Natural Language Processing: Part II Overview of Natural Language Processing (L90): ACS Lecture 4

Paula Buttery (Materials by Ann Copestake)

Computer Laboratory
University of Cambridge

October 2018

Parsing

Syntactic structure in analysis:

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

Next lecture — alternative to 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);
2. a set of terminal symbols (i.e., the words);
3. a set of rules (productions), where the LHS (mother) is a single non-terminal and the RHS is a sequence of one or more non-terminal or terminal symbols (daughters);

   ```
   S -> NP VP
   V -> fish
   ```
4. a start symbol, conventionally S, which is a non-terminal.

Exclude empty productions, NOT e.g.:

$$\text{NP} -> \epsilon$$

A simple CFG for a fragment of English

rules

```
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
```

Analyses in the simple CFG

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))))

Analyses in the simple CFG

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))))

Analyses in the simple CFG

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))))

Structural ambiguity without lexical ambiguity

they fish in rivers in December

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

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

Consider: "They fish in rivers in Alaska"

Structural ambiguity without lexical ambiguity

they fish in rivers in December

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

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

Consider: "They fish in rivers in Alaska"

Structural ambiguity without lexical ambiguity

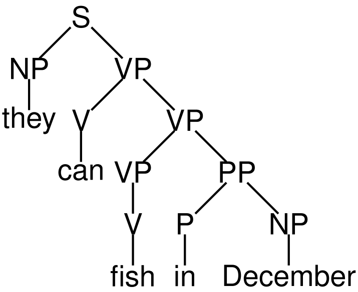
they fish in rivers in December

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

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

Consider: "They fish in rivers in Alaska"

Parse trees




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

Chart parsing

A dynamic programming algorithm (memoisation):

chart store partial results of parsing in a vector

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
5     2    3    V     (fish)
6     2    3    VP    (5)
7     1    3    VP    (3 6)
```

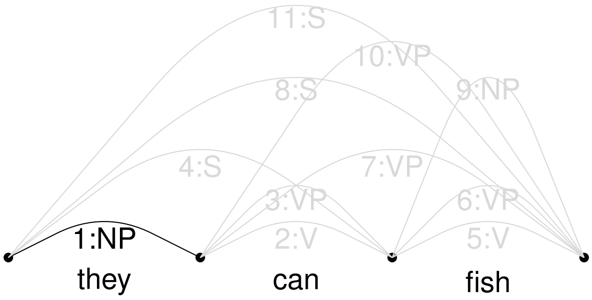
A bottom-up passive chart parser

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

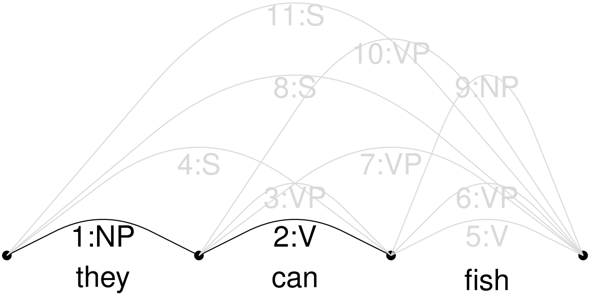
Inner function

Add new edge *from*, *to*, *category*, *dtrs*:
Put edge in chart: [*id*,*from*,*to*, *category*,*dtrs*]
For each *rule lhs* $\rightarrow$ $cat_1 \ldots cat_{n-1}$,*category*
Find sets of contiguous edges
[id_1,$from_1$,to_1, cat_1,$dtrs_1$] ...
[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*, ($id_1 \ldots id$)

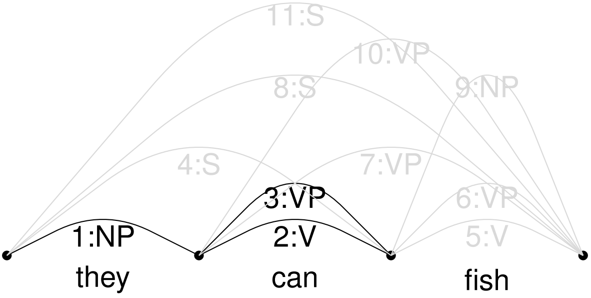
Bottom up parsing: edges



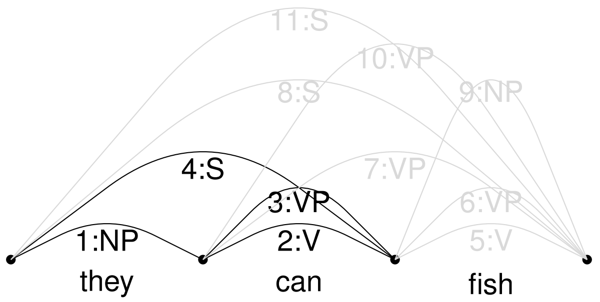
Bottom up parsing: edges



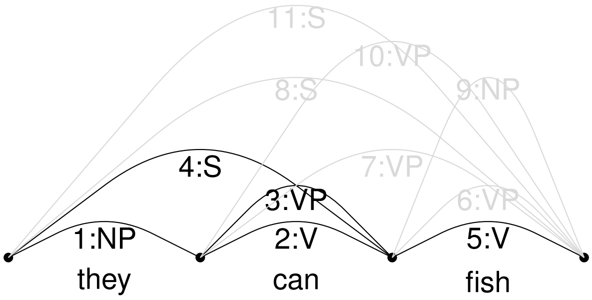
Bottom up parsing: edges



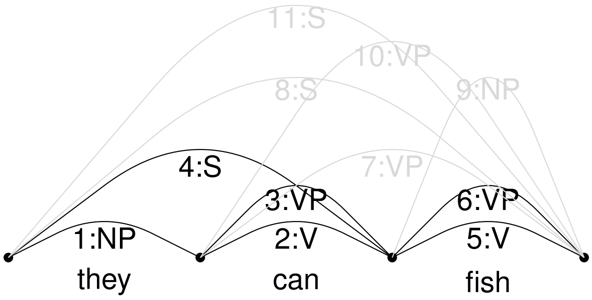
Bottom up parsing: edges



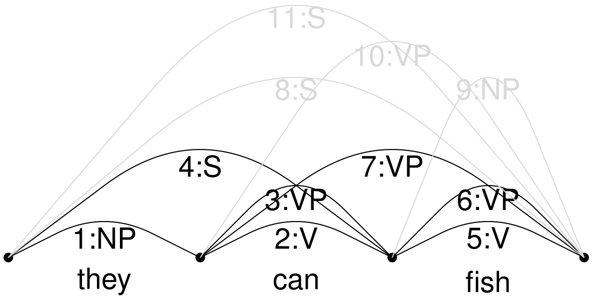
Bottom up parsing: edges



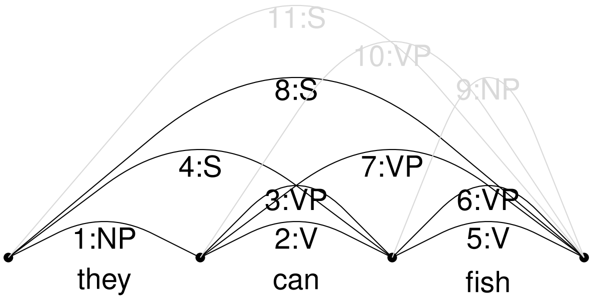
Bottom up parsing: edges



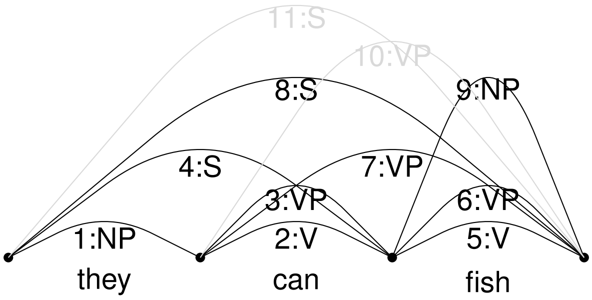
Bottom up parsing: edges



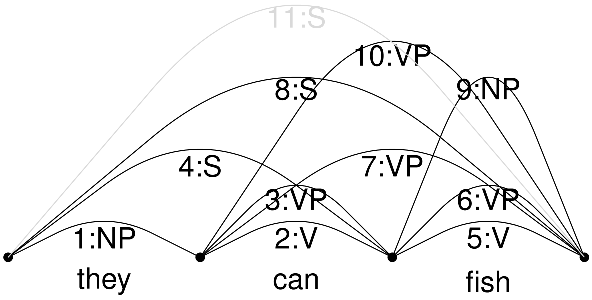
Bottom up parsing: edges



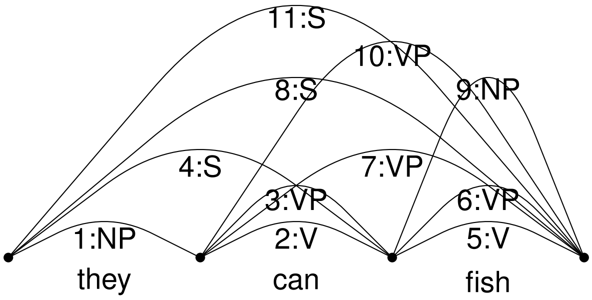
Bottom up parsing: edges



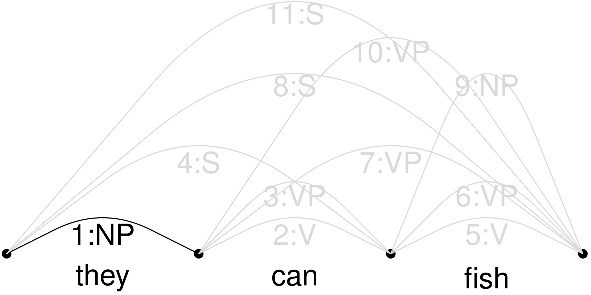
Bottom up parsing: edges



Bottom up parsing: edges

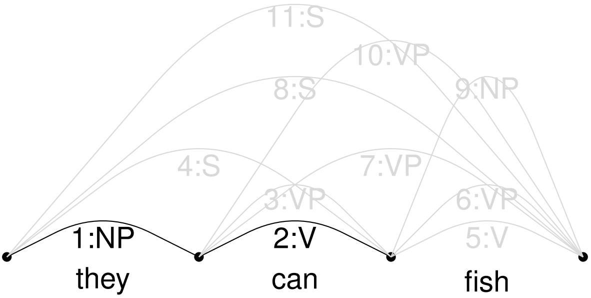


Parse construction



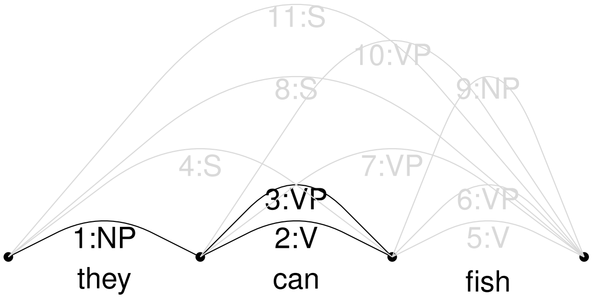
word = they, categories = {NP}
Add new edge 0, 1, NP, (they)
Matching grammar rules: {VP→V NP, PP→P NP}
No matching edges corresponding to V or P

Parse construction



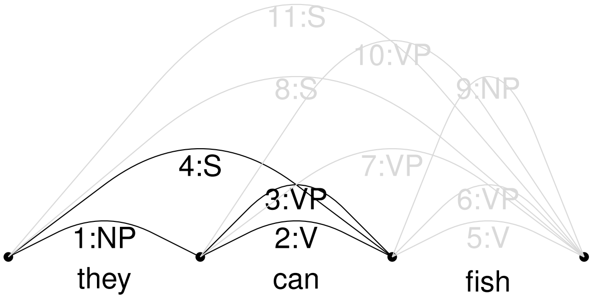
word = can, categories = {V}
Add new edge 1, 2, V, (can)
Matching grammar rules: {VP→V}
recurse on edges {(2)}

Parse construction



Add new edge 1, 2, VP, (2)
Matching grammar rules: {S→NP VP, VP→V VP}
recurse on edges {(1,3)}

Parse construction



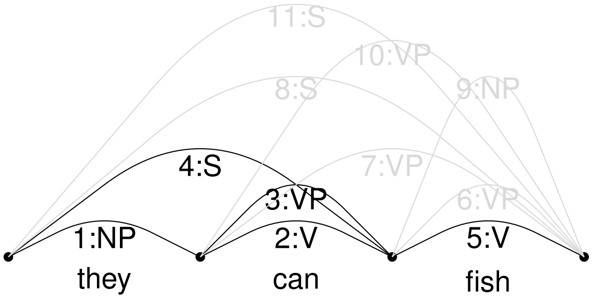
Add new edge 0, 2, S, (1, 3)

No matching grammar rules for S

Matching grammar rules: {S→NP VP, VP→V VP}

No edges for V VP

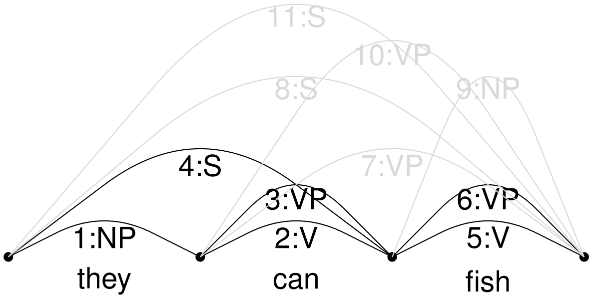
Parse construction



word = fish, categories = {V, NP}
Add new edge 2, 3, V, (fish)
Matching grammar rules: {VP→V}
recurse on edges {(5)}

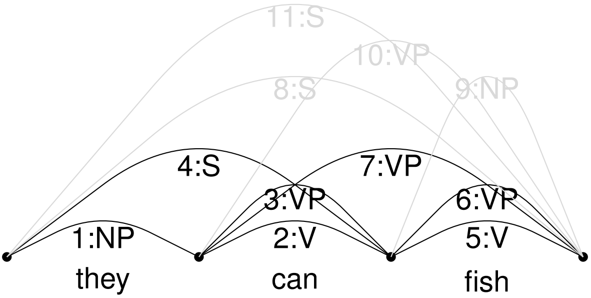
NB: fish as V

Parse construction



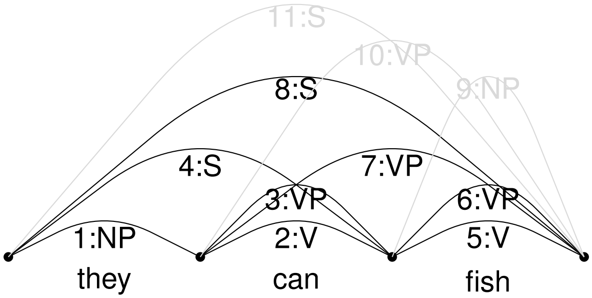
Add new edge 2, 3, VP, (5)
Matching grammar rules: {S →NP VP, VP →V VP}
No edges match NP
recurse on edges for V VP: {(2,6)}

Parse construction



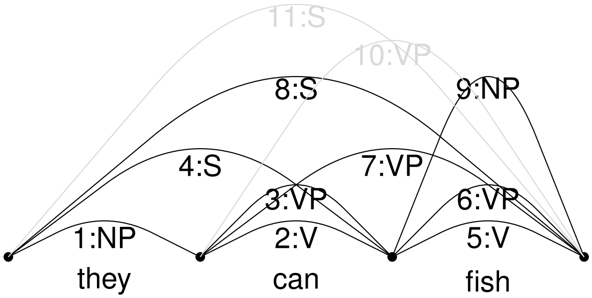
Add new edge 1, 3, VP, (2, 6)
Matching grammar rules: {S→NP VP, VP→V VP}
recurse on edges for NP VP: {(1,7)}

Parse construction



Add new edge 0, 3, S, (1, 7)
No matching grammar rules for S
Matching grammar rules: {S→NP VP, VP →V VP}
No edges matching V

Parse construction



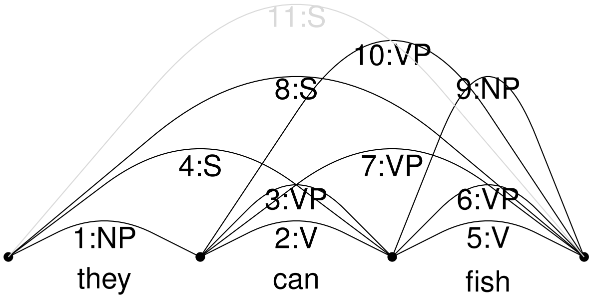
Add new edge 2, 3, NP, (fish)

NB: fish as NP

Matching grammar rules: {VP→V NP, PP→P NP}

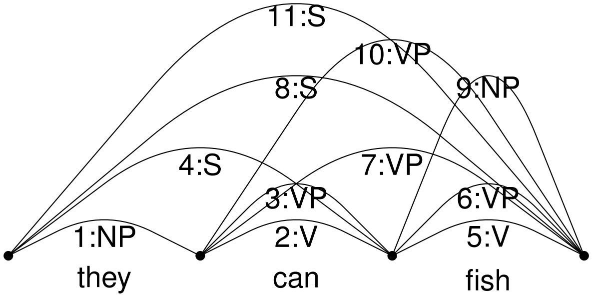
recurse on edges for V NP {(2,9)}

Parse construction



Add new edge 1, 3, VP, (2, 9)
Matching grammar rules: {S→NP VP, VP→V VP}
recurse on edges for NP VP: {(1, 10)}

Parse construction



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

No matching grammar rules for S

Matching grammar rules: {S→NP VP, VP→V VP}

No edges corresponding to V VP

Matching grammar rules: {VP→V NP, PP→P NP}

No edges corresponding to P NP

Output results for spanning edges

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)))
```

How does this compare to other parsing methods you know about?

Output results for spanning edges

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)))
```

How does this compare to other parsing methods you know about?

Packing

- exponential number of parses means exponential time
- body can be cubic time: don't add equivalent edges as whole new edges
- *dtrs* is a set of lists of edges (to allow for alternatives)

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* ∪ *dtrs*]

and do not recurse on it: never need to continue computation with a packable edge.

Packing example

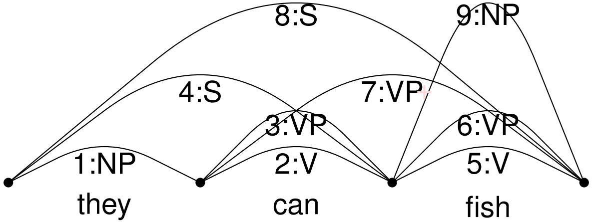
1	0	1	NP	{(they)}
2	1	2	V	{(can)}
3	1	2	VP	{(2)}
4	0	2	S	{(1 3)}
5	2	3	V	{(fish)}
6	2	3	VP	{(5)}
7	1	3	VP	{(2 6)}
8	0	3	S	{(1 7)}
9	2	3	NP	{(fish)}

Instead of edge `10 1 3 VP {(2 9)}`

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

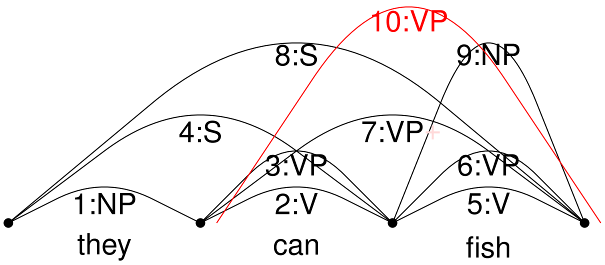
and we're done

Packing example



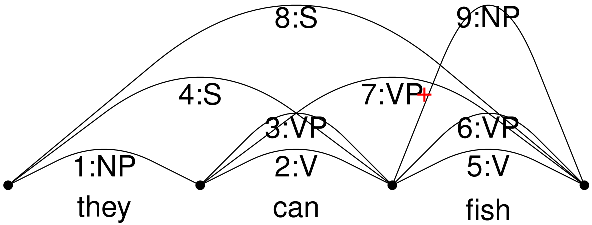
Both spanning results can now be extracted from edge 8.

Packing example



Both spanning results can now be extracted from edge 8.

Packing example



Both spanning results can now be extracted from edge 8.

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:

$$A \rightarrow \alpha A \beta$$

generate grammars of the form $a^n b^n$.
For instance:

the students the police arrested complained

However, limits on human memory / processing ability:

? the students the police the journalists criticised arrested complained

More importantly:

1. FSM grammars are extremely redundant
2. FSM grammars don't support composition of semantics

Why not FSA?

centre-embedding:

$$A \to \alpha A \beta$$

generate grammars of the form $a^n b^n$.
For instance:

the students the police arrested complained

However, limits on human memory / processing ability:

? the students the police the journalists criticised arrested complained

More importantly:

1. FSM grammars are extremely redundant
2. FSM grammars don't support composition of semantics

Overgeneration in atomic category CFGs

- agreement: subject verb agreement. e.g., *they fish*, *it fishes*, **it fish*, **they fishes*. * means ungrammatical
- case: pronouns (and maybe *who/whom*) e.g., *they like them*, **they like they*

```
S -> NP-sg-nom VP-sg          NP-sg-nom -> he
S -> NP-pl-nom VP-pl          NP-sg-acc -> him
VP-sg -> V-sg NP-sg-acc       NP-sg-nom -> fish
VP-sg -> V-sg NP-pl-acc       NP-pl-nom -> fish
VP-pl -> V-pl NP-sg-acc       NP-sg-acc -> fish
VP-pl -> V-pl NP-pl-acc       NP-pl-acc -> fish
```

BUT: very large grammar, misses generalizations, no way of saying when we don't care about agreement.

Overgeneration in atomic category CFGs

- agreement: subject verb agreement. e.g., *they fish*, *it fishes*, **it fish*, **they fishes*. * means ungrammatical
- case: pronouns (and maybe *who/whom*) e.g., *they like them*, **they like they*

```
S -> NP-sg-nom VP-sg          NP-sg-nom -> he
S -> NP-pl-nom VP-pl          NP-sg-acc -> him
VP-sg -> V-sg NP-sg-acc       NP-sg-nom -> fish
VP-sg -> V-sg NP-pl-acc       NP-pl-nom -> fish
VP-pl -> V-pl NP-sg-acc       NP-sg-acc -> fish
VP-pl -> V-pl NP-pl-acc       NP-pl-acc -> fish
```

BUT: very large grammar, misses generalizations, no way of saying when we don’t care about agreement.

Subcategorization

- intransitive vs transitive etc
- verbs (and other types of words) have different numbers and types of syntactic arguments:

  **Kim adored*
  **Kim gave Sandy*
  **Kim adored to sleep*
  Kim liked to sleep
  **Kim devoured*
  Kim ate
- Subcategorization is correlated with semantics, but not determined by it.

Overgeneration because of missing subcategorization

Overgeneration:

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

- intransitive takes no following arguments (complements)
- simple transitive takes one NP complement
- *like* may be a simple transitive or take an infinitival complement, etc

Long-distance dependencies

1. which problem did you say you don't understand?
2. who do you think Kim asked Sandy to hit?
3. which kids did you say were making all that noise?

'gaps' (underscores below)

1. which problem did you say you don't understand _?
2. who do you think Kim asked Sandy to hit _?
3. which kids did you say _ were making all that noise?

In 3, the verb *were* shows plural agreement.

* what kid did you say _ were making all that noise?

The gap filler has to be plural.

Context-free grammar and language phenomena

- CFGs can encode long-distance dependencies
- Language phenomena that CFGs cannot model (without a bound) are unusual — probably none in English.
- BUT: CFG modelling for English or another NL could be trillions of rules
- Enriched formalisms: CFG equivalent or greater power