Parsing (and generation)

Syntactic structure in analysis:

- as a step in assigning semantics
- checking grammaticality
- corpus-based investigations, lexical acquisition etc

This lecture:

1. generative grammar
2. a simple context free grammar of a fragment of English
3. random generation
4. simple chart parsing
5. refinements to chart parsing
6. why not FSAs?

Next lecture — beyond simple CFGs

Generative grammar

a formally specified grammar that can generate all and only the acceptable sentences of a natural language

Internal structure:

the big dog slept

can be bracketed

((the (big dog)) slept)

constituent a phrase whose components 'go together' ...

weak equivalence grammars generate the same strings

strong equivalence grammars generate the same strings with same brackets

Context free grammars

1. a set of non-terminal symbols (e.g., S, VP), conventionally written in uppercase;
2. a set of terminal symbols (i.e., the words), conventionally written in lowercase;
3. a set of rules (productions), where the left hand side (the mother) is a single non-terminal and the right hand side is a sequence of one or more non-terminal or terminal symbols (the daughters);

   ```
   S -> NP VP
   V -> fish
   ```

4. a start symbol, conventionally S, which is a member of the set of non-terminal symbols.

Note: exclude empty productions (complicate parsing, arguable linguistic status).

NOT: NP $->$ ϵ

A simple CFG for a fragment of English

```
S -> NP VP
VP -> VP PP
VP -> V
VP -> V NP
VP -> V VP
NP -> NP PP
PP -> P NP

;;; lexicon
V -> can
V -> fish
NP -> fish
NP -> rivers
NP -> pools
NP -> December
NP -> Scotland
NP -> it
NP -> they
P -> in
```

they fish
(S (NP they) (VP (V fish)))

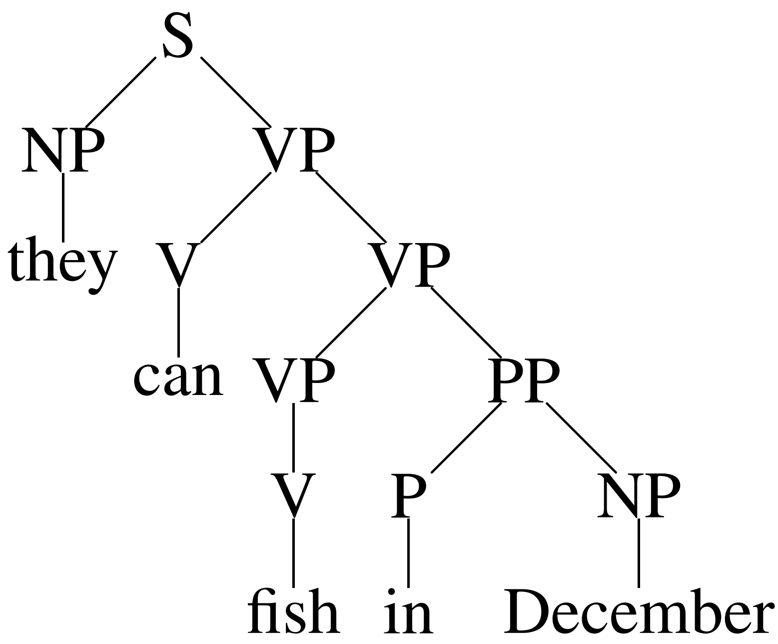
they can fish
(S (NP they) (VP (V can) (VP (V fish))))
(S (NP they) (VP (V can) (NP fish)))

they fish in rivers
(S (NP they) (VP (VP (V fish))
                    (PP (P in) (NP rivers))))

they fish in rivers in December
(S (NP they)
  (VP (VP (V fish))
          (PP (P in) (NP (NP rivers)
               (PP (P in) (NP December))))))
(S (NP they)
   (VP (VP (VP (V fish))
          (PP (P in) (NP (NP rivers))))
        (PP (P in) (NP December))))

Parse trees

they can fish in December




```
(S (NP they)
   (VP (V can)
       (VP (VP (V fish))
           (PP (P in)
               (NP December)))))
```

Using a grammar as a random generator

Expand cat *category sentence-record*:

Let *possibilities* be a set containing all lexical items which match *category* and all rules with left-hand side *category*

If *possibilities* is empty,

then fail

else

> Randomly select a possibility *chosen* from *possibilities*
>
> **If** *chosen* is lexical,
>
> **then** append it to *sentence-record*
>
> **else**
>
> > **expand cat** on each rhs category in *chosen* (left to right) with the updated *sentence-record*
>
> **return** *sentence-record*

Expand cat S ()

possibilities = `S -> NP VP`

chosen = `S -> NP VP`

> **Expand cat** NP ()
>
> possibilities = it, they, fish
>
> chosen = fish
>
> sentence-record = (fish)
>
> **Expand cat** VP (fish)
>
> possibilities = `VP -> V`, `VP -> V VP`, `VP -> V NP`
>
> chosen = `VP -> V`
>
> > **Expand cat** V (fish)
> >
> > possibilities = fish, can
> >
> > chosen = fish
> >
> > sentence-record = (fish fish)

Chart parsing

chart store partial results of parsing

edge representation of a rule application

Edge data structure:

[*id*,*left_vtx*, *right_vtx*,*mother_category*, *dtrs*]

```
. they . can . fish .
0      1     2      3
```

Fragment of chart:

id	l	r	ma	dtrs
3	1	2	V	(can)
4	2	3	NP	(fish)
5	2	3	V	(fish)
6	2	3	VP	(5)
7	1	3	VP	(3 5)
8	1	3	VP	(3 4)

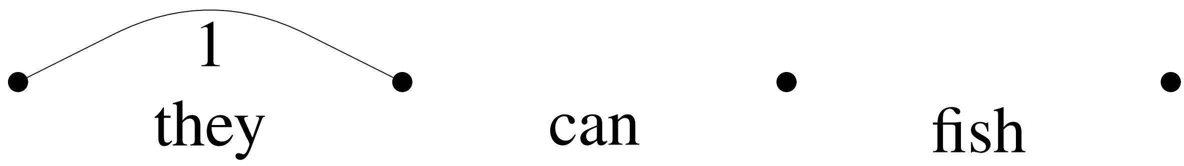
A bottom-up passive chart parser

Parse:
Initialize the chart
For each word *word*, let *from* be left vtx, *to* right vtx and *dtrs* be (*word*)
For each category *category* lexically associated with *word*
Add new edge *from*, *to*, *category*, *dtrs*
Output results for all spanning edges

Add new edge *from*, *to*, *category*, *dtrs*:
Put edge in chart: [*id*,*from*,*to*, *category*,*dtrs*]
For each *rule lhs* $->$ $cat_1 \ldots cat_{n-1}$,*category*
Find sets of contiguous edges
[id_1,$from_1$,to_1, cat_1,$dtrs_1$] $\ldots$
[id_{n-1},$from_{n-1}$,*from*, cat_{n-1},$dtrs_{n-1}$]
(such that $to_1 = from_2$ etc)
For each set of edges,
Add new edge $from_1$, *to*, *lhs*, ($id_1 \ldots id$)

id	l	r	ma	dtrs
1	0	1	NP	(they)
2	1	2	V	(can)
3	1	2	VP	(2)
4	0	2	S	(1 3)
5	2	3	V	(fish)
6	2	3	VP	(5)
7	1	3	VP	(2 6)
8	0	3	S	(1 7)
9	2	3	NP	(fish)
10	1	3	VP	(2 9)
11	0	3	S	(1 10)

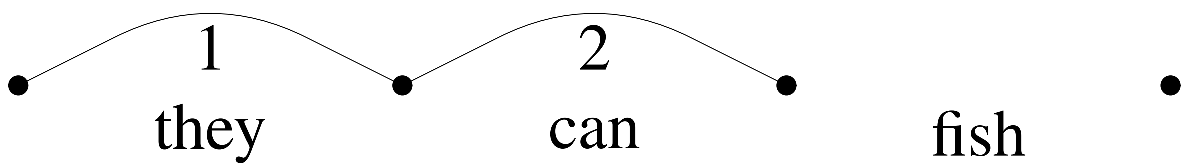
Parse
word = they
categories = NP
Add new edge 0, 1, NP, (they)



Matching grammar rules are:

```
VP -> V NP
PP -> P NP
```

No matching edges corresponding to V or P
word = can
categories = V
Add new edge 1, 2, V, (can)

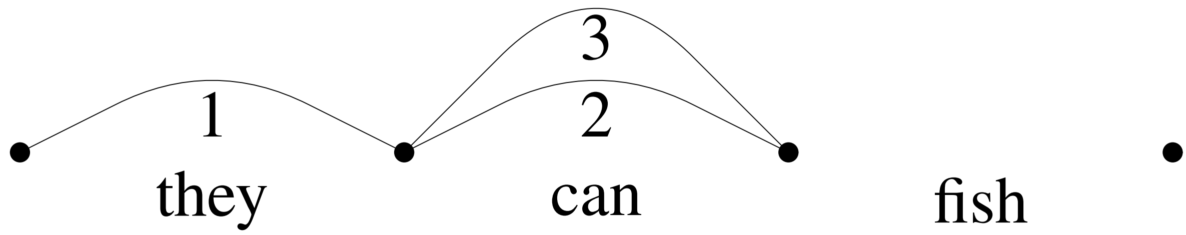


Matching grammar rules are:

```
VP -> V
```

set of edge lists = $\{(2)\}$

Add new edge 1, 2, VP, (2)

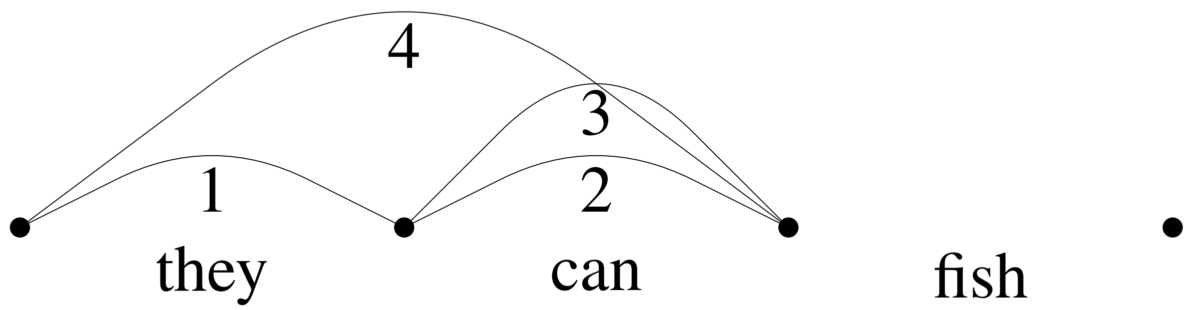


Matching grammar rules are:

```
S -> NP VP
VP -> V VP
```

set of edge lists corresponding to NP VP
$= \{(1, 3)\}$

Add new edge 0, 2, S, (1, 3)

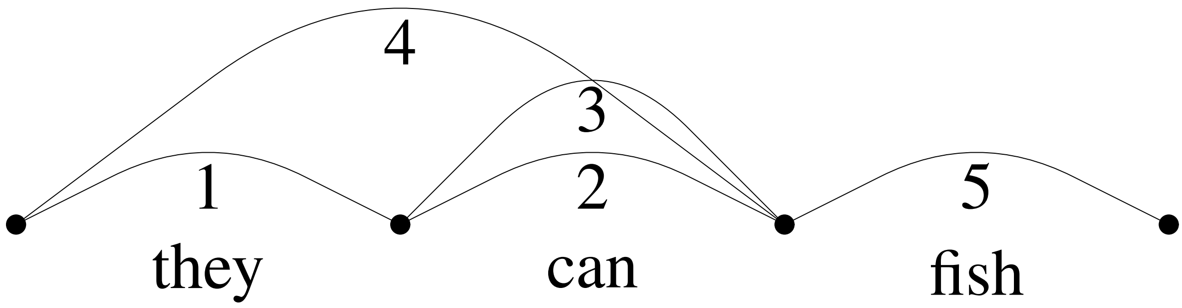


No matching grammar rules for S

No edges matching V VP

word = fish
categories = V, NP

Add new edge 2, 3, V, (fish)

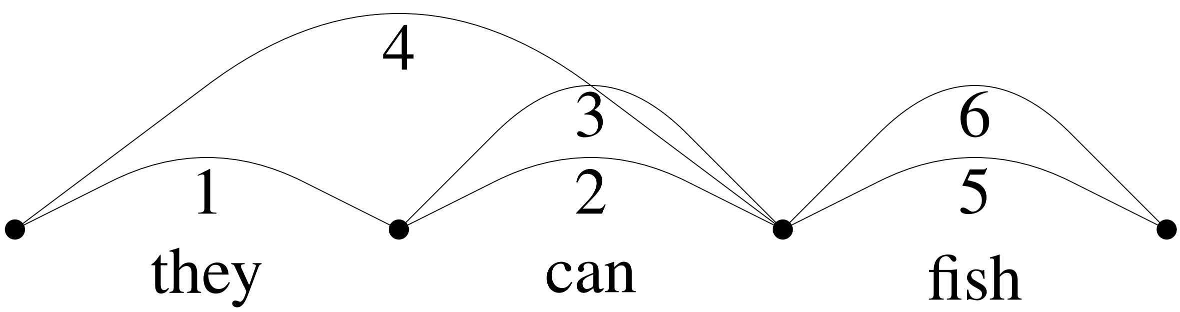


Matching grammar rules are:

```
VP -> V
```

set of edge lists $= \{(5)\}$

Add new edge 2, 3, VP, (5)

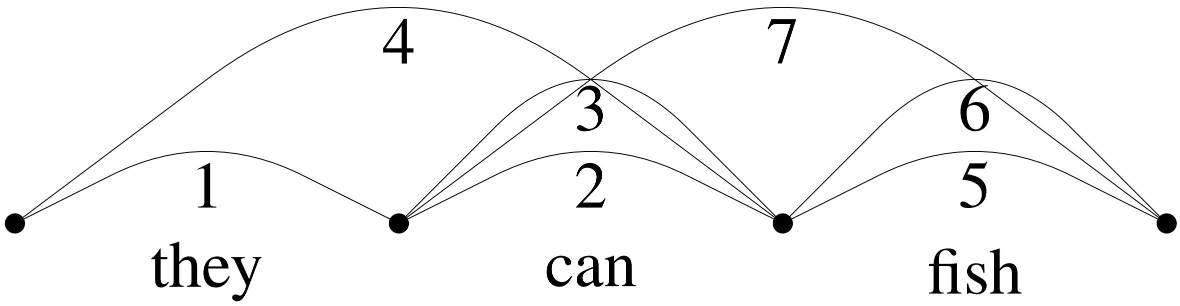


Matching grammar rules are:

```
S -> NP VP
VP -> V VP
```

No edges match NP
set of edge lists for V VP $= \{(2, 6)\}$

Add new edge 1, 3, VP, (2, 6)

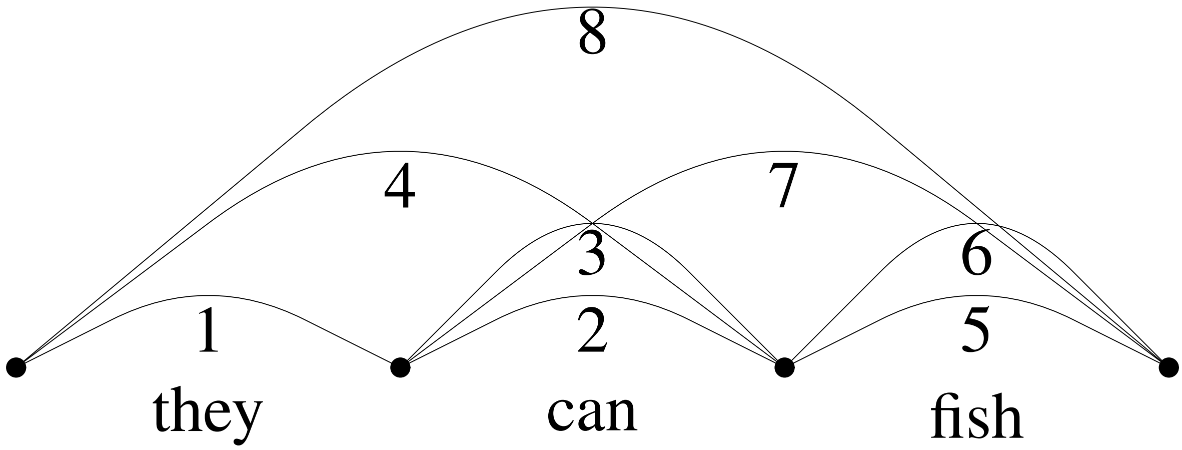


Matching grammar rules are:

```
S -> NP VP
VP -> V VP
```

set of edge lists for NP VP $= \{(1, 7)\}$

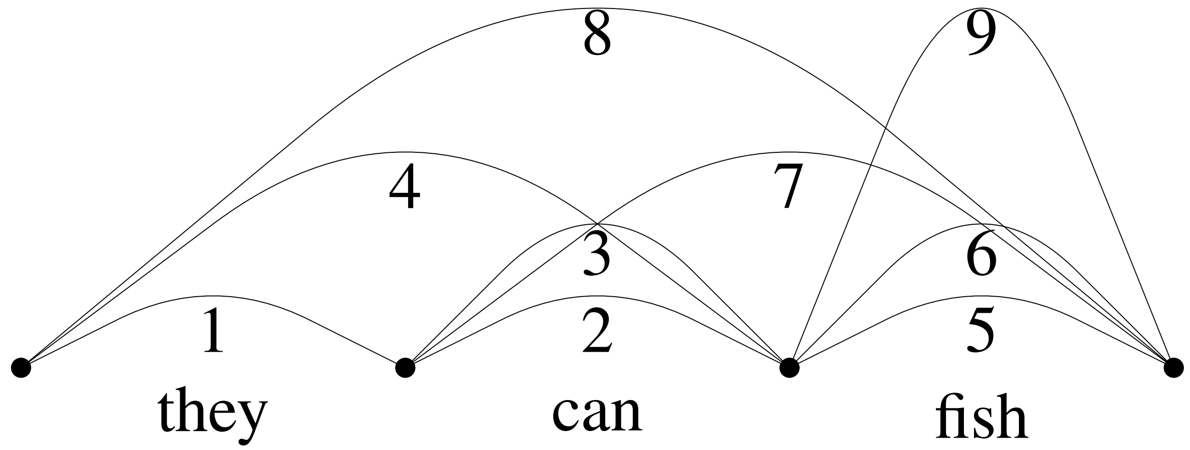
Add new edge 0, 3, S, (1, 7)



No matching grammar rules for S

No edges matching V

Add new edge 2, 3, NP, (fish)

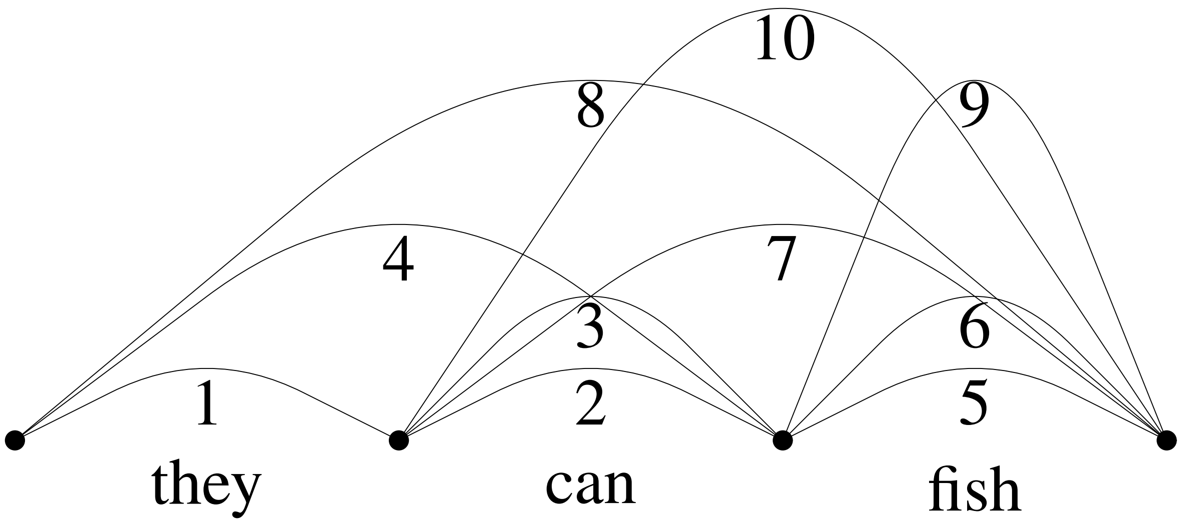


Matching grammar rules are:

```
VP -> V NP
PP -> P NP
```

set of edge lists corresponding to V NP = $\{(2, 9)\}$

Add new edge 1, 3, VP, (2, 9)

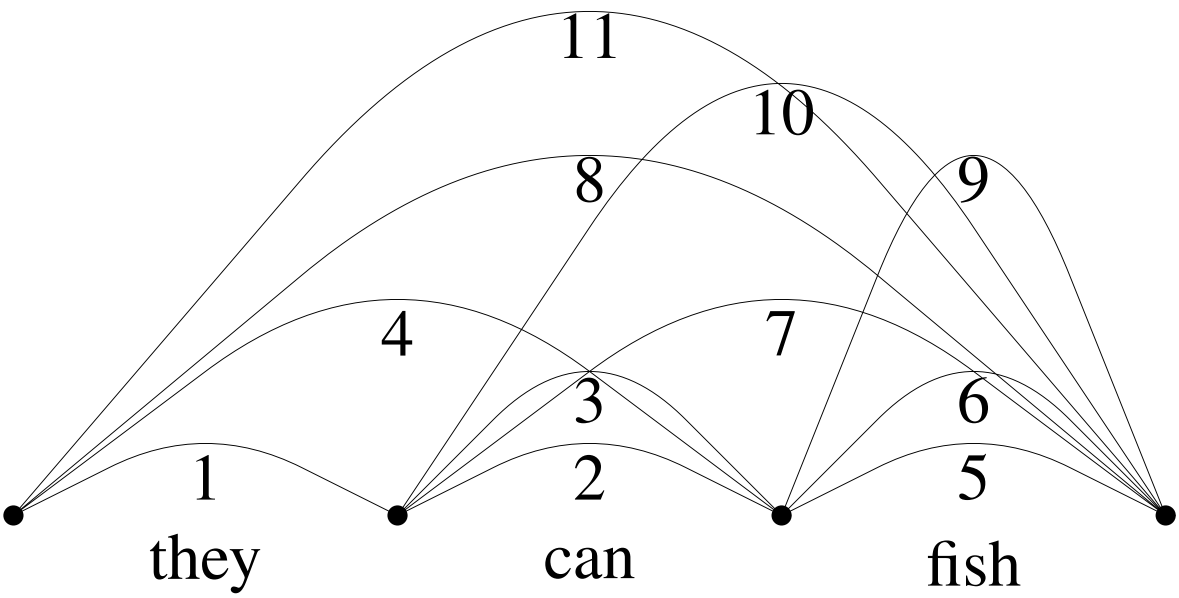


Matching grammar rules are:

```
S -> NP VP
VP -> V VP
```

set of edge lists corresponding to NP VP $= \{(1, 10)\}$

Add new edge 0, 3, S, (1, 10)



No matching grammar rules for S

No edges corresponding to V VP

No edges corresponding to P NP

No further words in input

Spanning edges are 8 and 11: Output results for 8

```
(S (NP they) (VP (V can)
                 (VP (V fish))))
```

Output results for 11

```
(S (NP they) (VP (V can)
                 (NP fish)))
```

Note: sample chart parsing code in Java is downloadable from the course web page.

Packing

- exponential number of parses means exponential time
- body can be cubic time: don't add equivalent edges

about to add:

[*id*,*l_vtx*, *right_vtx*,*ma_cat*, *dtrs*]

and there is an existing edge:

[*id-old*,*l_vtx*, *right_vtx*,*ma_cat*, *dtrs-old*]

we simply modify the old edge to record the new dtrs:

[*id-old*,*l_vtx*, *right_vtx*,*ma_cat*, *dtrs-old* $\sqcup$ *dtrs*]

id	l	r	ma	dtrs
1	0	1	NP	{(they)}
2	1	2	V	{(can)}
3	1	2	VP	{(2)}
4	0	2	S	{(1 3)}
5	2	3	V	{(fish)}
6	2	3	VP	{(5)}
7	1	3	VP	{(2 6)}
8	0	3	S	{(1 7)}
9	2	3	NP	{(fish)}

instead of:

```
10  1   3   VP   {(2 9)}
```

we have:

```
7   1   3   VP   {(2 6), (2 9)}
```

Active chart parsing

id	l	r	ma	exp	dtrs
1	0	1	NP		(they)
2	0	1	S	VP	(1 ?)
3	0	1	NP	PP	(1,?)
4	1	2	V		(fish)
5	1	2	VP	PP	(4,?)
6	1	2	VP		(4)
7	1	2	VP	NP	(4,?)
8	1	2	VP	VP	(4,?)
9	0	2	S		(2,6)

- store partial rule applications
- record expected input as well as seen
- one active can create more than one passive. e.g., *they fish in Scotland* — edge 2 completed by *fish* and *fish in Scotland*. NP is combined with rule once not twice.
- active edges can be packed

Ordering the search space

- agenda: order edges in chart by priority
- top-down parsing: predict possible edges

Producing n-best parses:

- manual weight assignment
- probabilistic CFG — trained on a treebank
  - automatic grammar induction
  - automatic weight assignment to existing grammar
- beam-search

Why not FSA?

centre-embedding:

$A \rightarrow \alpha A \beta$

and which generate grammars of the form $a^n b^n$.

For instance:

the students the police arrested complained

However:

? the students the police the journalists criticised arrested complained

Limits on human memory / processing ability

More importantly for practical application:

1. FSM grammars are very redundant: difficult to build and maintain
2. FSM grammars don't support composition of semantics

but FSMs useful for:

1. tokenizers (dates, times etc)
2. named entity recognition in information extraction etc
3. approximating CFGs in speech recognition