Introduction to Computational Linguistics

Lecture 4: Grammars and Parsing

Pavlina Ivanova

University of Plovdiv, Bulgaria

Thanks to Daniel Jurafsky for many of these slides!

Outline for Grammar/Parsing

- Context-Free Grammars and Constituency
- Some common CFG phenomena for English
  - Sentence-level constructions
  - NP, PP, VP
  - Coordination
  - Subcategorization
- Top-down and Bottom-up Parsing
- Problems with these parsers
  - Ambiguity
  - Left-recursion
  - Repeated work
- Solution: Dynamic Programming parsing
  - CKY
  - Earley
- Quick sketch of probabilistic parsing

Review

- Parts of Speech
  - Basic syntactic/morphological categories that words belong to
- Part of Speech tagging
  - Assigning parts of speech to all the words in a sentence

Syntax

- Syntax: from Greek syntaxis, "setting out together, arrangement'
- Refers to the way words are arranged together, and the relationship between them.
- Distinction:
  - **Prescriptive grammar: how people ought to talk**
  - Descriptive grammar: how they do talk
- Goal of syntax is to model the knowledge of that people unconsciously have about the grammar of their native language

Syntax

- Applications
  - Grammar checkers
  - Database access
  - Question answering
  - Information retrieval
  - Information extraction
  - Machine translation
  - Summarization
  - Text generating

3 key ideas of syntax

- Constituency

  Groups of words may behave as a single unit.

  Ex.: the dog, a big dog, the dog that barked

- Grammatical relations

  Formalization of ideas from traditional grammar such as SUBJECTS and OBJECTS, and other related notions.

  Ex. She ate a mammoth breakfast.

- Subcategorization and dependency relations (local and long-distance dependencies)

  –Refer to certain kinds of relations between words and phrases. Ex.: the verb *want* can be followed by an infinitive, as in *I want to fly to Detroit*, or a noun phrase, as in *I want a flight to Detroit*. But the verb *find* cannot be followed by an infinitive *(*I found to fly to Dallas*).

Context-Free Grammars

- Capture constituency and ordering
  - Ordering:
    - What are the rules that govern the ordering of words and bigger units in the language?
  - Constituency:

    How words group into units and how the various kinds of units behave
  - Constituent:

    Group of words that may behave as a single unit or phrase

Constituency

- Noun phrases (NPs)
  - Three parties from Brooklyn
  - A high-class spot such as Mindy's
  - The Broadway coppers
  - They
  - Harry the Horse
  - The reason he comes into the Hot Box
- How do we know these form a constituent?
  - They can all appear before a verb:
    - Three parties from Brooklyn arrive…
    - A high-class spot such as Mindy's attracts…
    - The Broadway coppers love…
    - They sit…

Constituency (II)

- They can all appear before a verb:
  - Three parties from Brooklyn arrive…
  - A high-class spot such as Mindy's attracts…
  - The Broadway coppers love…
  - They sit
- But individual words can't always appear before verbs:
  - *from arrive…
  - *as attracts…
  - *the is
  - *spot is…
- Must be able to state generalizations like:
  - Noun phrases occur before verbs

Constituency (IV)

- Practical test for constituent:
  - May appears independently
  - May be replaced with other
  - May be moved (in the beginning, at the end)

Context-Free Grammars (CFG)

- Also called Phrase-Structure Grammar
- Equivalent to Backus-Naur Form (BNF)
- Consists of:
  - Set of **rules** (productions) – express the ways that symbols of the language can be grouped and ordered together
  - **Lexicon** of words (symbols)

    Symbols are divided into 2 classes:
    - Terminal symbols – correspond to words in the language
    - Non-terminal symbols – express generalizations

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 it says that this is the only place that NPs and VPs occur

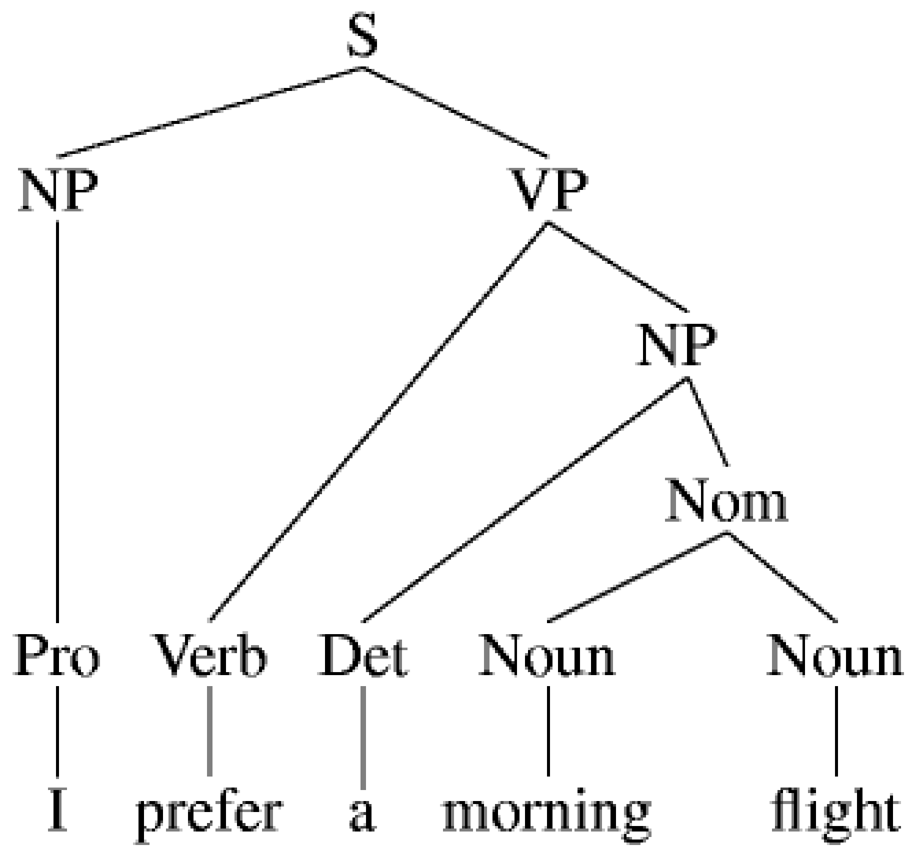
Generativity

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

Derivations

- A derivation 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 (Parse Tree)



Parsing

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

Context?

- The notion of context in CFGs has nothing to do with the ordinary meaning of the word context in language.
- All it really 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

Key Constituents (English)

- Sentences
- Noun phrases
- Verb phrases
- Prepositional phrases

Sentence-Types

- Declaratives: A plane left
  S → NP VP
- Imperatives: 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

NPs

- NP → Pronoun
  - I came, you saw it, they conquered
- NP → Proper-Noun
  - Los Angeles is west of Texas
  - John Hennesey is the president of Stanford
- NP → Det Nominal
- Nominal → Noun
  - The president
- Nominal → Nominal Noun
  - A morning flight to Denver

VPs

- $VP \rightarrow Verb\ NP$
  - prefer a morning flight
- $VP \rightarrow Verb\ NP\ PP$
  - leave Boston in the morning
- $VP \rightarrow Verb\ PP$
  - leaving on Thursday

PPs

- PP → Preposition NP
  - From LA
  - To Boston
  - On Tuesday
  - With lunch

Recursion

- 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).

  NP → NP PP   [[The 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]] [to Miami]]

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

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

  Etc.

Implications of recursion and context-freeness

- If you have a rule like
  - VP → V NP
  - It only cares that the thing after the verb is an NP. It doesn't have to know about the internal affairs of that NP

The Point

- VP → V NP
- I hate

  flights from Denver

  Flights from Denver to Miami

  Flights from Denver to Miami in February

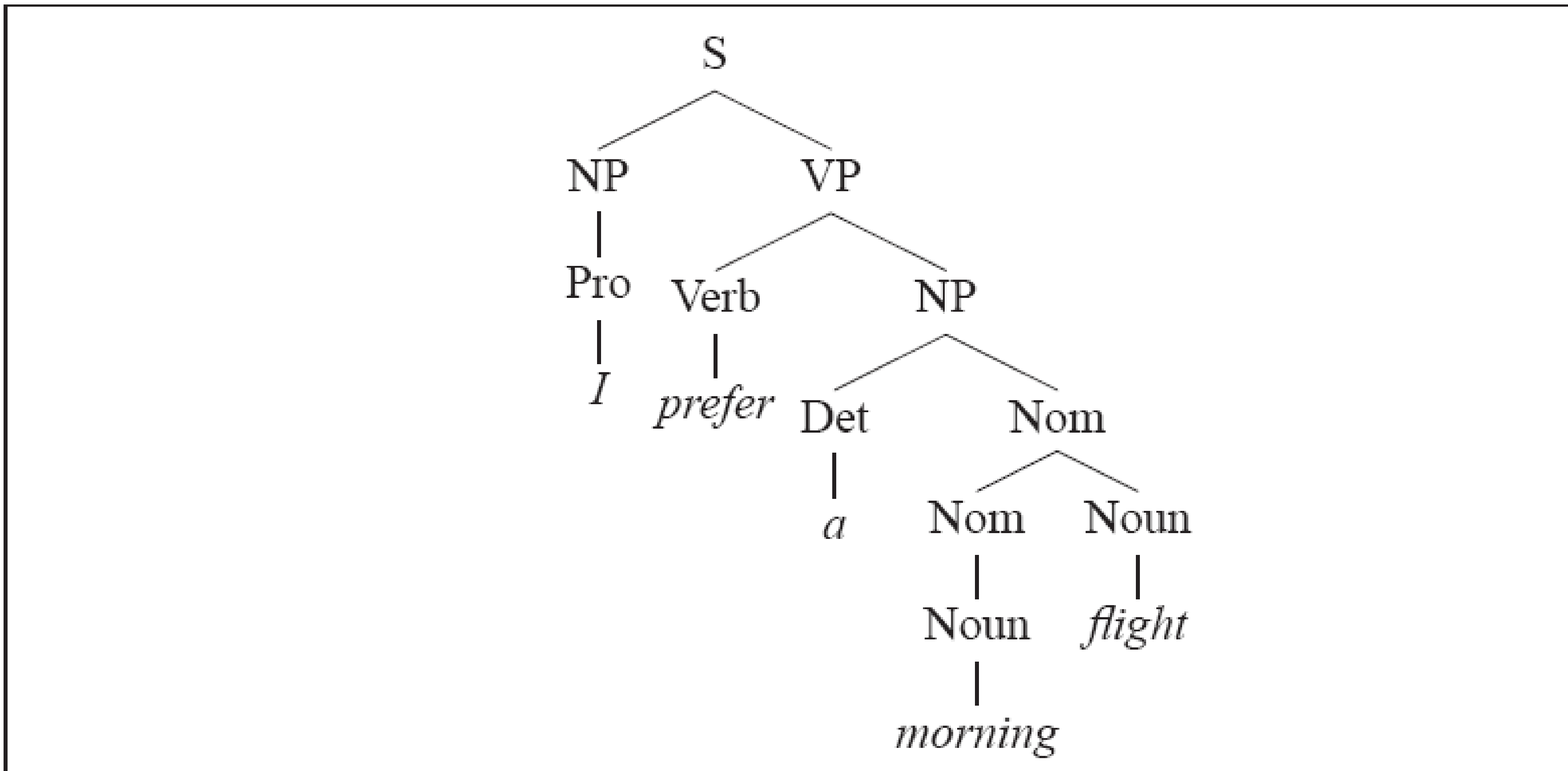
  Flights from Denver to Miami in February on a Friday

  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

Bracketed Notation

$[_S\ [_{NP}\ [_{PRO}\ \text{I}]]\ [_{VP}\ [_V\ \text{prefer}\ [_{NP}\ [_{NP}\ [_{Det}\ \text{a}][_{Nom}\ [_{Nom}\ [_N\ \text{morning}]]\ [_N\ \text{flight}]]]]$



Coordination Constructions

- S → S and S
  - John went to NY and Mary followed him
- NP → NP and NP
- VP → VP and VP
- …
- In fact the right rule for English is

  X → X and X

Problems

- Agreement
- Subcategorization
- Movement

Agreement

- This dog
- Those dogs

- *This dogs
- *Those dog

- This dog eats
- Those dogs eat

- *This dog eat
- *Those dogs eats

Possible CFG Solution

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

One way is to expand our grammar with multiple sets of rules.

- 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

- It doubles the size of grammar

These problems are compounded in languages like Bulgarian, German or French, which not only have number-agreement as in English, but also have gender agreement.

Subcategorization

- Verbs can be subcategorized by the types of complements/arguments they expect.
- Sneeze: John sneezed
- Find: Please find $[\text{a flight to NY}]_{NP}$
- Give: Give $[\text{me}]_{NP}[\text{a cheaper fare}]_{NP}$
- Help: Can you help $[\text{me}]_{NP}[\text{with a flight}]_{PP}$
- Prefer: I prefer $[\text{to leave earlier}]_{TO\text{-}VP}$
- Said: You said $[\text{United has a flight}]_{S}$
- …

Subcategorization

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

- Subcat expresses the constraints that a predicate (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

Subcategorization

The possible sets of complements of a verb are called its subcategorization frame

- 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]$_{S}$
- …

Another way of talking about the relation between the verb and these other constituents is to think of the verb as a logical predicate and the constituents as logical arguments of the predicate.

Forward Pointer

- It turns out that verb subcategorization facts will provide a key element for semantic analysis (determining who did what to who in an event).

Possible CFG Solution

- VP -> V
- VP -> V NP
- VP -> V NP PP
- …

Subtypes of verbs:
  - intransitive
  - transitive

Each rule could be modified to require the appropriate verb subtypes

- VP -> IntransV
- VP -> TransV NP
- VP -> TransVwPP NP PP
- …

Problem: The vast explosion in the number of rules.

Movement

- Core 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. 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.

CFGs: a summary

- 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.
- There are simpler, more elegant, solutions that take us out of the CFG framework (beyond its formal power)
- Syntactic theories: HPSG, LFG, Minimalism, etc

Other Syntactic stuff

- Grammatical Relations
  - Subject
    - I booked a flight to New York
    - The flight was booked by my agent.
  - Object
    - I booked a flight to New York
  - Complement
    - I said that I wanted to leave

Dependency Grammars

- The syntactic structure of a sentence is described purely in terms of words and binary semantic or syntactic relations between these words.

Dependency Parsing

- Word to word links instead of constituency
- Based on the European rather than American traditions
- But dates back to the Greeks
- The original notions of Subject, Object and the progenitor of subcategorization (called ‘valence’) came out of Dependency theory.
- Dependency parsing is quite popular as a computational model since relationships between words are quite useful

Dependency Grammars

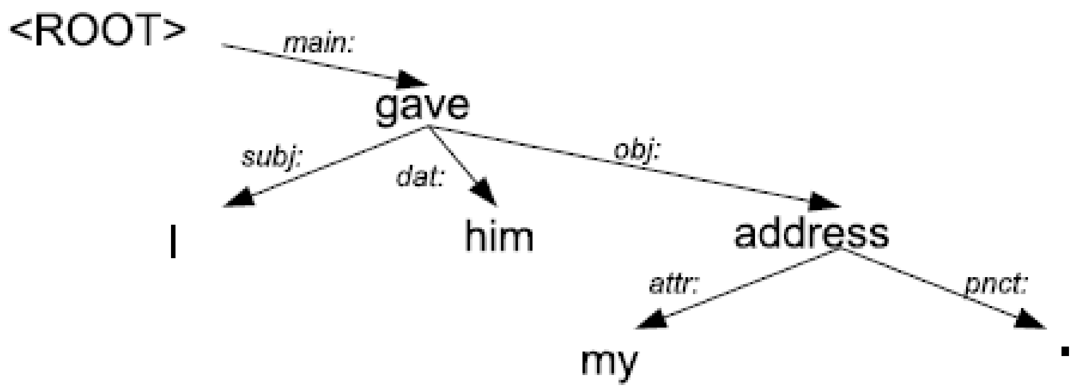


Figure 10.14 A sample dependency grammar parse, using the dependency formalism of Karlsson et al. (1995), after Järvinen and Tapanainen (1997).

Dependency Grammars

- One of the main advantages of pure dependency grammars is their ability to handle languages with relatively **free word order**.

Parsing

- Parsing: assigning correct trees to input strings
- Correct tree: a tree that covers all and only the elements of the input and has an S at the top
- For now: enumerate all possible trees
  - A further task: disambiguation: means choosing the correct tree from among all the possible trees.

Parsing

- The Link Grammar parser
  - http://www.link.cs.cmu.edu/link/
- Colorado parser
  - http://sds.colorado.edu/SEPA
- The Connexor dependency parser
  - http://www.connexor.com/demos/syntax_en.html

Treebanks

- Parsed corpora in the form of trees
- The Penn Treebank
  - The Brown corpus
  - The WSJ corpus
- Tgrep
  - http://www.ldc.upenn.edu/ldc/online/treebank/
  - http://www.ldc.upenn.edu/ldc/online/

TreeBanks

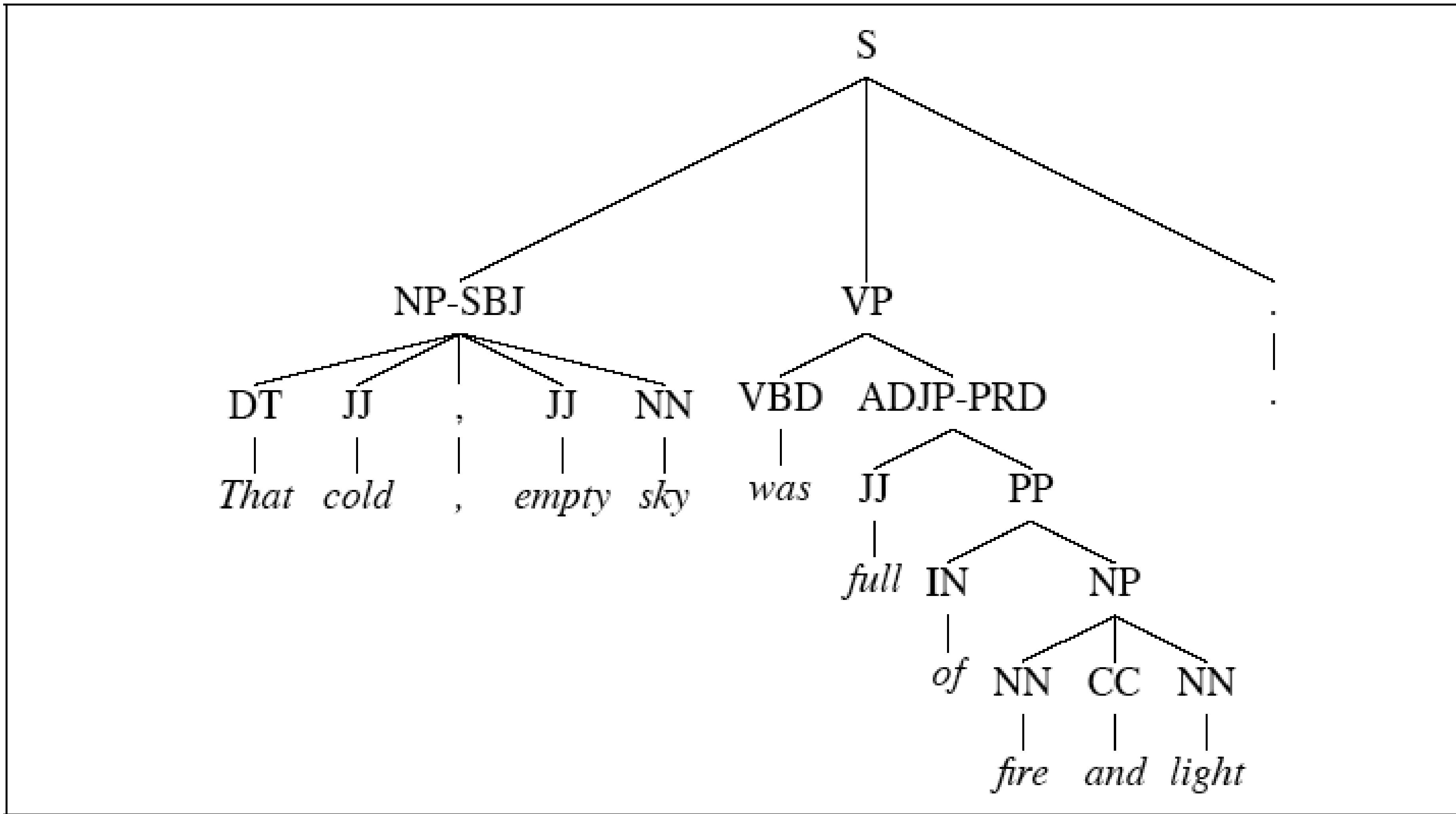
```
((S
   (NP-SBJ (DT That)
     (JJ cold) (, ,)
     (JJ empty) (NN sky) )
   (VP (VBD was)
     (ADJP-PRD (JJ full)
       (PP (IN of)
         (NP (NN fire)
           (CC and)
           (NN light) ))))
   (. .) ))
```

(a)

```
((S
   (NP-SBJ The/DT flight/NN )
   (VP should/MD
     (VP arrive/VB
       (PP-TMP at/IN
         (NP eleven/CD a.m/RB ))
       (NP-TMP tomorrow/NN )))))
```

(b)

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 ('' '')
    (S-TPC-2
      (NP-SBJ-1 (PRP We) )
      (VP (MD would)
        (VP (VB have)
          (S
            (NP-SBJ (-NONE- *-1) )
            (VP (TO to)
              (VP (VB wait)
                (SBAR-TMP (IN until)
                  (S
                    (NP-SBJ (PRP we) )
                    (VP (VBP have)
                      (VP (VBN collected)
                        (PP-CLR (IN on)
                          (NP (DT those) (NNS assets) )))))))))))))
    (, ,) ('' '')
    (NP-SBJ (PRP he) )
    (VP (VBD said)
      (S (-NONE- *T*-2) ))
    (. .) ))
```

Treebank Grammars

S	→	*NP VP .*
		NP VP
		" S " , NP VP .
		-NONE-
		DT NN
		DT NN NNS
		NN CC NN
		CD RB
NP	→	*DT JJ , JJ NN*
		PRP
		-NONE-
VP	→	*MD VP*
		VBD ADJP
		VBD S
		VB PP
		VB S
		VB SBAR
		VBP VP
		VBN VP
		TO VP
SBAR	→	*IN S*
ADJP	→	*JJ PP*
PP	→	*IN NP*

PRP	→	*we* | *he*
DT	→	*the* | *that* | *those*
JJ	→	*cold* | *empty* | *full*
NN	→	*sky* | *fire* | *light* | *flight*
NNS	→	*assets*
CC	→	*and*
IN	→	*of* | *at* | *until* | *on*
CD	→	*eleven*
RB	→	*a.m*
VB	→	*arrive* | *have* | *wait*
VBD	→	*said*
VBP	→	*have*
VBN	→	*collected*
MD	→	*should* | *would*
TO	→	*to*

Lots of flat rules

```
NP → DT JJ NN
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 sentences from those rules

- Total: over 17,000 different grammar rules in the 1-million word Treebank corpus

(9.19) [$_{\text{DT}}$ The] [$_{\text{JJ}}$ state-owned] [$_{\text{JJ}}$ industrial] [$_{\text{VBG}}$ holding] [$_{\text{NN}}$ company] [$_{\text{NNP}}$ Instituto] [$_{\text{NNP}}$ Nacional] [$_{\text{FW}}$ de] [$_{\text{NNP}}$ Industria]

(9.20) [$_{\text{NP}}$ Shearson's] [$_{\text{JJ}}$ easy-to-film], [$_{\text{JJ}}$ black-and-white] "[$_{\text{SBAR}}$ Where We Stand]" [$_{\text{NNS}}$ commercials]

Parsing

- The parser can be viewed as searching through the space of possible parse trees to find the correct parse tree for a given sentence.
- As with everything of interest, parsing involves a search which involves the making of choices
- We’ll start with some basic (meaning bad) methods before moving on to the one or two that you need to know

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

Parsing

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

Figure 11.1 The $\mathscr{L}_1$ miniature English grammar and lexicon.

Parsing

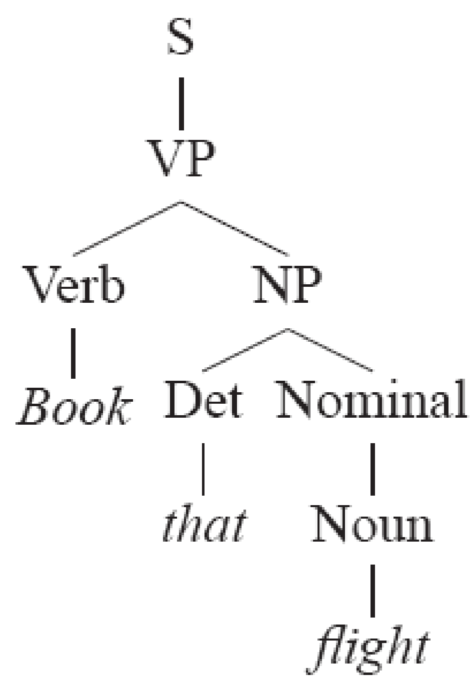
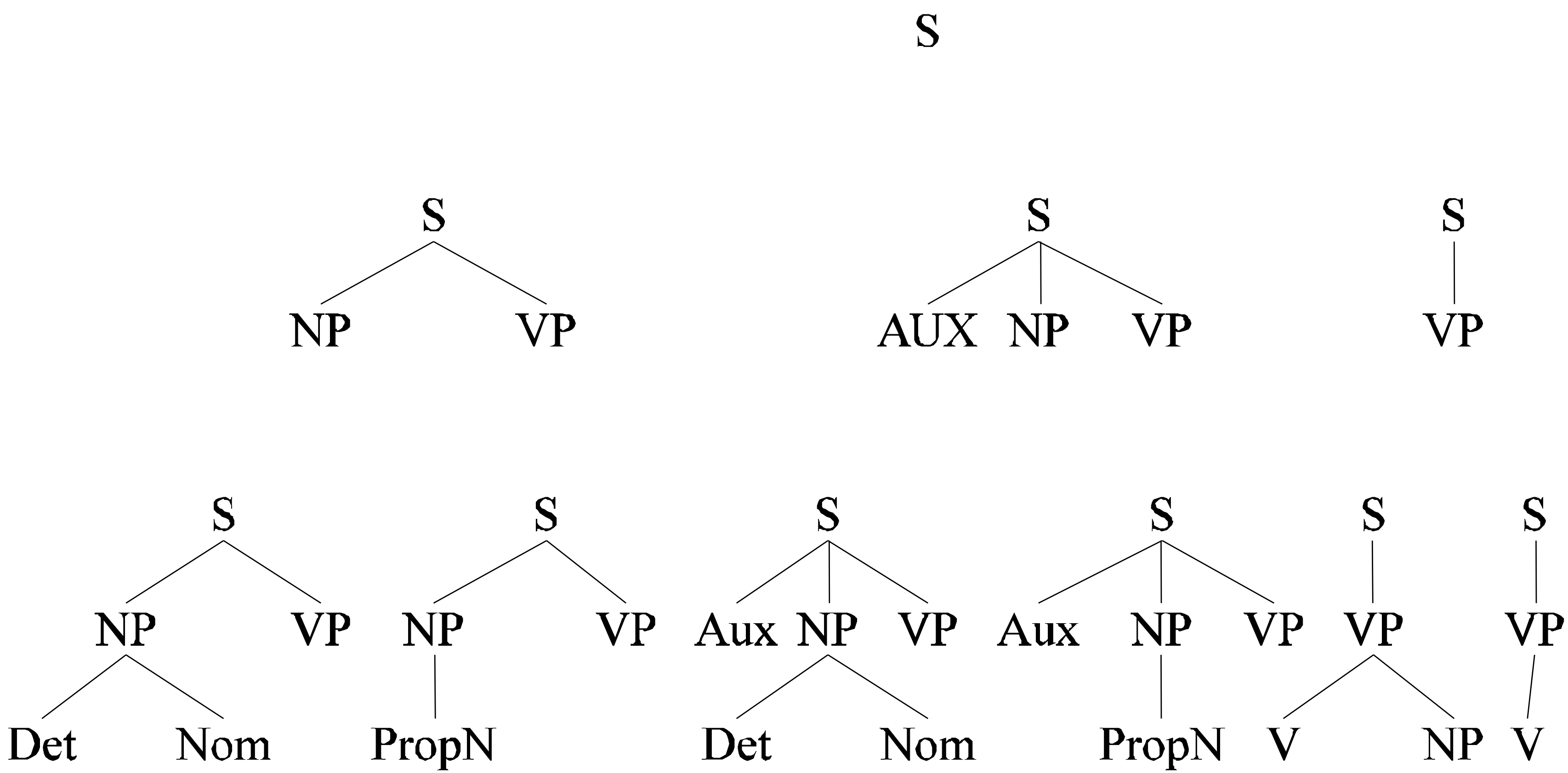


Figure 11.2 The parse tree for the sentence *Book that flight* according to grammar $\mathscr{L}_1$.

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.

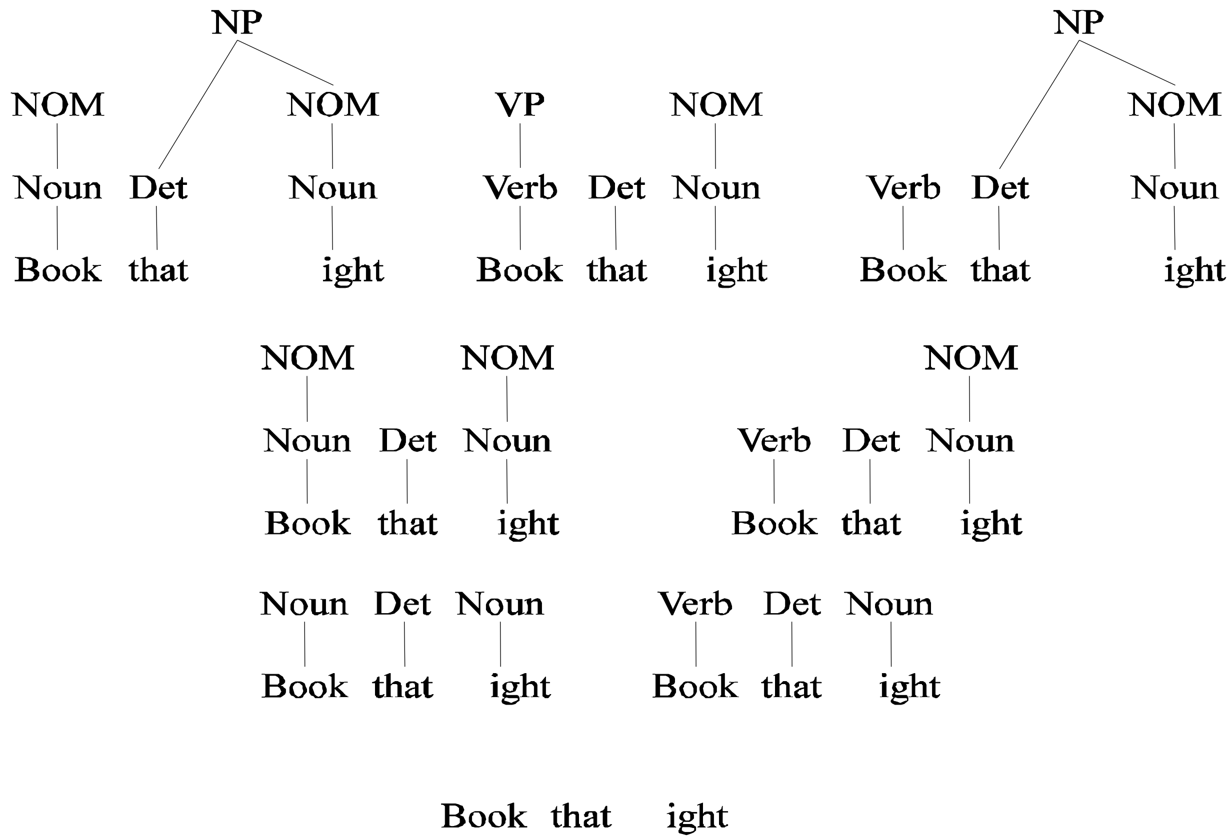
Top Down Space



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.

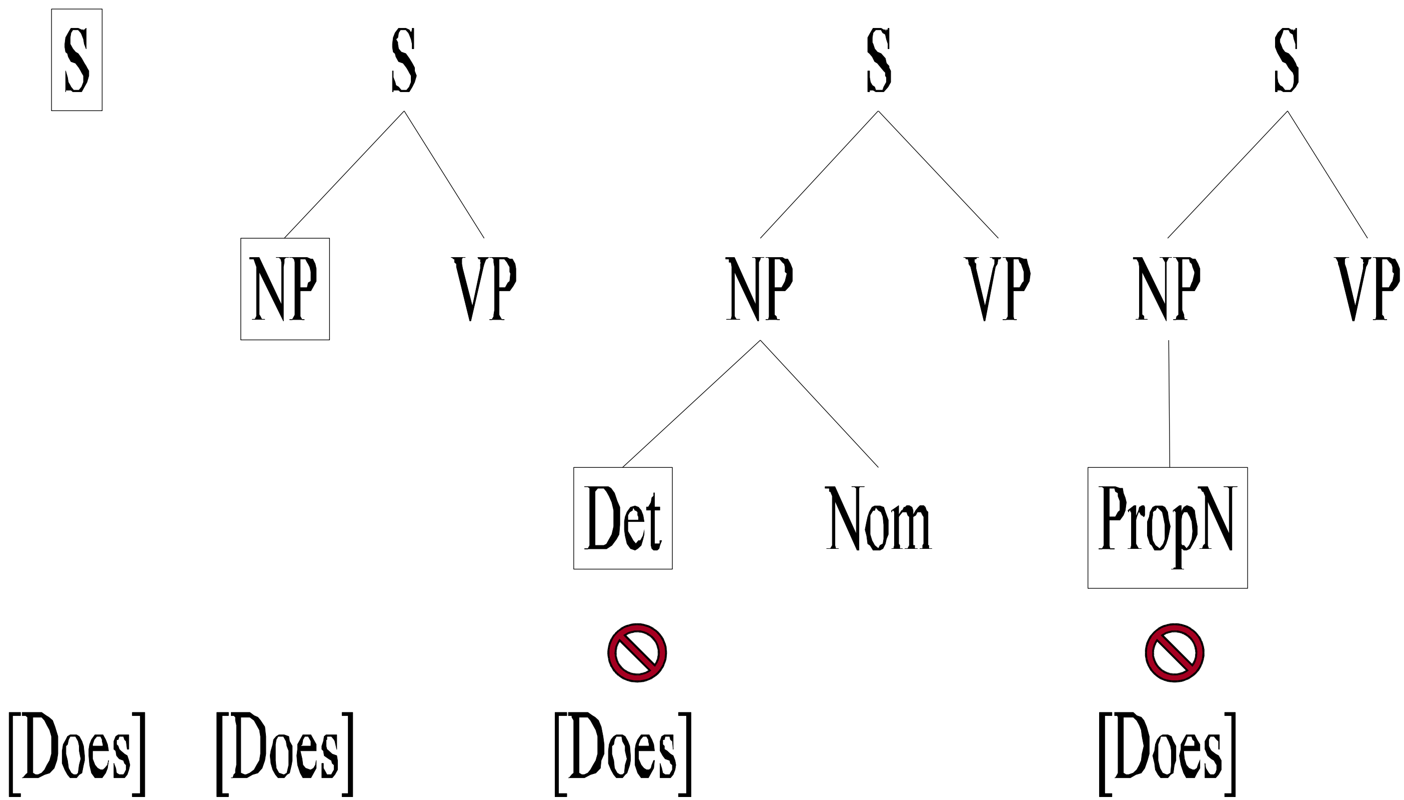
Bottom-Up Space



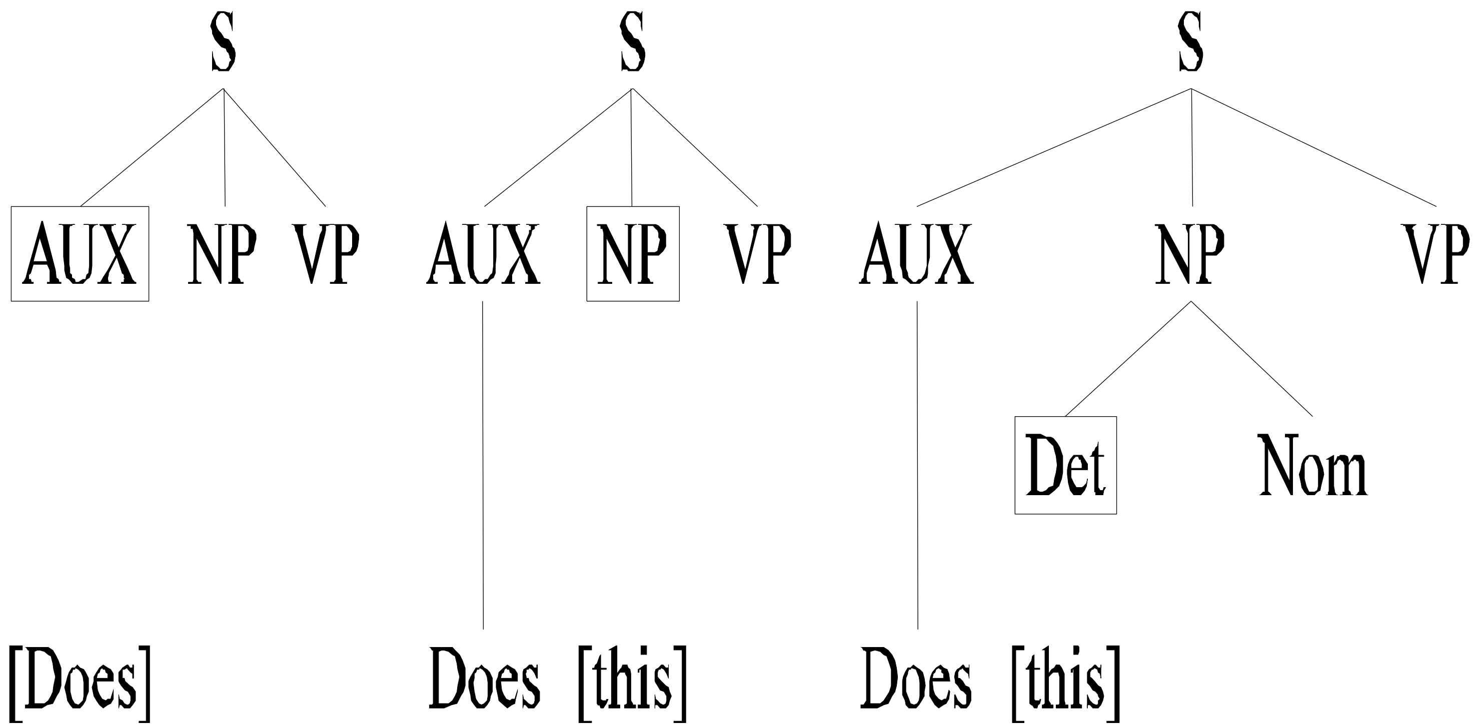
Control

- 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

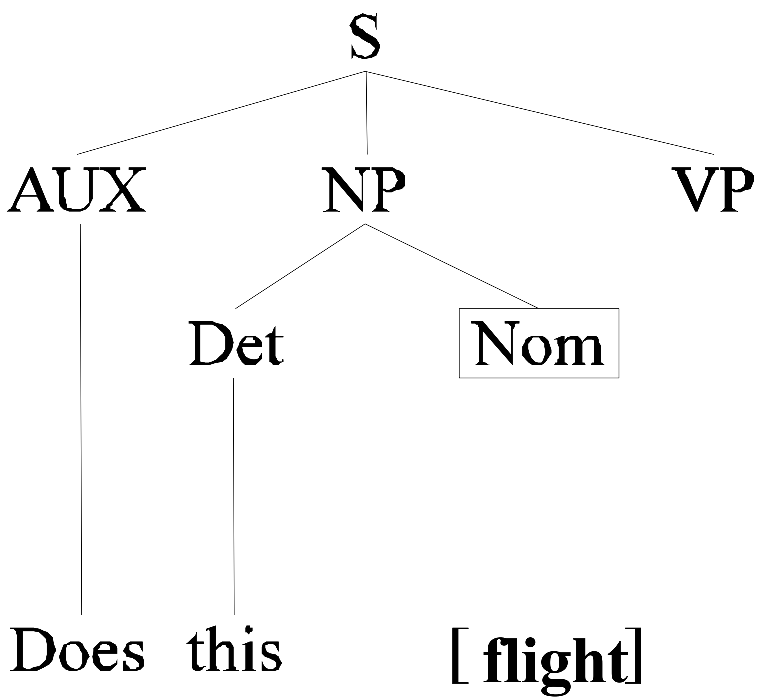
Top-Down, Depth-First, Left-to-Right Search

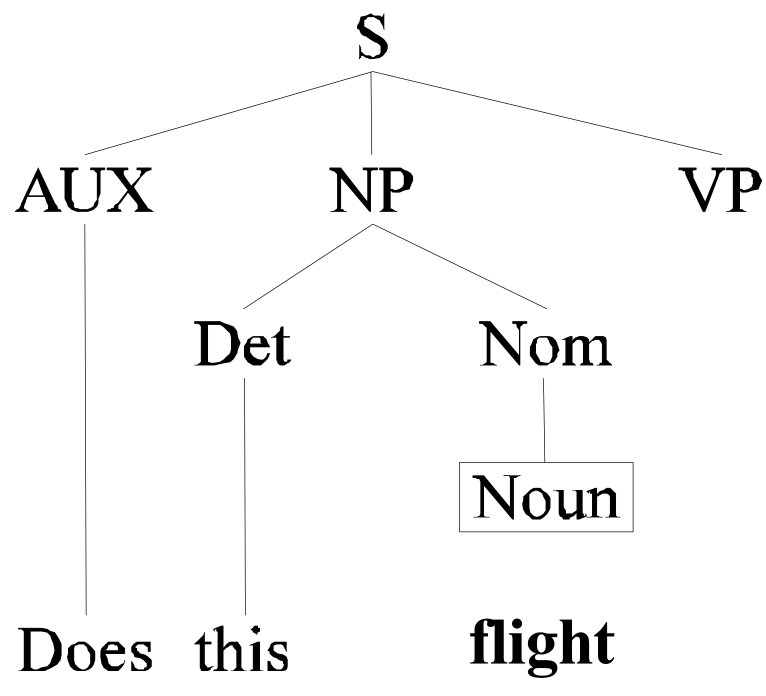


Example

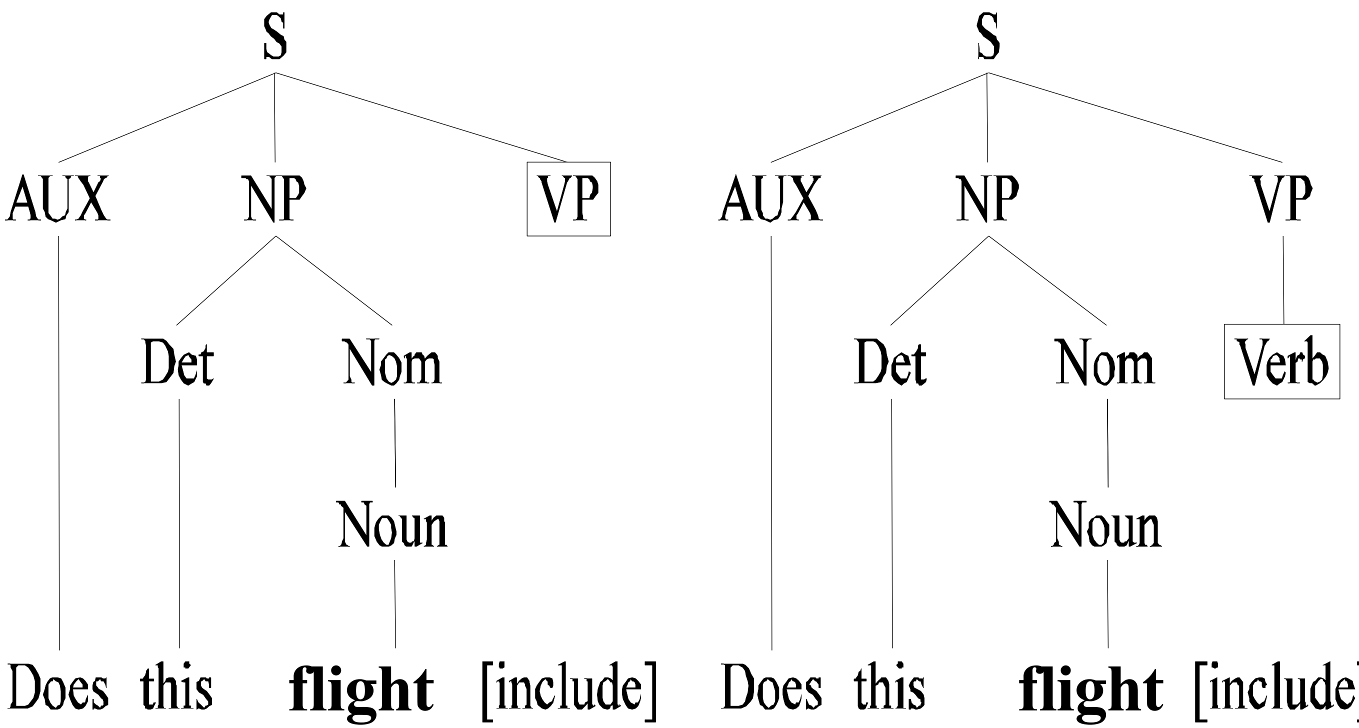

S
AUX
NP
VP
[Does]
S
AUX
NP
VP
Does
[this]
S
AUX
NP
VP
Det
Nom
Does
[this]

TopDownDepthFirstLeftoRight





TopDownDepthFirstLeftoRight



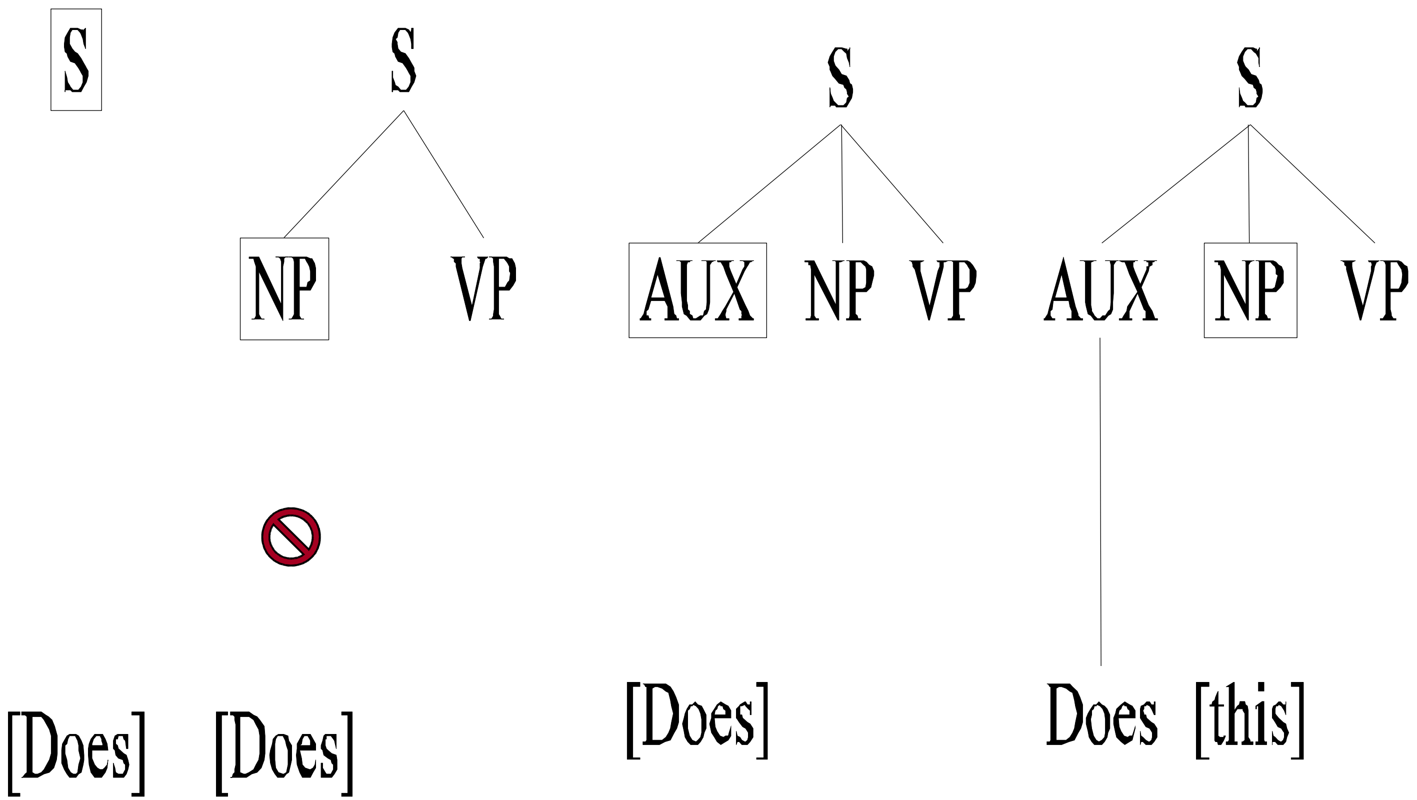
Top-Down and Bottom-Up

- Top-down
  - Only searches for trees that can be answers (i.e. S's)
  - But also suggests trees that are not consistent with the words
- Bottom-up
  - Only forms trees consistent with the words
  - Suggest trees that make no sense globally

So Combine Them

- There are a million ways to combine top-down expectations with bottom-up data to get more efficient searches
- Most use one kind as the control and the other as a filter
  - As in top-down parsing with bottom-up filtering

Adding Bottom-Up Filtering



3 problems with TDDFLtR Parser

- Left-Recursion
- Ambiguity
- Inefficient reparsing of subtrees

Left-Recursion

- What happens in the following situation
  - S -> NP VP
  - S -> Aux NP VP
  - NP -> NP PP
  - NP -> Det Nominal
  - …
  - With the sentence starting with
    - Did the flight…

Ambiguity

- One morning I shot an elephant in my pyjamas. How he got into my pajamas I don’t know.   (Groucho Marx)

Lots of ambiguity

- VP -> VP PP
- NP -> NP PP
- Show me the meal on flight 286 from SF to Denver
- 14 parses!

Lots of ambiguity

- Church and Patil (1982)
  - Number of parses for such sentences grows at rate of number of parenthesizations of arithmetic expressions
  - Which grow with Catalan numbers

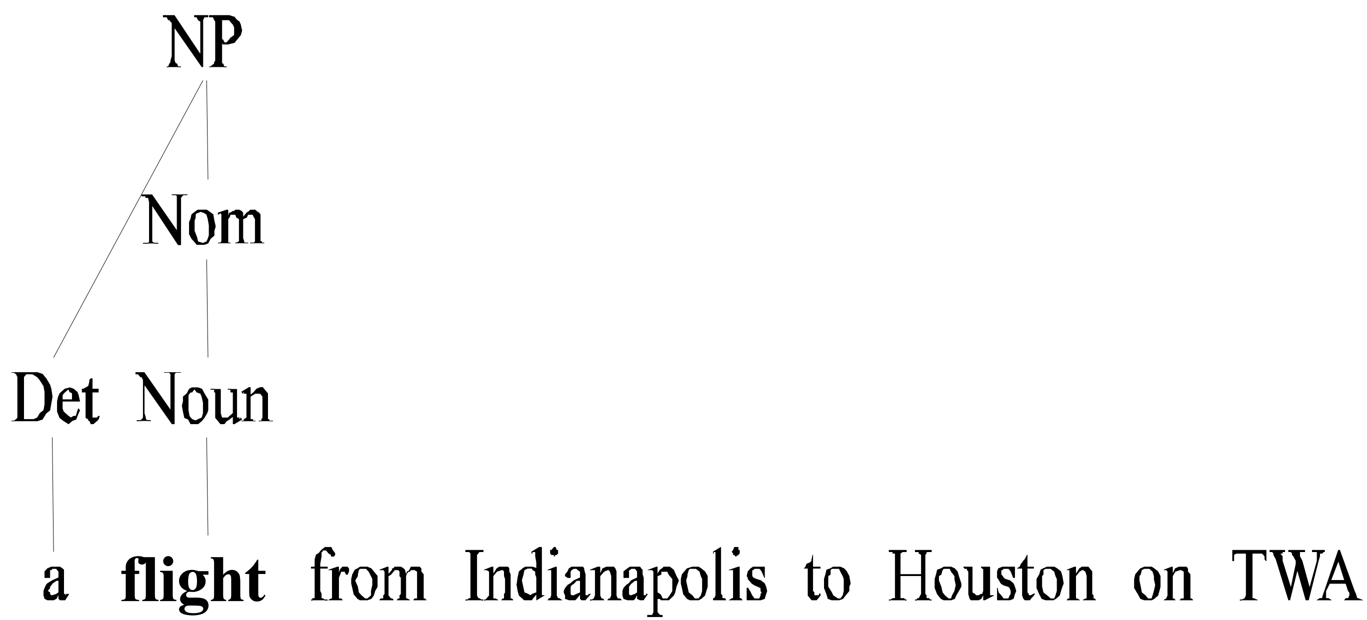
$$C(n) = \frac{1}{n+1}\binom{2n}{n}$$

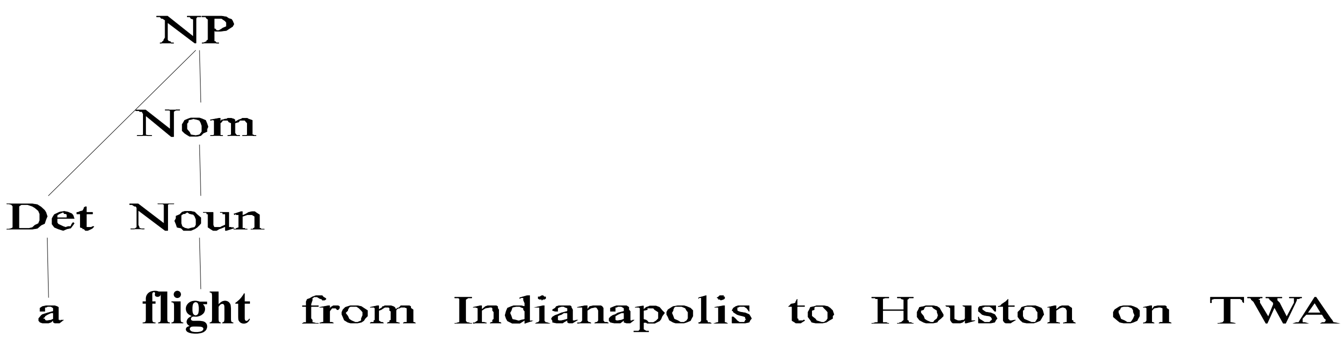
PPs	**Parses**
1	**2**
2	**5**
3	**14**
4	**132**
5	**469**
6	**1430**

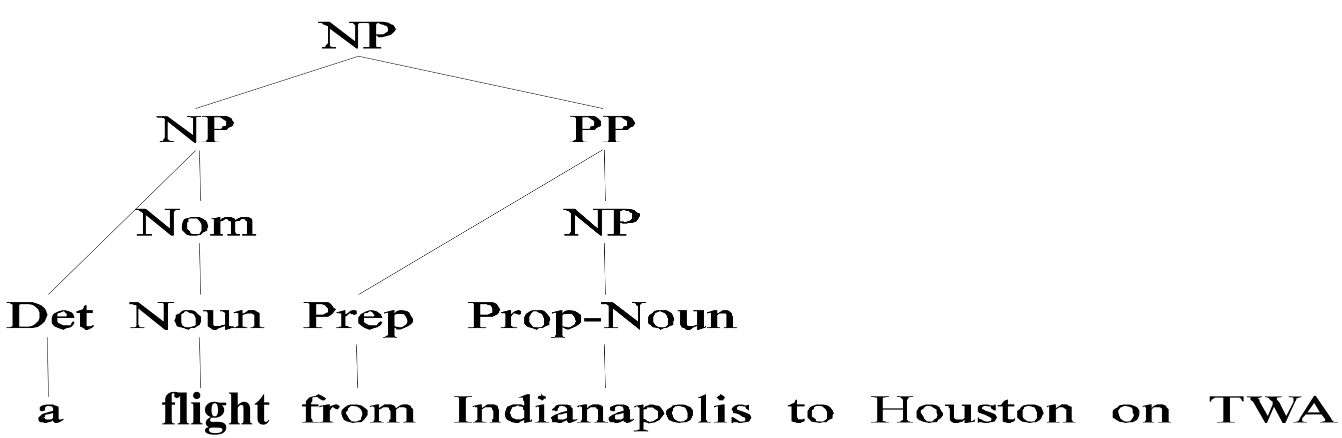
Avoiding Repeated Work

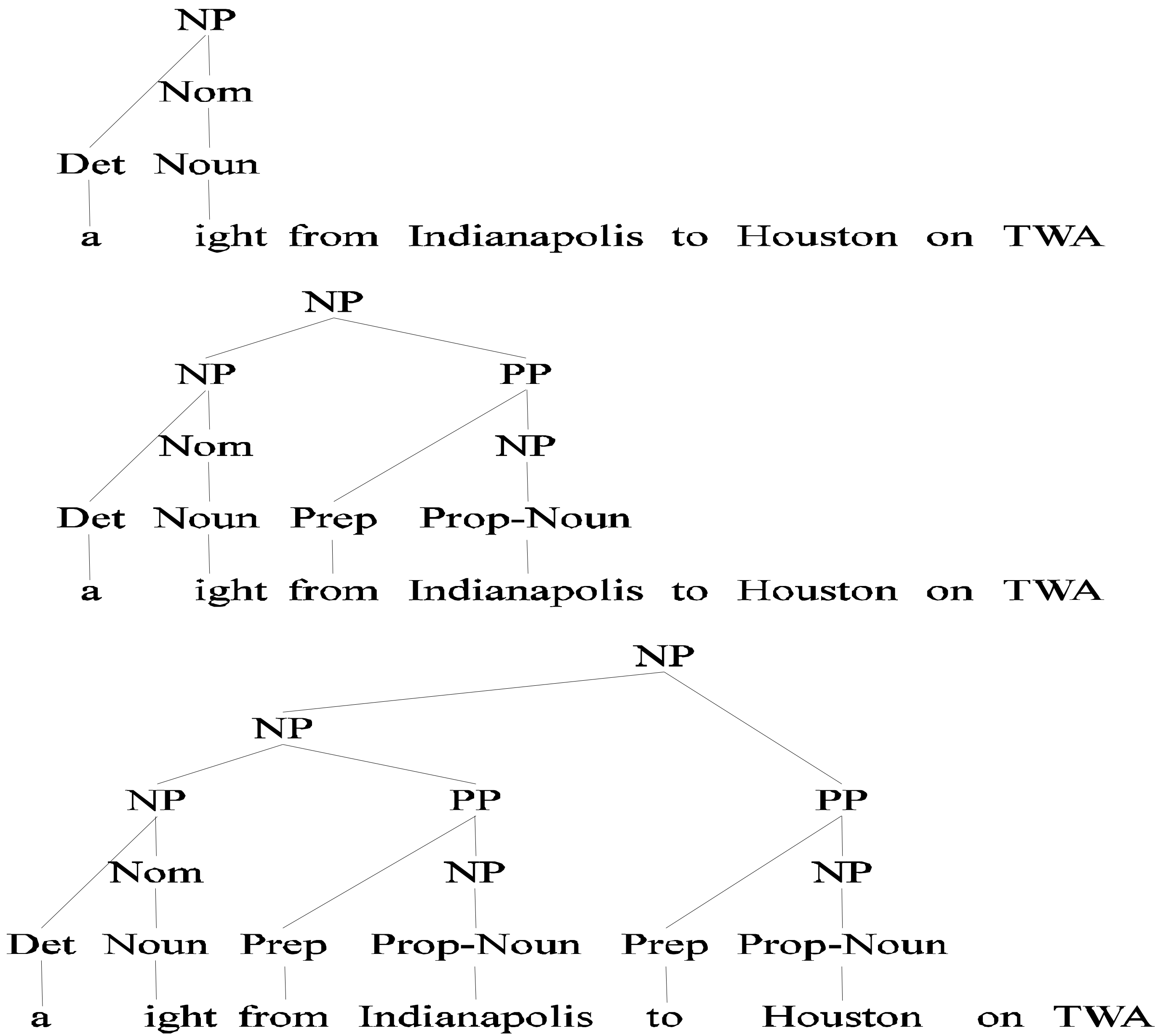
- Parsing is hard, and slow. It’s wasteful to redo stuff over and over and over.
- Consider an attempt to top-down parse the following as an NP

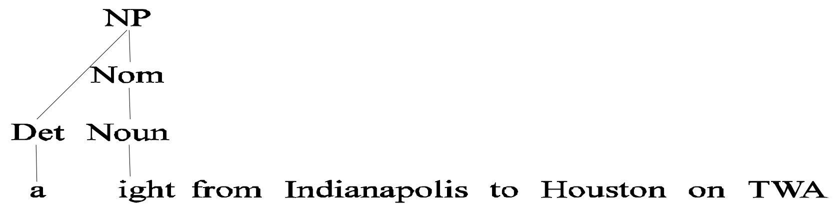
A flight from Indianapolis to Houston on TWA

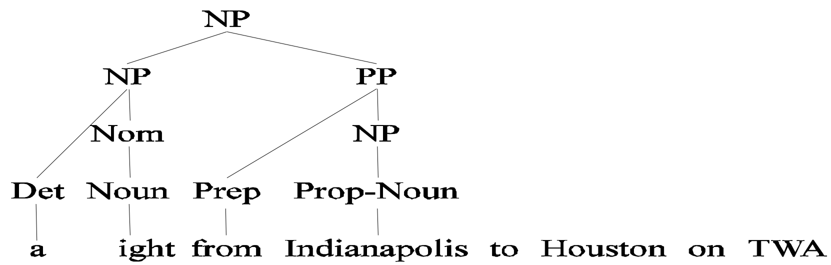

NP
Nom
Det
Noun
a **flight** from Indianapolis to Houston on TWA

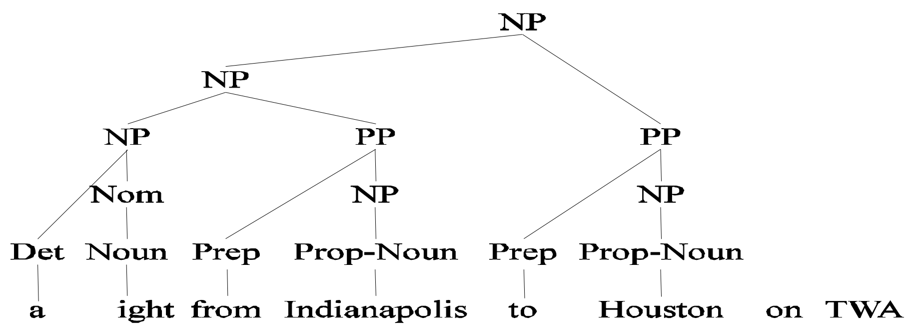

NP
Nom
Det Noun
a flight from Indianapolis to Houston on TWA

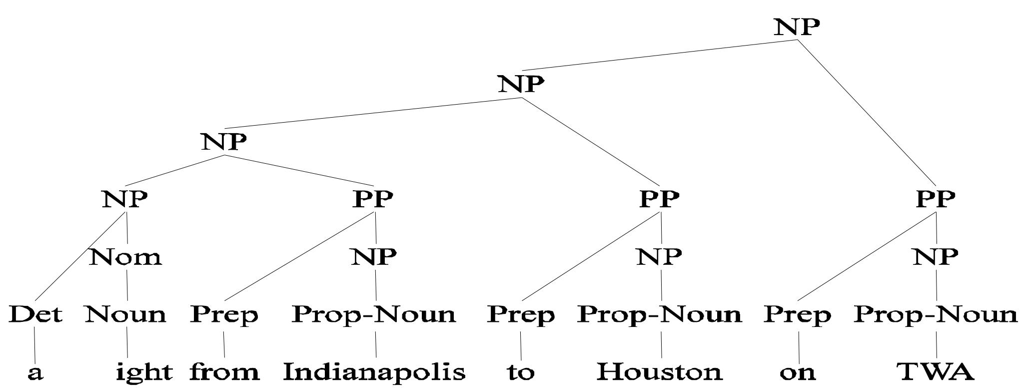

NP
NP PP
Nom NP
Det Noun Prep Prop-Noun
a flight from Indianapolis to Houston on TWA


NP
Nom
Det Noun
a ight from Indianapolis to Houston on TWA
NP
NP PP
Nom NP
Det Noun Prep Prop-Noun
a ight from Indianapolis to Houston on TWA
NP
NP
NP PP PP
Nom NP NP
Det Noun Prep Prop-Noun Prep Prop-Noun
a ight from Indianapolis to Houston on TWA


NP
Nom
Det Noun
a ight from Indianapolis to Houston on TWA


NP
NP PP
Nom NP
Det Noun Prep Prop-Noun
a ight from Indianapolis to Houston on TWA


NP
NP
NP PP PP
Nom NP NP
Det Noun Prep Prop-Noun Prep Prop-Noun
a ight from Indianapolis to Houston on TWA


NP
NP
NP
NP PP PP PP
Nom NP NP NP
Det Noun Prep Prop-Noun Prep Prop-Noun Prep Prop-Noun
a ight from Indianapolis to Houston on TWA

Dynamic Programming

- We need a method that fills a table with partial results that
  - Does not do (avoidable) repeated work
  - Does not fall prey to left-recursion
  - Can find all the pieces of an exponential number of trees in polynomial time.
- We'll introduce 2
  - CKY
  - Earley

The CKY (Cocke-Kasami-Younger) Algorithm

- Requires the grammar be in Chomsky Normal Form (CNF)
  - All rules must be in following form:
    - A -> B C
    - A -> w
- Any grammar can be converted automatically to Chomsky Normal Form

Converting to CNF

- Rules that mix terminals and non-terminals
  - Introduce a new dummy non-terminal that covers the terminal
    - INFVP -> to VP   replaced by:
    - INFVP -> TO VP
    - TO -> to
- Rules that have a single non-terminal on right ("unit productions")
  - Rewrite each unit production with the RHS of their expansions
- Rules whose right hand side length >2
  - Introduce dummy non-terminals that spread the right-hand side

Automatic Conversion to CNF

Original L0 Grammar	CNF
$S \rightarrow NP\ VP$	$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$	$S \rightarrow X1\ VP$
	$X1 \rightarrow Aux\ NP$
$S \rightarrow VP$	$S \rightarrow book \mid include \mid prefer$
	$S \rightarrow Verb\ NP$
	$S \rightarrow VP\ PP$
$NP \rightarrow Det\ Nominal$	$NP \rightarrow Det\ Nominal$
$NP \rightarrow Proper\text{-}Noun$	$NP \rightarrow TWA \mid Houston$
$NP \rightarrow Pronoun$	$NP \rightarrow I \mid she \mid me$
$Nominal \rightarrow Noun$	$Nominal \rightarrow book \mid flight \mid meal \mid money$
$Nominal \rightarrow Noun\ Nominal$	$Nominal \rightarrow Noun\ Nominal$
$Nominal \rightarrow Nominal\ PP$	$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow Verb$	$VP \rightarrow book \mid include \mid prefer$
$VP \rightarrow Verb\ NP$	$VP \rightarrow Verb\ NP$
$VP \rightarrow VP\ PP$	$VP \rightarrow VP\ PP$
$PP \rightarrow Prep\ NP$	$PP \rightarrow Prep\ NP$

Figure 10.15 Original L0 Grammar and its conversion to CNF

CKY Recognition

- We will use a simple two-dimensional matrix to encode the structure of a parse tree
  - Like other dynamic programming methods!
- For a sentence of length *n*
  - We will use the upper-triangular portion
    - Of an (n+1) x (n+1) matrix
  - Each cell [i,j] contains the set of constituents that span positions i thru j of the input:
  - NP[1,3]
  - ________________
  - 0 Book 1 the 2 flight 3 through 4 Chicago 5

CKY Recognition

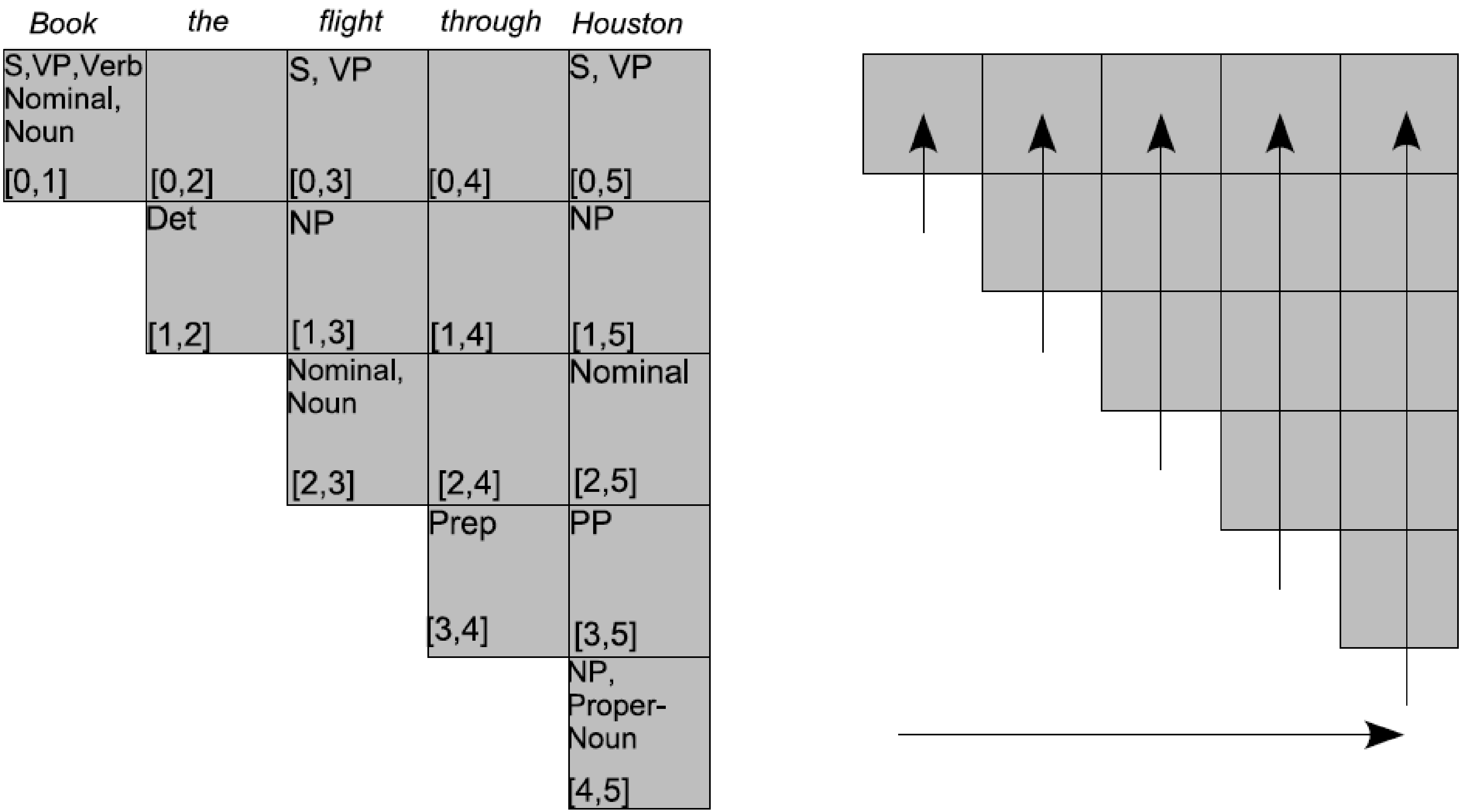
- Each cell [i,j] contains the set of constituents that span positions i thru j of the input
- CNF -> Each non-terminal has exactly 2 daughters
- Therefore, for each constituent covering [i,j]
  - There must be a point k, i < k < j, where it can be split
  - Given such a k, the first constituent [i,k] lies to left
  - And the second constituent [k,j] lies beneath on column j

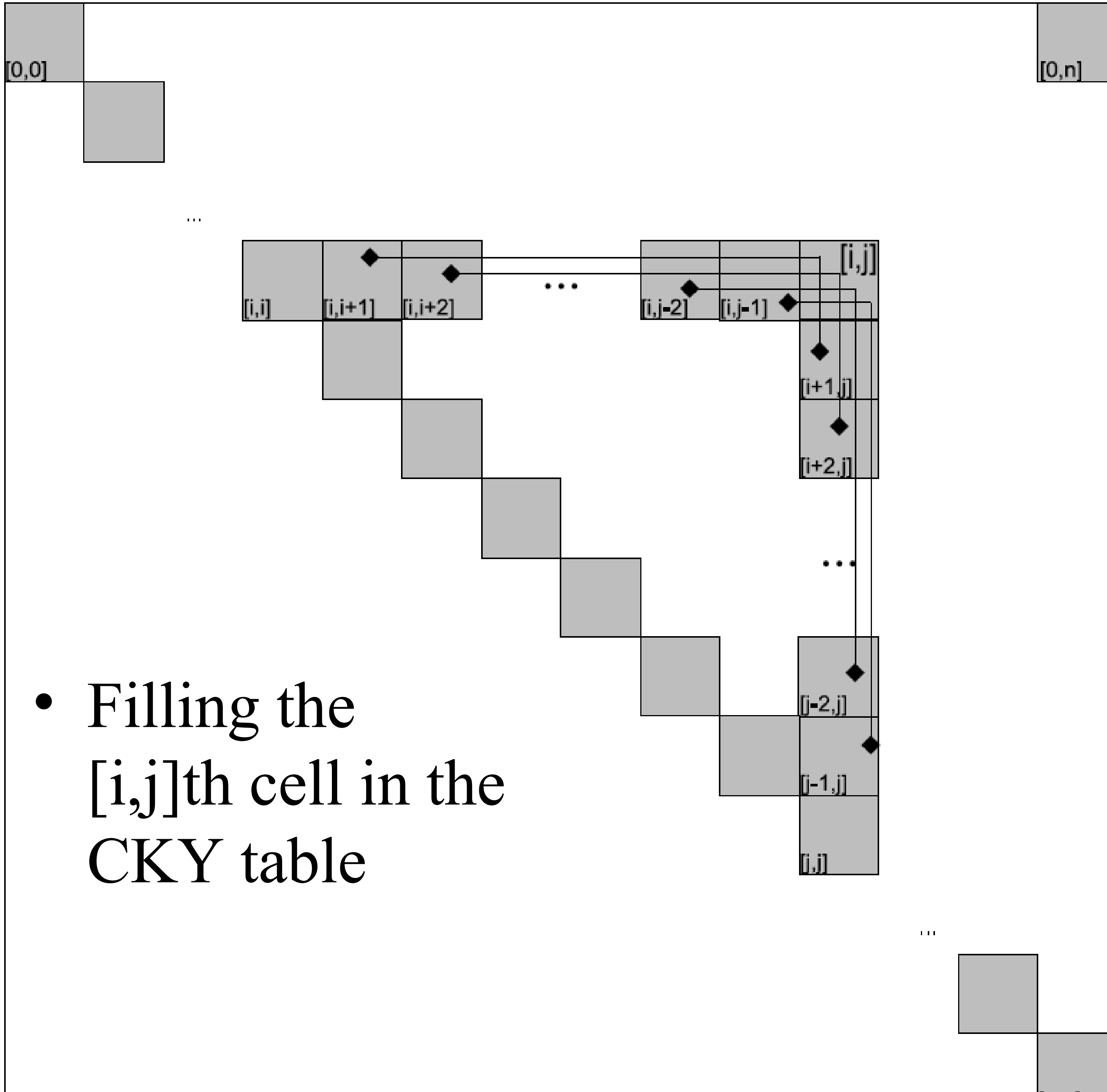
CKY Algorithm

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**
      $table[i,j] \leftarrow table[i,j] \cup$
        $\{A \mid A \rightarrow BC \in grammar,$
          $B \in table[i,k],$
          $C \in table[k,j]\}$

0 Book 1 the 2 flight 3 through 4 Chicago 5



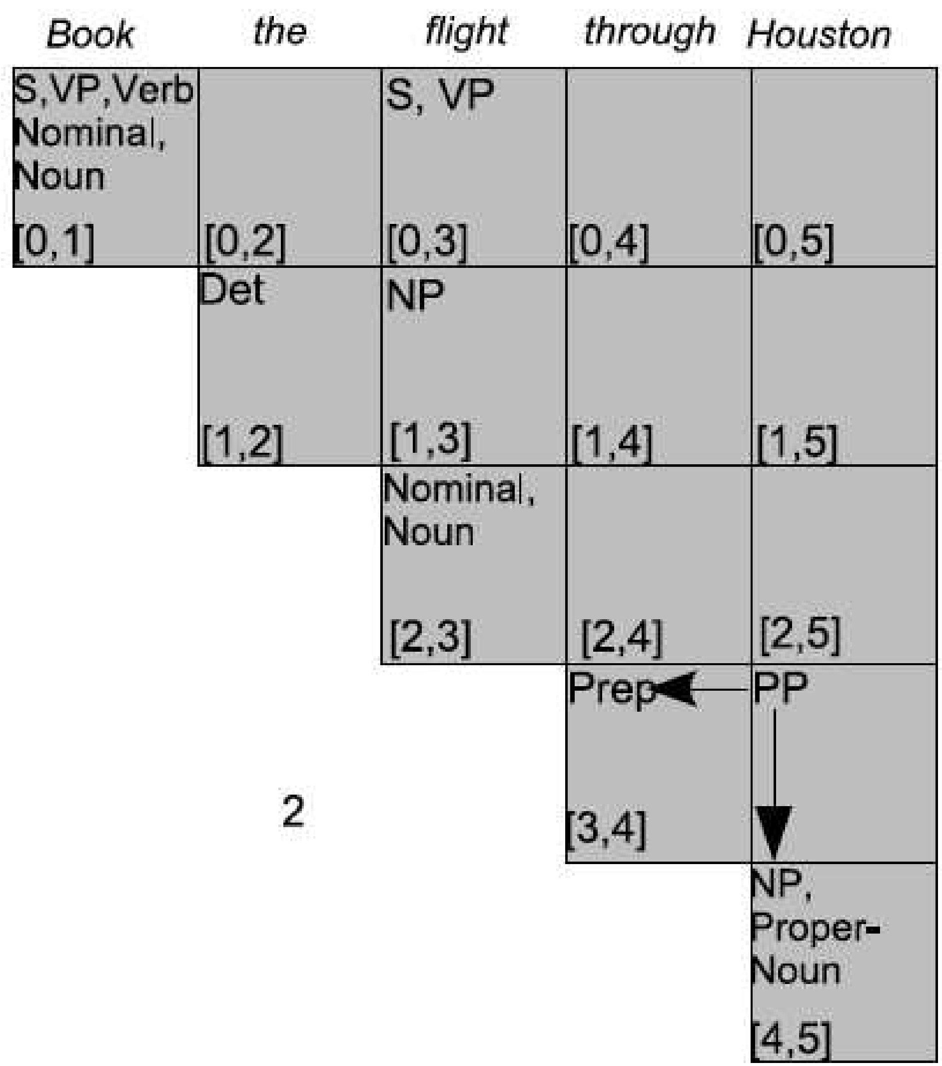


- Filling the [i,j]th cell in the CKY table

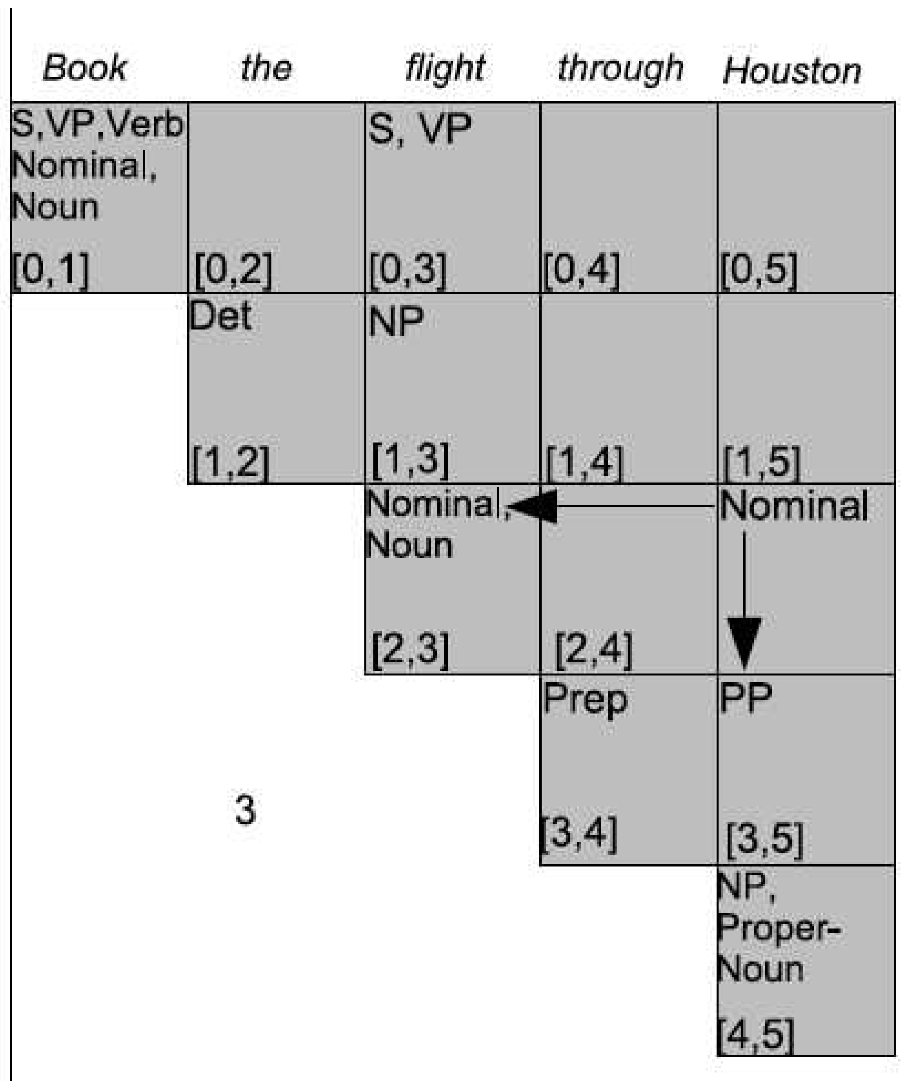
Filling the last column after reading the word Houston

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

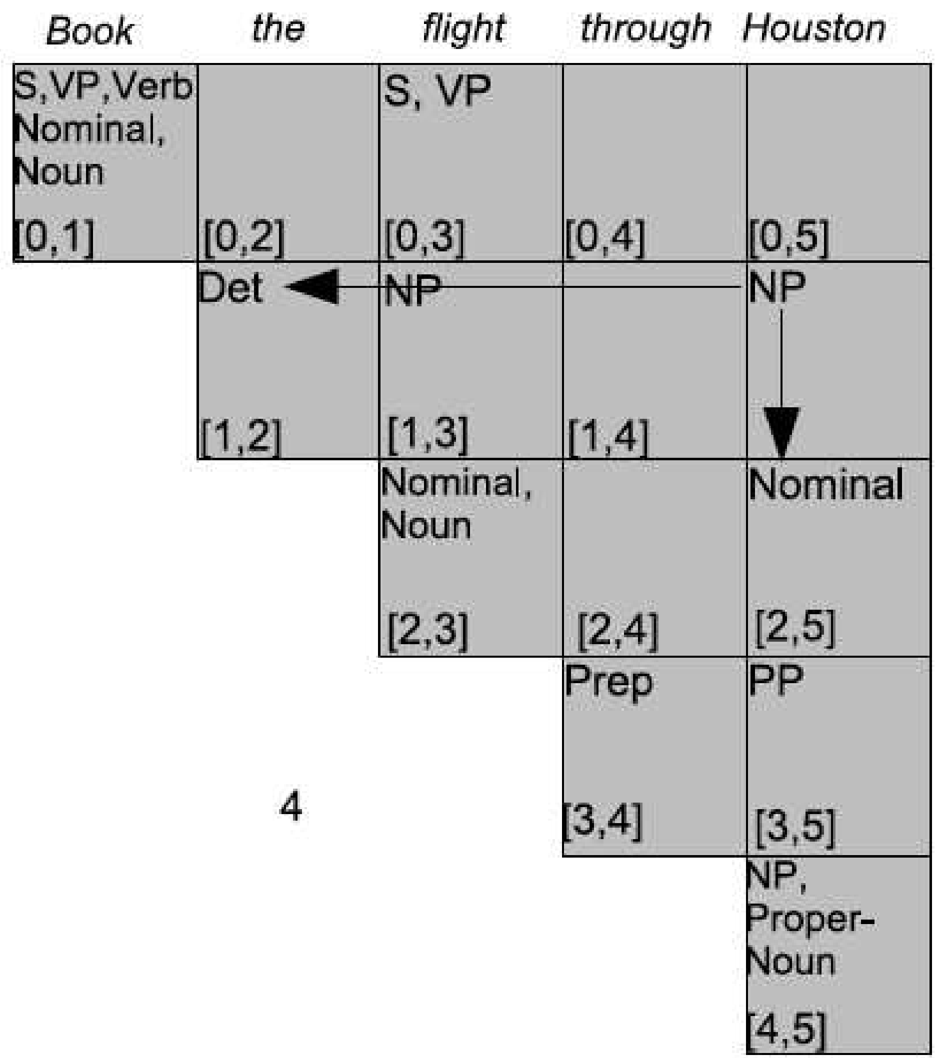
Filling the last column after reading the word Chicago



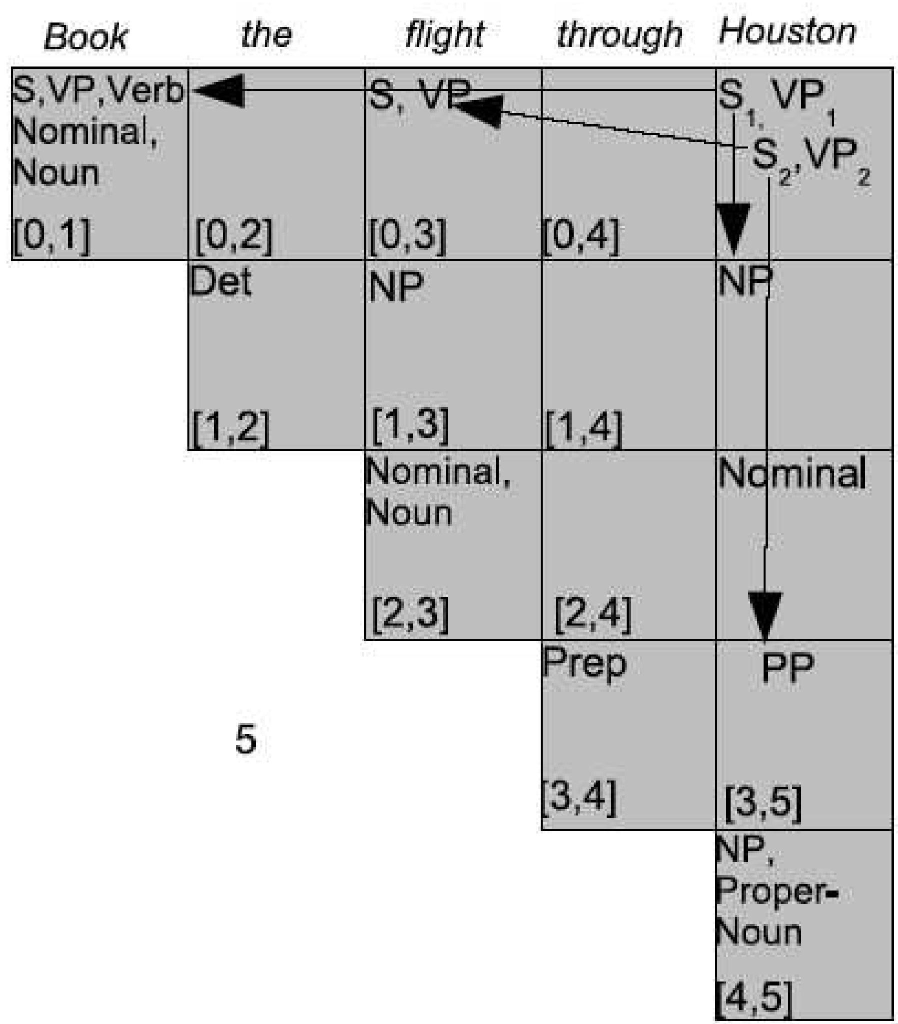
Filling the last column after reading the word Chicago



Filling the last column after reading the word Chicago



Filling the last column after reading the word Chicago



Parsing and Ambiguity

- We can store all the different parses efficiently
- But retrieving parses, we still have to do all the exponential work
- So in practice, we will need some way to do disambiguation as we go, so we don't have to store every parse of very ambiguous sentences.

Earley Parsing

- Doesn't require CNF grammars
- Where CKY is bottom-up, Earley is top-down
- Fills a table in a single sweep over the input words
  - Table is length N+1; N is number of words
  - Table entries represent
    - Completed constituents and their locations
    - In-progress constituents
    - Predicted constituents

Earley States

- The table-entries are called states and are represented with dotted-rules.

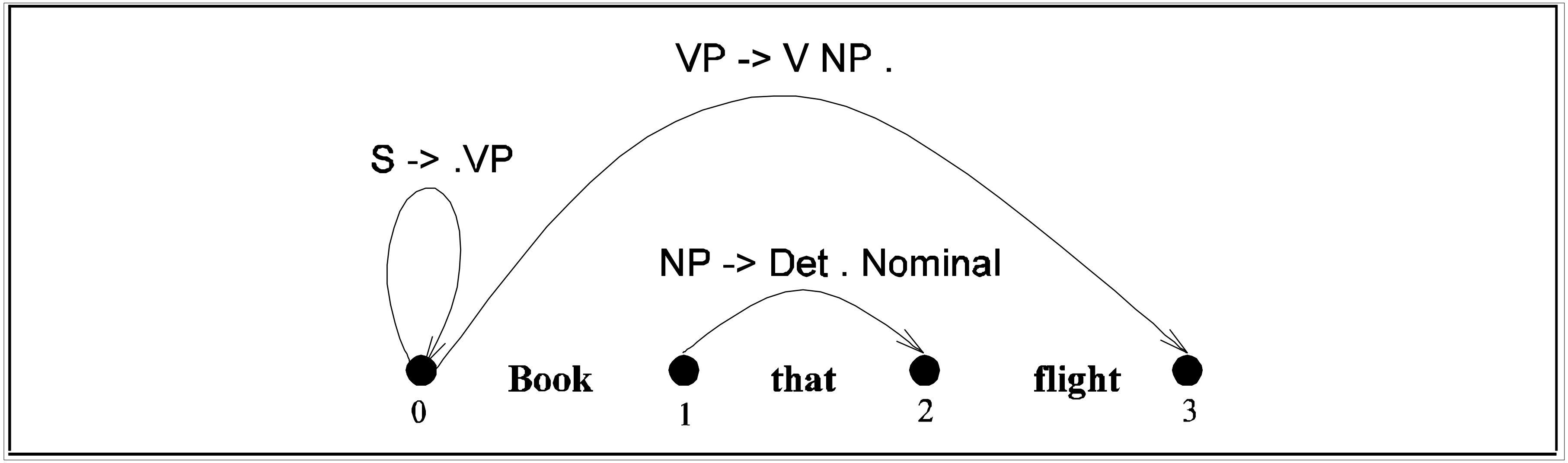
S -> · VP	A VP is predicted
NP -> Det · Nominal	An NP is in progress
VP -> V NP ·	A VP has been found

Earley States/Locations

- We need to know where these things are in the input:

S -> · VP [0,0]	A VP is predicted at the start of the sentence
NP -> Det · Nominal	[1,2] An NP is in progress; the Det goes from 1 to 2
VP -> V NP · [0,3]	A VP has been found starting at 0 and ending at 3

Graphically



Earley Algorithm

- March through chart left-to-right.
- At each step, apply 1 of 3 operators
  - Predictor
    - Create new states representing top-down expectations
  - Scanner
    - Match word predictions (rule with word after dot) to words
  - Completer
    - When a state is complete, see what rules were looking for that completed constituent

Predictor

- Given a state
  – With a non-terminal to right of dot
  – That is not a part-of-speech category
  – Create a new state for each expansion of the non-terminal
  – Place these new states into same chart entry as generated state, beginning and ending where generating state ends.
  – So predictor looking at
    - S -> . VP [0,0]
  – results in
    - VP -> . Verb [0,0]
    - VP -> . Verb NP [0,0]

Scanner

- Given a state
  - With a non-terminal to right of dot
  - That is a part-of-speech category
  - If the next word in the input matches this part-of-speech
  - Create a new state with dot moved over the non-terminal
  - So scanner looking at
    - VP -> . Verb NP [0,0]
  - If the next word, “book”, can be a verb, add new state:
    - VP -> Verb . NP [0,1]
  - Add this state to chart entry following current one
  - Note: Earley algorithm uses top-down input to disambiguate POS! Only POS predicted by some state can get added to chart

Completer

- Applied to a state when its dot has reached right end of role.
- Parser has discovered a category over some span of input.
- Find and advance all previous states that were looking for this category
  - copy state, move dot, insert in current chart entry
- Given:
  - NP -> Det Nominal . [1,3]
  - VP -> Verb. NP [0,1]
- Add
  - VP -> Verb NP . [0,3]

Earley: how do we know we are done?

- How do we know when we are done?.
- Find an S state in the final column that spans from 0 to n+1 and is complete.
- If that's the case you're done.
  - $S \rightarrow \alpha \cdot [0,n+1]$

Earley

- So sweep through the table from 0 to n+1…
  - New predicted states are created by starting top-down from S
  - New incomplete states are created by advancing existing states as new constituents are discovered
  - New complete states are created in the same way.

Earley

- More specifically…
  1. Predict all the states you can upfront
  2. Read a word
     1. Extend states based on matches
     2. Add new predictions
     3. Go to 2
  3. Look at N+1 to see if you have a winner

Example

- Book that flight
- We should find… an S from 0 to 3 that is a completed state…

Example

Chart[0]	S0	$\gamma \rightarrow \bullet S$	[0,0]	Dummy start state
	S1	$S \rightarrow \bullet NP\ VP$	[0,0]	Predictor
	S2	$S \rightarrow \bullet Aux\ NP\ VP$	[0,0]	Predictor
	S3	$S \rightarrow \bullet VP$	[0,0]	Predictor
	S4	$NP \rightarrow \bullet Pronoun$	[0,0]	Predictor
	S5	$NP \rightarrow \bullet Proper\text{-}Noun$	[0,0]	Predictor
	S6	$NP \rightarrow \bullet Det\ Nominal$	[0,0]	Predictor
	S7	$VP \rightarrow \bullet Verb$	[0,0]	Predictor
	S8	$VP \rightarrow \bullet Verb\ NP$	[0,0]	Predictor
	S9	$VP \rightarrow \bullet Verb\ NP\ PP$	[0,0]	Predictor
	S10	$VP \rightarrow \bullet Verb\ PP$	[0,0]	Predictor
	S11	$VP \rightarrow \bullet VP\ PP$	[0,0]	Predictor

Example

Chart[0]	S0	$\gamma \rightarrow \bullet S$	[0,0]	Dummy start state
	S1	$S \rightarrow \bullet NP\ VP$	[0,0]	Predictor
	S2	$S \rightarrow \bullet Aux\ NP\ VP$	[0,0]	Predictor
	S3	$S \rightarrow \bullet VP$	[0,0]	Predictor
	S4	$NP \rightarrow \bullet Pronoun$	[0,0]	Predictor
	S5	$NP \rightarrow \bullet Proper\text{-}Noun$	[0,0]	Predictor
	S6	$NP \rightarrow \bullet Det\ Nominal$	[0,0]	Predictor
	S7	$VP \rightarrow \bullet Verb$	[0,0]	Predictor
	S8	$VP \rightarrow \bullet Verb\ NP$	[0,0]	Predictor
	S9	$VP \rightarrow \bullet Verb\ NP\ PP$	[0,0]	Predictor
	S10	$VP \rightarrow \bullet Verb\ PP$	[0,0]	Predictor
	S11	$VP \rightarrow \bullet VP\ PP$	[0,0]	Predictor
Chart[1]	S12	$Verb \rightarrow book \bullet$	[0,1]	Scanner
	S13	$VP \rightarrow Verb \bullet$	[0,1]	Completer
	S14	$VP \rightarrow Verb \bullet NP$	[0,1]	Completer
	S15	$VP \rightarrow Verb \bullet NP\ PP$	[0,0]	Predictor
	S16	$VP \rightarrow Verb \bullet PP$	[0,0]	Predictor
	S17	$S \rightarrow VP \bullet$	[0,1]	Completer
	S18	$VP \rightarrow VP \bullet PP$	[0,1]	Completer
	S19	$NP \rightarrow \bullet Pronoun$	[1,1]	Predictor
	S20	$NP \rightarrow \bullet Proper\text{-}Noun$	[1,1]	Predictor
	S21	$NP \rightarrow \bullet Det\ Nominal$	[1,1]	Predictor
	S22	$PP \rightarrow \bullet Prep\ NP$	[1,1]	Predictor

Earley example cont'd

Chart[1]	S12	*Verb* → *book* •	[0,1]	Scanner
	S13	*VP* → *Verb* •	[0,1]	Completer
	S14	*VP* → *Verb* • *NP*	[0,1]	Completer
	S15	*VP* → *Verb* • *NP PP*	[0,0]	Predictor
	S16	*VP* → *Verb* • *PP*	[0,0]	Predictor
	S17	*S* → *VP* •	[0,1]	Completer
	S18	*VP* → *VP* • *PP*	[0,1]	Completer
	S19	*NP* → • *Pronoun*	[1,1]	Predictor
	S20	*NP* → • *Proper-Noun*	[1,1]	Predictor
	S21	*NP* → • *Det Nominal*	[1,1]	Predictor
	S22	*PP* → • *Prep NP*	[1,1]	Predictor
Chart[2]	S23	*Det* → *that* •	[1,2]	Scanner
	S24	*NP* → *Det* • *Nominal*	[1,2]	Completer
	S25	*Nominal* → • *Noun*	[2,2]	Predictor
	S26	*Nominal* → • *Nominal Noun*	[2,2]	Predictor
	S27	*Nominal* → • *Nominal PP*	[2,2]	Predictor

Example

Chart[2]	S23	*Det* → *that* •	[1,2]	Scanner
	S24	*NP* → *Det* • *Nominal*	[1,2]	Completer
	S25	*Nominal* → • *Noun*	[2,2]	Predictor
	S26	*Nominal* → • *Nominal Noun*	[2,2]	Predictor
	S27	*Nominal* → • *Nominal PP*	[2,2]	Predictor
Chart[3]	S28	*Noun* → *flight* •	[2,3]	Scanner
	S29	*Nominal* → *Noun* •	[2,3]	Completer
	S30	*NP* → *Det Nominal* •	[1,3]	Completer
	S31	*Nominal* → *Nominal* • *Noun*	[2,3]	Completer
	S32	*Nominal* → *Nominal* • *PP*	[2,3]	Completer
	S33	*VP* → *Verb NP* •	[0,3]	Completer
	S34	*VP* → *Verb NP* • *PP*	[0,3]	Completer
	S35	*PP* → • *Prep NP*	[3,3]	Predictor
	S36	*S* → *VP* •	[0,3]	Completer

What is it?

- What kind of parser did we just describe (trick question).
  - Earley parser… yes
  - Not a parser – a recognizer
    - The presence of an S state with the right attributes in the right place indicates a successful recognition.
    - But no parse tree… no parser
    - That’s how we solve (not) an exponential problem in polynomial time

Converting Earley from Recognizer to Parser

- With the addition of a few pointers we have a parser
- Augment the “Completer” to point to where we came from.

Augmenting the chart with structural information

Chart[1]	S12	*Verb* → *book* •	[0,1]	Scanner
Chart[2]	S23	*Det* → *that* •	[1,2]	Scanner
Chart[3]	S28	*Noun* → *flight* •	[2,3]	Scanner
	S29	*Nominal* → *Noun* •	[2,3]	(S28)
	S30	*NP* → *Det Nominal* •	[1,3]	(S23, S29)
	S33	*VP* → *Verb NP* •	[0,3]	(S12, S30)
	S36	*S* → *VP* •	[0,3]	(S33)

Figure 11.15 States that participate in the final parse of *Book that flight*, including structural parse information.

Retrieving Parse Trees from Chart

- All the possible parses for an input are in the table
- We just need to read off all the backpointers from every complete S in the last column of the table
- Find all the S -> X . [0,N+1]
- Follow the structural traces from the Completer
- Of course, this won't be polynomial time, since there could be an exponential number of trees
- So we can at least represent ambiguity efficiently

Earley and Left Recursion

- Earley solves the left-recursion problem without having to alter the grammar or artificially limiting the search.
  - Never place a state into the chart that's already there
  - Copy states before advancing them

Earley and Left Recursion: 1

- S -> NP VP
- NP -> NP PP

- Predictor, given first rule:
  - S -> · NP VP [0,0]
- Predicts:
  - NP -> · NP PP [0,0]
  - stops there since predicting same again would be redundant

Earley and Left Recursion: 2

- When a state gets advanced make a copy and leave the original alone…

- Say we have NP -> · NP PP [0,0]
- We find an NP from 0 to 2 so we create

  NP -> NP · PP [0,2]
- But we leave the original state as is

Dynamic Programming Approaches

- Earley
  - Top-down, no filtering, no restriction on grammar form
- CYK
  - Bottom-up, no filtering, grammars restricted to Chomsky-Normal Form (CNF)
- Details are not important...
  - Bottom-up vs. top-down
  - With or without filters
  - With restrictions on grammar form or not

How to do parse disambiguation

- Probabilistic methods
- Augment the grammar with probabilities
- Then modify the parser to keep only most probable parses
- And at the end, return the most probable parse

Probabilistic CFGs

- The probabilistic model
  - Assigning probabilities to parse trees
- Getting the probabilities for the model
- Parsing with probabilities
  - Slight modification to dynamic programming approach
  - Task is to find the max probability tree for an input

Probability Model

- Attach probabilities to grammar rules
- The expansions for a given non-terminal sum to 1

  VP -> Verb .55

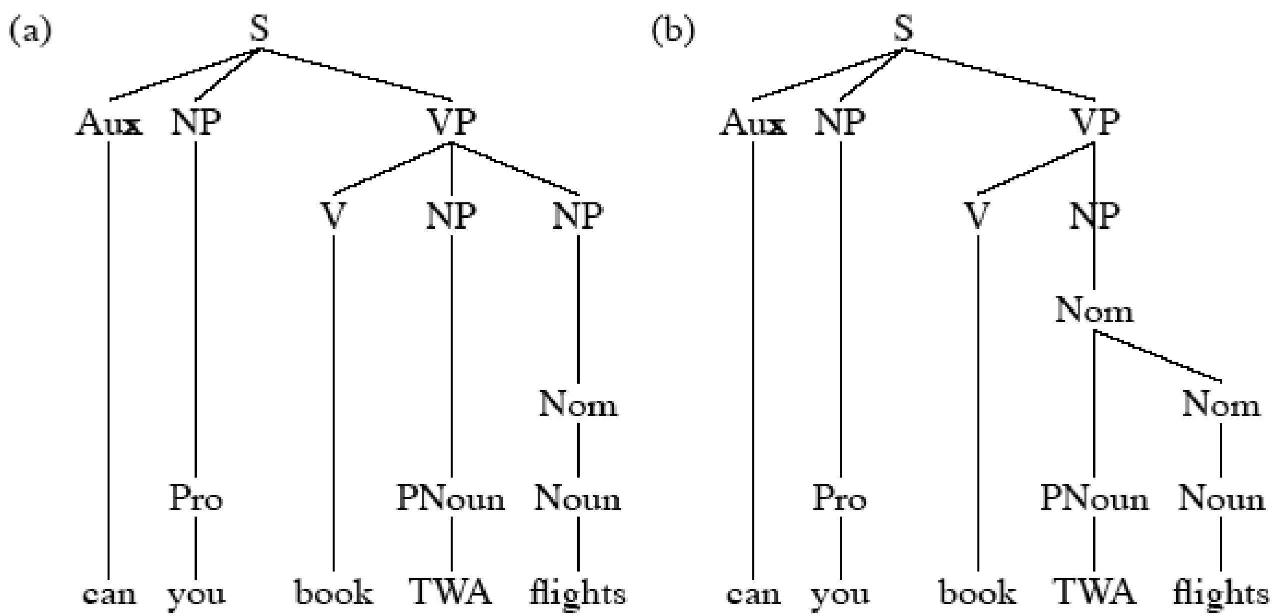
  VP -> Verb NP .40

  VP -> Verb NP NP .05

  – Read this as P(Specific rule | LHS)

PCFG

S → *NP VP*	[.80]	*Det* → *that* [.05] | *the* [.80] | *a* [.15]	
S → *Aux NP VP*	[.15]	*Noun* → *book*	[.10]
S → *VP*	[.05]	*Noun* → *flights*	[.50]
NP → *Det Nom*	[.20]	*Noun* → *meal*	[.40]
NP → *Proper-Noun*	[.35]	*Verb* → *book*	[.30]
NP → *Nom*	[.05]	*Verb* → *include*	[.30]
NP → *Pronoun*	[.40]	*Verb* → *want*	[.40]
Nom → *Noun*	[.75]	*Aux* → *can*	[.40]
Nom → *Noun Nom*	[.20]	*Aux* → *does*	[.30]
Nom → *Proper-Noun Nom*	[.05]	*Aux* → *do*	[.30]
VP → *Verb*	[.55]	*Proper-Noun* → *TWA*	[.40]
VP → *Verb NP*	[.40]	*Proper-Noun* → *Denver*	[.40]
VP → *Verb NP NP*	[.05]	*Pronoun* → *you* [.40] | *I* [.60]	



Rules			P	Rules			P
S	→	Aux NP VP	.15	S	→	Aux NP VP	.15
NP	→	Pro	.40	NP	→	Pro	.40
VP	→	V NP NP	.05	VP	→	V NP	.40
NP	→	Nom	.05	NP	→	Nom	.05
NP	→	PNoun	.35	Nom	→	PNoun Nom	.05
Nom	→	Noun	.75	Nom	→	Noun	.75
Aux	→	Can	.40	Aux	→	Can	.40
NP	→	Pro	.40	NP	→	Pro	.40
Pro	→	you	.40	Pro	→	you	.40
Verb	→	book	.30	Verb	→	book	.30
PNoun	→	TWA	.40	Pnoun	→	TWA	.40
Noun	→	flights	.50	Noun	→	flights	.50

Probability Model (1)

- A derivation (tree) consists of the set of grammar rules that are in the tree

- The probability of a tree is just the product of the probabilities of the rules in the derivation.

Probability model

$$P(T,S) = \prod_{n \in T} p(r_n)$$

- P(T,S) = P(T)P(S|T) = P(T); since P(S|T

$$\begin{aligned} P(T_l) &= .15 * .40 * .05 * .05 * .35 * .75 * .40 * .40 * .40 \\ &\quad * .30 * .40 * .50 \\ &= 1.5 \times 10^{-6} \end{aligned}$$

$$\begin{aligned} P(T_r) &= .15 * .40 * .40 * .05 * .05 * .75 * .40 * .40 * .40 \\ &\quad * .30 * .40 * .50 \\ &= 1.7 \times 10^{-6} \end{aligned}$$

Probability Model (1.1)

- The probability of a word sequence P(S) is the probability of its tree in the unambiguous case.
- It's the sum of the probabilities of the trees in the ambiguous case.

Getting the Probabilities

- From an annotated database (a treebank)
  - So for example, to get the probability for a particular VP rule just count all the times the rule is used and divide by the number of VPs overall.

Probabilistic Grammar Assumptions

- We're assuming that there is a grammar to be used to parse with.
- We're assuming the existence of a large robust dictionary with parts of speech
- We're assuming the ability to parse (i.e. a parser)
- Given all that… we can parse probabilistically

Typical Approach

- Bottom-up (CYK) dynamic programming approach
- Assign probabilities to constituents as they are completed and placed in the table
- Use the max probability for each constituent going up

What's that last bullet mean?

- Say we're talking about a final part of a parse
  - $S\text{->}_0NP_iVP_j$

The probability of the S is…

P(S->NP VP)*P(NP)*P(VP)

The green stuff is already known. We're doing bottom-up parsing

Modern parsers: lexicalized PCFG

- Modern CFG-based parsers use lexicalized PCFGs
  - Collins parser (Bikel version of this in Java)
  - Charniak parser
  - Stanford parser
- Also recent probabilistic versions of
  - HPSG parser
  - LFG parser

Summary

- Context-Free Grammars
- Parsing
  - Top Down, Bottom Up Metaphors
  - Dynamic Programming Parsers: CKY. Earley
- Disambiguation:
  - PCFG
  - Probabilistic Augmentations to Parsers
  - Treebanks