UNIT-II

Grammars and Parsing

Grammars and Parsing- Top- Down and Bottom-Up Parsers, Transition Network Grammars, Feature Systems and Augmented Grammars, Morphological Analysis and the Lexicon, Parsing with Features, Augmented Transition Networks, Bayees Rule, Shannon game, Entropy and Cross Entropy.

[1]Grammars and Parsing

- The most common way of representing how a sentence is broken into its major subparts. and how those subparts are broken up in turn, is as a tree.
- The tree representation for the sentence *John ate the cat* is shown in Figure.
- This illustration can be read as follows: The sentence (5) consists of an initial noun phrase (NP) and a verb phrase (VP).
- The initial noun phrase is made of the simple NAME *John.*
- The verb phrase is composed of a verb (V) *ate* and an NP, which consists of an article (ART) *the* and a common noun (N) *cat.* In list notation this same structure could be represented as

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

1.	S → NP VP	5.	NAME → John
2.	VP → V NP	6.	V → ate
3.	NP → NAME	7.	ART → the
4.	NP → ART N	8.	N → cat

Grammar 3.2 A simple grammar

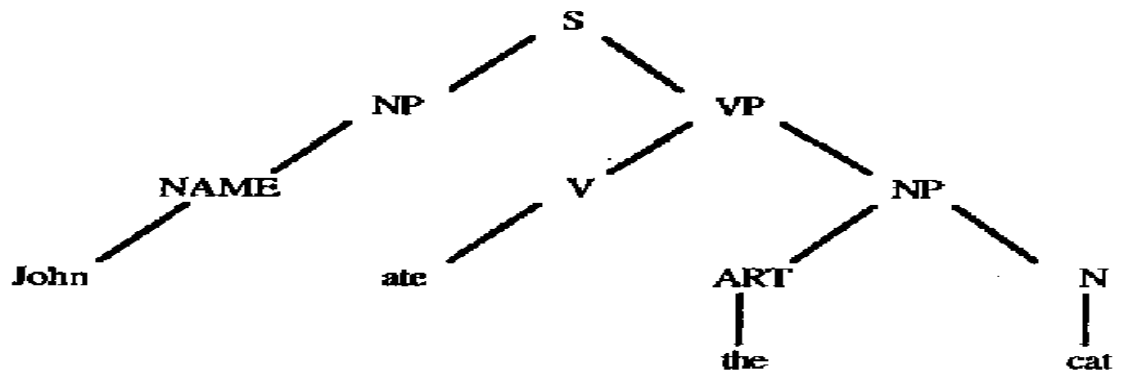


Figure 3.1 A tree representation of *John ate the cat*

- Trees are a special form of graph, which are structures consisting of labeled nodes (for example, the nodes are labeled 5, NP, and so on in Figure 3.1) connected by links.
- They are called trees because they resemble upside-down trees, and much of the terminology is derived from this analogy with actual trees.

• The node at the top is called the root of the tree, while the nodes at the bottom are called the leaves.
• We say a link points from a parent node to a child node.
• The node labeled S in Figure 3.1 is the parent node of the nodes labeled NP and VP, and the node labeled NP is in turn the parent node of the node labeled NAME.
• While every child node has a unique parent, a parent may point to many child nodes. An ancestor of a node N is defined as N's parent, or the parent of its parent, and so on.
• A node is dominated by its ancestor nodes. The root node dominates all other nodes in the tree.

TopDown Parsing

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

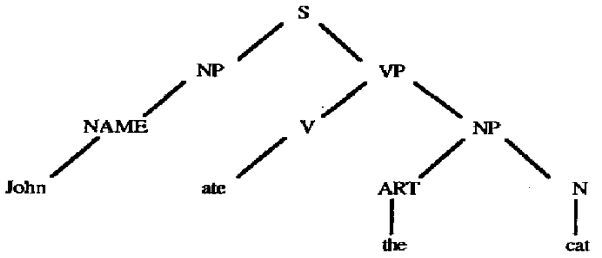


Figure 3.1 A tree representation of *John ate the cat*

Bottom up Parsing

=> john ate the cat (rewriting John)
=> NAME ate the cat (rewriting John)
=> NAME V the cat (rewriting ate)
=> NAME V ART cat (rewriting the)
=> NAME V ART N (rewriting cat)
=> NP V ART N (rewriting NAME)
=> NP V NP (rewriting ART N)
=> NP VP (rewriting V NP)
=> S (rewriting NP VP)

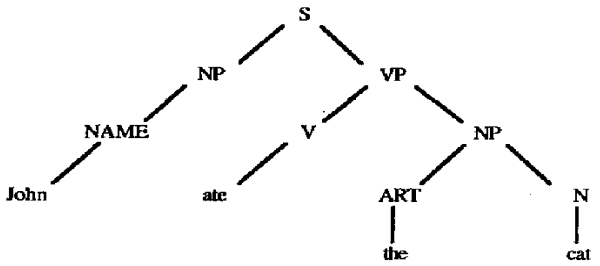


Figure 3.1 A tree representation of *John ate the cat*

A Top-Down Parser

- A top-down parser starts with the S symbol and attempts to rewrite it into a sequence of terminal symbols that matches the classes of the words in the input sentence.
- The state of the parse at any given time can be represented as a list of symbols that are the result of operations applied so far, called the symbol list.
- For example, the parser starts in the state and after applying the rule S -> NP VP the symbol list will be (NP VP).
- If it then applies the rule NP ->ART N, the symbol list will be (ART N VP), and so on.

Sample Grammar

1. **S -> NP VP**
2. **NP -> ART N**
3. **NP -> ART ADJ N**
4. **VP -> V**
5. **VP -> V NP**

Input:

1 The **2** dogs **3** cried **4**

cried: V
dogs: N, V
the: ART

A Simple Top-Down Parsing Algorithm

- The algorithm manipulates a list of possible states, called the possibilities list. The first element of this list is the current state, which consists of a symbol list
- -and a word position in the sentence, and the remaining elements of the search state are the backup states, each indicating an alternate symbol-list—word-position pair. For example, the possibilities list
- (((N) 2) ((NAME) 1) ((ADJ N) 1))
- indicates that the current state consists of the symbol list (N) at position 2, and that there are two possible backup states: one consisting of the symbol list (NAME) at position 1 and the other consisting of the symbol list (ADJ N) at position 1.

Top-down depth-first parse of 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 I
3	**((ART N VP) 1)**		rewriting NP by rules 2 & 3
		((ART ADJN VP) I)	
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 5— 8
		((V NP) 3)	
		((ART ADJ N VP) 1)	
7			the parse succeeds as V is
			matched to *cried,* leaving
			an empty grammatical symbol
			list with an empty sentence

Algorithm

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

1. Select the current state: Take the first state off the possibilities list and call it C. If the possibilities list is empty, then the algorithm fails (that is, no successful parse is possible).
2. If C consists of an empty symbol list and the word position is at the end of the sentence, then the algorithm succeeds.
3. Otherwise, generate the next possible states.

3.1. If the first symbol on the symbol list of C is a 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 and updating the word position, and add it to the possibilities list.

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

Example:

1 The 2 old 3 man 4 cried 5

the: ART
old: ADJ, N
man: N, V
cried: V

Step	Current State	Backup States	Comment
1.	((S) 1)		
2.	((NP VP) 1)		S rewritten to NP VP
3.	((ART N VP) 1)	((ART ADJ N VP) 1)	NP rewritten producing two new states
4.	((N VP) 2)	((ART ADJ N VP) 1)	
5.	((VP) 3)	((ART ADJ N VP) 1)	the backup state remains
6.	((V) 3)	((V NP) 3) ((ART ADJ N VP) 1)	
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!

Figure 3.6 A top-down parse of $_1$ *The* $_2$ *old* $_3$ *man* $_4$ *cried* $_5$

A search tree for two parse strategies (depth first strategy on Left, breadth first on right)

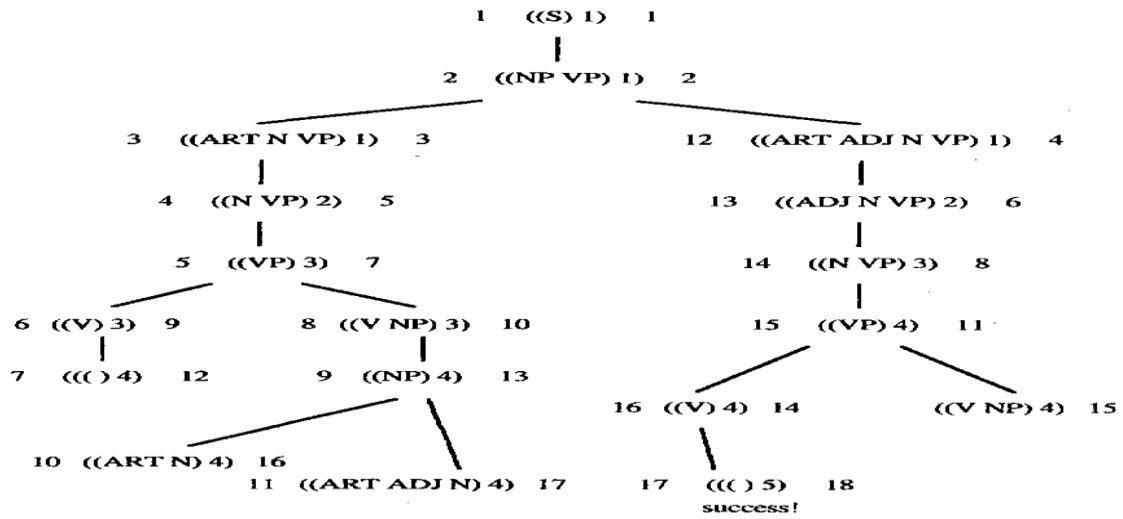


Figure 3.7 Search tree for two parse strategies (depth-first strategy on left; breadth-first on right)

A Bottom-Up Chart Parser

• The main difference between top-down and bottom-up parsers is the way the grammar rules are used. For example, consider the rule

• **NP -> ART ADJ N**

• In a top-down system you use the rule to find an NP by looking for the sequence ART ADJ N. In a bottom-up parser you use the rule to take a sequence ART ADJ N that you have found and identify it as an NP. The basic operation in bottom-up parsing then is to take a sequence of symbols and match it to the right-hand side of the rules. You could build a bottom-up parser simply by formulating this matching process as a search process. The state would simply consist of a symbol list, starting with the words in the sentence. Successor states could be generated by exploring all possible ways to

• **rewrite a word by its possible lexical categories**

• **replace a sequence of symbols that matches the right-hand side of a grammar rule by its left-hand side symbol**

1. **S -> N PVP**
2. **NP -> ART ADJ N**
3. **NP -> ART N**
4. **NP -> ADJ N**
5. **VP -> AUX VP**
6. **VP -> V NP**

Grammar: A simple context-free grammar

• Matches are always considered from the point of view of one constituent, called the key. To find rules that match a string involving the key, look for rules that start with the key, or for rules that have already been started by earlier keys and require the present key either to complete the rule or to extend the rule. For instance, consider Grammar 3.8.

• Assume you are parsing a sentence that starts with an ART. With this ART as the key, rules 2 and 3 are matched because they start with ART. To record this for analyzing the next key, you need to record that rules 2 and 3 could be continued at the point after the ART. You denote this fact by writing the rule with a dot (o), indicating what has been seen so far. Thus you record

• **2'. NP -> ART o ADJ N**

• **3'. NP -> ART o N**

• If the next input key is an ADJ, then rule 4 may be started, and the modi fied rule 2 may be extended to give

• **2''. NP -> ART ADJ o N**

• The chart maintains the record of all the constituents derived from the sentence so far in the parse. It also maintains the record of rules that have matched partially but are not complete. These are called the active arcs.

• For example, after seeing an initial ART followed by an ADJ in the preceding example, you would have the chart shown in Figure 3.9. You should interpret this figure as follows.

• There are two completed constituents on the chart: ART1 from position 1 to 2 and ADJ1 from position 2 to 3. There are four active arcs indicating possible constituents.

• These are indicated by the arrows and are interpreted as follows (from top to bottom).

• There is a potential NP starting at position 1, which needs an ADJ starting at position 2. There is another potential NP starting at position 1, which needs an N starting at position 2.

• There is a potential NP starting at position 2 with an ADS, which needs an N starting at position 3. Finally, there is a potential NP starting at position 1 with an ART and then an ADJ, which needs an N starting at position 3.

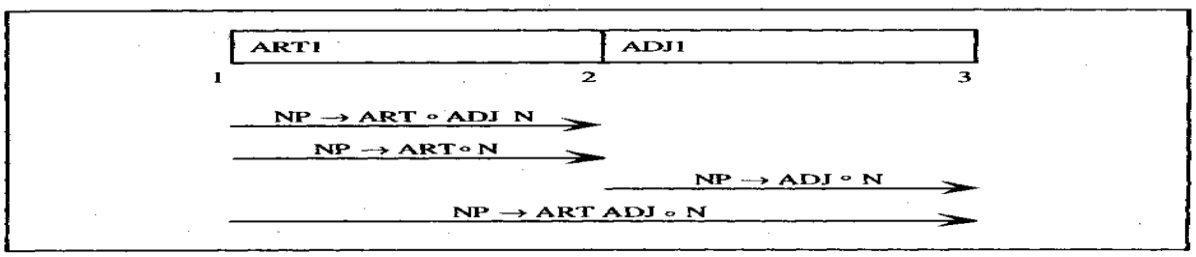


Figure 3.9 The chart after seeing an ADJ in position 2

The arc extension algorithm

To add a constituent C from position p_1 to p_2:

1. Insert C into the chart from position p_1 to p_2.
2. For any active arc of the form $X \rightarrow X_1 \ldots \circ C \ldots X_n$ from position p_0 to p_1, add a new active arc $X \rightarrow X_1 \ldots C \circ \ldots X_n$ from position p_0 to p_2.
3. For any active arc of the form $X \rightarrow X_1 \ldots X_n \circ C$ from position p_0 to p_1, then add a new constituent of type X from p_0 to p_2 to the agenda.

Figure 3.10 The arc extension algorithm

A bottom-up chart parsing algorithm

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 p1 to p2).

3. For each rule in the grammar of form X -> C X1 ... Xn, add an active arc of form X -> C o C o X1 .... Xn from position p1 to p2.

4. Add C to the chart using the arc extension algorithm above.

Example

• Consider using the algorithm on the sentence *The large can can hold the water* using Grammar 3.8 with the following lexicon:

the: ART
large: ADS
can: N,AUX,V
hold: N,V
water: N, V

• To best understand the example, draw the chart as it is extended at each step of the algorithm. The agenda is initially empty, so the word *the* is read and a constituent ARTL placed on the agenda.

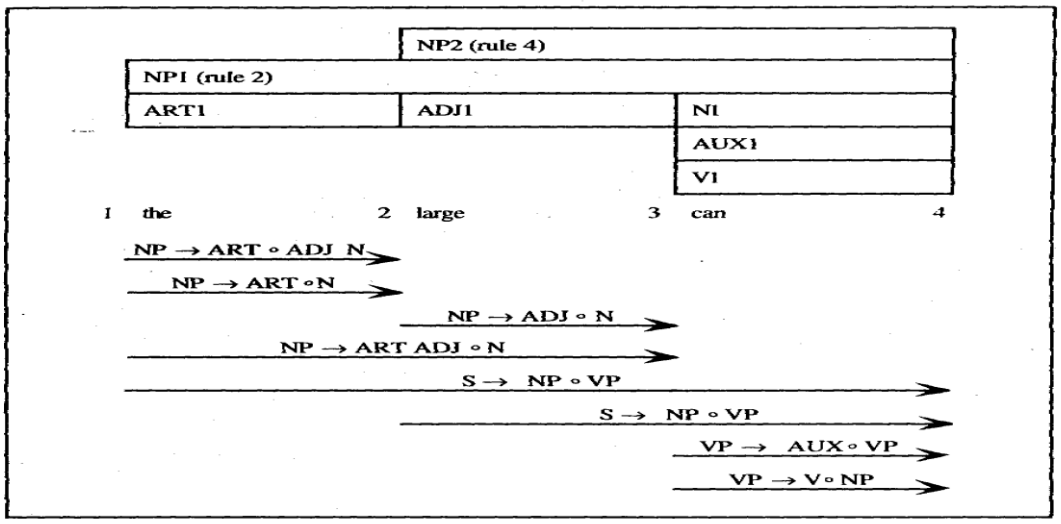


Figure 3.12 After parsing *the large can*

- **Entering ART1: (the from 1 to 2)**
- **Adds active arc NP -> ART o ADJ N from 1 to 2**
- **Adds active arc NP -> ART o N from 1 to 2**
- **Entering ADJ1: (large from 2 to 3)**
- **Adds arc NP -> ADS o N from 2 to 3**
- **Adds arc NP -> ART ADJ o N from 1 to 3**
- For the next word, *can,* three constituents, Ni, AUX1, and V1 are created for its three interpretations.
- **Entering NP1 :an NP (*the large can* from 1 to 4)**
- **Adding active arc S -> NP o VP from 1 to 4**
- **Entering NP2: an NP (*large can* from 2 to 4)**
- **Adding arc S -> NP o VP from 2 to 4**
- **Entering AUX1: (*can* from 3 to 4)**
- **Adding arc VP -> AUX o VP from 3 to 4**
- **Entering V1: (*can* from 3 to 4)**
- **Adding arc VP —> V o NP from 3 to 4**

• The chart is shown in Figure 3.12, which illustrates all the completed constituents (NP2, NP1, ART1, ADJ1, N1, AUX1, V1) and all the uncompleted active arcs entered so far. The next word is *can* again. and N2, AUX, and V2 are created.

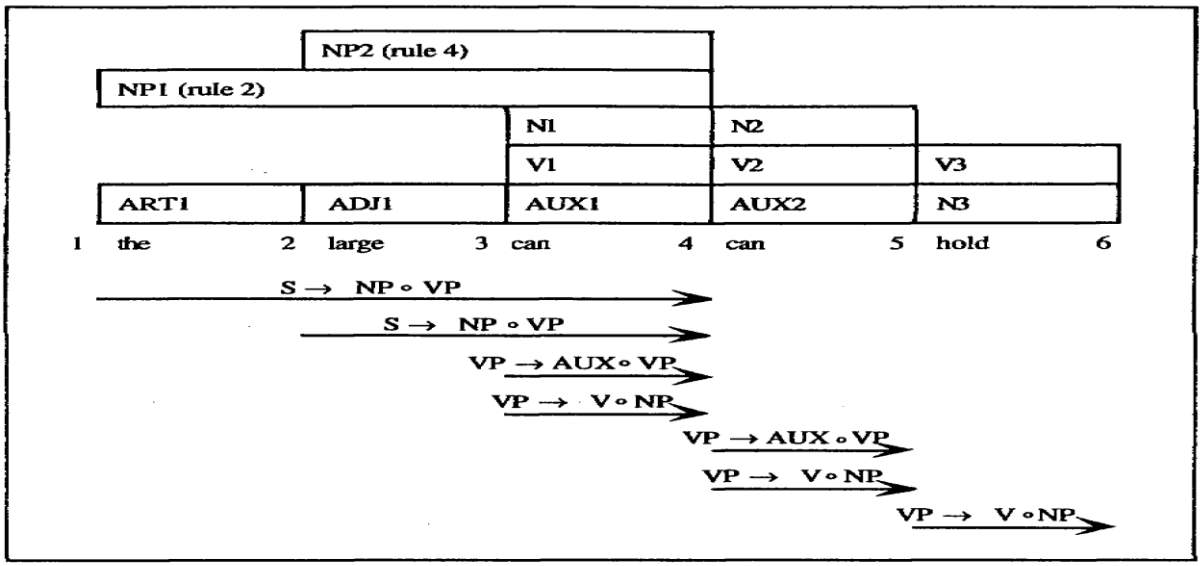


Figure 3.13 The chart after adding *hold*, omitting arcs generated for the first NP

- **Entering N2: (can from 4 to 5, the second can)**
- **Adds no active arcs**
- **Entering AUX2: (can from 4 to 5)**
- **Adds arc VP -> AUX o VP from 4 to 5**
- **Entering V2: (can from 4 to 5)**
- **Adds arc VP -> V o NP from 4 to 5**
- The next word is *hold*, and N3 and V3 are created.
- **Entering N3: (hold from 5 to 6)**
- **Adds no active arcs**
- **Entering V3: (hold from 5 to 6)**
- **Adds arc VP -> V o NP from 5 to 6**

The chart in Figure 3.13 shows all the completed constituents built so far, together with all the active arcs, except for those used in the first NP.

- **Entering ART2: (*the* from 6 to 7)**
- **Adding arc NP -> ART o ADJ N from 6 to 7**
- **Adding arc NP -> ART o N from 6 to 7**

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

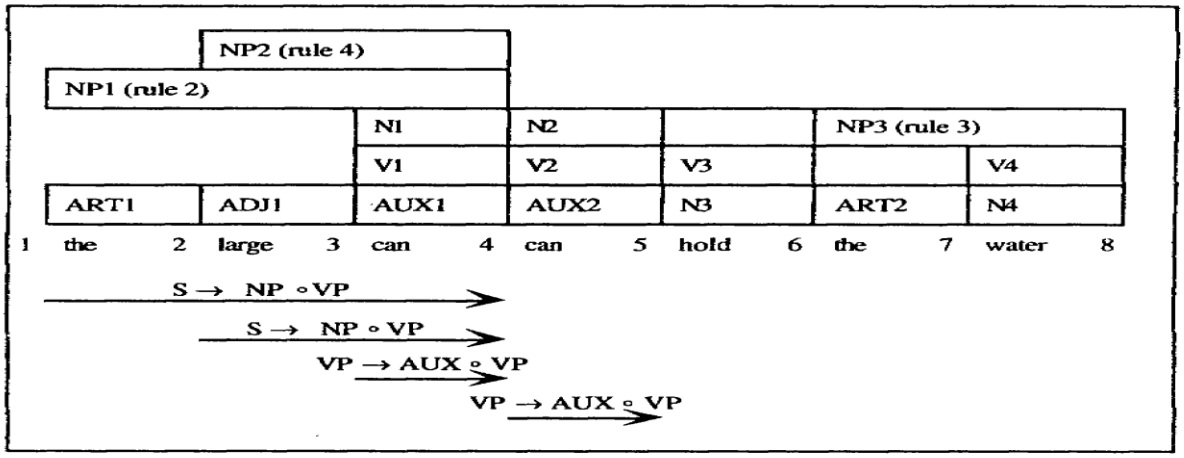


Figure 3.14 The chart after all the NPs are found, omitting all but the crucial active arcs

• Entering N4: (*water* from 7 to 8)
• No active arcs added in step 3
• An NP, NP3, from 6 to 8 is pushed onto the agenda, by completing arc
• NP -> ART o N from 6 to 7
• Entering NP3: (*the water* from 6 to 8)
• A VP, VP1, from 5 to 8 is pushed onto the agenda, by completing VP ->
• V o NP from 5 to 6
• Adds arc S -> NP o VP from 6 to 8

The chart at this stage is shown in Figure 3.14, but only the active arcs to be used in the remainder of the parse are shown.
• Entering VP1: (*hold the water* from 5 to 8)
• A VP, VP2, from 4 to 8 is pushed onto the agenda, by completing
• VP -> AUX o VP from 4 to S
• Entering VP2: (*can hold the water* from 4 to 8)
• An S, S1, is added from 1 to 8, by completing
• arcS -> NP o VP from 1 to 4
• A VP, VP3, is added from 3 to 8, by completing
• arc VP -> AUX o VP from 3 to 4
• An S, S2, is added from 2 to 8, by completing
• arc S -> NP o VP from 2 to 4

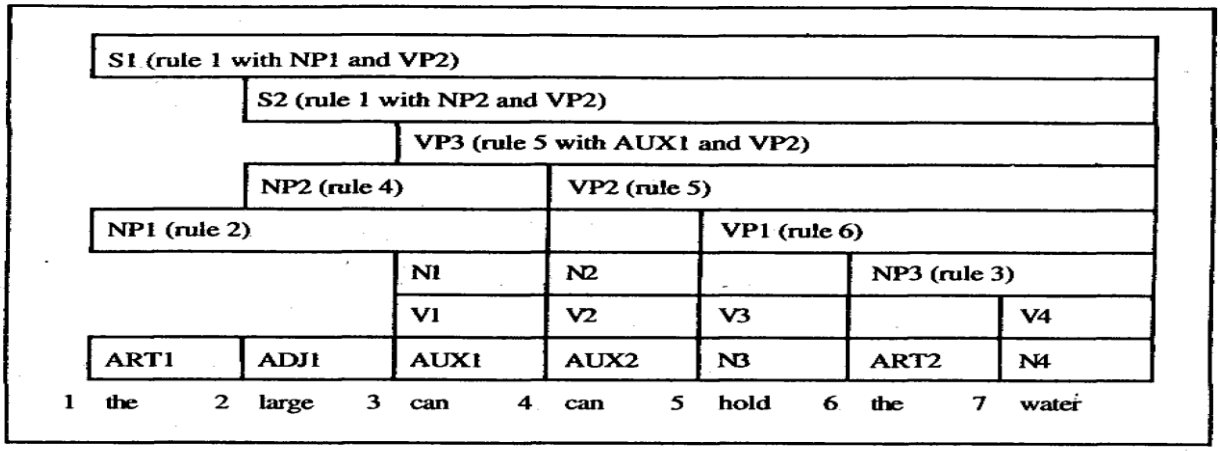
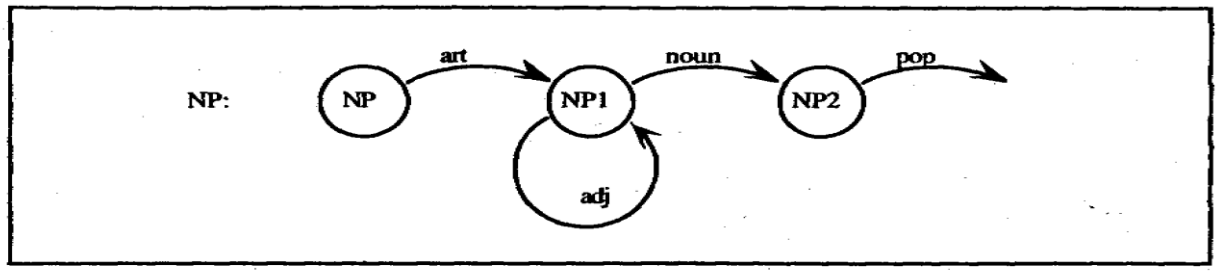


Figure 3.15 The final chart

[2]Transition Network Grammars:

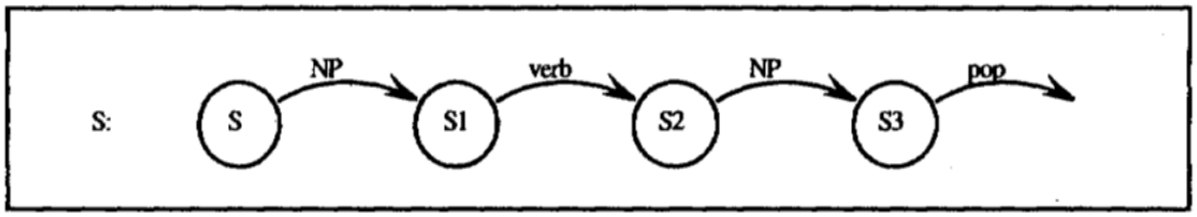
• It is based on the notion of a transition network consisting of nodes and labeled arcs.
• One of the nodes is specified as the initial state, or start state.
• Consider the network named NP in Grammar 3.16, with the initial state labeled NP and each arc labeled with a word category.
• Starting at the initial state, you can traverse an arc if the current word in the sentence is in the category on the arc.
• If the arc is followed, the current word is updated to the next word.



Grammar 3.16

NP -> ART NP1
NP1 -> ADJ NP1
NP1 -> N

• Consider parsing the NP ***a purple cow*** with this network.
• Starting at the node NP, you can follow the arc labeled art, since the current word is an **article**—namely, ***a***. From node NP1 you can follow the arc labeled **adj** using the **adjective *purple***, and finally, again from NP1, you can follow the arc labeled **noun** using the **noun *cow***.
• Since you have reached a pop arc, ***a purple cow*** is a leg al NP.
• Simple transition networks are often called **finite state machines (FSMs).**
• Finite state machines are equivalent in expressive power to regular grammars, and thus are not powerful enough to describe all languages that can be described by a CFG.
• To get the descriptive power of CFGs, you need a notion of recursion in the network grammar.
• A **recursive transition network (RTN)** is like a simple transition network, except that it allows arc labels to refer to other networks as well as word categories.
• Thus, given the NP network in Grammar 3.16, a network for simple English sentences can be expressed as shown in Grammar 3.17.
• Uppercase labels refer to networks.
• The arc from S to Si can be followed only if the NP network can be successfully traversed to a pop arc.



Grammar 3.17

• Consider finding a path through the S network for the sentence ***The purple cow ate the grass***.
• Starting at node S, to follow the arc labeled NP, you need to traverse the NP network.
• Starting at node NP, traverse the network as before for the input ***the purple cow.***
• Following the pop arc in the NP network, return to the S network and traverse the arc to node S1.
• From node S1 you follow the arc labeled verb using the word ***ate.***
• Finally, the arc labeled NP can be followed if you can traverse the NP network again.
• This time the remaining input consists of the words ***the grass.***
• You follow the arc labeled art and then the arc labeled noun in the NP network; then take the pop arc from node NP2 and then another pop from **node S3**.
• Since you have traversed the network and used all the words in the sentence, ***The purple cow ate the grass*** is accepted as a legal sentence.

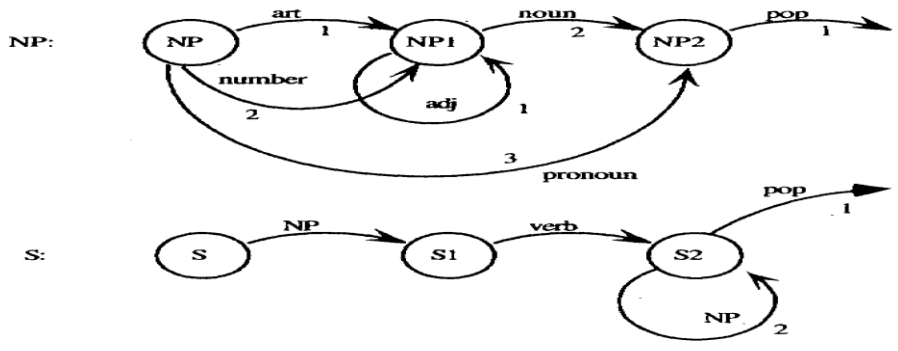
The arc labels for RTNs (Recursive Transition Network)

Arc Type	Example	Arc Type Example How Used
CAT	noun	succeeds only if current word is of the named category
WRD	of	succeeds only if current word is identical to the label
PUSH	NP	succeeds only if named network can be successfully traversed
JUMP	jump	always succeeds
POP	pop	succeeds and signals the successful end of the network

• Above Figure summarizes the arc types, together with the notation that will be used in this book to indicate these arc types.
• According to this terminology, arcs that are labeled with networks are called push arcs, and arcs labeled with word categories are called cat arcs.
• In addition, an arc that can always be followed is called a jump arc.

Top-Down Parsing with Recursive Transition Networks

• An algorithm for parsing with RTNs can be developed along the same lines as the algorithms for parsing CFGs.
• The state of the parse at any moment can be represented by the following:
• **current position** - a pointer to the next word to be parsed.
• **current node** - the node at which you are located in the network.
• **return points** - a stack of nodes in other networks where you will continue if you pop from the current network.



Grammar 3.19

- You can leave the current node and traverse an arc in the following cases:
- **Case 1:**
  - If arc names **word category** and next word in sentence is in that category,
  - Then
  - (1) update **current position** to start at the next word;
  - (2) update **current node** to the destination of the arc.
- **Case 2:**
  - If arc is a **push** arc to a network N,

Then

- (1) add the destination of the arc onto return points;
- (2) update current node to the starting node in network N.

- **Case 3:**
  - If arc is a **pop** arc and return points list is not empty,

  Then
  - (1) remove first return point and make it current node.
- **Case 4:**
  - If arc is a **pop arc**, return points **list is empty** and there are **no words left,**

  Then

  (1) parse completes successfully.

Step	Current Node	Current Position	Return Points	Arc to be Followed	Comments
1.	(S,	1,	NIL)	S/1	initial position
2.	(NP,	1,	(S1))	NP/1	followed push arc to NP network, to return ultimately to S1
3.	(NP1,	2,	(S1))	NP1/1	followed arc NP/1 (*the*)
4.	(NP1,	3,	(S1))	NP1/2	followed arc NP1/1 (*old*)
5.	(NP2,	4,	(S1))	NP2/2	followed arc NP1/2 (*man*) since NP1/1 is not applicable
6.	(S1,	4,	NIL)	S1/1	the pop arc gets us back to S1
7.	(S2,	5,	NIL)	S2/1	followed arc S2/1 (*cried*)
8.					parse succeeds on pop arc from S2

Figure 3.20 A trace of a top-down parse

- Figure 3.20 demonstrates that the grammar accepts the sentence
- **1 The 2 old 3 man 4 cried 5**
- by showing the sequence of parse states that can be generated by the algorithm. In the trace, each arc is identified by the name of the node that it leaves plus the number identifier.
- Thus arc S/1 is the arc labeled 1 leaving the S node. If you start at node 5, the only possible arc to follow is the push arc NP.
- As specified in case 2 of the algorithm, the new parse state is computed by setting the current node to NP and putting node S1 on the return points list.
- From node NP, arc NP/1 is followed and, as specified in case 1 of the algorithm, the input is checked for a word in category art.
- Since this check succeeds, the arc is followed and the current position is updated (step 3).
- The parse continues in this manner to step 5, when a pop arc is followed, causing the current node to be reset to S1 (that is, the NP arc succeeded).
- The parse succeeds after finding a verb in step 6 and following the pop arc from the S network in step 7.

[3]Feature Systems and Augmented Grammars:

• extension to the basic context-free mechanism that defines constituents by a set of features.

• This extension allows aspects of natural language such as agreement and subcategorization to be handled in an intuitive and concise way.

• In natural languages there are often agreement restrictions between words and phrases.

• For example, the NP "***a men***" is not correct English because the article **a indicates a single object** while the **noun "*men*" indicates a plural object**; the noun phrase does not satisfy the number agreement restriction of English.

• There are many other forms of agreement, including **subject-verb agreement**, **gender agreement for pronouns**, **restrictions** between the **head of a phrase and the form of its complement**, and so on.

• To handle such phenomena conveniently, the grammatical formalism is extended to allow constituents to have features.

• For example, we might define a feature **NUMBER** that may take a value of either ***s*** (for sin gular) or **p** (for plural), and we then might write an augmented CFG rule such as

NP -> ART N only when NUMBER1 agrees with NUMBER2

• This rule says that a legal noun phrase consists of an article followed by a noun, but only when the number feature of the first word agrees with the number feature of the second.

• This one rule is equivalent to two CFG rules that would use different terminal symbols for encoding singular and plural forms of all noun phrases, such as

NP-SING -> ART-SING N-SING

NP-PLURAL -> ART-PLURAL N-PLURAL

• While the two approaches seem similar in ease-of-use in this one example, consider that all rules in the grammar that use an NP on the right-hand side

Feature structures

• Constituents can be defined as feature structures that map features to values

• Features can be shared between constituents

• Some basic features for English:

– **Number, Gender and Person agreement**

– Verb form features and sub-categorizations

• **Complex Feature Structures**: Feature values can themselves be feature structures

• **Feature names** in formulas will be written in **boldface**.

• For example, a feature structure for a constituent **ART1** that represents a particular use of the word **a** might be written as follows:

ART1:

(CAT ART
ROOT a
NUMBER s**)**

• This says it is a constituent in the category ART that has as its root the word a and is singular.

• In this abbreviated form, constituent ART1 would be written as

ART1: (ART **ROOT** a **NUMBER** s**)**

• Feature structures can be used to represent larger constituents as well. To do this, feature structures themselves can occur as values.

• Special features based on the integers - 1, 2, 3, and so on - will stand for the first subconstituent, second subconstituent, and so on, as needed.

• With this, the representation of the NP constituent for the phrase "*a fish*" could be

Larger Constituent

- The representation of the NP constituent for the phrase "***a fish***" could be
- NP1:(NP NUMBERs
  1 (ART ROOT a
  NUMBER s)
  2 (N ROOT fish
  NUMBER s))

Feature Structure

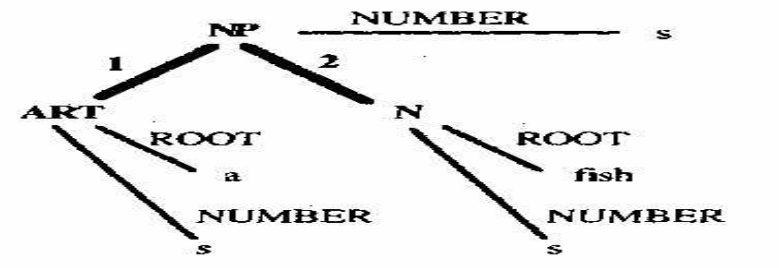


Figure 4.1 Viewing a feature structure as an extended parse tree

Augmented Rules

• The rules in an augmented grammar are stated in terms of feature structures rather than simple categories.

• Variables are allowed as feature values so that a rule can apply to a wide range of situations. For example, a rule for simple noun phrases would be as follows:

• (NP NUMBER ?n) - (ART NUMBER ?n) (N NUMBER ?n)

• This says that an NP constituent can consist of two subconstituents, the first being an ART and the second being an N, in which the NUMBER feature in all three constituents is identical.

(NP **1** (ART **NUMBER** s)
2 (N **NUMBER** s))

• is not allowed by this rule because there is no NUMBER feature in the NP, and the constituent

*(NP **NUMBER** s
1 (ART **NUMBER** s)
2 (N **NUMBER** p))

• is not allowed because the NUMBER feature of the N constituent is not identical to the other two NUMBER features.

Using Variable to Express Ambiguities

• The word fish is ambiguous between a singular and a plural reading.

• Thus the word might have two entries in the lexicon that differ only by the value of the NUMBER feature.

• Alternatively, we could define a single entry that uses a variable as the value of the NUMBER feature, that is,

(N ROOT fish NUMBER ?n)

• This works because any value of the NUMBER feature is allowed for the word fish. In many cases, however, not just any value would work, but a range of values is possible

• To handle these cases, we introduce constrained variables, which are variables that can only take a value out of a specified list. For example, the variable ?n{s p} would be a variable that can take the value s or the value p.

• when we write such variables, we will drop the variable name altogether and just list the possible values. Given this, the word fish might be represented by the constituent

(N ROOT fish NUMBER ?n{sp})

or more simply as

(N ROOT fish NUMBER {s p})

Formalizing Feature Structures

BOX 4.1 Formalizing Feature Structures

There is an active area of research in the formal properties of feature structures. This work views a feature system as a formal logic. A feature structure is defined as a partial function from features to feature values. For example, the feature structure

ART1: (CAT ART
ROOT a
NUMBER s)

is treated as an abbreviation of the following statement in FOPC:

$$ART1(CAT) = ART \wedge ART1(ROOT) = a \wedge ART1(NUMBER) = s$$

Feature structures with disjunctive values map to disjunctions. The structure

THE1: (CAT ART
ROOT the
NUMBER {s p})

would be represented as

$$THE1(CAT) = ART \wedge THE1(ROOT) = the \wedge (THE1(NUMBER) = s \vee THE1(NUMBER) = p)$$

Given this, agreement between feature values can be defined as equality equations.

Some Basic Feature Systems for English

- **Person and Number Features**
- The possible values of this dimension are
- **First Person (1):** The noun phrase refers to the speaker, or a group of people including the speaker (for example, I, we, you,...
- **Second Person (2):** The noun phrase refers to the listener, or a group including the listener but not including the speaker (for example, you, all of you).
- **Third Person (3):** The noun phrase refers to one or more objects, not including the speaker or hearer.
- it is convenient to combine the two into a single feature, that has six possible values:
- first person singular (1s),
- second person singular (2s),
- third person singular (3s),
- first, second and third person plural (ip, 2p, and 3p, respectively).

Verb-Form Features and Verb Subcategorization

- we will use the following feature values for the feature VFORM:

base - base form (for example, go, be, say, decide)

pres - simple present tense (for example, go, goes, am, is, say, says, decide)

past - simple past tense (for example, went, was, said, decided)

ing - present participle (for example, going, being, saying, deciding)

pastprt - past participle (for example, gone, been, said, decided)

[4]Morphological Analysis and the Lexicon:

• **What is Morphology?**

• The study of word formation – how words are built up from smaller pieces.

• Identification, analysis, and description of the structure of a given language's MORPHEMES and other linguistic units, such as root words, affixes, parts of speech, intonations and stresses, or implied context.

• **Morphology**: is the study of word formation, of the structure of words.

• Examples:

- Care-*less*
- *Un*-happy
- Teach-*er*
- *Wash-ing*
- *Brows-er*
- *Rat-s*

Morpheme

• **Morpheme** is a minimal unit of meaning or grammatical function.

• It is a meaningful linguistic unit consisting of a **word**, such as **man**, or a word **element**, such as **-ed** in **walked**, that cannot be divided into smaller meaningful parts.

• For example, the English word **unacceptable** can be segmented into **three morphemes**, **un, accept, able**, each of which carries a certain semantic meaning and cannot be further segmented.

• It is a smallest morphological unit that cannot be divided into smaller parts.

• A morpheme may be a **complete word** (e.g. boy, scout, accept) or an **affix** (e.g. -s, un-, -able, -hood).

• A word of **one morpheme is called one-morpheme word** and **a word of two two-morpheme word**.

• The word **boy** contains **one morpheme** and the word **boys** contains **two morphemes.**

Types of Morpheme

• **Free Morpheme**: Free morphemes are those that can stand alone as words. They may be lexical morphemes ({serve}, {press}), or grammatical(functional) morphemes ({at}, {and}).

• in English, free morphemes can be identified as the set of separate word forms such as basic nouns, adjectives, verbs, etc.

• e.g. care, teach, help, above….

• **Bound Morpheme**: the morphemes that occur only in combination are called bound morpheme. (e.g. –ed, -s, -ing)

• E.g. un-, -er, -less, -ed, -ing……

• Unhappy, teacher, careless, talked, teaching ……

Morphological Analysis

INPUT	Morphologically analysed output
Cats	Cat+N+PL
Cat	Cat+N+SG
Cities	City+N+PL
Goose	Goose+N+SG or Goose+V

Morphological Analysis and the Lexicon

• Before you can specify a grammar, you must define the **lexicon**

• The lexicon must contain information about all the different words that can be used, including all the relevant feature value restrictions.

• When a word is ambiguous, it may be described by multiple entries in the lexicon, one for each different use.

• Because words tend to follow regular morphological patterns, however, many forms of words need not be explicitly included in the lexicon.

• Most English verbs, for example, use the same set of suffixes to indicate different forms: -s is added for third person singular present tense, -ed for past tense, -ing for the present participle, and so on.

• Without any morphological analysis, the lexicon would have to contain every one of these forms. For the verb want this would require six entries, for want (both in base and present form), wants, wanting, and wanted (both in past and past participle form).

• Consider the following rule for present tense verbs:

(V **ROOT** ?r **SUBCAT** ?s **VFORM** pres **AGR** 3s) -> (V **ROOT** ?r **SUBCAT** ?s **VFORM** base) (+S)

• where +S is a new lexical category that contains only the suffix morpheme -s. This rule, coupled with the lexicon entry

• want:

(V **ROOT** want
SUBCAT {_np_vp:inf _np_vp:inf}
VFORM base)

• would produce the following constituent given the input string want –s

• want:

(V **ROOT** want
SUBCAT {_np_vp:inf _np_vp:inf}
VFORM pres
AGR 3s)

• Another rule would generate the constituents for the present tense form not in third person singular, which for most verbs is identical to the root form:

(V **ROOT** ?r **SUBCAT** ?s **VFORM** pres **AGR** {ls 2s lp 2p 3p})
—> (V **ROOT** ?r **SUBCAT** ?s **VFORM** base)

• But this rule needs to be modified in order to avoid generating erroneous interpretations.

• Currently, it can transform any base form verb into a present tense form, which is clearly wrong for some irregular verbs. For instance, the base form be cannot be used as a present form (for example, *We be at the store).

• To cover these cases, a feature is introduced to identify irregular forms. Specifically, verbs with the binary feature +IRREG PRES have irregular present tense forms. Now the rule above can be stated correctly:

(V **ROOT** ?r **SUBCAT** ?s **VFORM** pres **AGR** (ls 2s lp 2p 3p)) —
> (V **ROOT** ?r **SUBCAT** ?s **VFORM** base **IRREG-PRES** -)

PROBLEMS IN MORPHOLOGICAL ANALYSIS

- **False Analysis**
- False analysis Words such as **hospitable, sizeable.**
- They don't have the meaning "**to be able**"
- Analyzing them as the words containing suffix -able leads to false analysis

Some lexical rules for common suffixes on verbs and nouns

```
Present Tense
1.  (V ROOT ?r SUBCAT ?s VFORM pres AGR 3s) →
        (V ROOT ?r SUBCAT ?s VFORM base IRREG-PRES –) +S
2.  (V ROOT ?r SUBCAT ?s VFORM pres AGR {1s 2s 1p 2p 3p}) →
        (V ROOT ?r SUBCAT ?s VFORM base IRREG-PRES –)
Past Tense
3.  (V ROOT ?r SUBCAT ?s VFORM past AGR {1s 2s 3s 1p 2p 3p}) →
        (V ROOT ?r SUBCAT ?s VFORM base IRREG-PAST –) +ED
Past Participle
4.  (V ROOT ?r SUBCAT ?s VFORM pastprt) →
        (V ROOT ?r SUBCAT ?s VFORM base EN-PASTPRT –) +ED
5.  (V ROOT ?r SUBCAT ?s VFORM pastprt) →
        (V ROOT ?r SUBCAT ?s VFORM base EN-PASTPRT +) +EN
Present Participle
6.  (V ROOT ?r SUBCAT ?s VFORM ing) →
        (V ROOT ?r SUBCAT ?s VFORM base) +ING
Plural Nouns
7.  (N ROOT ?r AGR 3p) →
        (N ROOT ?r AGR 3s IRREG-PL –) +S
```

Grammar 4.5 Some lexical rules for common suffixes on verbs and nouns

A lexicon

```
a:      (CAT ART
         ROOT A1
         AGR 3s)
be:     (CAT V
         ROOT BE1
         VFORM base
         IRREG-PRES +
         IRREG-PAST +
         SUBCAT {_adjp _np})
cry:    (CAT V
         ROOT CRY1
         VFORM base
         SUBCAT _none)
dog:    (CAT N
         ROOT DOG1
         AGR 3s)
fish:   (CAT N
         ROOT FISH1
         AGR {3s 3p}
         IRREG-PL +)
happy:  (CAT ADJ
         SUBCAT _vp:inf)
he:     (CAT PRO
         ROOT HE1
         AGR 3s)
is:     (CAT V
         ROOT BE1
         VFORM pres
         SUBCAT {_adjp _np}
         AGR 3s)
Jack:   (CAT NAME
         AGR 3s)
man:    (CAT N1
         ROOT MAN1
         AGR 3s)
men:    (CAT N
         ROOT MAN1
         AGR 3p)
saw:    (CAT N
         ROOT SAW1
         AGR 3s)
saw:    (CAT V
         ROOT SAW2
         VFORM base
         SUBCAT _np)
saw:    (CAT V
         ROOT SEE1
         VFORM past
         SUBCAT _np)
see:    (CAT V
         ROOT SEE1
         VFORM base
         SUBCAT _np
         IRREG-PAST +
         EN-PASTPRT +)
seed:   (CAT N
         ROOT SEED1
         AGR 3s)
the:    (CAT ART
         ROOT THE1
         AGR {3s 3p})
to:     (CAT TO)
want:   (CAT V
         ROOT WANT1
         VFORM base
         SUBCAT {_np _vp:inf _np_vp:inf})
was:    (CAT V
         ROOT BE1
         VFORM past
         AGR {1s 3s}
         SUBCAT {_adjp _np})
were:   (CAT V
         ROOT BE
         VFORM past
         AGR {2s 1p 2p 3p}
         SUBCAT {_adjp _np})
```

Figure 4.6 A lexicon

Grammar

1. S[–inv] → (NP **AGR** ?a) (*VP* [{*pres past*}] ***AGR*** *?a*)
2. NP → (ART **AGR** ?a) (*N* ***AGR*** *?a*)
3. NP → *PRO*
4. VP → *V*[*_none*]
5. VP → *V*[*_np*] NP
6. VP → *V*[*_vp:inf*] VP[inf]
7. VP → *V*[*_np_vp:inf*] NP VP[inf]
8. VP → *V*[*_adjp*] ADJP
9. VP[inf] → *TO* VP[base]
10. ADJP → *ADJ*
11. ADJP → *ADJ* [*_vp:inf*] VP[inf]

Head features for S, VP: **VFORM**, **AGR**
Head features for NP: **AGR**

Grammar 4.7 A simple grammar in abbreviated form

Expanded Grammar

1. (S **INV** – **VFORM** ?v{pres past} **AGR** ?a) →
   (NP **AGR** ?a) (*VP* ***VFORM*** *?v* {*pres past* } ***AGR*** *?a*)
2. (NP **AGR** ?a) → (ART **AGR** ?a) (*N* ***AGR*** *?a*)
3. (NP **AGR** ?a) → (*PRO* ***AGR*** *?a*)
4. (VP **AGR** ?a **VFORM** ?v) → (*V* ***SUBCAT*** *_none* ***AGR*** *?a* ***VFORM*** *?v*)
5. (VP **AGR** ?a **VFORM** ?v) → (*V* ***SUBCAT*** *_np* ***AGR*** *?a* ***VFORM*** *?v*) NP
6. (VP **AGR** ?a **VFORM** ?v) →
   (*V* ***SUBCAT*** *_vp:inf* ***AGR*** *?a* ***VFORM*** *?v*) (VP **VFORM** inf)
7. (VP **AGR** ?a **VFORM** ?v) →
   (*V* ***SUBCAT*** *_np_vp:inf* ***AGR*** *?a* ***VFORM*** *?v*) NP (VP **VFORM** inf)
8. (VP **AGR** ?a **VFORM** ?v) →
   (*V* ***SUBCAT*** *_adjp* ***AGR*** *?a* ***VFORM*** *?v*) ADJP
9. (VP **SUBCAT** inf **AGR** ?a **VFORM** inf) →
   (*TO* ***AGR*** *?a* ***VFORM*** *inf*) (VP **VFORM** base)
10. ADJP → *ADJ*
11. ADJP → (*ADJ* ***SUBCAT*** *_inf*) (VP **VFORM** inf)

Grammar 4.8 The expanded grammar showing all features

Sample Parse Tree

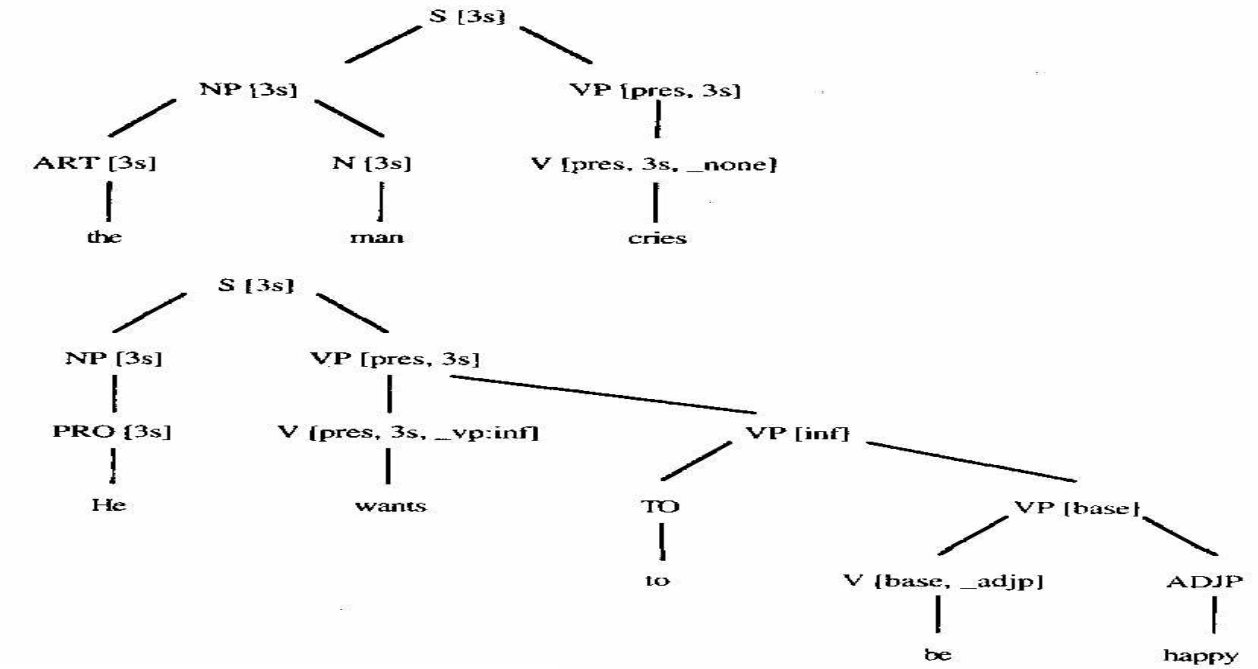


Figure 4.9 Two sample parse trees with feature values

[5]Parsing with Features:

To add a constituent C from position p_1 to p_2:

1. Insert C into the chart from position p_1 to p_2.
2. For any active arc of the form $X \rightarrow X_1 \ldots \circ C \ldots X_n$ from position p_0 to p_1, add a new active arc $X \rightarrow X_1 \ldots C \circ \ldots X_n$ from position p_0 to p_2.
3. For any active arc of the form $X \rightarrow X_1 \ldots X_n \circ C$ from position p_0 to p_1, then add a new constituent of type X from p_0 to p_2 to the agenda.

Figure 3.10 The arc extension algorithm

Bottom up Parsing Algorithm

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 p1 to p2).
3. For each rule in the grammar of form **X -> C X1 ... Xn**, add an active arc of form **X -> C o X1 .... Xn** from position p1 to p2.
4. Add C to the chart using the arc extension algorithm above.

• **context-free grammars.** This involves generalizing the algorithm for matching rules to constituents. For instance, the chart-parsing algorithms developed in Chapter 3 all used an operation for extending active arcs with a new constituent. A constituent X could extend an arc of the form

C -> C1 ... Ci o X ... Cn

• to produce a new arc of the form

C -> C1 ... Ci X o ... Cn

• A rule such as

1. (NP **AGR** ?a) -> o (**ART AGR** ?a) (N **AGR** ?a)

• says that an NP can be constructed out of an ART and an N if all three agree on the AGR feature. It does not place any restrictions on any other features that the NP, ART, or N may have. Thus, when matching constituents against this rule, the only thing that matters is the AGR feature. All other features in the constituent can be ignored. For instance, consider extending arc 1 with the constituent

2. (**ART ROOT** A **AGR** 3s)

Example: A DOG

1. (NP **AGR** ?a) -> o (**ART AGR** ?a) (N **AGR** ?a)

• For instance, consider extending arc 1 with the constituent

2. (**ART ROOT** A **AGR** 3s)

• To make arc 1 applicable, the variable ?a must be instantiated to 3s, producing

3. (NP **AGR** 3s) -> o (**ART AGR** 3s) (N **AGR** 3s)

• This arc can now be extended because every feature in the rule is in constituent 2:

4. (NP **AGR** 3s) -> (**ART AGR** 3s) o (N **AGR** 3s)

• Now, consider extending this arc with the constituent for the word dog:

5. (N **ROOT** DOG1 **AGR** 3s)

• This can be done because the AGR features agree. This completes the arc

6. (NP **AGR** 3s) —> (**ART AGR** 3s) (N **AGR** 3s)

• This means the parser has found a constituent of the form (NP AGR 3s).

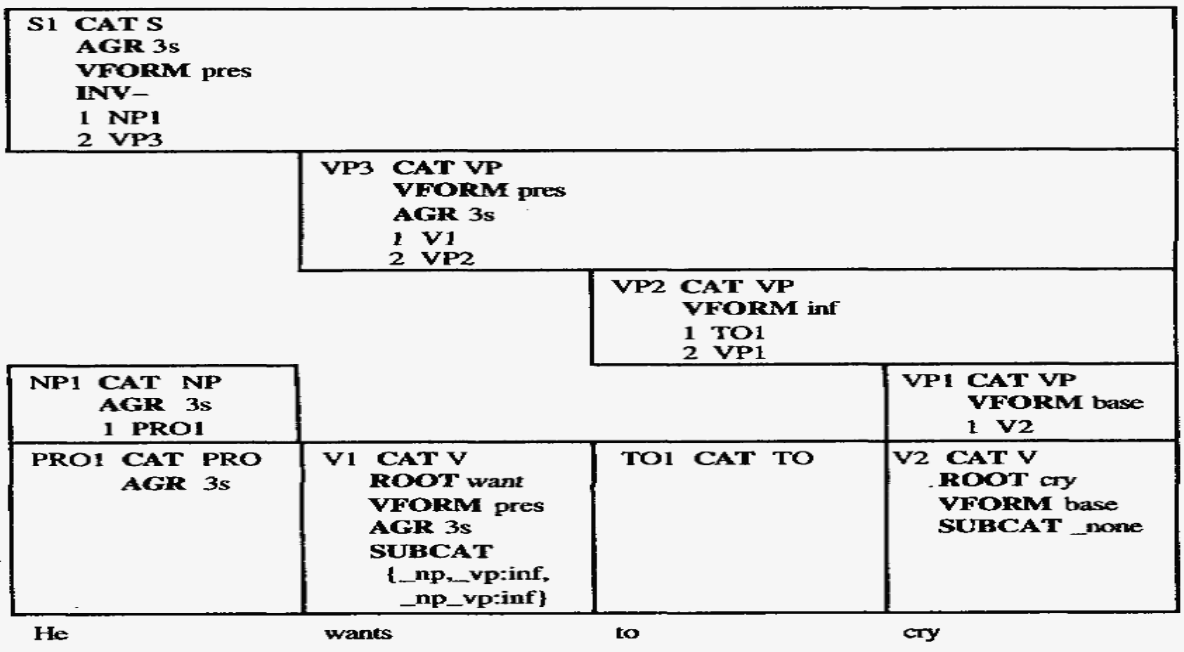


Figure 4.10 The chart for *He wants to cry.*

Algorithm

This algorithm can be specified more precisely as follows:

• Given an arc A, where the constituent following the dot is called NEXT, and a new constituent X, which is being used to extend the arc,

• **a**. Find an instantiation of the variables such that all the features specified in NEXT are found in X.

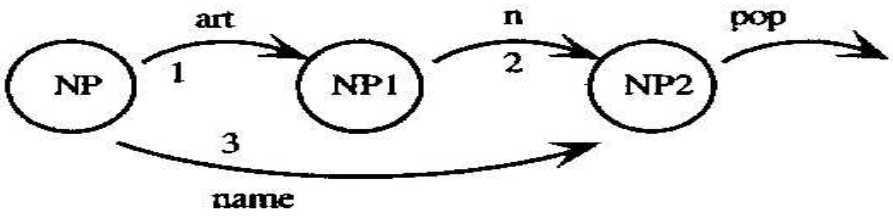
• **b.** Create a new arc A', which is a copy of A except for the instantiations of the variables determined in step(a).

• **c.** Update A' as usual in a chart parser.

[6]Augmented Transition Networks:

- Features can also be added to a recursive transition network to produce an augmented transition network (ATN).
- Features in an ATN are traditionally called registers.
- Constituent structures are created by allowing each network to have a set of registers.
- Each time a new network is pushed, a new set of registers is created. As the network is traversed, these registers are set to values by actions associated with each arc.
- When the network is popped, the registers are assembled to form a constituent structure, with the CAT slot being the network name.
- Grammar 4.11 is a simple NP network. The actions are listed in the table below the network. ATNs use a special mechanism to extract the result of following an arc.
- When a lexical arc, such as arc 1, is followed, the constituent built from the word in the input is put into a special variable named "*". The action
- **DET := ***
- then assigns this constituent to the DET register. The second action on this arc,
- **AGR := AGR***
- assigns the AGR register of the network to the value of the AGR register of the new word (the constituent in "*").
- Agreement checks are specified in the tests. A test is an expression that succeeds if it returns a nonempty value and fails if it returns the empty set or nil.

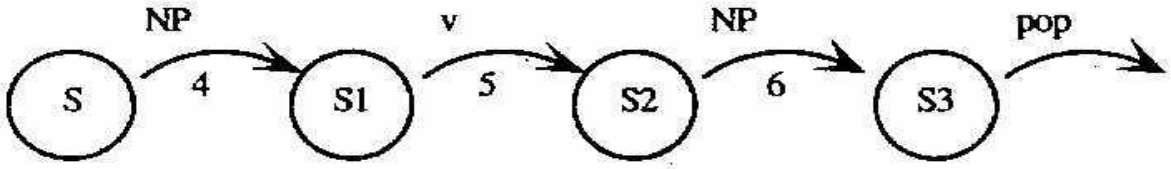
Grammar 4.11 A simple NP network



Arc	Test	Actions
1	none	DET := * AGR := AGR*
2	AGR ∩ AGR*	HEAD := * AGR := AGR ∩ AGR*
3	none	NAME := * AGR := AGR*

Grammar 4.11 A simple NP network

A simple S-network for "The Dog saw jack"



Arc	Test	Actions
4	none	SUBJ := *
5	AGR_{SUBJ} ∩ AGR*	MAIN-V := * AGR := AGR_{SUBJ} ∩ AGR*
6	none	OBJ := *

Grammar 4.12 A simple S network

Tracing S-Network

Trace of S Network

Step	Node	Position	Arc Followed	Registers Set
1.	S	1	arc 4 succeeds (for recursive call see trace below)	SUBJ ← (NP DET the HEAD dog AGR 3s)
5.	S1	3	arc 5 (checks if 3p ∩ 3p)	MAIN-V ← saw AGR ← 3p
6.	S2	4	arc 6 (for recursive call trace, see below)	OBJ ← (NP NAME Jack AGR 3s)
9.	S3	5	pop arc succeeds	returns (S SUBJ (NP DET the HEAD dog AGR 3s) MAIN-V saw AGR 3p OBJ (NP NAME Jack AGR 3s))

Trace of First NP Call: Arc 4

Step	Node	Position	Arc Followed	Registers Set
2.	NP	1	1	DET ← the AGR ← {3s 3p}
3.	NP1	2	2 (checks if {3s 3p} ∩ 3p)	HEAD ← dog
4.	NP2	3	pop	returns (NP DET the HEAD dog AGR 3s)

Trace of Second NP Call: Arc 6

Step	Node	Position	Arc Followed	Registers Set
7.	NP	4	3	NAME ← John AGR ← 3s
8.	NP2	5	pop	returns (NP NAME John AGR 3s)

Figure 4.13 Trace tests and actions used with 1 *The* 2 *dog* 3 *saw* 4 *Jack* 5

Figure 4.13 Trace tests and actions used with "1 The 2 dog 3 saw 4 Jack 5"

• With the lexicon in Section 4.3, the ATN accepts the following sentences:
• The dog cried.
• The dogs saw Jack.
• Jack saw the dogs.
• Consider an example. A trace of a parse of the sentence "The dog saw Jack" is shown in Figure4.13
• It indicates the current node in the network, the current word position, the arc that is followed from the node, and the register manipulations that are performed for the successful parse.
• It starts in the S network but moves immediately to the NP network from the call on arc 4.
• The NP network checks for number agreement as it accepts the word sequence The dog.
• It constructs a noun phrase with the AGR feature plural. When the pop arc is followed, it completes arc 4 in the S network.
• The NP is assigned to the SUBJ register and then checked for agreement with the verb when arc 3 is followed.
• The NP "Jack" is accepted in another call to the NP network.

[7]**Bayes' rule**

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

Conditional Probability : 1. P(A/B)=P(A ∩ B)/P(B)

2. P(B/A)= P(B∩A)/P(A)

1. P(A∩B)= P(A/B) * P(B)
2. P(B∩A)= P(B/A) * P(A)

Basic Probability Theory

• Probability is the measure of the likelihood that an event will occur. Probability is quantified as a number between 0 and 1.

• A probability of 1 indicates that the event is certain to occur, while a probability of 0 indicates that the event definitely will not occur.

• Any number between 0 and 1 indicates some degree of uncertainty.

• A probability of .5 would indicate that the event is equally likely to occur as not to occur, that is, a "50/50" bet.

• An event with probability .5 would occur exactly half of the time.

• An event with probability .1 would occur once in every 10 opportunities (1/10), whereas an event with probability .75 would occur 75 times out of 100 (3/4).

• Probability can be defined in terms of a **random variable**, which may range over a predefined set of values.

• While random variables may range over infinite sets and continuous values, here we will use only random variables that range over a finite set of values. For instance, consider tossing a coin.

• The random variable TOSS representing the result of a coin toss would range over two possible values: heads (h) or tails (t).

• One possible event is that the coin comes up heads - TOSS=h; the other is that the coin comes up tails - TOSS=t.

• No other value for TOSS is possible, reflecting the fact that a tossed coin always comes up either heads or tails.

Probability Function Properties

- A probability function, PROB, assigns a probability to every possible value of a random variable.
- Every probability function must have the following properties,
- where e1,...,en are the possible distinct values of a random variable E:
    1. PROB(ei) S 0, for all i
    2. PROB(ei) R 1, for all i
    3. Ìi=1,n PROB(ei)= 1

Example

- A particular horse, Harry, ran 100 races in his career. The result of a race is represented as a random variable R that has one of two values, Win or Lose. Say that Harry won 20 times overall.
- Thus the probability of Harry winning the race is **PROB**(R=Win) = .2 and the probability of him losing is **PROB**(R=Lose) = .8.
- Continuing the racing example, consider another random variable W representing the state of the weather and ranging over the values Rain and Shine.

- Let us say that it was raining **30 times** out of the **100 times** a race was run. Of these **30 races**, Harry won 15 times. Intuitively, if you were given the fact that it was raining - that is, that **W=Rain** - the probability that **R=Win** would be **.5 (15 out of 30).**
- This intuition is captured by the concept of conditional probability and is written as **PROB**(**Win | Rain**), which is often described as the probability of the event Win given the event Rain.

Conditional probability

- Conditional probability is defined by the formula
  PROB(e | e') = PROB(e & e') / PROB(e')
- where **PROB**(e & e') is the probability of the two events e and e' occurring simultaneously.
- You know that **PROB**(Rain) = **.3** and **PROB**(Win & Rain) = **.15**, and using the definition of conditional probability you can compute PROB(Win | Rain) and see that it agrees with your intuition:
- **PROB(Win | Rain) = PROB(Win & Rain) / PROB(Rain)**
  = .15 / .30
  = .5
- An important theorem relating conditional probabilities is called **Bayes' rule**. This rule relates the conditional probability of an event A given B to the conditional probability of B given A:

BAYE's RULE

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

- Where A, B Are the Events
- Using Bayes' rule we can compute the probability that it rained on a day that Harry won a race:

✞ **PROB(Rain | Win) = (PROB(Win | Rain)*PROB(Rain)) /**
PROB(Win) = (.5 * .3) / .2
= .75

✞ it is the same value as if we calculated the conditional probability directly from its definition:

PROB(Rain | Win) = (PROB(Rain & Win)* PROB(Win)
= .15 / .20
= .75

[8]Shannon's Game:

- Shannon's Game
- Guess the next letter:
  What do you think the next letter is?
- Guess the next word:
  What do you think the next word is?