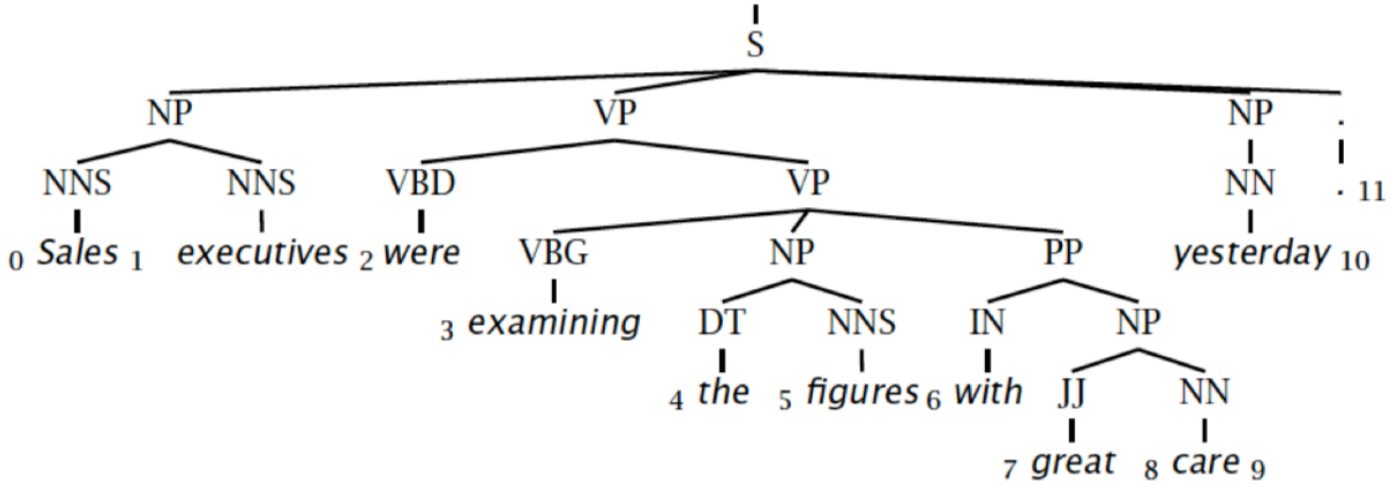
How to get probabilities for PCFG

- One method: Count rules from a treebank
- P(S→NP VP) = count(S→NP VP) / count(S)
- We can apply various smoothing techniques

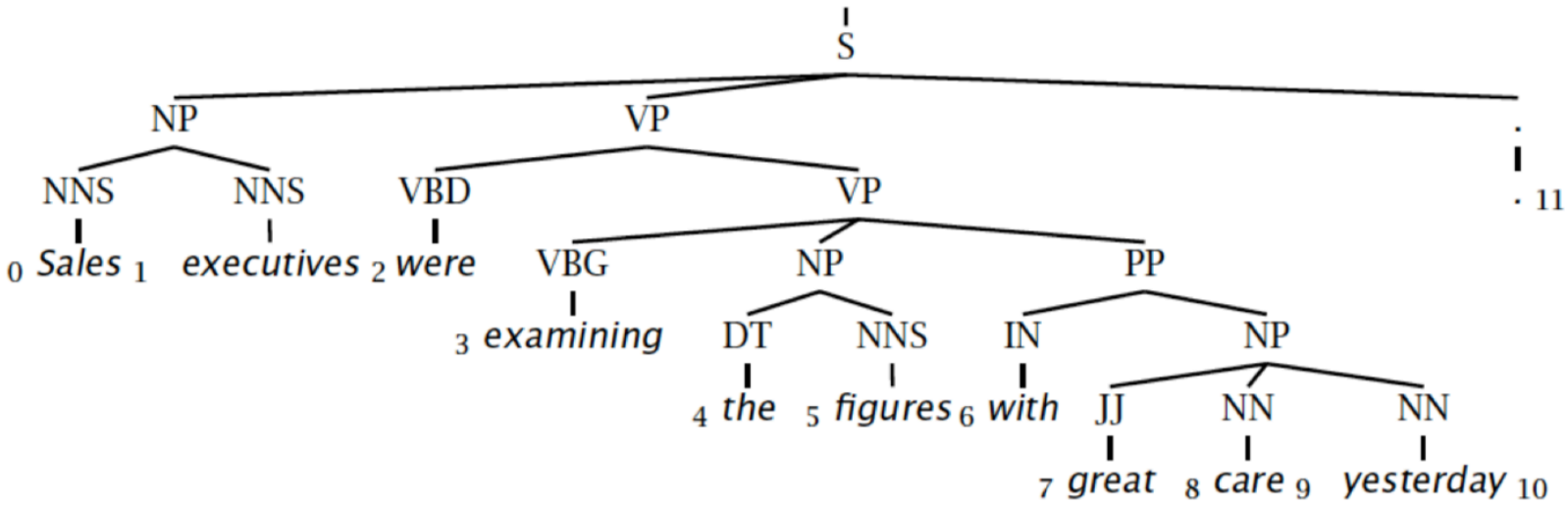
- Large research literature:
  - How to get best PCFG from treebank?
  - What if we don't have a treebank?

Evaluating Parse Results

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)



From: Yoav Artzi. http://www.cs.cornell.edu/courses/cs5740/2018sp/lectures/10-parsing-const.pdf

Evaluating Parse Results

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)

Recall = # correct candidate constituents / # gold constituents
= 3/8 = 37.5%

Precision = # correct candidate constituents / # constituents in candidate
= 3/7 = 42.9%

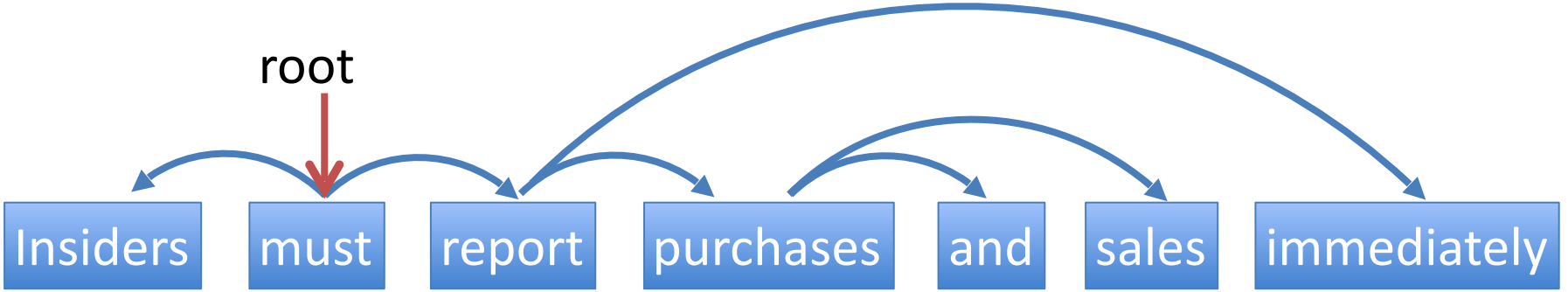
F1 = (2*precision*recall)/(precision+recall) i.e. harmonic mean
= 40%

From: Yoav Artzi. http://www.cs.cornell.edu/courses/cs5740/2018sp/lectures/10-parsing-const.pdf

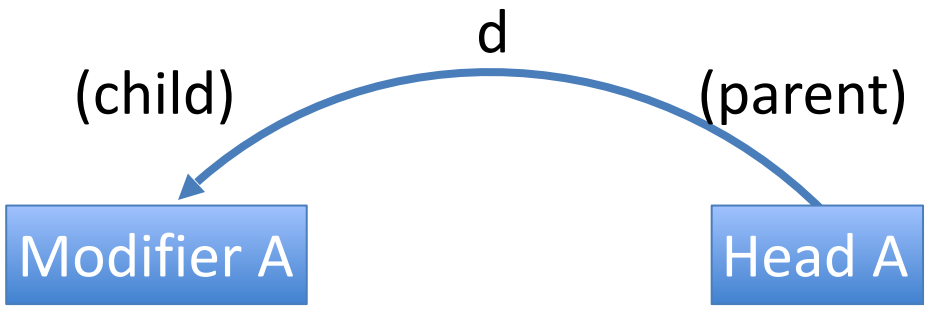
Overview

- Constituency Parsing
  - Parsing as Search; Dynamic Programming
  - CKY Algorithm
- Treebanks and Evaluation
- Dependency Parsing
  - Definitions
  - Eisner Algorithm

Dependency Tree

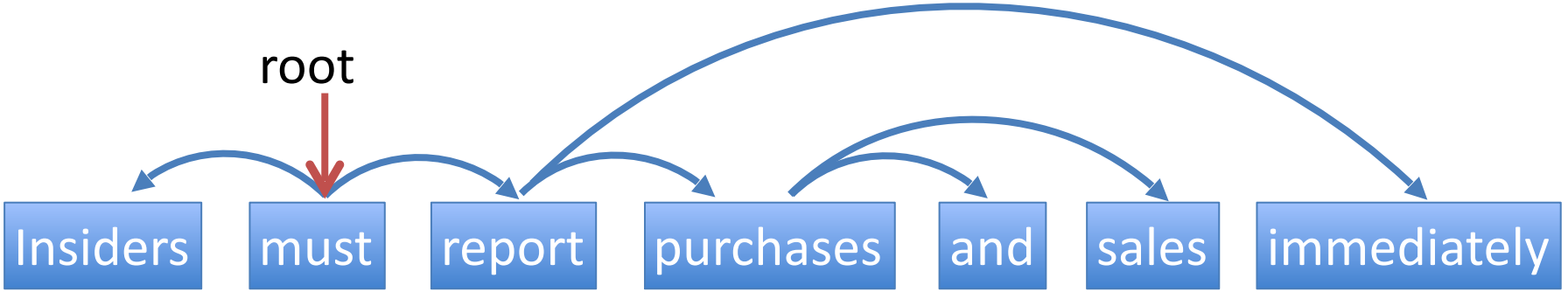


Represents head and modifier relationship



From: Akifumi Yoshimoto. http://hylos.jp/about

Dependency Parsing



- Input: Sentence (Word sequence)

$$S = \{w_1, \ldots, w_n\}$$

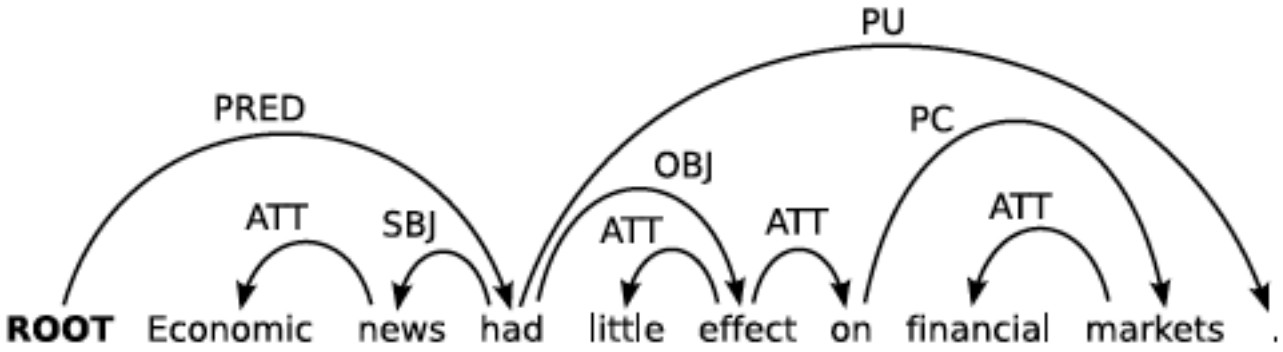
- Output: Dependency

$$D = \{d_1, \ldots, d_n\}$$

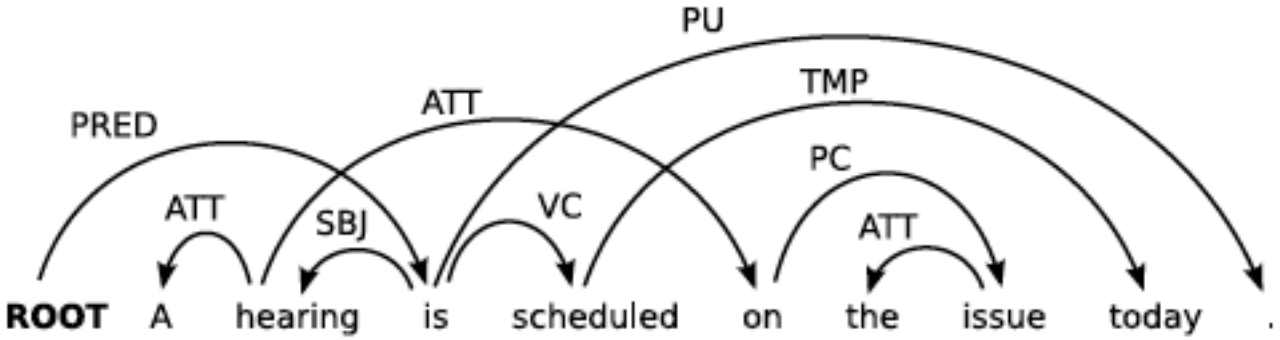
- All elements are acyclic, connected

From: Akifumi Yoshimoto. http://hylos.jp/about

Projective Dependency Trees: no crossings

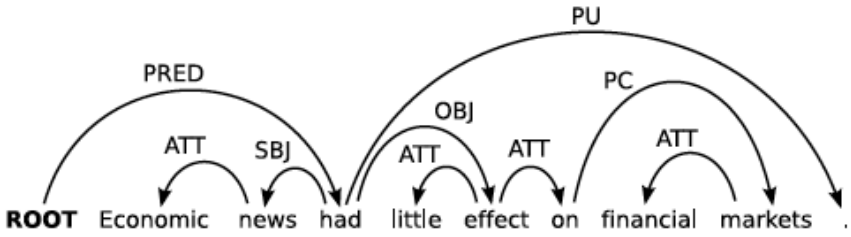


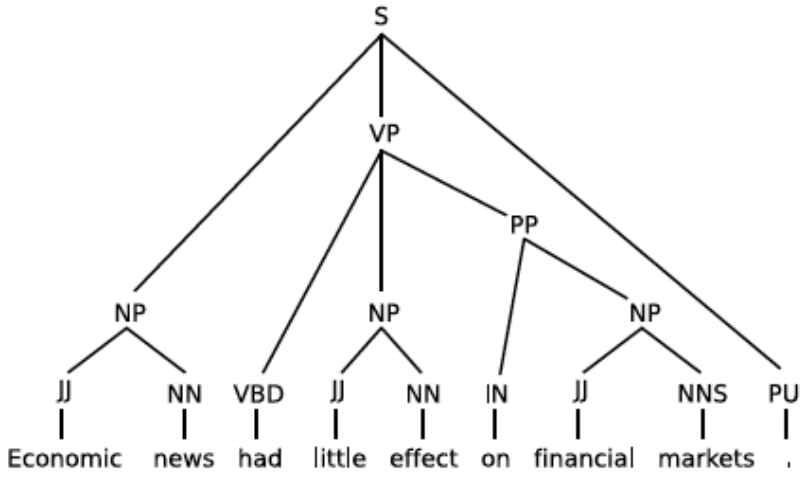
Non-Projective Dependency Trees



From: Kubler, McDonald, Nivre (2009). Dependency Parsing. Synthesis Lectures in HLT, Morgan & Claypool

Dependency vs Constituency

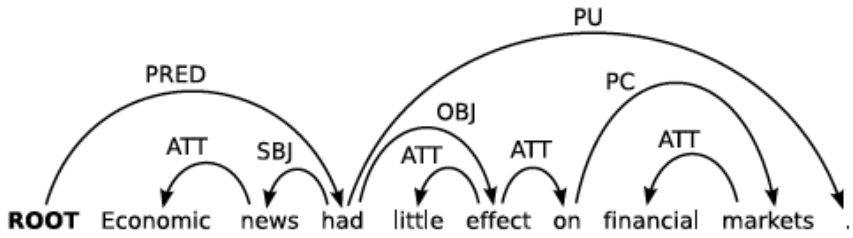


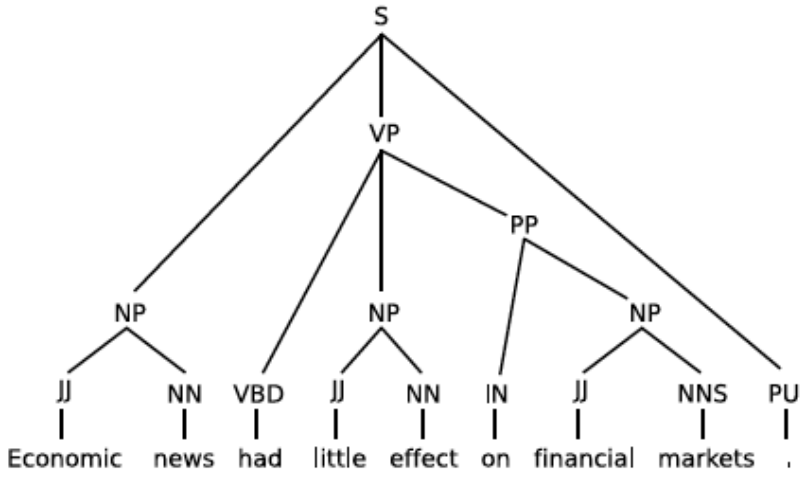


From: Kubler, McDonald, Nivre (2009). Dependency Parsing. Synthesis Lectures in HLT, Morgan & Claypool

Dependency vs Constituency

- Different ways to think about syntax
  - E.g. free word order in Czech

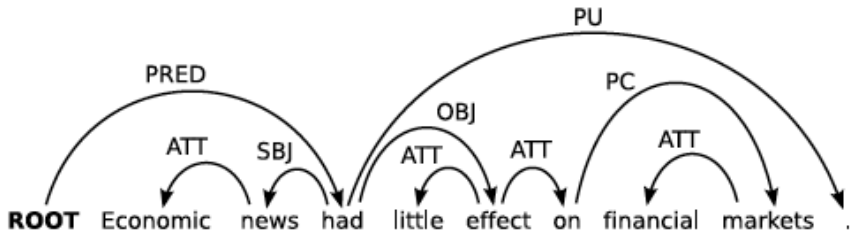


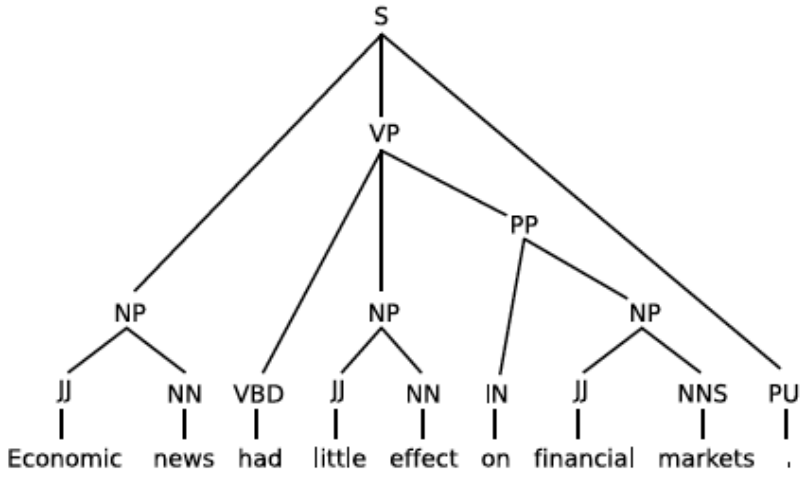


From: Kubler, McDonald, Nivre (2009). Dependency Parsing. Synthesis Lectures in HLT, Morgan & Claypool

Dependency vs Constituency

- Different ways to think about syntax
  - E.g. free word order in Czech
- Different applications need different formalisms

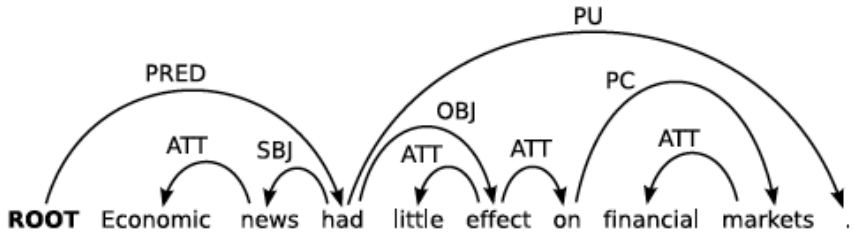


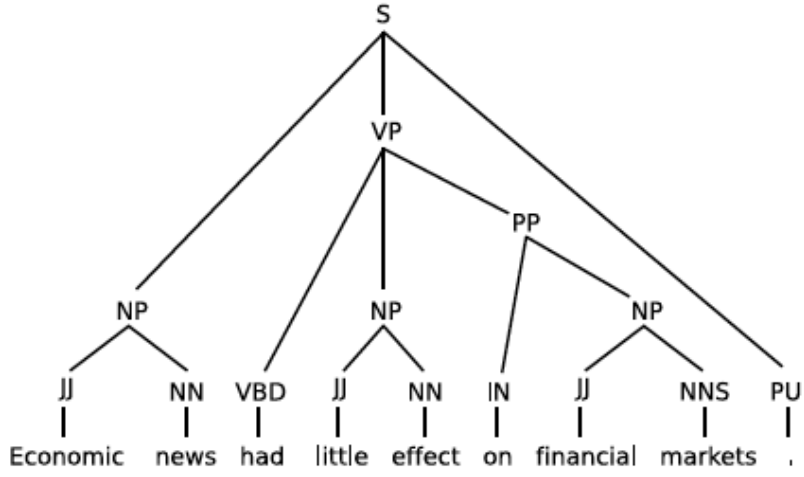


From: Kubler, McDonald, Nivre (2009). Dependency Parsing. Synthesis Lectures in HLT, Morgan & Claypool

Dependency vs Constituency

- Different ways to think about syntax
  - E.g. free word order in Czech
- Different applications need different formalisms
- Linguistic tradition based on language
  - E. Practical concerns:
  - Dependency treebanks might be easier to annotate
  - Constituency-Dependency conversion possible



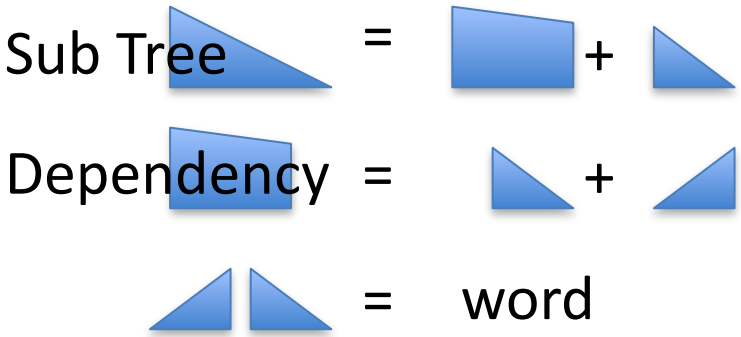


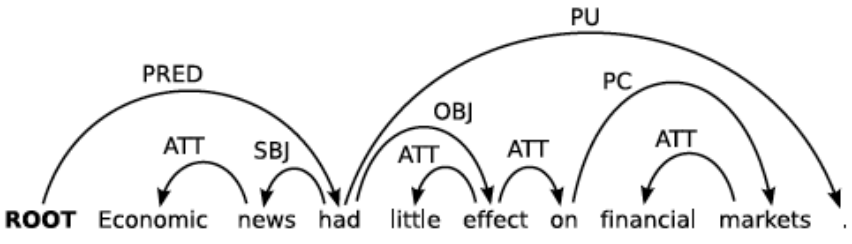
From: Kubler, McDonald, Nivre (2009). Dependency Parsing. Synthesis Lectures in HLT, Morgan & Claypool

The Eisner Algorithm

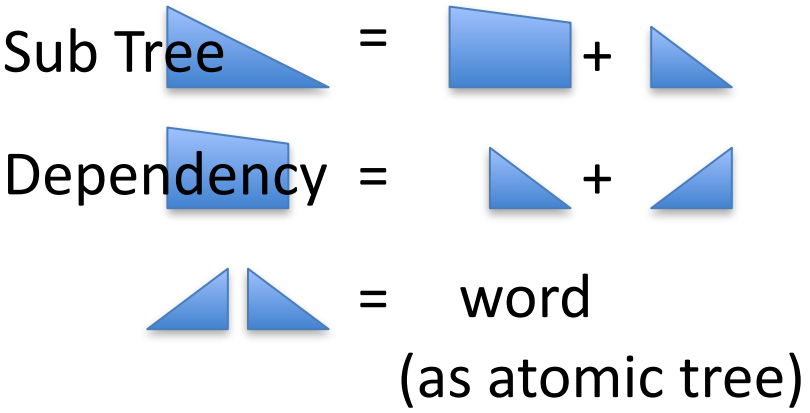
- A Bottom-Up Dynamic Program

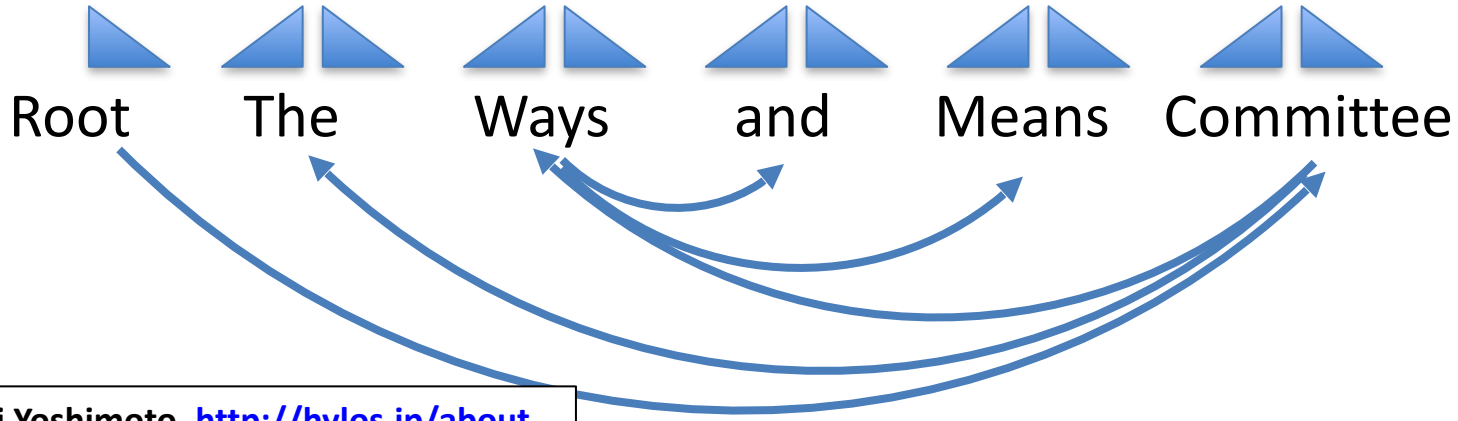
Rules





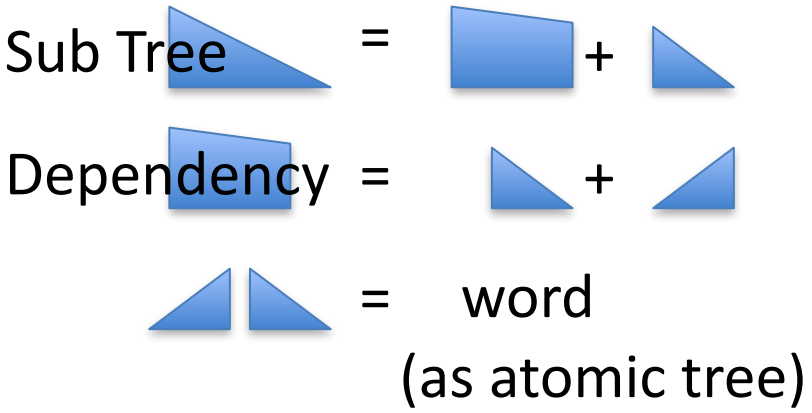
The Eisner Algorithm

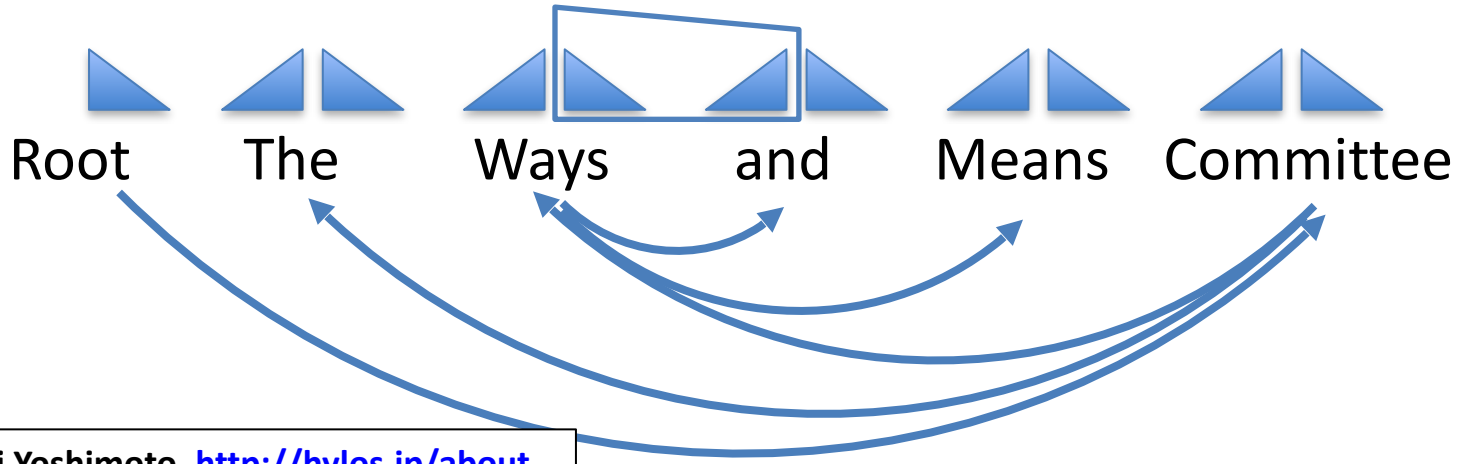




From: Akifumi Yoshimoto. http://hylos.jp/about

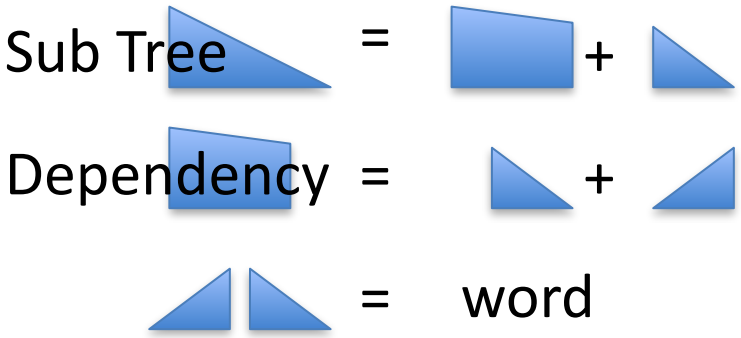
The Eisner Algorithm

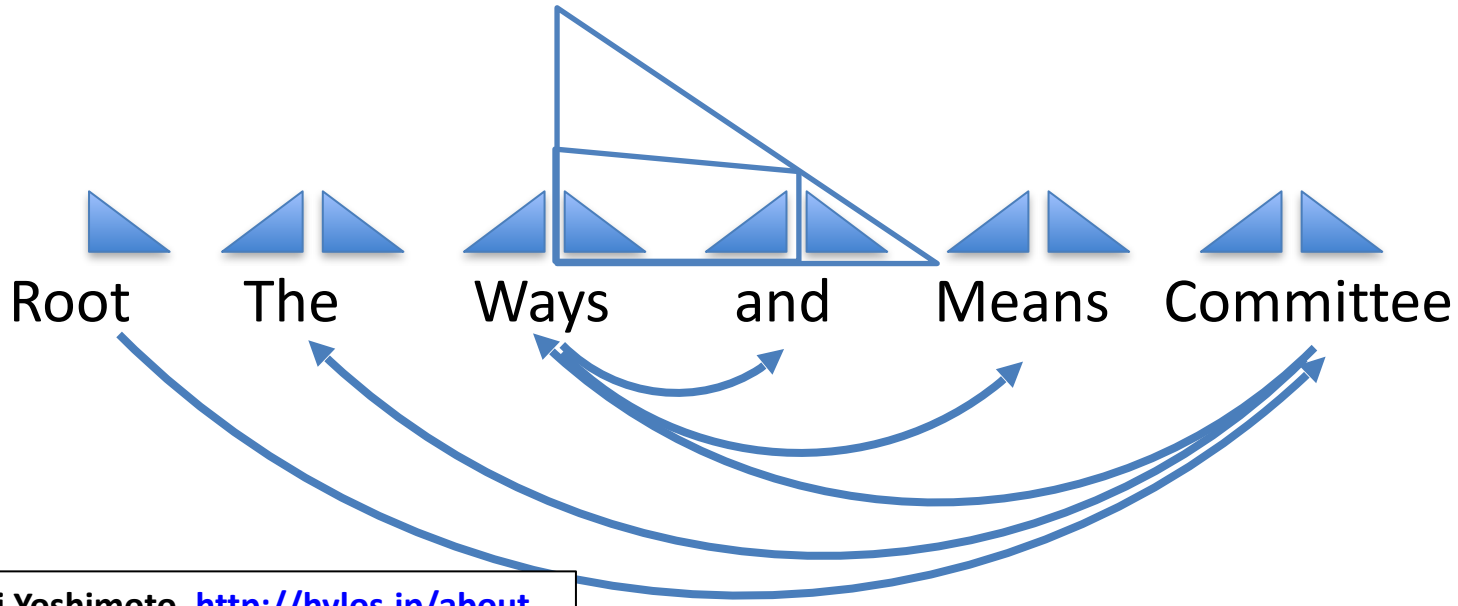




From: Akifumi Yoshimoto. http://hylos.jp/about

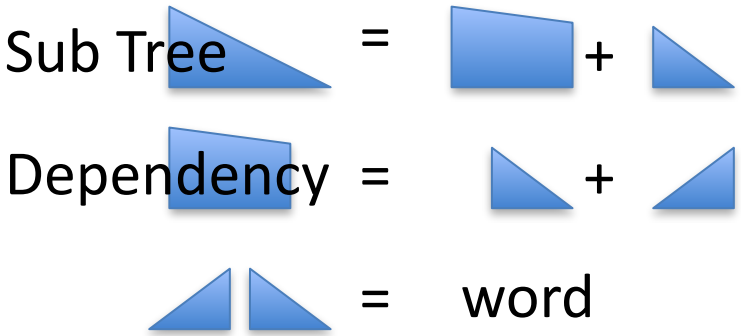
The Eisner Algorithm

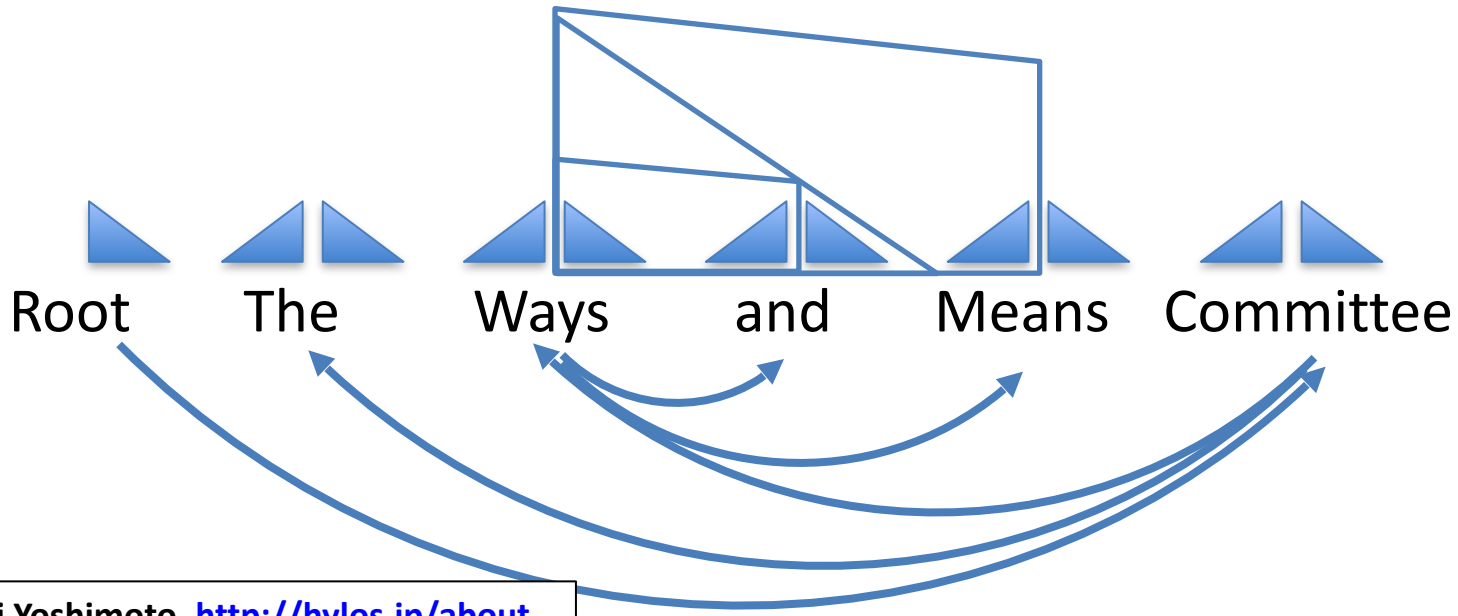




From: Akifumi Yoshimoto. http://hylos.jp/about

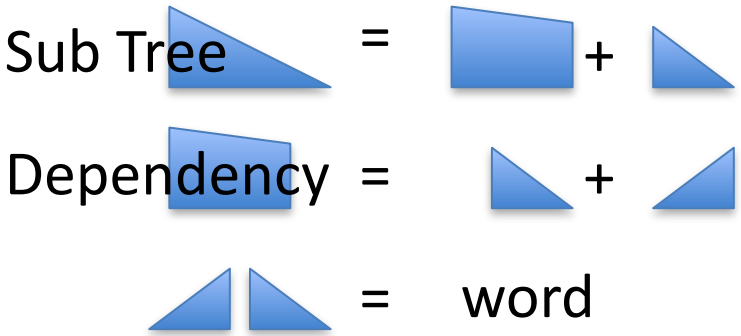
The Eisner Algorithm

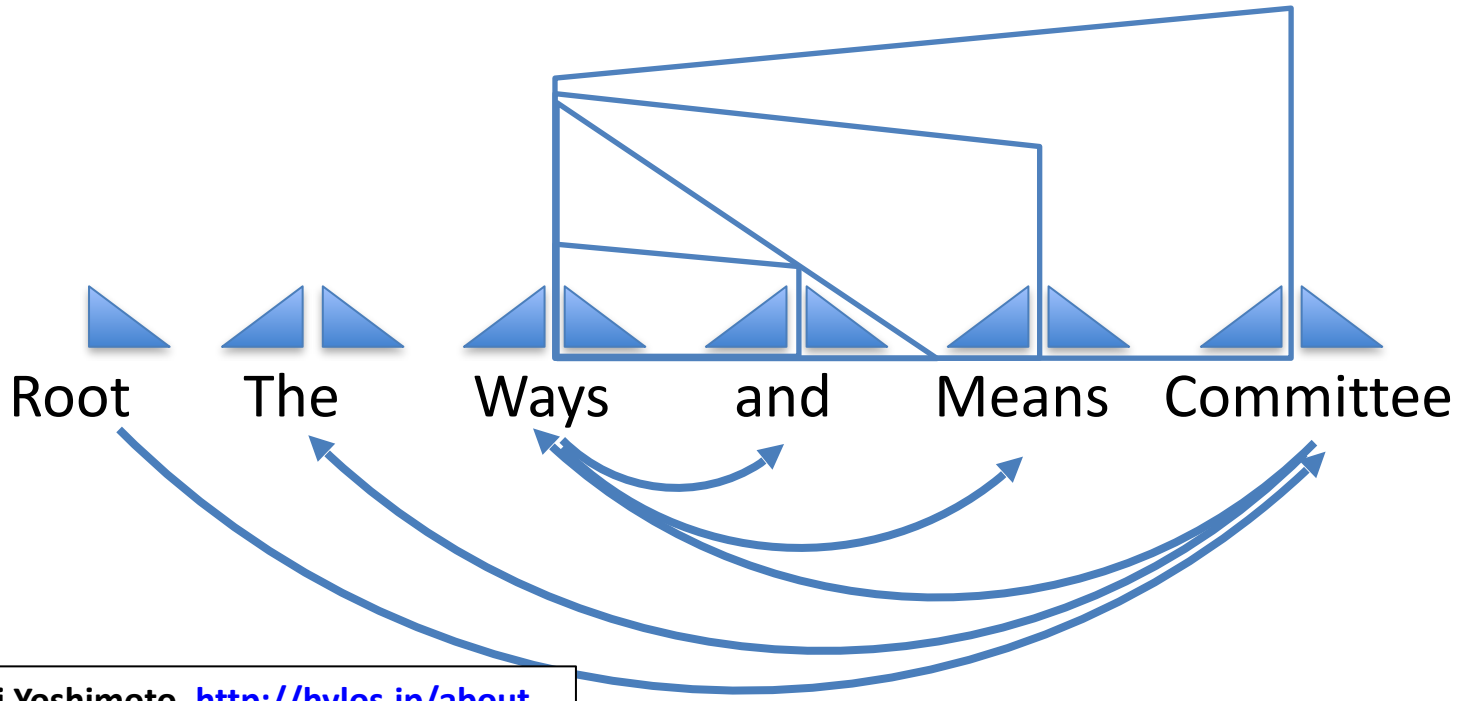




From: Akifumi Yoshimoto. http://hylos.jp/about

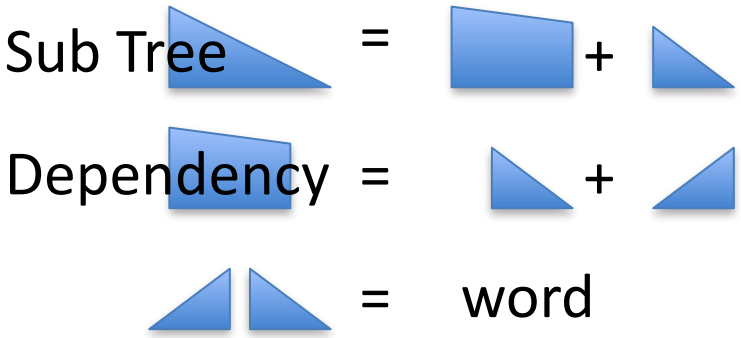
The Eisner Algorithm

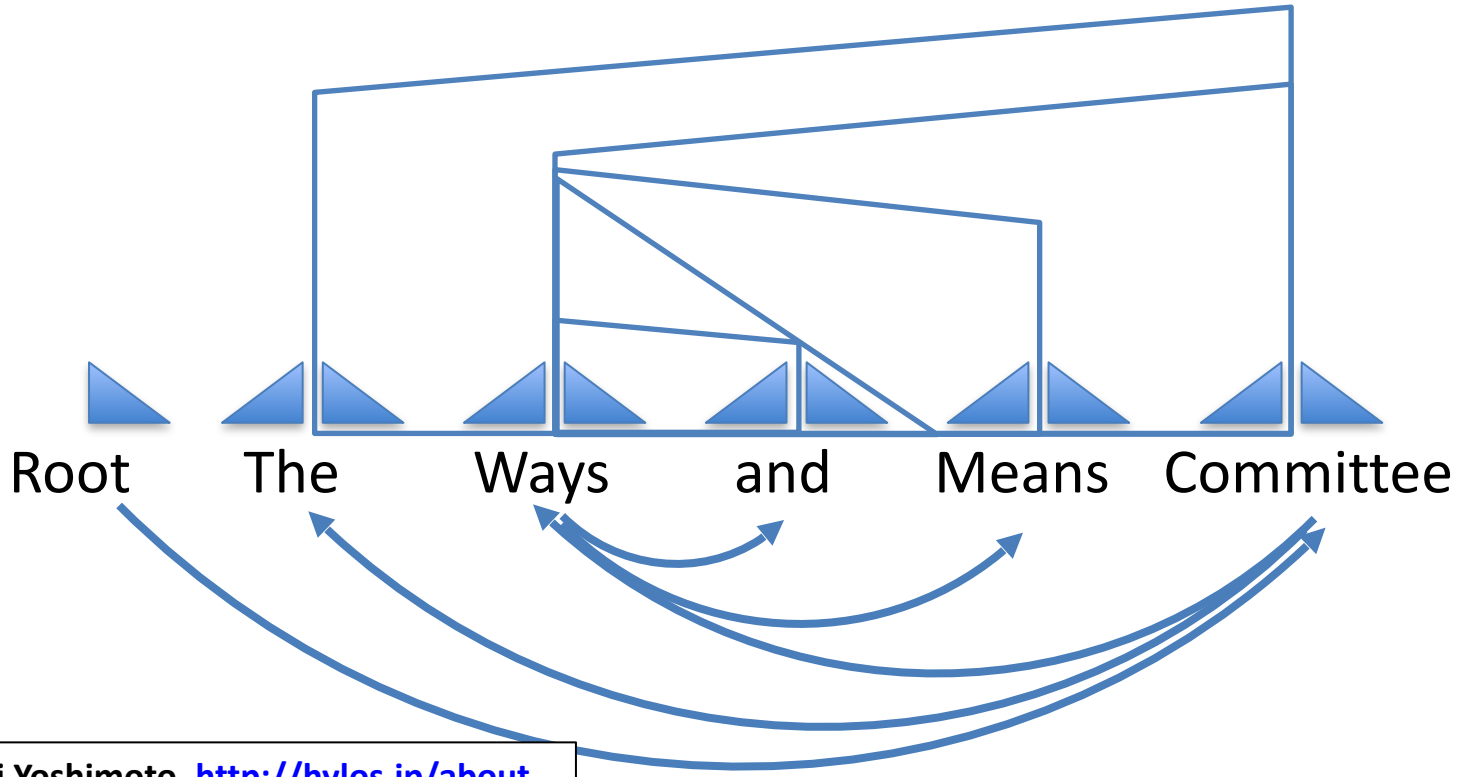




From: Akifumi Yoshimoto. http://hylos.jp/about

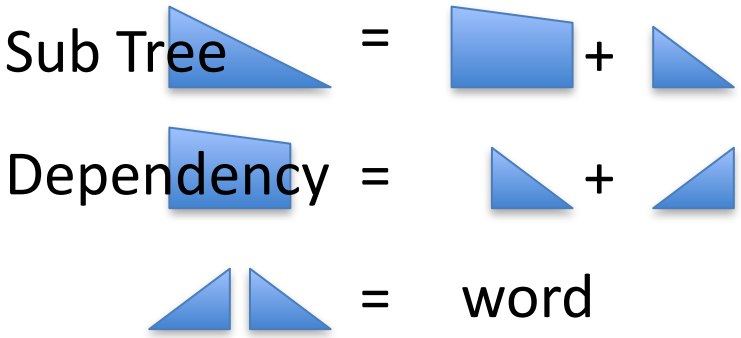
The Eisner Algorithm

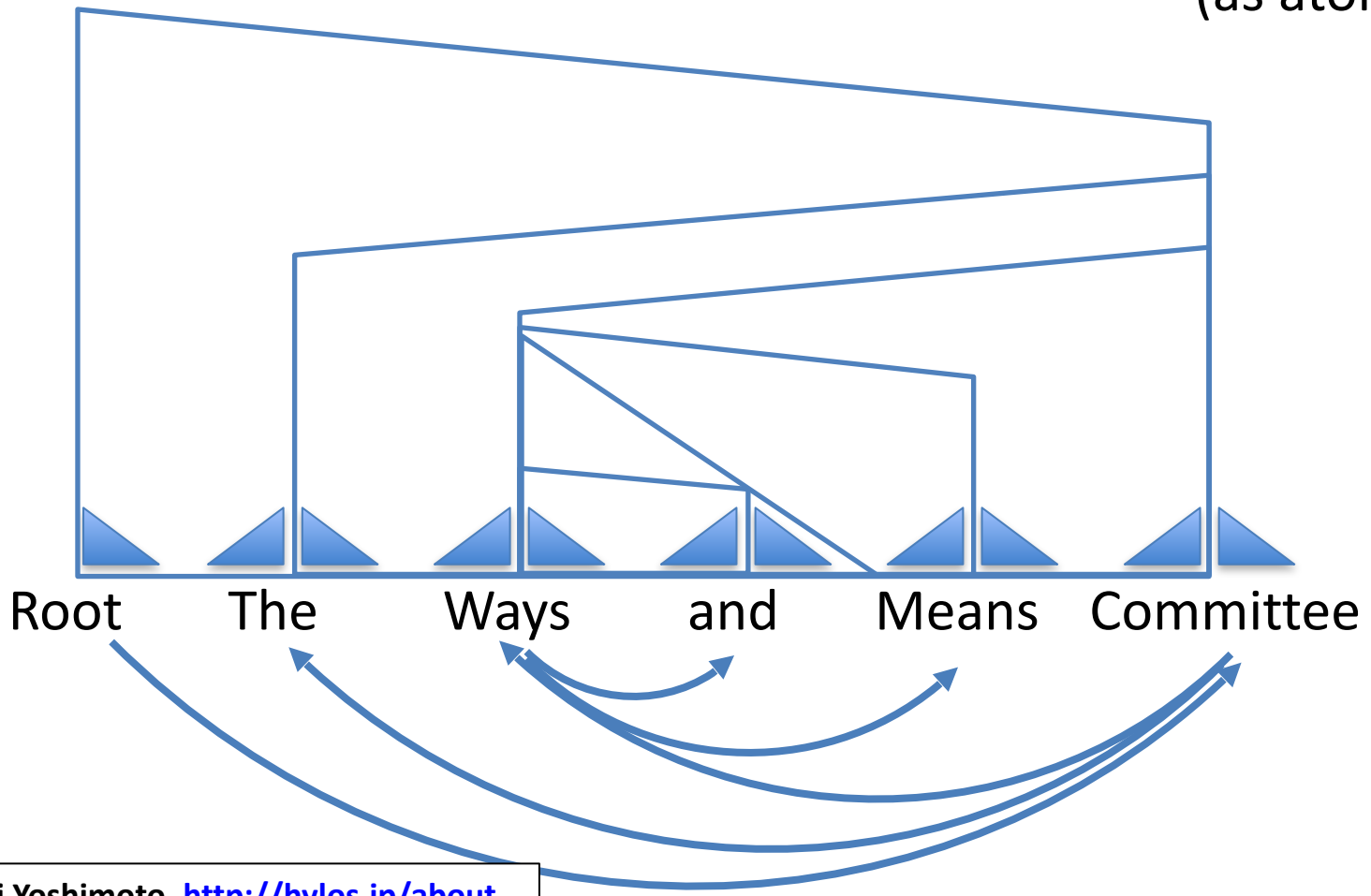




From: Akifumi Yoshimoto. http://hylos.jp/about

The Eisner Algorithm

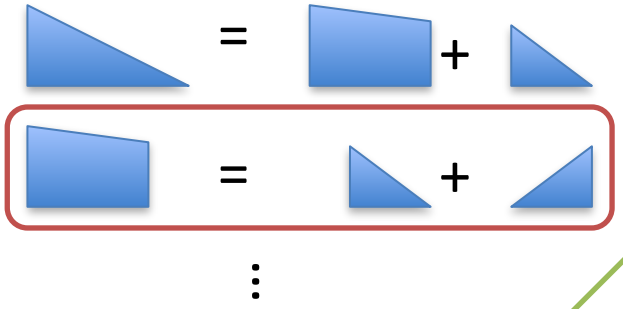




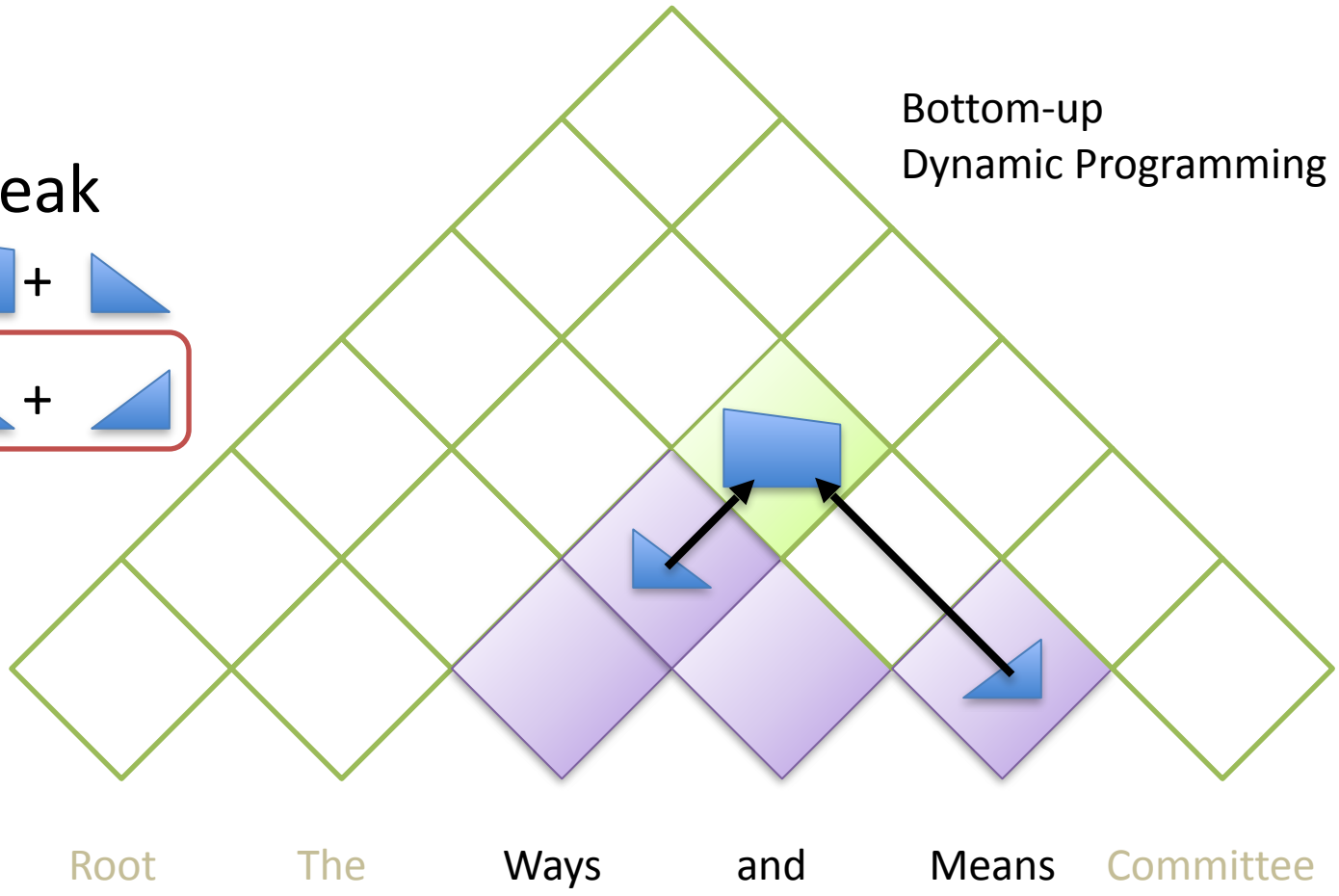
From: Akifumi Yoshimoto. http://hylos.jp/about

Eisner algorithm extends CKY

```
CKY Table: C[start][end][rule]
for k: 1..n
   for s: 1..n
      t = s+k
      if t>n then break
```



```
   end for
end for
```



From: Akifumi Yoshimoto. http://hylos.jp/about

Overview

- Constituency Parsing
  - Parsing as Search; Dynamic Programming
  - CKY Algorithm
- Treebanks and Evaluation
- Dependency Parsing
  - Definitions
  - Eisner Algorithm

Recurring Theme!