Concepts and Applications in NLP

Grammars and Parsing

Marion Di Marco

November 19, 2024

Outline

- How to describe (English) sentences in a formal way
- What structures can we use to describe them?
- How can we find the structure of a given sentence?

Outline

Syntactic Constituency

- Syntactic constituents: groups of words that behave as single units
- Noun phrase: sequence of words surrounding at least one noun
  - *a green car*
  - *the reason he came yesterday*
  - *they* (Pronoun)
- Why do these sequences form constituents?
  → similar syntactic environments
  - *they* **swim**
  - *a green car* **drives**
  - *the reason he came yesterday* **was**
- Only complete noun phrases can occur before a verb, not parts thereof
  - **the* **was**

How to Identify Constituents?

- Asking for setence fragments:
  - The students eat pizza. → *Who* eats pizza?
  - The students eat pizza. → *What* eat the students?
- Coordination: constituents of the same type can be coordinated
  - We peeled the potatoes. → We washed and peeled the potatoes.
- Substitution, for example with pronouns:
  - The boy ate ice cream. → He ate ice cream.
  - We prepared pizza with pineapples last week. → We did that last week.
- Topicalization: move a constituent to the beginning of the sentence
  - He put is bike into the garden → Into the garden he put his bike.
- Clefting
  - He put sprinkles on the cake. → On the cake is where he put sprinkles.

Constituents

- A sequence of words that forms a constituent in one context is not necessarily a constituent in another context

(7) a. The students wondered how cheap textbooks could be obtained.
b. The students wondered how cheap textbooks could be.
a.' The students wondered how [**cheap textbooks**] could be obtained.
b.' The students wondered [**how cheap**] textbooks could be.

- [cheap textbooks] is a constituent in sentence 7(a)
  - *the students wondered how they could be obtained*

- [cheap textbooks] is **not** a constituent in sentence 7(b)
  - [textbooks] is a separate constituent:
    the students wondered how cheap they could be.
  - [how cheap] is a constituent:
    the students wondered what textbooks could be.

Example from *Understanding Syntax*, Tallermann (2011), Chapter 5

Outline

Constituents in Grammars: Idea

- Notion of constituency → abstraction
- Groups of words (= constituents) behave as single units
- We "know" how to group constituents of particular types
  for example, *noun phrases* can go before *verbs*

- Formal system for modeling constituent structure:
  Context Free Grammars (CFG) or phrase-structure grammars
- General idea: segment a constituent into smaller constituents
  up to the level of words

Outline

Context-Free Grammars: Example

- **Context-Free Grammars (CFG)** or **phrase-structure grammars**
- A CFG consists of a set of rules and a lexicon
- **Rules** or **productions** express how symbols of the language can be grouped and ordered
- The **lexicon** contains words and symbols
- Example grammar for a noun phrase:

Rules

NP	→	Det Nominal
NP	→	ProperNoun
Nominal	→	Noun | Nominal Noun

Lexicon

Det	→	a
Det	→	the
Noun	→	flight

Context-Free Grammars: Example

- Example grammar for a noun phrase:

Rules

NP	→	Det Nominal
NP	→	ProperNoun
Nominal	→	Noun | Nominal Noun

Lexicon

Det	→	a
Det	→	the
Noun	→	flight

- **Terminal symbols** in the lexicon correspond to the words
- **Non-terminal symbols** express abstractions over the terminals
- Context-free rule:
  - left side: one non-terminal symbol
  - right side: ordered list of one or more terminals and non-terminals
- The left-side non-terminal in the lexicon: the word's lexical category

Context-Free Grammar: Derivation

- A CFG can be thought of in two ways:
  - as a device for generating sentences
  - as a device for assigning a structure to a given sentence
- Viewing a CFG as a generator: read the → as *rewrite the symbol on the left with the string of symbols on the right*
- The string *a flight* can be derived from the non-terminal *NP*:

start from symbol:	*NP*
rewrite *NP* as:	*Det Nominal*
rewrite *Nominal* as:	*Noun*
rewrite the part-of-speech tags as:	*a flight*

- The sequence of rule expansions is called a **derivation**

Context-Free Grammar: Parse Tree

- Derivations can be represented by a parse tree

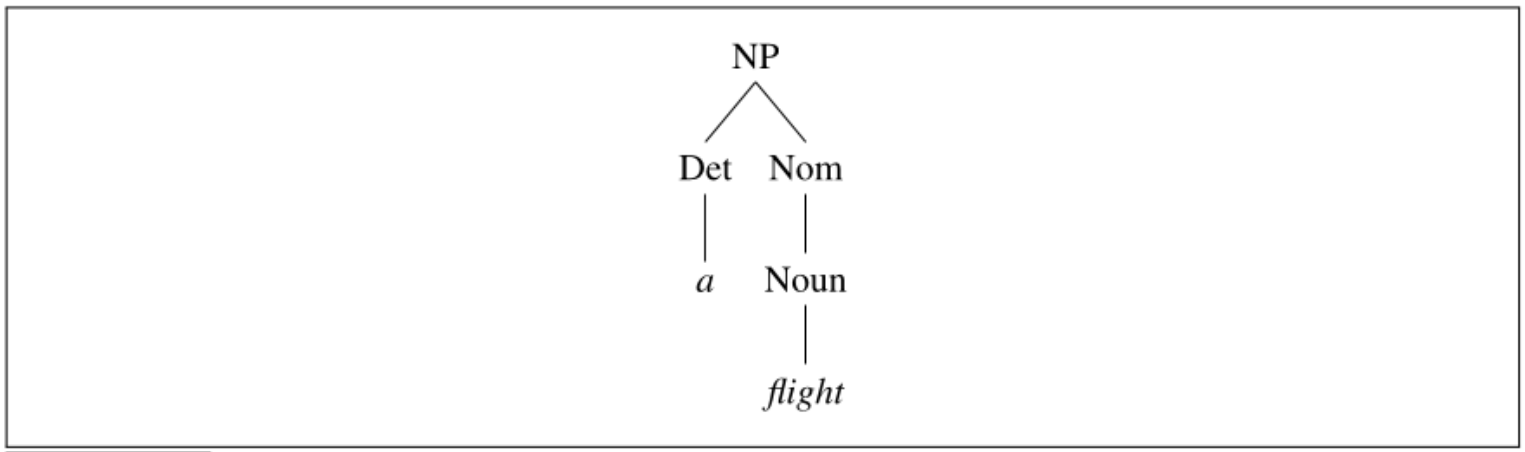


Figure 18.1 A parse tree for "a flight".

- The node *NP* **dominates** all the nodes in the tree (*Det*, *Nom*, *Noun*, *a*, *flight*)

- *NP* immediately dominates the nodes *Det* and *Nom*

Context-Free Grammar: Start Symbol

- Each grammar must have one designated **start symbol** S
- CFGs are often used to define sentences: S is often interpreted as the *sentence node*
- The formal language defined by a CFG:
  set of strings that are derivable from S

Context-Free Grammar Example: Simple Sentences

Rule			Example
S	→	NP VP	I prefer a morning flight
VP	→	Verb NP	prefer a morning flight
VP	→	Verb NP PP	leave Boston in the morning
VP	→	Verb PP	leaving on Thursday
PP	→	Prep NP	to Seattle

Noun →	*flights* | *flight* | *breeze* | *trip* | *morning*
Verb →	*is* | *prefer* | *like* | *need* | *want* | *fly* | *do*
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* | *in*
Conjunction →	*and* | *or* | *but*

Figure 18.2 The lexicon for $\mathcal{L}_0$.

Context-Free Grammar Example: Simple Sentences

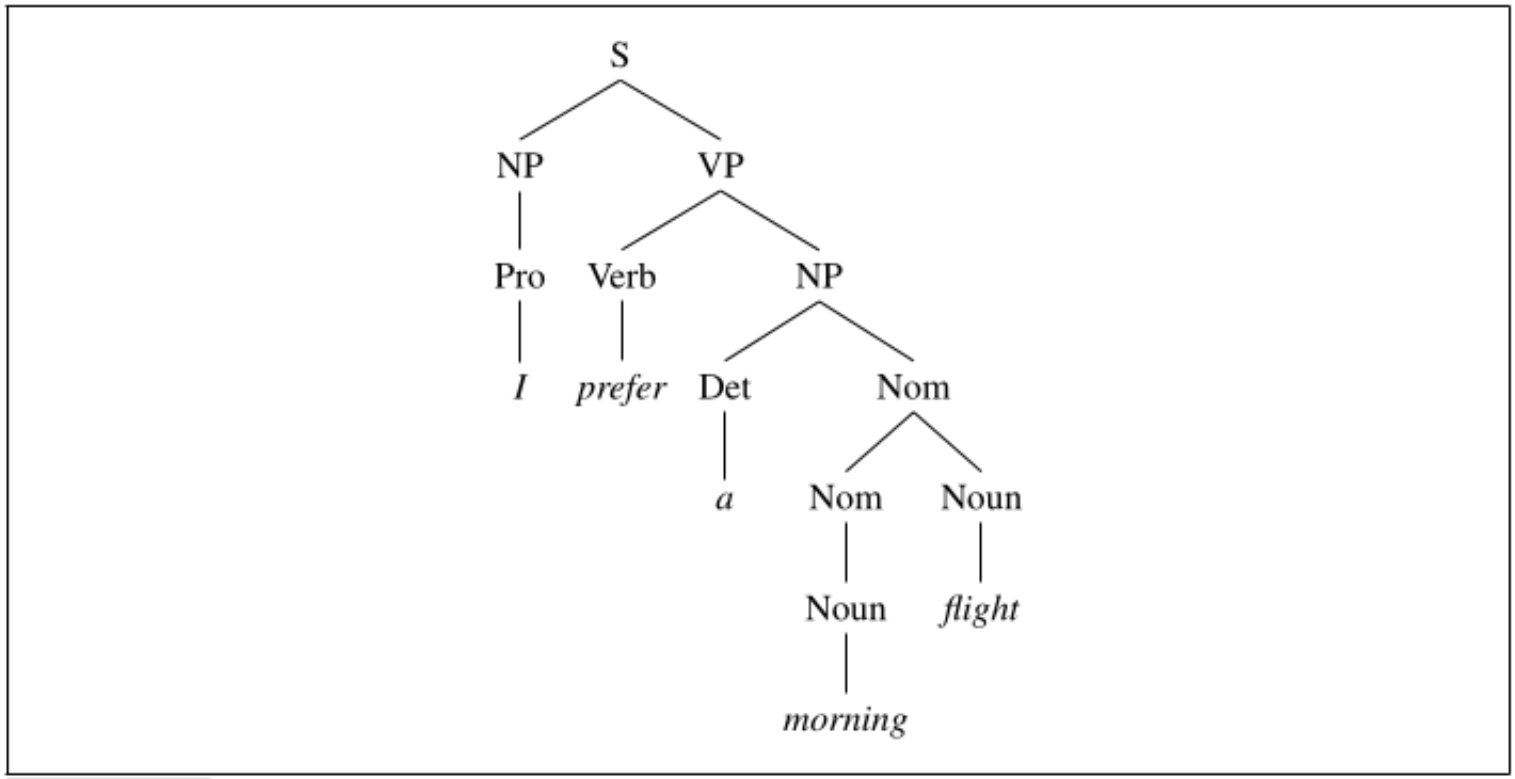


Figure 18.4 The parse tree for "I prefer a morning flight" according to grammar $\mathcal{L}_0$.

Bracketed notation:

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

Grammatical vs. Ungrammatical Sentences

- Sentences that can be derived by a grammar are in the formal language defined by that grammar: **grammatical** sentences
- Sentences that cannot be derived by a grammar are not in the formal language defined by that grammar: **ungrammatical** sentences
- Formal languages are only a very simplified model of natural languages
  often difficult to decide whether a sentence is a valid sentence of e.g. English
- Formal languages to model natural languages: **generative grammar**
  the language is defined by the set of possible sentences "generated" by the grammar

Formal Definition of Context-Free Grammar

- A context-free grammar G is defined by four parameters

N	a set of **non-terminal symbols** (or **variables**)
Σ	a set of **terminal symbols** (disjoint from N)
R	a set of **rules** or productions, each of the form $A \rightarrow \beta$,
	where A is a non-terminal,
	β is a string of symbols from the infinite set of strings $(\Sigma \cup N)^*$
S	a designated **start symbol** and a member of N

Outline

Parsing

- Parsing: determine the rules of a sentence given the rules of a grammar

$\Rightarrow$ Is the sentence valid?

$\Rightarrow$ What is the structure of the sentence?

CYK Parsing

- CYK (or CKY) algorithm (Cocke-Kasami-Younger):
  a dynamic bottom-up approach to parsing in a context-free grammar
  - bottom-up: start with the words
  - dynamic programming approach: solve complex problems by breaking them down into smaller subproblems; save their results in a table and re-use them
- CYK first detects smaller constituents and stores them for merging into larger constituents
- The CKY algorithm requires grammars to first be in Chomsky Normal Form (CNF)
- Complexity: $O(N^3|G|)$ where N = input length, $|G|$ = grammar size

Chomsky Normal Form

- A CFG is in **Chomsky Normal form** if
  - each production is of the form $A \rightarrow BC$ or $A \rightarrow a$
  - each production is ϵ-free (ϵ = empty string)
- Chomsky normal form grammars are **binary branching**
- Any CFG can be converted into a (weakly equivalent) Chomsky Normal Form Grammar
- Two grammars are **weakly equivalent** if they generate the same set of strings but not the same phrase structure to each sentence.
- Formally, CFGs can have empty productions (NP $\rightarrow \epsilon$)
  Empty productions can be eliminated without changing the language of the grammar
- We assume that there are no ϵ productions in the grammar!

Chomsky Normal Form

- How to transform a CFG into Chomsky Normal Form
- **Rules with more than two non-terminals**

$\Rightarrow$ Use "dummy" non-terminals:

$A \rightarrow B\ C\ D$

$A \rightarrow B\ X$
$X \rightarrow C\ D$

- **Unit productions:** rules with a single non-terminal on the right side

$\Rightarrow$ Rewrite the right-hand side of the original rules with the right-hand side of all the non-unit production rules that they ultimately lead to

Chomsky Normal Form: Example Grammar

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

Figure 18.10 $\mathcal{L}_1$ Grammar and its conversion to CNF. Note that although they aren't shown here, all the original lexical entries from $\mathcal{L}_1$ carry over unchanged as well.

CYK Recognition

- Each non-terminal node above the part-of-speech level in a parse tree has exactly two daughters
- General idea: apply binary productions to merge adjacent symbols into larger constituents
- Bottom-up parsing: starting from words, apply a series of merges that ultimately result in the start symbol S covering the input
- CYK: systematically construct a table in which each cell contains the set of non-terminals that derive the covered span of words
- Filling the table bottom up: all cells contributing to the current cell are already filled

CYK Parsing

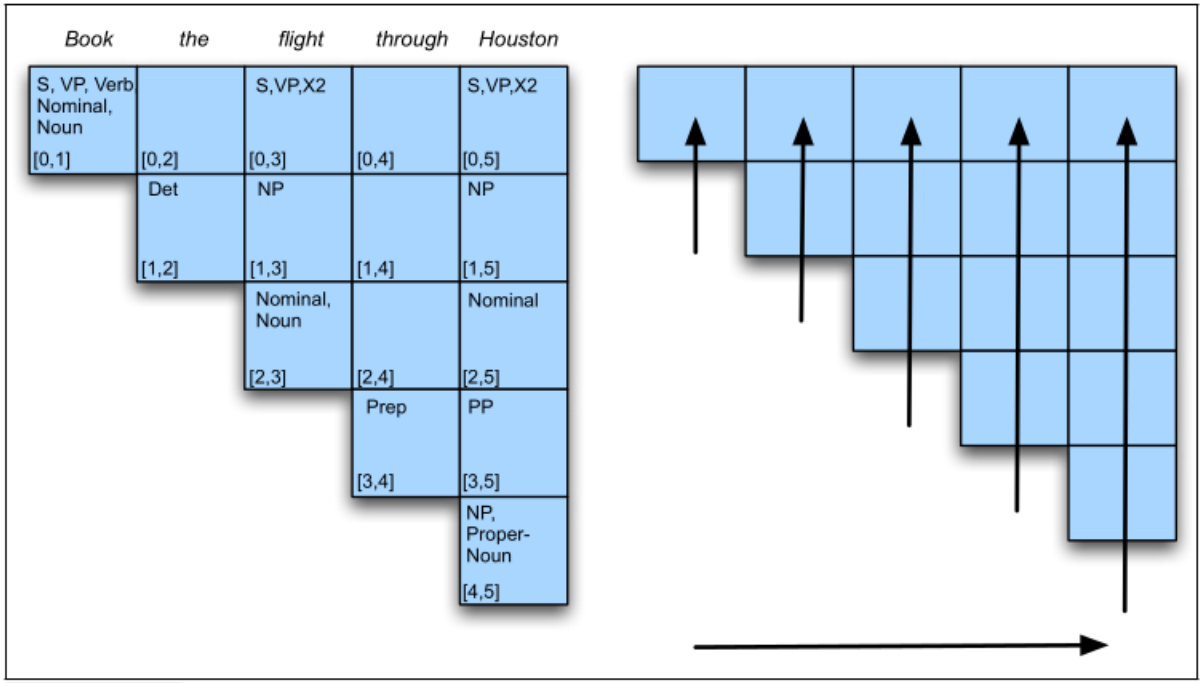


Figure 18.11 Completed parse table for *Book the flight through Houston.*

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

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

Figure 18.12 The CKY algorithm.

CYK Parsing

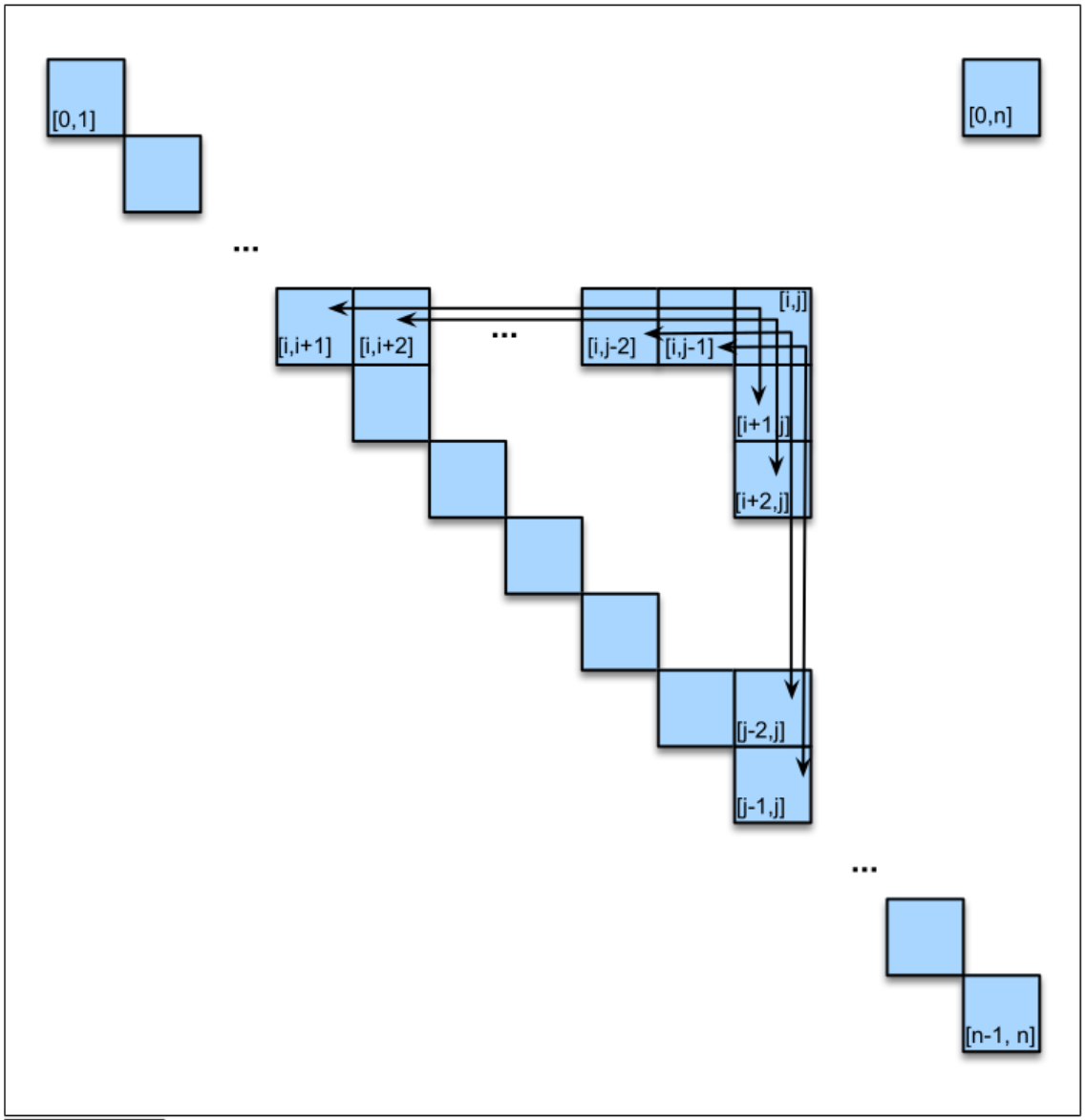
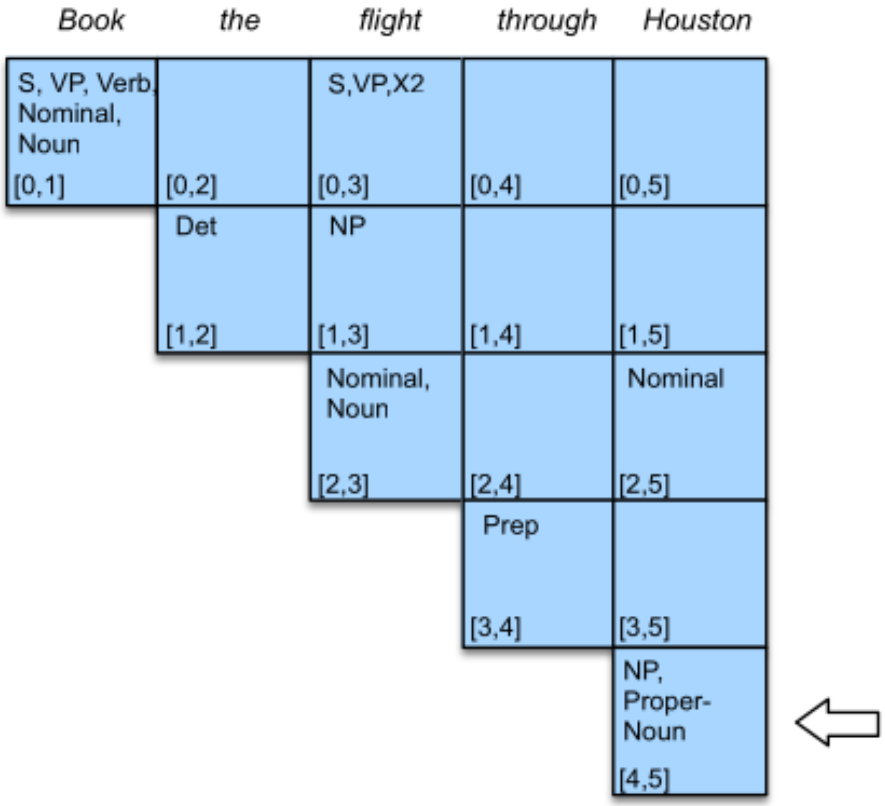
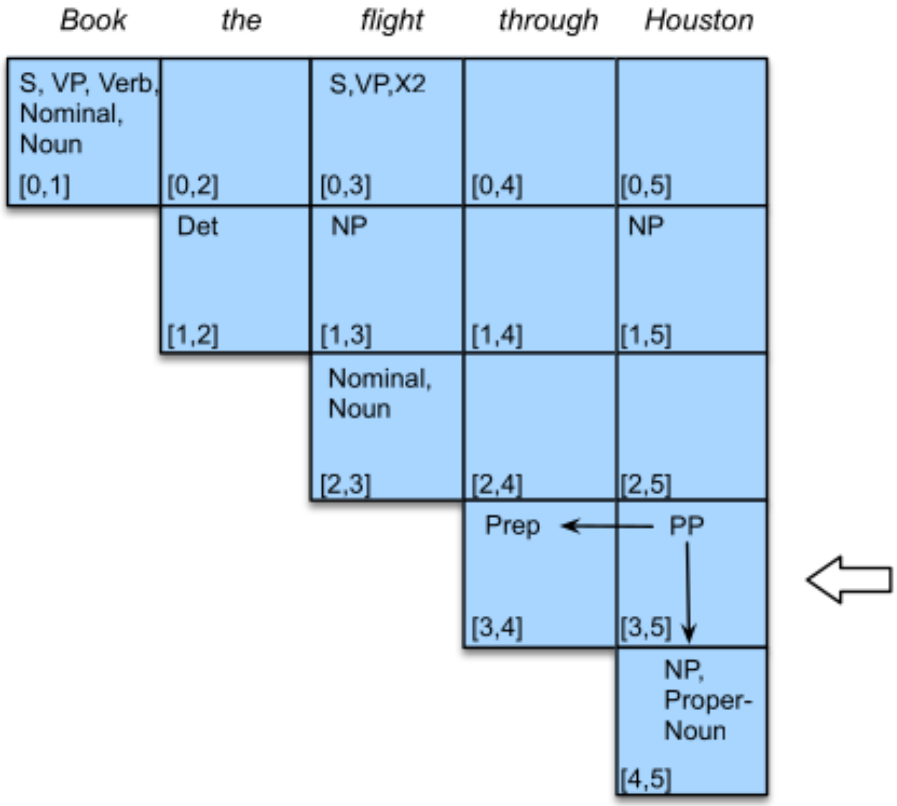


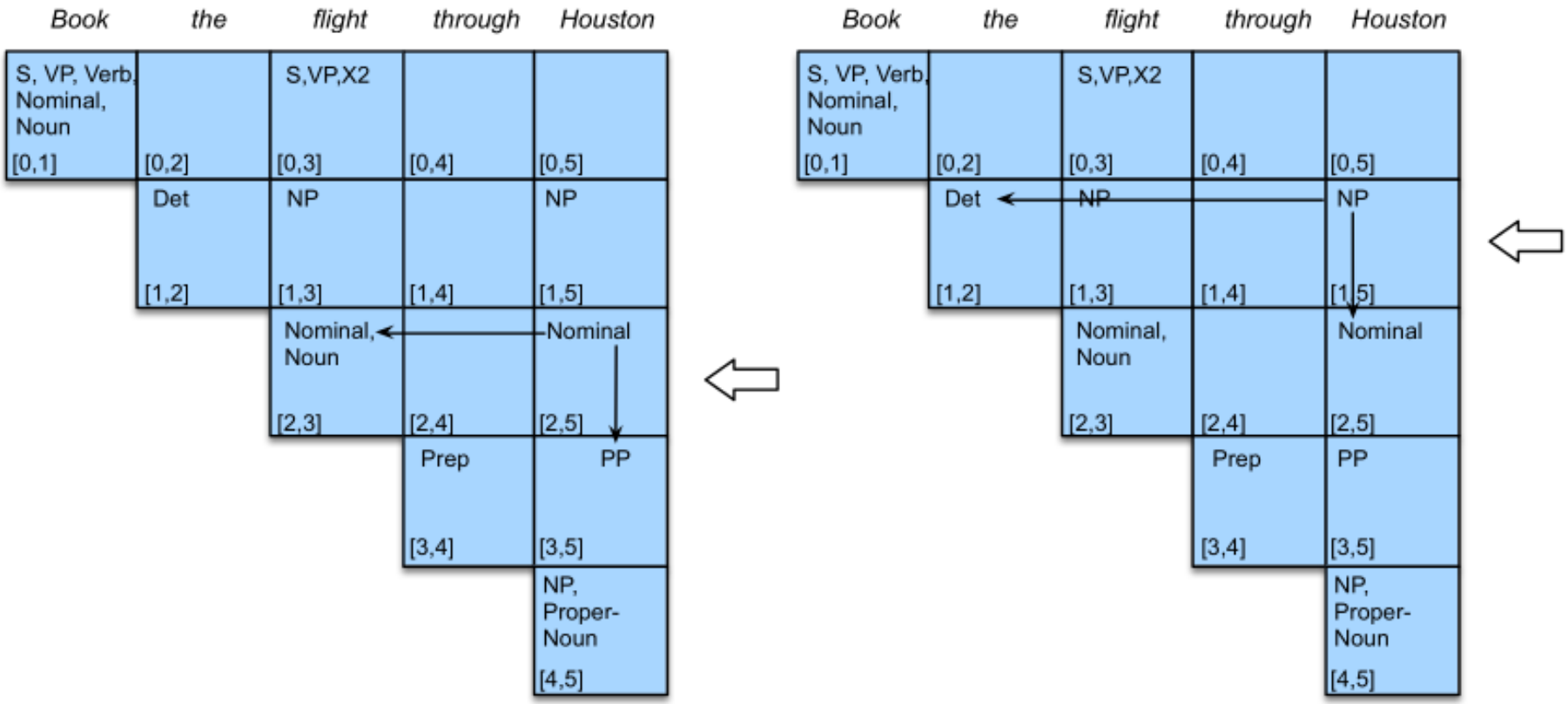
Figure 18.13 All the ways to fill the $[i, j]$th cell in the CKY table.

CYK Parsing: Example

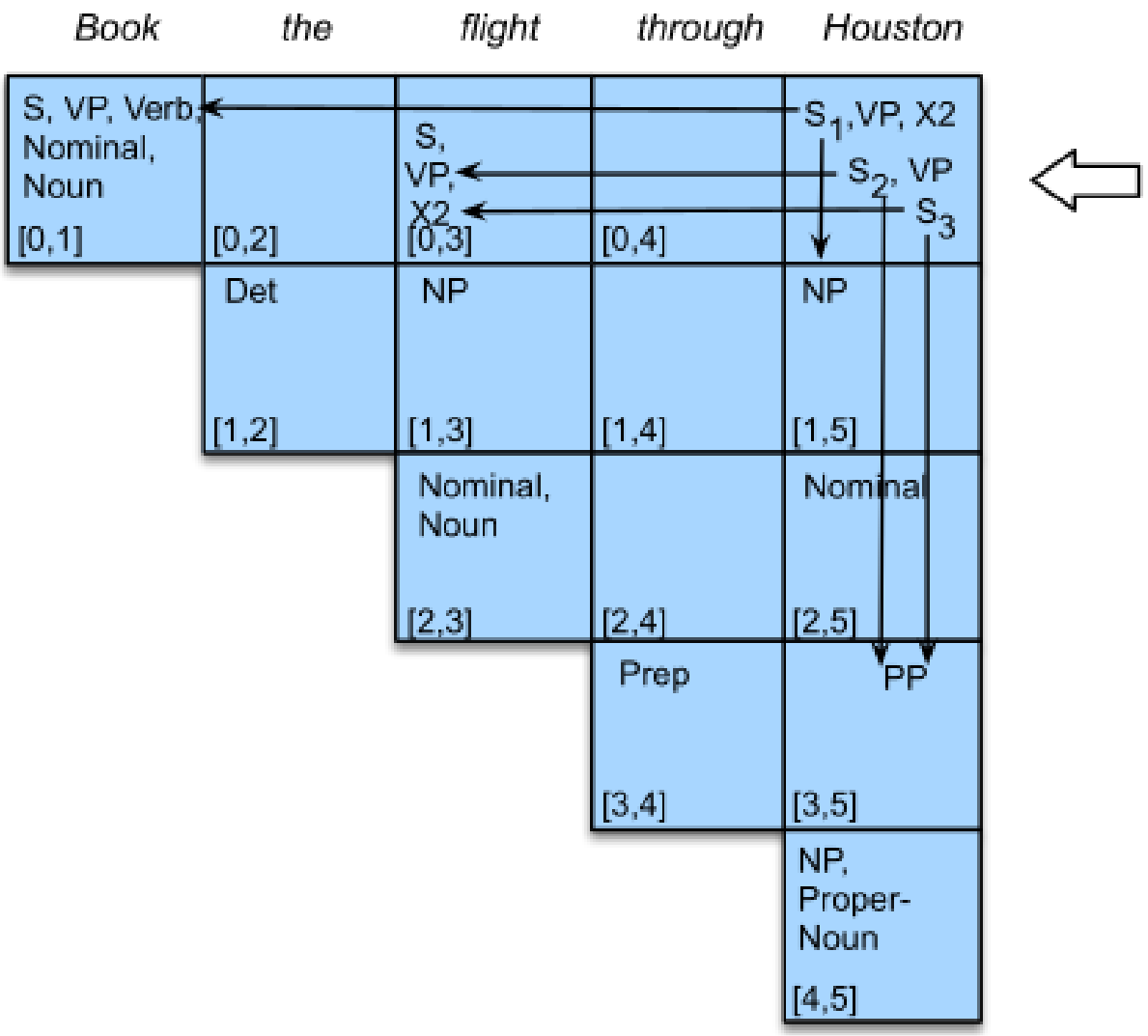




CYK Parsing: Example



CYK Parsing: Example



CYK Recognition vs. Parsing

- Technically, the algorithm in the pseudo-code is a recognizer, not a parser
  - it tells whether a valid parse exists
  - valid if there is an S spanning the sentence in cell [0,n]
- Does not yet provide a derivation
  - augment the entries in the table so that each non-terminal is paired with pointers to the table entries from which it was derived

$\Rightarrow$ The completed table contains all the possible parses for a given input

- Return an arbitrary parse:
  chose an S from cell [0,n] and retrieve its constituents
- Return the best parse?

Outline

Ambiguities

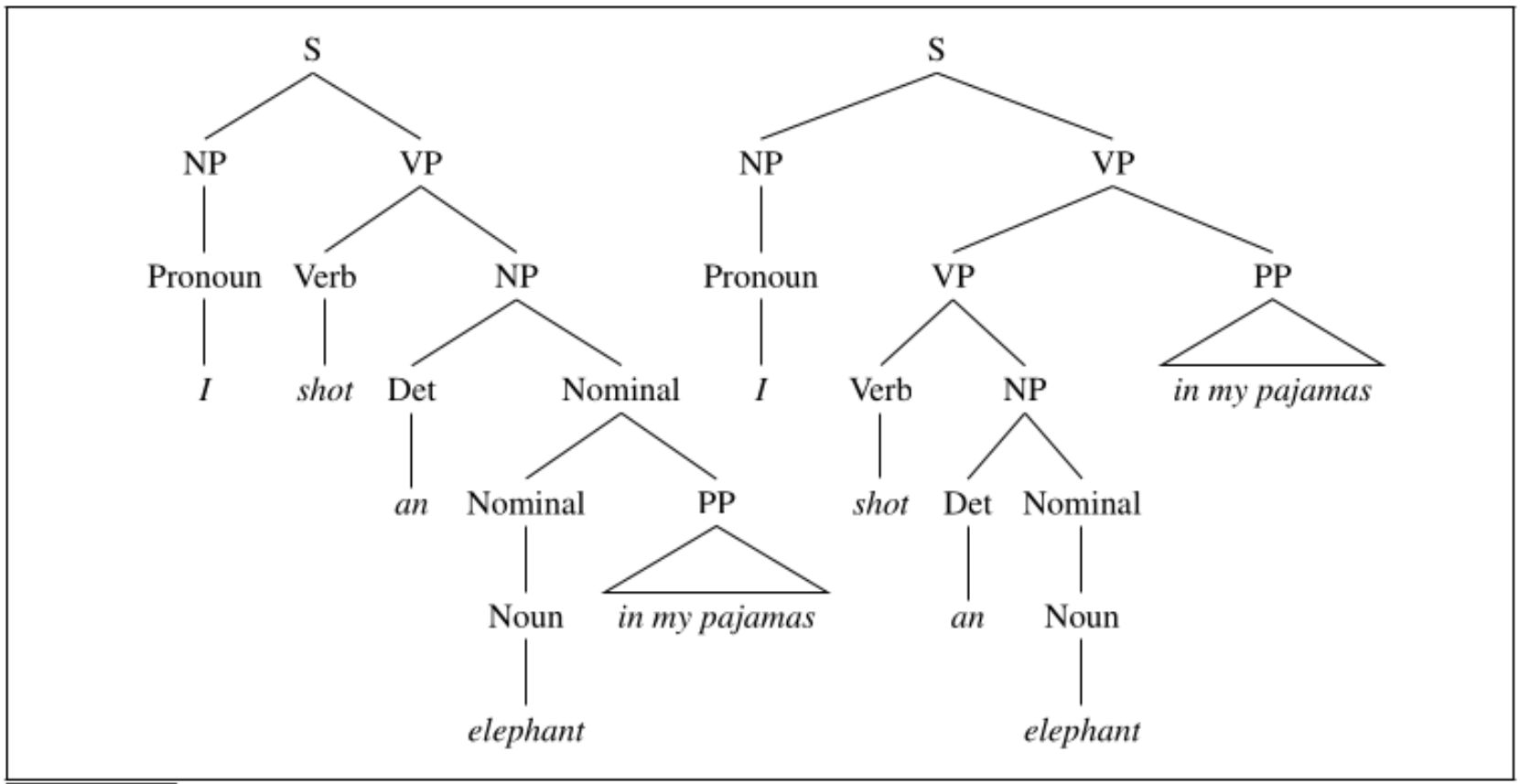
- **Structural ambiguity**: a grammar can assign more than one parse
- For example: PP-attachment

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

Figure 18.8 The $\mathcal{L}_1$ miniature English grammar and lexicon.

PP-Attachment

"I shot an elephant in my pajamas"



Structural Ambiguity

- Structural ambiguity comes in many forms
- Attachment ambiguity: a particular constituent can be attached to the parse tree at more than one place
  - *"I see the man with the telescope."*
  - *"We eat sushi with chopsticks"* vs. *"We eat sushi with salmon"*
- Coordination ambiguity: phrases can be conjoined by a conjunction
  - *"old men and women"* → *[old [men and women]]*
[old men] and [women]
- Modifier scope: *[plastic [cup holder]]* vs. *[[plastic cup] holder]*
- Complement structure: *"The students complained to the professor that they didn't understand"*
  - *that they didn't understand* → *complained*
  - *that they didn't understand* → *the professor*

Structural Ambiguity: Side Note

- Ambiguities are not necessarily the same across languages

English We eat sushi with chopsticks:
French *On mange des sushis* **avec** *des baguettes*

English We eat sushi with salmon:
French *On mange des sushis* **au** *saumon*

Ambiguities: Local Solutions

- Ambiguous PP-attachment:

  (a) *We met the president on Monday.*
  (b) *We met the president of Mexico.*

  $\Rightarrow$ the PP can be attached to the verb (*met*) or the NP (*the president*)

- Given a labeled corpus: compare the likelihood of observing the preposition alongside each candidate attachment point:

  $p(on|met) \lessgtr p(on|President)$
  $p(of|met) \lessgtr p(of|President)$

- With sufficient labeled data, some instances of attachment ambiguity can be solved by supervised classification

- Limitation: short toy examples vs. realistic sentences

Outline

Probabilistic Context-Free Grammar

- A Probabilistic Context-Free Grammar (PCFG) consists of
  - a set of **terminals** W^k (k = 1,...,V)
  - a set of **non-terminals** N^i
  - a designated **start symbol** S (member of N)
  - a set of **rules** $N^i \rightarrow \zeta^j$

    (where ζ^j is a sequence of terminals and non-terminals)
  - A corresponding set of probabilities on rules such that

    $\forall i \sum_j P(N^i \rightarrow \zeta^j) = 1$
- Probability of a sentence (according to a grammar G):

  $P(w_{1m}) = \sum_t P(w_{1m}, t)$ where t is a parse tree of the sentence
- Probability of a tree: multiply the probabilities of the rules that built the subtrees

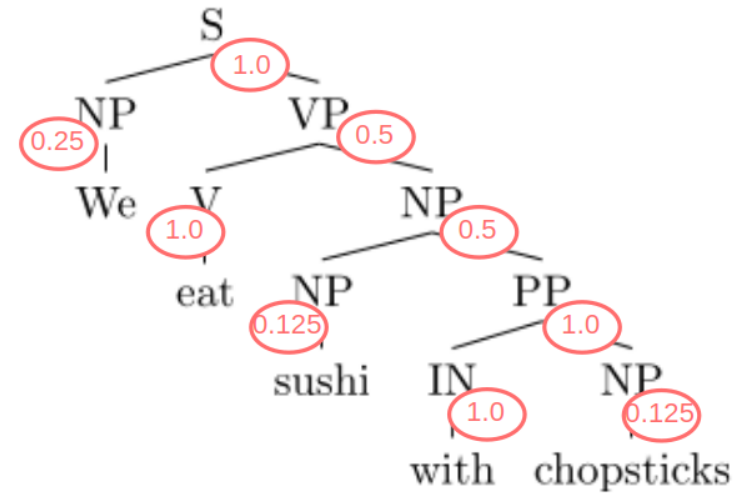
PCFG: Assumptions

- **Place invariance.** The probability of a subtree does not depend on where in the string the words it dominates are
- **Context-free.** The probability of a subtree does not depend on words not dominated by the subtree
- **Ancestor-free.** The probability of a subtree does not depend on nodes in the derivation outside the subtree

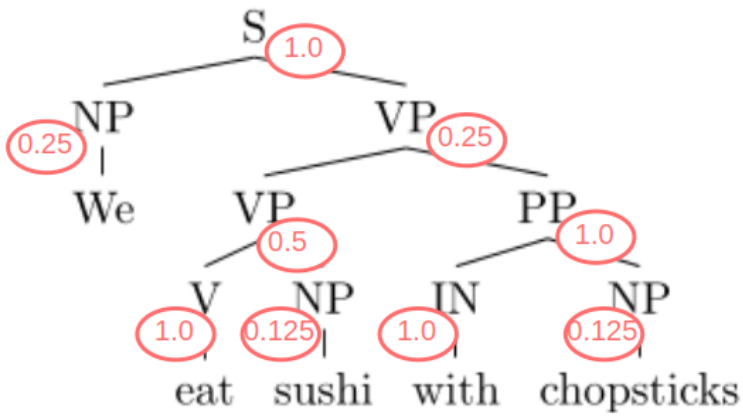
- Using these conditions we can justify the calculation of the probability of a tree in terms of just multiplying probabilities attached to rules

PCFG: Example Grammar

S	→	NP VP	1
NP	→	NP PP	0.5
	→	*we*	0.25
	→	*sushi*	0.125
	→	*chopsticks*	0.125
PP	→	IN PP	1
IN	→	with	1
VP	→	V NP	0.5
	→	VP PP	0.25
	→	MD V	0.25
V	→	eat	1



($tree_1$)



($tree_2$)

- $P(tree_1) = 1.0 \times 0.25 \times 0.5 \times 1.0 \times 0.5 \times 0.125 \times 1.0 \times 1.0 \times 0.125$
- $P(tree_2) = 1.0 \times 0.25 \times 0.25 \times 0.5 \times 1.0 \times 1.0 \times 0.125 \times 1.0 \times 0.125$

Context and Independence Assumptions

- Humans make wide use of the context of an utterance to disambiguate language
- The prior discourse context influences the interpretation of later sentences ("priming" in the psychological literature)
- People will find semantically intuitive readings for sentences in preference to weird ones
- In our PCFG model: making independence assumptions that none such factors are relevant
- But: context information is relevant to and might be usable for disambiguating probabilistic parses

Lexicalization

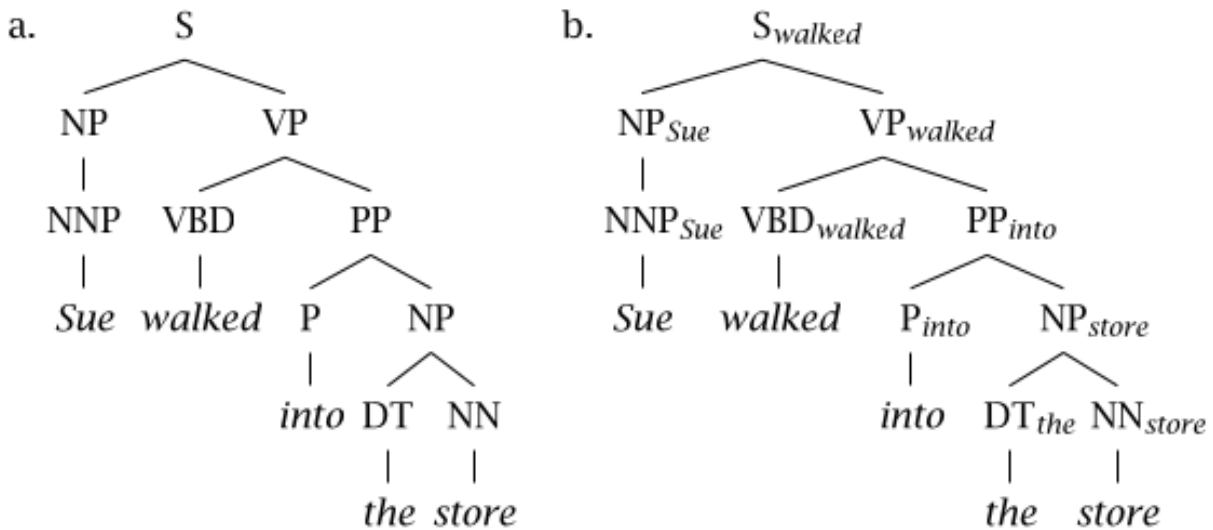
- Lack of lexicalization in PCFG
- Example: expansion of a VP is independent of the verb
  - → different subcategorization frames of intransitive, transitive and intransitive verbs

	Verb			
Local tree	*come*	*take*	*think*	*want*
VP → V	9.5%	2.6%	4.6%	5.7%
VP → V NP	1.1%	32.1%	0.2%	13.9%
VP → V PP	34.5%	3.1%	7.1%	0.3%
VP → V SBAR	6.6%	0.3%	73.0%	0.2%
VP → V S	2.2%	1.3%	4.8%	70.8%
VP → V NP S	0.1%	5.7%	0.0%	0.3%
VP → V PRT NP	0.3%	5.8%	0.0%	0.0%
VP → V PRT PP	6.1%	1.5%	0.2%	0.0%

Table 12.2 Frequency of common subcategorization frames (local trees expanding VP) for selected verbs. The data show that the rule used to expand VP is highly dependent on the lexical identity of the verb. The counts ignore distinctions in verbal form tags. Phrase names are as in table 12.1, and tags are Penn Treebank tags (tables 4.5 and 4.6).

Lexicalization

- Choosing phrasal attachment positions
  - lexical content of phrases often provides relevant information
  - syntactic category of the phrase provides only little information
- PCFGs fail to capture the lexical dependencies between words
- Combine lexical information and richer model than linear n-grams
- Lexicalize a CFG: mark phrasal node by its head word



Structural Context

- PCFGs are also deficient on purely structural grounds
- Assumption that probabilities are context-free:
  probability of a noun phrase expanding in a certain way is independent of where the NP is in the tree
- But: this is not true when looking at data

Expansion	% as Subj	% as Obj
NP → PRP	13.7%	2.1%
NP → NNP	3.5%	0.9%
NP → DT NN	5.6%	4.6%
NP → NN	1.4%	2.8%
NP → NP SBAR	0.5%	2.6%
NP → NP PP	5.6%	14.1%

Table 12.3 Selected common expansions of NP as Subject vs. Object, ordered by log odds ratio. The data show that the rule used to expand NP is highly dependent on its parent node(s), which corresponds to either a subject or an object.

Outline

Measuring Parsing Performance

- Assume we have a set of *reference parses* and a set of *system parses*
- Per-sentence accuracy: proportion of sentences on which the system and reference parses exactly match
- Give partial credit when correct parts match with the reference

  PARSEval metrics:

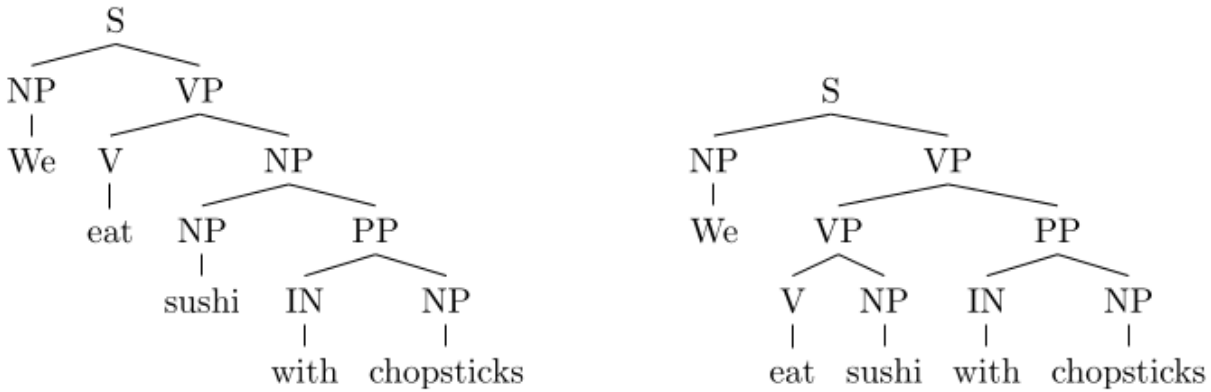
  Precision $= \frac{\#\ of\ correct\ constituents\ in\ hypothesis\ parse\ of\ s}{\#of\ total\ constituents\ in\ hypothesis\ parse\ of\ s}$

  Recall $= \frac{\#\ of\ correct\ constituents\ in\ hypothesis\ parse\ of\ s}{\#of\ total\ constituents\ in\ reference\ parse\ of\ s}$

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

- **Labeled** precision/recall: phrase type for each constituent must match
  Unlabeled precision/recall: only constituent structure must match

Measuring Parsing Performance: Example



(a) system output (b) reference

Figure 10.2: Two possible analyses from the grammar in Table 10.1

- S $\rightarrow w_{1:5}$ is *true positive*, because it appears in both trees
- VP $\rightarrow w_{2:5}$ is *true positive* as well
- NP $\rightarrow w_{3:5}$ is *false positive*, because it appears only in the system output
- PP $\rightarrow w_{4:5}$ is *true positive*, because it appears in both trees
- VP $\rightarrow w_{2:3}$ is *false negative*, because it appears only in the reference
- Precision $= \frac{3}{4}$, Recall $= \frac{3}{4}$, $F_1 = \frac{3}{4}$

Outline

Treebanks

- **Treebank**: collection of correctly parsed sentences
- Useful for building statistical parsers
- Penn Treebank:
  - the first publicly available syntactically annotated corpus
  - Wall Street Journal (50,000 sentences, 1 million words)
  - Other corpora: Brown Corpus, ATIS
  - Automatically parsed, manual correction
  - Standard data set for English phrase-structure parsers

Penn Treebank: Example

```
( (S (NP-SBJ The move)
     (VP followed
         (NP (NP a round)
             (PP of
                 (NP (NP similar increases)
                     (PP by
                         (NP other lenders))
                     (PP against
                         (NP Arizona real estate loans)))))
         ,
         (S-ADV (NP-SBJ *)
                (VP reflecting
                    (NP (NP a continuing decline)
                        (PP-LOC in
                                (NP that market))))))
     .))
```

Figure 12.2 A Penn Treebank tree.

S	Simple clause (sentence)	CONJP	Multiword conjunction phrases
SBAR	S′ clause with complementizer	FRAG	Fragment
SBARQ	*Wh*-question S′ clause	INTJ	Interjection
SQ	Inverted *Yes/No* question S′ clause	LST	List marker
SINV	Declarative inverted S′ clause	NAC	Not A Constituent grouping
ADJP	Adjective Phrase	NX	Nominal constituent inside NP
ADVP	Adverbial Phrase	PRN	Parenthetical
NP	Noun Phrase	PRT	Particle
PP	Prepositional Phrase	RRC	Reduced Relative Clause
QP	Quantifier Phrase (inside NP)	UCP	Unlike Coordinated Phrase
VP	Verb Phrase	X	Unknown or uncertain
WHNP	*Wh*- Noun Phrase	WHADJP	*Wh*- Adjective Phrase
WHPP	*Wh*- Prepositional Phrase	WHADVP	*Wh*- Adverb Phrase

Table 12.1 Abbreviations for phrasal categories in the Penn Treebank. The common categories are gathered in the left column. The categorization includes a number of rare categories for various oddities.

Penn Treebank: Some Characteristics

- Representation in Lisp notation (bracketing)
- Grouping of words into phrases is fairly flat:
  for example, no disambiguation of the compound noun in the phrase *Arizona real estate loans*
- Some attempt to indicate grammatical and semantic functions:
  the -SBJ and -LOC tags in the figure
- Empty nodes to indicate understood subjects and extraction gaps
  the understood subject of the adverbial clause, where the empty node is marked as *

Outline

Dependency Grammars – Intro

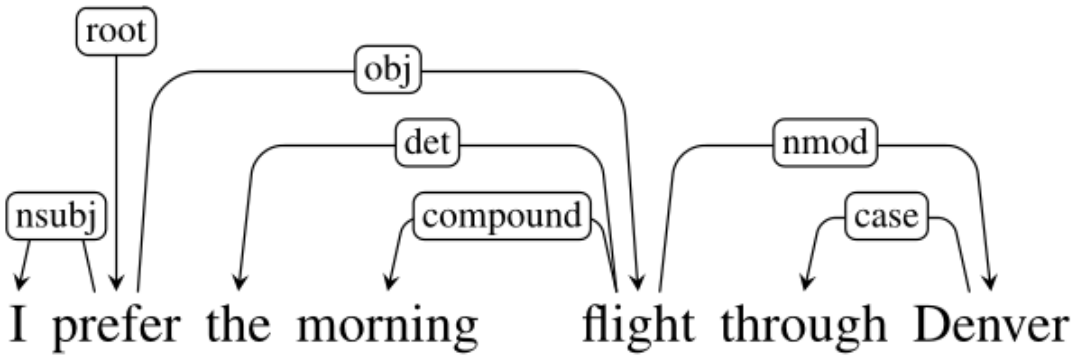
Tout mot qui fait partie d'une phrase... Entre lui et ses voisins, l'esprit aperçoit des connexions, dont l'ensemble forme la charpente de la phrase.

[Between each word in a sentence and its neighbors, the mind perceives **connections**. These connections together form the scaffolding of the sentence.]

Lucien Tesnière. 1959. Éléments de syntaxe structurale, A.1.§4

Dependency Grammars

- Syntactic structure is described in terms of directed binary grammatical relations



- Relations among the words represented as **directed, labeled arcs from heads to dependents**
- **Typed dependency structure**: fixed inventory of grammatical relations
- **Root node**: marks the head of the tree

Dependency and Phrasal Structures Compared

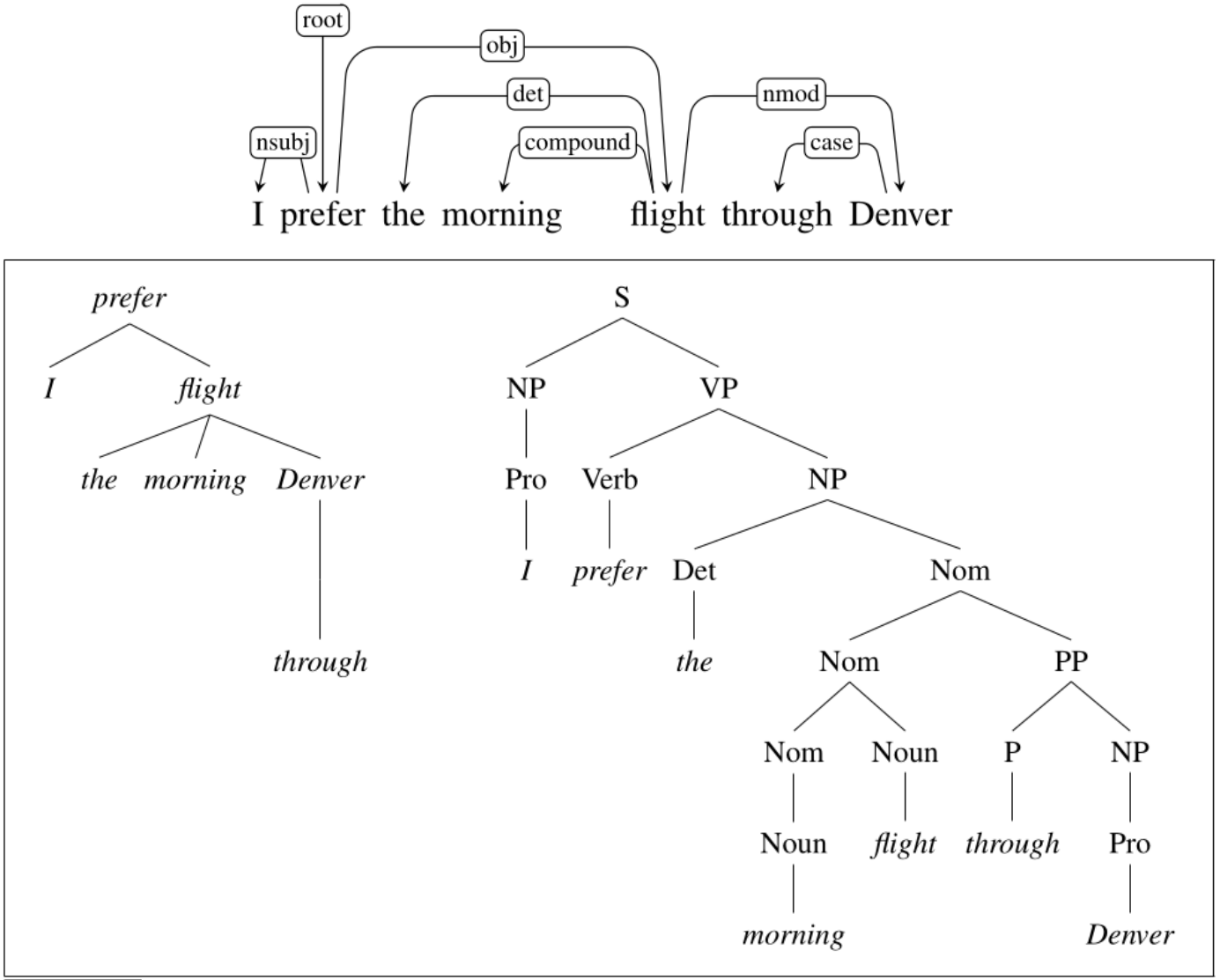


Figure 19.1 Dependency and constituent analyses for *I prefer the morning flight through Denver.*

Dependency Grammars

- Absence of nodes corresponding to phrasal constituents or lexical categories in the dependency parse
- Internal structure of the dependency parse: directed relations between words
- Head-dependent relationships encode important information which is often buried the more complex phrase-structure parses
- Example: verb arguments
  - arguments (= subject, object) are directly linked to *prefer*
  - connection to main verb is more distant in the phrase-structure tree
- Head-dependent relations: good proxy for semantic relations
  → dependency grammars are currently more common than constituency grammars

Word Order

- Ability to deal with free word order
- an *object* might occur before or after a *location adverbial* → separate rules in a phrase-structure grammar for each possibility
- dependency-based approach: link type representing the relation

→ abstract away from word order information

Dependency Relations

- **Grammatical Relations** provide basis for dependency structures
- Arguments of grammatical relations: **head** and **dependent**
  → directly linking heads to words that are immediately dependent on them
- Kind of grammatical relation: grammatical function
  - subject
  - direct object
  - indirect object
- Cross-linguistic standards for grammatical relations
- **Universal Dependencies (UD)** project de Marneffe et al. (2021)
  - open community effort to annotate dependencies
  - across more than 100 languages

Dependency Relations: UD

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

Figure 19.2 Some of the Universal Dependency relations (de Marneffe et al., 2021).

Dependency Relations: UD

Relation	Examples with *head* and **dependent**
NSUBJ	**United** *canceled* the flight.
OBJ	United *diverted* the **flight** to Reno.
	We *booked* her the first **flight** to Miami.
IOBJ	We *booked* **her** the flight to Miami.
COMPOUND	We took the **morning** *flight*.
NMOD	*flight* to **Houston**.
AMOD	Book the **cheapest** *flight*.
APPOS	*United*, a **unit** of UAL, matched the fares.
DET	**The** *flight* was canceled.
	Which *flight* was delayed?
CONJ	We *flew* to Denver and **drove** to Steamboat.
CC	We flew to Denver **and** *drove* to Steamboat.
CASE	Book the flight **through** *Houston*.

Figure 19.3 Examples of some Universal Dependency relations.

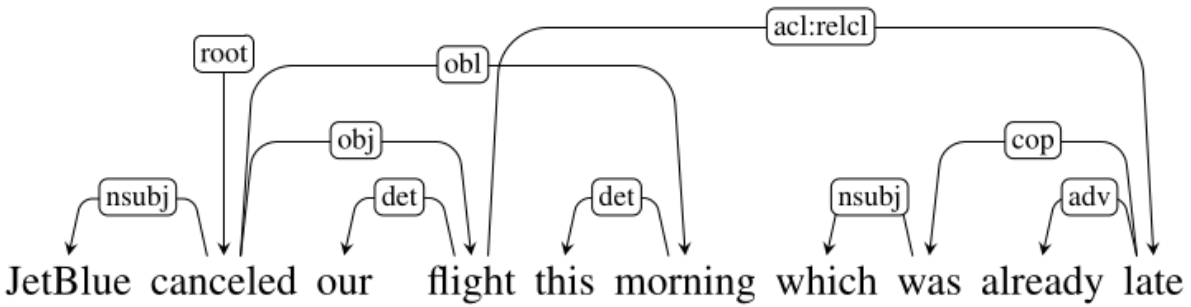
Outline

Dependency Formalisms

- Dependency structure: directed graph $G = (V, A)$
  - a set of vertices V
  - a set of ordered pairs of vertices A (arcs)
- V corresponds to words in the sentence
- A captures the head-dependent and grammatical function relationships between the elements in V
- **Dependency tree**: a directed graph
  - there is a single designated root node that has no incoming arcs
  - with the exception of the root node, each vertex has exactly one incoming arc
  - there is a unique path from the root node to each vertex in V

Projectivity

- **Projective**: there is a path from the head to every word that lies between the head and the dependent
- All dependency trees so far have been projective
- Many valid constructions can lead to non-projective trees



(19.3)

- A dependency tree is projective: it can be drawn with no crossing edges
- Computational limitations to many widely used parsing algorithms

Outline

Transition-based parsing

- Dependency parsing with **transition-based parsing**
  - **stack**: where we build the parse
  - **buffer**: tokens to be parsed
  - **oracle**: decision about next step in building the parse
- Parser walks through the sentence from left to right
  - shift items from buffer to stack
  - examine the top two words on the stack: what transition to apply
- Transition possibilities
  - assign the current word as the head of a previously seen word
  - assign a previously seen word as the head of the current word
  - postpone dealing with the current word, storing it for later processing

Transition-based parsing

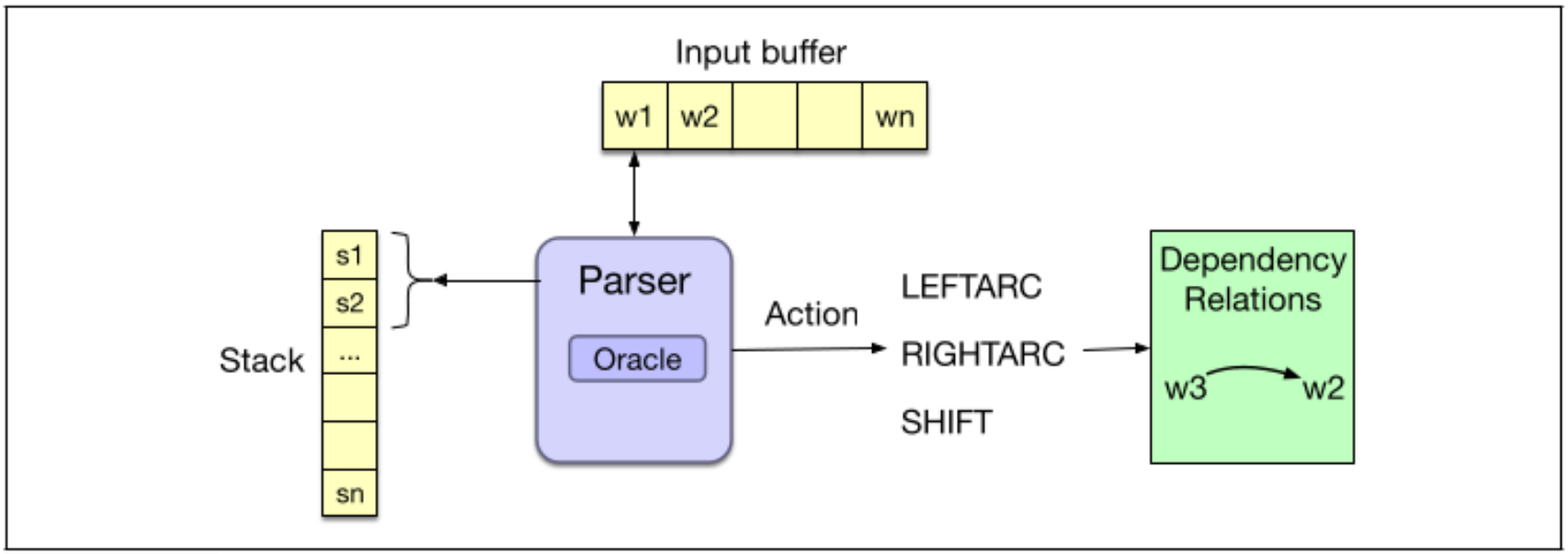


Figure 19.4 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action by consulting an oracle that examines the current configuration.

Transitions

- LEFTARC: assert a head-dependent relation between the word at the top of the stack and the second word; remove the second word from the stack
- RIGHTARC: assert a head-dependent relation between the second word on the stack and the word at the top; remove the top word from the stack
- SHIFT: Remove the word from the front of the input buffer and push it onto the stack
- **Configuration**: represents the current state of the parse
  stack, input buffer of words, and a set of relations representing a dependency tree

Algorithm

function DEPENDENCYPARSE(*words*) **returns** dependency tree

state ← {[root], [*words*], [] } ; initial configuration
while *state* **not final**
t ← ORACLE(*state*) ; choose a transition operator to apply
state ← APPLY(*t*, *state*) ; apply it, creating a new state
return *state*

Figure 19.5 A generic transition-based dependency parser

- Initial configuration: stack: ROOT node; buffer: all tokens; and an empty set of relations
- Final configuration:
  stack and buffer: empty; the set of relations represents the parse.
- Consulting an **oracle**: → comes later!
  provides the correct transition operator given the current configuration

Algorithm: Some Notes

Complexity: linear in the length of the sentence

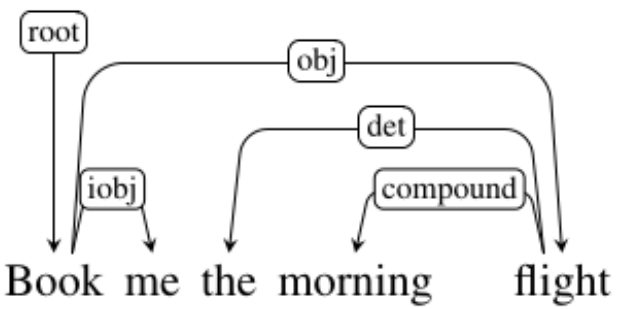
single left-to-right pass: shift and reduce

- Straightforward greedy algorithm
  - oracle provides a single choice
  - no other options explored, no backtracking
  - a single parse is returned
- Assumption that the oracle always provides the correct operator
  - likely not true in practice
  - incorrect choices → incorrect parses
- There are techniques to explore the search space more fully → book

Transition-Based Parsing: Example

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

Figure 19.6 Trace of a transition-based parse.



(19.7)

To produce labeled dependency trees: parameterize LEFTARC and RIGHTARC with dependency labels such as LEFTARC(NSUBJ)

Transition-Based Parsing: Example

Stack	Word List	Relations
[root, book, me]	[the, morning, flight]	

RIGHTARC assigns book as the head of *me* and pops *me* from the stack

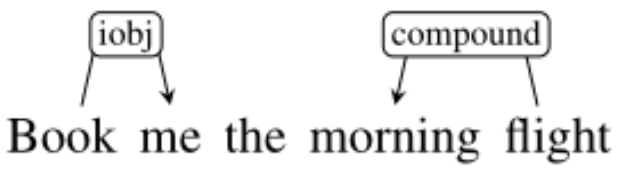
Stack	Word List	Relations
[root, book]	[the, morning, flight]	(book → me)

several subsequent applications of the SHIFT operator

Stack	Word List	Relations
[root, book, the, morning, flight]	[]	(book → me)

All the remaining words have been passed onto the stack.
Apply LEFTARC operator

Stack	Word List	Relations
[root, book, the, flight]	[]	(book → me) (morning ← flight)



Creating an Oracle: Overview

- Oracle is trained by supervised machine learning
- Supervised machine learning → we need training data: configurations annotated with the correct transition to take
- Extract features from the configurations to train a classifier
  - manually designed features, for example feature templates including word forms, lemmas, parts of speech in combination with dependency relations (cf. last lecture)
  - neural classifiers that represent the configuration via embeddings

Creating an Oracle: Generating Training Data

- Oracle takes as input a configuration and returns a transition operator
- we cannot get this directly from parsed trees in a tree bank
- To generate training data:
  - use training sentences with their corresponding reference parses from a treebank
  - simulate the operation of the parser to give us correct transition operators for each successive configuration
- Gold standard reference parse for training sentences: we know which dependency relations are valid

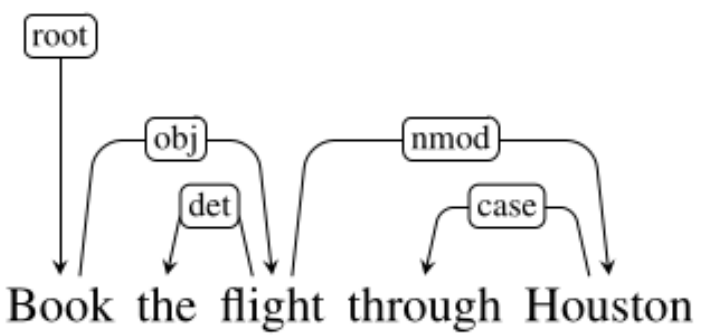
Creating an Oracle: Generating Training Data

- Reference parse to guide the selection of operators
- Given a reference parse and a configuration, the training oracle proceeds as follows:
  - Choose LEFTARC if it produces a correct head-dependent relation given the reference parse and the current configuration
  - Otherwise, choose RIGHTARC if (1) it produces a correct head-dependent relation given the reference parse and (2) all of the dependents of the word at the top of the stack have already been assigned
  - Otherwise, choose SHIFT
- Restriction on RIGHTARC: to ensure that a word is not popped from the stack before all its dependents have been assigned to it

Creating an Oracle: Generating Training Data

Step	Stack	Word List	Predicted Action
0	[root]	[book, the, flight, through, houston]	SHIFT
1	[root, book]	[the, flight, through, houston]	SHIFT
2	[root, book, the]	[flight, through, houston]	SHIFT
3	[root, book, the, flight]	[through, houston]	LEFTARC
4	[root, book, flight]	[through, houston]	SHIFT
5	[root, book, flight, through]	[houston]	SHIFT
6	[root, book, flight, through, houston]	[]	LEFTARC
7	[root, book, flight, houston]	[]	RIGHTARC
8	[root, book, flight]	[]	RIGHTARC
9	[root, book]	[]	RIGHTARC
10	[root]	[]	Done

Figure 19.7 Generating training items consisting of configuration/predicted action pairs by simulating a parse with a given reference parse.



Creating an Oracle: Generating Training Data

- Step 1:
  - LEFTARC is not applicable: (root ← book) is not in the reference
  - RIGHTARC: (root → book) is a valid relation, but *book* has not been attached to its dependents
  - defer to SHIFT

- Step 3: LEFTARC to link *the* to its head *flight*

Stack	Word buffer	Relations
[root, book, flight]	[through, Houston]	(the ← flight)

- Step 4:
  Add relation between *book* and *flight*?
  This would remove *flight* from the stack and prevent the later attachment of *Houston*

Feature-Based Classifier

- Feature templates to extract features from training configurations
- Some features: word forms, lemmas, parts of speech
- Template:

$$\langle s_1.w, op\rangle, \langle s_2.w, op\rangle \langle s_1.t, op\rangle, \langle s_2.t, op\rangle$$
$$\langle b_1.w, op\rangle, \langle b_1.t, op\rangle \langle s_1.wt, op\rangle$$

- Configuration from training oracle (next operation is SHIFT):

Stack	Word buffer	Relations
[root, canceled, flights]	[to Houston]	(canceled → United)
		(flights → morning)
		(flights → the)

- Apply feature templates:

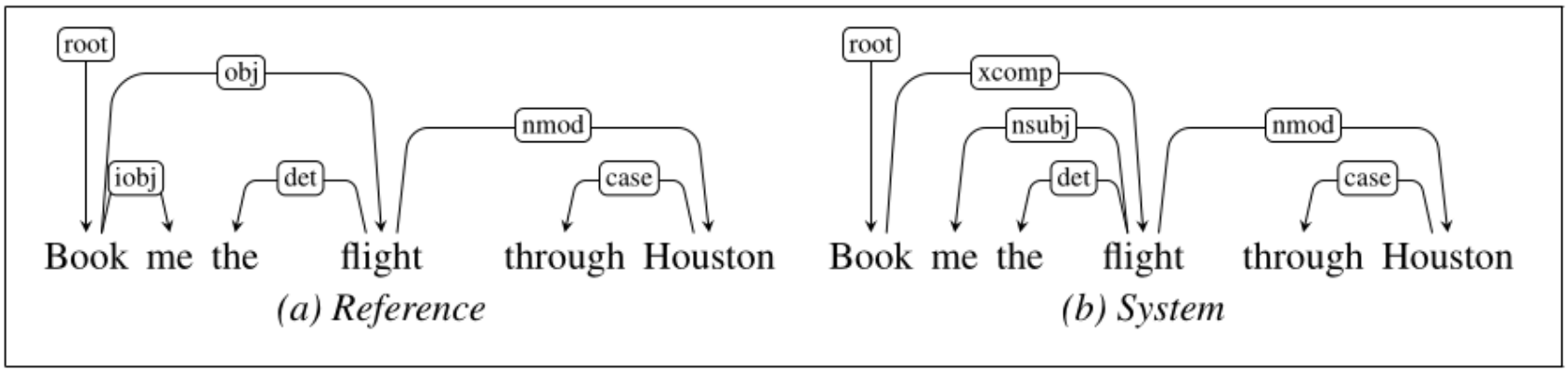
$$\langle s_1.w = flights, op = shift\rangle$$
$$\langle s_2.w = canceled, op = shift\rangle$$
$$\langle s_1.t = NNS, op = shift\rangle$$
$$\langle s_2.t = VBD, op = shift\rangle$$
$$\langle b_1.w = to, op = shift\rangle$$
$$\langle b_1.t = TO, op = shift\rangle$$
$$\langle s_1.wt = flightsNNS, op = shift\rangle \quad \langle s_1.t \circ s_2.t = NNSVBD, op = shift\rangle$$

Outline

Evaluation

- Exact match: quite pessimistic, not fine-grained enough
- Evaluating dependency parsers: **labeled** and **unlabeled attachment accuracy**
  - labeled attachment: proper assignment of a word to its head along with the correct dependency relation
  - unlabeled attachment: correctness of the assigned head, ignoring the dependency relation
- LAS: Labeled attachment score
  UAS: Unlabeled attachment score
  LAS: Label Accuracy score (percentage of token with correct labels)

Evaluation



- LAS: 4 of 6 dependency relations: LAS $= 2/3$
- UAS: 5/6 (relation between *book* and *flight* is a head-dependent relation in the reference)

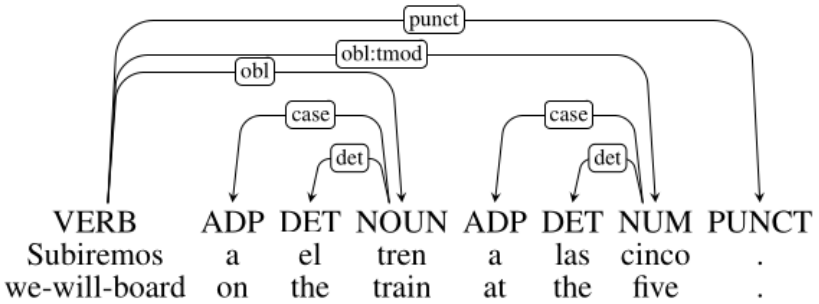
- Beyond attachment scores:
  how well is a system for a particular kind of dependency relation?

Outline

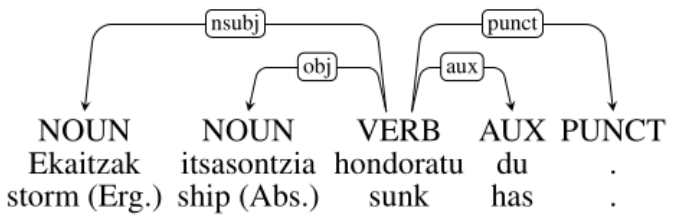
Dependency Treebanks

- Treebanks: relevant for development and evaluation of dependency parsing
  - parser training → gold labels
  - information for corpus linguistics
- Manual annotation: directly creating dependency structures or correction of a parser
- Universal Dependency project: https://universaldependencies.org
  treebanks for more than 100 languages

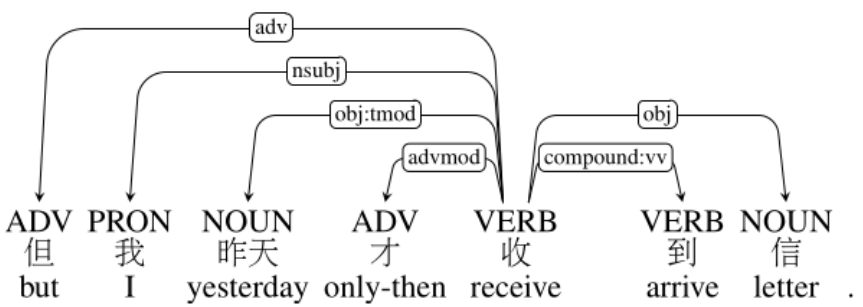
UD Treebank Examples



[Spanish] Subiremos al tren a las cinco. "We will be boarding the train at five." (19.4)



[Basque] Ekaitzak itsasontzia hondoratu du. "The storm has sunk the ship." (19.5)



[Chinese] 但我昨天才收到信 "But I didn't receive the letter until yesterday." (19.6)

Outline

Credits

The slides contain content and examples from

- *Speech and Language Processing*
  (Jurafsky and Martin): Chapters 18 and 19
- *Foundations of Statistical Natural Language Processing*
  (Manning and Schütze): Chapters 11 and 12
- *Natural Language Processing*
  (Eisenstein): Chapter 10