Modularisation of Finnish Finite-State Language Description—Towards Wide Collaboration in Open Source Development of Morphological Analyser

Tommi A Pirinen
University of Helsinki
Helsinki, Finland
tommi.pirinen@helsinki.fi

Abstract

In this paper we present an open source implementation for Finnish morphological parser. We shortly evaluate it against contemporary criticism towards monolithic and unmaintainable finite-state language description. We use it to demonstrate way of writing finite-state language description that is used for varying set of projects, that typically need morphological analyser, such as POS tagging, morphological analysis, hyphenation, spell checking and correction, rule-based machine translation and syntactic analysis. The language description is done using available open source methods for building finite-state descriptions coupled with autotools-style build system, which is de facto standard in open source projects.

1 Introduction

Writing maintainable language descriptions for finite-state systems has traditionally been a laborious task. Even though finite-state technology has been de facto standard for writing computational language descriptions for more than two decades now (Beesley and Karttunen, 2003), it has some recognised flaws and problems both caused by shortcomings of actual implementations and background technology (Wintner, 2008). Commonly language description is performed by a single linguist or language technologist. The descriptions typically wind up being complex enough that modifying them requires a great amount of studying and understanding before one is able to do the smallest of modifications to the system. In the current times that all proper, healthy, scientific projects should be open source and globally developed, this poses a challenge for such project's internal structure. Another source of problem in such collaboration is that background of contributors for language description varies from computer scientists to linguists (Maxwell and David, 2008) to computer-savvy native language speakers, all of whom should be able to contribute to the project. The solutions we propose for this is to embrace proper modularisation in language descriptions to allow multiple specific entry points to contributors.

In this paper we describe a new implementation of the Finnish language description called omorfi[1], made to support large variety of NLP applications and different audiences. While the background theory for implementing finite-state description of Finnish was laid out already in Koskenniemi (1983), and morphophonological system does not have significant changes, the actual system was rewritten from the scratch. The rewriting was originally done by single linguist as usual, in a master's thesis project (Pirinen, 2008), but afterwards it has been extended as full-fledged open source project and used in various contexts. This extended development has necessitated a better modularised framework to allow people of varying level of familiarity with finite-state technology and Finnish to contribute on their prospective parts of the description without causing of modularisation problems for other applications of the finite-state analysers.

The projects that have used and use omorfi as language description include spell checking and correction (Pirinen and Lindén, 2010b), lemmatising for IR applications (e.g. Kurola (2010)), named entity recognition, rule-based machine translation Forcada et al. (2010)[2], and syntactic disambiguation and analysis. The demands for even the basic morphology with all these different applications are very different with regards to productivity; lexical coverage and accuracy as well as depth

[1]http://home.gna.org/omorfi
[2]http://www.apertium.org/

of tagging, so it has became obvious that no one lexical automaton will work for everyone. For this reason the modularisation has to provide easily configurable options and modifiability for all end-points.

One of the key points in modular structure here is that we ensure that modifying will not typically break already working parts, so contributors adding new words or moving hyphens will not cause problems in other parts of description as much as possible.

2 Modularisation of Finite-State Language Description

The modularisation scheme we ended up with in finite-state description of Finnish has grown organically around rather standard description of finite-state morphology. The further development followed from development of finite-state technology along years from initial implementation of omorfi at publication of Pirinen (2008).

In omorfi we use a hierarchical set of abstract modules implemented to encapsulate the system. As mentioned, the classical modules of morphotactic combinatorics (i.e. Xerox compatible lexc language description) and morphophonology (i.e. Xerox compatible twolc description) is still present. The morphotactic combinatorics has already been split to sub modules for two reasons. First is primarily practical fact that code base for morphotactic combinatorics for words of Finnish is huge. Second and perhaps the more important distinction is the fact that central and integral part of the life force of morphophonological description of the all languages is to keep up with constant influx of new lexical items to the language; neologisms, proper nouns and other coinages. From further new modules, orthographical variations was implemented to create detached support for certain obvious variations of Finnish written data, e.g. the typewriter and SF7-ASCII era digraphs like sh and zh in stead of š and ž respectively. The hyphenation and syllabification of Finnish language is also one obvious service for morphological dictionary to provide; for Finnish the compound boundaries cannot typically be discriminated without a dictionary (Lindén and Pirinen, 2009a). One of the features that has become rather obvious along years for morphological parsers is the fact that all computational linguistic applications must require their own very special version of morphological analysis, so in omorfi we have chosen to avoid lock-ins for any specific type of *tag sets*, and instead go for one version of analysis to contain certain superset of all needed forms and rewrite as needed. The statistical models is one of recent developments of finite-state technology, and there is a lot to offer for language models here so the whole family of weighted finite-state training and models is also implemented in omorfi as a separate module here, which for most intents and purposes does work independently of any other part of the language description.

3 Implementation

Here we briefly discuss implementation of modules, mainly to discuss about practices that help the cooperation. Naturally full discussion of the modules is found in the documentation of the system[3]. The system implementation is harnessing the autotools framework and unix style tools of HFST to incrementally build the finite-state automata using finite-state algebra, such as composition to extend them, originally noted even in Beesley and Karttunen (2003). The crucial thing for this modular approach is that it can be applied incrementally, each module can be replaced or disabled entirely at needs of end application, and with autotools framework all this can be performed by simple command-line switch to `./configure`.

3.1 Lexical Data—Lexicon and Features of Lexical Items

The initial part of morphotactics deals with lexicographical data. This is the part where most modification and cooperation can be used, the lexical items in language change all the time in introduction of new word forms, and the expertise needed to extend the lexicon does not require significant expertise beyond understanding of the language. For this case we provide different entry formats for new lexical data; csv, XML and so on. The minimal data to enter for new word form is morphological part of speech. Additionally a paradigm classification is typically needed for working inflection and derivation. While this is facilitated as much as possible, further research for easy lexicon management is still required.

The other practical example as to why easy modification of lexical data is crucial is that for example for rule-based machine translation benefits

[3] `http://home.gna.org/omorfi/`

from mapping between lexical units of source language and target language. Similarly for forthcoming syntactic constraint grammar work in vein of (Karlsson, 1990). For this reason easy access to lexical units is required for users of morphology.

3.2 Traditional Morphotactics and Phonology—The Lexc and Twolc Model

The various lexical data sources are joined back to traditional lexc format, which is combined with word stem variation definitions and inflectional data to produce lexical automaton. This is compose-intersected with morphophonology descriptions to produce the analyser already; as these parts rarely need changes beyond bug-fixes and are unlikely to benefit from open source cooperation beyond initial linguist work, they are still in same form as traditional finite-state morphology by Beesley and Karttunen (2003), even if it was deemed monolithic and fragile for such collaboration.

3.3 Analysis Formats and Sets

Another thing that is quickly obvious for interoperability is that all projects using morphological analyser, for whatever purpose, require their own analysis format. Instead of converging to standard we have temporarily solved this by making our analyser to contain superset of required features at all times, and providing rules to rewrite the tagsets. The rulesets can be compiled to finite-state networks and composed like usual. Typical rules are of course relatively simple contextless rewriting, for example the annotation for singular nominative is `+sg+nom` or `<sg><nom>`, for different applications, so a simple composition in style of `NUM=SG:+sg` is enough for providing the singular nominatives to that analysis style. Ideally of course this would be solved by using more suitable abstract data type for the annotations than character string (Wintner, 2008), ideally derived from standardized set of features, such as ISOCat as is also suggested by (Maxwell and David, 2008).

3.4 Orthographical variations

When dealing with data from various sources, such as old literature or *spoken* standard language found in instant messaging etc., there are certain variations on spelling rules. These has also been implemented as independent rule set compiled to composable finite-state automaton. Incidentally both mapping of typewriter digraphs sh and zh to š and ž correspondingly and omission of final component of i-final diphthong of spoken language are both definable as rule working on morphological analyser as an independent unit.

3.5 Hyphenation and syllabification

Hyphenation is in practice also one of the applications of the language. It has been defined as a rule set over half-build morphological analyser, since it can neatly abuse build-time information of the analyser, such as word and morpheme boundaries. The syllables could also conversely be used by other parts of the description if needed.

3.6 Error models

Error model is a crucial part of spell-checking system, for the correction task. This is implemented as finite-state filter that can be applied with on-the-fly composition (Pirinen and Lindén, 2010a) to perform the error correction for spell checking, or for example error-tolerant analysis.

3.7 Statistical models

Statistical models provide for disambiguating language models and spell-checking tasks for example. The statistical models used are simple finite-state automata or training sets combinable to the language description by use of composition (Lindén and Pirinen, 2009a; Lindén and Pirinen, 2009b).

3.8 Filtering the Analyser

The models needed for different task may need widely different dictionaries and allowed word-forms, and not always the statistical models are sufficient to discriminate between good word forms. So we also provide filter rule sets, to limit features, such as derivation and compounding, and lexical units, such as archaic or dialectal words. For example for the spell-checker's error detection lexicon or information retrieval task compounding and derivation can be largely allowed, whereas in the spelling correction the suggestions should be relatively conservative for plausible but non-existing compounds and derivations.

4 Discussion and Future Work

In this article we have showed that finite-state description can be implemented in modularised manner enabling wide cooperation in the open source context for people with varying background. Furthermore we have demonstrated the ease of proper abstraction in finite-state language description using easily available open source tools while still providing open source community with the de facto standard build system of autotoolset for wide distribution, packaging and deployment.

What we did not address here is the easy way of coupling up-to-date documentation with our modularised language description. The next step to research is to see into integrating the notion of literate programming in this framework. This topic has already been widely researched by Maxwell and David (2008), specifically in case of finite-state language descriptions.

Acknowledgements

We thank Donald Killian for pointing us towards the ongoing discussion about shortcomings of finite-state morphologies and the HFST research group, and our colleagues for fruitful discussions.

References

Kenneth R Beesley and Lauri Karttunen. 2003. *Finite State Morphology*. CSLI publications.

Mikel L. Forcada, Mireia Ginestí i 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. to appear*.

Fred Karlsson. 1990. Constraint grammar as a framework for parsing unrestricted text. In H. Karlgren, editor, *Proceedings of the 13th International Conference of Computational Linguistics*, volume 3, pages 168–173, Helsinki.

Kimmo Koskenniemi. 1983. *Two-level Morphology: A General Computational Model for Word-Form Recognition and Production*. Ph.D. thesis, University of Helsinki.

Joel Kurola. 2010. Työpaikkailmoitusten sisällön ja osaamisvaatimusten käsittely. Bachelor's thesis (in Finnish).

Krister Lindén and Tommi Pirinen. 2009a. Weighted finite-state morphological analysis of finnish compounds. In Kristiina Jokinen and Eckhard Bick, editors, *Nodalida 2009*, volume 4 of *NEALT Proceedings*.

Krister Lindén and Tommi Pirinen. 2009b. Weighting finite-state morphological analyzers using hfst tools. In Bruce Watson, Derrick Courie, Loek Cleophas, and Pierre Rautenbach, editors, *FSMNLP 2009*, 13 July.

Michael Maxwell and Anne David. 2008. Joint grammar development by linguists and computer scientists. In *Workshop on NLP for Less Privileged Languages, Third International Joint Conference on Natural Language Processing*, Hyderabad, India.

Tommi A Pirinen and Krister Lindén. 2010a. Building and using existing hunspell dictionaries and TEX hyphenators as finite-state automata. In *Proccedings of Computational Linguistics - Applications, 2010*, pages 25–32, Wisła, Poland.

Tommi A Pirinen and Krister Lindén. 2010b. Finite-state spell-checking with weighted language and error models. In *Proceedings of the Seventh SaLTMiL workshop on creation and use of basic lexical resources for less-resourced languagages*, pages 13–18, Valletta, Malta.

Tommi Pirinen. 2008. Suomen kielen äärellistilainen automaattinen morfologinen analyysi avoimen lähdekoodin menetelmin. Master's thesis, Helsingin yliopisto.

Shuly Wintner. 2008. Strengths and weaknesses of finite-state technology: A case study in morphological grammar development. *Nat. Lang. Eng.*, 14:457–469, October.