ITCS 4111/5111: Introduction to NLP

Syntax and Grammars

Syntactic Parsing

Razvan C. Bunescu

Department of Computer Science @ CCI

rbunescu@uncc.edu

Syntax and Grammars

- **Syntax** [Greek *syntaxis*] ≡ the way words are arranged together:
  - Not all combinations of words are *well formed*.
- Syntactic constraints can be captured using:
  - N-gram models.
  - Part-of-speech categories.
  - **Formal Grammars**:
    - Regular Grammars.
    - Context Free Grammars.
    - Dependency Grammars.
    - Categorial Grammars.

The **dog** that ***my cat*** chased yesterday **barked** at the moon.

Syntax and Constituency

- Groups of words behave as a single unit or phrase, called a **constituent**.
- Constituents for coherent classes that behave in similar ways:
  - **Internal structure**:
    - a common structure for all constituents in a class (use CFGs):
      - the white whale, the yellow fever, the scarlet letter, …
      - the elephant in the room, the man on the moon, …
  - **External behavior**:
    - similar behavior with respect to other language units:
      - three parties from Brooklyn *arrive* …
      - a high-class spot such as Mindy's *attracts* …
      - they *sit* …

Constituency and Context Free Grammars

- **Context Free Grammar (CFG)** ≡ a formal model commonly used to describe constituent structure:
  - A sentence as a hierarchy of constituents [Wilhelm Wundt, 1900].
  - Formalizations:
    - **Phrase Structure Grammars** [Chomsky, 1956].
    - **Backus-Naur Form** [Backus, 1959].

- A **CFG** is-a **generative grammar** is-a **formal grammar**:
  - Panini (4th century BC): the earliest known grammar of Sanskrit.
  - Chomsky (1950s): first formalized generative grammars.

Generative Grammars

- A **grammar** is tuple $G = (\Sigma, N, P, S)$:
  - A finite set Σ of **terminal symbols**.
    - the words of a natural language.
    - the tokens of a programming language.
  - A finite set N of **nonterminal symbols**, disjoint from Σ.
    - the constituent classes in a NL (noun phrase, verb phrase, sentence).
    - expressions, statement, type declarations in a PL.
  - A finite set P of **production rules**.
    - $P : (\Sigma \cup N)^* \, N \, (\Sigma \cup N)^* \rightarrow (\Sigma \cup N)^*$
  - A distinguished **start symbol** $S \in N$.

Generative Grammars

- The language L associated with a formal grammar G is the set of strings from Σ^* that can be generated as follows:
  - start with the start symbol S;
  - apply the production rules in P until no more nonterminal symbols are present.
- Example:
  - $\Sigma = \{a,b,c\}$, $N=\{S,B\}$
  - P consists of the following production rules:
    1. $S \rightarrow aBSc$
    2. $S \rightarrow abc$
    3. $Ba \rightarrow aB$
    4. $Bb \rightarrow bb$

Generative Grammars

- Production rules:
    1. $S \rightarrow aBSc$
    2. $S \rightarrow abc$
    3. $Ba \rightarrow aB$
    4. $Bb \rightarrow bb$
- **Derivations** of strings in the language L(G):
    - $S \Rightarrow_2 abc$
    - $S \Rightarrow_1 aBSc \Rightarrow_2 aBabcc \Rightarrow_3 aaBbcc \Rightarrow_4 aabbcc$
    - $S \Rightarrow \ldots \Rightarrow aaabbbccc$
- $L(G) = \{a^n b^n c^n \mid n > 0\}$

Generative Grammars

- A **derivation** is a repeated application of rules, starting with the start symbol and ending with a **sentence**:
  - A string of symbols in a derivation is a *sentential form*.
  - A *sentence* is a sentential form that has only terminal symbols.

- A **parse tree** is a hierarchical representation of a derivation:
  - The root and intermediate nodes are nonterminals.
  - The leaf nodes are terminals.
  - For each rule used in a derivation step:
    - the LHS is a parent node.
    - the symbols in the RHS are children nodes (from left to right).

Chomsky Hierarchy (1956)

- Type 0 grammars (unrestricted grammars)
  - Includes all formal grammars.
- Type 1 grammars (context-sensitive grammars).
  - Rules restricted to: $\alpha A \beta \rightarrow \alpha \gamma \beta$, where A is a non-terminal, and α, β, γ strings of terminals and non-terminals.
- **Type 2** grammars (**context-free grammars**).
  - Rules restricted to $A \rightarrow \gamma$, where A is a non-terminal, and γ a string of terminals and non-terminals
- **Type 3** grammars (**regular grammars**).
  - Rules restricted to $A \rightarrow \gamma$, where A is a non-terminal, and γ:
    - the empty string, or a single terminal symbol followed optionally by a non-terminal symbol.

Context Free Grammars (Type 2)

- Example:
  - $\Sigma = \{a,b\}$, $N=\{S\}$
  - P consists of the following production rules:
    1. $S \rightarrow aSb$
    2. $S \rightarrow \varepsilon$
  - L(G) = ?

CFGs provide the formal **syntax specification** of most programming languages.

$$S \rightarrow aSb \rightarrow aaSbb \rightarrow aaaSbbb \rightarrow aaabbb$$

Regular Grammars (Type 3)

- Example:
  - $\Sigma = \{a,b,c\}$, $N=\{S,A,B\}$
  - P consists of the following production rules:
    1. $S \rightarrow aS$
    2. $S \rightarrow cB$
    3. $B \rightarrow bB$
    4. $B \rightarrow \varepsilon$
  - $L(G) = ?$

Regular Grammars/Expressions provide the formal **lexical specification** of most programming languages.

A Simple CFG for English

- **Lexicon** = the rules that generalize the terminal symbols:
  - *token* → *lexeme* // in programming languages.
  - *part-of-speech* → *word* // in natural languages.

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

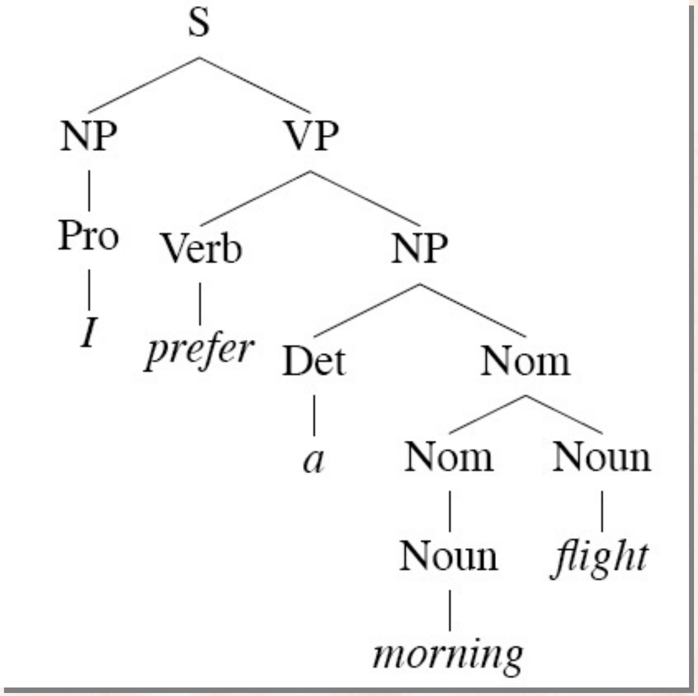
A Simple CFG for English

Grammar Rules			Examples
S	→	*NP VP*	I + want a morning flight
NP	→	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	→	*Nominal Noun*	morning + flight
	|	*Noun*	flights
VP	→	*Verb*	do
	|	*Verb NP*	want + a flight
	|	*Verb NP PP*	leave + Boston + in the morning
	|	*Verb PP*	leaving + on Thursday
PP	→	*Preposition NP*	from + Los Angeles

A Simple CFG for English

- Leftmost Derivation & Parse Tree:

S => NP VP
=> Pro VP
=> *I* VP
=> *I* Verb NP
=> *I prefer* NP
=> *I prefer* Det Nom
=> *I prefer a* Nom
=> *I prefer a* Nom Noun
=> *I prefer a* Noun Noun
=> *I prefer a morning* Noun
=> *I prefer a morning flight*



Syntactic Parsing

- **Syntactic Parsing (Analysis)** = a computing problem:
  - Input:
    - a context free grammar.
    - a sequence of tokens.
  - Output:
    - *YES* if the input can be generated by the CFG.
      - The parse tree $\Rightarrow$ need **unambiguous** grammar.
    - *NO* if the input cannot be generated by the CFG.
      - Find all syntax errors; for each, produce an appropriate diagnostic message and recover quickly.

Some Grammar Rules for English

- Sentences:
  - and Clauses.
- Noun Phrases:
  - Agreement.
- Verb Phrases:
  - Subcategorization.

Sentence Types

- Declaratives: *A plane left.*

  S → NP VP

- Imperatives: *Show me the cheapest fare that has lunch.*

  S → VP

- Yes-No Questions: *Do any of these flights have stops?*

  S → Aux NP VP

- WH Questions:

  long distance dependency

  – *What airlines fly from Burbank to Denver?*

    S → WH-NP VP

  – *What flights do you have for Washington DC?*

    S → WH-NP Aux NP VP

Sentences and Clauses

- **Clauses** are modeled using the *S* nonterminal:
  - Sentences are clauses.
  - "They form a complete thought"
  - Can appear both on the LHS and RHS of a rule:
    - `S => NP VP`
    - `VP => Verb S`

[$_S$ [$_{NP}$ You] [$_{VP}$ [$_{VB}$ said] [$_S$ there were two flights to Denver]]]

sentential complement

Noun Phrases

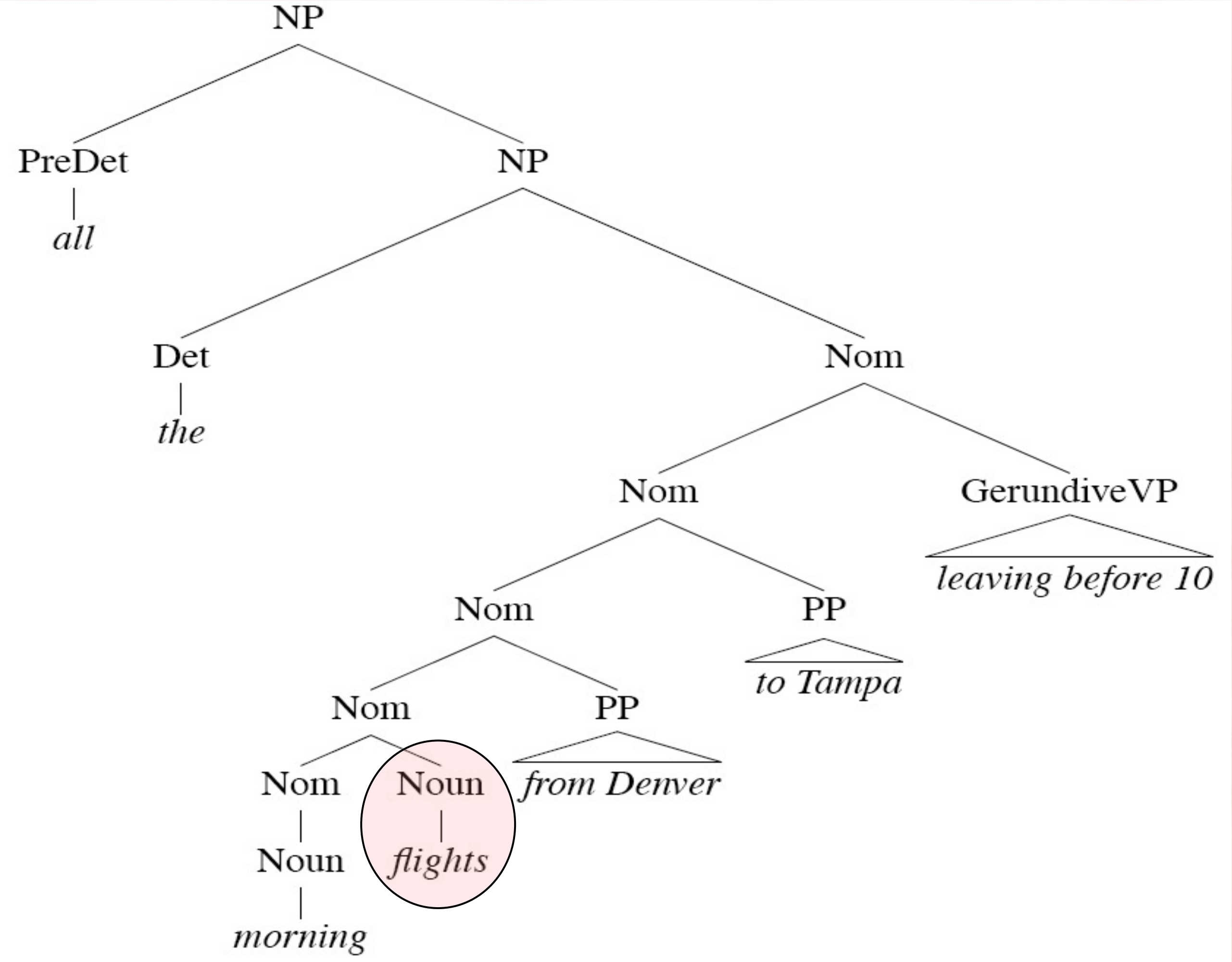
- "All the morning *flights* from Denver to Tampa leaving before 10":
  - Clearly this NP is about *flights.*
    - the **head** of the NP, i.e. its central noun.

- Context free rules:

```
NP => PreDet NP
NP => Det Nominal
Nominal => ?
```

the head, along with various modifiers that can appear before and after the head

Noun Phrases



Determiners

- Noun phrases can start with **determiners**:
  - Exceptions: plural nouns, mass nouns, …
- Determiners can be
  - Simple lexical items: *the, this, a, an*, etc.
    - *a* car, *the* car, *those* flights, *any* flights, *some* flights, …
  - Or simple possessives:
    - John's car
  - Or complex recursive versions of that
    - John's sister's husband's son's car

```
NP => Det Nom
Det => NP POS
```

Nominals: Premodifiers

- **Nominals** contain the head and any pre- and post- modifiers:
  - **Premodifiers**:
    - Quantifiers, cardinals, ordinals...
      - *Three* cars
    - Adjectives and Adjectival Phrases
      - *large* cars
    - Ordering constraints
      - *Three large* cars
      - ?large three cars

    `NP => [Det] [Card] [Ord] [Quant] [AP] Nom`

Nominals: Postmodifiers

- Three kinds of **postmodifiers**:
  - Prepositional phrases:
    - all flights *from Seattle*
  - Non-finite clauses:
    - any flights *arriving before noon*
  - Relative clauses:
    - a flight *that serve breakfast*
- Same general (recursive) rule to handle these
  - *Nominal* $\rightarrow$ *Nominal PP*
  - *Nominal* $\rightarrow$ *Nominal GerundVP*
  - *Nominal* $\rightarrow$ *Nominal RelClause*

Nominals: Agreement Constraints

- **Number Agreement**:
  - subject & verb
    - flights leave *flights leaves
    - do you have *does you have
  - determiner & head noun:
    - this flight *this flights
    - those flights *those flight
- **Case Agreement**:
  - nominative: I, she, he, they
  - accusative: me, her, him, them
- **Gender Agreement**:
  - le petit prince * la petite prince

Agreement Constraints

- NP rules so far are deficient:
  - *NP → Det Nominal*
    - Accepts, and assigns correct structures, to grammatical examples (*this flight*)
    - But it's also happy with incorrect examples (**these flight*).
  - The rule is said to **overgenerate**.
- VP rules are deficient too:
  - **subcategorization** constraints.

Verb Phrases

- "*flies* from Milwakee to Orlando in less than 4 hours"
  - The VP is about the action of *flying*.

  ⇒ *flies* is the **head** of the VP.

- Verb Phrase structure:
  - a head verb along with 0 or more following constituents called **complements**:
    - **arguments** (core complements).
    - **adjuncts** (modifiers).

VP	$\rightarrow$	$Verb$	disappear
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

Verb Subcategorization

- **Subcategorization** ≡ the tendency of heads to place restrictions on the types and number of arguments.
  - 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]$_{S}$
  - Want: I want [to fly from Milwakee to Orlando]$_{TO\text{-}VP}$

- Framenet: http://framenet.icsi.berkeley.edu/

Verb Subcategorization

- Right now, the various rules for VPs **overgenerate**:
  - *John sneezed the book
  - *I prefer United has a flight
  - *Give with a flight

"All grammars leak" [Sapir, 1921]

- We can **subcategorize** the verbs in a language according to the sets of VP rules that they participate in:
  - generalization of the traditional notion of transitive/intransitive.
  - Modern grammars may have 100s of such classes.
  - **Subcategorization Frame**: the possible sets of arguments for a given verb.

Agreement and Subcategorization

- Should these constraints be modeled through CFG rules?

SgS → SgNP SgVP
PlS → PlNp PlVP
SgNP → SgDet SgNom
PlNP → PlDet PlNom
PlVP → PlV NP
SgVP → SgV Np
…

agreement

- Similar approach for subcategorization ⇒ proliferation of rules.

Agreement and Subcategorization

- How to avoid bloated grammars in natural languages?
  - parameterize each non-terminal with **feature structures**.
  - use **unification** to enforce constraints.
  - more details in Ch. 15 "Features and Unification" in J&M.

- Similar to static semantic constraints in programming languages:
  - type compatibility rules (e.g. Java cannot assign float to integer)
  - use **attribute grammars** [Knuth, 1968], where a CFG is augmented to carry some semantic info on parse tree nodes.
  - approach (implicitly) used by compiler writers.

Agreement and Subcategorization

- CFGs appear to be just about what we need to account for a lot of basic syntactic structure in English:
  - It doesn’t scale all that well because the interaction among the various constraints explodes the number of rules in our grammar.
- There are simpler, more elegant solutions that take us out of the CFG framework (beyond its formal power):
  - Lexical Functional Grammar (LFG).
  - Head-driven Phrase Structure Grammar (HPSG),
  - Construction Grammar,
  - Tree Adjoining Grammar (TAG),
  - …

Treebanks

- **Treebank** = a corpus in which every sentence is syntactically annotated with a parse tree.

- Generally created in two steps:
  1) First parse the collection with an automatic parser.
  2) Then human annotators correct each parse as necessary.
     - Requires detailed annotation guidelines:
       – a POS tagset, a grammar.
       – Instructions for how to deal with particular grammatical constructions.
         » ftp://ftp.cis.upenn.edu/pub/treebank/doc/manual/

The Penn Treebank

- Brown, Switchboard, ATIS, and Wall Street Journal.
  - also Arabic and Chinese.

```
( (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) ))
    (. .) ))
```

traces and co-indexing for long distance dependencies

tags indicate grammatical function: surface subject, logical topic, cleft, …

Treebank Grammars

- Treebanks implicitly define a grammar for the language covered in the treebank.
  - Simply take the local rules that make up the sub-trees.
  - Not complete, but if you have decent size corpus, you'll have a grammar with decent coverage.

```
((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)

$\Rightarrow$

Treebank Grammars

Grammar	Lexicon
$S \rightarrow NP\ VP\ .$	$PRP \rightarrow we \mid he$
$S \rightarrow NP\ VP$	$DT \rightarrow the \mid that \mid those$
$S \rightarrow$ " S " , NP VP .	$JJ \rightarrow cold \mid empty \mid full$
$S \rightarrow$ -NONE-	$NN \rightarrow sky \mid fire \mid light \mid flight \mid tomorrow$
$NP \rightarrow DT\ NN$	$NNS \rightarrow assets$
$NP \rightarrow DT\ NNS$	$CC \rightarrow and$
$NP \rightarrow NN\ CC\ NN$	$IN \rightarrow of \mid at \mid until \mid on$
$NP \rightarrow CD\ RB$	$CD \rightarrow eleven$
$NP \rightarrow DT\ JJ\ ,\ JJ\ NN$	$RB \rightarrow$ a.m.
$NP \rightarrow PRP$	$VB \rightarrow arrive \mid have \mid wait$
$NP \rightarrow$ -NONE-	$VBD \rightarrow was \mid said$
$VP \rightarrow MD\ VP$	$VBP \rightarrow have$
$VP \rightarrow VBD\ ADJP$	$VBN \rightarrow collected$
$VP \rightarrow VBD\ S$	$MD \rightarrow should \mid would$
$VP \rightarrow VBN\ PP$	$TO \rightarrow to$
$VP \rightarrow VB\ S$	
$VP \rightarrow VB\ SBAR$	
$VP \rightarrow VBP\ VP$	
$VP \rightarrow VBN\ PP$	
$VP \rightarrow TO\ VP$	
$SBAR \rightarrow IN\ S$	
$ADJP \rightarrow JJ\ PP$	
$PP \rightarrow IN\ NP$	

Treebank Grammars

- Treebank grammars tend to be very flat:
  - They tend to avoid recursion, to ease the annotators burden.
  - The Penn Treebank has 4500 different rules for VPs.

$$
\begin{array}{l}
\mathrm{VP} \rightarrow \mathrm{VBD}\ \mathrm{PP} \\
\mathrm{VP} \rightarrow \mathrm{VBD}\ \mathrm{PP}\ \mathrm{PP} \\
\mathrm{VP} \rightarrow \mathrm{VBD}\ \mathrm{PP}\ \mathrm{PP}\ \mathrm{PP} \\
\mathrm{VP} \rightarrow \mathrm{VBD}\ \mathrm{PP}\ \mathrm{PP}\ \mathrm{PP}\ \mathrm{PP}
\end{array}
$$

  - Even longer: VP → VBP PP PP PP PP PP ADVP PP

    "This mostly happens because we *go from football in the fall to lifting in the winter to football again in the spring*".

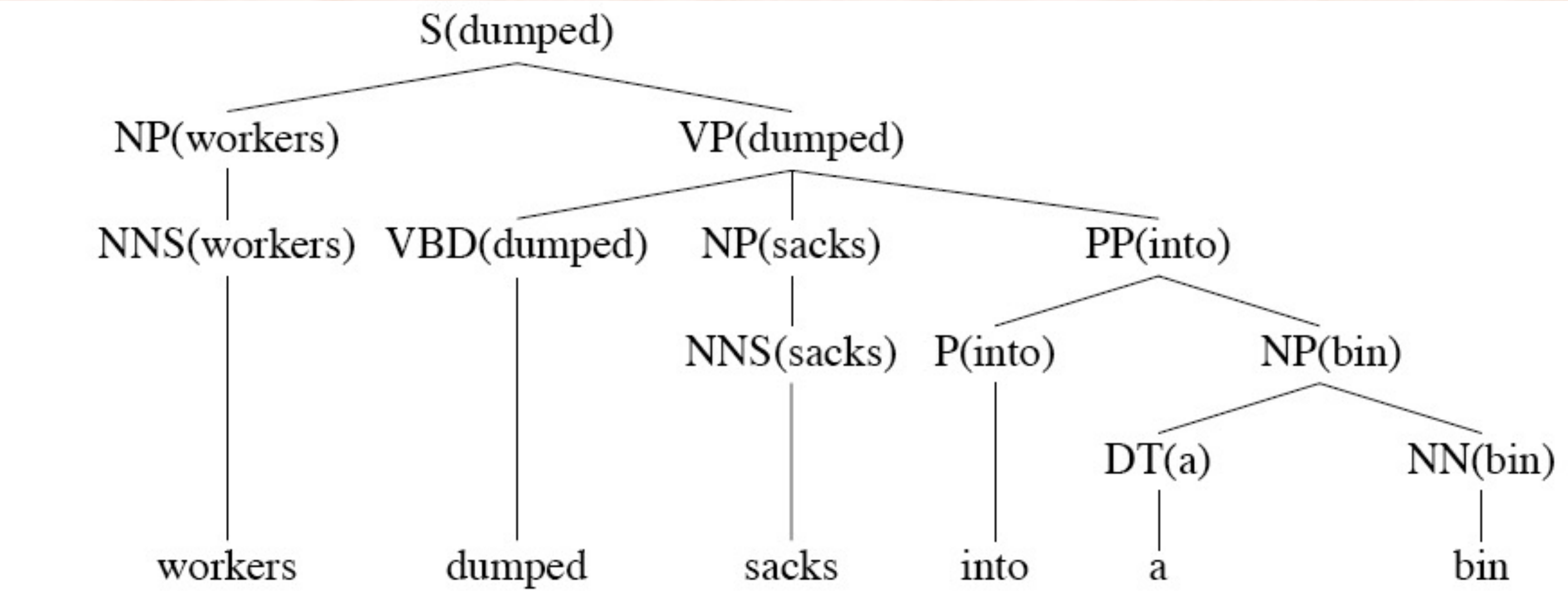
- Typically "normalized" to make them amenable to porbabilistic parsing algorithms.

Heads in Trees

- Each syntactic constituent can be associated with a lexical **head**:
  - nouns for NPs, verbs for VPs and clauses, … [Bloomfield, 1914].
  - central for HPSGs, corpus linguistics, statistical parsing with lexicalized grammars.
    - statistical parsers are trained on treebanks ⇒ need to be able to automatically find heads in trees.

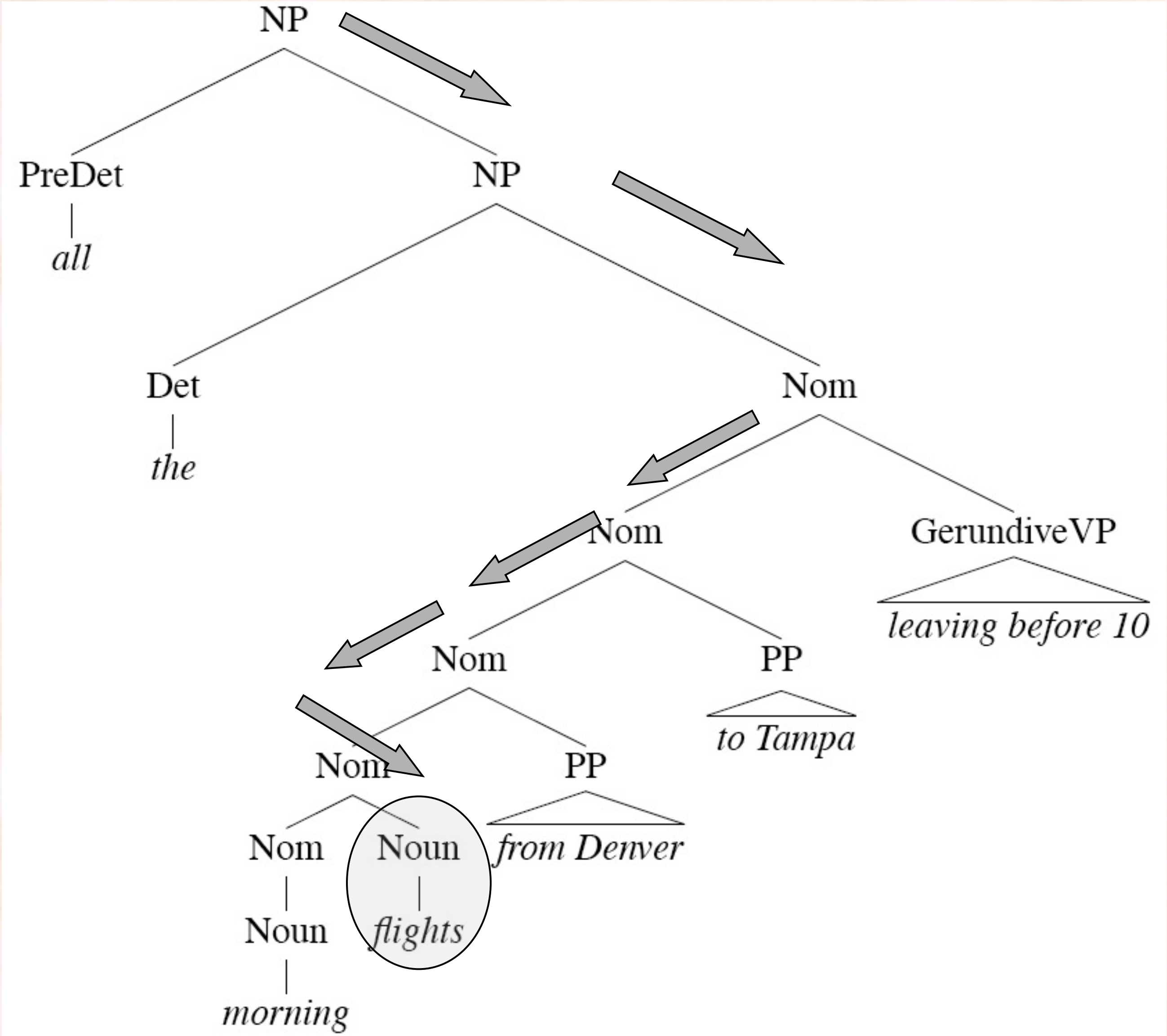
- Finding heads:
  - visualize this task by annotating the nodes of a parse tree with the heads of each corresponding node.

Head Decorated Parse Tree



Head Finding

- Use a simple set of hand-written rules:
  - For Penn Treebank [Magerman, 1995; Collins, 1999].
  - Rules for NPs:
    - If the last word is tagged POS, return last word.
    - Else search from R to L for the first child which is an NN, NNP, NNPS, NX, POS, or JJR.
    - Else search from L to R for the first child which is an NP.
    - Else search from R to L for the first child which is a $, ADJP or PRN.
    - Else search from R to L for the first child which is a CD.
    - Else search from R to L for the first child which is a JJ, JJS, RB or QP.
    - Else return the last word.


NP
PreDet
all
NP
Det
the
Nom
Nom
GerundiveVP
leaving before 10
Nom
PP
to Tampa
Nom
PP
from Denver
Nom
Noun
Noun
flights
morning

Head Percolation Table

Parent	Direction	Priority List
ADJP	Left	NNS QP NN $ ADVP JJ VBN VBG ADJP JJR NP JJS DT FW RBR RBS SBAR RB
ADVP	Right	RB RBR RBS FW ADVP TO CD JJR JJ IN NP JJS NN
PRN	Left	
PRT	Right	RP
QP	Left	$ IN NNS NN JJ RB DT CD NCD QP JJR JJS
S	Left	TO IN VP S SBAR ADJP UCP NP
SBAR	Left	WHNP WHPP WHADVP WHADJP IN DT S SQ SINV SBAR FRAG
VP	Left	TO VBD VBN MD VBZ VB VBG VBP VP ADJP NN NNS NP

Treebank Searching: Tgrep2

Link	Explanation
`A < B`	A is the parent of (immediately dominates) B.
`A > B`	A is the child of B.
`A <N B`	B is the Nth child of A (the first child is `<1`).
`A >N B`	A is the Nth child of B (the first child is `>1`).
`A <, B`	Synonymous with `A <1 B`.
`A >, B`	Synonymous with `A >1 B`.
`A <-N B`	B is the Nth-to-last child of A (the last child is `<-1`).
`A >-N B`	A is the Nth-to-last child of B (the last child is `>-1`).
`A <- B`	B is the last child of A (synonymous with `A <-1 B`).
`A >- B`	A is the last child of B (synonymous with `A >-1 B`).
`A <‘ B`	B is the last child of A (also synonymous with `A <-1 B`).
`A >‘ B`	A is the last child of B (also synonymous with `A >-1 B`).
`A <: B`	B is the only child of A.
`A >: B`	A is the only child of B.

Treebank Searching: Tgrep2

`A << B`	A dominates B (A is an ancestor of B).
`A >> B`	A is dominated by B (A is a descendant of B).
`A <<, B`	B is the leftmost descendant of A.
`A >>, B`	A is the leftmost descendant of B.
`A <<‘ B`	B is the rightmost descendant of A.
`A >>‘ B`	A is the rightmost descendant of B.
`A <<: B`	There is a single path of descent from A and B is on it.
`A >>: B`	There is a single path of descent from B and A is on it.
`A . B`	A immediately precedes B.
`A , B`	A immediately follows B.
`A .. B`	A precedes B.
`A ,, B`	A follows B.
`A $ B`	A is a sister of B (and A $\neq$ B).
`A $. B`	A is a sister of and immediately precedes B.
`A $, B`	A is a sister of and immediately follows B.
`A $.. B`	A is a sister of and precedes B.
`A $,, B`	A is a sister of and follows B.

Is English a Regular Language?

1) Certain syntactic structures cannot be described with RGs:
   - Recursive center-embedding rules:
     - $A \rightarrow_* \alpha\ A\ \beta$
       - The luggage arrived.
       - The luggage that the passengers checked arrived.
       - The luggage that the passengers that the storm delayed checked arrived.
     - Matching parantheses in programming languages.

2) Even when expressive enough, regular grammars do not produce structures of immediate use in semantic analysis.
   - Syntax-directed semantic analysis / translation.

Dependency Grammars

- In CFG-style **phrase-structure grammars** the main focus is on **constituents**.
- In **dependency grammars**, the focus is on **binary relations among the words** in an utterance.
  - In a parse tree:
    - The nodes stand for the words in an utterance
    - The links between the nodes represent dependency relations between pairs of words.
      - Relations may be typed (labeled), or not.

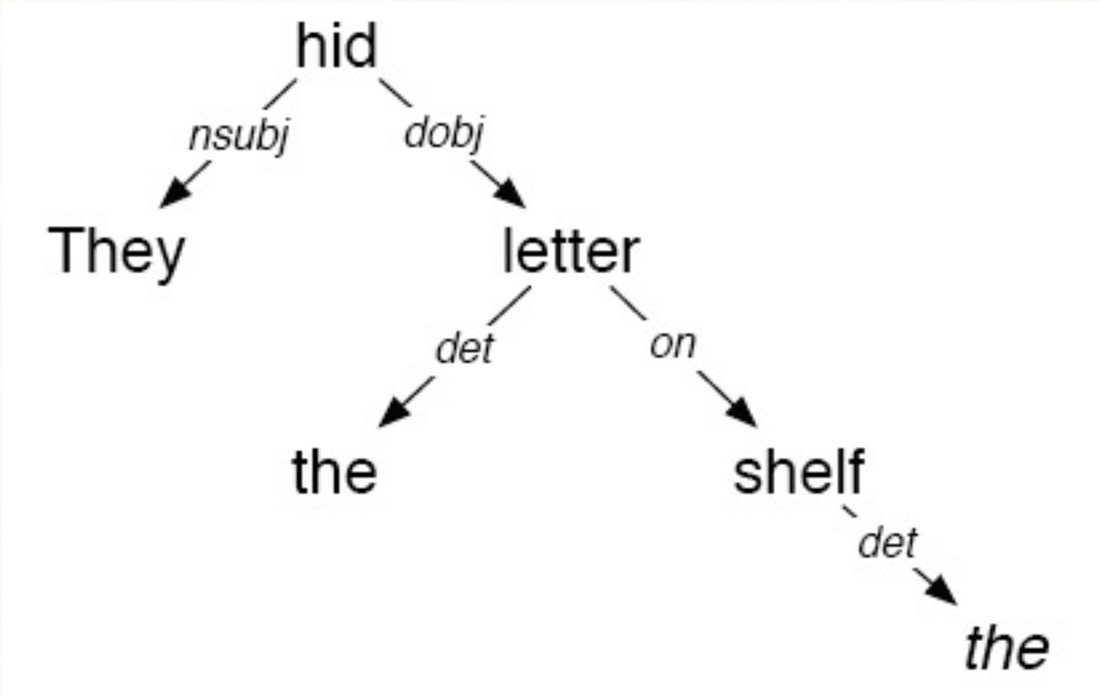
Dependency Relations

[de Marneffe et al., 2006]

Argument Dependencies	Description
nsubj	nominal subject
csubj	clausal subject
dobj	direct object
iobj	indirect object
pobj	object of preposition
Modifier Dependencies	**Description**
tmod	temporal modifier
appos	appositional modifier
det	determiner
prep	prepositional modifier

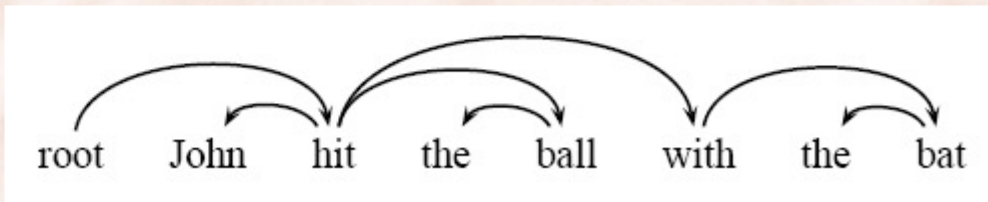
Dependency Parse

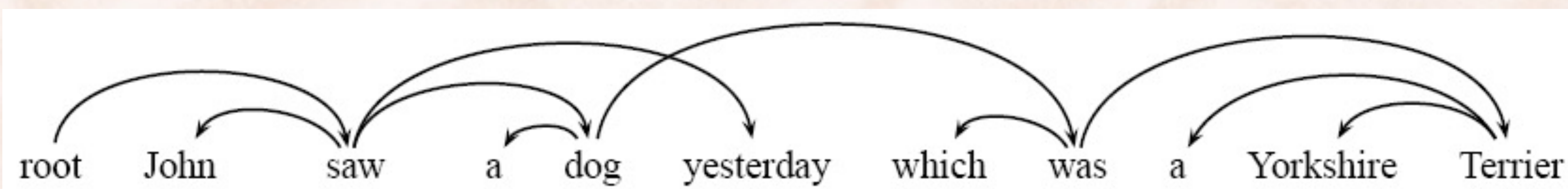
[de Marneffe et al., 2006]

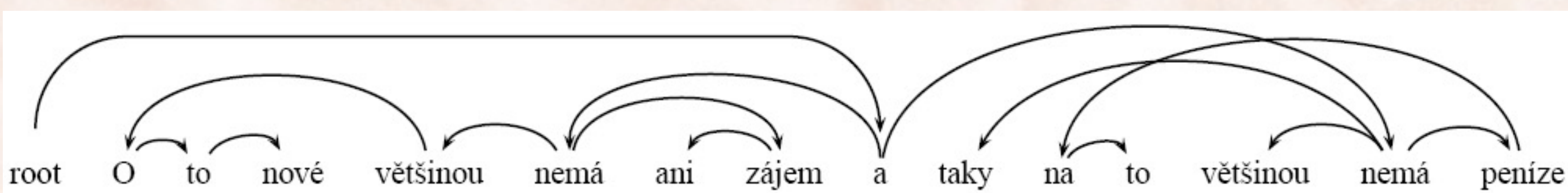


They hid the letter on the shelf

Projective vs. Non-Projective Dependencies







He is mostly not even interested in the new things and in most cases, he has no money for it either.

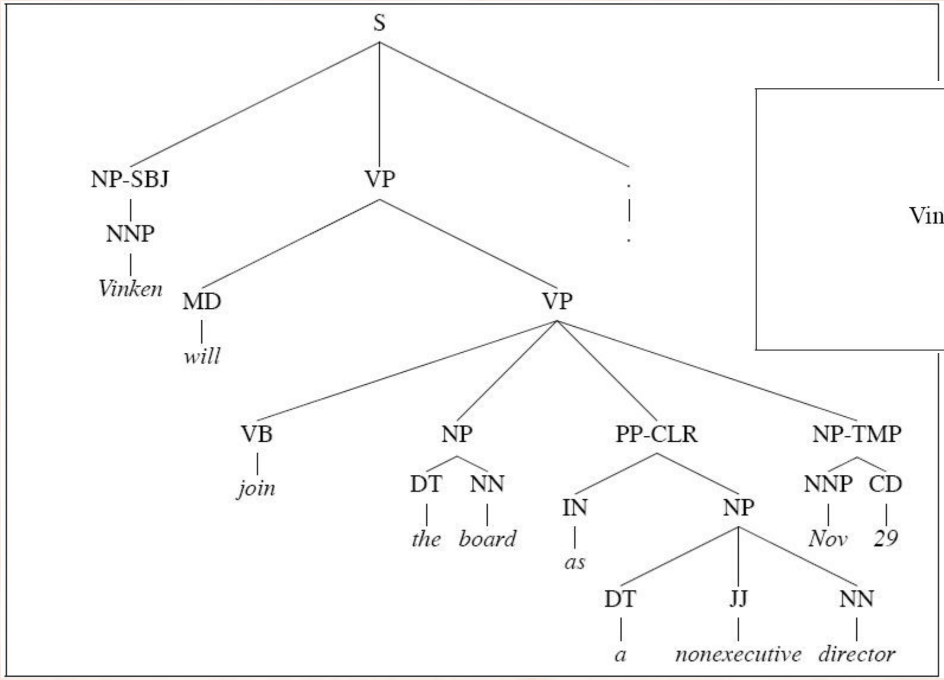
[McDonald et al., EMNLP'05]

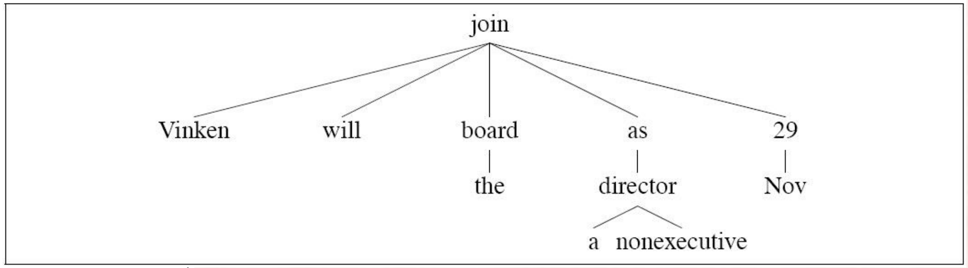
Dependency Parsing

- **Dependency** Parsing vs. **Phrase-Structure** Parsing:
  - Ability to handle languages with relatively free word order.
    - In Czech, an object may occur before or after a location adverbial
      - In a CFG, need separate rule.
      - In a DG, need only one link type.
  - Parsing is much faster.
  - CFGs are often used to extract the same syntactic relations anyway.
    - dependency trees extracted automatically from constituent trees.
- Implementations:
  - Link Grammar (Sleator and Temperley, 1993), Constraint Grammar (Karlsson et al.), MINIPAR (Lin, 2003), Stanford Parser (de Marneffe et al., 2006).

From Constituent Trees to Dependency Trees

1) Mark the head child of each node.
2) For every parent node, create a dependency link between the head of a non-head child node and the head of the head-child.





3) Type dependencies using hand written patterns [de Marneffe et al., 2006].

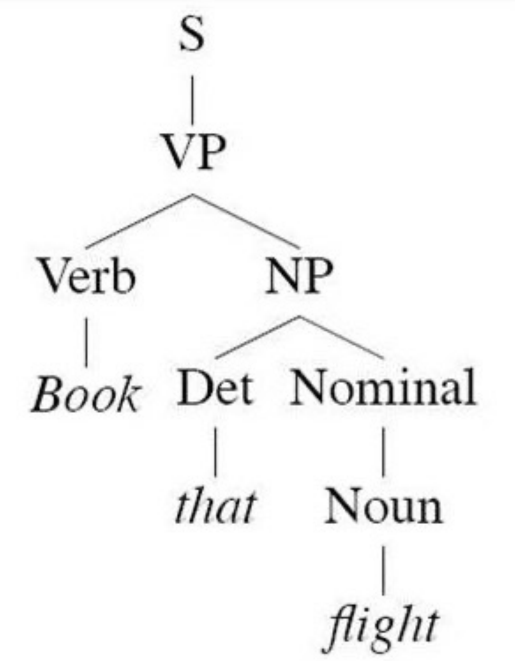
Syntactic Parsing

Syntactic Parsing

- **Syntactic Parsing** = assigning a syntactic structure to a sentence.
  - For CFGs: assigning a *phrase-tructure tree* to a sentence.

Grammar	Lexicon
S → NP VP	*Det → that | this | a*
S → Aux NP VP	*Noun → book | flight | meal | money*
S → VP	*Verb → book | include | prefer*
NP → Pronoun	*Pronoun → I | she | me*
NP → Proper-Noun	*Proper-Noun → Houston | NWA*
NP → Det Nominal	*Aux → does*
Nominal → Noun	*Preposition → from | to | on | near | through*
Nominal → Nominal Noun	
Nominal → Nominal PP	
VP → Verb	
VP → Verb NP	
VP → Verb NP PP	
VP → Verb PP	
VP → VP PP	
PP → Preposition NP	

Book that flight.



Syntactic Parsing as Search

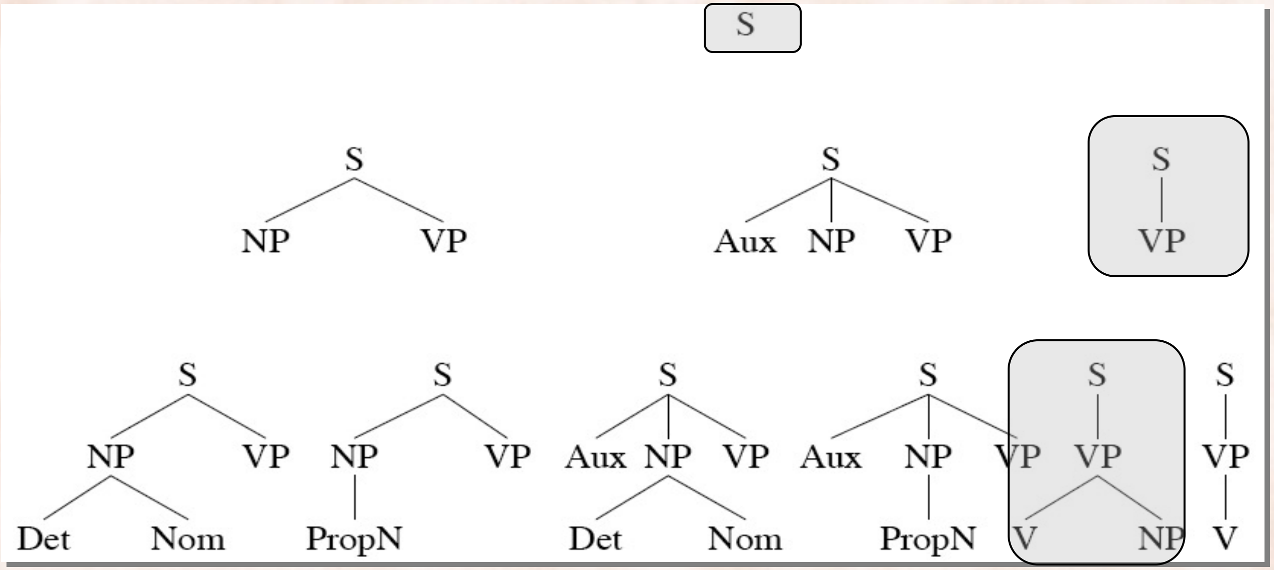
- Parsing ≡ search through the space of all possible parse trees such that:
  1. The leaves of the final parse tree coincide with the words in the input sentence.
  2. The root of the parse tree is the symbol S, i.e. complete parse tree.

⇒ 2 search strategies:

- **Top-Down** parsing (goal-directed search).
- **Bottom-Up** parsing (data-directed search).

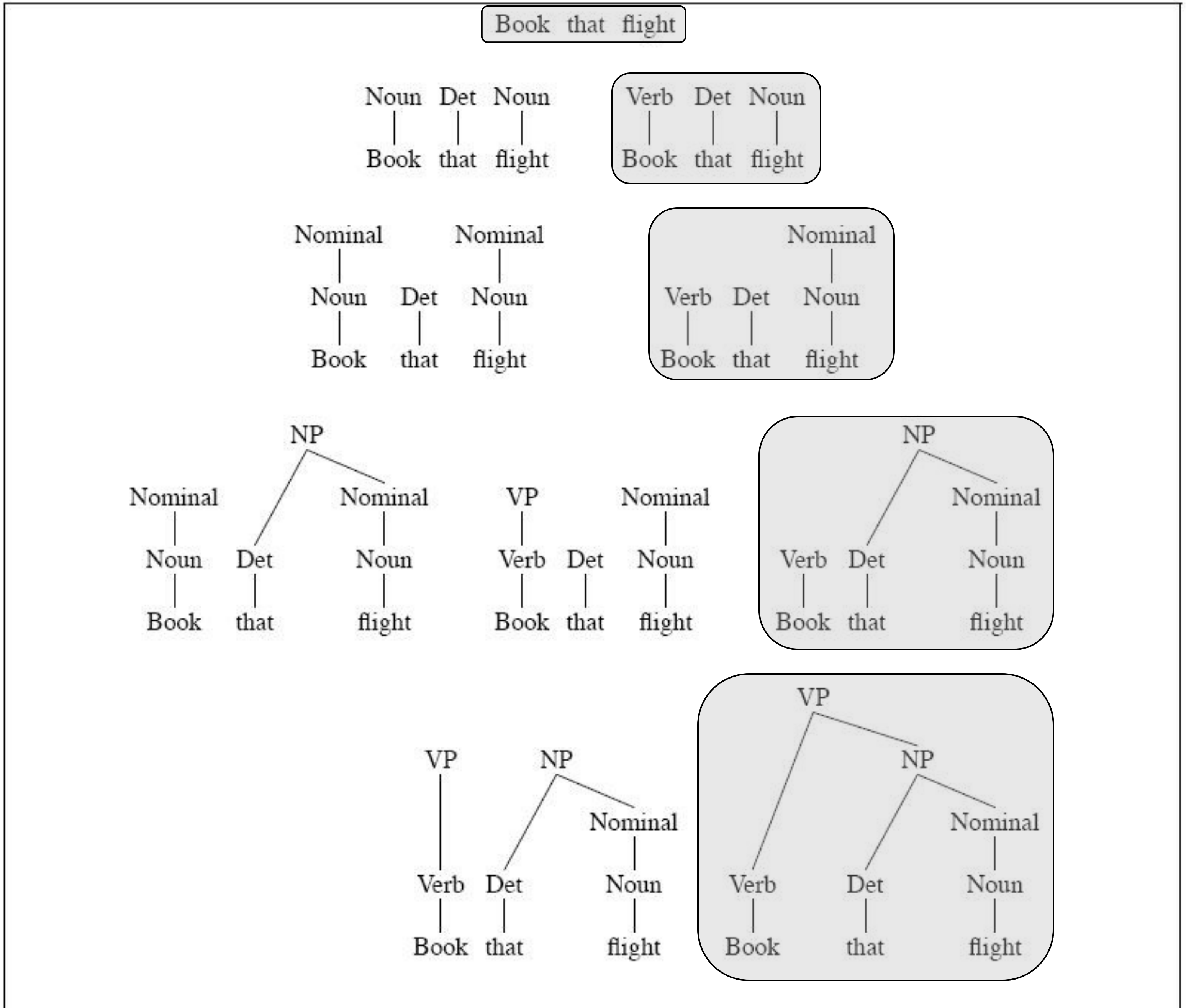
Top-Down Parsing

- Build the parse tree from the root *S* down to the leaves:
  - Expand tree nodes *N* by using CFG rules $N \rightarrow N_1 \ldots N_k$.
  - Grow trees downward until reaching the POS categories at the bottom of the tree.
  - Reject trees that do not match all the words in the input.



Bottom-Up Parsing

- Build the parse tree from the leaf words up to the root S:
  - Find root nodes $N_1 \dots N_k$ in the current forest such that they match a CFG rule $N \rightarrow N_1 \dots N_k$.
  - Reject sub-trees that cannot lead to the start symbol S.


Book that flight
Noun Det Noun
Book that flight
Verb Det Noun
Book that flight
Nominal Nominal
Noun Det Noun
Book that flight
Nominal
Verb Det Noun
Book that flight
NP
Nominal Nominal
Noun Det Noun
Book that flight
VP Nominal
Verb Det Noun
Book that flight
NP
Nominal
Verb Det Noun
Book that flight
VP NP
Nominal
Verb Det Noun
Book that flight
VP
NP
Nominal
Verb Det Noun
Book that flight

Top-Down vs. Bottom-Up

- Top-down:
  - Only searches for trees that are complete (i.e. S's)
  - But also suggests trees that are not consistent with any of the words.
- Bottom-up:
  - Only forms trees consistent with the words.
  - But also suggests trees that make no sense globally.
- How expensive is the entire search process?

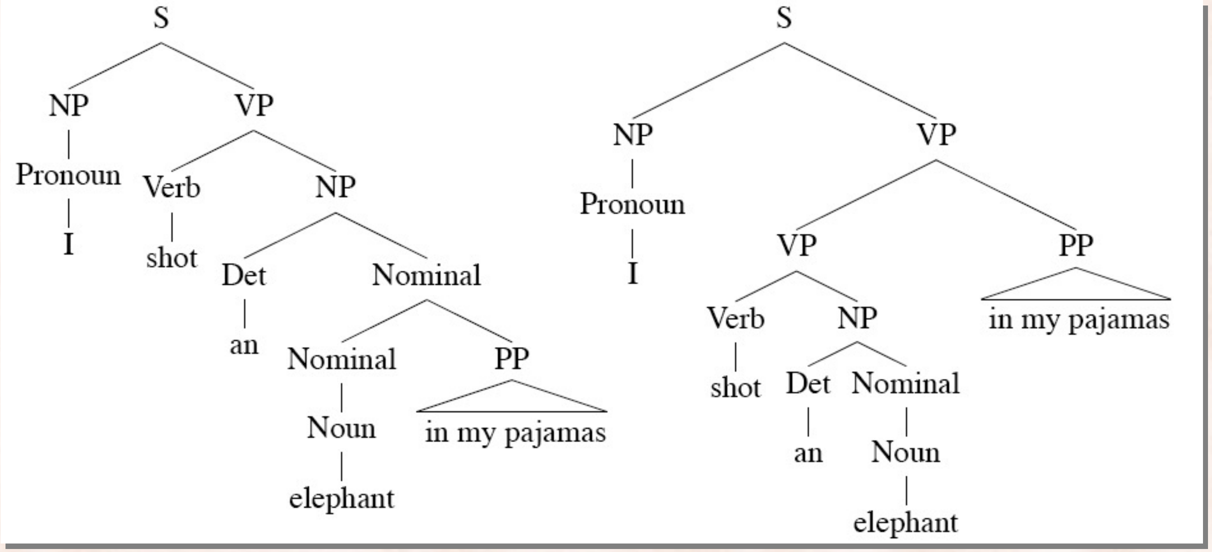
Syntactic Parsing as Search

- 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.

- Backtracking (naïve implementation of parsing):
  - Expand the search space incrementally, choose a state to expand in the search space (depth-first, breadth-first, or other strategies).
  - If strategy arrives at an inconsistent tree, backtrack to an unexplored search on the agenda.
  - Doomed because of *large search space* and *redundant work* due to shared subproblems.

Large Search Space

- Global Ambiguity:
  - coordination: *old men and women*
  - attachment: *we saw the Eiffel Tower flying to Paris*



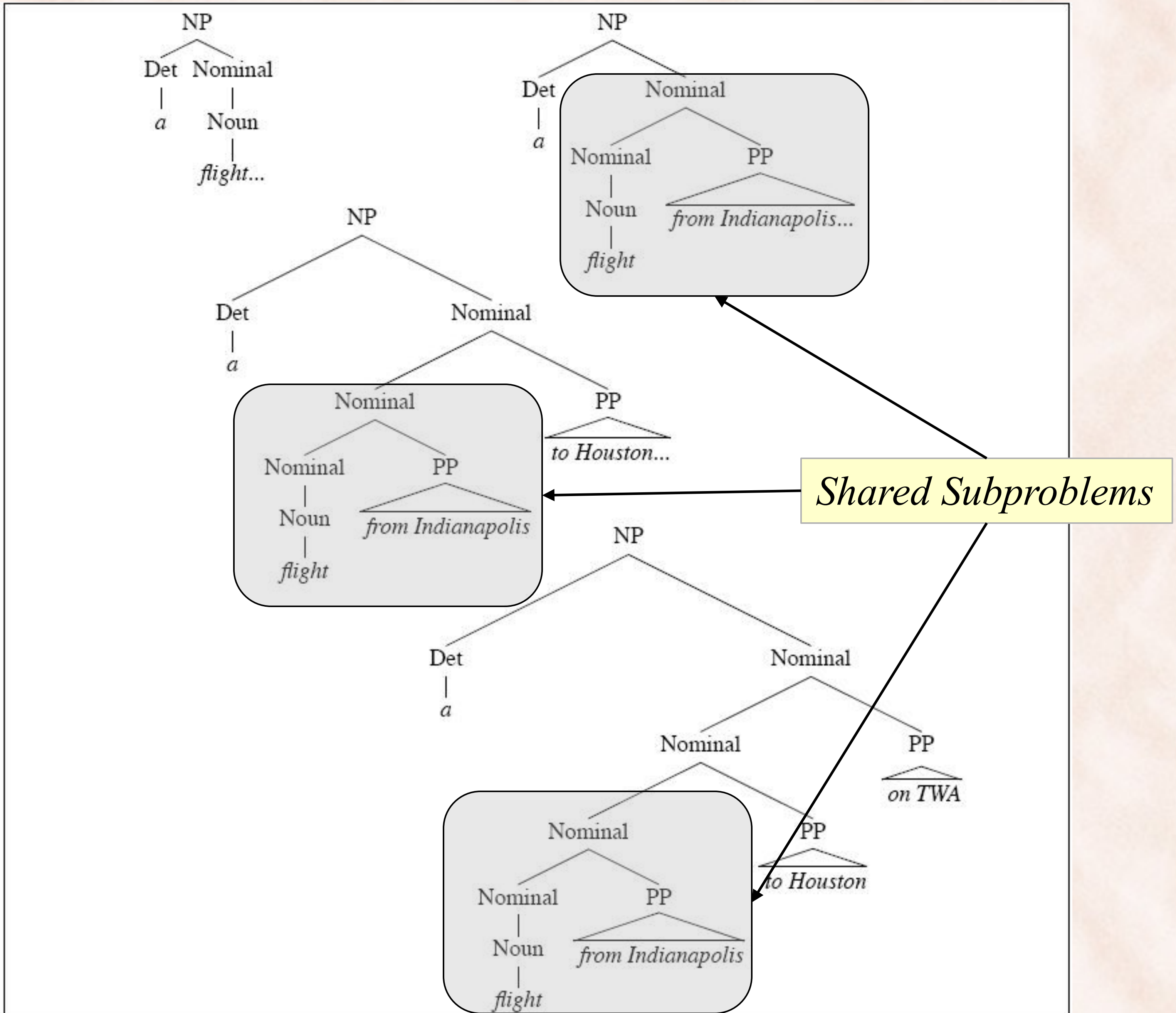
- Local Ambiguity

Shared Subproblems

- Parse the sentence:

  "a flight from Indianapolis to Houston on TWA"

- Use backtracking with a top-down, depth-first, left-to-right strategy:
  - Assume a top-down parse making choices among the various Nominal rules, in particular, between these two:
    - Nominal $\rightarrow$ Noun
    - Nominal $\rightarrow$ Nominal PP
  - Statically choosing the rules in this order leads to the following bad results, in which every part of the final tree is derived more than once:



NP
Det
Nominal
a
Noun
flight...
NP
Det
Nominal
a
Nominal
PP
Noun
from Indianapolis...
flight
NP
Det
Nominal
a
Nominal
PP
to Houston...
Nominal
PP
Noun
from Indianapolis
flight
Shared Subproblems
NP
Det
Nominal
a
Nominal
PP
on TWA
Nominal
PP
to Houston
Nominal
PP
Noun
from Indianapolis
flight

Syntactic Parsing using Dynamic Programming

- Shared subproblems ⇒ **dynamic programming** could help.
- Dynamic Programming:
  - **CKY** algorithm (bottom-up search).
    - Need to transform the CFG into Chomsky Normal Form (CNF).
    - Any CFG can be transformed into CNF automatically.
  - **Earley** algorithm (top-down search).
    - does not require a normalized grammar.
    - a single left-to-right pass that fills an array/chart of size $n + 1$.
    - more complex than CKY.
  - **Chart parsing**:
    - more general, retain completed phrases in a chart, can combine top-down and bottom-up search.

CKY Parsing: Chomsky Normal Form

- All rules should be of one of two forms:

  $A \rightarrow B\ C$ or $A \rightarrow w$

- CNF conversion procedure:
  1. Convert terminals to dummy non-terminals:

     INF-VP → *to* VP ⇔ INF-VP → TO VP and TO → *to*
  2. Convert unit productions

     Nominal → Noun
     Noun → *book* | *flight*
     ⇔ Nominal → *book* | *flight*
  3. Make all rules binary by adding new non-terminals:

     VP → Verb NP PP ⇔ VP → VX PP
     VX → Verb NP

L_1 Grammar

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

$\mathscr{L}_1$ **Grammar**	$\mathscr{L}_1$ **in CNF**
$S \rightarrow NP\ VP$	$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$	$S \rightarrow X1\ VP$
	$X1 \rightarrow Aux\ NP$
$S \rightarrow VP$	$S \rightarrow book \mid include \mid prefer$
	$S \rightarrow Verb\ NP$
	$S \rightarrow X2\ PP$
	$S \rightarrow Verb\ PP$
	$S \rightarrow VP\ PP$
$NP \rightarrow Pronoun$	$NP \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$NP \rightarrow TWA \mid Houston$
$NP \rightarrow Det\ Nominal$	$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow Noun$	$Nominal \rightarrow book \mid flight \mid meal \mid money$
$Nominal \rightarrow Nominal\ Noun$	$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$	$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow Verb$	$VP \rightarrow book \mid include \mid prefer$
$VP \rightarrow Verb\ NP$	$VP \rightarrow Verb\ NP$
$VP \rightarrow Verb\ NP\ PP$	$VP \rightarrow X2\ PP$
	$X2 \rightarrow Verb\ NP$
$VP \rightarrow Verb\ PP$	$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$	$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$	$PP \rightarrow Preposition\ NP$

CKY Parsing: Dynamic Programming

- Use indeces to point at gaps between words:

  $_0$ Book $_1$ the $_2$ flight $_3$ through $_4$ Houston $_5$

- A sentence with n words $\Rightarrow$ $n + 1$ positions.
  - *words*[1] = "book", *words*[2] = "the", …

- Define a $(n + 1)\times(n + 1)$ matrix T:
  - T[i,j] = the set of non-terminals that can generate the sequence of words between gaps i and j.
  - T[0,n] contains S $\Leftrightarrow$ the sentence can be generated by the CFG.

- How can we compute T[i,j]?
  - Only interested in the upper-triangular portion (i.e. $i < j$).

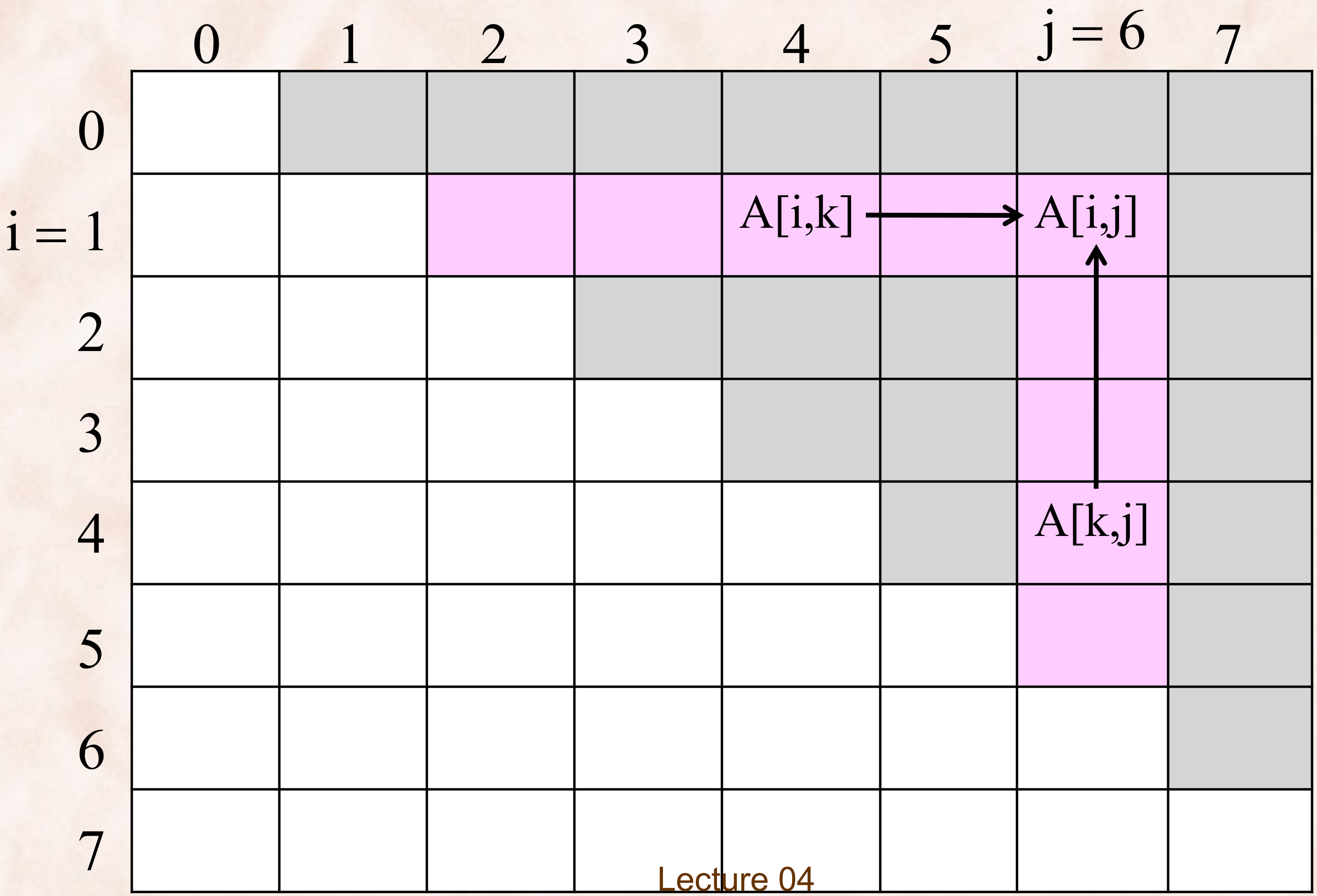
CKY: Dynamic Programming

- Recursively define the table values:
  1. $A \in T[i-1,i]$ if and only if there is a rule $A \rightarrow$ *words*$[i]$.
  2. $A \in T[i,j]$ if and only if $\exists$ k, $i < k < j$, such that:
     - $B \in T[i,k]$ and $C \in T[k,j]$.
     - There is a rule $A \rightarrow B\ C$ in the CFG.

- Bottom-up computation:
  – In order to compute the set $T[i,j]$, the sets $T[i,k]$ and $T[k,j]$ need to have been computed already, for all $i < k < j$.

  $\Rightarrow$ (at least) two possible orderings:
     - which one is more "natural"?

CKY: Bottom-Up Computation



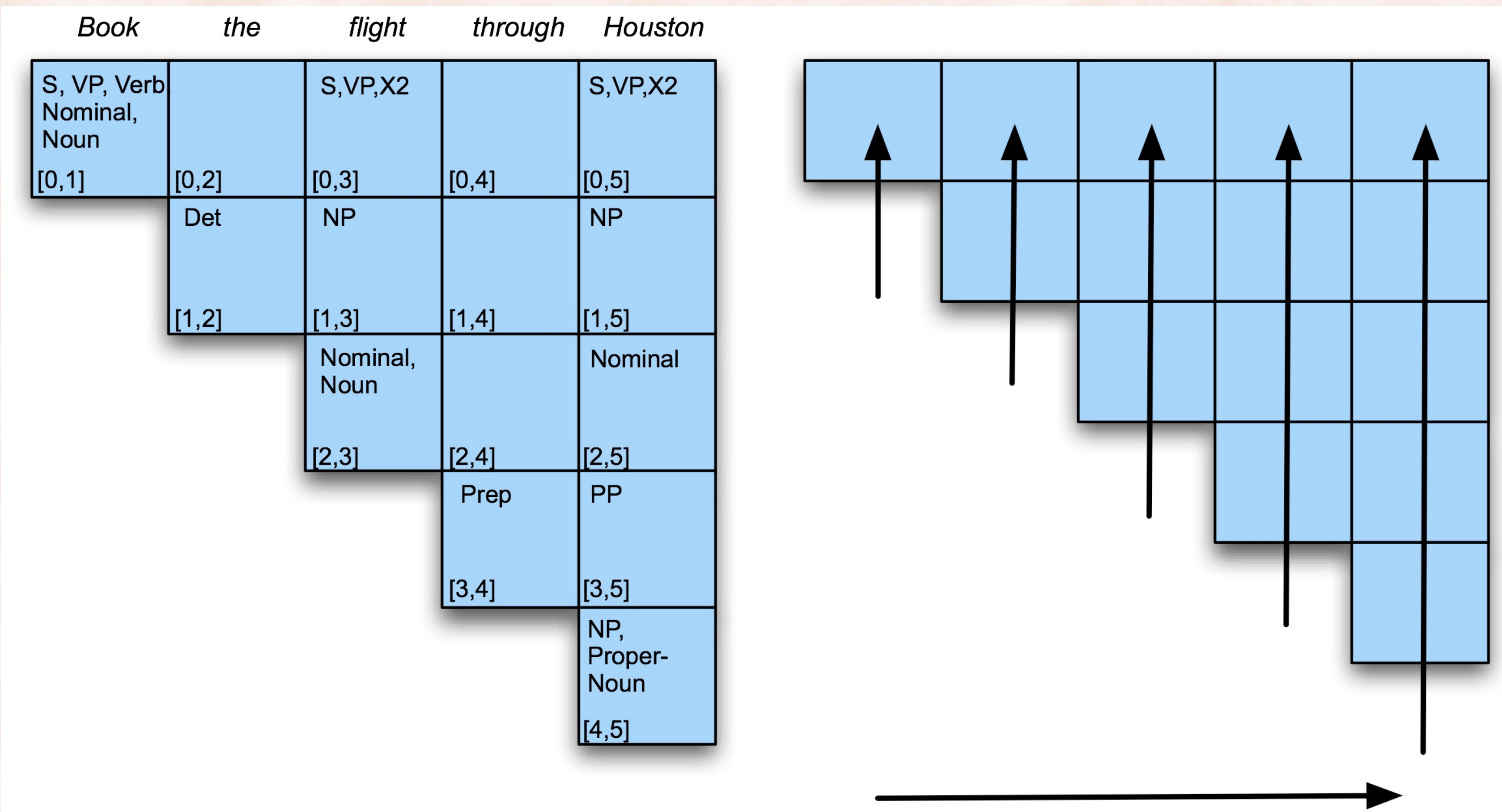
CKY Parsing

- Fill the table a column at a time, left to right, bottom to top.

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

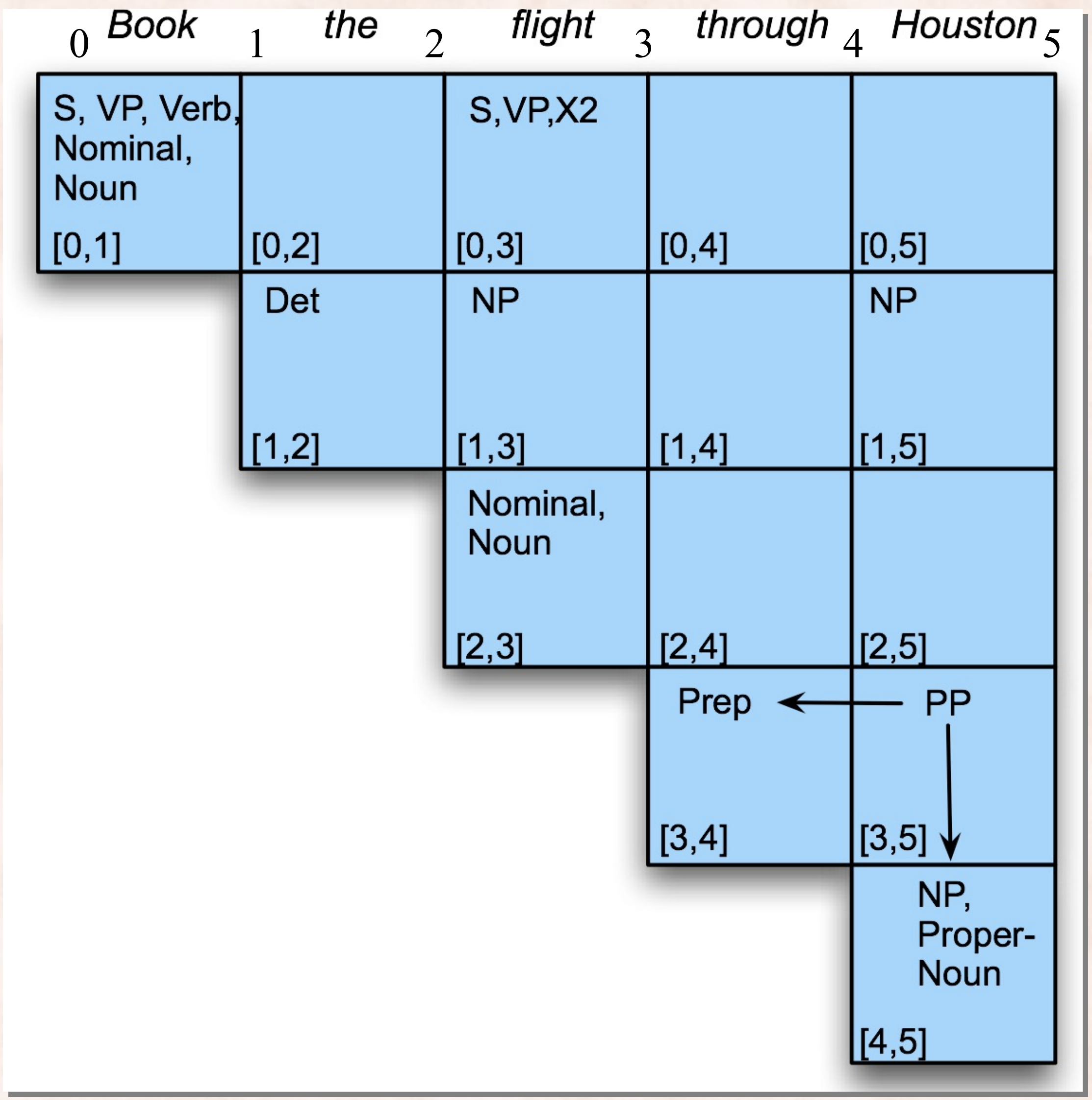
CKY Parsing: Example



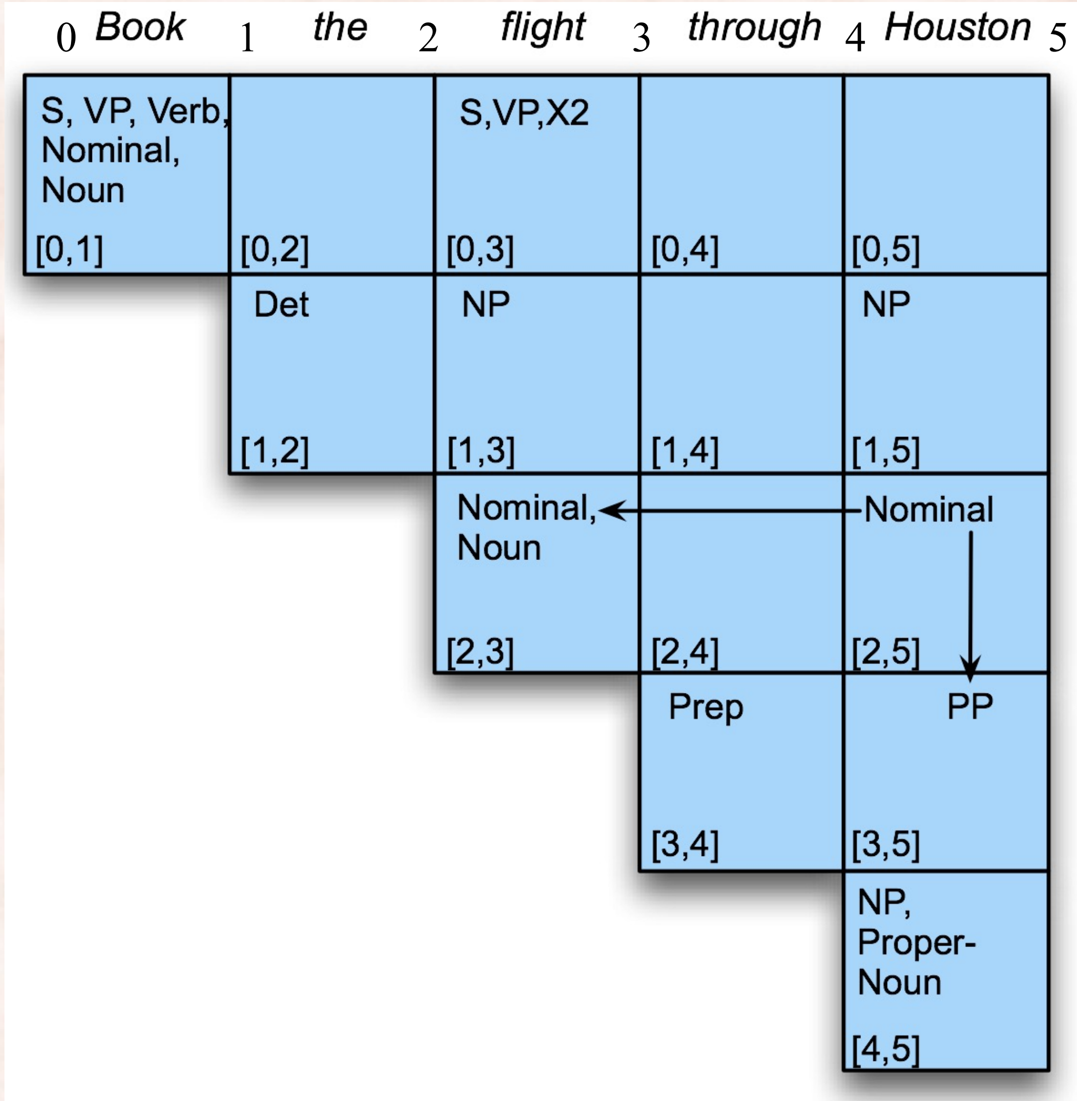
0 *Book* 1 *the* 2 *flight* 3 *through* 4 *Houston* 5

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 [2,5]
			Prep [3,4]	[3,5]
				NP, Proper-Noun [4,5]

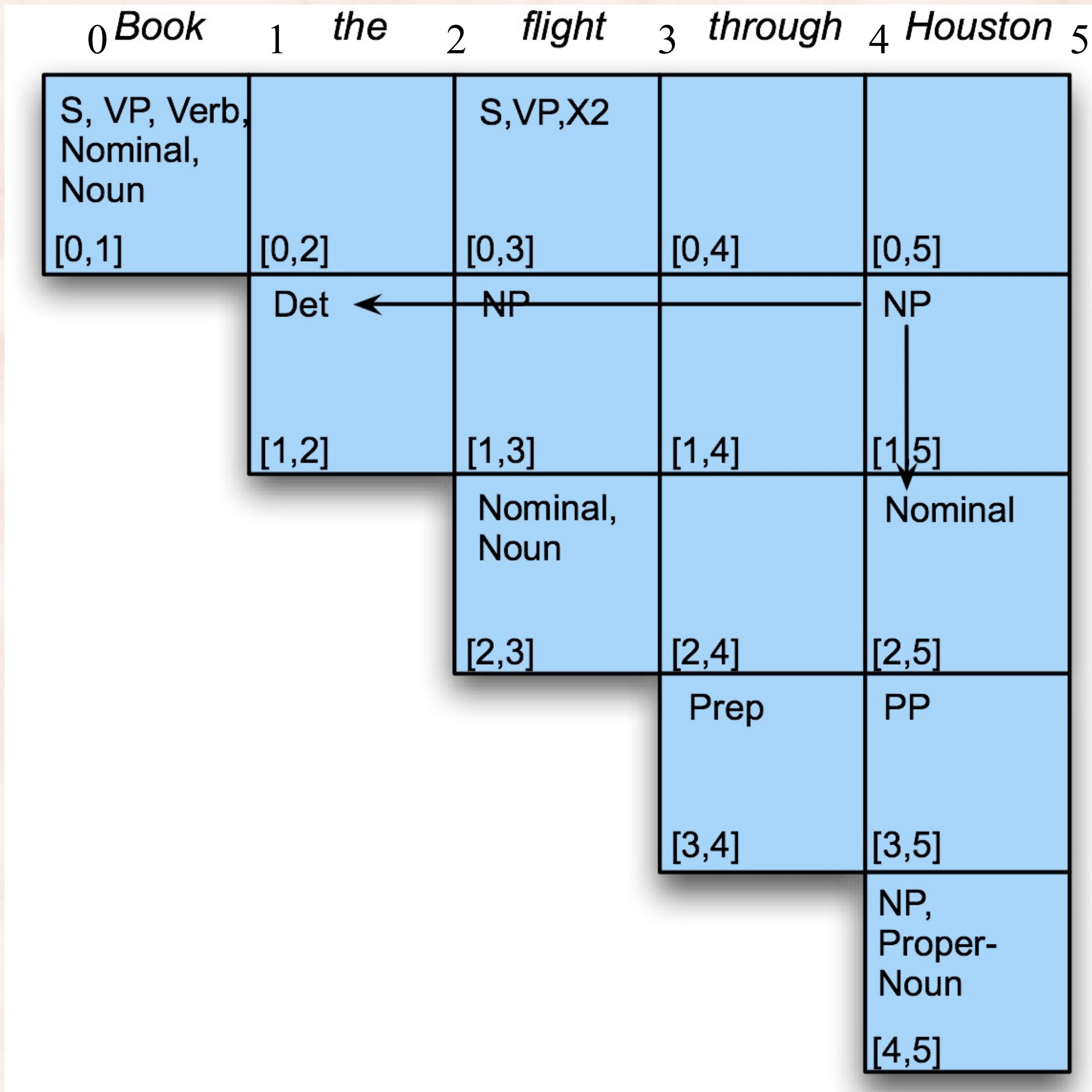
S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include | prefer
S → Verb NP
S → X2 NP
X2 → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → VP PP
VP → X2 PP
PP → Prep NP



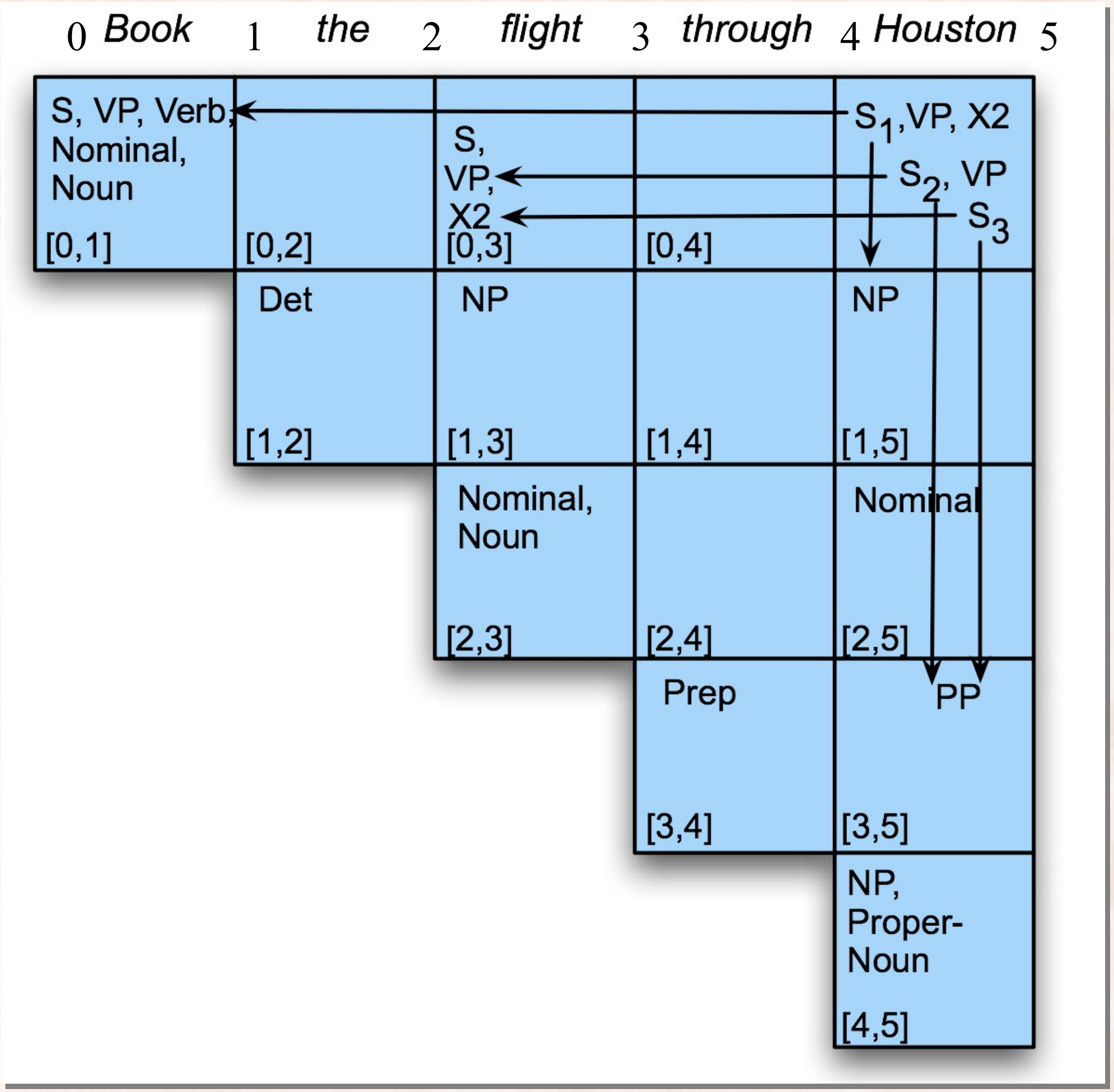
S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include | prefer
S → Verb NP
S → X2 NP
X2 → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → VP PP
VP → X2 PP
PP → Prep NP



S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include | prefer
S → Verb NP
S → X2 NP
X2 → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → VP PP
VP → X2 PP
PP → Prep NP



S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include | prefer
S → Verb NP
S → X2 NP
X2 → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → VP PP
VP → X2 PP
PP → Prep NP



S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include | prefer
S → Verb NP
S → X2 NP
X2 → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → VP PP
VP → X2 PP
PP → Prep NP

CKY Parsing

- How do we change the algorithm to output the parse trees?
- Time complexity:
  - for computing the table?
  - for computing all parses?

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

```
for j ← from 1 to LENGTH(words) do
    table[j−1, j] ← {A | A → words[j] ∈ grammar}
    for i ← from j−2 downto 0 do
        for k ← i+1 to j−1 do
            table[i,j] ← table[i,j] ∪
                         {A | A → BC ∈ grammar,
                              B ∈ table[i,k],
                              C ∈ table[k,j]}
```

CKY Parsing

- The parse trees correspond to the CNF grammar, not the original CFG:

  ⇒ complicates subsequent syntax-direct semantic analysis.

- Post-processing of the parse tree:
  - For binary productions:
    - delete the new dummy non-terminals and promote their daughters to restore the original tree.
  - For unit productions:
    - alter the basic CKY algorithm to handle them directly.
      - homework exercise 13.3

CKY Parsing

- Does CKY solve ambiguity?
  - Book the flight through Houston.

Use *probabilistic* CKY parsing, output *highest probability* tree.

- Will probabilistic CKY solve all ambiguity?
  - One morning I shot an elephant in my pajamas.
    - How he got into my pajamas I don’t know.

Shallow Parsing: Chunking

- **Chunking** = find all non-recursive major types of phrases:
  - [$_{NP}$ The morning flight] [$_{PP}$ from] [$_{NP}$ Denver] [$_{VP}$ has arrived]
  - [$_{NP}$ The morning flight] from [$_{NP}$ Denver] has arrived
- Chunking can be approached as **Sequence Labeling**.
- Evaluation:

$$\text{Precision (P)} = \frac{\#\,\text{correct chunks found}}{\text{total}\,\#\,\text{chunks found}}$$

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

$$\text{Recall (R)} = \frac{\#\,\text{correct chunks found}}{\text{total}\,\#\,\text{actual chunks}}$$

$$F_1 = \frac{2PR}{P + R}$$

Currently, best NP chunking system obtains F_1=96%.

Supplemental Reading

- Sections 12 and 13 from Jurafsky & Martin.
- Section 9 and 10 from Eisenstein.