Tool Demonstration: Functional Morphology

Markus Forsberg & Aarne Ranta

Department of Computing Science
Chalmers University of Technology and the University of Gothenburg
SE-412 96 Gothenburg, Sweden
{markus, aarne}@cs.chalmers.se

1 System Description

We will present *Functional Morphology*[1] [5], abbreviated *FM*, which is a tool that implements a methodology for constructing natural language morphologies in the functional language Haskell [8]. FM has its own runtime system that supports morphological analysis and synthesis. Moreover, a morphology implemented in FM can be compiled to many other source formats.

FM adopts a *word-and-paradigm* view of morphology: it represents a morphology as a set of inflection tables, *paradigms*, and the lexicon as a set of dictionary words each tagged with a pointer to an inflection table.

The basic idea behind FM is simple, instead of working with untyped regular expressions, which is the state of the art of morphology in computational linguistics, we use finite functions over hereditarily finite algebraic data types. These data types and functions constitute the language-dependent part of the morphology. The language-independent part consists of an untyped dictionary format which is used for synthesis of word forms, translations to other formats, and a decorated trie, which is used for analysis.

Functional Morphology builds on ideas introduced by Huet [6] in his computational linguistics toolkit Zen, which he has used to implement the morphology of Sanskrit. In particular, Huet's ideas about sandhi in Sanskrit have been adopted to a language independent description of compound analysis in FM.

The goal of FM has been to make it easy for linguists, who are not trained as functional programmers, to implement the morphology of a new language. In addition to the ease of programming, FM attempts to exploit the high level of abstraction provided by functional programming to make it possible to capture linguistic generalizations.

A morphology written in FM has a type system, which defines the inflectional and inherent parameters of the language described. By using algebraic data types, the type system can guarantee that no spurious parameter combinations appear in the morphology description, at the same time as all meaningful combinations are defined.

The use of the functional language Haskell provides, besides typing, many other language features that simplify the development of a morphology: *higher-order functions*, functions as first class objects, give the possibility of defining

[1] FM homepage: `http://www.cs.chalmers.se/~markus/FM/`

a paradigm in terms of another paradigm; the *class system* is used for sharing code between morphologies of different languages.

The lifetime of a digital linguistic resource such as morphology depends on which system it has been developed in [3]. If the resource has been developed in a proprietary system with a binary-only format, and the system is no longer supported after some years, it may be impossible to access the resource. FM offers a solution to this problem by supporting translation to a multiple of different formats, such as XFST source code [2], GF [10] source code, SQL source code, full form lexicon, full form tables etc. This feature will hopefully prolong the lifetime of a morphology developed in FM. Furthermore, the system is completely open source, which should improve the situation even more.

2 Results

The following morphologies have been implemented in Functional Morphology: a Swedish inflection machinery and a lexicon of 20,000 words; a Spanish inflection machinery + lexicon of 10,000 words [1]; major parts of the inflection machinery + lexicon for Russian [4], Italian, Estonian [7], and Latin. Comprehensive inflection engines for Finnish, French, German, and Norwegian have been written following the same method but using GF as source language [9]. Since FM can generate GF source code, there exists a seamless connection between GF grammars and morphologies defined in FM.

References

1. I. Andersson and T. Söderberg. Spanish Morphology – implemented in a functional programming language. Master's Thesis in Computational Linguistics, 2003. `http://www.cling.gu.se/theses/finished.html`.
2. K. R. Beesley and L. Karttunen. *Finite State Morphology.* CSLI Publications, Stanford University, United States,, 2003.
3. S. Bird and G. Simons. Seven dimensions of portability for language documentation and description. *Language*, 79:557–582, 2003.
4. L. Bogavac. Functional Morphology for Russian. Master's Thesis in Computing Science, 2004.
5. M. Forsberg and A. Ranta. Functional Morphology. *Proceedings of the Ninth ACM SIGPLAN International Conference of Functional Programming, Snowbird, Utah*, pages 213–223, 2004.
6. G. Huet. The Zen Computational Linguistics Toolkit. `http://pauillac.inria.fr/~huet/`, 2002.
7. M. Pellauer. A Functional Morphology for Estonian. Term Paper, 2005.
8. S. Peyton Jones and J. Hughes. Report on the Programming Language Haskell 98, a Non-strict, Purely Functional Language. Available from `http://www.haskell.org`, February 1999.
9. A. Ranta. Grammatical Framework Homepage, 2000–2004. `http://www.cs.chalmers.se/~aarne/GF/`.
10. A. Ranta. Grammatical Framework: A Type-theoretical Grammar Formalism. *The Journal of Functional Programming*, 14(2):145–189, 2004.