Language Technologies

"New Media and eScience" MSc Programme
Jožef Stefan International Postgraduate School

Winter/Spring Semester, 2006/07

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*[2]", i.e. base form is identified, e.g. *banks* → *bank*
4. Further information about the word is retrieved from the lexicon

Chomsky Hierarchy

Artificial languages	Recogniser/generator	Natural languages
Regular languages	**Regular expressions Finite state automata**	**Phonology Morphology**
Context-free languages	Context-free rules Push-down automata	Syntax?
Context-sensitive languages	Context-sensitive rls. Weird automata	Syntax?
Turing machine languages	Arbitrary computer program	Semantics? Pragmatics?

Regular expressions

- A RE recognises a (possibly infinite) set of strings
- Literals: a,b,c,č,…
- Operators: concatenation, disjunction, repetition, grouping
- Basic examples:
  - /abc/ recognises {*abc*}
  - /(a|b)/ recognises {*a, b*}
  - /ab./ recognises {*aba, abb, abc*,…}
  - /ab*/ recognises {*a, ab, abb*, …}
- Extensions: sets ([abc], [^abc]), special characters (\., \t, \n, \d)
- Not only search, but also substitution: s/a(.)c/x$1y/ (*abc* to x*b*y)
- 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)
- Not as simple as it looks:
  kvačka, 23rd, teacher's, [2,3H]dexamethasone, etc., kogarkoli, http://nl2.ijs.si/cgi-bin/corpus-search?Display=KWIC&Context=60&Corpus=ORW-SL&Query="hoditi",
  "So," said Dr. A. B. "who cares?"
- 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).*

→

```
<seg 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>page</w>
  <w type="dig">6</w><c type="close">)</c>
  <c>.</c>
</seg>
```

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

Identifying signatures

```
<S>V Bruslju, 15. aprila 1958</S>
<S>V Frankfurtu na Maini, 21.junija 2001</S>                (no space after day)
<S>V Bruslju 19. julija 1999</S>                            (no comma after place)
<S>V Bruslju, dne 27 oktobra1998.</S>                       (no space after month)
<S>V Bruslju,  2000</S>                                     (just year)
<S>V Helsinksih, sedemnajstega marca tisočdevetstodvaindevetdeset</S> (words!)
<S>V Luksemburgu</S>                                        (no date)
<S>V Dne</S>                                                (just template)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/<S>V\s     #Start of sentence, 'In', space
            [A-TV-Z]                                #Capital letter that starts place name, but not
            'U'(redba)                              #"Uredba"
            .{2,20}                                 #whatever, but not too long
            [\s,]\d                                 #some whitespace or comma, day of month
            .{0,3}                                  #whatever, but not too long
            (
            (januar|februar|marec|marca|april       #month
            |maj|junij|julij|avgust|september       #in two forms (cases) only
            |septembra|oktober|oktobra|november     #when change of stem
            |novembra|december|decembra)
            |
            1?\d                                    #or month as number
            )
            .{0,3}                                  #whatever, but not too long
            (19\d\d | 20\d\d)                       #exactly four digits for the year
            \.?                                     #maybe full stop
            .{0,100}                                #trailing blues..
<\/S>                                               #and end of sentence
/x
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Matches 7820 times with no errors: precision = 100%, recall=?

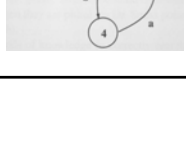
2. Finite state automata and morphology

- 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

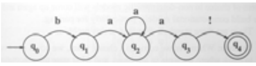
- Talking sheep:
  - The language: {baa!, baaa!, baaaa!, …}
  - Regular expression: /baaa*!/
  - FSA:

- Mystery FSA:

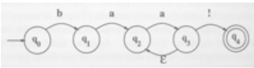


“Extensions”

- Non-deterministic FSAs



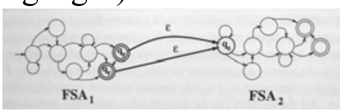
- FSAs with ε moves



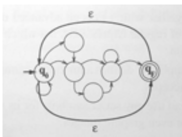
- But metods exist that convert εFSA to NDFSAs to DFSAs. (however, the size can increase significantly)

Operations on FSAs (and their languages)

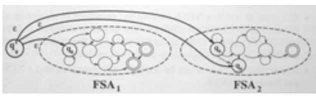
- Concatenation



- Closure



- Union



- Intersection!

Morphological analysis with the two-level model

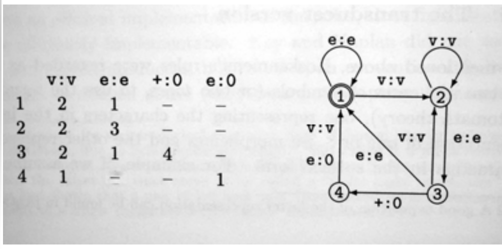
- Task: to arrive from the surface realisation of morphemes to their deep (lexical) structure, e.g. $dog_{[N]}+s_{[pl]}$ ← *dogs* but $\mathbf{wolf_{[N]}+s_{[pl]}}$ ← *wolves*
- Practical benefit: this results in a smaller, easier to organise lexicon
- The surface structure differs from the lexical one 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:
  m o v e + e d (in the lexicon)
  m o v e 0 0 d (in the text)
- The model can also be used generativelly

A FST rule

- Accepted input:
  m:m o:o v:v e:e +:0 e:0 d:d
- Rejected input:
  m:m o:o v:v e:e +:0 e:e d:d
- We assume a lexicon with move+ed
- Would need to extend left and right context



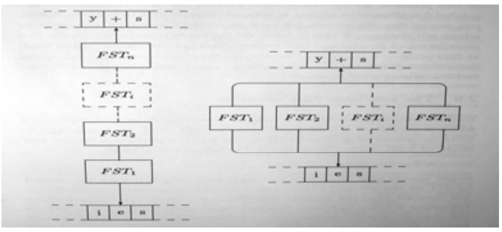
	v:v	e:e	+:0	e:0
1	2	1	–	–
2	2	3	–	–
3	2	1	4	–
4	1	–	–	1

Rule notation

- Rules are easier to understand than FSTs; compiler from rules to FSTs
- devoicing:
  - surface *mabap* to lexical *mabab*
  - b:p ⇔ ___#
  - 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 <u>here</u>

FST composition

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



Stochastic FSAs

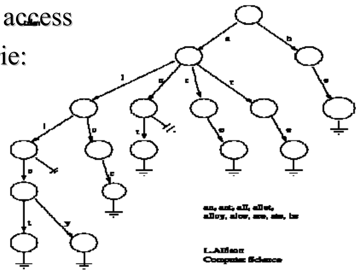
- Finite state machines can be supplemented by arc probabilities
- This makes then useful for statisticaly based processing: Hidden Markov Models, Viterbi algorithm

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 HTL
- 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 though (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