Computational Morphology: Xerox finite state tool, lexc

Yulia Zinova

15 February 2016 – 19 February 2016

Overview

What is lexc?

Making lexical transducers

LEXC

- The Finite-State Lexicon Compiler (lexc) is an authoring tool for creating lexicons and lexical transducers.
- **lexc** is designed to be used in conjunction with transducers produced with the Xerox Two-level Rule Compiler (Karttunen, 2003).
- These slides are based on the Karttunen (2003) book and the Karttunen (1993) technical report.

What is **lexc**?

- A lexical transducer is a specialized finite-state automaton that maps lexical forms and morphological specifications to corresponding inflected forms, and vice versa.
- The input to the lexicon compiler is a text file in a format similar to lexicons accepted by Kimmo Koskenniemi's two-level program and Evan Antworth's PC-KIMMO.
- The output from the compiler is a simple finite-state automaton or a transducer.
- The result of the compilation may be composed with a rule transducer or transducers that effect morphological alternations or in other ways modify the initial result.

Commands

- To start the compiler, type ‘`lexc`’
- There are three groups of commands: Input/Output, Operations, and Display.
- Each group is further divided into subsections of related commands.
- To get the list of commands, type the question mark.
- ‘`help`’ command, followed by a name of a particular command, gives advice on what the commands do;
- ‘`help all`’ prints out all help messages.

Main commands

- Three main commands: `'compile-source'`, `'read-rules'`, `'compose-result.'`
- *Source net* is the automaton (either a simple lexicon or a transducer) compiled from the input lexicon.
- The command `'compile-source'` creates the source net.
- *Rule nets* refers to a set of rule transducers created by the two-level rule compiler (or by other means) and saved in binary form.
- The command `'read-rules'` reads the rule nets from a file.
- The command `'compose-result'` composes the source net with the rule net or nets and produces the result net.

A very simple example

- Let us construct a lexicon that contains the forms of two lexemes, *dine* and *line*.
- *line*: a noun and a verb, *dine*: a verb.
- The simplest way to define the lexicon is to list all the forms under the heading LEXICON Root (ex0-lex.txt).
- Compilation of this lexicon creates a finite-state machine (fsm) that accepts all only the listed word forms.

Basic rules

- There may be any number of sublexicons, but one of them must have the name Root.
- The other sublexicons, if any, are named according to the needs and tastes of the lexicon writer.
- Entries are separated by semicolons.
- Each entry consists of a form, possibly null, and a continuation class.
- The continuation class must be the name of another sublexicon or #, a terminator symbol.
- The word END at the end of the file is optional.
- The exclamation point, !, marks a comment; the rest of the line is ignored.

Word classes

- Let us define another version of the lexicon that distinguishes stems from suffixes (ex1-lex.txt).
- Some entries have no content other than the continuation class, some have a sublexicon name as the continuation class.
- compile it!
  `compile-source ex1-lex.txt`
- The command ‘`save-source`’ saves the compiled source net as a binary file.

Exercise: A lexicon of dates

- Let us construct a lexicon that contains all valid dates of the form "<month> <date>, <year>", such as *March 14, 1993*.
- Restrictions: exact format (a space between the month and the day, a comma and a space between the day and the year), <year> between 0 and 9999.
- Every date in the lexicon should be valid; *April 31, 1993* should not be included (30 days in April). Assume that February has 29 days.
- Whitespace characters (blank, tab, carriage return, linefeed) are ignored by the compiler. % should be placed in front of a blank that should not count as whitespace.
- The numeral 0 is also special and has to be prefixed with % to be an ordinary digit.
  June 10, 2000 $\rightarrow$ June% 1%0,% 2%0%0%0

Exercise: Eliminate unwanted leap days

- February has 29 days only in years that are divisible by four, except that centuries are leap years only if the number is still divisible by four when the last two zeros are left out.
- For example, 1900 was not a leap year but in the year 2000 there were 29 days in February.
- Write a transducer that eliminates unnecessary leap days (use xfst). Compile this transducer and store it as ex3-rule.fst

Exercise: Combine lexicon and transducer

- Load the lexicon...

  `compile-source ex2-lex.txt`

Exercise: Combine lexicon and transducer

- Load the lexicon...
  `compile-source ex2-lex.txt`
- and the transducer...
  `read-rules ex3-rule.fst`

Exercise: Combine lexicon and transducer

- Load the lexicon...
  `compile-source ex2-lex.txt`
- and the transducer...
  `read-rules ex3-rule.fst`
- and compose them...
  `compose-result`
- now try the `status` command

Exercise: Combine lexicon and transducer

- Load the lexicon...
  `compile-source ex2-lex.txt`
- and the transducer...
  `read-rules ex3-rule.fst`
- and compose them...
  `compose-result`
- now try the `status` command
- Are you on the right track?
- Try `lookup February 29, 1900`

Exercise: Combine lexicon and transducer

- Load the lexicon...
  `compile-source ex2-lex.txt`
- and the transducer...
  `read-rules ex3-rule.fst`
- and compose them...
  `compose-result`
- now try the `status` command
- Are you on the right track?
- Try `lookup February 29, 1900`
- If everything is fine, `save-result ex3-lex.fsm`

A simple lexical transducer (1)

- A lexical transducer is a lemmatizer that maps inflected surface forms to their canonical dictionary representations and vice versa.
- In simple cases, such transducers can be compiled directly from a source lexicon.
- We can modify the minilexicon we created to make it a morphological analyzer/generator for the forms of *dine* and *line*.
- In addition to the citation form of the word itself, the entries now contain morphological information:
  - part of speech (+N, +V)
  - type of inflection (+Pl, +Sg, +Base, +Sg3, +Past, +PastPart)

A simple lexical transducer (1)

- Some of the entries now contain a pair of forms: a lexical form and a surface form, separated by a colon.
- The compiler interprets such a pair as a regular relation, a sequence of pairwise correspondences between the symbols.
- For example, the entry **+Pl:s** pairs the lexical form **+Pl** with the surface form **s**.
- In other words, in forms constructed with this entry, plural is realized as **s**.
- The command '`compile-source`' converts the lexicon (ex4-lex.txt) into a transducer, which can be used bidirectionally for analysis ('`lookup`') as well as generation ('`lookdown`').
- Exercise: add *swim/swam/swum* to the lexicon.

Composing with rule transducers

- Let us consider augmenting our minilexicon with the present participle forms *dining*, *lining*, and *swimming*.
- Problem: the deletion of the stem final *e* in *dining* and *lining* and the gemination of *m* in *swimming*

  `dine:din`

  `line:lin`

  `swim:swimm`
- Exercise: write rules for this and compose them with the modified lexicon (include participles)
- Test: `lookup swimming`, `lookup dining`, `lookdown line+V+PresPart`

Cascade of compositions

- For languages with a more complex morphology than English, it may be practical to describe morphological alternations by two separate rule systems.
- The lexicon is composed with the first rule system and the resulting transducer is composed with the second set of rules.
- In a cascaded composition, the result net of the first composition becomes the source net for the second step.
- The command `'result-to-source'` makes that switch.
- When a new set of rules has been read in, the composition can be continued.
- At every point in the cascade, the result net maps the lexical forms to the output of the last rule or set of rules in the composition.

Checking the result

- Commands for single-item testing: `'random'`, `'lookup'`, and `'lookdown'`.
- The `'check-all'` command systematically compares the source net to the result net and displays the effect of the compose operation.
- It looks at every word in the source net and tries to find all words in the result net that have been derived from it by composition.
- Try it!

References:

Karttunen, L. (1993). Finite-state lexicon compiler. Technical report, Xerox Palo Alto Research Center, Palo Alto, California.

Karttunen, L. (2003). Finite-state morphology.