CLARIN and Free Open Source Finite-State Tools

Kimmo Koskenniemi and Anssi Yli-Jyrä

Department of General Linguistics
P.O.Box 9, Siltavuorenpenger, room 328, FIN-00014
University of Helsinki, Finland
kimmo.koskenniemi@helsinki.fi, aylljyra@cc.helsinki.fi

CLARIN stands for Common Language Resources and Technologies Research Infrastructure and it is one of the 35 infrastructure projects listed in the ESFRI roadmap of European research infrastructures for various areas. CLARIN has now entered its 3 year preparatory phase under a grant from the EU Commission. The preparatory phase of CLARIN has 32 partner organizations, (see www.clarin.eu for more details).

There are quite a number of language resources around in Europe consisting of text and speech corpora with possible annotations, lexical materials, standards and norms and programs for parsing and processing of such data. These resources are fragmented in several ways, and it is difficult:

– to find out whether certain types of resources exist and if they exists, where they are,
– to get a permission for using materials and tools found, and
– to use the materials efficiently because the tools and the materials are not compatible with each other.

The goal of CLARIN is to overcome all these obstacles by creating (1) systems with metadata to ease the locating of the resources, (2) a harmonized system for licensing, online authorization and authentication for enabling easy application and granting of permissions, and (3) standards for text and speech corpora, lexicons and tools in order to guarantee interoperability. Nationally funded projects will convert existing contents and gradually provide new materials and tools during the building phase of CLARIN. The total cost of building CLARIN is estimated to be more than 100 million euros. Ultimately CLARIN might contain all relevant European written and spoken materials, ancient and current, all smoothly accessible by the research community. There are some 100 relevant European languages which ought to be handled on an equal basis. Existing language specific programs, if coded separately for each language are not practical for an environment where so many languages are used in parallel.

Morphological analysis of a wide range of languages can be implemented using finite-state technologies without problems and using pure finite-state transducers which do not resort to ad hoc additions such as registers or special procedures or conditions embedded in states or transitions. Purity makes it possible to combine operations algebraically. It also makes the common handling of the relevant languages attractive for the software engineer. Programs may process lots of different languages using nontrivial full scale inflectional dictionaries in order to perform efficient and accurate searches, frequency calculations, lemmatization of online concordances etc. The software only needs to select an appropriate transducer in order to process each language correctly. The programmer would need no language specific code, just use different transducers according to the target language. The programmer can, thus, ignore special characteristics of individual languages. Specifically, there need not be a separate program or subroutine for each of the ca. 100 languages. Furthermore, one operation, i.e. transduction can be used for a wide range of different tasks e.g. preprocessing of the input, stemming, lemmatization, disambiguation, generation of inflected forms, etc.

Pure finite-state transducers appear to be practical at least for word-level processing and similar phrase-level processes such as detection of named entities (i.e. information extraction). Full scale syntactic description of languages is usually done with more powerful frameworks, even if finite-state devices are used in some parts of them.

Several libraries and compilers for finite-state calculus have been built during the past decades. Some commercial ones by Xerox and AT&T are well tested and efficient, but they are proprietary.
They cannot be used freely and it may be difficult or impossible to modify them or combine them with other relevant software.

On the other hand, there are several open source implementations of the finite-state calculus, e.g. SFST (by Helmut Schmid), Vaucanson (Jacques Sakarovitch) and OpenFST (by Michael Riley, Mehryar Mohri et al.). For an implementor of finite-state language tools, these packages pose some problems. In some packages, the functions of the basic calculus are more or less integrated with some interface for compiling rules or interpreting regular expressions and most packages have only limited documentation which makes it difficult for a programmer to extend or utilize the existing code. For a developer further compilers or other tools, choosing the package has to be made in advance. Once decided, it will be difficult to change into another base system.

We have started an initiative called HFST aiming at a more unified and productive implementations of finite-state tools for language processing. We have three full time programmers during 2008 for HFST and the work is supervised by Krister Lindén. The two authors of this paper participate the work as advisors.

The goal of HFST is to ease the above problems by defining and documenting a generic interface for the finite-state calculus so that the building or finite-state based tools would be easier and safer. The interface is first implemented for a few most promising basic finite-state calculus packages, initially for SFST and OpenFST. Thus, the author of a new finite-state based tool needs not commit himself to one underlying implementation of the calculus. HFST has put some effort to set up the underlying concepts to be used in defining the individual functions. The functions, data types and structures of the interface are described using a well known DOXYGEN source code documentation system which produces an online HTML documentation which is automatically produced out of the comments in the source code of the interface.

When the interface is available for downloading later this year, the HFST project also implements and distributes a lexicon compiler and a two-level rule compiler which are similar to the LEXC and TWOLC compilers of Xerox Corporation. All HFST software will be available under GNU GPL or LGPL software and other contributors are invited to build further tools on top of the HFST interface, or integrate further finite-state calculi under the HFST interface.

By making it possible to compare different calculus software packages against each other, we also hope that this will stimulate theoretical studies and software engineering efforts to improve the implementations of the basic finite-state calculus.

As a summary, thanks to national Finnish funding related to CLARIN, we are hope to get the HFST effort up to a level where it is already usable and would attract other researchers to continue the work on different aspects including the creation of new language tools, improving the existing HFST tools and improving the underlying finite-state calculus.