Lecture 6: EFFICIENT CFG PARSING

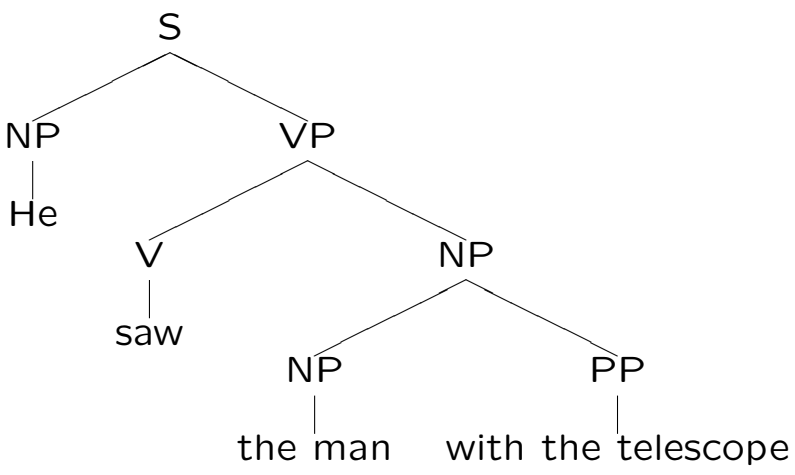
stephen.pulman@comlab.ox.ac.uk

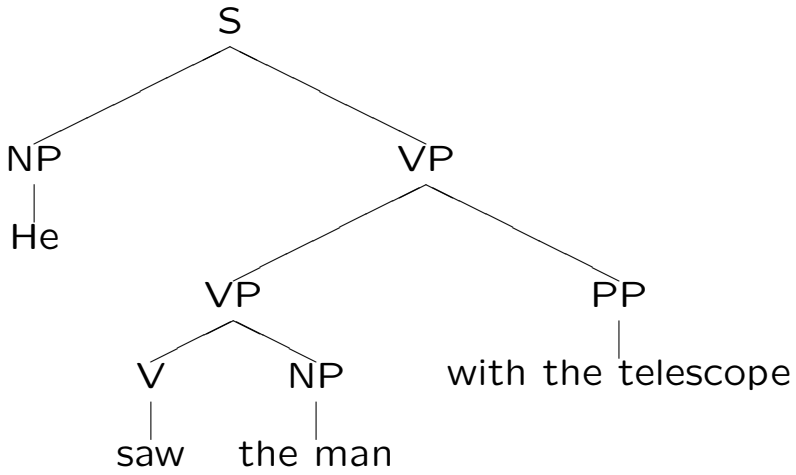
- The chart parsing framework
- Packing
- Reachability and lookahead
- Probabilistic parsing

J. Allen Chapters 3 and 4.

Jurafsky and Martin Chap 10 (First Edition), Chapter 13 (Second Edition)

We saw earlier that naive non-determinism leads to a lot of repeated work:





If we are parsing top down or bottom up, there will at some point be items of the form:

< .... [VP [VP saw the man] • PP], [with the telescope]>
< .... [NP [NP the man] • PP], [with the telescope]>

and both of these will lead to exactly the same sequence of actions to parse the PP 'with the telescope'. This is very wasteful - consider what would happen in a sentence like:

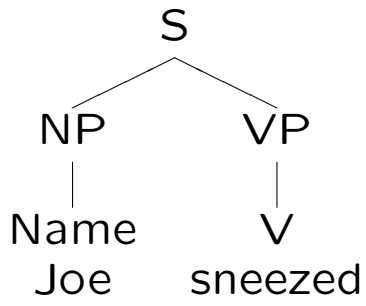
> *He saw the man with a telescope in a park near the museum with the exhibition about dinosaurs*

CHART PARSING

Although the CYK algorithm partly avoids this problem, it is not always convenient to have to transform the grammar to CNF, particularly since - as we will see later - syntax rules are used to do semantic interpretation. Most people use some form of 'chart' parsing.

Id	Constituent	From	To	Containing
c1	Name	0	1	Joe
c2	V	1	2	sneezed
c3	NP	0	1	c1
c4	VP	1	2	c2
c5	S	0	2	c3, c4

The equivalent tree representation would be:



A **chart** consists of a sequence of n+1 **vertexes/vertices** (for an n word sentence) representing positions between the words. Between vertices we record the items (usually called **edges**) constructed so far for the words between those vertices on a parse of the sentence. Edges contain an identifier, the rule that gave rise to them, the position (dot) in the rule so far, and also pointers to edges that are subconstituents of this one. Edges can be complete (dot at the end of the rule) or incomplete.

We can implement almost any parsing algorithm using a chart framework. We need (a) a mechanism for **proposing** new edges, given a grammar and a set of lexical entries, and (b) a method for **combining** complete and incomplete edges and (c) a regime for processing newly created edges (usually stored on an **agenda** before they are incorporated into the chart.)

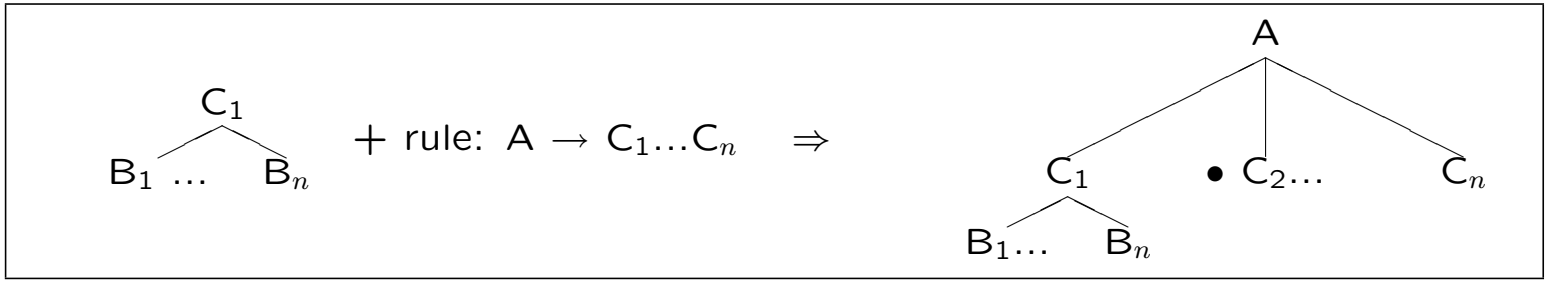
We could implement our simple bottom up parsing algorithm in a chart framework as follows:

Propose:

IF C_1 from X to Y, AND Rule A $\rightarrow$ $C_1 C_2 ... C_n$

THEN $A \rightarrow$ $C_1 \bullet C_2 ... C_n$ from X to Y

i.e. if there is a constituent C_1 from position X to position Y, and there is a grammar rule which builds an A from a series of constituents beginning with a C_1, then begin a new constituent, an incomplete A, consisting just of the recognised C_1, but expecting to find the remainder of the constituents. The notation
A $\rightarrow$ $C_1 \bullet C_2 ... C_n$ is to be read as 'something which will be an A when we have found $C_2 ... C_n$'.

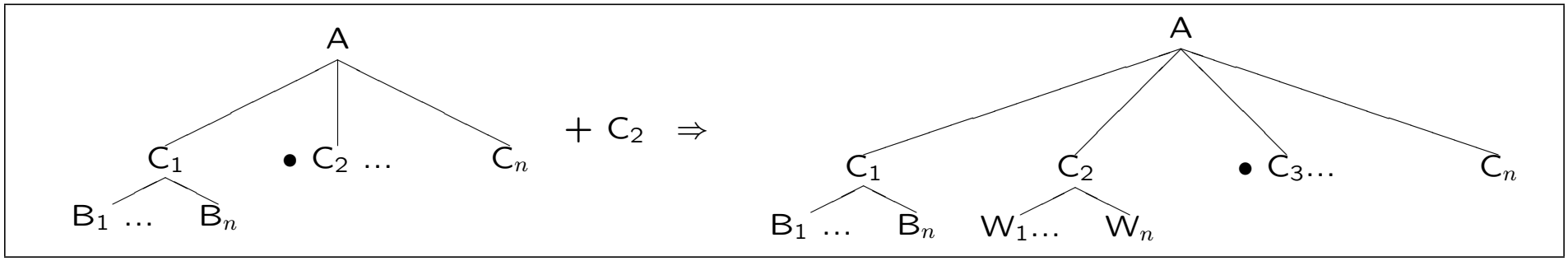


The other main rule is 'Combine' which combines an existing incomplete constituent with an existing complete one.

Combine:

IF $A \rightarrow C_1 ... \bullet C_i ... C_m$ from X to Y, AND C_i from Y to Z

THEN $A \rightarrow C_1 ... C_i \bullet C_{i+1} ... C_m$ from X to Z



Main Chart Loop

Initialise: For word$_i$ from 1 to N, create an edge from vertex i-1 to i for the word and put on agenda

Run:
Until agenda is empty:
take next edge E off agenda and put in chart
PROPOSE any new edges arising from E,
and put them on the agenda.
COMBINE E with any edges already in the chart,
putting resulting new edges on the agenda

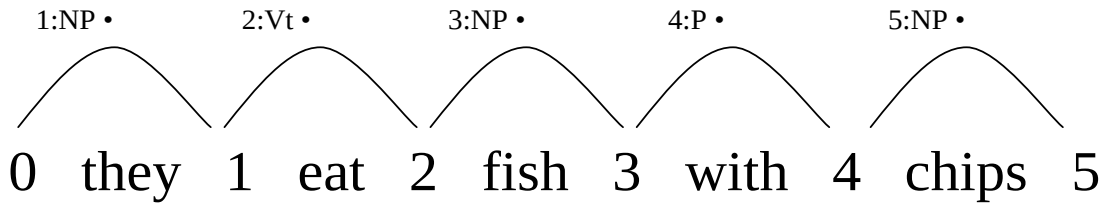
Check for complete edges of desired category spanning start to finish. The parse tree can be recovered by following the pointers from the complete edge to its components.

We can choose how to take edges off the agenda. If we treat the agenda as a stack, we will get depth first search behaviour. If we treat it as a queue we will get breadth first behaviour. Alternatively, we can try some heuristics to choose the next edge. This will only matter if we are interested in getting the first parse quickly, since we will eventually get all the parses anyway, whatever order we take edges off the agenda.

S $\rightarrow$ NP VP
NP $\rightarrow$ they | fish | chips | forks
NP $\rightarrow$ NP PP
VP $\rightarrow$ VP PP
VP $\rightarrow$ Vt NP
Vt $\rightarrow$ eat
PP $\rightarrow$ P NP
P $\rightarrow$ with | without

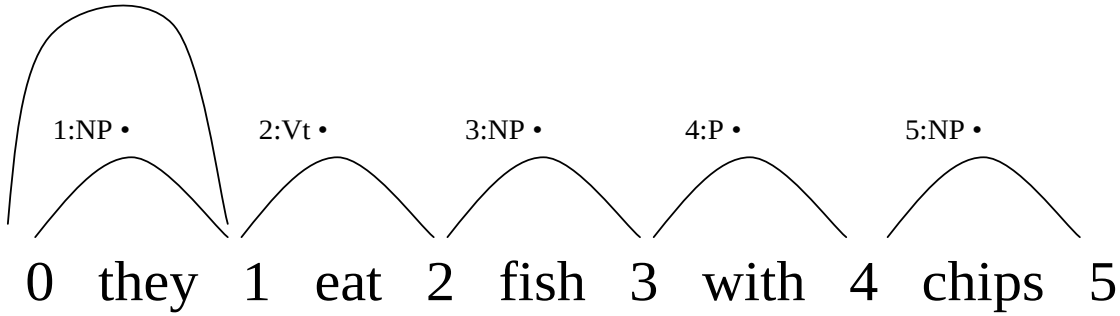
Worked example:

0 they 1 eat 2 fish 3 with 4 chips 5

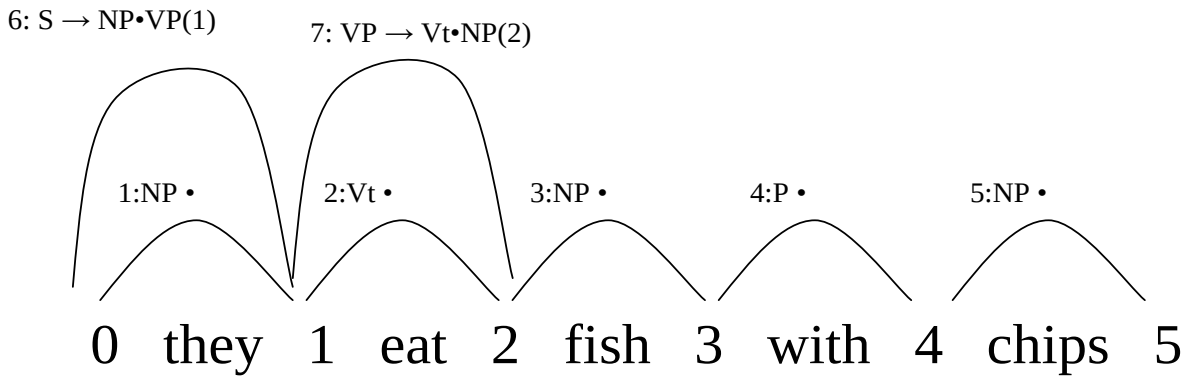


Agenda: 1, 2, 3, 4, 5

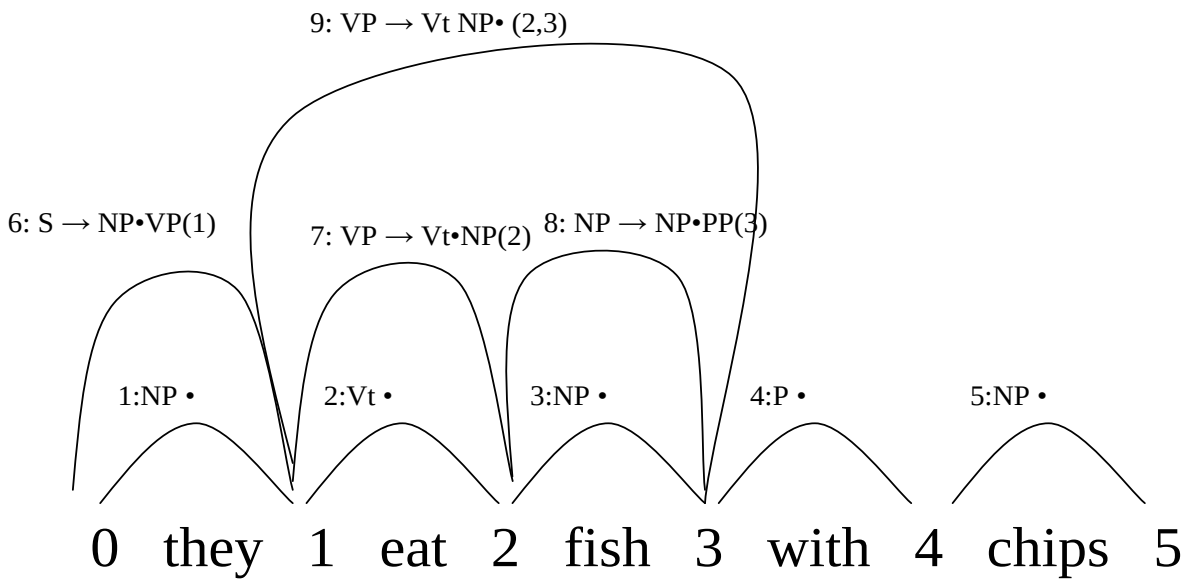
6: S → NP•VP(1)



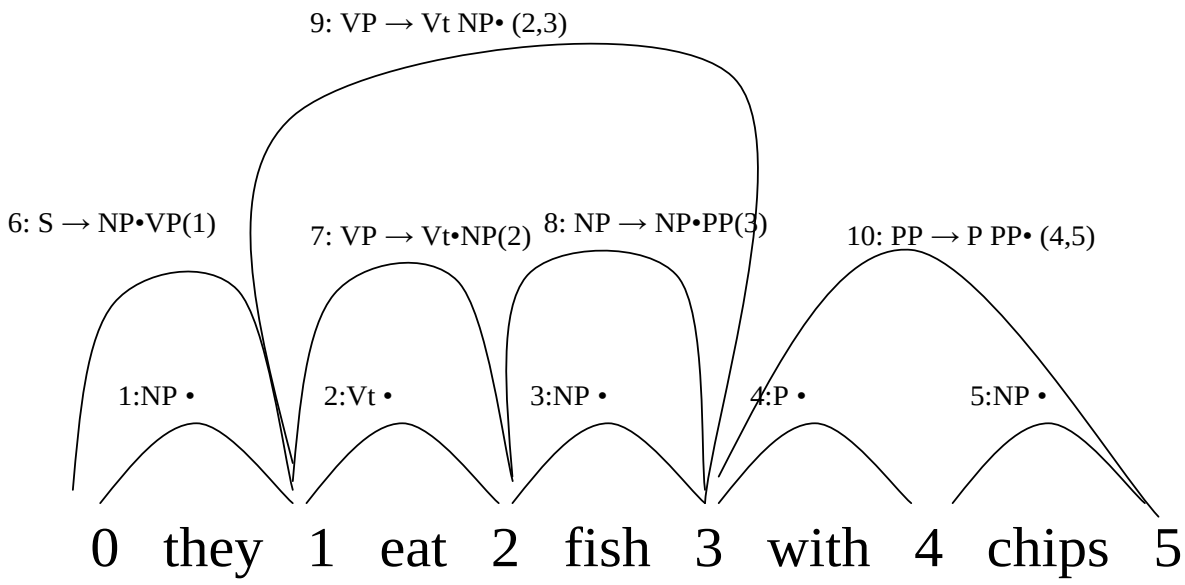
Agenda: 2, 3, 4, 5, 6



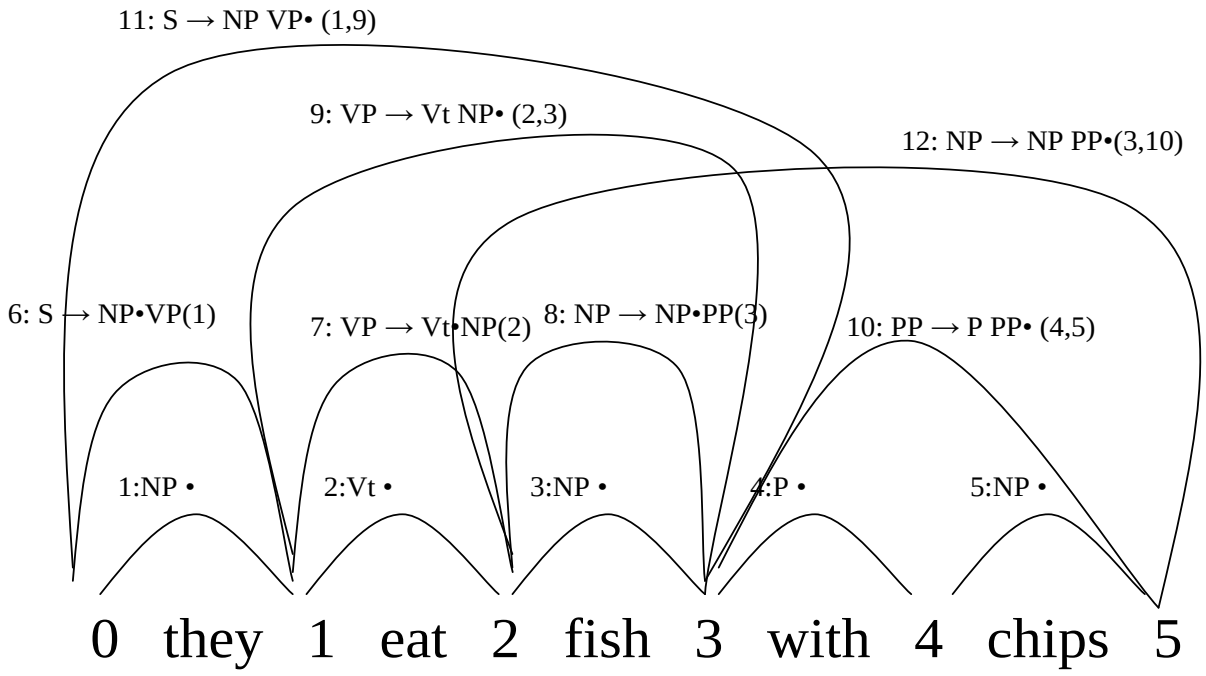
Agenda: 3, 4, 5, 6, 7



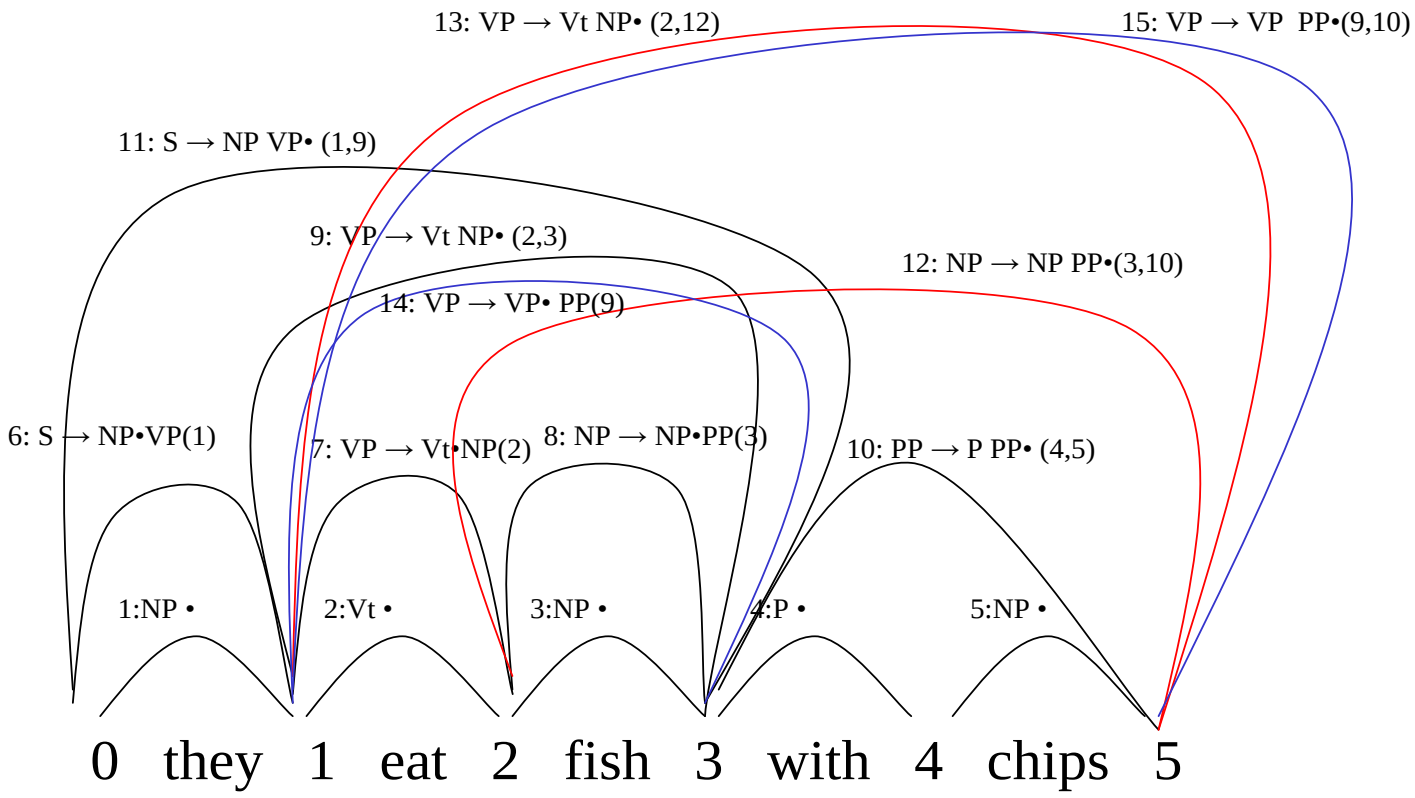
Agenda: 4, 5, 6, 8, 9



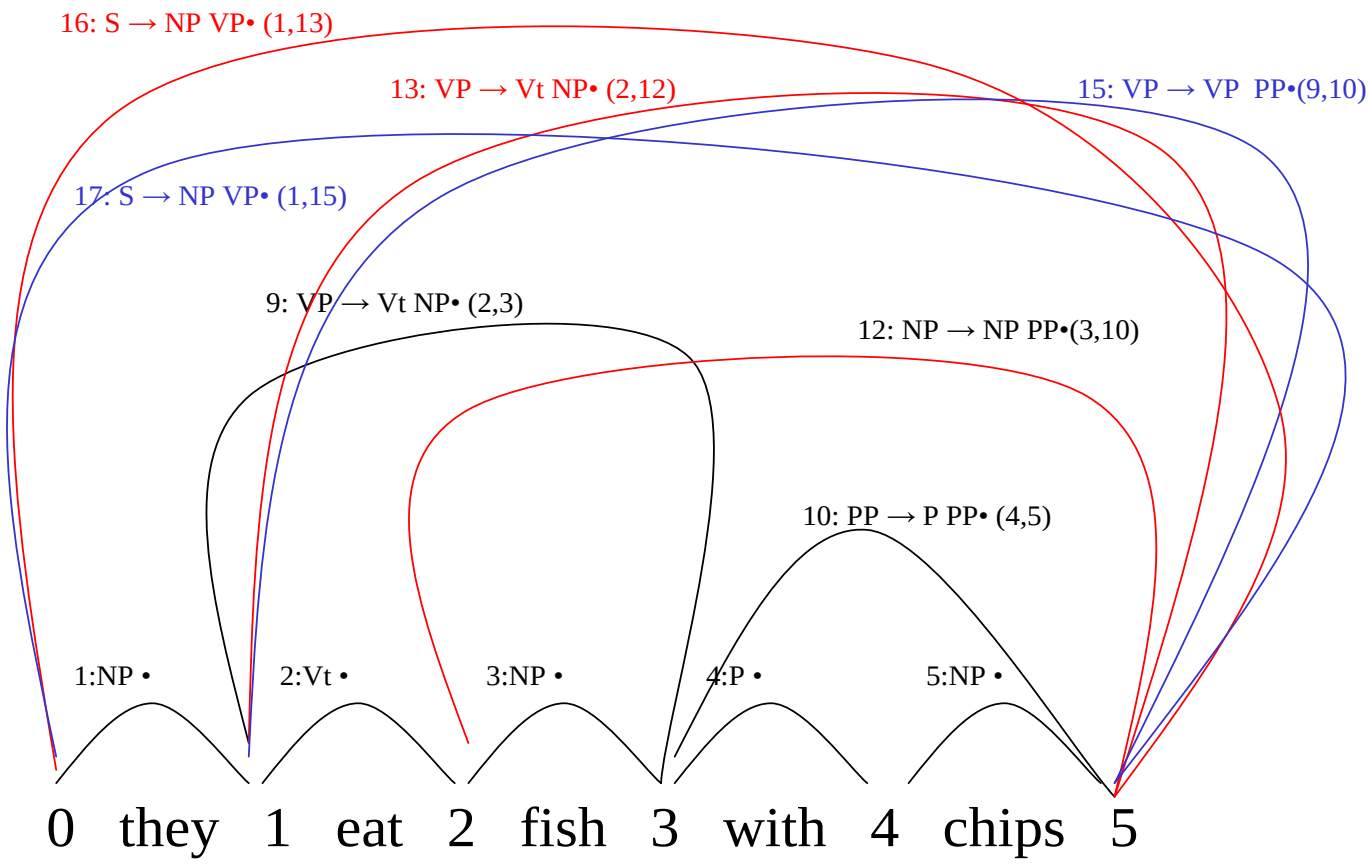
Agenda: 6, 8, 9, 10


11: S → NP VP• (1,9)
9: VP → Vt NP• (2,3)
12: NP → NP PP•(3,10)
6: S → NP•VP(1)
7: VP → Vt•NP(2)
8: NP → NP•PP(3)
10: PP → P PP• (4,5)
1:NP •
2:Vt •
3:NP •
4:P •
5:NP •
0 they 1 eat 2 fish 3 with 4 chips 5

Agenda: 11, 12



Agenda: 11, 12, 13



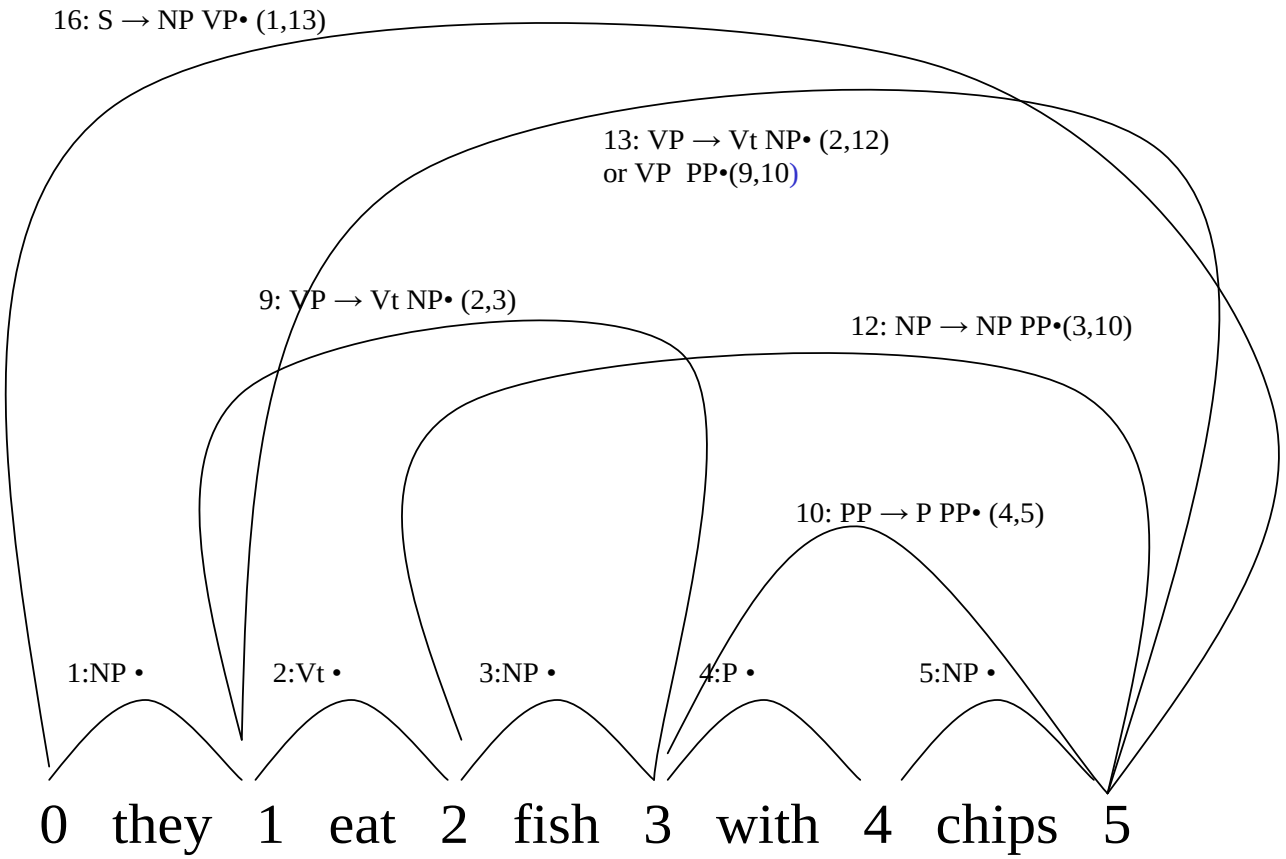
Agenda: empty – look for complete S edges from 0 to 5

PACKING

Note that the bottom-up chart algorithm still contains some redundancy. The fact that there are two ways of building VPs leads to two S edges, and in general will lead to separate edges for every parse. (Note that the CYK algorithm did not have this problem).

But if an NP can combine with one VP from X to Y to form an S, then it is obvious that it can also combine with any other VP from X to Y. At the point at which this move is made the fact that the two VPs have different internal structure is irrelevant, provided that structure does not affect the NP or S involved.

We can use a so-called 'packed' representation to avoid this duplication of effort. We represent all constituents of the same type between the same vertices by a single edge: this may contain the disjunction of the component edges.



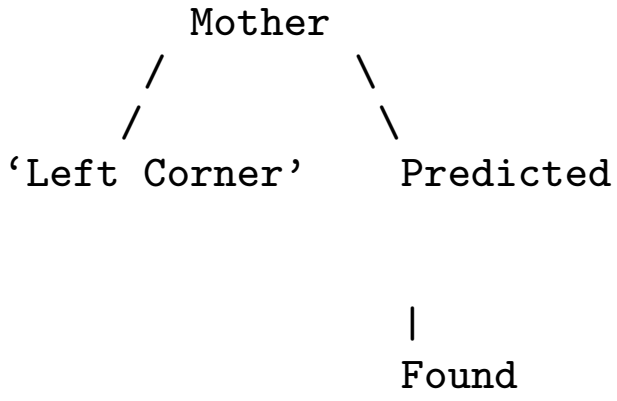
Packed representation

BOTTOM-UP WITH TOP-DOWN FILTERING

Our existing bottom-up algorithm is unintelligent in several ways. For example, every NP found will trigger a hypothesis that it is the start of a new sentence, even if we know that it cannot be:

They can [S[NP fish] •VP...]

If we augment our bottom-up algorithms with information about what constituent we are trying to build top-down, then we can eliminate many of these hypotheses.

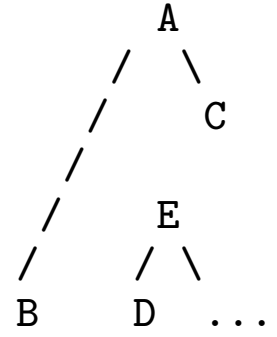


Is the 'found' constituent a possible left corner of 'predicted'?

For Propose, we compile a table from the grammar showing the 'left corner of' relation (NB extra Sigma start symbol to make things tidy).

Sigma	→ S	S	→ NP VP
NP	→ they | fish	VP	→ Aux VP
VP	→ Vi	VP	→ Vt NP
Aux	→ can	Vi	→ fish
Vt	→ can		

```
(<XisLeftCornerOf,Y> : <NP,Sigma>  <NP,S>  <Aux,VP>  <Vi,VP> <Vt,VP>
```




```
Looking for an A
Found a B, looking for a C
Found a D, start a new hypothesis E -> D ...
but only if E is a left corner of C.
```

For a recent paper comparing a variant of this technique to some classical parsing algorithms see: Improved Left-Corner Chart Parsing for Large Context-Free Grammars, by Robert C. Moore, available (via Oxford University subscription) at: `http://www.springerlink.com/content/xm5j782lq582xj71/`

Probabilistic Parsing

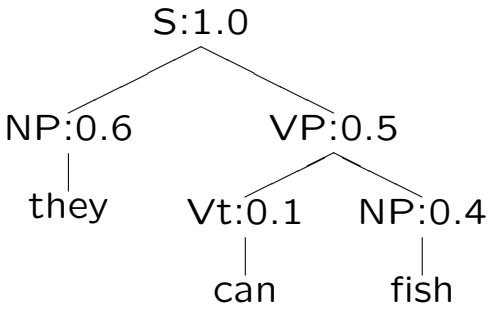
If we have a CFG grammar (or a treebank) we can associate probabilities with the rules. We want to find the tree that maximises P(Tree | Sentence), which here will be the same as maximising P(Tree,Sentence).

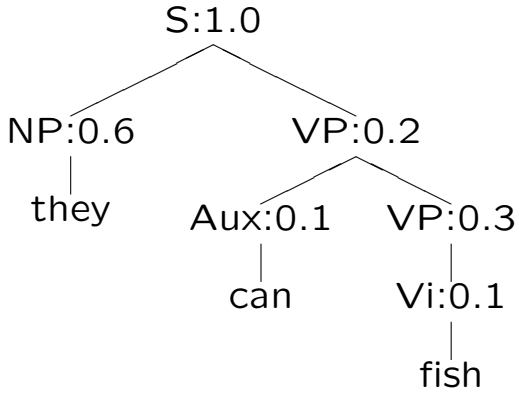
S	$\rightarrow$	NP VP	1.0
NP	$\rightarrow$	they	0.6
NP	$\rightarrow$	fish	0.4
vVP	$\rightarrow$	Aux VP	0.2
VP	$\rightarrow$	Vi	0.3
VP	$\rightarrow$	Vt NP	0.5
Aux	$\rightarrow$	can	0.1 (etc)
Vi	$\rightarrow$	fish	0.1 (etc)
Vt	$\rightarrow$	can	0.1 (etc)

P(Tree,Sentence) can be obtained by multiplying the probabilities of the rules figuring in it:

$$\text{P(Tree,Sentence)} = \prod_i \text{P(Rule}_i\text{)}$$

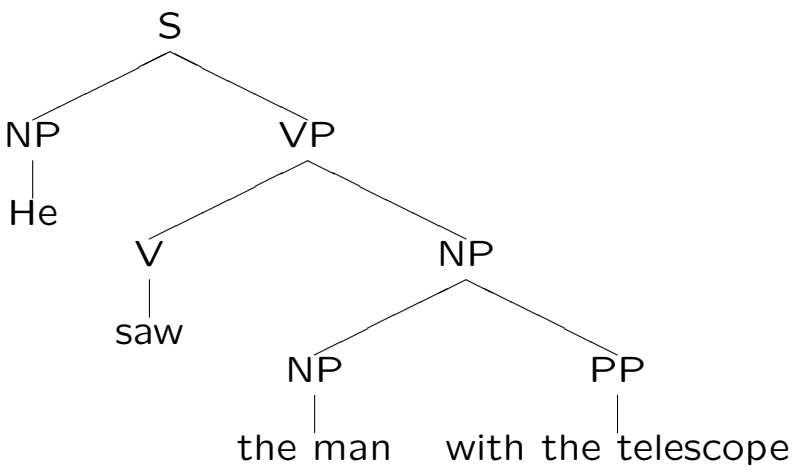
If you have a parsed corpus, $\text{P(A} \rightarrow \text{B)} = \frac{\text{count(A} \rightarrow \text{B)}}{\text{count(A} \rightarrow \text{anything)}}$

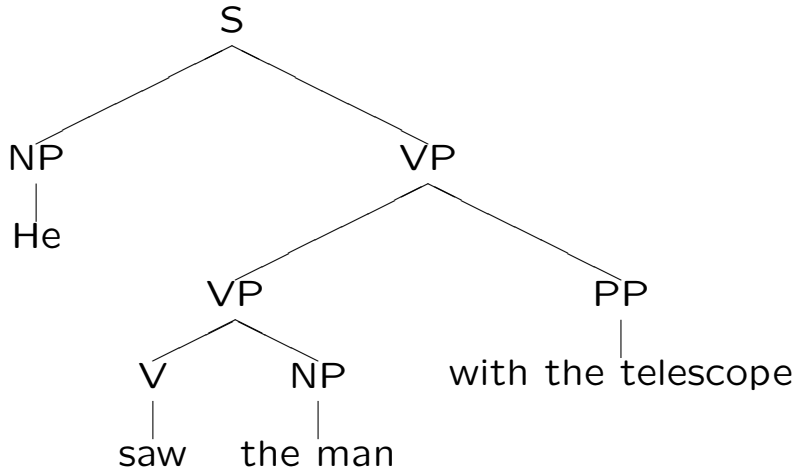




P(Tree1) = 1.0*0.6*0.5*0.1*0.4 = 0.012	P(Tree2)= 1.0*0.6*0.2*0.1*0.3*0.1 = 0.00036

Unfortunately, associating probabilities with rules is not very flexible





The probabilities would prefer one parse ALWAYS, irrespective of the words present:

He ate the pizza with a fork
He ate the pizza with the cheese topping

We need to find some way of making the probabilities more sensitive to the actual words used.