UNIT–III GrammarsforNaturalLanguage

Grammars for Natural Language, Movement Phenomenon in Language, Handling questionsin Context Free Grammars, Hold Mechanismsin ATNs, Gap Threading, Human Preferences in Parsing, Shift Reduce Parsers, Deterministic Parsers.

MovementPhenomenainLanguage

Many sentence structures appear to be simple variants of other sentence structures. In some cases, simple words or phrases appear to be locally reordered; sentences are identical except that a phrase apparently is moved from its expected position in a basic sentence. This section explores techniques for exploiting these generalities to cover questions in English.

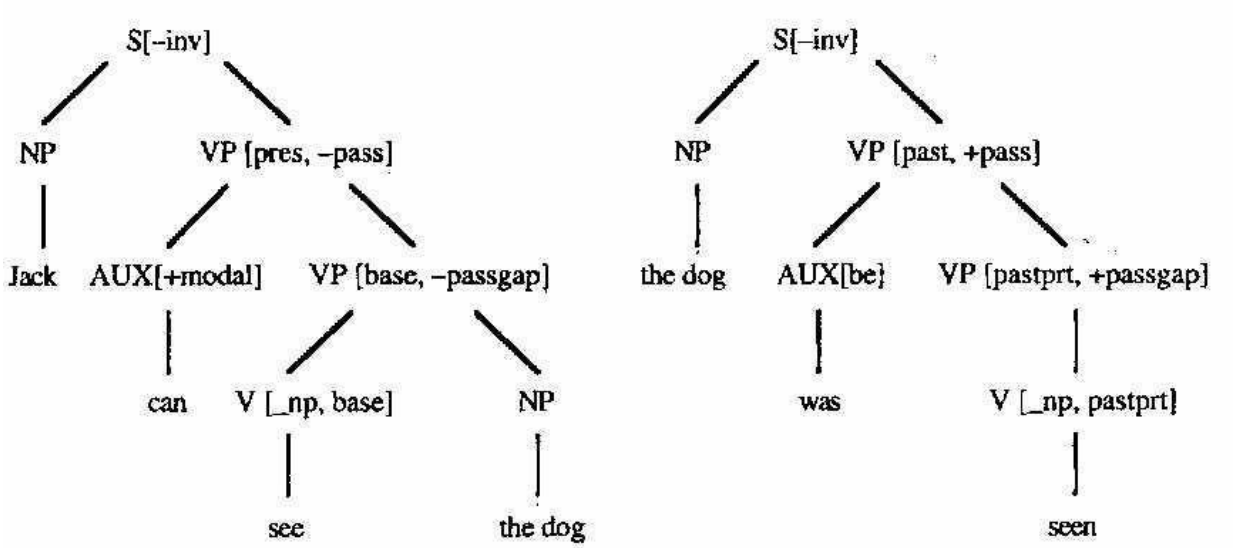


Figure 5.4 An active and a passive form sentence

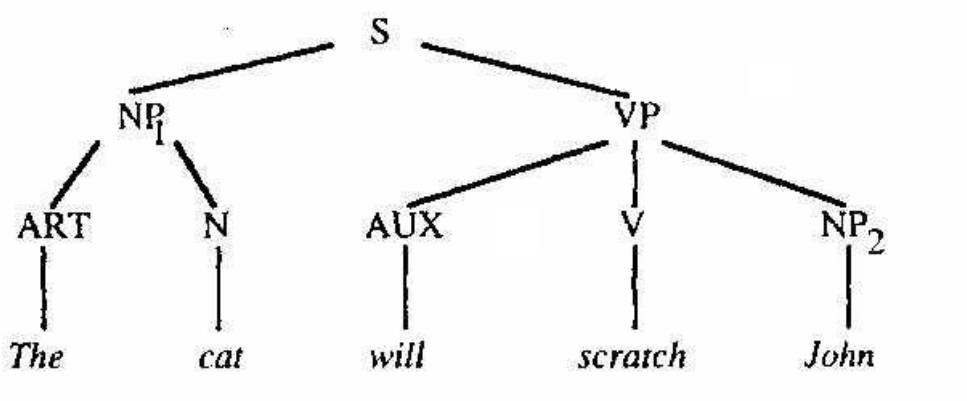
Asastartingexample,considerthestructureofyes/noquestionsandhowtheyrelatetotheir assertional counterpart. In particular, consider the following examples:

Jackis givingSueaback rub.
Hewillruninthemarathonnextyear. Is
Jack giving Sue a back rub?
Willheruninthemarathonnextyear?
As you can readily see, yes/no questions appear identical in structure to their assertional counterparts except that the subject NPs and first auxiliaries have swapped positions. If there is no auxiliary in the assertional sentence, an auxiliary of root "*do"*, in the appropriate tense, isused:
Johnwenttothestore.Henrygoestoschooleveryday.
DidJohngo tothestore?DoesHenry goto schooleveryday?
Taking a term from linguistics, this rearranging of the subject and the auxiliaryis called subject-aux inversion.

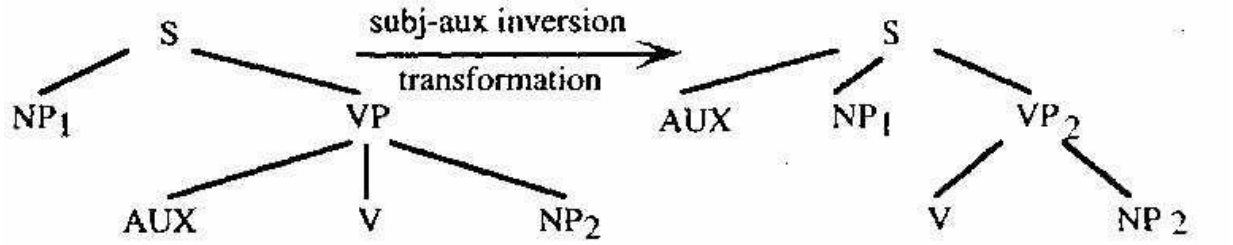
Ontheotherhand,ifyouareinterestedinhowitisdone,youmightaskoneofthefollowing questions:
Howwill thefatmanput thebook in thecorner?
Inwhatwaywill thefatmanput thebookin thecorner?
Ifyouareinterestedinotheraspects, youmightaskoneofthese questions:
Whatwillthefatmanangrilyputinthecorner? Where
will the fat man angrily put the book?
Inwhatcornerwillthefatmanangrilyputthebook? What
will the fat man angrily put the book in?
Each question has the same form as the original assertion, except that the part being questionedis removed and replaced by a wh-phrase at the beginning of the sentence. In addition, except when the part being queried is the subject NP, the subject and the auxiliary are apparently inverted, as in yes/no questions. This similarity with yes/no questions even holds for sentences without auxiliaries. In both cases, a "*do"*auxiliary is inserted:
Ifound a bookcase.
DidI find a bookcase?
WhatdidIfind?

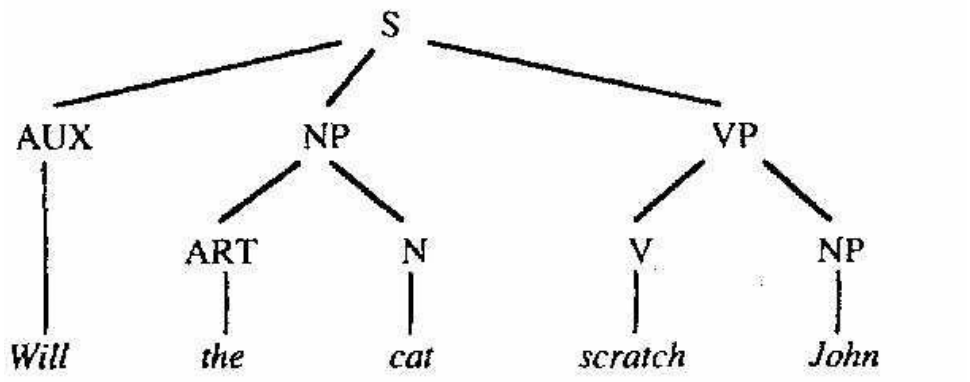
The term *movement* arose in transformational grammar (TG). TG posited two distinct levels of structural representation: surface structure, which corresponds to the actual sentence structure, and deep structure. A CFG generates the deep structure, and a set of transformations map the deep structure to the surface structure.

Forexample,the deepstructureof"*Willthecat scratchJohn?"*would be:



Theyes/noquestionisthengeneratedfromthisdeepstructurebyatransformationexpressed schematically as follows





HandlingQuestionsinContext-FreeGrammars

The main goal is to extend a context-free grammar minimally so that it can **handle questions**. We want to reuse as much of the original grammar as possible. For yes/no questions, this iseasily done. We can extend following Grammar with one rule that allows an auxiliarybefore the first NP and handles most examples:

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

S[+inv]-> (AUXAGR ?a SUBCAT?v)(NPAGR?a) (VPVFORM?v)
This enforces subject-verb agreement between the AUX and the subject NP, and ensures that the VP has the right VFORM to follow the AUX. This one rule is all that is needed to handle yes/no questions, and all of the original grammar tor assertions can be used directly for yes/noquestions.

An algorithm for automatically adding GAP features to a grammar is shown in Figure 5.5. Note that it does not modify any rule that explicitly sets the GAP feature already, allowing the grammardesignertointroducerulesthatdonotfollowtheconventionsencodedin thealgorithm. In particular, the rule for subject-aux inversion cannot allow the gap to propagate to the subject NP.

For each rule $Y \rightarrow X_1 \ldots H_i \ldots X_n$ with head constituent H_i

1. If the rule specifies a GAP feature in some constituent already, then skip.
2. If the head H_i is not a lexical category, then add a GAP feature to the head and the mother, and –GAP to the other subconstituents, producing a rule of form:
   $(Y\ \mathbf{GAP}\ ?g) \rightarrow (X_1\ \mathbf{GAP}-) \ldots (H_i\ \mathbf{GAP}\ ?g) \ldots (X_n\ \mathbf{GAP}-)$
3. If the head H_i is a lexical category, then for each nonlexical subconstituent X_j, add a rule of the form:
   $(Y\ \mathbf{GAP}\ ?g) \rightarrow (X_1\ \mathbf{GAP}-) \ldots (X_j\ \mathbf{GAP}\ ?g) \ldots (X_n\ \mathbf{GAP}-)$

Figure 5.5 An algorithm for adding GAP features to a grammar

Using this procedure, a new grammar can be created that handles gaps. All that is left to do is analyze wherethe fillers for the gaps come from. In wh questions, the fillers aretypicallyNPs or PPs at the start of the sentence and are identified by a new feature WH that identifies a class of phrases that can introduce questions. The WH feature is signaled by words such as *who, what, when, where, why,* and *how* (as in *how many* and *how carefully).* These words fall into several different grammatical categories, as can be seen by considering what type of phrases they replace.In particular, *who, whom,* and *what* can appear as pronouns and can be used to specify simple NPs:

Whoatethepizza?

Whatdid youputthebook in?

Thewords"*what"*and"*which"*mayappearasdeterminers innounphrases,as in

Whatbook didhesteal?

Wordssuchas"*where"*and"*when"*appear asprepositionalphrases:

Wheredid youputthebook?

Theword"*how"*actsasanadverbialmodifierto adjectiveandadverbialphrases:

Howquicklydidherun?

Finally,theword"*whose"*actsasapossessivepronoun:

Whosebook didyoufind?

AsimpleNPandPPgrammarhandlingwh-words

1. (NP **POSS** ?p **WH** ?w) → (*PRO* ***POSS*** *?p* ***WH*** *?w*)
2. (NP **WH** ?w) → (DET **WH** ?w **AGR** ?a) (*CNP* ***AGR*** *?a*)
3. CNP → *N*
4. CNP → ADJ *N*
5. DET → *ART*
6. (DET **WH** ?w) → (*NP*[*+POSS*] ***WH*** *?a*)
7. (DET **WH** ?w) → (*QDET* ***WH*** *?w*)
8. (PP **WH** ?w) → *P* (NP **WH** ?w)
9. (PP **WH** ?w) → (*PP-WRD* ***WH*** *?w*)

Head feature for NP, DET and CNP: AGR
Head feature for PP: PFORM

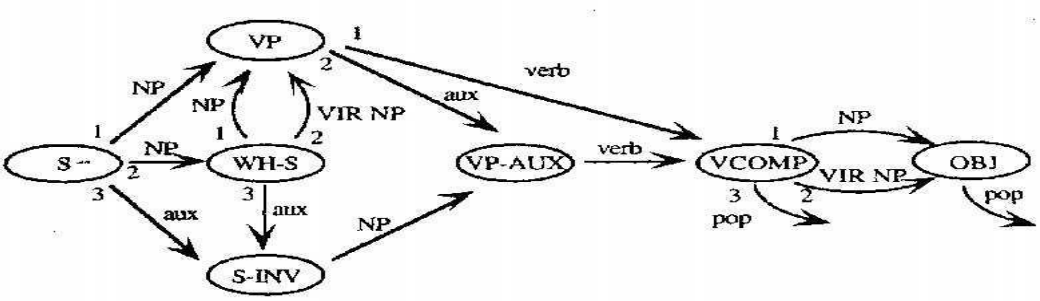
Grammar 5.7 A simple NP and PP grammar handling wh-words

HoldMechanismsinATNs

Another technique for handling movement was first developed with the ATN framework. A data structure called the hold list maintains the constituents that are to be moved. Unlike GAP features, more than one constituent may be on the hold list at a single time. Constituents are added to the hold list by a new action on arcs, the hold action, which takes a constituent and places it on the hold list.

The hold action can store a constituent currently in a register (for example, the action HOLD SUBJ holds the constituent thatis inthe SUBJ register). To ensure that a held constituent is always used to fill a gap, an ATN system does not allow a pop arc to succeed from a network until any constituent held by an action on an arc in that network has been used. That is, the held constituent must have been used to fill a gap in the current constituent or in one of its subconstituents.

Finally, you need a mechanism to detect and fill gaps. A new arc called VIR (for virtual) that takes a constituent name as an argument can be followed if a constituent of the named categoryis present on the hold list. If the arcis followed successfully, the constituent is removed from the hold list and returned as the value of the arc in the identical form that a PUSH arc returns a constituent.



Arc	Test	Actions
S/1		INV := – SUBJ := *
S/2	WH* ∩ {Q R}	WH := WH* HOLD *
S/3	VFORM* ∩ {past pres}	INV := + AUX := *
WH-S/1		SUBJ := *
WH-S/2		SUBJ := *
WH-S/3	VFORM* ∩ {past pres}	AUX := *
VP/1	AGR$_{SUBJ}$ ∩ AGR*	MAIN-V := *
	VFORM* ∩ {past pres}	
VP/2	AGR$_{SUBJ}$ ∩ AGR*	AUX := *
	VFORM* ∩ {past pres}	
S-INV/1	AGR$_{AUX}$ ∩ AGR*	SUBJ := *
VP-AUX/1	SUBCAT$_{AUX}$ ∩ FORM*	MAIN-V := *
VCOMP/1	SUBCAT$_{MAIN-V}$ ∩ _np	OBJ := *
VCOMP/2	SUBCAT$_{MAIN-V}$ ∩ _np	OBJ := *
VCOMP/3	SUBCAT$_{MAIN-V}$ ∩ _none	

Grammar 5.14 An S network for questions and relative clauses

The network is organized so that all +INV sentences go through node S-INV, while all -INV sentences gothroughnodeVP.Allwh-questionsandrelativeclauses gothroughnodeWH-Sand then are redirected back into the standard network based on whether or not the sentence is inverted. The initial NP is put on the hold list when arc Sf2 is followed. For noninverted questions,suchas *Who atethepizza?,* and forrelativeclauses intheNPof form *themanwhoate the pizza*, the held NP is immediately used bythe VIR arc WH-S/2. For other relative clauses, as in the NP *the man who we saw,* arc WH-S/1 is used to accept the subject in the relative clause, and the held NP is used later on arc VCOMP/2. For inverted questions, such as *Who do I see?,* arc WH-S/3 is followed to accept the auxiliary, and the held NP must be used later.

This network only accepts verbs with SUBCAT values _none and _np but could easily be extended to handle other verbcomplements. When extended, there would beamuch widerrange of locations where the held NP might be used. Figure 5.15 shows a trace of the sentence "*The man who we saw cried".* In parsing the relative clause, the relative pronoun *who* is held in step *5* and used by a VIR arc in step 8.

Notethat this ATNwould not accept **Who did themanseetheboy?,* as theheld constituent *who* is not usedbyanyVIR arc; thus thepop arcfrom theS networkcannot be taken. Similarly, **The man who the boy cried ate the pie* is unacceptable, as the relative pronoun is not used by a VIR arc in the S network that analyzes the relative clause.

GapThreading

A third method for handling gaps combines aspects of both the GAP feature approach and the hold list approach. This technique is usually called gap threading. It is often used in logic grammars, where two extra argument positions are added to each predicate—one argument for a list of fillers that might be used in the current constituent, and one for the resulting list of fillers that were not used after the constituent is parsed. Thus the predicate

s(position-in,position-out,fillers-in,fillers-out)

is true onlyif there is a legal S constituent between *position-in* and *position-out* of the input. If a gap was used to build the 5, its filler will be present in *fillers-in,* but not in *fillers-out.* For example,anSconstituentwithanNP gapwouldcorrespondtothepredicates(ln,Out,(NP],nil). In cases where there are no gaps in a constituent, the *fillers-in* and *fillers-out* will be identical.

Consideranexampledealingwithrelativeclauses.TherulesrequiredareshowninGrammar 5.16.Thevariousfeaturerestrictionsthatwouldbeneededtoenforceagreementandsubcategorization are not shown so as to keep the example simple.

Alogical Grammar forGAP threading:

```
1.  s(In, Out, FillersIn, FillersOut) :- np(In, In1, FillersIn, Fillers1),
                                         vp(In1, Out, Fillers1, FillersOut)
2.  vp(In, Out, FillersIn, FillersOut) :- v(In, In1)
3.  vp(In, Out, FillersIn, FillersOut) :- v(In, In1), np(In1, Out, FillersIn, FillersOut)
4.  np(In, Out, Fillers, Fillers) :- art(In, In1), cnp(In1, Out)
5.  np(In, Out, Fillers, Fillers) :- pro(In, Out)
6.  cnp(In, Out) :- n(In, In1), np-comp(In1, Out)
7.  np-comp(In, In) :-
    (This covers the case where there is no NP complement.)
8.  np-comp(In, Out) :- rel-intro(In, In1, Filler),
                        s(In1, Out, (Filler nil), nil)
    (Here we hold the Rel-Intro constituent, and must use it in the following S.)
9.  rel-intro(In, Out, [NP]) :- relpro(In, Out)
    (where relpro accepts any pronoun with WH feature R)
10. np(In, In, [NP | Fillers], Fillers) :-
    (This rule builds an empty np from a filler.)
```

Grammar 5.16 A logic grammar using gap threading

Toseetheserulesinuse, considertheparseofthesentence*Themanwhowesawcried*inFigure 5.17. The relative clause is analyzed starting with step 7. Using rule 9, the word *who* is recognized as a relative pronoun, and the variable Filler is bound to the list [NP]. This filler is then passed into the embedded S (step 9), to the NP (step 10), and then on to the VP (step 12), since it is not used in the NP. From there it is passed to the NP predicate in step 14, which uses the filler according to rule 10. Note that no other NP rule could have applied at this point, because the filler must be used since the FillersOut variable is nil. Only rules that consume the fillercan apply. Oncethis gap is used,theentireNP from positions1 to6has been foundand the rest of the parse is straightforward.

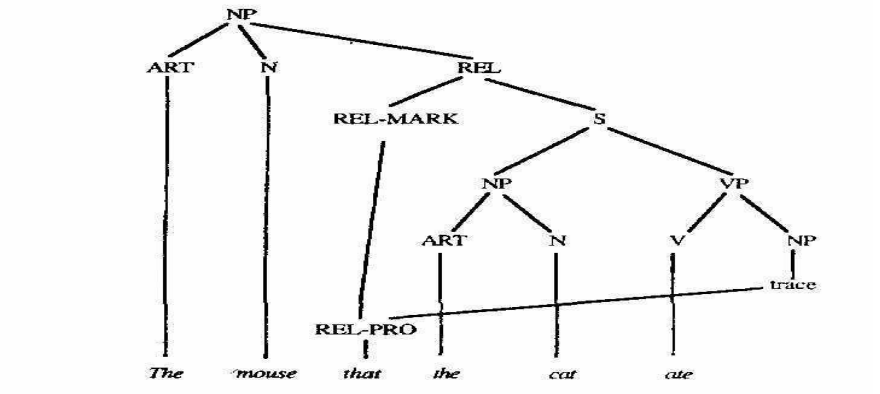


Figure 5.18 A parse tree in extraposition grammar

HumanPreferencesinParsing

Generally the people willknow simple rules to parse sentences. The Psycholinguists have conducted many investigations into parsing using a variety of techniques. These studies haverevealed some general principles concerning how people resolve ambiguity.

1.1	S → NP VP	1.4	NP → ART N
1.2	VP → V NP PP	1.5	NP → NP PP
1.3	VP → V NP	1.6	PP → P NP

Grammar 6.1 A simple CFG

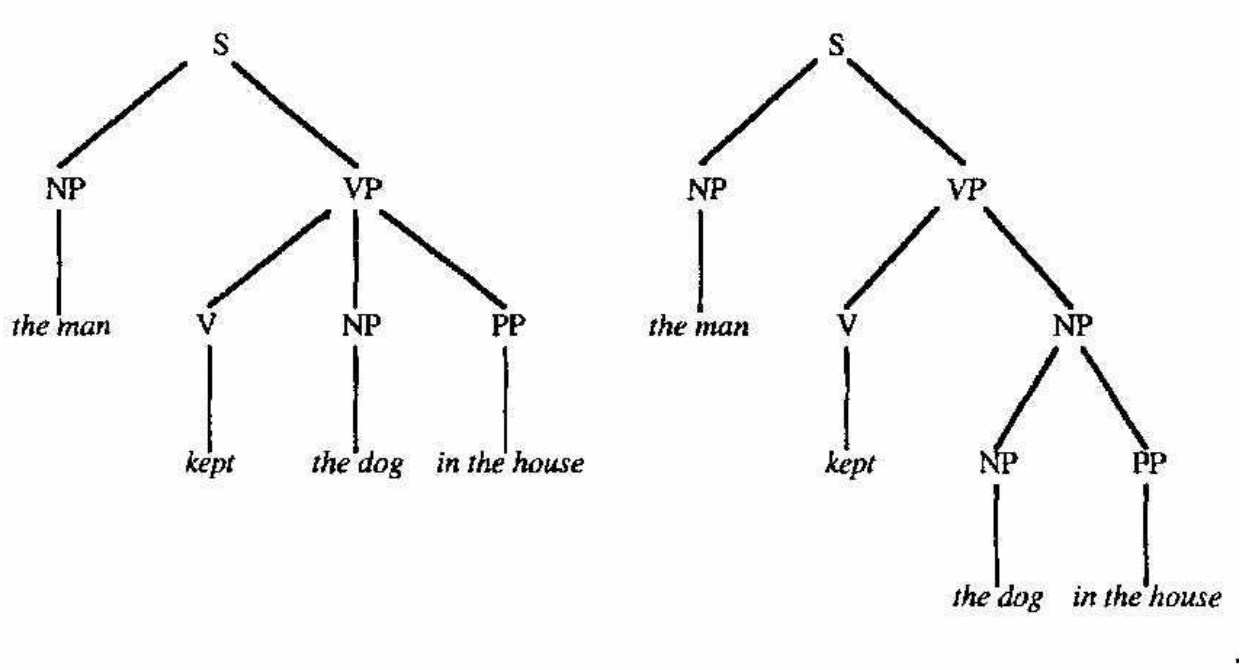


Figure 6.2 The interpretation on the left is preferred by the minimal attachment principle

The most basic result from these studies is that people do not give equal weight to all possible syntactic interpretations. This can be illustrated by sentences that are temporarily ambiguous, which cause a conscious feeling of having pursued the wrong analysis, as in the sentence "*The raft floated down the river sank"*. When you read the word "*sank"*you realize that the interpretation you have constructed so far for the sentence is not correct. In the literature, such sentences are often called "**garden-path"**sentences, based on the expression about leading someone down a garden path. Here are a few of the general principles that appear to predict when garden paths will arise.

MinimalAttachment

The most general principle is called the **minimal attachment** principle**,** which states that there is a preference for the syntactic analysis that creates the least number of nodes in the parse tree. Thus, given Grammar 6.1, the sentence "*The man kept the dog in the house"*would be interpreted with the PP"*inthehouse"*modifyingtheverbratherthantheNP"*thedog"*.Thesetwointerpretationsare

shown in Figure 6.2. The interpretation with the PP attached to the VP is derived usingrules 1.1, 1.2, and 1.6 and three applications of rule 1.4 for the NPs. The parse tree has a total of 14 nodes. The interpretation with the PP attached to the NP is derived using rules 1.1, 1.3, 1.5, and 1.6 and three applications of rule 1.4, producing a total of 15 nodes in the parse tree. Thus this principle predicts that the first interpretation is preferred, which probably agrees with your intuition.

This principle appears to be so strong that it can cause certain sentences to be almost impossible to parse correctly. One example is the sentence

Wepaintedallthewallswithcracks.

which, against allcommon sense, is often readas meaningthatcracks werepaintedontothe walls, or that cracks were somehow used as an instrument to paint the walls. Both these anomalous readings arise from the PP being attached to the VP *(paint)* rather than the NP *(the walls).* Another classic example is the sentence

Thehorseracedpastthebarn fell.

which has a reasonable interpretation corresponding to the meaning of the sentence "*The horse that was raced past the barn fell".* In the initial sentence, however, creating a reduced relative clausewhen the word "*raced"*is encountered introduces many more nodes than the simple analysis where "*raced"*is the main verb of the sentence. Of course, this second interpretation renders the sentence unanalyzable when the wordfell is encountered.

RightAssociation

The second principle is called right association or late closure. This principle states that, all other things being equal, new constituents tend to be interpreted as being part of the current constituent under construction (rather than part of some constituent higher in the parse tree). Thus, given the sentence

GeorgesaidthatHenryleftinhiscar.

the preferred interpretation is that Henry left in the car rather than that George spoke in the car. Both interpretations are, of course, syntactically acceptable analyses. The two interpretations are shown in Figure 6.3. The former attaches the PP to the VP immediately preceding it, whereas the latterattaches the PP to the VP higher in the tree. Thus the right association principle prefers the former. Similarly, the preferred interpretation for the sentence "*I thought it would rain yesterday"*is that yesterday was when it was thought to rain, rather than the time of the thinking.

LexicalPreferences

In certain cases the two preceding principles seem to conflict with each other. In the sentence "*The man kept the dog in the house",* the principle of right association appears to favor the interpretation in which the PP modifies the dog, while the minimal attachment principle appears tofavorthePPmodifyingtheVP.Youmightsuggestthatminimalattachmenttakespriorityover right association in such cases; however, the relationship appears to be more complex than that. Consider the sentences

1. Iwantedthedoginthehouse.
2. Ikeptthedogin thehouse.
3. Iputthedoginthe house.

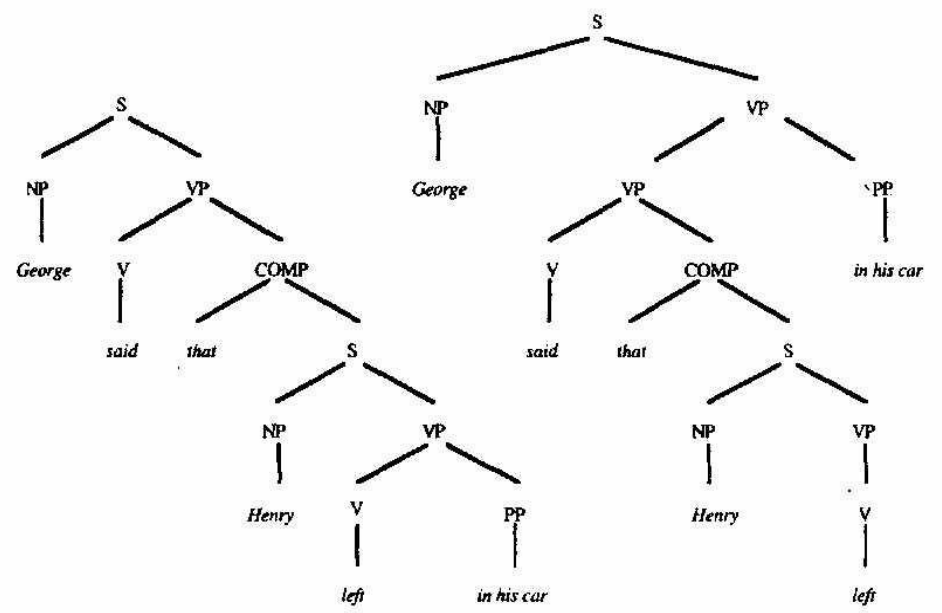


Figure 6.3 Two interpretations of *George said that Henry left in his car.*

ThePP "*in thehouse"*insentence Iseems most likelyto bemodifying "*dog"*(al though theother interpretationispossible,asinthesense"*Iwantedthedogtobeinthehouse").* Insentence2, the PP seems most likely to be modifying the VP (although modifying the NP is possible, as in "*I kept the dog that was in the house").* Finally, in sentence 3, the PP is definitely attached to the VP, and no alternative reading is possible.

These examples demonstrate that lexical items, in this case the verb used, can influence parsing preferences. In many cases, the lexical preferences will override the preferences based on the general principles. For example, if a verb subcategorizes for a prepositional phrase, then somePP must be attached to the VP. Other PP5 might also be identified as having a strong preference forattachmentwithinthe VP. Ifneitherofthese casesholds,thePP willbeattachedaccordingto the general principles.
Thus, for the preceding verbs, "*want"*has no preference for any PPs, whereas "*keep"*mightprefer PPs with prepositions "*in", "on",* or "*by"*to be attached to the VP. Finally, the verb "*put"*requires (subcategorizes for) a PP beginning with "*in", "on", "by",* and so on, which must be attached to the VP.

2.1 $S \rightarrow NP\ VP$	2.3 $VP \rightarrow AUX\ V\ NP$
2.2 $NP \rightarrow ART\ N$	2.4 $VP \rightarrow V\ NP$

Grammar 6.4 A simple grammar with an AUX/V ambiguity

ShiftReduceParsers

The Shift Reduce parsers are used to improve the efficiency of parsers in NLP. The uncertaintyis passed forward through the parse to the point where the input eliminates all but one of the possibilities. The efficiency of the technique described in this section arises from the fact that all the possibilities are considered in advance, and the information is stored in a table that controls the parser, resulting in parsing algorithms that can be much faster than described thus far.

These techniques were developed for use with unambiguous context-free grammars - grammars for which there is at most one interpretation for any given sentence. While this constraint is reasonable for programming languages, it is clear that there is no unambiguous grammar for natural language. But these techniques can be extended in various ways to make them applicable to natural language parsing.

SpecifyingtheParserState

Consider using this approach on the small grammar in Grammar 6.4. The technique involves predetermining all possible parser states and determining the transitions from one state to another. A parser state is defined as the complete set of dotted rules (that is, the labels on the active arcs in a chart parser) applicable at that position in the parse. It is complete in the sense that if a state contains a rule of the form **Y -> ... o X ...**, where **X** is a nonterminal, then all rules for **X** are also contained in the state. For instance, the initial state of the parser would include the rule

S-> o NP VP

aswellasalltherulesforNP,whichinGrammar6.4is only

NP-> oART N

Thustheinitialstate,S0,couldbesummarizedasfollows:

InitialStateS0:S->oNPVP NP -

> o ART N

In other words, the parser starts in a state where it is looking for an NP to start building an S and looking for an ART to build the NP. What states could follow this initial state? To calculate this, consider advancing the dot over a terminal or a nonterminal and deriving a new state. If you pick the symbol

ART,theresultingstateis

State S1: NP-> ARTo N

IfyoupickthesymbolNP,theruleis

S-> NP o VP

inthenewstate.NowifyouexpandouttheVPtofindallitspossiblestartingsymbols, you getthe following:

StateS2:S->NPoVP

VP -> o AUX V NPVP

-> o V NP

Now,expandingSI,ifyou havetheinputN,you getastateconsistingofacompleted rule:

StateS1':NP->ARTNo

Expanding52,aVwouldresultinthestate

StateS3:VP->VoNP NP

-> o ART N

AnAUX fromS2wouldresultinthestate

State S4: VP-> AUXo VNP
andaVPfrom52wouldresultinthestate
State S2': S-> NPVPo
Continuingfromstate53withanART,you find yourselfinstateSiagain,asyouwouldalso ifyou expand from S0 with an ART. Continuing from S3
withanNP,ontheotherhand,yieldsthenew state
State S3':VP ->V NP o

ContinuingfromS4withaV yields
StateS5:VP->AUXVoNP NP ->
o ART N

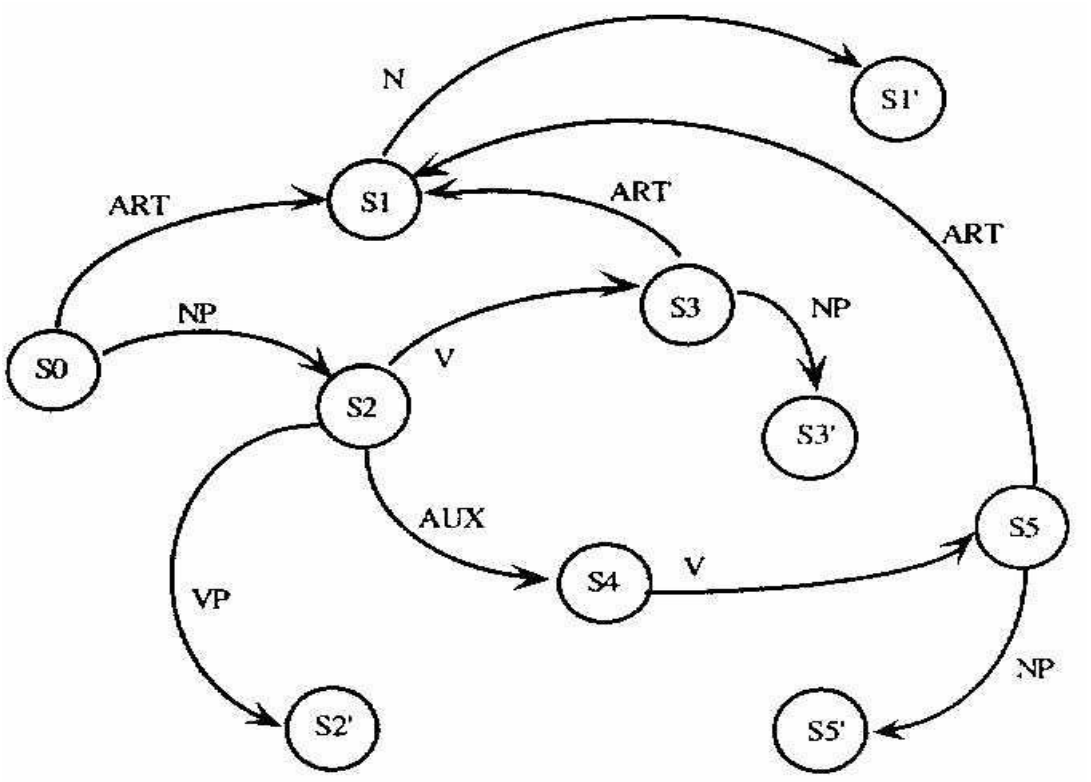


Figure 6.5 A transition graph derived from Grammar 6.1

andcontinuingfromS5withanARTwouldproducestateS1again.Finally,continuingfromS5with an NP would produce the state
State S5':VP->AUXV NP o
Now that this process is completed, you can derive a transition graph that can be used to control the parsing of sentences, as is shown in Figure 6.5.

AShift-ReduceParser

These states can be used to control a parser that maintains two stacks: the**parse stack**, which contains parse states (that is, the nodes in Figure 6.5) and grammar symbols; and the input stack, which contains the input and some grammar symbols. At any time the parser operates using the information specified for the top state on the parse stack. The states are interpreted as follows. The states that consist of a single rule with the dot at the far right-hand side, such as S2',

S-> NP VP o

indicate that the parser should rewrite the top symbols on the parse stack according to this rule. This is called a reduce action. The newly derived symbol(S in this case) is pushed onto the top of the input stack.

Anyother state not containing anycompleted rules is interpreted bythe transition diagram. If the top input symbol matches an arc, then it and the new state (at the end of the arc) are pushed onto theparse stack. This is called the **shift action.** Using this interpretation of states you can construct a table, called the oracle, that tells the parser whattodo in everysituation.Theoracle for Grammar 6.4 is shown in Figure 6.6. For each state and possible input, it specifies the action and the next state. Reduce actions can be applied regardless of the next input, and the accept action only is possible when the input stackis empty(that is, the next symbol is theemptysymbol e).Theparsingalgorithm for using an oracle is specified in Figure 6.7.

Considerparsing"*Themanatethecarrot"*.Theinitialstateoftheparseris

ParseStack	**InputStack**
(S0)	**(Themanatethecarrot)**

LookinguptheentryinthetableinFigure6.6forstateSOfortheinputART(thecategoryofthe word *the),* you see a shift action and a move to state S1:

ParseStack	**InputStack**
(S1ARTS0)	**(manatethecarrot)**

LookinguptheentryforstateS1fortheinputN,youseeashiftactionandamovetostateS1:

Parse Stack	**Input Stack**
(S1' N S1 ART S0)	**(atethecarrot)**

Lookinguptheentryforstate Si',youthenreducebyrule2.2,whichremovestheSi,,N,51,and ART from the parse stack and adds NP to the input stack:

ParseStack	**InputStack**
(S0)	**(NPatethe carrot)**

Again,consultingthetableforstateS0withinputNP, younowdoashiftandmovetostateS2:

ParseStack	**InputStack**
(S2NPS0)	**(atethe carrot)**

Next,thethreeremainingwordsallcauseshiftsandamovetoanewstate,endingupwiththeparse state:

Parse Stack	**InputStack**
(S1' N S1 ART S3 V S2 NP S0)	**( )**

Thereduceactionbyrule2.2specifiedinstateS1'popstheNandARTfromthestack(thereby popping S1 and S1' as well), producing the state:

ParseStackInput Stack	
(S3VS2NPS0)	**(NP)**

YouarenowbackatstateS3,withanNPintheinput,andafterashifttostateS3',youreduceby rule 2.4, producing:

ParseStack	**InputStack**

This algorithm uses the following information:

- *Action*(S, W)—a function that maps a state and an input constituent to one of the values *shift*, *reduce i*, or *accept*
- *GoTo*(S, W)—a function that maps a state and an input constituent to a new state
- a parse stack of form $(S_n\ C_n \ldots S_1\ C_1\ S_0)$, where S_i are parse states and C_i are constituents
- an input stack of form $(W_1 \ldots W_n)$, where W_i is a constituent symbol or word

The parser operates by continually executing the following steps until success or failure:

1. If *Action*(S_n, W_1) = *Shift*, and *GoTo*$(S_n, W_1) = S$, then remove W_1 from the input stack and push it on the parse stack, and then push S onto the parse stack, resulting in the following stacks:

   parse stack: $(S\ W_1\ S_n\ C_n \ldots S_1\ C_1\ S_0)$
   input stack: $(W_2 \ldots W_n)$

2. If *Action*(S_n, W_1) = *Reduce i* and grammar rule i has n constituents on its right-hand side, then remove 2n elements from the parse stack, and push the left-hand side of rule i onto the input stack. For example, if rule i were NP → ART N, then the new state would be

   parse stack: $(S_{n-2}\ C_{n-2} \ldots S_1\ C_1\ S_0)$
   input stack: $(NP\ W_1 \ldots W_n)$

3. If *Action*(S_n, W_1) = *Accept*, then the parser has succeeded.
4. If *Action*(S_n, W_1) is not defined, then the parser has failed.

Figure 6.7 The parsing algorithm for a shift-reduce parser

State	Top Input Symbol	Action	GoTo
S0	ART	Shift	S1
S0	NP	Shift	S2
S0	S	Shift	S0'
S0'	ε	Succeed	-----
S1	N	Shift	S1'
S1'	----------	Reduce by rule 2.2	-----
S2	V	Shift	S3
S2	AUX	Shift	S4
S2	VP	Shift	S2'
S2'	----------	Reduce by rule 2.1	-----
S3	ART	Shift	S1
S3	NP	Shift	S3'
S3'	----------	Reduce by rule 2.4	-----
S4	V	Shift	S5
S5	ART	Shift	S1
S5	NP	Shift	S5'
S5'	----------	Reduce by rule 2.3	-----

Figure 6.6 The oracle for Grammar 6.4

DeterministicParsers.

A deterministic parser can be built that depends entirely on matching parse states to direct its operation. Instead of allowing only shift and reduce actions, however, a richer set of actions is allowed that operates on an input stack called the buffer. (**The cat ate the fish**).

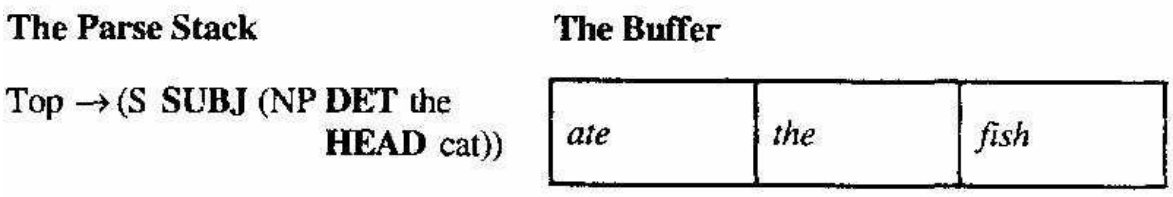


Figure 6.8 A situation during a parse

Ratherthanshiftingconstituentsontotheparsestacktobelaterconsumed byareduceaction,the parser builds constituents incrementally by attaching buffer elements into their parentconstituent, an operation similar to feature assignment. Rather than shifting an NP onto the stack to be used later in a reduction S -> NP VP, an S constituent is created on the parse stack and the NP is attached to it. Specifically, this parser has the following operations:

- Createanewnodeontheparsestack(to pushthesymbolonto thestack)
- Attachan inputconstituent to thetop nodeonthe parse stack
- Dropthe topnodeinthe parsestack intothe buffer

The drop action allows a completed constituent to be reexamined by the parser, which will then assign it a role in a higher constituent still on the parse stack. This technique makes the limited lookahead technique surprisingly powerful.

To get a feeling for these operations, consider the situation in Figure 6.8, which might occur in parsing the sentence "*The cat ate the fish".* Assume that the first NP has been parsed andassigned to the SUBJ feature of the S constituent on the parse stack. The operations introduced earlier can be used to complete the analysis. Note that the actual mechanism for deciding what operations to do has not yet been discussed, but the effect of the operations is shown here to provide intuition about the data structure. The operation

AttachtoMAIN-V

wouldremovethelexical entryfor*ate* from thebufferand assign ittothe MAIN-Vfeatureinthe S on the parse stack. Next the operation

CreateNP

wouldpushanemptyNPconstituentontotheparsestack,creatingthesituationinFigure6.9.Next the two operations

AttachtoDET

AttachtoHEAD

would successfully build the NP from the lexical entries for "*the"*and "*fish".* The input buffer would now be empty.

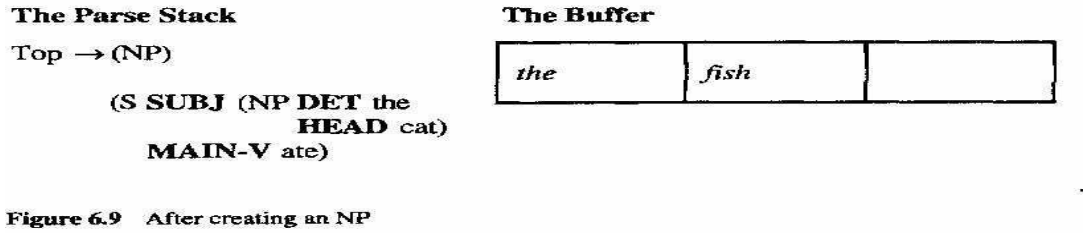


Figure 6.9 After creating an NP

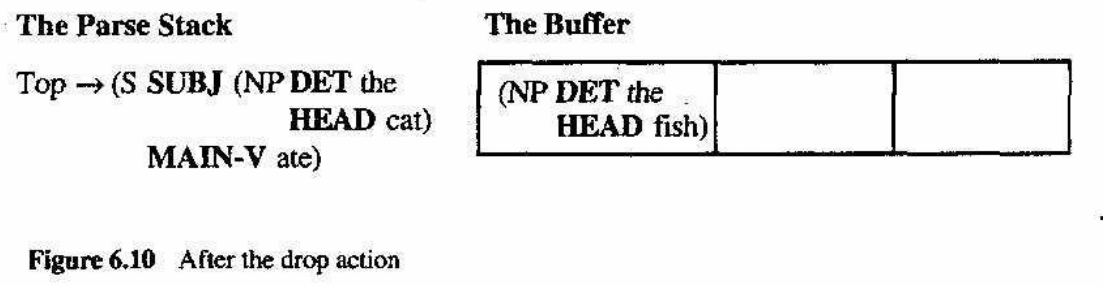


Figure 6.10 After the drop action

Theoperation

Drop

pops the NP from the parse stack and pushes it back onto the buffer, creating the situation in Figure 6.10.