Introduction to Syntactic Analysis

고려대학교
컴퓨터학과
임희석

KOREA UNIVERSITY GLOBAL PRIDE

Index

- Grammar
  - Context-Free Grammars
  - Grammars for English
- Method for the syntactic analysis
  - Bottom-up
  - Top-down
  - Ambiguity
  - CKY parsing

What is the Syntax

- Set of rules for arranging words into meaningful sentences
- Set of rules, principles, and processes that govern the structure of sentences in a given language, specifically word order
- Grammars are key components in many applications
  - Grammar checkers
  - Dialogue management
  - Question answering
  - Information extraction
  - Machine translation
  - ….

Grammar

Constituency

- The basic idea is that groups of words within utterances can be shown to act as single units.

- In a given language, these units form coherent classes that can be shown to behave in similar ways
  - ✓ With respect to their internal structure
  - ✓ And with respect to other units in the language

Constituency

- Internal structure
  - ✓ We can describe an internal structure to the class

- External behavior
  - ✓ For example, we can say that noun phrases can come after verbs

Constituency

- For example, it makes sense to say that the following are all ***noun phrases*** in English...

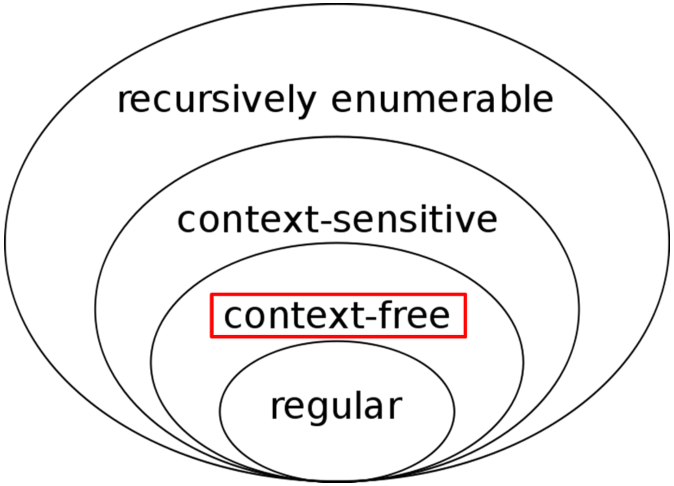
Harry the Horse	a high-class spot such as Mindy's
the Broadway coppers	the reason he comes into the Hot Box
they	three parties from Brooklyn

- Why? One piece of evidence is that verbs can precede all of them.
  - This is external evidence

Grammars and Constituency

- There's nothing easy or obvious about how we come up with right set of constituents and the rules that govern how they combine...

- That's why there are so many different theories of grammar and competing analyses of the same data.



Grammar	Languages	Automaton	Production rules (constraints)
Type-0	Recursively enumerable	Turing machine	$\alpha \rightarrow \beta$ (no restrictions)
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type-2	Context-free	Non-deterministic pushdown automaton	$A \rightarrow \gamma$
Type-3	Regular	Finite state automaton	$A \rightarrow a$ and $A \rightarrow aB$

Chomsky hierarchy

Context-Free Grammars

- Context-free grammars (CFGs)
  - Also known as
    - Phrase structure grammars
    - Backus-Naur Form (BNF)
- Consist of
  - Terminals
  - Non-terminals
  - Rules

Context-Free Grammars

- Terminals
  - We'll take these to be words (for now)
- Non-Terminals
  - The constituents in a language
  - Like noun phrase, verb phrase and sentence
- Rules
  - Rules are equations that consist of a single non-terminal on the left and any number of terminals and non-terminals on the right.

Some NP Rules

- Here are some rules for our noun phrases

$$
\begin{aligned}
NP &\rightarrow Det\ Nominal \\
NP &\rightarrow ProperNoun \\
Nominal &\rightarrow Noun \mid Nominal\ Noun
\end{aligned}
$$

- Together, these describe two kinds of NPs.
  - ✓ One that consists of a determiner followed by a nominal
  - ✓ And another that says that proper names are NPs.
  - ✓ The third rule illustrates two things
    - An explicit disjunction
      - ➢ Two kinds of nominals
    - A recursive definition
      - ➢ Same non-terminal on the right and left-side of the rule

L0 Grammar

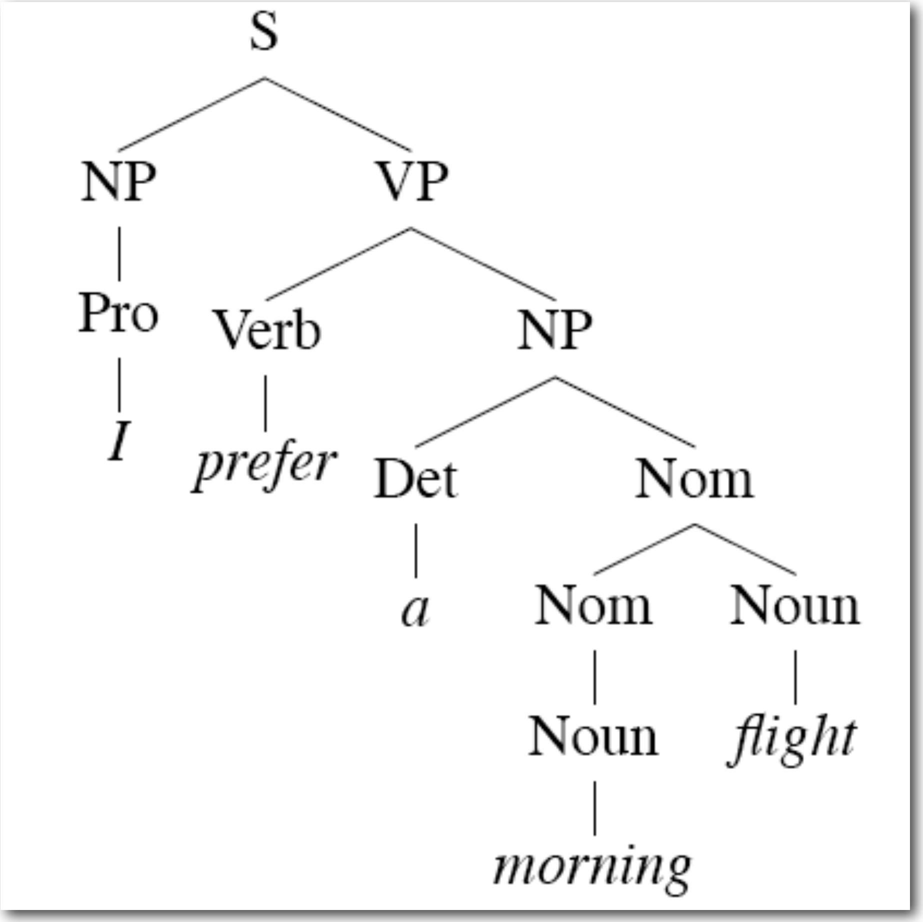
Grammar Rules			Examples
S	$\rightarrow$	$NP\ VP$	I + want a morning flight
NP	$\rightarrow$	$Pronoun$	I
	$\mid$	$Proper\text{-}Noun$	Los Angeles
	$\mid$	$Det\ Nominal$	a + flight
$Nominal$	$\rightarrow$	$Nominal\ Noun$	morning + flight
	$\mid$	$Noun$	flights
VP	$\rightarrow$	$Verb$	do
	$\mid$	$Verb\ NP$	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

Generativity

- As with FSA(Finite State Automata)s and FST(Finite State Transducers)s, you can view these rules as either analysis or synthesis machines
  - Generate strings in the language
  - Reject strings not in the language
  - Impose structures (trees) on strings in the language

Derivations

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



Definition

- More formally, a CFG consists of

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**

Parsing

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

An English Grammar Fragment

An English Grammar Fragment

- Sentences
- Noun phrases
  - ✓ Agreement
- Verb phrases
  - ✓ Subcategorization

Sentence Types

- Declaratives: *A plane left.*

  $S \rightarrow NP\ VP$
- Imperatives: *Leave!*

  $S \rightarrow VP$
- Yes-No Questions: *Did the plane leave?*

  $S \rightarrow Aux\ NP\ VP$
- WH Questions: *When did the plane leave?*

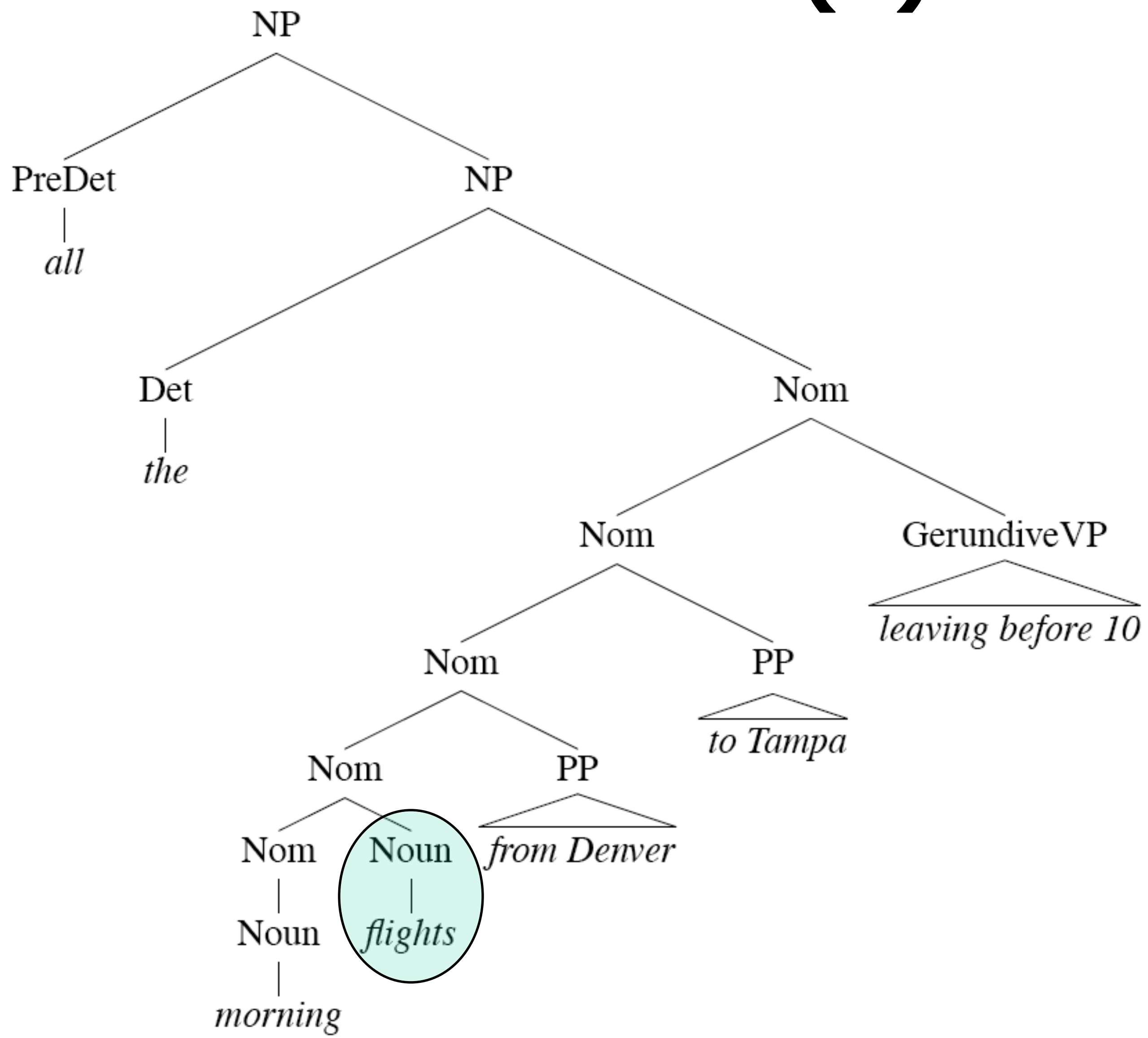
  $S \rightarrow WH\text{-}NP\ Aux\ NP\ VP$

Noun Phrases

- Let's consider the following rule in more detail...

  NP $\longrightarrow$ *Det Nominal*
- Most of the complexity of English noun phrases is hidden in this rule.
- Consider the derivation for the following example
  - ✓ *All the morning flights from Denver to Tampa leaving before 10*

Noun Phrases(2)



Noun Phrases(3)-NP Structure

- Clearly this NP is really about *flights.* That's the central critical noun in this NP. Let's call that the *head*.
- We can dissect this kind of NP into the stuff that can come before the head, and the stuff that can come after it.

Noun Phrases(4)-Determiners

- Noun phrases can start with determiners...
- Determiners can be
  - ✓ Simple lexical items: *the, this, a, an*, etc.
    - ➢ A car
  - ✓ Or simple possessives
    - ➢ John's car
  - ✓ Or complex recursive versions of that
    - ➢ John's sister's husband's son's car

Noun Phrases(5)-Nominals

- Contains the head and any pre- and post-modifiers of the head.
  - ✓ Pre-
    - Quantifiers, cardinals, ordinals...
      - Three cars
    - Adjectives and Aps
      - large cars
    - Ordering constraints
      - Three large cars
      - large three cars

Noun Phrases(6)- Nominals - Postmodifiers

- ✓ Post-
  - Three kinds
    - ✓ Prepositional phrases
      - ➢ From Seattle
    - ✓ Non-finite clauses
      - ➢ Arriving before noon
    - ✓ Relative clauses
      - ➢ That serve breakfast
- Same general (recursive) rule to handle these
  - ✓ *Nominal → Nominal PP*
  - ✓ *Nominal → Nominal GerundVP*
  - ✓ *Nominal → Nominal RelClause*

Noun Phrases(7)-Agreement

- By ***agreement***, we have in mind constraints that hold among various constituents that take part in a rule or set of rules
- For example, in English, determiners and the head nouns in NPs have to agree in their number.

This flight

Those flights

*This flights

*Those flight

Noun Phrases(8)-Problem

- Our earlier NP rules are clearly deficient since they don't capture this constraint

  NP → Det Nominal

  - ✓ Accepts, and assigns correct structures, to grammatical examples (*this flight*)
  - ✓ But, incorrect examples (*these flight)

- Such a rule is said to *overgenerate.*

Verb Phrases

- English *VPs* consist of a head verb along with 0 or more following constituents which we'll call *arguments*.

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 Phrases(2)-Subcategorization

- But, even though there are many valid VP rules in English, not all verbs are allowed to participate in all those VP rules.
- We can subcategorize the verbs in a language according to the sets of VP rules that they participate in.
- This is a modern take on the traditional notion of transitive/intransitive.
- Modern grammars may have 100s or such classes.

Verb Phrases(3)-Subcategorization

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

Verb Phrases(4)-Subcategorization

*John sneezed the book

*I prefer United has a flight

*Give with a flight

- As with agreement phenomena, we need a way to formally express the constraints

Parsing with CFGs

KOREA UNIVERSITY GLOBAL PRIDE

Parsing

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

Parsing

- As with everything of interest, parsing involves a search which involves the making of choices
- We'll start with some basic (meaning bad) methods before moving on to the one or two that you need to know

For Now

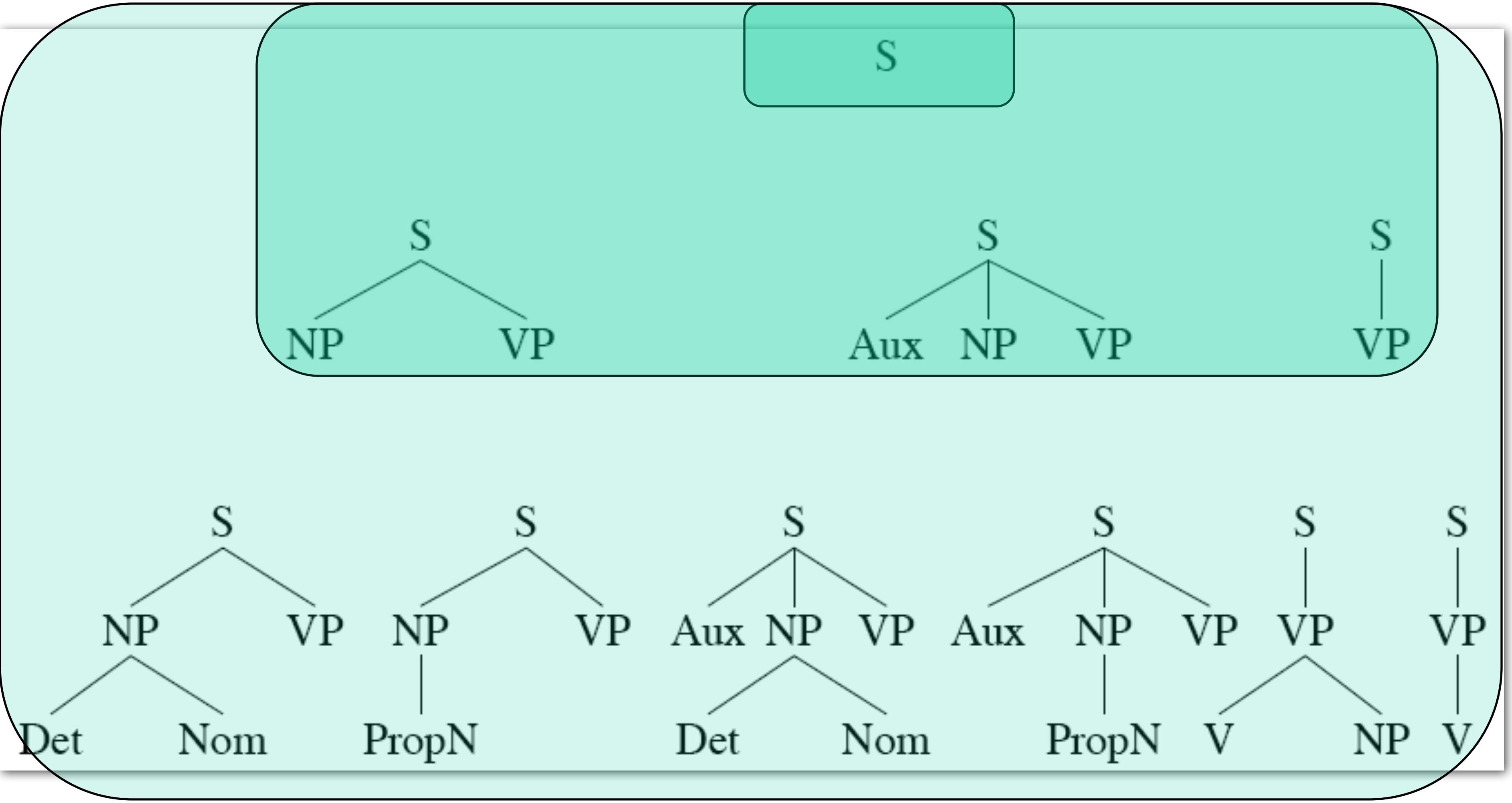
- Assume…
  - ✓ You have all the words already in some buffer
  - ✓ The input isn't POS tagged
  - ✓ We won't worry about morphological analysis
  - ✓ All the words are known

  - ✓ These are all problematic in various ways, and would have to be addressed in real applications.

Top-Down Search

- Since we're trying to find trees rooted with an *S* (Sentences), why not start with the rules that give us an *S*.
- Then we can work our way down from there to the words.

Top Down Space



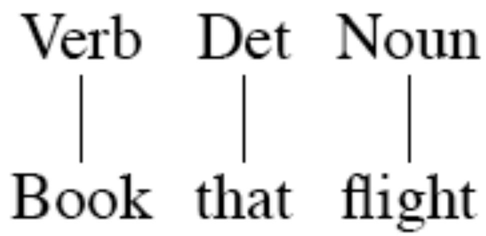
Bottom-Up Parsing

- Of course, we also want trees that cover the input words. So we might also start with trees that link up with the words in the right way.
- Then work your way up from there to larger and larger trees.

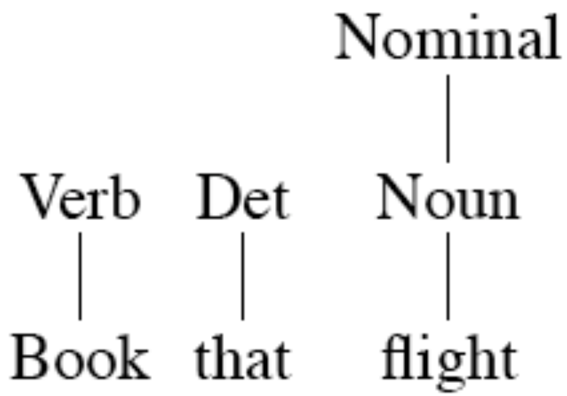
Bottom-Up Search

Book that flight

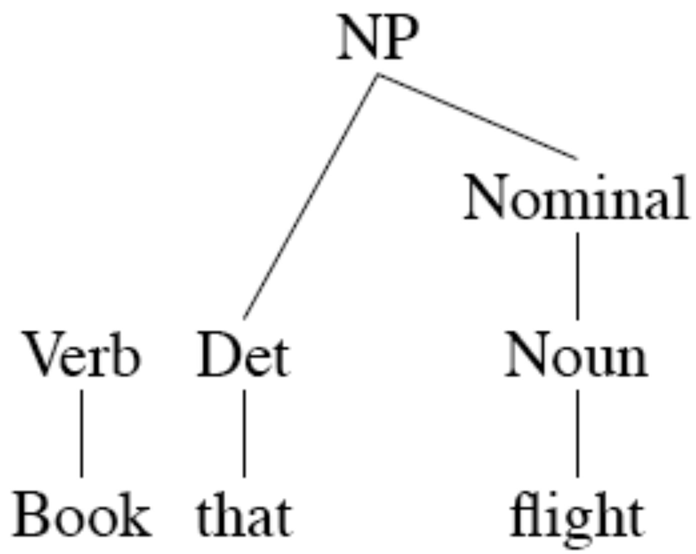
Bottom-Up Search



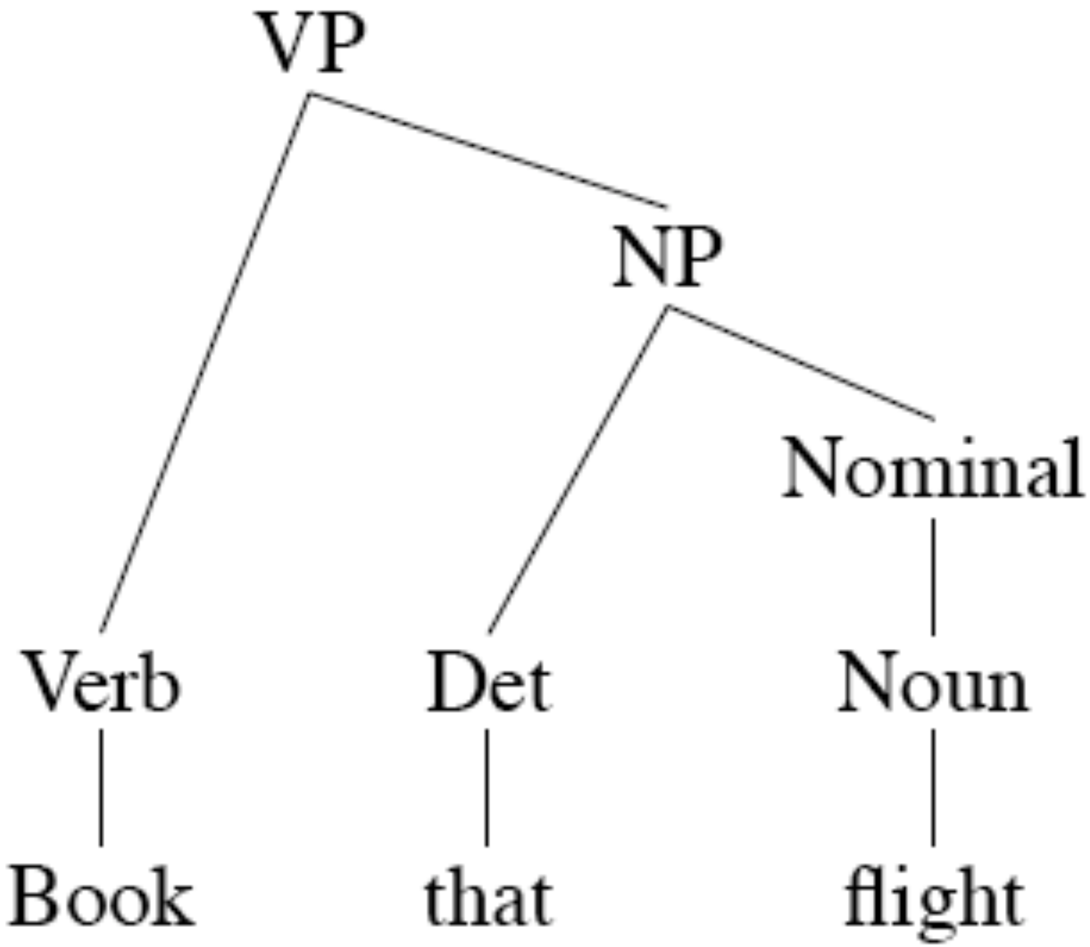
Bottom-Up Search



Bottom-Up Search



Bottom-Up Search



Top-Down and Bottom-Up

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

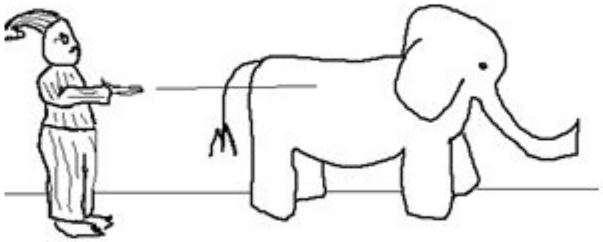
Control

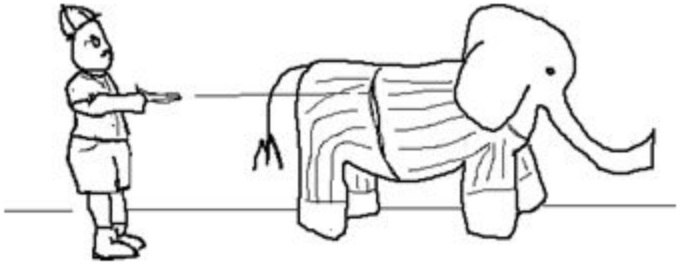
- Of course, in both cases we left out how to keep track of the search space and how to make choices
  - ✓ Which node to try to expand next
  - ✓ Which grammar rule to use to expand a node
- One approach is called backtracking.
  - ✓ Make a choice, if it works out then fine
  - ✓ If not then back up and make a different choice

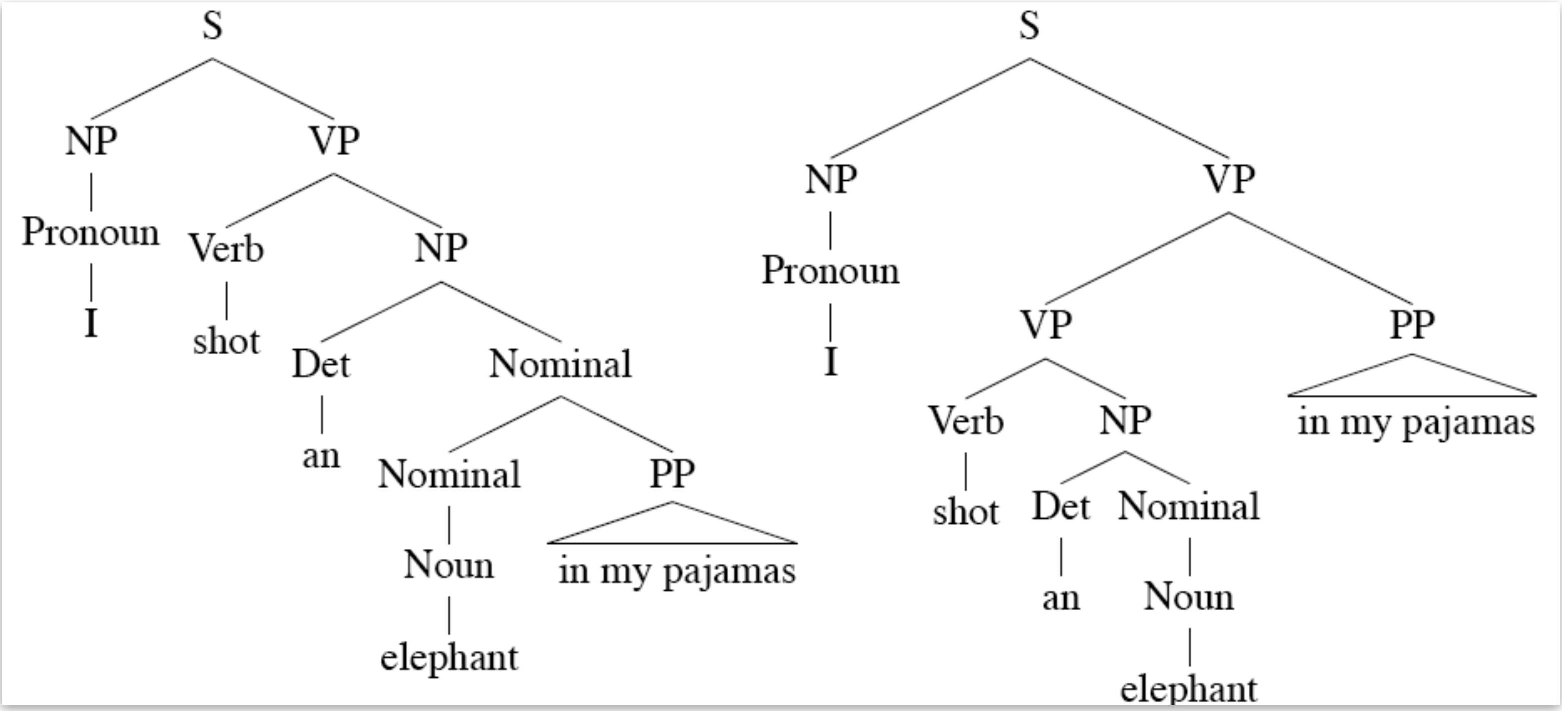
Problems

- Even with the best filtering, backtracking methods are doomed because of two inter-related problems
  - ✓ Ambiguity

Syntactic Ambiguity







CKY (Cocke-Kasami-Younger) Parsing

- One of the earliest recognition and parsing algorithms
- The standard version of CKY can only recognize languages defined by context-free grammars in Chomsky Normal Form (CNF).
- It is also possible to extend the CKY algorithm to handle some grammars which are not in CNF
  - ✓ Harder to understand
- Based on a "dynamic programming" approach:
  - ✓ Build solutions compositionally from sub-solutions
- Uses the grammar directly.

Demo → http://lxmls.it.pt/2015/cky.html

CKY (Cocke-Kasami-Younger) Parsing

- Considers every possible consecutive subsequence of letters and sets $K \in T[i,j]$ if the sequence of letters starting from i to j can be generated from the non-terminal K.
- Once it has considered sequences of length 1, it goes on to sequences of length 2, and so on.
- For subsequences of length 2 and greater, it considers every possible partition of the subsequence into two halves, and checks to see if there is some production A -> BC such that B matches the first half and C matches the second half. If so, it records A as matching the whole subsequence.
- Once this process is completed, the sentence is recognized by the grammar if the entire string is matched by the start symbol.

CKY Algorithm

- Observation: any portion of the input string spanning i to j can be split at k, and structure can then be built using sub-solutions spanning i to k and sub-solutions spanning k to j .
- Meaning: Solution to problem [i, j] can constructed from solution to sub problem [i, k] and solution to sub problem [k ,j].

CKY Algorithm

- Consider the grammar G given by:

$S \rightarrow e \mid AB \mid XB$

$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$

CKY Algorithm

- w = aaabbb :

S → e | AB | XB

T → AB | XB

X → AT

A → a

B → b

a a a b b b

CKY Algorithm

- Write variables for all length 1 substrings

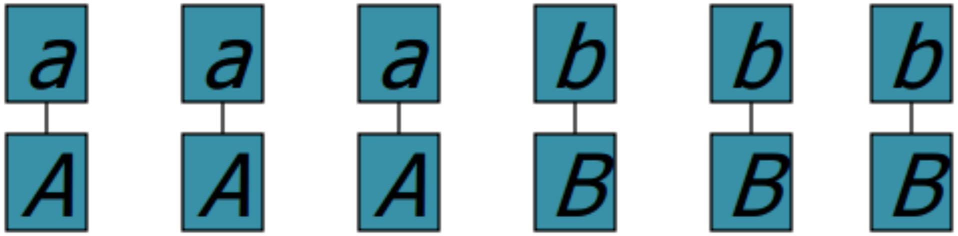
S → e | AB | XB

T → AB | XB

X → AT

A → a

B → b



CKY Algorithm

- Write variables for all length 2 substrings

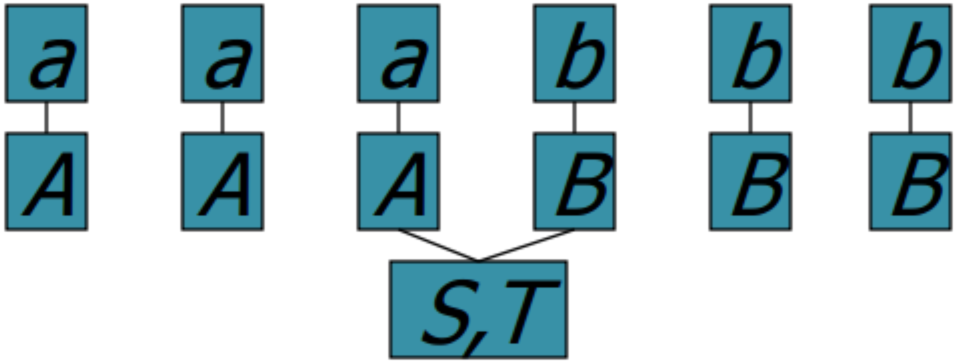
S → e | AB | XB

T → AB | XB

X → AT

A → a

B → b



CKY Algorithm

- Write variables for all length 3 substrings

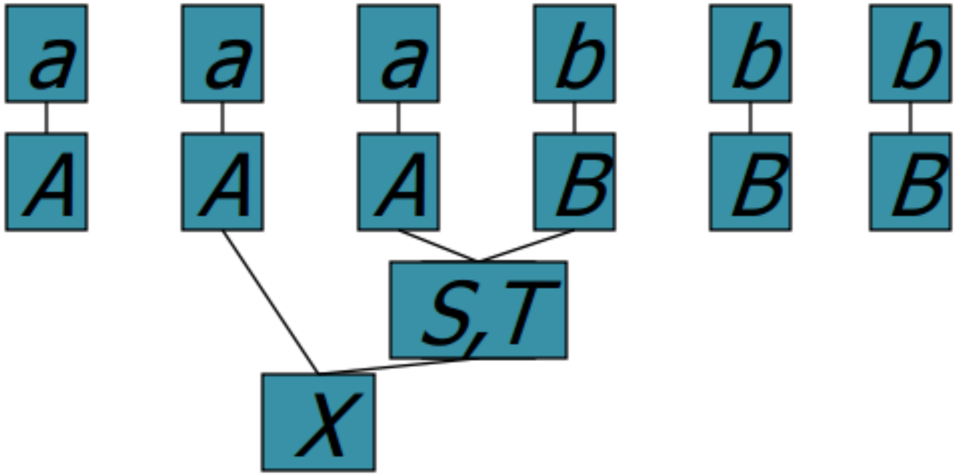
S → e | AB | XB

T → AB | XB

X → AT

A → a

B → b



CKY Algorithm

- Write variables for all length 4 substrings

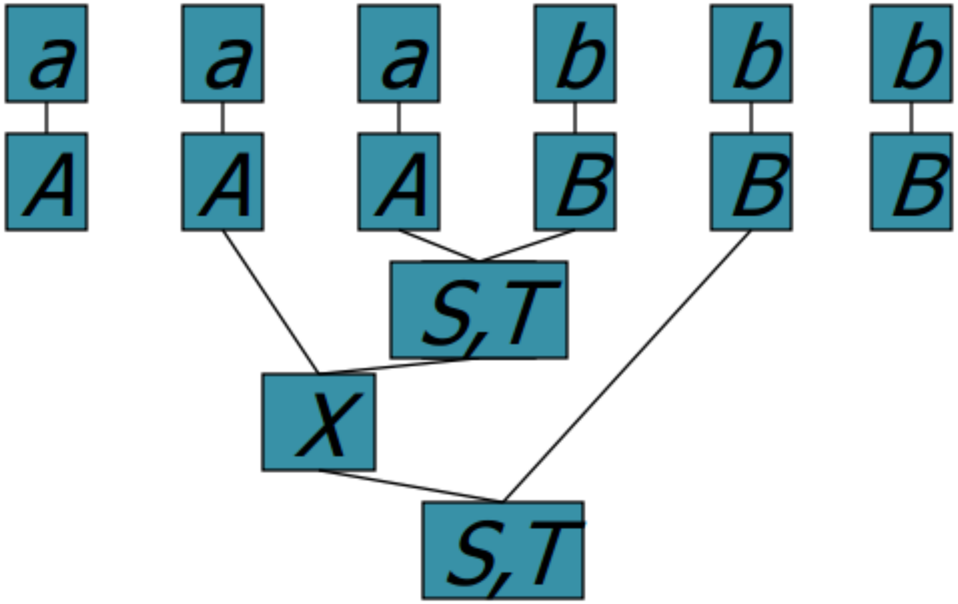
S → e | AB | XB

T → AB | XB

X → AT

A → a

B → b



CKY Algorithm

- Write variables for all length 5 substrings.

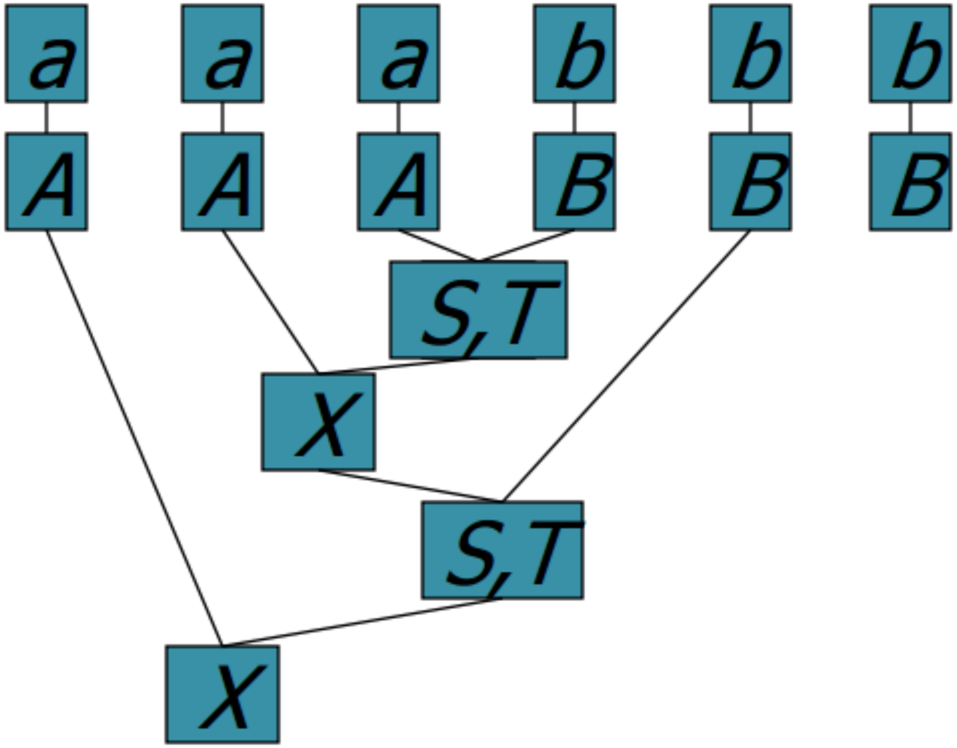
S → e | AB | XB

T → AB | XB

X → AT

A → a

B → b



CKY Algorithm

- Write variables for all length 6 substrings.

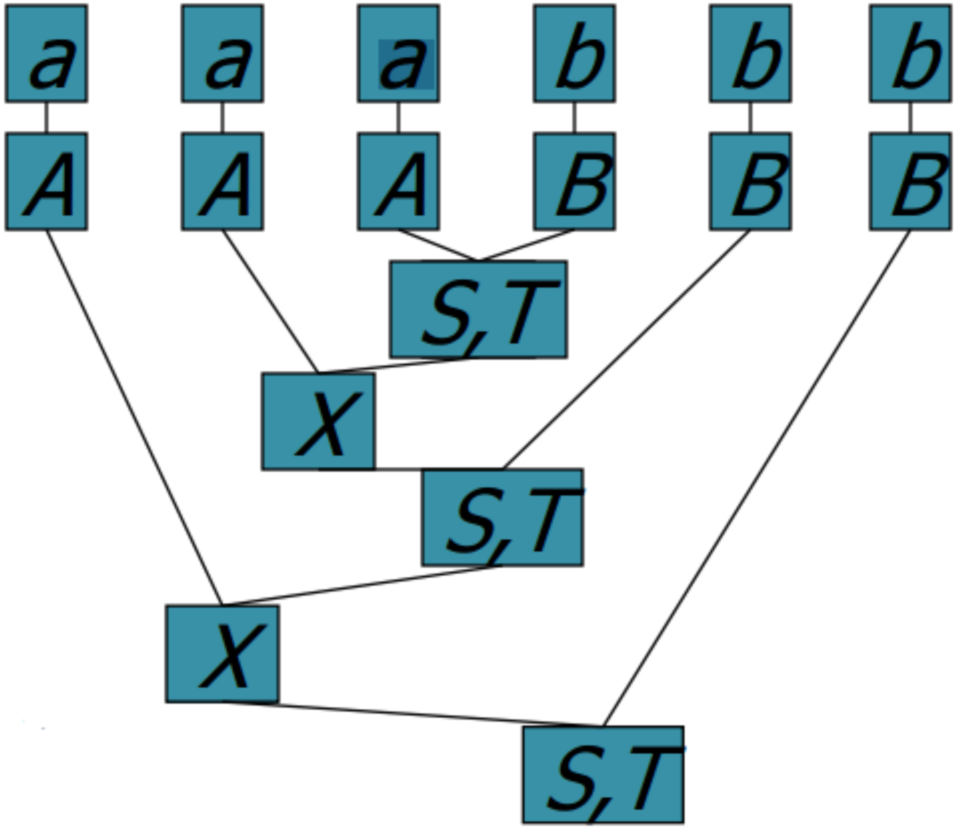
S → e | AB | XB

T → AB | XB

X → AT

A → a

B → b



CKY Algorithm

- The table chart used by the algorithm:

j / i	1 a	2 a	3 a	4 b	5 b	6 b
0	A	-	-	-	X	S,T
1		A	-	X	S,T	-
2			A	S,T	-	-
3				B	-	-
4					B	-
5						B

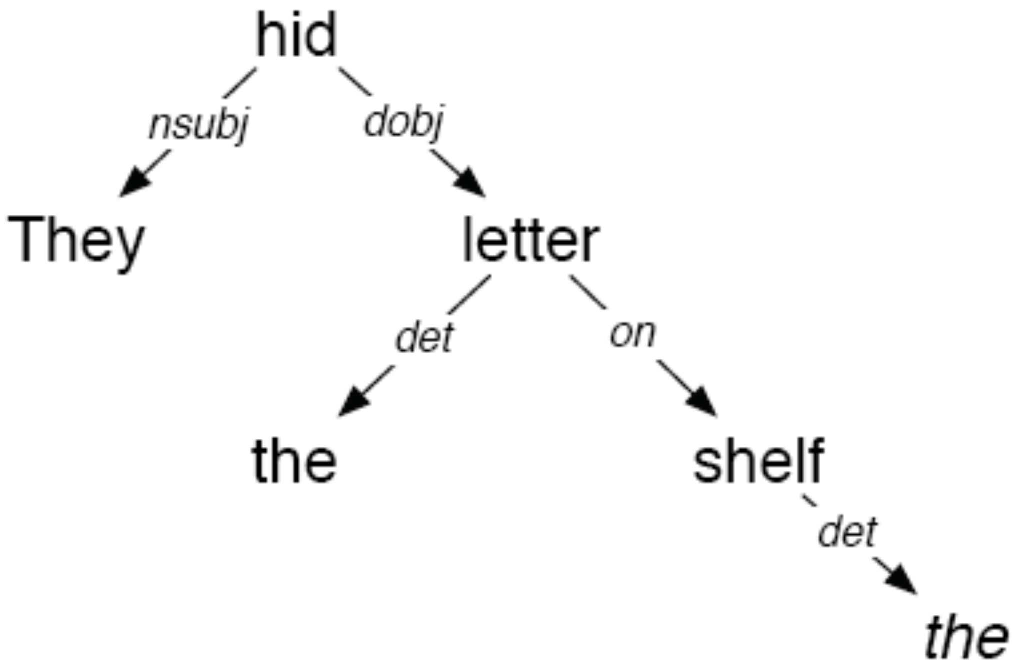
Dependency Grammars

- In CFG-style phrase-structure grammars the main focus is on *constituents.*
- But it turns out you can get a lot done with just binary relations among the words in an utterance.
- In a dependency grammar framework, a parse is a tree where
  - ✓ the nodes stand for the words in an utterance
  - ✓ The links between the words represent dependency relations between pairs of words.
    - ➢ Relations may be typed (labeled), or not.

Dependency Relations

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



They hid the letter on the shelf

Dependency Parsing

- The dependency approach has a number of advantages over full phrase-structure parsing.
  - ✓ Deals well with free word order languages where the constituent structure is quite fluid
  - ✓ Parsing is much faster than CFG-bases parsers
  - ✓ Dependency structure often captures the syntactic relations needed by later applications
    - ➢ CFG-based approaches often extract this same information from trees anyway.

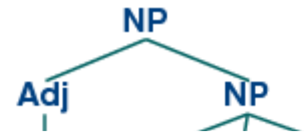
Dependency Parsing

- There are two modern approaches to dependency parsing
  - ✓ Optimization-based approaches that search a space of trees for the tree that *best* matches some criteria
  - ✓ Shift-reduce approaches that greedily take actions based on the current word and state.

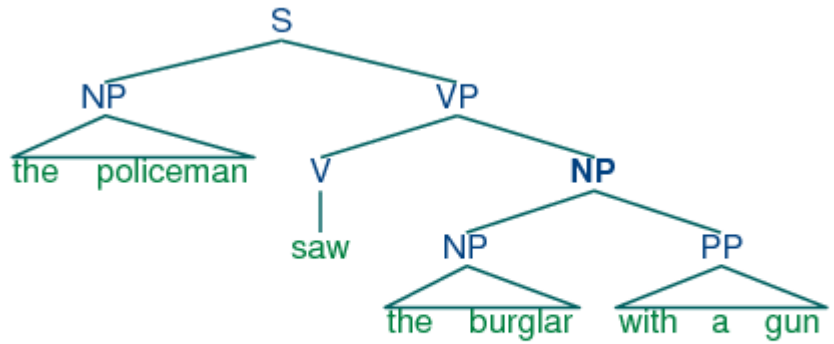
a. old (men and women)

b. (old men) and women

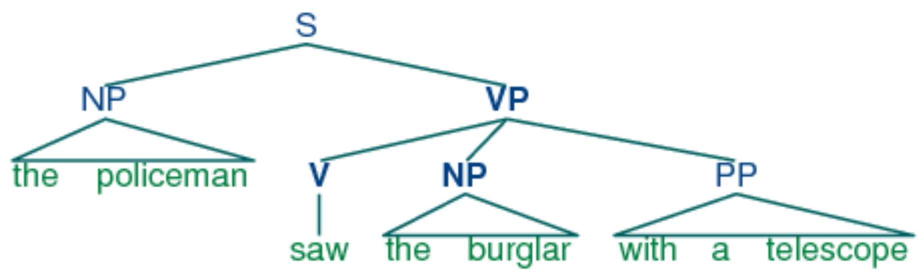
a.



a.



b.




```
>>> s1 = '(S (NP the policeman) (VP (V saw) (NP (NP the burglar) (PP with a gun))))'
>>> s2 = '(S (NP the policeman) (VP (V saw) (NP the burglar) (PP with a telescope)))'
>>> tree1 = nltk.bracket_parse(s1)
>>> tree2 = nltk.bracket_parse(s2)
```

```
>>> tree = nltk.bracket_parse('(NP (Adj old) (NP (N men) (Conj and) (N women)))')
>>> tree.draw()
```

http://nltk.sourceforge.net/doc/en/ch07.html

Syntaxnet

You'll need to install :

- Python 2.7
- Pip
- Bazel
  - Version 0.3.0-0.3.1
- Swig
- Numpy
- mock

Syntaxnet

- You'll need to install :
  - git clone –recursive https://github.com/tensorflow/models.git
  - cd models/syntaxnet/tensorflow
  - ./configure
  - cd ..
  - bazel test syntaxnet/... util/utf8/...
  - # On Mac, run the following:
  - bazel test --linkopt=-headerpad_max_install_names \
  - syntaxnet/... util/utf8/...

Syntaxnet

echo 'Bob brought the pizza to Alice.' | syntaxnet/demo.sh

```
Input: Bob brought the pizza to Alice .
Parse:
brought VBD ROOT
 +-- Bob NNP nsubj
 +-- pizza NN dobj
 |   +-- the DT det
 +-- to IN prep
 |   +-- Alice NNP pobj
 +-- . . punct
```

CYK algorithm 실습

과제 목적

- CYK algorithm에 대해 python을 통해 코딩을 함으로 CYK algorithm에 대한 이해를 확장하며, 실제 코드의 동작 절차를 이해하는데 목적이 있음

Grammar 설정

```
import nltk
gram = nltk.CFG.fromstring("""
S -> NP VP
NP -> Det N | NP PP
VP -> V NP | VP PP
PP -> P NP
Det -> 'the'
N -> 'kids' | 'box' | 'floor'
V -> 'opened'
P -> 'on'
""")
```

테이블 초기화 코드

```
def init_nfst(tok, gram):
    numtokens1 = len(tok)
     # fill w/ dots
    nfst = [["." for i in range(numtokens1+1)] for j in
range(numtokens1+1)]
    # fill in diagonal
    for i in range(numtokens1):
        prod= gram.productions(rhs=tok[i])
        nfst[i][i+1] = prod[0].lhs()
    return nfst
```

테이블 완성 코드

```
def complete_nfst(nfst, tok, trace=False):
    index1 = {}
    for prod in gram.productions():
        index1[prod.rhs()] = prod.lhs()
    numtokens1 = len(tok)
    for span in range(2, numtokens1+1):
        for start in range(numtokens1+1-span):
            end = start + span
            for mid in range(start+1, end):
                nt1, nt2 = nfst[start][mid], nfst[mid][end]
                if (nt1,nt2) in index1:
                    if trace:
                        print "[%s] %3s [%s] %3s [%s] ==>
[%s] %3s [%s]" %  (start, nt1, mid, nt2, end, start,
index1[(nt1,nt2)], end)
```

테이블 출력 코드

```
def display(wfst, tok):
    print 'nWFST ' + ' '.join([("%-4d" % i) for i in range(1,
len(wfst))])
    for i in range(len(wfst)-1):
        print "    %d " % i,
        for j in range(1, len(wfst)):
            print "%-4s" % wfst[i][j],
        print
```

실행 코드

```
tok = ["the", "kids", "opened", "the", "box", "on", "the", "floor"]
res1 = init_nfst(tok, gram)
display(res1, tok)
res2 = complete_nfst(res1,tok,1)
display(res2, tok)
```

실행결과(1)

res1 = init_nfst(tok, gram)

display(res1, tok)

nWFST	1	2	3	4	5	6	7	8
0	Det	.	.	.	.	.	.	.
1	.	N	.	.	.	.	.	.
2	.	.	V	.	.	.	.	.
3	.	.	.	Det	.	.	.	.
4	.	.	.	.	N	.	.	.
5	.	.	.	.	.	P	.	.
6	.	.	.	.	.	.	Det	.
7	.	.	.	.	.	.	.	N

실행결과(2)

res2 = complete_nfst(res1,tok,1)

```
[0] Det [1]   N [2] ==> [0]  NP [2]
[3] Det [4]   N [5] ==> [3]  NP [5]
[6] Det [7]   N [8] ==> [6]  NP [8]
[2]   V [3]  NP [5] ==> [2]  VP [5]
[5]   P [6]  NP [8] ==> [5]  PP [8]
[0]  NP [2]  VP [5] ==> [0]   S [5]
[3]  NP [5]  PP [8] ==> [3]  NP [8]
[2]   V [3]  NP [8] ==> [2]  VP [8]
[2]  VP [5]  PP [8] ==> [2]  VP [8]
[0]  NP [2]  VP [8] ==> [0]   S [8]
```

실행결과(3)

display(res2, tok)

```
nWFST 1    2    3    4    5    6    7    8
   0  Det  NP   .    .    S    .    .    S
   1  .    N    .    .    .    .    .    .
   2  .    .    V    .    VP   .    .    VP
   3  .    .    .    Det  NP   .    .    NP
   4  .    .    .    .    N    .    .    .
   5  .    .    .    .    .    P    .    PP
   6  .    .    .    .    .    .    Det  NP
   7  .    .    .    .    .    .    .    N
```

Thank you!

Heuiseok Lim
Brain-neuro Language Processing Lab
Korea University, Seoul
limhseok@korea.ac.kr