Natural Language Processing 1

Lecture 3: Modelling structure: Morphology and Syntax

Katia Shutova

ILLC
University of Amsterdam

8 November

Outline.

Stems and affixes

- morpheme: the minimal information carrying unit
- affix: morpheme which only occurs in conjunction with other morphemes
- words made up of stem (more than one for compounds) and zero or more affixes.
  e.g., *dog+s*, *book+shop+s*
- *slither*, *slide*, *slip* etc have somewhat similar meanings, but *sl-* not a morpheme.

Affixation

- **suffix**: *dog +s*, *truth +ful*
- **prefix**: *un+ wise* (derivational only)
- **infix**: Arabic stem *k_t_b*: *kataba* (he wrote); *kotob* (books)
  In English: *sang* (stem *sing*): not productive
  e.g., (maybe) *absobloodylutely*
- **circumfix**: not in English
  German *ge+kauf+t* (stem *kauf*, affix *ge-t*)

Productivity

productivity: whether affix applies generally, whether it applies to new words
sing, *sang*, *sung*
ring, *rang*, *rung*
BUT: *ping*, *pinged*, *pinged*
So this infixation pattern is not productive:
sing, *ring* are irregular

Productivity

productivity: whether affix applies generally, whether it applies to new words
sing, *sang*, *sung*
ring, *rang*, *rung*
BUT: *ping*, *pinged*, *pinged*
So this infixation pattern is not productive:
sing, *ring* are irregular

Inflectional morphology

- e.g., plural suffix *+s*, past participle *+ed*
- sets slots in some paradigm
  e.g., tense, aspect, number, person, gender, case
- inflectional affixes are not combined in English
- generally fully productive (except irregular forms)
  e.g., *texted*

Derivational morphology

- e.g., *un-*, *re-*, *anti-*, *-ism*, *-ist* etc
- broad range of semantic possibilities, may change part of speech
- indefinite combinations
  e.g., *antiantidisestablishmentarianism*
  anti-anti-dis-establish-ment-arian-ism
- generally semi-productive: e.g., *escapee*, *textee*, *?dropee*, *?snoree*, **cricketee* (* and ?)
- zero-derivation: e.g. *tango*, *waltz*

Guess the structure...

- ruined
- settlement
- inventive
- archive
- unionised

Guess the structure...

- ruined
- settlement
- inventive
- archive
- unionised

Guess the structure...

- ruined
- settlement
- inventive
- archive
- unionised

Guess the structure...

- ruined
- settlement
- inventive
- archive
- unionised

Guess the structure...

- ruined
- settlement
- inventive
- archive
- unionised

Internal structure and ambiguity

Morpheme ambiguity: stems and affixes may be individually ambiguous: e.g. *dog* (noun or verb), *+s* (plural or 3persg-verb)

Structural ambiguity: e.g., *shorts* or *short -s*
unionised could be *union -ise -ed* or *un- ion -ise -ed*

Bracketing: *un- ion -ise -ed*

- *((un- ion) -ise) -ed
- *un- ((ion -ise) -ed)*

Using morphological processing in NLP

- compiling a full-form lexicon
- stemming for IR (not linguistic stem)
- lemmatization, i.e. morphological analysis:
  - finding stems and affixes as a precursor to parsing (often inflections only)
- generation
  - Morphological processing may be bidirectional: i.e., parsing and generation.

    ```
    party + PLURAL <-> parties
    sleep + PAST_VERB <-> slept
    ```

Compiling a full form lexicon

run
runs
ran
running

Compiling a full form lexicon

run
runs
ran
running

Бегаю
Бегу
Бегаешь
Бежишь
Бегает
Бежит
Бегаем
Бежим
Бегаете
Бежите
Бегают
Бегут

Бегал
Бежал
Побежал
Бегала
Бежала
Побежала
Бегало
Бежало
Побежало
Бегали
Бежали
Побежали
Бегай
Беги
Побеги
Бегайте
Бегите
Побегите

Побегу
Побежишь
Побежит
Побежим
Побежите
Побегут
Бегущий
Бежавший
Бежавшая
Бегущая
Бегущее
Бежавшее
Побежавший
Побежавшая
Побежавшее
Побежав
Побегав
Бегая

Using morphological processing in NLP

- compiling a full-form lexicon
- stemming for IR (not linguistic stem)
- lemmatization, i.e. morphological analysis:
  - finding stems and affixes as a precursor to parsing (often inflections only)
- generation
  - Morphological processing may be bidirectional: i.e., parsing and generation.

    ```
    party + PLURAL <-> parties
    sleep + PAST_VERB <-> slept
    ```

Morphological processing

Surface form mapped to stem(s) and affixes (or abstractions of affixes):

Segmentation	*pinged / ping-ed*
Lemmatisation	*pinged / ping PAST_VERB* *pinged / ping PSP_VERB* *sang / sing PAST_VERB* *sung / sing PSP_VERB*

Lexical requirements for morphological processing

- affixes, plus the associated information conveyed by the affix

```
ed PAST_VERB
ed PSP_VERB
s PLURAL_NOUN
```

- irregular forms, with associated information similar to that for affixes

```
began PAST_VERB begin
begun PSP_VERB begin
```

- stems with syntactic categories

  e.g. to avoid *corpus* being analysed as *corpu -s*

Lexical requirements for morphological processing

- affixes, plus the associated information conveyed by the affix

```
ed PAST_VERB
ed PSP_VERB
s PLURAL_NOUN
```

- irregular forms, with associated information similar to that for affixes

```
began PAST_VERB begin
begun PSP_VERB begin
```

- stems with syntactic categories
  e.g. to avoid *corpus* being analysed as *corpu -s*

How is morphological processing implemented?

- rule-based methods
  - finate state transducer for full morphological analysis
  - stemming with e.g. the Porter stemmer
    (used in the practical)
- morphological segmentation or lemmatisation as supervised classification
- neural models with character-level input
  (discussed later in the course)

Outline.

Morphology

Syntax and formal grammars

Syntactic parsing

Why is syntax important?

- Last time we saw models of word sequences – n-grams
- Why is this insufficient?
- Because language has long-distance dependencies:

  The computer which I had just put into the machine room on the fifth floor is crashing.

- We want models that can capture these dependencies.

Syntactic parsing

Modelling syntactic structure of phrases and sentences.

Why is it useful?

- as a step in assigning semantics
- checking grammaticality
- applications: e.g. produce features for classification in sentiment analysis

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 form a coherent unit

The internal structures are typically given labels, e.g. *the big dog* is a noun phrase (NP) and *slept* is a verb phrase (VP)

Phrases and substitutability

- POS categories indicate which *words* are substitutable. For e.g., substituting adjectives:

  I saw a red cat
  I saw a sleepy cat

- Phrasal categories indicate which *phrases* are substitutable. For e.g., substituting noun phrases:

  Dogs sleep soundly
  My next-door neighbours sleep soundly
  Green ideas sleep soundly

- Examples of phrasal categories: Noun Phrase (NP), Verb Phrase (VP), Prepositional Phrase (PP), etc.

  We want to capture substitutability at the phrasal level!

Phrases and substitutability

- POS categories indicate which *words* are substitutable. For e.g., substituting adjectives:

  I saw a red cat

  I saw a sleepy cat

- Phrasal categories indicate which *phrases* are substitutable. For e.g., substituting noun phrases:

  Dogs sleep soundly
  My next-door neighbours sleep soundly
  Green ideas sleep soundly

- Examples of phrasal categories: Noun Phrase (NP), Verb Phrase (VP), Prepositional Phrase (PP), etc.

  We want to capture substitutability at the phrasal level!

Phrases and substitutability

- POS categories indicate which *words* are substitutable. For e.g., substituting adjectives:

  I saw a red cat

  I saw a sleepy cat

- Phrasal categories indicate which *phrases* are substitutable. For e.g., substituting noun phrases:

  Dogs sleep soundly
  My next-door neighbours sleep soundly
  Green ideas sleep soundly

- Examples of phrasal categories: Noun Phrase (NP), Verb Phrase (VP), Prepositional Phrase (PP), etc.

  We want to capture substitutability at the phrasal level!

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.:

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

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

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

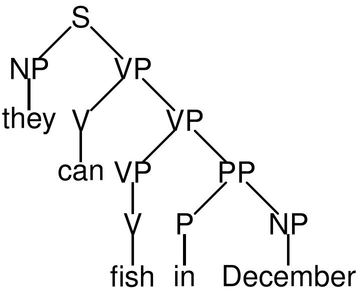
Structural ambiguity without lexical ambiguity

they fish in rivers in December

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

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

Parse trees




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

Outline.

Morphology

Syntax and formal grammars

Syntactic parsing

Chart parsing

chart store partial results of parsing in a vector

edge representation of a rule application

Edge data structure:

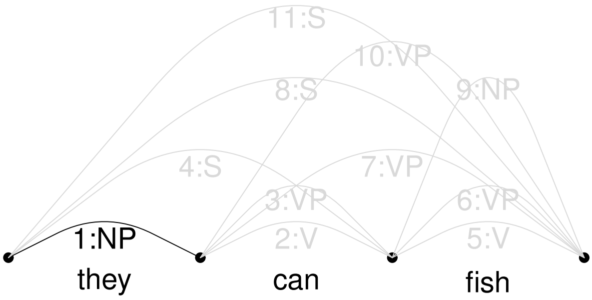
[*id*,*left_vtx*, *right_vtx*,*mother_category*, *dtrs*]

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

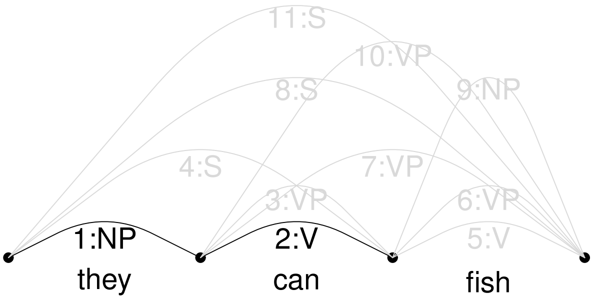
Fragment of chart:

id	left	right	mother	daughters
1	0	1	NP	(they)
2	1	2	V	(can)
3	1	2	VP	(2)
4	0	2	S	(1 3)

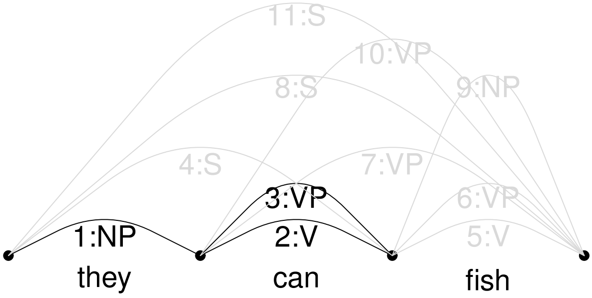
Bottom up parsing: example



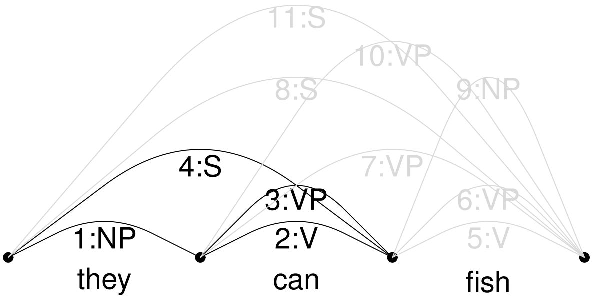
Bottom up parsing: example



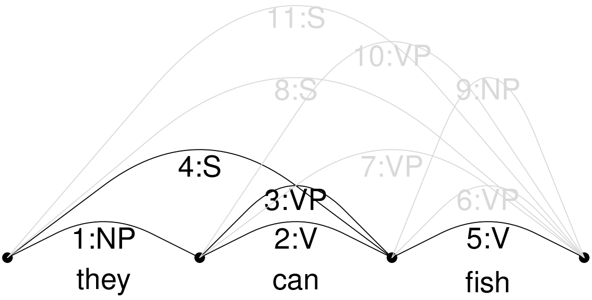
Bottom up parsing: example



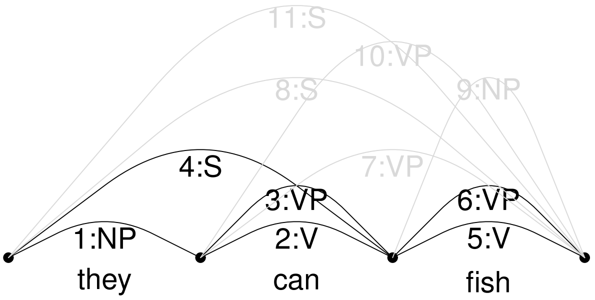
Bottom up parsing: example



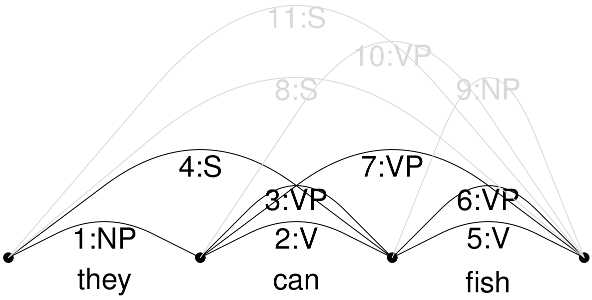
Bottom up parsing: example



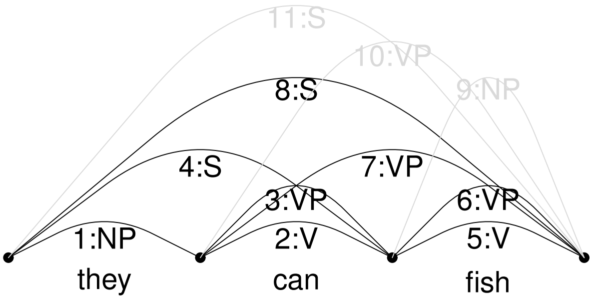
Bottom up parsing: example



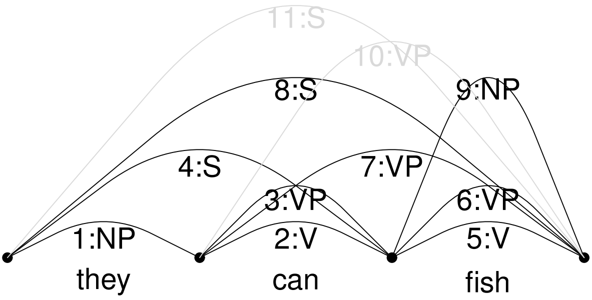
Bottom up parsing: example



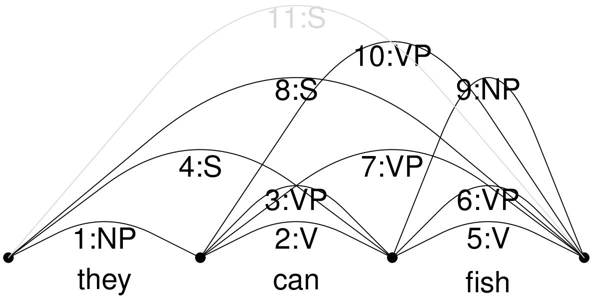
Bottom up parsing: example



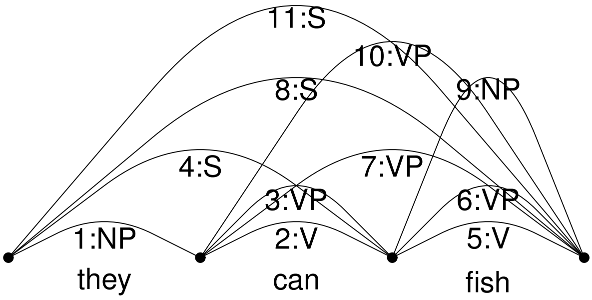
Bottom up parsing: example



Bottom up parsing: example



Bottom up parsing: example



Resulting chart

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

id	left	right	mother	daughters
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)

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)))
```

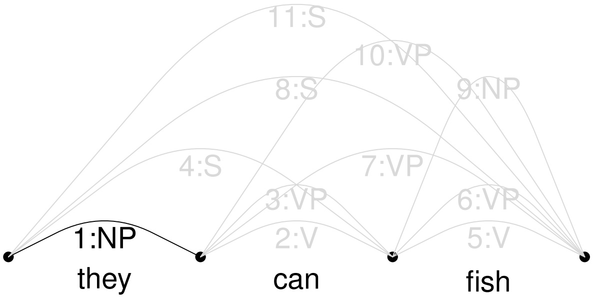
A bottom-up chart parser

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

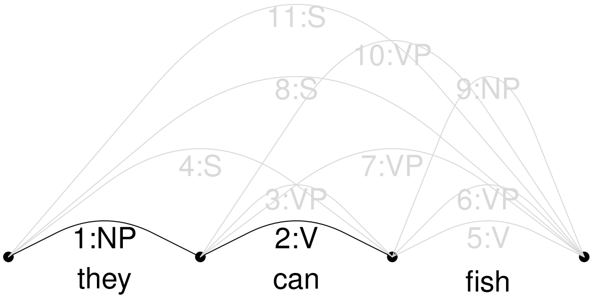
Add new edge *from*, *to*, *category*, *dtrs*:
Put edge in chart: [*id*,*from*,*to*, *category*,*dtrs*]
For each *rule lhs* $\rightarrow$ $cat_1 \dots cat_{n-1}$,*category*
Find sets of contiguous edges
[id_1,$from_1$,to_1, cat_1,$dtrs_1$] ...
[id_{n-1},$from_{n-1}$,*from*, cat_{n-1},$dtrs_{n-1}$]
(such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

Parse construction



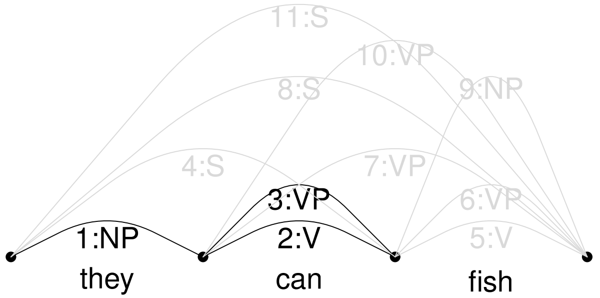
word = they, categories = {NP}
Add new edge 0, 1, NP, (they)
Matching grammar rules: {VP→V NP, PP→P NP}
No matching edges corresponding to V or P

Parse construction



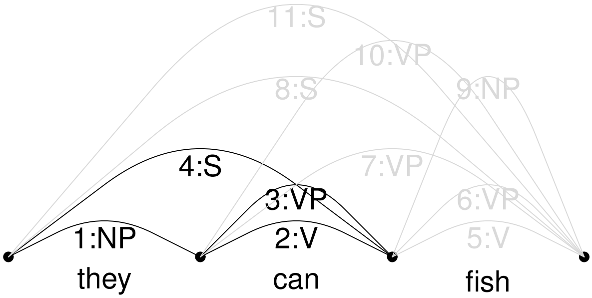
word = can, categories = {V}
Add new edge 1, 2, V, (can)
Matching grammar rules: {VP→V}
recurse on edges {(2)}

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



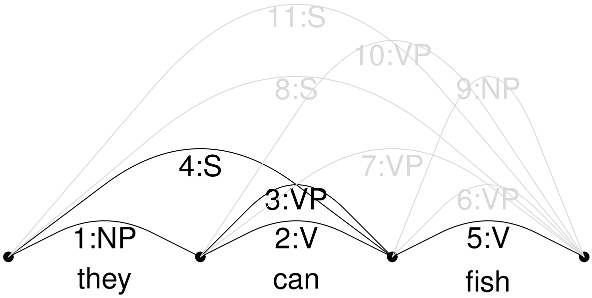
Add new edge 0, 2, S, (1, 3)

No matching grammar rules for S

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

No edges for V VP

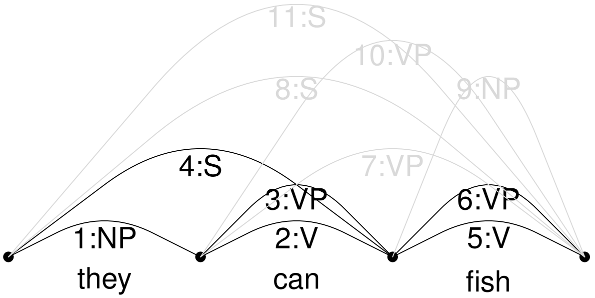
Parse construction



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

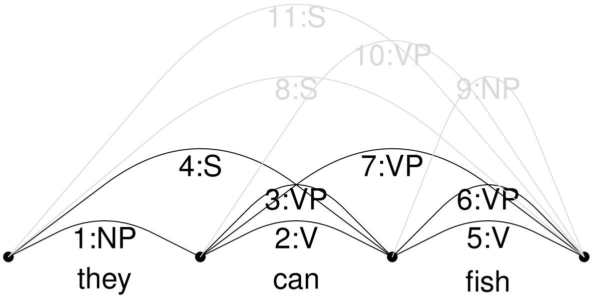
NB: fish as V

Parse construction



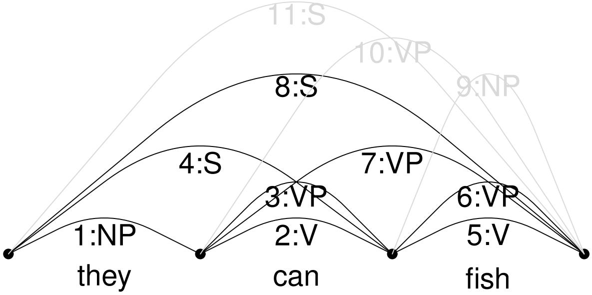
Add new edge 2, 3, VP, (5)
Matching grammar rules: {S →NP VP, VP →V VP}
No edges match NP
recurse on edges for V VP: {(2,6)}

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



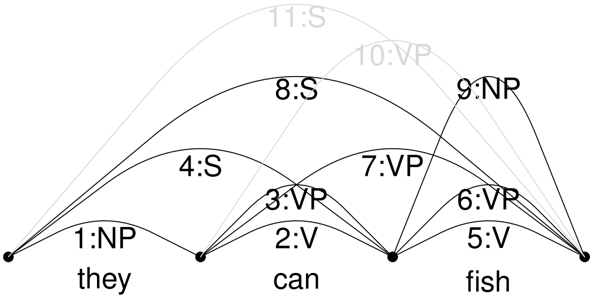
Add new edge 0, 3, S, (1, 7)

No matching grammar rules for S

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

No edges matching V

Parse construction



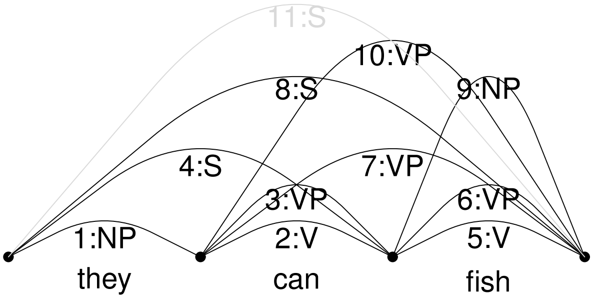
Add new edge 2, 3, NP, (fish)

NB: fish as NP

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

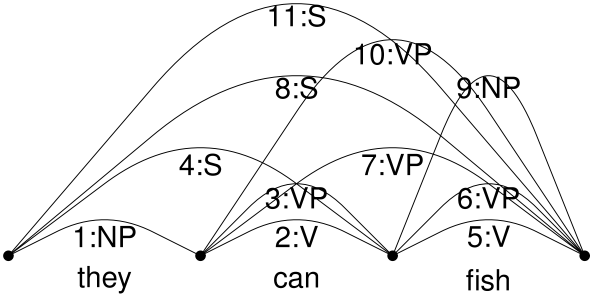
recurse on edges for V NP {(2,9)}

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

Packing

To make parsing more efficient:

- 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

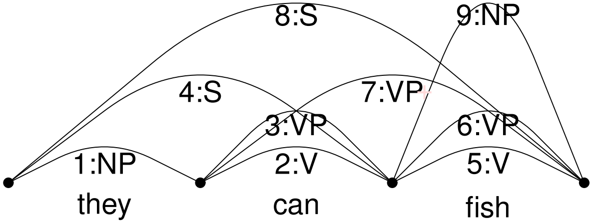
```
1   0   1   NP    {(they)}
2   1   2   V     {(can)}
3   1   2   VP    {(2)}
4   0   2   S     {(1 3)}
5   2   3   V     {(fish)}
6   2   3   VP    {(5)}
7   1   3   VP    {(2 6)}
8   0   3   S     {(1 7)}
9   2   3   NP    {(fish)}
```

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

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

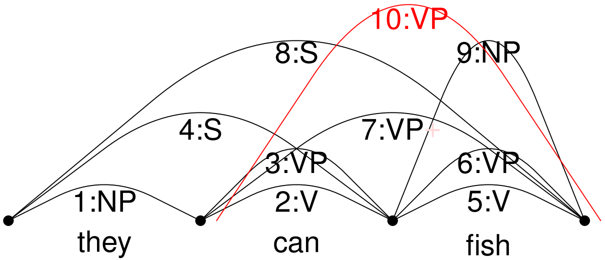
and we're done

Packing example



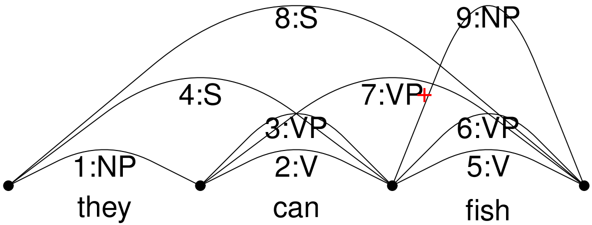
Both spanning results can now be extracted from edge 8.

Packing example



Both spanning results can now be extracted from edge 8.

Packing example



Both spanning results can now be extracted from edge 8.

Probabilistic Parsing

- How can we choose the correct tree for a given sentence?
- Traditional approach: grammar rules hand-written by linguists
  - constraints added to limit unlikely parses for sentences
  - hand-written grammars are not robust: often fail to parse new sentences.
- Current approach: use probabilities
  - Probabilitistic CFG (PCFG)
  - a CFG where each rule is augmented with a probability

An Example PCFG

$S \rightarrow NP\ VP$	.8
$S \rightarrow VP$	.2
$NP \rightarrow D\ N$	.4
$NP \rightarrow NP\ PP$	.4
$NP \rightarrow PN$	.2
$VP \rightarrow V\ NP$	.7
$VP \rightarrow VP\ PP$	.3
$PP \rightarrow P\ NP$	1

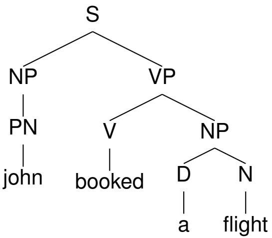
$D \rightarrow$ *the*	.8
$D \rightarrow$ *a*	.2
$N \rightarrow$ *flight*	1
$PN \rightarrow$ *john*	.9
$PN \rightarrow$ *schiphol*	.1
$V \rightarrow$ *booked*	1
$P \rightarrow$ *from*	1

How to compute the probability of a parse tree?

Computing the probability of a parse tree

The probability of a parse tree for a given sentence:

- the product of the probabilities of all the grammar rules used in the sentence derivation.



$$
\begin{aligned}
P(t) = & P(S \rightarrow NP\ VP) \times P(NP \rightarrow PN) \times P(PN \rightarrow john) \times \\
& P(VP \rightarrow V\ NP) \times P(V \rightarrow booked) \times \\
& P(NP \rightarrow D\ N) \times P(D \rightarrow a) \times P(N \rightarrow flight)
\end{aligned}
$$

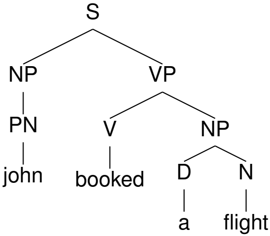
$$= .8 \times .2 \times .9 \times .7 \times .4 \times .2 \times 1$$

$$= .008064$$

Computing the probability of a parse tree

The probability of a parse tree for a given sentence:

- the product of the probabilities of all the grammar rules used in the sentence derivation.

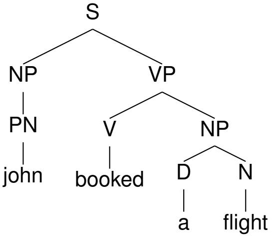


$$
\begin{aligned}
P(t) = & P(S \rightarrow NP\ VP) \times P(NP \rightarrow PN) \times P(PN \rightarrow john) \times \\
& P(VP \rightarrow V\ NP) \times P(V \rightarrow booked) \times \\
& P(NP \rightarrow D\ N) \times P(D \rightarrow a) \times P(N \rightarrow flight) \\
= & .8 \times .2 \times .9 \times .7 \times .4 \times .2 \times 1 \\
= & .008064
\end{aligned}
$$

Computing the probability of a parse tree

The probability of a parse tree for a given sentence:

- the product of the probabilities of all the grammar rules used in the sentence derivation.



$$
\begin{aligned}
P(t) = & P(S \rightarrow NP\ VP) \times P(NP \rightarrow PN) \times P(PN \rightarrow john) \times \\
& P(VP \rightarrow V\ NP) \times P(V \rightarrow booked) \times \\
& P(NP \rightarrow D\ N) \times P(D \rightarrow a) \times P(N \rightarrow flight)
\end{aligned}
$$

$$= .8 \times .2 \times .9 \times .7 \times .4 \times .2 \times 1$$

$$= .008064$$

Disambiguation with PCFGs

These probabilities can provide a criterion for disambiguation:

- i.e. a ranking over possible parses for any sentence
- we can choose the parse tree with the highest probability.

Disambiguation with PCFGs

$S \rightarrow NP\ VP$	.8
$S \rightarrow VP$	.2
$NP \rightarrow D\ N$	.4
$NP \rightarrow NP\ PP$	.4
$NP \rightarrow PN$	.2
$VP \rightarrow V\ NP$	.7
$VP \rightarrow VP\ PP$	.3
$PP \rightarrow P\ NP$	1

$D \rightarrow$ *the*	.8
$D \rightarrow$ *a*	.2
$N \rightarrow$ *flight*	1
$PN \rightarrow$ *john*	.9
$PN \rightarrow$ *schiphol*	.1
$V \rightarrow$ *booked*	1
$P \rightarrow$ *from*	1

John booked a flight from Schiphol

$P(t_1) = 6.4512 \times 10^{-5}$

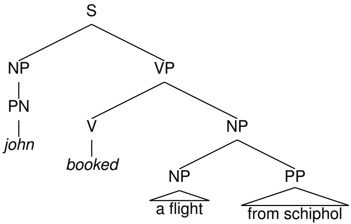
$P(t_2) = 4.8384 \times 10^{-5}$

Disambiguation with PCFGs

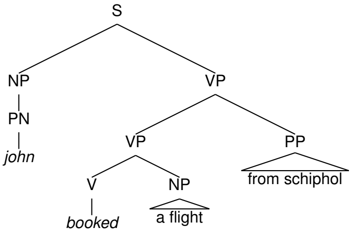
$S \rightarrow NP\ VP$	.8
$S \rightarrow VP$	.2
$NP \rightarrow D\ N$	.4
$NP \rightarrow NP\ PP$	.4
$NP \rightarrow PN$	.2
$VP \rightarrow V\ NP$	.7
$VP \rightarrow VP\ PP$	.3
$PP \rightarrow P\ NP$	1

$D \rightarrow the$	.8
$D \rightarrow a$	.2
$N \rightarrow flight$	1
$PN \rightarrow john$	.9
$PN \rightarrow schiphol$	.1
$V \rightarrow booked$	1
$P \rightarrow from$	1

John booked a flight from Schiphol



$P(t_1) = 6.4512 \times 10^{-5}$



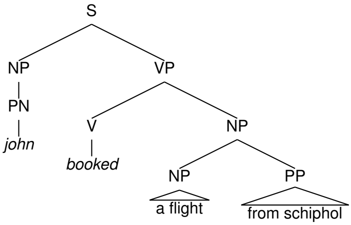
$P(t_2) = 4.8384 \times 10^{-5}$

Disambiguation with PCFGs

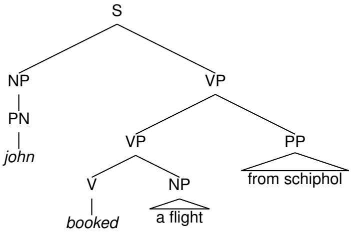
$S \rightarrow NP\ VP$	.8
$S \rightarrow VP$	.2
$NP \rightarrow D\ N$	.4
$NP \rightarrow NP\ PP$	.4
$NP \rightarrow PN$	.2
$VP \rightarrow V\ NP$	.7
$VP \rightarrow VP\ PP$	.3
$PP \rightarrow P\ NP$	1

$D \rightarrow the$	.8
$D \rightarrow a$	.2
$N \rightarrow flight$	1
$PN \rightarrow john$	.9
$PN \rightarrow schiphol$	.1
$V \rightarrow booked$	1
$P \rightarrow from$	1

John booked a flight from Schiphol



$P(t_1) = 6.4512 \times 10^{-5}$



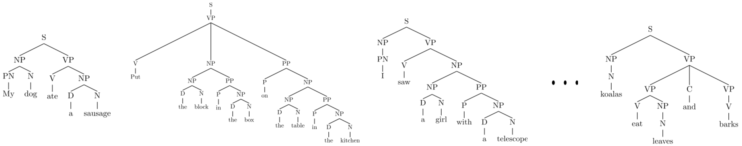
$P(t_2) = 4.8384 \times 10^{-5}$

Treebank PCFGs

- Treebanks: instead of paying linguists to write a grammar, pay them to annotate real sentences with parse trees.
- This way, we implicitly get a grammar (for CFG: read the rules off the trees)
- And we get probabilities for those rules
- We can use these probabilities to improve disambiguation
- and also speed up parsing.

Estimating rule probabilities from a treebank

A treebank: a collection of sentences annotated with constituent trees



An estimated probability of a rule (**maximum likelihood** estimates):

$$p(X \rightarrow \alpha) = \frac{C(X \rightarrow \alpha)}{C(X)}$$

The number of times the rule used in the corpus

The number of times the nonterminal X appears in the treebank

Dependency structure

A dependency structure consists of dependency relations, which are binary and asymmetric.

John hit the ball

A relation consists of

- a head (H) — hit
- a dependent (D) — John
- a label identifying the relation between H and D — Subject

Dependency structure

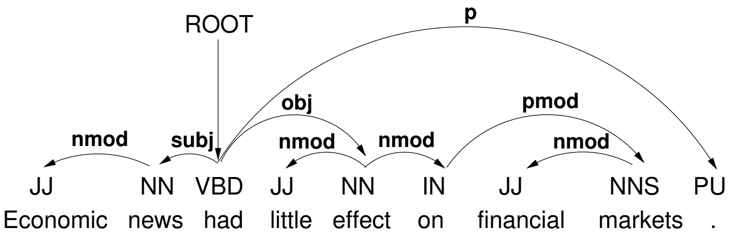
A dependency structure consists of dependency relations, which are binary and asymmetric.

John hit the ball

A relation consists of

- a head (H) — hit
- a dependent (D) — ball
- a label identifying the relation between H and D — Object

Example dependency structure



[From Joakim Nivre, Dependency Grammar and Dependency Parsing.]

Dependency parsing

Output a list of dependencies between words in the sentence.

John hit the ball.

```
(SUBJ head=hit dep=John)
(OBJ head=hit dep=ball)
(DET head=ball dep=the)
```

Why is it useful?

- dependencies provide an interface to semantics

  "Who did what to whom"

The cost of parsing errors...

Incorrect dependencies

```
(SUBJ head=hit dep=ball)
(OBJ head=hit dep=John)
(DET head=ball dep=the)
```

Acknowledgement

Some slides were adapted from Ann Copestake and Tejaswini Deoskar