自然语言处理

week-4

凌震华

2024年3月21日

- Syntax (句法) and Grammer (语法)
  - CFG, Dependency Grammer, TreeBanks
- Parsing (剖析)
  - CKY Parsing Algorithm
- Partial Parsing
  - Rule-Based / Machine Learning-Based

Syntax

- syntax（句法)
  - The way words are arranged together

- Why should you care?
  - Grammar checkers
  - Question answering
  - Information extraction
  - Machine translation
  - Search: people type queries in English rather than keywords

Context-Free Grammars
上下文无关语法

- A set of rules (规则) or productions (产生式)
- A lexicon (词表) of words and symbols
  - Non-terminal (非终极符号) / terminal (终极符号) symbols
  - A start symbol
- Capture ordering and constituency
  - Ordering (顺序性)

    What are the rules that govern the ordering of words and bigger units in the language
  - Constituency (组成性)

    How words group into units and how the various kinds of units behave with one another

CFG Examples

- S -> NP VP
- NP -> Det NOMINAL
- NOMINAL -> Noun
- VP -> Verb
- Det -> *a*
- Noun -> *flight*
- Verb -> *left*

CFGs

- S -> NP VP
  - This says that there are units called S, NP, and VP in this language
  - That an S consists of an NP followed immediately by a VP
  - Doesn't say that that's the only kind of S
  - Nor does it say that this is the only place that NPs and VPs occur

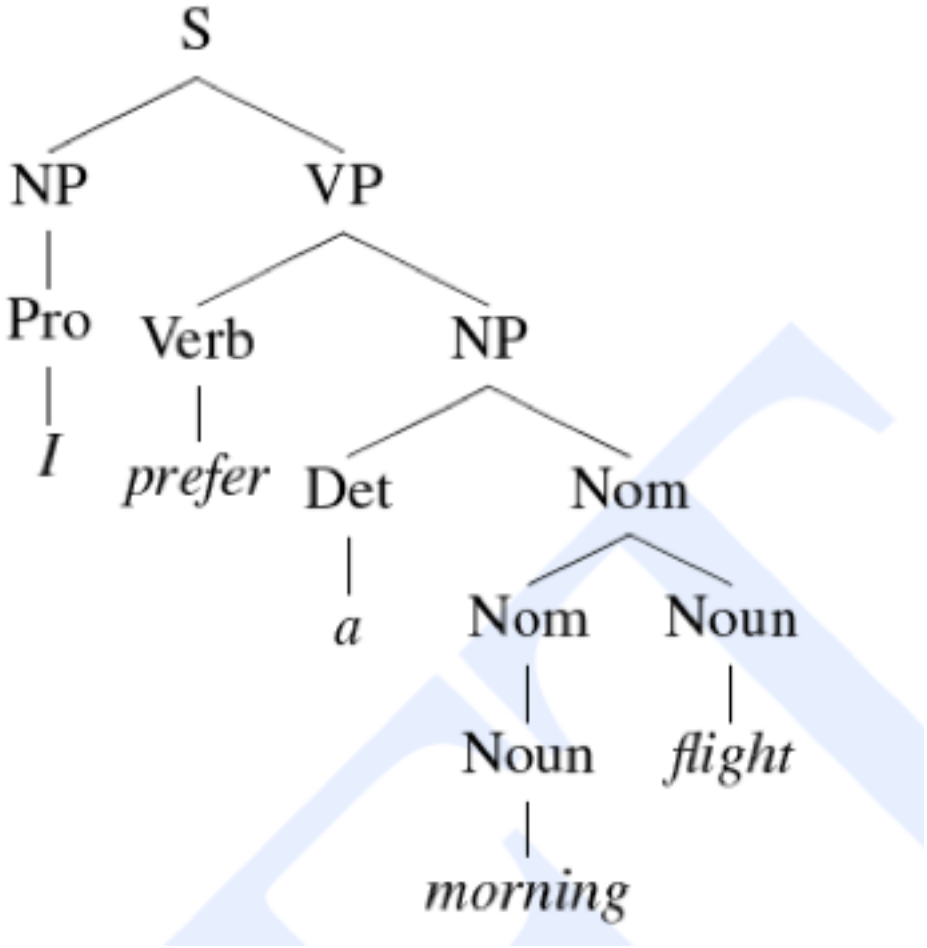
Generativity

- As with FSAs you can view these rules as either analysis or synthesis machines
  - Generate strings in the language
  - Reject strings not in the language
  - Impose structures (trees) on strings in the language

Derivations

- A derivation (推导) is a sequence of rules applied to a string that accounts for that string
  - Covers all the elements in the string
  - Covers only the elements in the string

Derivations as Trees



Parsing

- Parsing (剖析) is the process of taking a string and a grammar and returning a (many?) parse tree(s) (剖析树) for that string

Context-Free

- Context-Free in CFGs means is that the non-terminal on the left-hand side of a rule is out there all by itself (free of context)

  A -> B C

  Means that

  – I can rewrite an A as a B followed by a C regardless of the context in which A is found
  – Or when I see a B followed by a C I can infer an A regardless of the surrounding context

Key Constituents (English)

- Sentences 句子
- Noun phrases 名词短语
- Verb phrases 动词短语
- Prepositional phrases 介词短语
- … …

Sentences, Clauses, Phrases

- “Complete thought”
  - A node whose verb has all of its “arguments”
- Sentences (句子) (and some clauses) are complete thoughts
- Clauses (子句) have a verb
  - “When I reached home”
- Phrases (短语) are very general groupings
  - Need not have subject nor verb
  - “After dinner”

Sentence-Types

- Declaratives 陈述式: A plane left
  S -> NP VP
- Imperatives 命令式: Leave!
  S -> VP
- Yes-No Questions: Did the plane leave?
  S -> Aux NP VP
- WH Questions: When did the plane leave?
  S -> WH Aux NP VP

Recursion (递归)

- We’ ll have to deal with rules such as the following where the non-terminal on the left also appears somewhere on the right (directly).

  Nominal -> Nominal PP [[flight] [to Boston]]

  VP -> VP PP [[departed Miami] [at noon]]

Recursion

- Of course, this is what makes syntax interesting
  - Flights from Denver
  - Flights from Denver to Miami
  - Flights from Denver to Miami in February
  - Flights from Denver to Miami in February on a Friday
  - Flights from Denver to Miami in February on a Friday under $300
  - Flights from Denver to Miami in February on a Friday under $300 with lunch

Recursion

- Of course, this is what makes syntax interesting

  Flights from Denver

  [[Flights] [from Denver]]

  [[[Flights] [from Denver]] [to Miami]]

  [[[[Flights] [from Denver]] [to Miami]] [in February]]

  [[[[[Flights] [from Denver]] [to Miami]] [in February]] [on a Friday]]

  etc.

Problems

- Agreement (一致关系)
- Subcategorization (次范畴化)
- Movement (移位)

Agreement

- This dog
- Those dogs

- This dog eats
- Those dogs eat

- *This dogs
- *Those dog

- *This dog eat
- *Those dogs eats

Possible CFG Solution

- S -> NP VP
- NP -> Det Nominal
- VP -> V NP
- …

- SgS -> SgNP SgVP
- PlS -> PlNp PlVP
- SgNP -> SgDet SgNom
- PlNP -> PlDet PlNom
- PlVP -> PlV NP
- SgVP ->SgV NP
- …

CFG Solution for Agreement

- It works and stays within the power of CFGs
- But its ugly
- And it doesn't scale all that well

Subcategorization

- Sneeze: John sneezed
- Find: Please find [a flight to NY]$_{NP}$
- Give: Give [me]$_{NP}$[a cheaper fare]$_{NP}$
- Help: Can you help [me]$_{NP}$[with a flight]$_{PP}$
- Prefer: I prefer [to leave earlier]$_{TO\text{-}VP}$
- Told: I was told [United has a flight]$_{S}$
- …

Subcategorization

- *John sneezed the book
- *I prefer United has a flight
- *Give with a flight

- Subcat expresses the constraints that a predicate (谓语) (verb for now) places on the number and syntactic types of arguments it wants to take (occur with).

So?

- So the various rules for VPs overgenerate
  - They permit the presence of strings containing verbs and arguments that don't go together
  - For example
  - VP -> V NP therefore

    Sneezed the book is a VP since "sneeze" is a verb and "the book" is a valid NP

So What?

- Now *overgeneration* is a problem for a generative approach.
  - The grammar is supposed to account for all and only the strings in a language

- verb subcategorization facts will provide a key element for semantic (语义) analysis (determining who did what to who in an event)

Subcategorization Frame

- A verb subcategorizes for a particular phrase
  - “find” subcategorizes (次范畴化) for NP
- Subcat constituents are called *complements* (补语)
- Set of possible constituents = *subcat frame* (次范畴化框架)
- In semantics, this refers to predicate structure

Examples of subcat frames

Frame	Verb	Example
$\emptyset$	eat, sleep	I ate
NP	prefer, find, leave	Find [$_{NP}$ the flight from Pittsburgh to Boston]
NP NP	show, give	Show [$_{NP}$ me] [$_{NP}$ airlines with flights from Pittsburgh]
PP_{from} PP_{to}	fly, travel	I would like to fly [$_{PP}$ from Boston] [$_{PP}$ to Philadelphia]
NP PP_{with}	help, load	Can you help [$_{NP}$ me] [$_{PP}$ with a flight]
VPto	prefer, want, need	I would prefer [$_{VPto}$ to go by United airlines]
VPbrst	can, would, might	I can [$_{VPbrst}$ go from Boston]
S	mean	Does this mean [$_{S}$ AA has a hub in Boston]

Figure 11.6 Subcategorization frames for a set of example verbs.

Solution for subcat

- Subtypes for verb category
  - V-with-NP-complement, V-with-S-complement, etc
- Make each VP rule to require verb subcat
  - VP -> V-with-NP-comp NP
  - VP -> V-with-S-comp S
- Number of rules increases significantly
- Loss of generality

Movement

- Core (canonical) example
  - My travel agent booked the flight

Movement

- Core example
  - [[My travel agent]$_{NP}$ [booked [the flight]$_{NP}$]$_{VP}$]$_{S}$

i.e. “book” is a straightforward transitive verb (及物动词). It expects a single NP arg within the VP as an argument, and a single NP arg as the subject.

Movement

- What about?
  - Which flight do you want me to have the travel agent book?
  - The direct object argument to “book” isn't appearing in the right place. It is in fact a long way from where its supposed to appear.
  - And note that its separated from its verb by 2 other verbs.

The Point

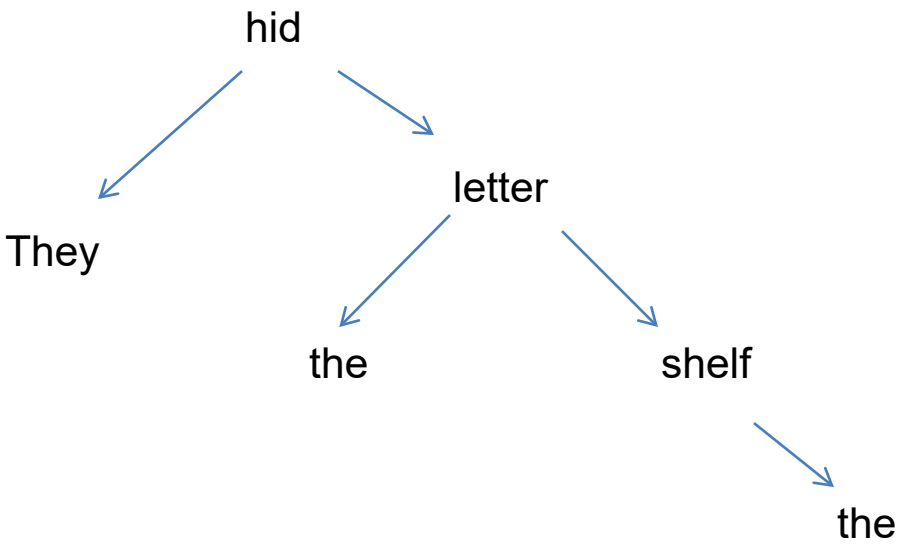
- CFGs appear to be just about what we need to account for a lot of basic syntactic structure in English.
- But there are problems
  - That can be dealt with adequately, although not elegantly, by staying within the CFG framework.
  - i.e. lexicalized CFG
- There are simpler, more elegant, solutions that take us out of the CFG framework (beyond its formal power)

Dependency (依存) Grammars

- More appropriate for languages other than English
  - Popular in Europe, India
- Key notion is thematic roles (论旨角色) between words
  - Often binary in nature

Simple Example

They hid the letter on the shelf



Roles are the links. Note they are directional and binary.

Differences with CFG

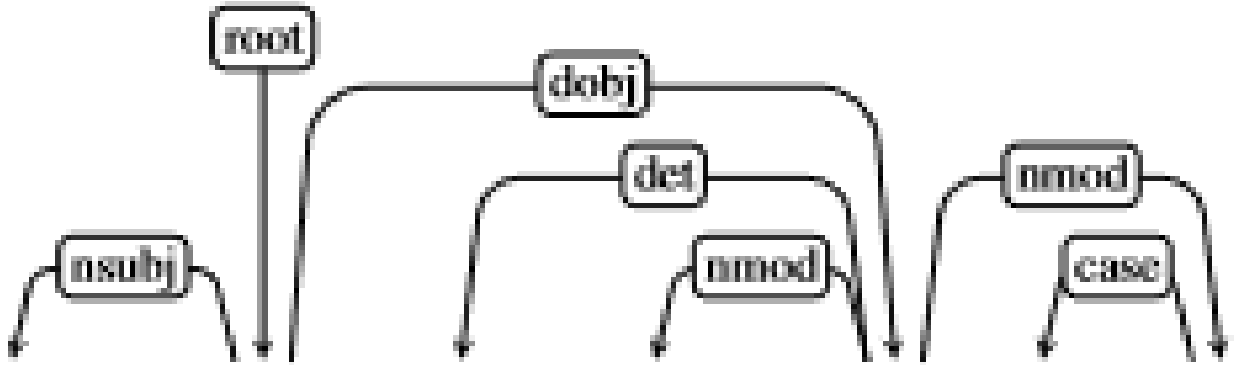
- No non-terminals
- Each link between 2 lexical nodes
- Dependency parse
  - Each link can be labeled
  - Useful for languages with flexible ordering
- Can convert CFG parse to dependency parse

Dependency Relations

Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Figure 14.2 Selected dependency relations from the Universal Dependency set. (de Marneffe et al., 2014)

Another example (with labels)



(14.2)

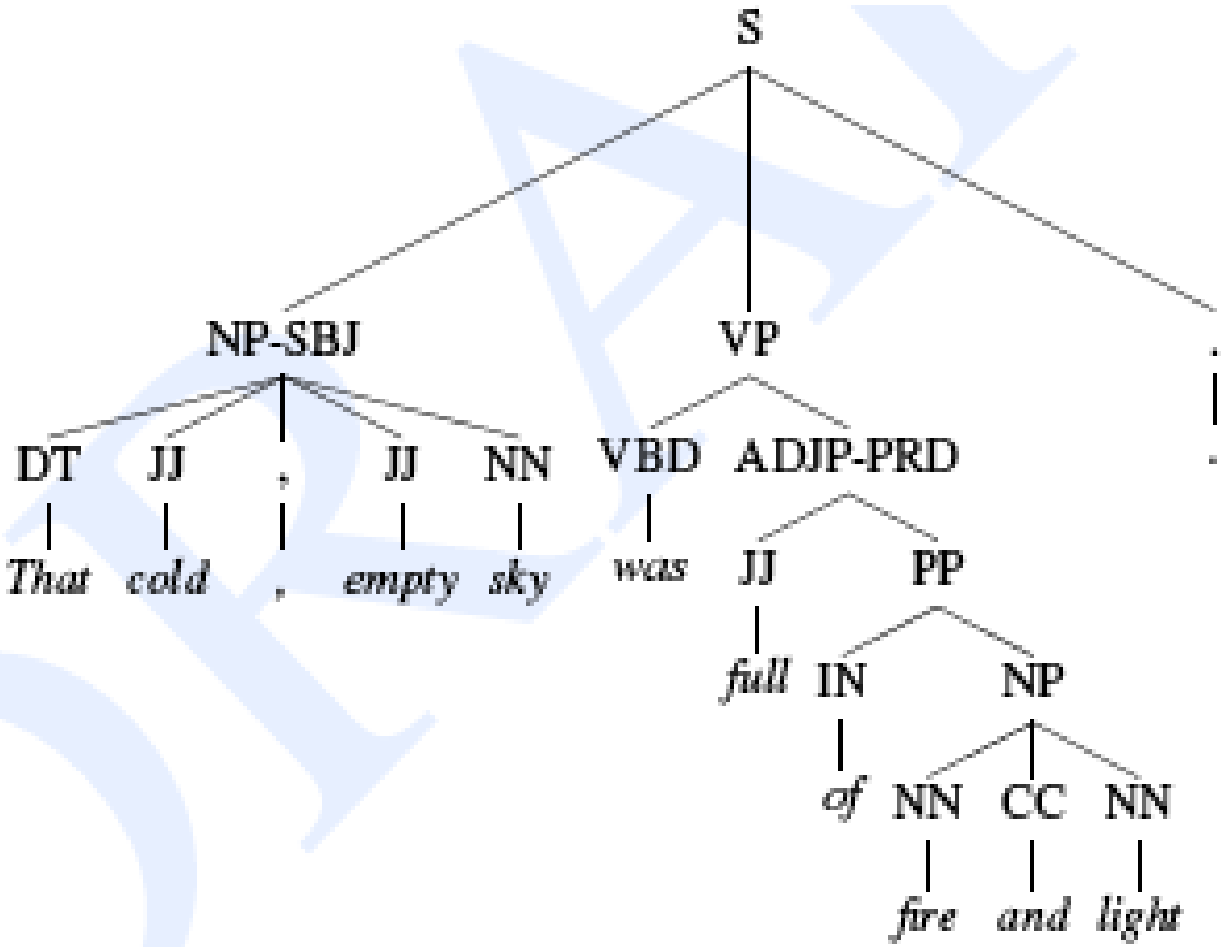
Dependency Parsing

- Helpful in machine translation systems
- Closer to semantics than CFG
- Abstracts out the word-order variation

Grammars: Origins

- Before you can parse you need a grammar.
- So where do grammars come from?
  - Grammar Engineering
    - Hand-crafted decades-long efforts by humans to write grammars
  - TreeBanks (树库)
    - Semi-automatically generated sets of parse trees for the sentences in some corpus
    - The grammar is the set of rules (local subtrees) that occur in the annotated corpus
    - They tend to the flat and redundant
    - Penn TreeBank (III) has about 17500 grammar rules under this definition.

TreeBanks


S
NP-SBJ
VP
.
DT
JJ
,
JJ
NN
VBD
ADJP-PRD
That
cold
,
empty
sky
was
JJ
PP
full
IN
NP
of
NN
CC
NN
fire
and
light

TreeBanks

```
((S
    (NP-SBJ (DT That)
      (JJ cold) (, ,)
      (JJ empty) (NN sky) )
    (VP (VBD was)
      (ADJP-PRD (JJ full)
        (PP (IN of)
          (NP (NN fire)
            (CC and)
            (NN light) ))))
    (. .) ))
```

Sample Rules

NP → DT JJ NNS
NP → DT JJ NN NN
NP → DT JJ JJ NN
NP → DT JJ CD NNS
NP → RB DT JJ NN NN
NP → RB DT JJ JJ NNS
NP → DT JJ JJ NNP NNS
NP → DT NNP NNP NNP NNP JJ NN
NP → DT JJ NNP CC JJ JJ NN NNS
NP → RB DT JJS NN NN SBAR
NP → DT VBG JJ NNP NNP CC NNP
NP → DT JJ NNS , NNS CC NN NNS NN
NP → DT JJ JJ VBG NN NNP NNP FW NNP
NP → NP JJ , JJ `` SBAR '' NNS

Example

NP → NP JJ , JJ `` SBAR '' NNS

(11.10) [NP Shearson's] [JJ easy-to-film], [JJ black-and-white] "[SBAR Where We Stand]" [NNS commercials]

TreeBanks

- Treebanks provide a grammar because we can read the rules of the grammar out of the treebank.
- But how did the trees get in there in the first place? There must have been a grammar theory in there someplace…
- Typically, not all of the sentences are hand-annotated by humans.
- They are automatically parsed and then hand-corrected.

- Syntax (句法) and Grammer (语法)
  - CFG, Dependency Grammer, TreeBanks
- Parsing (剖析)
  - CKY Parsing Algorithm
- Partial Parsing
  - Rule-Based / Machine Learning-Based

Parsing (剖析)

- Parsing with CFGs refers to the task of assigning correct trees to input strings
- Correct here means a tree that covers all and only the elements of the input and has an S at the top
- It doesn't actually mean that the system can select the correct tree from among all the possible trees

Parsing (剖析)

- As with everything of interest, parsing involves a search which involves the making of choices

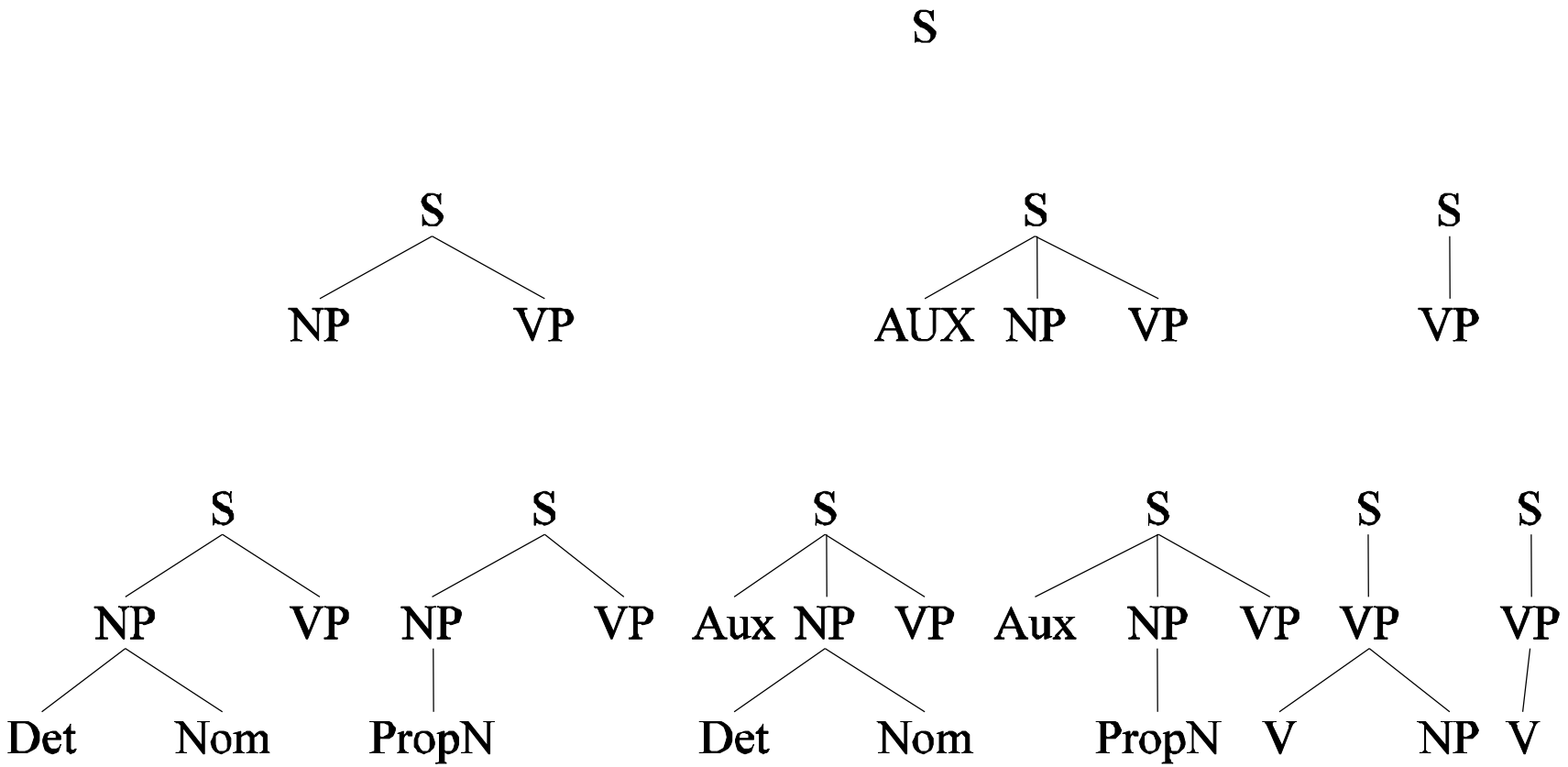
For Now

- Assume…
  - You have all the words already in some buffer
  - The input isn't POS tagged
  - We won't worry about morphological analysis
  - All the words are known

Top-Down Parsing

- Since we’ re trying to find trees rooted with an S (Sentences) start with the rules that give us an S.
- Then work your way down from there to the words.

Top-Down Space

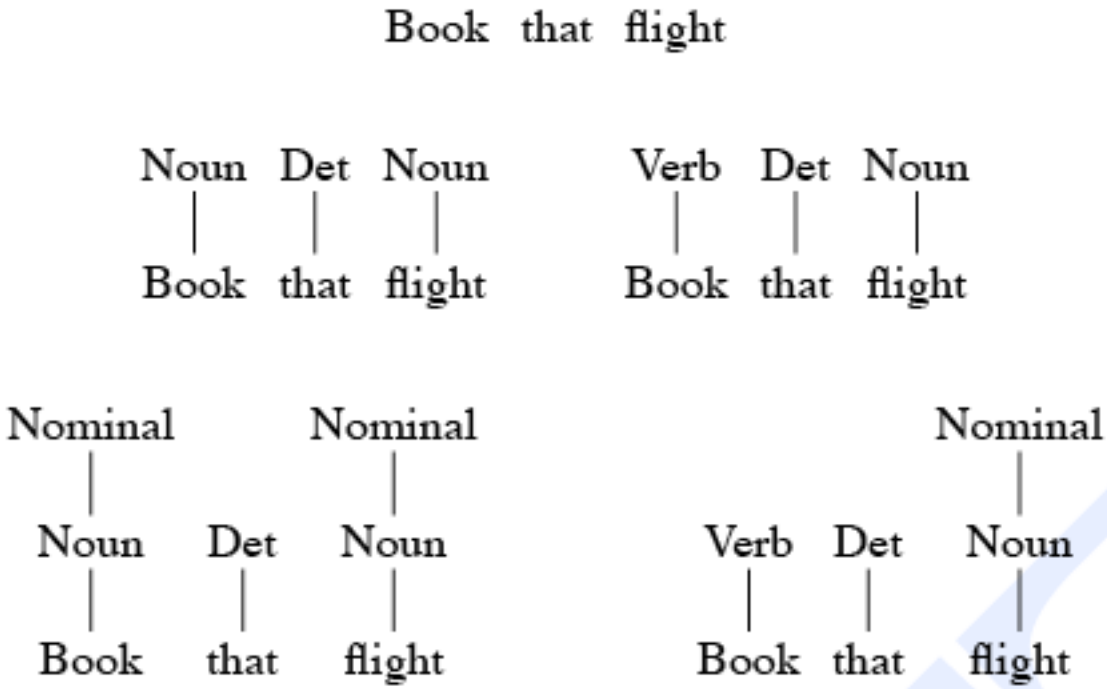


Bottom-Up Parsing

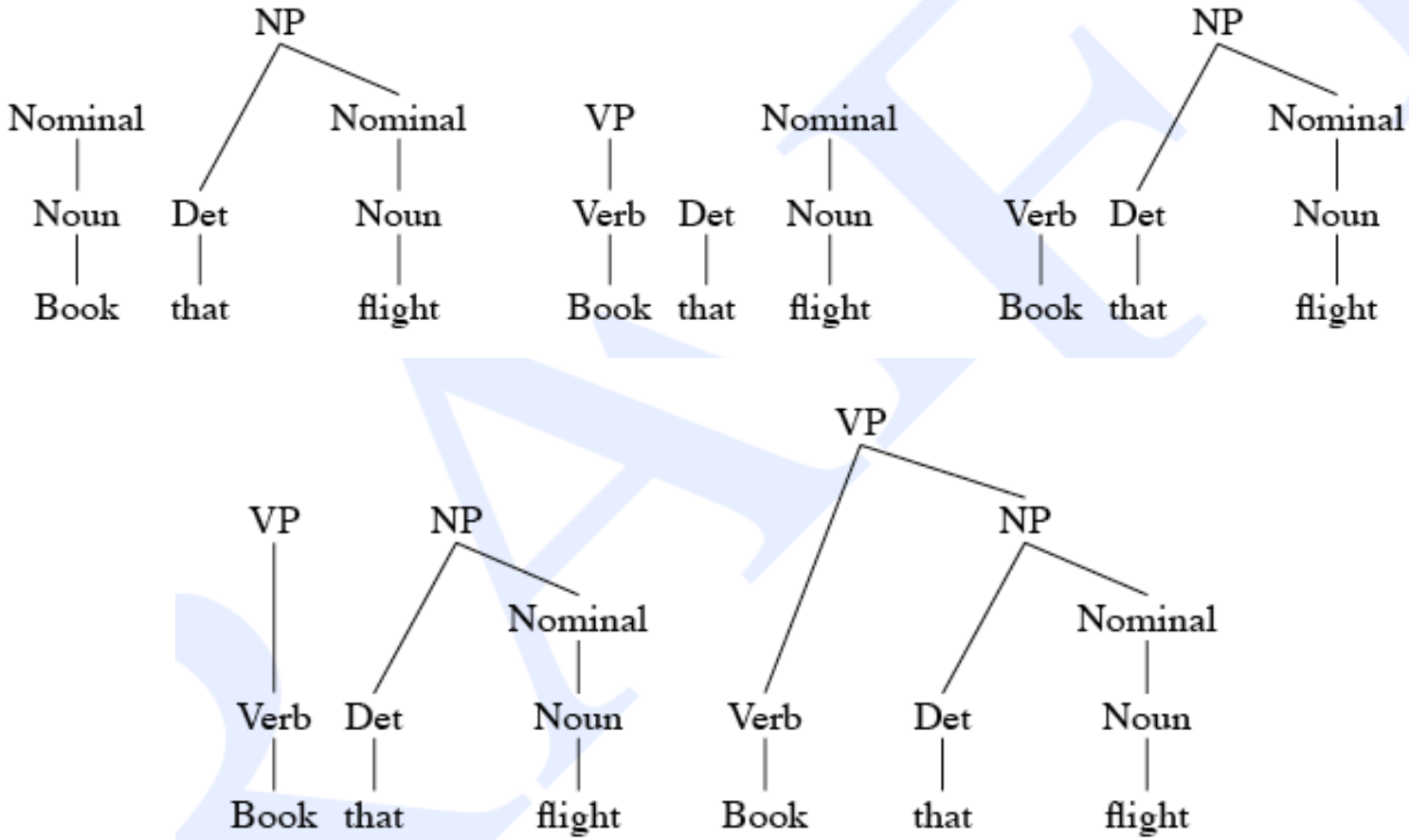
- Of course, we also want trees that cover the input words. So start with trees that link up with the words in the right way.

- Then work your way up from there.

Bottom-Up Space



Bottom-Up Space



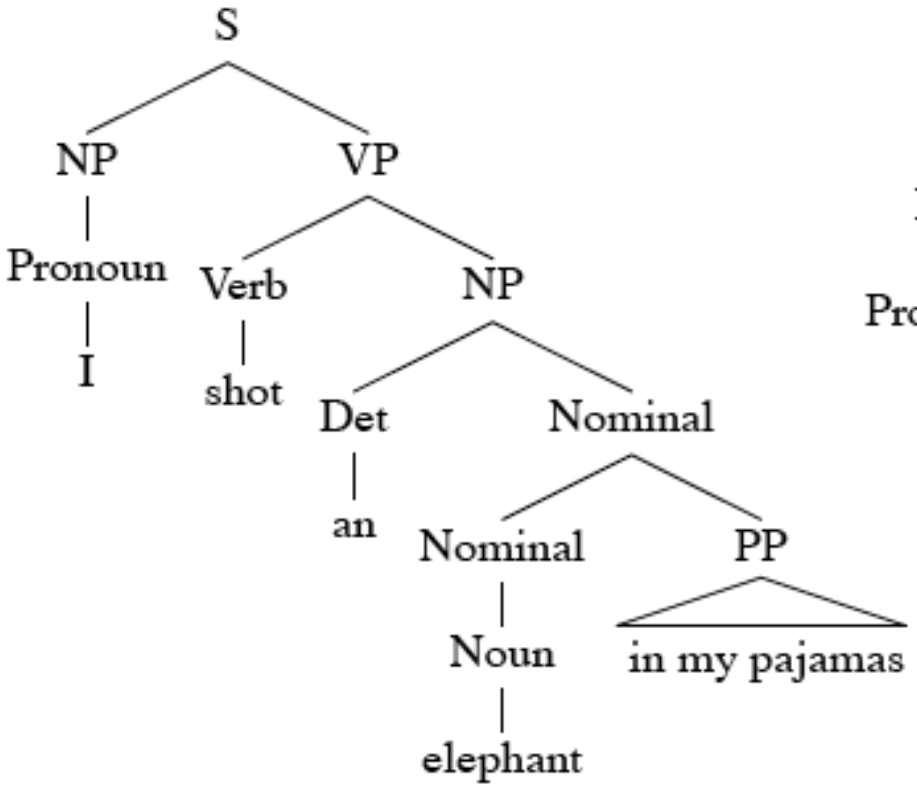
Control

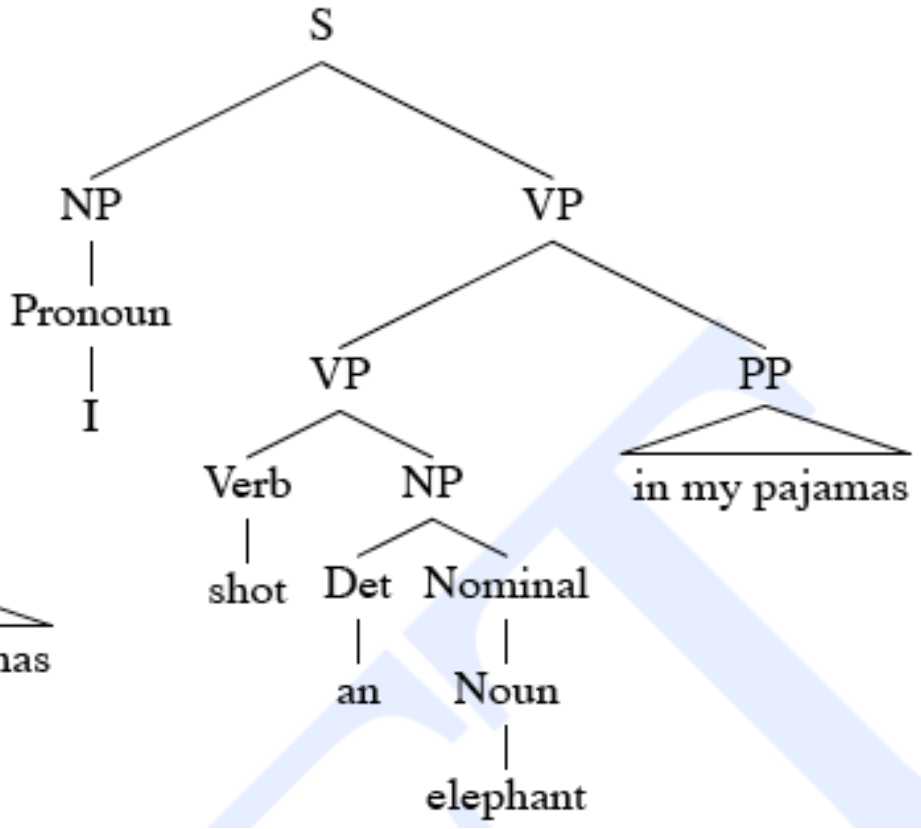
- Of course, in both cases we left out how to keep track of the search space and how to make choices
  - Which node to try to expand next
  - Which grammar rule to use to expand a node

Top-Down and Bottom-Up

- Top-down
  - Only searches for trees that can be answers
  - But also suggests trees that are not consistent with any of the words
- Bottom-up
  - Only forms trees consistent with the words
  - But suggest trees that make no sense globally

Ambiguity (歧义)





Dynamic Programming Parsing

- DP methods fill tables with partial results and
  - Do not do too much avoidable repeated work
  - Solve exponential problems in polynomial time (sort of)
  - Efficiently store ambiguous structures with shared sub-parts.
- Dynamic programming solutions that run in O(n^3) time.
  - CKY is bottom-up (going from words to S)
  - Earley is top-down (going from S to words)

Sample Grammar

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

CKY Parsing

- Limit our grammar Chomsky normal form (CNF) (Chomsky范式)
  - A->BC or A->w
  - *either 2 non-terminals or to a single terminal*
- Consider the rule A -> BC
  - If there is an A in the input then there must be a B followed by a C in the output.
  - If the A spans from i to j in the input then there must be some k s.t. i<k<j
    - i.e. The B splits from the C someplace.

Note on index convention

- 0 to n
- Think of them as the inter-word space
  - 0 = before 1st word
  - 1 = before 2nd word (or AFTER 1st word)
  - n= after n-th word
- 0 The 1 book 2 is 3 on 4 the 5 table 6
- A rule that spans 0 to 6 will span the sentence

CKY

- So let's build a table so that an A spanning from i to j in the input is placed in cell [i,j] in the table.
- So a non-terminal spanning the entire string will sit in cell [0, n]
- If we build the table bottom up we'll know that the parts of the A must go from i to k and from k to j

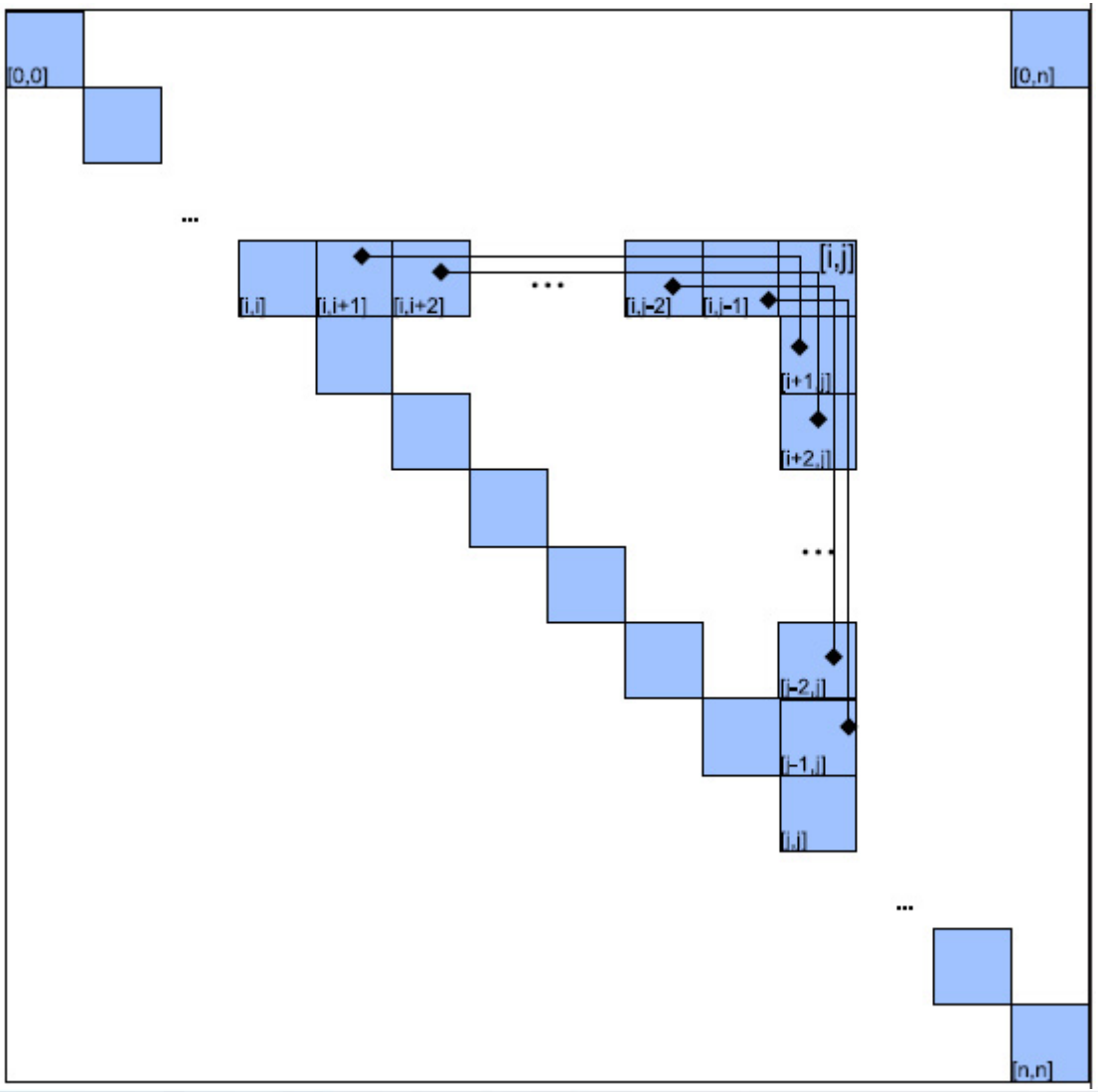
CKY

- Meaning that for a rule like A -> B C we should look for a B in [i,k] and a C in [k,j].
- In other words, if we think there might be an A spanning i,j in the input… AND
- A -> B C is a rule in the grammar THEN
- There must be a B in [i,k] and a C in [k,j] for some i<k<j

CKY

- So to fill the table loop over the cell[i,j] values in some systematic way
  - What constraint should we put on that?
    - For each cell loop over the appropriate k values to search for things to add.
- Note the binary constraint: each rule has exactly 2 non-terminals in RHS

CKY Table



CKY Algorithm

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

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do** Iterate over columns

  $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$

  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do** Iterate over rows (from bottom up)

    for $k \leftarrow i+1$ **to** $j-1$ **do** Iterate over all possible splits

      $table[i,j] \leftarrow table[i,j] \cup$

        $\{A \mid A \rightarrow BC \in grammar,$

          $B \in table[i,k],$

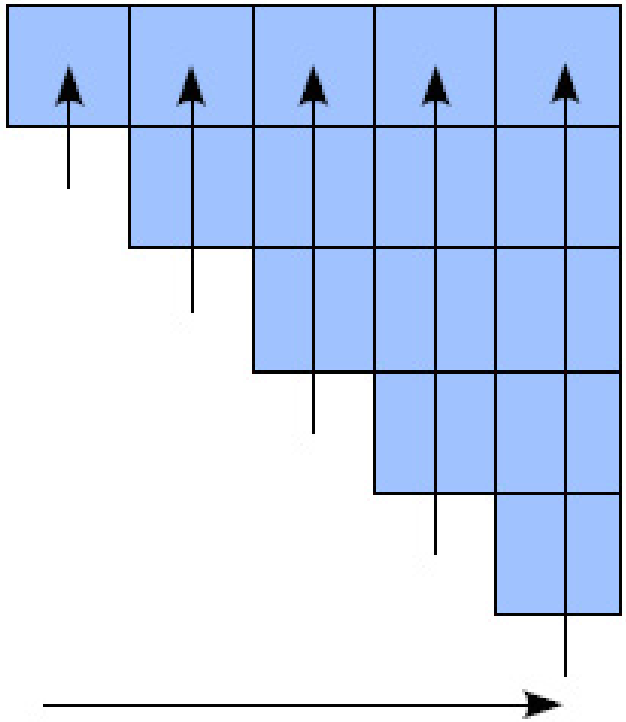
          $C \in table[k,j]\}$

Note

- We arranged the loops to fill the table a column at a time, from left to right, bottom to top.
  - This assures us that whenever we're filling a cell, the parts needed to fill it are already in the table (to the left and below)

Example

Book	*the*	*flight*	*through*	*Houston*
S,VP,Verb Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	S, VP [0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper-Noun [4,5]

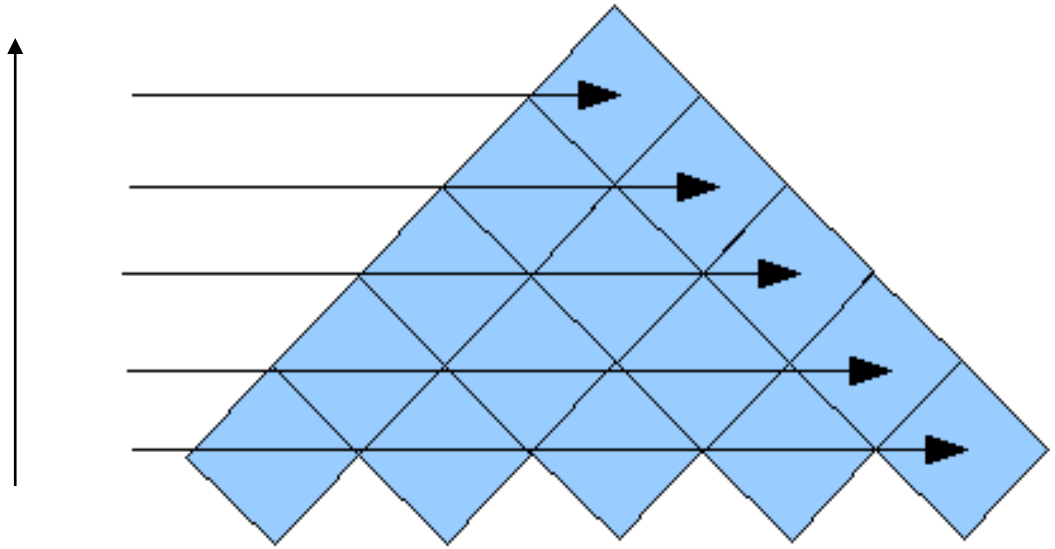


CKY Parsing

- Is that really a parser?
- Recognize != parse
- Need to keep track of backpointers

Other Ways to Do It?

- Are there any other sensible ways to fill the table that still guarantee that the cells we need are already filled?



Sample Grammar

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

Problem

- What if your grammar isn't binary?
  - As in the case of the TreeBank grammar?
- Convert it to binary… any arbitrary CFG can be rewritten into CNF automatically.
- What does this mean?
  - The resulting grammar accepts (and rejects) the same set of strings as the original grammar.
  - But the resulting derivations (trees) are different.

Binarization Intuition

- Introduce new intermediate non-terminals into the grammar that distribute rules with length > 2 over several rules. So…

  S -> A B C

  turns into

  S -> X C

  X - A B

  where X is a symbol that doesn't occur anywhere else in the grammar.

CNF Conversion

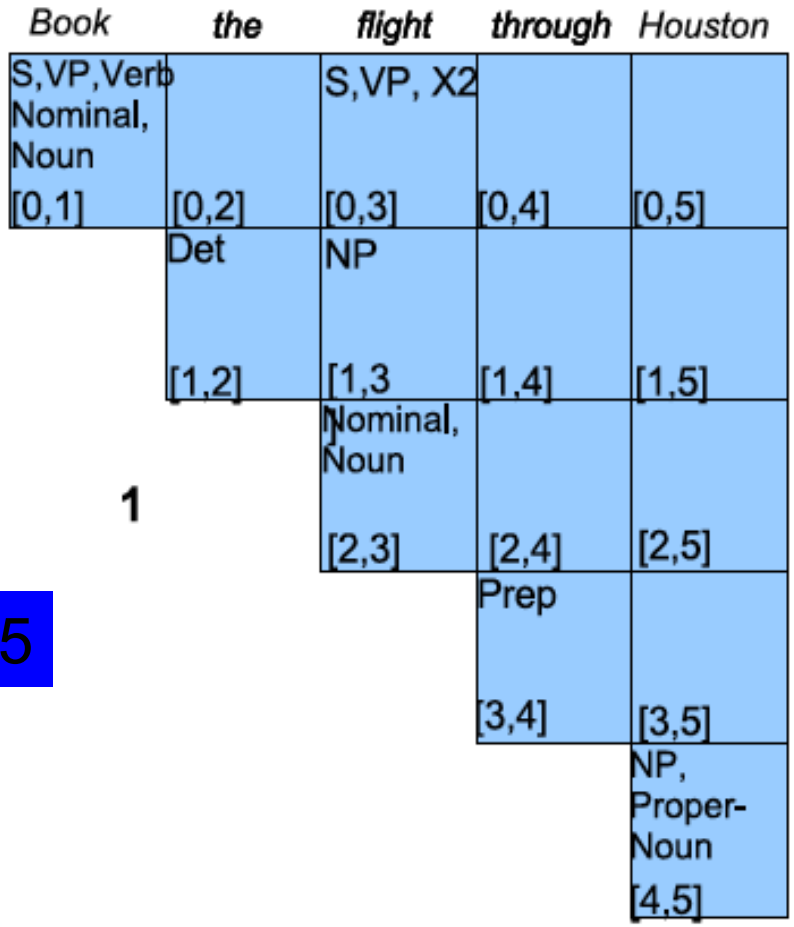
$S \rightarrow NP\ VP$	$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$	$S \rightarrow X1\ VP$
	$X1 \rightarrow Aux\ NP$
$S \rightarrow VP$	$S \rightarrow book \mid include \mid prefer$
	$S \rightarrow Verb\ NP$
	$S \rightarrow X2\ PP$
	$S \rightarrow Verb\ PP$
	$S \rightarrow VP\ PP$
$NP \rightarrow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

CKY Algorithm

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

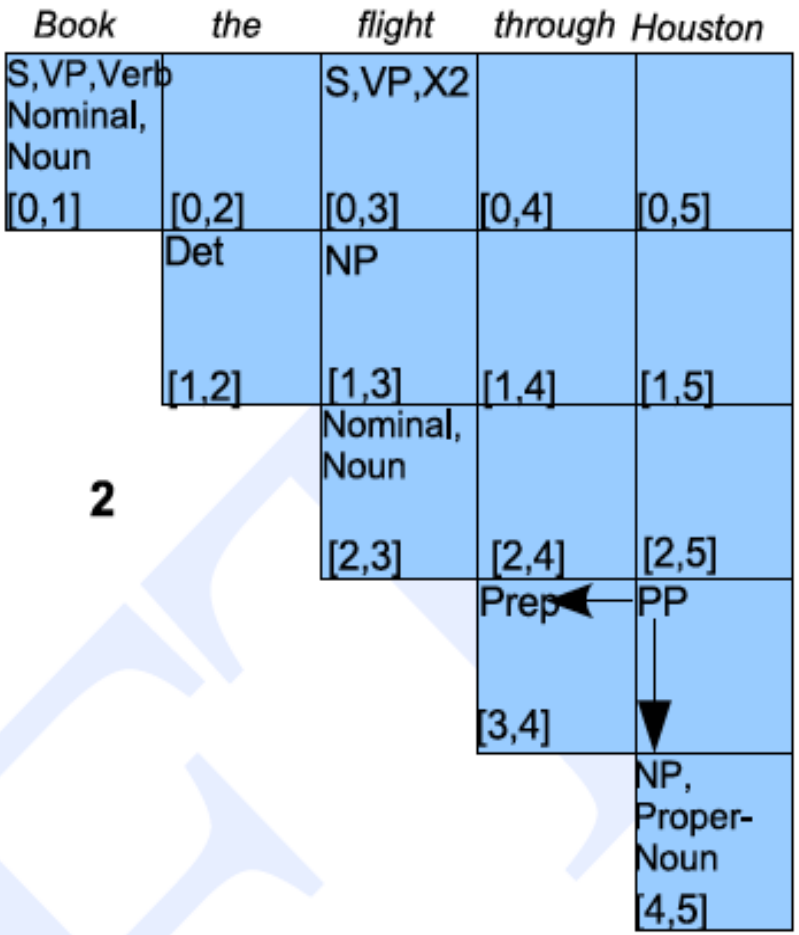
for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      $table[i,j] \leftarrow table[i,j] \cup$
        $\{A \mid A \rightarrow BC \in grammar,$
          $B \in table[i,k],$
          $C \in table[k,j]\}$

Example

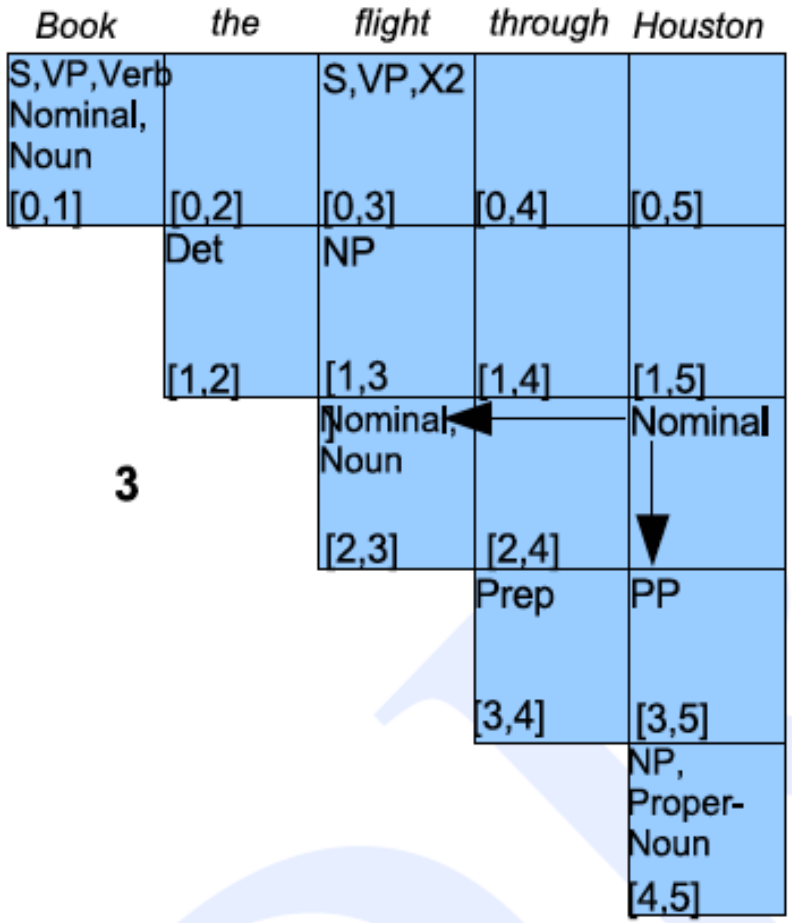


Filling column 5

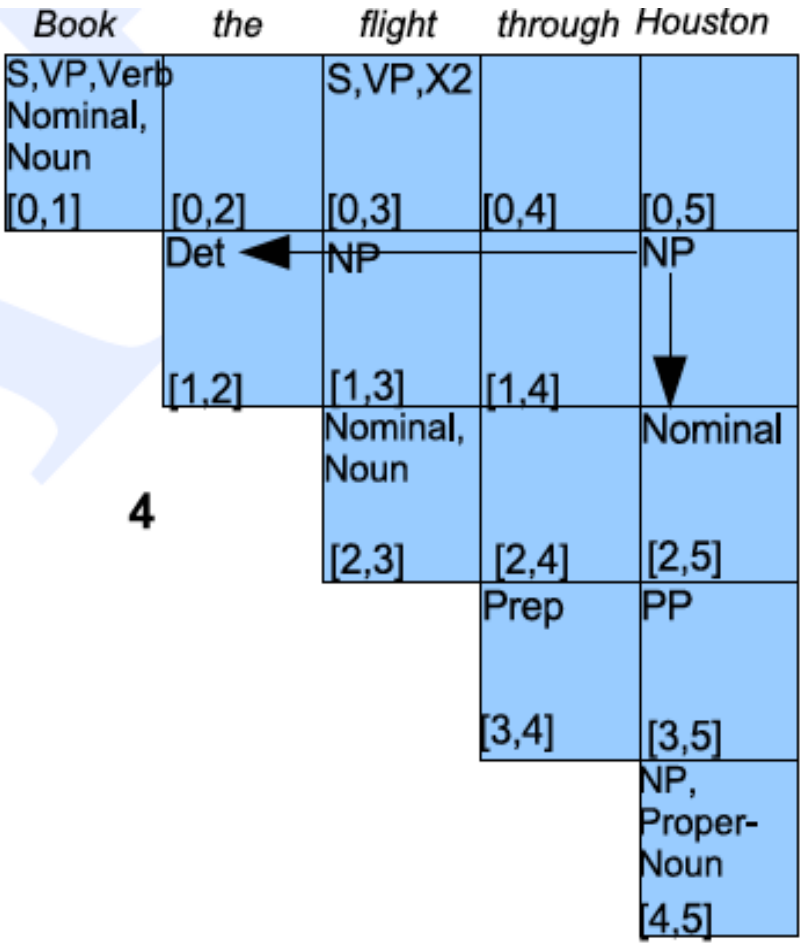
Example


Book
the
flight
through
Houston
S,VP,Verb
Nominal,
Noun
[0,1]
[0,2]
S,VP,X2
[0,3]
[0,4]
[0,5]
Det
[1,2]
NP
[1,3]
[1,4]
[1,5]
Nominal,
Noun
[2,3]
[2,4]
[2,5]
Prep
[3,4]
PP
NP,
Proper-
Noun
[4,5]
2

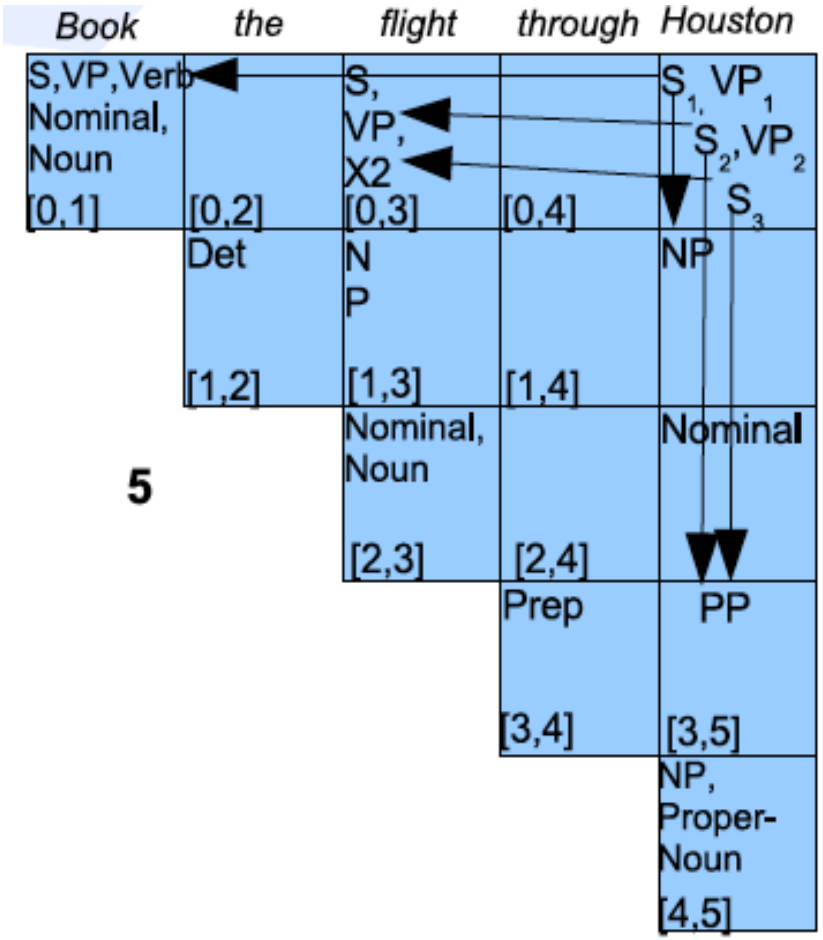
Example


Book
the
flight
through
Houston
S,VP,Verb
Nominal,
Noun
[0,1]
[0,2]
S,VP,X2
[0,3]
[0,4]
[0,5]
Det
[1,2]
NP
[1,3
[1,4]
[1,5]
Nominal,
Noun
[2,3]
[2,4]
Nominal
Prep
[3,4]
PP
[3,5]
NP,
Proper-
Noun
[4,5]
3

Example


Book
the
flight
through
Houston
S,VP,Verb
Nominal,
Noun
[0,1]
[0,2]
S,VP,X2
[0,3]
[0,4]
[0,5]
Det
NP
NP
[1,2]
[1,3]
[1,4]
Nominal,
Noun
[2,3]
Nominal
[2,4]
[2,5]
Prep
[3,4]
PP
[3,5]
NP,
Proper-
Noun
[4,5]

Example


Book
the
flight
through
Houston
S,VP,Verb
Nominal,
Noun
[0,1]
[0,2]
S,
VP,
X2
[0,3]
[0,4]
S1, VP1
S2,VP2
S3
Det
[1,2]
N
P
[1,3]
[1,4]
NP
Nominal,
Noun
[2,3]
[2,4]
Nominal
Prep
[3,4]
PP
[3,5]
NP,
Proper-
Noun
[4,5]

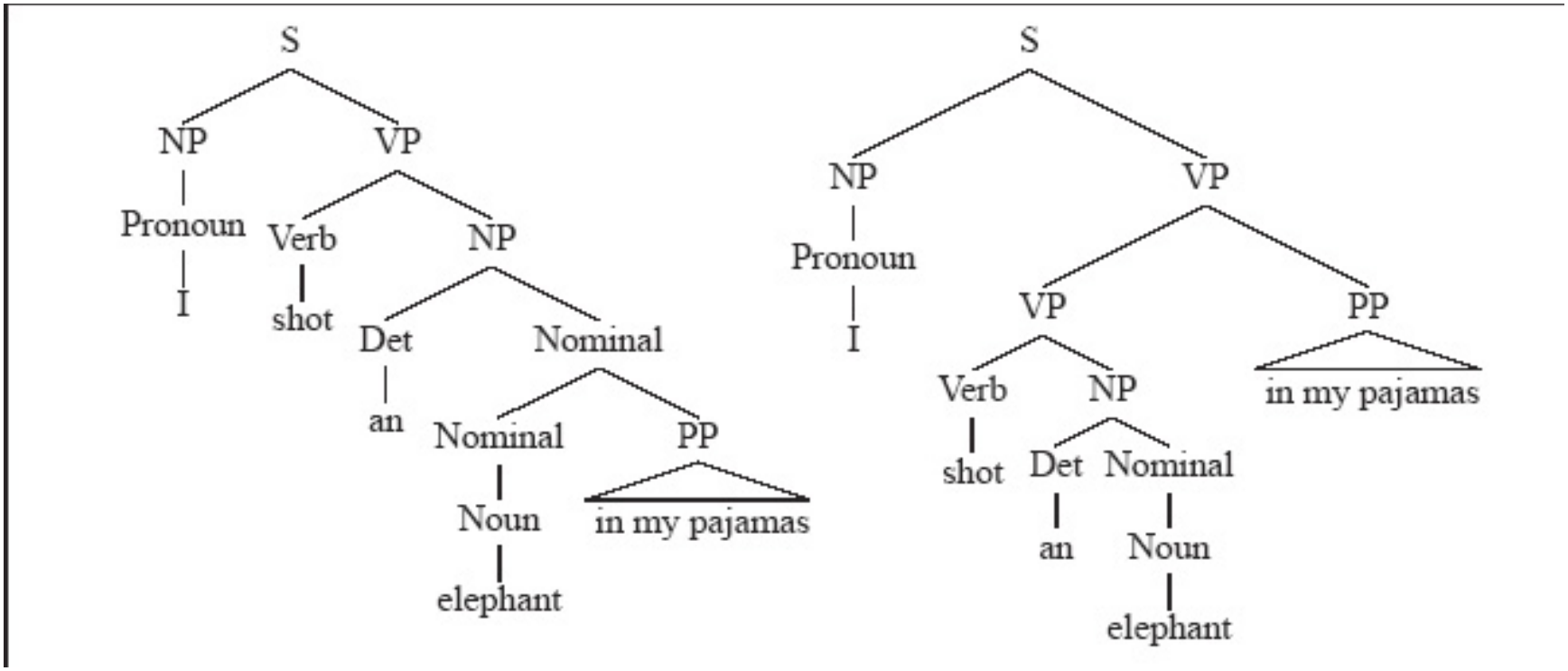
CKY Notes

- Since it's bottom up, CKY populates the table with a lot of phantom constituents.
  - Segments that by themselves are constituents but cannot really occur in the context in which they are being suggested.
  - To avoid this we can switch to a top-down control strategy (Earley Parsing) or
  - We can add some kind of filtering that blocks constituents where they can not happen in a final analysis.

Back to Ambiguity

- Did we solve it?

Ambiguity


S
NP
VP
Pronoun
Verb
NP
I
shot
Det
Nominal
an
Nominal
PP
Noun
in my pajamas
elephant
S
NP
VP
Pronoun
VP
PP
I
Verb
NP
in my pajamas
shot
Det
Nominal
an
Noun
elephant

Ambiguity

- No…
  - Both CKY and Earley will result in multiple S structures for the [0,n] table entry.
  - They both efficiently store the sub-parts that are shared between multiple parses.
  - And they obviously avoid re-deriving those sub-parts.
  - But neither can tell us which one is right.

- Syntax (句法) and Grammer (语法)
  - CFG, Dependency Grammer, TreeBanks
- Parsing (剖析)
  - CKY Parsing Algorithm
- Partial Parsing
  - Rule-Based / Machine Learning-Based

Full Syntactic Parsing

- Probably necessary for deep semantic analysis of texts
- Probably not practical for many applications (given typical resources)
  - O(n^3) for straight parsing
  - Too slow for applications that need to process texts in real time (search engines)
  - Or that need to deal with large volumes of new material over short periods of time

Partial Parsing

- For many applications you don't really need a full-blown syntactic parse. You just need a good idea of where the base syntactic units are.
  - Often referred to as chunks (组块).

- For example, if you're interested in locating all the people, places and organizations in a text it might be useful to know where all the NPs are.

Examples

[$_{NP}$ The morning flight] [$_{PP}$ from] [$_{NP}$ Denver] [$_{VP}$ has arrived.]

[$_{NP}$ a flight] [$_{PP}$ from] [$_{NP}$ Indianapolis][$_{PP}$ to][$_{NP}$ Houston][$_{PP}$ on][$_{NP}$ TWA]

[$_{NP}$ The morning flight] from [$_{NP}$ Denver] has arrived.

- The first two are examples of full partial parsing or chunking. All of the elements in the text are part of a chunk. And the chunks are non-overlapping.
- The last example illustrates base-NP chunking. Ignore anything that isn't in the kind of chunk you're looking for.

Rule-Based Partial Parsing

- Typical Architecture
  - Phase 1: Part of speech tags
  - Phase 2: Base syntactic phrases
  - Phase 3: Larger verb and noun groups
  - Phase 4: Sentential level rules

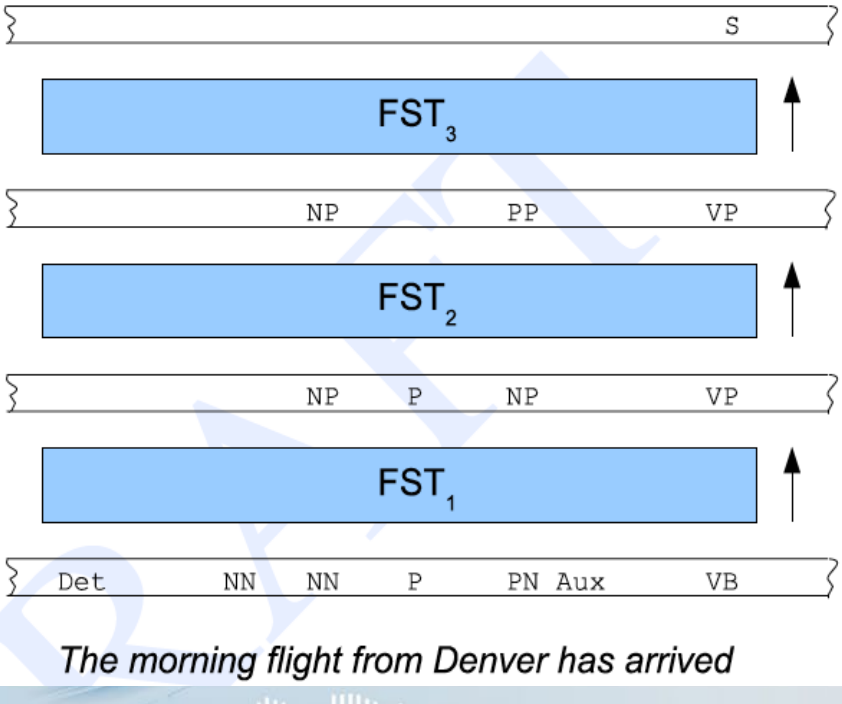
Rule-Based Partial Parsing

$$NP \rightarrow (Det)\ Noun^*\ Noun$$
$$NP \rightarrow Proper\text{-}Noun$$
$$VP \rightarrow Verb$$
$$VP \rightarrow Aux\ Verb$$

- No direct or indirect recursion allowed in these rules.

Rule-Based Partial Parsing

- This cascaded approach can be used to find the sequence of flat chunks you're interested in.
- Or it can be used to approximate the kind of hierarchical trees you get from full parsing with a CFG.



Phase 1: Part of speech tags
Phase 2: Base syntactic phrases
Phase 3: Larger verb and noun groups
Phase 4: Sentential level rules

Machine Learning-Based Chunking

- View this task as sequential classification
- IOB tagging
  - Beginning(B) Internal(I) Outside(O)
  - e.g.

[$_{NP}$ The morning flight] from [$_{NP}$ Denver] has arrived.

The morning flight from Denver has arrived

B_NP I_NP I_NP O B_NP O O

Machine Learning-Based Chunking

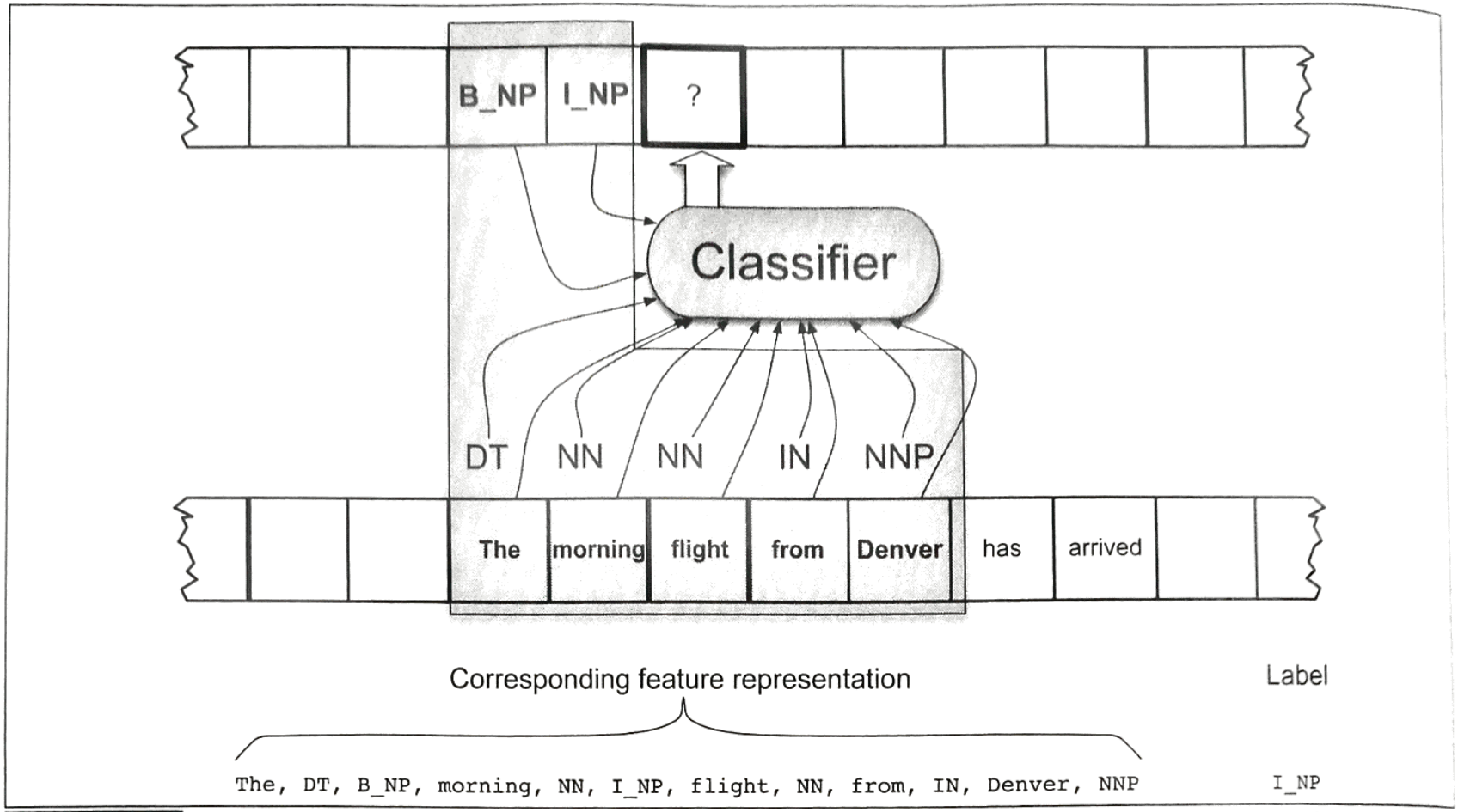


Figure 13.19 A sequential-classifier-based approach to chunking. The chunker slides a context window over the sentence, classifying words as it proceeds. At this point, the classifier is attempting to label *flight*. Features derived from the context typically include the words, part-of-speech tags as well as the previously assigned chunk tags.