

Lecture 12

Context Free Grammars

and Syntactic Parsing

CS 6320

Outline

- Formal Grammars
 - Context-free grammar
 - Grammars for English
 - Treebanks
 - Dependency grammars
- Syntactic Parsing
 - Bottom-up, top-down
 - Ambiguity
 - CKY parsing

Syntax

- **Syntax:** provides rules to put together words to form components of sentence and to put together these components to form sentences.
- Knowledge of syntax is **useful** for:
 - Parsing
 - QA
 - IE
 - Generation
 - Translation, etc.
- **Grammar** is the formal specification of rules of a language.
- **Parsing** is a method to perform syntactic analysis of a sentence.

Syntax

- Key notions that we'll cover
 - Constituency
 - Grammatical relations and Dependency
 - Heads
- Key formalism
 - Context-free grammars
- Resources
 - Treebanks

Constituency

- The basic idea here is that groups of words within utterances can be shown to act as single units.
- And 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 (might have to use disjunctions of somewhat unlike sub-classes to do this).
- External behavior
 - For example, we can say that noun phrases can come before verbs

Constituency

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

Harry the Horse

the Broadway coppers

they

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

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

Grammars and Constituency

- Of course, 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.
- The approach to grammar, and the analyses, adopted here are very generic (and don't correspond to any modern linguistic theory of grammar).

Chomsky's Classification 1/2

- Chomsky identifies four classes of grammars:
 - **Class 0:** unrestricted phrase-structure grammars. No restriction on type of rules. (Turing equivalent)

$$x \rightarrow y$$

- **Class 1:** context sensitive grammars.

$$xAy \rightarrow xzy$$

Rewrite a non-terminal A in context xAy

- **Class 2:** Context free grammars

$$A \rightarrow x$$

A is a nonterminal.

x is a sequence of terminals and/or nonterminal symbols.

Chomsky's Classification 2/2

- **Class 3:** regular grammars

$$A \rightarrow Bt$$

or

$$A \rightarrow t$$

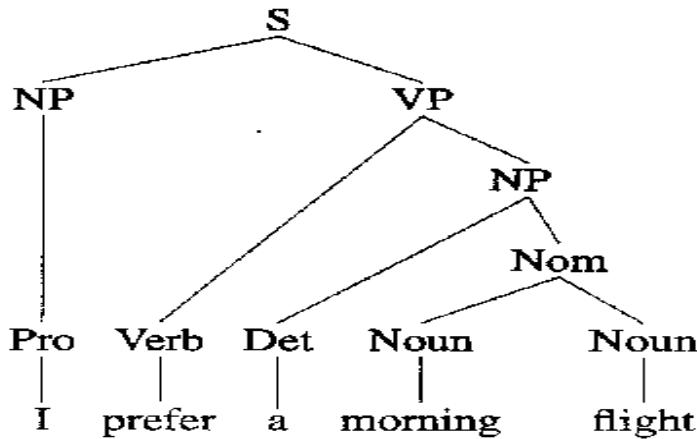
A, B are nonterminals

t is a terminal.

Note: The higher the class the more restrictive it is.

Context Free Grammars

- Just as with FSAs, one can view the grammar rules as either structure imposing device or generative device.
- A derivation is a sequence of rule applications.
- Derivations can be visualized as parse trees.



- Compare CFG with:
 - Regular expressions (too weak)
 - Context sensitive grammars (too strong)
 - Turing machines (way too strong)

Context-Free Grammars

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

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

$$NP \rightarrow Det\ Nominal$$
$$NP \rightarrow ProperNoun$$
$$Nominal \rightarrow Noun \mid Nominal\ Nominal$$

- 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

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**

L0 Grammar

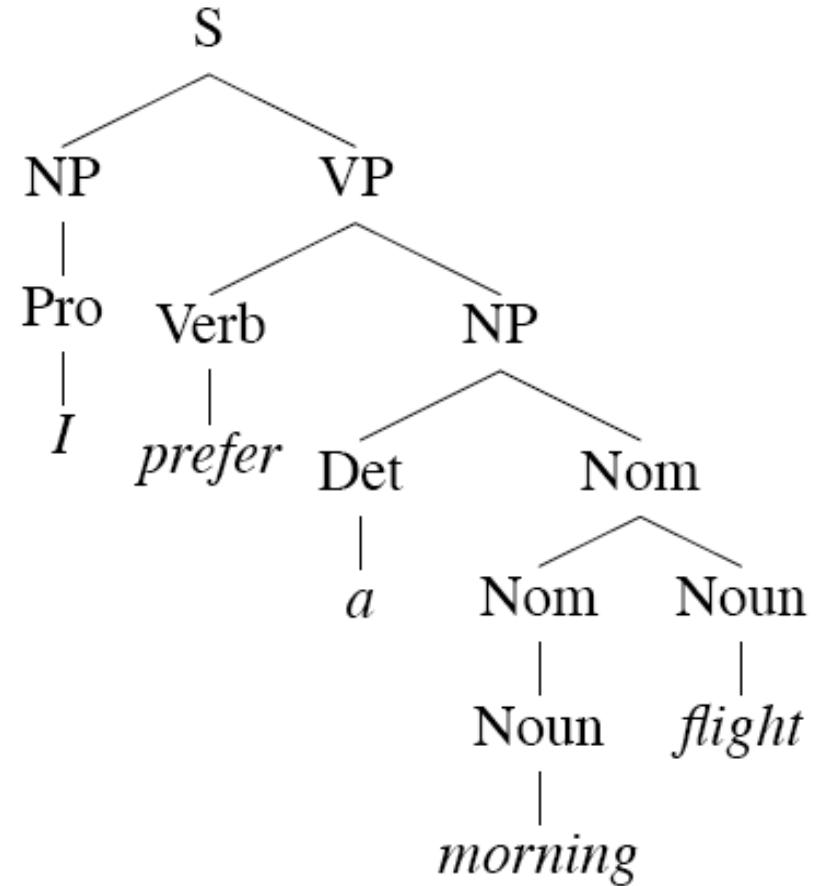
Grammar Rules	Examples
$S \rightarrow NP\ VP$	I + want a morning flight
$NP \rightarrow Pronoun$	I
$Proper-Noun$	Los Angeles
$Det\ Nominal$	a + flight
$Nominal \rightarrow Nominal\ Noun$	morning + flight
$Noun$	flights
$VP \rightarrow Verb$	do
$Verb\ NP$	want + a flight
$Verb\ NP\ PP$	leave + Boston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

Generativity

- As with FSAs and FSTs, you can view these rules as either analysis or synthesis machines
 - Generate strings in the language
 - Reject strings not in the language
 - 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



Parsing

- Parsing is the process of taking a string and a grammar and returning a (multiple?) parse tree(s) for that string
- It is completely analogous to running a finite-state transducer with a tape
 - It's just more powerful
 - Remember this means that there are languages we can capture with CFGs that we can't capture with finite-state methods

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?*

$S \rightarrow Aux NP VP$

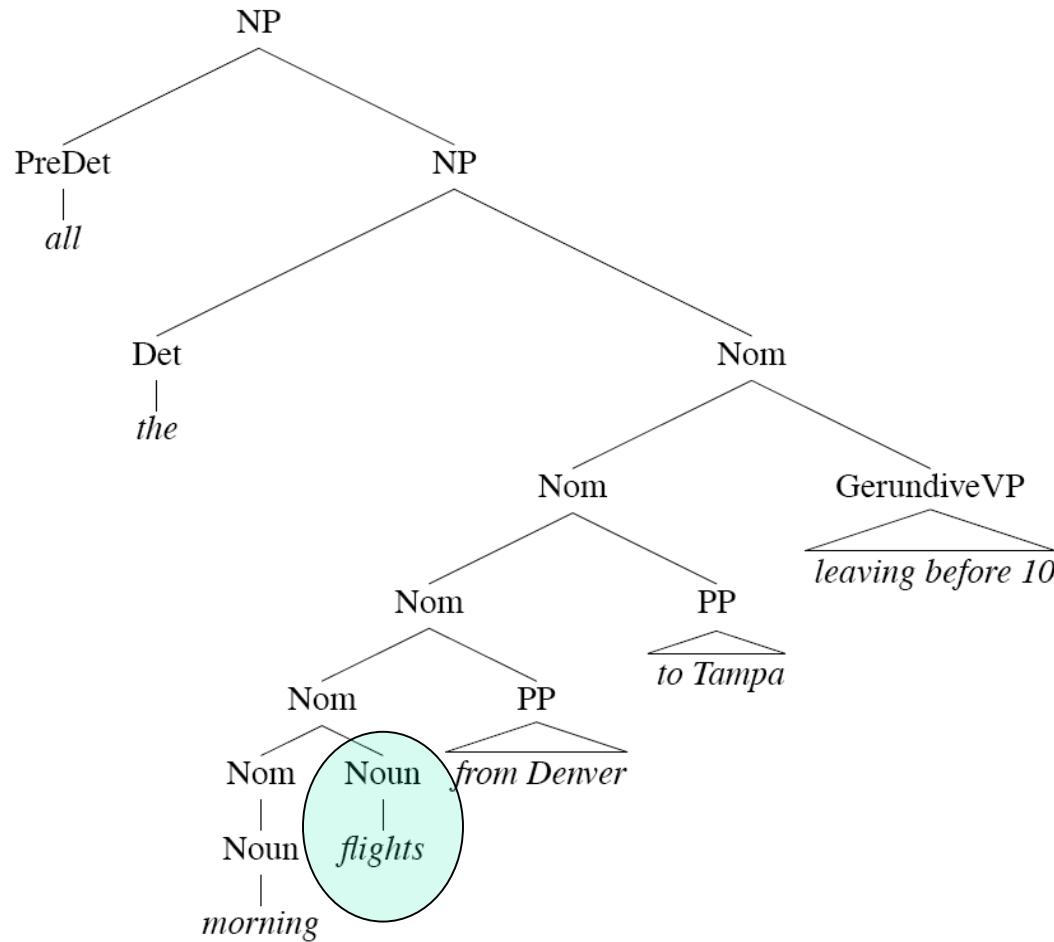
- WH Questions: *When did the plane leave?*

$S \rightarrow WH-NP Aux NP VP$

Noun Phrases

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



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.

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

Nominals

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

Postmodifiers

- 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*

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

Does_{NP} [this flight] stop in Dallas?

$S \rightarrow Aux\ NP\ VP$

Such rules need to have agreements in number, gender, case.

Problem

- Our earlier NP rules are clearly deficient since they don't capture this constraint
 - $NP \rightarrow Det\ Nominal$
 - Accepts, and assigns correct structures, to grammatical examples (*this flight*)
 - But it's also happy with incorrect examples (**these flight*)
 - Such a rule is said to *overgenerate*.
 - We'll come back to this in a bit

Verb Phrases

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

VP → *Verb* disappear

VP → *Verb NP* prefer a morning flight

VP → *Verb NP PP* leave Boston in the morning

VP → *Verb PP* leaving on Thursday

Some Difficulties

- Subcategorization:
 - Verbs have preference for the kind of constituents they cooccur with. Not every verb is compatible with every verb phrase.
Example: want can be used with NP complement, or VP complement.

I want a flight.

I want to fly.

But not other verbs:

*I found to fly...

Subcategorization

- We say that *find* subcategorizes for an NP while *want* subcategorizes for NP or a nonfinite VP.
- Complements, are called subcategorization frames.

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

- Movement:

I looked up his grade.
I looked his grade up.

Subcategorization

- 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.

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-VP}
- Told: I was told [United has a flight]_S
- ...

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

Why?

- Right now, the various rules for VPs *overgenerate*.
 - They permit the presence of strings containing verbs and arguments that don't go together
 - For example
 - VP -> V NP therefore

Sneezed the book is a VP since “sneeze” is a verb and “the book” is a valid NP

Recursive Structures

- Recursive rules: one rule where the nonterminal on the left-hand side also appears on the righthand side.

$NP \rightarrow NP\ PP$ *The flight from Boston*

$VP \rightarrow VP\ PP$ *departed Miami at noon.*

- This allows us to do the following:

Flights to Miami

Flights to Miami from Boston

Flights to Miami from Boston in April

Flights to Miami from Boston in April on Friday

Flights to Miami from Boston in April on Friday under \$300

Flights to Miami from Boston in April on Friday under \$300
with lunch

Conjunctions

$S \rightarrow S \text{ and } S$

$NP \rightarrow NP \text{ and } NP$

$VP \rightarrow VP \text{ and } VP$

- Any phrasal constituent can be conjoined with a constituent of the same type to form a new constituent of that type. We can say that English has the rule:

$X \rightarrow X \text{ and } X$

Treebanks

- Treebanks are corpora in which each sentence has been paired with a parse tree (presumably the right one).
- These are generally created
 - By first parsing the collection with an automatic parser
 - And then having human annotators correct each parse as necessary.
- This generally requires detailed annotation guidelines that provide a POS tagset, a grammar and instructions for how to deal with particular grammatical constructions.

Penn Treebank

- Penn TreeBank is a widely used treebank.

- Most well known is the Wall Street Journal section of the Penn TreeBank.

- 1 M words from the 1987-1989 Wall Street Journal.

```
( (S (‘ ‘))
  (S-TPC-2
    (NP-SBJ-1 (PRP We) )
    (VP (MD would)
      (VP (VB have)
        (S
          (NP-SBJ (-NONE- *-1) )
          (VP (TO to)
            (VP (VB wait)
              (SBAR-TMP (IN until)
                (S
                  (NP-SBJ (PRP we) )
                  (VP (VBP have)
                    (VP (VBN collected)
                      (PP-CLR (IN on)
                        (NP (DT those)(NNS assets))))))))))))
        (, ,) (‘ ‘)
        (NP-SBJ (PRP he) )
        (VP (VBD said)
          (S (-NONE- *T*-2) )))
        (. .) ))
```

Treebank Grammars

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

Treebank Grammars

- Such grammars tend to be very flat due to the fact that they tend to avoid recursion.
 - To ease the annotators burden
- For example, the Penn Treebank has 4500 different rules for VPs. Among them...

VP → VBD PP

VP → VBD PP PP

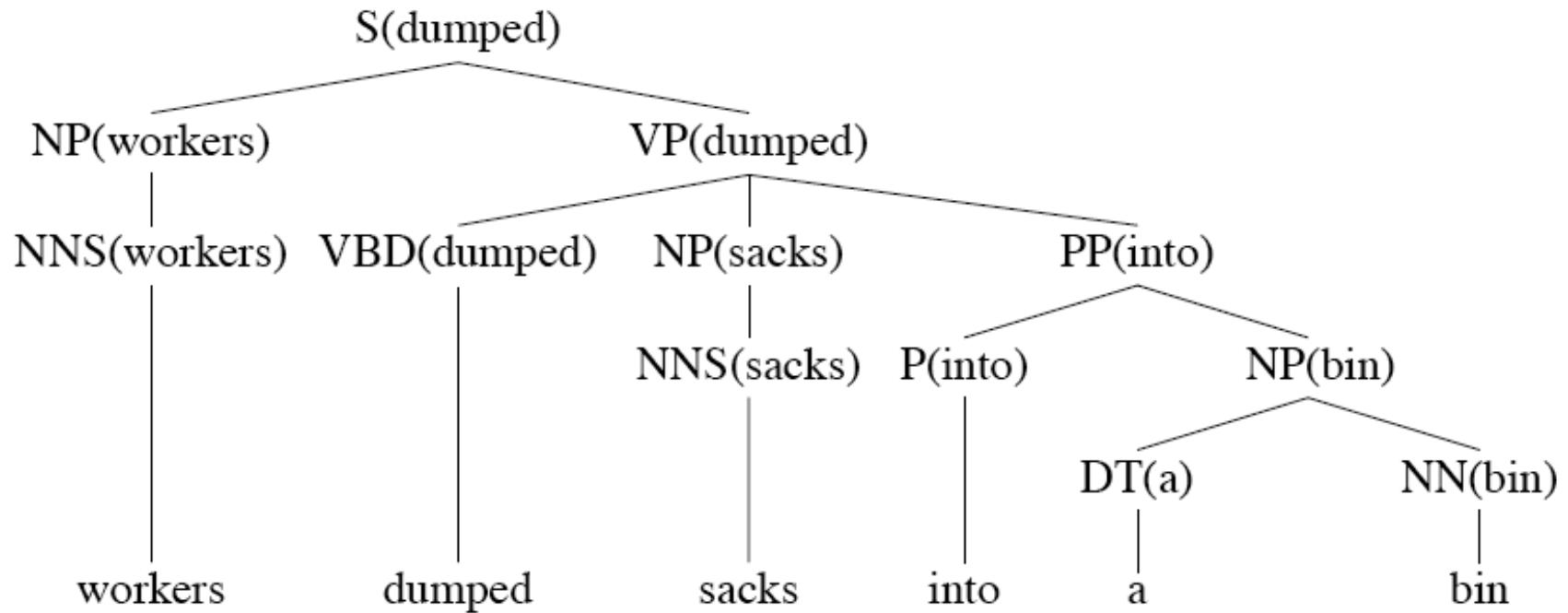
VP → VBD PP PP PP

VP → VBD PP PP PP PP

Heads in Trees

- Finding heads in treebank trees is a task that arises frequently in many applications.
 - Particularly important in statistical parsing
- We can visualize this task by annotating the nodes of a parse tree with the heads of each corresponding node.

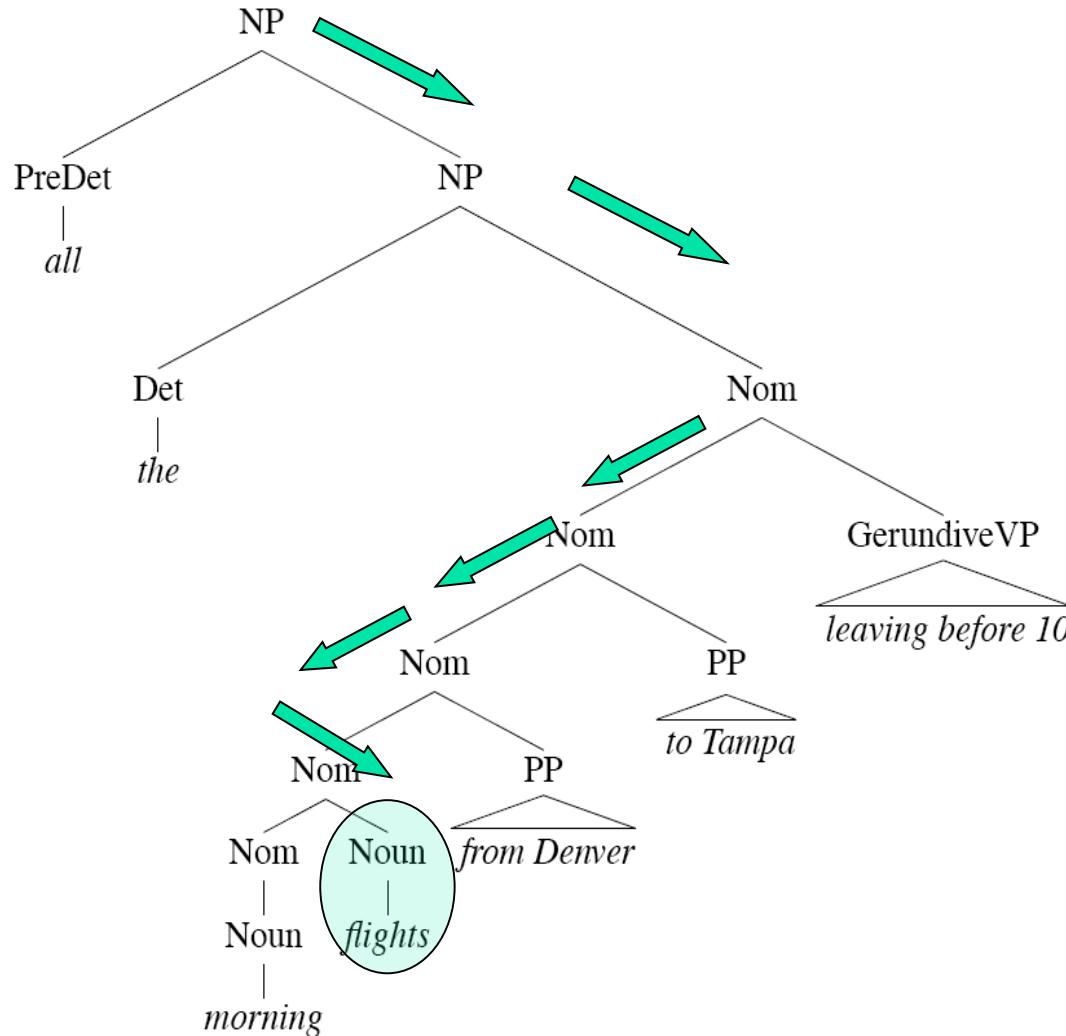
Lexically Decorated Tree



Head Finding

- The standard way to do head finding is to use a simple set of tree traversal rules specific to each non-terminal in the grammar.

Noun Phrases



Treebank Uses

- Treebanks (and headfinding) are particularly critical to the development of statistical parsers
 - Chapter 14
- Also valuable to *Corpus Linguistics*
 - Investigating the empirical details of various constructions in a given language

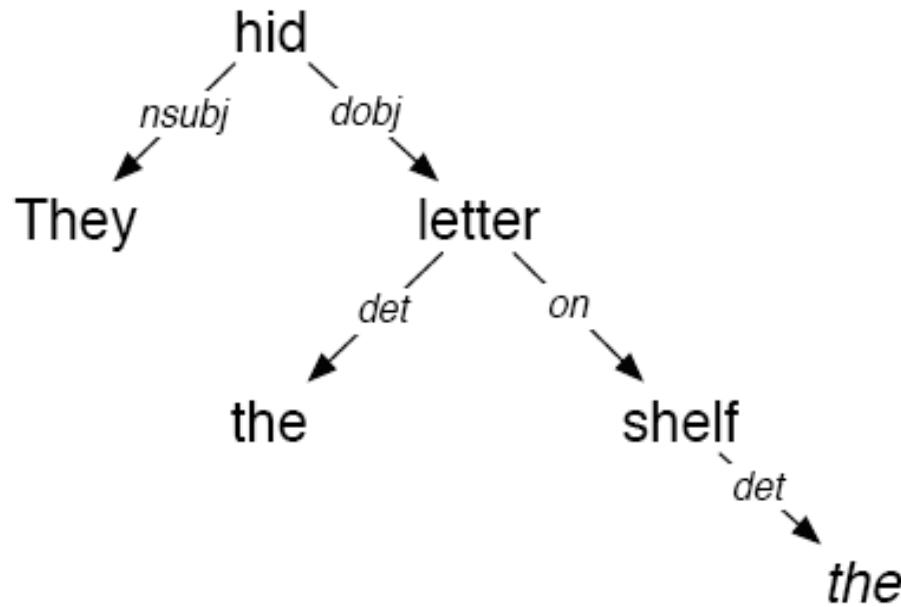
Dependency Grammars

- 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.

Summary of CFG

- Context-free grammars can be used to model various facts about the syntax of a language.
- When paired with parsers, such grammars constitute a critical component in many applications.
- Constituency is a key phenomena easily captured with CFG rules.
 - But agreement and subcategorization do pose significant problems
- Treebanks pair sentences in corpus with their corresponding trees.

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.

A Simple Grammar

$S \rightarrow NP\ VP$

$VP \rightarrow V\ NP$

$NP \rightarrow NAME$

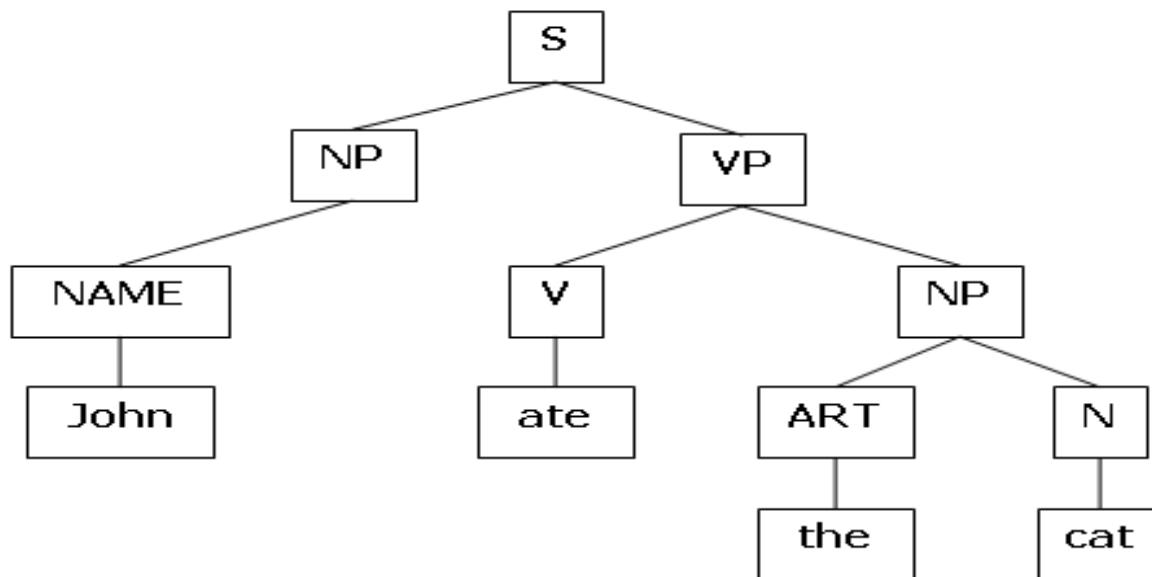
$NP \rightarrow ART\ N$

$NAME \rightarrow John$

$V \rightarrow ate$

$ART \rightarrow the$

$N \rightarrow cat$



Syntactic Parsing

- Representation of a parsed sentence:

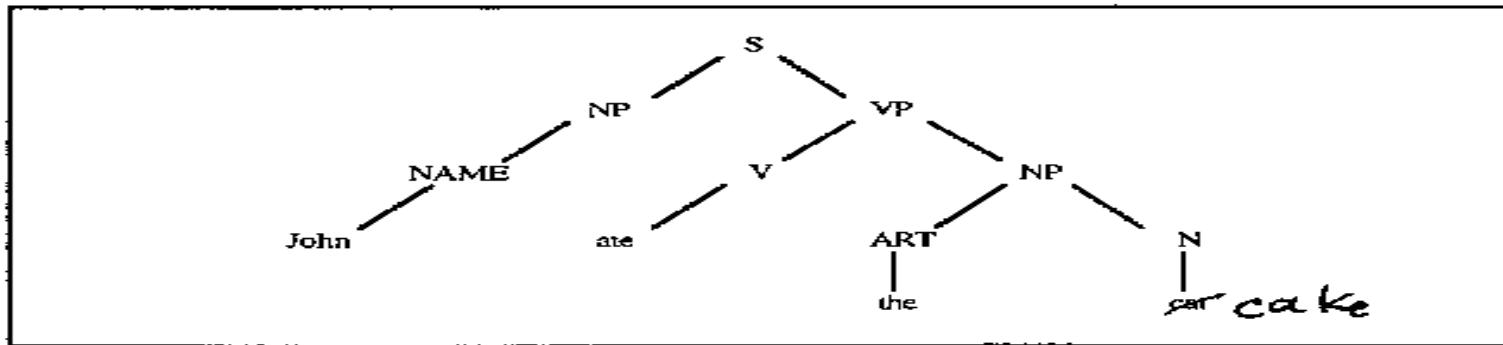
[John]NP[[ate]V[[the]ART[cat]N]NP]]

(S (NP (NAME John))

 (VP (V ate)

 (NP (ART the)

 (N cat))))



A tree representation of *John ate the cat*

- | | |
|----------------------------|----------------------------|
| 1. $S \rightarrow NP\ VP$ | 5. $NAME \rightarrow John$ |
| 2. $VP \rightarrow V\ NP$ | 6. $V \rightarrow ate$ |
| 3. $NP \rightarrow NAME$ | 7. $ART \rightarrow the$ |
| 4. $NP \rightarrow ART\ N$ | 8. $N \rightarrow cat$ |

A simple grammar

Parsing

Top-down Parsing

S
⇒ NP VP
⇒ NAME VP
⇒ John VP
⇒ John V NP
⇒ John ate NP
⇒ John ate ART N
⇒ John ate the N
⇒ John ate the cat

(rewriting S)
(rewriting NP)
(rewriting NAME)
(rewriting VP)
(rewriting V)
(rewriting NP)
(rewriting ART)
(rewriting N)

Bottom-up Parsing

⇒ NAME ate the cat
⇒ NAME V the cat
⇒ NAME V ART cat
⇒ NAME V ART N
⇒ NP V ART N
⇒ NP V NP
⇒ NP VP
⇒ S

(rewriting John)
(rewriting ate)
(rewriting the)
(rewriting cat)
(rewriting NAME)
(rewriting ART N)
(rewriting V NP)
(rewriting NP VP)

Rules are applied from left to right.

Rules are applied from right to left.

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.

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.

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

- 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

Syntactic Parsing

- Define a lexicon:

Cried: V

Dogs: N, V

The: ART

Rewrite S into a sequence of terminals symbols. We want the result as soon as we can. A state of the parse is a pair: symbol list and a number indicating the current position in the sentence.

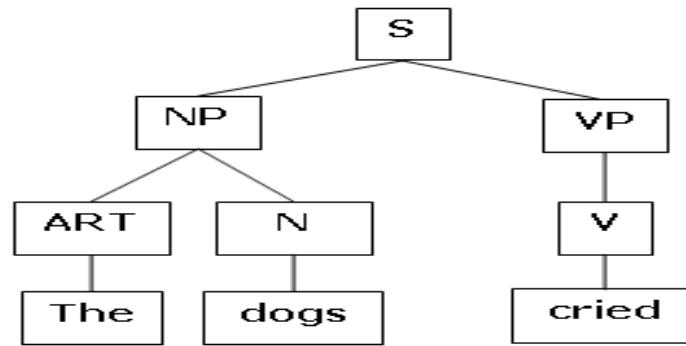
Syntactic Parsing

- 1 The 2 dogs 3 cried 4

Step	Current State	Backup States	Comment
1.	((S) 1)		initial position
2.	((NP VP) 1)		rewriting S by rule 1
3.	((ART N VP) 1)		rewriting NP by rules 2 & 3
4.	((N VP) 2)	((ART ADJ N VP) 1)	matching ART with <i>the</i>
5.	((VP) 3)	((ART ADJ N VP) 1)	matching N with <i>dogs</i>
6.	((V) 3)	((ART ADJ N VP) 1)	rewriting VP by rules 5–8
7.		((V NP) 3) ((ART ADJ N VP) 1)	the parse succeeds as V is matched to <i>cried</i> , leaving an empty grammatical symbol list with an empty sentence

- Another example:
 - Parse the sentence using the same grammar with lexicon:
1 The 2 old 3 man
4 cried 5
the: ART
old: ADJ, N
man: N, V
cried: V

Top-down depth-first parse of 1 *The 2 dogs 3 cried 4*

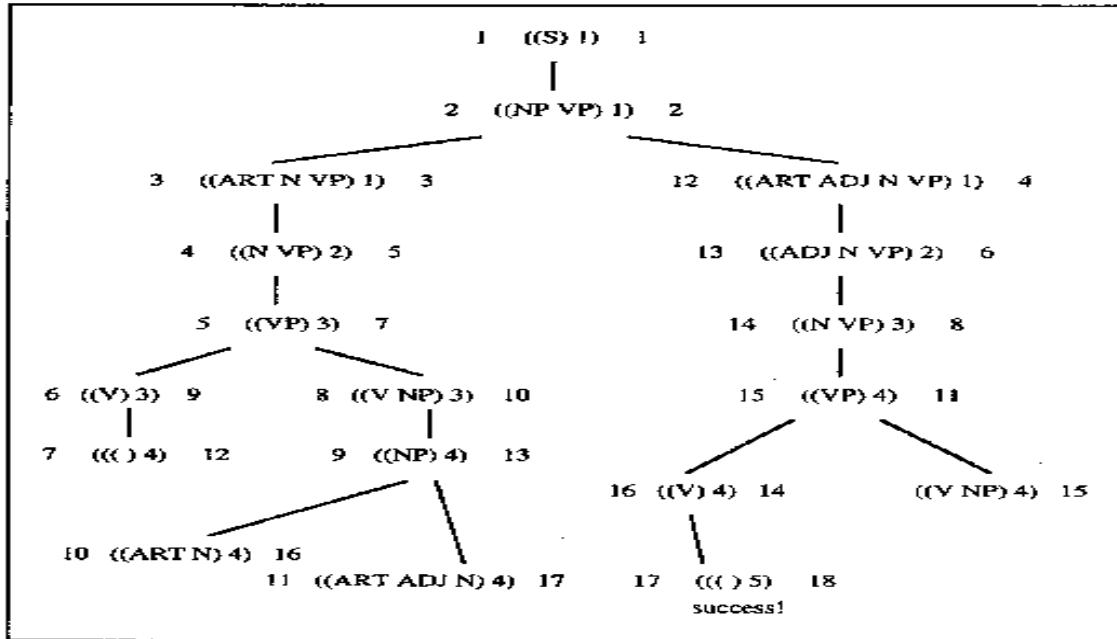


Syntactic Parsing

Step	Current State	Backup States	Comment
1.	((S) 1)		
2.	((NP VP) 1)		
3.	((ART N VP) 1)	((ART ADJ N VP) 1)	S rewritten to NP VP NP rewritten producing two new states
4.	((N VP) 2)	((ART ADJ N VP) 1)	
5.	((VP) 3)	((ART ADJ N VP) 1)	
6.	((V) 3)	((V NP) 3) ((ART ADJ N VP) 1)	the backup state remains
7.	(() 4)	((V NP) 3) ((ART ADJ N VP) 1)	
8.	((V NP) 3)	((ART ADJ N VP) 1)	the first backup is chosen
9.	((NP) 4)	((ART ADJ N VP) 1)	
10.	((ART N) 4)	((ART ADJ N) 4) ((ART ADJ N VP) 1)	looking for ART at 4 fails
11.	((ART ADJ N) 4)	((ART ADJ N VP) 1)	fails again
12.	((ART ADJ N VP) 1)		now exploring backup state saved in step 3
13.	((ADJ N VP) 2)		
14.	((N VP) 3)		
15.	((VP) 4)		
16.	((V) 4)	((V NP) 4)	
17.	(() 5)		success!

A top-down parse of *The_2 old_3 man_4 cried_5*

Syntactic Parsing



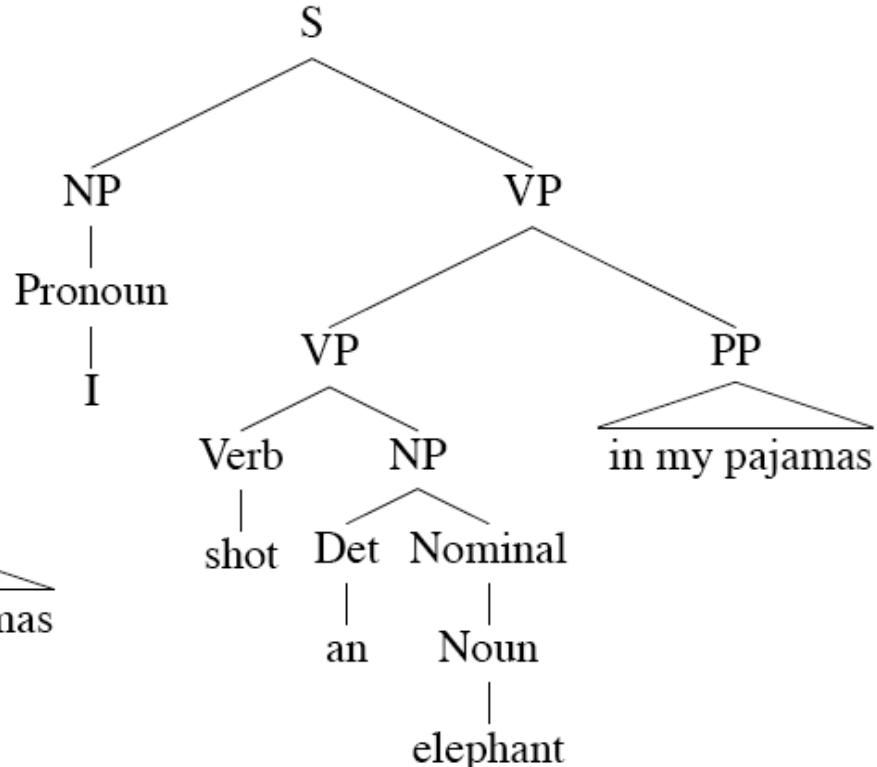
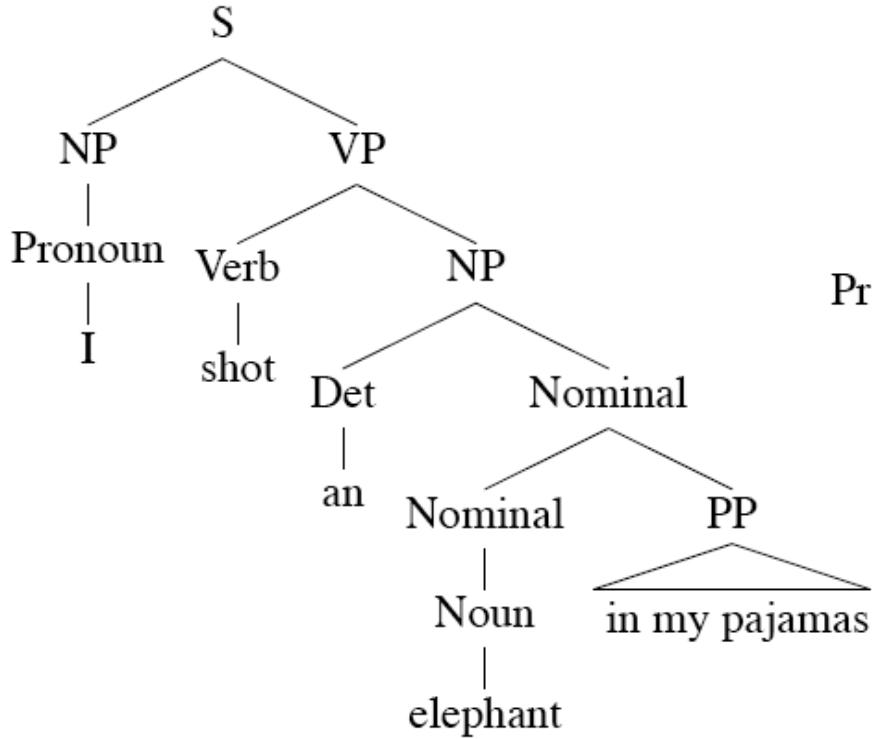
Search tree for two parse strategies (depth-first strategy on left; breadth-first on right)

- Depth-first search: the states form a stack LIFO policy.
- Breath-first search: the states form a queue FIFO policy.
Note the large number of states for a small sentence.

Problems

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

Ambiguity

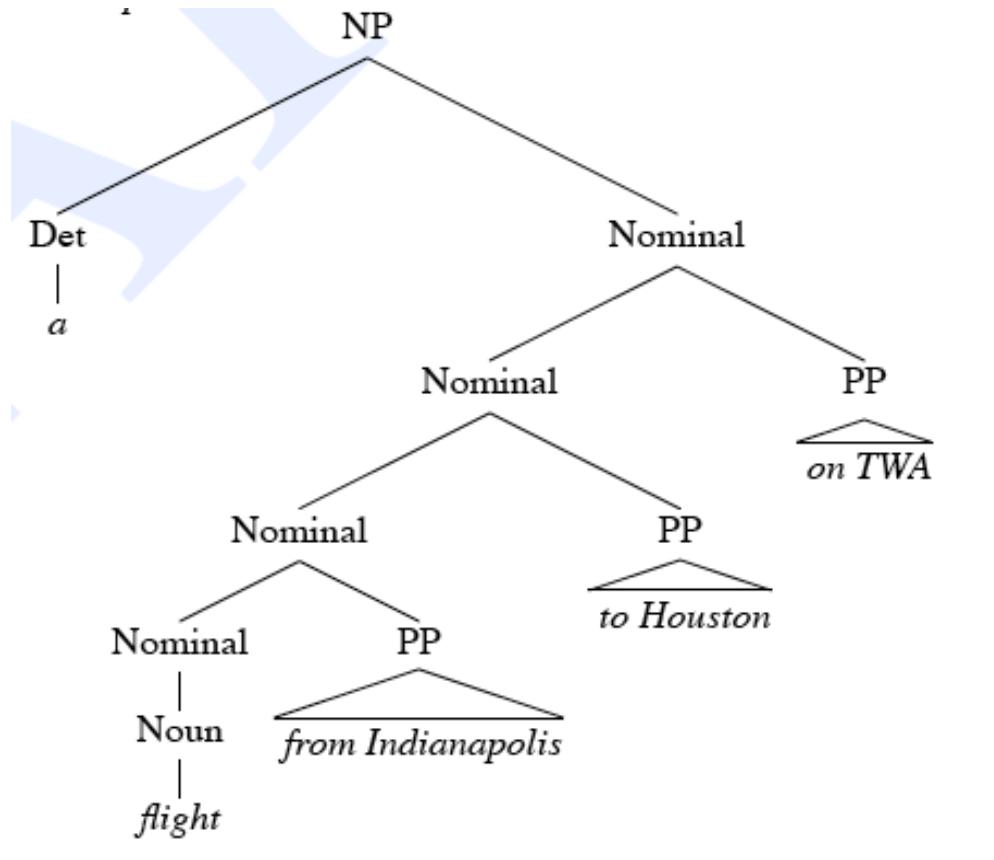


Shared Sub-Problems

- No matter what kind of search (top-down or bottom-up or mixed) that we choose.
 - We don't want to redo work we've already done.
 - Unfortunately, naïve backtracking will lead to duplicated work.

Shared Sub-Problems

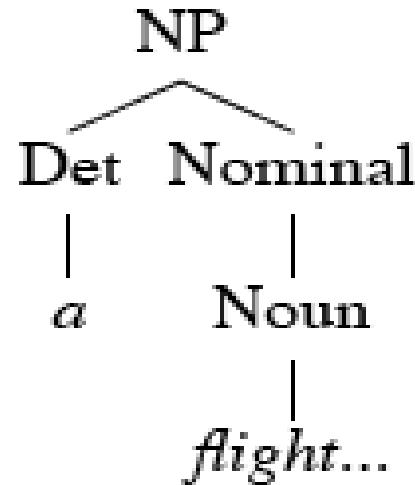
- Consider
 - A flight from Indianapolis to Houston on TWA



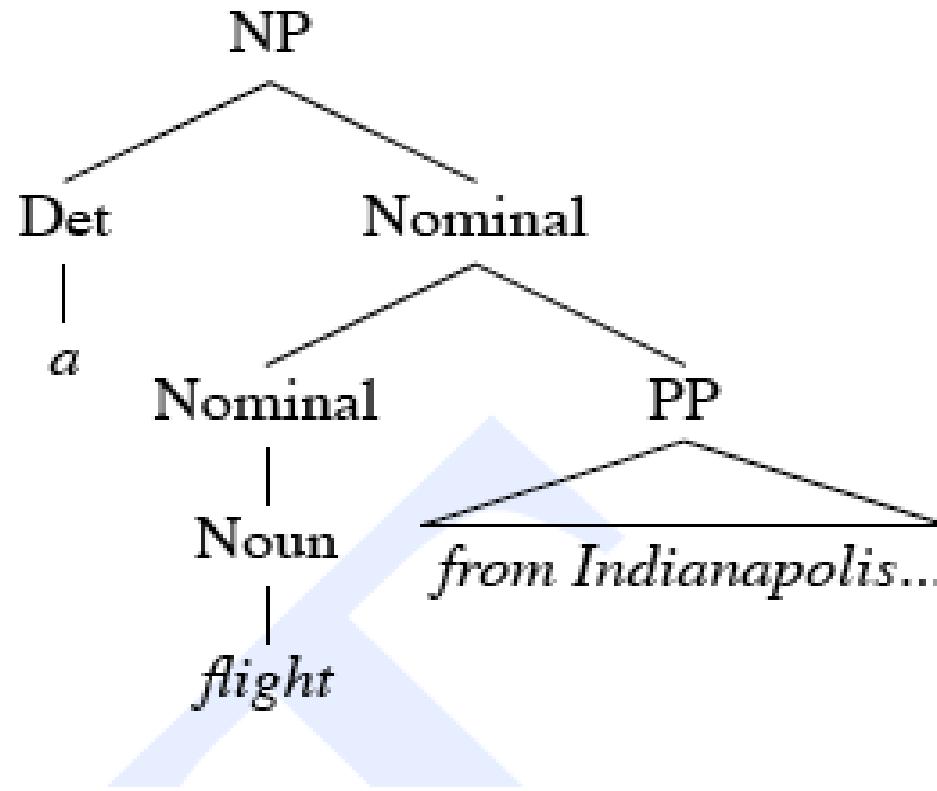
Shared Sub-Problems

- Assume a top-down parse making choices among the various Nominal rules.
- In particular, between these two
 - Nominal -> Noun
 - Nominal -> Nominal PP
- Statically choosing the rules in this order leads to the following bad results...

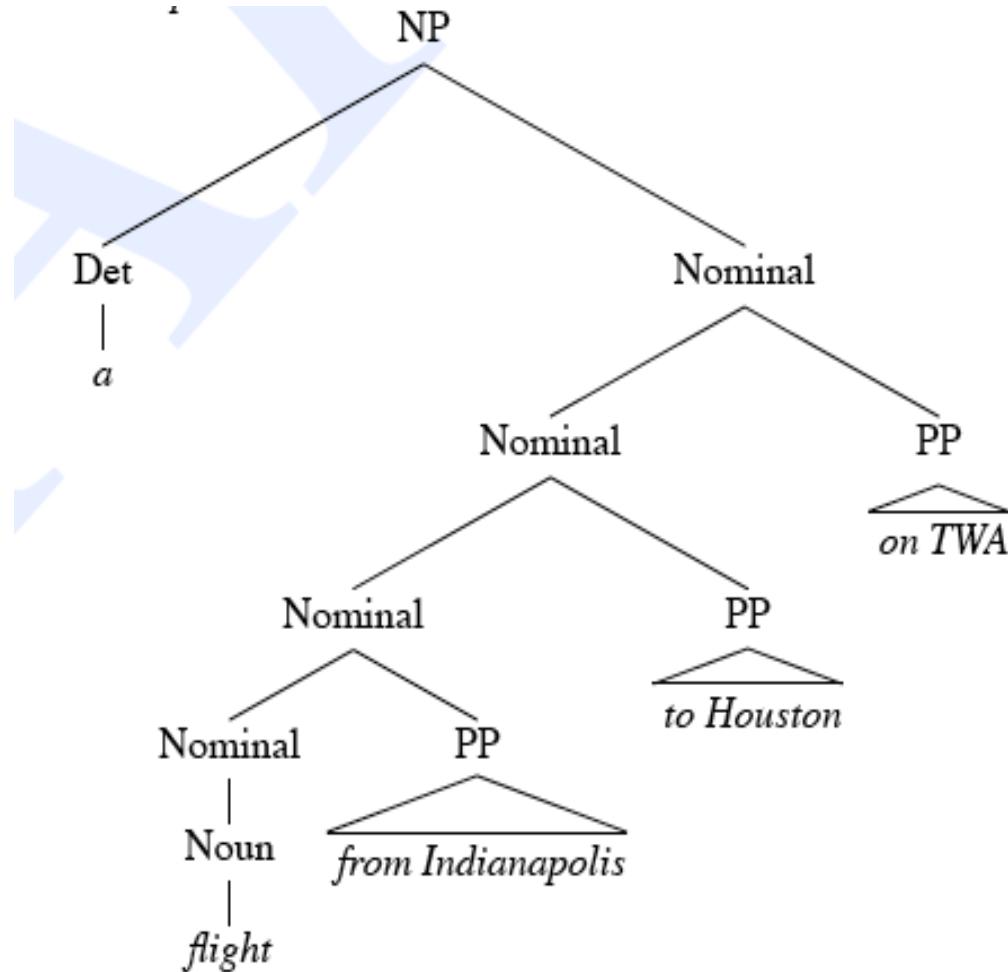
Shared Sub-Problems



Shared Sub-Problems



Shared Sub-Problems



Dynamic Programming

- DP search methods fill tables with partial results and thereby
 - Avoid doing avoidable repeated work
 - Solve exponential problems in polynomial time (well, no not really)
 - Efficiently store ambiguous structures with shared sub-parts.
- We'll cover two approaches that roughly correspond to top-down and bottom-up approaches.
 - CKY
 - Earley

CKY Parsing

- First we'll limit our grammar to epsilon-free, binary rules (more later)
- Consider the rule $A \rightarrow BC$
 - If there is an A somewhere in the input then there must be a B followed by a C in the input.
 - If the A spans from i to j in the input then there must be some k st. $i < k < j$
 - Ie. The B splits from the C someplace.

Problem

- What if your grammar isn't binary?
 - As in the case of the TreeBank grammar?
- Convert it to binary... any arbitrary CFG can be rewritten into Chomsky-Normal Form automatically.
- What does this mean?
 - The resulting grammar accepts (and rejects) the same set of strings as the original grammar.
 - **But** the resulting derivations (trees) are different.

Problem

- More specifically, we want our rules to be of the form

$A \rightarrow BC$

Or

$A \rightarrow w$

That is, rules can expand to either 2 non-terminals or to a single terminal.

Binarization Intuition

- Eliminate chains of unit productions.
- Introduce new intermediate non-terminals into the grammar that distribute rules with **length > 2** over several rules.

■ So... $S \rightarrow A B C$ turns into

$S \rightarrow X C$ and

$X \rightarrow A B$

Where X is a symbol that doesn't occur anywhere else in the grammar.

Sample L1 Grammar

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

CNF Conversion

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

CKY

- So let's build a table so that an A spanning from i to j in the input is placed in cell $[i, j]$ in the table.
- So a non-terminal spanning an entire string will sit in cell $[0, n]$
 - Hopefully an S
- If we build the table bottom-up, we'll know that the parts of the A must go from i to k and from k to j , for some k .

CKY

- Meaning that for a rule like $A \rightarrow B C$ we should look for a B in $[i,k]$ and a C in $[k,j]$.
- In other words, if we think there might be an A spanning i,j in the input... AND
 $A \rightarrow B C$ is a rule in the grammar THEN
- There must be a B in $[i,k]$ and a C in $[k,j]$ for some $i < k < j$

- So to fill the table loop over the $\text{cell}[i,j]$ values in some systematic way
 - What constraint should we put on that systematic search?
 - For each cell, loop over the appropriate k values to search for things to add.

CKY Algorithm

```
function CKY-PARSE(words, grammar) returns table
  for j  $\leftarrow$  from 1 to LENGTH(words) do
    table[j - 1, j]  $\leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$ 
  for i  $\leftarrow$  from j - 2 downto 0 do
    for k  $\leftarrow$  i + 1 to j - 1 do
      table[i, j]  $\leftarrow$  table[i, j]  $\cup$ 
       $\{A \mid A \rightarrow BC \in grammar,$ 
       $B \in table[i, k],$ 
       $C \in table[k, j]\}$ 
```

CKY Parsing

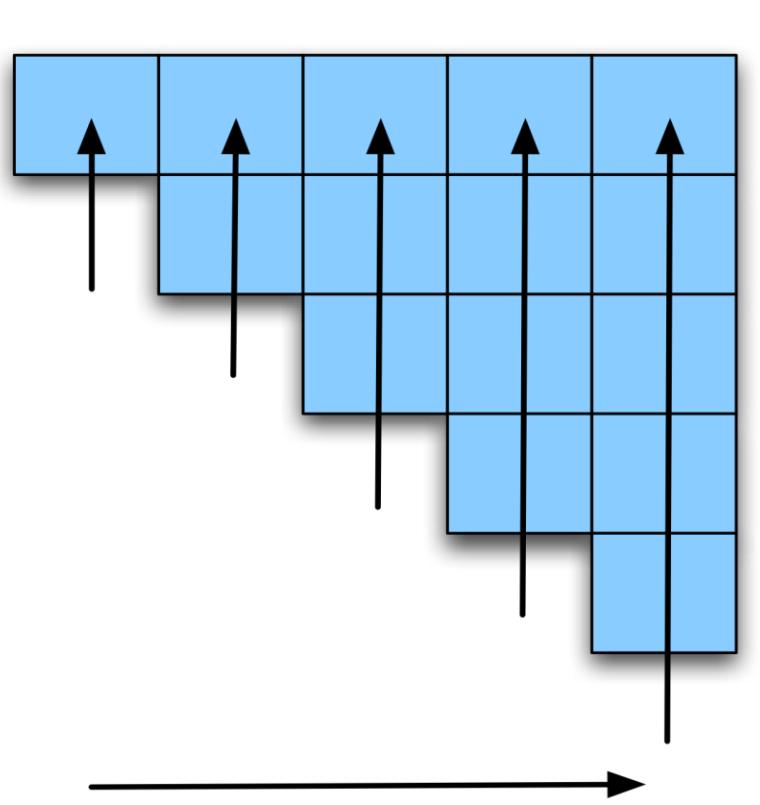
- Is that really a parser?

Note

- We arranged the loops to fill the table a column at a time, from left to right, bottom to top.
 - This assures us that whenever we're filling a cell, the parts needed to fill it are already in the table (to the left and below)
 - It's somewhat natural in that it processes the input a left to right a word at a time
 - Known as online

Example

<i>Book</i>	<i>the</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
S, VP, Verb Nominal, Noun [0,1]		S,VP,X2 [0,3]		S,VP,X2 [0,5]
Det [1,2]	NP [1,3]			NP [1,5]
	Nominal, Noun [2,3]			Nominal [2,5]
		Prep [3,4]	PP [3,5]	
			NP, Proper- Noun [4,5]	



Example

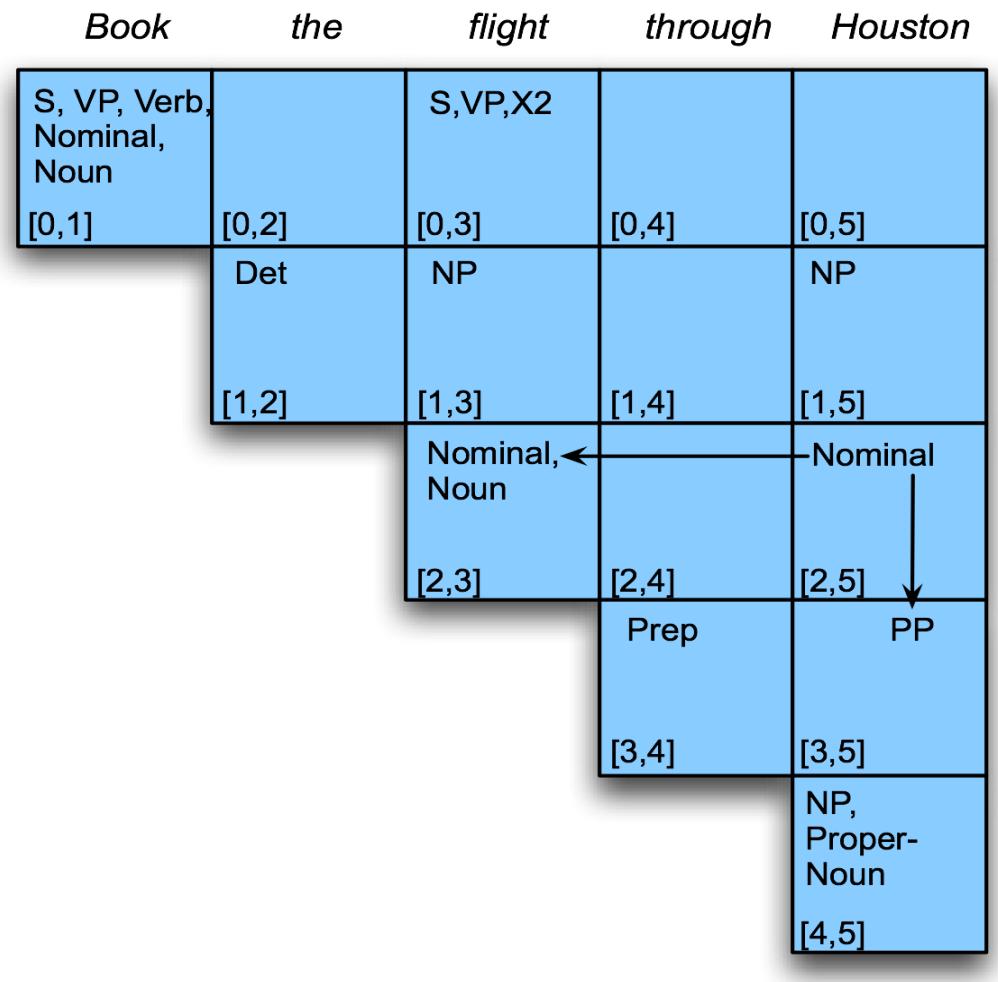
<i>Book</i>	<i>the</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
Det [1,2]	NP [1,3]		[1,4]	[1,5]
	Nominal, Noun [2,3]			Nominal [2,5]
	Prep [3,4]		[3,5]	
		NP, Proper- Noun [4,5]		

Filling column 5

Example

<i>Book</i>	<i>the</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
Det [1,2]	NP [1,3]			NP [1,5]
	Nominal, Noun [2,3]	[2,4]	[2,5]	
	Prep ← [3,4]	PP [3,5] ↓		NP, Proper- Noun [4,5]

Example



Example

Book the flight through Houston

S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
	Det ← NP [1,2]	NP [1,3]		NP [1,5]
		Nominal, Noun [2,3]		Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper- Noun [4,5]

Example

Book *the* *flight* *through* *Houston*

S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S, VP, X2 [0,3]	[0,4]	S ₁ , VP, X2 S ₂ , VP S ₃ NP [1,5] Nominal [2,5] PP [3,5] NP, Proper- Noun [4,5]
Det [1,2]	NP [1,3]	[1,4]	[1,5]	Nominal [2,3] [2,4] [2,5] Prep [3,4] [3,5] NP, Proper- Noun [4,5]

CKY Notes

- Since it's bottom up, CKY populates the table with a lot of phantom constituents.
 - Segments that by themselves are constituents but cannot really occur in the context in which they are being suggested.
 - To avoid this we can switch to a top-down control strategy
 - Or we can add some kind of filtering that blocks constituents where they can not happen in a final analysis.

Back to Ambiguity

- Did we solve it?

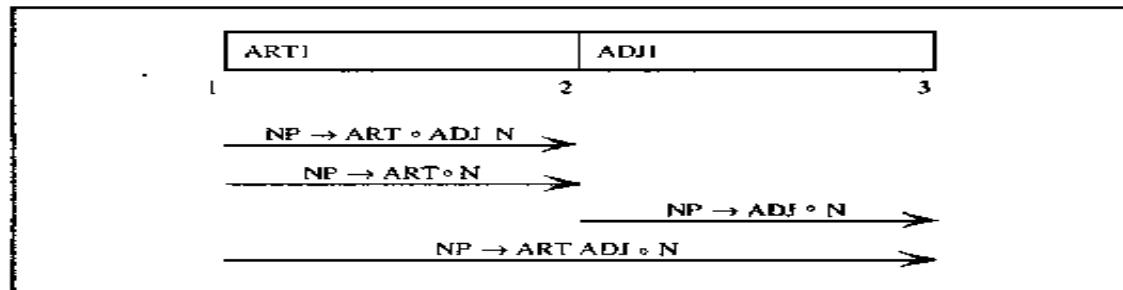
Ambiguity

- No...
 - Both CKY and Earley will result in multiple **S** structures for the **[0,N]** table entry.
 - They both efficiently store the sub-parts that are shared between multiple parses.
 - And they obviously avoid re-deriving those sub-parts.
 - But neither can tell us which one is right.
- In most cases, humans don't notice incidental ambiguity (lexical or syntactic). It is resolved on the fly and never noticed.
- We'll try to model that with probabilities.

Example

- A bottom up chart parser:
 - The idea is to match a sequence of symbols to the right hand side of each rule to determine if a rule is applicable.
 - To reduce the search space use a data structure called a chart that keeps track of successful rules. The process stops when the entire sentence is covered. Efficiency results from not repeating the construction of sentence blocks (or constituents).
- Grammar:

1. $S \rightarrow NP VP$
2. $NP \rightarrow ART ADJ N$
3. $NP \rightarrow ART N$
4. $NP \rightarrow ADJ N$
5. $VP \rightarrow AUX VP$
6. $VP \rightarrow V NP$



The chart after seeing an ADJ in position 2

Example

Algorithm

Do until there is no input left:

1. If the agenda is empty, look up the interpretations for the next word in the input and add them to the agenda.
2. Select a constituent from the agenda (let's call it constituent C from position p_1 to p_2).
3. For each rule in the grammar of form $X \rightarrow C X_1 \dots X_n$, add an active arc of form $X \rightarrow^* C X_1 \dots X_n$ from position p_1 to p_2 .
4. Add C to the chart using the arc extension algorithm above.

A bottom-up chart parsing algorithm

Lexicon:

the: ART

large: ADJ

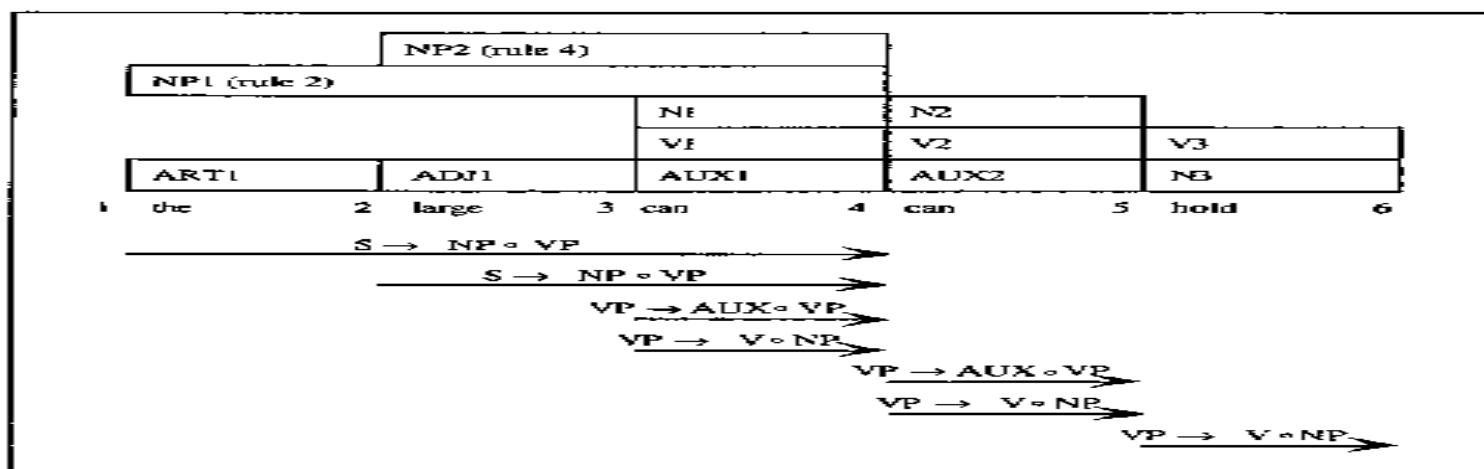
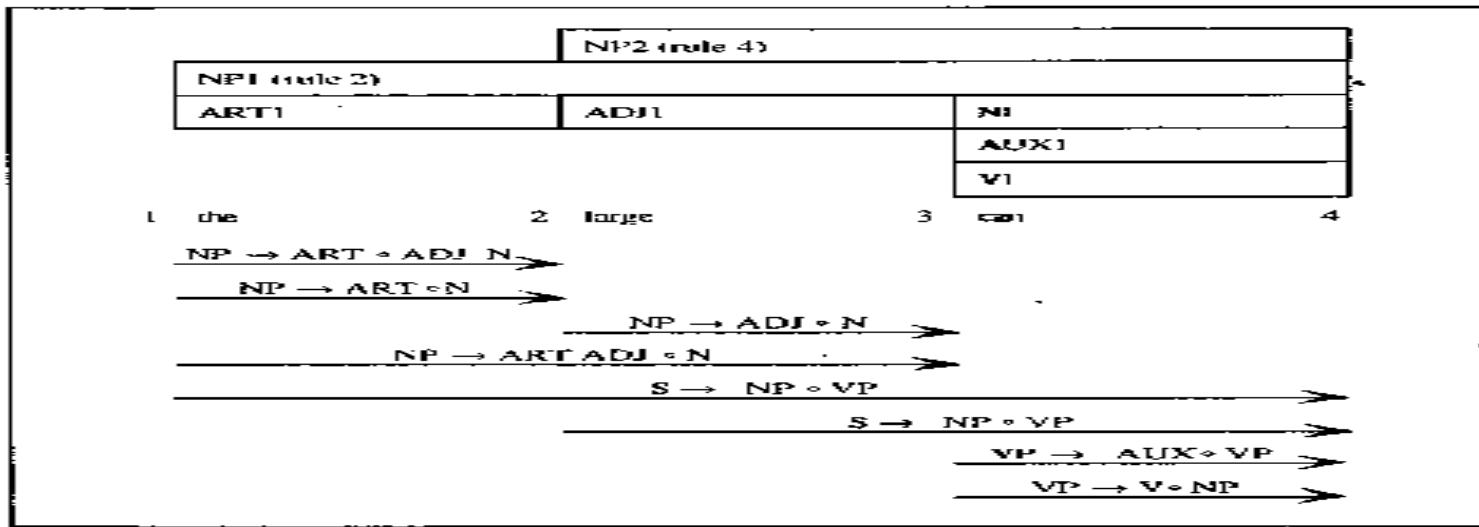
can: N, AUX, V

hold: N, V

water: N, V

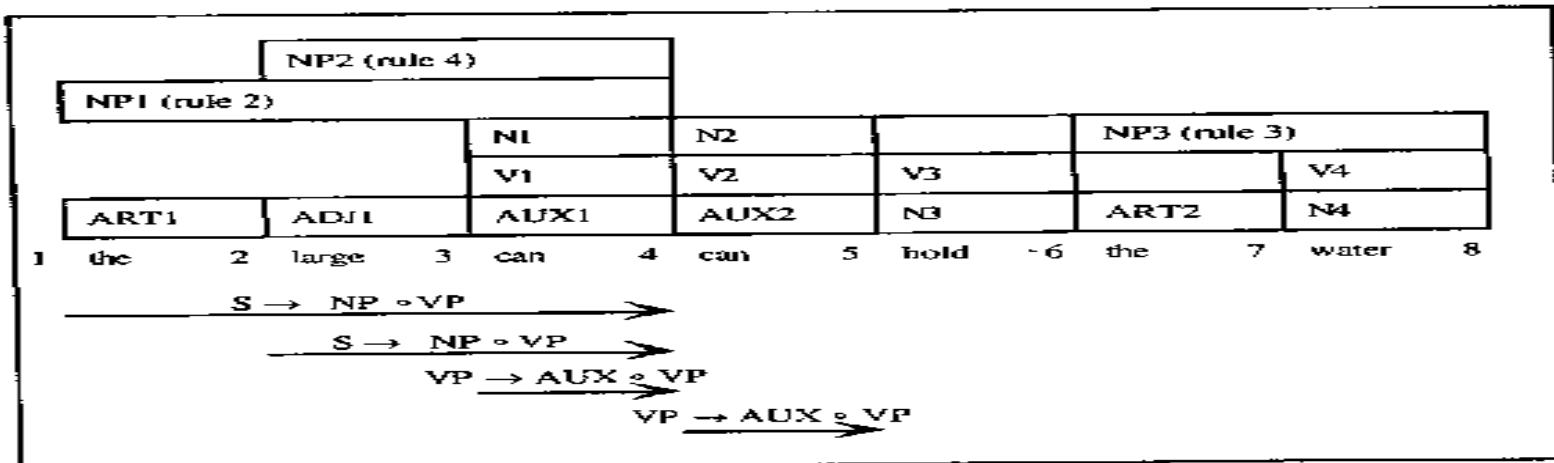
1 The 2 large 3 can 4 can 5 hold 6 the 7 water 8

Example

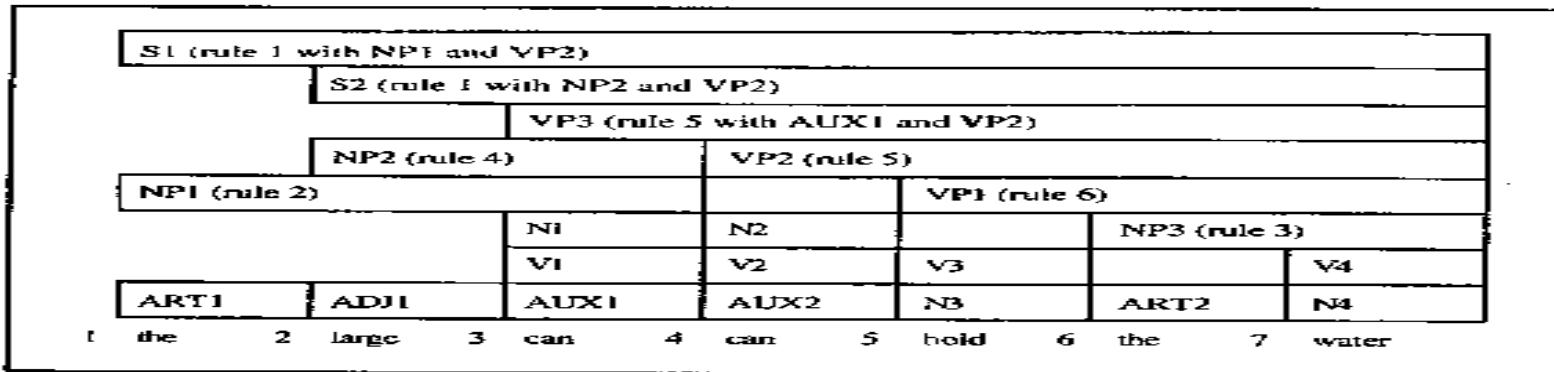


The chart after adding *hold*, omitting arcs generated for the first NP

Example



The chart after all the NPs are found, omitting all but the crucial active arcs.



The final chart