UNIT III: Grammars for Natural Language

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

UNIT III: Grammars for Natural Language:

1) Grammars for Natural Language.
2) Movement Phenomenon in Language,
3) Handling questions in Context Free Grammars,
4) Hold Mechanisms in ATNs,
5) Gap Threading,
6) Human Preferences in Parsing,
7) Shift Reduce Parsers,
8) Deterministic Parsers.

[1]Grammar for Natural Language:

5.1 Auxiliary Verbs and Verb Phrases

English **sentences** typically contain a **sequence of auxiliary verbs** followed by **a main verb,** as in the following:

I can see the house.

I will have seen the house.

I was watching the movie.

I should have been watching the movie.

These may at first appear to be **arbitrary sequences of verbs**, including **have, be, do, can, will, and so on,** but in fact there is a **rich structure.**

Consider how the **auxiliaries constrain** the verb that follows them. In particular, the auxiliary have must be followed by

a) **a past participle** form (either another auxiliary or the main verb),

b) and the **auxiliary** be must be followed by a **present participle** form, or, in the case of **passive sentences**,

c) by the **past participle** form. The auxiliary do usually occurs alone but can accept a base form following it **(I did eat my carrots!)**. **Auxiliaries** such as can and must always be followed by a **base form**. In addition, the first auxiliary (or verb) in the sequence must agree with the subject in **simple declarative sentences** and be in a finite form **(past or present).**

For **example:**

I going, *we be gone,* and **they am are all **unacceptable.**

This section explores how to capture the **structure of auxiliary forms** using a combination of **new rules** and **feature restrictions.**

The **principal idea** is that **auxiliary verbs** have **subcategorization features** that restrict their **verb phrase complements.**

To **develop** this, a clear distinction is made between **auxiliary** and **main verbs**. While some **auxiliary verbs** have many of the **properties of regular verbs,** it is important to distinguish them.

For **example**, **auxiliary verbs** can be placed **before** an **adverbial *not*** in a sentence, whereas a main verb cannot:

I am not going!
You did not try it.
He could not have seen the car.

Auxiliary	**COMPFORM**	**Construction**	**Example**
modal	base	modal	can *see the house*
have	pastprt	perfect	have *seen the house*
be	ing	progressive	is *lifting the box*
be	pastprt	passive	was *seen by the crowd*

Figure 5.1 The COMPFORM restrictions for auxiliary verbs

In addition, **only auxiliary verbs** can precede the **subject NP in yes/no questions:**

Did you see the car?

Can I try it?

*** Eat John the pizza?**

In contrast, **main verbs** may appear as the **sole verb in a sentence,** and if made into a **yes/no** question require the addition of the auxiliary do:

I ate the pizza.	Did I eat the pizza?
The boy climbed in the window.	Did the boy climb in the window?
I have a pen.	Do I have a pen?

The **primary auxiliaries** are based on the **root** forms **"be" and "have".**

The **other auxiliaries** are called **modal auxiliaries** and generally appear only in the **finite tense forms** (**simple present** and **past**).

These **include** the following **verbs organized** in pairs corresponding roughly to **present and past verb** forms **"do (did)", "can (could)","may (might)", "shall (should)", "will (would)", "must", "need", and "dare**".

In addition, there are phrases that also serve a **modal auxiliary function**, such as **"ought to", "used to",** and **"be going to".**

Notice that **"have"** and **"be"** can be either an **auxiliary** or a main verb by these tests. Because they behave **quite differently**, these words have different **lexical entries as auxiliaries** and **main verbs** to allow for **different properties**;

Example:

The auxiliary "have" requires a **past-participle** verb phrase to follow it, whereas the **verb "have"** requires an **NP complement.**

The **basic idea** for handling **auxiliaries is to treat** them as verbs that take a **VP** as a **complement**. This **VP may itself** consist of another **auxiliary verb** and **another VP**, or be a **VP headed by a main verb.**

VP -> (AUX COMPFORM ?s) (VP VFORM ?s)

The **COMPFORM** feature indicates the **VFORM of the VP complement**. The values of this feature for the **auxiliaries** are shown in **Figure 5.1.**

There are **other restrictions** on the **auxiliary sequence**. In particular, **auxiliaries can appear only** in the following order:

	Modal +	have +	be (progressive) +	be (passive)
The song	might	have	been	being played as they left

A **modal auxiliary** can never follow **"have" or "be"** in the auxiliary sequence.
For **example,** the sentence

*** He has might see the movie already.**

If you consider **auxiliary sequences appearing** in VP complements for certain **verbs,** such as **"regret",** the **participle** forms of **"have"** as an auxiliary can be required, as in

I regret having been chosen to go.

*** I must be having been singing.**

A **binary head** feature **MAIN** could be introduced that is

+ for any main verb,

— for auxiliary verbs.

This way we can restrict the **VP complement** for "be" as follows:

VP -> AUX [be] VP[ing, +main]

For instance, treat the **"be"** in the **passive** construction as a **main verb** form rather than an auxiliary.

Another way would be **simply to add another rule** allowing a complement in the **passive form,** using **a new binary feature PASS**, which is + only if the **VP involves passive**:

VP -> AUX[be] VP[ing, +pass]

The passive rule would then be:

VP[+pass] -> AUX [be] VP[pastprt, main]

can:	(**CAT** AUX **MODAL** + **VFORM** pres **AGR** {1s 2s 3s 1p 2p 3p} **COMPFORM** base)	could:	(**CAT** AUX **MODAL** + **VFORM** {pres past} **AGR** {1s 2s 3s 1p 2p 3p} **COMPFORM** base)
do:	(**CAT** AUX **MODAL** + **VFORM** pres **AGR** {1s 2s 1p 2p 3p} **COMPFORM** base)	did:	(**CAT** AUX **MODAL** + **VFORM** past **AGR** {1s 2s 3s 1p 2p 3p} **COMPFORM** base)
be:	(**CAT** AUX **VFORM** base **ROOT** be **COMPFORM** ing)	have:	(**CAT** AUX **VFORM** base **ROOT** have **COMPFORM** pastprt)

Figure 5.2 Lexicon entries for some auxiliary verbs

o Passives:

This **form involves** using the **normal "object position"** NP as the **first NP** in the sentence and either **omitting the NP** usually in the **subject position** or putting it in a **PP** with the preposition **"by".**

For **example,** the **active voice sentences**

I will hide my hat in the drawer.

I hid my hat in the drawer.

I was hiding my hat in the drawer.

can be rephrased as the following passive voice sentences:

My hat will be hidden in the drawer.

My hat was hidden in the drawer.

My hat was being hidden in the drawer.

The **complication** here is that the **VP** in the **passive construction** is missing the **object NP.**

One way to solve this **problem** to **add a new grammatical rule** for every verb **subcategorization** that is usable only for **passive forms**, namely all rules that allow an NP to follow the verb.

A **program** can easily be **written** that would **automatically generate** such passive rules given a grammar. A

1. S[–inv] → (NP AGR ?a) (*VP* [*fin*] *AGR ?a*)
2. VP → (*AUX COMPFORM ?v*) (VP VFORM ?v)
3. VP → *AUX*[*be*] VP[ing, +main]
4. VP → *AUX*[*be*] VP[ing, +pass]
5. VP[+pass] → *AUX* [*be*] VP[pastprt, main, +passgap]
6. VP[–passgap, +main] → *V*[*_none*]
7. VP[–passgap, +main] → *V*[*_np*] NP
8. VP[+passgap, +main] → *V*[*_np*]
9. NP → (ART AGR ?a) (*N AGR ?a*)
10. NP → *NAME*
11. NP → *PRO*

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

Figure 5.3 A fragment handling auxiliaries including passives

[2] Movement Phenomenon in Language:

Many **sentence structures** appear to be **simple variants** of **other sentence structures**.

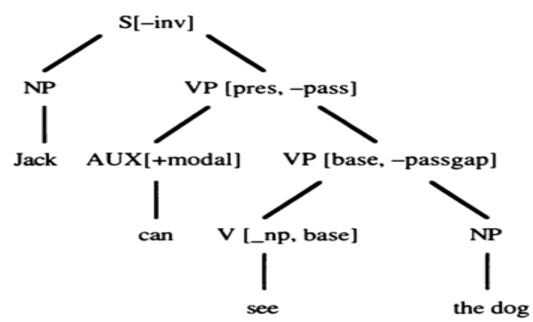
In some cases, **simple words** or phrases appear to be **locally reordered;**

sentences are **identical except** that a phrase apparently is **moved from its expected.**

1. S[−inv] → (NP **AGR** ?a) (*VP* [*fin*] ***AGR*** *?a*)
2. VP → (*AUX* ***COMPFORM*** *?v*) (VP **VFORM** ?v)
3. VP → *AUX*[*be*] VP[ing, +main]
4. VP → *AUX*[*be*] VP[ing, +pass]
5. VP[+pass] → *AUX* [*be*] VP[pastprt, main, +passgap]
6. VP[−passgap, +main] → *V*[_*none*]
7. VP[−passgap, +main] → *V*[_*np*] NP
8. VP[+passgap, +main] → *V*[_*np*]
9. NP → (ART **AGR** ?a) (*N* ***AGR*** *?a*)
10. NP → *NAME*
11. NP → *PRO*

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

Figure 5.3 A fragment handling auxiliaries including passives



1. S[−inv] → (NP **AGR** ?a) (*VP* [*fin*] ***AGR*** *?a*)
2. VP → (*AUX* ***COMPFORM*** *?v*) (VP **VFORM** ?v)
3. VP → *AUX*[*be*] VP[ing, +main]
4. VP → *AUX*[*be*] VP[ing, +pass]
5. VP[+pass] → *AUX* [*be*] VP[pastprt, main, +passgap]
6. VP[−passgap, +main] → *V*[_*none*]
7. VP[−passgap, +main] → *V*[_*np*] NP
8. VP[+passgap, +main] → *V*[_*np*]
9. NP → (ART **AGR** ?a) (*N* ***AGR*** *?a*)
10. NP → *NAME*
11. NP → *PRO*

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

Figure 5.3 A fragment handling auxiliaries including passives

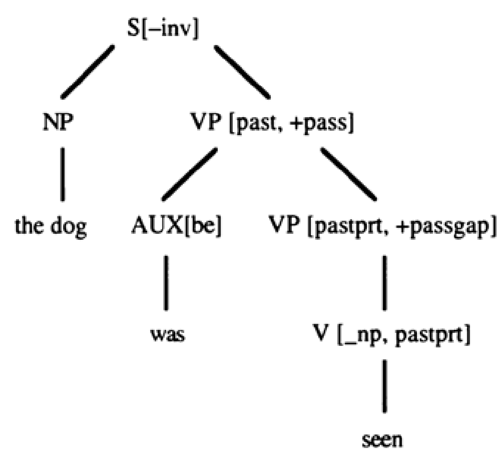


Figure 5.4 An **active** and a **passive** form **sentence.**

Jack is giving Sue a back rub. He will run in the marathon next year.
Is Jack giving Sue a back rub? Will he run in the marathon next year?

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, is used:

John went to the store. Henry goes to school every day.
Did John go to the store? Does Henry go to school every day?

Taking a term from linguistics, this rearranging of the subject and the auxiliary is called **subject-aux inversion**.

This rearrangement is precisely within the scope of the limited number of rules - local or bounded movement.

in wh-questions, it is unbounded.

The fat man will angrily put the book in the corner.

On the **other hand**, if you are **interested in how it is done**, you might ask one of the following questions:

How will the fat man put the book in the corner?

In what way will the fat man put the book in the corner?

If you are **interested in other aspects**, you **might ask one of these questions:**

What will the fat man angrily put in the corner?

Where will the fat man angrily put the book?

In what corner will the fat man angrily put the book?

What will the fat man angrily put the book in?

This similarity with **yes/no questions** even holds for **sentences without auxiliaries**. In both cases, a **"do" auxiliary is inserted**:

I found a bookcase.

Did I find a bookcase?

What did I find?

For **example,** consider the italicized VP in the sentence

What will the fat man angrily put in the corner?

While this is an **acceptable sentence, "angrily put in the corner"** does not appear to be an acceptable VP because you cannot allow sentences such as ***"I angrily put in the corner".**

Only in situations like **wh-questions** can such a **VP be allowed**, and then it is allowed only if the **wh-constituent** is of the **right form** to make a **legal VP** if it were **inserted in the sentence.**

For **example,** "What will the fat man angrily put in the corner?" is acceptable, but

***"Where will the fat man angrily put in the corner?" is not.**

The place where a **subconstituent** is missing is called the **gap,**
and the **constituent** that is moved is called the **filler.**
The techniques that follow all involve ways of allowing gaps in constituents when there is an appropriate filler available.

gap - place where subconstituent is missing

filler - constituent that is moved

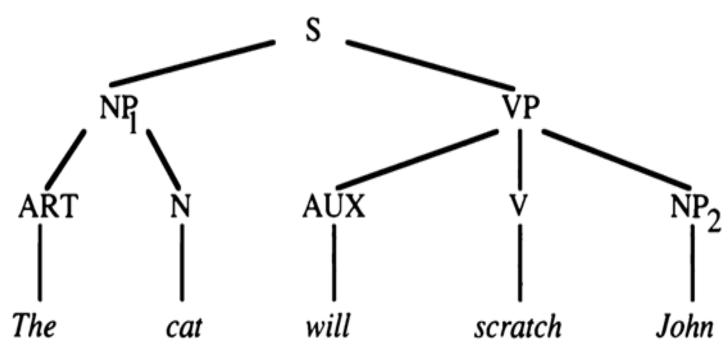
BOX 5.1 Movement in Linguistics:

The term movement arose in **transformational grammar (TG).**

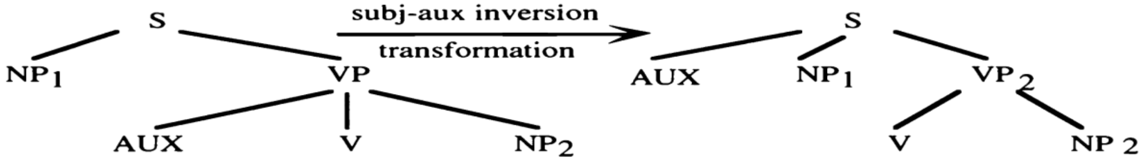
TG posited **two distinct levels** of structural representation: **surface structure,** which corresponds to the actual sentence structure, and **deep structure**.

A **CFG generates** the deep structure, and a set of **transformations** map the deep structure to the surface structure.

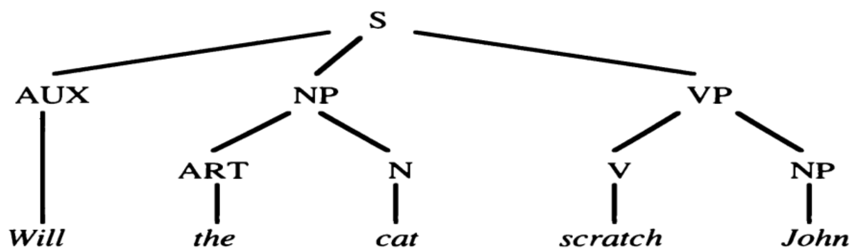
For **example,** the deep structure of "Will the cat scratch John?" would be:



The yes/no question is then generated from this deep structure by a transformation expressed schematically as follows:



With this transformation the surface form will be



BOX 5.2 Different Types of Movement

Wh-movement - move a wh-term to the front of the sentence to form a wh-question

topicalization - move a constituent to the beginning of the sentence for emphasis, as in

I never liked this picture.

This picture, I never liked.

adverb preposing - move an adverb to the beginning of the sentence, as in

I will see you tomorrow.

Tomorrow, I will see you.

extraposition - move certain NP complements to the sentence final position, as in

A book discussing evolution was written.

A book was written discussing evolution.

As you consider **strategies to handle movement**, remember that constituents cannot be moved from any **arbitrary position** to the front to make a question.

For **example,**

The man who was holding the two balloons will put the box in the corner.

is a **well-formed sentence**, but you **cannot ask the following question,** where the **gap** is indicated by a dash:

*What will the man who was holding - put the box in the corner?

[3] Handling questions in Context Free Grammars:

Extend grammar by adding the rule to handle yes/no questions.

S [+inv] -> (AUXAGR ?a SUBCAT ?v) (NP AGR ?a) (VP VFORM ?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

- GAP feature is used to handle wh-questions.
- this feature is passed from mother to sub constituent until appropriate place for gap is found in the sentence.
- at that place, an appropriate constituent can be constructed using no input.

[4]Hold Mechanisms in ATNs:

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.

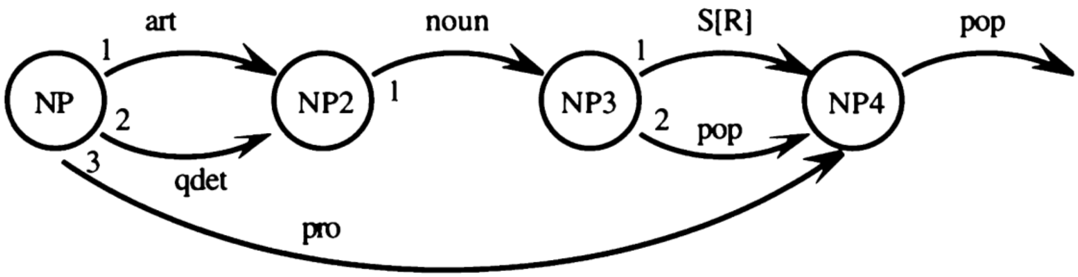
The hold action can store a constituent currently in a register (for example, the action HOLD SUBJ holds the constituent that is in the SUBJ register).

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.

The **held constituent** must have been used to **fill a gap** in the **current constituent** or in one of its **sub constituents**.

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 category is **present on the hold list.**

If the **arc** is 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.

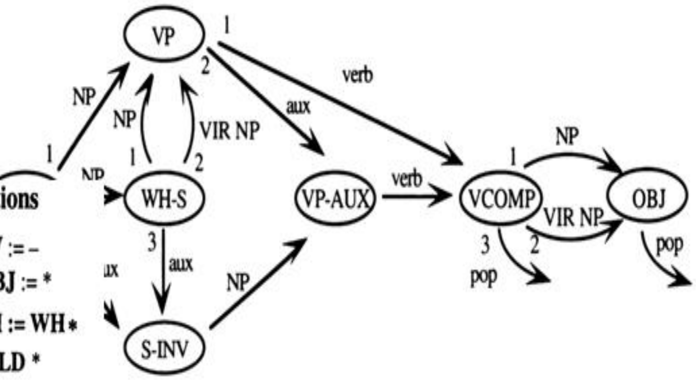


Arc	Test	Actions
NP/1	----------	DET := * AGR := AGR_*
NP/2	----------	DET := * WH := Q AGR := AGR_*
NP/3	----------	PRO := * WH := WH_* AGR := AGR_*
NP2/1	$AGR \cap AGR_*$	HEAD := * AGR := $AGR \cap AGR_*$
NP3/1	$WH_* \cap R$	MOD := *

Grammar 5.13 An NP network including wh-words

- **data structure** ‘holdlist’ maintains the **constituents** that are to be **moved**
- more than one constituent may be in **holdlist at a single time.**
- **constituents** are added to holdlist by **new action on arcs, hold action, also adds to register**
- to ensure that the held constituent is always used to **fill the gap**, ATN system does not allow a **pop arc to succeed** from a network until any constituent held by an action on an arc has been used
- to detect and fill gaps, **VIR (virtual)** arc is used.

Handling Questions in CFGs



Arc	Test	Actions
S/1		INV := – SUBJ := *
S/2	$WH_* \cap \{Q\ R\}$	WH := WH* HOLD *
S/3	$VFORM_* \cap \{past\ pres\}$	INV := + AUX := *
WH-S/1		SUBJ := *
WH-S/2		SUBJ := *
WH-S/3	$VFORM_* \cap \{past\ pres\}$	AUX := *
VP/1	$AGR_{SUBJ} \cap AGR_*$ $VFORM_* \cap \{past\ pres\}$	MAIN-V := *
VP/2	$AGR_{SUBJ} \cap AGR_*$ $VFORM_* \cap \{past\ pres\}$	AUX := *
S-INV/1	$AGR_{AUX} \cap AGR_*$	SUBJ := *
VP-AUX/1	$SUBCAT_{AUX} \cap FORM_*$	MAIN-V := *
VCOMP/1	$SUBCAT_{MAIN\text{-}V} \cap \_np$	OBJ := *
VCOMP/2	$SUBCAT_{MAIN\text{-}V} \cap \_np$	OBJ := *
VCOMP/3	$SUBCAT_{MAIN\text{-}V} \cap \_none$	

Grammar 5.14 An S network for questions and relative clauses

A trace of an ATN parse for $_{1}$ *The* $_{2}$ *man* $_{3}$ *who* $_{4}$ *we* $_{5}$ *saw* $_{6}$ *cried* $_{7}$

Trace of First NP Call: Arc S/1

Step	Node	Position	Arc Followed	Registers
2.	NP	1	NP/1	**DET** ← the **AGR** ← {3s 3p}
3.	NP2	2	NP2/1	**HEAD** ← man **AGR** ← 3s
4.	NP3	3	NP3/1 (for recursive call see trace below)	**MOD** ← (S **WH** R **SUBJ** we **MAIN-V** saw **OBJ** who)
10.	NP4	6	NP4/1 pop	returns (NP **DET** the **HEAD** man **AGR** 3s **MOD** (S who we saw))

Trace of S Network

Step	Node	Position	Arc Followed	Registers
1.	S	1	S/1 (for recursive call see trace below)	**SUBJ** ← (NP **DET** the **HEAD** man **AGR** 3s **MOD** (S who we saw))
11.	VP	6	VP/1	**MAIN-V** ← cried
12.	VCOMP	7	VCOMP/3 succeeds since no words left	returns (S **SUBJ** (NP **DET** the **HEAD** man **AGR** 3p **MOD** (S who we saw)) **MAIN-V** cried)

Trace of Recursive Call to S on Arc NP3/1

Step	Node	Position	Arc Followed	Registers
5.	S	3	S/2 (call to NP network not shown)	**WH** ← {Q R} **HOLDING** (NP **PRO** who **WH** {Q R})
6.	WH-S	4	WH-S/1	**WH** ← R **SUBJ** ← (NP **PRO** we ...)
7.	VP	5	VP/1	**MAIN-V** ← saw
8.	VCOMP	6	VCOMP/2 (uses the NP on the hold list)	**OBJ** ← (NP **PRO** who ...)
9.	OBJ	6	OBJ/1 pop	returns (S **WH** R **SUBJ** we **MAIN-V** saw **OBJ** who)

[5] **Gap Threading:**

- combines gap feature approach and hold list approach.
- often used in **logic grammars** where **two extra argument positions** are added to each predicate
    - one argument for **list of fillers** that might be used in the **current constituent**
    - one for the resulting list of fillers that were **not used after the constituent is parsed.**

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

- true only if there is a **legal S constituent** between **position-in** and **position-out** of the **input**
- if a **gap used, filler** will be in **fillers-in** but **not in fillers-out**
- **S** constituent with **NP gap** corresponds to

s(In, Out, [NP], nil)

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

Step	State	Next Operation
1.	s(1, 7, nil, nil)	applying rule 1
2.	np(1, In1,nil, Fillers1) vp(In1, 7, Fillers1, nil)	applying rule 4
3.	art(1, In2) cnp(In2, In1)	proved art(1,2)
4.	cnp(2, In1)	applying rule 6
5.	n(2, In3) np-comp(In3, In1)	proved n(2,3)
6.	np-comp(3, In1)	applying rule 8
7.	rel-intro(3, In4, Filler) s(In4, In1, Filler, nil)	applying rule 9
8.	relpro(3, In4)	proved relpro(3,4) proved rel-intro(3,4,[NP])
9.	s(4, In3, [NP], nil)	applying rule 1
10.	np(4, In5, [NP], Fillers1) vp(In5, In1, Fillers1, nil)	
		applying rule 5
11.	pro(4, In5)	proved pro(4,5) proved np(4, 5, [NP], [NP])
12.	vp(5, In1,[NP], nil)	applying rule 4
13.	v(5, In6) np(In6, In1, [NP], nil)	proved v(5, 6)
14.	np(6, In1, [NP], nil)	proved np(6, 6, [NP], nil) proved vp(5, 6, [NP], nil) proved s(4, 6, [NP], nil) proved np-comp(3, 6) proved cnp(2, 6) proved np(1, 6, nil, nil)
15.	vp(6, 7, nil, nil)	applying rule 2
16.	v(6, 7)	proved v(6, 7) proved vp(6, 7, nil, nil) proved s(1, 7, nil, nil)

Figure 5.17 A trace of the parse of ${}_1$ *The* ${}_2$ *man* ${}_3$ *who* ${}_4$ *we* ${}_5$ *saw* ${}_6$ *cried* ${}_7$

Toward Efficient Parsing:

- **human parsing** closer to **deterministic process**
- **2 different issues**
    - improve efficiency by reducing search but not final outcome

  - techniques for choosing between different interpretations.
- people does not give equal weight to all possible syntactic interpretations.
- garden-path sentences
  - eg. The raft floated down the river sank
  - by reading sank realizes that interpretations constructed so far is not correct
- approaches to predict when garden paths will arise - Minimal attachment principle and Right Association
- Minimal attachment principle - preference for the **syntactic analysis** which **creates least number of nodes in the parse tree**

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

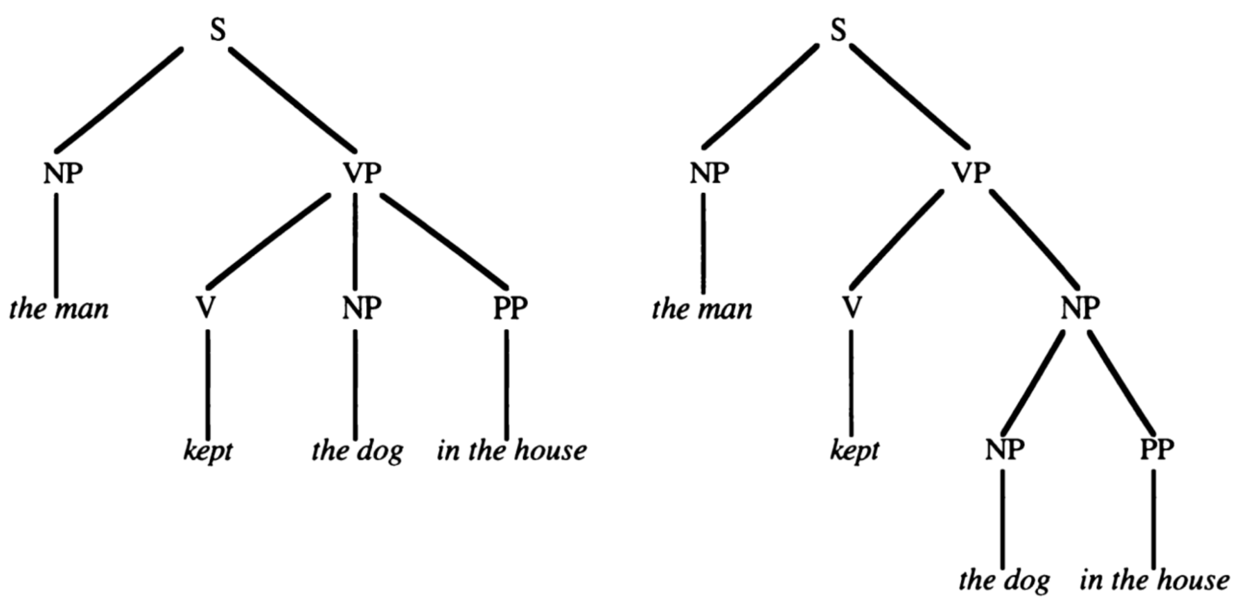


Figure 6.2 The interpretation on the left is preferred by the minimal attachment principle

- Right association/ late closure - new constituents tend to be interpreted as part of the current constituent under construction rather than part of some constituent higher in the parse tree

George said that Henry left in his car.

Preferred **interpretation** is that **Henry left** in the car rather than **George spoke**

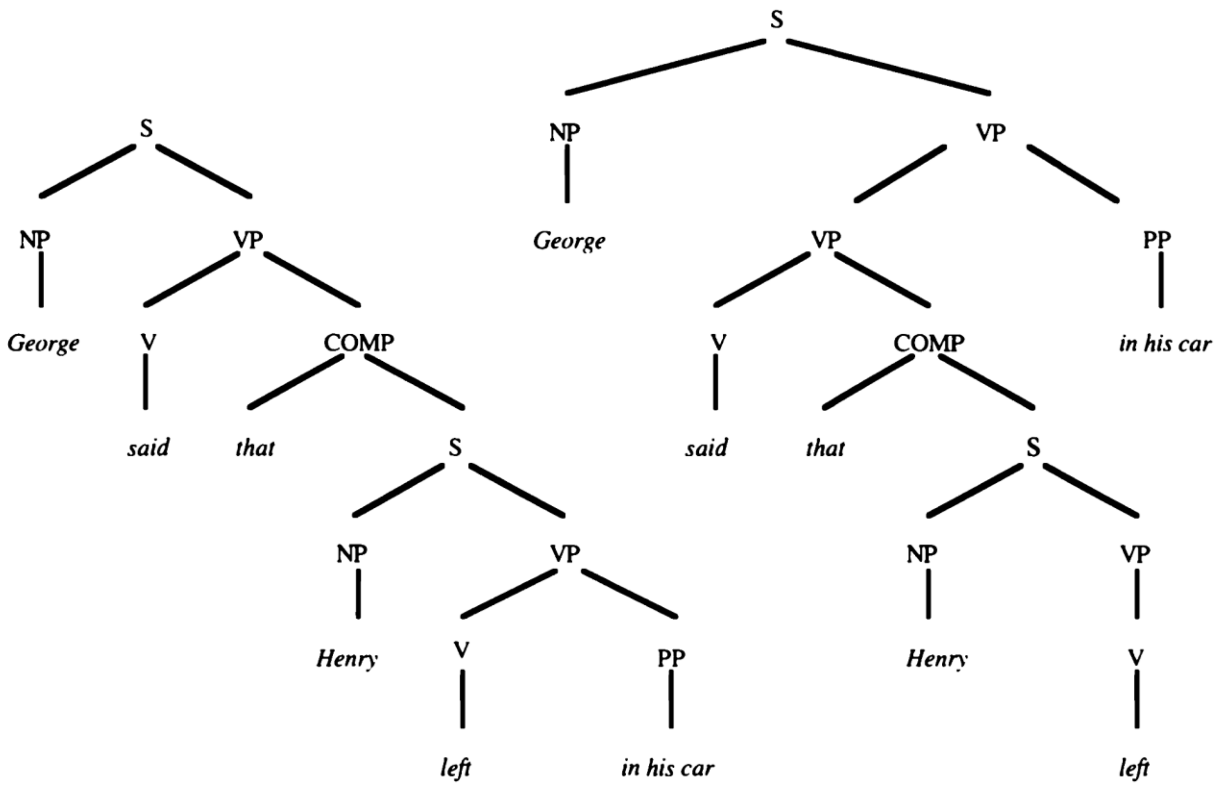


Figure 6.3 Two interpretations of *George said that Henry left in his car.*

[ ]Encoding Uncertainty: Shift-Reduce Parsers:

One way to **improve** the efficiency of parsers is to **use techniques** that encode uncertainty, so that the parser need not make an arbitrary choice and **later backtrack.**

Specifying the Parser State:

Consider using this approach on the small grammar in Grammar 6.4.

2.1	S → NP VP	2.3	VP → AUX V NP
2.2	NP → ART N	2.4	VP → V NP

Grammar 6.4 A simple grammar with an AUX/V ambiguity

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

as well as all the rules for NP, which in Grammar 6.4 is only

NP -> o ART N

Thus the **initial state, S0,** could be summarized as follows:

Initial State S0: S -> o NP VP
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**, the resulting state is

State S1: NP -> ART o N

If you **pick the symbol NP**, the rule is
S -> NP o VP
in the **new state.** Now if you **expand** out the **VP** to find all its possible **starting symbols**, you get the following:
State S2: S -> NP o VP
VP -> o AUX V NP
VP -> o V NP
Now, **expanding SI,** if you have the **input N,** you get a state **consisting of a completed rule:**

State S1': NP -> ART N o

Expanding 52, a V would result in the state

State S3: VP -> V o NP
NP -> o ART N

An **AUX from S2** would result in the state

State S4: VP -> AUX o V NP

and a **VP from 52** would result in the state

State S2': S -> NP VP o

Continuing from state 53 with an ART, you find yourself in state **Si again**, as you would also if you expand from **S0 with an ART**. Continuing from S3 with an NP, on the other hand, yields the new state.

State S3': VP -> V NP o

Continuing from S4 with a V yields

State S5: VP -> AUX V o NP
NP -> o ART N

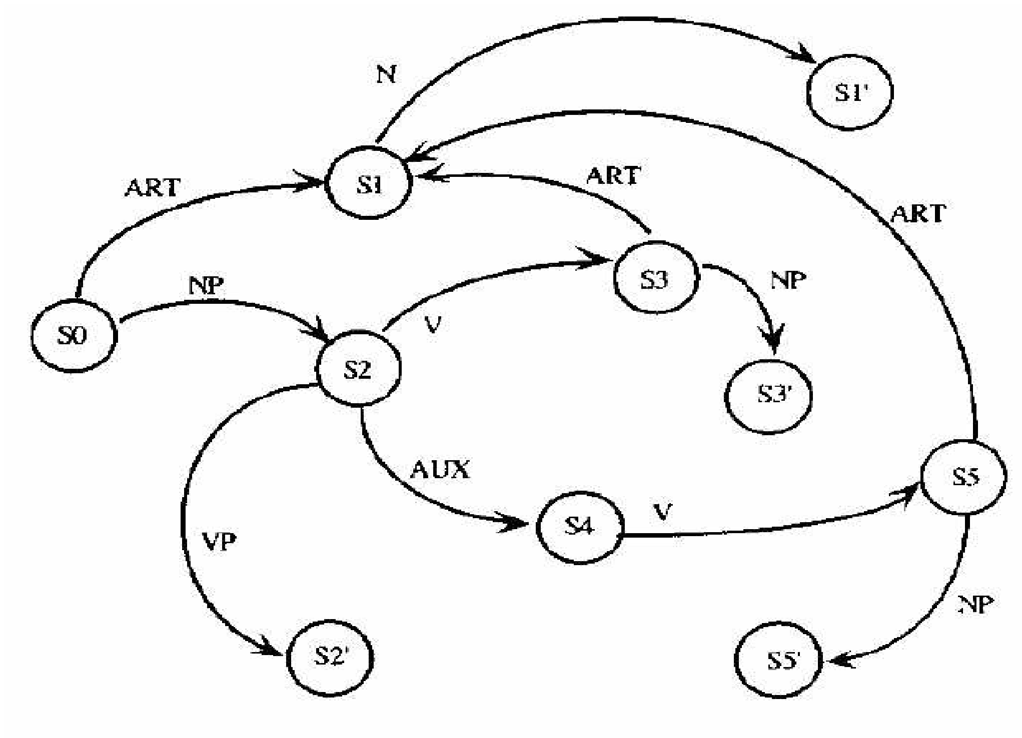


Figure 6.5 A transition graph derived from Grammar 6.1

and continuing from **S5 with an ART** would produce **state S1 again**. Finally, continuing from **S5** with an **NP** would produce the state.
State S5': VP -> AUX V NP o

A Shift-Reduce Parser:

- Shift reduce parsers are for unambiguous CFGs
    - maintains two stacks
        - **input stack - input symbols** and some **grammar symbols**

- **Parse stack - parse states** and **grammar symbols**

Shift Reduce Parser:

2.1 S → NP VP

2.2 NP → ART N

2.3 VP → AUX V NP

2.4 VP → V NP

A simple grammar with an AUX/V ambiguity

[7] A Deterministic Parser:

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.**

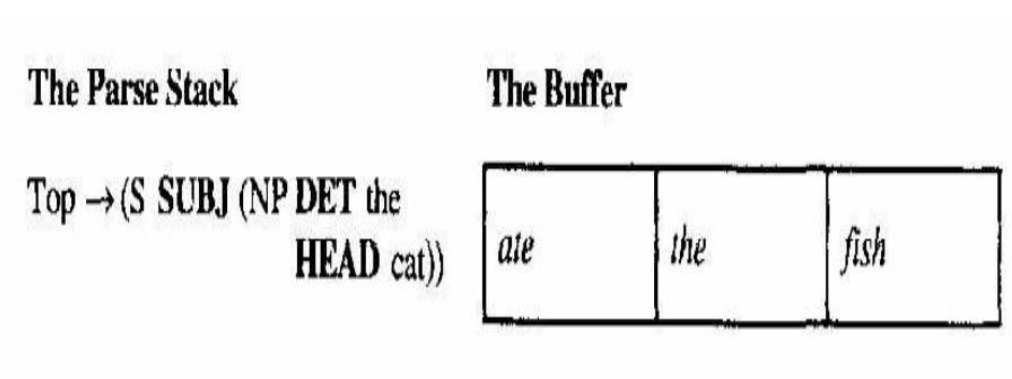


Figure 6.8 A situation during a parse

Rather than **shifting constituents** onto the **parse stack** to be later consumed by a **reduce action,** the parser builds constituents incrementally by attaching **buffer elements** into their **parent constituent**, 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:**

a)**Create a new node on the parse stack** (to push the symbol onto the stack). NP->S

b) **Attach an input** constituent to the top node on the parse stack.

c) **Drop the top** node in the parse stack into the **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.

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 and **assigned to the SUBJ** feature of the **S constituent** on the **parse stack.**

The operation **Attach to MAIN-V**

would remove the lexical entry for ate from the buffer and assign it to the **MAIN-V** feature in the **S** on the parse stack. **Next the operation.**

Create NP

would push an empty NP constituent onto the parse stack, creating the situation in Figure 6.9.

Next the two operations

Attach to DET

Attach to HEAD

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

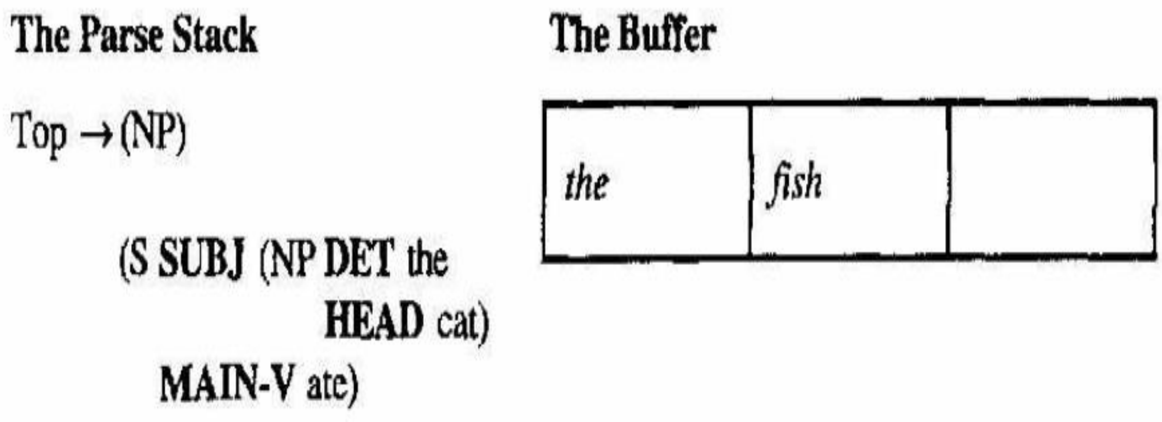


Figure 6.9 After creating an NP

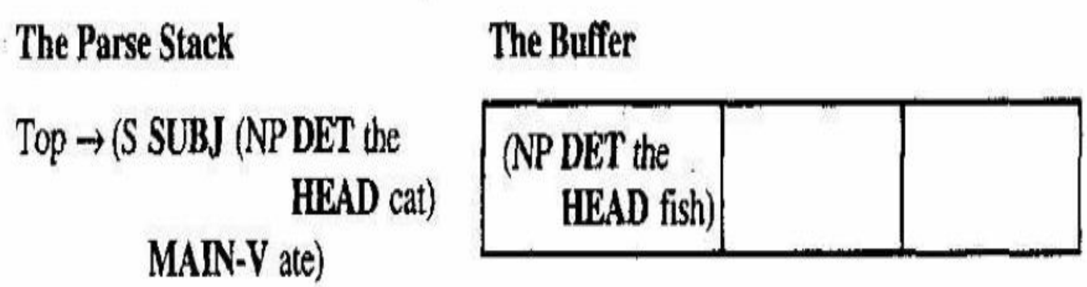


Figure 6.10 After the drop action

The operation

Drop

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

The parser is now in a situation to build the final structure with the operation

Attach to OBJ

which takes the **NP** from the **buffer** and assigns it to the **OBJ slot** in the S constituent.

Three other operations prove very useful in **capturing** generalizations in **natural languages:**

Switch the nodes in the **first two** buffer positions.

Insert a specific lexical item into a specified buffer slot.

Insert an empty NP into the first buffer slot

Additional **actions are available** for **changing the parser** state by selecting **which packets** to use. In particular, there are actions to

Activate a packet (that is, all its rules are to be used to interpret the next input)**.**

Deactivate a packet.

Pattern	Actions	Priority
Packet BUILD-AUX:		
1. <=AUX, HAVE> <=V, pastprt>	Attach to **PERF**	10
2. <=AUX, BE> <=V, ing>	Attach to **PROG**	10
3. <=AUX, BE> <=V, pastprt>	Attach to **PASSIVE**	10
4. <=AUX, +modal> <=V, inf>	Attach to **MODAL**	10
5. <=AUX, DO> <=V, inf>	Attach to **DO**	10
6. <true>	Drop	15

Grammar 6.11 The rules for packet BUILD-AUX

Consider the **example rules** shown in Grammar 6.11, which deals with **parsing the auxiliary structure.**

The pattern for each **rule indicates** the feature tests that must succeed on each **buffer position** for the **rule to be applicable.**

Thus the pattern **<=AUX, HAVE>** <=V, **pastprt>** is true only if the **first buffer** is an AUX structure' with ROOT feature value HAVE, and the **second is a V structure** with the VFORM feature **pastprt.** The **priority associated** with each rule is used to decide between **conflicting rules.** The **lower the number,** the **higher the priority.**

In particular, **rule 6,** with the pattern , will always match, but since its **priority is low,** it will never be used if another **one of the rules also matches.**

It simply covers the case when none of the rules match, and it completes the parsing of the **auxiliary and verb structure.**

The Parse Stack

Nodes	Active Packets
Top → (AUXS)	(BUILD-AUX)
(S MOOD DECL SUBJ (NP NAME John))	(PARSE-AUX CPOOL)

The Input Buffer

(AUX ROOT HAVE FORM pres NUM {3s})	(V ROOT SEE FORM en)	(ART ROOT A NUM {3s})

Figure 6.12 A typical state of the parser

Figure **6.12 shows** a parse state in which the state **BUILD-AUX is active**.

It contains an **AUXS structure** on the **top of the stack** with **packet BULLD AUX active,** and an **S structure above** with packets **PARSE-AUX and CPOOL** that will become active once the **AUXS constituent** is dropped into the buffer.

Given this situation and the rules in Figure 6.11, the parser's next action is determined by seeing which rules match. Rules 1 and 6 succeed, and I is

chosen because of its higher priority. Applying the actions of rule 1 produces the state in Figure 6.13. Now the rules in BUILD-AUX are applied again.

This time only rule 6 succeeds, so the next action is a drop, creating the state in Figure 6.14.

At this stage the rules in packets PARSE-AUX and CPOOL are active; they compete to determine the next move of the parser (which would he to attach

the AUXS structure into the S structure).

The Parse Stack

Nodes	**Active Packets**
Top → (AUXS **PERF** has)	(BUILD-AUX)
(S **MOOD** DECL (S **SUBJ** (NP **NAME** John))	(PARSE-AUX CPOOL)

The Input Buffer

(V **ROOT** SEE **VFORM** pastprt)	(ART **ROOT** A **NUM** {3s})	(N **ROOT** DAY **NUM** {3s})

Figure 6.13 After rule 1 is applied

The Parse Stack

Nodes	**Active Packets**
Top → (S **MOOD** DECL **SUBJ** (NP **NAME** John))	(PARSE-AUX CPOOL)

The Input Buffer

(AUXS **PERF** has)	(V **ROOT** SEE **VFORM** pastprt)	(ART **ROOT** A **NUM** {3s})

Figure 6.14 The parse state after a drop action