# Using HFST–Helsinki Finite-State Technology for Recognizing Semantic Frames

Krister Lindén. Sam Hardwick, Miikka Silfverberg, Erik Axelson

University of Helsinki
{krister.linden, sam.hardwick, miikka.silfverberg,
erik.axelson}@helsinki.fi

**Abstract.** To recognize semantic frames in languages with a rich morphology, we need computational morphology. In this paper, we look at one particular framework, HFST–Helsinki Finite-State Technology, and how to use it for recognizing semantic frames in context. HFST enables tokenization, morphological analysis, tagging and frame annotation in one single framework.

## Introduction

Language technology enables text mining, e.g. by recognizing semantic frames. In this paper we will look at one particular framework, HFST–Helsinki Finite-State Technology, and its use in processing text from tokenization to recognizing semantic frames in context.

HFST–Helsinki Finite Technology is a framework for building morphologies including morphological lexicons [13], [14], [15]. We present how HFST *identifies semantic frames in context*. To do so, we first present how HFST supports building tokenizers and taggers, which is the minimum requirement for recognizing semantic frames in languages with a rich morphology. In Section 1, we get an overview of the HFST p-match syntax and some examples of how to develop a tokenizer based on a lexicon containing multi-word expressions. In Section 2, we get an introduction to building morphological taggers with HFST using machine learning. In Section 3, we get an introduction to semantic frame recognition with HFST. In Section 4, we get a brief evaluation of developing a rule set for semantic frame annotation. In Section 5, we discuss our results compared with other approaches to semantic frame annotation. In Section 6, we conclude the presentation.

## 1 Tokenization using `hfst-pmatch`

Tokenization is a necessary first step in most text-based natural language processing tasks. For some languages, e.g. English, it is often considered to be a mechanical preprocessing task without linguistic importance, and for others, e.g. Chinese, it is an intricate task called segmentation. However, even in languages that generally insert spaces between words, there are issues that influence the quality or feasibility of tools down the pipeline. We may, for example, want to be able to identify multi-word units, identify

compound words and mark their internal boundaries, control various dimensions of normalization, or produce possible part-of-speech tags or deeper morphological analyses. We describe a general approach to these issues based on morphological transducers, regular expressions and the pattern matching operation `pmatch` [10].

### 1.1 A Short Introduction to `pmatch`

`pmatch` [13] is a pattern-matching operation for text based on regular expressions. In HFST, it has been further developed from the ideas in Xerox `fst`. The regular expressions, i.e. *rules*, are named, and are invoked ultimately by a root expression , i.e. the *top level*, which by convention has the name `TOP`. Expressions may refer to themselves or each other circularly by special arcs which are interpreted at runtime, allowing context-free grammars to be expressed.

Matching operates in a loop, accepting the largest possible amount of input from the current position, possibly modifying it according to the rules and tagging left and right boundaries of sub-rules, and continuing on the the next position in the input. When the rules successfully accept (and possibly transform) some length of input, that is a *match*. When the match has triggered the operation of a tagging directive, e.g. `EndTag(TagName)` or `[].t(TagName)`, the enclosed length of the input is tagged with *TagName*. For example, here is a very naïve tokenizer for English

```
define TOP [[ ("'") Alpha+ ] | Sigma({,.;!?})]] EndTag(w);
```

where `Sigma()` is a function that extracts the alphabet of its argument, which in this case is some punctuation marks given as a string denoted by curly braces. When operated on the sentence *"If I am out of my mind, it's all right with me, thought Moses Herzog."*, it produces output that looks like this

```
<w>If</w> <w>I</w> <w>am</w> <w>out</w> <w>of</w> <w>my</w>
<w>mind</w> <w>,</w> <w>it</w> <w>'s</w> <w>all</w> <w>right</w>
<w>with</w> <w>me</w><w>,</w> <w>thought</w> <w>Moses</w>
<w>Herzog</w> <w>.</w>
```

in normal *matching mode*. The runtime operation of matching can be controlled to only output the matched parts, or give positions and lengths of tagged parts in *locate mode* as well as operating as a more conventional tokenizer outputting one token per line in *extract-matches mode*.

### 1.2 Tokenizing with a Dictionary

A tokenizer consists of the input side of a morphological dictionary. Good coverage in vocabulary and derivation can satisfactorily solve many tokenization headaches on its own. For example, consider the plural possessive of the compound in

(1) The Attorney-Generals' biographies are over there.

To get the tokenization of the example exactly right, a tokenization rule needs to understand that the hyphen is joining parts of a compound word, unlike in e.g. *Borg-McEnroe*, and that the apostrophe is indicating the possessive form, not the end of a quotation. A dictionary can also be augmented to recover from formatting or digitization issues. For example, a text may split words at line boundaries with hyphens, as in

(2) He seemed suddenly to have been endowed with super-
human strength

In this example, the correct tokenization is *superhuman* rather than *super* and *human*, but a dictionary would miss this possibility. However, we can use a finite-state operation to allow the string `-\n` (hyphen followed by a newline) to appear anywhere inside the words in the dictionary. In regular expressions this operation is sometimes called *ignoring*.

### 1.3 Preserving the Parts of a Multi-word Unit

Dictionaries are often equipped with a collection of short idioms, e.g. *in view of*, and other tokens which include whitespace, e.g. *New York*. While these are useful, it may be too early at this stage to fix the tokenization as the longest possible match. A discriminative tagger may not be able to make the correct choice in

(3) The ball was in view of the referee.

if it only sees a tokenization where *in view of* is a single token.

We can extend the dictionary in a simple way to also contain the other possible tokenizations and, in the case of a morphological dictionary, the analyses, as follows

```
define combined_tokens [dict].u .o. [dict | [" " dict]*]
```

where `dict` is our dictionary and `[dict].u` is its input projection. We compose it with arbitrarily many copies of itself, interspersed with space characters. The result contains every multi-word expression both as itself, and as a combination of other words found in the dictionary.

In addition to bare tokens, many downstream tools use analysis cohorts, i.e. the full set of possible base forms and morphological tags for the token in question. The `hfst-pmatch` utility exposes an API that allows retrieval of the position, length, input, output, tag and weight of each of the longest matches, so cohort formatters can be written. For example, suppose our dictionary includes the following entries

| | | | |
|---|---|---|---|
| **in** | in AVP | in NN0 | in PRP |
| **view** | view NN1 | view VVB | view VVI |
| **of** | of PRF | of PRP | |
| **in view of** | in view of PRP | | |

when tokenizing *in view of*. The combined dictionary will then produce the full set of combinations which may be formatted as follows

```
"<in view of>"
   "in view of" PRP
   "in" AVP "view" VVI "of" PRF
   "in" NN0 "view" VVB "of" PRF
   "in" NN0 "view" VVB "of" PRP
   "in" NN0 "view" VVI "of" PRF
   etc.
```

Since in `pmatch` multiple rules operate on the same input, it is possible to integrate higher-level tokenization, such as chunking, named-entity recognition, grouping tokens into sentences and sentences into paragraphs in the same ruleset.

## 2 Morphological Tagging using `hfst-finnpos`

In this section, we describe the morphological tagger `hfst-finnpos`.

FinnPos [20] is a data driven *morphological tagging* toolkit distributed with the HFST interface. The term morphological tagging [6] refers to assigning one full morphological label, including for example part-of-speech, tense, case and number, to each word in a text. It can be contrasted with POS tagging where the task is to infer the correct part-of-speech for each word.

The FinnPos toolkit is based on the Conditional Random Field (CRF) framework [12] for data driven learning. Most work on CRF taggers and other discriminative taggers has concentrated on POS tagging for English, which has a very limited selection of productive morphological phenomena. In contrast, FinnPos is especially geared toward morphologically rich languages with large label sets, that cause data sparsity and slow down estimation when using standard solutions. FinnPos gives state-of-the-art results for the morphologically rich language Finnish [20] both with regard to runtime and accuracy. In addition to morphological tagging, FinnPos also performs data driven lemmatization. Moreover, it can be combined with a morphological analyzer to make a data-driven morphological disambiguator. The capability of FinnPos to take advantage of the linguistic choices made by developers of morphological lexicons is the reason for including FinnPos in the HFST tool set.

In this section, we will focus on describing FinnPos from a practical point of view. A more detailed description of the theoretical foundations as well as evaluation can be found in [20].

### 2.1 FinnPos for Morphologically Rich Languages

In part-of-speech (POS) tagging, the label sets are usually fairly small. For example, the Penn Treebank uses only 45 distinct label types. When tagging morphologically

complex languages, where full morphological labels are required, vastly larger label sets are used. Label sets of around 1,000 distinct label types frequently occur.

Large label sets create a data sparsity problem. For example, for a second order language model and a label set of 1,000 distinct label types, an overwhelming majority of the one billion possible ($1{,}000^3$) label trigrams are never seen in a training corpus of realistic scope. Even label unigrams may be rare as many label unigrams tyically occur only a couple of times in a training corpus.

Although morphological label sets can be very large, individual labels are usually created by combining smaller sub-units from a relatively small inventory. A typical example of such a structured morphological label is the label `Noun|Sg|Nom`, which consists of three sub units: the main word class `Noun`, the singular number `Sg` and the nominative case `Nom`. FinnPos utilizes the internal structure of complex labels by extracting features for sub-units as well as for the entire labels [19]. This alleviates the data sparsity problem because features relating to sub-units of entire tags are used as fall-back. Additionally, sub-unit features allow FinnPos to model grammatical generalizations such as case congruence in isolation of the full labels.

In addition to data sparsity, large label sets cause long training times because the complexity of standard CRF training of an $n$th order model depends on the $(n+1)$st power of the label set size. To speed up training, FinnPos uses an adaptive beam search and a label guesser [20] during inference and estimation. These substantially reduce run-time.

### 2.2 Training and Using a Model

FinnPos uses an averaged perceptron algorithm with early stopping for estimation of model parameters. The error-driven perceptron training algorithm iterates through the training corpus one sentence at a time, labels the sentences and adjusts model weights when erroneous labels are detected. Usually the Viterbi algorithm [2] is used for labeling. This, however, is too slow in practice when dealing with large label sets.

Instead of the Viterbi algorithm, FinnPos uses beam search with an adaptive beam width [18]. Additionally FinnPos uses a generative label guesser modeled after the OOV word model used in [1] to restrict label candidates during training. Because of inexact inference during the training phase, FinnPos additionally uses violation fixing [8].

### 2.3 FinnPos and Morphological Analyzers

FinnPos benefits from a morphological analyzer for morphological disambiguation. The analyzer can be used in two ways: to provide label candidates for words and as a generator of features. For words not recognized by the analyzer, FinnPos will use a data-driven suffix-based guesser to generate label candidates. In addition to the morpological label, FinnPos also uses the morphological analyzer for determining the lemma of a given word. For words not recognized by the analyzer, a data-driven lemmatizer is used instead. The data-driven components are learned from the training corpora, which means that the FinnPos tagger could be used without a morphological analyzer, but a lexicon with reasonable coverage improves the tagging performance.

# 3 Semantic Tagging using `hfst-pmatch`

In this section, we outline a scheme for extracting semantic frames from text using hand-written rules. The rules and approach has been demonstrated in [7]. The current paper is more extensive and includes an evaluation of the rule set. While it does not currently represent a system for extracting a large number of different frames, the `hfst-pmatch` tool has been extensively tested in a full-fledged named-entity recognizer for Swedish [11]. Our motivation here is to present additional capabilities of `hfst-pmatch` as a natural language processing system for extracting factoids from textual data to be used in text and data mining.

## 3.1 Introduction

A semantic frame [5] is a description of a *type* of event, relation or entity and related participants. For example, in FrameNet, a database of semantic frames, the description of an `Entity` in terms of physical space occupied by it is an instance of the semantic frame `Size`. The frame is evoked by a lexical unit (LU), also known as a frame evoking element (FEE), which is a word, in this case an adjective, such as *big* or *tiny*, descriptive of the size of the `Entity`. Apart from an `Entity`, which is a core or compulsory element, the frame may identify a `Degree` to which the `Entity` deviates from the norm, e.g., *a **really** big dog*, and a `Standard` with which it is compared, e.g., *tall **for a jockey***.

| | |
|---|---|
| Lexical Unit (LU) | Adjective describing magnitude (large, tiny, ...) |
| Entity (E) | That which is being described (house, debt, ...) |
| Degree (D), optional | Intensity or extent of description (really, quite, ...) |
| Standard (S), optional | A point of comparison (for a jockey, ...) |

Table 1: The semantic frame *Size*.

For example:

$$\left[_{\text{Size}}\left[_{\text{E}}\text{He}\right]\text{is}\left[_{\text{D}}\text{quite}\right]\left[_{\text{LU}}\text{tall}\right]\left[_{\text{S}}\text{for a jockey}\right]\right]$$

Table 2: A tagged example of *Size*

## 3.2 A Rule

A simple and common syntactic realization of the `Size` frame is a single noun phrase containing one of the LUs, such as *the big brown dog that ran away*. Here we would like to identify *big* as `LU`, *brown dog* as `Entity` and the combination as `Size`. Our first rule for identifying this type of construction might be

```
define LU {small} | {large} | {big} EndTag(LU);
define Size1 LU (Adjective) [Noun EndTag(Entity)].t(Entity);
define TOP Size1 EndTag(Size);
```

Table 3: A simplified first rule

This rule set has been simplified for brevity – it only has a few of the permitted LUs, and word boundary issues have not been addressed. The `[].t()` syntax in the definition of `Size1` is a tag delimiter controlling the area tagged as `Entity`. The extra `Adjective` is optional, which is conveyed by the surrounding parentheses.

We can verify that our rules extract instances of our intended pattern by compiling them with `hfst-pmatch2fst` and running the compiled result with `hfst-pmatch --extract-tags`. In the following we have input the text of the King James Bible from Project Gutenberg[1] and allowed some extra characters on both sides for a concordance-like effect

```
...
there lay a <Size><LU>small</LU> round <Entity>thing</Entity></Size>
...
there was a <Size><LU>great</LU> <Entity>cry</Entity></Size> in Egypt
...
saw that <Size><LU>great</LU> <Entity>work</Entity></Size> which
...
```

A natural next step is to add optional non-core elements, such as an adverb preceding the LU being tagged as `Degree` and a noun phrase beginning with *for a* following it as `Standard`.

```
define Size1 [Adverb].t(Degree) LU (Adjective) [Noun].t(Entity)
             [{for a} NP].t(Standard);
```

Table 4: Extending the rule with optional elements

and here are some examples this rule finds in the British National Corpus [2]

[1] `http://gutenberg.org`
[2] http://www.natcorp.ox.ac.uk/

```
...
presence of an <Size><Degree>arbitrarily</Degree>
  <LU>small</LU> <Entity>amount</Entity></Size> of dust
...
one <Size><LU>small</LU> <Entity>step</Entity>
  <Standard>for a man</Standard> </Size>
...
```

We can see that in *small amount of dust*, we might want to tag not just the immediate noun as `Entity` but the entire noun phrase which could be implemented up to a context-free definition of a noun phrase, and in *one small step for a man* a common indirect use of the `Standard` construction. As well as correct matches, such as *small round thing* in the biblical example, we have metaphorical meanings of `Size`, such as *great cry*. This may or may not be desired – perhaps we wish to do further processing to identify the target domains of such metaphors, or perhaps we wish to be able to annotate physical size and physical size only.

### 3.3 Incorporating Semantic Information

Size is a very metaphorical concept, and syntactic rules as above will produce a large amount of matches that relate to such uses, e.g., *a great cry* or *a big deal*. If we wish to refine our rules to detect such uses, there are a few avenues to explore. First of all, some LUs are much more metaphorical than others. A *great man* is almost certainly a metaphorical use, whereas a *tall man* is almost certainly concrete. Accuracy may be improved by requiring *great* to be used together with common nouns meaning several individuals like a *great crowd*. In addition, there are semantic classifications of words, such as WordNet [17]. We may compile the set of hyponyms of *physical entity* and require them to appear as the nouns in our rules as shown in Table 5.

```
define phys_entity  @txt"phys_entity.txt";
```

Table 5: Reading an external linguistic resource

### 3.4 Incorporating Part-of-speech Information

We have so far used named rules for matching word classes like `Noun`, without specifying how they are identified. Also our collection of LUs might need some closer attention – for example *little* could be an adverb. Considering that in writing our rules, we are effectively doing shallow syntactic parsing, even a very simple way to identify parts of speech may suffice, e.g. a morphological dictionary. For example, a finite-state transducer representing English morphology may be used to define the class of common

nouns as in Table 6. If we have the use of a part-of-speech tagger, we may write our rules to act on its output, as in Table 7 where W refers to some word delimiter.

```
! The lexicon we want to read
define English @bin"english.hfst";
! We compose it with a noun filter and extract the input side
define Noun  [ English .o. [?+ "<NN1>" | "<NN2>"] ].u;
! (NN1 is singular, NN2 plural)
```

Table 6: Using a dictionary to extract words of a given word-class

```
define Noun LC(W) Wordchar+ ["<NN1>"|"<NN2>"] RC(W);
```

Table 7: Using tags in pre-tagged text

### 3.5 Increasing Coverage

Having considered for each rule where Degree and Standard may occur, coverage may be evaluated by also finding those cases where a LU is used as an adjective but does not match the current rules, e.g.

```
define TOP Size1 | Size2 | [LU].t(NonmatchingLU);
```

The valid match is always the longest possible one, so NonmatchingLU will be the tag only if no subsuming SizeN rule applies. For example in

```
the moving human body is <NonmatchingLU>large</NonmatchingLU>,
obtrusive and highly visible
```

we see another realization of the Size frame: the Entity is followed by a copula, and the LU appears to the right. We can write a new rule Size2 to capture this, adding positions for non-core elements either by linguistic reasoning or by searching the corpus.

## 4 Evaluation

FrameNet has published a tagged extract of the American National Corpus [3] [4], consisting of 24 texts. Of these, one uses the `Size` frame 35 times, but the remainder use it only an additional 6 times for a total of 41 times. This is too thin a selection, and suggestive of some inconsistency in the use of this frame vs. some alternative ones such as `Dimension`, and various metaphorical sub-cases of that frame. Evaluating the extraction of the `Size` frame on the basis of this minute corpus was unfeasible, but we used it as a reference when developing our own training and test set.

To develop our rule set, we took 200 sentences of the British National Corpus containing, as a token, one of the `LU`s, and tagged them by hand. We considered a LU to be any inflected form of a word of the synonyms to *size* given by WordNet including metaphorical meanings of size. The sentences had POS tags from the original material, but punctuation and information about multi-word units was removed before developing the rule set. This corresponds to running surface text through a POS tagger which does not recognize multi-word expressions before running the frame extractor.

We had one person spend a working day developing rules based on our set of training samples, iterating a process of spotting the difference between the hand-tagged samples and the tagging produced by our rules, and modifying the rule set. This resulted in two top-level rules, one corresponding to cases where the `LU` precedes the `Entity`, and one to cases where it follows as these were the only compulsory elements in the frame. Overall, the rule set was 46 lines long, excluding comments and whitespace.

To get an idea of the quality of the rules, we also hand-tagged another 100 sentences from the same corpus. These do not necessarily contain the `Size` frame to test that the rules do not over-generate. Of these sentences, 81 were tagged completely correctly by the rule set. Results by `LU` are in Table 8.

| | |
|---|---|
| Number of sentences | 100 |
| Number of LUs | 113 |
| Number of LUs corresponding to a `Size` frame | 56 |
| Number thereof matched by the rules | 50 |
| Total number of matches made by the rules | 54 |
| Coverage | 89% |
| Accuracy | 93% |

Table 8: LU-level semantic tagging performance on the 100 sentence test set

In Table 8, a match in the test material is considered correct if the relevant `LU` is correctly identified. We explore some further details regarding the quality of both correct and incorrect test matches in Table 9.

We note that the test tagging was not independent of us but no other tagging existed and that the overall amount of both training and test material is rather small. We do not

[3] `http://www.anc.org`

[4] The FrameNet-annotated texts are at
`https://framenet.icsi.berkeley.edu/fndrupal/index.php?q=fulltextIndex`

| | |
|---|---|
| Matches where wrong `Entity` was tagged | 4 (8%) |
| Matches where `Entity` was partially wrongly tagged | 8 (16%) |
| Matches where `Degree` was incorrectly tagged | 2 (33% of hand-tagged `Degrees`) |
| Incorrect tagging due to insufficient rule sophistication | 9 (53% of mistakes) |
| Incorrect tagging due to mistakes in POS tagging | 5 (29% of mistakes) |
| Incorrect tagging due to lacking multi-word unit information | 2 (12%) |
| Incorrect tagging due to lacking punctuation information | 1 (6%) |

Table 9: Quality of matches made by the rules in the test samples

think this is a conclusive result, but it is an indication of the semantic tagger that could be developed in a relatively small amount of time with this approach.

## 5 Discussion

In this section, we contrast HFST with some other semantic frameworks for recognizing semantic frames, i.e. Shalmaneser [4], LTH [9] and SEMAFOR [3].

Shalmanser treats semantic frame extraction as a pipeline of syntactic parsing, frame identification, semantic argument identification and semantic role labeling. Syntactic parsing uses an external toolkit. Note that frame identification precedes role labeling, i.e. they are not done in parallel. However, Erk and Pado [4] claim that this would give very small gains in accuracy while incurring huge CPU cost. Shalmaneser can be trained for any semantic annotation scheme provided appropriate training data exists. Users can replace some components of the system with customized components. Full scale models for English and German are available. Evaluation was done on manually annotated data. FrameNet 1.2 for English and the SALSA corpus for German. Evaluation is with regard to to the F1-score on unlabeled argument chunks and labeling accuracy for argument labels. The F1-score for argument chunks was 0.751 for English and 0.6 for German. Argument label accuracy was 0.784 for English and 0.673 for German.

LTH also treats semantic frame extraction as a pipeline of syntactic parsing, frame identification, semantic argument identification and semantic role labeling. In contrast to many other systems, LTH uses a dependency syntactic parser instead of a constituent parser. Frame identification is accomplished using a classifier based on input words and dependency structure. To aid argument identification, the FrameNet lexical database was extended with WordNet data. A classifier was trained to identify words that were likely to belong to a given semantic frame. Evaluation was with regard to F1-score for frames and frame elements. As training data, FrameNet 1.3 was used and, as test data, three manually annotated segments from the American National Corpus. The data sets come from the SemEval 2007 joint task on frame semantic structure extraction. The F1-score for English on the test data was 0.621.

The basic architecture of SEMAFOR is similar to Shalmaneser and LTH. The frame parsing task is divided into two sub-tasks: predicate identification and argument identification. SEMAFOR features a latent-variable model, semi-supervised extension of the predicate lexicon and joint identification of the entire argument set of a predicate using linear programming. This allows for integration of linguistic constraints on the argu-

ment sets in a principled way. A model for English is available. The evaluation and data was the same as for LTH. The F1-score on English is 0.645.

In contrast, HFST treats semantic frame extraction as a pipeline in only two stages: morphological tagging and semantic labeling, i.e. frame identification, semantic argument identification and semantic role labeling are done in parallel. The fact that HFST recognizes the whole frame in one step, means that HFST has access to the whole frame element configuration when making the decision to commit to the frame and the argument labels. In addition, HFST can take linguistic constraints into consideration both in the morphological and the frame and role labeling tasks. This contributes to the high coverage and accuracy in the evaluation which no doubt is still much too limited. When the whole semantic frame and all its argument roles are considered at the same time, HFST removes part of the need for syntactic processing as an intermediate step, but nothing prevents a user from replacing or enriching the morphological tagging with information from a syntactic parser. Future work is a large-scale evaluation of HFST for semantic frame and role labeling of a semantically rich language like Finnish where we will draw on the availability of FinnWordNet [16] to extend the lexical unit coverage.

## 6 Conclusion

In this paper, we have outlined the steps involved when using HFST–Helsinki Finite-State Technology for recognizing semantic frames in context. A small-scale evaluation indicates that the setup is capable of highly accurate semantic information labeling.

## References

1. Brants, T.: TnT: A statistical part-of-speech tagger. In: Proceedings of the sixth conference on Applied natural language processing. pp. 224–231 (2000)
2. Collins, M.: Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In: Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10. pp. 1–8. EMNLP '02, Association for Computational Linguistics, Stroudsburg, PA, USA (2002)
3. Das, D., Chen, D., Martins, A., Schneider, N., Smith, N.: Frame-semantic parsing. Computation Linguistics (2014)
4. Erk, K., Pado, S.: Shalmaneser – a flexible toolbox for semantic role assignment. In: Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06) (2006)
5. Fillmore, C.J.: Frame semantics and the nature of language. Annals of the New York Academy of Sciences: Conference on the Origin and Development of Language and Speech 280(1), 20–32 (1976)
6. Grzegorz Chrupala, G.D., van Genabith, J.: Learning morphology with Morfette. In: Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08). European Language Resources Association (ELRA), Marrakech, Morocco (May 2008)
7. Harwick, S., Silfverberg, M., Lindén, K.: Extracting semantic frames using hfst-pmatch. In: Proceedings from NODALIDA 2015. pp. 305–308. Vilnius, Lithuainia (May 2015)

8. Huang, L., Fayong, S., Guo, Y.: Structured perceptron with inexact search. In: Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 142–151 (2012)
9. Johansson, R., Nugues, P.: LTH: Semantic structure extraction using non-projective dependency trees. In: Proceedings of SEMEVAL 2007 (2007)
10. Karttunen, L.: Beyond morphology: Pattern matching with FST. In: Mahlow, C., Piotrowski, M. (eds.) Systems and Frameworks for Computational Morphology. Communications in Computer and Information Science, vol. 100, pp. 1–13. Springer, Berlin Heidelberg (2011)
11. Kokkinakis, D., Niemi, J., Hardwick, S., Lindén, K., Borin, L.: Hfst-sweNER - A New NER Resource for Swedish. In: Proceedings of the 9th edition of the Language Resources and Evaluation Conference (LREC'14), Reykjavik 26 - 31 May 2014. pp. 2537–2543 (2014)
12. Lafferty, J.D., McCallum, A., Pereira, F.C.N.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the Eighteenth International Conference on Machine Learning. pp. 282–289. ICML '01, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001)
13. Lindén, K., Axelson, E., Drobac, S., Hardwick, S., Kuokkala, J., Niemi, J., Pirinen, T., Silfverberg, M.: HFST – a system for creating NLP tools. In: Mahlow, C., Piotrowski, M. (eds.) Systems and Frameworks for Computational Morphology. Communications in Computer and Information Science, vol. 380, pp. 53–71. Springer, Berlin Heidelberg (2013)
14. Lindén, K., Axelson, E., Hardwick, S., Pirinen, T.A., Silfverberg, M.: HFST—framework for compiling and applying morphologies. In: Mahlow, C., Piotrowski, M. (eds.) Systems and Frameworks for Computational Morphology. Communications in Computer and Information Science, vol. 100, pp. 67–85. Springer, Berlin Heidelberg (2011)
15. Lindén, K., Silfverberg, M., Pirinen, T.: HFST tools for morphology—an efficient open-source package for construction of morphological analyzers. In: Mahlow, C., Piotrowski, M. (eds.) Systems and Frameworks for Computational Morphology. Lecture Notes in Computer Science, vol. 41, pp. 28–47. Springer (2009)
16. Lindén, K., Carlson, L.: FinnWordNet – WordNet på finska via översättning (in Swedish with an English abstract). LexicoNordica – Nordic Journal of Lexicography 17, 119–140 (2010)
17. Miller, G.A.: Wordnet: A lexical database for English. Communications of the ACM 38(11), 39–41 (1995)
18. Pal, C., Sutton, C., McCallum, A.: Sparse forward-backward using minimum divergence beams for fast training of conditional random fields. In: Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on. vol. 5, pp. V–581–V–584. IEEE (2006)
19. Silfverberg, M., Ruokolainen, T., Lindén, K., Kurimo, M.: Part-of-speech tagging using conditional random fields: Exploiting sub-label dependencies for improved accuracy. pp. 259–264. Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Association for Computational Linguistics (2014)
20. Silfverberg, M., Ruokolainen, T., Lindén, K., Kurimo, M.: FinnPos: An Open-Source Morphological Tagging and Lemmatization Toolkit for Finnish. Language Resources and Evaluation, Springer (2015)