Natural Language Processing

Assist. Prof. Dr. Tuğba YILDIZ

İSTANBUL BİLGİ UNIVERSITY
Department of Computer Engineering

March 31, 2017

Syntax

- Syntax is the study of formal relationships between words
- Syntax refers to the way that words are arranged together.
- Last week,
  - how words are clustered into classes called part-of-speech (POS)
  - tagsets for POS
  - methods of POS Tagging

Syntax

- In this week,
  - how they group with their neighbors into phrases
  - context-free grammars
  - parsing

Syntax

- There are three main new ideas:
  - constituency
  - grammatical relations
  - subcategorization and dependencies

Constituency

- **constituency**: groups of words may behave as a single unit or phrase
- noun phrase or noun groups : This is a sequence of words surrounding at least one noun.
  - three parties from Brooklyn
  - a high-class spot such as Mindy's
  - the Broadway coppers
  - they

Constituency

- One piece of **evidence** is that they can all appear in similar syntactic environments, for example **before a verb.**
  - three parties from Brooklyn arrive
  - a high-class spot such as Mindy's attract
  - the Broadway coppers love
  - they sit

Constituency

- context-free grammars are also called Phrase-Structure Grammars
- a context-free grammar(CFG) consists of a set of rules or productions
- each rule expresses the ways that symbols of the language can be grouped and ordered together, and a lexicon of words and symbols.

Constituency

- for example, the following productions expresses that a **NP** (or noun phrase), can be composed of either a **ProperNoun** or of a **Det** (determiner) followed by a **Nominal**
- a **Nominal** can be one or more Nouns.
- NP $\rightarrow$ Det Nominal
- NP $\rightarrow$ ProperNoun
- Nominal $\rightarrow$ Noun $|$ Noun Nominal

Constituency

- Context free rules can be hierarchically embedded, so we could combine the previous rule with others
- Det $\rightarrow$ a
- Det $\rightarrow$ the
- Noun $\rightarrow$ flight

Context-Free Grammar

- the symbols that are used in a CFG are divided into two classes:
  1. terminal symbols:
     - the symbols that correspond to words in the language (“the”, “club”) are called terminal symbols
     - the lexicon is the set of rules that introduce these terminal symbols
  2. nonterminal symbols:
     - the symbols that express clusters or generalizations of these are called nonterminals
- in each context-free rule, the item to the right of the arrow is an ordered list of one or more terminals and nonterminals
- while to the left of the arrow is a single nonterminal symbol expressing some cluster or generalization

Context-Free Grammar

- a CFG is usually thought of in two ways:
  1. as a device for **generating sentences**
  2. as a device for **assigning a structure** to a given sentence.
- as a generator, we could read the arrow as “rewrite the symbol on the left with the string of symbols on the right”
- rewrite NP as Det Nominal
- Det Nominal
- rewrite Nominal as Noun
- Det Noun
- a flight

Context-Free Grammar

- we say the string **a flight** can be derived from the nonterminal NP
- Thus a CFG can be used to randomly generate a series of strings
- This sequence of rule expansions is called a derivation of the string of words
- It is common to represent a derivation by a parse tree

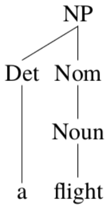

Fig.1 Parse tree of a flight

Context-Free Grammar

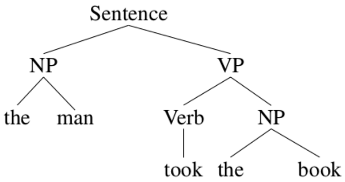

Fig 2. The first context-free grammar parse tree (Chomsky, 1956)

Context-Free Grammar

- the formal language defined by a CFG is the set of strings that are derivable from the designated start symbol.
- each grammar must have one designated start symbol, which is often called S.
- since context-free grammars are often used to define sentences, S is usually interpreted as the “sentence” node

Context-Free Grammar

- S $\rightarrow$ NP VP : I prefer a morning flight
- A verb phrase in English consists of a verb followed by assorted other things
  - VP $\rightarrow$ Verb NP : prefer a morning flight
- Or the verb phrase may have a noun phrase and a prepositional phrase
  - VP $\rightarrow$ Verb NP PP : leave Boston in the morning
- Or the verb may be followed just by a preposition-phrase
  - VP $\rightarrow$ Verb PP : leaving on Thursday

Context-Free Grammar

- A prepositional phrase generally has a preposition followed by a noun phrase.
- For example, a very common type of prepositional phrase in the ATIS corpus is used to indicate location or direction:
  - PP $\rightarrow$ Preposition NP : from Los Angeles

Context-Free Grammar

$$
\begin{aligned}
Noun &\rightarrow flights \mid breeze \mid trip \mid morning \mid \ldots \\
Verb &\rightarrow is \mid prefer \mid like \mid need \mid want \mid fly \\
Adjective &\rightarrow cheapest \mid non-stop \mid first \mid latest \\
&\quad \mid other \mid direct \mid \ldots \\
Pronoun &\rightarrow me \mid I \mid you \mid it \mid \ldots \\
Proper\text{-}Noun &\rightarrow Alaska \mid Baltimore \mid Los\ Angeles \\
&\quad \mid Chicago \mid United \mid American \mid \ldots \\
Determiner &\rightarrow the \mid a \mid an \mid this \mid these \mid that \mid \ldots \\
Preposition &\rightarrow from \mid to \mid on \mid near \mid \ldots \\
Conjunction &\rightarrow and \mid or \mid but \mid \ldots
\end{aligned}
$$

Fig.3 The lexicon for L0 .

Context-Free Grammar

S	$\rightarrow$	$NP\ VP$	I + want a morning flight
NP	$\rightarrow$	$Pronoun$	I
	$\mid$	$Proper\text{-}Noun$	Los Angeles
	$\mid$	$Det\ Nominal$	a + flight
Nominal	$\rightarrow$	$Noun\ Nominal$	morning + flight
	$\mid$	$Noun$	flights
VP	$\rightarrow$	$Verb$	do
	$\mid$	$Verb\ NP$	want + a flight
	$\mid$	$Verb\ NP\ PP$	leave + Boston + in the morning
	$\mid$	$Verb\ PP$	leaving + on Thursday
PP	$\rightarrow$	$Preposition\ NP$	from + Los Angeles

Fig. 4 The grammar for L0 with example phrases for each rule.

Context-Free Grammar

- We can use this grammar to generate sentences
  "I prefer a morning flight"
- We start with S, expand it to NP VP,
- then choose a random expansion of NP
- (let's say to I), and a random expansion of VP (let's say to Verb NP),
- and so on until we generate the string I prefer a morning flight.

Context-Free Grammar

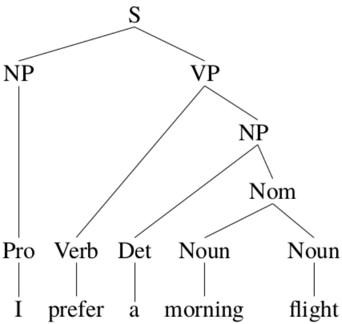

Fig. 5 The parse tree for ‘I prefer a morning flight’ according to grammar L0

Context-Free Grammar

- it is sometimes convenient to represent a parse tree in a more compact format called bracketed notation

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

Fig. 6 The parse tree for 'I prefer a morning flight' according to grammar L0

Context-Free Grammar

- a CFG like that of L0 defines a formal language
- a formal language is a set of strings (baa!, baaa! ...)
- sentences (strings of words) that can be derived by a grammar are in the formal language defined by that grammar and are called grammatical sentences.
- sentences that cannot be derived by a given formal grammar are not in the language defined by that grammar and are referred to as ungrammatical

Context-Free Grammar

- We conclude this section by way of summary with a quick formal description of a CFG and the language it generates.
- A CFG has four parameters (technically 'is a 4-tuple'):
  1. a set of non-terminal symbols (or "variables") N
  2. a set of terminal symbols $\sum$ (disjoint from N)
  3. a set of productions P, each of the form A $\rightarrow \alpha$, where A is a non-terminal and α is a string of symbols from the infinite set of strings $(\sum \cup N)*$.
  4. a designated start symbol S
- A language is defined via the concept of derivation.
- One string derives another one if it can be rewritten as the second one via some series of rule applications.

Sentence Level Constructions

- Sentence Level Constructions
- there are 4 particularly structures common and important:
    - declarative structure
    - imperative structure
    - yes-no question structure
    - wh-question structure

Sentence Level Constructions

- Sentence Level Constructions
- there are 4 particularly structures common and important:
  - **declarative structure :** have a subject noun phrase followed by a verb phrase, like
    "I prefer a morning flight"
    S → NP VP
  - **imperative structure :** often begin with a verb phrase, and have no subject, like
    "Show the lowest fare"
    S → VP
  - **yes-no question structure:** used to ask questions, and begin with a auxiliary verb, followed by a subject NP, followed by a VP, like
    "Do any of these flights have stops¿'
    S → Aux NP VP
  - **wh-subject-question / wh-nonsubject-question**
    S → Wh-NP VP S → Wh-NP Aux NP VP

Noun Phrases

- Noun Phrases:
  - Each noun phrase has a head noun : a book
  - A noun phrase the head noun may be preceded by pre-nominal modifiers and followed by post-nominal modifiers
  - Pre-Nominal Modifiers:
    - Determiner : a, the, that, this, any, some / a book
    - mass-nouns do not require determiners
    - Pre-Determiners : all / all the flights, all flights
    - Cardinal Numbers : one, two / two friends, one man
    - Ordinal Numbers : first,second,next,last,other / the last flight
    - Quantifiers : many,several,few / many fares
    - Adjective Phrases : the least expensive fare
  - A simplified rule:
  - NP $\rightarrow$ (PreDet) (Det) (Card) (Ord) (Quan) (AP) NOM

Noun Phrases

- Noun Phrases:
- Post-Nominal Modifiers:
  - Three common post-modifiers:
  - prepositional phrases : all flights from Ankara
  - relative clauses : a flight that serves dinner
  - non-finite clauses : any flight arriving after 5 p.m.
    - three common non-finite post-modifiers: gerundive, -ed, and infinitive forms.
- NOM→ NOM PP (PP) (PP)
- NOM → NOM GerundVP
- NOM → NOM RelClause
- GerundVP → GerundV | GerundV NP | GerundV PP | GerundV NP PP
- GerundV → arriving | preferring |
- RelClause → who VP | that VP

Coordination

- a coordinate noun phrase can consist of two other noun phrases separated by a conjunction
- Conjunctions:
- noun phrases and other phrases can be conjoined with conjunctions such as and, or, but,
- **table** and **chair**
- the flights that **leaving Ankara** and **arriving in Istanbul**
- **he came from Ankara** and **he went to Istanbul**
- NP $\rightarrow$ NP and NP
- VP $\rightarrow$ VP and VP
- S $\rightarrow$ S and S

Difficulties

- some Difficulties in Grammar Development:
- agreement:
- what flights leave vs What flight leaves
- he flies vs he fly
- I fly vs I flies
- this book vs this books
- those books vs those book

Difficulties

- how can we modify our grammar to handle these agreement phenomena?
- one way is to expand our grammar with multiple sets of rules, one rule set for 3sg subjects, and one for non-3sg subjects.
- S $\rightarrow$ Aux NP VP
- we could replace this with two rules of the following form:
  S $\rightarrow$ 3sgAux 3sgNP VP
  S $\rightarrow$ Non3sgAux Non3sgNP VP
- We could then add rules for the lexicon like these:
  3sgAux $\rightarrow$ does | has | can | ...
  Non3sgAux $\rightarrow$ do | have | can | ...

Difficulties

- a problem with this method of dealing with number agreement is that it doubles the size of the grammar.
- for example, every rule that refers to a noun or a verb needs to have a 'singular' version and a 'plural' version.

Verb Phrase and Subcategorization

- Verb Phrase and Subcategorization
- the verb phrase consists of the verb and a number of other constituents.
- subcategorization and dependency relations refer to certain kinds of relations between words and phrases.
- for example, the verb **want** can be followed by an infinitive
  I want to fly to Detroit
- the verb **want** can be followed a noun phrase
  I want a flight to Detroit
- These are called facts about the subcategory of the verb

Verb Phrase and Subcategorization

- 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 Phrase and Subcategorization

- Although a verb phrase can have many possible of constituents, not every verb is compatible with every verb phrase.
- Verbs have preferences for the kinds of constituents they co-occur with.
- I disappeared the cat. (disappear cannot be followed by a noun phrase)

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

Fig.7 Subcategorization frames

Auxiliary

- Auxiliary:
- The subclass of verbs called auxiliaries or helping verbs have particular syntactic constraints which can be viewed as a kind of subcategorization.
- Auxiliaries include the modal verbs can, could, may, might, must, will, would, shall, and should,
- the perfect auxiliary have
- the progressive auxiliary be

Recursion

- recursion in a grammar occurs when an expansion of a non-terminal includes the non-terminal itself
- recursive rules may appear in our grammars.
- NP $\rightarrow$ NP PP : the flight from Ankara
- VP $\rightarrow$ VP PP : departed Ankara at 5 p.m.
- these rules allow us the following:
- flights to Ankara
- flights to Ankara from Istanbul
- flights to Ankara from Istanbul in March
- flights to Ankara from Istanbul in March on Friday
- flights to Ankara from Istanbul in March on Friday under $100
- flights to Ankara from Istanbul in March on Friday under $100 with lunch

Parsing

- parsing (Syntactic Parsing) is the combination of recognizing an input string and assigning some structure to it.
- in syntactic parsing, the parser can be viewed as searching through the space of all possible parse trees to find the correct parse tree for the sentence.

Parsing

- searching is imporatant!
- the goal of a parsing search is to find all trees whose root is the start symbol S, which cover exactly the words in the input.

Parsing

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

Fig. 8 A miniature English grammar and lexicon

Parsing

- there are clearly two kinds of constraints that should help guide the search.
- **one kind of constraint** comes from the data, i.e. the input sentence itself (book that flight)
- we know that there must be three leaves, and they must be the words book, that, and flight
- **the second kind of constraint** comes from the grammar
- we know that whatever else is true of the final parse tree, it must have one root, which must be the start symbol S

Parsing

- these two constraints give rise to the two strategies underlying most parsers:
- top-down or goal-directed search
- bottom-up or data-directed search

Top-down Parsing

- A top-down parser searches for a parse tree by trying to build from the root node S down to the leaves.
- it builds all possible trees in parallel
- The algorithm starts by assuming the input can be derived by the designated start symbol S.
- The next step is to find the tops of all trees which can start with S, by looking for all the grammar rules with S on the left-hand side.

Top-down Parsing

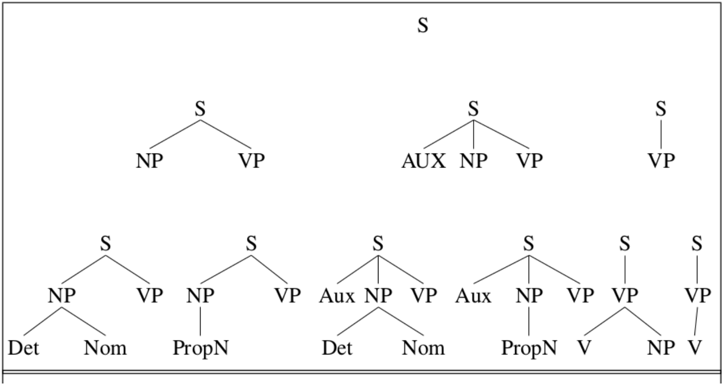

Fig.9 Top-down search space for the sentence "Book that flight"

Bottom-Up Parsing

- the parser starts with the words of the input and tries to build trees from the words up
- again by applying rules from the grammar one at a time
- the parse is successful if the parser succeeds in building a tree rooted in the start symbol S that covers all of the input

Bottom-Up Parsing

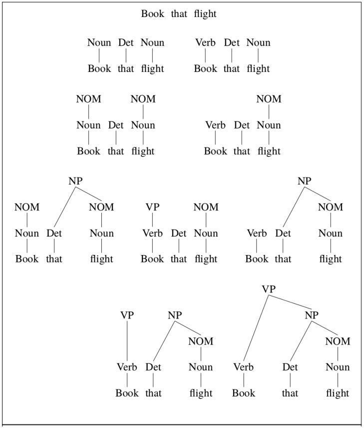

Fig. 10 Bottom-up search space for the sentence "Book that flight"

Comparing Top-down vs. Bottom-Up Parsing

- each of these two architectures has its own advantages and disadvantages
- **the top-down strategy** never wastes time exploring trees that cannot result in an S, since it begins by generating just those trees.
- this means it also never explores subtrees that cannot find a place in some S-rooted tree.
- **in the bottom-up strategy**, by contrast, trees that have no hope of leading to an S, or fitting in with any of their neighbors
- for example the left branch of the search space is completely wasted effort in Fig. 10

Comparing Top-down vs. Bottom-Up Parsing

- **the top-down approach** has its own inefficiencies.
- while it does not waste time with trees that do not lead to an S, it does spend considerable effort on S trees that are not consistent with the input.
- note that the first four of the six trees all have left branches that cannot match the word book.
- none of these trees could possibly be used in parsing this sentence.
- this weakness in top-down parsers arises from the fact that they can generate trees before ever examining the input.
- **solution:** incorporates features of both the top-down and bottom-up approaches.

Combining Top-down vs. Bottom-Up Parsing

- there are any number of ways of combining the best features of top-down and bottom-up parsing into a single algorithm.
- one fairly straightforward approach is to adopt one technique as the primary control strategy used to generate trees
- and then use constraints from the other technique to filter out inappropriate parses on the fly
- the parser we develop in this section uses **a top-down control strategy** augmented with a bottom-up filtering mechanism.

Top-down Depth-First Left-to-Right Parsing

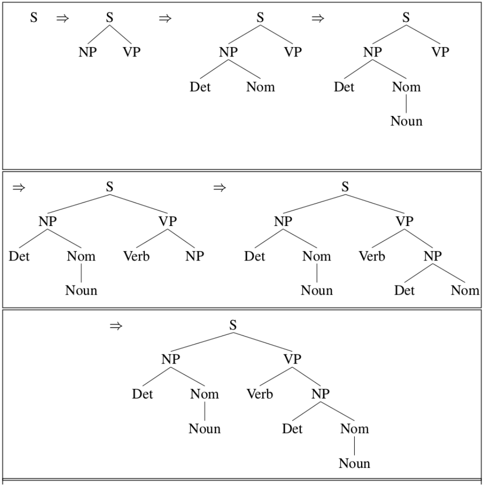

Fig.11 Top-down depth-first left-to-right parse tree

Top-down Depth-First Left-to-Right Parsing

- Does this flight include a meal?

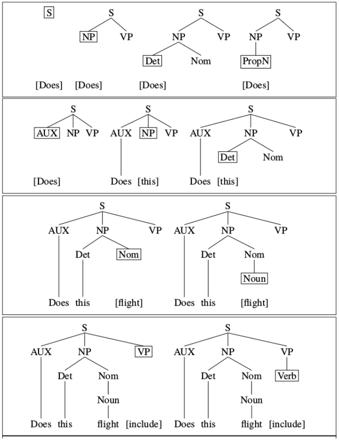

Fig. 12 Top-down depth-first left-to-right parse tree

Top-down Depth-First Left-to-Right Parsing

- Does this flight include a meal?

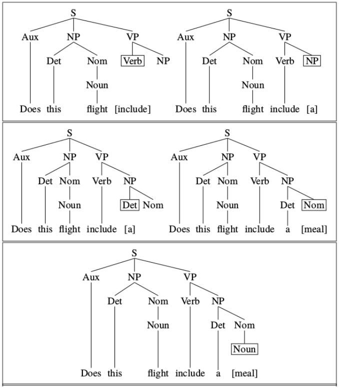

Fig. 13 Top-down depth-first left-to-right parse tree

Top-Down Parsing with Bottom-Up Filtering

- When we choose applicable rules, we can use bottom-up information.
- For example, in our grammar we have:
- S $\rightarrow$ NP VP
- S $\rightarrow$ Aux NP VP
- S $\rightarrow$ VP
- If we want to parse the input:
- Does this flight serve a meal?
- Although all three of these rules are applicable, the first and the third ones will definitely fail because NP and VP cannot derive to strings starting with **does (an auxiliary verb here)**
- Can we make this decision before we choose an applicable rule?
- Yes. We can use **left-corner filtering**

Filtering with Left Corners

- The parser should not consider any grammar rule if the current input serve as the first word along the left edge of some derivation from this rule.
- the first word along the left edge of a derivation is called as the left-corner of the tree.
- B is a left-corner of A if the following relation holds:
- $A \implies ^* B\alpha$
- In other words, B can be the left-corner of A if there is a derivation of A that begins with B.

Filtering with Left Corners

- Does this flight include a meal?
- three rules:
- S $\rightarrow$ NP VP
- S $\rightarrow$ Aux NP VP
- S $\rightarrow$ VP
- using the left-corner notion, it is easy to see that only the S $\rightarrow$ Aux NP VP rule is a candidate
  since the word Does can not serve as the left-corner of either the NP or the VP required by the other two S rules.

Category	Left Corners
S	Det, Proper-Noun, Aux, Verb
NP	Det, Proper-Noun
Nominal	Noun
VP	Verb

Fig. 14 Top-down depth-first left-to-right parse tree

Problems with basic top-down parser

- three problems are:
- left-recursion
- ambiguity
- inefficient reparsing of subtrees

Left recursion

- When left-recursive grammars are used, top-down depth-first left-to-right parsers can dive into an infinite path.
- A grammar is left-recursive if it contains at least one non-terminal A such that:
- $A \implies {}^* A\alpha$
- This kind of structures are common in natural language grammars.
- NP $\rightarrow$ NP PP
- We can convert a left-recursive grammar into an equivalent grammar which is not left-recursive.

  $A \rightarrow A\alpha \mid \beta$

  $A \rightarrow \beta A'$

  $A' \rightarrow \alpha A' \mid \epsilon$
- Unfortunately, the resulting grammar may no longer be the most grammatically natural way to represent syntactic structures.

Left Recursion

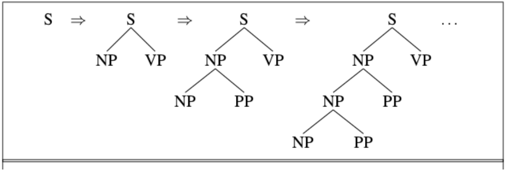

Fig. 15 top-down depth-first left-to-right parse tree

Ambiguity

- One morning I shot an elephant in my pajamas. How he got into my pajamas I don't know. (Groucho Marx, Animal Crackers, 1930)
- not efficient at handling ambiguity
- structural ambiguity
  - attachment ambiguity
  - coordination ambiguity
  - noun-phrase bracketing ambiguity

Left Recursion

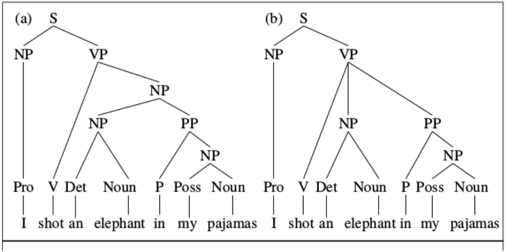

Fig. 16 Two parse trees for an ambiguous sentence.

Ambiguity

- a particular constituent can be attached to the parse tree at more than one place.
- PP-attachment ambiguity in example
- coordination ambiguity, in which there are different sets of phrases that can be conjoined by a conjunction like and.
- old men and women

Inefficient reparsing of subtrees

- the parser often builds valid trees for portion of the input,
- then discards them during backtracking, only to find that it has to rebuild them again.
- the parser creates small parse trees that fail because they do not cover all the input.
- the parser backtracks to cover more input, and recreates subtrees again and again.
- the same thing is repeated more than once unnecessarily.

Inefficient reparsing of subtrees

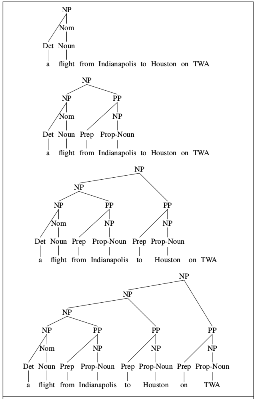

Fig. 17 Reduplicated effort caused by backtracking in top-down parsing

Dynamic Programming

- We want a parsing algorithm (using dynamic programming technique) that fills a table with solutions to subproblems that:
  - Does not do repeated work
  - Does top-down search with bottom-up filtering
  - Solves the left-recursion problem
  - Solves an exponential problem in O(N^3) time.
- The answer is Earley Algorithm.

Early Algorithm

- Next Week!
- NLTK!

References

- Speech and Language Processing (3rd ed. draft) by D. Jurafsky & J. H. Martin (web.stanford.edu)