

Emores — Empirical Morphological Reasoning

Toni Arnold
8048 Zürich, Switzerland
toni_arnold@bluewin.ch

Abstract

Emores is an experimental software (GPL-ed) for aiding lexicon extraction from corpora. It is based on the Stuttgart Finite State Transducer Tools [4] and the German morphology SMOR [5]. For new word forms from a corpus text, it generates the lemmas producing them (induction) and enumerates all word forms each of these lemma generates (deduction). All data is written to a relational database and analyzed with SQL to measure the explanatory power of the guessed lemmas with respect to the corpora (abduction).

1. Introduction

For learners of a language, it is quite common to infer inflectional classes or word classes from encountered word forms: I have always known the word “word” as a noun, but I have inferred that it can also be used as a verb, after repeatedly hearing the form “wording”. The absence of the form “wordest” to me is also absence of evidence to memorize it as an adjective.¹

In English, the word class often can be inferred from the word position in a sentence and this in turn determines the inflectional class. In more heavily inflected languages such as German each word class has several inflectional classes which usually are learned by example. Therefore it is even common to annotate the inflectional class in terms of an abbreviated notation of the possible word forms in printed spelling dictionaries: “Wort -[e]s”.

The aim of emores is to formally model and compute this process (by brute force) to facilitate semi-automatic lexicon acquisition from corpora. The development of the software was carried out in the spirit of old-fashioned top-down design.

¹The existing “wordiest” is based on the derivation “wordy”, while “word” as an adjective would be a distinct lemma.

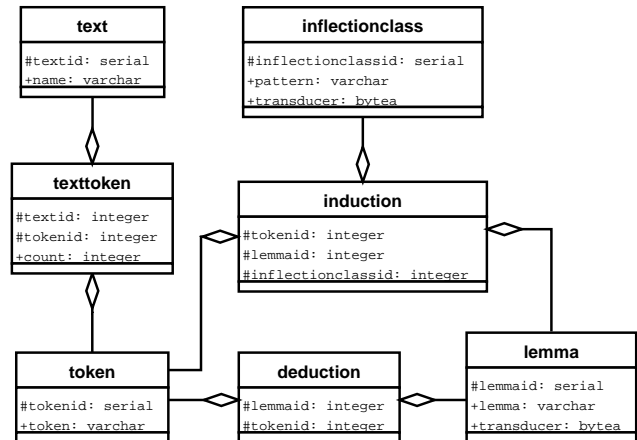


Figure 1. database model

2. Relational model

The main idea behind emores is the evidence that the linguistic data a finite state morphology operates on can be modeled relationally: Corpora can be viewed as sets of inflected word forms (tokens). The morphology uses a lexicon as a set of lemmas and maps word forms to lemmas when analyzing them. Distinct inflected forms of a word are analyzed with the same lemma — a typical one-to-many relationship ($1..n \in \mathbb{N}_1$).

Extending the lexicon means finding the correct lemmas for unknown word forms. As the lemma encodes the inflectional class of a word, it is correct if the word forms it produces can be found in corpora, and it is wrong if it generates word forms that don’t exist in the language. The occurrence of a (hypothetical) word form in corpora texts is also a one-to-many relationship ($1..n \in \mathbb{N}_0$).

Figure 1 shows a simplified model of the emores database. The detailed model was drawn with GNOME Dia in UML notation, which has the advantage that it can be automatically converted to SQL DDL using the tedia2sql script. This upgrades the diagram to the actual source code in the sense of the GPL as “the preferred form of the work for making modifications to it”[1].

The difference between $n \in \mathbb{N}_1$ (the deduction link table in the model) and $n \in \mathbb{N}_0$ (the texttoken link table in the model) is the key for performing abductive inference.

3. Inference model

Emores' abductive inference model is based on an idealization of practice in empirical science: (1) hypothesize rules based on empirical data (induction), (2) apply these rules to the data (deduction) and (3) test which model fits the data best (abduction). But in empirical science, a hypothesis normally is based on well-grounded reasons, thus hypothesis overgeneration is not so much a problem as in the context of emores, where choosing between arbitrary hypotheses is the main challenge.

It is controversial whether the term "abduction" should be used in linguistics at all [2]. Deutscher argues that while Peirce has introduced it, he used it in two incompatible senses in distinct periods of his writings, which now leads to confusion. He proposes to subsume "inference to the best explanation" under "inductive inference", so I need to point out how and why the terms are used in the emores software.²

3.1. Induction

The induction step corresponds to the induction link table in the database model and is used in the narrow sense of concluding rules (lemmas) from real observations as empirically found word forms (tokens). Deutscher calls this "enumerative induction" [2, pg. 476]. In emores, induction is performed without restrictions and therefore each token generates many (linguistically mostly wrong) inflection rules (lemmas), as most inflectional class patterns match a specific token.

The corresponding finite state transducer is stored in the lemma table for later use. It consists of the morphology compiled with a lexicon containing just this one lemma.

3.2. Deduction

The deduction step corresponds to the deduction link table in the model. It enumerates all tokens that a lemma can analyse in a given morphology, i.e. any strings the corresponding transducer generates. As the transducer found in the induction step is stored, carrying out the deduction step is separated from the induction step.

²From an epistemological point of view, the inference model is embedded in a pure deduction: The lexical categories of the linguistic morphology and its implementation using finite state transducers are taken for granted and can't be challenged within the formalism. A philosopher of science would identify the whole terminology as an abuse concealing the absence of falsifiability, and justifiably so.

3.3. Abduction

Abduction here means "inference to the best explanation" [2, pg. 476]. There is no physical data table representing abduction. Instead, after all the data generated by induction and deduction has been written to the database, the explanatory power of the lemmas is measured using SQL in terms of the saturation level of a lemma.

The definition of saturation is based on the set of tokens a lemma has generated in the deduction step: The saturation level is the quotient of the number of tokens actually referenced by texttoken with the whole number of tokens for the lemma. It yields 1 if all tokens a lemma can generate are covered by empirically found tokens and decreases if the generation set isn't supported by the data.

4. Lemma guessing

Guessing a lemma for an unknown token is not as trivial as it first might seem, as a finite state morphology normally just rejects strings it can not analyse. To make it accept not lexicalized token, I have used a concept of inflectional classes: An inflectional class lemma is derived from a lexical lemma by replacing the defining lexeme string with the dot wildcard (mapping any letter in the language alphabet onto itself) and Kleene's plus operator .+

This way, the inflectional class of the fully specified lemma `<Stem>word<N><base><native><NounReg>` becomes `<Stem>.+<N><base><native><NounReg>`. But the analysis strings generated by a transducer containing all inflectional classes do not determine which lemma has actually been used to match the input token. A first precondition for lemma guessing is therefore to compile a separate transducer for each single inflectional class and to apply them separately.

But the situation is even worse: the word stem in the analysis string returned by such a transducer doesn't necessarily equal the lexeme string used in the lemma, as the lemma sits somewhere in the middle of a transducer composition chain between the surface string and the analysis string. Submatch extraction is beyond the generative capacity of finite state transducers, a technique commonly used in recursive regular expression algorithms by using parentheses like (.+) which store the searched submatch string in a variable to be retrieved later on.

Confronted with this problem, I considered manually constructing a separate lemma guesser morphology for a given language morphology to be much too tedious and error prone. The core idea has been to build a specialized computer algebra system (CAS) for SFST-PL finite state transducer sources.

5. The SFST CAS

The key to the formalism is the inversion operator $\hat{_}$, which switches the upper and lower level of a transducer. Figure 2 shows the basic structure of the transformation to extract the lemma from a morphology mapping the surface to the analysis (each element denotes a two-level transducer):

```
analysis || lemma || surface
          ↓
 $\hat{\_}$  analysis || analysis || lemma || surface
```

Figure 2. lemma extraction

But an SFST-PL program consists of a large set of variable definitions (assigning intermediate transducers to variables) and a final line with the variable(s) denoting the result transducer the compiler will output.

Figure 3 shows a simple toy morphology derived from the algebra doctest for the Python interface to the SFST tools (pysfst). It is deliberately somewhat confusing (comments are stripped) to justify the need for an automated system.

```
>>> import sfst
>>> src = '''
... ALPHABET = [a-z]<pl>
... $LEX$ = <stem> .+ <reginf>
... $MAP1$ = <>:<stem> .* <>:<reginf> .*
... $MAP2$ = <stem>:<> .* <reginf>:<>
... $LEX$ = $MAP1$ || $LEX$ || $MAP2$
... $LEMMA$ = {lemma}:{surface}
... $REGMOR$ = $LEX$ || $LEMMA$
... $INFL$ = <pl>:s
... $ANLS$ = .* {analysis}:{lemma} .*
... $MORPH$ = $ANLS$ || $REGMOR$ $INFL$
... $MORPH$
... '''
>>> transducer = sfst.compile(src)
>>> print transducer.analyze('surfaces')
['analysis<pl>']
```

Figure 3. example morphology

The lexicon containing the inflectional class (the lemma with a wildcard) is assigned to the first occurrence of the $\$LEX\$$ variable. The core “morphology” itself consists of two parts: $\$LEMMAS$ maps the surface string to the lemma, and $\$INFL\$$ recognizes the plural form. The two $\$MAPx\$$ transducers delete the symbols not wanted on the surface or the analysis level, respectively. The result transducer maps the string “surfaces” to the string “analysis”, recognizing the trailing “s” as the plural form $\langle pl \rangle$.

Now we have a complex expression with an embedded variable whose value we’d like to know. “Have you seen it before”? [3] Yes! This might somehow be analogous to a

linear equation system containing unknowns. A CAS (computer algebra system) is a symbolic variable resolver for this very problem. So I’ve built one for finite state transducer algebra. Figure 4 shows it in action.

```
>>> solved_src = sfst.solve(src, '$LEX$')
>>> print solved_src
ALPHABET = [a-z]<pl>
$LEX$ = <stem> .+ <reginf>
$MAP1$ = <>:<stem> .* <>:<reginf> .*
$MAP2$ = <stem>:<> .* <reginf>:<>
$LEX_pysfst_2$ = $MAP1$ || $LEX$ || $MAP2$
$LEMMAS$ = {lemma}:{surface}
$REGMOR$ = $LEX_pysfst_2$ || $LEMMAS$
$INFL$ = <pl>:s
$ANLS$ = .* {analysis}:{lemma} .*
$MORPH$ = $ANLS$ || $REGMOR$ $INFL$
(_$LEX$) (_$INFL$) ||  $\hat{\_}$  ( $ANLS$ || $MAP1$ (
  _$INFL$ ) || $ANLS$ || $MAP1$ (_$INFL$) ||
  $LEX$ (_$INFL$) || $MAP2$ (_$INFL$) ||
  $LEMMAS$ $INFL$
)
>>> guesser = sfst.compile(solved_src)
>>> guesser.analyze('surfaces')
['<stem>lemma<reginf><pl>']
```

Figure 4. the example solved

When solving for $\$LEX\$$, the CAS generates a new SFST-PL source bringing the intermediate level down to the analysis. Doesn’t the cryptic last source line nicely look like what one expects from a mathematical CAS solving equations for a variable?

The analysis string produced by the solved transducer now in fact returns the lemma string the $\$LEX\$$ transducer actually matched for the input string “surfaces”, together with the analysis symbol $\langle pl \rangle$. But what happened in detail?

The CAS flattens the variable assignment structure into one single composition chain such that the $\$LEX\$$ variable becomes visible in the result and then composes it with the inversion of the composition chain between the analysis and the lemma. This reverses any manipulations that have been done between the lemma and the analysis.

Instead of applying inversion, one could be content with the flattened composition with a visible lemma and strip the composition left to it. In fact, the inversion part even for the complex SMOR morphology is rather small. The reason for using the full composition with inversion is some sort of mathematical stubbornness: “Did you use all the data? Did you use the whole condition?” [3] Leaving out the part between the lemma and the analysis would yield a “no” to that question, leaving the mule feeling queasy.

5.1. Underlying postulates

The first important postulate for the symbolical conversion is the law of associativity which holds for transducer

compositions. This makes it possible to replace a variable in a composition with the expression the variable stands for (look at variables as implicitly parenthesized expressions).

The second, less obvious postulate, is the law that concatenation is distributive over composition. Therefore in the example, the analysis/deep language of \$INFL\$ (extracted with the domain operator `_` with its scope restricted using parentheses) must be concatenated with each element in the composition chain below it.

Two additional problems had to be solved for the CAS to become usable: First, variables can be redefined later on and therefore become ambiguous in a symbolic transformation of an SFST-PL source, as the defining order gets lost in the variable solving algorithm. Therefore, homonymous variables automatically get renamed by appending a numbered suffix (\$LEX_pysfst_2\$ in the example).

Second, the declarative nature of the SFST programming language hides the fact that transducer minimization only takes place on variable assignments. While that declarativeness is a prerequisite for computing the presented conversion, it is an abstraction that leaks [6]: the compilation of a composition of not minimized transducers of a real size morphology can literally explode in terms of CPU and memory usage. Which intermediate variables were introduced for clarity and conciseness and which were introduced for computability reasons is not apparent in the SFST-PL source. The algebra therefore provides a `solve_precompile` method which compiles the result transducer by incrementally assigning each element of the composition to an intermediate variable, thus ensuring that it always gets minimized. This turned out to be much slower than compiling the original source of the SMOR morphology — a hint that it may have been optimized without explicitly telling this to the reader of the code.

5.2. Limitations of the CAS

The algebraic calculus presented here cannot handle disjunction. I am not aware of a formal definition, but such a calculus should multiply out the discrete resolution paths along both sides of the operator and return multiple algebraic solutions — just like the present implementation uses multiple discrete transducers for any single lemma in the lemma disjunction assigned to \$LEX\$. It is therefore a prerequisite to reformulate the morphology that it should operate on by removing all and any disjunctions in the assignment path of the variable that the morphology is solved for.

In fact, the mechanism currently implemented in `pysfst` rejects all SFST operators except composition and concatenation. If a variable to be expanded takes part of such an operator, the operator needs to be refactored out by introducing an intermediate variable such that the assignment path bypasses it. But that kind of refactoring is impossible for

disjunction, which always propagates to the top. That fact mirrors a conceptual limitation: derivation and composition consists of several lemmas joined together. A morphology will analyse a given word form either as lexicalized or as derived — an essential disjunction. But if two compound lemmas are unknown for a given word form, the lexeme split point could be assigned anywhere in its middle, yielding a multiplicity of two-valued guesses. Therefore I considered limiting emores to lexicalized lemmas by commenting out all disjunction expressions as reasonable: derivation and composition should be tried with known lemmas.

6. Lemma clustering

Using a database for storing all the data produced by the finite state transducers introduces a layer of abstraction which hides these concrete machines from data analysis tasks. This makes it possible to easily use existing tools like GNU R for statistical analysis.

An example for visualizing enumerative induction and deduction is the cluster dendrogram shown in figure 5. It groups the guessed lemmas for the token “word” according to their generative similarity. As one token generates many lemmas in the induction step and each of these lemmas generate many tokens, the generated token sets overlap (the initial token is by definition generated by all lemmas, therefore the need for the dummy token “_” to make all crosstab variables truly binary). It is instructive to see how closely the generated lemmas for a token resemble to each other with respect to the token sets they generate.

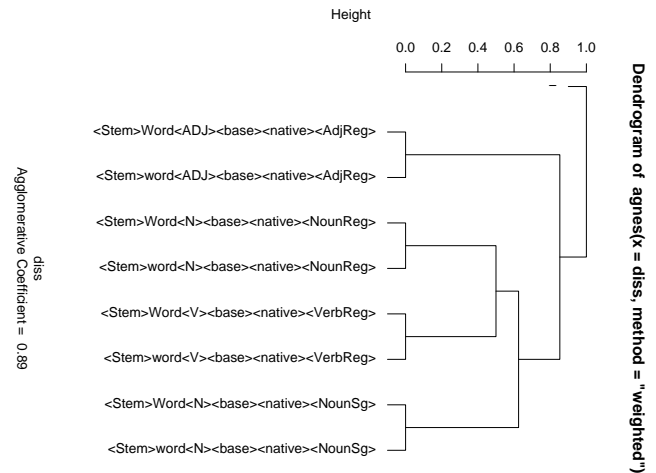


Figure 5. grouped lemmas for “word”

Obviously, the induction/deduction always generates pairs of lemmas with a dissimilarity of 0.0 which differ only by the capitalization of the first letter. This is an artifact of the capitalization mechanism in the morphology.

The higher dissimilarities reflect the English word classes, except the <NounSg> lemma, which is used for irregular inflection when two lemmas are used for the same word to consistently generate the same word stem in the analysis. As the infix inflection used for mapping “mice” to “mouse” doesn’t match the given token, <NounPl> is not generated.

The dendrogram is simple, because the underlying morphology is the simple XMOR example morphology for the weakly inflected English language. For the German SMOR morphology, the enumerative induction usually generates hundreds of lemmas, causing the dendrogram to spawn several pages. The groups of generatively identical lemmas (with a dissimilarity of 0.0) tend to be surprisingly large (sometimes several dozens).

I am too unfamiliar with the linguistic aspects of the published SMOR implementation to reasonably comment on that. Undoubtedly, German nouns of masculine and neuter gender share their inflectional classes and therefore are always indistinguishable by emores’ calculus — but the lemma explicitly encodes that information which therefore must be attained externally. But information like <nativ> vs. <fremd> (foreign words) in German changes the preference for subtle details in the declination, a probabilistic concept that is not implemented in the SFST tools.

However, printed cluster dendrograms generated with the emores software could be a valuable aid for further linguistic work on that topic, as discussions based on easy-to-generate analytical data tend to become more seminal.

7. The implementation infrastructure

As the SFST tools were brought to the public under the GPL, it is crucial for all software used with emores to be GPL compatible Free Software.

The core application has been written in Python. The main reasons for choosing Python as the glue language are the availability of good SQL database interfaces (psycopg2 and SQLAlchemy) and the relative ease to write a Python interface to the SFST tools with the SWIG interface generator (pysfst).

Being an interactive scripting language, Python is well suited for building prototypical research software, and its powerful OO capabilities support systematic code organization well. Additionally, Python has excellent support for unit testing including doctests, a special flavour of unit tests written as text documents containing interactive Python sessions to exercise an API. This is sometimes referred to as a variant of literate programming. The figures in the section about the CAS in fact use doctest syntax.

PostgreSQL is used as the database backend mainly for its scalability, standard compliance and programmability,

the latter being especially useful for the reporting scripts for interfacing with GNU R.

Emores and the pysfst interface to the SFST tools are hosted by Gna! at <http://gna.org>.

8. Conclusion

The current implementation of emores finally seems to be mathematically correct, and for the simple English XMOR morphology, it even delivers results in a reasonable amount of processing time. But for the much more complex German SMOR morphology, it operates too slow to be productively usable for a substantial word list.

Currently, there are two main computational bottlenecks, and neither of them has anything to do with implementing emores in an interpreted (thus comparatively slow) scripting language: One is the compilation of single-lemma-transducers for guessed lemmas with SFST (written in C++). As with SMOR, each token generates hundreds of lemmas, this step really needs to be optimized to the theoretical maximum. But even with the present implementation, after running emores for some days, the database SQL for performing abductive inference becomes slow (response time to be measured in minutes). At present, the SQL constructs are written purely in a declarative manner, without considering performance at all. So there should be room for improvement.

The implementation of emores aims at automatically processing very large word lists derived from corpora for the German SMOR morphology. But the lemma clustering shows at a glance that the abductive selectivity is too weak for magically generating a correct lexicon, even if all possible word forms were found in the corpora — which is unlikely, as Zanchetta and Baroni observed for Italian. [7]

The presented inference logic should be a valuable aid for a human lexicographer, and as it is all Free Software, it can be adapted to many needs.

References

- [1] Gnu general public license, version 3.
- [2] G. Deutscher. On the misuse of the notion of ‘abduction’ in linguistics. *Journal of Linguistics*, 38(03):469–485, 2002.
- [3] G. Polya. *How to solve it*. Princeton University Press, 1945.
- [4] H. Schmid. Stuttgart finite state transducer tools.
- [5] H. Schmid, A. Fitschen, and U. Heid. Smor: A german computational morphology covering derivation, composition, and inflection. 2004.
- [6] J. Spolsky. The law of leaky abstractions. *Joel on Software*, 2002.
- [7] E. Zanchetta and M. Baroni. Morph-it! a free corpus-based morphological resource for the italian language. *Corpus Linguistics 2005*, 1(1), 2005.