



HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

HFST: Modular Compatibility for Open Source Finite-state Tools

Kimmo Koskenniemi

9 June 2008

University of Helsinki

Department of General Linguistics



HFST team

- Krister Lindén, PhD, leader of the team
- Anssi Yli-Jyrä, PhD, researcher providing theory and compilation formulas
- Antoine Trux, MSc, PhD student
- Erik Axelsson, Miikka Silfverberg, Tommi Pirinen, MA/MSc students, programming the HFST interface and tools based on it
- Kimmo Koskenniemi, professor, funding and consulting



Existing finite-state software

- [Lots of finite-state packages](#) have been created by different researchers and teams (maybe 50-100), (including NooJ).
- Some are free [open source](#), others commercial or with other restrictions (e.g. shareware).
- Some packages only provide the [basic finite-state calculus](#) as a library or as separate programs.
- Some integrate a calculus with an interpreter (or compiler) tailored for a specific rule or lexicon formalism.
- Some are actively [maintained and developed](#), many have neither maintenance nor other activities.



HFST: Helsinki Finite State Toolkit

- Does not implement yet another finite-state calculus, but rather utilizes existing free open source implementations:
 - SFST (Helmut Schmid) transducers without weights,
 - OpenFST (M. Riley, J. Schalkwyk, W. Skut, C. Allauzen and M. Mohri) with weighted transducers, and others such as
 - Vaucanson (Jacques Sakarovitch).
- Intends to provide practical and general purpose free open source finite-state tools.
- Provides HFST, a carefully defined and well documented common interface to several FSM engines.
- Implements useful tools on top of this HFST interface.
- Full scale language modules for testing.



Goals of the HFST

- Create [convergence and cooperation](#) within the community which develops finite-state calculus and tools.
- Create a [neutral platform](#) where different implementations of the FS calculus can [coexist and compete with each other](#).
- Create a [critical mass of research](#) for improving the basic algorithms of the calculus, and compilation algorithms.
- Stimulate the production of free open source [software for compiling](#) dictionaries, grammars and rules into FSTs.
- Stimulate the production of language [resources](#) (e.g. dictionaries, grammars, rules) [to be compiled into FSTs](#).
- Inspired by [CLARIN infrastructure](#) and supports it.



First tools to be implemented with HFST

- Lexicon compilers: HLEXC like Xerox LEXC useful for complex morphological phenomena such as agglutination, derivation and compounding. Derivation and compounding can be cyclic.
- Morphophonemic rule compilers:
 - HTWOLC: Two-level compiler (like Xerox TWOLC)
 - Cascaded replace operations (like SFST and Xerox XFST)
- Challenges: Finnish has lots of word-forms ($> 10^{24}$) and a lot of morphophonological alternations (consonant gradation, vowel harmony, stem final alternations). Northern Sámi is even more complex.



Lexicon compiler

- Words are composed out of morphemes. Morphemes correspond to entries in the lexicon.
- Morphemes are grouped into classes with similar distribution: morphemes belong to the same class, if they can be interchanged (regarding what precedes). Classes are represented as sublexicons.
- Morphemes may still differ regarding what classes of morphemes may follow, i.e. there is a continuation class (which points to a sublexicon from which the next morpheme can be chosen).
- The lexicon compiler reads in a lexicon and produces a FST out of it.



An example of a lexicon

LEXICON Root

talo N;

kaTu N;

LEXICON N;

+N+Sg+Nom:0 #;

+N+Sg+Gen:n #:

+N+Sg+Ine:ssA #

+N+Pl+Gen:jen #;

Root;

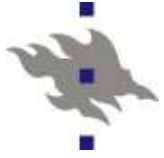
n Root;

END



Two-level rule compiler

- Morphemes are considered to have two representations
 - lemma and features to be shown as the result of the analysis
 - a morphophonemic representation of the morpheme
- The task for the rule component is to relate the morphophonemic representation to the surface form, e.g.
 - p e t t ä ä +V +Pss +Pcp2 +Sup +Pl +Ine
p e t T Ä t T U I m P I s s A
p e t 0 e t 0 y i m m i s s ä
- Rules are parallel with no rule ordering, e.g.
 - T:0 <=> t _ Vowel (I:i) Cons (Cons | :#)
 - I:j <=> :Vowel _ :Vowel
 - A:a <=> (:a | :o | u:) :* _ (elsewhere ä)



The SFST structure

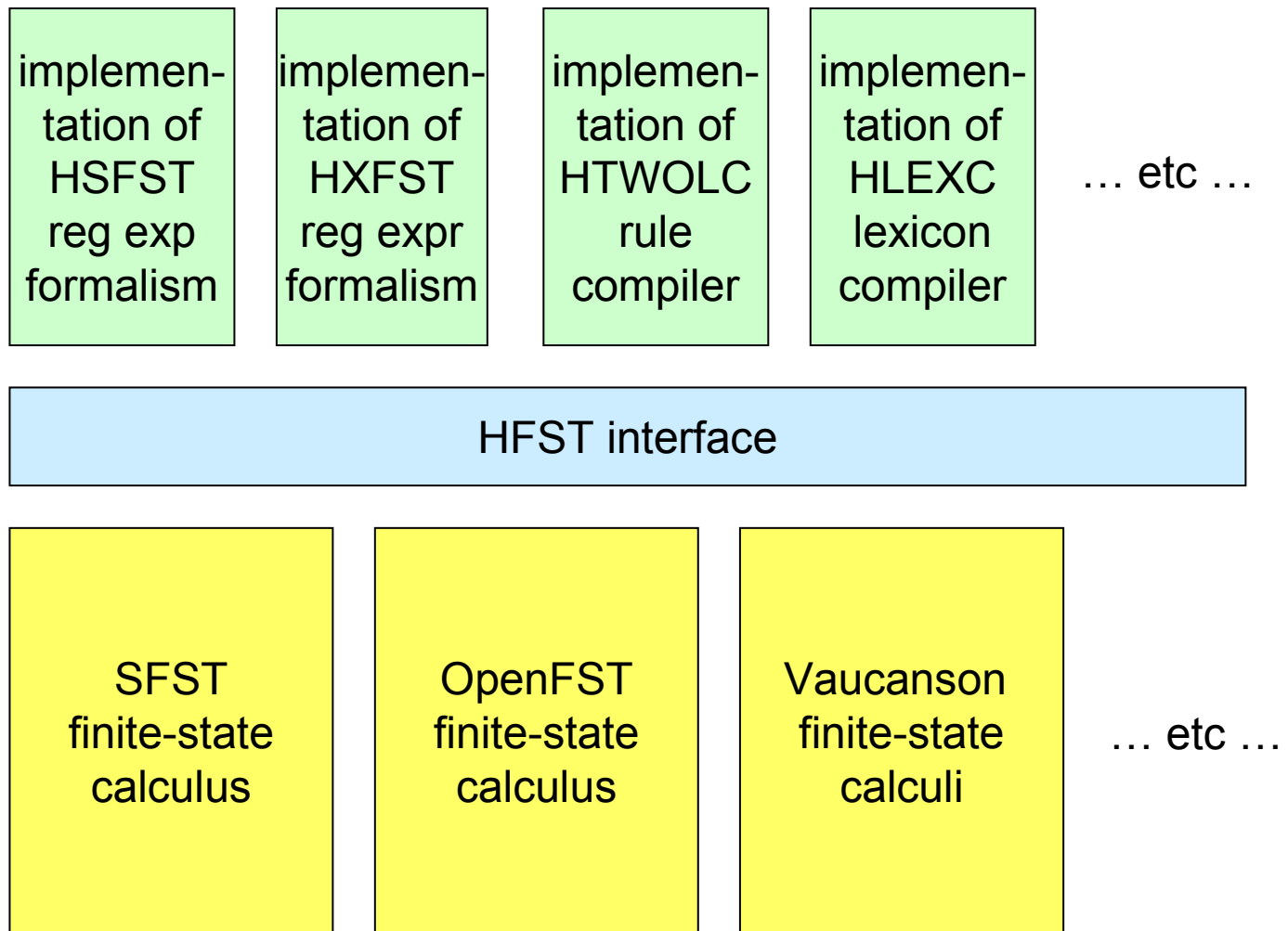
regular expression
formalism

translation into
function calls

functions for
the finite-state
calculus



Design of the HFST





HFST interface

- Definitions of underlying concepts and functions.
- Consistent [naming of functions](#) and use of parameters.
- C++ code as needed to achieve similar behaviour of different calculus packages.
- [DOXYGEN documentation](#)
www.ling.helsinki.fi/~eaxelson/hfst/namespaceHFST.html
- TWiki documentation
<https://kitwiki.csc.fi/twiki/bin/view/KitWiki/HfstHome>
- [File formats](#) (from AT&T, OpenFST, SFST) for exchanging binary and text mode FSTs.



Rule compilation

- Kaplan and Kay found a way to [compile rewrite rules](#) into FSTs. Koskenniemi modified that for two-level rules.
- Anssi Yli-Jyrä discovered a new simpler and more general way to compile rules, so called [general restriction](#).
- With an auxiliary symbol § and alphabet S , one can express various rules as implications, e.g.

$$S^* \S t:d \S S^* \Rightarrow S^* a \S S^* \S a S^*$$

says that $t:d$ may only occur between two a 's. Alternative contexts (the right part) can be expressed as disjunctions of the right-hand side (whereas Kaplan and Kay had a much more complex compilation scheme for that).



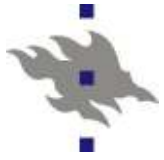
HFST and NooJ

- FSTs produced with HFST tools could be [reused in NooJ](#), e.g. morphological analysis of languages with complex alternations and inflection. (A Finnish verb has some 12,000 distinct forms and several distinct stems.)
- Heuristic methods for [collecting named entities and new lexemes](#) could be based on HFST tools, and the results used in NooJ.
- NooJ dictionaries could be reused with the HFST tools.
- Some NooJ/INTEX functionalities could be implemented using the HFST API.



OMor: open source morphologies

- For research, we need free open source resources which we can improve and develop as needed. In parallel with HFST, we have a OMor project for creating free morphological analyzers.
- OMorFi is creates a Finnish transducer lexicon based on a [list of words of a dictionary](#) with inflectional codes.
- A project in Norway has created two-level morphological analyzers for [Northern and Lule Sámi](#) using Xerox tools. These will be reimplemented with HFST tools as open source.
- HFST [demos](#) already exist for [Finnish](#) (OMorFi), [Swedish](#), [English](#) and [French](#).



HFST commercial and open source applications

- HFST is written under GNU LGPL (compatible with GPL), SFST is GPL, OpenFST is under Apache license. These are all free open source licenses.
- Any further tools developed from HFST or combined with it will remain under (L)GPL.
- Any FSTs produced with HFST tools will remain under the same conditions as the input lexicon and rules, i.e. either proprietary commercial or free open source.
- Thus, HFST tools can be used both for open source and proprietary projects.



A note on finite-state patents

- Xerox and AT&T have claimed several [patents on finite-state techniques](#) and their applications (maybe 100).
- If they are "[software patents](#)", they are not in effect in Europe. (Some may be and others perhaps not.)
- Many patents are invalid because there is previous commercial use of the method or the method is obvious and the only solution. Invalidating patents is expensive. (Patents are a bit like a minefields.)
- Patents [do not prevent private or "fair use"](#).
- Most use of HFST (and other finite-state tools) is, thus, [safe](#). The use of FSTs produced this way should also be safe.