Outline of today’s lecture

Lecture 2: Morphology and finite state techniques
  A brief introduction to morphology
  Using morphology in NLP
  Aspects of morphological processing
  Finite state techniques
  More applications for finite state techniques
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
A brief introduction to morphology

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* is not a possible form, so not *((un- ion) -ise) -ed*
- *un-* is ambiguous:
  - with verbs: means ‘reversal’ (e.g., *untie*)
  - with adjectives: means ‘not’ (e.g., *unwise, unsurprised*)
- therefore *(un- ((ion -ise) -ed))**
Using morphological processing in NLP

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

  party + PLURAL $\leftrightarrow$ parties
  sleep + PAST_VERB $\leftrightarrow$ slept
Using morphological processing in NLP

run
runs
ran
running
Using morphological processing in NLP

<table>
<thead>
<tr>
<th>run</th>
<th>Бегал</th>
<th>Побегу</th>
</tr>
</thead>
<tbody>
<tr>
<td>runs</td>
<td>Бежал</td>
<td>Побежищий</td>
</tr>
<tr>
<td>ran</td>
<td>Побежал</td>
<td>Побежит</td>
</tr>
<tr>
<td>running</td>
<td>Бегала</td>
<td>Побежим</td>
</tr>
</tbody>
</table>

Побежала
Побежала
Побежите
Побегут

Бегаю
Бегу
Бегаешь
Бежишь
Бегает
Бежит
Бегаем
Бежим
Бегаете
Бежите
Бегают
Бегут
Using morphological processing in NLP

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

  party + PLURAL $\leftrightarrow$ parties
  sleep + PAST_VERB $\leftrightarrow$ slept
Lecture 2: Morphology and finite state techniques
Aspects of morphological processing

Morphological processing

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

   OPTION 1  \( pinged / ping-ed \)
   OPTION 2  \( pinged / ping \ PAST\_VERB \)
               \( pinged / ping \ PSP\_VERB \)
               \( sang / sing \ PAST\_VERB \)
               \( sung / sing \ PSP\_VERB \)

2. Internal structure / bracketing (e.g., \( (un- ((ion -ise) -ed)) \)).

3. Syntactic and semantic effects
   parsing can filter results of previous stages.
   e.g., \( feed \) analysed as \( fee-ed \) (as well as \( feed \))
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

\[
\varepsilon \rightarrow e/ \left\{ \begin{array}{c}
s \\ x \\ z \\
\end{array} \right\} ^{-} s
\]

- map ‘underlying’ form to surface form
- mapping is left of the slash, context to the right
- notation:
  - _ position of mapping
  - \( \varepsilon \) 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

\[ \varepsilon \rightarrow e/ \left\{ s \right\}^\text{^} \_ s \]

- map ‘underlying’ form to surface form
- mapping is left of the slash, context to the right
- notation:
  - _ position of mapping
  - \varepsilon 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

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

- map ‘underlying’ form to surface form
- mapping is left of the slash, context to the right
- notation:
  - _ position of mapping
  - $\varepsilon$ 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

\[
\varepsilon \rightarrow e/ \left\{ \begin{array}{c}
  s \\
  x \\
  z
\end{array} \right\} ^\_ s
\]

- map ‘underlying’ form to surface form
- mapping is left of the slash, context to the right
- notation:
  
  _ \hspace{1cm} \text{position of mapping}

  \varepsilon \hspace{1cm} \text{empty string}

  ^ \hspace{1cm} \text{affix boundary — stem} \hspace{1cm} \text{^ affix}
Finite state transducer

\[
\begin{align*}
\varepsilon & : \wedge \\
e & : e \\
other & : other \\
s & : s \\
x & : x \\
z & : z
\end{align*}
\]

Surface : underlying
- cakes ↔ cake \wedge s
- boxes ↔ box \wedge s

\[
\varepsilon \rightarrow e/ \left\{ \begin{array}{l} s \\ x \\ z\end{array} \right\} \wedge - s
\]
Analysing boxes

Input: boxes
Accept output: box

Input: box
Accept output: box

Input: box
Accept output: box

Input: box e
Accept output: box e

Input: box e s
Accept output: box 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.
  - can’t condition on prior transitions, so potential redundancy
Some other uses of finite state techniques in NLP

Dialogue models for spoken dialogue systems (SDS)
e.g. obtaining a date:

1. No information. System prompts for month and day.
2. Month only is known. System prompts for day.
3. Day only is known. System prompts for month.
4. Month and day known.
Example FSA for dialogue
Example of probabilistic FSA for dialogue