Lecture II.
Processing words

Tomaž Erjavec

The HLT low road:
Processing words

- Identifying words: regular expressions and tokenisation
- Analyzing words: finite state machines and morphology

What is a word?

- Smallest phonetic and semantic unit of language (more or less)
- We can distinguish several meanings of “word”:
  - Word-form in text (word¹):
    “The banks are closed today.”
  - The abstract lexical unit (word²)
    word¹ banks is the plural form of the word² bank
Basic steps in processing words

1. Tokenisation: word-forms are first identified in the text
e.g. “The banks are closed”  
   the+banks+are+closed

2. Morphological analysis: the word-forms are associated with their grammatical information
e.g. bank+s  → noun+plural

3. Lemmatisation: the “word” i.e. base form is identified, e.g. banks  → bank

4. Further information about the word is retrieved from the lexicon

Chomsky Hierarchy

<table>
<thead>
<tr>
<th>Artificial languages</th>
<th>Regular languages</th>
<th>Regular expressions</th>
<th>Finite state automata</th>
<th>Phonology</th>
<th>Morphology</th>
<th>Syntax?</th>
<th>Semantics?</th>
<th>Pragmatics?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbitrary computer program</td>
<td>Context-free languages</td>
<td>Context-free languages</td>
<td>Finite state automata</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recogniser/generator</td>
<td>Regular expressions</td>
<td>Regular expressions</td>
<td>Finite state automata</td>
<td>Phonology</td>
<td>Morphology</td>
<td>Syntax?</td>
<td>Semantics?</td>
<td>Pragmatics?</td>
</tr>
<tr>
<td>Natural languages</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Regular expressions

- A RE recognises a (possibly infinite) set of strings
- Literals: a,b,c,…
- Operators: concatenation, disjunction, repetition, grouping
- Basic examples:
  - /abc/ recognises {abc/}
  - /(ab)/ recognises {a b/}
  - /ab/ recognises {ab, abb, abbb, …}
- Extensions: sets (ab), ([abc]), special characters (‘,’ ‘,’ ‘,’ ‘,’ ‘,’
- Not only search, but also substitution: s/a(abc)/x$1y/(a=to x=by)
- Fast operation, implemented in many computer languages (esp. on Unix: grep, awk, Perl)
Text pre-processing

- Splitting the raw text into words and punctuation symbols (tokenisation), and sentences (segmentation)
- In free text there are also errors
- Also, different rules for different languages: 4., itd., das Haus, ...

Result of tokenisation

→ Euromoney's assessment of economic changes in Slovenia has been downgraded (page 6).
→
<s id="ecmr.en.17"> <w>Euromoney</w> <w type="rsplit">'s</w> <w>assessment</w> <w>of</w> <w>economic</w> <w>changes</w> <w>in</w> <w>Slovenia</w> <w>has</w> <w>been</w> <w>downgraded</w> <c type="open">(</c> <w type="dig">6</w> <c type="close">)</c>
</s>

Other uses of regular expressions

- Identifying named entities (person and geographical names, dates, amounts)
- Structural up-translation
- Searching in corpora
- Swiss army knife for HLT
2. Finite state automata and morphology

- It is simple to make a regular expression generator, difficult to make an efficient recogniser.
- FSAs are extremely fast, and only use a constant amount of memory.
- The languages of finite state automata (FSAs) are equivalent to those of regular expressions.
- A FSA consists of:
  - a set of characters (alphabet)
  - a set of states
  - a set of transitions between states, labeled by characters
  - an initial state
  - a set of final states
- A word / string is in the language of the FSA, if, starting at the initial state, we can traverse the FSA via the transitions, consuming one character at a time, to arrive at a final state with the empty string.

Some simple FSAs

- Talking sheep:
  - The language: `{baa!, baaa!, baaaa!, ...}`
  - Regular expression: `/baaa*!/`
  - FSA:

- Mystery FSA:
“Extensions”

- Non-deterministic FSAs
- FSAs with $\varepsilon$ moves
- But methods exist that convert $\varepsilon$FSA to NDFSA to DFSAs. (however, the size can increase significantly)

Operations on FSAs (and their languages)

- Concatenation
- Closure
- Union
- Intersection

Morphological analysis with the two-level model

- Task: to arrive from the surface realisation of morphemes to their deep (lexical) structure, e.g. $\text{dog}\_\text{N}\_\text{pl} \leftrightarrow \text{dogs}$ but $\text{wolf}\_\text{N}\_\text{pl} \leftrightarrow \text{wolves}$
- Practical benefit: this results in a smaller, easier to organise lexicon
- The surface structure differs from the lexical one because of the effect of (morpho-)phonological rules
- Such rules can be expressed with a special kind of FSAs, so called Finite State Transducers
Finite State Transducers

- The alphabet is taken to be composed of character pairs, one from the surface and the other from the lexical alphabet.
- The model is extended with the non-deterministic addition of pairs containing the null character.
- Input to transducer:
  - move + ed (in the lexicon)
  - move 0 0 d (in the text)
- The model can also be used generatively.

A FST rule

- Accepted input: move + ed
- Rejected input: move 0 0 d

We assume a lexicon with move + ed.

Rule notation

- Rules are easier to understand than FSTs; compiler from rules to FSTs.
- Devoicing:
  - surface mabap to lexical mabab
  - b:p & 0
  - Lexical b corresponds to surface p if and only if the pair occurs in the word-final position.
- 'e' insertion:
  - wish+s & wishes
  - + :e <= {s x z [s c] h} __ s
  - A lexical morph boundary between s, x, z, sh, or ch on the left side and an s on the right side must correspond to an e on the surface level. It makes no statements about other contexts where ' + ' may map to an 'e'.
- More examples from Slovene here.
FST composition

- Serial: original Hall&Chomsky proposal; feeding and bleeding rules (c.f. generative phonology)
- Parallel: Koskenniemmi approach; less ‘transformational’; rule conflicts

Stochastic FSAs

- Finite state machines can be supplemented by arc probabilities
- This makes them useful for statistically based processing: Hidden Markov Models, Viterbi algorithm

3. Storing words: the lexicon

- From initial systems where the lexicon was “the junkyard of exceptions” lexica have come to play a central role in CL and HLT
- What is a lexical entry? (multi-word entries, homonyms, multiple senses)
- Lexica can contain a vast amount of information about an entry:
  - Spelling and pronunciation
  - Formal syntactic and morphological properties
  - Definition (in a formalism) and qualifiers
  - Examples (frequency counts)
  - Translation(s)
  - Related words (thesaurus / ontology)
  - Other links (external knowledge sources)
- An extremely valuable resource for HLT of a particular language
- MRDs are useful as a basis for lexicon development, but less than may be thought (vague, sloppy)
Lexicon as a FSA

- The FSA approach is also used to encompass the lexicon: efficient storage, fast access
- A trie: 

Hierarchical organisation

- With emphasis on lexica, each entry can contain lots of information
- But much of it is repeated over and over
- The lexicon can be organised in a hierarchy with information inherited along this hierarchy
- Various types of inheritance, and associated problems: multiple inheritance, default inheritance

Summary

The lecture concentrated on processing words, esp. on two basic tasks:
- Identifying words: regular expressions and tokenisation
- Analyzing words: finite state machines and morphology