Computational Linguistics

6. Formal Grammars and Syntactic Parsing

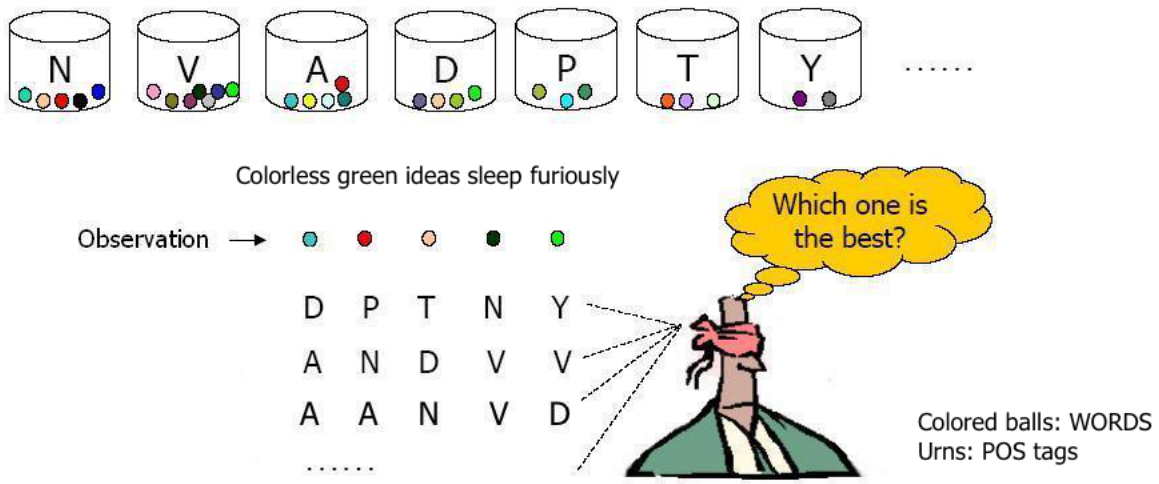
Xiaojing Bai

Tsinghua University

https://bxjthu.github.io/CompLing

Recap: HMM POS tagging as a decoding task

Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1 o_2 \cdots o_T$, find the most probable sequence of states $Q = q_1 q_2 q_3 \cdots q_T$.



Hidden Markov models are characterized by three fundamental problems:

- Problem 1: Likelihood
- Problem 2: Decoding
- Problem 3: Learning

Recap: The basic equation of HMM tagging

The most probable tag sequence given the observation sequence of n words w_1^n:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n)$$

$\hat{t}_1^n$ means 'the estimate of the sequence of n tags'

$\underset{x}{\operatorname{argmax}} P(x)$ means 'the x such that P(x) is maximized'

Recap: The basic equation of HMM tagging

The most probable tag sequence given the observation sequence of n words w_1^n:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(w_1^n|t_1^n)P(t_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

Two simplifying assumptions and the estimates

$$P(w_i|t_i) = \frac{\text{Frequency of } w_i \text{ tagged as } t_i \text{ in the training corpus}}{\text{Frequency of } t_i \text{ in the training corpus}}$$

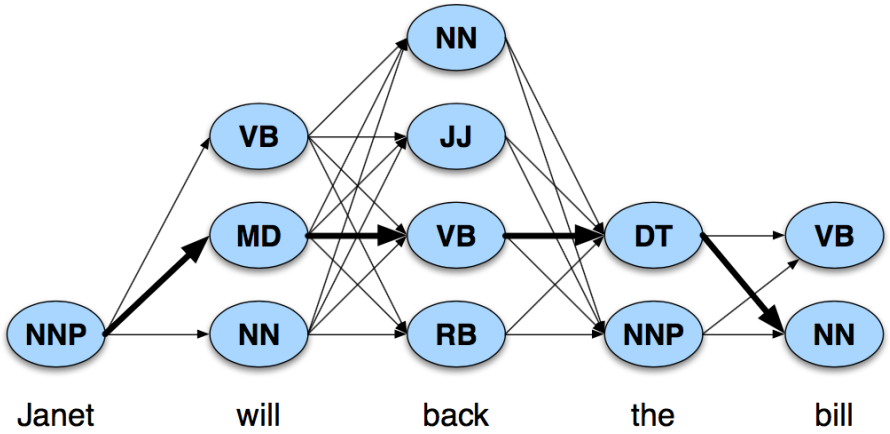
$$P(t_i|t_{i-1}) = \frac{\text{Frequency of } t_i \text{ after } t_{i-1} \text{ in the training corpus}}{\text{Frequency of } t_{i-1} \text{ in the training corpus}}$$

At the end of this session you will

- understand the rationale behind dynamic programming;
- know how the Viterbi algorithm works for POS tagging;
- know how to describe a language using regular and context-free grammars;
- know how top-down parsing and bottom-up parsing work;
- understand the strengths and weaknesses of these two parsing algorithms;
- understand how language models and parsing algorithms work together;
- learn to analyze sentence structure with NLTK.

POS tagging: an example

E.g. Janet will back the bill



Janet/NNP will/MD back/VB the/DT bill/NN

The HMM defined by two tables

- Transition probabilities between pos tags
- Likelihoods of words given tags

	NNP	MD	VB	JJ	NN	RB	DT
<s>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

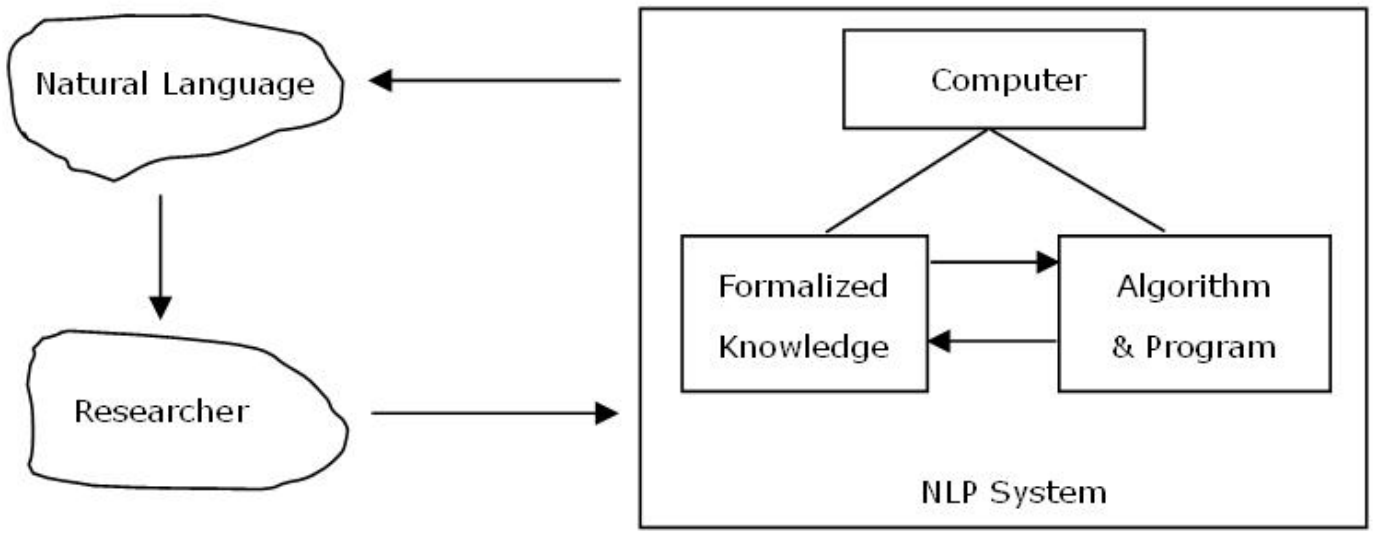
Figure 10.5 The A transition probabilities $P(t_i|t_{i-1})$ computed from the WSJ corpus without smoothing. Rows are labeled with the conditioning event; thus $P(VB|MD)$ is 0.7968.

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0.000097	0
NN	0	0.000200	0.000223	0.000006	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Figure 10.6 Observation likelihoods B computed from the WSJ corpus without smoothing.

Models and algorithms

How NLP systems work?



- **A language model:** a formal description of linguistic knowledge to show how a language works

- **An algorithm:** a series of steps in a computer program to solve a problem in an efficient way

Viterbi algorithm

- Andrew Viterbi, 1967
- A dynamic programming algorithm
- Maximize the probabilities
- Find the most likely sequence of tags
- Algorithm: exponential complexity

  With a tagset of N tags, for a sequence of M words,

  there are, in the worse case, N^M possible paths.
- Efficiency: N^M vs. $N^2 \times M$

Rationale behind dynamic programming

Aka: dynamic optimization

- Solve a complex problem by breaking it down into a collection of simpler subproblems
- Solve each of those subproblems just once and store their solutions
- Look up the previously computed solution the next time the same subproblem occurs
- Thereby save computation time at the expense of a hopefully modest expenditure in storage space

Memoization: storing solutions to subproblems instead of recomputing them

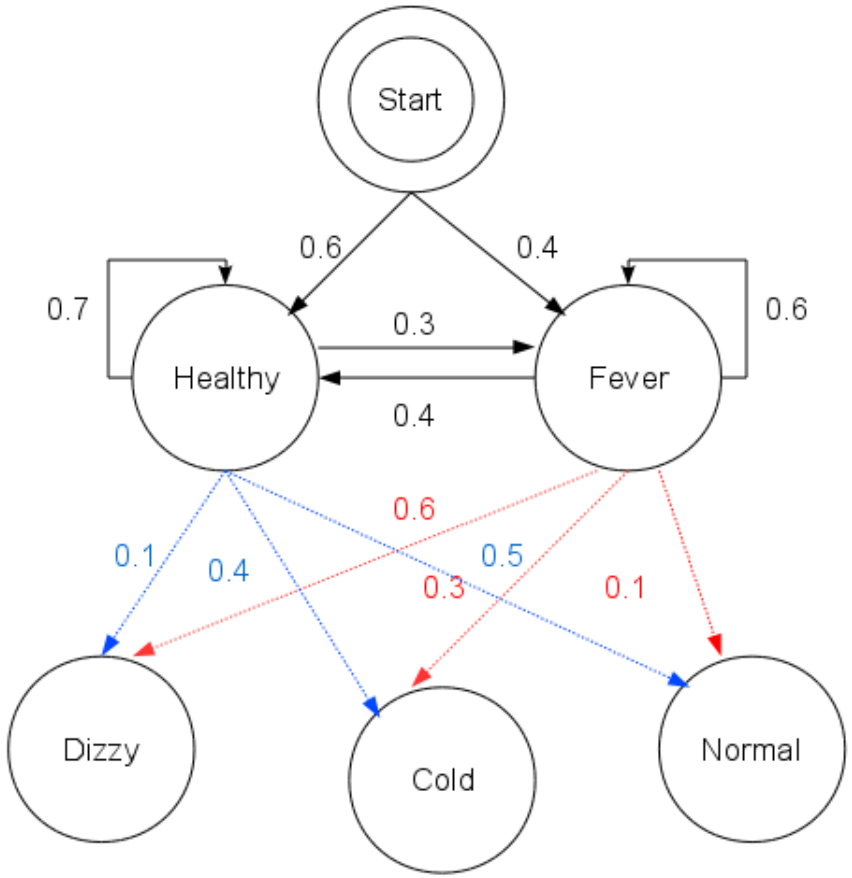
Viterbi algorithm: an example

Consider a village where all villagers are either healthy or have a fever and only the village doctor can determine whether each has a fever. The doctor diagnoses fever by asking patients how they feel. The villagers may only answer that they feel normal, dizzy, or cold.

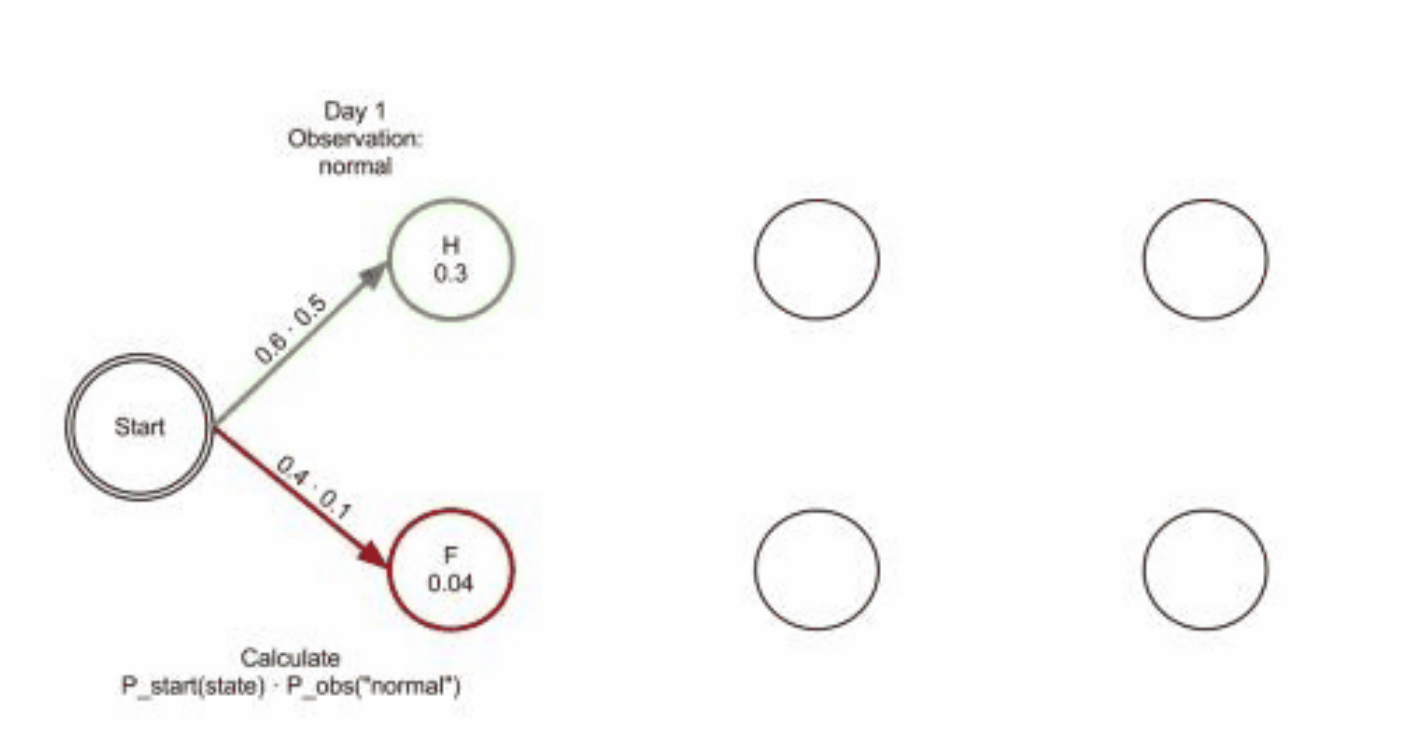
Python representation of the given HMM:

```
obs = ('normal', 'cold', 'dizzy')
states = ('Healthy', 'Fever')
start_p = {'Healthy': 0.6, 'Fever': 0.4}
trans_p = {
   'Healthy' : {'Healthy': 0.7, 'Fever': 0.3},
   'Fever' : {'Healthy': 0.4, 'Fever': 0.6}
   }
emit_p = {
   'Healthy' : {'normal': 0.5, 'cold': 0.4, 'dizzy': 0.1},
   'Fever' : {'normal': 0.1, 'cold': 0.3, 'dizzy': 0.6}
   }
```

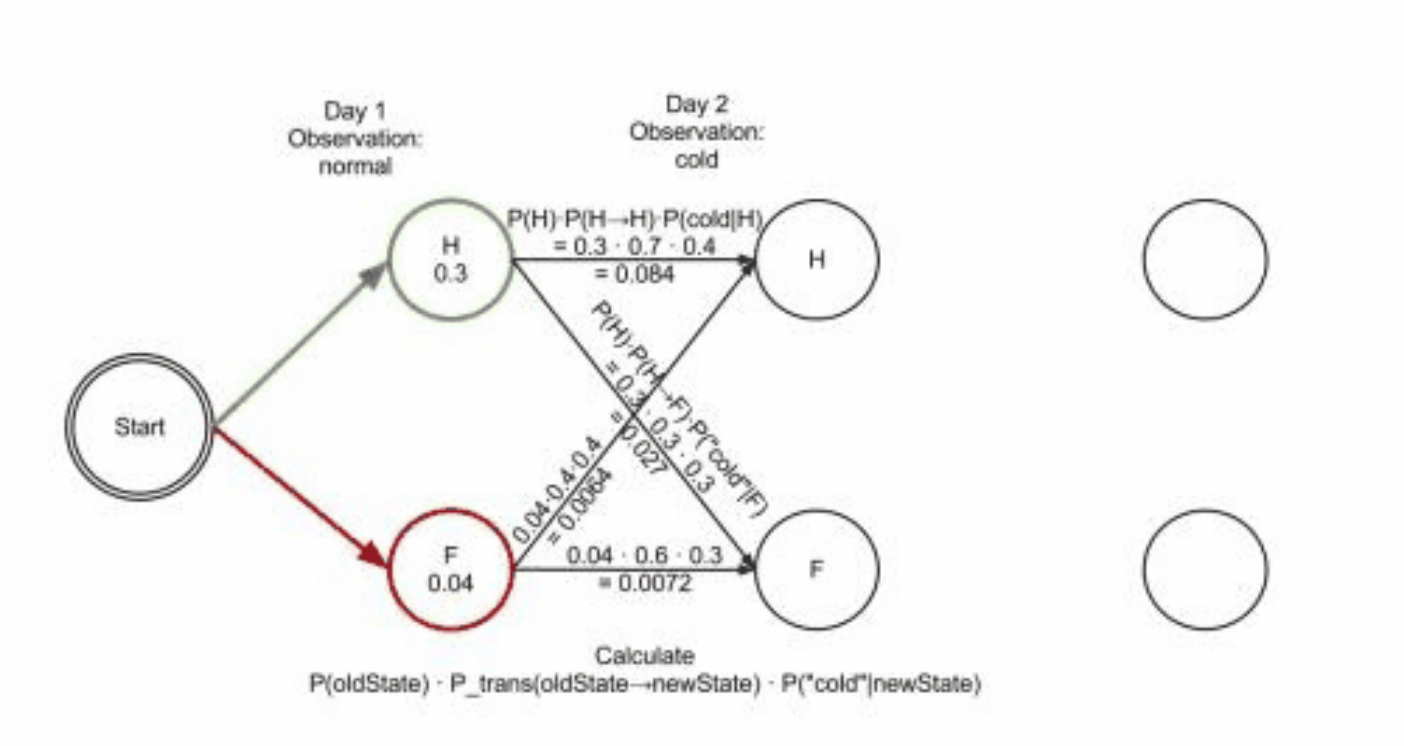
Graphical representation of the given HMM



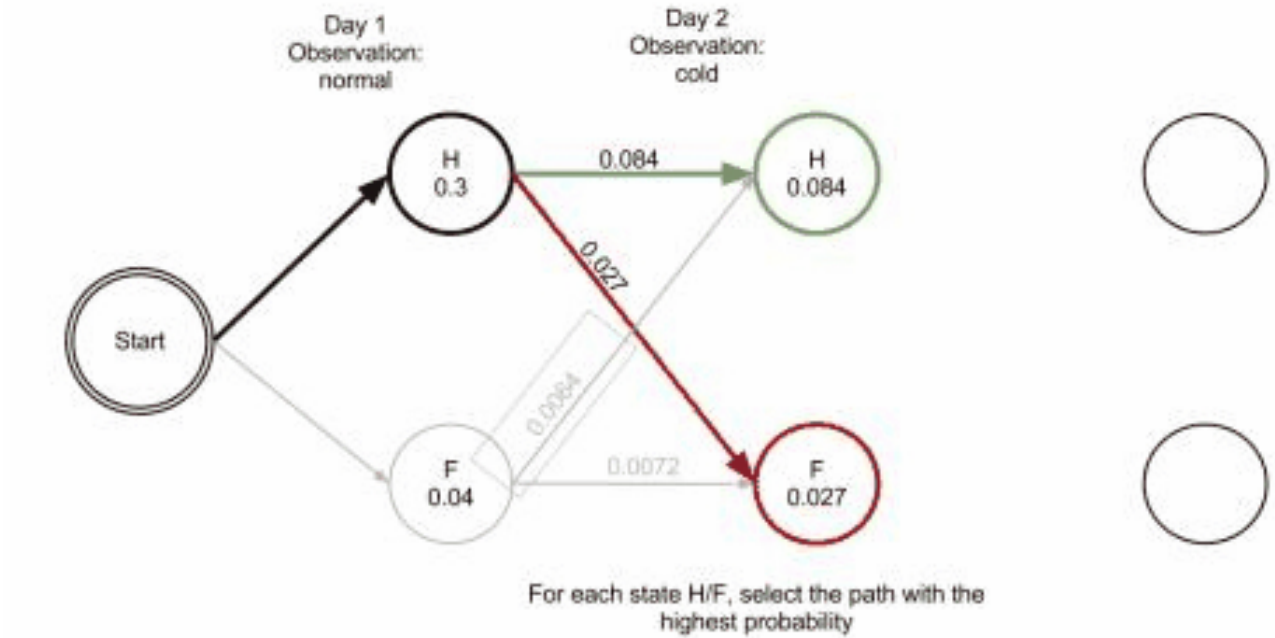
Viterbi algorithm: an example



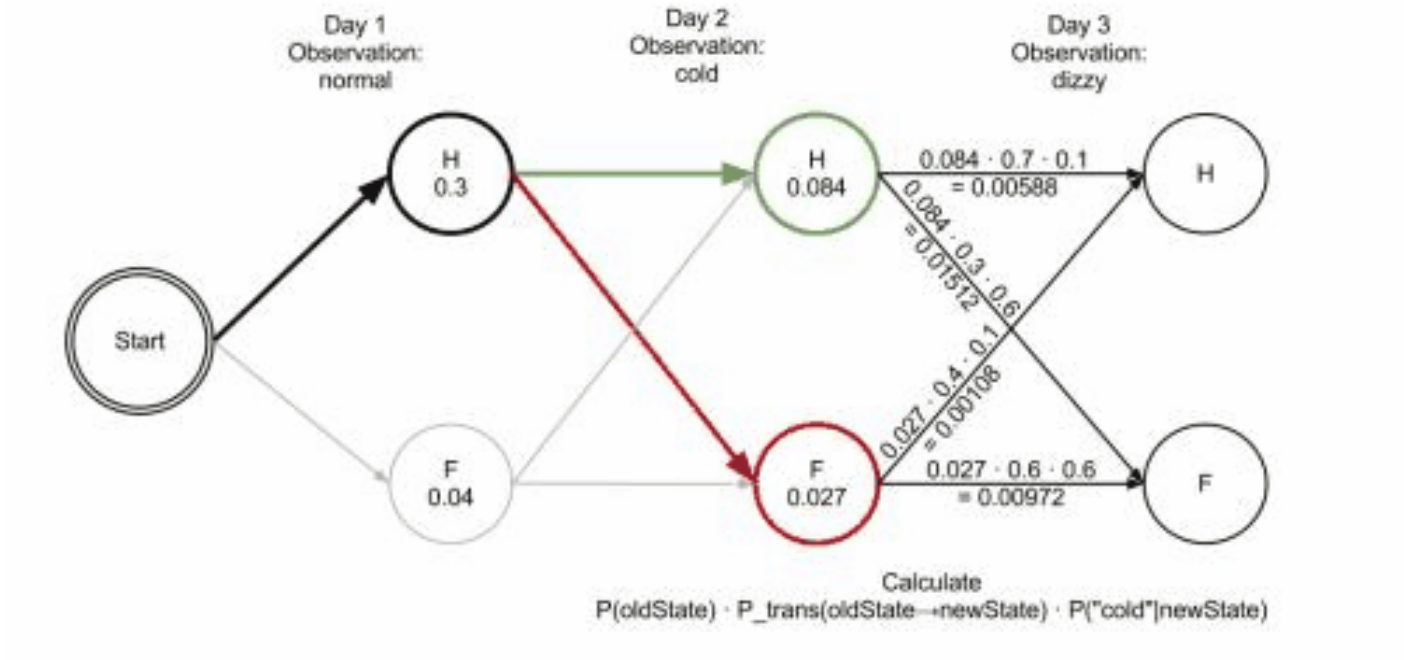
Viterbi algorithm: an example



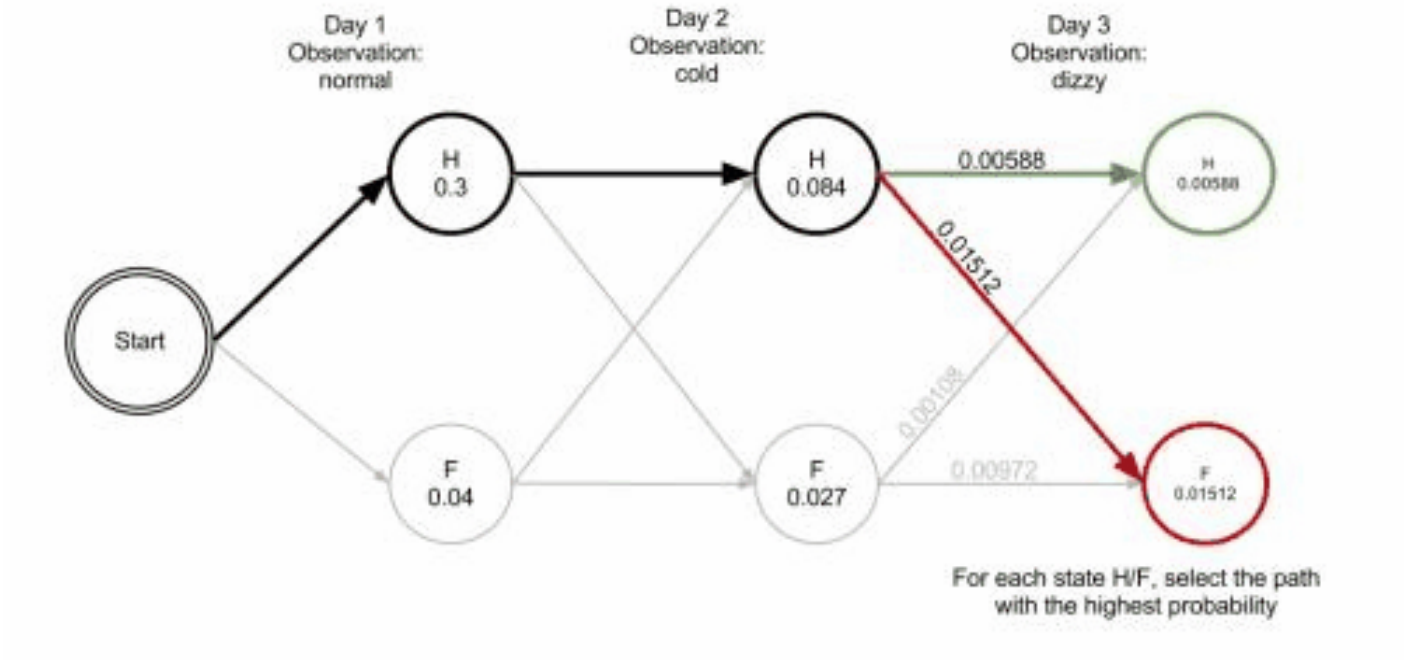
Viterbi algorithm: an example

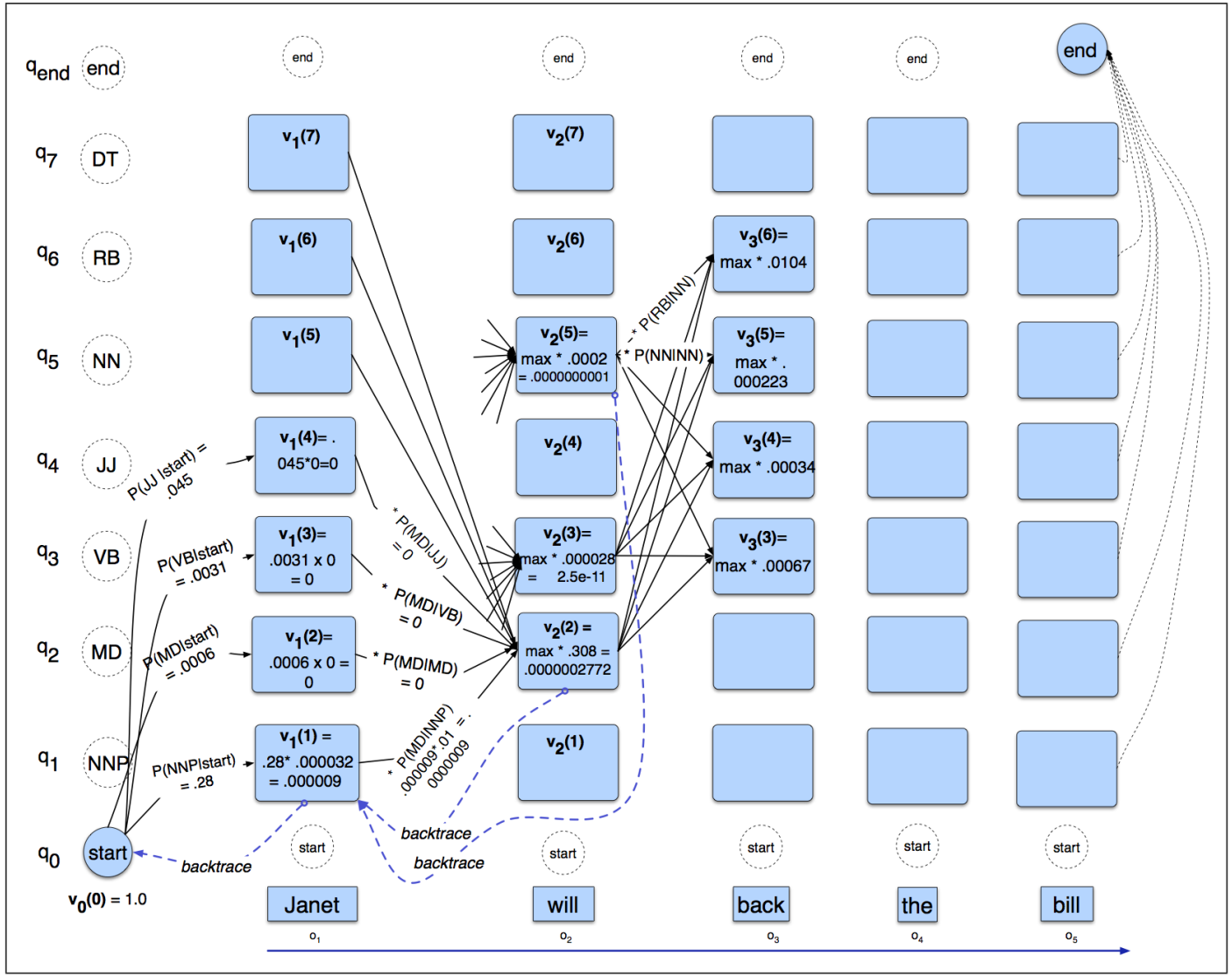


Viterbi algorithm: an example



Viterbi algorithm: an example





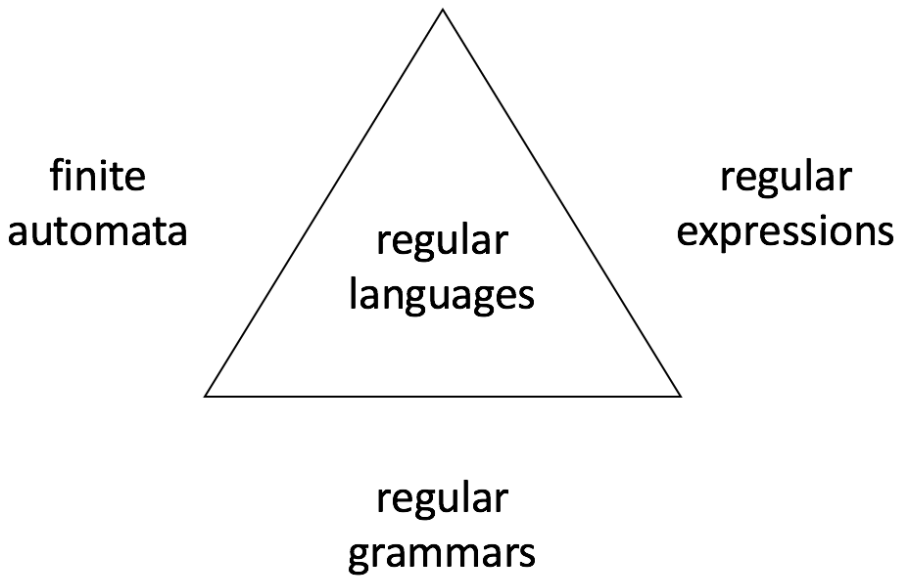
- The first few entries in the individual state columns for the Viterbi algorithm.
- Each cell keeps the probability of the best path so far and a pointer to the previous cell along that path.

Syntax

- The way words are arranged together
- Syntactic notions
  - Regular expressions: representing the sequences of words
  - N-grams: computing probabilities for the sequences of words
  - Parts-of-speech: acting as a kind of equivalence class for words
  - Models of syntax and grammar
    - Regular grammars
    - Context-free grammars
    - ...

Representing an FSA as a grammar

Three equivalent ways of describing regular languages



- The Chomsky hierarchy
- Natural language and its complexity
- Formal models and formal languages
- Power of formal models: complexity of the phenomena they can describe

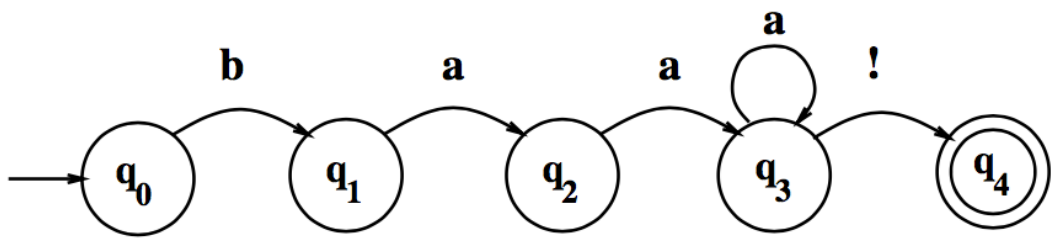
A formal grammar for the sheep talk

starting symbol = S
non-terminals = {S, A, B, C}
terminals = {b, a, $!$}

Rewrite rules or production rules

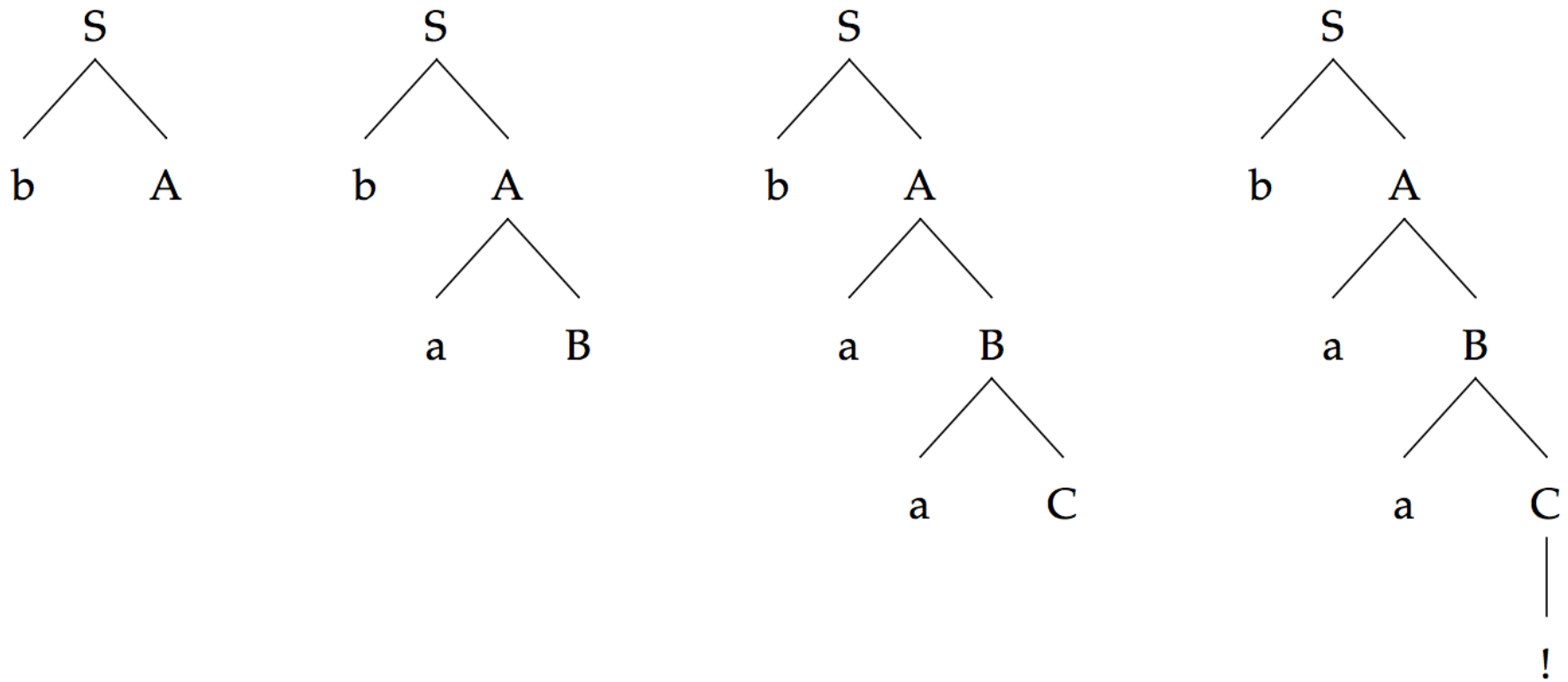
$S \rightarrow bA$
$A \rightarrow aB$
$B \rightarrow aC$
$C \rightarrow aC$
$C \rightarrow !$

/baa+!/



$$Q = \{q_0, q_1, q_2, q_3, q_4\}$$
$$\Sigma = \{b, a, !\}$$
$$q_0 = q_0$$
$$F = \{q_4\}$$

Derivation of the sheep talk



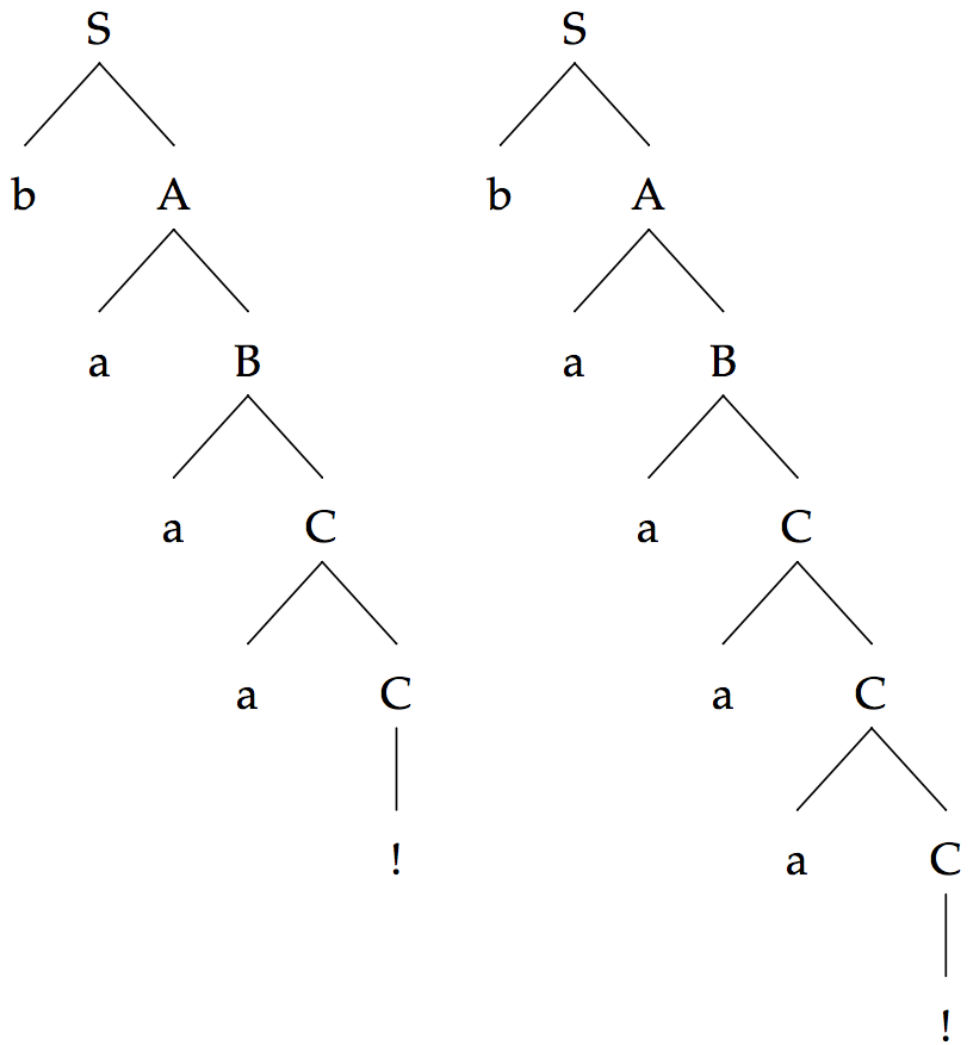
$S \to bA$ $A \to aB$ $B \to aC$ $C \to !$

Derivation of the sheep talk

starting symbol = S
non-terminals = {S, A, B, C}
terminals = {b, a, $!$}

Rewrite rules or production rules

$S \rightarrow bA$
$A \rightarrow aB$
$B \rightarrow aC$
$C \rightarrow aC$
$C \rightarrow !$

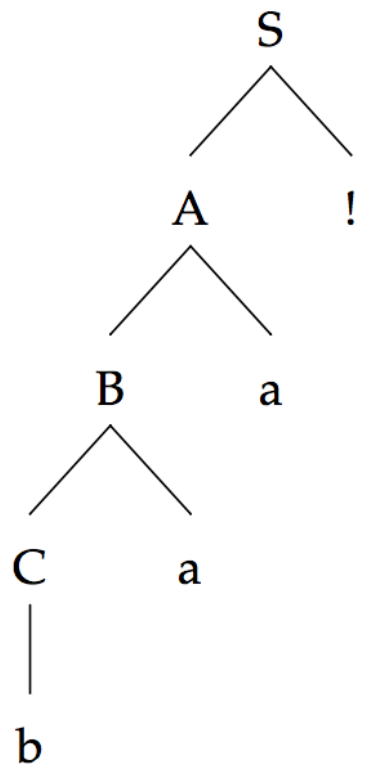


Rewrite rules of a regular grammar

Left-branching structures

$X \rightarrow Ya$
$X \rightarrow a$

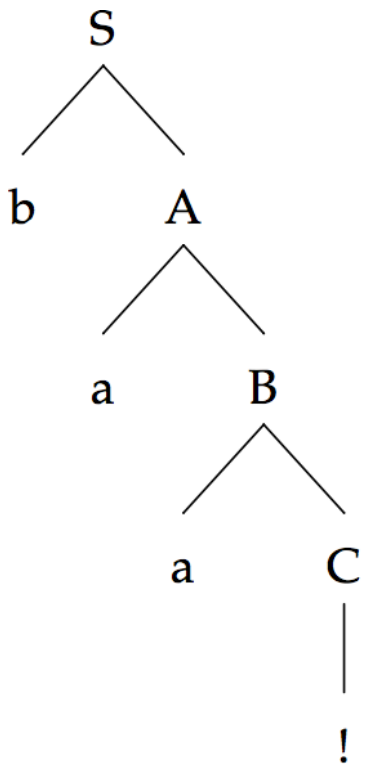
$S \rightarrow A!$
$A \rightarrow Ba$
$B \rightarrow Ca$
$C \rightarrow Ca$
$C \rightarrow b$



Right-branching structures

$X \rightarrow aY$
$X \rightarrow a$

$S \rightarrow bA$
$A \rightarrow aB$
$B \rightarrow aC$
$C \rightarrow aC$
$C \rightarrow !$



Rewrite rules of a regular grammar

Strong equivalence vs. **weak equivalence**

- Two grammars are **strongly equivalent** if they generate the same set of strings and if they assign the same phrase structure to each sentence (allowing merely for renaming of the non-terminal symbols).
- Two grammars are **weakly equivalent** if they generate the same set of strings but do not assign the same phrase structure to each sentence.

Regular grammars for natural language: problems

- Redundancy
- Centre Embedding

  E.g.

  The students the police arrested complained.

  The luggage that the passengers checked arrived.

  The luggage that the passengers that the storm delayed checked arrived.

Context-free grammars for natural language

- Less restrictive and hence more powerful
- Aka: phrase-structure grammars
- Equivalent to Backus-Naur Form (BNF)
- Backbone of many formal models of the syntax of natural language
- Applications
  - syntactic parsing, semantic interpretation etc.
  - grammar checking, semantic interpretation, dialogue understanding, machine translation etc.
- Computationally tractable

Context-free grammars: the formal definition

S: a designated start symbol;

Σ: a set of terminal symbols;

N: a set of non-terminal symbols;

R: a set of rewrite rules of the form $A \rightarrow \beta$
where A is a non-terminal
and β is a string of elements from the infinite set $(\Sigma \cup N)^*$.

Or most commonly written as **Chomsky Normal Form** (CNF):

$A \rightarrow BC$, or, $A \rightarrow a$, where, A, B, $C \in N$, and, $a \in \Sigma$.

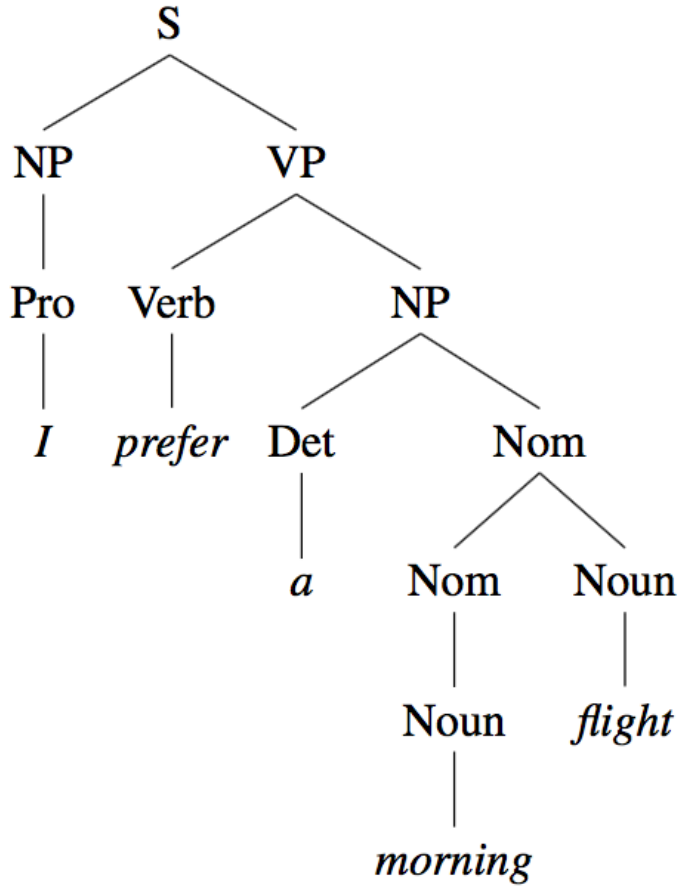
Context-free grammars: an example

E.g. I prefer a morning flight

Bracketed notation:

[S [NP [Pro I]] [VP [V prefer] [NP [Det a] [Nom [N morning] [Nom [N flight]]]]]]

Parse tree:



Context-free grammars: an example

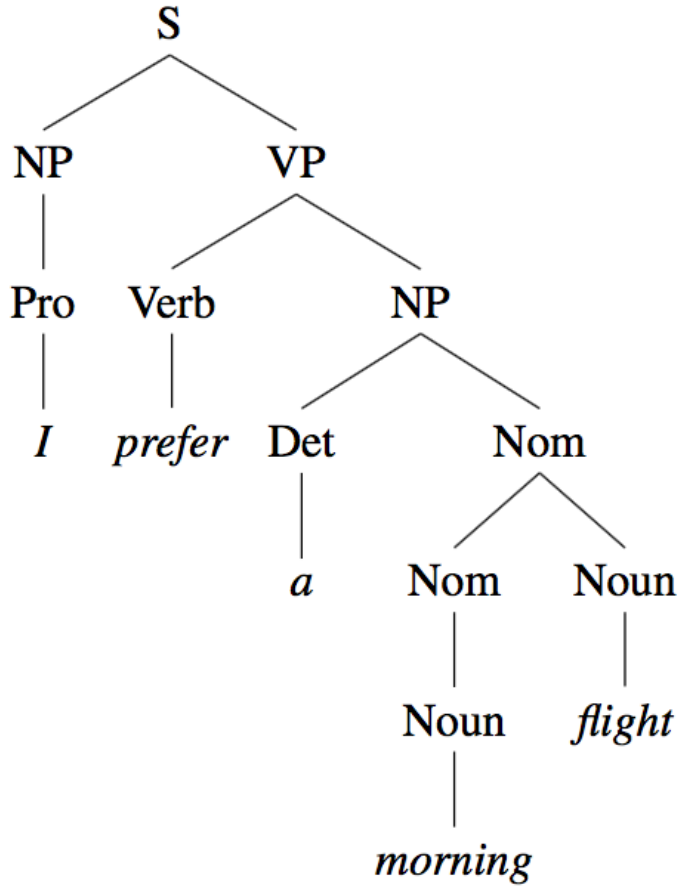
Noun	→	*flights* | *breeze* | *trip* | *morning*
Verb	→	*is* | *prefer* | *like* | *need* | *want* | *fly*
Adjective	→	*cheapest* | *non-stop* | *first* | *latest* | *other* | *direct*
Pronoun	→	*me* | *I* | *you* | *it*
Proper-Noun	→	*Alaska* | *Baltimore* | *Los Angeles* | *Chicago* | *United* | *American*
Determiner	→	*the* | *a* | *an* | *this* | *these* | *that*
Preposition	→	*from* | *to* | *on* | *near*
Conjunction	→	*and* | *or* | *but*

Figure 11.2 The lexicon for $\mathscr{L}_0$.

Grammar Rules			Examples
S	→	*NP VP*	I + want a morning flight
NP	→	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	→	*Nominal Noun*	morning + flight
	|	*Noun*	flights
VP	→	*Verb*	do
	|	*Verb NP*	want + a flight
	|	*Verb NP PP*	leave + Boston + in the morning
	|	*Verb PP*	leaving + on Thursday
PP	→	*Preposition NP*	from + Los Angeles

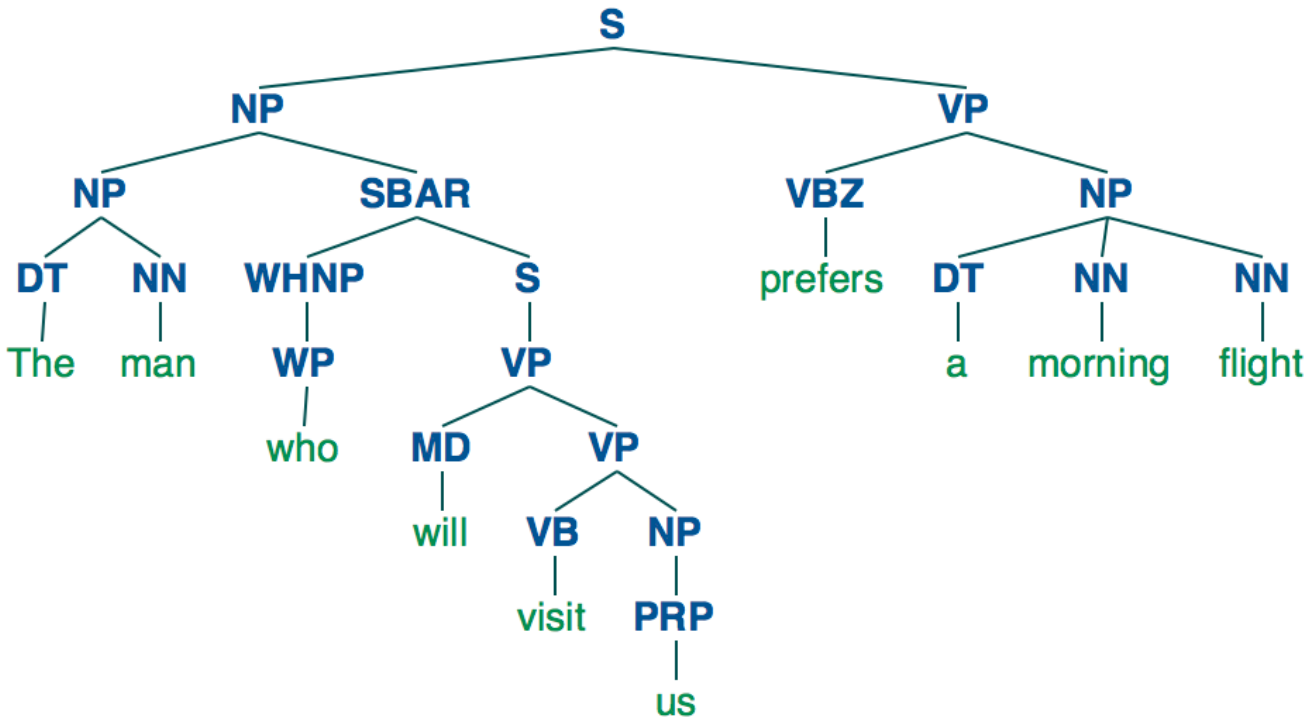
Figure 11.3 The grammar for $\mathscr{L}_0$, with example phrases for each rule.

Parse tree:



Context-free grammars: another example

E.g. The man who will visit us prefers a morning flight



What are formal grammars used for?

- Generating sentences
- Recognizing grammatical and ungrammatical sentences
- Parsing sentences

Syntactic parsing with context-free grammars

- **Parsing**: recognizing an input string and assigning a structure to the string
- **Syntactic parsing**: recognizing a sentence and assigning a syntactic structure to the sentence
  - **A context-free grammar**: a declarative formalism of how words combine to form larger linguistic units (phrases and sentences)
  - **A parsing algorithm**: employing the grammar to produce parse trees for a given sentence

Parsing as a search problem

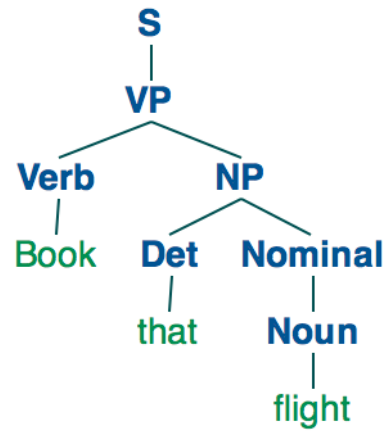
Goal: to find all the trees whose root is the start symbol S and which cover exactly the words in the input (leaves)

E.g. *Book that flight*

Constraint 1: Whatever else is true of the final parse tree, its leaves must be three words from the input: *book*, *that*, and *flight*.

Constraint 2: Whatever else is true of the final parse tree, it must have one root: the start symbol *S*.

Grammar	Lexicon
$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid NWA$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal\ Noun$	
$Nominal \rightarrow Nominal\ PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	
$VP \rightarrow Verb\ NP\ PP$	
$VP \rightarrow Verb\ PP$	
$VP \rightarrow VP\ PP$	
$PP \rightarrow Preposition\ NP$	



Top-down parsing

Start with the root and keep employing rules whenever possible to expand the tree .

With a context-free grammar, the top-down algorithm inspects the left-hand side of the rules to look for matches.

a

Bottom-up parsing

Start with the leaves and keep employing rules whenever possible to combine the tree pieces.

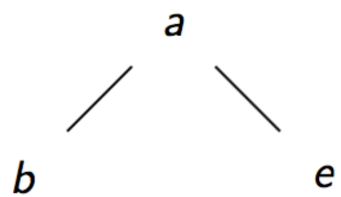
With a context-free grammar, the bottom-up algorithm inspects the right-hand side of the rules to look for matches.

c *d* *f*

Top-down parsing

Start with the root and keep employing rules whenever possible to expand the tree .

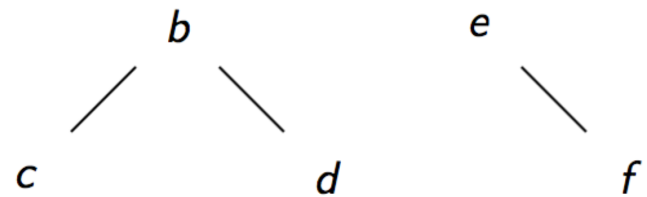
With a context-free grammar, the top-down algorithm inspects the left-hand side of the rules to look for matches.



Bottom-up parsing

Start with the leaves and keep employing rules whenever possible to combine the tree pieces.

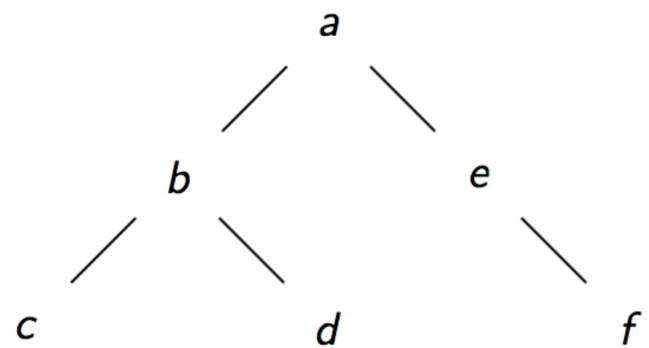
With a context-free grammar, the bottom-up algorithm inspects the right-hand side of the rules to look for matches.



Top-down parsing

Start with the root and keep employing rules whenever possible to expand the tree .

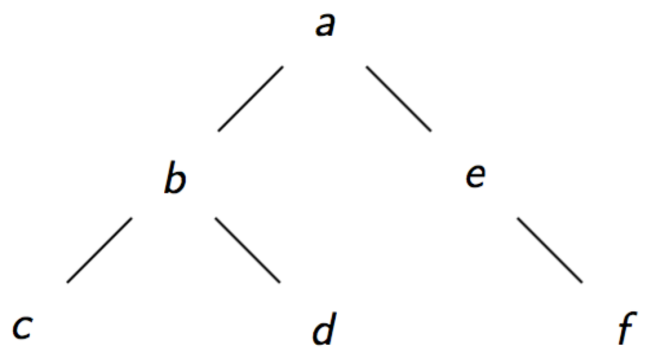
With a context-free grammar, the top-down algorithm inspects the left-hand side of the rules to look for matches.



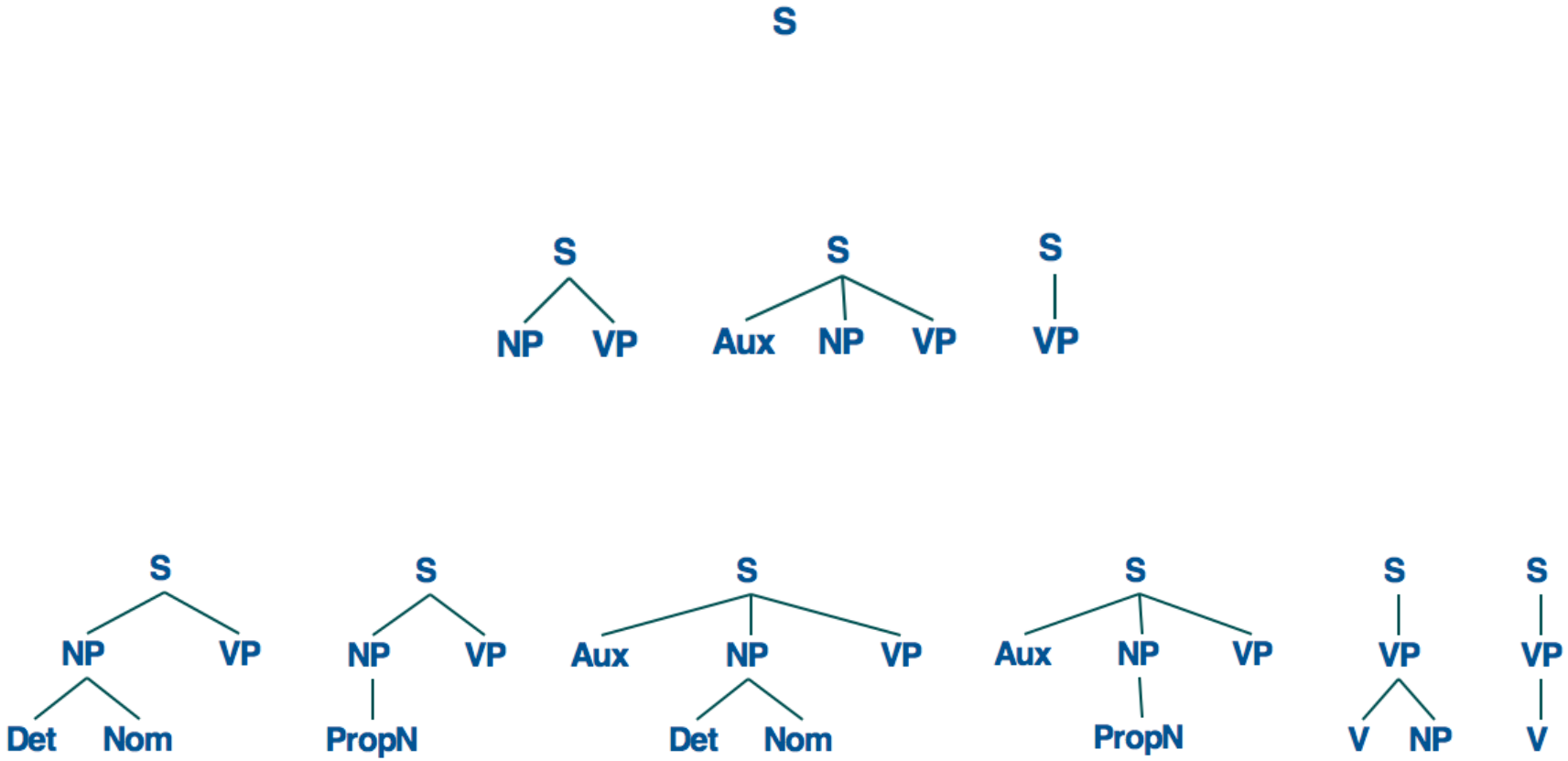
Bottom-up parsing

Start with the leaves and keep employing rules whenever possible to combine the tree pieces.

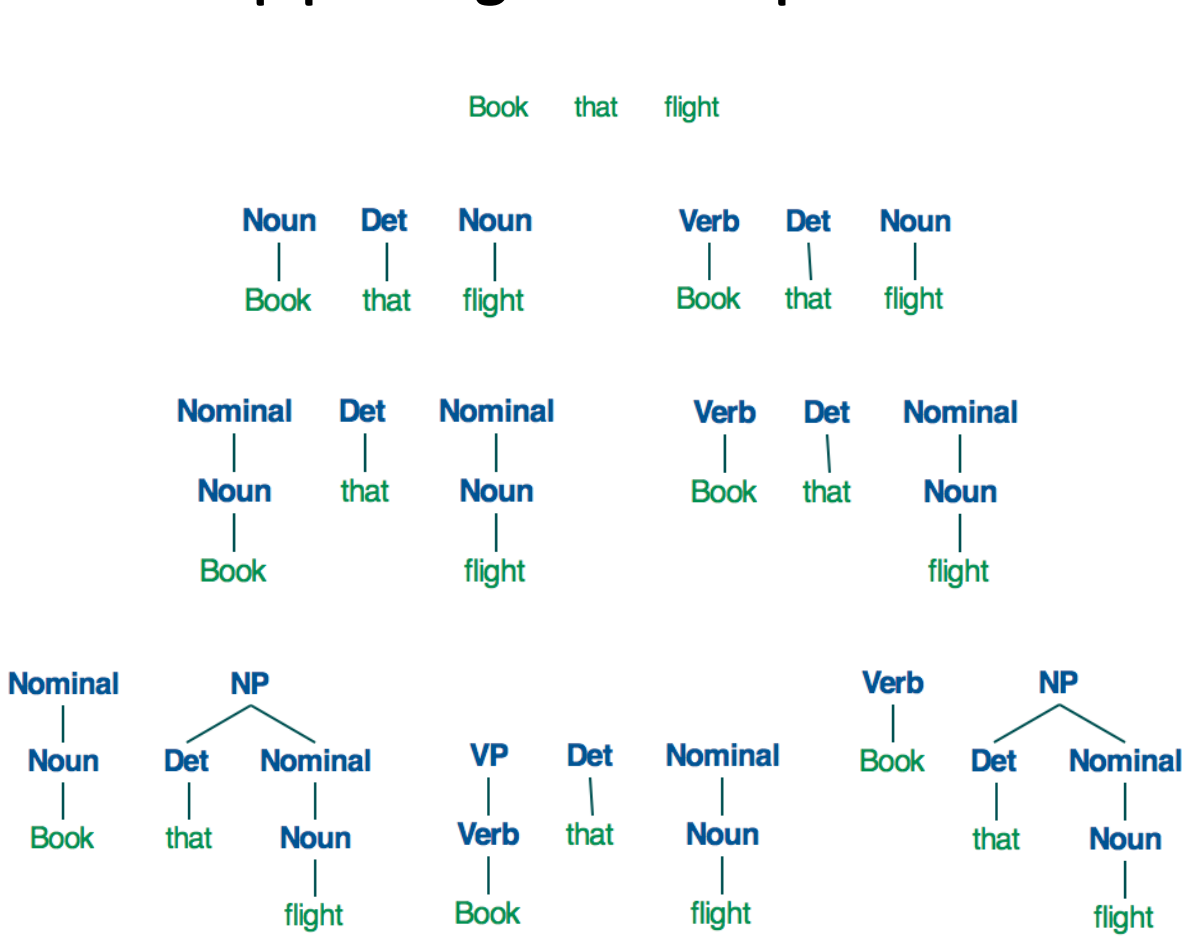
With a context-free grammar, the bottom-up algorithm inspects the right-hand side of the rules to look for matches.

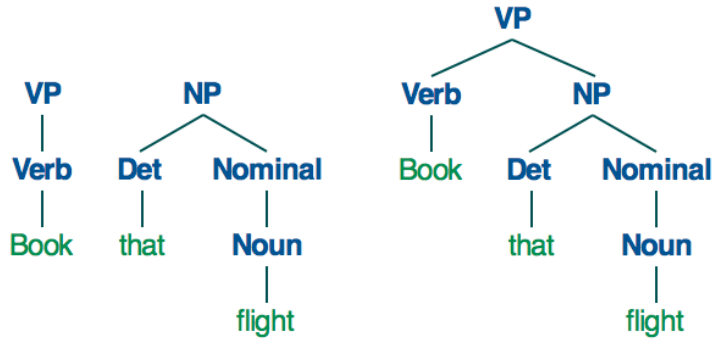


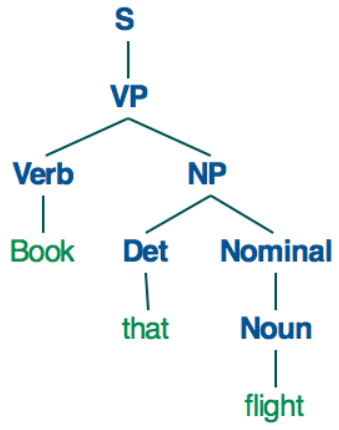
Top-down parsing: an example



Bottom-up parsing: an example







Strengths and weaknesses

The top-down algorithm

- Does not explore trees that cannot result in an S, nor subtrees that cannot find a place in some S-rooted trees.
- Does waste time on trees that are not consistent with the input.

The bottom-up algorithm

- Does not explore trees that are not consistent with the input.
- Does waste time on subtrees that cannot lead to an S or fit in with any of their neighbors.

Models and algorithms

- Language models: correct parsing
  - Lexical-functional grammar
  - Head-driven phrase structure grammar
  - Link grammar
  - Dependency grammar
  - Probabilistic context-free grammar …
- Parsing algorithms: efficient parsing
  - CYK parsing (Cocke-Younger-Kasami algorithm, Chomsky Normal Form)
  - Earley parsing
  - Chart parsing
  - Left–corner parsing
  - ATN parsing …

Practical 6: analyzing sentence structure with NLTK

http://www.nltk.org/book/ch08.html

- Define a simple grammar to parse a simple sentence
- Play with the recursive descent parser application
- Parse with CFG (RecursiveDescentParser, ShiftReduceParser)
- Write your own grammars in a text file
- Draw a parse tree
- Develop a grammar by using the Treebank corpus
- Use the Stanford Parser to parse sentences and draw trees

At the end of this session you will

- understand the rationale behind dynamic programming;
- know how the Viterbi algorithm works for POS tagging;
- know how to describe a language using regular and context-free grammars;
- know how top-down parsing and bottom-up parsing work;
- understand the strengths and weaknesses of these two parsing algorithms;
- understand how language models and parsing algorithms work together;
- learn to analyze sentence structure with NLTK.

Homework

- Read/Review (Quiz 5 on Nov. 7, 2018)
  - J+M 8 (8.1-8.4; 8.7)
  - J+M 10
- Read
  - J+M 11
- Practice
  - http://www.nltk.org/book/ch08.html
- Practical 5 assignment - submit your codes on GradeScope (DDL: Nov. 5)

Next session

Statistical Parsing and Dependency Parsing