HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Building Real-World Finite-State Spell-Checkers With HFST

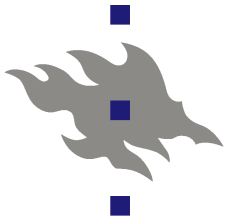
in FSMNLP 2012 at Donostia

Tommi A Pirinen

‹tommi.pirinen@helsinki.fi›

August 2, 2012

University of Helsinki



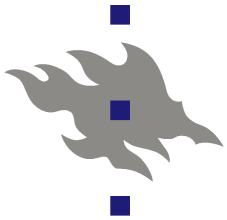
Outline

Introduction

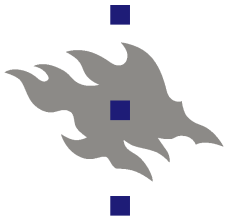
Language Models

Error Models

Misc. practicalities

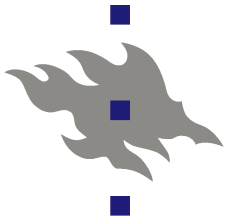


Outline



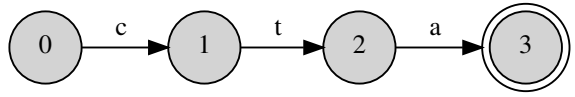
Finite-State Spell-Checking

- A simple task: go through all words in text to see if they belong to language LL, if not, modify them with relation EE to fit into language LL
- In Finite-State world language model LL is any (weighted) single tape finite-state automaton recognising the words of the language
- The error model EE is any (weighted) two-tape automaton, that describes spelling errors, i.e. mapping from misspelt word into the correct one
- In this tutorial we build simple WFST language model and piece together error model from different error types stored in FSTs

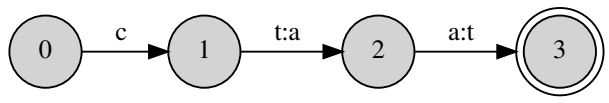


(Weighted) finite-state spell-checking graphically

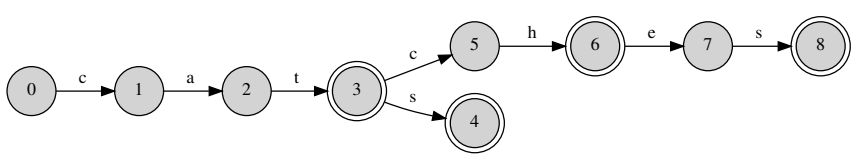
From this misspelled word as automaton:

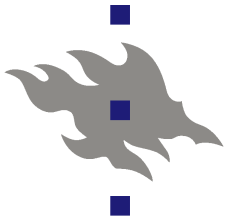


Applying this very specific error (correction) model:



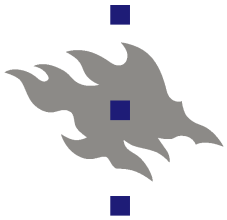
We can compose or intersect with a word in this dictionary:



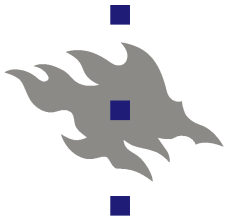


With HFST Tools (but mostly generic FST algebra)

- The tools we use to build the automata are from HFST project `http://hfst.sf.net`
- but most what I show here is finite-state algebra, which your favorite fst tools should support as well
- The chain of libraries to get HFST automata spellers to any desktop application is HFST ospell→voikko→enchant (with few exceptional application not supporting this).
  `https://kitwiki.csc.fi/twiki/bin/view/KitWiki/HfstUseAsSpellChecker`

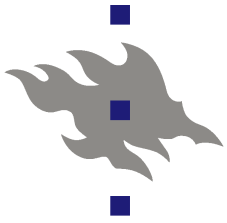


Outline



Easy baseline for language model—a word-list

- The most basic language model for checking if word is correctly written is a word-form list: `abacus, ..., zygotes`, right?
- Fast way to build such word-list now: grab an online text collection and separate it to a words
- Doing it like this, we also get: probabilities! $P(w) = \frac{c(w)}{CS}$
- In the end we'd probably mangle the probabilities into tropical logprobs that our wfst's use by default, say: $-\log P(w)$



Compiling corpus into a spell-checker with HFST

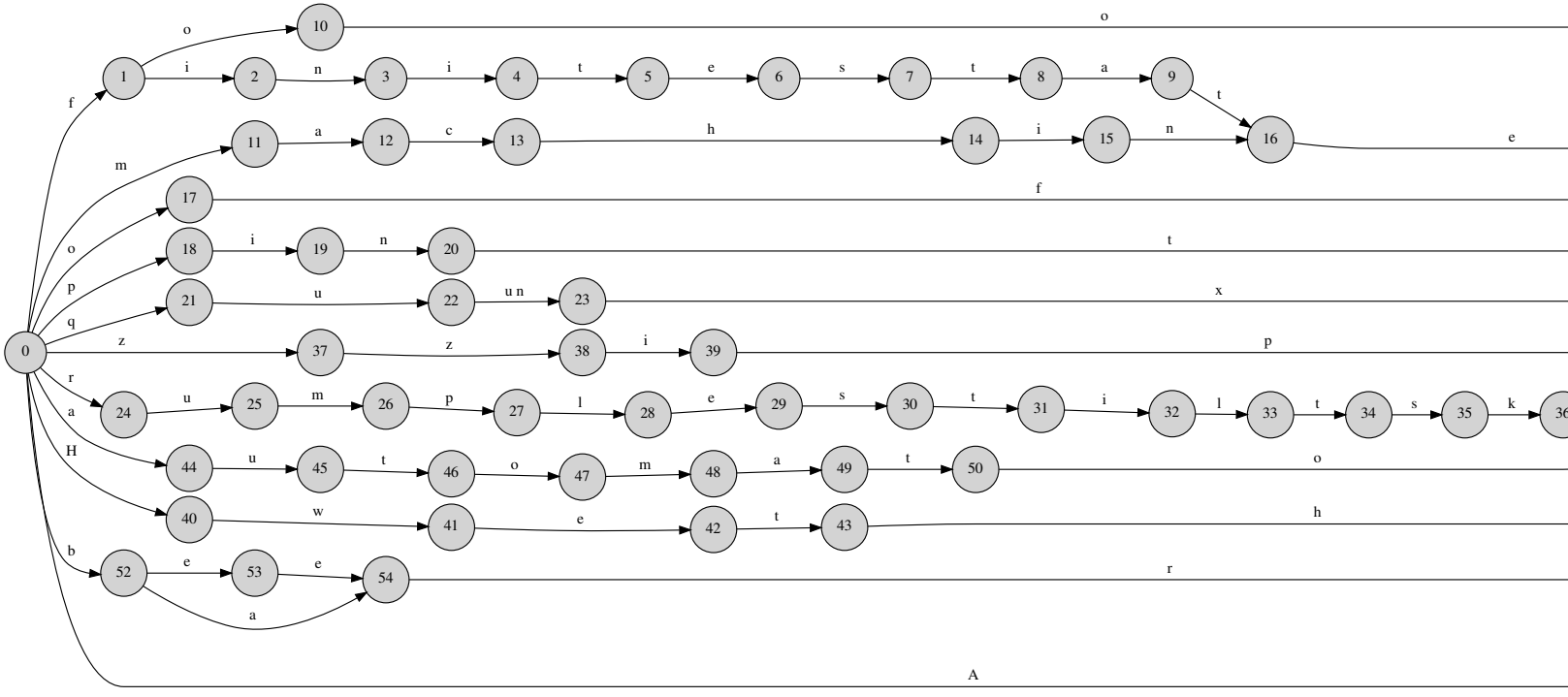
- A simple single command-line (no line breaks):

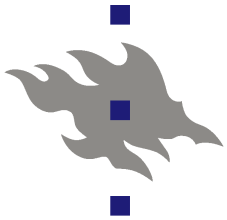
```
tr -d '[:punct:]'  <
demo-corpus.txt | tr -s
'[:space:]'  '\n' | sort | uniq -c
| awk -f tropicalize-uniq-c.awk
-assign=CS=20 | hfst-txt2fst -f
openfst -o demo-lm.hfst
```

- In order: clean punctuation, tokenize, sort, count, log probs and compile
- Let's try; you may grab corpus from , or Wikipedia dump, write one on the fly if you want to test it now

A language model as a FST

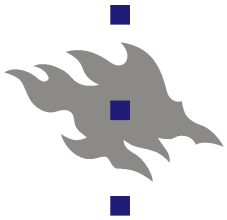
Resulting language model should look like this:



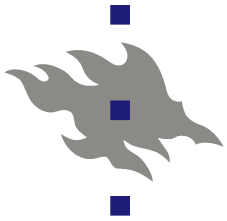


Other language models

- A word-form list is only good for very basic demoing for mostly isolating languages
- Since any given finite-state dictionary is usable, we can use—and have used—e.g.:
  - Xerox style morphologies (lexc, twolc, xfst) compiled with foma or hfst tools
  - apertium dictionaries
  - hunspell dictionaries—with a very kludgey collection of conversion scripts
- from a morphological analyser you may want to project off the analysis tape, we currently won't use it
- weighting language models afterwards from corpora is simple, cf. for example my past publications for HFST scripts of that

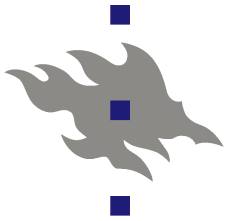


Outline



Building error models from scratch

- Error models are two-tape automata that map misspelled word-forms into a correctly spelled one
- Basic error types / corrections covered here: typing errors (Levenshtein–Damerau), phonemic/orthographic competence errors, common confusables word-forms
- Other relatively easy corrections not covered here: phonemic folding, errors in context. . .



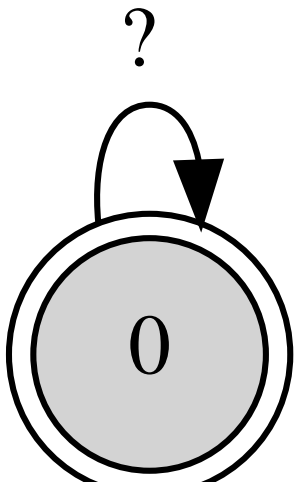
Building error models from scratch

- Since types of errors that can appear are very different, we build now separate automata components for each, and then connect them together
- Here is a good time to look back at the dictionary to see what the weights are and scale error weights accordingly using the Stetson-Harrison algorithm

Basic component 1: Run of correctly spelled characters

We assume most misspelled words will have most of the letters correct, to account these parts we need the identity automaton:

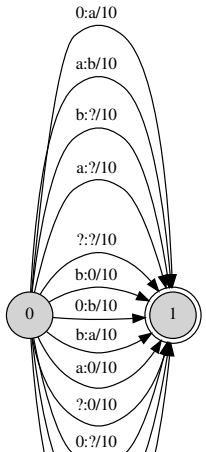
```
echo '?*' | hfst-regexp2fst -o
anystar.hfst
```

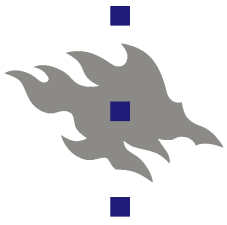


Basic component 2: Typing error

A single typing error, as in popular Levenshtein measure, is a) removal, b) addition or c) change of a character, using fixed alphabet for example with a and b, and let's say this error is less unlikely than any other for all combinations (= 10):

```
echo 'a:0::10 | 0:a::10 | a:b::10 |
b:0::10 | 0:b::10 | b:a::10' |
hfst-regexp2fst -o edit1.hfst
```

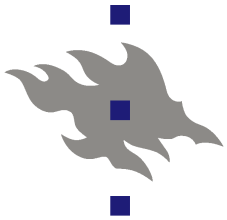




Excursion 1: Swap in Levenshtein-Damerau measure

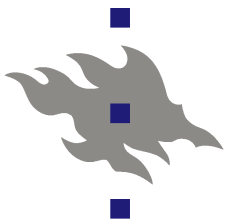
The fourth type of error in edit distance algorithm, transposition or swapping adjacent characters, is useful, but makes the automaton a bit messy and tedious to write (which is why we have few scripts to generate them). This is the additional component, that must be added *per each pair of characters in alphabet*:

```
echo 'a:b::10 (b:a) | b:a::10 (a:b) |
hfst-regexp2fst
```

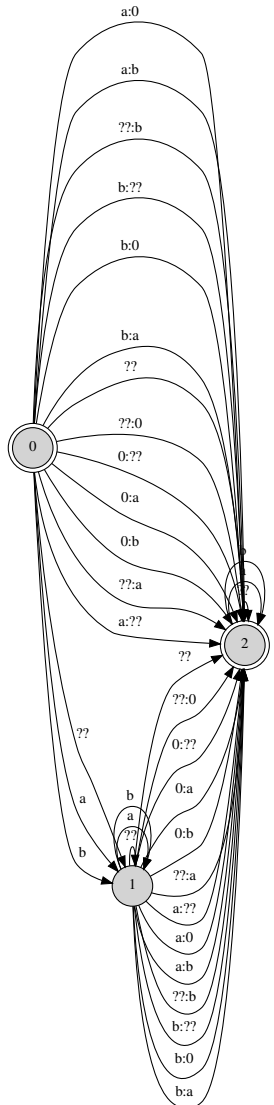


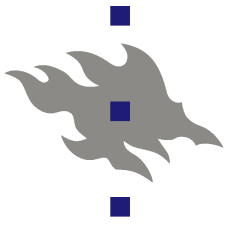
Edit distance automata

- Combining the correctly written parts and a single edit gets us a nice baseline error model:
```
hfst-concatenate anystar.hfst
edit1.hfst | hfst-concatenate -
anystar.hfst -o errm1.hfst
```
- Since it's just an automaton, all regular tricks of FST algebra just work, particularly:
```
hfst-repeat -f 1 -t 7 -i
errm1.hfst -o errm7.hfst
```
- (A script to automate writing of the combinations of swaps: `python editdist.py -d 1 -S -a demo.hfst | hfst-txt2fst -o edit1.hfst`)



Edit distance automaton graphically

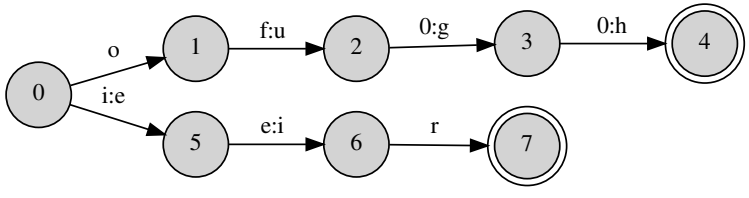


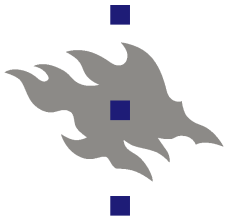


Common orthography-based competence errors

English orthography sometimes leads to errors that are not easily solvable by basic edit distance, so we can extend the edit distance errors by set of commonly confused substrings (N.B. input to strings2fst expects TAB between strings and weight 5 i.e. bit more likely than before):

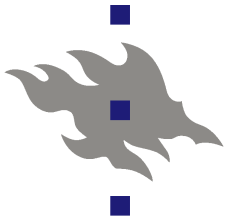
```
hfst-strings2fst -j -o en-orth.hfst
of:ough 5
ier:eir 5
```





Edit Distance Plus Orthographic Errors

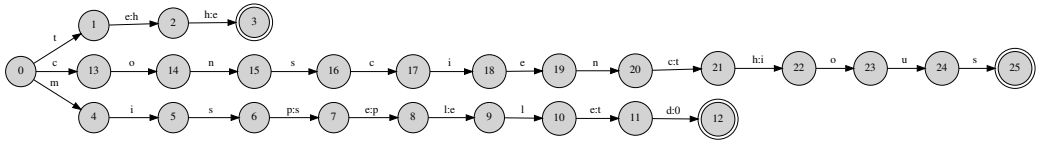
- Combine orthographic errors with edit1 error before making the full-word covering error-model: `hfst-disjunct edit1.hfst en-orth.hfst -o edit1+en-orth.hfst`
- And rest as before: `hfst-concatenate anystar.hfst edit1+en-orth.hfst | hfst-concatenate - anystar.hfst -o errm.edit1+en-orth.hfst`

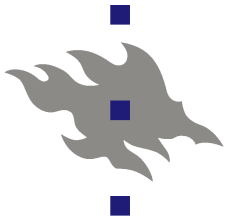


Confusion set of words

Very simple, yet very important piece of error model, common word-specific mistakes, of course compiled from a list of mistakes:

```
hfst-strings2fst -j -o
en-confusables.hfst
teh:the 2.5
mispelled:misspelt 2.5
conscienchous:conscientious 2.5
```

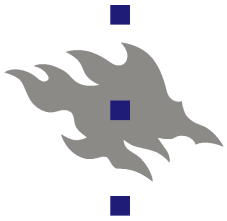




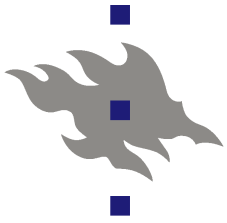
Finally: hacking all these together

- adding one more component to the mixture; the full-word errors can be just disjuncted to any existing full error model: `hfst-disjunct en-confusables errm.edit1+en-orth.hfst -o errm.everything.hfst`
- should your language happen to use productive compounding you may wish to `hfst-repeat -f 1` the full-word errors before disjuncting

(The resulting automaton is too much for graphviz already)

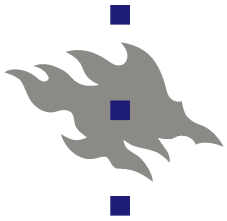


Outline



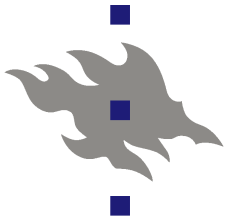
Getting it used in real-world applications

- With help of few libraries it's relatively easy to get these automata to any typical desktop application
- libvoikko was originally for Finnish spell-checking based on malaga (lag) grammars, and it still requires that format of metadata present. The installation location is also inherited from here
- HFST ospell file format was specced openly on mailing list with few FST people—result: XML-based metadata in zip archive with very specific filenaming conventions.



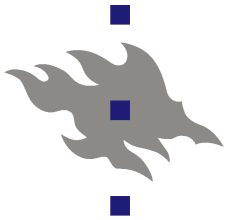
Metadata

- Two parts of metadata: `voikko-fi_FI.pro` file saying language code and few other details in MIME-headerish format (line-based, key: value)
- XML metadata about automatons, how to lay out dictionaries and error models, also lot of UI metadata like names and descriptions
- The most meaningful setting in these files is the language code, it must match the locale settings and the name of the locale in program settings where applicable

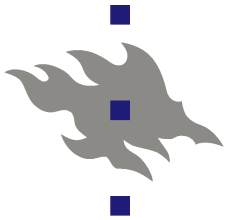


voikko-fi_FI.pro

```
info: Voikko-Dictionary-Format: 2
info: Language-Code: fi_FI
info: Language-Variant: hfst
info: Description: Finnish HFST dictionary
info: Morphology-Backend: hfst
info: Speller-Backend: hfst
info: Suggestion-Backend: hfst
info: Hyphenator-Backend: hfst
```

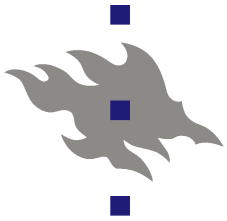


```
<hfstspeller dtdversion="1.0" hfstversion=
  <info>
    <locale>fi</locale>
    ...
  </info>
  <acceptor type="general" id="acceptor.de
    ...
  </acceptor>
  <errmodel>
    ...
  </errmodel>
</hfstspeller>
```



Zipping and Installation Location

- The dictionary or language model needs to be stored in `acceptor.default.hfst` (configurable) in special format:
  ```
  hfst-fst2fst -f olw -i
  demo-lm.hfst -o
  acceptor.default.hfst
  ```
- The error models are `errmodel.default.hfst` and likewise in special format: `hfst-fst2fst -f olw -i errm.everythin.hfst -o errmodel.default.hfst`
- The XML metadata goes into index.xml, then zip it to `speller.zhfst` like so:
  ```
  zip -9 speller.zhfst
  ```



URLs and references

- http://hfst.sf.net
- https://kitwiki.csc.fi/twiki/bin/view/KitWiki/HfstUseAsSpellChecker
- http://hfst.svn.sourceforge.net/viewvc/hfst/trunk/articles/fsmnlp-2012-spellers/
- http://www.helsinki.fi/~tapirine/publications/
- http://divvun.no/future/proofing/lexfile-spec.html
- tommi.pirinen@helsinki.fi