Processing historical texts with contemporary NLP tools

Krzysztof Jassem*, **Paweł Skórzewski***

*Adam Mickiewicz University in Poznań
Umultowska 87, 61-614 Poznań, Poland
{jassem, pawel.skorzewski}@amu.edu.pl

Abstract

The paper outlines a method for processing historical texts: A text is converted into its modernized equivalent by a tool called diachronic normalizer, which is applied in a pipeline along with other text processors. The solution presented in the paper consists in embedding the diachronic normalizer into PSI-Toolkit. The toolkit architecture allows for late disambiguation, which is crucial whilst dealing with inter-word dependencies in diachronic normalization. The pipeline feature enables to further process normalized texts with contemporary NLP tools, such as parsers or translators. The solution is open for use and may be easily extended to other user-customized normalization tasks.

1. Introduction

Increasing volumes of historical texts are becoming available in the digitized form. This opens up opportunities to analyze them by means of natural language processing tools, like POS-taggers (Rayson et al., 2007) or named entity recognizers (Nissim et al., 2004). Some aspects of creating tools dedicated for processing historical texts are discussed in (Piotrowski, 2012). Historical texts may, however, be successfully treated with NLP tools designed for contemporary languages: texts processed with such tools may be post-processed manually, a tool may be re-trained on historical text, or a historical document may be pre-processed by transforming words to their contemporary forms. In our approach we use the last method, which we call diachronic normalization.

A standard NLP procedure applies modular approach, where the disambiguated output of one tool (e.g. POS-tagger) becomes the input of another (e.g. syntax analyzer). Diachronic normalization does not fully submit to such treatment, as in some cases disambiguation decisions need to be postponed to later stages. An example for the Polish language is the joint spelling of the particle *nie* ('not') with verbs (e.g. *niechciał*), which should be normalized to contemporary disjoint spelling (*nie chciał* '(he) did not want'). The normalization process should take into account that the verb stem may undergo changes, as is the case of the infinitive form: archaic: *módz*; contemporary: *móc* ('to be able to'). Thus, the appropriate normalization for the word *niemódz* should be carried out in the following steps: the original word is normalized to its contemporary writing (*niemóc*), a word separation rule produces *nie móc*, the morphological analyser confirms that *móc* is the verb, and only then is the transformation triggered. The process is illustrated in Table 1, where the treatment of the word *niemódz* is contrasted to that of the word *niedziela* ('Sunday').

The paper proposes a solution to the task – the diachronic normalization is inserted to a pipeline of NLP tools. Such an approach makes it possible to overcome difficulties exemplified above.

The paper is structured as follows: In Section 2. we give the historical insight into diachronic normalization and sketch the methods applied in our solution. Section 3. overviews NLP Toolkits designed for the Polish language. Section 4. describes the insertion of the diachronic normalization into one of those sets, PSI-Toolkit, and presents some scenarios for using the tool. In Section 5. we give a recipe on how to apply the presented method to customized text normalization (not necessarily diachronic normalization).

2. Diachronic normalization of historical texts

The first attempts at rule-based diachronic normalization used for historical text in English were described by (Rayson et al., 2007) and (Baron et al., 2009). Similar studies were conducted for German (Archer et al., 2006). There, context rules operated at the level of letters instead of words. The normalization rules may be derived from corpora, as (Hauser and Schulz, 2007) and (Bollmann et al., 2011) showed for German. Diachronic normalization may be also performed using a noisy channel model, as described by (Oravecz et al., 2010) on the example of Old Hungarian texts. The research on diachronic normalization was also conducted for Swedish (Pettersson et al., 2012), Slovene (Scherrer and Erjavec, 2013), Spanish (Porta et al., 2013) and Basque (Etxeberria et al., 2016).

Normalization of Polish historic texts was first tackled by Waszczuk in 2012[1] for the sake of the SYNAT project[2] (a reference to the SYNAT project may be found in (Mykowiecka et al., 2012), where the authors report on the paper-into-electronic conversion of an Old Polish dictionary). The solution, called *hist-pl-transliter*, is designed for the creation of transliteration rules, which allow for the usage of character classes within context constraints. The rule set is not large and consists of rules for transliteration of diacritic letters, and a few context-dependent rules.

The most recent effort in the area has been carried out within the KORBA project[3] (Bronikowska and Modrzejewski, 2017). The task consists in the transliteration of Polish texts originating from the 17th and 18th centuries. The solution uses the Ameba Supertool[4], which applies a

[1] https://hackage.haskell.org/package/hist-pl-transliter
[2] http://synat.nlp.ipipan.waw.pl
[3] http://clip.ipipan.waw.pl/KORBA
[4] https://bitbucket.org/jsbien/pol

input	spelling change	separation	confirmation	accept change?
niedziela	*niedziela*	↗ *nie dziela* ↘ *niedziela*	is *dziela* a verb?	No. Reject: *nie dziela* Accept: ***niedziela***
niemódz	*niemóc*	↗ *nie móc* ↘ *niemóc*	is *móc* a verb?	Yes. Accept: ***nie móc*** Reject: *niemóc*

Table 1: The process of normalization

large set of context-dependent rules defined by means of regex-based formalism.

Our approach differs from the two previous efforts for Polish in the computational aspect – it applies finite-state transducers, as well in the linguistic background – the set of rules for diachronic normalization is based on the historical description of Polish orthographic changes gathered in (Malinowski, 2011). We have designed a machine-readable, Thrax-compliant (Tai et al., 2011; Roark et al., 2012) rule formalism, into which we re-wrote the human-readable rules described by Malinowski. Details on the formalism may be found in (Jassem et al., 2017). An example rule is given here:

```
Infinitive_z = ("e" | "ó") ("dz" : "c");
Rule001_02 = CDRewrite[Infinitive_z,
                       NonEmptyString,
                       End,
                       Any*];
```

The first line of the above code is the declaration of a transducer `Infinitive_z` that replaces *dz* with *c* if it follows *e* or *ó*. The second line is the declaration of transducer `Rule001_02` that performs a context-dependent rewrite (`CDRewrite`) using the transducer `Infinitive_z` on any location in the text where the left-handed constraint (non-empty string) and the right-handed constraint (end of the word) are satisfied. The rule has been created to change the infinitive forms of verbs such as *biedz* or *módz* into their contemporary forms: *biec* ('to run'), *móc* ('to be able to'). We named our solution `iayko`, for an archaic word *iayko*, whose contemporary form is *jajko* ('an egg').

3. NLP toolkits for the Polish language

3.1. Language Tool

The first NLP Toolkit dealing with the Polish language was developed for the single task of error correction. Language Tool (Miłkowski, 2010) is an open-source proofreading tool. In order to perform the desired task Language Tool (LT) requires the following text resources and processors: an exhaustive dictionary of inflected forms, a tokenizer, a sentence-splitter, a lemmatizer (or at least a stemmer), and a shallow parser. As all of these elements are parts of the system, LT may be regarded as an integrated NLP Toolkit. The system uses an XML-compliant language for correction rules. The code below shows how the system deals with a common typographical mistake:

```
<rule id="Adress_Address"
      name="Address">
  <pattern>
    <token>adress</token>
  </pattern>
  <message>
    Did you mean
    <suggestion>address</suggestion>?
  </message>
</rule>
```

LT might be adapted for diachronic normalization. Here is an example of a diachronic rule, which would prompt to convert the archaic spelling of the verb infinitive *módz* into its contemporary spelling *móc*, formulated in the LT schema:

```
<rule id="MDZ_MC" name="módz->móc">
  <pattern>
    <token regexp='yes'>\bmódz\b</token>
  </pattern>
  <message>
    Contemporary spelling of módz is móc
  </message>
  <example correction=''>
    <marker>módz</marker> coś zrobić
  </example>
  <example>
    móc coś zrobić
  </example>
</rule>
```

The LT formalism allows for the usage of morphological tags. Here is an example of a potential diachronic rule that takes advantage of this feature. The rule separates the particle *nie* from a verb that follows the particle:

```
<rule id="NIEVERB" name="nie+Verb">
  <pattern>
    <token>nie</token>
    <token inflected='yes' postag='verb'/>
  </pattern>
  <message>
    Separate <suggestion>nie</suggestion>
    from a verb form
  </message>
  <example correction=''>
    niechce przyjść
  </example>
  <example>
    nie chce przyjść
  </example>
</rule>
```

3.2. Multiservice

Mulitiservice (Ogrodniczuk and Lenart, 2012) is a platform that integrates tools used by the linguistic community in Poland, such as morphosyntactic analyzers, shallow and deep parsers, sentiment recognizers, named entity recognizer, text summarizers, coreference analyzers or mention

detector. The tools may be linked into pipelines. The user of the on-line Multiservice demo version is helped by a smart assistant, which allows him/her to add only such tools to the pipeline which do not require any other information than that delivered by preceding tools: The morphosyntax analyzers may work on plain text and thus they are allowed to begin the pipeline. Other tools, which require morphosyntactic information, are allowed to be chained only after one of the morphosyntax analyzers has been added to the pipeline. Currently, Multiservice does not allow users to create their own tools based on the platform's architecture.

3.3. PSI-Toolkit

A distinct feature of the PSI-Toolkit is that all its processors operate on a common lattice-based data structure, called PSI-lattice. As described in (Graliński et al., 2013) PSI-lattice is defined as a graph where vertices represent the intra-character points in input string and edges represent input characters and subsequent annotations. This feature has proved crucial for our approach to diachronic normalization.

The current list of PSI-Toolkit processors is available at *http://psi-toolkit.amu.edu.pl/help/documentation.html*. The list consists of data readers, text segmenters, lemmatizers, POS-taggers, bilingual lexicons, syntactical parsers, machine translators and data writers in various formats (both textual and graphical).

The PSI-lattice allows for easy extension of the toolkit. For example, some external processors, such as Link Grammar parser (Sleator and Temperley, 1993) or Morfologik lemmatizer[5] (Miłkowski, 2010; Woliński et al., 2012) have been incorporated. This feature facilitated the embedding of our new tool, diachronic normalizer, into the toolkit.

PSI-Toolkit allows users to provide custom linguistic resources. This will enable to apply our solution for normalization tasks, other than diachronic normalization.

A PSI-Toolkit command may be formed as a pipeline of tools. A pipeline of processors may be shortened by means of aliases – one-word alternative names for processor sequences.

PSI-Toolkit commands may be called from the command line in Linux-type operating systems. In this scenario, PSI-tools may be pipe-lined with Linux filtering programs.

4. Embedding the diachronic normalizer into PSI-Toolkit

It is our intention to allow users create customized normalizers, either based on our set of rules, or totally independent, possibly intended for other normalization tasks.

4.1. Docker technology

In order to improve the system portability we applied a container technology, called Docker[6]. A container is a virtualization engine that allows for running processes in an isolated environment. In contrast to virtual machines, containers do not contain a full operating system, but only libraries and settings required for the process to run. As containers can share operating system components (such as the kernel), they are more lightweight than virtual machines. The Docker's infrastructure makes PSI-Toolkit portable and flexible.

4.2. Running PSI-Toolkit with Docker

To install Docker under Ubuntu Linux distribution, you should follow the instructions provided on the project's website: *https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu*.

After Docker's installation, the dockerized PSI-Toolkit can be used as if it were installed and configured on user's machine:

```
echo "text to process" | docker run -i
  skorzewski/psi-toolkit
  psi_toolkit_pipeline
```

Here is an example of a command that uses our diachronic normalization tool, `iayko`.

```
echo 'iayko czy nie yaiko' | docker run -i
  skorzewski/psi-toolkit iayko
```

As a result, a normailzed (contemporary) text will appear:

```
jajko czy nie jajko
```

Docker will download the relevant container if not found in the local Docker cache and run its contents. The first attempt may take some time because Docker needs to download the container image.

Docker can also be installed on Microsoft Windows 10 Professional or Enterprise 64-bit. The installer and the installation instructions can be found at *https://store.docker.com/editions/community/docker-ce-desktop-windows*.

4.3. Examples of usage

Here are some examples of a basic usage of `iayko`:

- A historic text is normalized to its modern version, using the default set of finite-state rules:
  - pipeline:

    ```
    iayko --lang pl
    ```

  - input: *naley w puhar i doday iayko*
  - output: *nalej w puchar i dodaj jajko*
- Normalized text may be used as an input for further processing, e.g. machine translation:
  - pipeline:

    ```
    iayko --lang pl ! parse ! bilexicon
      --lang pl --trg-lang en !
      transferer --lang pl --trg-lang en
    ```

  - input: *naley w puhar i doday iayko*

[5] http://morfologik.blogspot.com

[6] https://www.docker.com

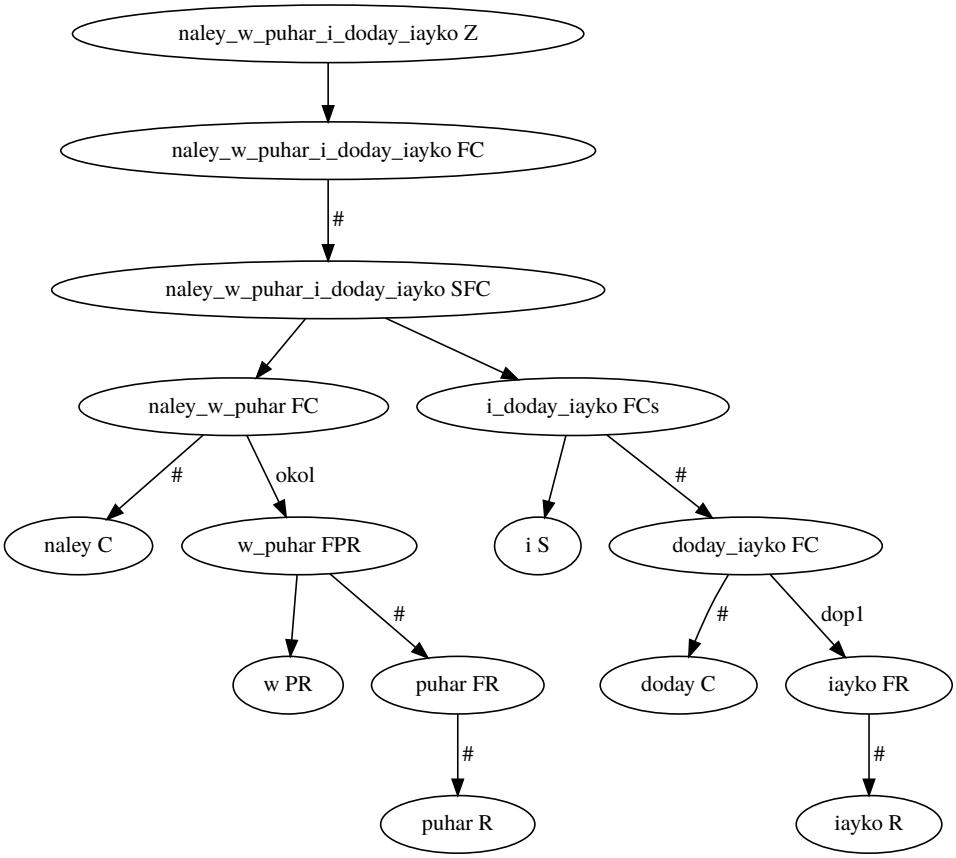

Figure 1: The parse tree for a historic text

  - output: *pour in the cup and add the egg*

- The normalization is used to obtain the parse tree for a historic text:
  - pipeline:

```
iayko --lang pl ! parse !
  draw-parse-tree
```

  - input: *naley w puhar i doday iayko*
  - output: see Figure 1

A comprehensive set of various usage examples may also be found at *http://psi-toolkit.wmi.amu.edu.pl/help/processor.psis?name=iayko*

4.4. Morphological diachronic normalizer

One of the limitations of `iayko` as a normalizer is that its normalization rules are indifferent to morphological features of normalized words. For example, the archaic rules of spelling required the particle *nie* ('not') to be spelled jointly with the verb, and the contemporary spelling of *nie* with verb is separate. On the other hand, the contemporary spelling of *nie* with adjectives or nouns is joint. So the normalizer should insert a space between the particle *nie* and the verb, but not between the particle *nie* and an adjective.

To address this issue, we created two new processors:

- a conditional diachronic normalizer, called `niema` (for an archaic word *niema*, whose contemporary form is *nie ma* 'there is no'),
- a selector annotator, called simply `selector`.

The lattice-based structure of PSI-Toolkit allows us to perform the process of morphological diachronic normalization by simply adding relevant edges to the initial lattice. The whole process is carried out by the following pipeline:

```
iayko ! niema ! lemmatizer ! selector
```

There is an alias `diachronizer` provided for this pipeline to facilitate its use.

Let's walk through all four steps of the pipeline on the example of input text *niemódz*:

1. `iayko` applies unconditional normalization rules to the input text. The rule *dz→c* is applied and the edge *niemóc* (transient form) is appended to the lattice.
2. `niema` applies conditional normalization rules on the output of `iayko`. The set of rules includes (among others) the separation of the particle *nie* before the verb. The edge sequence *nie móc* is appended to the lattice with the attached information about the constraint that the second part should be a verb.
3. A morphological analyzer (lemmatizer) adds morphological information for token edges in the lattice. In particular, the word *móc* is annotated as a verb.

4. `selector` finds that the constraint of *móc* being a verb is satisfied, and this edge is marked as selected. If the constraint weren't satisfied, a fallback edge (*niemóc*, in this case) would be marked as selected instead.

Like in the case of `iayko`, the rules for the conditional diachronic normalizer `niema` apply the Thrax formalism. However, they are supplemented by an additional line, starting with `%`, which specifies the morphological constraint. Here are two examples:

```
    separate_nie = ("nie" ("" : " ")
                   NonEmptyString);
%*+verb
    Rule1 = CDRewrite[separate_nie,
                      Beginning,
                      End,
                      Any*];
```

The rule `Rule1` sends to `selector` the request to accept the change only if the outcome consists of two tokens: the particle *nie* (`*` denotes that any morphological form is accepted) and a verb.

```
    add_y = "" : "y";
%verb&aspect=imperf&number=pl&person=pri
    Rule2 = CDRewrite[add_y,
                      AnyPrefix,
                      End,
                      Any*];
```

The rule `Rule2` sends to `selector` the request to accept the change only if the outcome consists of an adequate verb form (imperfective aspect, plural number, first person). This works for verbs such as *zrobim→zrobimy* ('we will do').

5. Creating a customized normalizer

The capabilities of PSI-Toolkit's normalizers `iayko` and `niema` can be extended by writing customized normalization rules. The normalization rules can be created using Thrax – a language for defining finite state grammars[7].

In fact, `iayko` and `niema` can be used not only for diachronic normalization, but with the right set of Thrax rules, they can act as any desired normalizers.

Suppose you have PSI-Toolkit installed as a Docker container, as described in Section 4.2. You can write a finite-state grammar using Thrax, and save it to a file, let's say *my_grammar.grm*. This file can be stored in any location, but you should remember to provide the whole file path when you use it:

```
docker run -i skorzewski/psi-toolkit iayko
  --grm path/to/my_grammar.grm
```

Grammars for `iayko` and `niema` can be also written in Markdown format, which allows for easier commenting, as well as for adding `niema` constraints:

```
docker run -i skorzewski/psi-toolkit niema
  --md path/to/my_grammar.md
```

[7] http://www.openfst.org/twiki/bin/view/GRM/Thrax

The processing can be limited to a selected transducer:

```
docker run -i skorzewski/psi-toolkit iayko
  --grm path/to/my_grammar.grm
  --fst MyTransducer
```

The normalization will run much slower compared to using the embedded grammars, because the system needs to compile the text grammar into the FAR archive. If you intend to use your grammar more than once, you can ask the system to save the compilation result to a FAR file for future use:

```
docker run -i skorzewski/psi-toolkit iayko
  --grm path/to/my_grammar.grm
  --fst MyTransducer
  --save-far path/to/compiled.far
```

Then, you can use the compiled grammar to normalize texts as fast as with the embedded normalizer:

```
docker run -i skorzewski/psi-toolkit iayko
  --far path/to/compiled.far
  --fst MyTransducer
```

6. Conclusions

The paper reports on an attempt to process historical texts with contemporary NLP tools. The approach consists in applying diachronic normalization executed by means of the Thrax platform. The normalization process is executed inside a pipeline of text annotators, included in PSI-Toolkit. The solution enables to reject unwanted changes, thus increasing accuracy on the one hand, and to process the historical input with NLP tools designed for contemporary texts on the other. The solution may be easily applied for other normalization tasks.

7. References

Archer, Dawn, Andrea Ernst-Gerlach, Sebastian Kempken, Thomas Pilz, and Paul Rayson, 2006. The identification of spelling variants in English and German historical texts: manual or automatic? In *Digital Humanities 2006*. Paris, France: CATI, Université Paris-Sorbonne.

Baron, Alistair, Paul Rayson, and Dawn Archer, 2009. Automatic standardization of spelling for historical text mining. In *Digital Humanities 2009*.

Bollmann, Marcel, Florian Petran, and Stefanie Dipper, 2011. Rule-based normalization of historical texts. In *Proceedings of the International Workshop on Language Technologies for Digital Humanities and Cultural Heritage*. Hissar, Bulgaria.

Bronikowska, Renata and Emanuel Modrzejewski, 2017. The enrichment of the lexical information and the corpus resources by using the results of the morphological analysis of historical texts.

Etxeberria, Izaskun, Iñaki Alegria, Larraitz Uria, and Mans Hulden, 2016. Evaluating the noisy channel model for the normalization of historical texts: Basque, Spanish and Slovene. In *In Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC-2016)*.

Graliński, Filip, Krzysztof Jassem, and Marcin Junczys-Dowmunt, 2013. PSI-Toolkit: A Natural Language Processing Pipeline. In Adam Przepiórkowski, Maciej Piasecki, Krzysztof Jassem, and Piotr Fuglewicz (eds.), *Computational Linguistics*, volume 458 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, pages 27–39.

Hauser, Andreas W and Klaus U Schulz, 2007. Unsupervised learning of edit distance weights for retrieving historical spelling variations. In *Proceedings of the First Workshop on Finite-State Techniques and Approximate Search*. Borovets, Bulgaria.

Jassem, Krzysztof, Filip Graliński, Tomasz Obrębski, and Piotr Wierzchoń, 2017. Automatic diachronic normalization of Polish texts. To appear.

Malinowski, Maciej, 2011. *Ortografia polska od II połowy XVIII wieku do współczesnoci. Kodyfikacja, reformy, recepcja*. Ph.D. thesis, Uniwersytet Śląski w Katowicach, Katowice.

Miłkowski, Marcin, 2010. Developing an open-source, rule-based proofreading tool. *Software: Practice and Experience*, 40(7):543–566.

Mykowiecka, Agnieszka, Piotr Rychlik, and Jakub Waszczuk, 2012. Building an electronic dictionary of Old Polish on the base of the paper resource. In Petya Osenov, Stelios Piperidis, Milena Slavcheva, and Cristina Vertan (eds.), *Proceedings of the Workshop on Adaptation of Language Resources and Tools for Processing Cultural Heritage at LREC 2012*. European Language Resources Association (ELRA).

Nissim, Malvina, Colin Matheson, and James Reid, 2004. Recognising geographical entities in Scottish historical documents. In *Proceedings of the workshop on geographic information retrieval at SIGIR ACM 2004*. Sheffield, UK.

Ogrodniczuk, Maciej and Michał Lenart, 2012. Web Service integration platform for Polish linguistic resources. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC 2012*. Istanbul, Turkey: ELRA.

Oravecz, Csaba, Bálint Sass, and Eszter Simon, 2010. Semi-automatic normalization of Old Hungarian codices. In *Proceedings of the ECAI 2010 Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH 2010)*. Lisbon, Portugal.

Pettersson, Eva, Beáta Megyesi, and Joakim Nivre, 2012. Rule-based normalisation of historical text – a diachronic study. In *Empirical Methods in Natural Language Processing: Proceedings of the 11th Conference on Natural Language Processing (KONVENS 2012)*. Vienna, Austria: Österreichische Gesellschaft für Artificial Intelligence (ÖGAI).

Piotrowski, Michael, 2012. *Natural Language Processing for Historical Texts*. San Rafael, CA, USA: Morgan & Claypool.

Porta, Jordi, José-Luis Sancho, and Javier Gómez, 2013. Edit transducers for spelling variation in Old Spanish. In *Proceedings of the workshop on computational historical linguistics at NODALIDA 2013; NEALT Proceedings Series 18*, number 87. Oslo, Norway: Linköping University Electronic Press; Linköpings Universitet.

Rayson, Paul, Dawn Archer, Alistair Baron, and Nicholas Smith, 2007. Tagging historical corpora – the problem of spelling variation. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum fr Informatik.

Roark, Brian, Richard Sproat, Cyril Allauzen, Michael Riley, Jeffrey Sorensen, and Terry Tai, 2012. The OpenGrm open-source finite-state grammar software libraries. In *Proceedings of the ACL 2012 System Demonstrations*. Jeju Island, Korea: Association for Computational Linguistics.

Scherrer, Yves and Tomaž Erjavec, 2013. Modernizing historical Slovene words with character-based SMT. In *BSNLP 2013-4th Biennial Workshop on Balto-Slavic Natural Language Processing*.

Sleator, Daniel and Davy Temperley, 1993. Parsing English with a Link Grammar. In *Third International Workshop on Parsing Technologies*.

Tai, Terry, Wojciech Skut, and Richard Sproat, 2011. Thrax: An open source grammar compiler built on OpenFst. In *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU 2011)*, volume 12. Waikoloa Resort, Hawaii.

Woliński, Marcin, Marcin Miłkowski, Maciej Ogrodniczuk, Adam Przepiórkowski, and Łukasz Szałkiewicz, 2012. Polimorf: a (not so) new open morphological dictionary for Polish. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC'12)*. Istanbul, Turkey: European Language Resources Association (ELRA).