Chapter 3: Grammar and Parsing

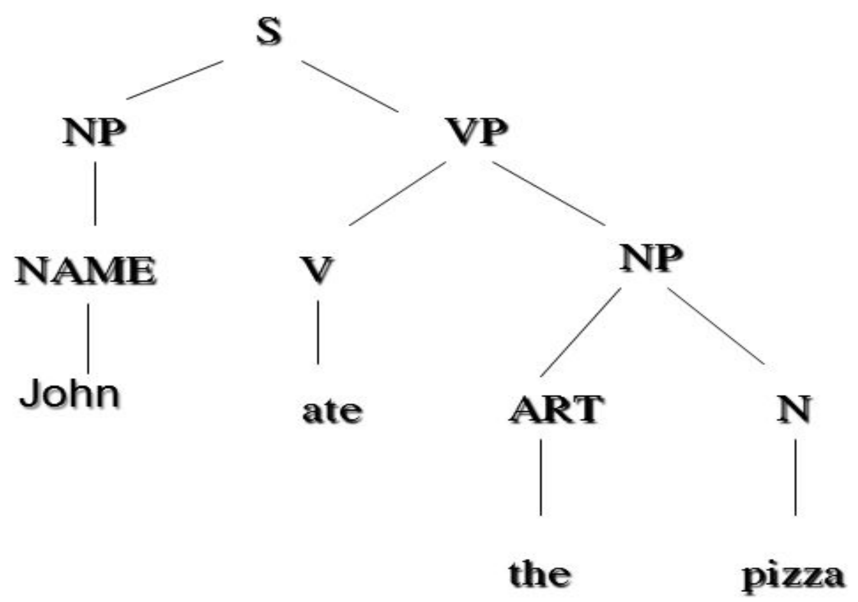
Context-Free Grammars

(CFG)

3.1 Context-Free Grammar (CFG)

3.1.1 Grammar and Sentences Structure

The most common method to study the structure of sentence is how sentence is broken into its major subparts and how those subparts are broken up in turn, is as a tree.



Hình 3.1: Syntactic structure of sentence "John ate the pizza"

3.1 Context-Free Grammar (CFG)

3.1.1 Grammar and Sentences Structure

S consists initial noun phrase NP and verb phrase VP.

initial NP is made of NAME *John*. Initial VP is composed of verb *ate* and NP, which consists ART *the* and N *pizza*

The structure of sentence may be representated by other way:

(S (NP (NAME John))

(VP (V ate) (NP (ART the)

(N pizza))))

3.1 Context-Free Grammar (CFG)

3.1.1 Grammar and Sentences Structure

Context Free grammar (CFG)

G =(S, P, N, T)

S: start symbol S,

P: production rules, which have the form: $A \rightarrow \alpha$;

N, T: set of lexical symbols (word categories).

N is set of non- terminal symbols; T is set of terminal symbols.

There are two important process based on derivations: sentence <u>generation</u> and sentence <u>parsing</u>

There are two methods of search (structure of sentence): Top down and Bottom up.

3.1 Context-Free Grammar (CFG)

3.1.2 What makes a Good Grammar

-To construct a grammar for language, we are interested in ***generality,*** the range of sentences the grammar analyzes correctly; ***selectivity,*** the range of non-sentences it identifies as problematic, and ***understandability***, the. simplicity of grammar itseft

Beginning of small grammars, such as those that describer only few types of sentences, one structural analysis of a sentence may appear as understandable as another

3.1 Context-Free Grammar (CFG)

3.1.2 What makes a Good Grammar

Then we attempt to extend a grammar to cover a wide range of sentences, however we often find one analysis is easily extendable, while the other requires complex modification.

- This analysis retains its simplicity and generality as it is extended is more desirable.
- To pay close attention to the way the sentence is divided into its subparts, called constituents.

3.1 Context-Free Grammar (CFG)

3.1.2 What makes a Good Grammar

- By using our intuition we can apply specific tests, as follows.

+ To decide that a group of words forms a particular constituent;

+ Try to construct a new sentence which involves that group of words in conjunction with another group of words classified as the same type of constituent.

- Example:

- *I ate a hamburger and a hot dogs (NP-NP).*

- *I will eat the hamburger and throw away the hot dog (VP-VP)*

3.1 Context-Free Grammar (CFG)

3.1.3 Top down Parser

- Parsing algorithm is a procedure that searches through various ways of grammar rules to find a combination that generates a tree that could be the structure of the input sentence.
- A top down parser starts with the S symbol and attempts to rewrite it into a sequence of terminal symbols that matches the classes (categories) of the words in the input sentence.

3.1 Context-Free Grammar (CFG)

3.1.3 Top down Parser

A Simple Top-Down Parsing Algorithm

The algorithm starts with the initial state ((S) 1) and no backup

1. Select the current state: take the first state of the possibilities list and call it C. If the possibilities list is empty, then the algorithm fails (no successful parse is possible)
2. If C consists of an empty list and the position of word is at the end of sentence, then the algorithm is succed

3.1 Context-Free Grammar (CFG)

3.1.3 Top down Parser

A Simple Top-Down Parsing Algorithm (continue)

3. Otherwise, generate the next possible states.

3.1 If the first symbol on the symbol list C is terminal (lexical symbol),, and the next word in the sentence can be in that class, then create a new state by removing the first symbol from the symbol list C, update the word position and add it to the possibilities list

3.2 If the first symbol of C is non-terminal, generate a new state of each rule in the grammar that can rewrite that non-terminal symbol and add them all to the possibilities list.

3.1 Context-Free Grammar (CFG)

3.1.3 Top down Parser

A Simple Top-Down Parsing Algorithm

Example: Parse the sentence: $_1$***the*** $_2$***dogs*** $_3$***cried***$_4$

Grammar:

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

3.1 Context-Free Grammar (CFG)

3.1.3 Top down Parser

A Simple Top-Down Parsing Algorithm (continue)

Step	current state	back state	note
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
		((ART ADJ N VP)1)	
4	((N VP)2)		matching ART with *the*
		((ART ADJ N VP)1)	
5	((VP)3)		matching N with *dogs*
		((ART ADJ N VP)1)	
6	((V)3)		rewriting VP by rules 4&5
		((V NP)3)	
		((ART ADJ N VP)1)	
7	(( )4)		The parser succeeds as maching V with ***cried***, leaving an empty grammatical symbol list with an empty input sentence

Figure 3.2: Top Down Depth first parser of *"The dog cried"*

3.1 Context-Free Grammar (CFG)

3.1.3 Top down Parser

Parsing as a Search Procedure

The top down parser is described as a search procedure that implements as follows.

The possibilities list is initially set to the start state S of the parser.

1. Select the first state from possibilities list, and remove it from the list.
2. Generate new states from a current state by trying every possible option from the selected (there may be none if we on a bad path).
3. Add the states generated in the step 2 to the possibilities list, repeat the step 1.

3.1 Context-Free Grammar (CFG)

3.1.3 Top down Parser

Parsing as a Search Procedure

Step	Current state	Backup state	comment
01	((S) 1)		
02	((NP VP) 1)		S rewritten to NP VP
03	((ART N VP) 1)	((ART ADJ N VP) 1)	NP rewritten producing two new states
04	((N VP) 2)	((ART ADJ N VP) 1)	
05	((VP) 3)	((ART ADJ N VP) 1)	The backup state remains
06	((V) 3)	((V NP) 3) ((ART ADJ N VP) 1)	
07	(( ) 4)	((V NP) 3) ((ART ADJ N VP) 1)	
08	((V NP) 3)	((ART ADJ N VP) 1)	The first backup is chosen
09	((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

Figure 3.3:. A Top Down Parse of "The old man ried"

3.1 Context-Free Grammar (CFG)

3.1.3 Top down Parser

Parsing as a Search Procedure

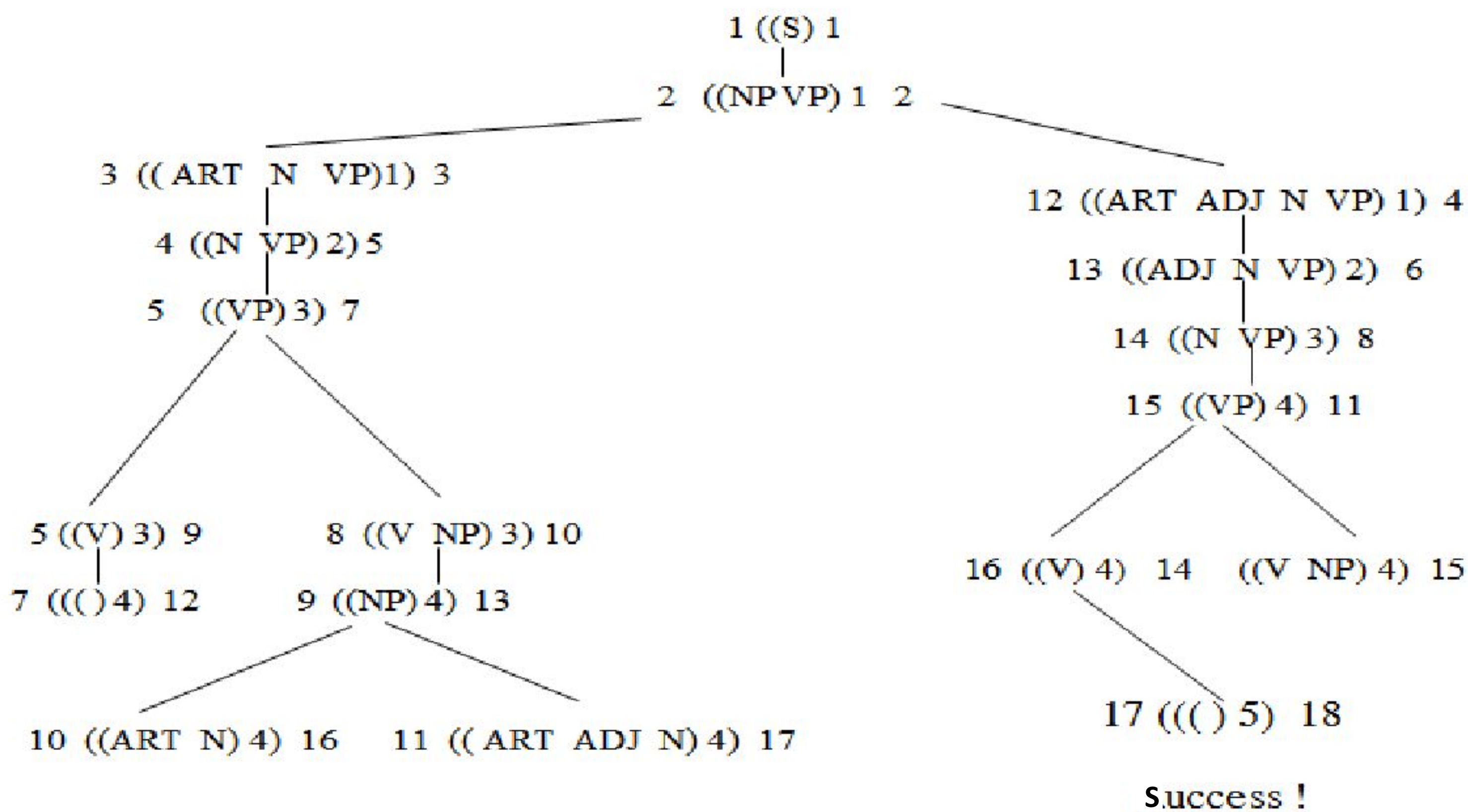


Figure 3.4:. Depth first strategy and breadth first strategy

3.1.4 A bottom-Up Chart Parser

The main difference between top-down and bottom-up parser is

the way the grammar rules are used.

❖ The extension algorithm

Add a constituent C from position p1 to p2:

1. Insert C into chart from p1 to p2;

2. For any active arc of the form: X □ X_1 … • C … X_n from p_0 to p_1, add a new active arc X □ X_1 … C •…X from p_0 to p_2

3. For any active arc of the form: X □ X_1 … • C from p_0 to p_1, add a new constituent of type X from p0 to p2 to the agenda.

3.1 Context-Free Grammar (CFG)

3.1.4 A bottom-Up Chart Parser

- **A Bottom-up Chart Parsing Algorithm**

- Do until there is no input left.

1. If the agenda is empty, then look up the interpretations (categories) of the new word in the input and add them to the agenda.

2. Select a constituent form the agenda (let's call it constituent C from p1 to p2).

3. For any grammar rule X CX1…Xn, add active arc of the form X $\bullet$ C $X_1 \ldots X_n$ from p1 to p2.

4. Add C to the chart by the extension algorithm.

--Example: To parse a sentence "***The large can can hold the water***"

3.1 Context-Free Grammar (CFG)

3.1.4 A bottom-Up Chart Parser

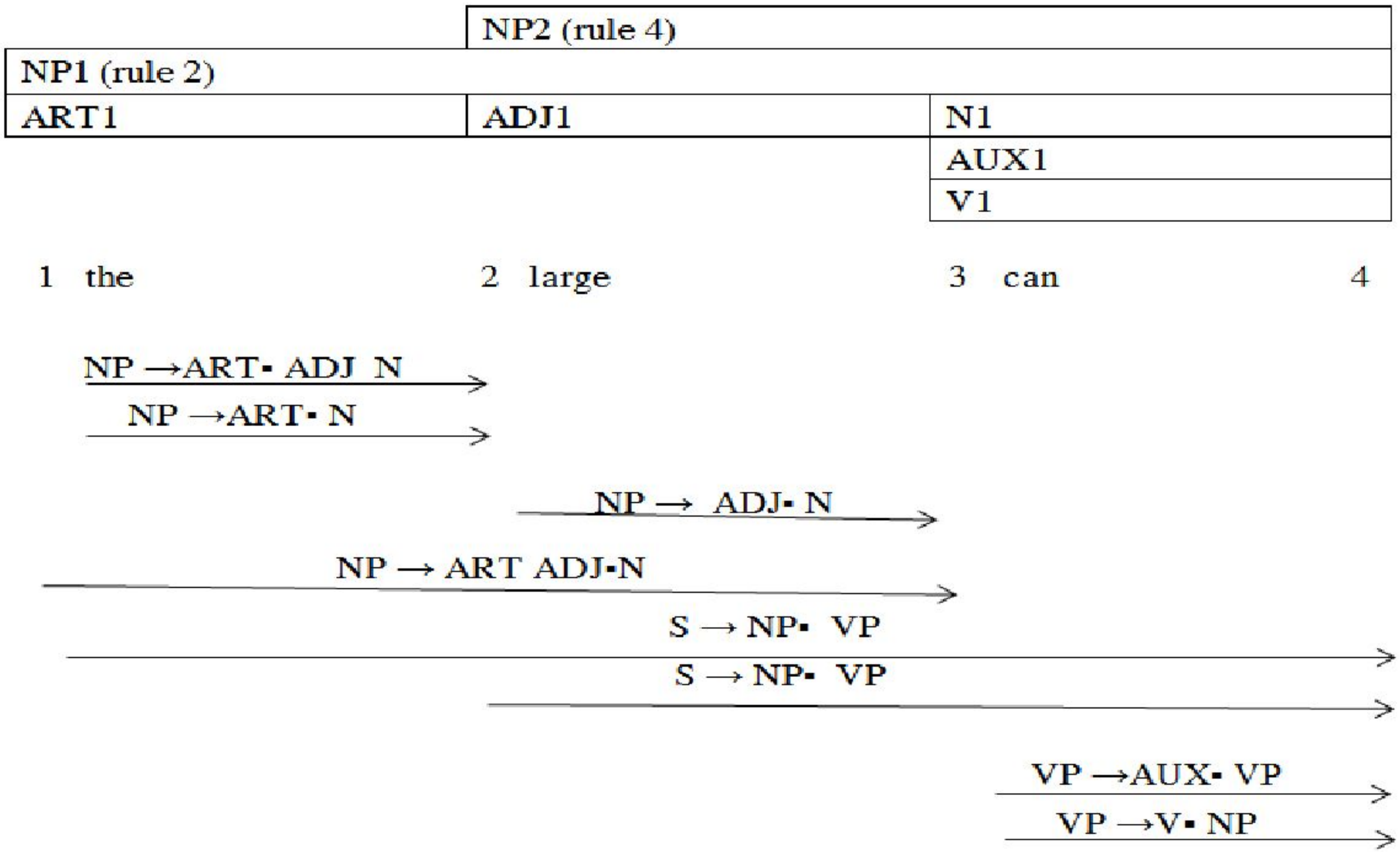


Figure 3.5: After parsing *the large can*

3.1 Context-Free Grammar (CFG)

3.1.4 A bottom-Up Chart Parser

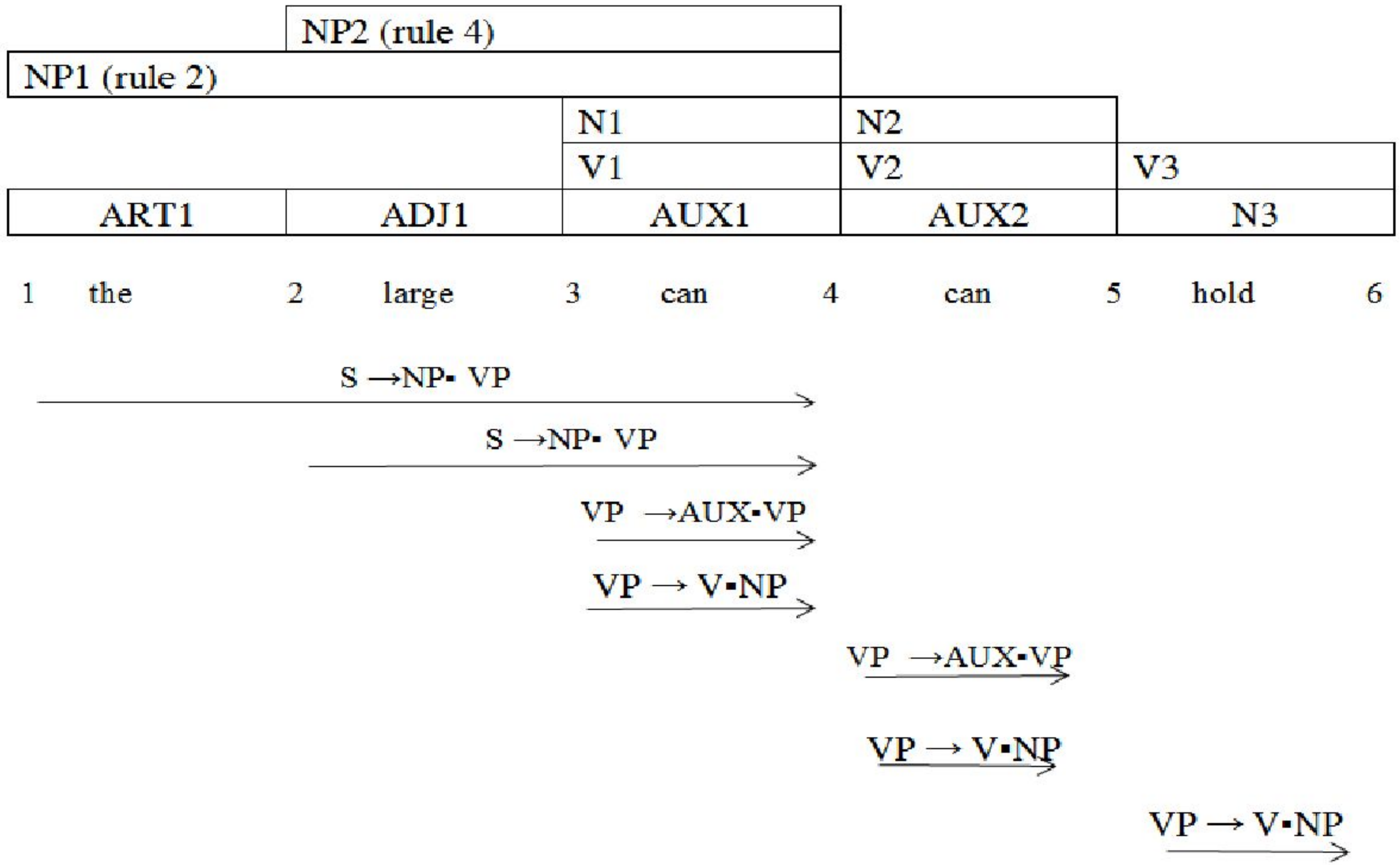


Figure 3.6: The chart after adding *hold,* omitting arcs generated for the first NP

3.1 Context-Free Grammar（CFG）

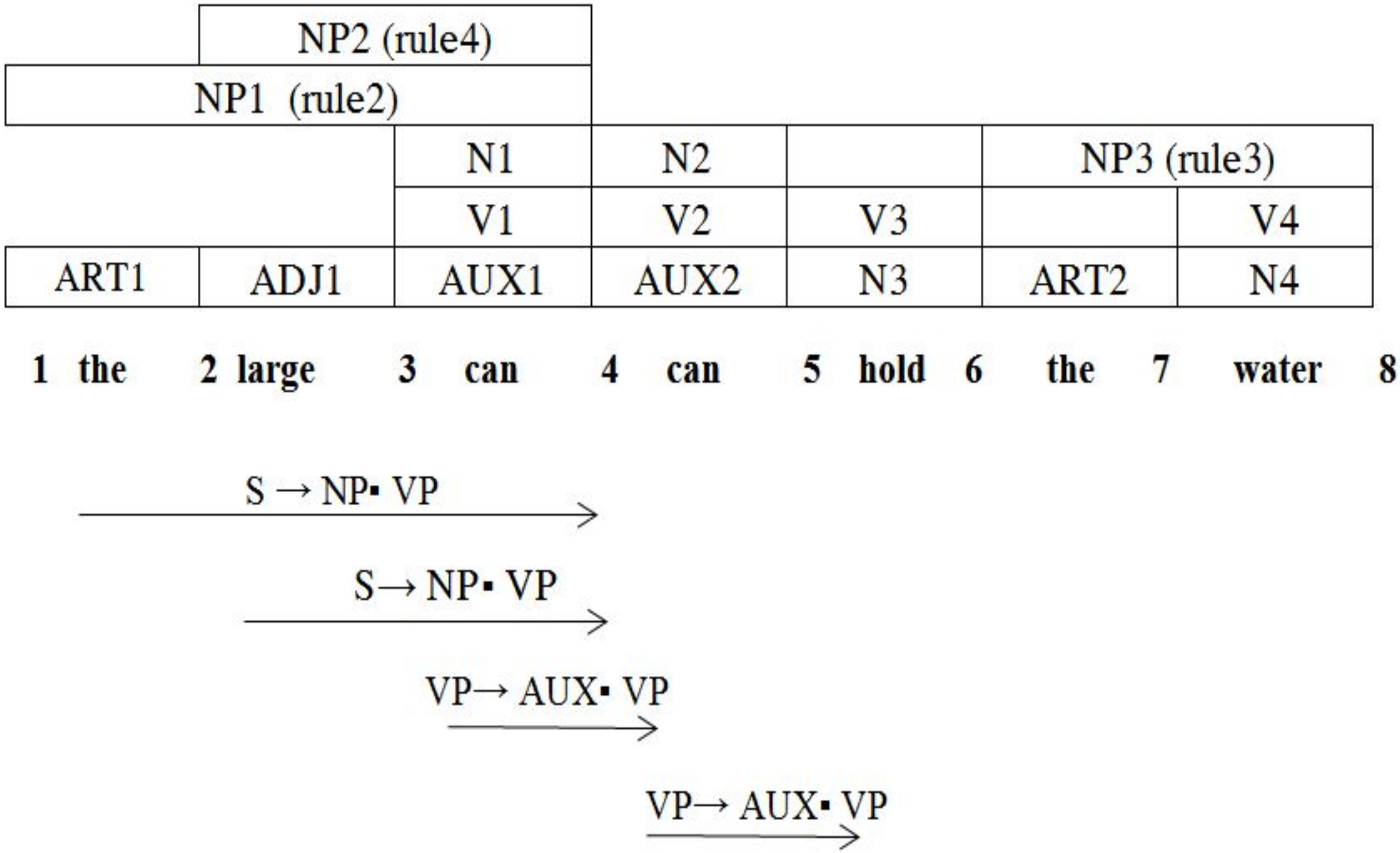


Figure 3.7: The chart after the NPs are found, omitting all but the crucial active arcs

3.1 Context-Free Grammar (CFG)

3.1. 4 A bottom-Up Chart Parser

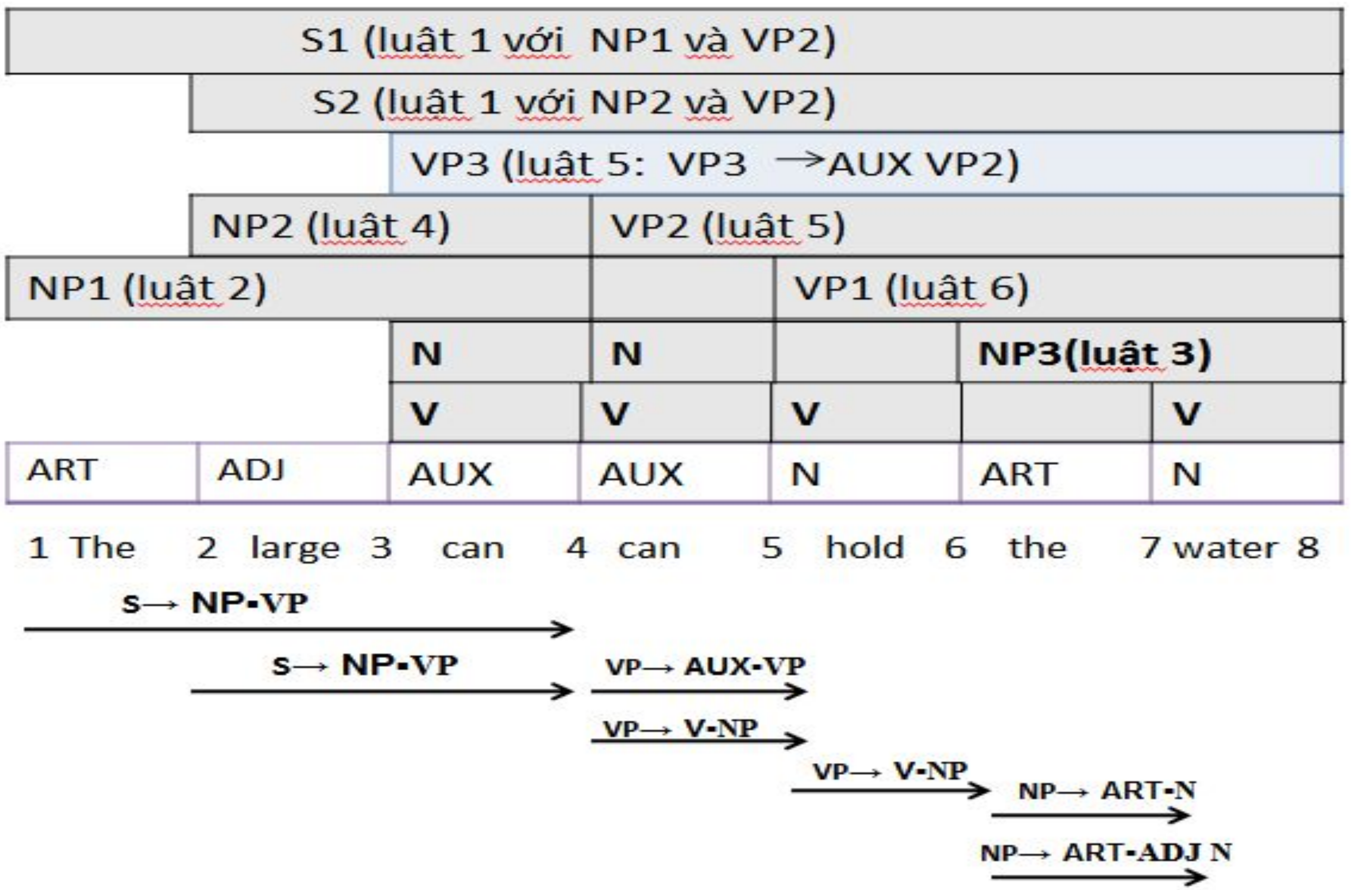


Figure 3.8: The final chart

3.1 Context-Free Grammar (CFG)

3.1. 4 A bottom-Up Chart Parser

Efficiency Considerations

- Chart –based parser can be considerably more efficient than parsers that rely only on a search because the same constituent never is constructed more than once.
- The complexity of the pure top down or bottom up parser could require up to C^n operations to parse a sentence of the length n, C is a constant that depends on specific algorithm we use.
- The complexity of the chart-based parser is $K*n^3$. K is a constant that depends on the algorithm, n is the length of sentence.
- The chart based parser would be up to the many times faster than pure parser.

3.1 Context-Free Grammar (CFG)

3.1.5 Top-Down Chart Parsing

So far we have seen a simple top-down method and a bottom-up chart-based method for parsing context free grammars.

Now a new method is presented actually captures the advantages of both, that is ***top-down chart parser*.**

❖ ***Top - down arc Introduction Algorithm***

To add an arc $S \square C_1 \ldots \bullet C_i \ldots C_n$ ending at position j,

do the following.

For any rule in the grammar of the form $C_i \square X_1 \ldots X_k$, recursively add new arc $C_i \square \bullet X_1 \ldots X_k$ from position j to j.

3.1 Context-Free Grammar (CFG)

3.1.5 Top-Down Chart Parsing

Top-Down Chart Parsing Algorithm

Initialization: for every rule in the grammar of the form $S \square X_1 \ldots X_k$ add an arc ;labeled $S \square \bullet X_1 \ldots X_k$ using the arc introduction algorithm.

Parsing: Do until there is no input left

1. If agenda is empty, look up the interpretation of the next word and add them to the agenda.
2. Select constituent from the agenda (call it constituent C).
3. Using the arc extension algorithm, combine C with every active arc on the chart. Any new constituents are added to the agenda.
4. For any active arcs created in the step 3, add them to the chart using the top-down arc introduction algorithm.

3.1 Context-Free Grammar (CFG)

3.1.5 Top-Down Chart Parsing

Example: $_1$the $_2$ large $_3$can $_4$can $_5$hold $_6$the $_7$water$_8$

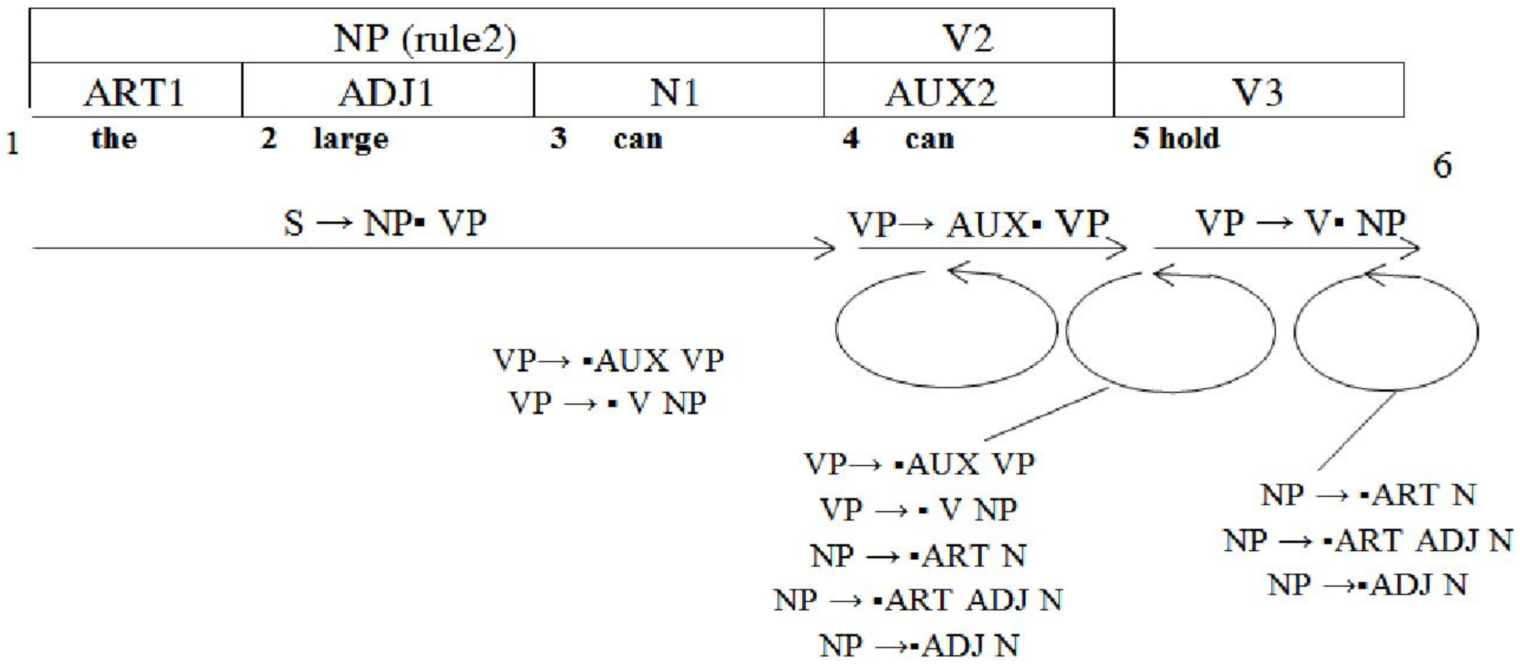


Figure 3.14: The chart after adding *hold*, omitting arcs generated for the first NP

3.1 Context-Free Grammar (CFG)

3.1.5 Top-Down Chart Parsing

Example: $_1$the $_2$ large $_3$can $_4$can $_5$hold $_6$the $_7$water$_8$

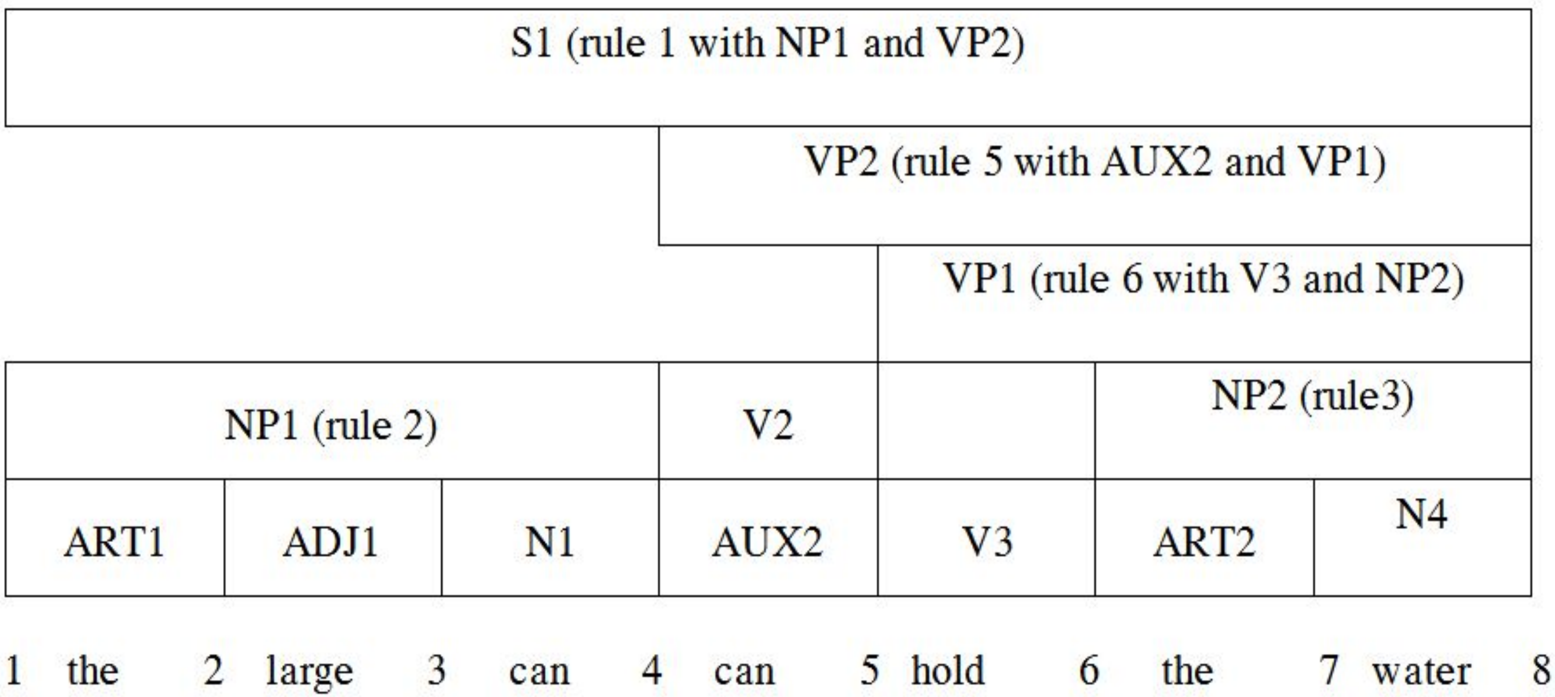


Figure 3.15: The final chart for top-down filtering algorithm

The Penn Treebank

```
( (S
    (NP-SBJ (DT The) (NN move))
    (VP (VBD followed)
      (NP
        (NP (DT a) (NN round))
        (PP (IN of)
          (NP
            (NP (JJ similar) (NNS increases))
            (PP (IN by)
              (NP (JJ other) (NNS lenders)))
            (PP (IN against)
              (NP (NNP Arizona) (JJ real) (NN estate) (NNS loans))))))
      (, ,)
      (S-ADV
        (NP-SBJ (-NONE- *))
        (VP (VBG reflecting)
          (NP
            (NP (DT a) (VBG continuing) (NN decline))
            (PP-LOC (IN in)
              (NP (DT that) (NN market)))))))
    (. .)))
```

Chapter 3:
Grammar and Parsing

Probabilistic Context-Free Grammars (PCFG)

3.2 Probabilistic – or stochastic – context-free grammars (PCFGs)

3.2.1 PCFGs

- G = (T, N, S, R, P)
  -
  -
  -
  -
  - P is a probability function
    - P: R $\longrightarrow$ [0,1]
    - A grammar G generates a language model L.

$$\sum_{\gamma \in T^*} P(\gamma) = 1$$

3.2 Probabilistic – or stochastic – context-free grammars (PCFGs)

3.2.1 PCFGs

$S \longrightarrow NP\ VP$	1.0
$VP \longrightarrow V\ NP$	0.6
$VP \longrightarrow V\ NP\ PP$	0.4
$NP \longrightarrow NP\ NP$	0.1
$NP \longrightarrow NP\ PP$	0.2
$NP \longrightarrow N$	0.7
$PP \longrightarrow P\ NP$	1.0

$N \longrightarrow$ *people*	0.5
$N \longrightarrow$ *fish*	0.2
$N \longrightarrow$ *tanks*	0.2
$N \longrightarrow$ *rods*	0.1
$V \longrightarrow$ *people*	0.1
$V \longrightarrow$ *fish*	0.6
$V \longrightarrow$ *tanks*	0.3
$P \longrightarrow$ *with*	1.0

3.2.2 The probability of trees and strings

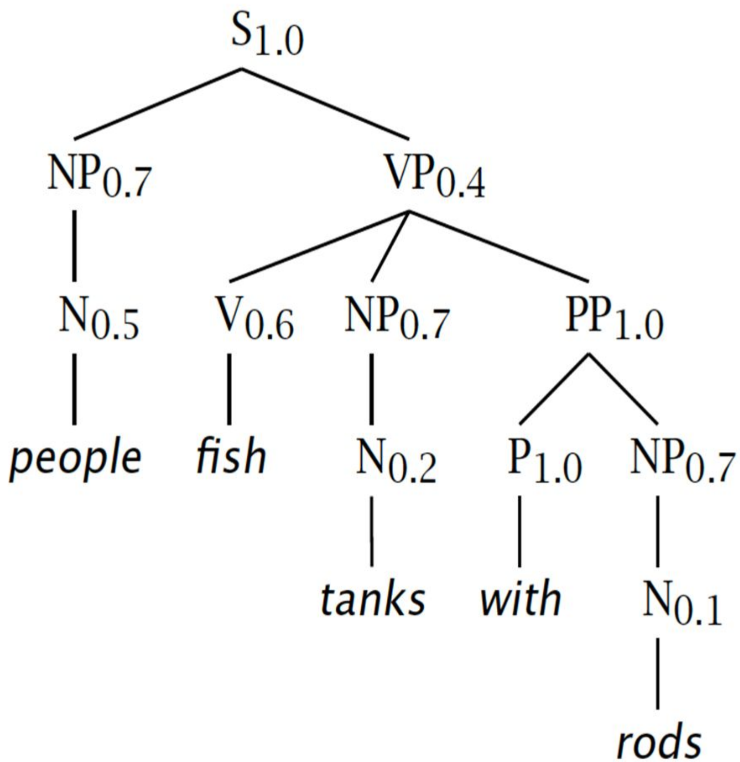
- $P(t)$ – The probability of a tree t is the product of the probabilities of the rules used to generate it.
- $P(s)$ – The probability of the string s is the sum of the probabilities of the trees which have that string as their yield

$$P(s) = \Sigma_j P(s, t) \quad \text{where } t \text{ is a parse of } s$$

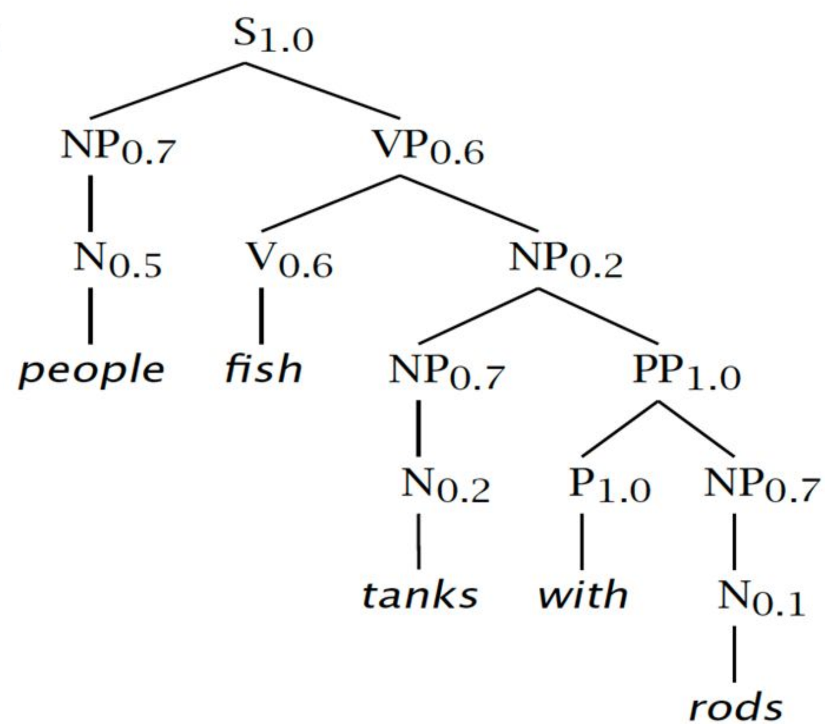
$$= \Sigma_j P(t)$$

3.2.2 The probability of trees and strings

t_1:



t_2:



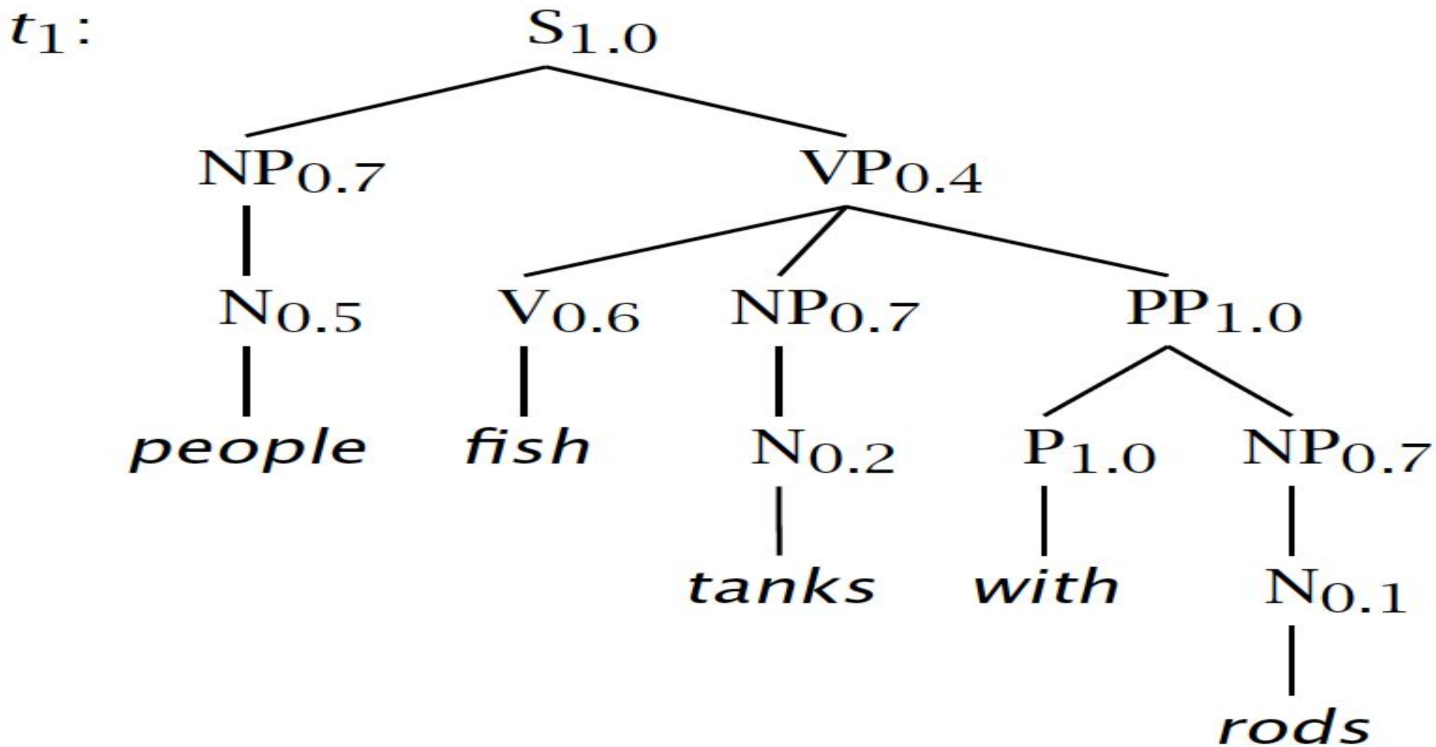
3.2.2 The probability of trees and strings

- s = *people fish tanks with rods*
- $P(t_1) = 1.0 \times 0.7 \times \underline{0.4} \times 0.5 \times 0.6 \times 0.7$
  $\times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1$
  $= 0.0008232$
- $P(t_2) = 1.0 \times 0.7 \times \underline{0.6} \times 0.5 \times 0.6 \times \underline{0.2}$
  $\times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1$
  $= 0.00024696$
- $P(s) = P(t_1) + P(t_2)$
  $= 0.0008232 + 0.00024696$
  $= 0.00107016$

Verb attach

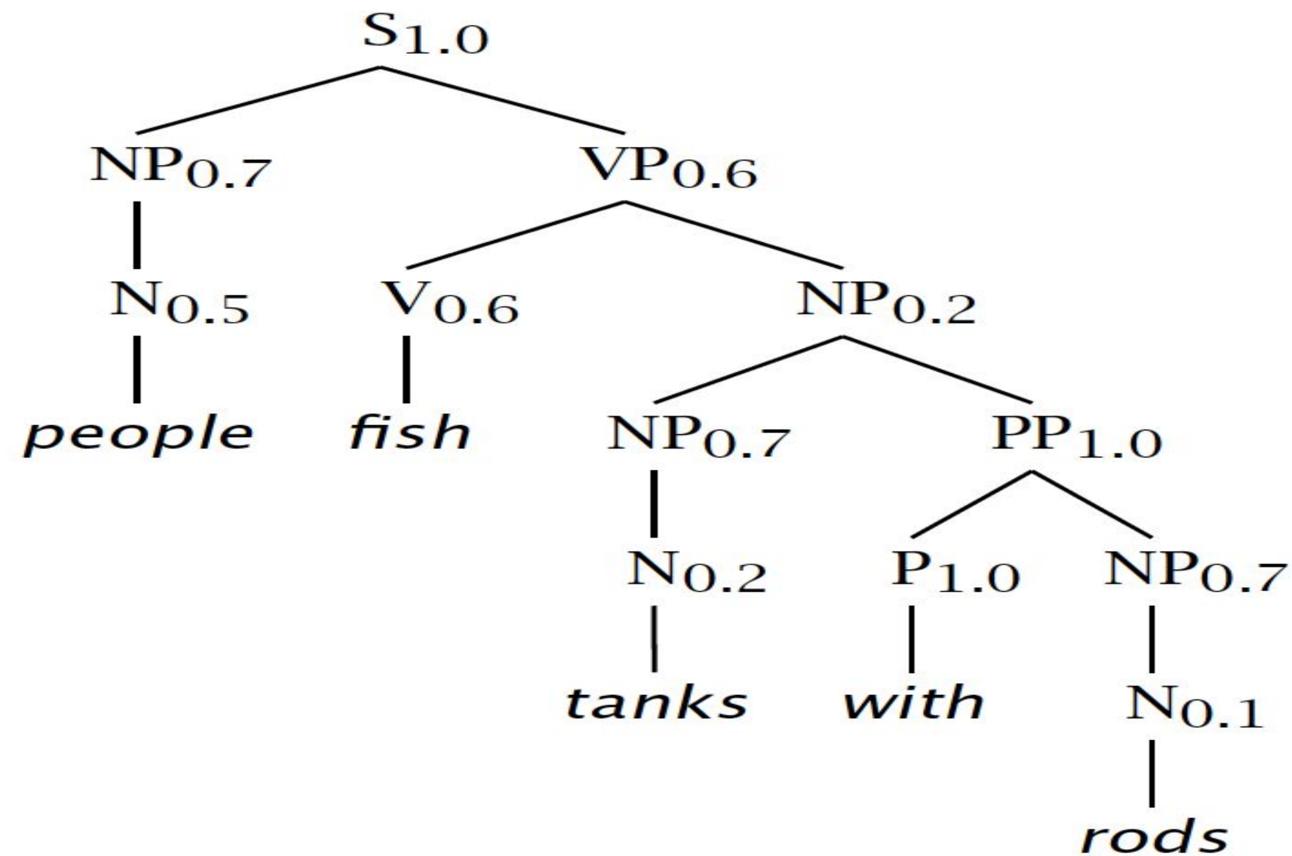
Noun attach

3.2.2 The probability of trees and strings



3.2.2 The probability of trees and strings

t_2:



Chapter 3: Grammar and Parsing

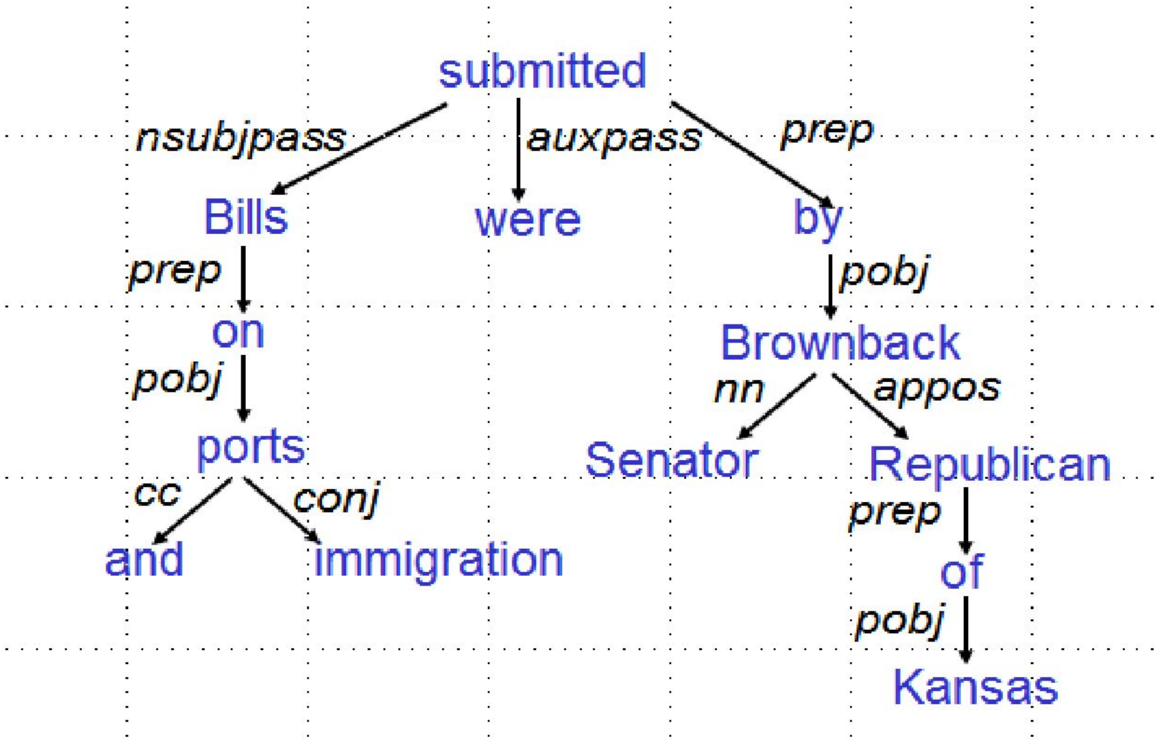
Dependency Parsing

3.3 Dependency Grammar and Dependency Structure

3.3.1 Dependency relation

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations ("arrows") called dependencies

The arrows are commonly typed with the name of grammatical relations (subject, prepositional object, apposition, etc.)



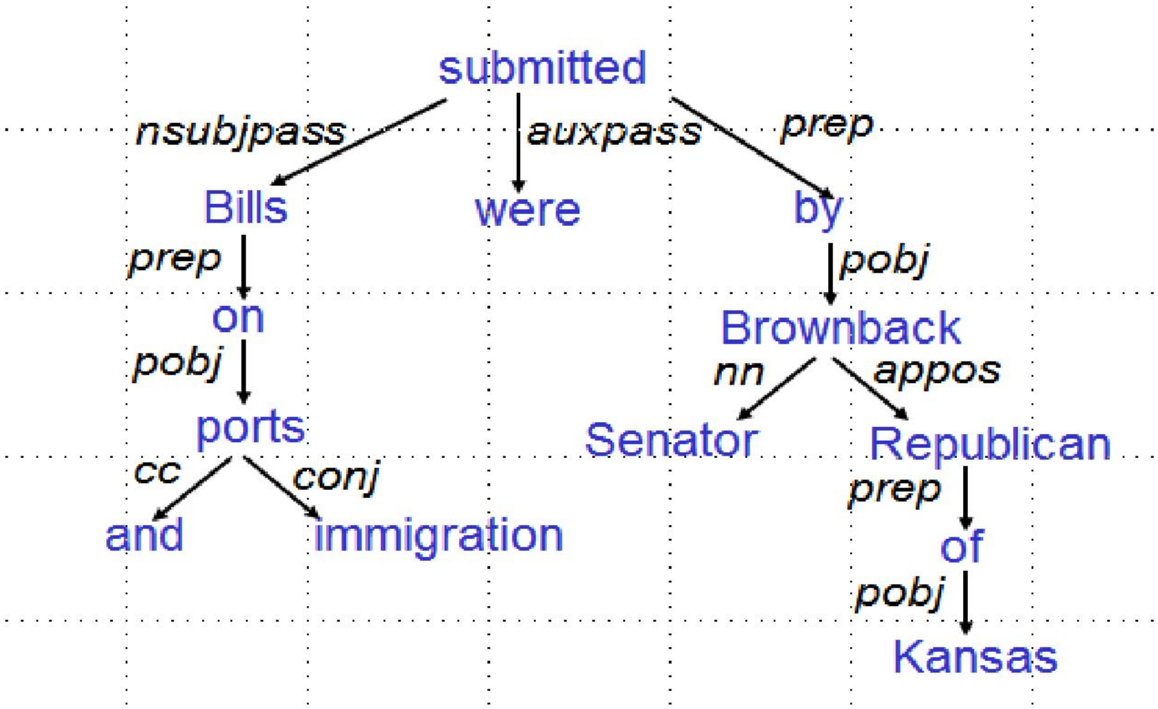
3.3 Dependency Grammar and Dependency Structure

3.3.1 Dependency relation

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations (“arrows”) called dependencies

The arrow connects a head (governor, superior, regent) with a dependent (modifier, inferior, subordinate)

Usually, dependencies form a tree (connected, acyclic, single-head)



3.3 Dependency Grammar and Dependency Structure

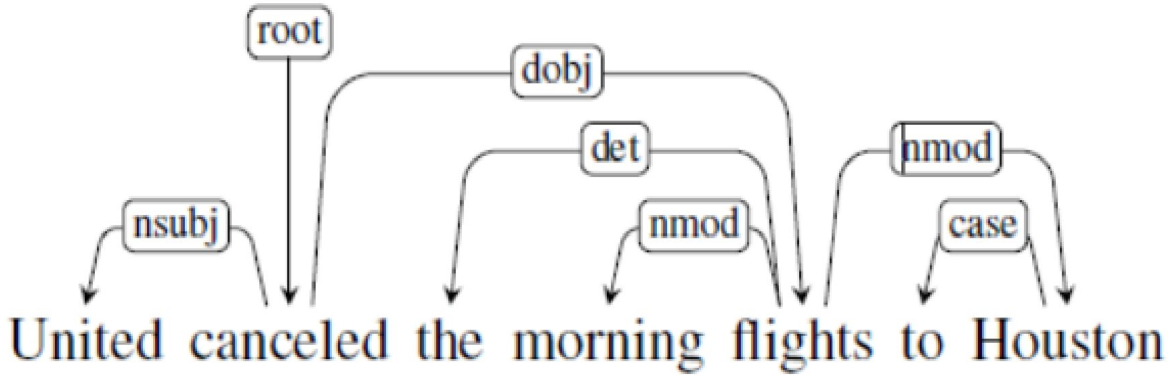
3.3.1 Dependency relation

Clausal Argument relations	**Description**
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal Complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
Other Notable Relations	**Description**
CONJ	Conjunct

Selected dependency relations from Universal Dependency set

3.3 Dependency Grammar and Dependency Structure

Example of dependency structure with universal dependency relations:



Clausal relations NSUBJ and DOBJ identify the subject and direct object of the predicate cancel, while NMOD, DET and CASE relations denote modifiers of the nouns flights and Houston

3.3 Dependency Grammar and Dependency Structure

3.3.2 Dependency formalisms

Dependency structure is directed graph:

G = (V, A) V: set of vertices A: set of ordered pairs of vertices, We will refer A as arcs.

- V corresponds exactly the set of words in the given sentence.
- A captures the head-dependent and grammatical function relationship between the elements in V

Dependency Tree

Dependency tree is directed graph that satisfies the following constraints:

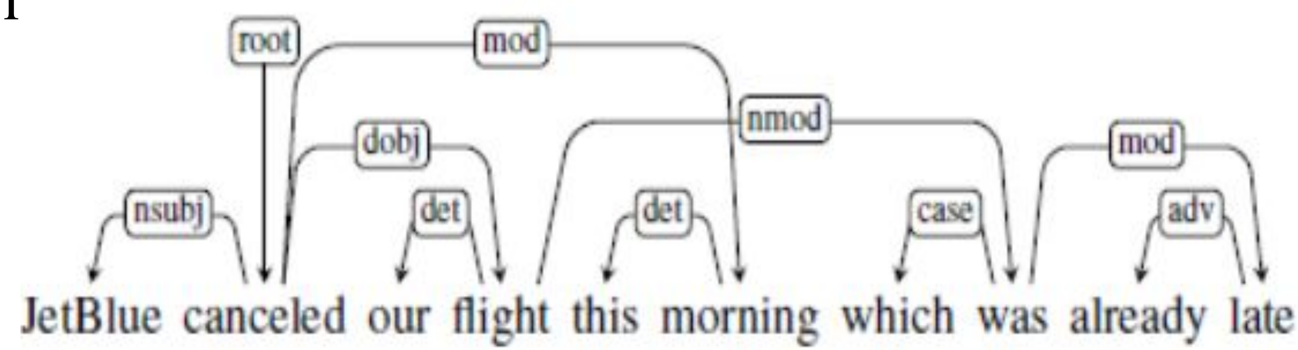
1. There is single designated root node that has no incoming arcs.
2. With the exception of root node, each vertex has exactly one incoming arc.
3. There is a unique path from the root node to each vertex in V.

3.3 Dependency Grammar and Dependency Structure

3.3.2 Dependency formalisms

Projectivity

- An arc from a head to a dependent is said to be projective if there is a path from the head to every word that lies between the head and the dependent in the sentence.
- A dependency tree is then said to be projective if all the arcs that make it up are projective.
- Consider the following example:

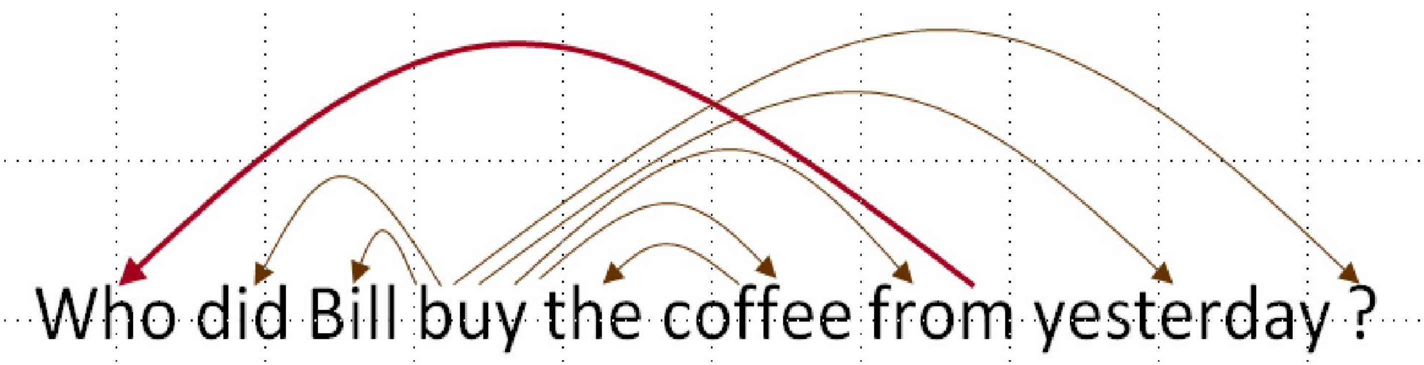


Arc from *flight* to its modifier *was* is non- projective, sine there is no path from *flight* to intervening words *this* and *morning*.

3.3.2 Dependency formalisms

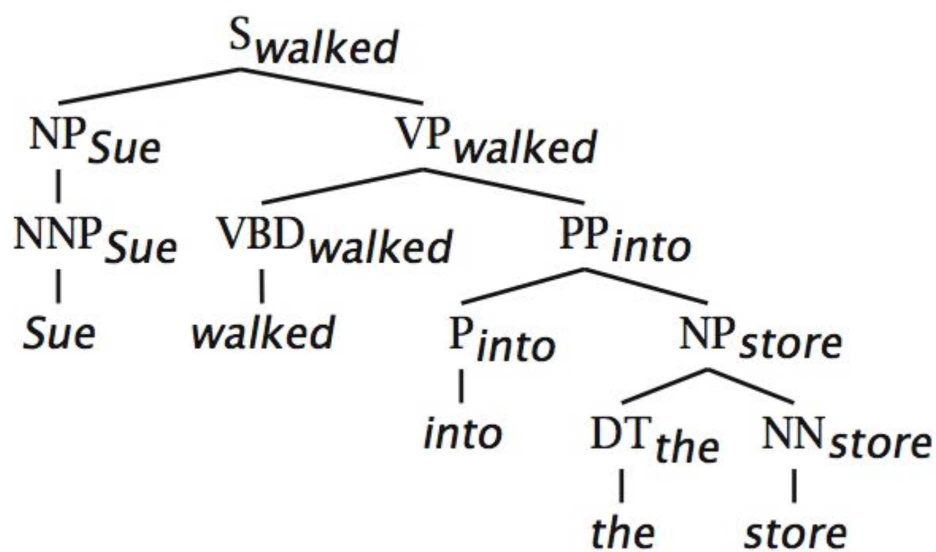
Projectivity

- Dependencies from a CFG tree using heads, must be projective
  - There must not be any crossing dependency arcs when the words are laid out in their linear order, with all arcs above the words.
- But dependency theory normally does allow non-projective structures to account for displaced constituents
  - You can't easily get the semantics of certain constructions right without these nonprojective dependencies



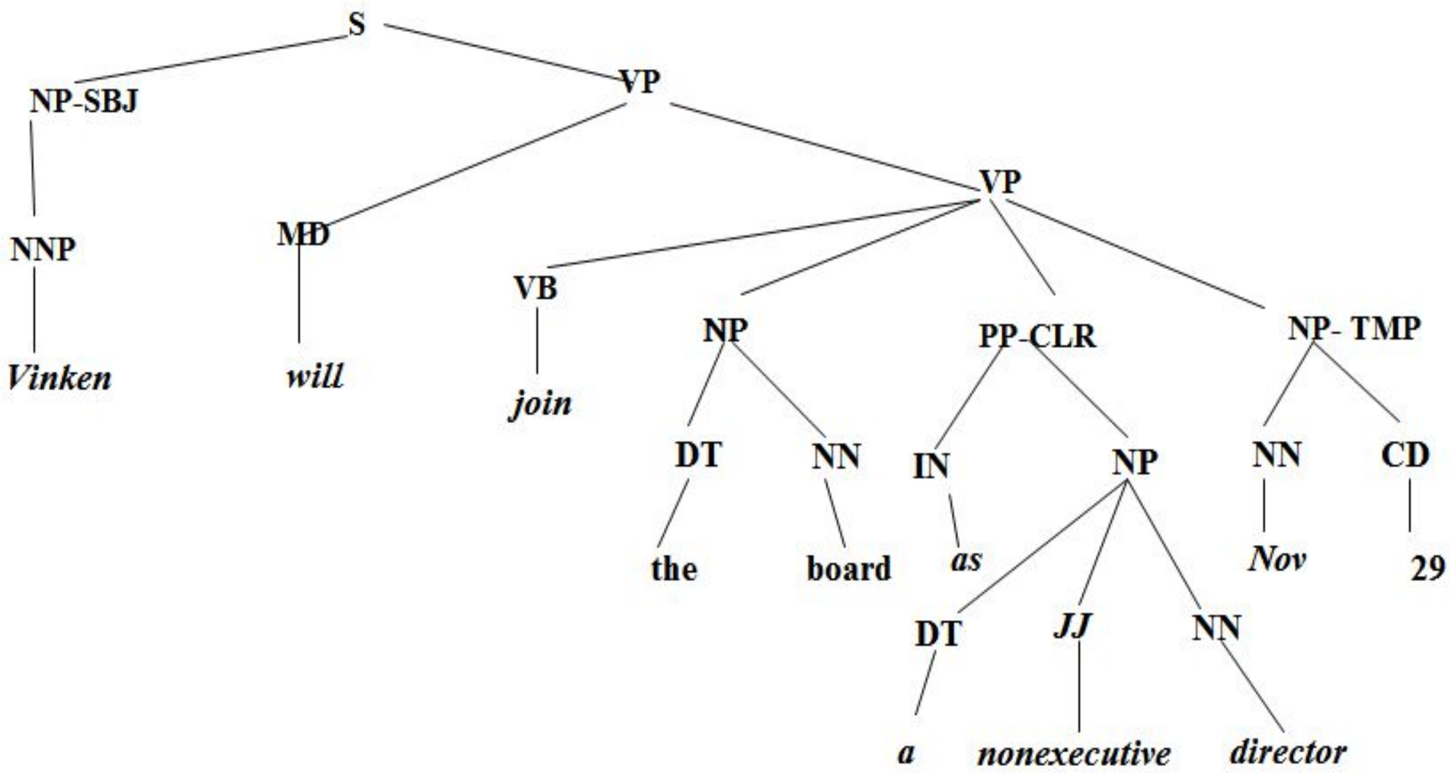
3.3.3 Relation between phrase structure and dependency structure

- A dependency grammar has a notion of a head. Officially, CFGs don't.
- But modern linguistic theory and all modern statistical parsers (Charniak, Collins, Stanford, …) do, via hand-written phrasal "head rules":
  - The head of a Noun Phrase is a noun/number/adj/…
  - The head of a Verb Phrase is a verb/modal/….
- The head rules can be used to extract a dependency parse from a CFG parse



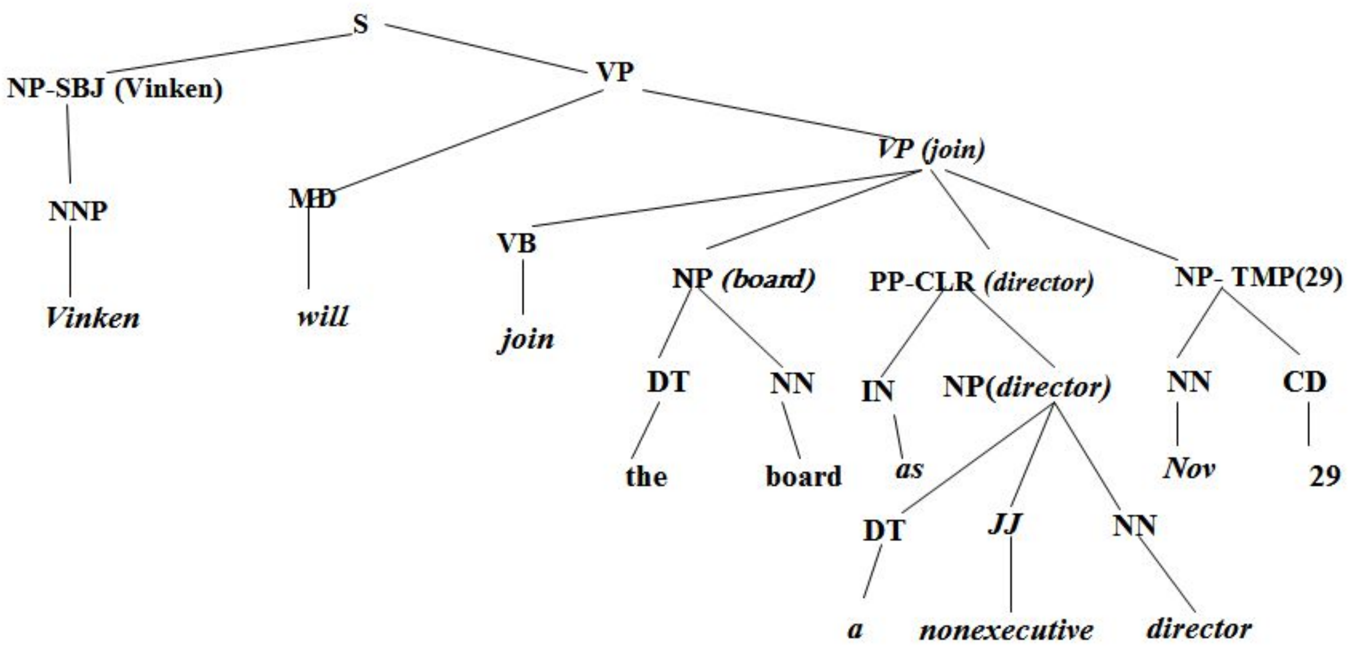
- The closure of dependencies give constituency from a dependency tree
- But the dependents of a word must be at the same level (i.e., "flat") – there can be no VP!

3.3.3 Relation between phrase structure and dependency structure



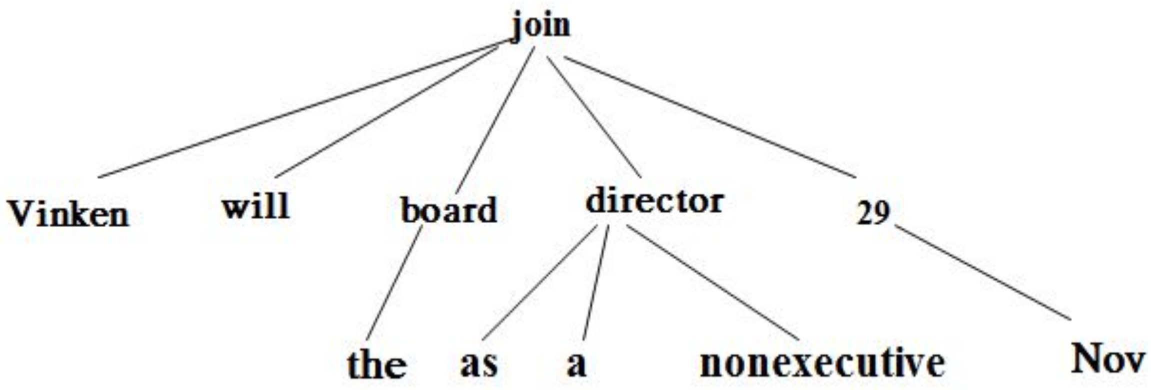
a*) Example of a phrase structure "Vinken will join the board as a nonexecutive director Nov 29*

3.3.3 Relation between phrase structure and dependency structure



b) To translate a) structure to dependency structure

3.3.3 Relation between phrase structure and dependency structure

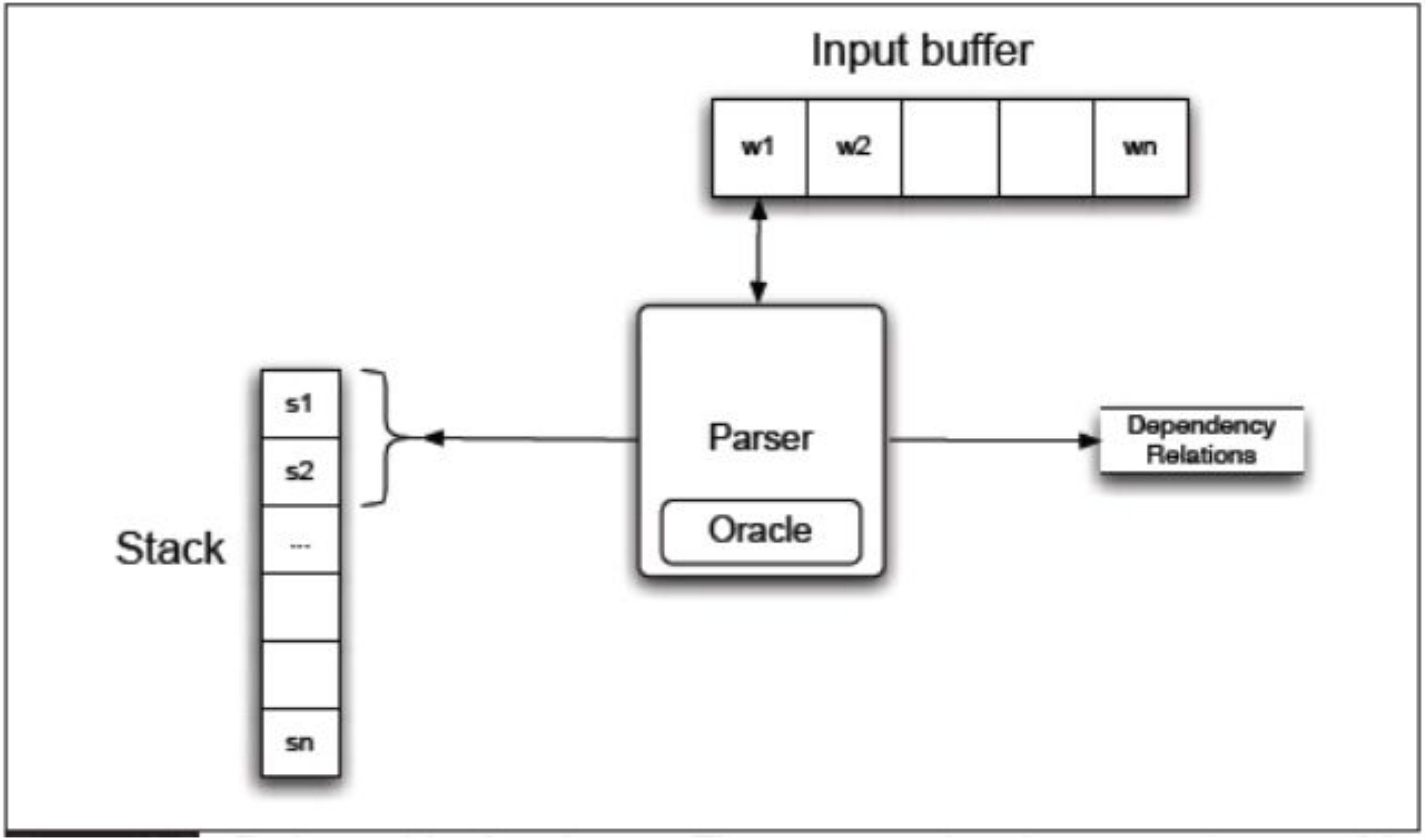


c) Dependency structure of sentence

3.3.4 Transition – Based Dependency parsing

- Dependency parsing is motivated by a stack –based approach called ***shift-reduce parsing.***
- ***Configuration*** consists of stack, an input buffer of words, or tokens and a set of relations representing a dependency tree.
- Parsing process consists of a sequence of transitions through the space of possible configurations.

3.3.4 Transition – Based Dependency parsing



Basic transition- based dependency parser

Quiz question

- Consider this sentence:

Retail sales drop in April cools afternoon market trading.

- Which word are these words a dependent of?
  1. sales
  2. April
  3. afternoon
  4. trading

3.3.5 MaltParser
[Nivre et al. 2008]

- A simple form of greedy discriminative dependency parser
- The parser does a sequence of bottom up actions
  - Roughly like "shift" or "reduce" in a shift-reduce parser, but the "reduce" actions are specialized to create dependencies with head on left or right
- The parser has:
  - a stack σ, written with top to the right
    - which starts with the ROOT symbol
  - a buffer β, written with top to the left
    - which starts with the input sentence
  - a set of dependency arcs A
    - which starts off empty
  - a set of actions

3.3.5 MaltParser

[Nivre et al. 2008]

Basic transition-based dependency parser

Start: σ = [ROOT], $\beta = w_1, \ldots, w_n$, A = $\varnothing$

1. Shift $\quad \sigma, w_i|\beta, A$ □ $\sigma|w_i, \beta, A$
2. Left-Arc$_r$ $\quad \sigma|w_i, w_j|\beta, A$ □ $\sigma, w_j|\beta, A \cup \{r(w_j,w_i)\}$
3. Right-Arc$_r$ $\quad \sigma|w_i, w_j|\beta, A$ □ $\sigma, w_i|\beta, A \cup \{r(w_i,w_j)\}$

Finish: $\beta = \varnothing$

Notes:

- Unlike the regular presentation of the CFG reduce step, dependencies combine one thing from each of stack and buffer

3.3.5MaltParser
[Nivre et al. 2008]

Actions ("arc-eager" dependency parser)

Start: $\sigma = [\text{ROOT}], \beta = w_1, \ldots, w_n, A = \varnothing$

1. Left-Arc$_r$ $\quad \sigma|w_i, w_j|\beta, A \Rightarrow \sigma, w_j|\beta, A \cup \{r(w_j,w_i)\}$

   Precondition: $r'(w_k, w_i) \notin A, w_i \neq \text{ROOT}$

2. Right-Arc$_r$ $\quad \sigma|w_i, w_j|\beta, A \Rightarrow \sigma|w_i|w_j, \beta, A \cup \{r(w_i,w_j)\}$
3. Reduce $\quad \sigma|w_i, \beta, A \Rightarrow \sigma, \beta, A$

   Precondition: $r'(w_k, w_i) \in A$

4. Shift $\quad \sigma, w_i|\beta, A \Rightarrow \sigma|w_i, \beta, A$

Finish: $\beta = \varnothing$

This is the common "arc-eager" variant: a head can immediately take a right dependent, before *its* dependents are found

Exmple: Happy children like to play with their friends

1. Left-Arc$_r$ $\sigma|w_i, w_j|\beta, A$ □ $\sigma, w_j|\beta, A \cup \{r(w_j,w_i)\}$
   Precondition: $(w_k, r', w_i) \notin A$, $w_i \neq$ ROOT
2. Right-Arc$_r$ $\sigma|w_i, w_j|\beta, A$ □ $\sigma|w_i|w_j, \beta, A \cup \{r(w_i,w_j)\}$
3. Reduce $\sigma|w_i, \beta, A$ □ σ, β, A
   Precondition: $(w_k, r', w_i) \in A$
4. Shift $\sigma, w_i|\beta, A$ □ $\sigma|w_i, \beta, A$

.	[ROOT]	[Happy, children, …]	$\varnothing$
Shift	[ROOT, Happy]	[children, like, …]	$\varnothing$
LA$_{amod}$	[ROOT]	[children, like, …]	{amod(children, happy)} = A_1
			(children $\longrightarrow$ happy)
Shift	[ROOT, children]	[like, to, …]	A_1
LA$_{nsubj}$	[ROOT]	[like, to, …]	$A_1 \cup$ {nsubj(like, children)} = A_2
			(like $\longrightarrow$ children)
RA$_{root}$	[ROOT, like]	[to, play, …]	$A_2 \cup$ {root(ROOT, like) = A_3
Shift	[ROOT, like, to]	[play, with, …]	A_3
LA$_{aux}$	[ROOT, like]	[play, with, …]	$A_3 \cup$ {aux(play, to) = A_4
			(play $\longrightarrow$ to)
RA$_{xcomp}$	[ROOT, like, play]	[with their, …]	$A_4 \cup$ {xcomp(like, play) = A_5
			(like $\longrightarrow$ play)

Example

1. Left-Arc$_r$ $\sigma|w_i, w_j|\beta, A$ □ $\sigma, w_j|\beta, A \cup \{r(w_j,w_i)\}$
   Precondition: $(w_k, r', w_i) \notin A$, $w_i \neq$ ROOT
2. Right-Arc$_r$ $\sigma|w_i, w_j|\beta, A$ □ $\sigma|w_i|w_j, \beta, A \cup \{r(w_i,w_j)\}$
3. Reduce $\sigma|w_i, \beta, A$ □ σ, β, A
   Precondition: $(w_k, r', w_i) \in A$
4. Shift $\sigma, w_i|\beta, A$ □ $\sigma|w_i, \beta, A$

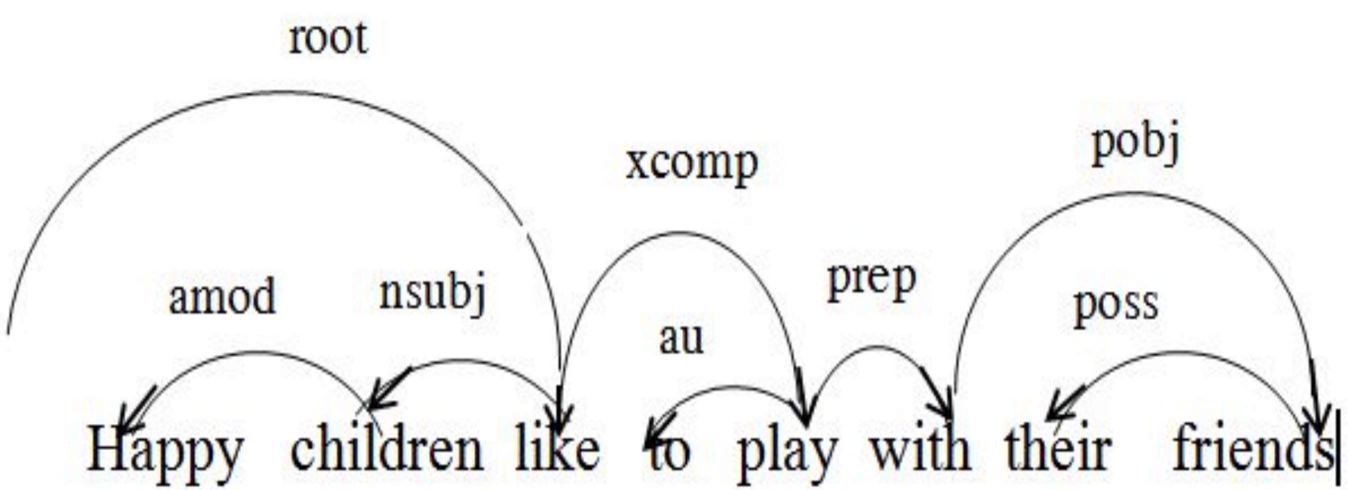
Happy children like to play with their friends .

RA$_{xcomp}$	[ROOT, like, play]	[with their, …]	$A_4 \cup$ {xcomp(like, play) = A_5
RA$_{prep}$	[ROOT, like, play, with]	[their, friends, …]	$A_5 \cup$ {prep(play, with) = A_6
Shift	[ROOT, like, play, with, their]	[friends, .]	A_6
LA$_{poss}$	[ROOT, like, play, with]	[friends, .]	$A_6 \cup$ {poss(friends, their) = A_7
RA$_{pobj}$	[ROOT, like, play, with, friends]	[.]	$A_7 \cup$ {pobj(with, friends) = A_8
Reduce	[ROOT, like, play, with]	[.]	A_8
Reduce	[ROOT, like, play]	[.]	A_8
Reduce	[ROOT, like]	[.]	A_8
RA$_{punc}$	[ROOT, like, .]	[]	$A_8 \cup$ {punc(like, .) = A_9

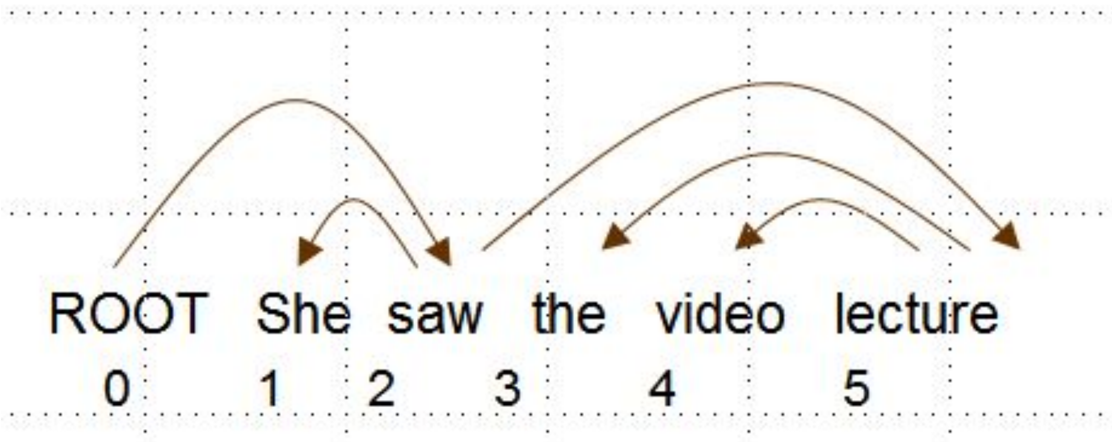
You terminate as soon as the buffer is empty. Dependencies = A_9

3.3.5MaltParser
[Nivre et al. 2008]

Exmple: ***Happy children like to play with their friends***



Evaluation of Dependency Parsing: (labeled) dependency accuracy



$$\text{Acc} = \frac{\#\text{ correct deps}}{\#\text{ of deps}}$$

UAS = 4 / 5 = 80%

LAS = 2 / 5 = 40%

Gold

1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	dobj

Parsed

1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

Dependencies encode relational structure

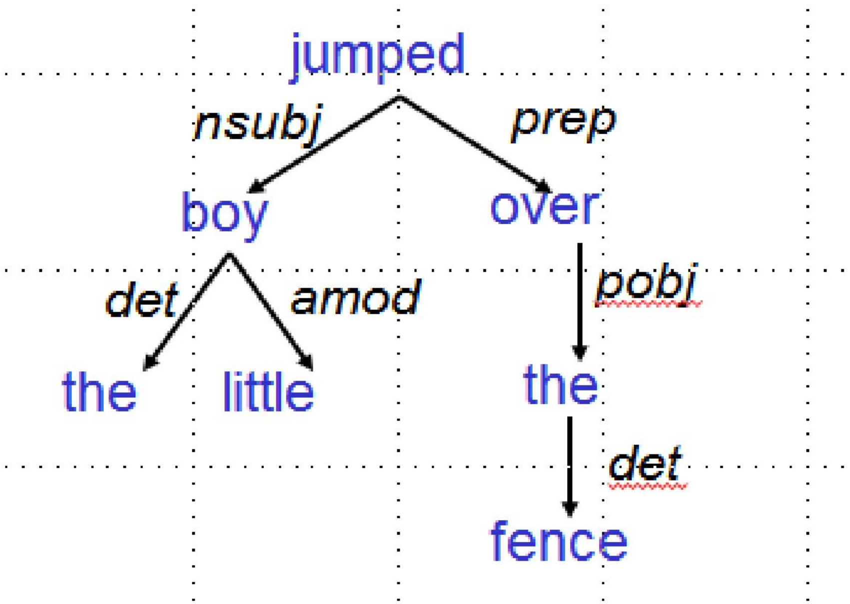
Relation Extraction

with Stanford Dependencies

3.4 Stanford Dependencies

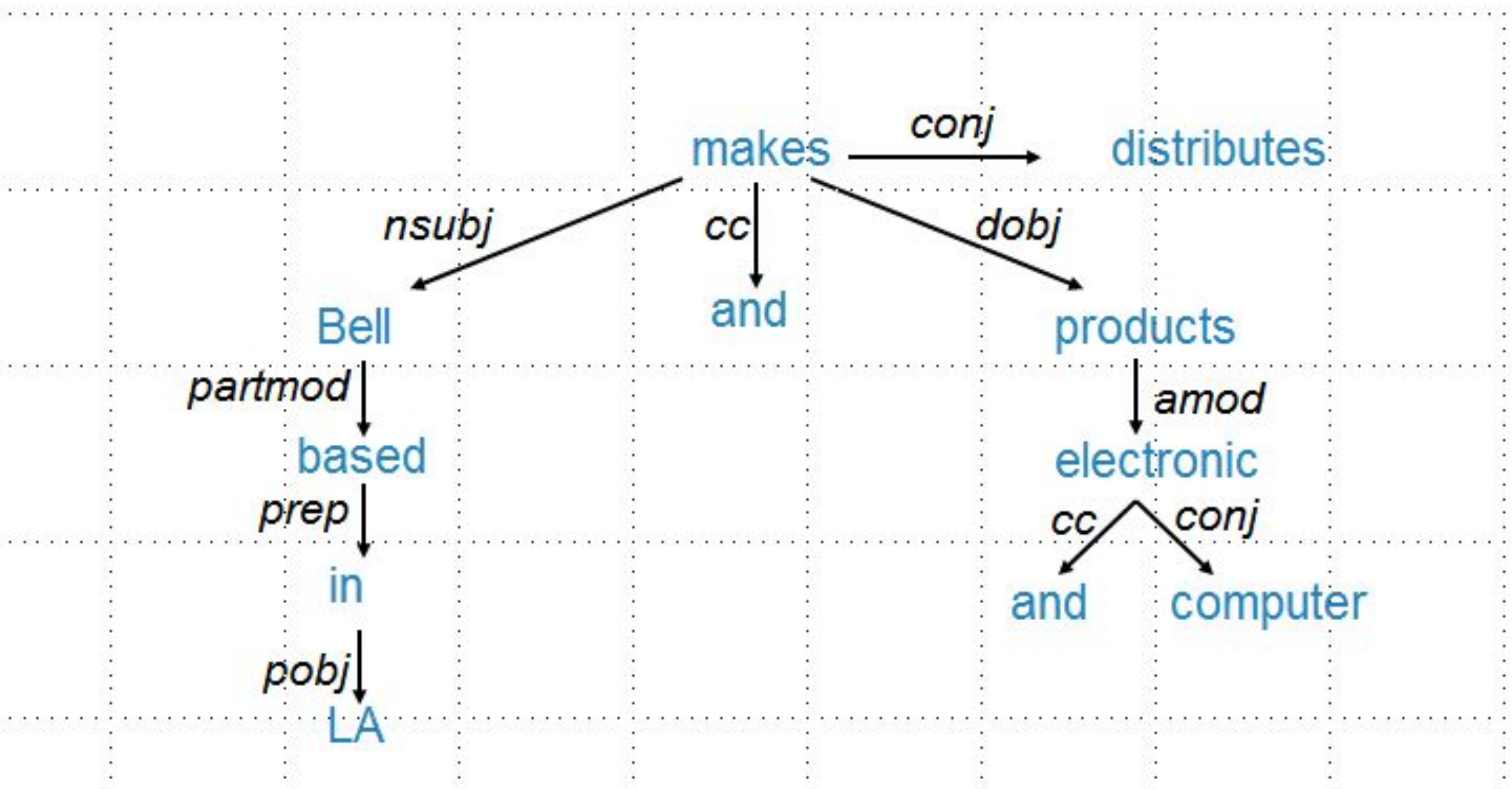
[de Marneffe et al. LREC 2006]

- The basic dependency representation is projective
- It can be generated by postprocessing headed phrase structure parses (Penn Treebank syntax)
- It can also be generated directly by dependency parsers, such as MaltParser, or the Easy-First Parser



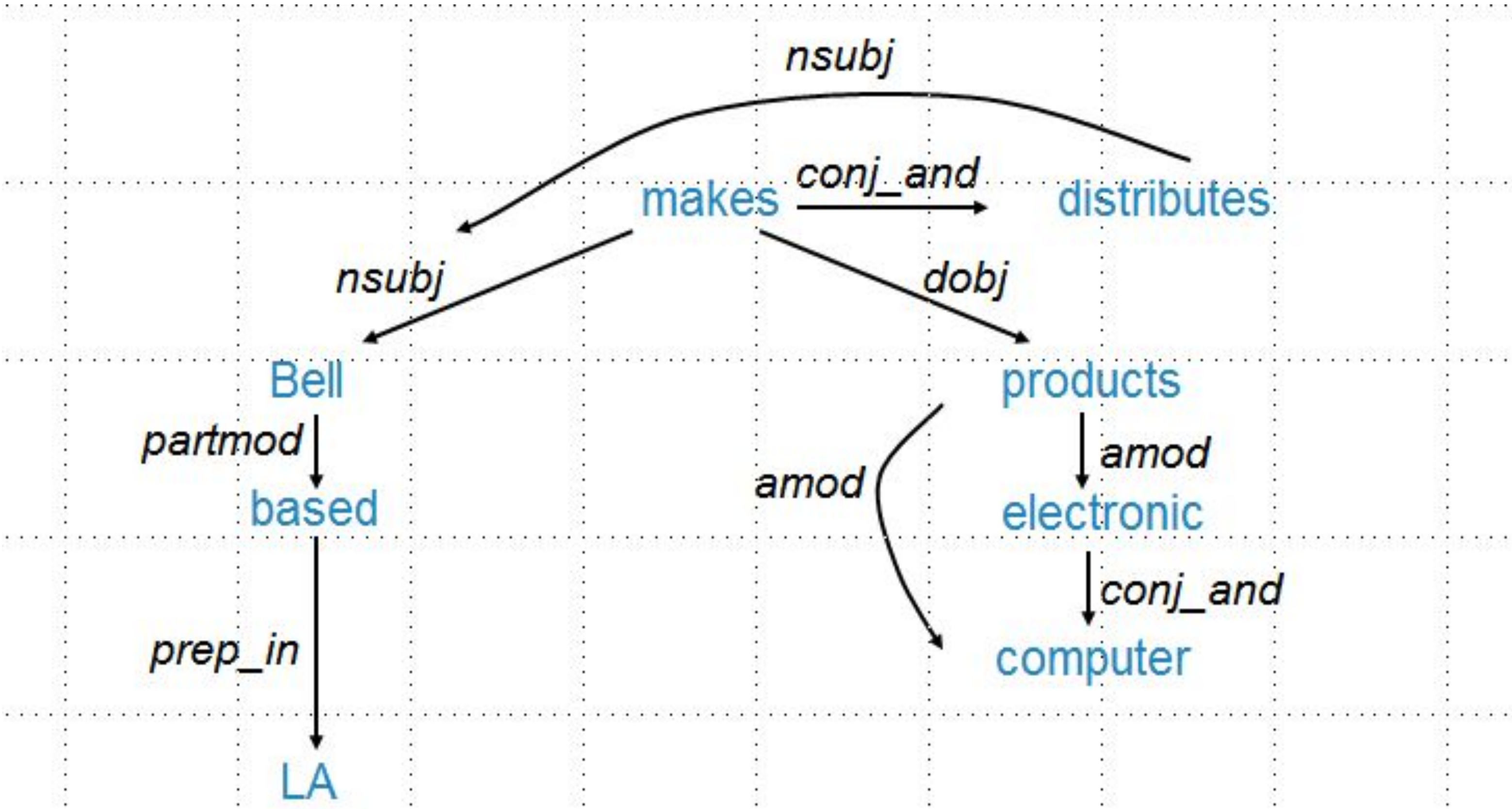
3.4.1 Graph modification to facilitate semantic analysis

Bell, based in LA, makes and distributes electronic and computer products.



3.4.1 Graph modification to facilitate semantic analysis

Bell, based in LA, makes and distributes electronic and computer products.



EXERCISES FOR CHAPTER 3

1/ Given the CFG:

$S \rightarrow NP\ VP$

$NP \rightarrow ART\ N$

$NP \rightarrow ART\ ADJ\ N$

$VP \rightarrow V$

$VP \rightarrow V\ NP$

a) Define a lexicon for the sentence:

“*The man walked the old dog* “.

b) Construct a top-down chart parse and bottom-up chart parse of the above sentence.

EXERCISES FOR CHAPTER 3

2. Use the dependency parser (3.3.5 ***MaltParser***) to parse the following sentences:

   a) He books me the morning flight

   b) Chúng tôi đăng ký vé máy bay ra Hà Nội.

Note: the Vietnamese sentence b) has six words: chúng tôi, đăng ký, vé, máy bay, ra, Hà Nội

Then, draw dependency trees by sets of dependency relations (dependency arcs) A, which are as the outputs of appropriate parses

REFERENCE OF CHAPTER 3

1. http://www.sfs.uni-tuebingen.de/~dm/10/ss/dep/dg-slides-2x2.pdf
2. https://web.stanford.edu/~jurafsky/NLPCourseraSlides.html,
3. Speech and Language Processing. Daniel Jurafsky & James H. Martin. Copyright c 2018.