Computational Linguistics SC579

Ch6. Chart parser

Prof. Feng Zhiwei

6.1 Well-Formed Sub-string Table (WFST)

6.1.1 Why we need WFST

-- A WFST is a mechanism enabling a parser to keep a record of structures it has already found, so that it can avoid looking for them again.

If we have the rules of PSG grammar as follows:

S → NP VP 1
VP → IV 2
VP → IV PP 3
VP → TV NP 4
VP → TV NP PP 5
VP → TV NP VP 6
NP → Det N 7
NP → Det N PP 8
PP → Prep NP 9

Here IV is intransitive verb, TV is transitive verb, Prep is preposition).

Lexicon:

the: <cat> = Det
her: <cat> = Det
her: <cat> = NP
they: <cat> = NP
nurses: <cat> = NP
nurses: <cat> = N
book: <cat> = N
travel: <cat> = N
report: <cat> = N
report: <cat> = IV
hear: <cat> = TV
see: <cat> = TV
on: <cat> = Prep

This PSG can generate following sentences:

Nurses hear her.

The nurses report.

They see the book on the nurses.

They hear her report on the nurses.

Consider the problem confronted by a top-down parser faced with the string

They saw the nurses report

Ignoring the parsing of subject noun phrase, the top of the research tree looks as shown in Fig.1 (For brevity, we have assumed that the parser is intelligent with lexical categories and does not waste time looking for lexical items that are not present).

The main part of the search tree (circled) shows the parser's efforts in looking for just a noun

phrase after the transitive verb ‘saw’.

All the paths in this part of the tree lead to failure, because there is an intransitive verb after the noun phrase, but nevertheless along the way a noun phrase is successfully parsed.

Abbreviation ‘stnr’ = saw the nurses report.

(a)

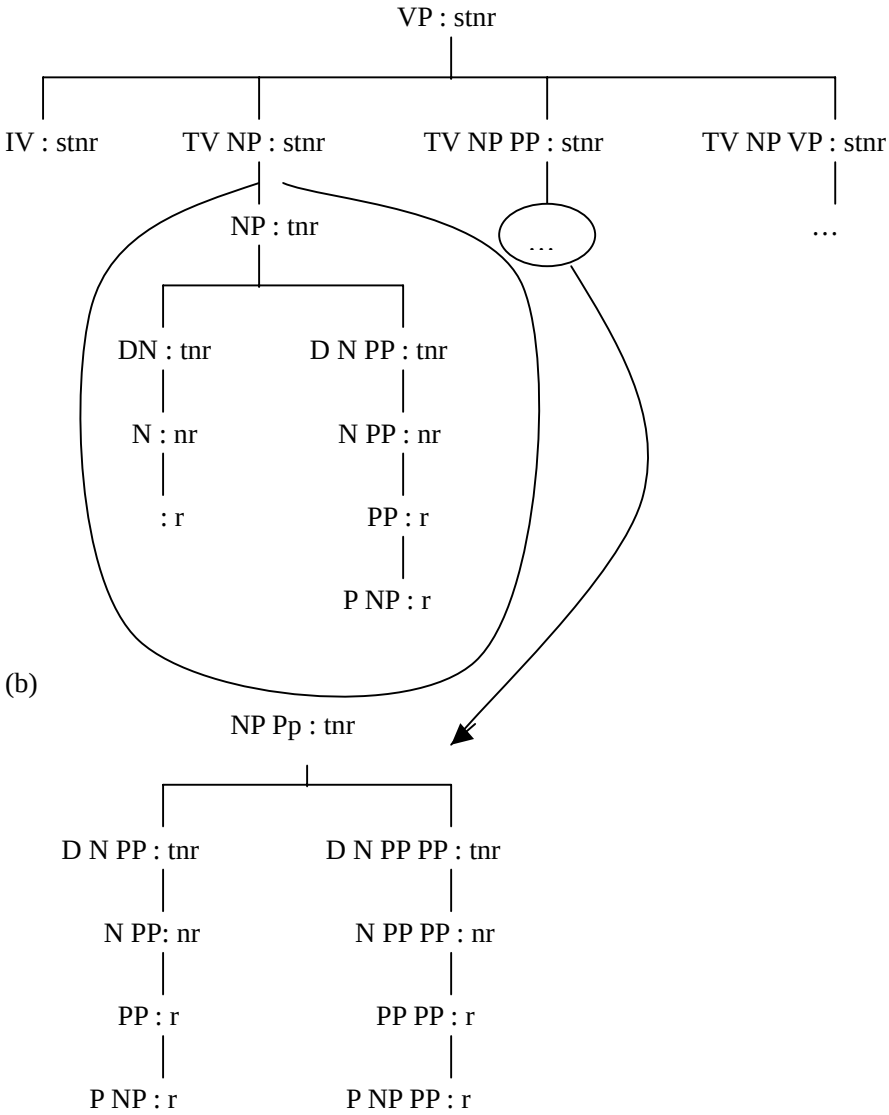


Fig. 1 parts of parsing search tree

The worrying things is that an almost identical tree is to be found when we look more closely at the next sub-tree (Fig. 1 (b)) Apart from the bottom left node, this tree is exactly like the previous one, except that there is an extra PP in front of each ‘:’. So the parser is having to do exactly the same work looking for a noun phrase in this part of the tree. Moreover, the same thing happens yet again in the third sub-tree, although this time there will be an extra PP in front of each ‘:’, but the work done looking for the noun phrase will be the same. Here, perhaps the extra work may not matter much, but parsing a noun phrase can sometimes be a lengthy process, and it would be wasteful to carry out the same work three times.

If our parser is exploring the search tree depth first from left to right, it will successfully parse the noun phrase ‘the nurses’ in the first sub-tree. But since the sub-tree leads to failure, everything in it will be discarded and the work will have to begin again in the second subtree. If the parser had kept some record of what it had found, it would not have had to repeat the work of finding it again.

The parser may record the correct result for this noun phrase as a complete structure as following:

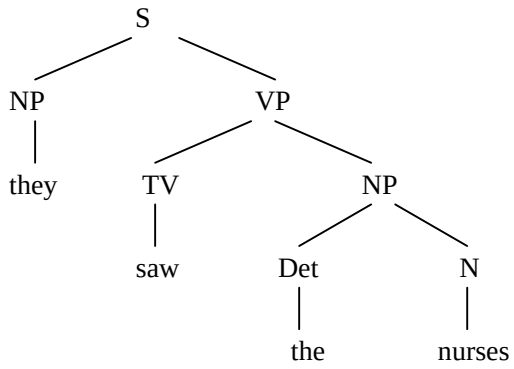


Fig.2 Representation a complete structure

When we parse third sub-tree and forth sub-tree, we can reuse this correct result for this noun phrase “the nurses”.

--If we use above CFG grammar to parse following sentence:

The nurses book her travel

In the lexicon of this grammar, the category of ‘book’ is only ‘N’:

book: <cat> = N

So our grammar can’t get the complete tree for this sentence. The result is a partial structure as follows:

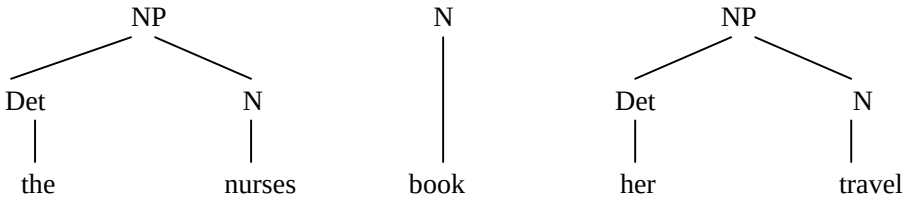


Fig. 3 representing partial structure

In the NLP, when there are ill-formed input, a grammatical construction that falls outside its own set of rules, a misspelling or a word that is not in its lexicon. In such case, a robust system will want the syntactic component to tell the semantic and pragmatic components as much as it has been able to find out about the syntactic structure, rather than simple breaking, reporting “No parse available”, and leaving other components to try and sort out the mess (disorder state) on their own on the basis of an unanalyzed string of words. If parser can record the partial structure, it will be helpful for the semantic component and pragmatic component.

--In the parsing, the ambiguous structure often can be disambiguated when the new information produced in the parsing process is accumulated enough to deal with the ambiguity. If the

ambiguous structure can be record, it is also useful.

For example, the sentence “they hear the report on the travel” is ambiguous. It can be parsed as following two alternative structures:

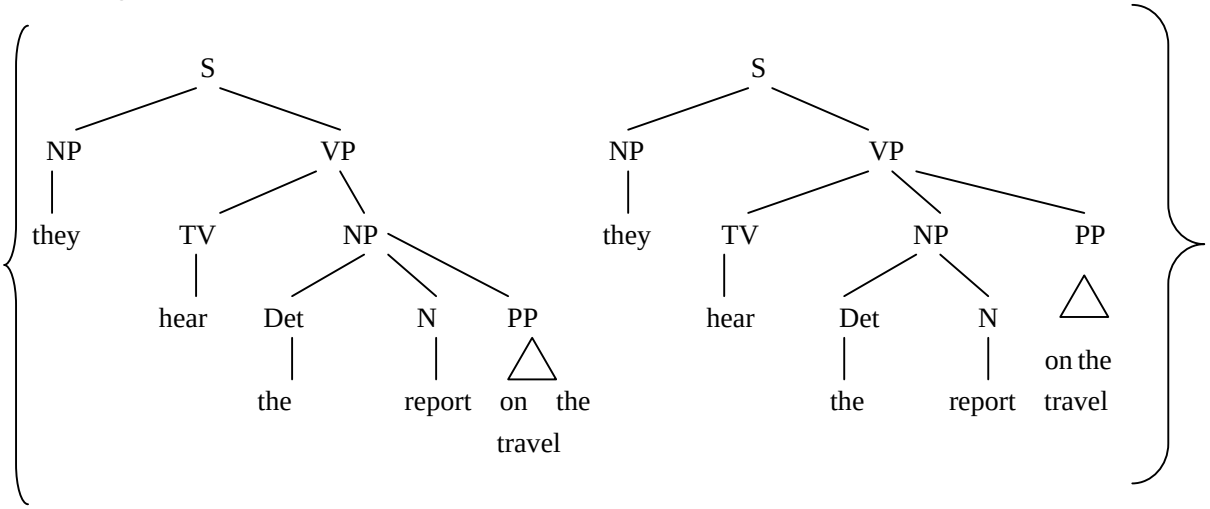


Fig. 4 Representing alternative structures

6.1.2 Description of WFST

- WFST is a directed (each arc has a particular direction), acyclic (containing no cycles) graph with unique first and last nodes, a graph whose nodes are (conveniently and conventionally) labeled from 0 (the first node) to n (the last node), where n is the number of words in the string, and whose arcs are labeled with syntactic categories and words.

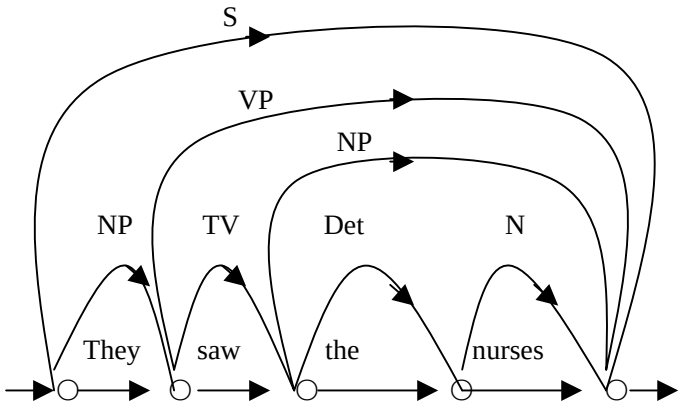


Fig. 5 complete structure in WFST

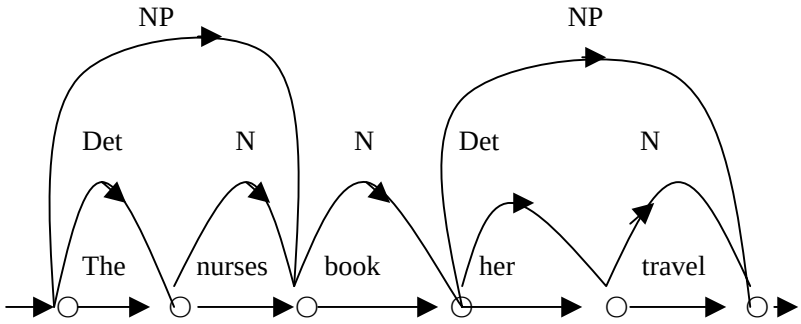


Fig. 6 partial structure in WFST

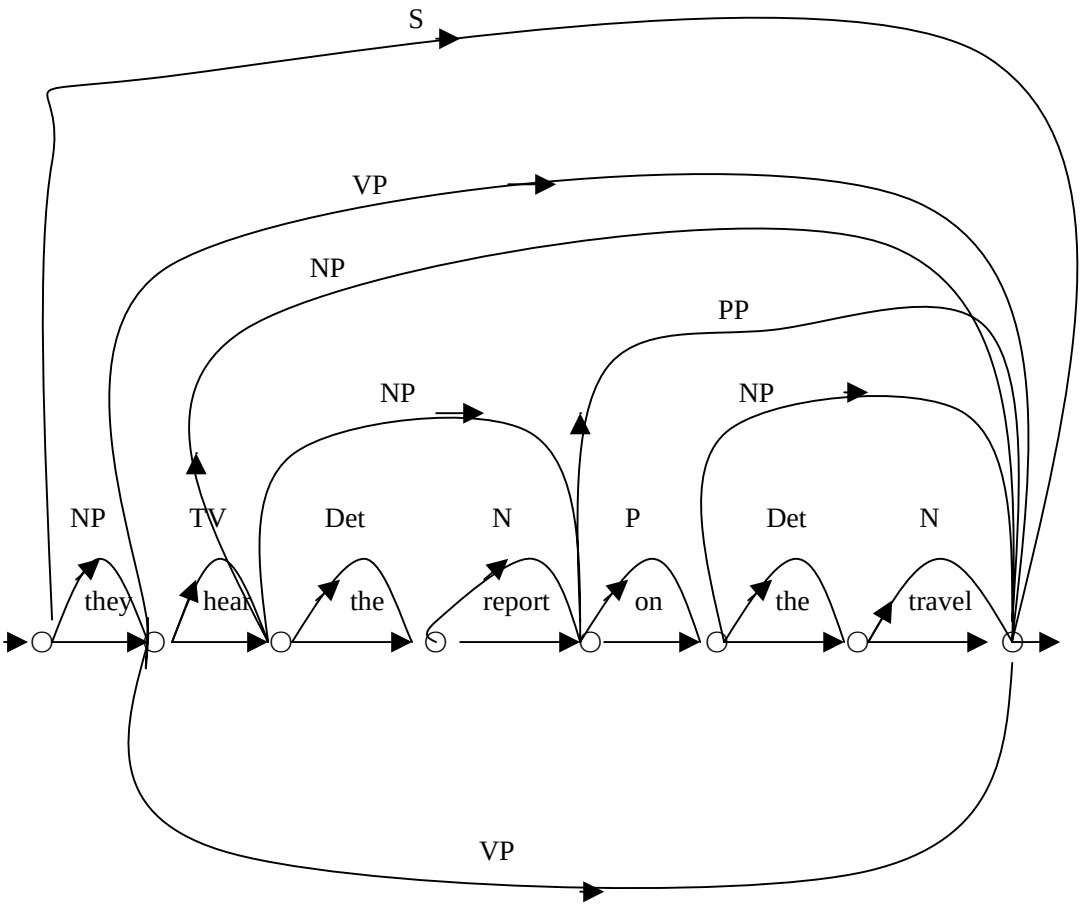


Fig. 7 alternative structure in WFST

- WFSR can be represented as a set of arcs, where an arc is a structure with the following attributes:

  <START> = … some integer …

  <FINISH> = … some integer …

  <LABEL> = … some category ..

E.g. NP arc:

<START> = 1

<FINISH> = 2

<LABEL> = NP

TV arc:

<START> = 2

<FINISH> = 3

<LABEL> = TV

Det Arc:

<START> = 3

<FINISH> = 4

<LABEL> = Det

……

6.1.3 Difference between WFST and PSG tree

- A phrase structure tree of the conventional kind is also a directed acyclic graph, but the nodes of phrase structure tree are labeled with categories and the arcs are unlabelled. In WFST, the arc carry the category labels and the nodes have essentially arbitrary names, although we have used integers and exploited the '<' relation on integers to implicitly encode the head and tail of the directed arcs.
- WFST encode the order of phrase directly: a constituent with an arc going from i to j in the table precedes a constituent with an arc from m to n just in case j < m. This is in contrast to tree, which encodes ordering directly for sister constituents only.

6.2 Chart parsing

The use of a simple WFST will save work because it means that a successfully found phrase only needs to be found once. The WFST is a good way of representing facts about structure, but WFST can not represents the information about structural hypotheses, conjecture (guess opinion based on incomplete information) or goals. In this case, we need to extend WFST to chart.

6.2.1 Rule form of chart parsing

We consider following graph:

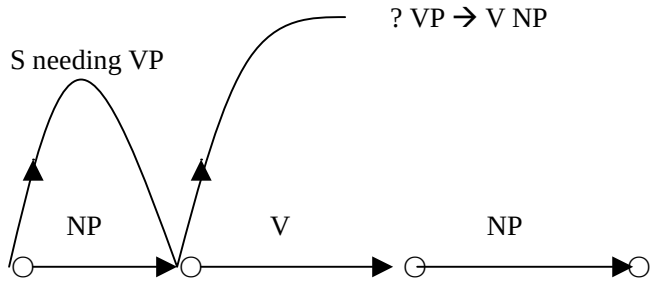


Fig. 1 Representing goals and hypothesis

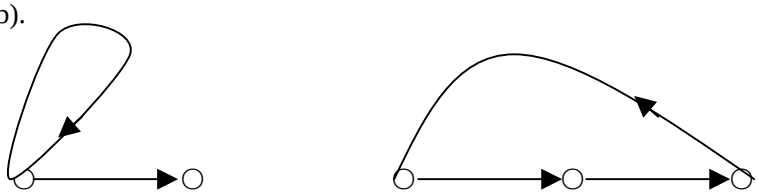
This wholly informal object is intended to represent the state of analysis of a parser that:

- has established that the string may consist of an NP V NP sequence.
- is exploring the hypothesis that it can be analyzed as an S consisting of an NP VP sequence.
- has got to the point where it has decided to treat the NP it has already established as identical with the NP of the NP VP sequence, and
- needs to establish that the subsequent V NP sequence can be covered by VP.

We have already seen that WFST can readily represent partial analyses, but what we have here is such hypothesis together with a hypothesis. We plainly cannot represent such hypothesis in our WFST data structure.

Two small changes to the data structure will give us what we need:

- Instead of requiring our directed graph to be strictly acyclic, we will relax things to permit single arcs (empty arcs) that cycle back to the node they start out from, as in Fig. 2 (a), but we will continue to prohibit graphs that have cycles like that shown in Fig 2 (b).



(a) (b)

Fig. 2 allowed cycle (a) and disallowed cycle (b) in chart

- Elaborate the label on arcs from a simple category to a decorated rule (dotted rules) of the grammar. If S → NP VP is a rule of the grammar, then the following objects (dotted rules) are all well-formed arc labels:

  S → .NP VP
  S → NP. VP
  S → NP VP.

Informly, the 'dot' in one of this labels indicates to what extent the hypothesis that this rule is applicable has been verified by the parser.

The first kind of label denotes that hypothesis that an S can be found covering that starts from the node in question and which covers that sub-string in virtue of the latter also being covered by an NP VP sequence.:

The second kind of label will only be found on arcs that cover an NP, that is, between nodes that are already spanned by the arc labeled 'NP → <category>', and thus indicate that the first part of the hypothesis, the presence of an initial NP, has been confirmed.

The third kind of label is the closest equivalent to the earlier category labels. It indicates a fully confirmed hypothesis and will be found on arcs that cover a string made up an NP sub-string followed by a VP sub-string.

A WFST that has been modified to include hypothesis in the manner described is known as an 'active chart', or simply 'chart'. And the parser that exploit the chart as a data structure are called as 'chart parser'.

A node in a chart is often referred to as a 'vertex' and the arcs as 'edges'.

Arcs that represent unconfirmed hypothesis are known as 'active edges' and those that represent confirmed hypothesis are known as 'inactive edges'.

Obviously, charts can represent everything that a WFST can. The Fig 3, 4, 5 show our earlier example WFST would appear in chart notation. These charts only exhibit inactive edges.

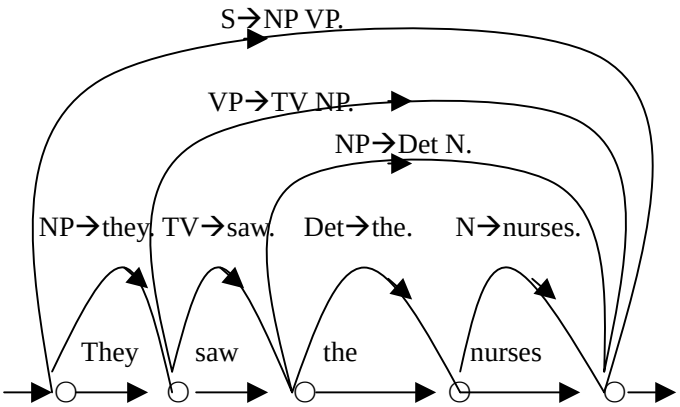


Fig. 3 complete structure in chart

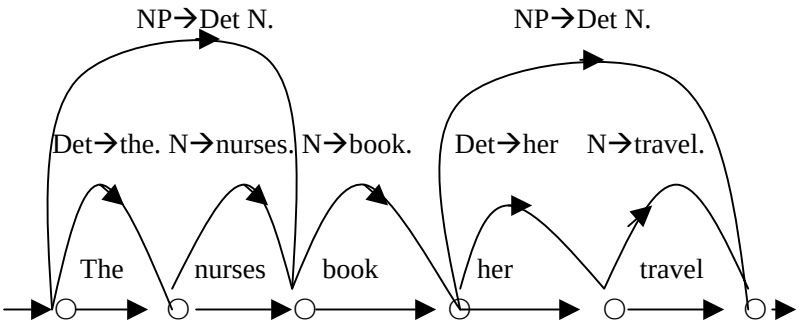


Fig. 4 partial structure in chart

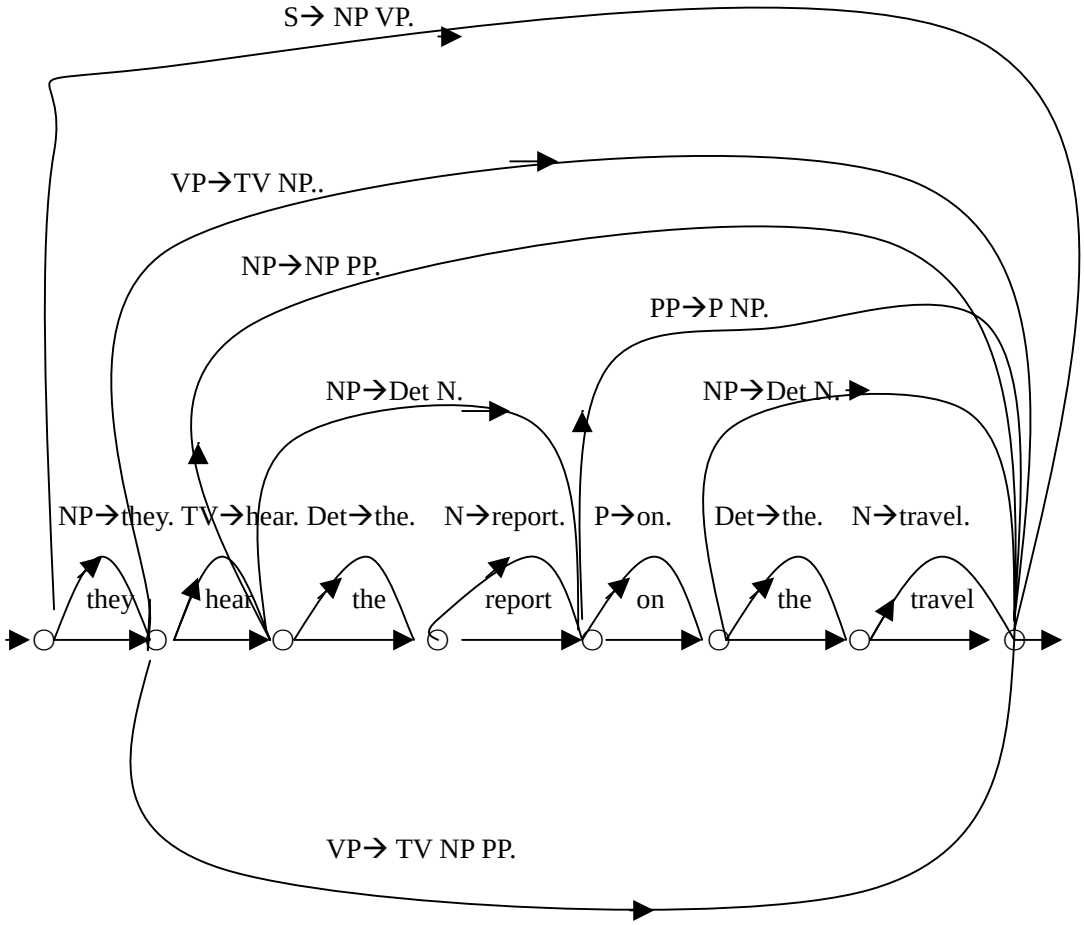


Fig. 5 alternative structure

Fig. 6 is the chart that corresponds to our informal hypothesis diagram in Fig.1. It contains two active edges.

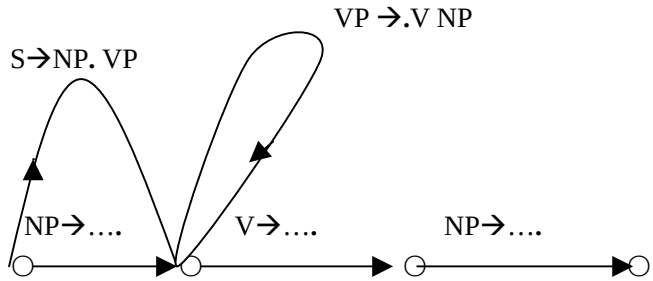


Fig. 6 Representing goals and hypothesis in a chart

We can represent a chart as a set of structures, each of which has the following attributes:

<START> = … some integer …

<FINISH> = … some integer …

<LABEL> = … some category …

<FOUND> = … some category sequence …

<TOFIND> = .. some category sequence …

Where <LABEL> is the LHS of the appropriate dotted rule, <FOUND> is the sequence of RHS categories to the left of the dot and <TOFIND> is the sequence of RHS categories to the right of the dot.

In this representation, an edge whose value for TOFIND is the empty sequence will be an inactive edge, and all other edges will be active.

We will occasionally use an n-tuple notation in the text to abbreviate such records. Thus, the three tuples:

<0,2,S→NP.VP>

<0,2,NP→Det N.>

<3,5,NP→Det N.>

represent the following active and inactive edges, respectively:

<START> = 0

<FINISH> = 2

<LABEL> = S

<FOUND> = <NP>

<TOFIND> = <VP>

<START> = 0

<FINISH> = 2

<LABEL> = NP

<FOUND> = <Det, N>

<TOFIND> = <>

<START> = 3

<FINISH> = 5

<LABEL> = NP

<FOUND> = <Det, N>

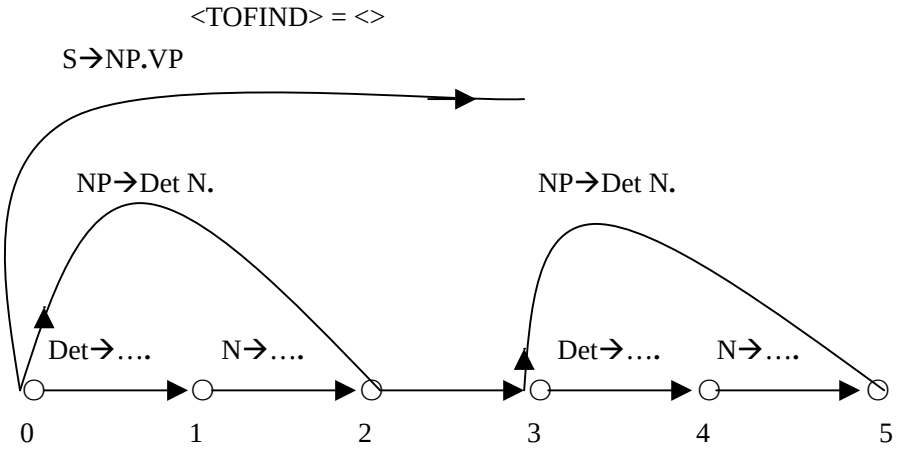


Fig. 7

6.2.2 Fundamental rule of chart parsing

Suppose that we are in the middle of the parsing process and we look at the chart to find it contains the following edges:

{<0,2,S→NP.VP>,
<2,3,VP→TV.NP PP>,
<3,5,NP→Det N.>,
<5,8,PP→P NP.>}

We can represent these edges diagrammatically in Fig. 8.

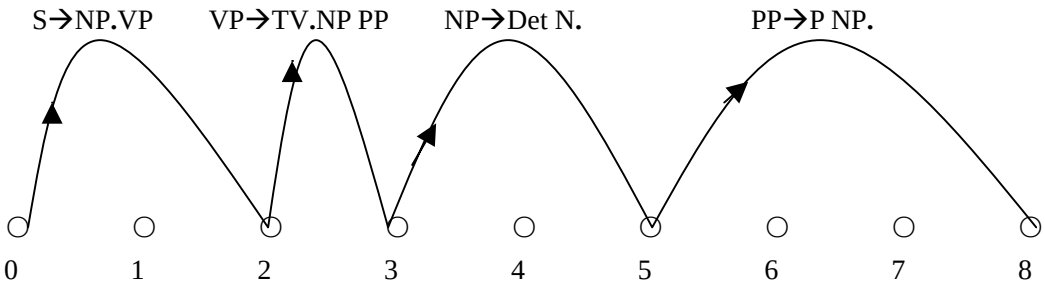


Fig. 8 example partial chart

Now we have two active edges and two inactive edges.

The latter constitute a noun phrase and a prepositional phrase that we have already found.

The first active edge represents a hypothesis about a sentence that has found a noun phrase, and is looking for a verb phrase.

The second active edge represents a hypothesis about a verb phrase that has found a transitive verb and is looking for a noun phrase followed by a prepositional phrase.

Consider the first active edge. To satisfy the hypothesis, we need to find an inactive verb phrase edge in the chart that starts at vertex 2. But there is no such edge.

Then turning our attention to the second active edge, we have a verb phrase hypothesis that is looking to find a noun phrase that starts at vertex 3. The edge 'NP→Det N.' is just this edge. Then we can add another active edge to the chart – namely, <2,5,VP→TV NP.PP>.

It is a hypothesis that is looking to find a prepositional phrase starting at vertex 5. We have one of

those in our chart, so this verb phrase hypothesis is fully confirmed and we can add an inactive edge to the chart to represent the verb phrase, namely, <2,8,VP→TV NP PP.>.

This is where we have got to:

{<0,2,S→NP.VP>,
<2,3,VP→TV.NP PP>,
<3,5,NP→Det N.>,
<5,8,PP→P NP.>
<2,5,VP→TV NP.PP>
<2,8,VP→TV NP PP.>}

In this case, if we return to our first active edge, we can see that its hypothesis is now also fully confirmed, since there is a verb phrase that starts at vertex 2. So, we can add the inactive edge <0.8.S→NP VP.> to our chart.

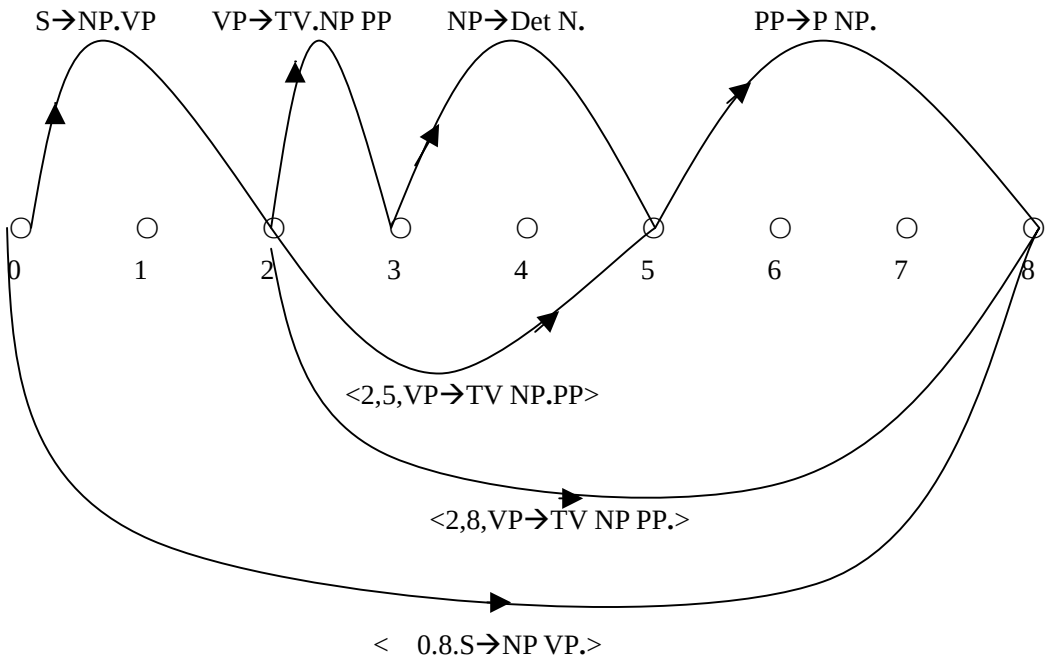


Fig. 9 example of a chart

In Fig.9, we have an S edge spanning the entire width of the graph. It means that we have succeeded in parsing the string as a sentence.

The process above represents the essence of the chart parsing. What we have done is to apply exactly the same rule three time: if an active edge meets an inactive edge of the desired category, then put a new edge onto the chart that spans both the active and inactive edges.

It is called the fundamental rule of chart parsing.

Strictly, the fundamental rule of chart parsing is as following:

If the chart contains edges <i,j,A→W1.B W2> and <j.k,B→W3.>, where A and B are categories and W1, W2 and W3 are (possibly empty) sequences of categories or words, then add edge <i,k,A→W1 B.W2> to the chart.

The rule does not say whether the new edge is active or inactive, but then it does not need to since this is fully determined by whether W2 is empty or not. If W2 is empty, then the new edge will be inactive edge; if W2 is not empty, then the new edge will be active edge.

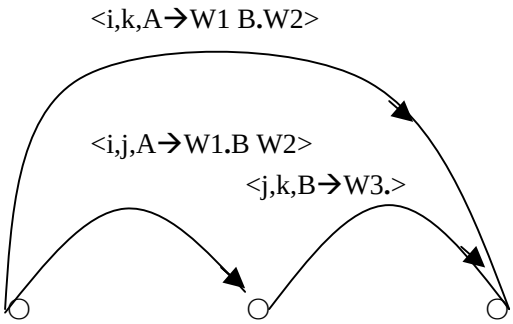


Fig. 10 fundamental rule of chart parsing

6.2.1 Initialization of chart parsing

We cannot apply the fundamental rule to a chart that contains no edges. There needs to be at least one active edge and one inactive edge for anything to happen.

■ Simple principle to insure such edges gets added:

Every time you add an inactive edge with label C to the chart, add a new empty active edge, starting from the same vertex, for every rule in the grammar that requires a constituent C as its leftmost daughter. This empty active edge starts from, and finishes at, the same vertex. It is just a cycle. In this case, the acyclic graph became to the graph with the cycle (cyclic graph). Then you can add the label to this empty active edge, the label is the rule which takes the constituent C as its leftmost element of RHS (leftmost daughter in the sub-tree).

S→.NP VP

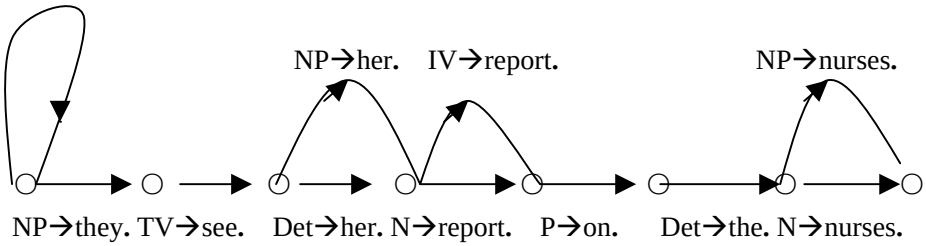


Fig. 1 chart initialization

In Fig.1, we only add the first active edge 'S→.NP VP' to the chart. Of course, we can add another active edges to the chart in according to same principle. Then we can use the fundamental rule to initialize the chart parsing process.

■ Bottom-up rule

If you are adding edge <i,j,C→W1.> to the chart, then for every rule in the grammar of the form B→C W2, add an edge <i,i,B→.C W2> to the chart.

B→.CW2

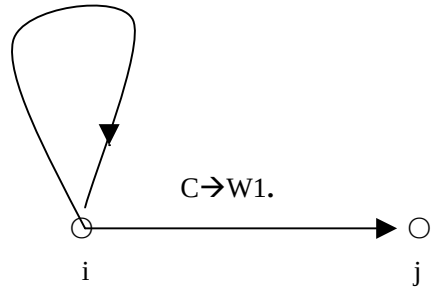


Fig. 2 bottom-up rule

The bottom-up rule will apply when most inactive edges are added to the chart, and the result is a

set of active edges to be added.

The active edges have been added to the chart corresponding to rules whose leftmost daughters have been found Clearly, there is plenty in the chart for the fundamental rule to work on, and, every time that the latter succeeds in adding an inactive edge to the chart, our bottom-up rule will also apply, providing more work for the fundamental rule to do.

For example, in the nodes 0, 1, 2 of Fig.1, we can add following active edges:

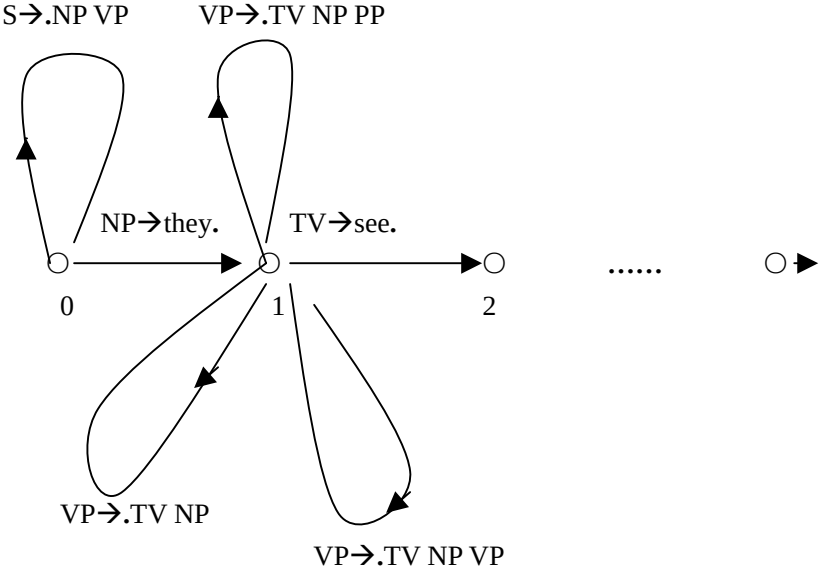


Fig. 3 Application of bottom-up rule to the initialized chart

Notice that in this parsing strategy a rule is retrieved as soon as the leftmost element on the RHS has been found. This indexing on the leftmost daughter means that the strategy is a kind of left-corner parsing..

■ Top-down rule

(1) At initialization, for each rule A→W, where A is a category that can span a chart (typically S), add <0,0,A→.W> to the chart.

(2) **If you are adding edge <i,j,C→W1.B W2> to the chart, then for each rule B→W, add <j,j,B→.W> to the chart.**

The first clause ensures that the parsing process gets started by a bunch of empty active S edges added to the first vertex.

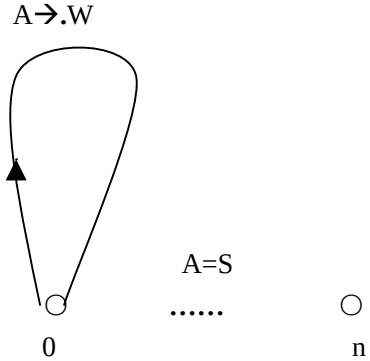


Fig. 4

The second clause entails that every active edge spawns (produces in large number) further empty active edges that are looking to find the first (non-preterminal, e.g NP, VP, etc.) category in the to-be-found list.

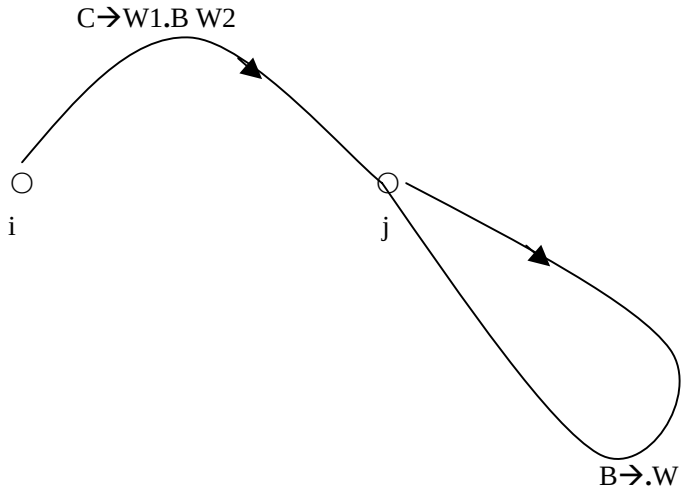


Fig. 5

Applying the first clause of our initialized chart example gives us the chart in Fig. 6, and applying the second clause gives that in Fig. 7

S→.NP VP

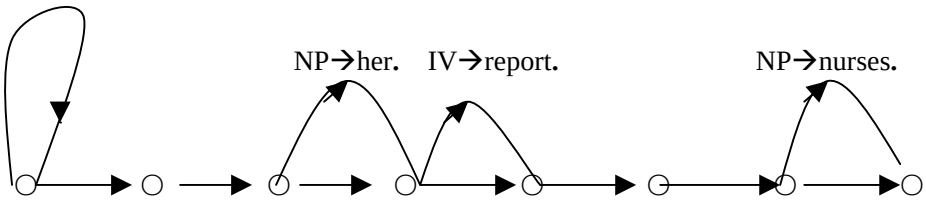


Fig. 6 Applying the top-down rule (1)

In Fig. 6, the RHS 'NP VP' in the rule S-->NP VP of grammar is same with the LHS of label 'NP-->they.' between 0 and 1, so we can add an active edge <0,0,S-->.NP VP> to initialize the chart parsing from top to down.

S→.NP VP

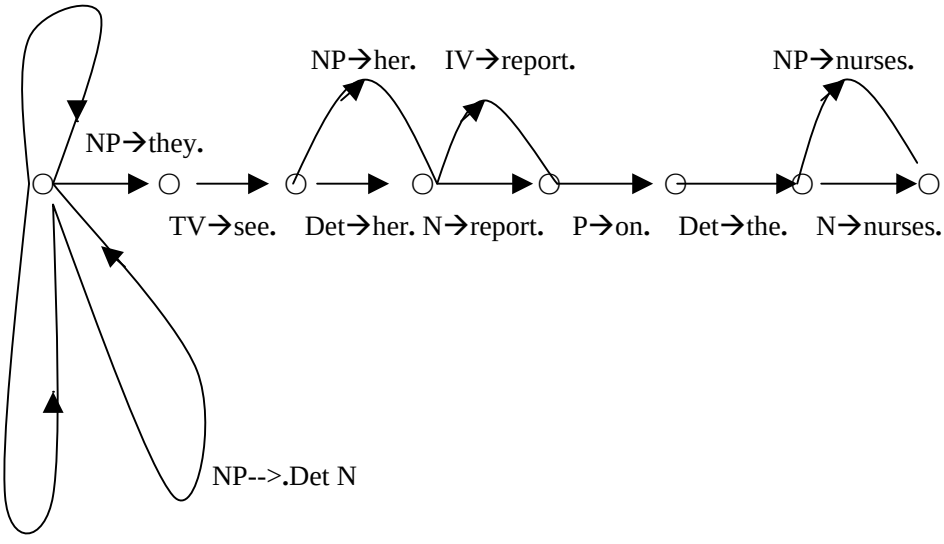


Fig. 7. Applying the top-down rule (1)

In Fig. 7, after we added the active edge 'S-->.NP VP' (in the label <i,j,C→W1.B W2>, i=0, j=0, C=S, W1= ϕ , B=NP, W2=VP),there are rules 'NP→Det N' and 'NP→Det N PP' in the grammar, its LHS is NP, so we can add the active edges <0,0,NP→.Det N>and <0,0,NP→.Det N PP> (in the label <j,j,B→.W>, j=0, B=NP, W=Det N, or W=Det N PP. These active edge can help to top-down parsing when we apply the fundamental rule.

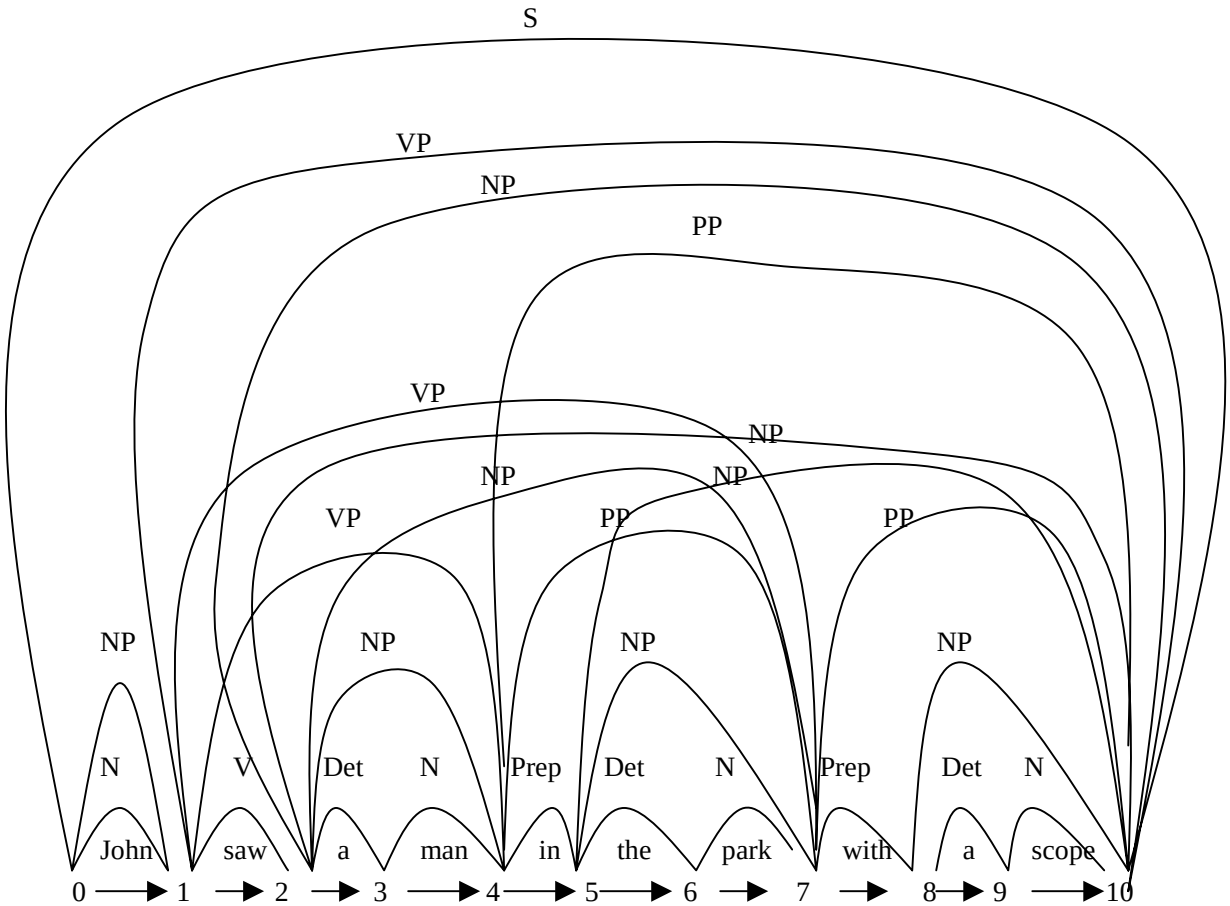
Example of Chart parser for an ambiguous sentence:

The CFG rules are as follows:

S → NP VP
NP → N
NP → Det N
NP → NP PP
PP → Prep NP
VP → V NP
VP → VP PP
N → {john, man, park, scope}
Det → {the, a}
V → {saw}
Prep → {in, with}

Using chart parser approach to analyze English sentence "John saw a man in the park with a scope", express the structure of this sentence by chart, write the analysis process of this sentence.

[answer] Chart:



Analysis process:

1. {0,0, N→.John}
2. {0,1, N→John.}
3. {0,0, NP→.N }
4. {0,1, NP→N.}
5. {1,1, V→.saw}
6. {1,2, V→saw.}
7. {2,2, Det→.a}
8. {2,3, Det→a.}
9. {3,3, N→.man}
10. {3,4, N→man.}
11. {2,2, NP→.Det N}
12. {2,3, NP→Det.N}
13. {2,4, NP→Det N.}
14. {1,1, V→.V NP}
15. {1,2, VP→V.NP}
16. {1,4, VP→V NP.}
17. {4,4, Prep→.in}
18. {4,5, Prep→in.}
19. {5,5, Det→.the}
20. {5,6, Det→the.}
21. {6,6, N→.park}
22. {6,7, N→park.}
23. {5,5, NP→.Det N}
24. {5,6, NP→Det.N}
25. {5,7, NP→Det N.}
26. {4,4, PP→.Prep NP}
27. {4,5, PP→Prep.NP}
28. {4,7, PP→Prep NP.}
29. {2,2, NP→.NP PP}
30. {2,4, NP→NP.PP}
31. {2,7, NP→NP PP.}
32. {1,1, VP→.V NP}
33. {1,2, VP→V.NP}
34. {1,7, VP→V NP.} (VPa)
35. {1,1, VP→.VP PP}
36. {1,4, VP→VP.PP}
37. {1,7, VP→VP PP.} (VPb)
38. {7,7, Prep→.with}
39. {7,8, Prep→with.}
40. {8,8, Det→.a}
41. {8,9, Det→a.}
42. {9,9, N→.scope}
43. {9,10, N→scope.}

44. {8,8, NP→.Det N}
45. {8,9, NP→Det.N}
46. {8,10, NP→Det N.}
47. {7,7, PP→.Prep NP}
48. {7,8, PP→Prep.NP}
49. {7,10, PP→Prep NP.}
50. {5,5, NP→.NP PP}
51. {5,7, NP→NP.PP}
52. {5,10, NP→NP PP.}
53. {4,4, PP→.Prep NP}
54. {4,5, PP→Prep.NP}
55. {4,10, PP→Prep NP.}
56. {2,2, NP→.NP PP}
57. {2,7, NP→NP.PP}
58. {2,10, NP→NP PP.} (NP1)
59. {2,2, NP→NP PP.}
60. {2,4, NP→NP.PP}
61. {2,10, NP→NP PP.} (NP2)
62. {1,1, VP→.V NP}
63. {1,2, VP→V.NP}
64. {1,10, VP→V NP.} (VP1) NP(2,10)→NP(2,4) + PP(4,10)
→NP(2,7) + PP(7,10)
65. {1,1, VP→.VP PP}
66. {1,4, VP→VP.PP}
67. {1,10, VP→VP PP.} (VP2)
68. {1,1, VP→.VP PP}
69. {1,7, VP→VP.PP}
70. (1,10, VP→VP PP.) (VP3) VP(1,7)→V(1,2) + NP(2,7)
→VP(1,4) + PP(4,7)
71. {0,0, S→.NP VP}
72. {0,1, S→NP.VP}
73. {0,10, S→NP VP.}

Comment:

- In VP1, NP(2,10) can be explained as “P(2,4) + PP (4,10)”or “P(2,7) + PP(7,10)”. There are two ambiguities.
- In VP3, VP(1,7) can be explained as “V(1,2) + NP(2,7)” or “VP(1,4) + Pp(4,7)”. There are two ambiguities.
- The total ambiguity number is 2 + 2 + 1 = 5.