Unit 3

Grammars and Parsing:

7.1 Natural Language:

→ Augmented context-free grammars provide a powerful formalism for capturing many generalities in natural language.

→ Auxiliary verbs and introduces features that capture the ordering and agreement constraints.

→ The general class of problems often characterized as movement phenomena

Movement Phenomena 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 position in a basic sentence.

Eg:- consider yes/no questions and how they relate to their assertional counterpart.

Eg:- Jack is giving Sue a back rub.
Is Jack giving Sue a back rub?

He will run in the marathon next year.
Will he run in the marathon next year?

→ 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:

Eg:-
1. John went to the store.
Did John go to the store?
2. Henry goes to school every day.
Does Henry go to school every day?

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

Local (or Bounded) Movement :-
Deriving yes/no questions from assertions by moving the constituents in the manner. The movement is considered local because the reordering of the constituents is specified precisely within the scope of a limited number of rules.

Unbounded Movement :-
→ Which occurs in wh-questions. In case of unbounded movement, constituents may be moved arbitrarily far from their original position.

Eg:
Consider the wh-questions that are related to the assertion

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

Questions :
→ Which man will angrily put the book in the corner?
→ Who will angrily put the book in the corner?
→ How will the fat man put the book in the corner?
→ In what way will the fat man put the book in the corner?
→ 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?
→ Each question has the same form as the original assertion, except that the part being questioned is removed and replaced by a wh-phrase at the beginning of the sentence.
→ 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:
Eg: → I found a bookcase
→ Did I find a bookcase?
→ What did I find?

→ Thus we may be able to reuse much of a grammar for yes/no questions for wh-questions.

→ A serious problem remains, however, concerning how to handle the fact that a constituent is missing from someplace later in the sentence.

Eg: VP in the sentence

What will the fat man angrily put in the corner?

↑ acceptable sentence

What will the fat man angrily ↑ put in the corner?

acceptable

Not acceptable

*I angrily put in the corner.

*Where will the fat man angrily put in the corner?

Gap :

If we constructed a special grammar for VPs in wh-questions, you would need a seperate grammar for each form of VP and each form of missing constituent. This would create a significant expansion in the size of the grammars

→ the place where a subconstituent is missing is called the gap.

Filler :

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.

Eg:- VP sentence

What will the fat man angrily put in the corner?

Parsed as

angrily put what in the corner.

What will the fat man angrily put the book in?

Parsed as

angrily put the book in what.

→ For the correctness of analysis all the tests like subject-verb agreement, the case of

pronouns (who vs whom) and verb transitivity operate as though the wh-term were actually filling the gap.

Eg: Transitive verb put

What did you put in the cupboard?

put is a transitive verb requires an object. The object is a gap filled by the wh-term, satisfying the transitivity constraint.

→ A sentence where the object is explicitly filled is unacceptable.

* What did you put the bottle in the cupboard?

* You put what the bottle in the cupboard?

→ Two objects not acceptable.

→ Accepts only for constraint on the standard object NP position.

Transformational Grammar

→ Many linguistic theories have been developed that are based on the intuition that a constituent can be moved from one location to another.

→ Significant generalizations can be made that greatly simplify the construction of a grammar.

→ A context-free grammar generated a base sentence, then a set of transformations converted the resulting syntactic tree into a different tree a set of transformations converted the resulting syntactic tree into a different tree by moving constituents.

→ Augmented transition networks offered a new formalism that captured much of the behavior in a more computationally effective manner.

Hold list:

→ A new structure called the hold list was introduced that allowed a constituent to be saved and used later in the parse by a new arc called the virtual (VIR) arc, this was the predominant computational mechanism for quite some time.

→ New techniques were developed in linguistics that are strongly influenced current computational systems.

pronouns (who vs whom) and verb transitivity operate as though the wh-term were actually filling the gap.

Eg: Transitive verb put

What did you put in the cupboard?

put is a transitive verb requires an object. The object is a gap filled by the wh-term, satisfying the transitivity constraint.

→ A sentence where the object is explicitly filled is unacceptable.

* What did you put the bottle in the cupboard?

* You put what the bottle in the cupboard?

→ Two objects not acceptable.

→ Accepts only for constraint on the standard object NP position.

Transformational Grammar

→ Many linguistic theories have been developed that are based on the intuition that a constituent can be moved from one location to another.

→ significant generalizations can be made that greatly simplify the construction of a grammar

→ A context-free grammar generated a base sentence, then a set of transformations converted the resulting syntactic tree into a different tree a set of transformations converted the resulting syntactic tree into a different tree by moving constituents.

→ Augmented transition networks offered a new formalism that captured much of the behavior in a more computationally effective manner.

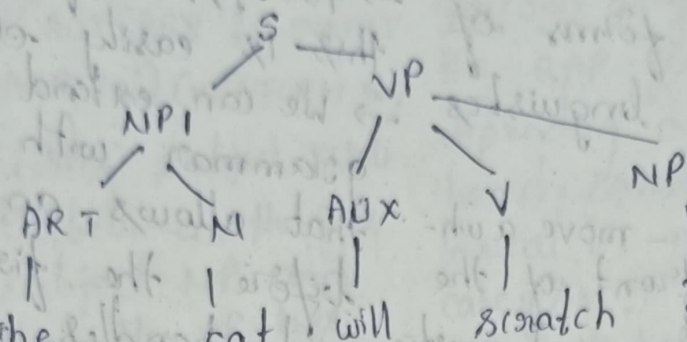
Hold list:

→ A new structure called the hold list was introduced that allowed a constituent to be saved and used later in the parse by a new arc called the virtual (VIR) arc. This was the predominant computational mechanism for quite some time.

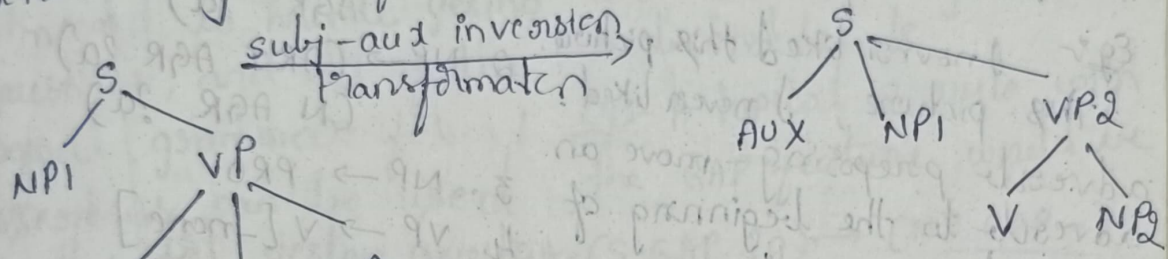
→ New techniques were developed in linguistics that are strongly influenced current computational systems

→ The first technique was the introduction of slash categories, which are the complex nonterminals of the form X/Y and stand for a constituent of type X with a subconstituent of type Y missing.
→ Given a context-free grammar, there is a simple algorithm that can derive new rules for such complex constituents.
→ Grammar writers have a convenient notation for expressing the constraints arising from unbounded movement.
→ The size of the resulting grammar can be significantly larger than the original grammar.
→ A better approach uses the feature system.
→ constituents are stored in a special feature called GAP and are passed from constituent to subconstituent to allow movement.
→ In this analysis the constituent S/NP is shorthand for an S constituent with the GAP feature NP.
→ To handle movement, constituents cannot be moved from any arbitrary position to the front to make a question.
General constraint on movement
Eg:- Well formed sentence
out of relative clause
The man who was holding the two balloons will put the box in the corner.
Gap indicated by dash.
we cannot ask question
* what will the man who was holding - put the box in the corner?
→ The term movement arose in transformational grammar (TG).
→ TG posited 2 distinct levels of structural representation:
1. 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.

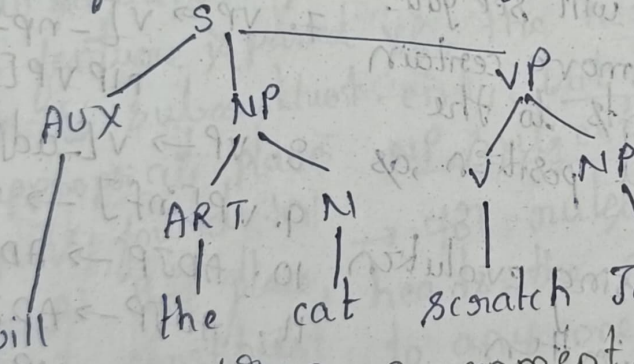
Eg- Deep structure of "will the cat scratch John?



→ The yes/no question is then generated from the deep structure by a transformation expressed schematically as follows ...



With this transformation the surface form will be.



→ In early TG, transforma -tions could do many opera -tions on trees, including moving, adding and deleting constituents.

→ Besides subj-aux inversion, there were transformational accounts of passives, wh-ques -tions and embedded sen -tences.

Government-Binding (GB) theory is the modern descendants of TG do not use transformations in the same way. A single transformation rule, Move-α allows any constituent to be moved anywhere!

→ The focus is on developing constraints on movement that prohibit the generation of illegal surface structures.

Different Types of Movement in

Most common forms of movement in linguistics literature

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

Eg: 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.

Handling Questions in Context-free Grammars

→ 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 is easily done.
→ We can extend the following grammar with one rule that allows an auxiliary before the first NP and handles most examples

1. S[-inv] → [NP AGR ?a)
   (VP [?t 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

A simple grammar in abbreviated form

from above grammar

S[+inv] → (AUX AGR ?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 that is needed to handle yes/no questions, and all of the original grammar for assertions can be used directly for yes/no questions.

→ A special feature GAP is introduced to handle wh-questions. This feature is passed from mother to subconstituent until the appropriate place for the gap is found in the sentence. At that place, the appropriate constituent can be constructed using no input. This can be done by introducing additional rules with empty right-hand sides.

Eg:

(NP GAP ((CAT NP) (AGR ?a)) AGR ?a) → ε

NP from no input if the NP sought has a GAP feature that is set to an NP.

→ There seem to be two general ways in which the GAP feature propogates depending on whether the head constituent is a lexical or non lexical category.

Non lexical category - the GAP feature is passed from the mother to the head and not to any other subconstituents.

Eg: A typical S rule with the GAP feature would be

(S GAP ?g) → (NP GAP -) (VP GAP ?g).

→ The GAP can be in the VP, the head subconstituent, but not in the NP subject.

→ For rules with lexical heads, the gap may move to any one of the nonlexical subconstituents.

Eg:

A typical S rule with the GAP feature would be

The rule for verbs with an np-vp:inf comp-ement,

VP → V [-np-vp:inf] NP PP would result in 2 rules involving gaps

(VP GAP ?g) → V [-np-vp:inf] (NP GAP ?g) (PP GAP -)

(VP GAP ?g) → V [-np-vp:inf] (NP GAP -) (PP GAP ?g)

→ The GAP may be in the NP or in the PP, but not both. Setting the GAP feature in all but one subconstituent to guarantees that a gap can be used in only one place.

→ An algorithm for automatically adding GAP features to a grammar is shown below.

Note:-

→ It does not modify any rule that explicitly sets the GAP feature already, allowing grammar designer to introduce rules that do not follow the conventions encoded in the algorithm.

→ The rule for subject-aux inversion cannot allow the gap to propogate to the subject NP.

An algorithm for adding GAP features to a grammar

for each rule $Y \rightarrow X_1 \cdots H_i \cdots X_n$ 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 GAP ?g) → (X1 GAP -) ... (Hi GAP ?g) ... (Xn GAP -)

3. If the head H_i is a lexical category, then for each non lexical subconstituent X_j, add a rule of the form:

(Y GAP ?g) → (X1 GAP -) ... (Xj GAP ?g) ... (Xn GAP -)

→ A new grammar can be created that handles gaps. All that is left to do is analyze where the fillers for the gaps come from.

→ In wh-questions, the fillers are typically NPs 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 (how many & how carefully).

Simple NPs:

Pronouns - who, whom and what

who ate the pizza?

What did you put the book in?

Determiners in noun phrases what and which

What book did he steal?

Prepositional phrases where and when

Where did you put the book?

The word how acts as an adverbial modifier to adjective and adverbial phrases:

How quickly did he run?

The word whose acts as a possessive pronoun:

Whose book did you find?

→ The wh-words also can act in different roles.
→ Wh-questions which will be indicated by the WH feature value Q. A subset of them can also be used to introduce relative clauses.
→ These will also have the WH feature value R to make the WH feature act appropriately, a grammar should satisfy the following constraint:

If a phrase contains a subphrase that has the WH feature, then the larger phrase also has the same WH feature.

→ Complex phrases containing a sub constituent with a WH word also can be used in questions.

Eg:-

In what store did you buy the picture?

PP in what store has WH value Q because NP what store has the WH value Q (because the determiner what has the WH value Q).

→ With this constraint, the final S constituent will also have the WH value Q, which will be used to indicate the sentence is a WH-question.

A lexicon for some of the WH-words

what: (CAT PRO
WH Q
AGR {3S, 3P})

what: (CAT QDET
WH Q
AGR {3S, 3P})

which: (CAT QDET
WH Q
AGR {3S, 3P})

which: (CAT PRO
WH R
AGR {3S 3P})

when: (CAT PP-WRD
WH {Q R}
PFORM TIME)

who: (CAT PRO
WH {Q R}
AGR {3S 3P})

where: (CAT PP-WRD
WH {Q R}
PFORM {LOC MOT})

whose: (CAT PRO
WH {Q R}
POSS +
AGR {3S 3P})

A simple NP and PP grammar handling wh-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 (common noun phrase)
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

→ New lexical categories:
QDET for determiners that introduce wh-terms
PP_WRD for words like when that act like prepositional phrases.

le NP and PP
ar handling
ds

POSS ?P WH ?w)
RO POSS ?P
H ?w)
WH ?w) →
H ?w AGR ?a)
AGR ?a)
mar Noun Phrase
ADJ N
ART
?w) →
SS] WH ?a)
H ?w) →
WH ?w)
?w) →
P WH ?w)
?w) →
WRD WH ?w)
for NP, DET
AGR
for PP: PFORM
categories
determiners
uce wh-terms
for words like
act like
l phrases.

The unexpanded S grammar for wh-questions

10. (S[-inv] WH ?w) →
(NP WH ?w AGR ?a)
(VP[fin] AGR ?a))

11. (S[+inv] WH ?w GAP ?g) →
(AUX COMPFORM ?s AGR ?a)
(NP WH ?w AGR ?a GAP -)
(VP VFORM ?s GAP ?g))

12. S → (NP[Q, -gap] AGR ?a)
(S[+inv] GAP (NP AGR ?a))
Subject Aux Inversion

13. S → (PP[Q, -gap] PFORM ?P)
(S[+inv] GAP (PP PFORM ?P))

14. VP → (AUX COMPFORM ?s)
(VP VFORM ?s)

15. VP → V[-none]

16. VP → V[-np] NP

17. VP → V[-vp:inf] VP[inf]

18. VP → V[-np-vp:inf]
NP VP[inf]

19. VP[inf] → TO VP[base]

The S grammar for wh-questions with the GAP feature

10. (S[-inv] WH ?w GAP ?g)
→ (NP WH ?w AGR ?a)
(VP[fin] AGR ?a GAP ?g)

11. (S[+inv] WH ?w GAP ?g)
→ (AUX COMPFORM ?s AGR ?a)
(NP WH ?w AGR ?a GAP -)
(VP VFORM ?s GAP ?g)

12. S → (NP[Q, -gap] AGR ?a)
(S[+inv] GAP (NP AGR ?a))

13. S → (PP[Q, -gap] PFORM ?P)
(S[+inv] GAP (PP PFORM ?P))

14. (VP GAP ?g) →
(AUX COMPFORM ?s)
(VP VFORM ?s GAP ?g)

15. VP → V[-none]

16. (VP GAP ?g) → V[-np]
(NP GAP ?g)

17. (VP GAP ?g) →
V[-vp:inf] (VP[inf] GAP ?g)

18. (VP GAP ?g) →
V[-np-vp:inf] (NP GAP ?g)
(VP[inf] GAP -)

18'. (VP GAP ?g) →
V[-np-vp:inf] (NP GAP -)
(VP[inf] GAP ?g)

19. (VP[inf] GAP ?g) →
TO (VP[base] GAP ?g)

The unexpanded S grammar for wh-questions

10. (S[-inv] WH ?w) →
(NP WH ?w AGR ?a)
(VP[fin] AGR ?a)

11. (S[+inv] WH ?w GAP ?g) →
(AUX COMPFORM ?s AGR ?a)
(NP WH ?w AGR ?a GAP -)
(VP VFORM ?s GAP ?g)

12. S → (NP[Q, -gap] AGR ?a)
(S[+inv] GAP (NP AGR ?a))
Subject Aux Inversion

13. S → (PP[Q, -gap] PFORM ?p)
(S[+inv] GAP (PP PFORM ?p))

14. VP → (AUX COMPFORM ?s)
(VP VFORM ?s)

15. VP → V[-none]

16. VP → V[-np] NP

17. VP → V[-vp:inf] VP[inf]

18. VP → V[-np-vp:inf]
NP VP[inf]

19. VP[inf] → TO VP[base]

The S grammar for wh-questions with the GAP feature

10. (S[-inv] WH ?w GAP ?g)
→ (NP WH ?w AGR ?a)
(VP[fin] AGR ?a GAP ?g)

11. (S[+inv] WH ?w GAP ?g)
→ (AUX COMPFORM ?s AGR ?a)
(NP WH ?w AGR ?a GAP -)
(VP VFORM ?s GAP ?g)

12. S → (NP[Q, -gap] AGR ?a)
(S[+inv] GAP (NP AGR ?a))

13. S → (PP[Q, -gap] PFORM ?p)
(S[+inv] GAP (PP PFORM ?p))

14. (VP GAP ?g) →
(AUX COMPFORM ?s)
(VP VFORM ?s GAP ?g)

15. VP → V[-none]

16. (VP GAP ?g) → V[-np]
(NP GAP ?g)

17. (VP GAP ?g) →
V[-vp:inf] (VP[inf] GAP ?g)

18. (VP GAP ?g) →
V[-np-vp:inf] (NP GAP ?g)
(VP[inf] GAP -)

18'. (VP GAP ?g) →
V[-np-vp:inf] (NP GAP -)
(VP[inf] GAP ?g)

19. (VP[inf] GAP ?g) →
TO (VP[base] GAP ?g)

20. VP → V[-np-pp:loc]
NP PP[loc]

Head features for S, VP:
VFORM, AGR

Parsing with GAPS:
→ A grammar that allows gap creates some new complications for parsing algorithms.
→ Rules that have an empty right-hand side such as
(NP AGR ?a1 GAP (NP AGR ?a)) → ϵ
may cause problems because they allow an empty NP constituent anywhere.
→ In a bottom-up strategy, for instance, this rule could apply at any position (or many times at any position) to create NPs that use no input.
→ A top-down strategy fares better, in that the rule would only be used when a gap is explicitly predicted.
→ Instead of using such rules, however, the arc extension algorithm can be modified to handle gaps automatically. This technique works with any parsing strategy.

20. (VP GAP ?g) → V[-np-pp:loc]
(NP GAP ?g) (PP[loc] GAP -)

20! (VP GAP ?g) → V[-np-pp:loc]
(NP GAP -) (PP[loc] GAP ?g)

Head features for S, VP:
VFORM, AGR

→ Consider what extensions are necessary. When a constituent has a GAP feature that matches the constituent itself, it must be realized by the empty constituent. This means that an arc whose next constituent is a gap can be extended immediately.

Eg:
(VP GAP (NP AGR 3S))
→ V[-np-pp:loc]
(NP GAP (NP AGR 3S))
PP [LOC]

The next constituent needed is an NP, but it also has a GAP feature that must be an NP. Thus the constituent must be empty. The parser inserts the constituent
(NP AGR 3S EMPTY +)

to the chart, which can then extend the arc to produce the new arc

(VP GAP (NP AGR 3S))
→ V[-np-PP:loc] (NP GAP (NP AGR 3S)) . PP[LOC]

The algorithm to insert empty constituents as needed whenever an arc of the form

X → ... • (CF1 V1 ... Fn Vn GAP (CG1 ?vg1 ... Gm ?vgm)) is suggested by the parser, and the constituent pattern that is the GAP feature, that is,

(CG1 ?vg1 ... Gm ?vgm)

matches the constituent itself

(CF1 V1 ... Fn Vn GAP (CG1 VG1 ... Gm VGm))

then add a new constituent

(CG1 ?vg1 ... Gm ?vgm EMPTY +), with the variables bound as necessary to the chart. Use this constituent to extend the original arc.

→ consider parsing "which dogs did he see?" using the bottom-up strategy. only the rules that contribute to the final analysis will be considered

Rule 7 (DET WH ?w) → (QDET WH ?w) applies to the constituent QDET1 (which) to build constituent DET1 (common Noun Phrase)

Rule 3 CNP → N applies to the constituent N1 (dogs) to build a constituent CNP1, which then combined with DET1 by rule 2 to build an NP constituent of the form

NP1 : (NP AGR 3P -WH Q |QDET1| 2 CNP1)

→ This NP introduces an arc based on rule 12 in Grammar

S → (NP [Q, -gap] AGR ?a) (S [+inv] GAP (NP AGR ?a))

→ The next word, did is an AUX constituent, which introduces an arc based on rule 11.

11. (S[+inv] WH ?w GAP ?g) →
(AUX COMPFORM ?s AGR ?a)
(NP WH ?w AGR ?a GAP -)
(VP VFORM ?s GAP ?g)

→ the next word, he, is a PRO, which creates an NP, NP2, using rule 1, and extends the arc based on rule 1.

(NP POSS ?p WH ?w) →
(PRO POSS ?p WH ?w)
and extends the arc based on rule 1.

→ the word see introduces a verb V1, which can extend rule 16. This adds the arc labeled

(VP GAP ?g) → V[-np] •
(NP GAP ?g)

Since the GAP value can match the required NP (because it is unconstrained), an empty NP is inserted in the chart with the form:

EMPTY-NP1: (NP AGR ?a)
GAP (NP AGR ?a))
EMPTY +)

this constituent can then be used to extend the VP arc, producing the constituent

VP1: (VP VFORM inf
GAP (NP AGR ?a)
1 V1
2 EMPTY-NP1)

Now VP1 can extend the arc based on rule 11 to form the S structure:

S1: (S GAP (NP AGR ?a)
INV +
1 AUX-1
2 NP2
3 VP1

Finally, NP1 and S1 can combine using rule 12 to produce an S constituent

Rule 12: S → (NP [Q, -gap]
AGR ?a) (S [+inv] GAP
(NP AGR ?a))

→ WH and GAP features appropriately and by extending the parser to insert empty constituents as needed to fill gaps, the original grammar for declarative assertions can be extended to handle yes/no questions and wh-questions with only three new rules.

NP1 WH Q AGR 3P 1 DET1 2 CNP1			
DET1 WHQ AGR 3P 1 QDET1	CNP1 AGR 3P 1 N1		NP2 AGR 3S 1 PRO1
QDET1 WH Q AGR 3P	N1 AGR 3P	AUX1 AGR 3S VFORM past SUBCAT base	PRO1 AGR 3S
which	dogs	did	he

S → NP[Q] • (S GAP (NP AGR 3P))

(S GAP ?g) → • AUX NP • (VP GAP ?g)

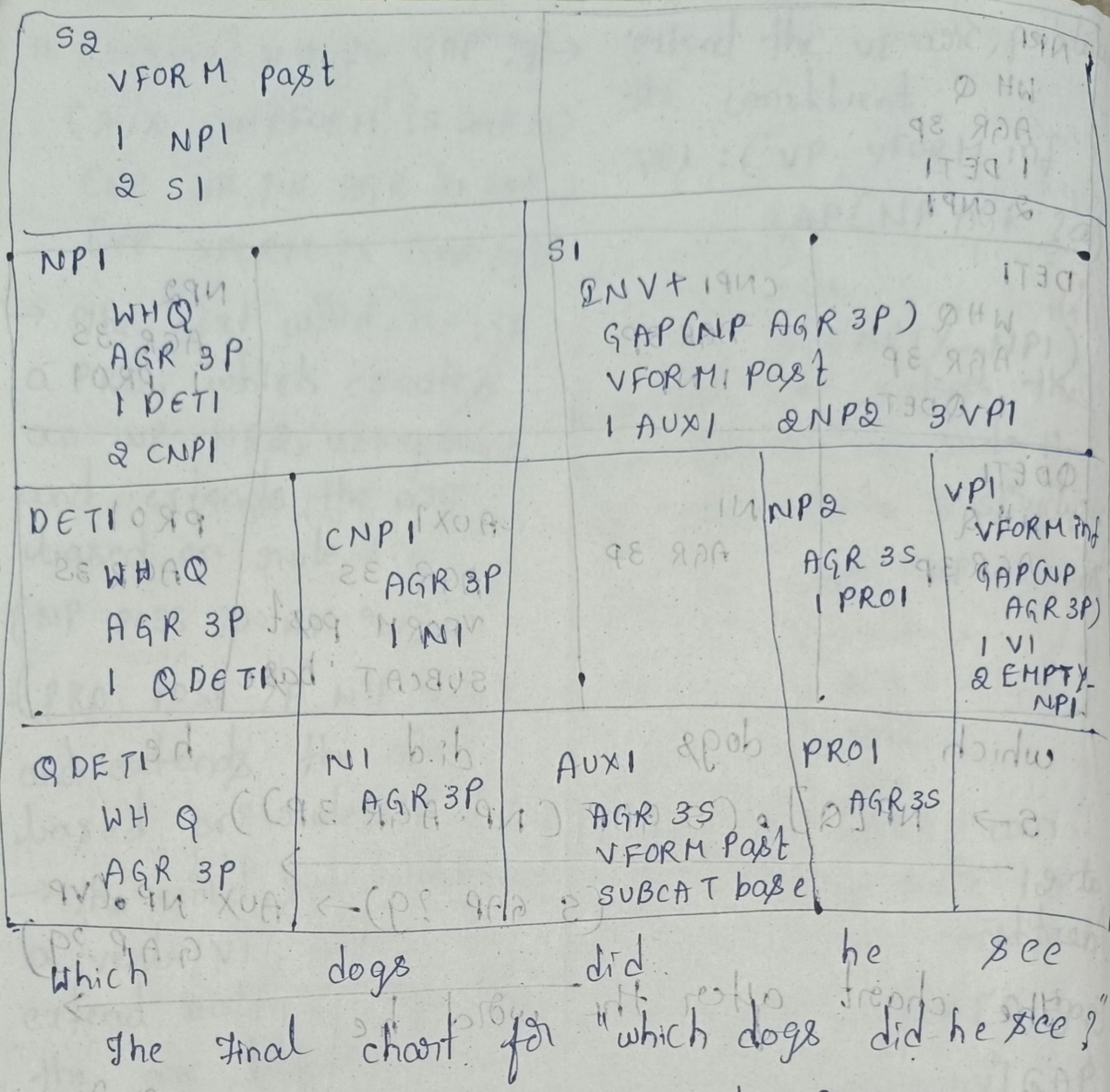
The chart after the word he

2. (NP WH ?w) → (DET WH ?w AGR ?a)
(CNP (AGR ?a)

3. CNP → N

11. (S [+inv] WH ?w GAP ?g) →
(AUX COMPFORM ?s AGR ?a)
(NP WH ?w AGR ?a GAP -)
(VP VFORM ?s GAP ?g)

which - pron
dogs - N, V
did - AUX
he - pron, Adj, N
see - V



The final chart for "which dogs did he see?"

which - Pronoun
dogs - N, V, ADJ
did - AUX V
he - Pro; Adj; N
see - V, N

12. S → (NP [Q, -gap] AGR ?a)
(S [+inv] GAP (NP AGR ?a))

2. (NP WH ?w) → (DET WH ?w AGR ?a) (CNP AGR ?a)

7. (DET WH ?w) → (Q DET WH ?w)

3. CNP → N

11. (S [+inv] WH ?w GAP ?g) →
(AUX COMPFORM ?s AGR ?a)
(NP WH ?w AGR ?a GAP -)
(VP VFORM ?s GAP ?g)

1. (NP POSS ?p WH ?w) → (PRO POSS ?p WH ?w)

16. (VP GAP ?g) → V[-np] (NP GAP ?g)

Rule 16
VP1 : (VP V FORM inf
GAP (NP AGR ?a)
| V |
& EMPTY-NP1)

The Movement constraints :-

→ In linguistics the principles that govern where gaps may occur are called island constraints. The term draws on the meta-phor of constituent movement. An island is a constituent from which no subconstituent can move out (just as a person cannot walk off an island).

1. The A over A constraint

No constituent of category A can be moved out of a constituent of type A. This means you cannot have an NP gap within an NP; a PP gap within a PP and so on and provides justification for not allowing non-finite null constituents of the form NP/NP, PP/PP and so on. This disallows sentences such as

* What book, did you meet the author of —1 ?

2. Complex-NP Constraint

No constituent may be moved out of a relative clause or noun complement. This constraint disallows sentences like

* To whom, did the man who gave the book —1 laughed ?

where the PP to whom would have been part of the relative clause who gave the book to whom (as in the man who gave the book to John laughed).

3. Sentential subject constraint

No constituent can be moved out of a constituent serving as the subject of a sentence. This overlaps with the other constraints when the subject is an NP, but non-NP subjects are possible as well, as in the sentence "for me to learn these constraints is impossible". This constraint eliminates the possibility of a question like

* what, is for me to learn —1 impossible

4. Wh-Island constraint

No constituent can be moved from an embedded sentence with a wh-complementizer.

Eg:- Did they wonder whether I took the book ?

We cannot ask

* What, did they wonder whether I took —, ?

5. Coordinate structure constraint

A constituent cannot be moved out of a coordinate structure.

Eg:- Did you see John and Sam

We cannot ask

* Who, did you see John and —, ?

The Hold Mechanism in ATNs:-

Another technique for handling movement was first developed with the ATN framework.

Holdlist :- 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.

Hold action :- 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 that currently in a register

Eg:- The action HOLD SUBJ holds the constituent that is in the SUBJ register.

→ To 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.

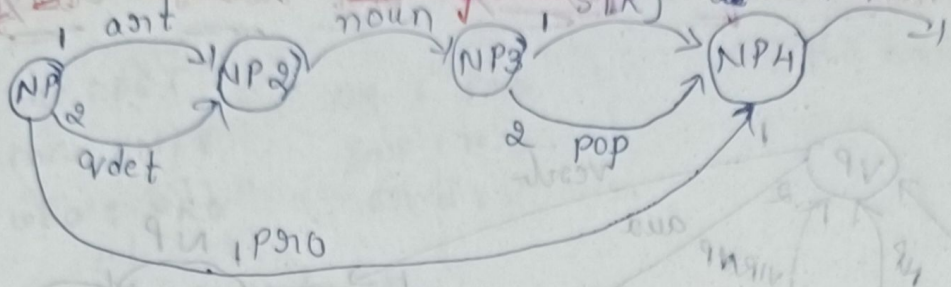
VIR (for virtual) :-

We 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

→ NP network that recognizes NPs involving wh-words as pronouns or determiners

→ As with CFGs, the feature

An NP Network including WH-words



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 ∧ AGR*	HEAD := * AGR := AGR ∧ AGR* MOD := *
NP3/1	WH* ∧ R	

→ WH is set to the value Q to indicate the NP is a wh-NP starting a question. It also accepts relative clauses with a push to the S network.

→ S network handles yes/no questions, wh-questions and relative clauses.

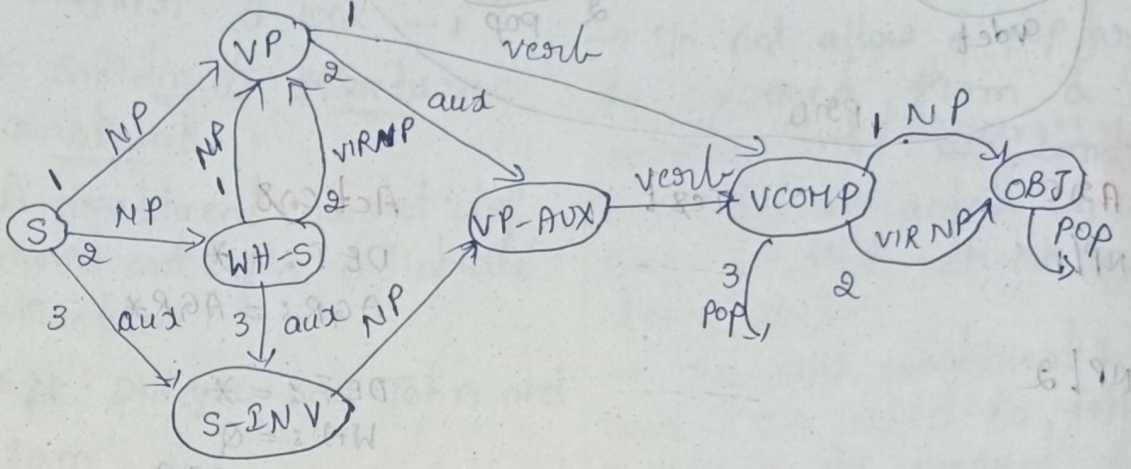
→ Wh-questions and relative clauses are processed by putting the initial NP (with a WH feature, Q or R) on to the hold list and later using it in a VIR arc.

→ The network is organized so that all +INV sentences go through node S-INV while all -INV sentences go through node VP.

→ All wh-questions and relative clauses go through node WH-S and then are redirected back into the Standard network based on whether or not the sentence is inverted.

→ All wh-questions and relative clauses go through node WH-S and then are redirected back into the

An S network for questions and relative clauses



Arc	Test	Actions
S/1		INV := - SUBJ := *
S/2	WH* ∩ {@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* VFORM* ∩ {past pres}	MAIN-V := *
VP/2	AGR$_{SUBJ}$ ∩ AGR* VFORM * ∩ {past pres}	AUX := *
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	

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

The : DET
man : N, V
who : PRO
we : PRO
saw : N, V
cried : V

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 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 WHR 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 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	returning (S WH R SUBJ we MAIN-V saw OBJ who)

the standard network, based on whether or not the sentence is inverted.

for non inverted Questions

Who ate the pizza?

for relative clauses in NP form

The man who ate the pizza held NP is immediately used by the VIR arc WH-S/2

for other relative clauses

The man who we saw arc WH-S/1 used to accept the subject in the relative clause and held NP used later on arc VCOMP/2.

for inverted Questions

Who do I see?

are WH-S/3 is followed to accept the auxiliary and held NP.

→ this network only accepts verbs with SUBCAT values _none and _np but could easily be extended to handle other verb complements.

Note:

ATN would not accept

* who did the man see the boy?

The held constituent who is not used by an VIR arc; thus the pop arc from the network cannot be taken.

* the man who the boy saved ate the pie
unacceptable as the relative pronoun is not used by a VIP arc in the S network that analyzes the relative clause.

Comparing the Methods :-

2 approaches to handling questions in grammars
1. use of the GAP features
2. use of the hold list in ATNs.

3 important considerations
1. coverage - whether the approach can handle all examples
2. selectivity - whether it rejects all ill-formed examples
3. conciseness - how easily rules can be specified.

→ under reasonable assumptions both methods appear to have the necessary coverage, that is, a grammar can be written with either approach that accepts any acceptable sentence.

GAP feature

1. GAP feature theory is that a symbol such as NP with an NP gap must be realized as the empty string ie) you could not have an NP with an NP gap inside it.

* Who did the man who saw ~~it~~ the boy?
not comprehensible

→ In relative clause, the relative pronoun would be taken as the subject and the initial query NP would be taken as the object. This sentence is not acceptable by any grammar using GAP features, however, because the GAP feature cannot be propogated into a relative clause since it is not the head constituent in the rule

CNP → CNP REL

Even if it were the head, the sentence would still not be acceptable, since the erroneous analysis would require the GAP feature to have two

ATN Hold list Mechanism

1. Such structure could be parsable using the ATN hold list mechanism

→ This sentence is accepted by ATN because hold list would contain two NPs one for each occurance of who.

→ The hold list mechanism in the ATN framework must be extended to capture these constraints, because there is no obvious way to prevent held constituents from being used anytime they are available. The only possible way to restrict it using the existing mechanism would be to use feature values to keep track of what held constituents are available in each context.

New action

HIDE - that temporarily hides the existing constituents on the hold list until either an explicit action.

values when it starts analyzing the relative clause.

→ GAP feature propagation was introduced with context-free grammars.

Common Benifits :-

→ we can develop an extension to CFGs that incorporates a hold list and uses it for gaps.

→ we might be able to develop some rules for GAP features propogation in an ATN.

→ Since GAP feature propogation rules depend on the notation of a head subconstituent, which doesn't have a direct correlate in ATN grammars.

UNHIDE - is executed, or the present constituent is completed. With this extension the ATN grammar to handle relative clauses could be modified to execute a HIDE action just before the hold action that holds the current constituent is performed.

→ Hold list mechanism was introduced with ATNs these techniques are not necessarily restricted to these formalisms.

Gap Threading :-

→ 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 resulting list of fillers that were not used after the constituent is parsed.

A logic grammar using gap 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.)

A trace of the parse of 1 The 2 man 3 who 4 we 5 saw 6 cried

step	state	Nekt 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)

16. VP(6,7, nil, nil)

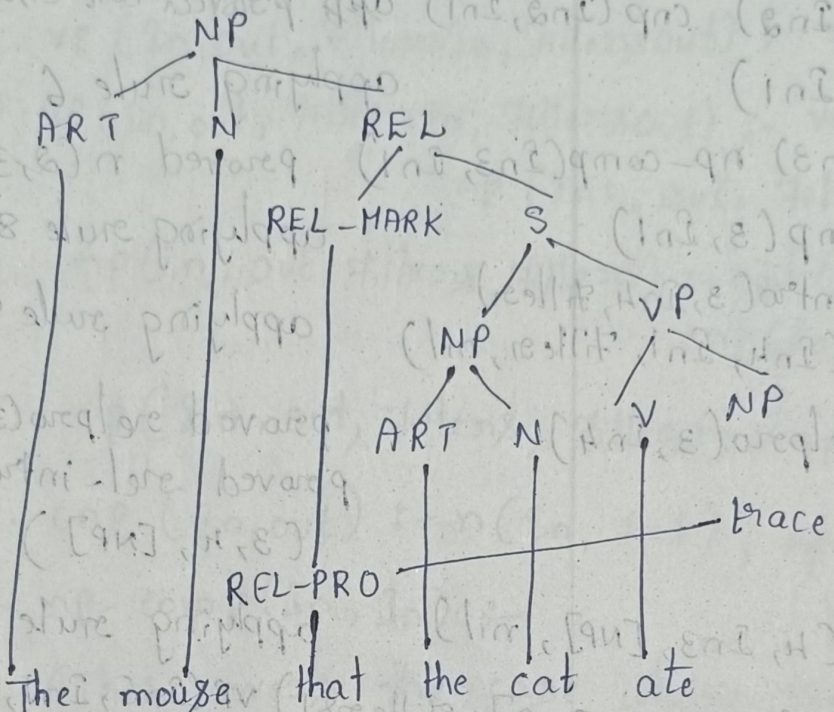
16. V(6,7).

proved V(6,7)

proved VP(6,7, nil, nil)

proved S(1,7, nil, nil).

A parse Tree in extraposition grammar :-



Thus the predicate

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

is true only if there is a legal 's' constituent between position-in and position-out of the input.

→ If a gap was used to build the S, its filler will be present in fillers-in but not in fillers-out.

Eg:- An S constituent with an NP gap would correspond to the predicate S(In, Out, [NP], nil). There are no gaps in a constituent, the fillers_in and fillersout will be identical.

→ Just as a convenient notation was designed for definite clause grammars, which then could be simply translated into a PROLOG program, a notation has been designed to facilitate the specification of grammars that handle gaps.

→ In particular, there is a formalism called extraposition grammar,

which, besides allowing normal context-free rules, allows rules of the form

REL-MARK ... TRACE → REL-PRO

which essentially says that the constituent REL-MARK, plus the constituent TRACE later in the sentence can be rewritten as a REL-PRO.

→ Such a rule violates the tree structure of syntactic forms and allows the analysis. Such rules can be compiled into a logic grammar using the gap-threading technique.

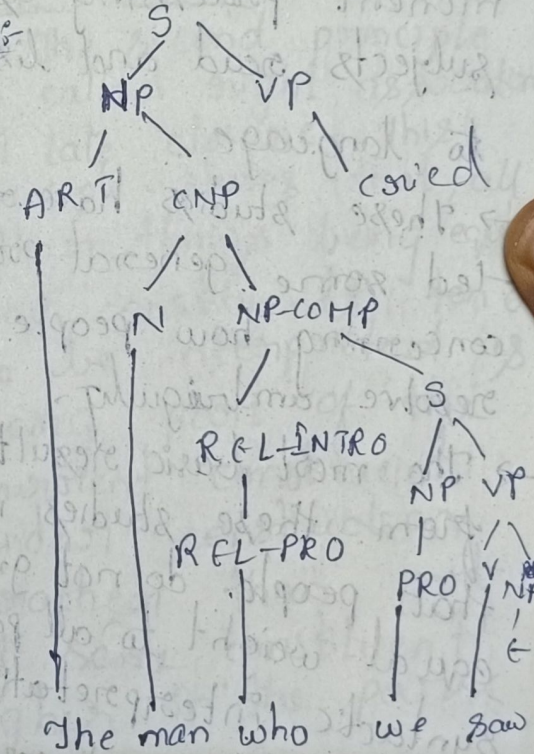
→ Consider how gap threading compares with the other approaches.

→ By using additional arguments on predicates, any of the approaches could be implemented. If viewed as simply an implementation technique, then the gap threading approach looks like a way to implement a hold list in a logic grammar.

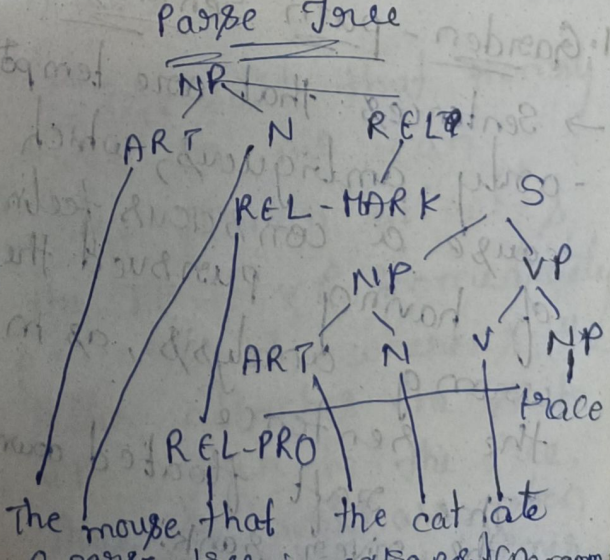
→ Since the grammar designer can decide to propogate the hold list or not in the grammar it provides the flexibility to avoid some of the problems with the simple hold list mechanism.

→ The propogation constraint must be explicitly enforced rather than being a consequence of the formalism.

Eg:



Parse Tree



Human Preferences in Parsing :-

→ Psycholinguists have conducted many investigations into this issue using a variety of techniques, from intuitions about preferred interpretations to detailed experiments monitoring moment-by-moment processing as subjects read and listen to language.

→ These studies have revealed some general principles concerning how people resolve ambiguity.

→ The most basic result from these studies is that people do not give equal weight to all possible syntactic interpretations.

1. Garden-path Sentences :-

→ 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 bank."

→ When you read the word "bank" you realize that the interpretation we have constructed so far for the sentence is not correct. Such sentences are often called garden-path sentences, based on the expression about leading someone down a garden path.

2. Minimal Attachment :-

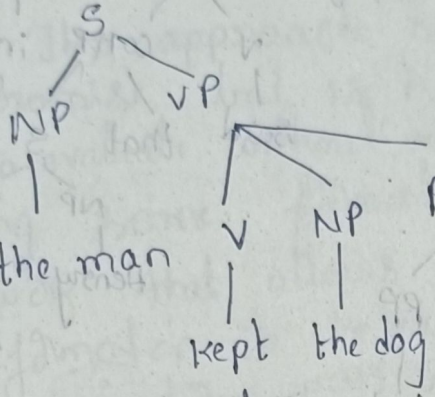
→ It states that there is a preference for the syntactic analysis that creates the least number of nodes in the parse tree.

Two interpretations for the simple CFG grammar

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

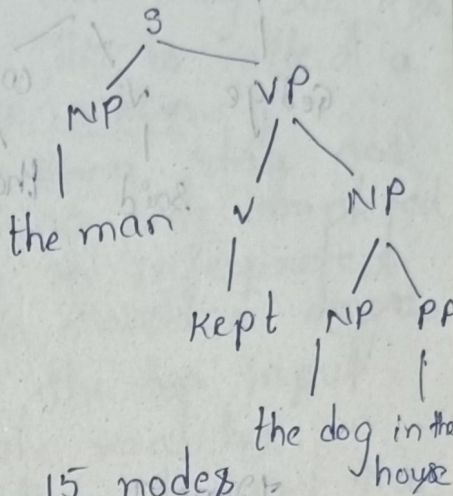
→ Given sentence "The man kept the dog in the house"

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



14 nodes

preferred



15 nodes

→ This principle appears to be so strong that it can cause certain sentences to be almost impossible to parse correctly.

Eg:- We painted all the walls with cracks which against all common sense, is often read as meaning that cracks were painted onto the 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).

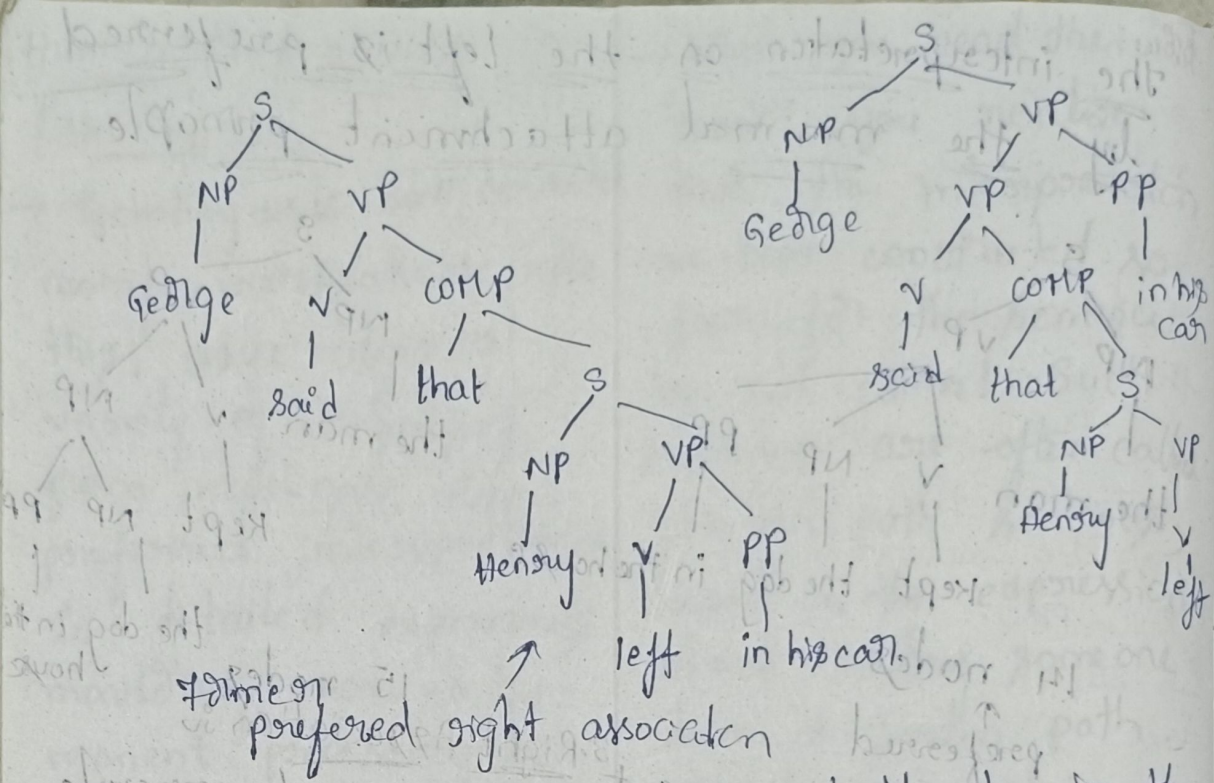
3. Right Association :-

→ 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).

Eg:-

George said that Henry left in his car

→ The former attaches the PP to VP immediately preceding it, whereas the latter attaches the PP to VP higher in the tree. Thus, the right association principle prefers the former.



Time flies like an arrow.

prefered right association

4. Lexical Preferences :-

→ In certain cases the two preceding principles seem to conflict with each other.

Eg:- 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 to favor the PP modifying the VP.

→ Minimal attachment takes priority over right association, the relationship appears to be more complex than that.

Eg:- I wanted the dog in the house.

2. I kept the dog in the house.

3. I put the dog in the house.

→ These examples demonstrate that lexical items, in this case the verb used, can influence parsing preferences.

→ Lexical preferences will override the preferences based on the general principles.

→ want verb has no preference for any PPs

→ keep might prefer PP with prepositions in, on, or by to be attached to the VP.

→ finally verb "put" requires (subcategorizes

for) a PP beginning with in, on, by and so on, which must be attached to the VP.

→ This approach has promise but is hard to evaluate without developing some formal framework that allows such information to be expressed in a uniform way will be addressed.

Shift Reduce Parsers :-

→ The states which can be used to control a parser that maintains two stacks: the parse stack, which contains parse states (i.e. nodes) 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 S_2, $S \rightarrow NP\ VP \bullet$ indicate that the parser should rewrite the top symbols on the parse stack according to this rule. This is called a reduce action.

→ Any other state not containing any completed rules is interpreted by the transition diagram.

→ If the top input symbol matches an arc, then it and the new state (at the end of the arc) are pushed on to the stack. This is called shift action.

→ Using this interpretation of states you can construct a table called the oracle that tells the parser what to do in every situation.

→ For each state and possible input, it specifies the action and the next state. Reduce action can be applied regardless of the next input, and the accept action only is possible when the input stack is empty (i.e. the next symbol is the empty symbol ϵ).

1. Consider a simple CFG and derive shift-reduce parsers with Oracle table and also parse the input stack for the sentence "The man ate the carrot".

Soln:

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

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

A transition graph derived from grammar given.

The Grade 781 given grammar

state	Top Input Symbol	Action	GoTo
S_0	ART	shift	S_1
S_0	NP	shift	S_2
S_0	S	shift	S_0'
S_0'	ϵ	Succeed	---
S_1	N	shift	S_1'
S_1'	---	Reduce by rule 2.2	---
S_2	V	shift	S_3
S_2	AUX	shift	S_4
S_2	VP	shift	S_2'
S_2'	---	Reduce by rule 2.1	---
S_3	ART	shift	S_1
S_3	NP	shift	S_3'
S_3'	---	Reduce by rule 2.4	---
S_4	V	shift	S_5
S_5	ART	shift	S_1
S_5	NP	shift	S_5'
S_5'	---	Reduce by rule 2.3	---

Parsing stack

Parse stack	Input stack	
(S0)	(The man ate the carrot)	
(S1 ART S0)	(man ate the carrot)	The: ART
(S1' N S1 ART S0)	(ate the carrot)	man: N, VI, NP
(S0)	(NP ate the carrot)	ate - V
(S2 NP S0)	(ate the carrot)	the: ART
(S1' N S1 ART S3 V S2 NP S0)	( )	carrot: N
(S3 V S2 NP S0)	(NP)	
(S2 NP S0)	(VP)	
(S0)	(S)	

1. parsing algorithm for using an oracle

→ The parsing algorithm for a shift-reduce parser

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 $(S_n C_n \dots S_1 C_1 S_0)$, where S_i are parse states and C_i are constituents.

→ an input stack of form $(W_1 \dots W_n)$, 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_i 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: $(S\ W_1\ S_n\ C_n \dots S_1 C_1 S_0)$

input stack: $(W_2 \dots W_n)$

2. If Action(S_n, W_1) = Reduce i and gramm-ar 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.

Eg:- If rule i were NP → ART N, then the new state would be

parse stack: $(S_{n-2} C_{n-2} \dots S_1 C_1 S_0)$

input stack: $(NP\ W_1 \dots W_n)$

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.

Shift-Reduce Parsers and Ambiguity :

→ shift-reduce parsers are efficient because the algorithm can delay decisions about which rule to apply.

Eg:

1. NP → ART N REL-PRO VP
2. NP → ART N PP

A top-down parser would have to generate two new active arcs, both of which would be extended if an ART were found in the input. The shift-reduce parser, on the other hand, represents both using a single state

Eg:

NP1 : NP → • ART N REL_PRO VP
NP → • ART N PP

If an ART is found, the next state will be

NP2 → NP → ART • N REL-PRO VP
NP → ART • N PP

Thus the ART can be recognized and shifted onto the parse stack without committing to which rule it is used in. Similarly, if an N is seen at state NP2, the state is

NP3 : NP → ART N • REL-PRO VP
NP → ART N • PP
PP → • P NP

Now from NP3, we finally can distinguish the cases. If a REL-PRO is found next, the state is

NP4 : NP → ART N REL-PRO • VP
VP → • V NP

which leads eventually to a reduction by rule 1. If a P is found, we move to a different state, which eventually builds a PP that is used in the reduction by rule 2. Thus the parser delays the decision until sufficient information is available to choose between rules.

Lexical Ambiguity :

→ This ability to postpone decisions can be used to deal with some lexical ambiguity by a simple extension to the parsing process.

→ We allow ambiguous words to be shifted onto the parse stack as they are, and delay their categorization until a reduction involving them is made.

→ To accomplish this extension, we must expand the number of states to include states that deal with ambiguities.

Eg: if "can" could be a V or an AUX, the stack cannot determine a unique action to perform from state S2. If it were a V we would shift to state S3 and if it were an AUX we would shift to state S4. Such ambiguities can be encoded, however, by generating a new state from S2 to cover both possibilities simultaneously. This new state will be the union of states S3 and S4:

S3-4: VP → AUX · V NP
VP → V · NP
NP → · ART N

→ In this case the input should resolve the ambiguity. If we see a V next, we will move to S5 (just as we would from state S4). If we see an ART next, we will move to S1 and if we see an NP next, we will move to S3 (just as we would from S3).

→ Thus, the new state maintains the ambiguity long enough for succeeding words to resolve the problem.

→ The next word might also be ambiguous, so the number of new states could be quite large.

→ If the grammar is unambiguous, then there is only one possible interpretation of the sentence once it is completely parsed. This technique can be used to delay the decision as long as necessary.

→ Construction of deterministic simulation of a nondeterministic finite automata.

Ambiguous Parse States

→ Ambiguity that arises when one rule is a proper prefix of another, such as the rules

3. NP → ART N
4. NP → ART N PP

We will generate a parse state containing the two dotted rules

NP → ART · N
NP → ART · N PP

If the next input in this state is an N, we would move to a state consisting of

NP5: NP → ART N.
NP → ART N. PP
PP → . P PP

→ Problem :- If the next input is a P, eventually a PP will be built and you return to state NP5. But the parser cannot decide whether to reduce by rule 3, leaving the PP to be attached later, or to shift the PP (and then reduce by rule 4).

→ In any reasonable grammar of English, both choices might lead to acceptable parses of the sentence.

→ 2 ways to deal with this problem.

1. The first strategy maintains a deterministic parser and accepts the fact that it may misparse certain sentences. The goal is to model human parsing performance and fail on only those that would give people trouble.

→ one recent suggestion claims the following heuristics favor more intuitive interpretations over less intuitive ones:
→ favor shift operations over reduce operations
→ resolve all reduce-reduce conflicts in favor of the longest rule (i.e. the one that uses the most symbols from the stack)

→ Using these strategies, we could build a deterministic parser that picks the most favored rule and ignores the others. It has been claimed that the first heuristic corresponds to the right-association preference and the second to the minimal attachment preference.

2. The second approach to dealing with the ambiguous states is to abandon the deterministic requirement and reintroduce a search. In this case we could consider each alternative by either a depth-first search with backtracking or a breadth-first search maintaining several interpretations in parallel.
→ Both of these approaches can yield efficient parsers.
→ The depth-first approach can be integrated with the preference strategies so that the preferred interpretations are tried first.

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, a richer set of actions is allowed that operates on an input stack called the buffer.

→ 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 :-

→ create a new node on the parse stack (to push the symbol onto the stack).

→ Attach an input constituent to the top node on the parse stack.

→ 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.

→ This technique makes the limited lookahead technique surprisingly powerful.

Eg:-

Consider the situation which might occur in parsing the sentence "The cat ate the fish".

The Parse Stack	The Buffer
Top → (S SUBJ (NP DET the HEAD cat))	ate the fish

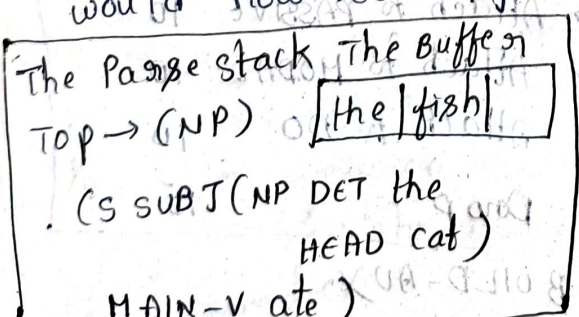
A situation during a parse.

→ Assume that the first NP has been parsed and assigned to the SUBJ feature of the

S constituent on the parse stack. The operators introduced earlier can be used to complete the analysis:

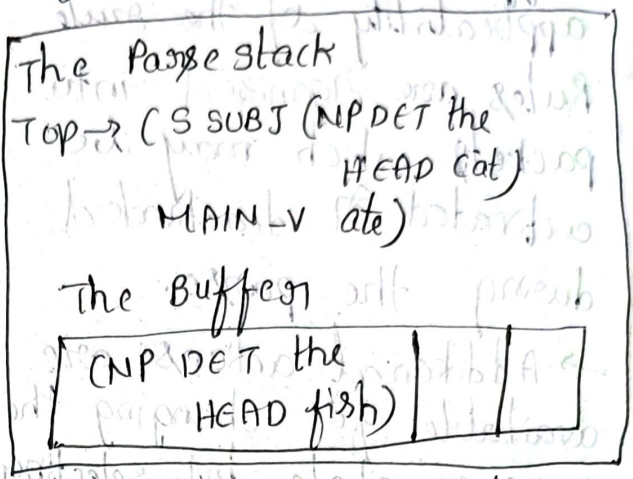
→ The operator Attach to MAIN-V would remove the lexical entry for ate from the buffer and assign it to the MAIN-V feature in the S on the parse stack. Next the operator Create NP would push an empty NP constituent onto the parse stack, creating the situation

Next the two operators
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.



After creating an NP

→ The operator Drop pops the NP from the parse stack and pushes it back onto the buffer, creating the situation below



After the drop action

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

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

→ Three other operators 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.

→ Packets - Each rule has a pattern that contains feature checks on the buffer to determine the applicability of the rule. Rules are organized into packets which may be activated or deactivated during the parse.

→ 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 (that is, pushed on the stack) all the active packets associated with the previous top of the stack become inactive until that constituent is again on the top of the stack.

→ The drop action will always deactivate the rules associated with the top node.

→ Packets play a role similar to the states in the shift-reduce parser.

→ Unlike the states in the shift-reduce parser more than one packet may be active at a time.

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

The rules for packet BUILD-AUX

→ The active packets are associated with the symbols on the parse stack. If a new constituent is created

→ Thus, there would be no need to create packets consisting of the union of packets to deal with word ambiguity, since both can be activated simultaneously.

Eg:-

→ 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, pastpart>

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 pastpart.

→ The priority associated with each rule is used to decide between conflicting rules. The lower the number, the higher the priority.

→ In particular, rules with the pattern <true> 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 rule match, and it completes the parsing of the auxiliary and verb structure.

A typical state of the Parser :

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})	

→ Above figure 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 BUILD-AUX active, and an S structure below 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 BUILD-AUX, the parser's next action is determined by seeing which rules match.

→ Rules 1 and 6 succeed, and 1 is chosen because of its higher priority.

→ Applying the actions of rule 1 produces the state in below fig. Now the rules in BUILD-AUX are applied again.

PASE-AUX and CPOOL are active, they compete to determine the next move of the parser (which would be to attach the AUXS structure into the S structure).

→ The lookahead rules are restricted to examining at most the first three buffer elements, with one exception that occurs when an NP sub constituent needs to be constructed.

After rule 1 is applied

The Parse Stack

Nodes

Top → (AUXS PERF has)
(S MOOD DECL)
(S SUBJ NP NAME John))

Active Packets

(BUILD-AUX)

(PARSE-AUX CPOOL)

The Input Buffer

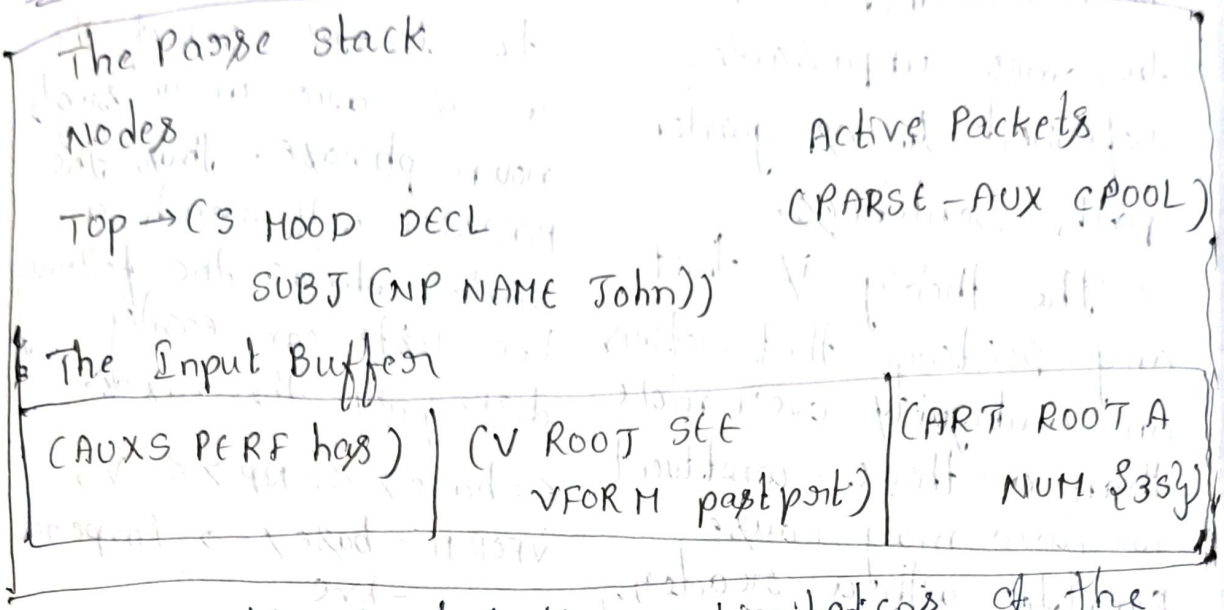
(V ROOT SEE VFORM pastprt)	(ART ROOT A NUM {3SG})	(N ROOT DAY NUM {3SG})

This time only rule 6 succeeds so that the next action is a drop, creating the state in below fig. At this stage the rules in packets

Attention Shifting

→ If the NP starts in the second or third buffer position then while it is being parsed, the position is used as through

The parse state after a drop action



it were the first buffer called attention shifting this circumstance is the only exception to the three-buffer restriction.

→ With this qualification a rule may still inspect only three buffer positions, but the starting position may be shifted.

→ Attention shifting is restricted so that under no circumstances can a rule inspect a position beyond the first five.

2. Psychological validity

→ Because of the limits on the lookahead and the deterministic nature of this parser.

→ Limitations of the mechanism itself account for various constraints on the form of language, such as the complex-NP constraint.

→ Rather than having to impose such a constraint in the grammar, this parser, because of its limitations could not operate in any other way.

→ Because of the limited lookahead, this mechanism must commit to certain structural analyses before the entire sentence has been examined.

→ In certain cases a sentence may begin in such a way

that the wrong decision is made and the sentence becomes unparsable. referred to as garden-path sentences.

→ The theory is that any sentence that retains an ambiguity over more than a three-constituent window may cause trouble with a reader.

→ Note that the lookahead is three constituents, not words, thus an ambiguity might be retained for quite some time without causing difficulty.

Eg: following two sentences are identical for the first seven words:

Have the students who missed the exam take it today.

Have the students who missed the exam taken it today?

→ The ambiguity between being an imperative or a sentence versus a yes/no question, however, never extends beyond three constituents, because six of the words are in a single noun phrase. Thus, the parser will reach a state where the following two tests can easily distinguish the cases:

<=have> <=NP> <=V>, VFORM = base> → imperative

<=have> <=NP> <=V>, VFORM = pastprt> → yes/no question

→ A sentence such as Have the soldiers given their medals by their sweethearts cannot be disambiguated using only three constituents, and this parser, like most people, will initially misinterpret the sentence as a yes/no question that is ill formed, rather than recognize the appropriate imperative reading corresponding to "Have their sweethearts give the soldiers their medals".