Omorfi—Free and open source morphological lexical database for Finnish

Tommi A Pirinen
Ollscoil Chathair Bhaile Átha Cliath
ADAPT Centre — School of Computing
Dublin City University, Dublin 9
tommi.pirinen@computing.dcu.ie

Abstract

This demonstration presents a freely available open source lexical database omorfi. Omorfi is a mature lexicographical database project, started out as a single-person single-purpose free open source morphological analyser project, omorfi has since grown to be used in variety of applications including spell-checking, statistical and rule-based machine translation, treebanking, joint syntactic and morphological parsing, poetry generation, information extraction. In this demonstration we hope to show both the variety of end-user facing applications as well as the tools and interfaces for computational linguists to make the best use of a developing product. We show a shallow database arrangement that has allowed a great variety of contributors from different projects to extend the lexical database while not breaking the continued use of existing end-applications. We hope to show both the best current practices for lexical data management and software engineering with regards to continuous external project integration of a constantly developing product. As case examples we show some of the integrations with following applications: Voikko spell-checking for Windows, Mac OS X, Linux and Android, statistical machine translation pipelines with moses, rule-based machine translation with apertium and traditional xerox style morphological analysis and generation. morphological segmentation, as well as application programming interfaces for python and Java.

1 Introduction

Omorfi[1] (open morfology for Finnish), as a project is centred around morphological analysis of Finnish. Morphology is a core for many if not most natural language processing systems, especially in the case of such morphology-heavy language as Finnish. However, the detail and even the formatting of the result of morphological processing varies from application to application. In order to produce all the different formats of output and details of data, one needs to maintain a large database of all the bits and pieces of lexical information that is necessary to produce the wanted readings, and that is exactly what omorfi has become over the years. What we have in the current version of omorfi, is a lexical database of roots, morphs and combinatorics, that can be *weaved* for use of different applications, documentations, and automatic test suites by use of simple scripting. The database is easy to maintain and update for linguists and contributors. It is robust enough to support a wide range of applications without the progress of each application and changes to their specific data bearing a negative effect to other applications.

2 Database

The word *database*, in context of omorfi is currently used in a very liberal sense, while we have structured our data in a manner that resembles relational database to the extent that it could be converted into a one by quite simple and fast process, we have opted to stick with basic *tab-separated-values* format for basically two reasons: firstly, computational linguists and computer scientists are already well-versed to handle this type of files with ease and efficiently on the command-line, they integrate with the basics of unix taught to any computational linguist in the past 20 years or

[1] https://github.com/flammie/omorfi/

so (Church, 1994), and the expected improvement in the use of real relational databases comes from the complexity of dozens of tables and billions of rows of data, whereas lexicon will likely not reach even million of root-words any time soon. For easy editing the TSV files are importable and exportable to all the commonly available free office products: e.g., LibreOffice[2] and OpenOffice.

3 Application Data Format Generation

The application of morphs and morphotactics is based on *finite state morphology* as documented by Beesley and Karttunen (2003). In order to be able to compile our lexical database into a finite-state automaton format for efficient processing, we generate a *lexc* file representation of the data, and compile it using *HFST* (Lindén et al., 2011) software that is available as a free and open source system. The translation from tab-separated-values into lexc is done by python scripts, an example of formats is in listings 1 and 2. Transformation is pretty straight-forward and easy to maintain.

```
lemma   homonym new_para    origin
Aabel   1       N_STADION   unihu
...
talo    1       N_TALO      kotus
...
Öösti   1       N_TYYLI     unihu
```

Figure 1: Lexical data in lexeme database, consisting of a *unique key* of lemma and homonym number, a paradigm for inflection and source of origin, which is all the obligatory information for each word to be added in the database. The origin's main importance is copyright's rather than linguistic information or database structure; in origins *unihu* refers to a yet unnamed project in University of Helsinki and *kotus* stands for Nykysuomen sanalista by kotus (RILF; research institute of languages in Finland)

The additional data can be added to each lexeme using the lemma and homonym number as an identifier, these data are stored in a separate TSV file that is joined to the database in figure 1 to produce a master database. E.g., a named-entity recognition project would add lines from `Aabel 1 first` to `Öösti 1 geo` into a file of named entity classes to add "first name" information to *Aabel* and "geographical place" information to *Öösti*. Then it can be accessed e.g., to generate readings of `PROPER=FIRST` and `PROPER=GEO` into the lexc code to be added to a specific named-entity categorising automaton, or the master database can be queried for this information when the given lemma is seen in the analysed text.

[2]http://libreoffice.org

4 Applications

The applications we are demonstrating in this paper are: statistical machine translation, rule-based machine translation, spell-checking and correction, morphological segmentation, analysis and generation.

For statistical machine translation, we are using moses[3] (Koehn et al., 2007). There are at least two ways morphological processing can be used in moses pipeline:*segmentation* (Dyer et al., 2008) and *factorisation* (Koehn and Hoang, 2007). In segmentation approach, the word-forms are reduced to smaller units, such as morphs, or just words (i.e. root morphs with sufixes intact but compounds broken), applying traditional statistical machine translation methods to morphs is supposed to improve the translation quality by decreasing the amount of unseen tokens and matching the morphs to many non-Finnish word-forms more regularly (e.g., aligning suffixes to preposition). In factored translation, results of morphological analysis are stored in a vector, each component of which can be used at any point of statistical machine translation: typical components of the vector are e.g., lemma, part-of-speech and full morphosyntactic description.

For rule-based machine translation setting we combine omorfi with apertium[4] (Forcada et al., 2010). In apertium's shallow rule-based machine translation, omorfi is used for morphological analysis, disambiguation and morphological generation.

For spell-checking we use voikko.[5] In spell-checking and correction omorfi is used to locate misspelled words and to find most likely corrections given mispelling (Pirinen and Lindén, 2014).

The morphological analysis and segmentation are tasks that the above-mentioned end-user programs depend on, but we also provide an API ac-

[3]http://www.statmt.org/moses/
[4]http://apertium.sf.net
[5]http://voikko.sf.net

```
LEXICON Nouns
[WORD_ID=Aabel][POS=NOUN][PROPER=FIRST]:0Aabel N_STADION       ;
...
[WORD_ID=talo][POS=NOUN]:0talo  N_TALO  ;
...
[WORD_ID=Öösti][POS=NOUN][PROPER=GEO]:0Ööst  N_TYYLI ;
```

Figure 2: Lexical data in lexc-compatible format for compilation.

cess for python and Java as well as convenience bash scripts on top of direct access to the automata.

Given this wide array of applications it is obvious how importance of lexical data management has gotten to more central position as the project has progressed: maximal coverage of word-forms for machine translation is not invariably a good thing for spell-checker.

Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement PIAP-GA-2012-324414 (Abu-MaTran).

Omorfi consists of a large body of work from numerous academic and open source contributors, including: Inari Listenmaa, Francis M Tyers, Ryan Johnson, and Juha Kuokkala.

References

Kenneth R Beesley and Lauri Karttunen. 2003. Finite-state morphology: Xerox tools and techniques. *CSLI, Stanford.*

Kenneth Ward Church. 1994. Unix™ for poets. *Notes of a course from the European Summer School on Language and Speech Communication, Corpus Based Methods.*

Christopher Dyer, Smaranda Muresan, and Philip Resnik. 2008. Generalizing word lattice translation. Technical report, DTIC Document.

Mikel L. Forcada, Mireia Ginestí Rosell, Jacob Nordfalk, Jim O'Regan, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Gema Ramírez-Sánchez, Felipe Sánchez-Martínez, and Francis M. Tyers. 2010. Apertium: a free/open-source platform for rule-based machine translation platform. *Machine Translation.*

Philipp Koehn and Hieu Hoang. 2007. Factored translation models. In *EMNLP-CoNLL*, pages 868–876.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 177–180. Association for Computational Linguistics.

Krister Lindén, Erik Axelson, Sam Hardwick, Tommi A Pirinen, and Miikka Silfverberg. 2011. Hfst—framework for compiling and applying morphologies. *Systems and Frameworks for Computational Morphology*, pages 67–85.

Tommi A Pirinen and Krister Lindén. 2014. State-of-the-art in weighted finite-state spell-checking. In *Computational Linguistics and Intelligent Text Processing*, pages 519–532. Springer.