Compiling Apertium dictionaries with HFST

leveraging generalised compilation formulas to get more and better end applications with fewer language description

Tommi A Pirinen, Francis Tyers
tommi.pirinen@helsinki.fi

University of Helsinki, Universitat d'Alacant

May 22, 2012

Outline

Finite-state automata and HFST and apertium

- Finite-state automata are one efficient way to encode dictionaries, morphological analysers etc.
- HFST stands for Helsinki Finite-State Technology— consisting of a library working as a compatibility layer between different open-source finite-state implementations,
  - SFST
  - OpenFST
  - Foma
- Also a set of finite-state tools built on top of the library, and set of end products using the automata in real-world applications (sold separately)
- HFST is still a research project in a computational linguistics' research group—not computer science or engineering
- apertium is a machine-translation platform that uses finite-state dictionaries

Compiling apertium dictionaries with HFST—rationale

- "just an engineering exercise"
- getting all language descriptions to compile natively in HFST (as opposed to converting compiled automata)
- using existing (and future) HFST algorithms to improve the resulting automata
- using bits of *linguistic* information to get better auxiliary automata for HFST end applications — data that may not be possible to induct from converted compiled automata
- possibility to integrate more complex features in of finite-state morphology in apertium dictionaries—morphophonetics, reduplication etc. that may be supported by other HFST tools
- this paper fits nicely in my PhD thesis under "State of the art of in language models"

Examples of immediate benefits to dictionary writers

- A lot of current work in building NLP software involves management of huge amounts of lexical data
- ...like generating different language models in different *morphology* programming formalisms: apertium, hunspell, xerox tools
- getting native and uniform compilation formulas for all lets you write dictionaries once and use everywhere
- or pick and mix tools and features from different formalisms

Examples of additional applications that can be generated from apertium dictionaries with this work

- Spell-checkers! A basic spell-checker with generic edit distance suggestion generator can be automatically generated—and used in majority of current open-source software without any extra effort
- Predictive text entry, for mobiles, such T9, XT9, possibly swype and keyboard as well
- Morphological analysers, lemmatisers, segmenters, tokenisers, etc., obviously

Examples of benefits that come for free—automatic optimisation

- depending on library / end format you choose for compiled dictionaries, you get speed–space tradeoffs (or improvements in both)
- This is work-in-progress, but once done it can be used in all dictionaries without modifications to sources
- automatic flag diacritic induction
- hyperminimisation
- all this can be based on things like finding homomorphic components from the finite-state automaton
- the linguistic concepts present in source code but missing from the compiled automaton should prove very useful here!

What now?

- The reference material for the article is in our svn `http://hfst.svn.sf.net/svnroot/hfst/trunk/lrec-2011-apertium`, includes compilation of spell-checkers for most apertium dictionaries
- what do we do to remove duplicate work, duplicate versions of dictionaries, conversion scripts...
- more compilers? Conversion scripts? New programming languages? New "standards" that everyone will use?
- I'll throw you this: I need more linguistic data and less engineering in the language model implementations to compile more applications from one source dictionary. Example: LR/RL concept in apertium or asymmetric flags in Xerox FSM is engineering hack POV; had the description called it *substandard* or *dialectal* word form it would already be usable in all applications!