CS 481

Artificial Intelligence Language Understanding

February 14, 2023

Announcements / Reminders

- **Please follow the Week 05 To Do List instructions**
- **PA #01 due on Monday (02/20/23) at 11:59 PM CST**

- **Exam dates:**
  - **Midterm:** **03/02/2023 during Thursday lecture time**
  - **Final:** **04/27/2023 during Thursday lecture time**

Plan for Today

- **Parts of Speech tagging – continued**
- **Context-Free Grammars**

Example

Given our synthetic corpus, what is the most like sequence of categories (tags) corresponding to a sentence:

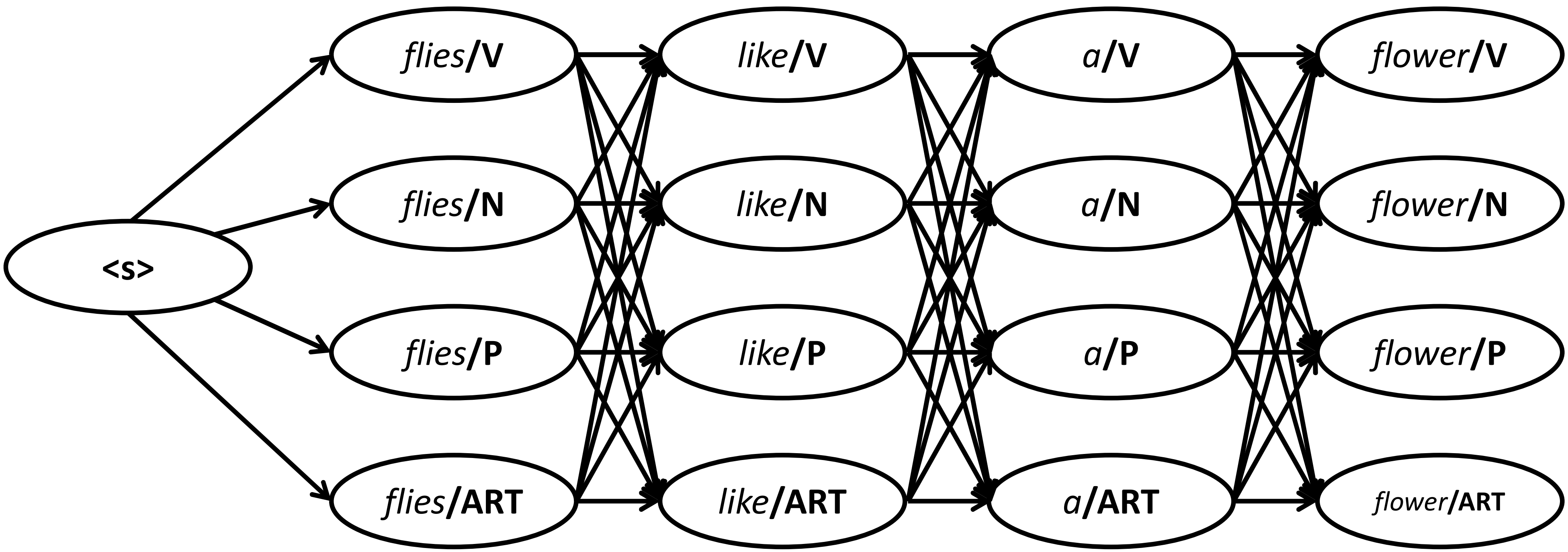
Flies like a flower

For example:

$$\boldsymbol{P(Flies, like, a, flower \mid N, V, ART, N) * P(N, V, ART, N)}$$

$$\cong \mathbf{5.4 * 10^{-5} * 0.081}$$

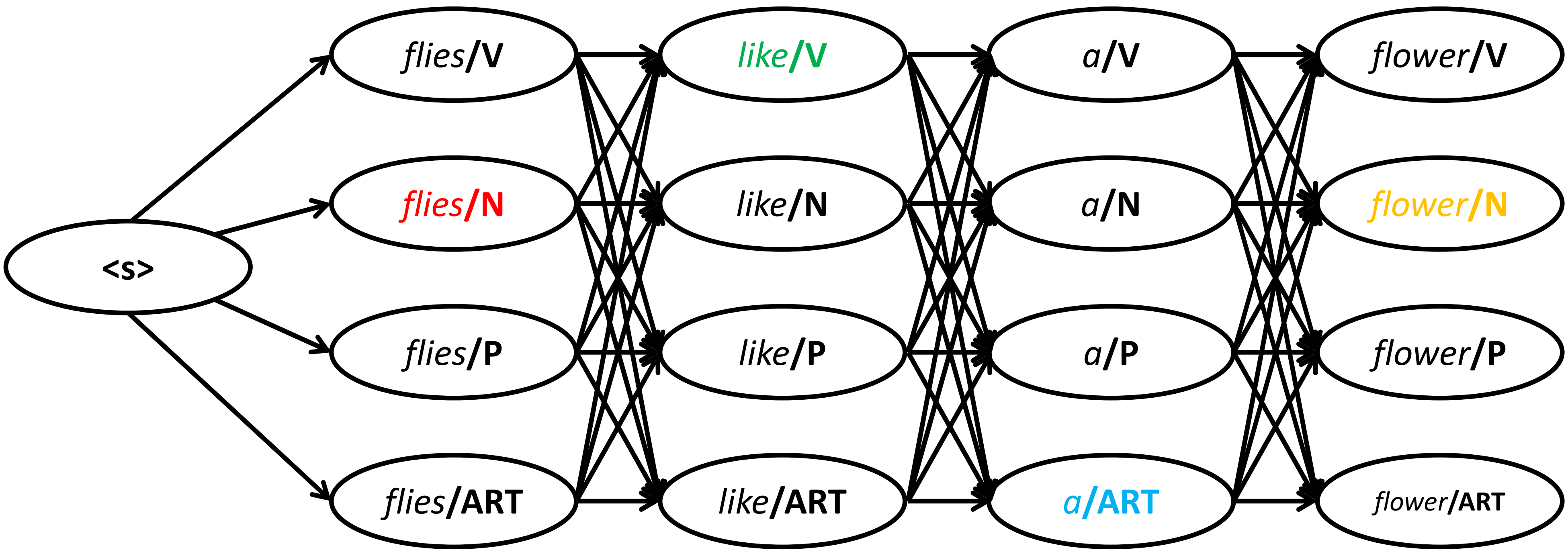
Example: All Possible Sequences



Every sequence can be assigned a probability:

$$P(w_1, w_2, w_3, \ldots, w_T \mid C_1, C_2, C_3, \ldots, C_T) \cong \prod_{i=1}^{T} P(w_i \mid C_i)$$

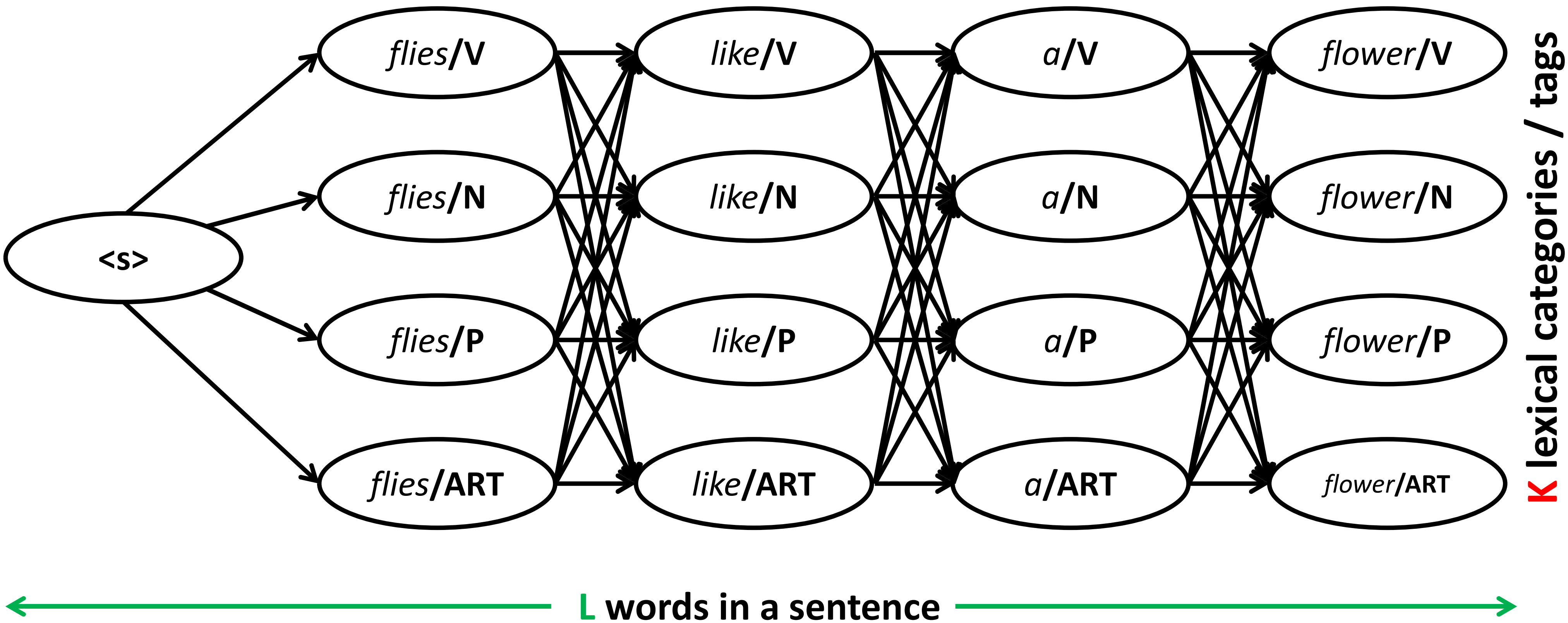
Example: Best Option



Best option will be:

$$\prod_{i=1}^{T} P(w_i \mid C_i) = P(flies|N) * P(like|V) * P(a|ART) * P(flower|N)$$

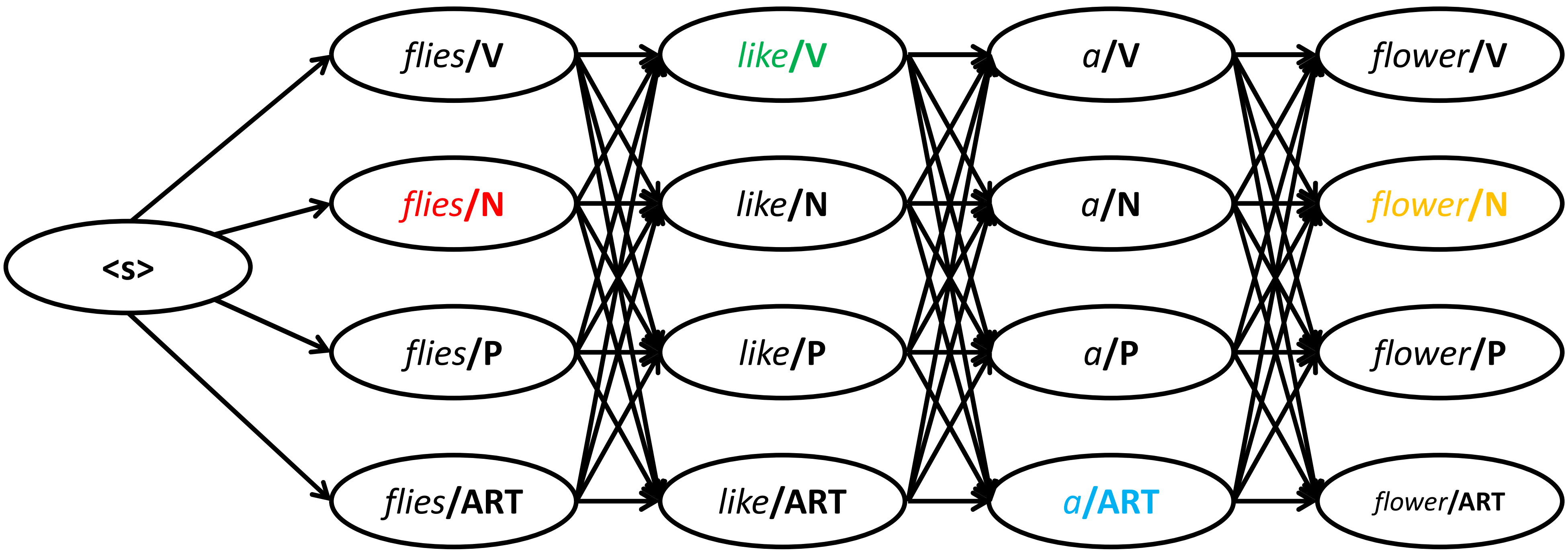
Example: All Possible Sequences



Brute force approach time complexity: O(K^L)

K = 20, L = 10 → 20^{10} = 10240000000000

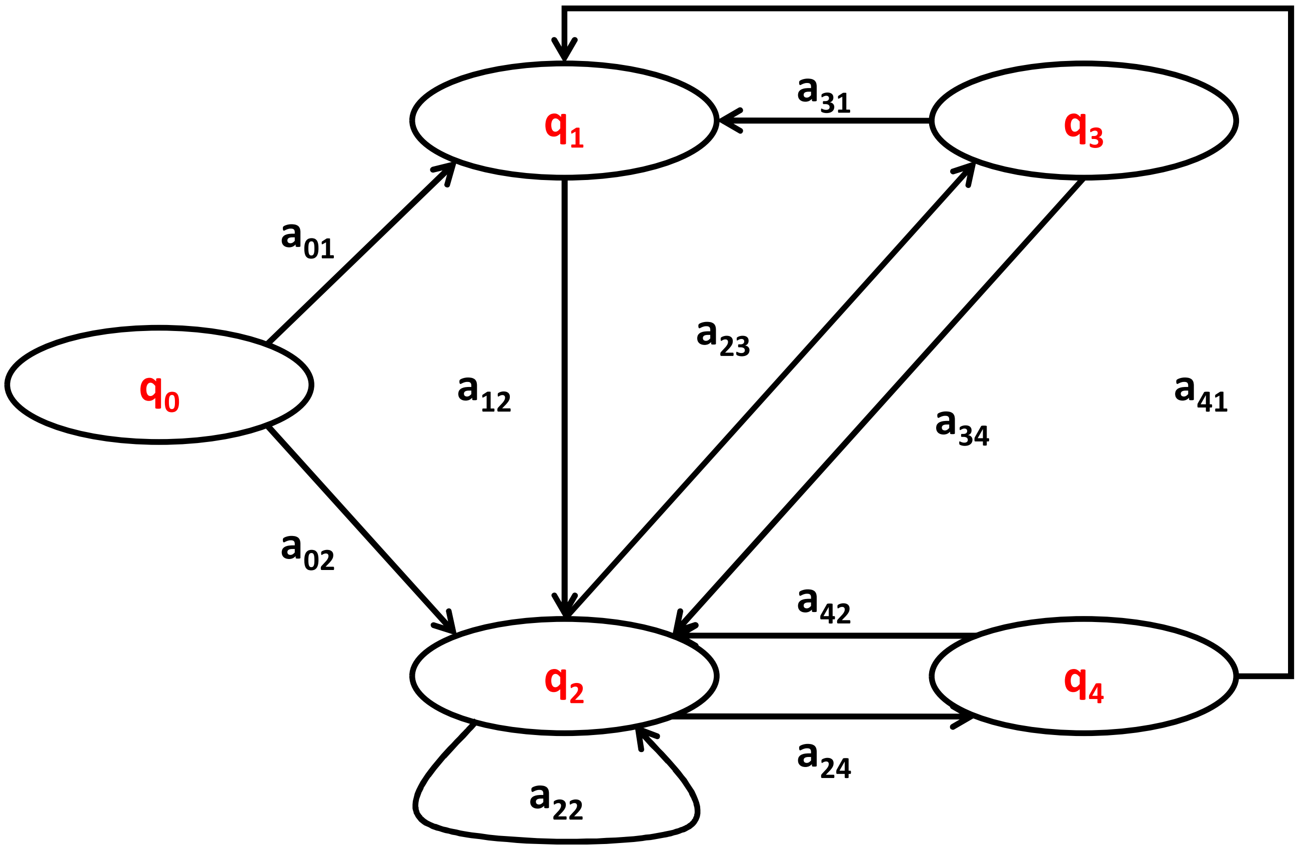
Example: Best Option



How can we efficiently find:

$$\prod_{i=1}^{T} P(w_i \mid C_i) = P(flies|N) * P(like|V) * P(a|ART) * P(flower|N)$$

Hidden Markov Model



Transition probability matrix A						
	q_0	q_1	q_2	q_3	q_4	Notes
q_0	a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	row sum = 1
q_1	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	row sum = 1
q_2	a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	row sum = 1
q_3	a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	row sum = 1
q_4	a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	row sum = 1

HMMs are specified with:

- A set of N states:

  $Q = \{q_1, q_2, \ldots, q_N\}$

- A transition probability matrix A, where each a_{ij} represents the probability of moving from state q_i to state q_j
- A sequence of observations O:

  $O = o_1, o_2, \ldots, o_T$

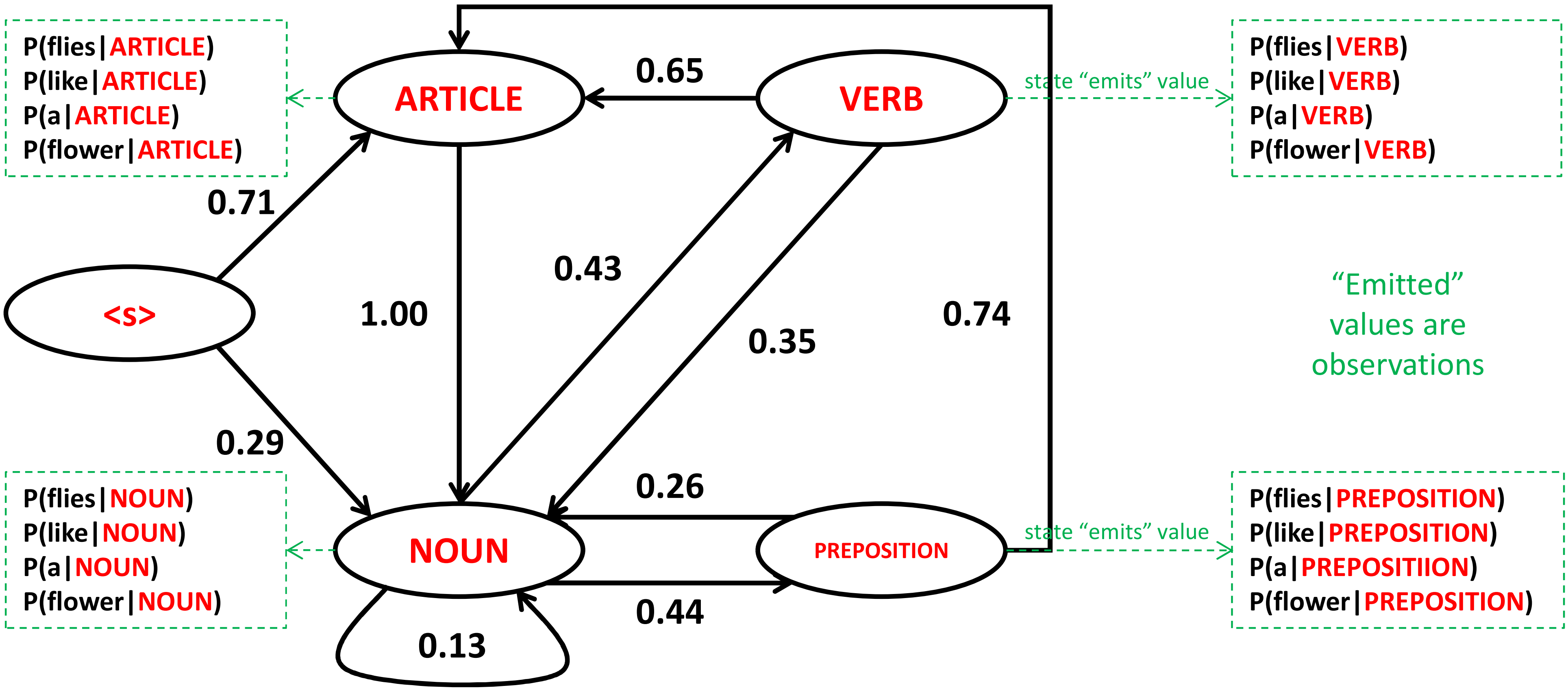
- A sequence of observation likelihoods (emission probabilities): probability of observation o_T being generated by a state q_i

  $B = b_i(o_t)$

- Special <s> and end (final) states

  q_0 and q_F

Hidden Markov Model



Transition probability matrix

	<s>	ARTICLE	NOUN	VERB	PREPOSITION
<s>	0.00	0.71	0.29	0.00	0.00
ARTICLE	0.00	0.00	1.00	0.00	0.00
NOUN	0.00	0.00	0.13	0.43	0.44
VERB	0.00	0.65	0.35	0.00	0.00
PREPOSITION	0.00	0.74	0.26	0.00	0.00

Emission probability matrix

	flies	like	a	flower
<s>	0.000	0.000	0.000	0.000
ARTICLE	0.000	0.000	0.360	0.000
NOUN	0.025	0.012	0.001	0.063
VERB	0.076	0.100	0.000	0.050
PREPOSITION	0.000	0.068	0.000	0.000

Hidden Markov Models: Decoding

The task of determining which sequence of variables is the underlying source of some sequence of observations is called the decoding:

Given as input an HMM $\alpha = (A, B)$ *and a sequence of observations* $o_1, o_2, \ldots, o_T$ *find the most probable sequence of states* $q_1, q_2, \ldots, q_T$ *.*

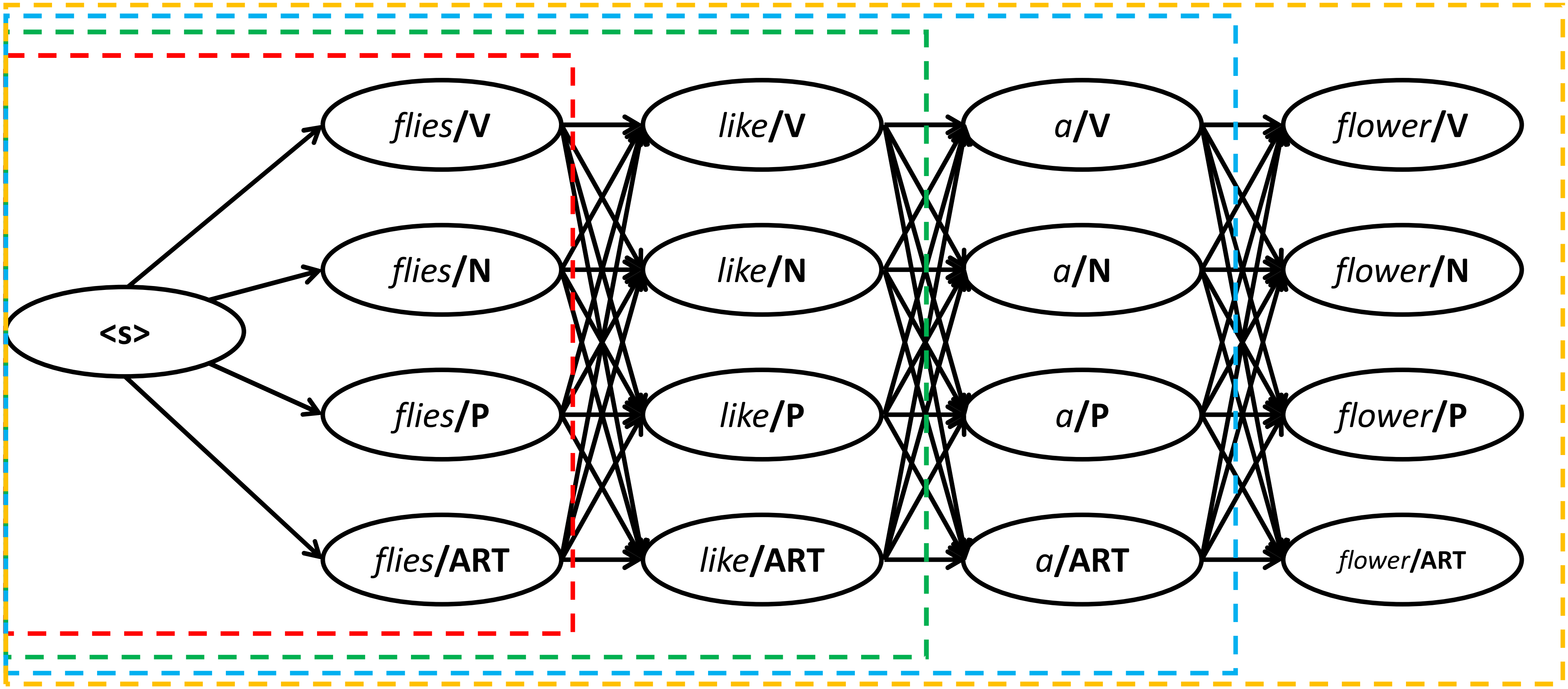
or in our case:

Given as input an HMM $\alpha = (A, B)$ *and a sequence of* ***words*** $w_1, w_2, \ldots, w_T$ *find the most probable sequence of* ***tags/states*** $C_1, C_2, \ldots, C_T$ *.*

A - transition probabilities matrix

B - emission probabilities matrix

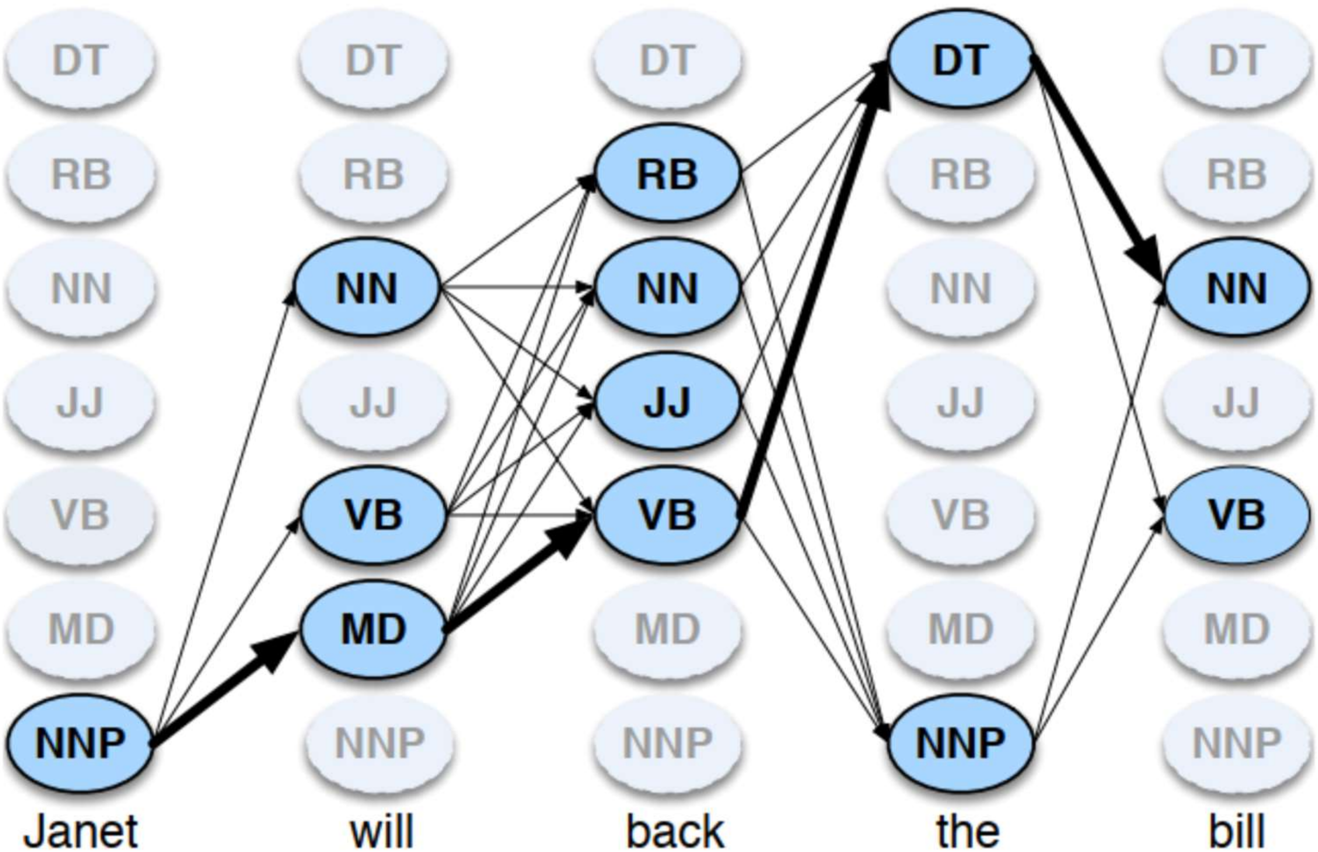
Viterbi Algorithm: the Idea



Maximizing means maximizing , , and .

In other words: maximize P() for all "sub-sentences".

Example: Possible Tag Sequences



Brown corpus tags:

NN - commn noun
NNP - singular proper noun
DT - singular determiner
VB - verb, base form
JJ - adjcective
MD - modal auxiliary
RB - adverb

	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

Transition probabilities matrix

	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	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Emission probabilities matrix

Viterbi Algorithm: Pseudocode

function VITERBI(*observations* of len T,*state-graph* of len N) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
    $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
    $backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    $$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$
    $$backpointer[s,t] \leftarrow \operatorname*{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$
$$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$$; termination step
$$bestpathpointer \leftarrow \operatorname*{argmax}_{s=1}^{N} viterbi[s,T]$$; termination step
bestpath ← the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

Viterbi Algorithm: Pseudocode

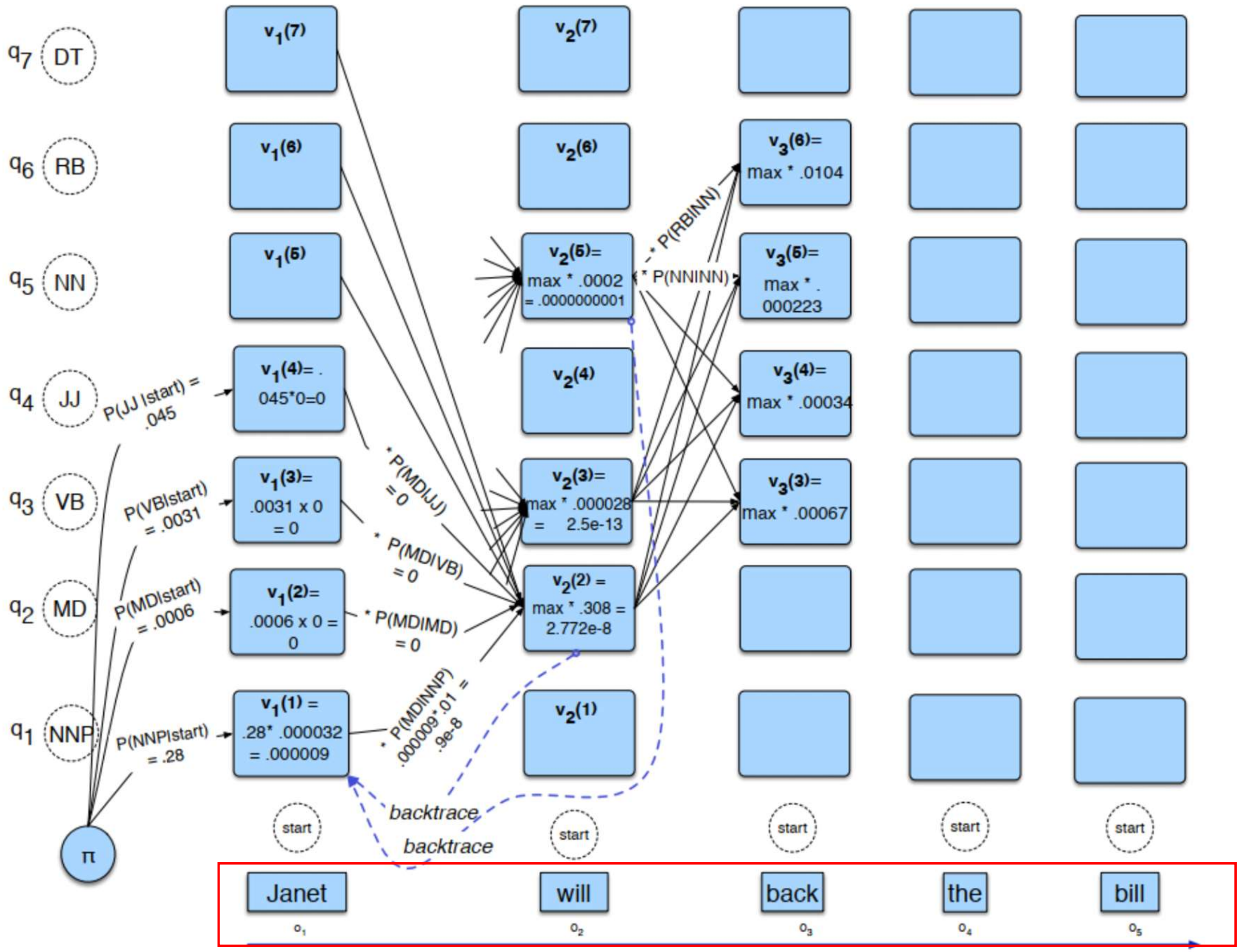
function VITERBI($\boxed{observations \text{ of len } T,}$*state-graph* of len N) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
    $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
    $backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    $$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$
    $$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$; termination step

$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$; termination step

bestpath $\leftarrow$ the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

Viterbi Algorithm: Example



Viterbi Algorithm: Pseudocode

function VITERBI(*observations* of len T,*state-graph* of len N) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi[N,T]*

$\boxed{\begin{array}{l}\textbf{for } \text{each state } s \textbf{ from } 1 \textbf{ to } N \textbf{ do} \\ \quad viterbi[s,1] \leftarrow \pi_s * b_s(o_1)\end{array}}$; initialization step

$\quad backpointer[s,1] \leftarrow 0$

for each time step t **from** 2 **to** T **do** ; recursion step

for each state s **from** 1 **to** N **do**

$$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

$$backpointer[s,t] \leftarrow \operatorname*{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

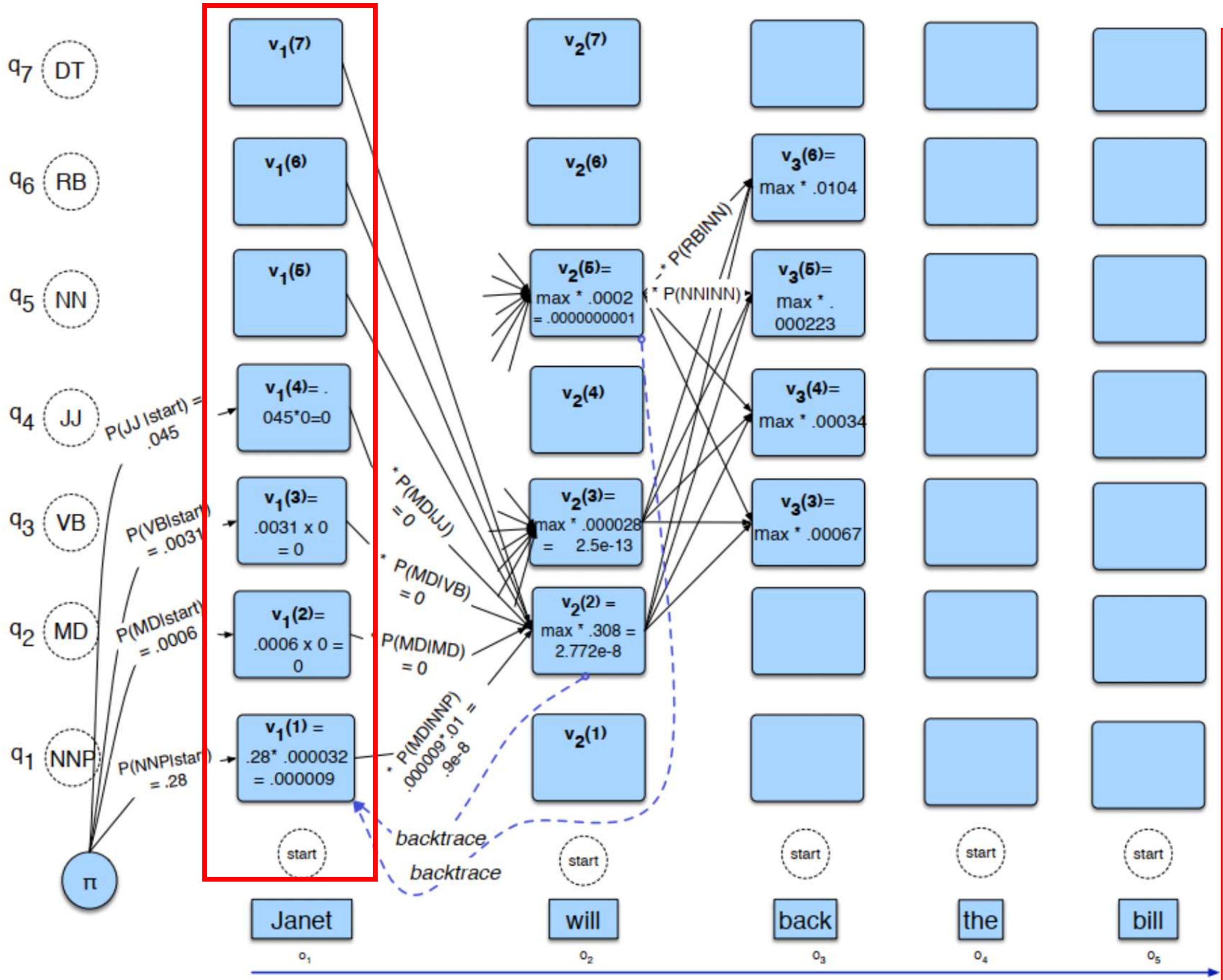
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$; termination step

$bestpathpointer \leftarrow \operatorname*{argmax}_{s=1}^{N} viterbi[s,T]$; termination step

bestpath $\leftarrow$ the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time

return *bestpath*, *bestpathprob*

Viterbi Algorithm: Example



Initialization step:

$v_1(1) = v_1(NNP) = P(T)*P(E) = (NNP|<s>)*P(Janet|NNP)$

$v_1(2) = v_1(MD) = P(T)*P(E) = (MD|<s>)*P(Janet|MD)$

$v_1(3) = v_1(VB) = P(T)*P(E) = (VB|<s>)*P(Janet|VB)$

$v_1(4) = v_1(JJ) = P(T)*P(E) = (JJ|<s>)*P(Janet|JJ)$

$v_1(5) = v_1(NN) = P(T)*P(E) = (NN|<s>)*P(Janet|NN)$

$v_1(6) = v_1(RB) = P(T)*P(E) = (RB|<s>)*P(Janet|RB)$

$v_1(7) = v_1(DT) = P(T)*P(E) = (DT|<s>)*P(Janet|DT)$

P(T) - P(transition)
P(E) - P(emission)

Viterbi Algorithm: Example

Transition probabilities matrix

	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

Emission probabilities matrix

	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	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Initialization step:

$v_1(1) = v_1(\text{NNP})$ = P(T)*P(E) = (NNP|<s>)*P(Janet|NNP)

$v_1(2) = v_1(\text{MD})$ = P(T)*P(E) = (MD|<s>)*P(Janet|MD)

$v_1(3) = v_1(\text{VB})$ = P(T)*P(E) = (VB|<s>)*P(Janet|VB)

$v_1(4) = v_1(\text{JJ})$ = P(T)*P(E) = (JJ|<s>)*P(Janet|JJ)

$v_1(5) = v_1(\text{NN})$ = P(T)*P(E) = (NN|<s>)*P(Janet|NN)

$v_1(6) = v_1(\text{RB})$ = P(T)*P(E) = (RB|<s>)*P(Janet|RB)

$v_1(7) = v_1(\text{DT})$ = P(T)*P(E) = (DT|<s>)*P(Janet|DT)

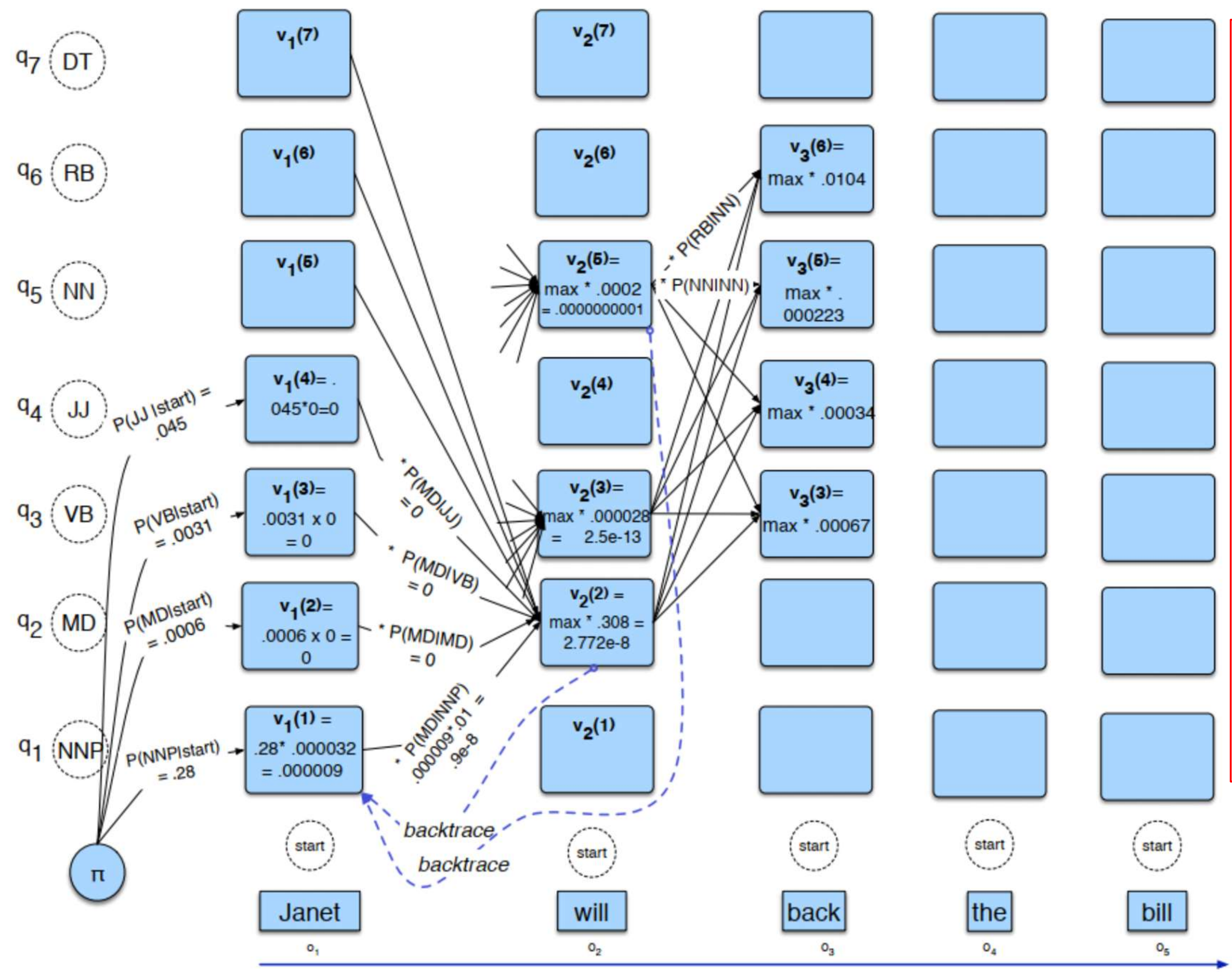
P(T) - P(transition)
P(E) - P(emission)

Viterbi Algorithm: Pseudocode

function VITERBI(*observations* of len T,*state-graph* of len N) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
  $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
  $backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$; termination step
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$; termination step
bestpath ← the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

Viterbi Algorithm: Example



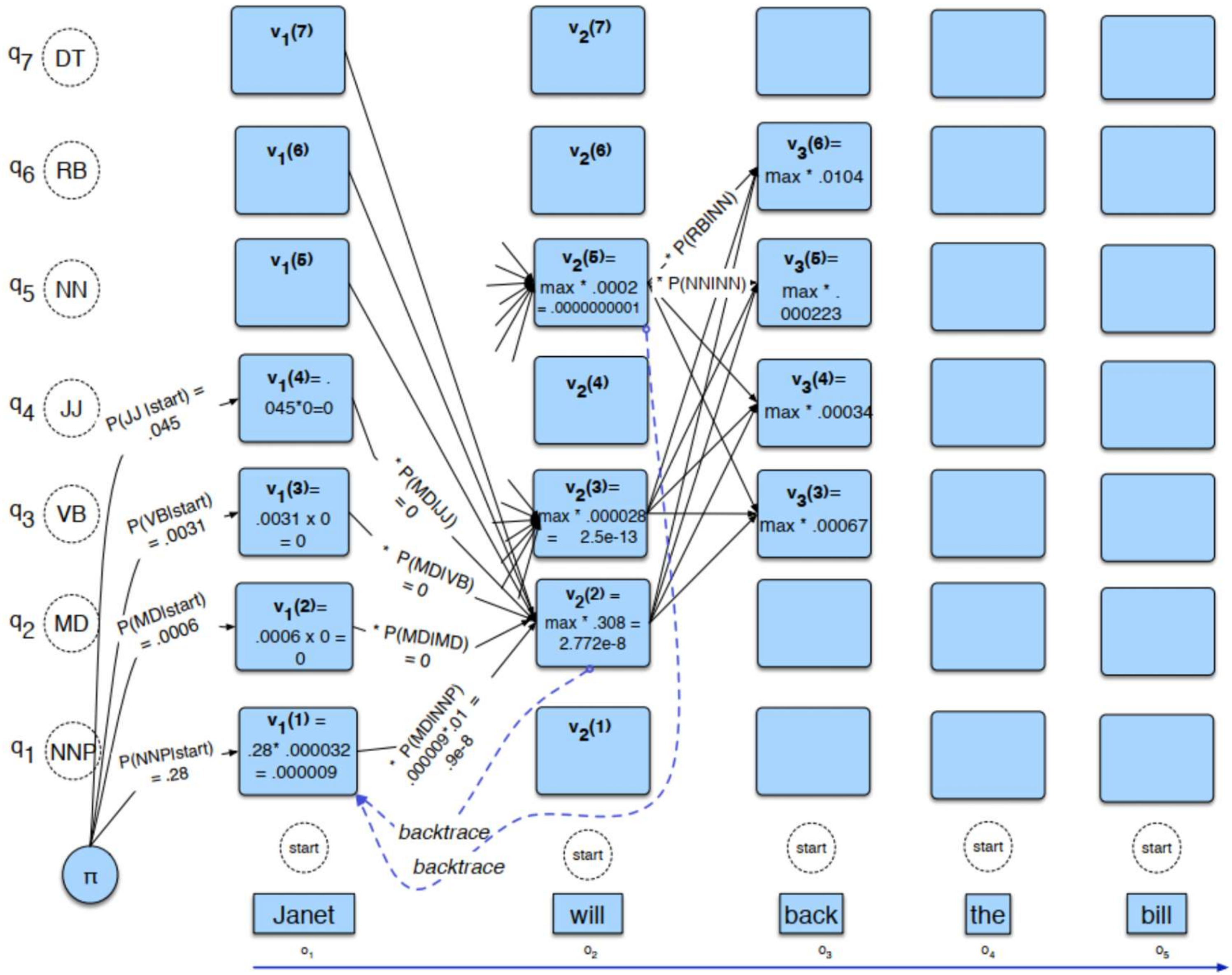
$v_{t-1}(j)$ - previous Viterbi path probability (from the previous time step)

a_{ij} - transition probability from previous state q_i to state q_j

$b_j(o_t)$ - observation likelihood of the observation symbol o_t given the current state q_j

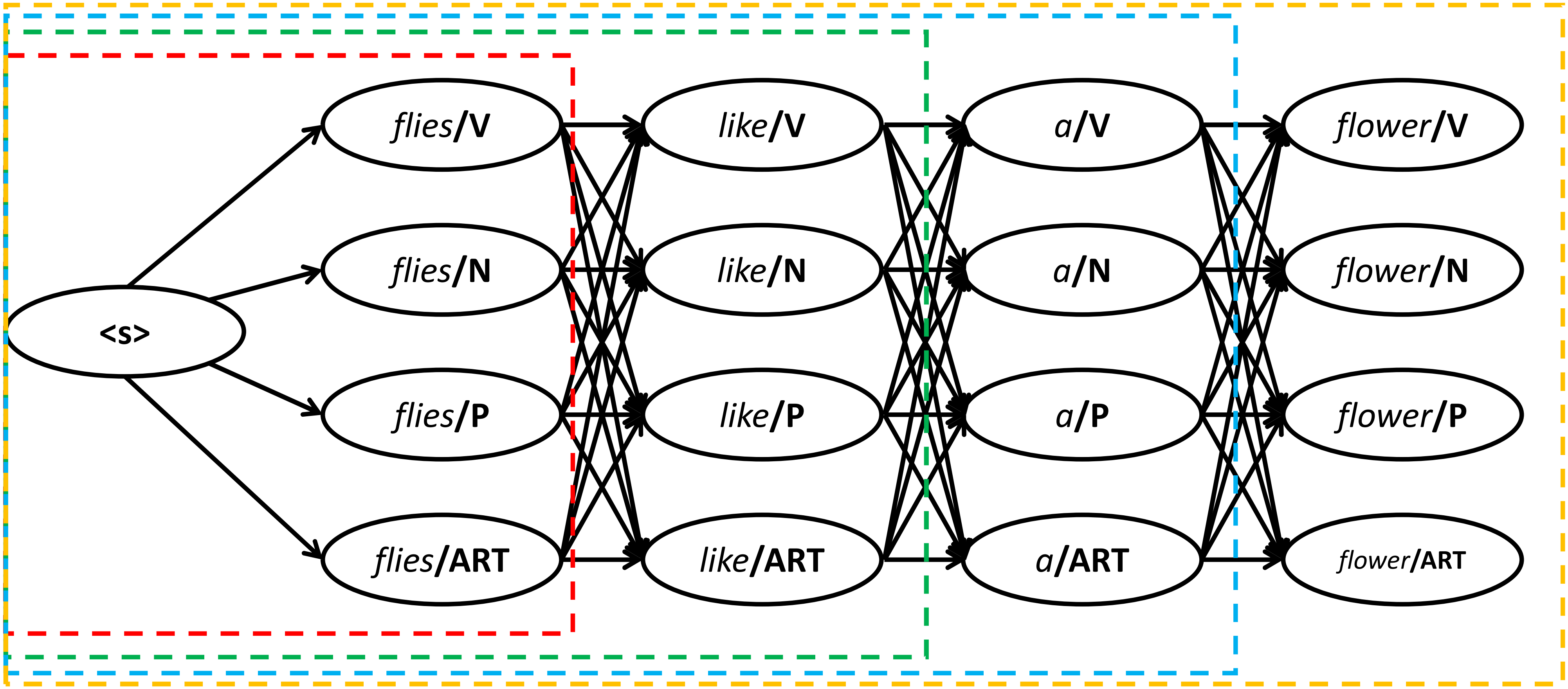
$$\textit{For each state } q_j \textit{ at time } t, \textit{compute: } v_t(j) = \max_{i=1to} v_{t-1}(i) * a_{ij} * b_j(o_t)$$

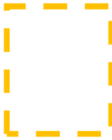
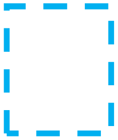
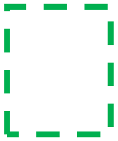
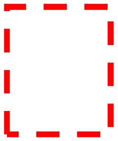
Viterbi Algorithm: Example



For each state q_j *at time t, compute:* $v_t(j) = \max_{i=1toN} v_{t-1}(i) * P(transition) * P(emisssion)$

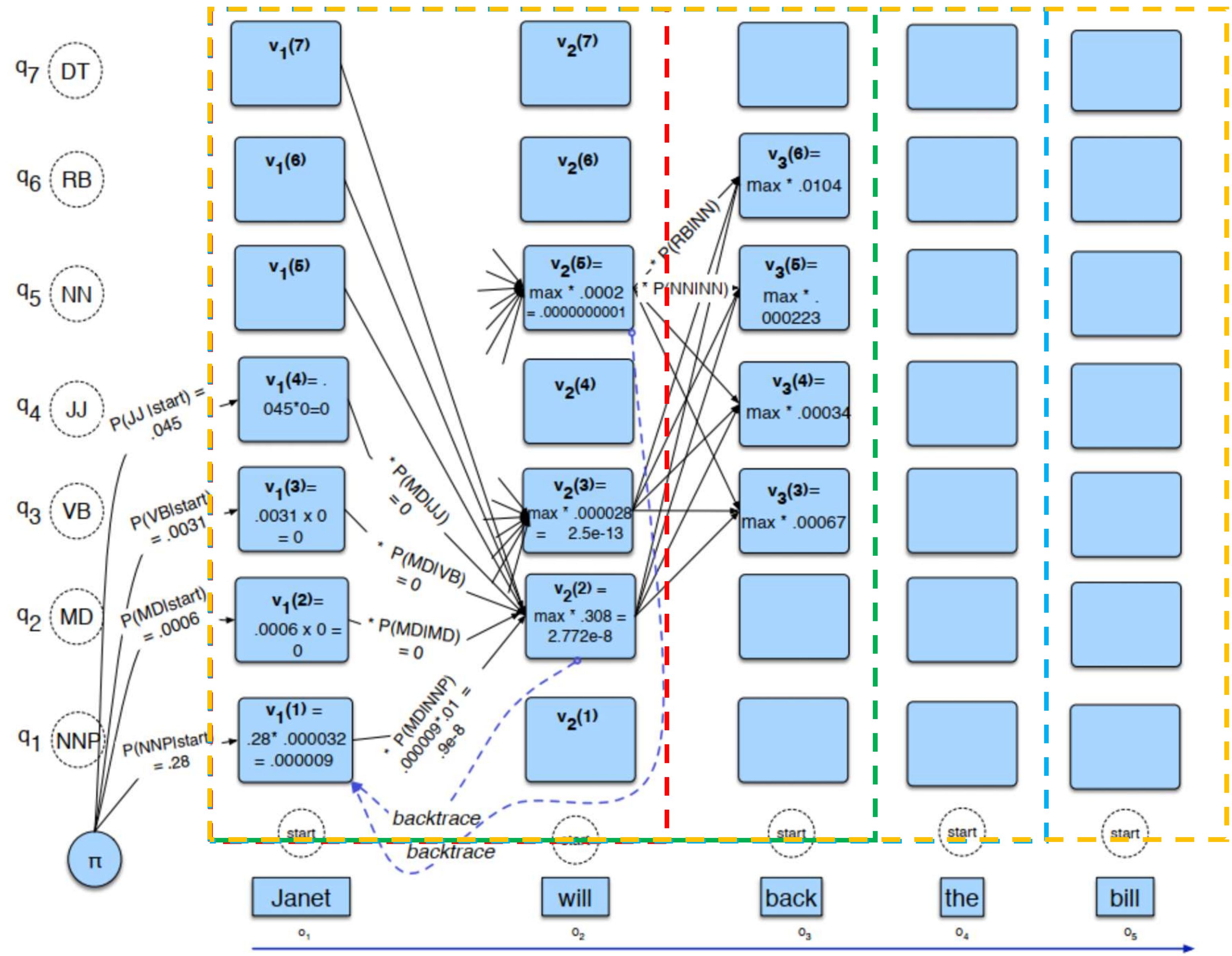
Viterbi Algorithm: the Idea



Maximizing  **means maximizing**  **,**  **, and**  **.**

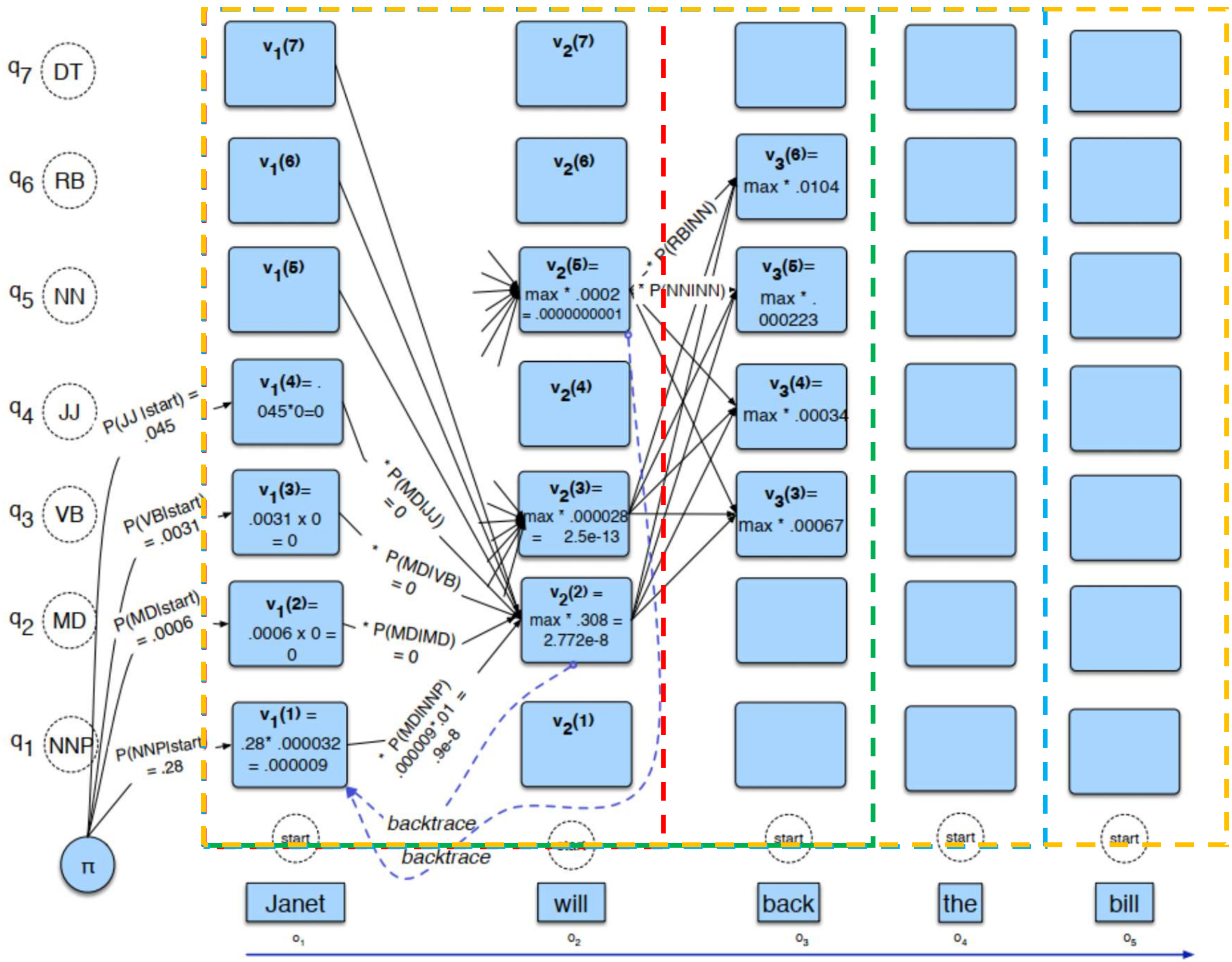
In other words: maximize P() for all "sub-sentences".

Viterbi Algorithm: Example



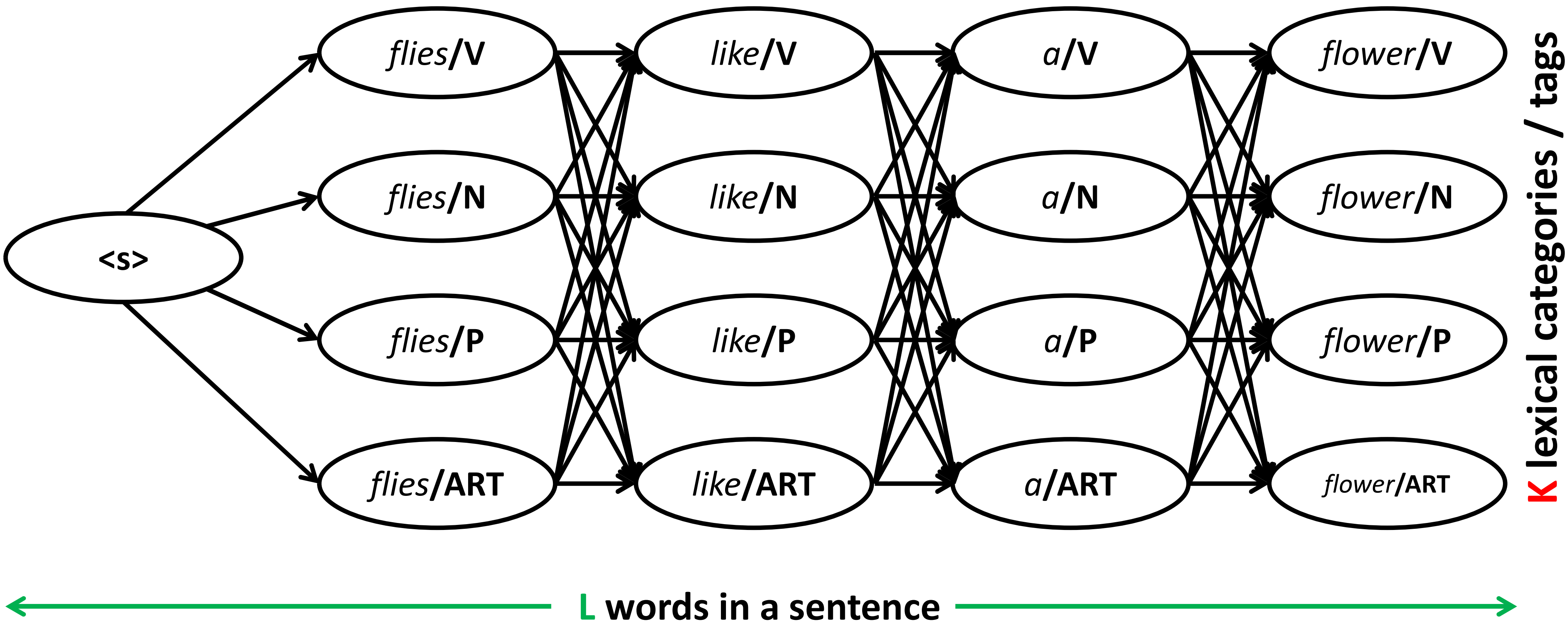
Maximizing [ ] means maximizing [ ], [ ], and [ ].

Viterbi Algorithm: Example



Some paths **need not to be explored**!

Example: Viterbi



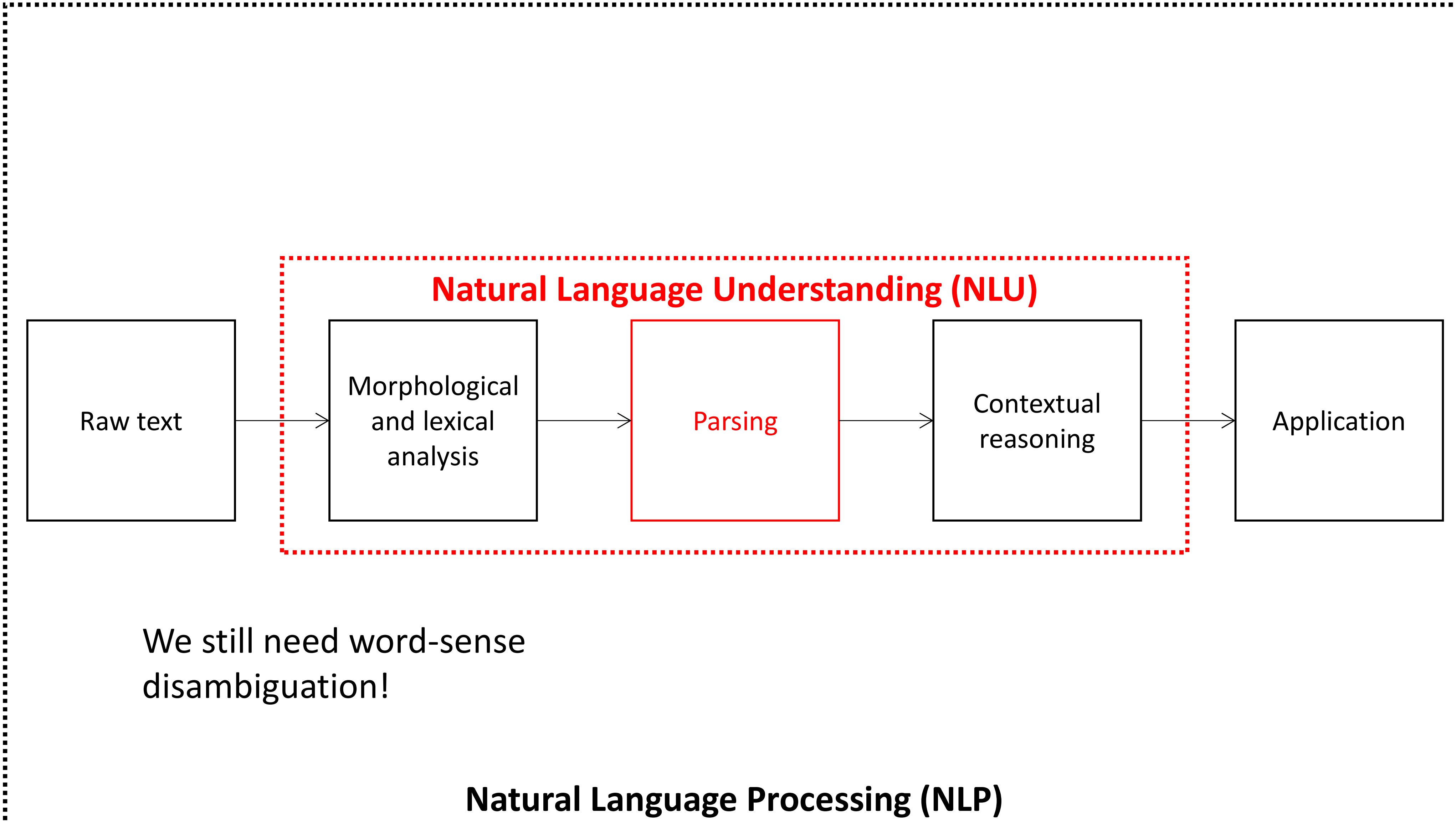
Viterbi algorithm approach time complexity: $O(L * K^2)$

$K = 20$, $L = 10 \rightarrow 10 * 20^2 = 4000$

versus brute force approach time complexity: $O(K^L)$

$K = 20$, $L = 10 \rightarrow 20^{10} = 10240000000000$

Basic NLP Text Processing Pipeline



Syntax

- **Words in sentences are not arranged randomly**
- **Syntax: "the way words are arranged together"**

Formal Languages

- In formal language theory, a language is defined as a set of strings of symbols that may be constrained by specific rules.

- Written English language is made up of groups of letters (words) separated by spaces. A valid (accepted) sentence in the language must follow particular rules, the grammar.

Regular Languages

- **A regular language is a language that can be expressed with a regular expression or a deterministic or non-deterministic finite automata or state machine.**

Context-Free Languages

- A **context-free language** is a **language generated by a context-free grammar**.
- More general than (but include) regular languages.
- The same context-free language might be generated by multiple context-free grammars.

The Concept of Constituency

Groups of words that may behave as a single unit or phrase are called a constituent.

Context-Free Grammar

Contex-Free Grammar: A mathematical system for modeling constituent structure in languages (English, other natural languages, etc.).

Also called Phase-Structure Grammars.

CFG Grammar = Rules + Lexicon

- **Rules** (also called productions): define how individual language symbols can be grouped and ordered together

+

- **Lexicon**: a set of language symbols (NLP: words)

Backus-Naur Form Notation

Backus–Naur form or Backus normal form (BNF) is a metasyntax notation for context-free grammars, often used to describe the syntax of languages used in computing, such as computer programming languages, document formats, instruction sets and communication protocols.

Simple Sample Grammar

$$
\begin{array}{lll}
S & \rightarrow & NP\ VP \\
NP & \rightarrow & Pronoun \\
 & | & Proper\text{-}Noun \\
 & | & Det\ Nominal \\
Nominal & \rightarrow & Nominal\ Noun \\
 & | & Noun \\
VP & \rightarrow & Verb \\
 & | & Verb\ NP \\
 & | & Verb\ NP\ PP \\
 & | & Verb\ PP \\
NP & \rightarrow & Preposition\ NP
\end{array}
$$

Legend:
S - start symbol
NP - noun phrase
VP - verb phrase
PP - prepositional phrase

The | ("or") symbol means that a non-terminal has alternate expansions.

Simple Grammar: Rules / Productions

$$
\begin{array}{lll}
S & \rightarrow & NP\ VP \\
NP & \rightarrow & Pronoun \\
 & | & Proper\text{-}Noun \\
 & | & Det\ Nominal \\
Nominal & \rightarrow & Nominal\ Noun \\
 & | & Noun \\
VP & \rightarrow & Verb \\
 & | & Verb\ NP \\
 & | & Verb\ NP\ PP \\
 & | & Verb\ PP \\
NP & \rightarrow & Preposition\ NP
\end{array}
$$

Grammar
rules /
productions

The | ("or") symbol means that a non-terminal has alternate expansions.

Simple Grammar: Expansions

$$
\begin{array}{lll}
S & \rightarrow & NP\ VP \\
NP & \rightarrow & Pronoun \\
 & | & Proper\text{-}Noun \\
 & | & Det\ Nominal \\
Nominal & \rightarrow & Nominal\ Noun \\
 & | & Noun \\
VP & \rightarrow & Verb \\
 & | & Verb\ NP \\
 & | & Verb\ NP\ PP \\
 & | & Verb\ PP \\
NP & \rightarrow & Preposition\ NP
\end{array}
$$

$$
\begin{array}{lll}
NP & \rightarrow & Pronoun \\
NP & \rightarrow & Proper\text{-}Noun \\
NP & \rightarrow & Det\ Nominal
\end{array}
$$

Multiple expansions of the rule

The | ("or") symbol means that a non-terminal has alternate expansions.

Simple Sample Lexicon

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

The | (“or”) symbol means that a non-terminal has alternate expansions.

Simple Sample Lexicon

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

Non-terminal symbols (generalizations)

Terminal symbols

Simple Grammar: Example Phrases

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*
NP	→ *Preposition NP*	*from Los Angeles*

The | ("or") symbol means that a non-terminal has alternate expansions.

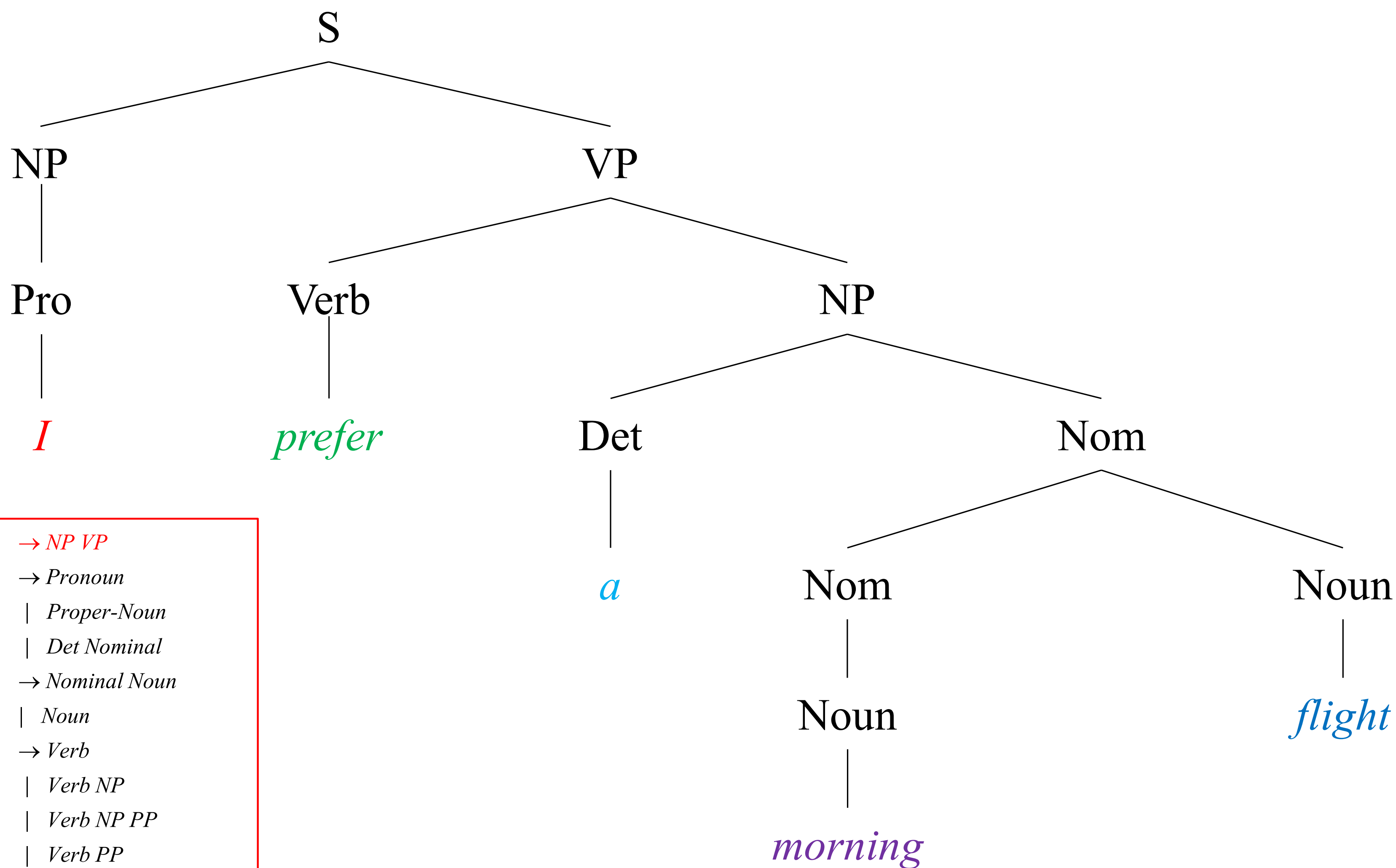
“Context-Free”

“expansion of a non-terminal does not depend on its neighbors”

Parse Tree: Example

Parse tree (representing a sequence of expansions = derivation) for sentence:

I prefer a morning flight



S	→ *NP VP*
NP	→ *Pronoun*
	| *Proper-Noun*
	| *Det Nominal*
Nominal	→ *Nominal Noun*
	| *Noun*
VP	→ *Verb*
	| *Verb NP*
	| *Verb NP PP*
	| *Verb PP*
NP	→ *Preposition NP*

Context Free Grammar

A context-free grammar G is defined by:

- **N: a set of non-terminal symbols (variables)**
- **Σ: a set of terminal symbols (disjoint from N)**
- **R: a set of rules or productions of the form $A \rightarrow \beta$, where:**
  - **A: a non-terminal symbol**
  - **β: a string of symbols from the infinite set of strings $(\Sigma \cup N)^*$**
- **S: a designated start symbol**

Direct Derivation: Formal Definition

One string derives another if it can be rewritten as the second one by some series of rule applications:

If $A \to \beta$ is a rule / production and α and γ are any strings in the set $(\Sigma \cup N)^*$, then we can say that

$\alpha A \gamma$ **directly derives** $\alpha \beta \gamma$

or

$$\alpha A \gamma \Rightarrow \alpha \beta \gamma$$

Derivation: Formal Definition

Derivation is a generalization of direct derivation:

Let $\alpha_1, \alpha_2, \ldots, \alpha_m$ **are strings in the set** $(\Sigma \cup N)^*$**,**
with $m \geq 1$ **such that**

$$\alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha_3, \ldots, \alpha_{m-1} \Rightarrow \alpha_m$$

and we can say that α_1 **derives** α_μ **, or** $\alpha_1 \stackrel{*}{\Rightarrow} \alpha_\mu$

Language vs. Grammar

Language L_G generated by a grammar G is the set of all strings composed of terminal symbols that can be derived from the designated start symbol S.

$$L_G = \{w \mid w \; is \; in \; \Sigma * \;\; and \; S \overset{*}{\Rightarrow} w\}$$

strings made up of terminals $\boxed{\text{derived}}$ from symbol S

Grammatical vs. Ungrammatical

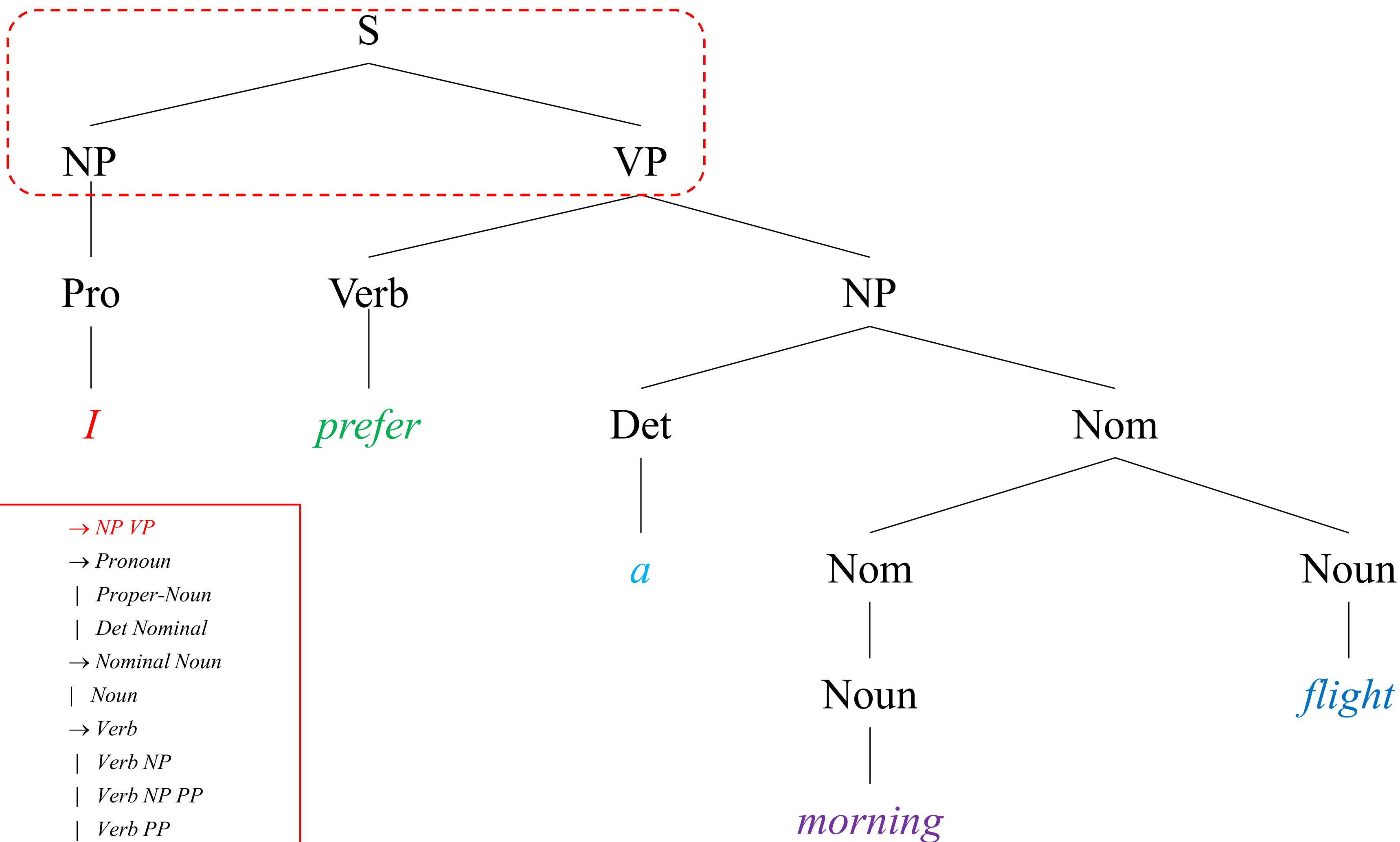
Formal language is made of sentences:

- **sentences that CAN be derived by a grammar are IN the formal language defined by that grammar are called grammatical sentences**
- **sentences that CANNOT be derived by a grammar are NOT IN the formal language defined by that grammar are called ungrammatical sentences**

Parse Tree: Example

Parse tree for:

I prefer a morning flight

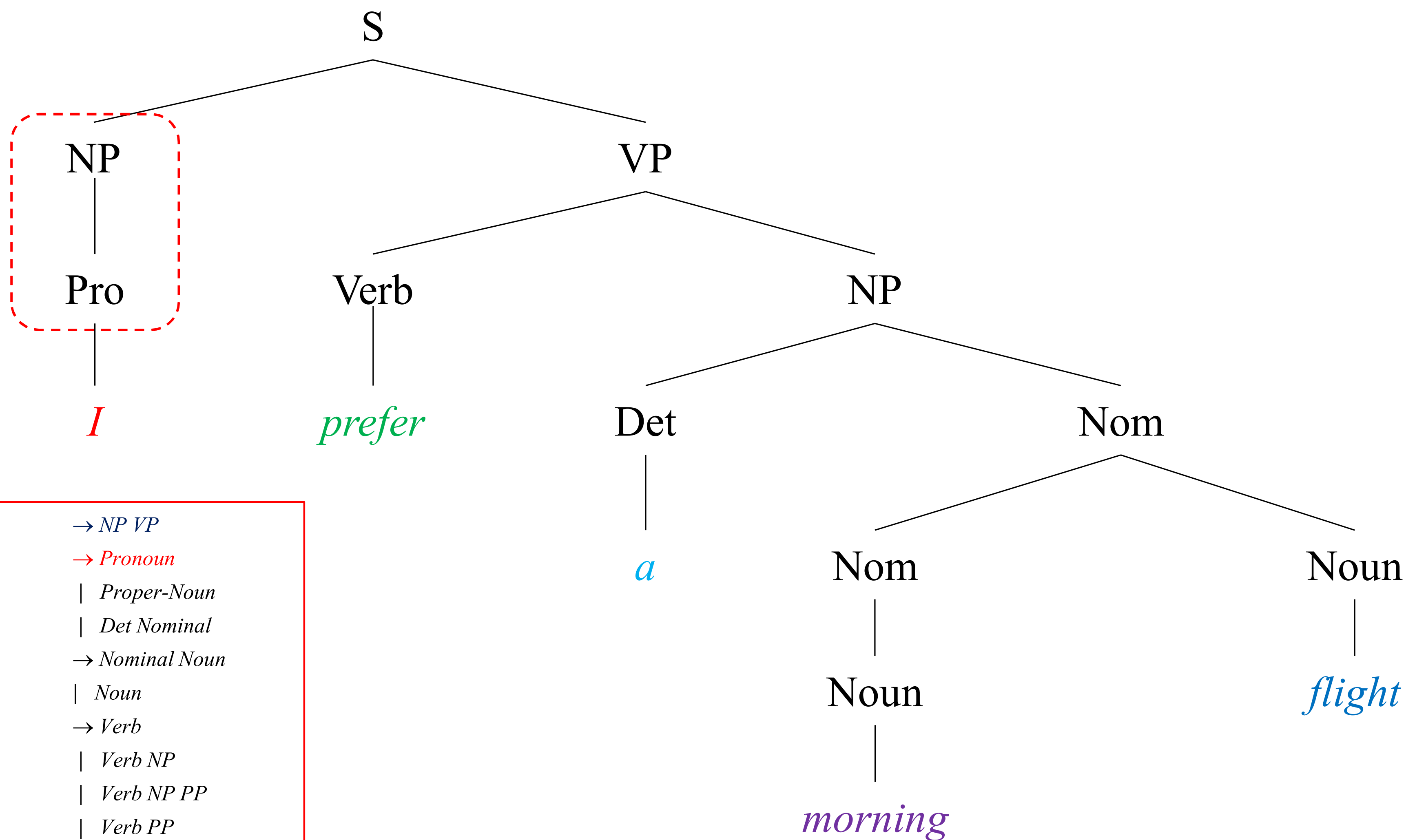


S	$\rightarrow$ *NP VP*
NP	$\rightarrow$ *Pronoun*
	| *Proper-Noun*
	| *Det Nominal*
Nominal	$\rightarrow$ *Nominal Noun*
	| *Noun*
VP	$\rightarrow$ *Verb*
	| *Verb NP*
	| *Verb NP PP*
	| *Verb PP*
NP	$\rightarrow$ *Preposition NP*

Parse Tree: Example

Parse tree for:

I prefer a morning flight

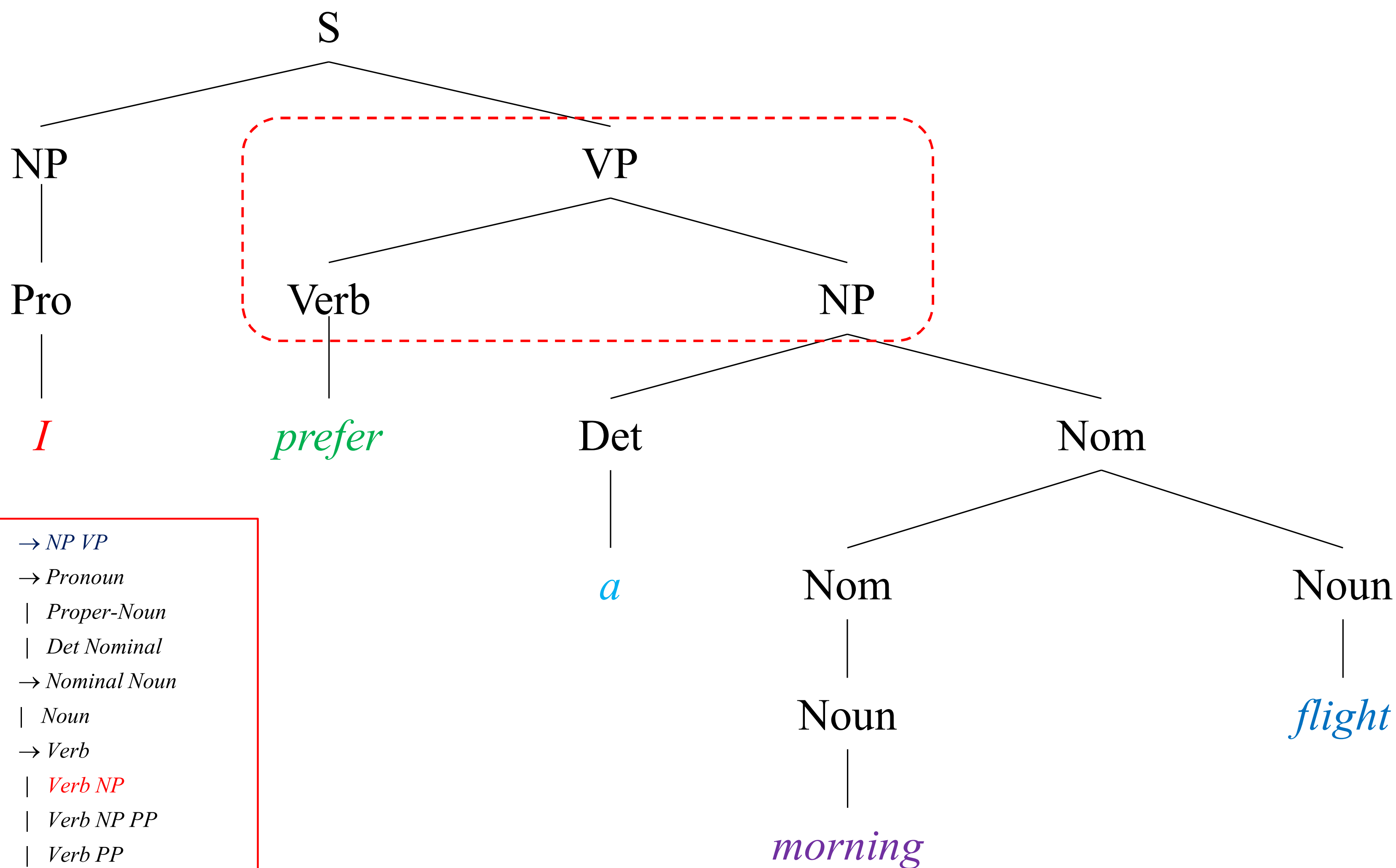


S	→ *NP VP*
NP	→ *Pronoun*
	| *Proper-Noun*
	| *Det Nominal*
Nominal	→ *Nominal Noun*
	| *Noun*
VP	→ *Verb*
	| *Verb NP*
	| *Verb NP PP*
	| *Verb PP*
NP	→ *Preposition NP*

Parse Tree: Example

Parse tree for:

I prefer a morning flight

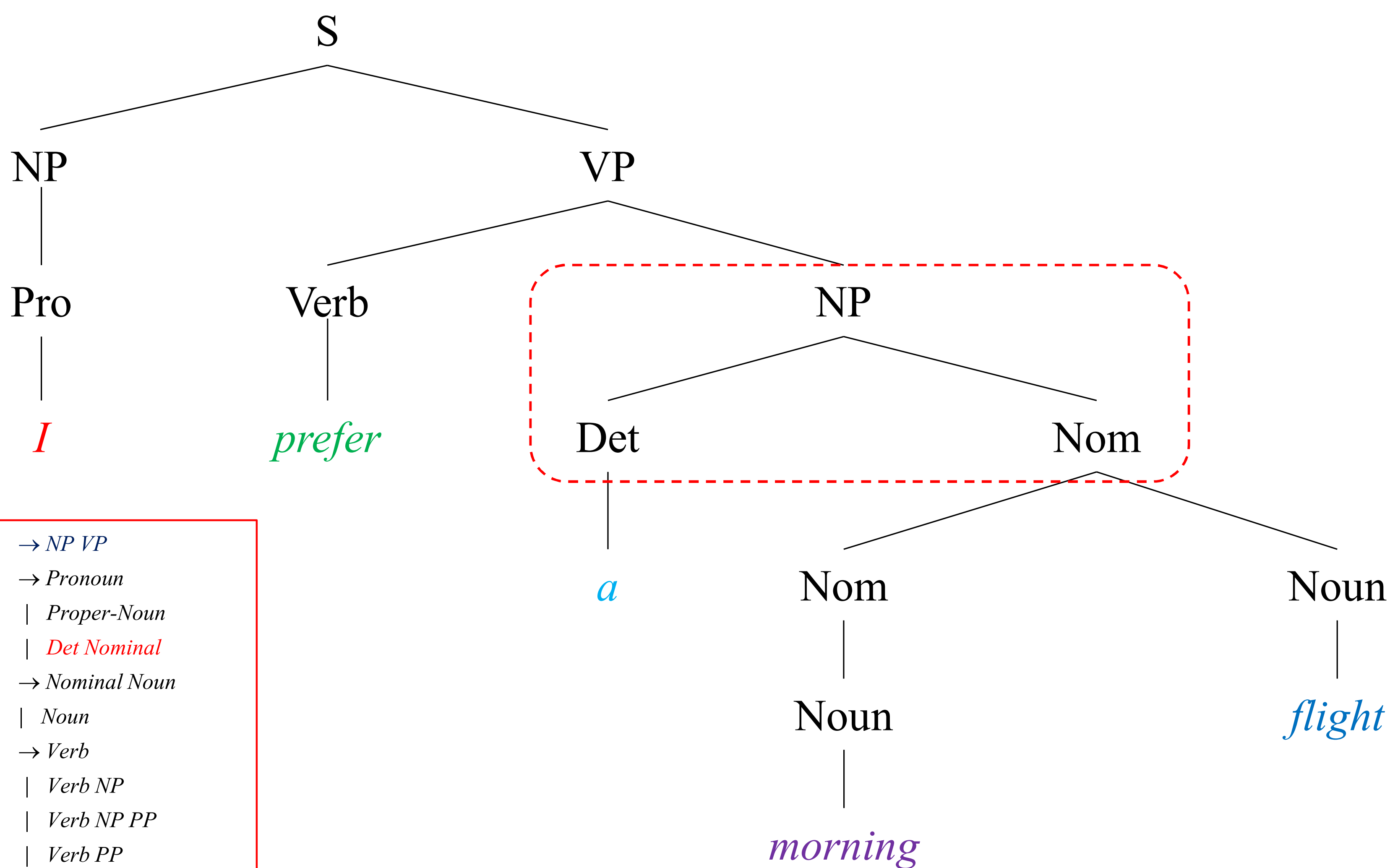


S	→ *NP VP*
NP	→ *Pronoun*
	| *Proper-Noun*
	| *Det Nominal*
Nominal	→ *Nominal Noun*
	| *Noun*
VP	→ *Verb*
	| *Verb NP*
	| *Verb NP PP*
	| *Verb PP*
NP	→ *Preposition NP*

Parse Tree: Example

Parse tree for:

I prefer a morning flight

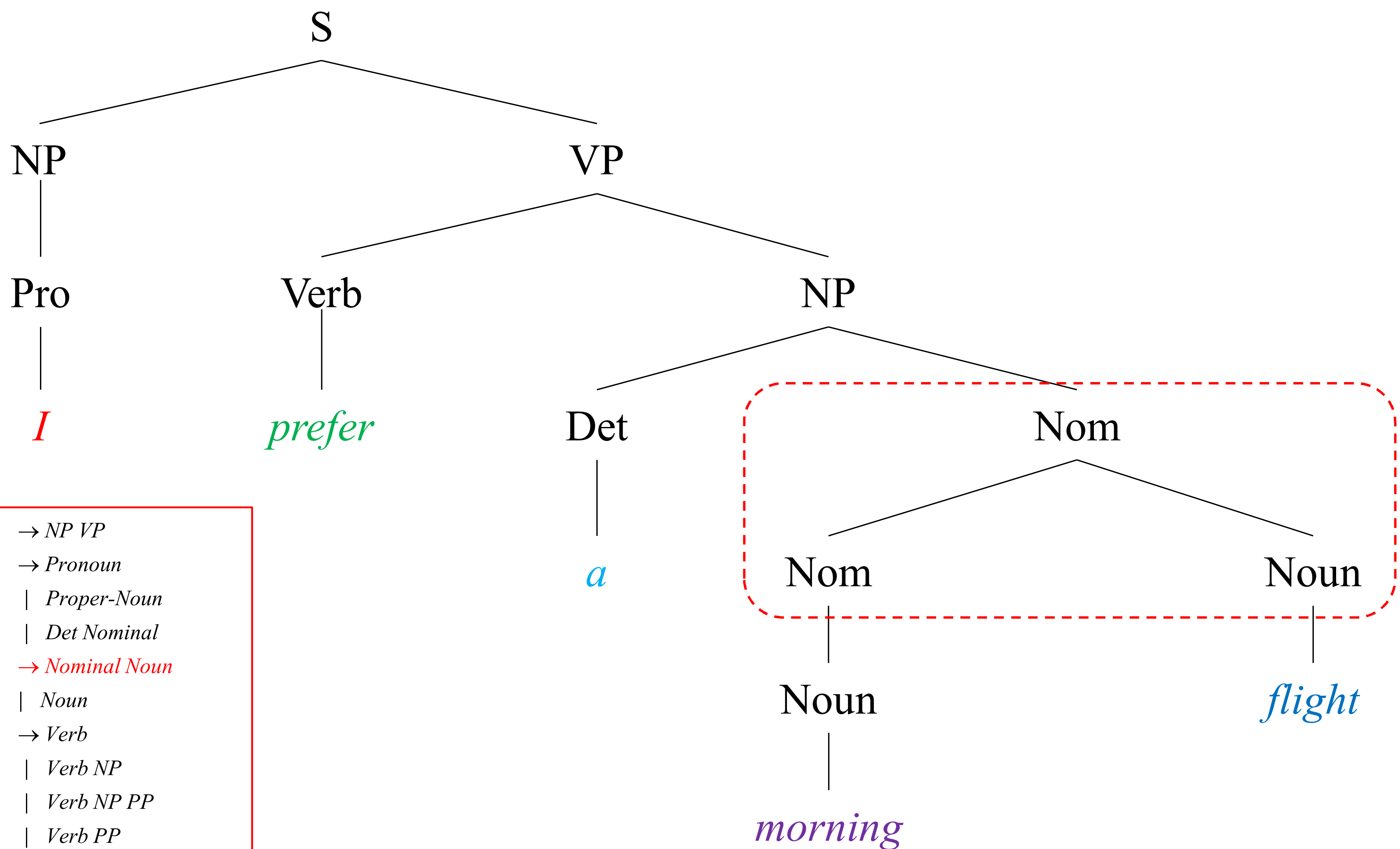


S	→ *NP VP*
NP	→ *Pronoun*
	| *Proper-Noun*
	| *Det Nominal*
Nominal	→ *Nominal Noun*
	| *Noun*
VP	→ *Verb*
	| *Verb NP*
	| *Verb NP PP*
	| *Verb PP*
NP	→ *Preposition NP*

Parse Tree: Example

Parse tree for:

I prefer a morning flight

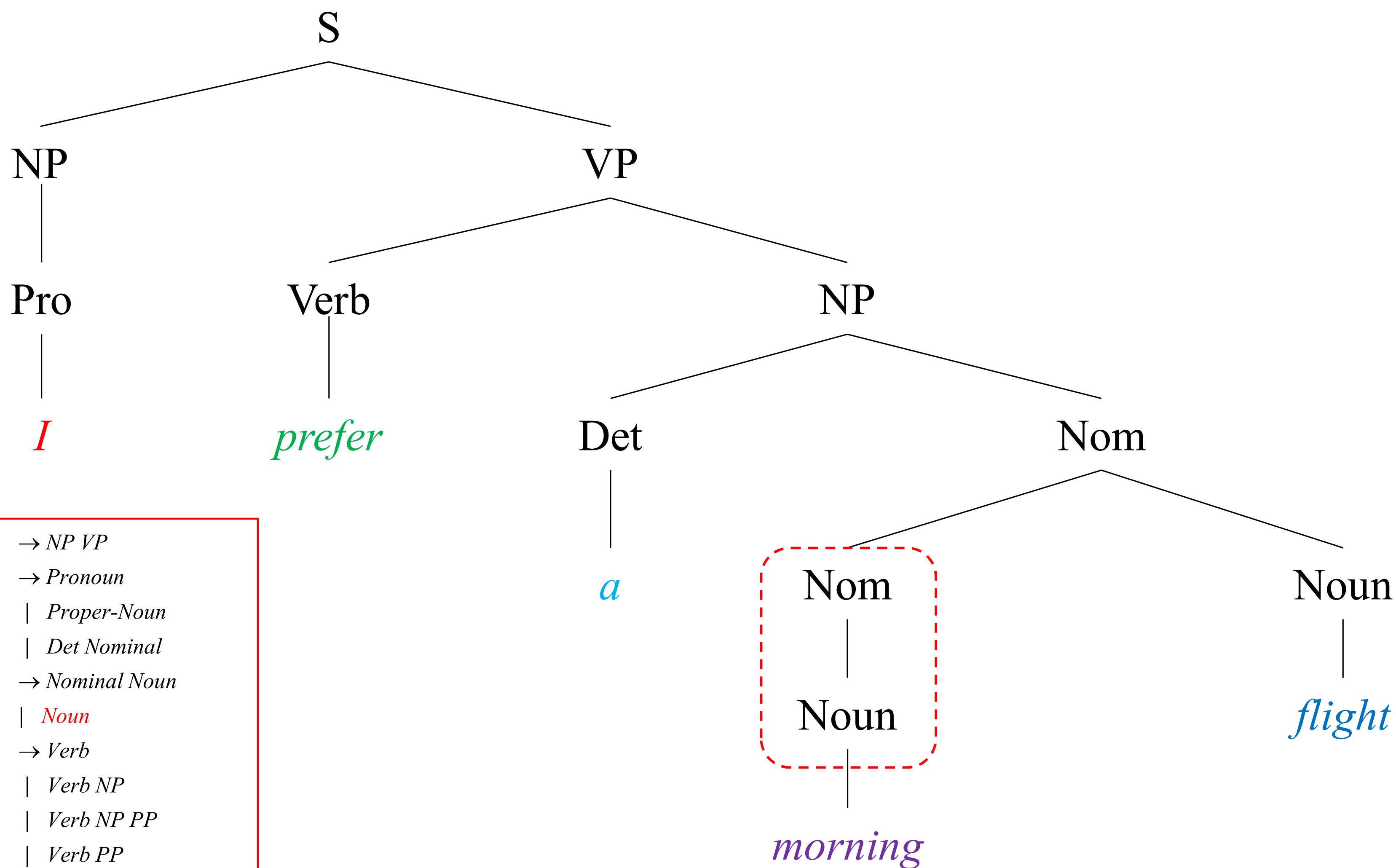


S	→ *NP VP*
NP	→ *Pronoun*
	| *Proper-Noun*
	| *Det Nominal*
Nominal	→ *Nominal Noun*
	| *Noun*
VP	→ *Verb*
	| *Verb NP*
	| *Verb NP PP*
	| *Verb PP*
NP	→ *Preposition NP*

Parse Tree: Example

Parse tree for:

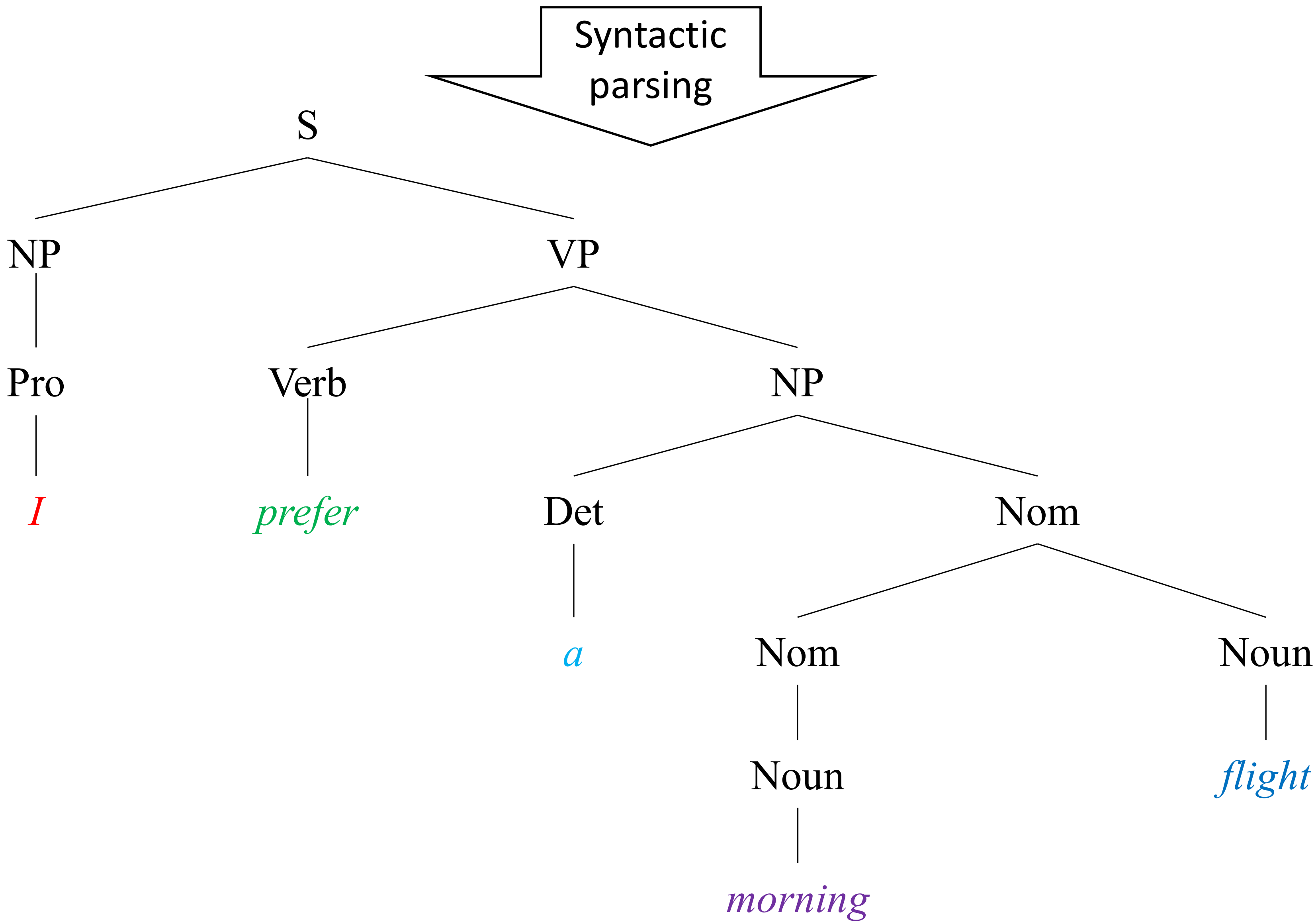
I prefer a morning flight



S	→	*NP VP*
NP	→	*Pronoun*
	|	*Proper-Noun*
	|	*Det Nominal*
Nominal	→	*Nominal Noun*
	|	*Noun*
VP	→	*Verb*
	|	*Verb NP*
	|	*Verb NP PP*
	|	*Verb PP*
NP	→	*Preposition NP*

Syntactic Parsing: Sentence → Tree

I prefer a morning flight



Parsing

The task of determining the parts of speech, phrases, clauses, and their relationship to one another is called parsing.

BNF Example: CNF Propositional Logic

$$
\begin{aligned}
CNFSentence &\rightarrow Clause_1 \wedge \cdots \wedge Clause_n \\
Clause &\rightarrow Literal_1 \vee \cdots \vee Literal_m \\
Fact &\rightarrow Symbol \\
Literal &\rightarrow Symbol \mid \neg Symbol \\
Symbol &\rightarrow P \mid Q \mid R \mid \ldots \\
HornClauseForm &\rightarrow DefiniteClauseForm \mid GoalClauseForm \\
DefiniteClauseForm &\rightarrow Fact \mid (Symbol_1 \wedge \cdots \wedge Symbol_l) \Rightarrow Symbol \\
GoalClauseForm &\rightarrow (Symbol_1 \wedge \cdots \wedge Symbol_l) \Rightarrow False
\end{aligned}
$$

- **BNF: Backus-Naur Form**
- **CNF: Conjunctive Normal Form**

Arithmetic Expressions Grammar

$$S \rightarrow S \; Op \; S \mid Num$$

$$Op \rightarrow + \mid - \mid \times \mid \div$$

$$Num \rightarrow Num \; Digit \mid Digit$$

$$Digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

The | ("or") symbol means that a non-terminal has alternate expansions.

Arithmetic Expressions Grammar

$$\begin{array}{ll} S & \rightarrow S\ Op\ S \mid Num \\ Op & \rightarrow + \mid - \mid \times \mid \div \\ Num & \rightarrow Num\ Digit \mid Digit \\ Digit & \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{array}$$

Lexicon

The | (“or”) symbol means that a non-terminal has alternate expansions.

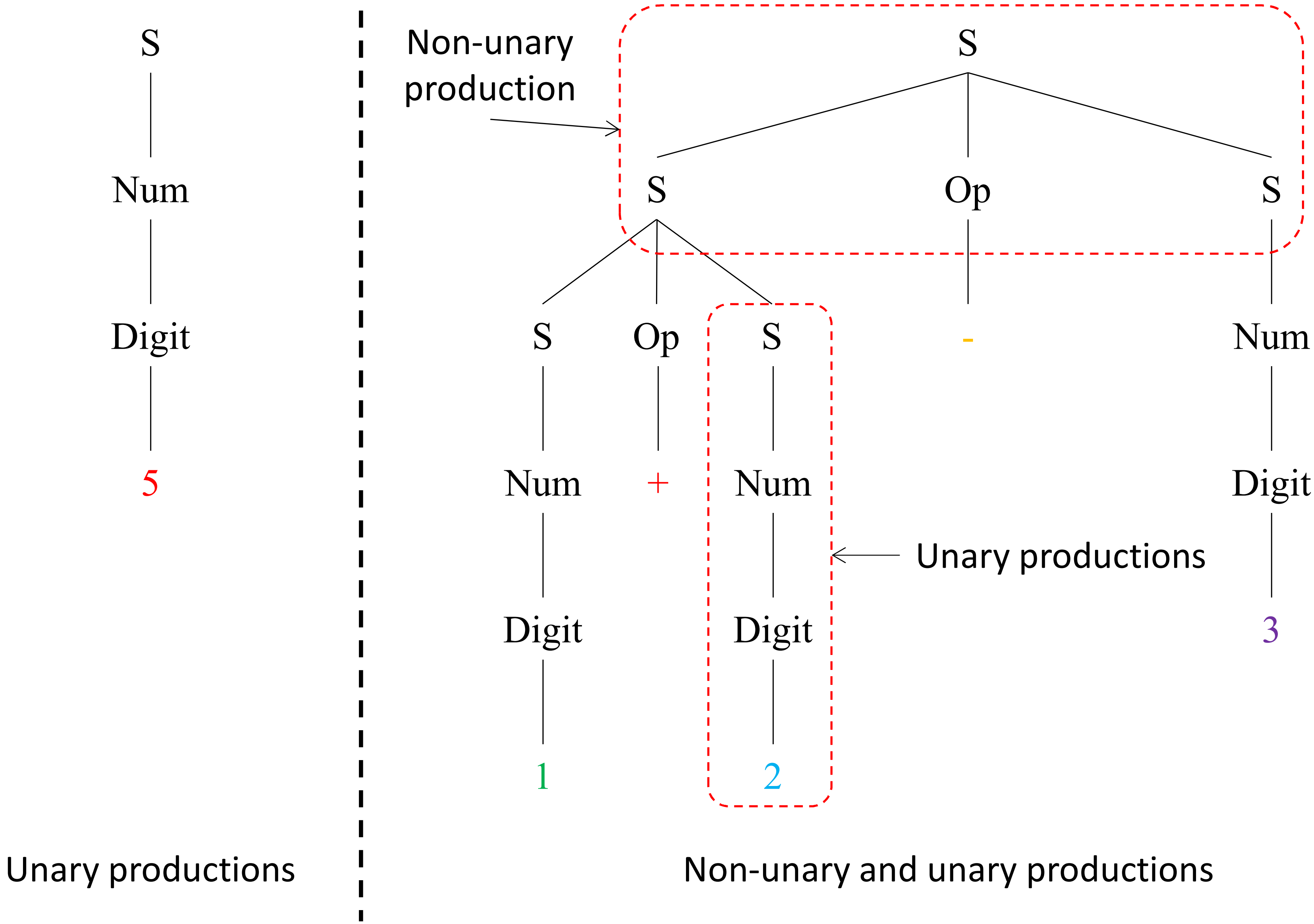
Arithmetic Expressions Grammar

$$
\begin{aligned}
S & \rightarrow S\ Op\ S \mid Num \\
Op & \rightarrow + \mid - \mid \times \mid \div \\
Num & \rightarrow Num\ Digit \mid Digit \\
Digit & \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}
$$

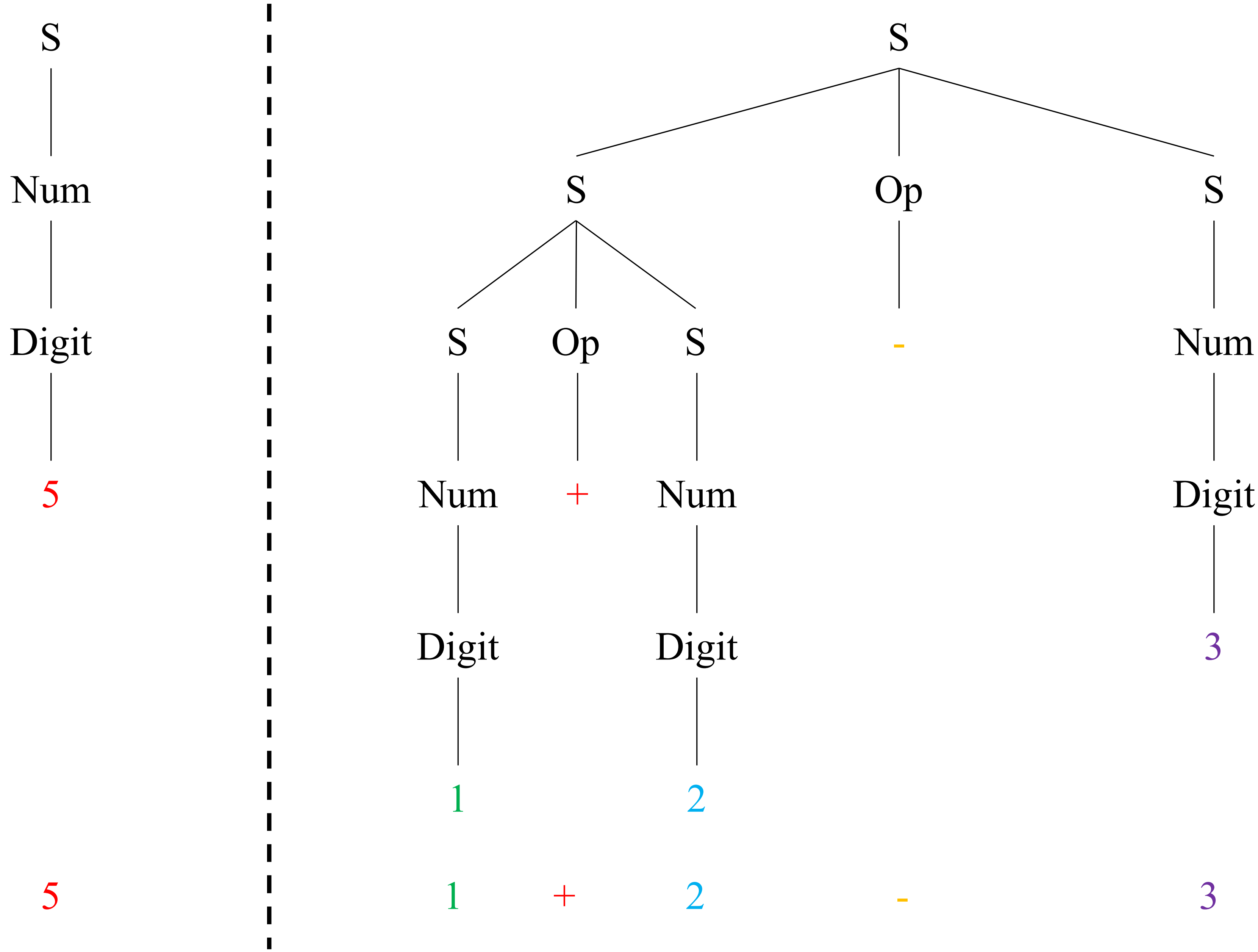
Rules

The | ("or") symbol means that a non-terminal has alternate expansions.

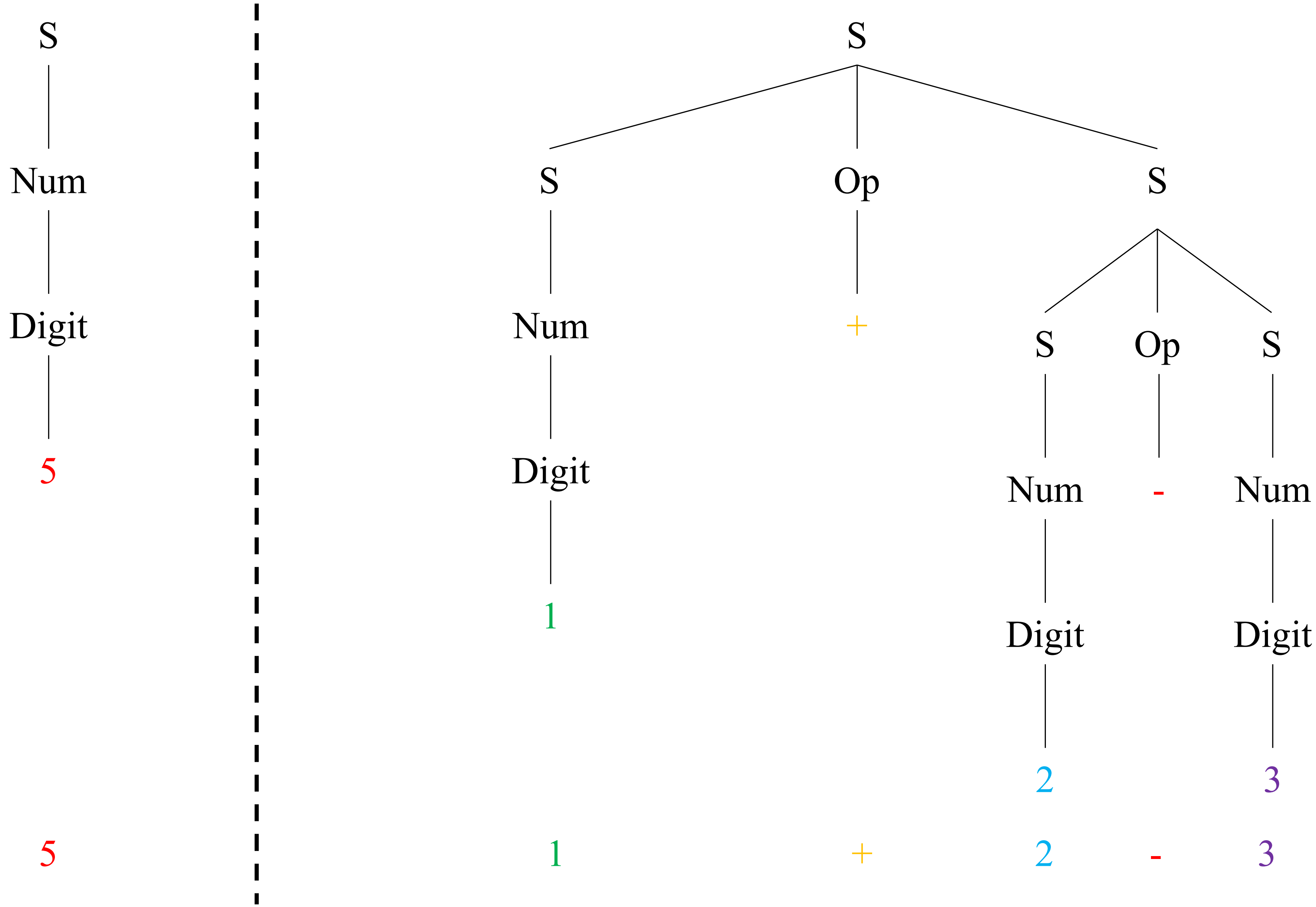
Grammar: Derivation / Parse Trees



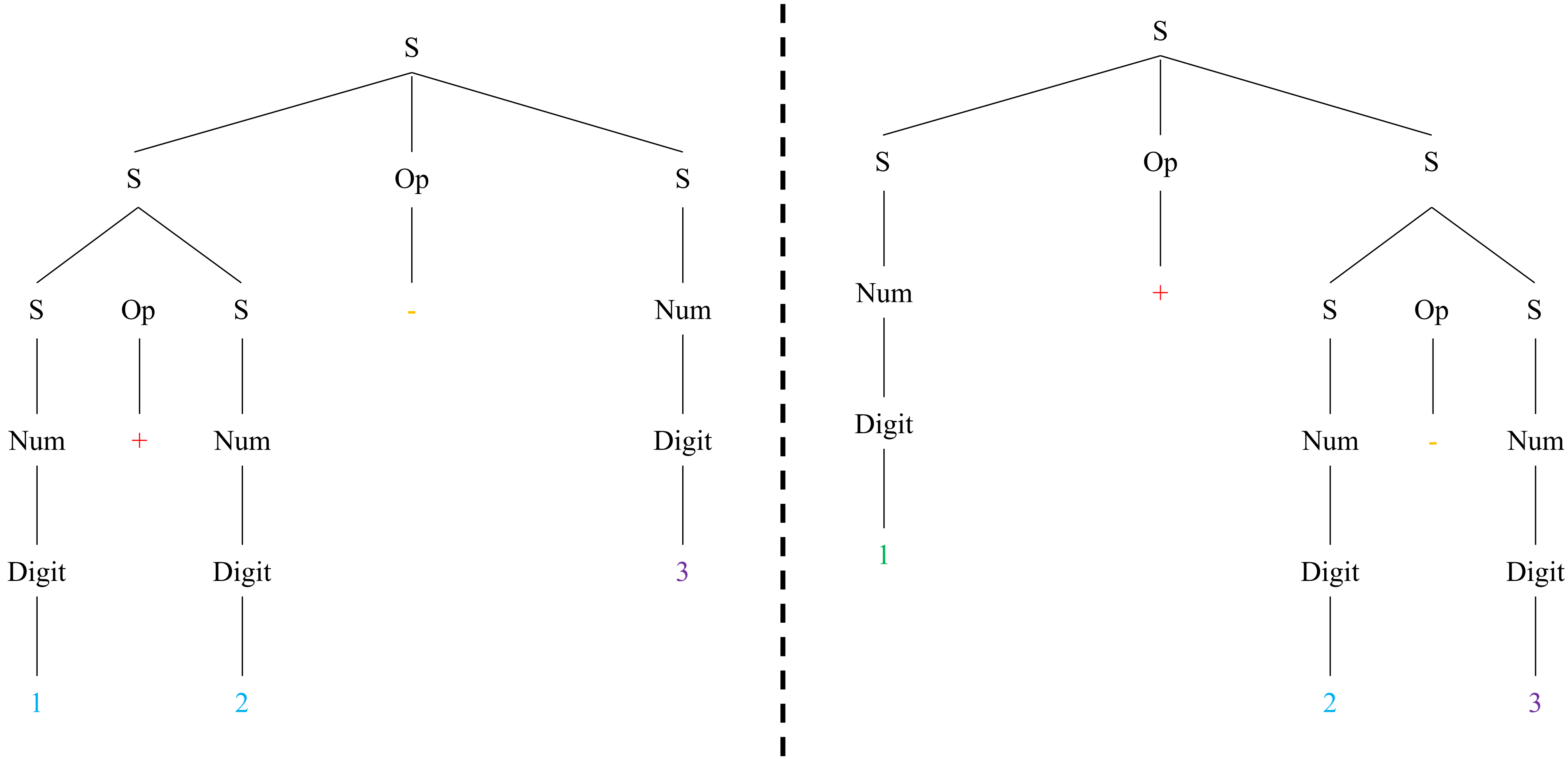
Grammar: Derivation / Parse Trees



Grammar: Derivation / Parse Trees

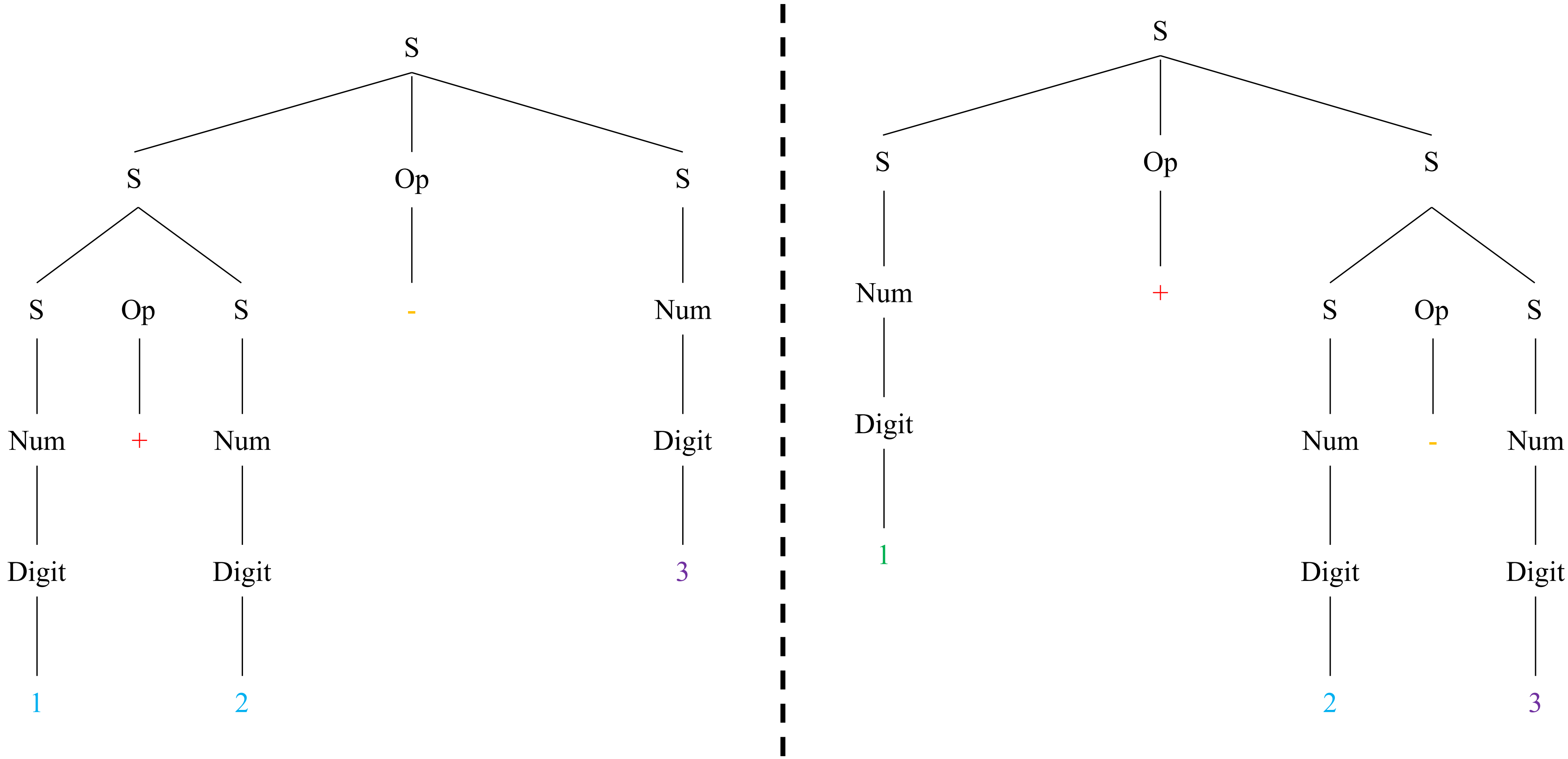


Ambiguous Grammar



A grammar is said to be ambiguous if it can generate the same string (here: 1 + 2 - 3) through multiple derivations.

Ambiguous Grammar



Derivations (generated by pre-order tree traversal):

Left: (S(S(S(Num(Digit 1)))(Op +)(S(Num(Digit 2))))(Op -)(S(Num(Digit 3))))

Right: (S(S(Num(Digit 1)))(Op +)(S(S(Num(Digit 2))(Op -)(S(Num(Digit 3))))))

Grammar Equivalence

Two grammars are equivalent if they can generate the same strings:

- **weak equivalence: generate the same strings**
- **strong equivalence: generate the same strings through the same derivations**

Grammar Equivalence: Example

Grammar A: $S \to aSB \mid ab$

Grammar B: $S \to aSB \mid aabb \mid ab$

Both can define language $a^n b^n$ (sequences of n as and n bs) for n > 0