

Natural Language Processing 1

Lecture 3: Modelling structure: morphology and syntax

Katia Shutova

ILLC
University of Amsterdam

4 November 2019

Outline of today's lecture

Morphology and finite state techniques

Formal grammars and syntactic parsing

Outline.

Morphology and finite state techniques

Formal grammars and syntactic parsing

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

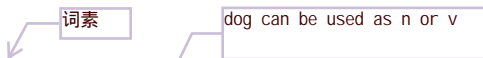
Guess the structure...

- ▶ 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
Бегают
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
Побежав
Бега

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):

OPTION 1 *pinged* / *ping-ed*

OPTION 2 *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*

Spelling rules

- ▶ English morphology is essentially **concatenative**
 - ▶ irregular morphology — inflectional forms have to be listed
- ▶ regular **phonological** and **spelling changes** associated with affixation, e.g.
 - ▶ -s is pronounced differently with stem ending in s, x or z
 - ▶ spelling reflects this with the addition of an e (*boxes* etc)
- ▶ in English, description is independent of particular stems/affixes

e-insertion

e.g. *box*^s to *boxes*

$$\varepsilon \rightarrow \mathbf{e} / \left\{ \begin{array}{c} \mathbf{s} \\ \mathbf{x} \\ \mathbf{z} \end{array} \right\} \wedge _ \mathbf{s}$$

- ▶ map ‘underlying’ form to surface form
- ▶ mapping is left of the slash, context to the right
- ▶ notation:

—	position of mapping
ε	empty string
^	affix boundary — stem ^ affix

- ▶ same rule for plural and 3sg verb
- ▶ formalisable/implementable as a finite state transducer

e-insertion

e.g. *box*^s to *boxes*

$$\varepsilon \rightarrow \text{e} / \left\{ \begin{array}{c} \text{s} \\ \text{x} \\ \text{z} \end{array} \right\} \wedge _ \text{s}$$

- ▶ map ‘underlying’ form to surface form
- ▶ mapping is left of the slash, context to the right
- ▶ notation:

_ position of mapping
 ε empty string
 ^ affix boundary — stem ^ affix

- ▶ same rule for plural and 3sg verb
- ▶ formalisable/implementable as a finite state transducer

e-insertion

e.g. *box*^s to *boxes*

$$\varepsilon \rightarrow \text{e} / \left\{ \begin{array}{c} \text{s} \\ \text{x} \\ \text{z} \end{array} \right\} \wedge _ \text{s}$$

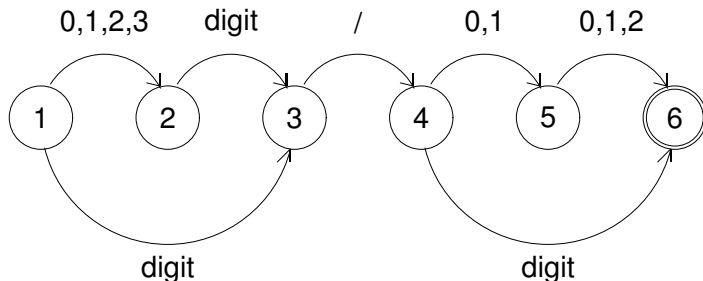
- ▶ map ‘underlying’ form to surface form
- ▶ mapping is left of the slash, context to the right
- ▶ notation:

_ position of mapping
 ε empty string
 ^ affix boundary — stem ^ affix

- ▶ same rule for plural and 3sg verb
- ▶ formalisable/implementable as a finite state transducer

Finite state automata for recognition

day/month pairs: e.g. 12/2, 1/12 etc.



- ▶ non-deterministic — after input of '2', in state 2 and state 3.
- ▶ double circle indicates accept state
- ▶ accepts e.g., 11/3 and 3/12
- ▶ also accepts 37/00 — overgeneration

e-insertion

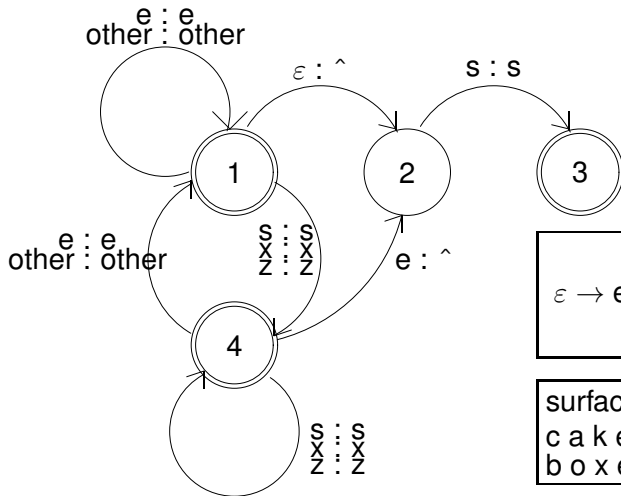
e.g. box^s to *boxes*

$$\varepsilon \rightarrow e / \left\{ \begin{array}{c} s \\ x \\ z \end{array} \right\} \wedge _ s$$

► notation:

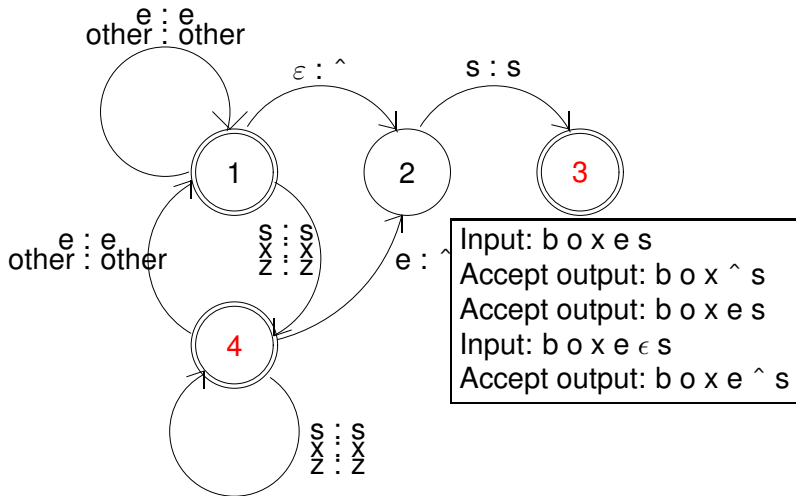
—	position of mapping
ε	empty string
\wedge	affix boundary — stem \wedge affix

Finite state transducer



$$\epsilon \rightarrow e / \left\{ \begin{array}{c} s \\ x \\ z \end{array} \right\} \wedge _ s$$

surface	: underlying
cakes	\leftrightarrow cake \wedge s
boxes	\leftrightarrow box \wedge s

Analysing *b o x e s*

Using FSTs

- ▶ FSTs assume **tokenization** (word boundaries) and words split into characters. One character pair per transition!
- ▶ **Analysis**: return character list with affix boundaries, so enabling lexical lookup.
- ▶ **Generation**: input comes from stem and affix lexicons.
- ▶ One FST per spelling rule: either compile to big FST or run in parallel.
- ▶ FSTs do not allow for internal structure:
 - ▶ can't model *un- ion -ize -d* bracketing.

How is morphological processing implemented?

- ▶ rule-based methods, e.g. the Porter stemmer
 - ▶ part of NLTK toolkit
 - ▶ used in the practical
- ▶ probabilistic models for morphological segmentation
- ▶ neural models with character-level input
(discussed later in the course)

Outline.

Morphology and finite state techniques

Formal grammars and 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**);

$S \rightarrow NP \ VP$

$V \rightarrow \text{fish}$

4. a start symbol, conventionally S, which is a non-terminal.

Exclude empty productions, NOT e.g.:

$NP \rightarrow \epsilon$

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



relating to the words
or vocabulary

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

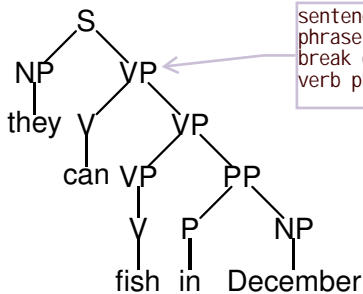
(S (NP they)
 (VP (VP (VP (V fish))
 (PP (P in) (NP rivers)))
 (PP (P in) (NP December))))

this is the correct structure. means:
 fish in december
 we human knows this structure makes
 sense, but the machine would provide
 2 ways to analyze the same sentence,
 which generate 2 different meanings.
 this is called "propositional phrase
 ambiguity". it is difficult for
 parsing to find out which structure
 is correct. this is a limitation of
 parsing

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

this is the wrong
 structure. it
 means: river in
 December

Parse trees



sentence is break down into noun phrase and verb phrase. the VP is break down into a verb and another verb phrase

```

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

Chart parsing

chart: a data structure to store info

chart **store** partial results of parsing in a vector

edge representation of a rule application

Edge data structure:

1 edge: 1 rule application

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

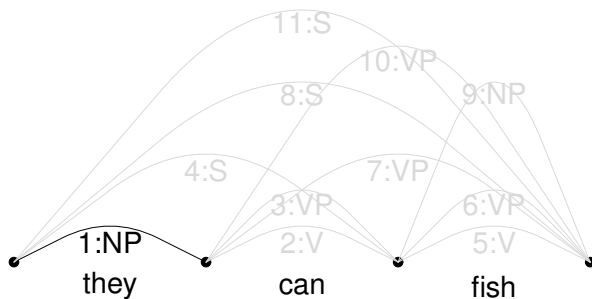
eg. id: an edge

. they . can . fish .
0 1 2 3

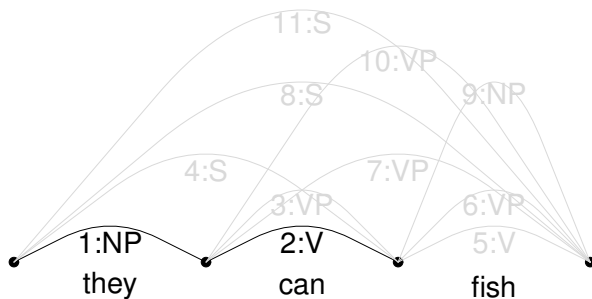
Fragment of chart:

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)

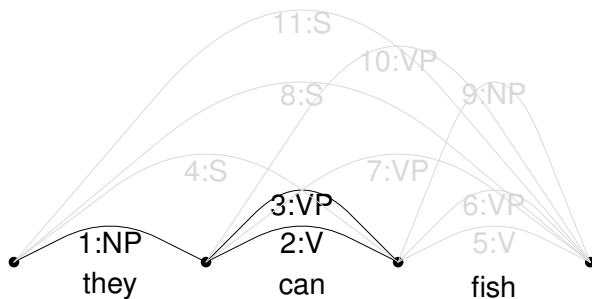
Bottom up parsing: edges



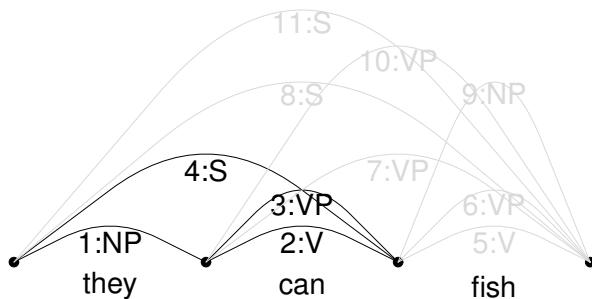
Bottom up parsing: edges



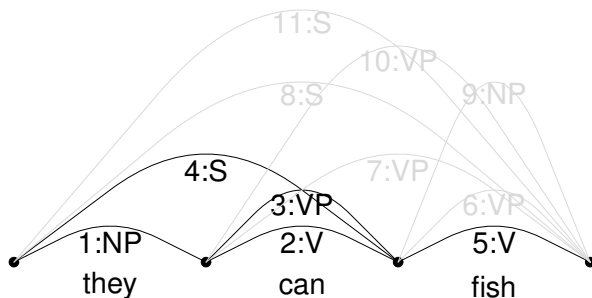
Bottom up parsing: edges



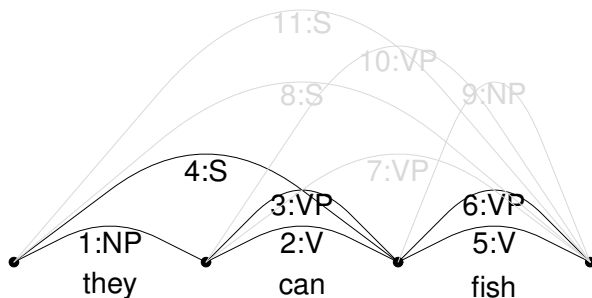
Bottom up parsing: edges



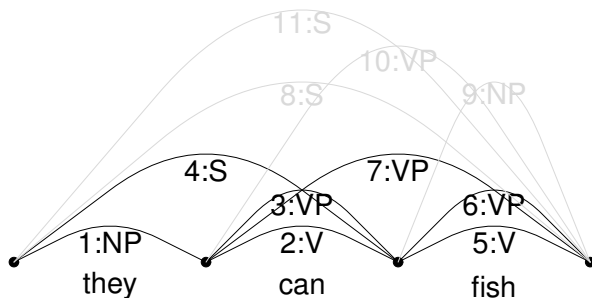
Bottom up parsing: edges



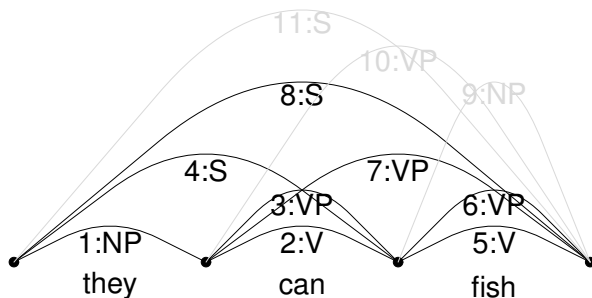
Bottom up parsing: edges



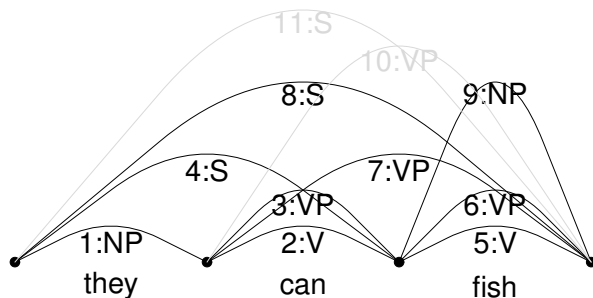
Bottom up parsing: edges



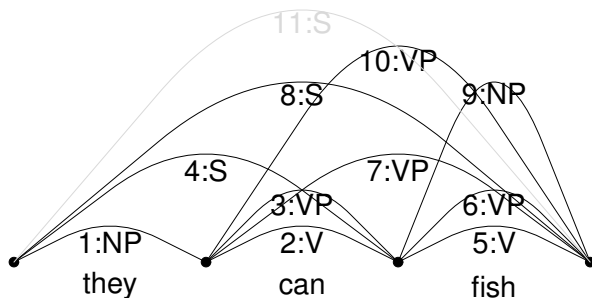
Bottom up parsing: edges



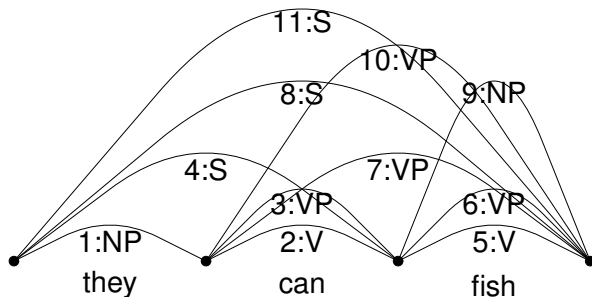
Bottom up parsing: edges



Bottom up parsing: edges



Bottom up parsing: edges



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

a recursive func
 after we add a new
 edge, we want to do
 the search for all
 possible analysis

indicate the span of the
 sentence (=where the
 sentence or word is)

mother category:
 egNP

daughter: fish
 daughter: is word itself

left word=left vertex

for word fish,
 we search: what grammar
 category can fish belong to?
 answer: fish ∈ {NP, VP, V}

Inner function

从左到右 left hand side
for this category (NP) at the right end, we want to
search for all matching rules which has this mother
category (NP) on the right end side 右边的最后一个

the category which we have
just added in the end of the
span of the sentence

Add new edge *from*, *to*, *category*, *dtrs*:

Put edge in chart: [*id*, *from*, *to*, *category*, *dtrs*]

For each *rule lhs* → *cat*₁ ... *cat*_{*n*-1}, *category*

Find sets of contiguous edges

[*id*₁, *from*₁, *to*₁, *cat*₁, *dtrs*₁] ...

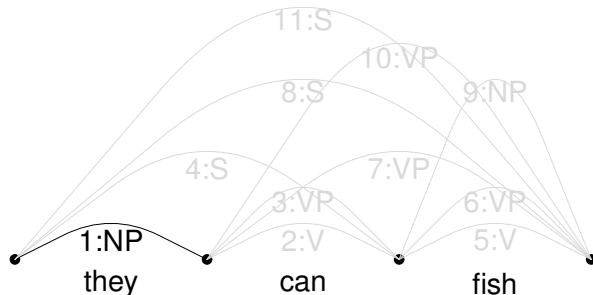
[*id*_{*n*-1}, *from*_{*n*-1}, *from*, *cat*_{*n*-1}, *dtrs*_{*n*-1}]

(such that *to*₁ = *from*₂ etc)

For each set of edges,

Add new edge *from*₁, *to*, *lhs*, (*id*₁ ... *id*)

Parse construction



word = they, categories = {NP}

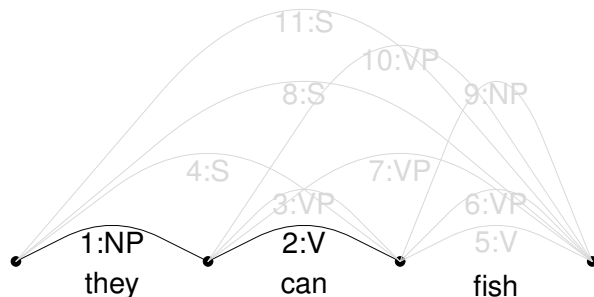
Add new edge 0, 1, NP, (they)

箭头：由...+...组成
VP is composed of V+NP

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

No matching edges corresponding to V or P

Parse construction



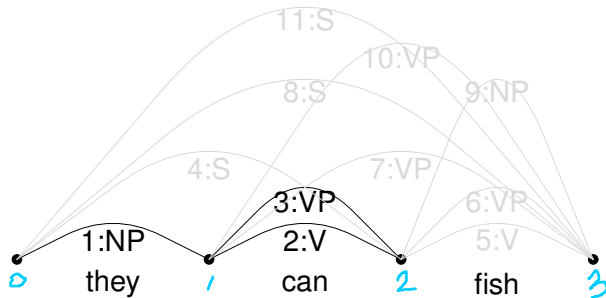
word = can, categories = {V}

Add new edge 1, 2, V, (can)

Matching grammar rules: $\{VP \rightarrow V\}$

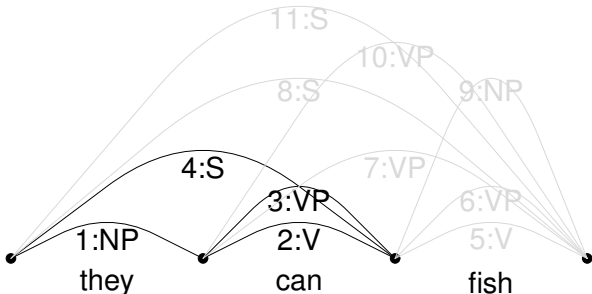
recurse on edges $\{(2)\}$

Parse construction



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

Matching grammar rules: $\{S \rightarrow NP VP, VP \rightarrow V VP\}$
 recurse on edges $\{(1,3)\}$



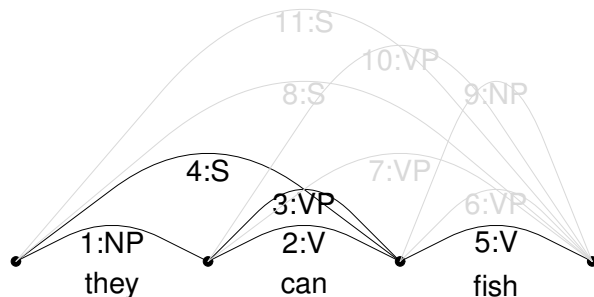
Add new edge 0, 2, S, (1, 3)

No matching grammar rules for S

Matching grammar rules: $\{S \rightarrow NP VP, VP \rightarrow V VP\}$

No edges for V VP

Parse construction



word = fish, categories = {V, NP}

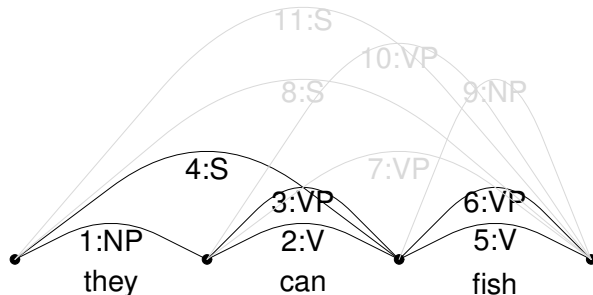
Add new edge 2, 3, V, (fish)

Matching grammar rules: $\{VP \rightarrow V\}$

recurse on edges $\{(5)\}$

NB: fish as V

Parse construction



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

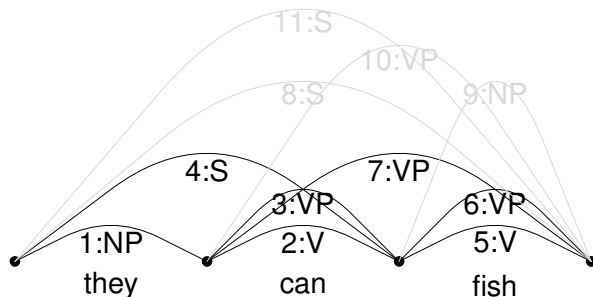
Matching grammar rules: $\{S \rightarrow NP VP, VP \rightarrow V VP\}$

No edges match NP

recurse on edges for V VP: $\{(2,6)\}$

edge 2 is V
edge 6 is VP

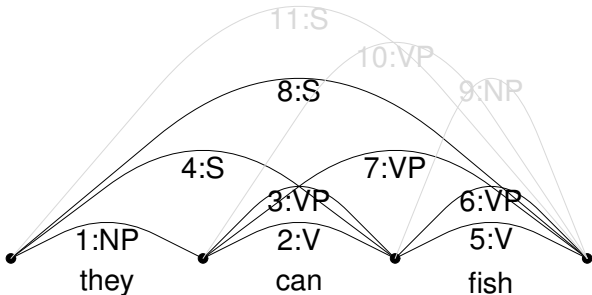
Parse construction



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

Matching grammar rules: $\{S \rightarrow NP VP, VP \rightarrow V VP\}$

recurse on edges for NP VP: $\{(1,7)\}$



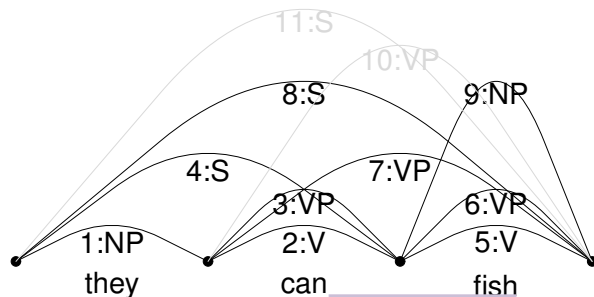
Add new edge 0, 3, S, (1, 7)

No matching grammar rules for S

Matching grammar rules: $\{S \rightarrow NP VP, VP \rightarrow V VP\}$

No edges matching V

Parse construction



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

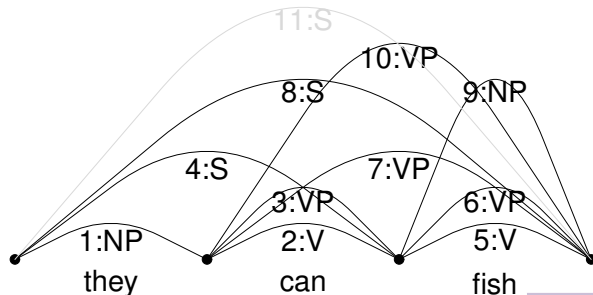
以NP结尾的rule 有哪些?
请看下一行

NB: fish as NP

Matching grammar rules: $\{VP \rightarrow V \text{ NP}, PP \rightarrow P \text{ NP}\}$

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

Parse construction



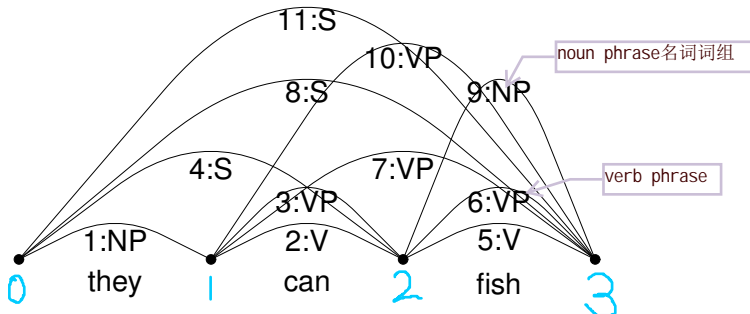
Add new edge 1, 3, VP, (2, 9)

Matching grammar rules: $\{S \rightarrow NP VP, VP \rightarrow V VP\}$

recurse on edges for NP VP: $\{(1, 10)\}$

以VP结尾的rule 有哪些?
请看下一行

Parse construction



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

No matching grammar rules for S

Matching grammar rules: $\{S \rightarrow NP VP, VP \rightarrow V VP\}$

No edges corresponding to V VP

Matching grammar rules: $\{VP \rightarrow V NP, PP \rightarrow P NP\}$

No edges corresponding to P NP

Resulting chart

. they . can . fish .

id	left	right	mother	daughters
0				
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)

grammar category of
mother edge

node 0

edge 2

edge 1 and edge 3 are the
daughters of edge "S"

edge 7 and 10 has the same
span. because they both
goes from node 1 to 3. so
we can pack them. we extend
the previous edge with
another daughter

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)))
```

Acknowledgement

Some slides were adapted from Ann Copestake and Tejaswini Deoskar