Lecture 10: Parsing I

Xu Ruifeng

Harbin Institute of Technology, Shenzhen

Today’s Class

- Chomsky Hierarchy and Grammar
- Parsing Approaches for Context-Free Grammar (CFG)
  - Top-down Approach
  - Bottom-up Approach
- Parsing Algorithms
  - Shift-reduce Parsing
  - CYK Parsing

Syntactic Analysis

Computer and Natural Languages are very **productive.** They are also highly **regular** in character. Hence, any grammar which is generated to accept all structurally correct strings must account for this productivity and regularity.

- To describe the regularity and productivity of languages, we provide rules of **syntax**

- Our rules of syntax should specify the syntactic structure of a string in our language.

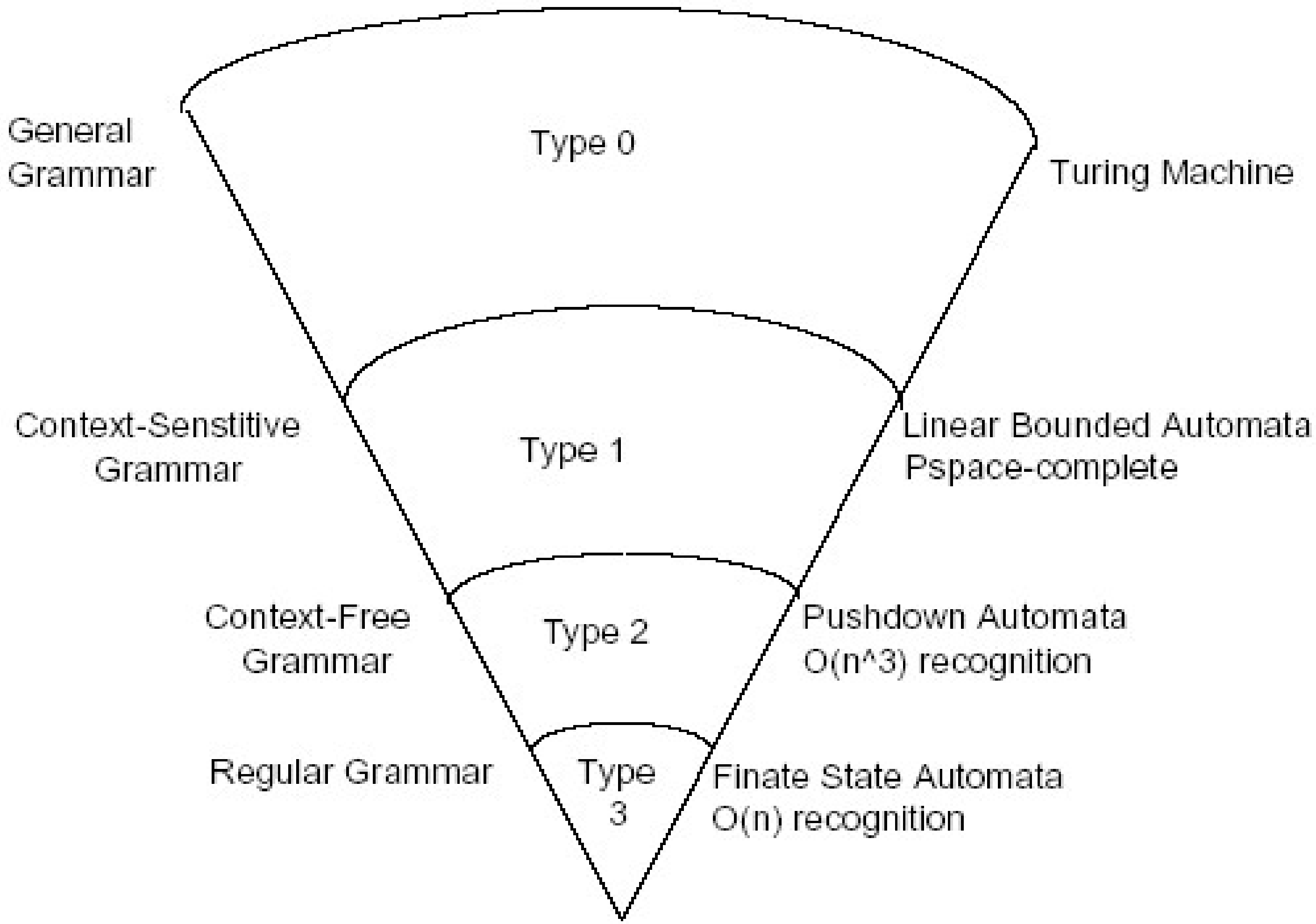
Syntactic Analysis

- To provide a syntactic analysis for a string, we must have:
- **A Grammar**: the formal specification of allowable structures.
- **A Parsing Algorithm**: a method of analyzing the sentence to determine its structure.
- **Generative grammars** can be thought of as a set of rules that generate valid phrases in a particular language. Noam Chomsky defined four classes of "complexity" for generative grammars, that are commonly organized into the **Chomsky Hierarchy**.

Generative Grammar

- A generative grammar G is expressed as G=(V, T, P, S), where:
  - V is a **finite** set of non-terminals or variables (use capital letters to designate them).
  - T is a **finite** set of terminals or tokens (use small letters to designate them).
  - P is a finite set of productions or rules of the form $\alpha \rightarrow \beta$.
  - S is a special non-terminal called the start symbol, $S \in V$.
- Normally $V \cap T = \emptyset$.
- A grammatical symbol is an element of $V \cup T$.

Chomsky Hierarchy



Regular Grammar (Type 3)

- A Regular Grammar is denoted as G=(V, T, P, S)
  - V are finite set of non-terminal variables
  - T are finite set of terminal variables
  - S are start symbol
  - P is a finite set of productions. Consist of productions like A→β
- Left linear grammar for *ab*
  - *B→Ab   A→a*

  Left hand side: one non-terminal symbol

  Right hand side: empty string / a terminal symbol/a non-terminal sysbol following a terminal symbol
- Right linear grammar
  - *A→aB   B→b*

Context-free Grammars (Type 2)

- A Context-free Grammar is denoted as G=(V, T, P, S)
  - V are finite set of non-terminal variables
  - T are finite set of terminal variables
  - S are start symbol
  - P is a finite set of productions. Consist of productions like $A \rightarrow \alpha$
  - Only one non-terminal symbol on the left hand side
- Push down Automata

Example

- Example Grammar, G:

  V = {S, VP, NP, Name, Det, Noun, Pro, Verb}

  T = {*Nyssa, chases, cat, cats, the, he, she*}

  S = {S}

  P = {S $\rightarrow$ NP VP
  VP $\rightarrow$ Verb NP
  NP $\rightarrow$ Name | Det Noun | Pro
  Name $\rightarrow$ *Nyssa*
  Det $\rightarrow$ *the*
  Noun $\rightarrow$ *cat* | *cats*
  Pro $\rightarrow$ *he* | *she*
  Verb $\rightarrow$ *chases*}

- To generate phrases in the language of grammar G, repeatedly apply productions to non-terminals beginning with the start symbol.

- If we apply $A \rightarrow \beta$ to the string $\alpha A \gamma$, we obtain $\alpha\beta\gamma$, obtaining a string in the language generated by G, L(G).

- L(G) is the set of strings such that:

1. Each string consists **solely** of terminals.
2. Each string is derived from S by applying the productions in P. Below is a string of derivations: $\alpha_1 \rightarrow \alpha_2 \rightarrow \alpha_3 \rightarrow ... \rightarrow \alpha_n$

- An important aspect of the CFGs is that the derivations involve variables that are independent of what surrounds them; hence, they are context independent.

- Importance of CFGs:

1. They are powerful enough to handle much of the productivity of computer and natural languages.

2. They are restrictive enough to construct efficient parsers, $O(n^3)$ generally, $O(n)$ for deterministic grammars (generally true for computer languages).

Context-Sensitive Grammars (Type 1)

- A Context-Sensitive Grammar is denoted as G=(V, T, P, S)
  - V are finite set of non-terminal variables
  - T are finite set of terminal variables
  - S are start symbol
  - P is a finite set of productions. Consist of productions like $\alpha \rightarrow \beta$
  - α and β are strings of symbol in (VUT)* and β is at least as long as α
  - NO productions of AB→C or AB→empty
  - $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$ where β can't be empty "A becomes β in the context of α_1 and α_1"

General Grammar (Type 0)

- A General Grammar is denoted as G=(V, T, P, S)
  - V are finite set of non-terminal variables
  - T are finite set of terminal variables
  - S are start symbol
  - P is a finite set of productions WITHOUT RESTRICATIONS
- Type 0 grammar are recursively enumerable and are accepted by a Turing Machine.
- Too complex a speciation for computer languages.

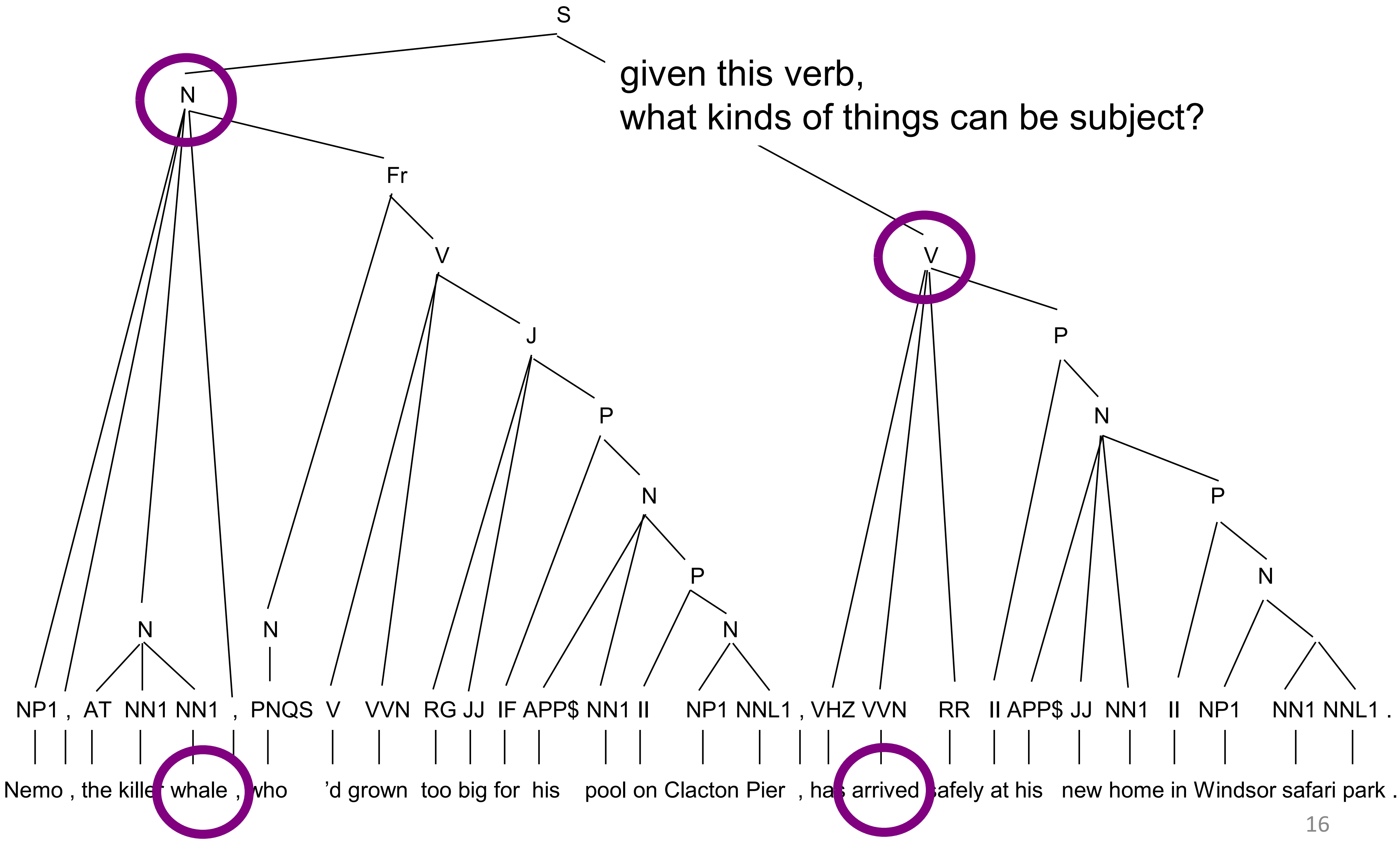
Parsing

- Parsing is the process of determining whether a string is in a grammar G.
- Without context, ambiguities occurs in natural language frequently. With more contextual information, most ambiguities can be removed
- The core of parsing natural language is to resolve the ambiguities in the syntactic structures.
- The goal of sentence processing is to understand the representations people form as they understand a sentence (syntax, meaning…).

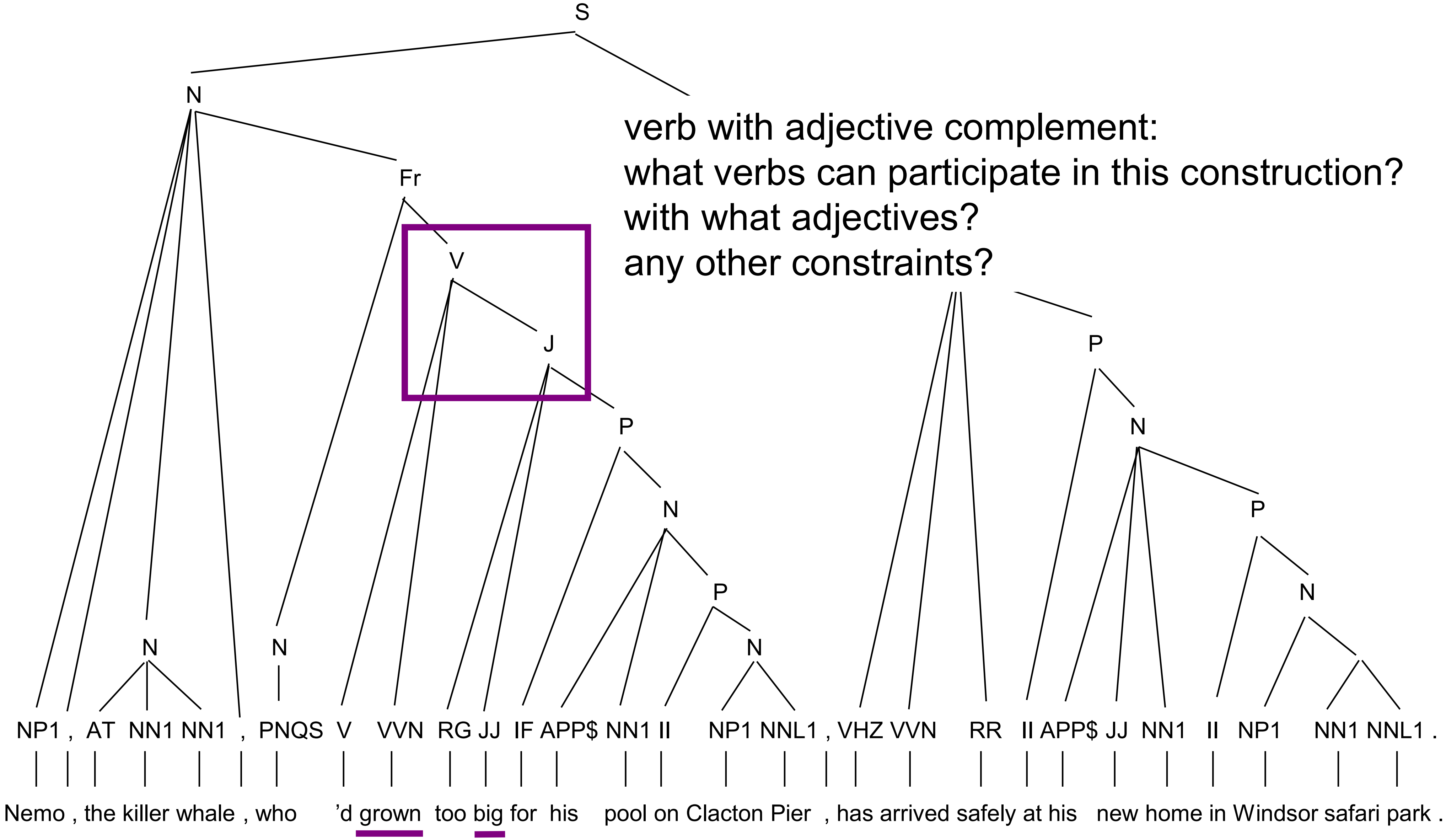
Parsing

- Parsing adds information about sentence structure and constituents
- Allows us to see what constructions words enter into
  - eg, transitivity, passivization, argument structure for verbs
- Allows us to see how words function relative to each other
  - eg, what words can modify / be modified by other words

[S[N Nemo_NP1 ,_, [N the_AT killer_NN1 whale_NN1 N] ,_, [Fr[N who_PNQS N][V 'd_VHD grown_VVN [J too_RG big_JJ [P for_IF [N his_APP$ pool_NN1 [P on_II [N Clacton_NP1 Pier_NNL1 N]P]N]P]J]V]Fr]N] ,_, [V has_VHZ arrived_VVN safely_RR [P at_II [N his_APP$ new_JJ home_NN1 [P in_II [N Windsor_NP1 [safari_NN1 park_NNL1]N]P]N]P]V] ._. S]



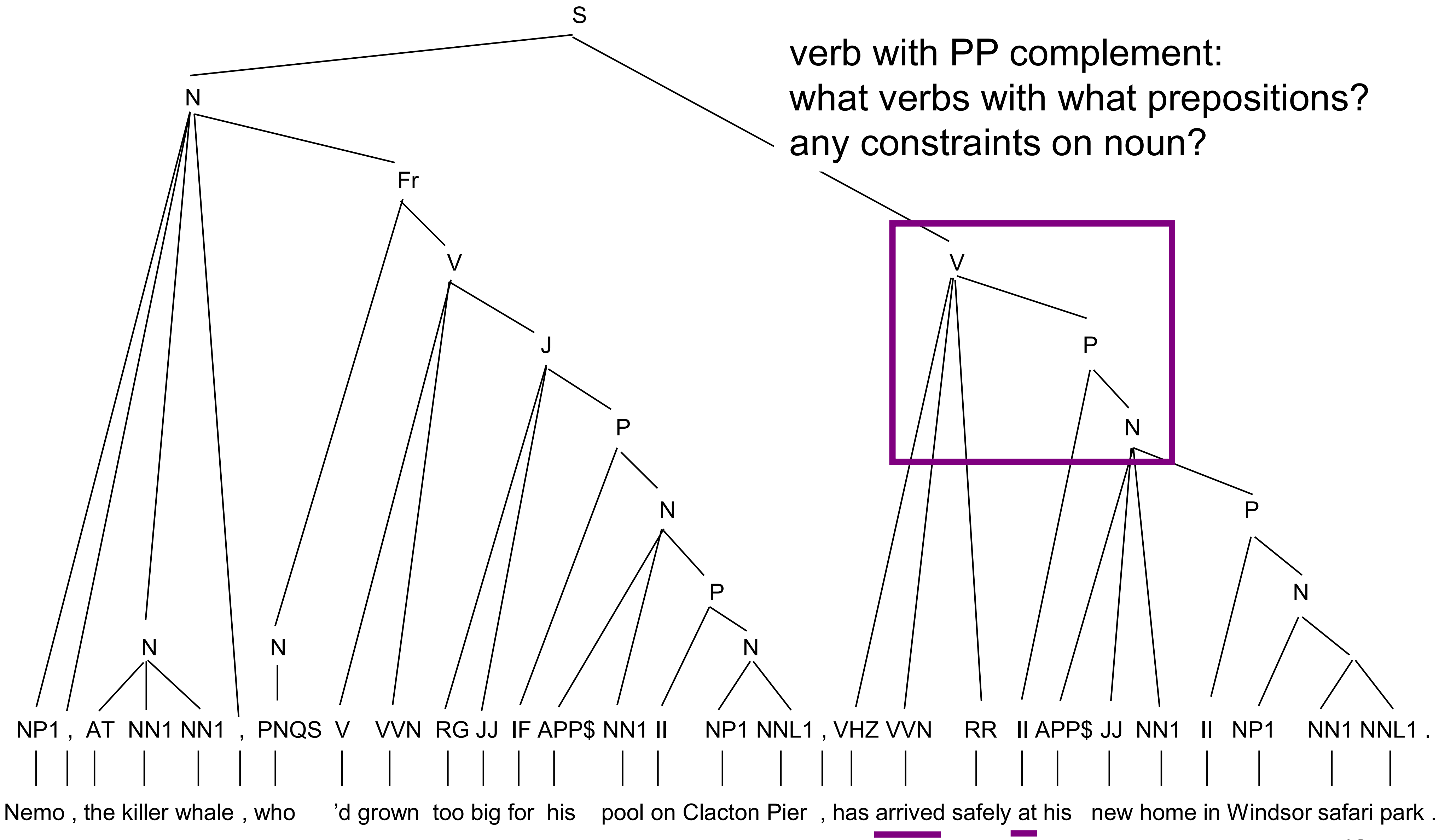
[S[N Nemo_NP1 ,_, [N the_AT killer_NN1 whale_NN1 N] ,_, [Fr[N who_PNQS N][V 'd_VHD grown_VVN [J too_RG big_JJ [P for_IF [N his_APP$ pool_NN1 [P on_II [N Clacton_NP1 Pier_NNL1 N]P]N]P]J]V]Fr]N] ,_, [V has_VHZ arrived_VVN safely_RR [P at_II [N his_APP$ new_JJ home_NN1 [P in_II [N Windsor_NP1 [safari_NN1 park_NNL1]N]P]N]P]V] ._. S]



verb with adjective complement:
what verbs can participate in this construction?
with what adjectives?
any other constraints?

[S[N Nemo_NP1 ,_, [N the_AT killer_NN1 whale_NN1 N] ,_, [Fr[N who_PNQS N][V 'd_VHD grown_VVN [J too_RG big_JJ [P for_IF [N his_APP$ pool_NN1 [P on_II [N Clacton_NP1 Pier_NNL1 N]P]N]P]J]V]Fr]N] ,_, [V has_VHZ arrived_VVN safely_RR [P at_II [N his_APP$ new_JJ home_NN1 [P in_II [N Windsor_NP1 [safari_NN1 park_NNL1]N]P]N]P]V] ._. S]

verb with PP complement:
what verbs with what prepositions?
any constraints on noun?



Parsing: difficulties

- Besides lexical ambiguities (usually resolved by tagger), language can be structurally ambiguous
  - global ambiguities due to ambiguous words and/or alternative possible combinations
  - local ambiguities, especially due to attachment ambiguities, and other combinatorial possibilities
  - sheer weight of alternatives available in the absence of (much) knowledge

Parsing: difficulties

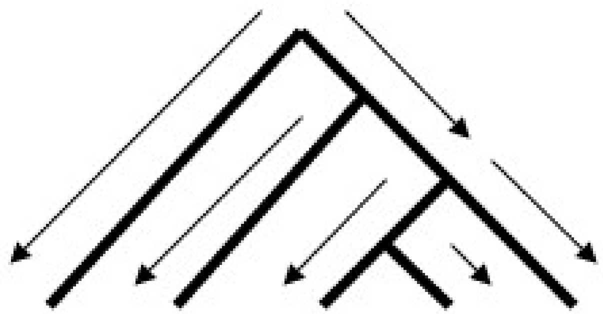
- Broad coverage necessary for parsing corpora of real text
- Long sentences:
  - structures are very complex
  - ambiguities proliferate
- Difficulty (even for human) to verify if parse is correct
  - because it is complex
  - because it may be genuinely ambiguous

Parsing

- Parsing Approaches
  - Top down parsers
  - Bottom up parsers
  - Left corner parsers

Top-down Parsing

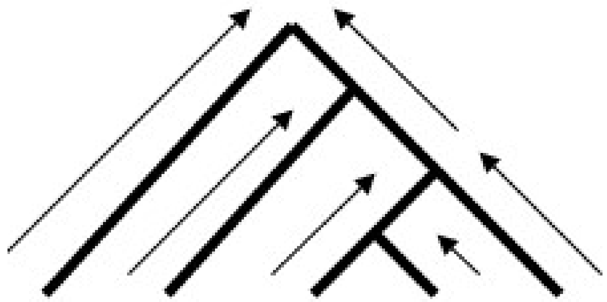
- Begin with the start symbol S and produce the right hand side (RHS) of the rule
- Match the left-hand side (LHS) of CFG rules to the non-terminals in the string, replacing them with the right-hand side of the rule.
- Continue until all the non-terminals are replaced by terminals, such that they correspond to the symbols in the sentence.
- The parse succeeds when all words in the sentence are generated.



- This is a left-most derivation for the sentence; each production is applied to the left-most non-terminals
- Based on Prediction
  - Produce the prediction for following component, then verify the prediction by matching words
  - If the prediction is verified, the target string will parsed according to the prediction
  - If no verified, use other predictions (backtracking)
  - If all predictions are rejected, the sentence is not valid → parser fails

Bottom-up Parsing

- Begin with Words.
- Apply the right-hand side RHS of the rules to the words to generate non-terminals from the LHS.
- The parse succeeds when the start non-terminal is generated.



- Based on Reduction
  - Match the words in the sentence
  - Generate possible non-terminal symbols iteratively
  - If S is generated, Parse succeeds
  - If S is not generated , use other symbols (backtracking)

Rule Set

Rule

(7) NP → R

(8) NP → N

(9) NP → Sϕ de

(10) VPϕ → V V

(11) VP → V NP

(12) Sϕ → NP VPϕ

(13) S → NP VP

Dictionary

(1) R→我

(2) N →县长

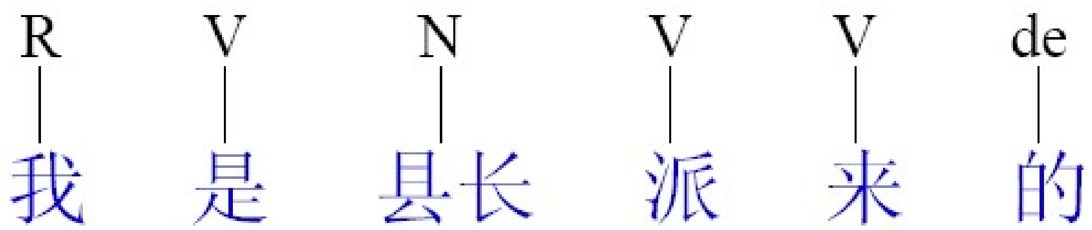
(3) V →是

(4) V →派

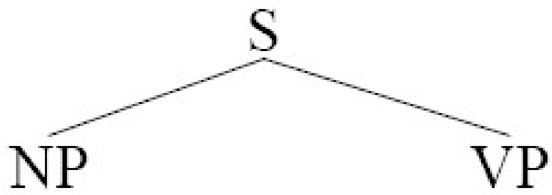
(5) V →来

(6) de →的

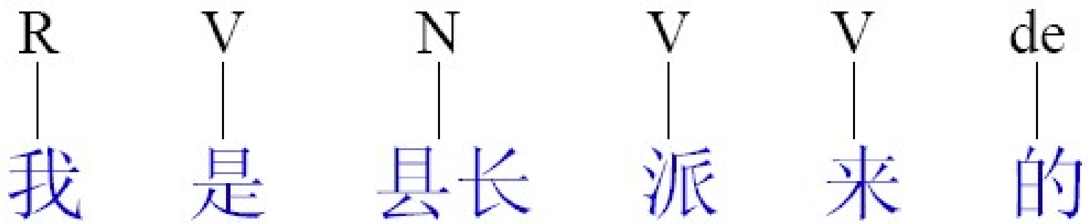
Top-Down Parsing Illustration -1



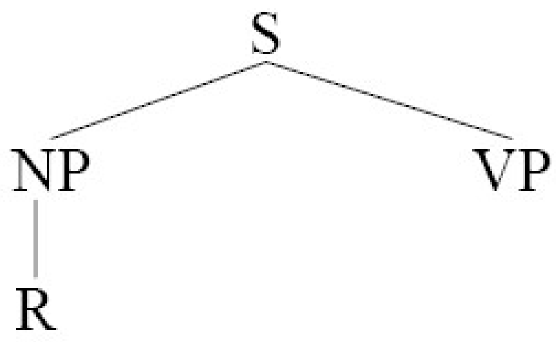
2



S → NP VP



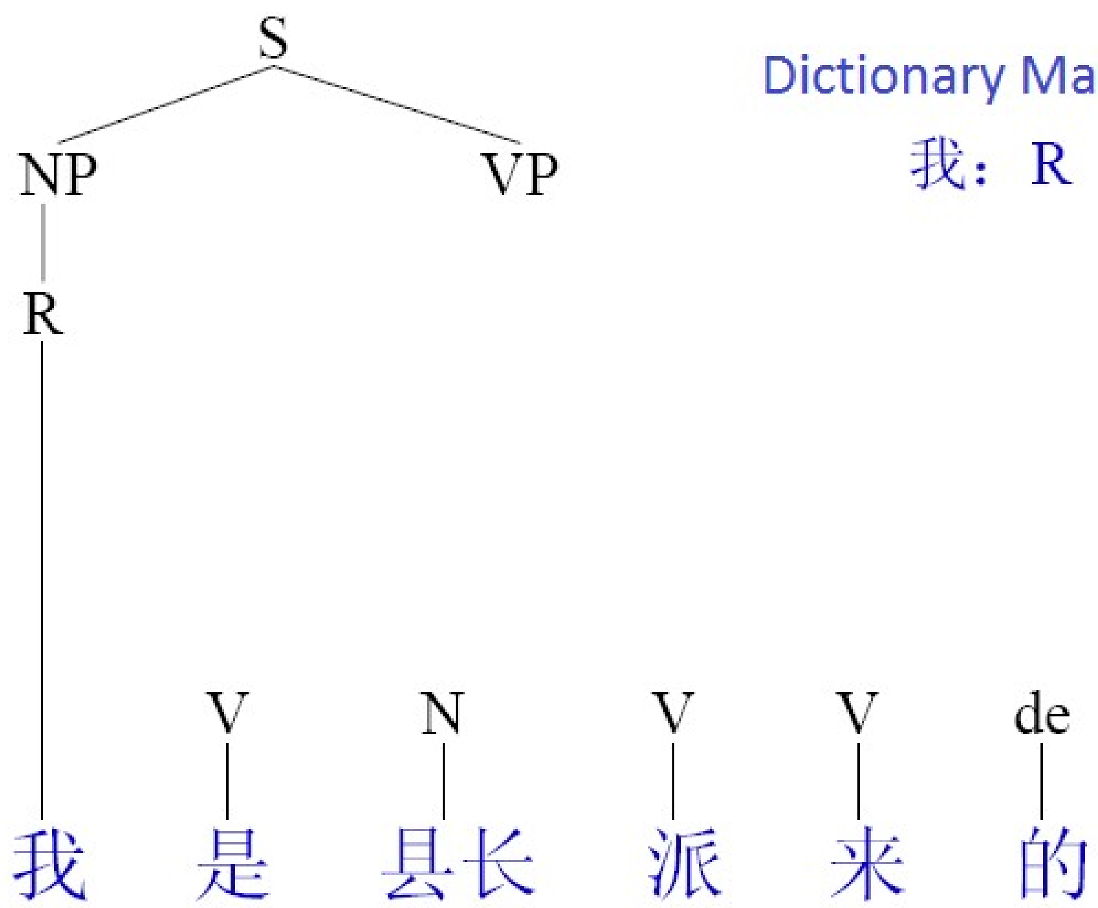
3



NP→R

R	V	N	V	V	de
我	是	县长	派	来	的

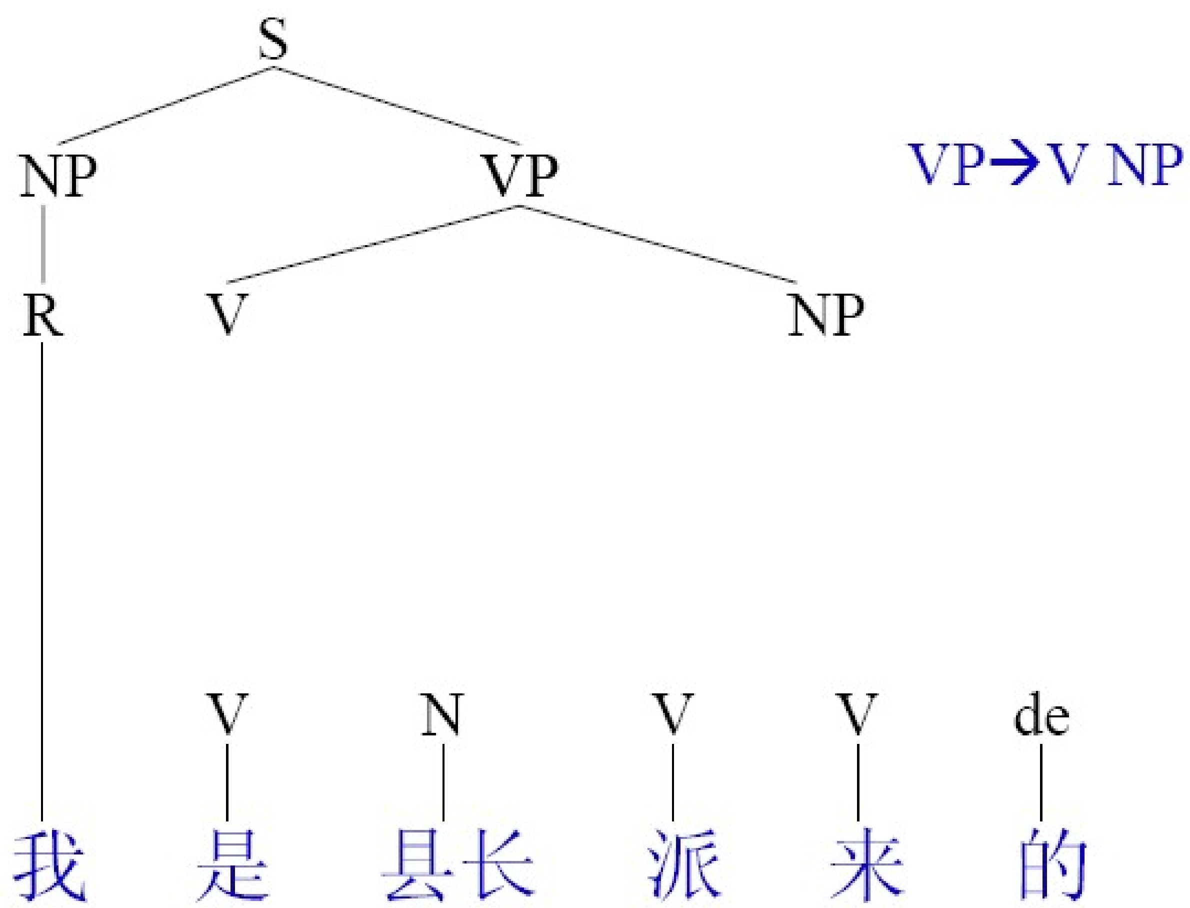
4



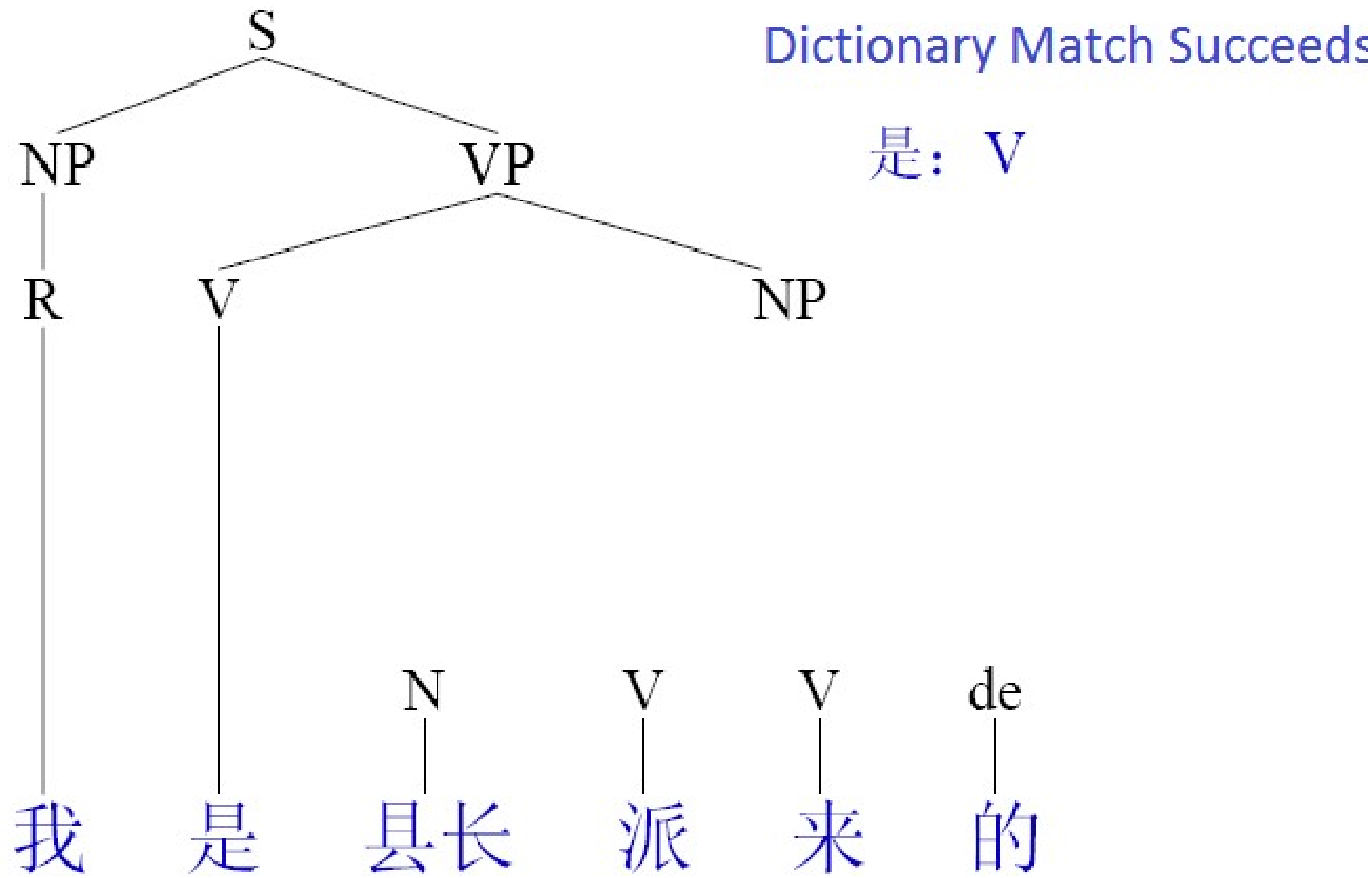
Dictionary Match Succeeds

我：R

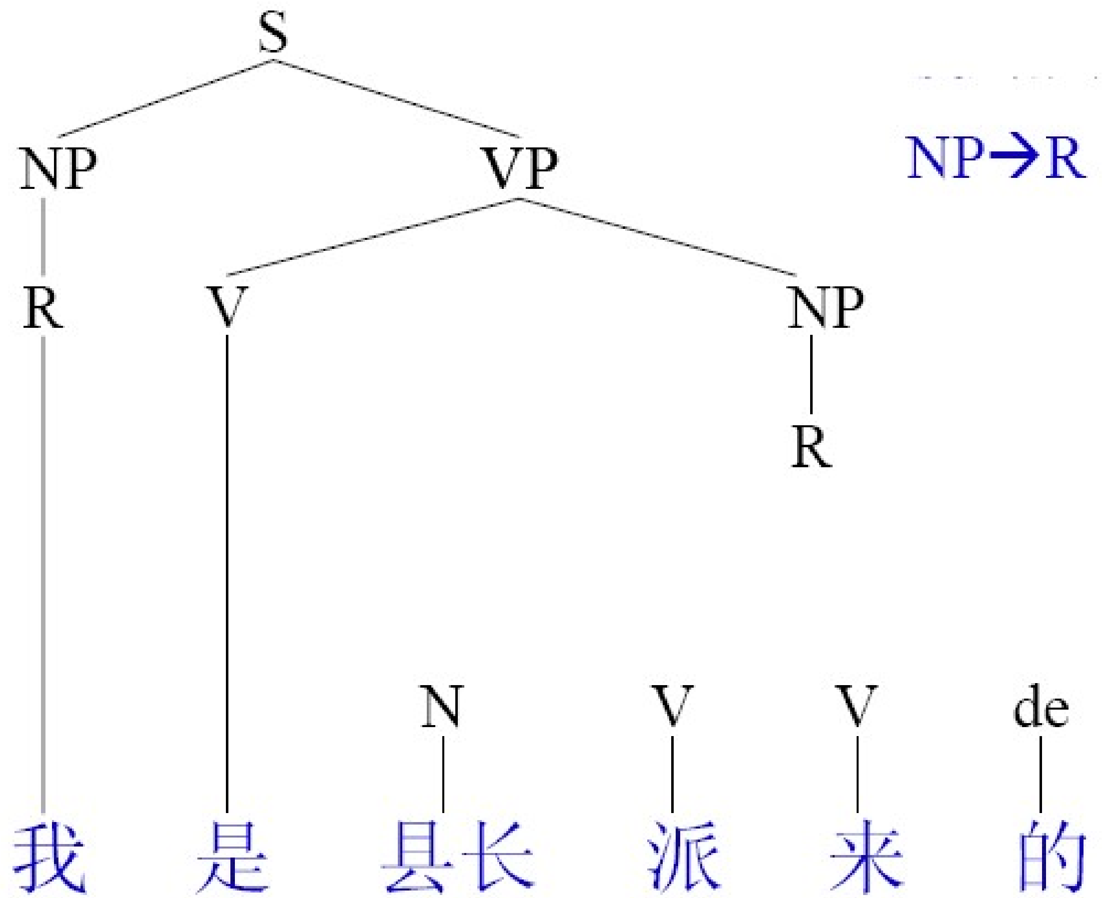
5



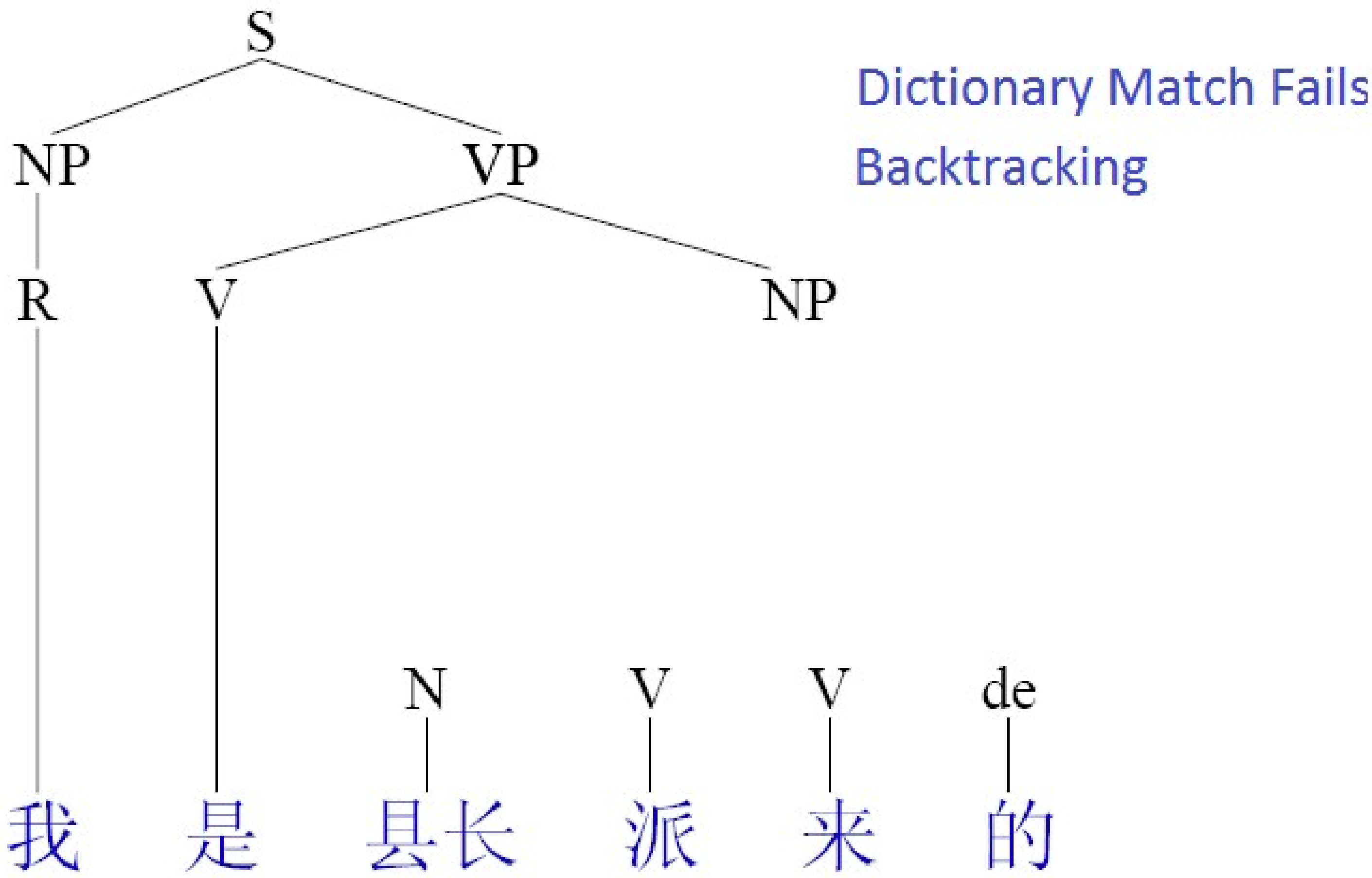
6



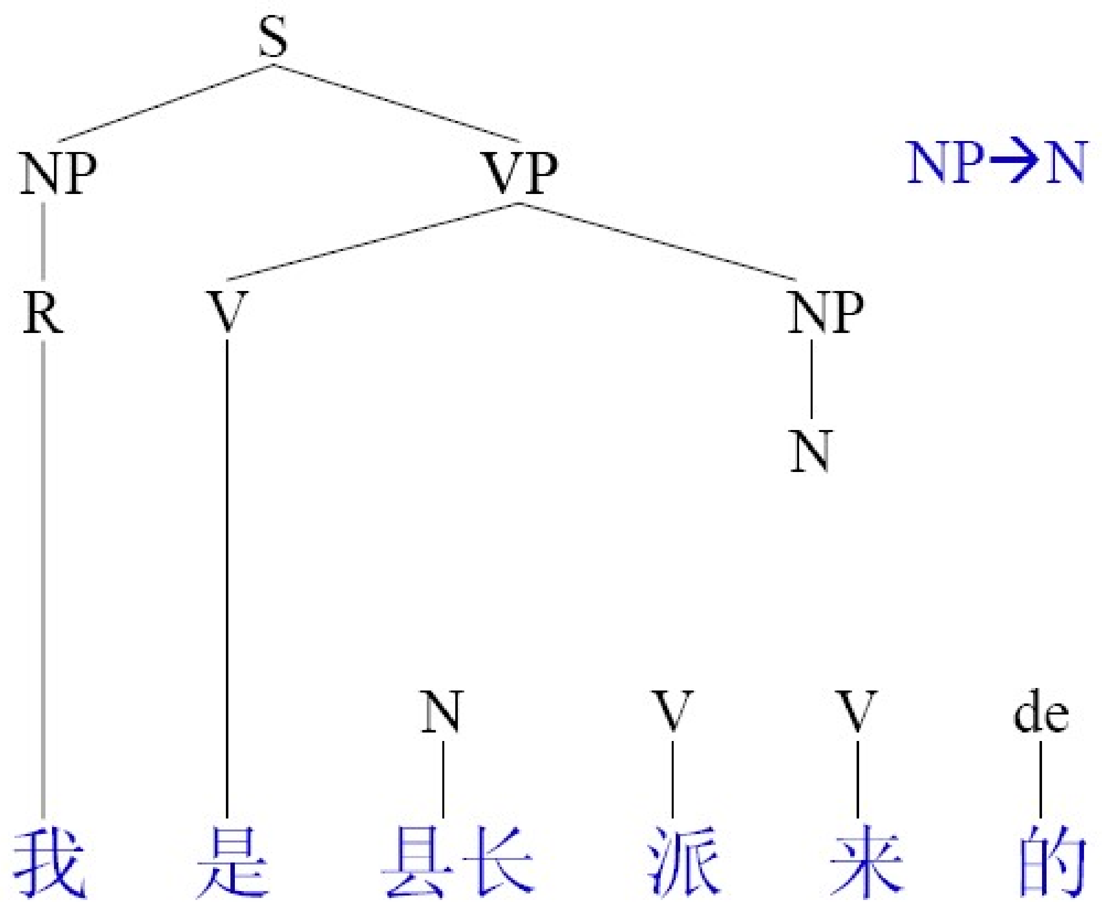
7



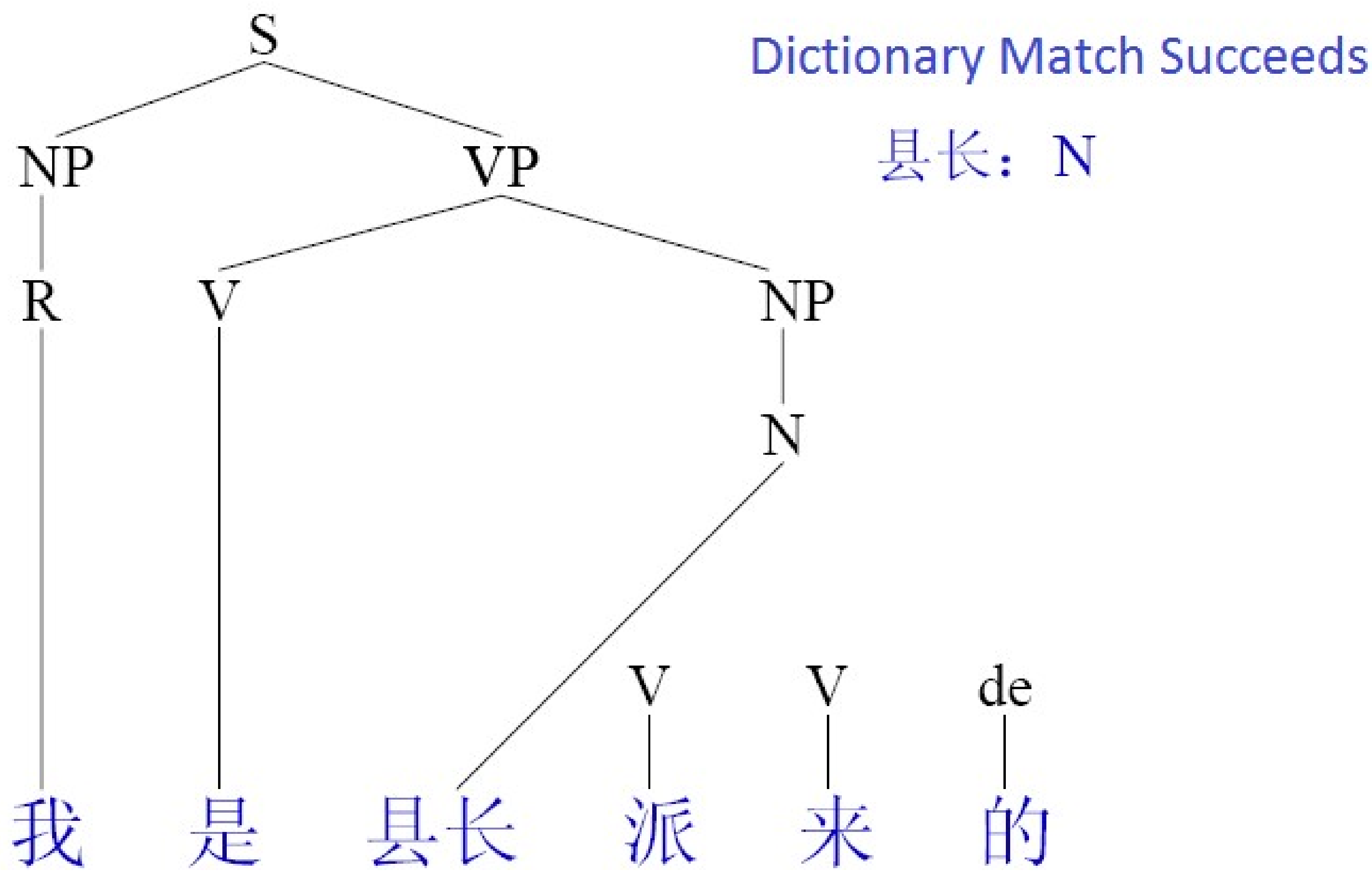
8


S
NP
VP
R
V
NP
N
V
V
de
我
是
县长
派
来
的
Dictionary Match Fails
Backtracking

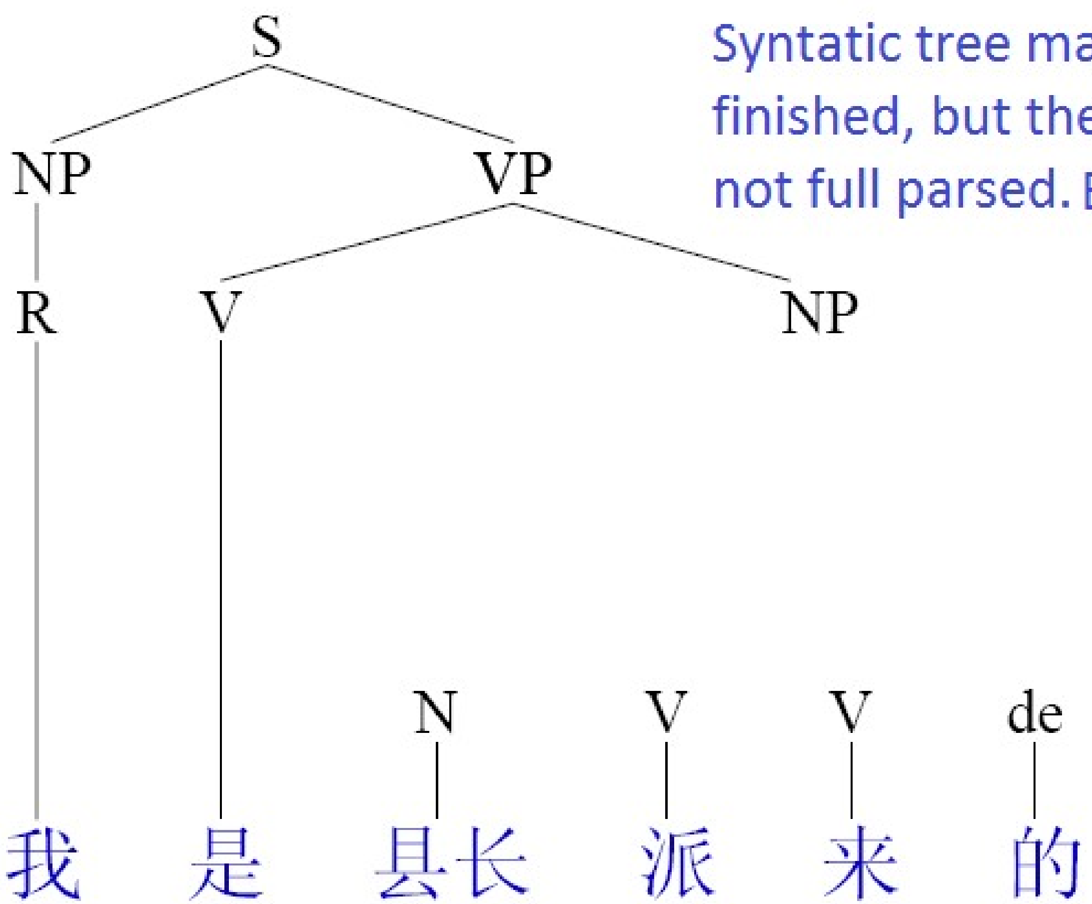
9



10

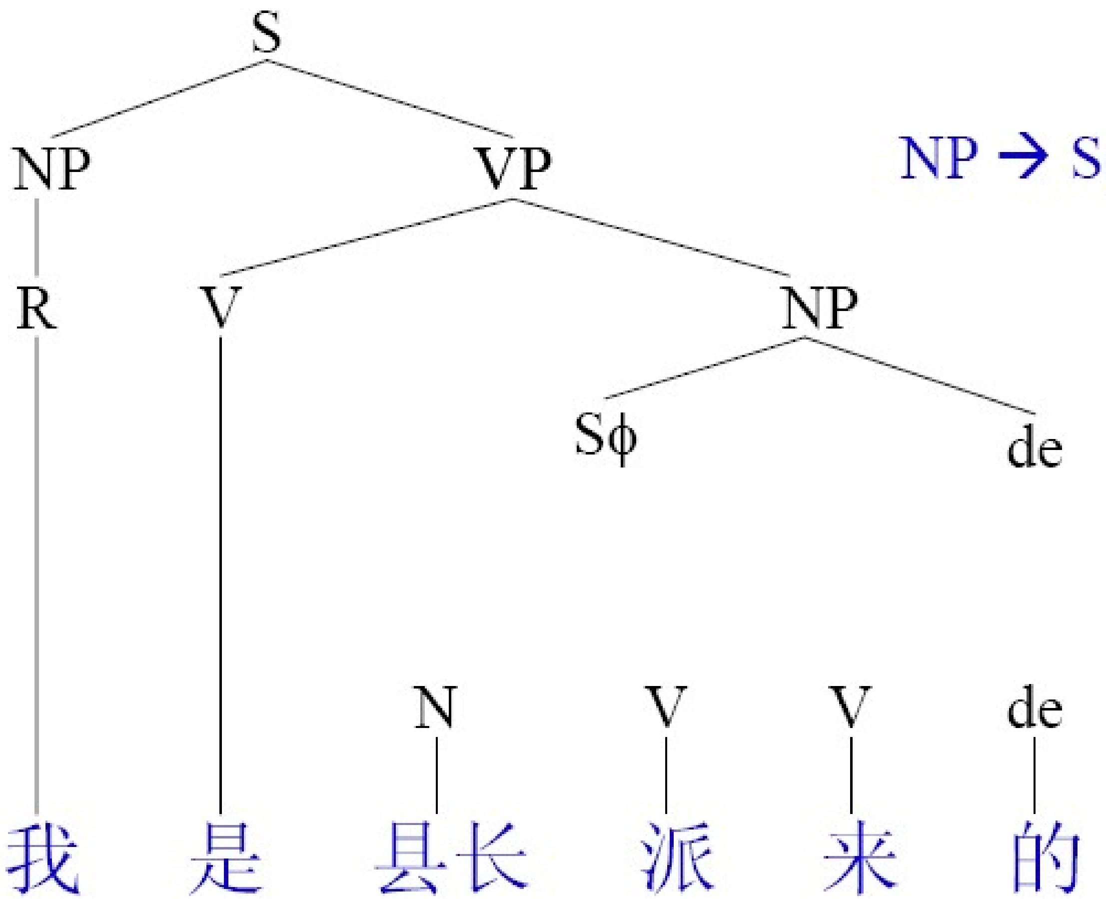


11



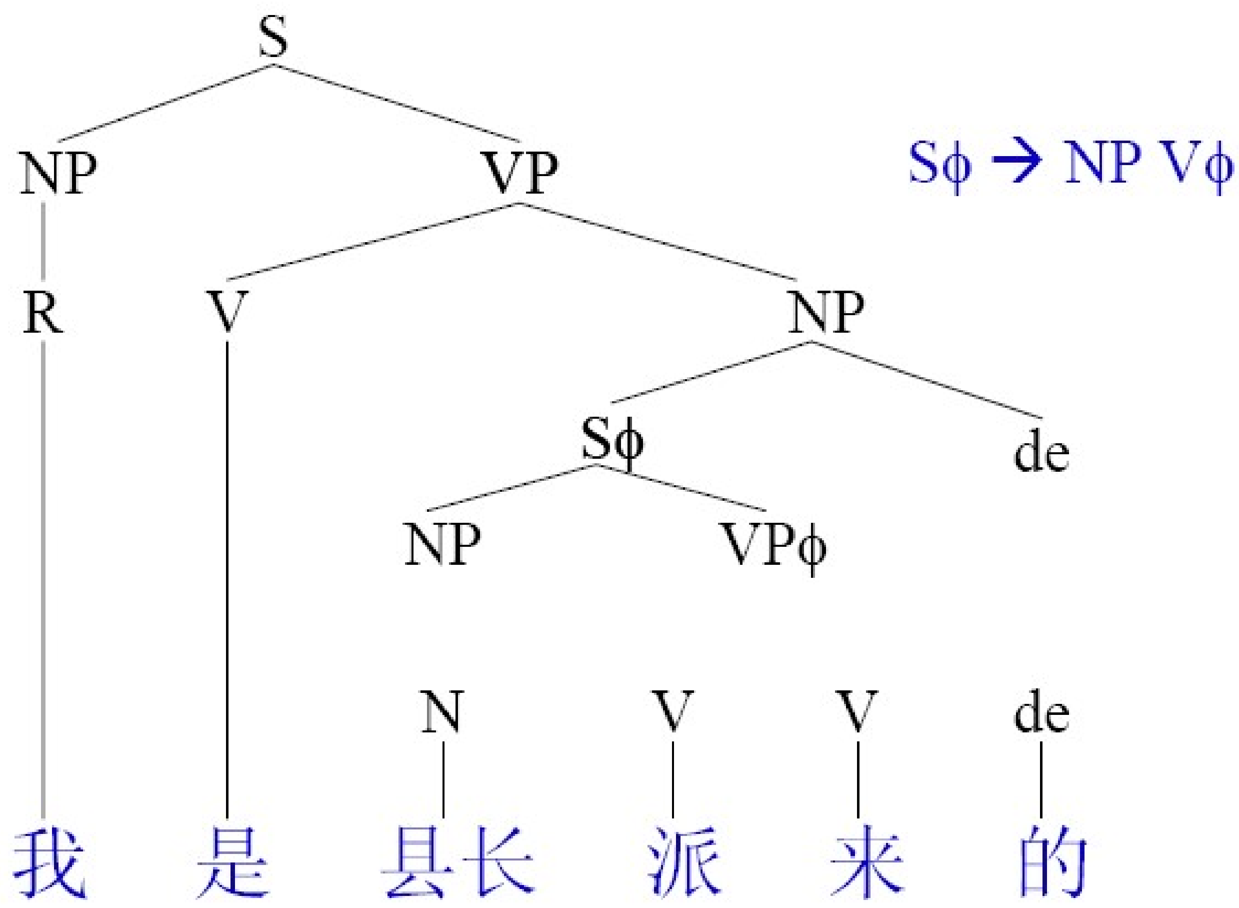
Syntatic tree matching finished, but the sentence is not full parsed. Backtracking

12

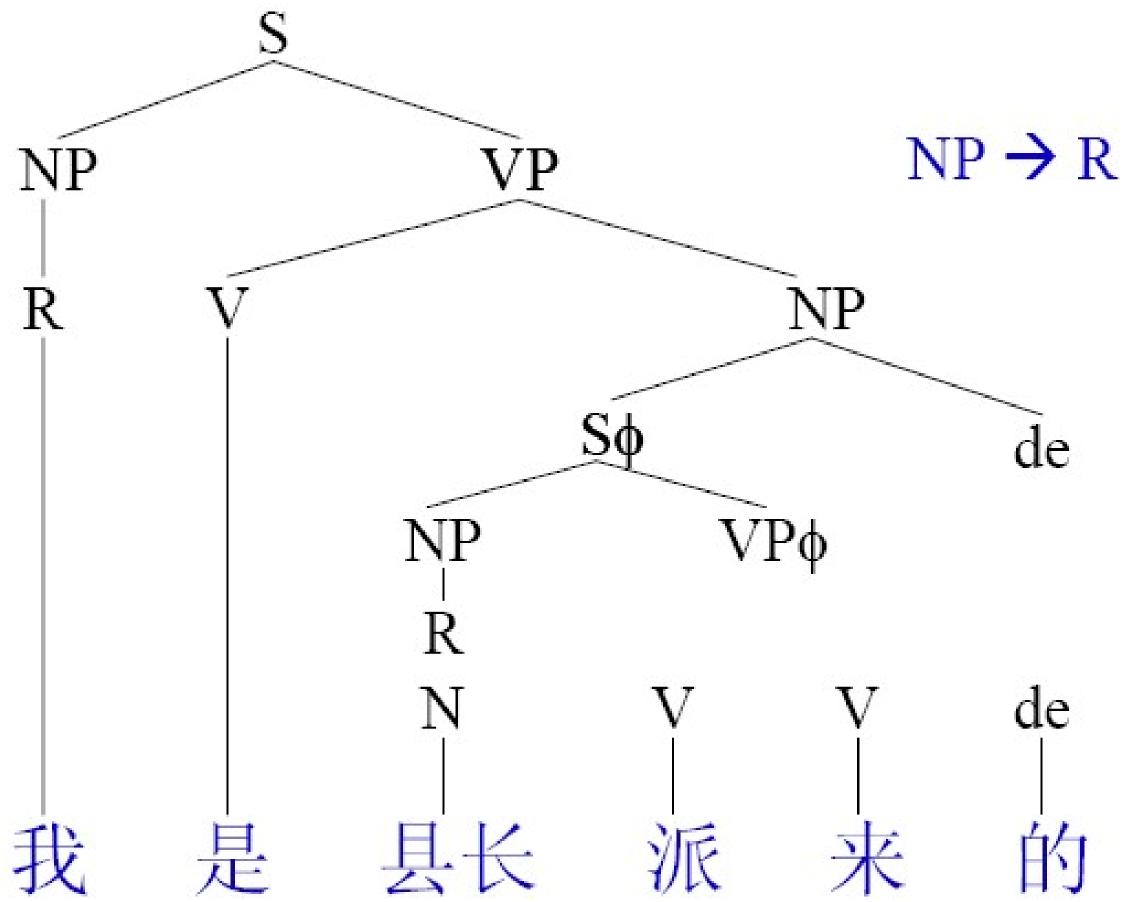


NP → Sϕ de

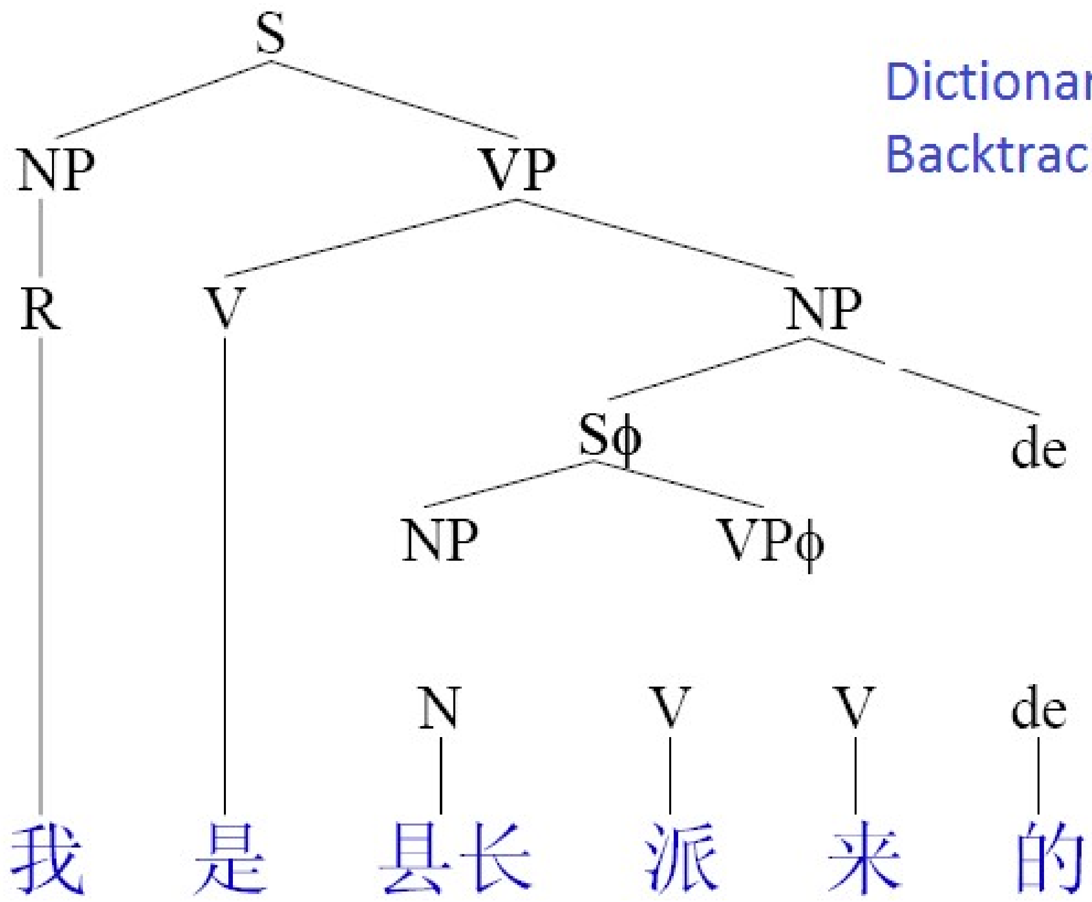
13



14

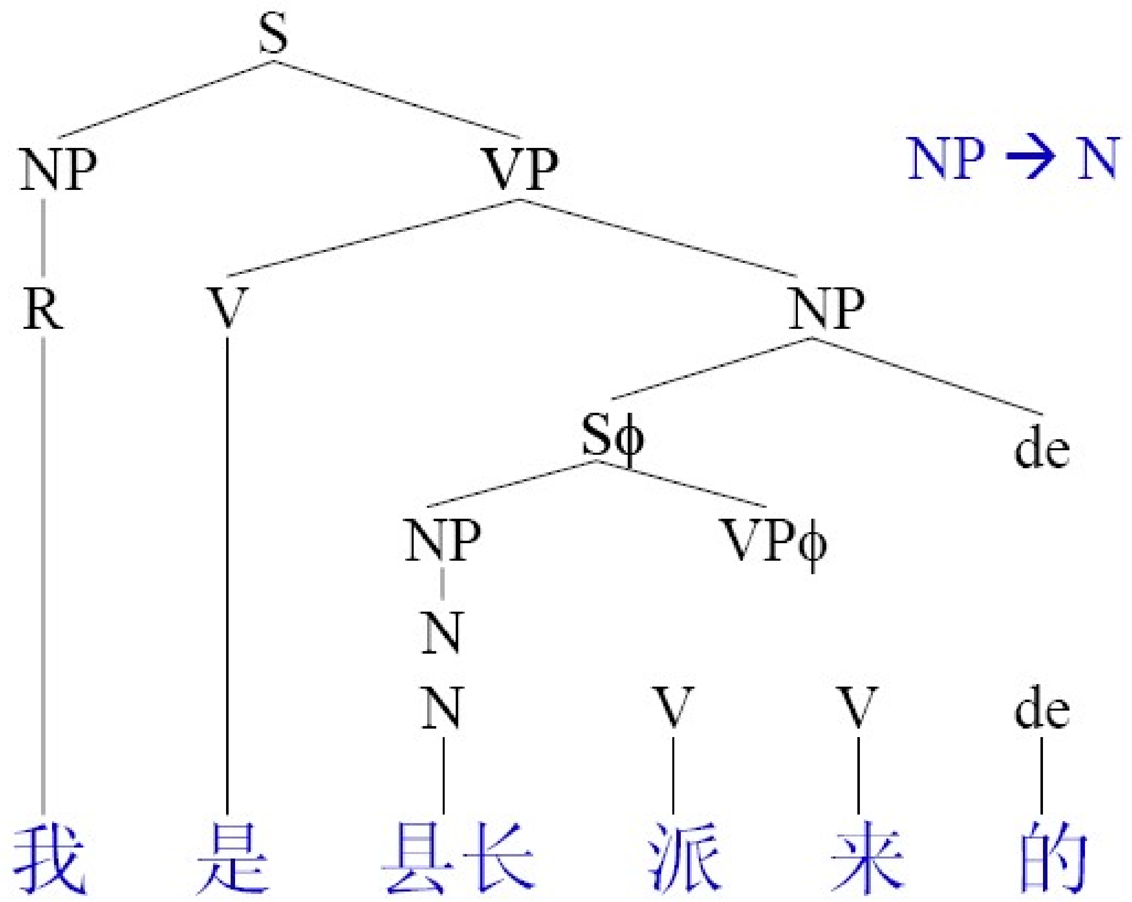


15

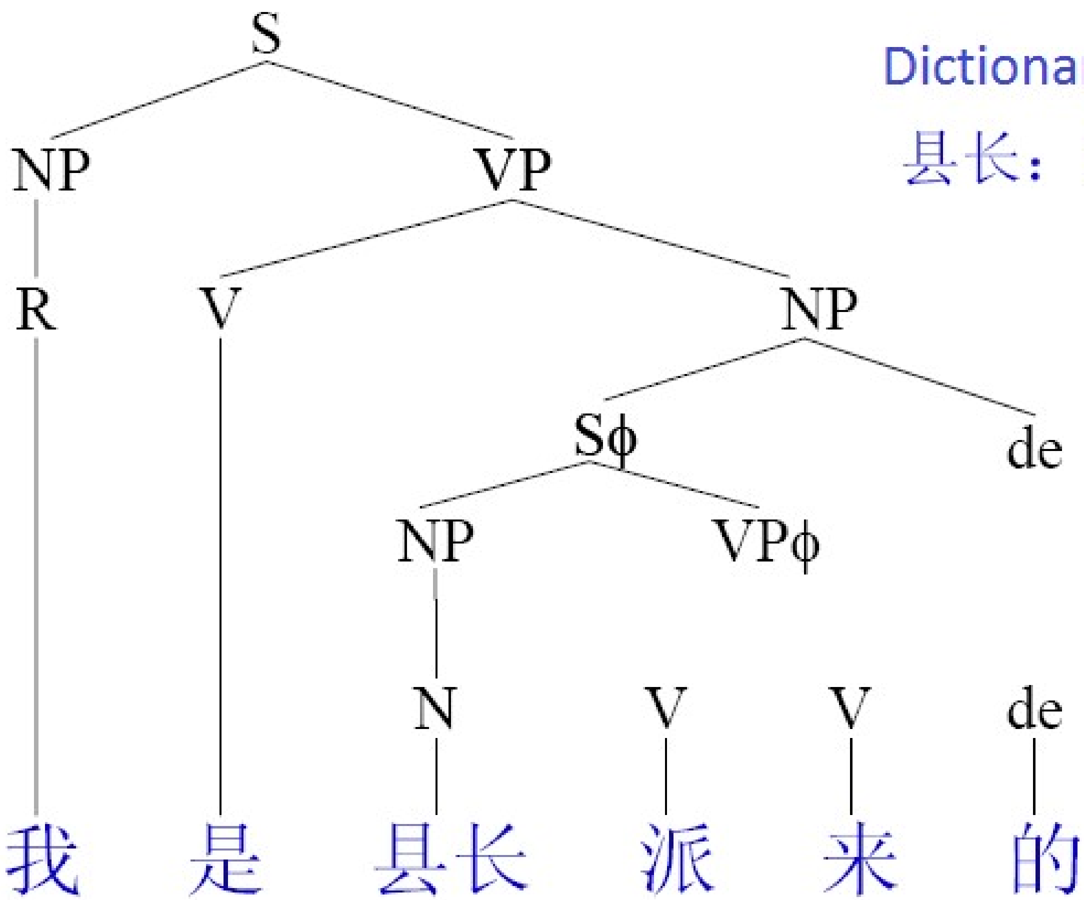


Dictionary match fails
Backtracking

16



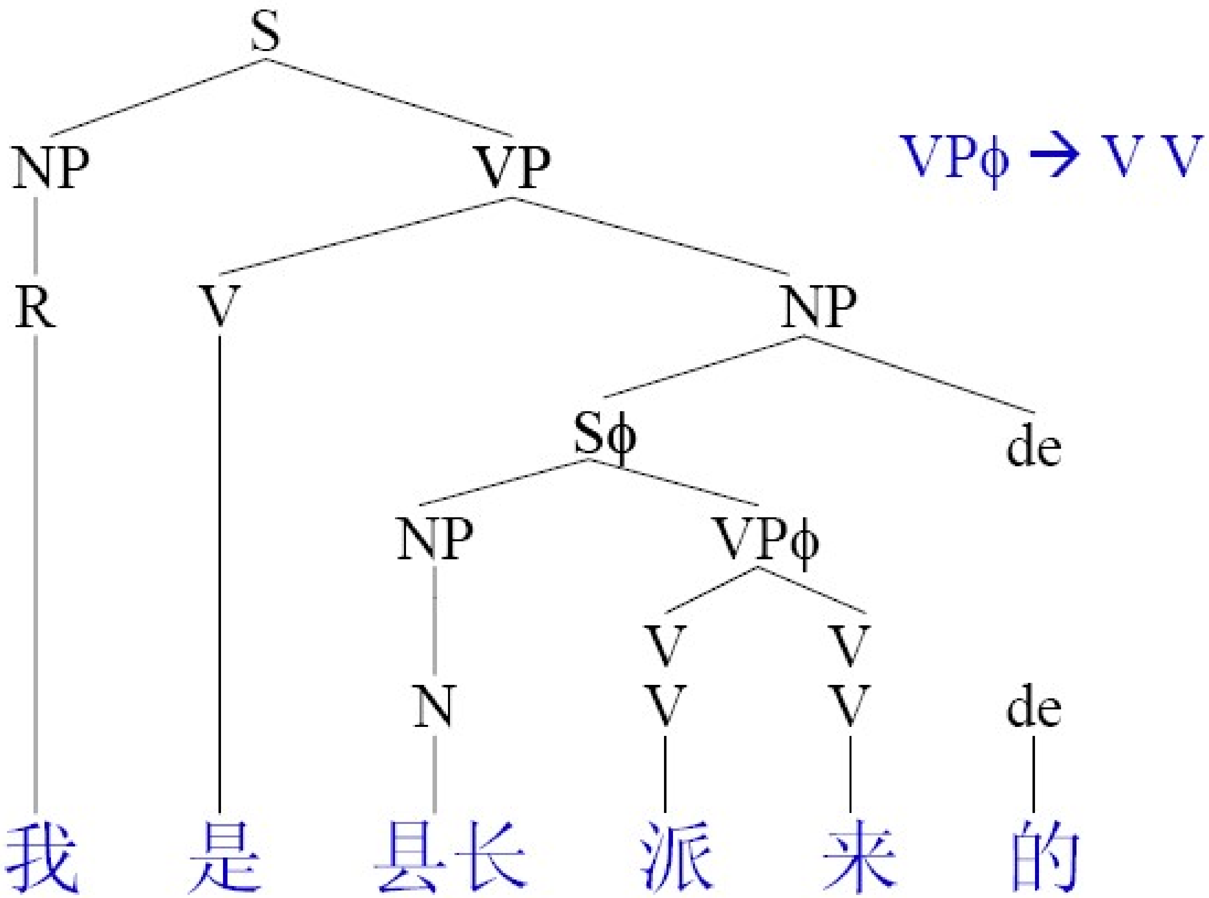
17



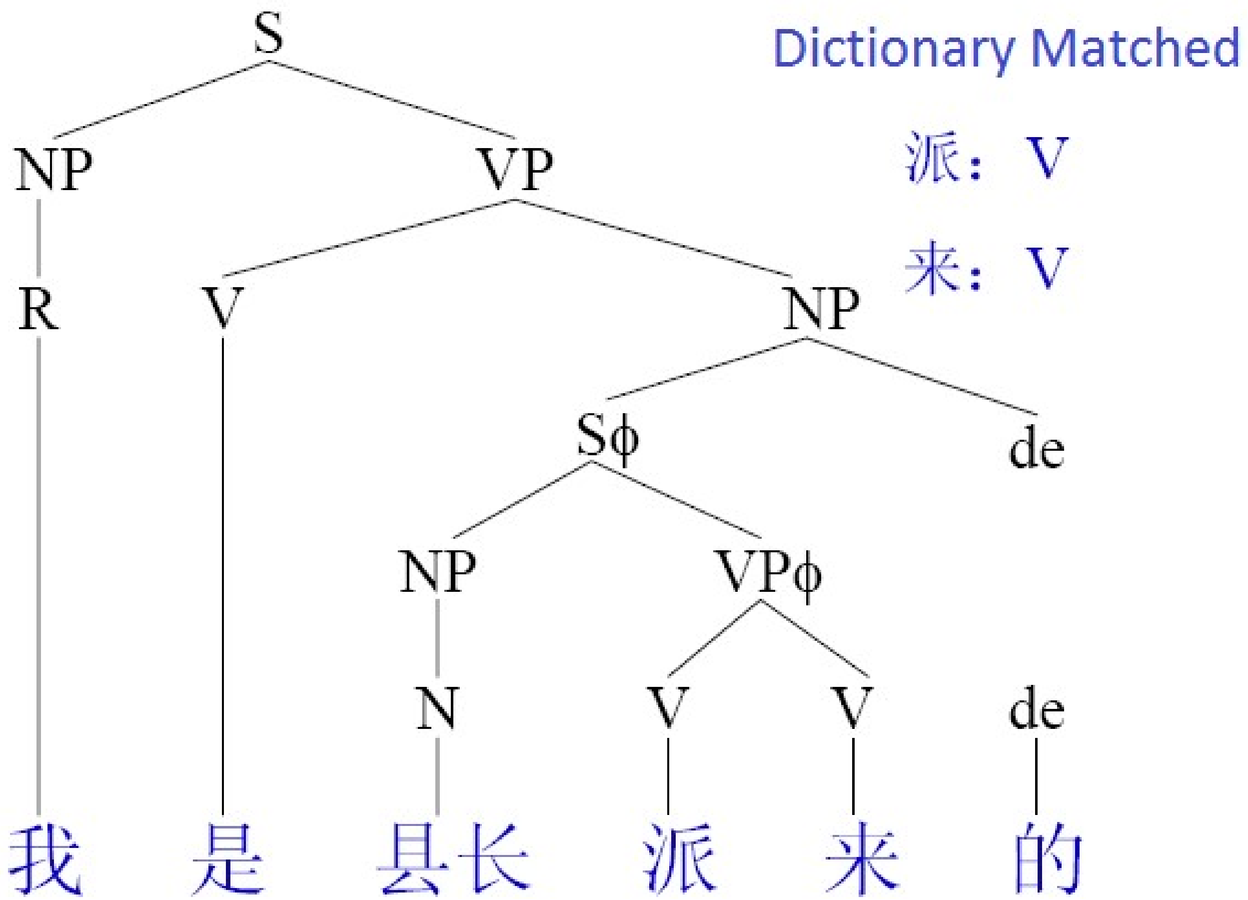
Dictionary matched

县长：N

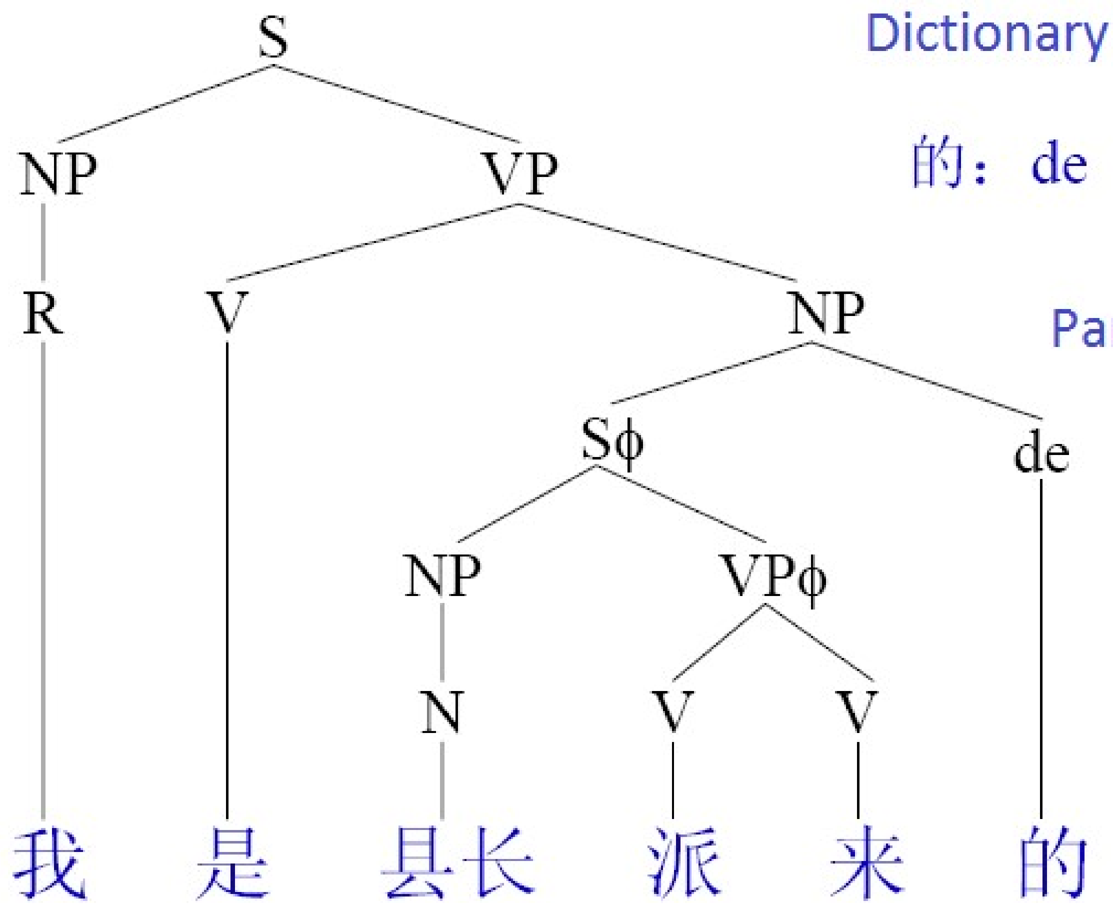
18



19


S
NP
VP
R
V
NP
Sϕ
de
NP
VPϕ
N
V
V
de
我
是
县长
派
来
的
Dictionary Matched
派：V
来：V

20

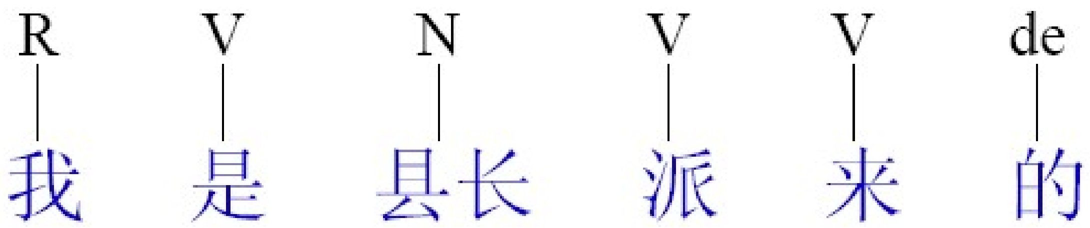


Dictionary Matched

的：de

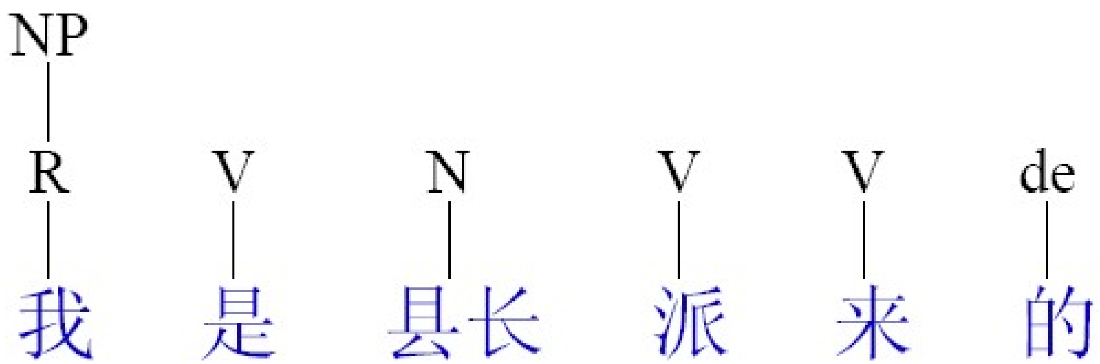
Parse Succeeds

Bottom-Up Parsing Illustration -1



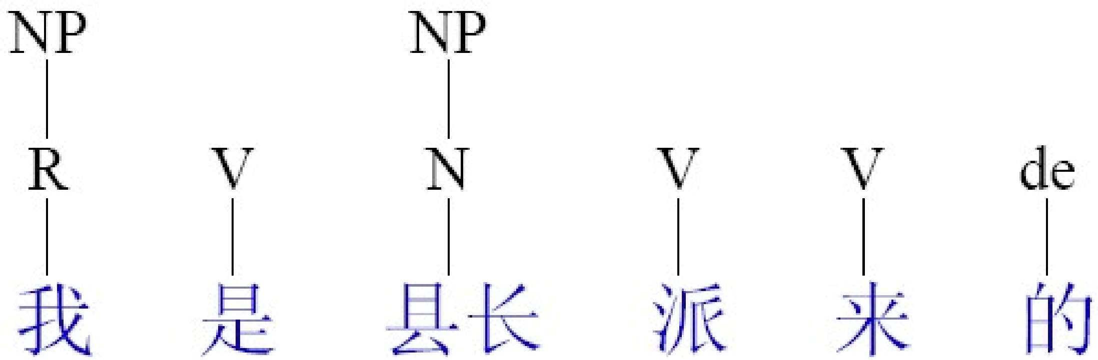
2

NP→R



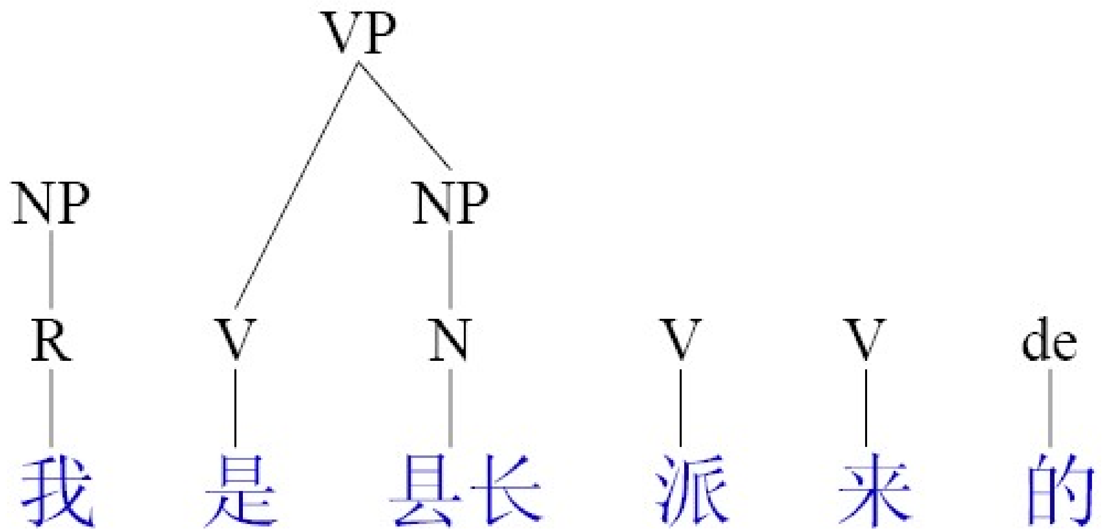
3

NP→N

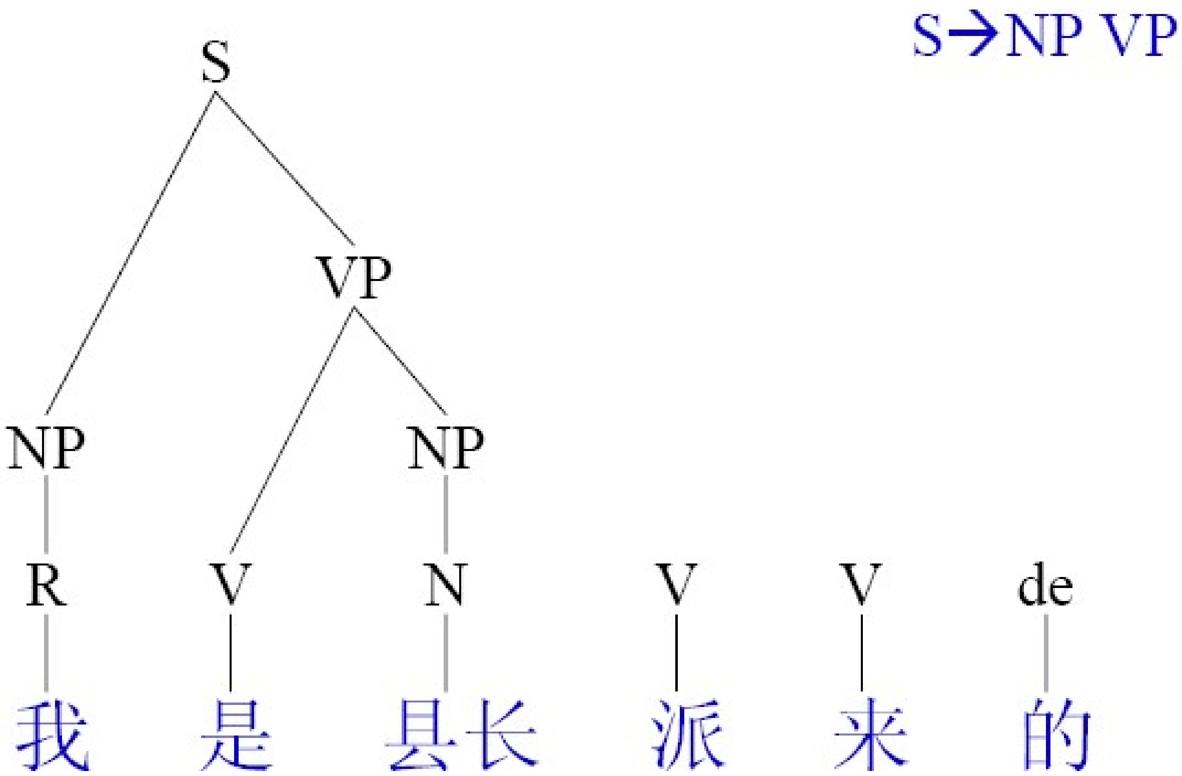


4

VP→V NP

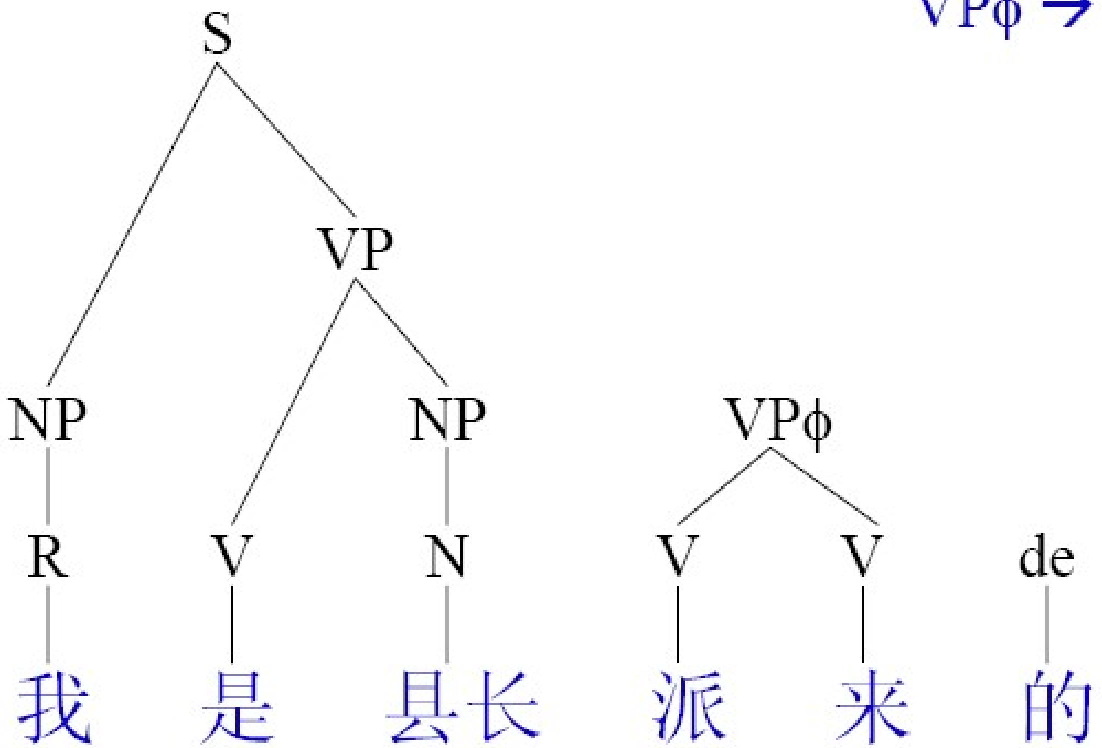


5

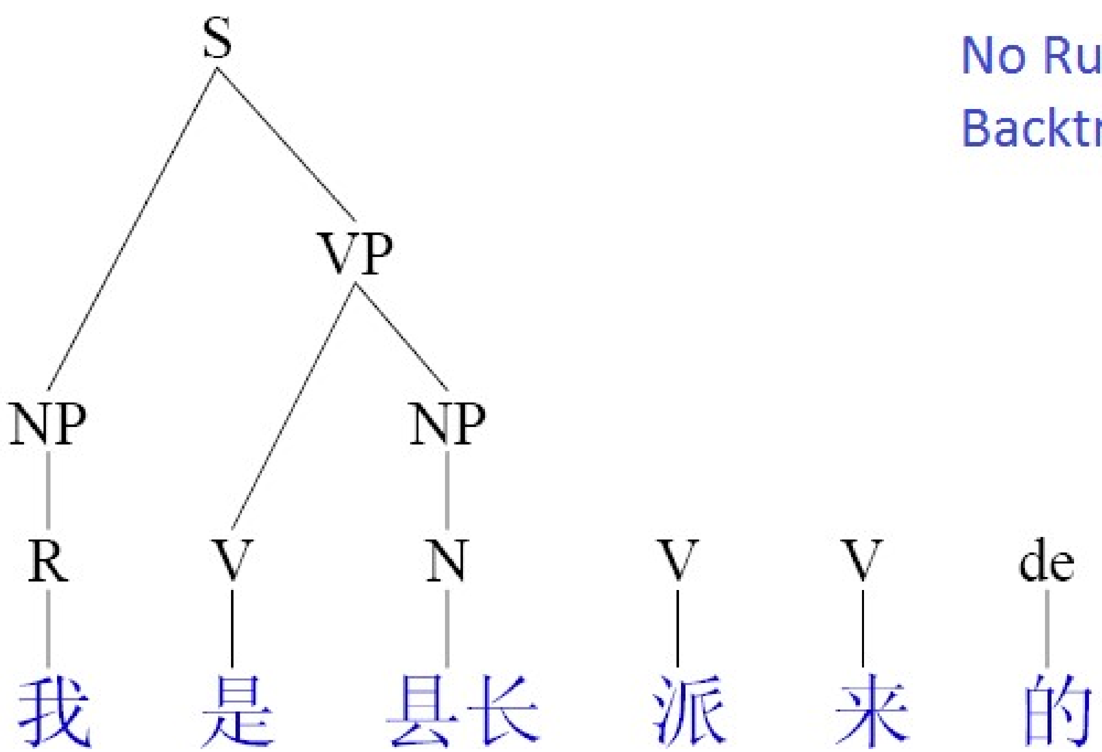


6

VPϕ → V V

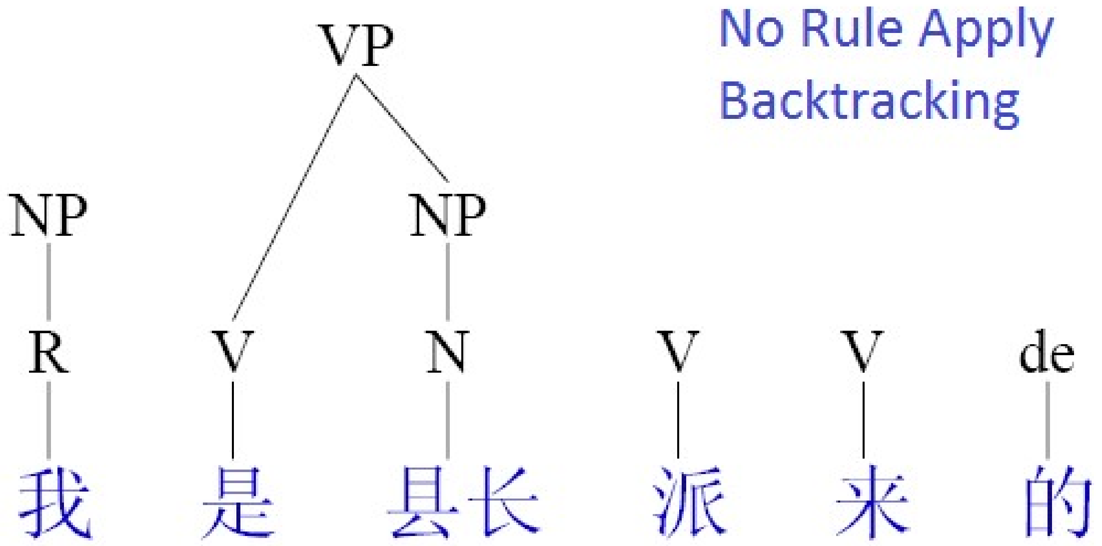


7



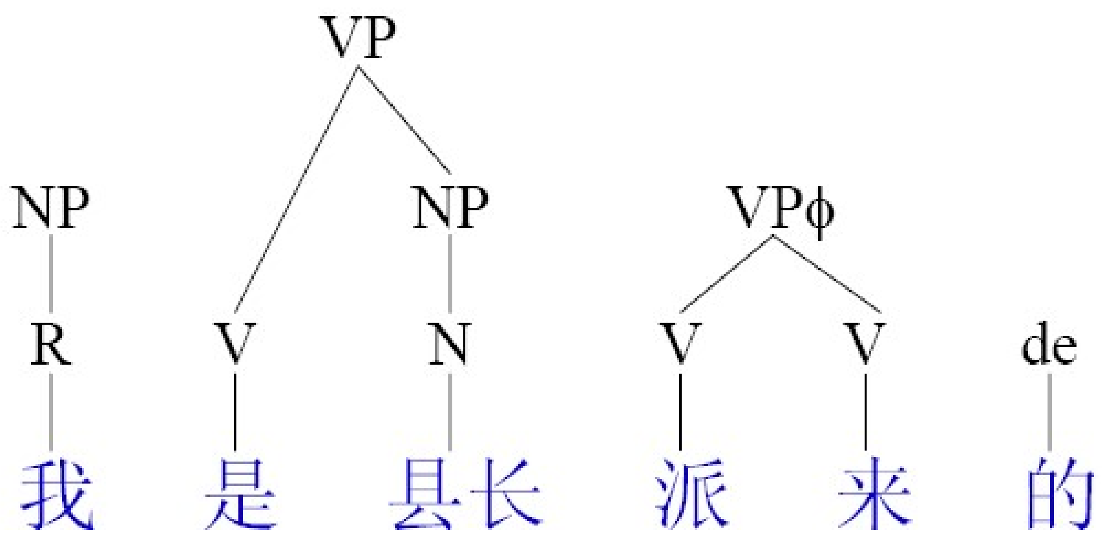
No Rule Apply
Backtracking

8



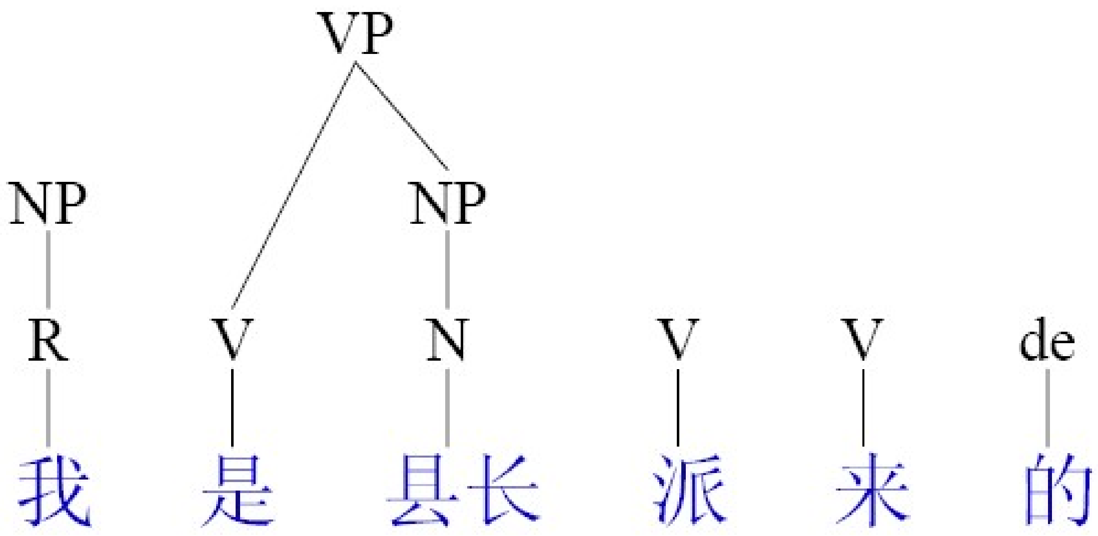
9

$VP\phi \rightarrow V\ V$



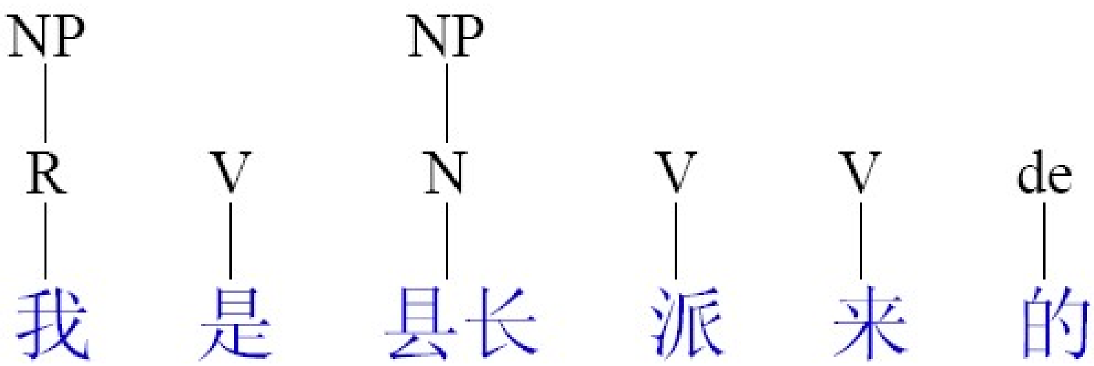
10

No Rule Apply
Backtracking



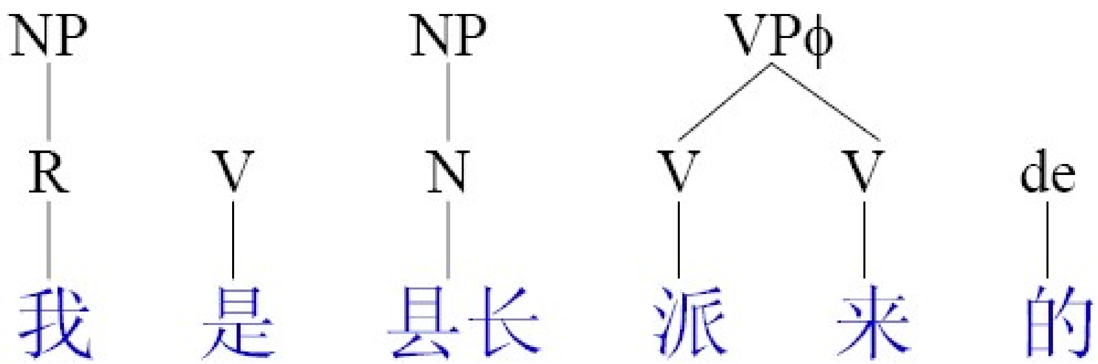
11

No Rule Apply
Backtracking



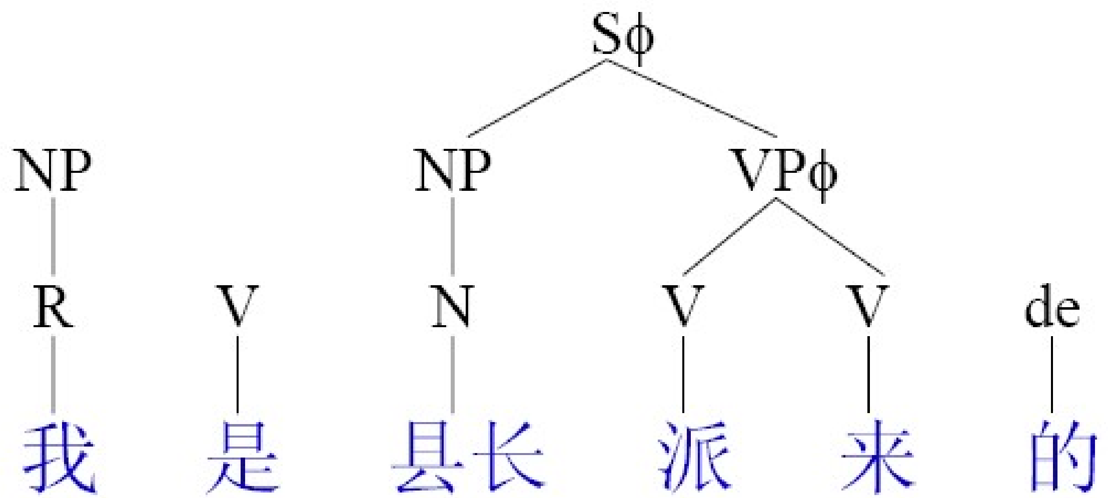
12

$VP\phi \rightarrow V\ V$



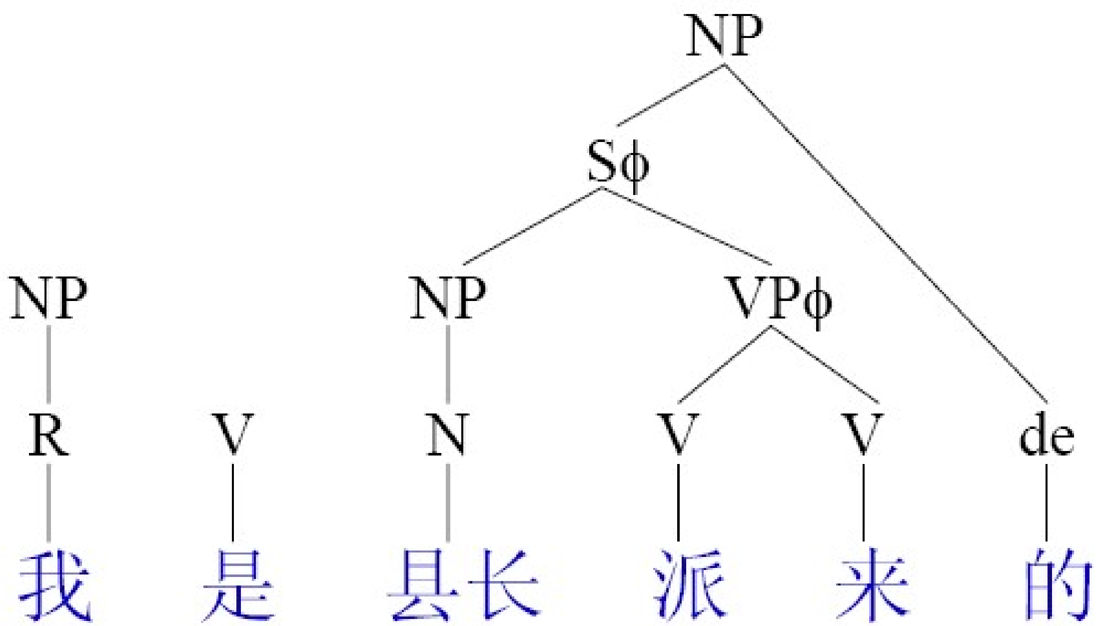
13

$S\phi \rightarrow NP\ VP\phi$

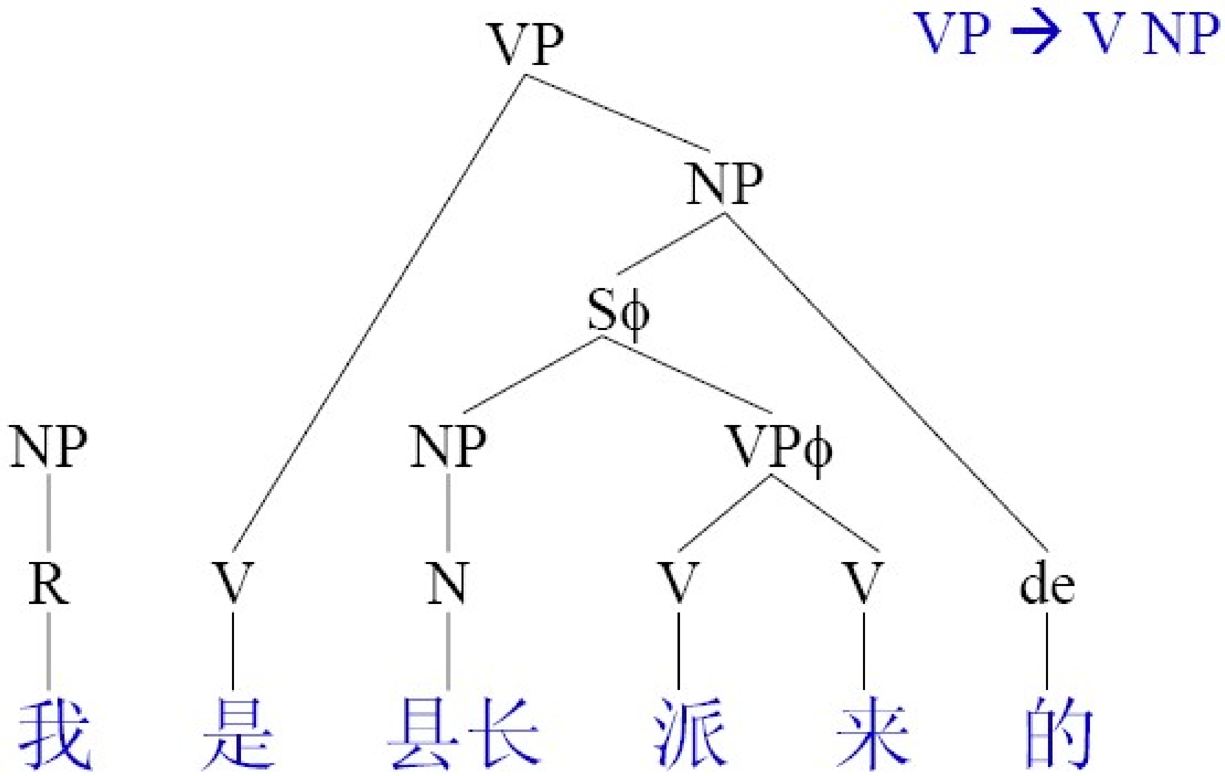


14

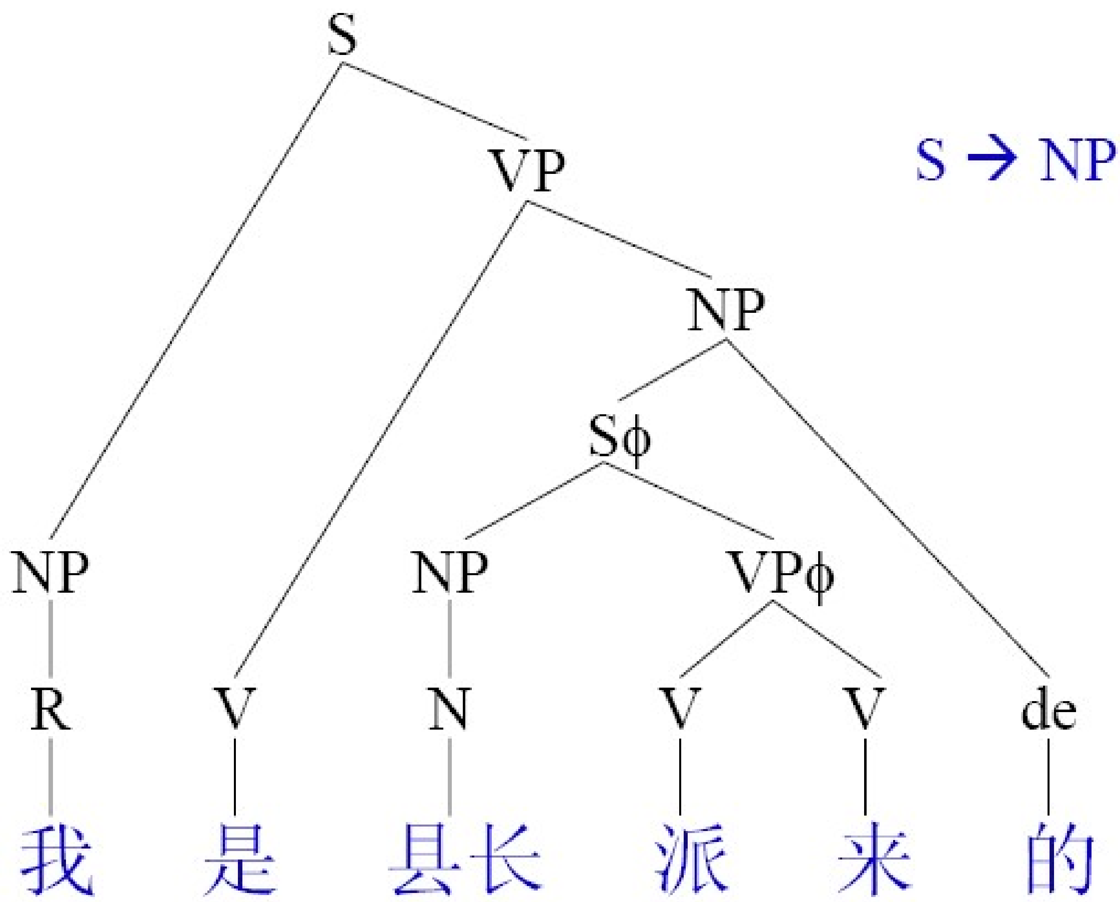
NP → Sϕ de



15



16



S → NP VP

Top Down Parsing - Remarks

- Top-down parsers do well if there is useful *grammar driven control*: search can be directed by the grammar.
- Not too many different rules for the same category
- Not too much distance between non terminal and terminal categories.
- Top-down is unsuitable for rewriting parts of speech (preterminals) with words (terminals). In practice that is always done bottom-up as lexical lookup.

Bottom Up Parsing - Remarks

- It is data-directed: it attempts to parse the words that are there.
- Does well, e.g. for lexical lookup.
- Does badly if there are many rules with similar RHS categories.
- Insufficient when there is great lexical ambiguity (grammar driven control might be helpful here)
- Empty categories: termination problem unless rewriting of empty constituents is somehow restricted (but it's generally incomplete)

CFG Parsing Algorithms

- Shift-Reduce Algorithm
- CYK Algorithm
- Marcus Analysis Algorithm
- Earley Algorithm (Top-down)
- Tomita Algorithm
- Chart Algorithm

Shift-Reduce Algorithm

- Similar to Push Down Automata
- Basic Data structure: Stack
- Four Operations in Shift-Reduced Algorithm
  - Shift 移进:
    - push next input on to top of stack
    - = Shift a terminator to top of stack from left of sentence
  - Reduce 归约 R:
    - top of stack should match RHS of rule
    - replace top of stack with LHS of rule
    - = replace several symbols on the top of stack to one symbol
  - Accept (shift EOF & can reduce what remains on stack to start symbol)
  - Error (shift all words to the stack, but stack have symbols more than S and cannot be reduced)

Shift-Reduce Algorithm

	Stack	Input	Operation	Rule
1	#	我 是 县长	Shift	
2	# 我	是 县长	Reduce	R→ 我
3	# R	是 县长	Reduce	NP→R
4	# NP	是 县长	Shift	
5	# NP 是	县长	Reduce	V→是
6	# NP V	县长	Shift	
7	# NP V 县长		Reduce	N→县长
8	# NP V N		Reduce	NP→N
9	# NP V NP		Reduce	VP→V NP
10	# NP VP		Reduce	S→NP VP
11	# S		Accept	

Ambiguities in Shift-Reduce Algorithm

- Two kinds of Ambiguities
  - Shift-Reduce ambiguity: The word can be both shifted and reduced
  - Reduce-Reduce ambiguity: The word can be reduced by different rules.
- Backtracking
  - For the ambiguous operations, give a selection order
  - Breakpoint Information: Maintain the non-terminator in stack and breakpoint information including the stack information and optional operations on the breakpoint.

Backtracking

- Backtracking Strategy
  - Shift-Reduce Ambiguity: Reduce first, Shift second
  - Reduce-Reduce Ambiguity: Sort the rules according to their priority. Apply the high priority rule first.
- Breakpoint Information
  - Current Applied Rule
  - Candidate Rules
  - Substituted node during reduce operation

Shift-Reduce Algorithm: Illustration

- Sort the rule set

Rule	Dictionary
(7) NP → R	(1) R→我
(8) NP → N	(2) N →县长
(9) NP → Sϕ de	(3) V →是
(10) VPϕ → V V	(4) V →派
(11) VP → V NP	(5) V →来
(12) Sϕ → NP VPϕ	(6) de →的
(13) S → NP VP	

2

	Stack	Input	Opr	Rule
1	#	我 是 县长 派 来 的	Shift	
2	# 我	是 县长 派 来 的	Reduce	(1) R→ 我
3	# R (1)	是 县长 派 来 的	Reduce	(7) NP→R
4	# NP (7)	是 县长 派 来 的	Shift	
5	# NP (7) 是	县长 派 来 的	Reduce	(3) V→是
6	# NP (7) V (3)	县长 派 来 的	Shift	
7	# NP (7) V (3)县长	派 来 的	Reduce	(2) N→县长
8	# NP (7) V (3) N (2)	派 来 的	Reduce	(8) NP→N
9	# NP (7) V (3) NP (8)	派 来 的	Reduce	(11) VP→V NP
10	# NP (7) VP (11)	派 来 的	Reduce	(13) S→NP VP
11	# S (13)	派 来 的	Shift	

3

	Stack	Input	Opr	Rule
12	# S ⒀ 派	来 的	Reduce	⑷ V→派
13	# S ⒀ V⑷	来 的	Shift	
14	# S ⒀ V⑷ 来	的	Reduce	⑸ V→来
15	# S ⒀ V⑷ V⑸	的	Reduce	⑽ VPϕ → V V
16	# S ⒀ VPϕ ⑽	的	Shift	
17	# S ⒀ VPϕ ⑽ 的		Reduce	⑹ de→的
18	# S ⒀ VPϕ ⑽ de⑹		Back	
19	# S ⒀ VPϕ ⑽ 的		Back	
20	# S ⒀ VPϕ ⑽	的	Back	
21	# S ⒀ V⑷ V ⑸	的	Back	
22	# S ⒀ V⑷ 来	的	Back	

4

	Stack	Input	Opr	Rule
23	# S ⒀ V⑷	来 的	Back	
24	# S ⒀ 派	来 的	Back	
25	# S ⒀	派 来 的	Back	
26	# NP ⑺ VP ⑾	派 来 的	Shift	⒀ S→NP VP
27	# NP ⑺ VP ⑾ 派	来 的	Reduce	⑷ V→派
28	# NP ⑺ VP ⑾ V⑷	来 的	Shift	
29	# NP ⑺ VP ⑾ V⑷ 来	的	Reduce	⑸ V→来
30	# NP ⑺ VP ⑾ V⑷ V⑸	的	Reduce	⑽ VPϕ → V V
31	# NP ⑺ VP ⑾ VPϕ ⑽	的	Shift	
32	# NP ⑺ VP ⑾ VPϕ ⑽ 的		Reduce	⑹ de→的
33	# NP ⑺ VP ⑾ VPϕ ⑽ de⑹		Back	

5

	Stack	Input	Opr	Rule
34	# NP (7) VP (11) VPϕ (10) 的		Back	
35	# NP (7) VP (11) VPϕ (10)	的	Back	
36	# NP (7) VP (11) V(4) V(5)	的	Back	
37	# NP (7) VP (11) V(4) 来	的	Back	
38	# NP (7) VP (11) V(4)	来 的	Back	
39	# NP (7) VP (11) 派	来 的	Back	
40	# NP (7) VP (11)	派 来 的	Back	
41	# NP (7) VP (11)	派 来 的	Back	
42	# NP (7) V (3) NP (8)	派 来 的	Shift	
43	# NP (7) V (3) NP (8) 派	来 的	Reduce	(4) V→派
44	# NP (7) V (3) NP (8) V(4)	来 的	Shift	

6

	Stack	Input	Opr	Rule
45	# NP (7) V (3) NP (8) V(4)来	的	Reduce	(5) V→来
46	# NP (7) V (3) NP (8) V(4) V(5)	的	Reduce	(10) VPφ → V V
47	# NP (7) V (3) NP (8) VPφ (10)	的	Reduce	(12) Sφ → NP VPφ
48	# NP (7) V (3) Sφ (12)	的	Shift	
49	# NP (7) V (3) Sφ (12) 的		Reduce	(6) de→的
50	# NP (7) V (3) Sφ (12) de (6)		Reduce	(9) NP → Sφ de
51	# NP (7) V (3) NP (9)		Reduce	(11) VP→V NP
52	# NP (7) VP (11)		Reduce	(13) S→NP VP
53	# S (13)		Accept	

Summary of Shift-Reduce Algorithm

- Shift-Reduce algorithm is a bottom-up analysis
- In order to obtain all possible parsing results, force backtracking is conducted for each success analysis, which leads to lots of redundant operations
- Revisions:
  – Introduce conditional operation rule
  – Introduce contextual operation rule
  – Introduce Cache (Marcus Algorithm)

CKY Parsing

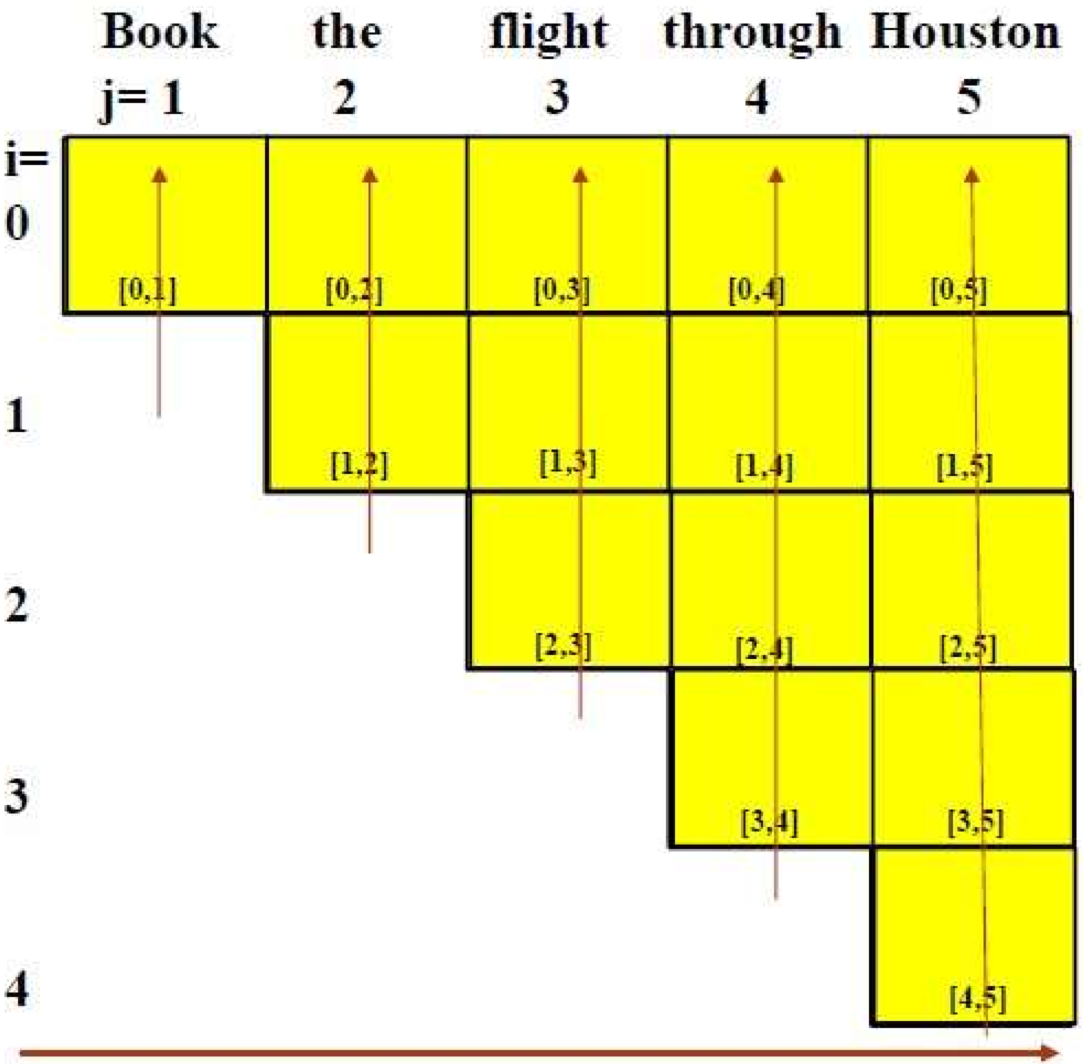
- CKY (Cocke-Kasami-Younger) algorithm
- Bottom-Up Parsing
- The productive rules must be formalized
- Dynamic Parsing ($O(n^3)$)
- The rules must be formalized to Chomsky normal
- form (CNF)， where the RHS of this rule must be 2 non-terminal or 1 terminal
- Paring from Bottom in a table.
- The Parsing for a string depends on the parsing for its sub-strings

CYK Parsing: Rule Formalization

Original Grammar	Chomsky Normal Form
S → NP VP	S → NP VP
S → Aux NP VP	S → X1 VP X1 → Aux NP
S → VP	S → book | include | prefer S → Verb NP S → VP PP
NP → Pronoun	NP → I | he | she | me
NP → Proper-Noun	NP → Houston | NWA
NP → Det Nominal	NP → Det Nominal
Nominal → Noun	Nominal → book | flight | meal | money
Nominal → Nominal Noun	Nominal → Nominal Noun
Nominal → Nominal PP	Nominal → Nominal PP
VP → Verb	VP → book | include | prefer
VP → Verb NP	VP → Verb NP
VP → VP PP	VP → VP PP
PP → Prep NP	PP → Prep NP

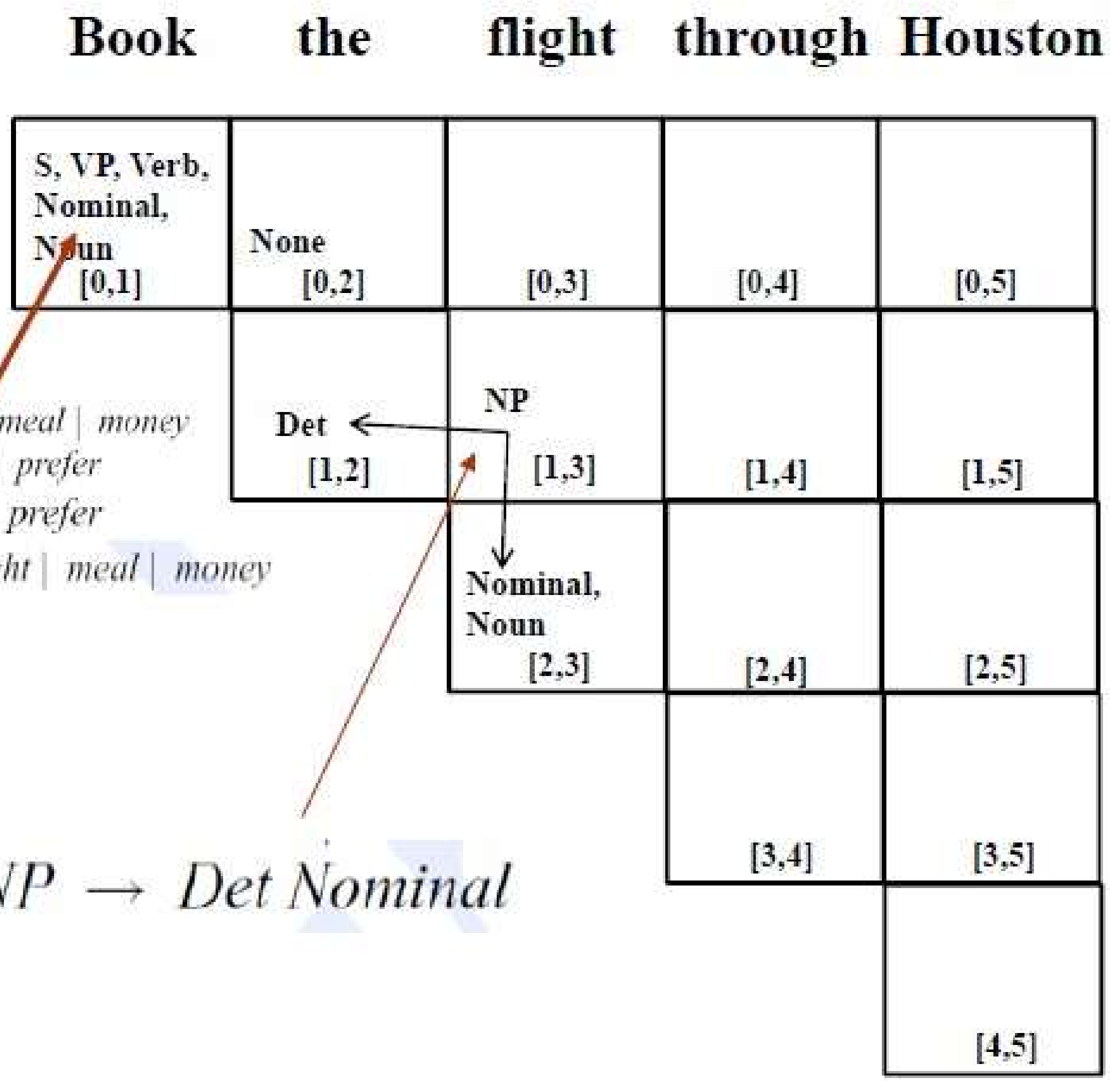
Det → *that* | *this* | *a*
Noun → *book* | *flight* | *meal* | *money*
Verb → *book* | *include* | *prefer*
Pronoun → *I* | *she* | *me*
Proper-Noun → *Houston* | *TWA*
Aux → *does*
Preposition → *from* | *to* | *on* | *near* | *through*

CYK Parsing



Cell[i,j] contains all of Non-terminals which covers Word_{i+1} to Word_j

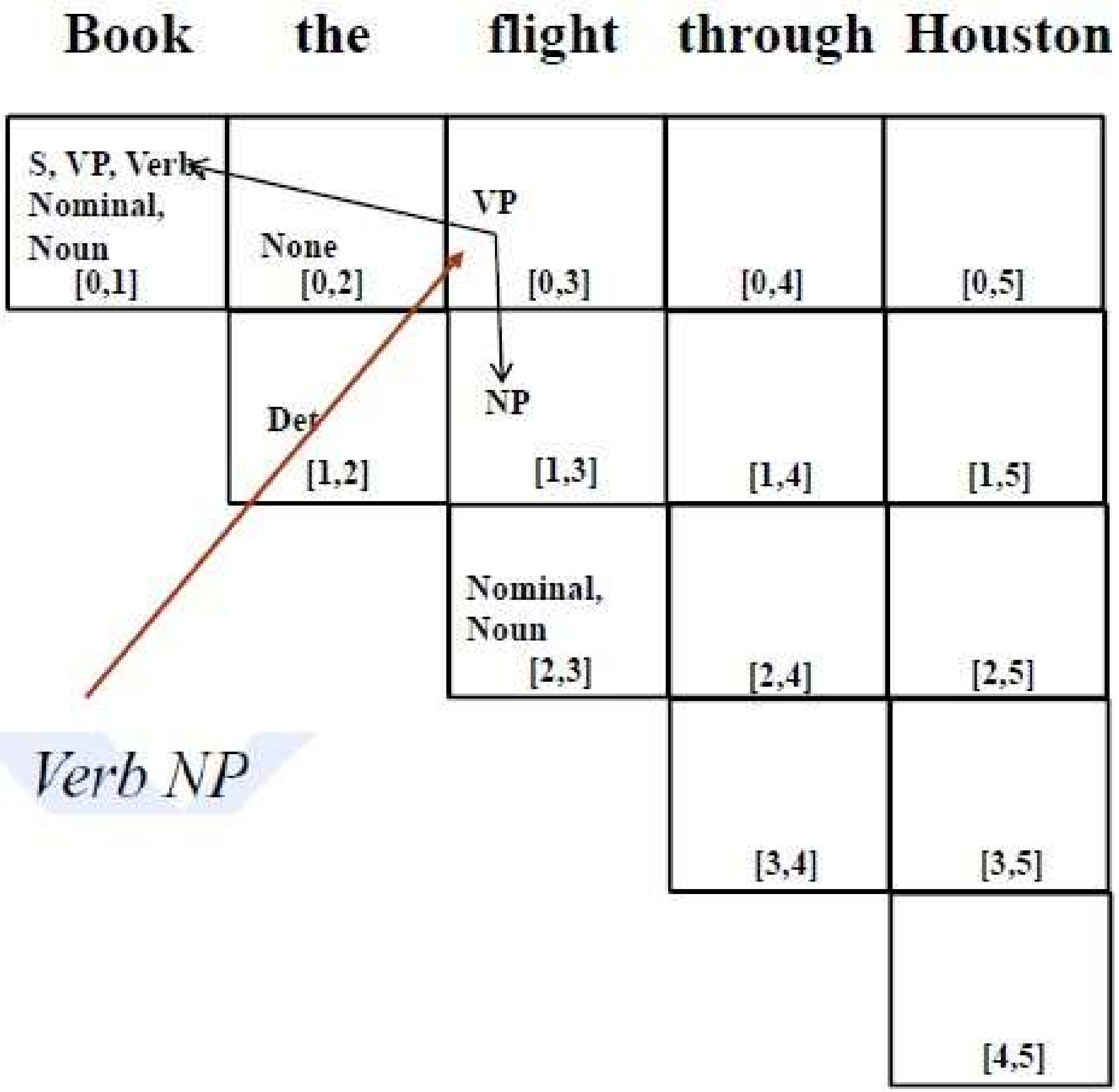
CYK Parsing



$Noun \rightarrow book \mid flight \mid meal \mid money$
$Verb \rightarrow book \mid include \mid prefer$
$S \rightarrow book \mid include \mid prefer$
$Nominal \rightarrow book \mid flight \mid meal \mid money$

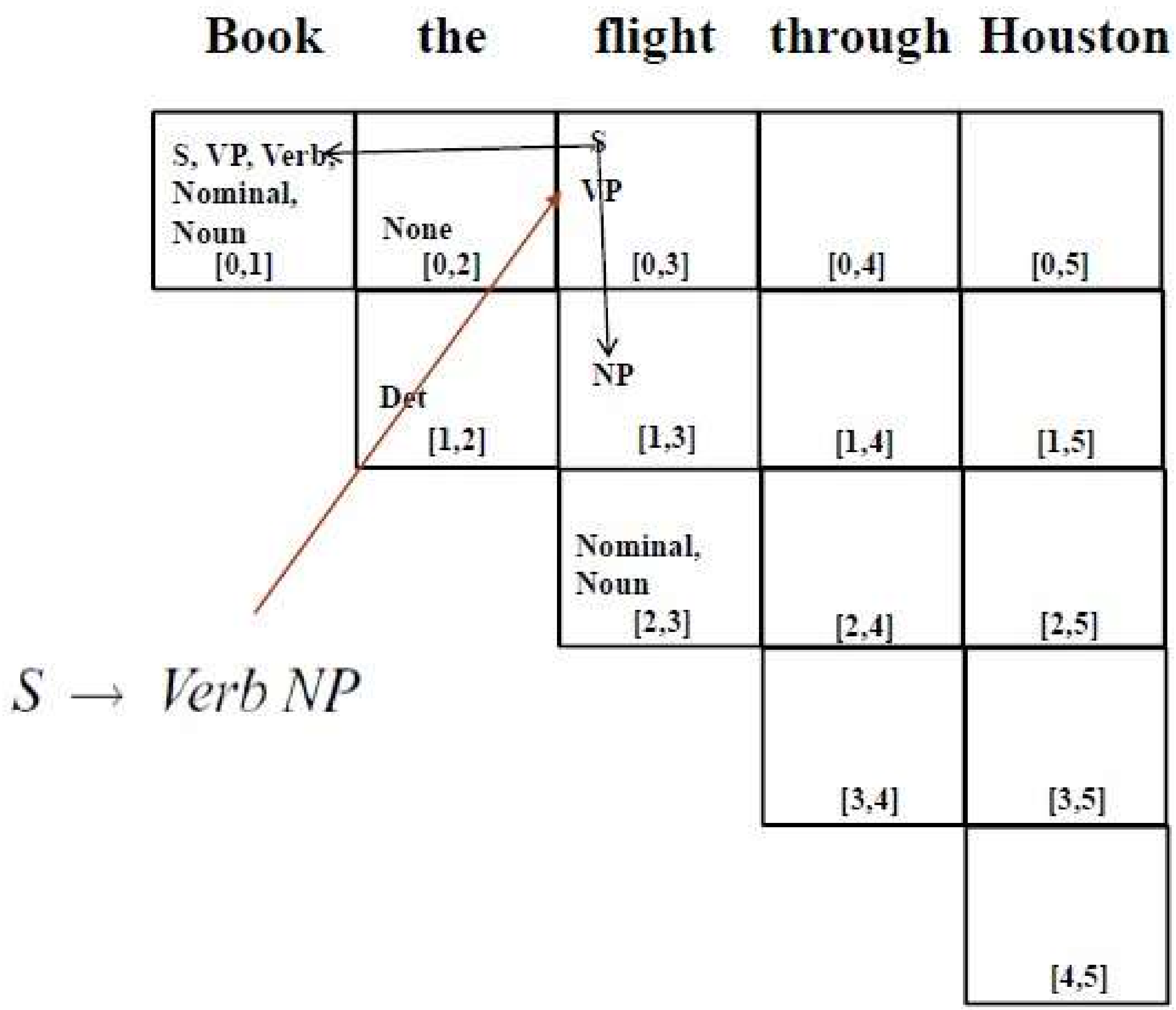
$$NP \rightarrow Det\ Nominal$$

CYK Parsing



$$VP \rightarrow Verb\ NP$$

CYK Parsing



Book the flight through Houston
S, VP, Verb, Nominal, Noun
[0,1]
None
[0,2]
S
VP
[0,3]
[0,4]
[0,5]
Det
[1,2]
NP
[1,3]
[1,4]
[1,5]
Nominal, Noun
[2,3]
[2,4]
[2,5]
[3,4]
[3,5]
[4,5]
$S \rightarrow Verb\ NP$

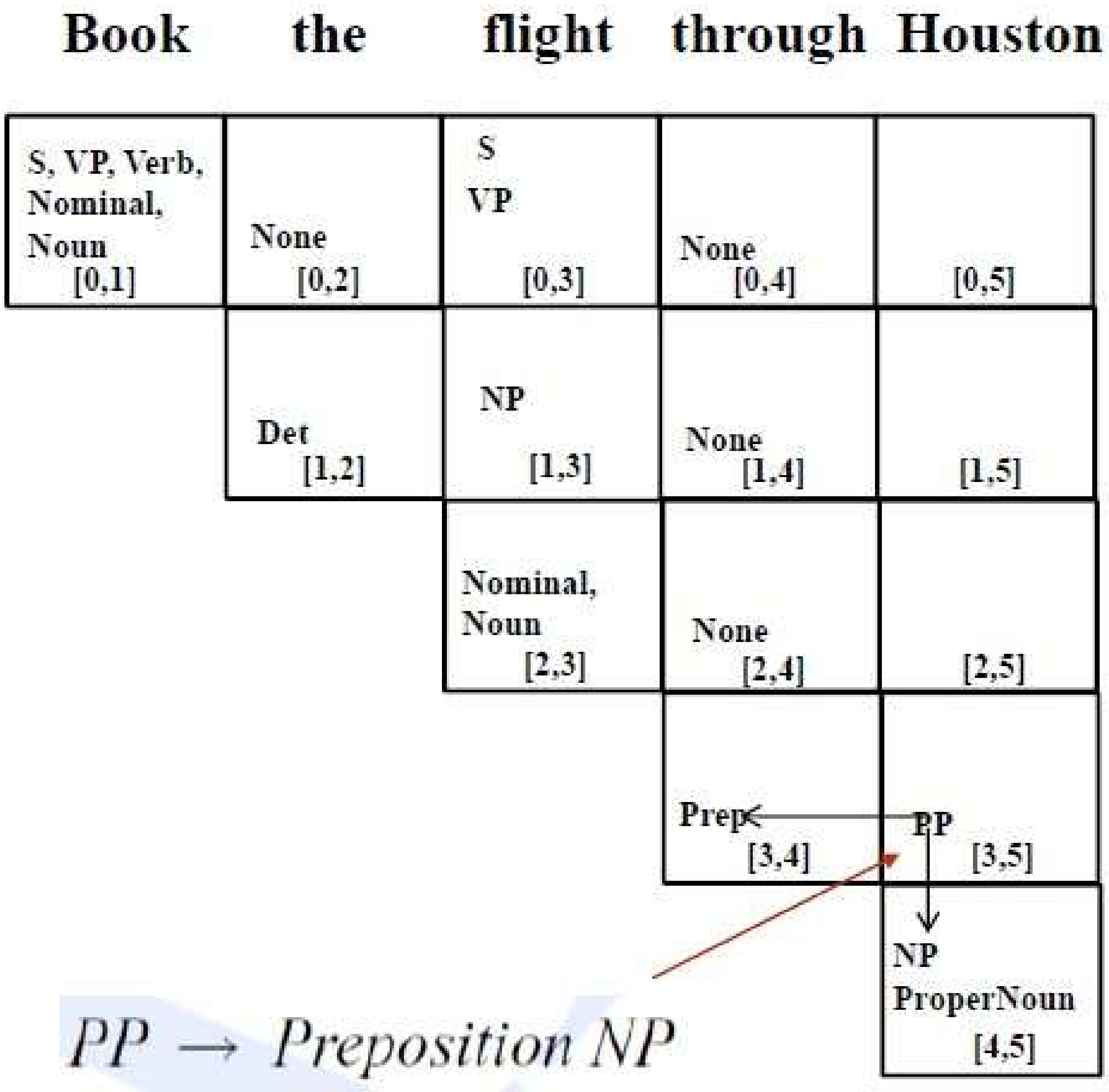
CYK Parsing

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun [0,1]	None [0,2]	S VP [0,3]	[0,4]	[0,5]
	Det [1,2]	NP [1,3]	[1,4]	[1,5]
		Nominal, Noun [2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

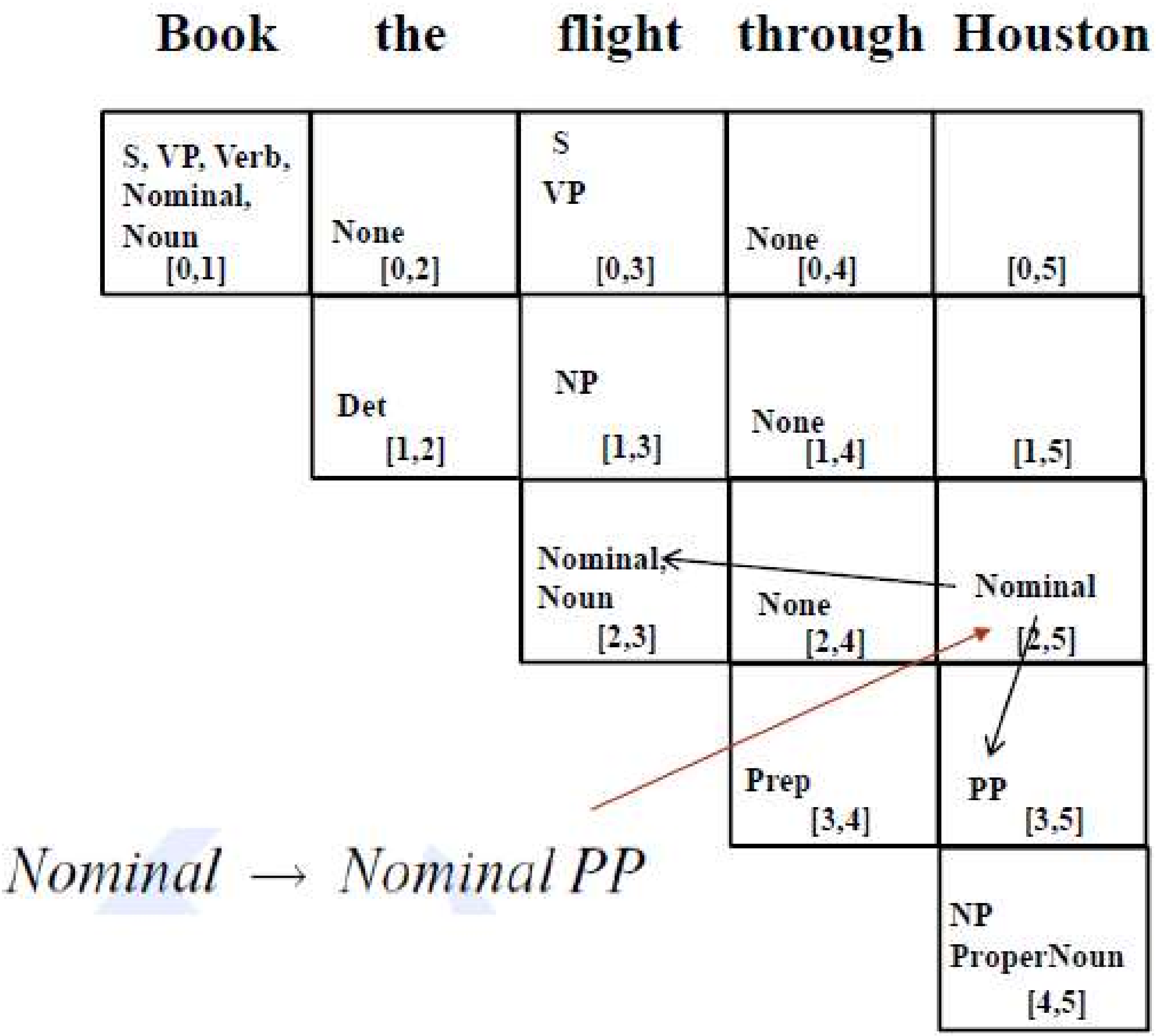
CYK Parsing

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun [0,1]	None [0,2]	S VP [0,3]	None [0,4]	[0,5]
	Det [1,2]	NP [1,3]	None [1,4]	[1,5]
		Nominal, Noun [2,3]	None [2,4]	[2,5]
			Prep [3,4]	[3,5]
				[4,5]

CYK Parsing

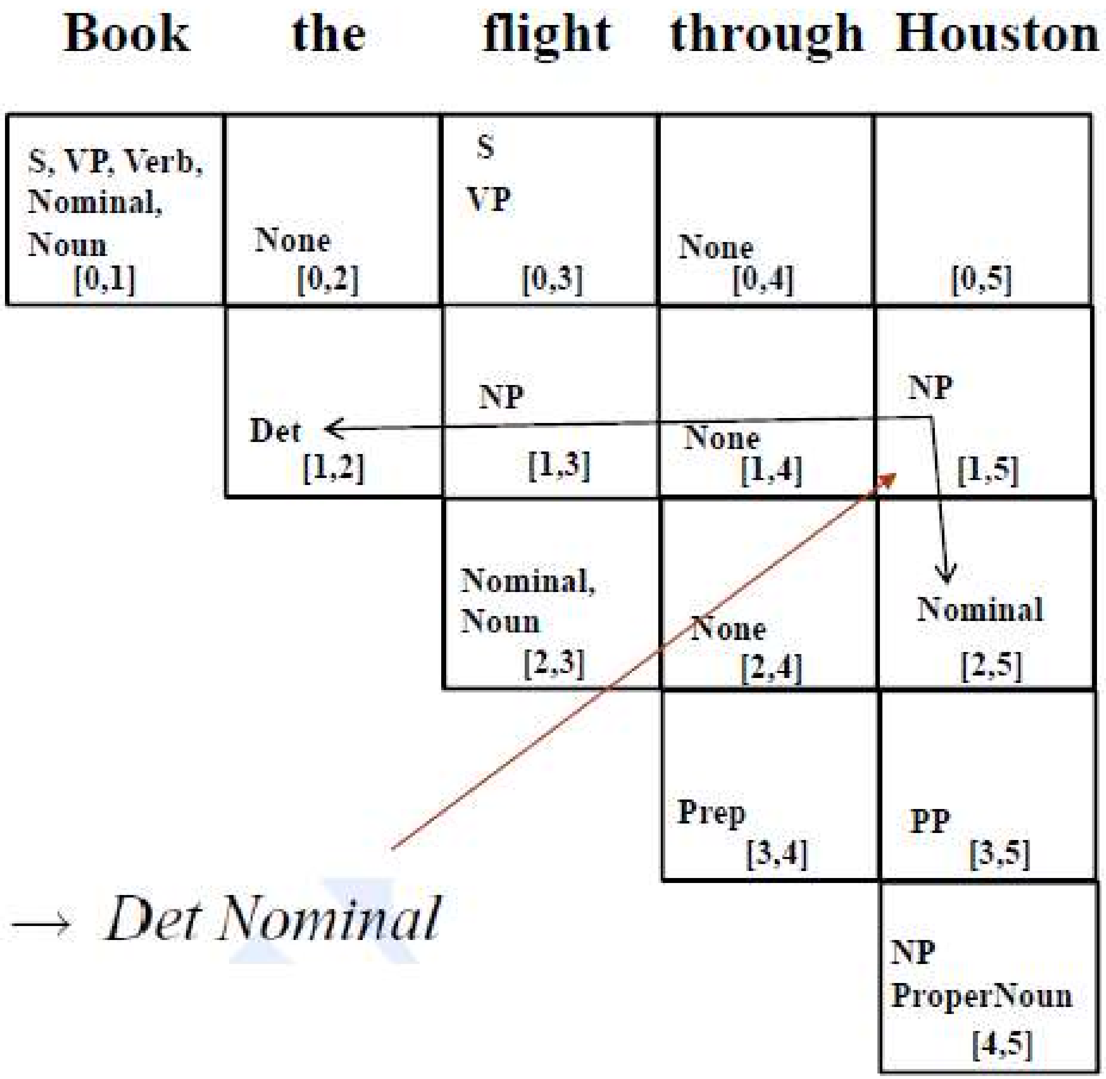

Book the flight through Houston
S, VP, Verb, Nominal, Noun [0,1]
None [0,2]
S VP [0,3]
None [0,4]
[0,5]
Det [1,2]
NP [1,3]
None [1,4]
[1,5]
Nominal, Noun [2,3]
None [2,4]
[2,5]
Prep [3,4]
PP [3,5]
NP ProperNoun [4,5]
PP → Preposition NP

CYK Parsing

- 

Book the flight through Houston
S, VP, Verb, Nominal, Noun
[0,1]
None
[0,2]
S
VP
[0,3]
None
[0,4]
[0,5]
Det
[1,2]
NP
[1,3]
None
[1,4]
[1,5]
Nominal, Noun
[2,3]
None
[2,4]
Nominal
[2,5]
Prep
[3,4]
PP
[3,5]
NP
ProperNoun
[4,5]
Nominal → Nominal PP

CYK Parsing



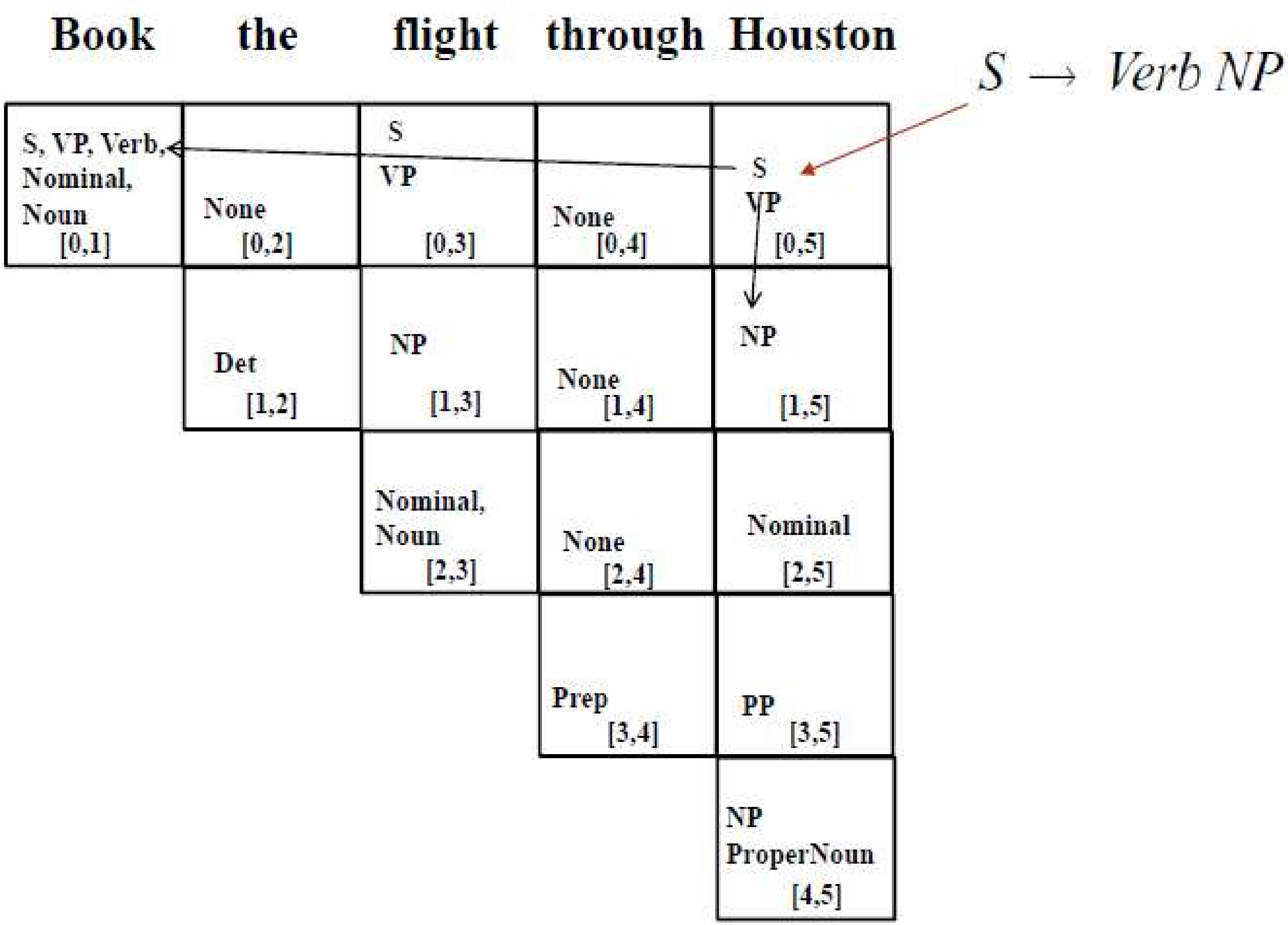
$$NP \rightarrow Det\ Nominal$$

CYK Parsing

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun [0,1]	None [0,2]	S VP [0,3]	None [0,4]	VP [0,5]
	Det [1,2]	NP [1,3]	None [1,4]	NP [1,5]
		Nominal, Noun [2,3]	None [2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP ProperNoun [4,5]

$$VP \rightarrow Verb\ NP$$

CYK Parsing



Book the flight through Houston
S → Verb NP
S, VP, Verb, Nominal, Noun [0,1]
None [0,2]
S VP [0,3]
None [0,4]
S VP [0,5]
Det [1,2]
NP [1,3]
None [1,4]
NP [1,5]
Nominal, Noun [2,3]
None [2,4]
Nominal [2,5]
Prep [3,4]
PP [3,5]
NP ProperNoun [4,5]

CYK Parsing

•

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun [0,1]	None [0,2]	S VP [0,3]	None [0,4]	VP S VP [0,5]
	Det [1,2]	NP [1,3]	None [1,4]	NP [1,5]
		Nominal, Noun [2,3]	None [2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP ProperNoun [4,5]

$$VP \rightarrow VP\ PP$$

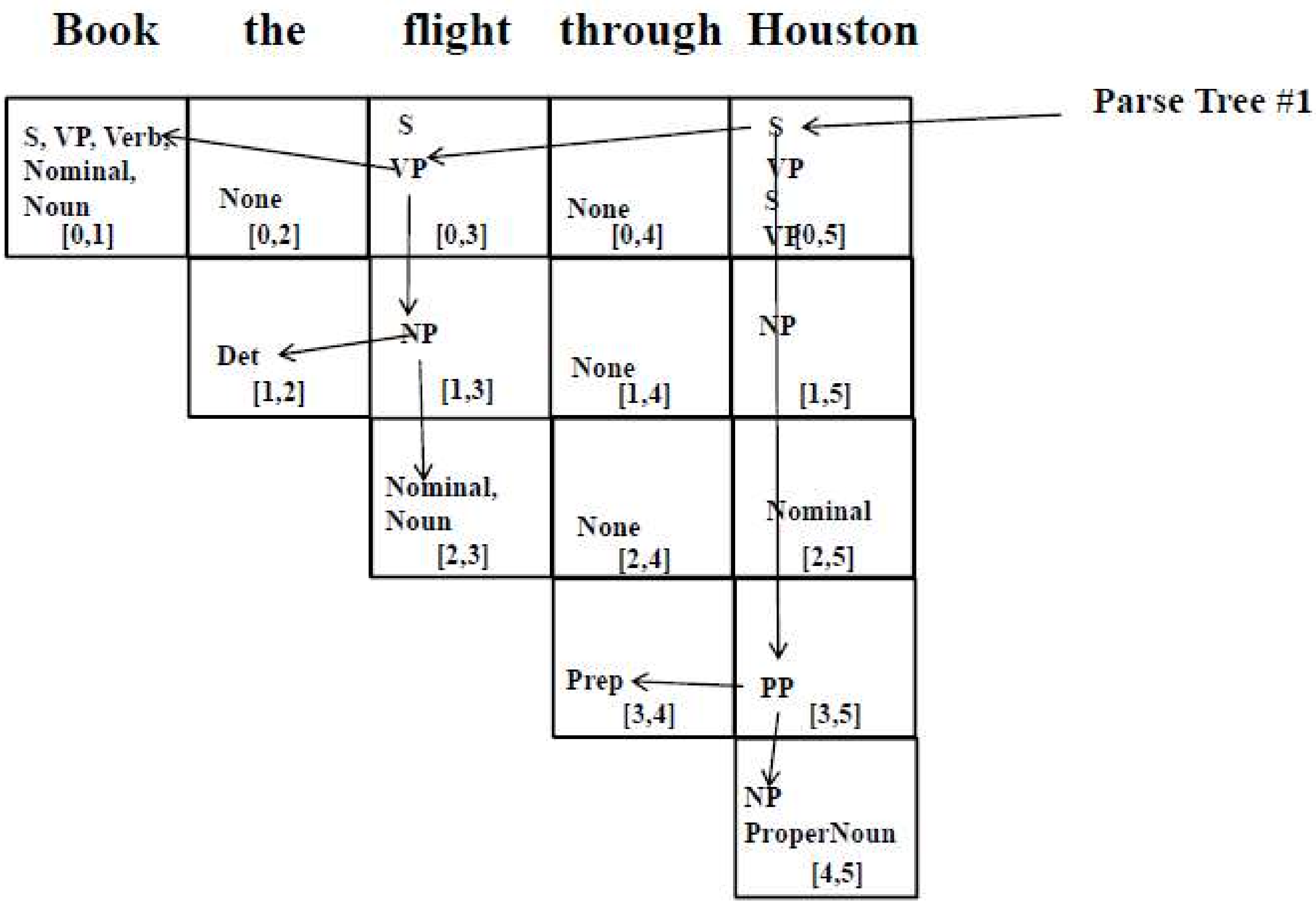
CYK Parsing

$$S \rightarrow VP\ PP$$

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun [0,1]	None [0,2]	S VP [0,3]	None [0,4]	S VP S VP [0,5]
	Det [1,2]	NP [1,3]	None [1,4]	NP [1,5]
		Nominal, Noun [2,3]	None [2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP ProperNoun [4,5]

CYK Parsing

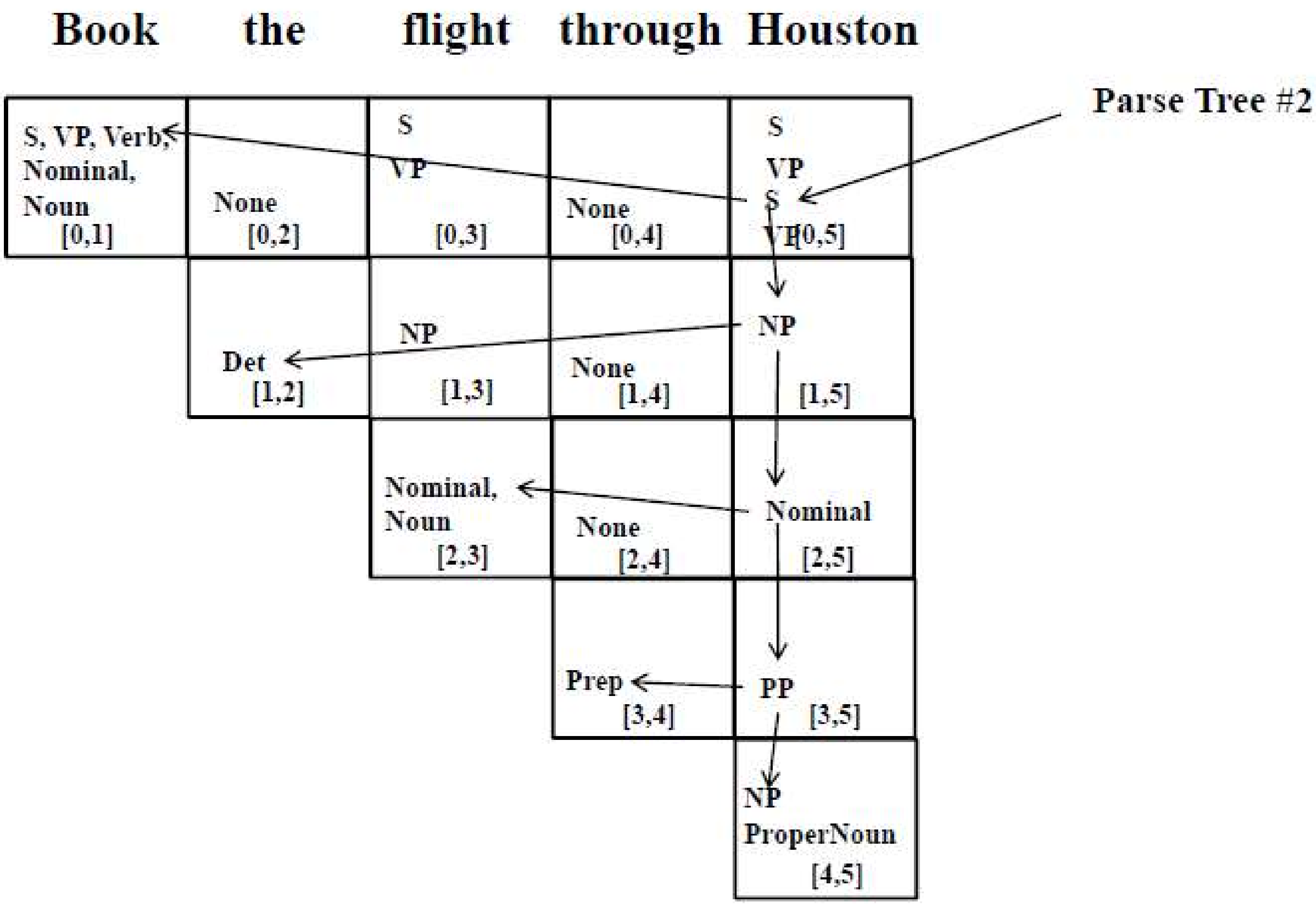
•



Book the flight through Houston
Parse Tree #1
S, VP, Verb, Nominal, Noun
[0,1]
None
[0,2]
S
VP
[0,3]
None
[0,4]
S
VP
S
VP
[0,5]
Det
[1,2]
NP
[1,3]
None
[1,4]
NP
[1,5]
Nominal, Noun
[2,3]
None
[2,4]
Nominal
[2,5]
Prep
[3,4]
PP
[3,5]
NP
ProperNoun
[4,5]

CYK Parsing

•



Book the flight through Houston
Parse Tree #2
S, VP, Verb, Nominal, Noun [0,1]
None [0,2]
S VP [0,3]
None [0,4]
S VP S VP [0,5]
Det [1,2]
NP [1,3]
None [1,4]
NP [1,5]
Nominal, Noun [2,3]
None [2,4]
Nominal [2,5]
Prep [3,4]
PP [3,5]
NP ProperNoun [4,5]

CYK Parsing

- Generate all possible parsing tree
- Complexity: $O(n^3)$
- Dynamic Parsing

- No disambiguation capability

The Next Lecture

- Lecture 11

  Parsing II