Natural Language Processing II (SC674)

Prof. Feng Zhiwei

Ch3. Parsing with PSG

3.1 Bottom-Up Parsing

3.1.1 Definition of parsing:

- Parsing means taking an input and producing some sort of structure for it. Parsing is a general conception not only for linguistics, but also for programming technique.
- In NLP, the parsing is a combination of recognizing an input string and assigning some structure to it.
- Syntactic parsing is the task of recognizing a sentence and assigning a syntactic structure (e..g. tree, chart) to it.

3.1.2 Parsing as search

In syntactic parsing, the parser can be reviewed as searching through the space of all possible parse trees to find the correct parse tree for the correct sentence.

E.g. If we have a small PSG for English:

1. S → NP VP
2. S → AUX NP VP
3. S → VP
4. NP → Det Nominal
5. Nominal → Noun
6. Nominal → Noun Nominal
7. Nominal → Nominal PP
8. NP → Proper Noun
9. VP → Verb
10. VP → Verb NP
11. Det → that | this | a
12. Noun → book | flight | meat | money
13. Verb → book | include | prefer
14. Aux → does
15. Prep → from | to | on
16. Proper Noun → Houston | ASIANA | KOREAN AIR | CAAC | Dragon Air

Using this PSG to parse sentence “Book that flight”, the correct parse tree that would be assigned to this sentence is as follows:

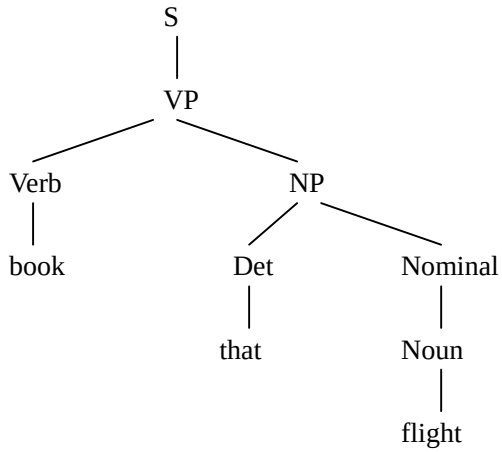


Fig. 1 parse tree

Regardless of the search algorithm we choose, there are two kinds of constraints that should help guide the search.

- Constraint coming from the data: The final parse tree must have three leaves (three words in the input sentence): “book, that flight”.
- Constraint coming from the grammar: the final parse tree must have one root: S (start symbol).

These two constraints give rise to the two search strategies:

- Bottom-up search (or data-directed search)
- Top-down search (or goal-directed search)

3.1.3 Bottom-Up Parsing

In bottom-up parsing, the parser starts with the words of the input and tries to build tree from the words up. The parsing is successful if the parser succeeds in building a tree rooted in the start symbol S that covers all of the input.

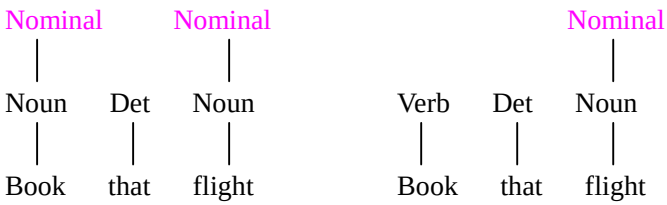
Example:

We use above small PSG to parse (Bottom-Up) sentence “Book that flight”

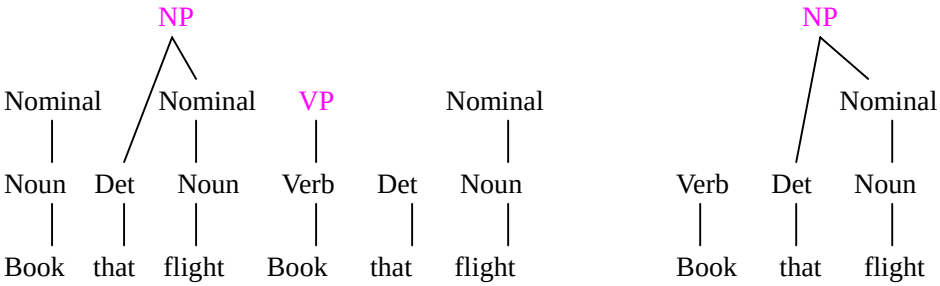
First ply: Book that flight

Second ply:

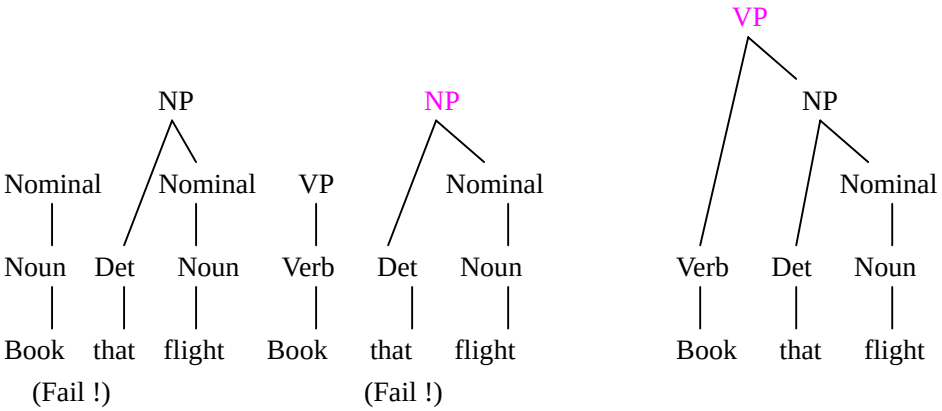
Third ply:



Fourth ply:



Fifth ply:



Sixth ply:

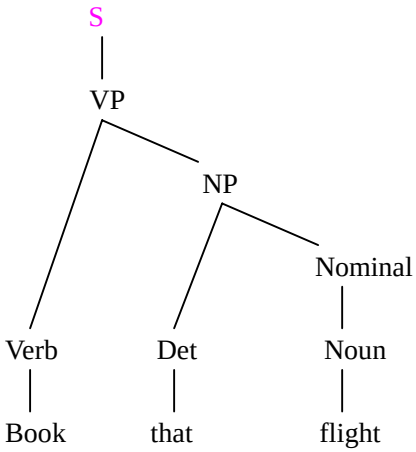


Fig. 2 Bottom-Up parser

In sixth ply, the root S covers all the input, our Bottom-Up parsing is success.

We can use Shift-Reduce algorithm to do the parsing.

In the shift-reduce algorithm, the stack is used for information access. The operation methods are shift, reduce, refuse and accept. In the shift, the symbol waiting to process is move to the top of stack. In the reduce, the symbol on stack top is replaced by RHS of grammar rule, if the RHS of the rule is matched with the symbol on stack top. If the input string is processed, and the symbol on stack top becomes S (initial symbol in the string), then the input string is accepted. Otherwise, it is refused.

Following is the shift-reduce process of sentence “Book that flight”

Stack	Operation	the rest part of input string
		Book that flight
++Book	shift	that flight
Noun	reduce by rule12	that flight
Noun that	shift	flight
Noun Det	reduce by rule 11	flight
Noun Det flight	shift	ϕ
Noun Det Noun	reduce by rule 12	ϕ
Noun Det Nominal	reduce by rule 5	ϕ
Noun NP	reduce by rule 4	ϕ

	[Backtracking to ++]	
+++ Verb	reduce by rule 13	that flight
VP	reduce by rule 9	that flight
VP that	shift	flight
VP Det	reduce by rule 11	flight
VP Det flight	shift	ϕ
VP Det Noun	reduce by rule 12	ϕ
VP Det Nominal	reduce by rule 5	ϕ
VP NP	reduce by rule 4	ϕ
	[Backtracking to +++]	
Verb that	shift	flight
Verb Det	reduce by rule 11	flight
Verb Det flight	shift	ϕ
Verb Det Noun	reduce by rule 12	ϕ
Verb Det Nominal	reduce by rule 5	ϕ
Verb NP	reduce by rule 4	ϕ
VP	reduce by rule 10	ϕ
S	reduce by rule 3	ϕ
	[Success !]	

3.2 Top-Down Parsing

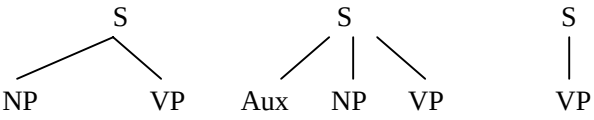
3.2.1 The process of Top-Down Parsing

A top-down parser searches for a parse tree by trying to build from the root node S down to the leaves. The algorithm starts symbol S. The next step is to find the tops of all trees which can start with S. Then expend the constituents in new trees. etc.

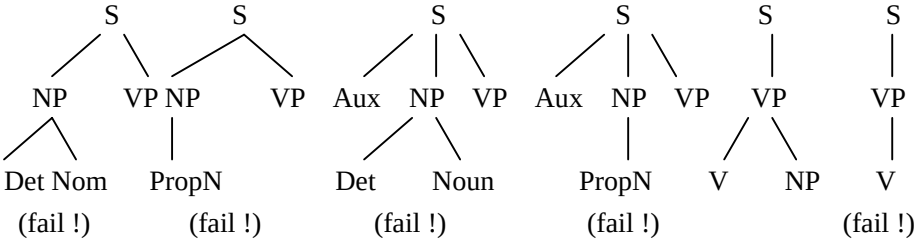
If we use above small PSG to parse (Top-Down) sentence “Book that flight”,, first 3 ply will be as follows:

First ply:

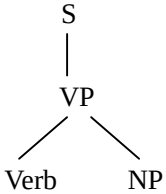
Second ply:



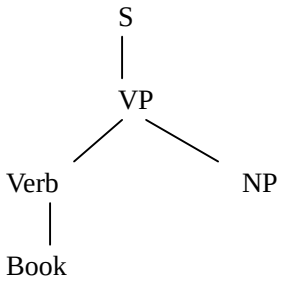
Third ply:



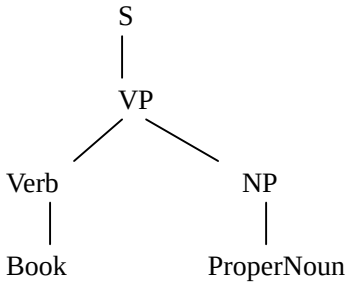
In this case, only the fifth parse tree will match the input sentence.



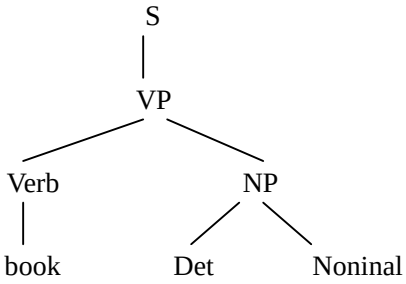
Fourth ply:



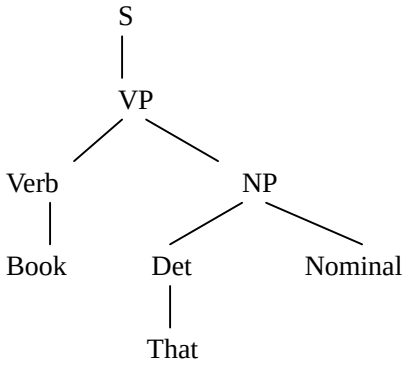
Fifth ply:



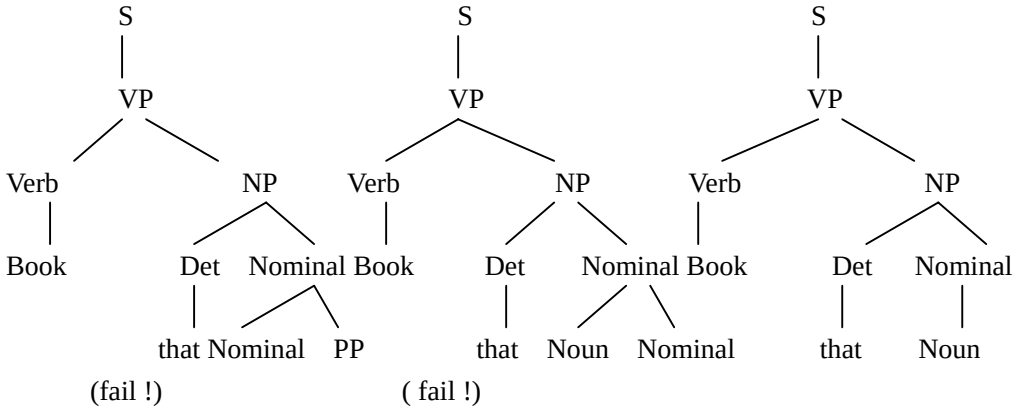
(fail !)



Sixth ply:



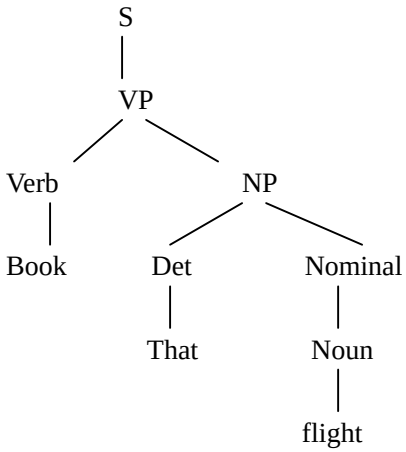
Seventh ply:



(fail !)

(fail !)

Eighth ply:



[Success !]

Fig. 3 Top-Down parsing

The search process of the sentence ”book that flight”:

Searching goal	Rule	The rest part of input string
++S		Book that flight
+NP VP	1	Book that flight
Det Nom VP	4	Book that flight
[backtracking to +]		
PropN VP	8	Book that flight
[backtracking to ++]		
Aux NP VP	2	Book that flight
[backtracking to ++]		Book that flight
+++VP	3	Book that flight
Verb	9	Book that flight
ϕ		that flight
[backtracking to +++]		Book that flight
++++Verb NP	10	Book that flight
PropN	8	that flight
[backtracking to ++++]		
Det Nominal	4	that flight
+++++ Nominal		flight
++++++Nominal PP	7	flight
Noun Nominal PP	6	flight
Nominal PP		ϕ
[backtracking to ++++++]		
Noun PP	5	flight
PP		ϕ
[backtracking to +++++]		
Noun Nominal	6	flight
Nominal		ϕ
[backtracking to +++++]		
Noun	5	flight
ϕ		ϕ

[Success !]

3.2.2 Comparing Top-Down and Bottom-Up Parsing

- Top-Down strategy never wastes time exploring trees that cannot result in an S, since it begins by generating just those trees. This means it never explores subtrees that cannot find a place in some S-rooted tree. By contrast, In the bottom-up strategy, trees that have no hope of leading to an S are generated with wild abandon. it will waste effort
- Top-Down strategy spends considerable effort on S trees that are not consistent with the input. It can generate trees before ever examining the input. Bottom-Up never suggest trees that are not locally grounded in the actual input.

Neither of these approaches adequately exploits the constraints presented by the grammar and the input words.

3.3.3 A basic Top-Down parser

3.3.3.1 left-corner: We call the first word along the left edge of a derivation the left-corner of the tree.

e.g.

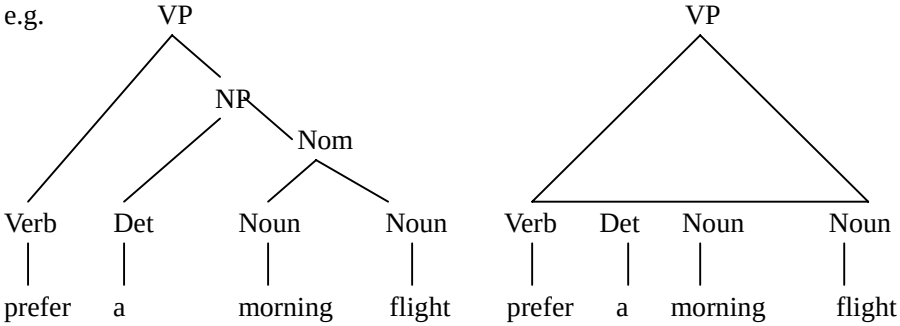


Fig. 4 left-corner

In Fig. 4, the node “verb” and the node “prefer” are both left-corner of VP.

Formally, we can say that for non-terminals A and B, B is a left-corner of A if the following relation holds:

$A \rightarrow B\,\alpha$

In other words, B can be left-corner of A if there is a derivation of A that begins with a B.

The parser should not consider any grammar rule if the current input cannot serve as the first word along the left edge of some derivation from the rule.

3.2.3 Bottom-Up Filtering

We can set up a table that list all the valid left-corner categories (it is part of speech, POS) for each non-terminal (e.g. S, VP, NP, etc) in the grammar. When a rule is considered, the table entry for the category (POS) that starts the right hand side of the rule in consulted. If it fails to contain any of the POS associated with the current input then the rule is eliminated from consideration. In this case, this table can be regarded as a bottom-up filter.

For our small Grammar, the left-corner table is as follows:

Non-terminal	Left-corner
S	Det, Proper Noun, Aux, Verb
NP	Det, Proper Noun
Nominal	Noun
VP	Verb

Fig. 5 left-corner table

Using this left-corner table, the process of sentence “book that flight” will become simple and quick. The process is as follows:

First ply:

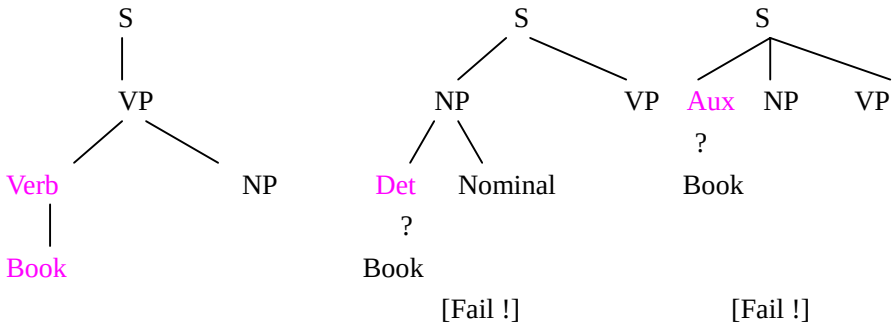


Fig. 5

Verb is the left-corner of S. “Det” and “Aux” can not match with “Book”.

Second ply:

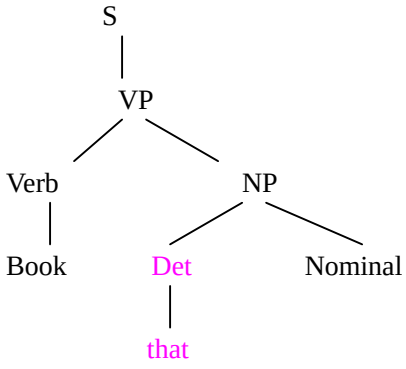


Fig. 6

Det is the left-corner of NP.

Third ply:

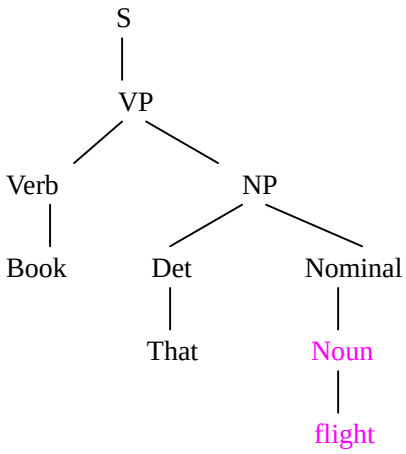


Fig. 7

Noun is the left-corner of Nominal.

The Top-Down parsing process using left-corner filter is as follows:

Searching goal	Rule	The rest part of input string
S		Book that flight
+ VP	3	Book that flight
Verb	9	Book that flight
ϕ		that flight
	[backtracking to +]	
Verb NP	10	Book that flight
NP		that flight
Det Nominal	4	that flight
Nominal		flight
Noun	5	flight
ϕ		ϕ
	[Success !]	

3.3 Problems with Top-Down Parser

3.3.1 Left-recursion:

In top-down, depth-first, left-to-right parser, It may dive down an infinitely deeper path never return to visit space if it use the left-recursive grammar.

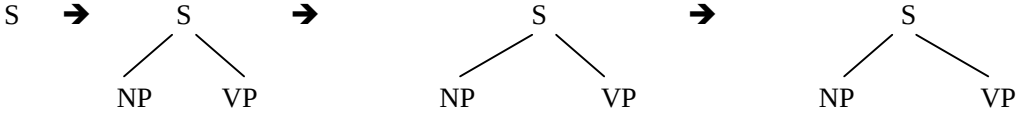
Formally, a grammar is left-recursive if it contains at least one non-terminal A, such that , A → α A β , for some α and β and α => ε . In other words, a grammar is left-recursive if it contains a non-terminal category that has a derivation that includes itself anywhere along its leftmost branch.

A more obvious and common case of left-recursion in natural language grammar involves immediately left-recursive rules. The left-recursive rules are rules of the form A → A β , where the first constituent of the RHS is identical to the LHS.

E.g. NP → NP PP
VP → VP PP
S → S and S

A left-recursive non-terminal can lead a top-down, depth-first, left-to-right parser to recursively expand the same non-terminal over again in exactly the same way, leading to an infinite expansion of the trees.

E.g. if we have the left recursive rule NP → NP PP as first rule in our small grammar, we may get the infinite search as following:



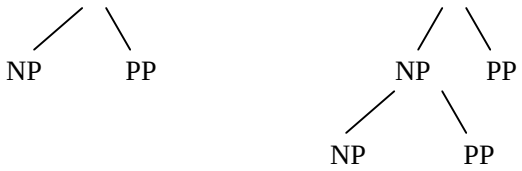


Fig. 8 infinite search .

3.3.2 Structure ambiguity

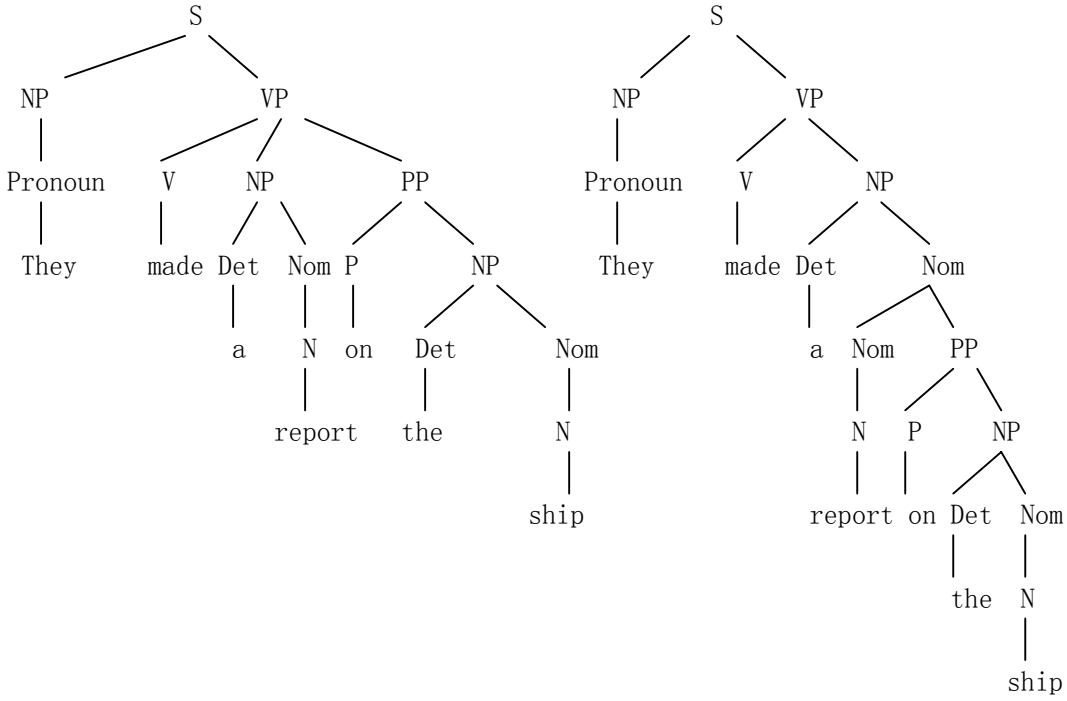
Structure ambiguity occurs when the grammar assigns more than one possible parse to sentence. Three common kinds of structure ambiguities are attachment ambiguity, coordination ambiguity and noun-phrase bracketing ambiguity.

3.3.2.1 Attachment ambiguity:

3.3.2.1.1 PP attachment ambiguity:

E.g.

1) They made a report about the ship.
   On the ship, they made a report.
   → They made a report on the ship.



PP is the modifier of V PP is the modifier of Nominal

Fig.9 PP attachment ambiguity

2) They made a decision concerning the boat.
   On the boat, they made a decision.
   → They made a decision on the boat.
3) He drove the car which was near the post office.
   Near the post office, he drove the car.
   → He drove the car near the post office.
4) They are walking around the lake which is situated in the park.
   In the park, they are walking around the lake.
   → They are walking around the lake in the park.
5) He shot at the man who was with a gun.

With a gun, he shot at the man.

→ He shot at the man with a gun.

6) The policeman arrested the thief who was in the room.

In the room, the policeman arrested the thief.

→ The policeman arrested the thief in the room.

Church and Patil (1982) showed that the number of parse for sentences of this type grows at the same rate as the number of parenthesization of arithmetic expressions. Such parenthesization problems (insertion problems) are known as grow exponentially in accordance with what are called the Catalan numbers:

$$C(n) = 1/n+1 \begin{pmatrix} 2n \\ n \end{pmatrix}$$

$$= \frac{1}{n+1} \times \frac{2n(2n-1)...(n+1)}{n!}$$

The following table shows the number of parses for a simple noun phrase as a function of the number of trailing prepositional phrases. We may see that this kind of ambiguity can very quickly make it imprudent to keep every possible parse around.

Number of PPs	Number of NP parses
2	2
3	5
4	14
5	21
6	132
7	429
8	1430
9	4867

Fig. 10

3.3.2.1.2 Gerundive attachment ambiguity:

E.g We saw the Eiffel tower flying to Paris.

The Gerundive phrase "flying to Paris" can modifies "saw" as the adverbial, it can also be the predicate in the clause "the Eiffel tower flying to Paris".

3.3.2.1.3 local ambiguity

Local ambiguity occurs when some part of a sentence is ambiguous, even if the whole sentence is not ambiguous. E.g. Sentence "book that flight" is unambiguous, but when the parser sees the first word "book", it can not know if it is a verb or a noun until later. Thus it must use backtracking or parallelism to consider both possible parses.

3.3.2.2 Coordination ambiguity (Ambiguity of 'and')

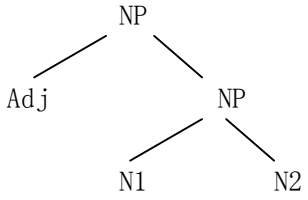
E,g

1) She looks care of old men and old women.
   She looks care of women and old men.
   → She looks care of old men and women.
2) Mr. John is a scientist of great fame and a professor of great fame.
   Mr. John is a professor of great fame and a scientist.
   → Mr. John is a scientist and a professor of great fame.
3) Someone tells me he's cheating, and I can't do anything about it.
   Someone tells me that he's cheating and that I can't do anything about it.
   → Someone tells me he's cheating and I can't do anything about it.
4) John will go, or Dick and Tom will go.
   John or Dick will go, and Tom will go.
   → John or Dick and Tom will go.

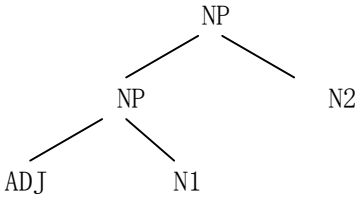
3.4.2.3 Noun-phrase bracketing ambiguity:

ADJ + N1 + N2

NP(ADJ(NP(N1 N2))):



NP(NP(ADJ N1)N2):

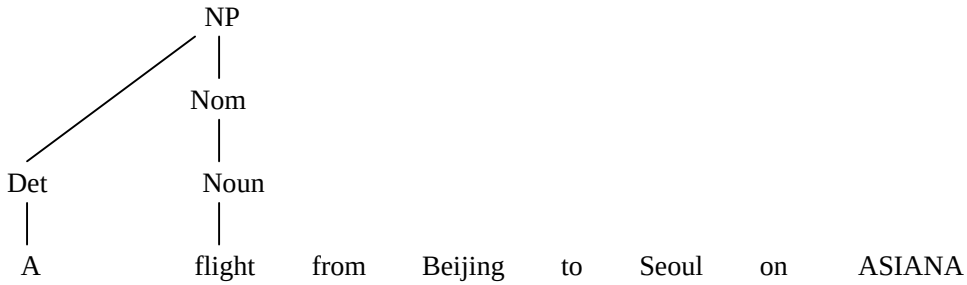


E.g.

1) The salesman who sells old cars is busy.
   The old salesman who sells cars is busy.
   → The old car salesman is busy.
2) He is a Department Head, who is from England.
   He is Head of the English Department.
   → He is an English Department Head.

3.3.3 Inefficient re-parsing of sub-tree

The parser often builds valid trees for portions of the input, then discards them during backtracking, only to find that it has to rebuild them again. The re-parsing of sub-tree is inefficient E,g. The noun phrase " a flight from Beijing to Seoul on ASIANA", its top-down parser process is as follows:



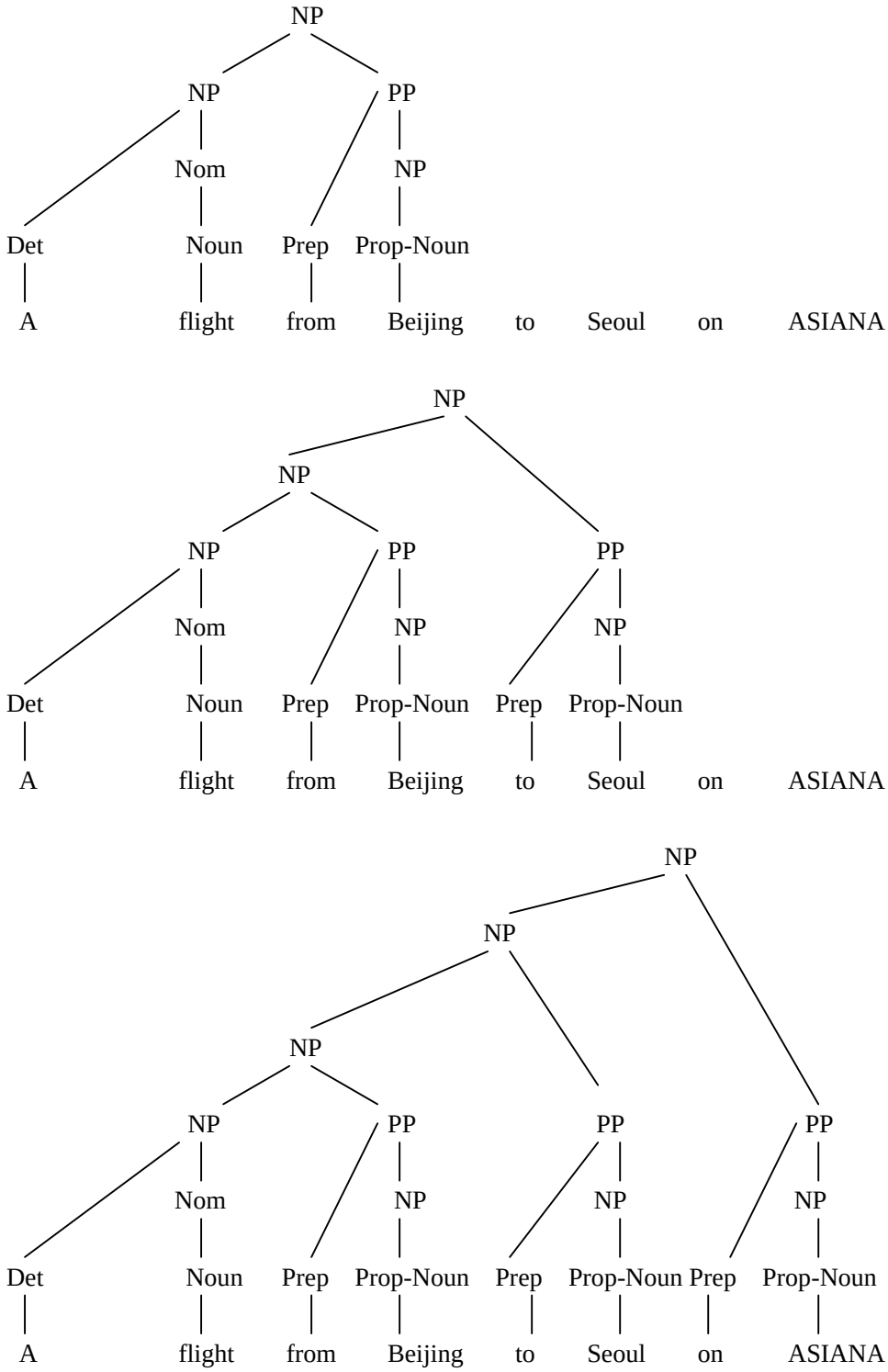


Fig. 11 Reduplication effort

Because of the way the rules are consulted in our top-down parsing, the parser is ;ed first to small parse trees that fail because they do not cover all the input. These successive failures trigger backtracking events which lead to parses that incrementally cover more and more of the input. In the backtracking, reduplication of work arises many times. Except for its topmost component, every part of the final tree is derived more than once in the backtracking process.

Component	reduplication times
A flight	4
From Beijing	3
To Seoul	2
On ASIANA	1
A flight from Beijing	3
A flight from Beijing to Seoul	2
A flight from Beijing to Seoul on ASIANA	1

Similar example of wasted effort also exists in the bottom-up parsing.

3.4 Some Algorithms

3.4.1 Earley algorithm

In order to solve the problems in parsing, Earley (1970) proposes Earley algorithm.

3.4.1.1 Chart and dotted rule

The core of Earley algorithm is chart. For each word position in the sentence, the chart contains a list of states representing that have been generated so far. By the end of the sentence, the chart compactly encodes all the possible parses of the input.

The state within each chart contain three kinds of information:

- A sub-tree corresponding to the single grammar rule;
- Information about the progress made in completing this sub-tree;
- Information about the position of the sub-tree with respect to the input.

These information is represented by a dotted rule. In the dotted rule, a state position with respect to the input is representd by two numbers indicating where the state begins and where its dot lies.

E.g.

The three rules which using to parser “book that flight” are as follows:

S → VP

NP → Det Nominal

VP → V NP

Some dotted rules of these three rules can be represented as follows:

S → **.**VP, [0,0]

NP → Det**.** Nominal, [1,2]

VP → V NP**.**, [0,3]

The state represented by these dotted rules can be expressed by following chart:

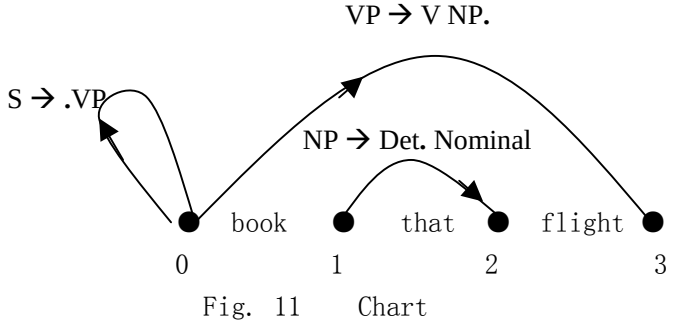


Fig. 11 Chart

This chart is a directed acyclic graph (DAG).

3.4.1.2 Three operators in Early algorithm

■ Predictor

The job of predictor is to create new state representing top-down expectations generated during the parsing process. The predictor is applied to any state that has a non-terminal to the right of the dot. This application results in the creation of one new state for each alternative expansion of that non-terminal provided by the grammar. These new states are placed into the same chart entry as the generated state. They begin and end at the point on the input where the generating state ends.:

E.g. applying the predicator to the state S → **.**VP, [0,0] results in adding the states VP → **.**Verb, [0,0] and VP → **.**Verb NP, [0,0] to the first chart entry.

■ Scanner

When a state has a POS category to the right of the dot, the scanner is called to examine the input and incorporate a state corresponding to the predicated POS into the chart. This is accomplished by creating a new state from the input state with the dot advanced over the predicted input category.

E.g. When the state VP → **.**Verb NP, [0,0] is processed, the Scanner consults the current word in the input since the category following the dot is a POS. The Scanner then notes that "book" can be a verb, matching the expectation in the current state. This results in the creation of new state VP → Verb**.** NP, [0,1]. The new state is then added to the chart entry that follows the one currently being processed.

■ Completer

The completer is applied to a state when its dot has reached the right end of the rule. Intuitively, the presence of such a state represents the fact that the parser has successfully discovered a particular grammatical category over some span of the input. The purpose of the completer is to find and advance all previously created states that were looking for this grammatical category at this position in the input. New states are then created by copying the old state, advancing the dot over expected category and installing the new state in the current chart entry.

E.g. When the state NP → Det Nominal**.**, [1,3] is processed, the completer looks for state ending at 1 expecting an NP. In the current example, it will find the state VP → Verb**.** NP, [0,1] created by the Scanner. This results in the addition of new completer state VP → Verb NP**.**, [0,3].

Martin Kay improved Early algorithm and proposed the fundamental rule of chart parsing.

Strictly, the fundamental rule of chart parsing is as following:

If the chart contains edges <A→W1.B W2, [i,j]> and <B→W3., [j,k]>, where A and B are categories and W1, W2 and W3 are (possibly empty) sequences of categories or words, then add edge <A→W1 B.W2, [i,k]> to the chart.

This fundamental rule can be represented by DAG:

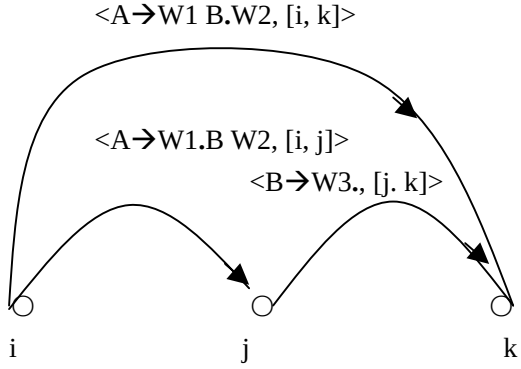


Fig. 12 fundamental rule of chart parsing

3.4.1.3 An example fro Early algorithm

The state sequence in chart while parsing “book that flight” using our small grammar:

	Chart [0]	
ɣ → .S	[0,0]	Dummy start state
S → .NP VP	[0,0]	Predictor
NP → .Det Nominal	[0,0]	Predictor
NP → .Proper-Noun	[0,0]	Predictor
S → .Aux NP VP	[0,0]	Predictor
S → .VP	[0,0]	Predictor
VP → .Verb	[0,0]	Predictor
VP → .Verb NP	[0,0]	Predictor
	Chart [1]	
Verb → book.	[0.1]	Scanner
VP → Verb.	[0,1]	Completer
S → VP.	[0,1]	Completer
VP → Verb. NP	[0,1]	Completer
NP → .Det Nominal	[1,1]	Predictor
NP → .Proper-Noun	[1,1]	Predictor
	Chart [2]	
Det → that.	[1,2]	Scanner
NP → Det. Nominal	[1,2]	Completer
Nominal → .Noun	[2,2]	Predictor
Nominal → .Noun Nominal	[2,2]	Predictor
	Chart [3]	
Noun → flight.	[2,3]	Scanner
Nominal → Noun.	[2,3]	Completer
Nominal → Noun. Nominal	[2,3]	Completer
NP → Det Nominal.	[1,3]	Completer
VP → Verb NP.	[0,3]	Completer
S → VP.	[0,3]	Completer
Nominal → .Noun	[3,3]	Predictor
Nominal → .Noun Nominal	[3,3]	Predictor

In chart [3], the presence of the state representing “flight” leads to completion of NP, transitive VP, and S. The presence of the state S → VP., [0,3] in the last chart entry means that our parser gets the success.

3.4.1.4 Retrieving parser trees from a Chart

The Earley algorithm just described is actually a recognizer not a parser. After processing, valid sentences will leave the state S → α., [0,N] (N is the word number in the sentence)

To return this algorithm into a parser, we must be able to extract individual parses from the chart. To do this, the representation of each state must be augmented with an additional field to store information about the completed states that generated its constituents.

Recall that the Completer creates new states by advancing older incomplete ones when the constituent following the dot is discovered. The only change necessary is to have Completer add a pointer to the older state onto the list of the previous states of the new state. Retrieving a parse tree from the chart is then merely a recursive retrieval starting with the state (or states) representing a complete S in the final chart entry. Following shows the chart produced by an updated completer.

Chart [0]

S0 γ → .S	[0,0]	[]	Dummy start state
S1 S → .NP VP	[0,0]	[]	Predictor
S2 NP → .Det Nominal	[0,0]	[]	Predictor
S3 NP → .Proper-Noun	[0,0]	[]	Predictor
S4 S → .Aux NP VP	[0,0]	[]	Predictor
S5 S → .VP	[0,0]	[]	Predictor
S6 VP → .Verb	[0,0]	[]	Predictor
S7 VP → .Verb NP	[0,0]	[]	Predictor

Chart [1]

S8 Verb → book.	[0.1]	[]	Scanner
S9 VP → Verb.	[0,1]	[S8]	Completer
S10 S → VP.	[0,1]	[S9]	Completer
S11 VP → Verb. NP	[0,1]	[S8]	Completer
S12 NP → .Det Nominal	[1,1]	[]	Predictor
S13 NP → .Proper-Noun	[1,1]	[]	Predictor

Chart [2]

S14 Det → that.	[1,2]	[]	Scanner
S15 NP → Det. Nominal	[1,2]	[S14]	Completer
S16 Nominal → .Noun	[2,2]	[]	Predictor
S17 Nominal → .Noun Nominal	[2,2]	[]	Predictor

Chart [3]

S18 Noun → flight.	[2,3]	[]	Scanner
S19 Nominal → Noun.	[2,3]	[S18]	Completer
S20 Nominal → Noun. Nominal	[2,3]	[S18]	Completer
S21 NP → Det Nominal.	[1,3]]	[S14, S19]	Completer
S22 VP → Verb NP.	[0,3]	[S8, S21]	Completer
S23 S → VP.	[0,3]	[S22]	Completer
S24 Nominal → .Noun	[3,3]	[]	Predictor
S25 Nominal → .Noun Nominal	[3,3]	[]	Predictor

The parsing process can be summarized as follows:

S8 Verb → book.	[0.1]	[]	Scanner
S9 VP → Verb.	[0,1]	[S8]	Completer
S10 S → VP.	[0,1]	[S9]	Completer
S11 VP → Verb. NP	[0,1]	[S8]	Completer
S14 Det → that.	[1,2]	[]	Scanner
S15 NP → Det. Nominal	[1,2]	[S14]	Completer
S18 Noun → flight.	[2,3]	[]	Scanner
S19 Nominal → Noun.	[2,3]	[S18]	Completer
S20 Nominal → Noun. Nominal	[2,3]	[S18]	Completer
S21 NP → Det Nominal.	[1,3]]	[S14, S19]	Completer
S22 VP → Verb NP.	[0,3]	[S8, S21]	Completer
S23 S → VP.	[0,3]	[S22]	Completer

The DAG representing the parse is as follows:

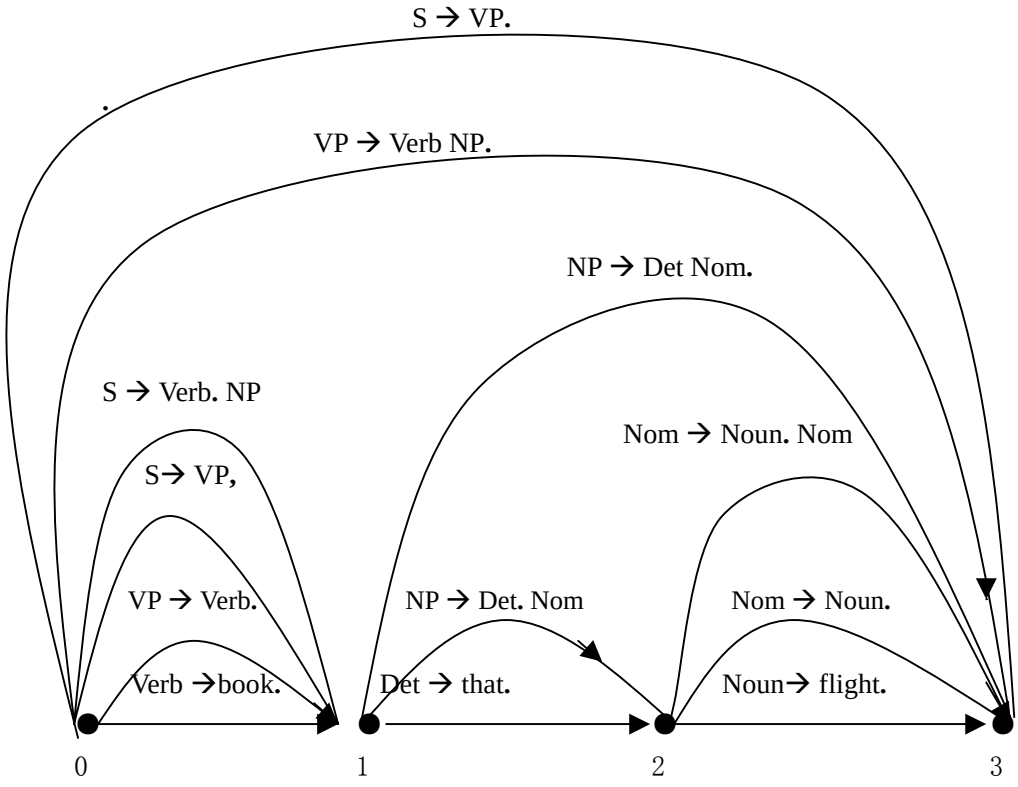


Fig. 13 DAG representing the parsing result

3.4.1.5 Another examples:

Example-1: Using Early algorithm to parse sentence "Does KA 852 have a first class section?"

In this sentence, "first" is "ord", so we shall add a new rule in our small grammar

NP → Ord Nom

The states are as follows:

● Does ● KA 852 ● have ●first ● class ● section ●

0 1 2 3 4 5 6

The state sequence in chart:

Chart [0]

ɣ → .S	[0,0]	Dummy start state
S → .NP VP	[0,0]	Predictor
NP → .Ord Nom	[0,0]	Predictor
NP → .PrN	[0,0]	Predictor
S → .Aux NP VP	[0,0]	Predictor
S → .VP	[0,0]	Predictor
VP → .V	[0,0]	Predictor
VP → .V NP	[0,0]	Predictor

Chart [1]

Aux → does.	[0,1]	Scanner
S → Aux. NP VP	[0,1]	Completer
NP → .Ord Nom	[1,1]	Predictor
NP → .PrN	[1,1]	Predictor

Chart [2]

PrN → KA 852.	[1,2]	Scanner
NP → PrN.	[1,2]	Completer
S → Aux NP. VP	[0,2]	Completer
VP → .V	[2,2]	Predictor
VP → .V NP	[2,2]	Predictor

Chart [3]

V → have.	[2,3]	Scanner
VP → V.	[2,3]	Completer
VP → V. NP	[2,3]	Completer
NP → ..Ord Nom	[3,3]	Predictor

Chart [4]

Ord → first.	[3,4]	Scanner
NP → Ord. Nom	[3,4]	Completer
Nom → .N Nom	[4,4]	Predictor
Nom → .N.	[4,4]	Predictor
Nom → .N PP	[4,4]	Predictor

Chart [5]

N → class.	[4,5]	Scanner
Nom → N.	[4,5]	Completer
NP → Ord Nom.	[3,5]	Completer
VP → V NP.	[2,5]	Completer
S → Aux NP VP.	[0,5]	Completer (S' span is 5, 5 < 6)
Nom → N. Nom	[4,5]	Completer

Nom → .N	[5,5]	Predictor
	Chart [6]	
N → section.	[5,6]	Scanner
Nom → N.	[5,6]	Completer
Nom→ N Nom.	[4,6]	Completer
NP → Ord Nom.	[3,6]	Completer
VP → V NP.	[2,6]	Completer
S → Aux NP VP.	[0,6]	Completer
	[Success !]	

The parsing process:

Aux → does.	[0,1]	Scanner
S → Aux. NP VP	[0,1]	Completer
PrN → KA 852.	[1,2]	Scanner
NP → PrN.	[1,2]	Completer
S → Aux NP. VP	[0,2]	Completer
V → have.	[2,3]	Scanner
VP → V.	[2,3]	Completer
VP → V. NP	[2,3]	Completer
Ord → first.	[3,4]	Scanner
NP → Ord. Nom	[3,4]	Completer
N → class.	[4,5]	Scanner
N → section.	[5,6]	Scanner
Nom → N.	[5,6]	Completer
Nom→ N Nom.	[4,6]	Completer
NP → Ord Nom.	[3,6]	Completer
VP → V NP.	[2,6]	Completer
S → Aux NP VP.	[0,6]	Completer
	[Success !]	

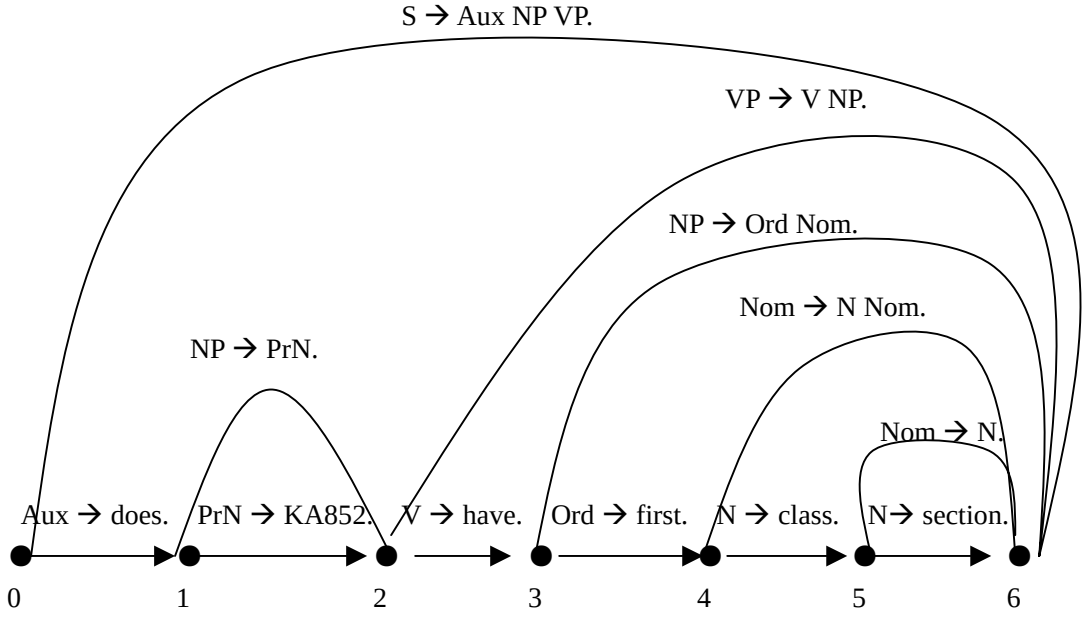


Fig. 14

Example-2: using Earley algorithm to parse sentence “It is a flight from Beijing to Seoul on ASIANA”

The states:

● it ● is ● a ● flight ● from ● Beijing ● to ● Seoul ● on ● ASIANA●

0 1 2 3 4 5 6 7 8 9 10

“It” is a pronoun, so we need to add a new rule in our grammar:

NP → Pron

and

PP → Prep NP

The state sequence is as follows:

Chart [0]

ɣ → .S	[0,0]	Dummy start state
S → .NP VP	[0,0]	Predictor
NP → .Pron	[0,0]	Predictor
NP → .PrN	[0,0]	Predictor
S → .Aux NP VP	[0,0]	Predictor
S → .VP	[0,0]	Predicator
VP → .V	[0,0]	Predicator
VP → .V NP	[0,0]	Predicator

Chart [1]

Pron → it	[0,1]	Scanner
NP → Pron.	[0,1,]	Completer
S → NP. VP	[0,1]	Completer
VP → .V	[1,1]	Predictor
VP → .V NP	[1,1]	Predictor

Chart [2]

V → is.	[1,2]	Scanner
VP → V.	[1,2]	Completer
S → NP VP.	[0,2]	Completer (S’ span is 2 < 10)
VP → V. NP	[1,2]	Completer
NP → .Det Nom	[2,2]	Predictor

Chart [3]

Det → a.	[2,3]	Scanner
NP → Det. Nom	[2,3]	Completer
Nom → N	[3,3]	Predictor
Nom → .N Nom	[3,3]	Predictor
Nom → .Nom PP	[3,3]	Predictor

Chart [4]

N → flight.	[3,4]	Scanner
Nom → N.	[3,4]	Completer

NP → Det Nom**.** [2,4] Completer

VP → V NP**..** [1,4] Completer

S → NP VP**.** [0,4] Completer (S' span is 4 < 10)

Nom → N**.** Nom [3,4] Completer

Attention: behind N, no Nom. So the process turns to following state:

Nom → Nom**.** PP [3,4] Completer

PP → **.**Prep NP [4,4] Predictor

Chart [5]

Prep → from. [4,5] Scanner

PP → Prep**.** NP [4,5] Completer

NP → **.**PrN [5,5] Predictor

Chart [6]

PrN → Beijing**.** [5,6] Scanner

NP → PrN. [5,6] Completer

PP → Prep NP [4,6] Completer

Nom → Nom PP**.** [3,6] Completer

Attention: The dot behind PP (this PP = "from Beijing"), it is inactive edge.

Nom → Nom**.** PP [3,6] Completer

Attention: The dot in front of PP (this PP = "to Seoul"), it is active edge.

PP → **.**Prep NP [6,6] Predictor

Chart [7]

Prep → to**.** [6,7] Scanner

PP → Prep**.** NP [6,7] Completer

NP → **.**PrN [7,7] Predictor

Chart [8]

PrN → Seoul**.** [7,8] Scanner

NP → PrN**.** [7,8] Completer

PP → Prep NP**.** [6,8] Completer

Nom → Nom PP**.** [3,8] Completer

Attention: The dot behind PP (this PP = "to Seoul"), it is inactive edge.

Nom → Nom**.** PP [3,8] Completer

Attention: The dot in front of PP (this PP = "on ASIANA"), it is active edged.

PP → **.**Prep NP [8,8] Predictor

Chart [9]

Prep → on**.** [8,9] Scanner

PP → Prep**.** NP [8,9] Completer

NP → **.**PrN [9,9] Predictor

Chart [10]

PrN → ASIANA.	[9,10]	Scanner
NP→PrN.	[9,10]	Completer
PP → Prep NP.	[8,10]	Completer
Nom → Nom PP.	[3,10]	Completer
NP → Det Nom.	[2,10]	Completer
VP → V NP	[1,10]	Completer
S → NP VP.	[0,10]	Completer

[Success !]

The parsing process:

Pron → it	[0,1]	Scanner
NP → Pron.	[0,1,]	Completer
S → NP. VP	[0,1]	Completer
V → is.	[1,2]	Scanner
VP → V. NP	[1,2]	Completer
Det → a.	[2,3]	Scanner
NP → Det. Nom	[2,3]	Completer
N → flight.	[3,4]	Scanner
Nom → N.	[3,4]	Completer
NP → Det Nom.	[2,4]	Completer
Nom → Nom. PP	[3,4]	Completer
Prep → from.	[4,5]	Scanner
PP → Prep. NP	[4,5]	Completer
PrN → Beijing.	[5,6]	Scanner
NP → PrN.	[5,6]	Completer
PP → Prep NP	[4,6]	Completer
Nom → Nom PP.	[3,6]	Completer

Attention: The dot behind PP (this PP = “from Beijing”), it is inactive edge.

Nom → Nom. PP	[3,6]	Completer

Attention: The dot in front of PP (this PP = “to Seoul”), it is active edge.

Prep → to.	[6,7]	Scanner
PP → Prep. NP	[6,7]	Completer
PrN → Seoul.	[7,8]	Scanner
NP → PrN.	[7,8]	Completer
PP → Prep NP.	[6,8]	Completer
Nom → Nom PP.	[3,8]	Completer

Attention: The dot behind PP (this PP = “to Seoul”), it is inactive edge.

Nom → Nom. PP	[3,8]	Completer

Attention: The dot in front of PP (this PP = “on ASIANA”), it is active edged.

Prep → on.	[8,9]	Scanner
PP → Prep. NP	[8,9]	Completer
PrN → ASIANA.	[9,10]	Scanner
NP→PrN .	[9,10]	Completer

PP → Prep NP .	[8,10]	Completer
Nom → Nom PP.	[3,10]	Completer
NP → Det Nom.	[2,10]	Completer
VP → V NP	[1,10]	Completer
S → NP VP.	[0,10]	Completer

[Success !]

The chart:

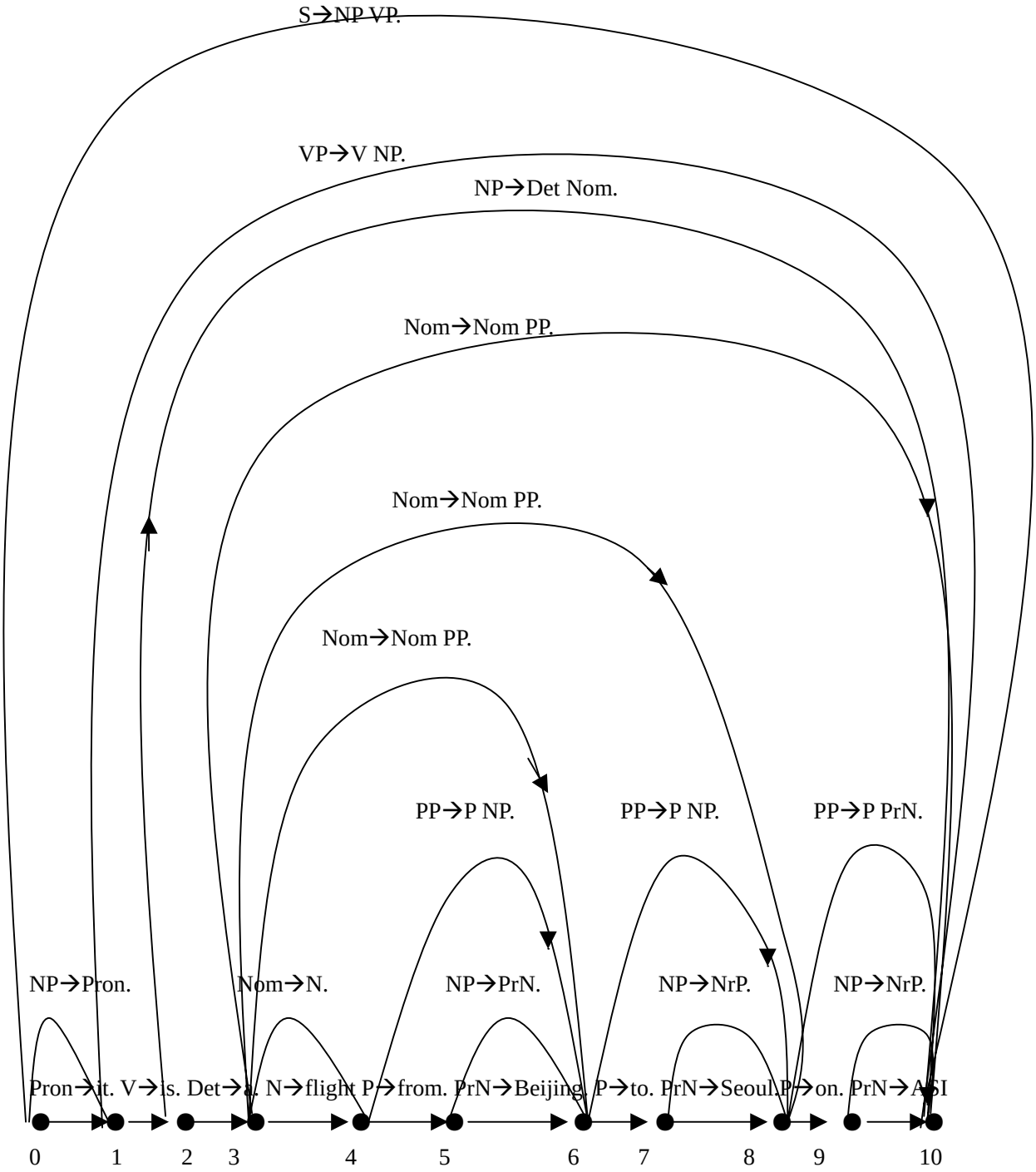


Fig. 15

In this parsing process, there is not backtracking as in the top-down parsing. The advantage of Early algorithm is obvious.

3.4.2. CYK approach:

CYK approach is abbreviation of Cocke-Younger-Kasami approach. It is a parallel parsing algorithm.

3.4.2.1 Table and box in CYK approach

If we have a CFG as follows:

S → NP VP
NP → Det N
VP → V NP

Obviously, this is a CFG with Chomsky Nornal Form because the form all the rules is A → BC. Following table can expresses the result of CYK parsing for the sentence “the boy hits a dog”:

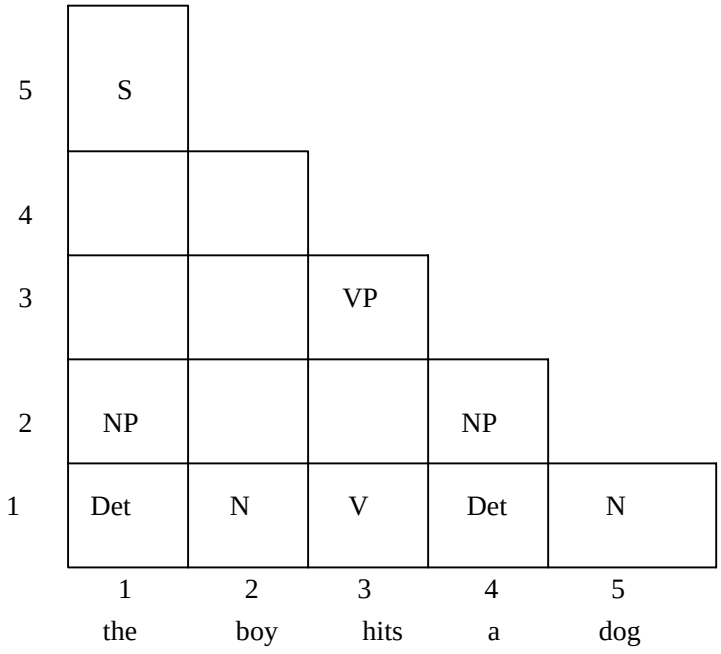


Fig. 16

In this table, the row number expresses the location of word in the sentence, the line number expresses the word number included in the grammatical category (e.g. N, V, NP, VP, S, etc). All the category is located in the box of the table. b_{ij} expresses the box that located in the row i and line j. Every grammatical category in the table can be expressed by b_{ij}.

‘Det belongs to b_{11}’ means : Det is located in row 1 and line 1.
‘N belongs to b_{21}’ means : N is located in row 2 and line 1.
‘V belongs to b_{31}’ means: V is located in row 3 and line 1.
‘Det belongs to b_{41}’ means; Det is located in row 4 and line 1.
‘N belongs to b_{51}’ means; N is located in row 5 and line 1.

By this reason,

The location of NP (the boy) is b_{12} (including 2 words),

The location of NP (a dog) is $b_{4\,2}$ (including 2 words),

The location of VP (hits a dog) is $b_{3\,3}$ (including 3 words),

The location of S (the boy hits a dog) is $b_{1\,5}$ (including 5 words).

Obviously, the table and the $b_{i\,j}$ in the table can describe the structure of the sentence. For every category $b_{i\,j}$, i describes its location in the sentence structure, j describes the word number included in this category. If we may create the table and the $b_{i\,j}$ in the table, the parsing is completed.

3.4.2.2 CYK Description of Chomsky Normal Form

In the Chomsky normal form A → BC,

if B belongs to $b_{i\,k}$, C belongs to $b_{i+k\,j-k}$,

Then A must belong to $b_{i\,j}$.

If we start from i-th word of the sentence create a sub-tree B including k words, and then from i+k-th word of the sentence create a sub-tree C including j-k words, then the tree graph A can be expressed as follows:

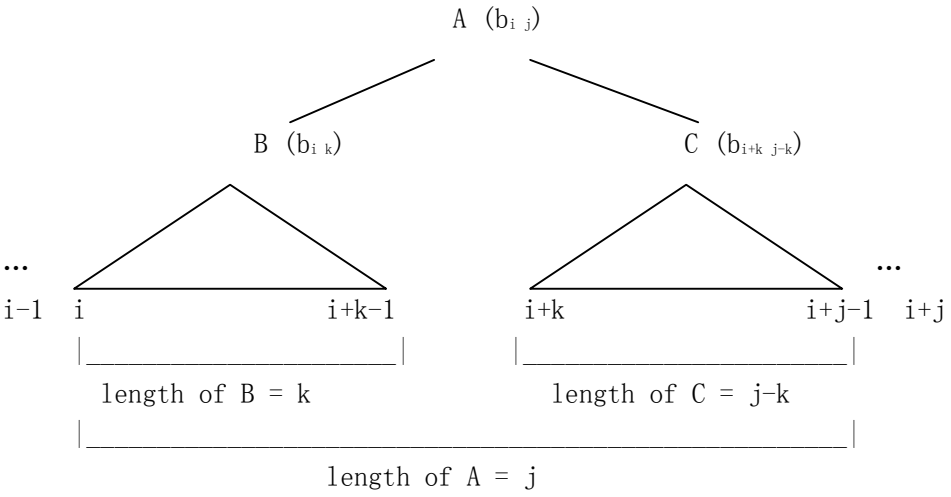


Fig. 17

For example, in Fig. 17, NP belongs to $b_{1\,2}$, Det belongs to $b_{1\,1}$, N belongs to $b_{2\,1}$, is represents the Chomsky normal form NP → Det N. In this case, i=1, k=1, j=2. Therefore, if we know the starting number i of B, the length k of B, the length j of A, then we can calculate the location of A, B and C in the CYK table: A belongs to $b_{i\,j}$, B belongs to $b_{i\,k}$, C belongs to $b_{i+k\,j-k}$.

In CYK approach, the important problem is how to calculate the location of A. The row number of A is always same as that of B, so if row number of B is i, then the row number of A must be i. The line number of A (=j) equals to the addition of the line number of B (=k) and the line number of C (=j-k): j = k + j - k.
Therefore, If we know the location of B and the location of C, it is easy to calculate the location of A.

If the length of input sentence is n, the CYK algorithm can be divided to two steps:

First step: start from i = 1, for every words Wi in input sentence (with length n), we have rewriting rule A → Wi, so we write the non-terminal symbol A of every Word Wi in the box of our table, and give the location number of box with b_{ij}. E.g, for sentence "The boy hits a dog", we give the location number respectively for every words of sentence is as follows: b_{11} (for Det [non-terminal symbol of 'the']), b_{21} (for N [non-terminal symbol of 'boy']), b_{31} (for V [non-terminal symbol of 'hits']), b_{41} (for second Det [non-terminal symbol of 'a']), b_{51} (for second N [non-terminal symbol of 'dog']).

Second step; For $1 \leqslant h \leqslant j$ and all i, create b_{ih}. non-terminal set including b_{ij} can be defined as follows:

b_{ij} = {A |for $1 \leqslant k \leqslant j$, B is included in b_{ik}, C is included in $b_{i+k\ j-k}$, and exists grammar rule A → BC that A is included in b_{ij}}.

If box b_{1n} includes initial symbol S, then input sentence will be accepted. The analysis gets success.

E.g, for rule 'NP → Det N' and Det belongs to b_{11}, N belongs to b_{21}, we can confirm that NP belongs to b_{12};

for rule 'NP → Det N' and 'Det" belongs to b_{41}, N belongs to b_{51}, we can confirm that NP belongs to b_{42};

for rule VP → V NP and V belongs to b_{31}, NP belongs to b_{42}, we can confirm that VP belongs to b_{33};

for rule S → NP VP and NP belongs to b_{12}, VP belongs to B_{33}, we can confirm that S belongs to b_{15}. In our input sentence, n=5, so our sentence is accepted.

3.4.2.3 A complex example for CYK algorithm

If the PSG grammar is as follows:

S → NP VP
NP → PrN
NP → DET N
NP → N WH VP
NP → DET N WH VP
VP → V
VP → V NP
VP → V that S

Use CYK approach to analyze sentence 'the table that lacks a leg hits Jack".

■ Transformation of rewriting rules to Chomsky normal form:

S → NP VP
NP → PrN It is not CNF and must be transformed to:
NP → Jack | John | Maria
NP → DET N
NP → N WH VP It must be transformed to:
NP → N CL
CL → WH VP
NP → DET N WH VP It must be transformed to:
NP → NP CL

NP → DET N

CL → WH VP

Here CL is WH clause, it = (that + VP)

VP → V It is not CNF and must be transformed to:

VP → cough | walk | …

VP → V NP

VP → V that S It must be transformed to:

VP → V TH

TH → WH S

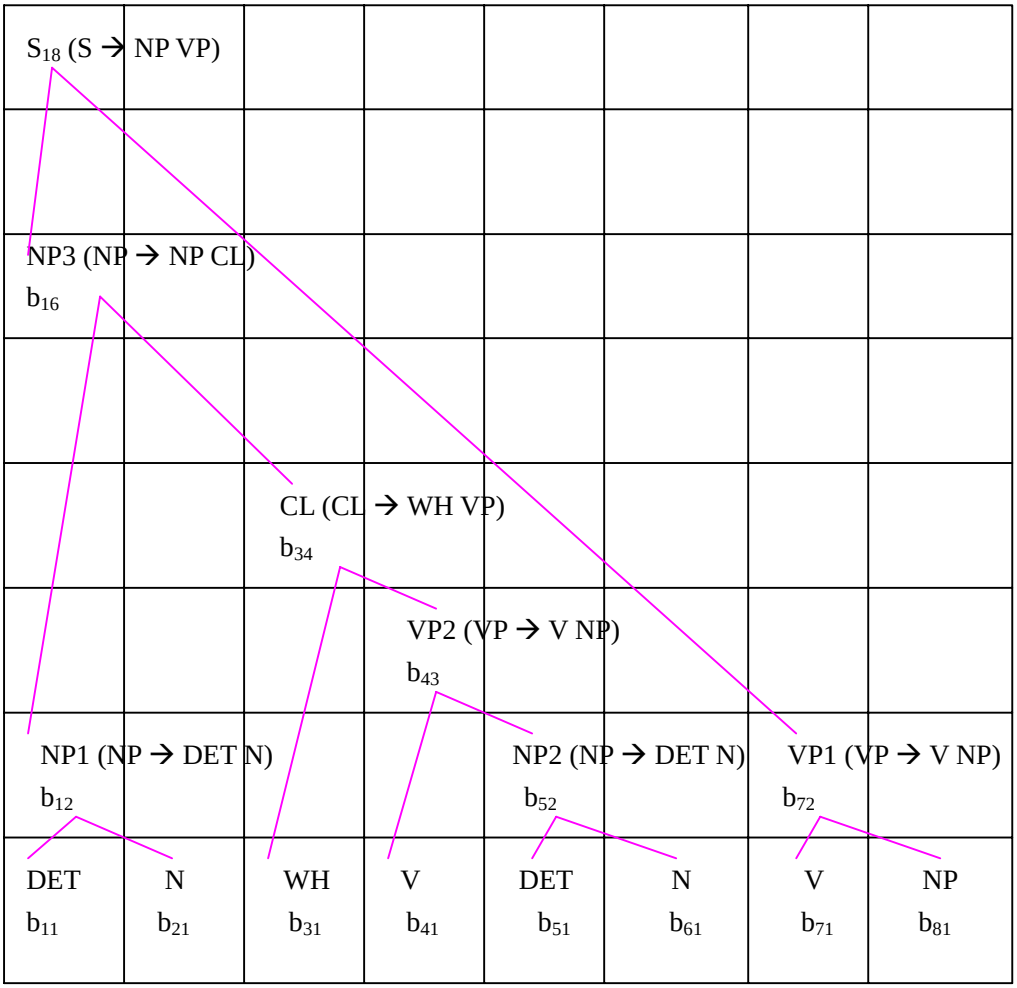
Here TH is that-clause, it = (that + S).

■ Calculation of the b_{ij} of non-terminal symbols:

--To arrange POS non-terminal symbols and calculate their b_{ij}

"The	table	that	lacks	a	leg	hits	Jack"
DET	N	WH	V	DET	N	V	PrN (NP)
b_{11}	b_{21}	b_{31}	b_{41}	b_{51}	b_{61}	b_{71}	b_{81}

--To calculate the b_{ij} of phrase non-terminal symbols



The table that lacks a leg hits Jack

Fig. 18

b_{ij} (NP1): i=1. j=1+1=2

b_{ij} (NP2): i=5, j=1+1=2

b_{ij} (VP1): i=7, j=1+1=2

b_{ij} (VP2): i=4, j=1+2=3

b_{ij} (CL): i=3, j=1+3=4

b_{ij} (NP3): i=1, j=2+4=6

b_{ij} (S): i=1, j=2+6=8

The length of this sentence is 8, and we get box line number of S is also 8, so the sentence was recognized.

By the CYK approach, we can create the pyramid in Fig. 15. This pyramid is also a tree graph.

3.4.2.5 another example

Now we use CYK to parse the sentence “book that flight”.

If the rules of our CFG are as above rules which we used to parse this sentence:

1. S → VP
2. VP → Verb NP
3. NP → Det Nominal
4. Nominal → Noun

The form of rule-1 is not CNF because its RHS includes only one single Non-terminal VP. Therefore we have to combine rule-1 with rule-2 which has CNF. In this case, rule-1 and rule-2 can be changes as following CNF:

S → Verb NP

The form rule-4 is not CNF because its RHS includes only one single Non-terminal symbol. Therefore we have to combine rule-4 with rule-3 which has CNF. In this case, rule-4 and rule-3 can be changed as following CNF:

NP → Det Noun

Now the rules with CNF can be:

S → Verb NP

NP → Det Noun

The CYK result of this sentence can be represented in following table:

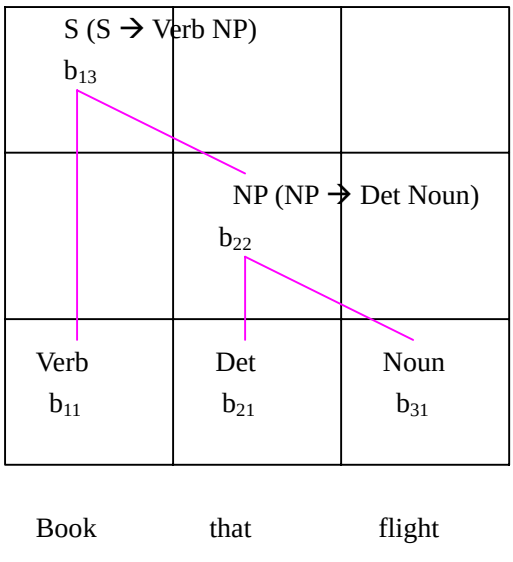


Fig. 19

b_{ij} (NP): i=2,. j=1+1=2

b_{ij} (S): i=1, j=1+2=3

We can also create the pyramid in the table of CYK. This pyramid is similar as a tree graph. We can see the CYK algorithm is so simple and effective.