Finite-State Canonicalization Techniques for Historical German

Bryan Jurish

Dissertation zur Erlangung des akademischen Grades
doctor philosophiæ (Dr. phil.)

eingereicht bei der
Humanwissenschaftliche Fakultät
der Universität Potsdam

Januar 2011

Gutachter: Prof. Dr. Peter Staudacher,
Prof. Dr. Wolfgang Klein

Datum der mündlichen Prüfung: 6. Oktober 2011

Published online at the
Institutional Repository of the University of Potsdam:
URL http://opus.kobv.de/ubp/volltexte/2012/5578/
URN urn:nbn:de:kobv:517-opus-55789
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-55789

Erklärung (Declaration in German)

Ich erkläre hiermit, dass ich an keiner anderen Hochschule ein Promotionsverfahren eröffnet habe. Zudem erkläre ich hiermit, dass die Dissertation selbständig und ohne unzulässige Hilfe Dritter verfasst wurde und bei der Abfassung nur die in der Dissertation angegebenen Hilfsmittel benutzt sowie alle wörtlich oder inhaltlich übernommenen Stellen als solche gekennzeichnet wurden.

Des Weiteren erkläre ich, dass die Dissertation in der gegenwärtigen oder einer anderen Fassung bei keiner anderen Fakultät einer wissenschaftlichen Hochschule zur Begutachtung im Rahmen eines Promotionsverfahrens vorgelegen hat.

Bryan Jurish
Berlin, Januar 2011

Acknowledgements

There are a great many people who supported and influenced this work and to whom thanks are due:

First, to my advisor Peter Staudacher, who despite (or perhaps because of) his professed ambivalence to impressing others has succeeded in impressing many of his students – myself included – with a taste for the formally rigorous study of natural language; and whose patience and repeated administrative intervention enabled this work eventually to be completed, if not exactly in a timely fashion.

Thanks to my colleagues past and present in the *Deutsches Textarchiv* project, for making the application of the techniques described here possible and available to the general public. In particular to Oliver Duntze and Susanne Haaf for language-historical consultations, and to Wolfgang Klein, whose conception of the *Deutsches Textarchiv* project anticipated the need for word recognition beyond the bounds of strict contemporary orthography, for allowing me to undertake the measures I deemed appropriate in that regard.

Thanks to Burkhard Brehm, Johannes Bubenzer, Jörg Didakowski, Thomas Hanneforth, Alexander Geyken, Lothar Lemnitzer, Kay-Michael Würzner, and the various articles' anonymous reviewers for both asking and answering relevant questions. Particular thanks are due to Thomas Hanneforth and Alexander Geyken for their tireless work on the TAGH morphology, without which none of the 'interesting' aspects of the current work would have been possible.

Thanks to Wolfgang Seeker, who wrote a prototype version of the heuristic rewrite transducer used in the current work; to Marko Drotschmann, who implemented the semi-automatic alignment procedure used to bootstrap the DTA evaluation corpus; and to Henriette Scharnhorst and Henriette Ast for providing the vast majority of the manual annotations used to evaluate the methods described here.

Thanks to my parents for supporting me throughout my undergraduate studies, and for their continuing interest in my academic activities involving 'something with computers'. Finally, 2^{10} thanks to Eva Brehm-Jurish for far too much to mention, including toleration of laundry unfolded, preparation of meals unappreciated, and for an ever-available auditory apparatus; and to Mathilda and Liam Jurish for putting up with a grumpy and preoccupied Daddy.

The work described here was supported by a *Deutsche Forschungsgemeinschaft* (DFG) grant to the *Deutsches Textarchiv* (DTA) project, and by the *Berlin-Brandenburg Akademie der Wissenschaften*.

Contents

The Big Picture **1**
Introduction . . . 1
Related Work . . . 2
Submitted Publications . . . 3
Article 1 . . . 3
Article 2 . . . 4
Article 3 . . . 5
Article 4 . . . 6
Conclusion . . . 7
Document Conventions . . . 8

I Submitted Publications **11**

1 Finding Canonical Forms **13**
1.1 Introduction . . . 13
1.2 Conflation by Phonetic Form . . . 14
1.2.1 Implementation . . . 15
1.2.2 Performance . . . 17
1.2.3 Coverage . . . 18
1.3 Conflation by Lemma Instantiation Heuristics . . . 19
1.3.1 Implementation . . . 19
1.3.2 Performance . . . 21
1.3.3 Coverage . . . 22
1.4 Summary & Outlook . . . 22

2 Efficient Online *k*-best Lookup **23**
2.1 Introduction . . . 23
2.1.1 Example Application . . . 24

2.1.2 Desiderata 25
2.2 Formal Background 26
2.3 Algorithms 28
2.3.1 Semiring Weights 28
2.3.2 Online Cascade Lookup 30
2.3.3 k-Best Final States 31
2.3.4 Cutoff Threshold 34
2.3.5 Label Strings 34
2.4 Summary 37

3 Comparing Canonicalizations **39**
3.1 Introduction 39
3.2 Canonicalization Methods 40
3.2.1 Phonetic Conflation 40
3.2.2 Levenshtein Edit Distance 41
3.2.3 Rewrite Transducer 42
3.3 Evaluation 43
3.3.1 Test Corpus 43
3.3.2 Evaluation Measures 44
3.3.3 Results 45
3.4 Conclusion & Outlook 47

4 More Than Words **49**
4.1 Introduction 49
4.2 Type-wise Conflation 50
4.2.1 String Identity 51
4.2.2 Transliteration 51
4.2.3 Phonetization 53
4.2.4 Rewrite Transduction 54
4.3 Token-wise Disambiguation 55
4.3.1 Basic Model 56
4.3.2 Transition Probabilities 57
4.3.3 Lexical Probabilities 58
4.3.4 Runtime Disambiguation 60
4.3.5 Expressive Power 61
4.4 Evaluation 61
4.4.1 Test Corpus 61
4.4.2 Evaluation Measures 62

4.4.3 Results 63
4.5 Conclusion 64

II Appendices **67**

A Finite-State Components **69**
A.1 Transliteration Rules 70
A.2 Phonetization Rules 73
A.2.1 Pre-processing Filters 73
A.2.2 Character Classes 74
A.2.3 Core LTS Rules 74
A.2.4 Post-processing Filters 80
A.2.5 Phonetization FST 83
A.3 TAGH Filters 84
A.3.1 Syntactic Category Filters 84
A.3.2 Lexical Stem Filters 85
A.3.3 Target Lexica 87
A.4 Morphological Security 88
A.5 Heuristic Rewrite Rules 90
A.5.1 Character Classes 90
A.5.2 Identity Rules 91
A.5.3 Consonant Rules 91
A.5.4 Vowel Rules 94
A.5.5 Explicit Elision Rules 97
A.5.6 Unrecognized Input Rules 98
A.5.7 Miscellaneous Rules 99
A.5.8 Rewrite Filters 101

B Selected Software **105**
B.1 `unicruft`: Transliteration 105
B.2 `gfsm` & `gfsmxl`: Finite-State Operations 105
B.3 `Lingua::LTS`: LTS Rule Compiler 106
B.4 `Taxi`: Structured Text Indices 106
B.5 `moot`: HMM Tagging/Disambiguation 107
B.6 `dta-tokwrap`: XML/TEI Serialization 107
B.7 `DTA::EvalCorpus`: Alignment and Annotation 107
B.8 `DTA::CAB`: Canonicalization 108

C Corpora **109**
C.1 DWB Verse Corpus 109
C.1.1 DWB Evaluation Subcorpus 110
C.2 DTA Corpus 111
C.2.1 DTA Evaluation Subcorpus 112

Glossary **115**

Acronyms **123**

Symbols and Notation **125**

Bibliography **129**

List of Tables

1.1 Phonetization transducer performance 18

3.1 Processing time for elementary canonicalization functions . . . 45
3.2 Evaluation results for the DWB verse corpus subset 46

4.1 Evaluation results for the DTA corpus subset 63

A.1 Latin-1 transliteration rules 70
A.2 Phonetizer modifications 75
A.3 TAGH Lexical stem filters 86
A.4 Rewrite rules: plosives 91
A.5 Rewrite rules: fricatives 92
A.6 Rewrite rules: affricates 93
A.7 Rewrite rules: sonorant consonants 94
A.8 Rewrite rules: front vowels 95
A.9 Rewrite rules: back vowels 96
A.10 Rewrite rules: diphthongs 96
A.11 Rewrite rules: elisions 97
A.12 Rewrite rules: miscellaneous 99

List of Figures

1.1 Example `festival` letter-to-sound transduction 17
1.2 Example phonetic conflations 19

3.1 Example spurious Levenshtein conflations 42
3.2 Example rewrite transducer heuristics 43

4.1 Example of HMM conflator disambiguation 56
4.2 Evaluation results for the DTA corpus subset 64

List of Algorithms

2.1 Dijkstra's algorithm 29
2.2 Dijkstra for online lookup cascades 30
2.3 Dijkstra for k-best successful cascade paths 32
2.4 Dijkstra for k-best cascade output strings 35

The Big Picture

Introduction

The four submitted publications (Jurish, 2008, 2010a,b,c) address issues in the robust automatic canonicalization of unconventional natural language text, in particular historical German. Five basic methods for automatic canonicalization using both weighted and unweighted finite-state techniques were introduced, formally defined, discussed, implemented, and empirically evaluated on corpora of historical German. This summary provides a brief overview over the canonicalization strategies discussed in the individual articles and their comparative strengths and weaknesses as indicated by the empirical evaluations.

Virtually all conventional text-based natural language processing techniques – from traditional information retrieval systems to full-fledged parsers – require reference to a fixed lexicon accessed by surface form, typically trained from or constructed for synchronic input text adhering strictly to contemporary orthographic conventions. Unconventional input such as historical text which violates these conventions therefore presents difficulties for any such system due to lexical variants present in the input but missing from the application lexicon.

Traditional approaches to the problems arising from an attempt to incorporate historical text into a conventional system rely on the use of additional application-specific lexical resources to explicitly encode known historical variants. Such specialized lexica are not only costly and time-consuming to create, but also – at least in their simplest form of finite static word lists – necessarily incomplete in the case of a morphologically productive language like German, since a simple finite lexicon cannot account for highly productive morphological processes such as nominal composition.

To facilitate the extension of synchronically-oriented natural language processing techniques to historical text while minimizing the need for special-

ized lexical resources, one may first attempt an automatic *canonicalization*[1] of the input text. Canonicalization approaches treat orthographic variation phenomena in historical text as instances of an error-correction problem, seeking to map each (unknown) word of the input text to one or more extant *canonical cognates*: synchronically active types which preserve both the root and morphosyntactic features of the associated historical form(s). To the extent that the canonicalization was successful, application-specific processing can then proceed normally using the returned canonical forms as input, without any need for additional modifications to the application lexicon.

Related Work

The current approach draws heavily on previous work in error correction (Kukich, 1992) and approximate string matching (Navarro, 2001), particularly those techniques descended from the traditional Damerau-Levenshtein (DL) string edit distance (Damerau, 1964; Levenshtein, 1966). Kernighan et al. (1990); Church and Gale (1991) describe the use of explicit error models implemented as edit cost matrices together with local context information in the form of word unigram and trigram probabilities in a spelling correction application. Mays et al. (1991) use a full-fledged word trigram language model to disambiguate confusion sets generated by a traditional DL matcher in sentential context. Brill and Moore (2000) extend these approaches to include generic local string-to-string edit operations, reporting significant improvements in accuracy.

Previous work on automatic conflation of historical variants with extant word forms has been performed for English by Robertson and Willett (1993), who investigated the utility of phonetic digest algorithms (Russell, 1918; Gadd, 1988, 1990) and approximate string matching techniques (Levenshtein, 1966; Pollock and Zamora, 1984) for improving recall in databases of historical English text. Rayson et al. (2005) describe an automatic "variant detector" for historical English which uses a manually constructed set of letter replacement heuristics to canonicalize[3] historical forms, reporting a substan-

[1] Despite its morphological redundance and the existence of a non-redundant cognate (*canonization*), the term *canonicalization* has established itself in the domain of information processing as the term of choice for denoting "a process for converting data ... into a 'standard', 'normal', or canonical form"[2], uses of the non-redundant *canonization* tending to restrict themselves to the religious domain.

[2] `http://en.wikipedia.org/wiki/Canonicalization`, accessed December 10, 2010.

[3] Rayson et al. use the verb "normalise" rather than "canonicalize" or "canonize" to denote the mapping of historical variants to equivalent extant forms.

tial improvement in accuracy on a small test set compared to conventional spell-checkers.

The RSNSR ('rule-based search in text databases with nonstandard orthography') research group at the University of Duisburg-Essen has performed extensive research on automatically mapping contemporary German word forms to their historical spelling variants (i.e. inverse canonicalization) for use in information retrieval applications (Kempken, 2005; Kempken et al., 2006; Pilz et al., 2006, 2008; Ernst-Gerlach and Fuhr, 2006, 2007, 2010), with particular focus on the use of specialized string distance measures trained from a set of manually confirmed evidence pairs using machine-learning techniques (Ristad and Yianilos, 1998). More relevant work on historical German has been performed by the IMPACT ('Improving Access to Text') research group at the University of Munich. The primary focus of the IMPACT project is the improvement of optical character recognition (OCR) applications for historical German documents. In this context, the Munich group has investigated the utility of heuristic canonicalization patterns[4] as well as traditional static "witnessed" dictionaries with respect to vocabulary coverage (Gotscharek et al., 2009c) and information retrieval tasks (Gotscharek et al., 2009b), while Reffle et al. (2009) use channel error profiles together with a local word trigram heuristic to identify and correct "false friends" in OCR output for historical German.

Finite-state approaches to error correction include that of Oflazer (1996), who defines a specialized algorithm for approximate matching of an input string with a regular language using traditional DL edit costs to guide the search. Schulz and Mihov (2002) present constructions for pre-compiling DL string matchers as finite state machines, while Pirinen and Lindén (2010) describe a spelling correction architecture expressed as a composition of weighted finite-state transducers, using the DL edit distance to represent the error modelling component.

Submitted Publications

Article 1 (Jurish, 2008)

The first submitted article (Jurish, 2008) introduces the basic canonicalization framework for historical text. Assuming a non-deterministic application analysis framework, the canonicalization task is characterized in terms of word type conflation, and two high-level type-wise conflation relations are proposed:

[4]Gotscharek et al. (2009c) refer to these in terms of a "hypothetical dictionary".

phonetic conflation and heuristic lemma instantiation. The proposed techniques are evaluated with respect to both runtime processing efficiency and coverage by the high-precision TAGH morphological analyzer for present-day German (Geyken and Hanneforth, 2006) over a 5.5 million word corpus of heterogeneous historical German verse extracted from dictionary quotation evidence.

In general, phonetic conflation equates those word types which share a common phonetic form, as determined by some computable phonetization function. The phonetization function used in the current work (Jurish, 2008, 2010b,c) was adapted from a deterministic rule-set (Möhler et al., 2001) originally designed for the popular text-to-speech synthesis system `festival`.[5] In this article, I present a method by which any such rule-set in the standard `festival` syntax may be converted to an equivalent finite state transducer, achieving an empirical reduction of over 84% in required processing time with respect to the native `festival` interpreter.

The second conflation method introduced in this article makes use of morphology induction techniques (Yarowsky and Wicentowski, 2000; Baroni et al., 2002) together with domain-specific restrictions to extend phonetic conflation sets for a corpus of dictionary quotation evidence by means of lemma instantiation heuritics. I show that by using the dictionary structure of the corpus to restrict a morphology induction algorithm, the number of required string comparisons can be reduced by a factor of over ten thousand, making the use of these techniques viable for large structured corpora. While not generally applicable to arbitrary corpora of historical text, these heuristics can be employed to "bootstrap" lexical resources if a dictionary corpus is available.

Use of the phonetic conflation strategy reduced type-wise TAGH coverage errors on the historical verse corpus by over 21.1%, and token-wise coverage errors by over 48.2%. Relaxation of the strict identity criterion by the domain-specific lemma instantiation heuristics provided an additional type-wise coverage error reduction of over 26.7% and an additional token-wise coverage error reduction of over 33.8%.

Article 2 (Jurish, 2010a)

The coverage results from Jurish (2008) showed that relaxing the strict identity criterion of the phonetic conflation technique could improve coverage by a conventional lexicon. Jurish (2010a) introduces an algorithm by means of which such relaxed conflation sets may be efficiently computed for arbitrary

[5] `http://www.cstr.ed.ac.uk/projects/festival`

input text even in the presence of an infinite target lexicon as exhibited by morphologically productive languages like German.

The problem is characterized in terms of k-best lookup operations in weighted finite-state transducer cascades. The case of relaxed conflation is realized as a simple cascade consisting of a weighted "editor" transducer which models likelihood of diachronic variation and a (possibly infinite) target acceptor representing the synchronic lexicon. Due to the combinatorial properties of the finite-state composition operation, even such a comparatively modest cascade cannot be compiled offline by conventional computing architectures.

The algorithm presented in this paper is based on the well-known Dijkstra algorithm (Dijkstra, 1959) for best-path search in weighted graphs. Necessary extensions to the algorithm to accomodate arbitrary bounded semiring weights, online expansion of cascade transitions, restriction to the k-best final states, and direct computation of the k-best output strings are presented. In particular the latter aspect differentiates this approach from other popular best-path algorithms (Cormen et al., 2001; Mohri, 2002). The correctness conditions and running time of the algorithm are discussed for both worst- and average-case scenarios, as are the implications of a greedy termination strategy and an external cutoff threshold. Some properties of "degenerate" cascades are identified, and the subclass of lookup cascades for which the algorithm is expected to terminate is restricted accordingly.

Article 3 (Jurish, 2010b)

One important question unaddressed by the previous papers is that of the reliability of the various canonicalization techniques. Improvement in coverage by a synchronic lexicon is of course desirable, but not if such improvement comes at the expense of reliability. In order to address this question, a small test corpus was extracted from the corpus of historical verse used in Jurish (2008), and a unique canonical form was manually assigned to each token. In Jurish (2010b), I use this test corpus as a gold standard to provide a quantitative evaluation of three non-trivial canonicalization strategies in terms of the information retrieval notions of precision and recall (van Rijsbergen, 1979).

The first of the three canonicalization techniques evaluated here is the phonetic conflation strategy from Jurish (2008). The remaining two techniques are both reducible to best-path lookup operations in simple weighted finite-state cascades, which were computed using a C library implementation of the algorithm from Jurish (2010a). Both cascade-based canonicalizers use the (infinite) input language of the TAGH morphology transducer to represent

the target lexicon, differing only in the definition of the "editor" transducer: the component responsible for modelling likelihood of diachronic variation. The first tested editor transducer is just a finite-state implementation of the traditional Levenshtein string edit distance (Levenshtein, 1966). The second editor was compiled from a heuristic two-level "rewrite" rule-set whose rules were manually constructed to reflect linguistically plausible patterns of diachronic variation as observed in the set of lemma-instance pairs extracted in Jurish (2008).

One additional naïve string identity technique was tested to simulate the lack of any canonicalization preprocessing. With both type- and token-wise precision values over 99%, this was by far the most precise of all tested techniques, but the poorest in terms of recall, supporting the claim that a synchronically-oriented lexicon cannot adequately account for the degree of graphematic variation in historical text. The phonetic and Levenshtein distance techniques performed similarly to one another at the type level, but a sharp drop in phonetic precision was observed at the token level, due to a small number of phonetic misconflations involving high-frequency types. The linguistically motivated rewrite cascade performed best overall, achieving a type-wise harmonic mean F=93.2% and a token-wise F=95.8%, providing a harmonic mean error reduction of over 45% with respect to both phonetic and Levenshtein-distance conflation. These results lend empirical support to the intuition that a fine-grained context-sensitive pseudo-metric incorporating linguistic knowledge can more accurately model diachronic processes than a general-purpose metric like the Levenshtein distance.

Article 4 (Jurish, 2010c)

The canonicalization techniques discussed in the previous articles have all been formally expressible as word type conflation relations: binary relations over the set of all word strings which induce a (pseudo-) equivalence class or "conflation set" for each input word type, independent of its surrounding context. Already apparent in the data from Jurish (2010b) is a typical precision-recall trade-off pattern among these type-wise conflation techniques: the ultra-conservative string identity conflator despite its near-perfect precision shows quite poor recall, while the more ambitious high-recall conflators such as phonetic identity or rewrite transduction tend to be disappointingly imprecise. In Jurish (2010c), I present a technique for disambiguation of type conflation sets at the token level using a Hidden Markov Model whose lexical probability matrix is dynamically computed from the candidate conflations, and evaluate its performance on a corpus of historical German prose.

The type-wise conflators used in this article are the phonetic and rewrite conflators from Jurish (2010b). Additionally, a conservative transliteration conflator is defined and used as an alternative to naïve string identity.[6] Treating the conflation sets returned by these subordinated type-wise conflators as canonicalization hypotheses, the disambiguator chooses an optimal sequence of token-wise unique canonical forms for each input sentence. The best-sequence optimization problem itself is computed by a standard application of the Viterbi algorithm (Viterbi, 1967). Since the "hidden" states of the disambiguator represent extant word forms, the model's transition probabilities can be estimated using standard maximum likelihood techniques from a corpus of contemporary text (Manning and Schütze, 1999). The model's lexical probabilities however require reference to both extant and historical forms, so these cannot be directly estimated in the absence of representative training data. Instead, lexical probabilities are dynamically re-computed for each input sentence using a Maxwell-Boltzmann distribution with a conflator-sensitive distance function.

The proposed disambiguation architecture was evaluated on an information retrieval task over a new gold standard corpus of manually confirmed canonicalizations of historical German prose, which was constructed by a two-phase procedure of automatic alignment and manual review. Use of the token-wise disambiguator provided a precision error reduction of over 94% with respect to the highest-recall method, and a recall error reduction of over 71% with respect to the most precise method. Overall, the proposed disambiguation method proved very competitive at the type level (F=96.9%, versus F=97.0% for the best type-wise method, the heuristic rewrite cascade), and outperformed all other tested methods at the token level, achieving a token-wise harmonic mean F=99.4%.

Conclusion

Five basic techniques for automatic canonicalization of historical text were introduced, implemented, and empirically evaluated: deterministic transliteration, phonetic conflation, lemma instantiation heuristics, best-path lookup in a weighted finite-state cascade, and dynamic Hidden Markov Model disambiguation. The poor coverage and low recall displayed by a naïve string

[6]Only the status of the transliteration function as a conflator in its own right is new here; the transliterator described in Jurish (2010c) was briefly mentioned already in Jurish (2008), where it was employed as an "orthographic preprocessor". The same function was implicitly used in Jurish (2010b) to preprocess the inputs passed to the phonetic and cascade conflators.

matching strategy support the initial hypothesis that a synchronically oriented lexicon cannot adequately account for graphematic variation observable in historical text.

The substantial improvement in recall by the simple phonetic conflator with respect to both string identity and transliteration suggests that for German, the phonological and phonetic properties of the language are more stable along the diachronic axis than its graphematic properties; or at least that phonetic and graphematic variations have tended to pattern disjointly. Further recall improvement by relaxation of the strict (phonetic) identity criterion through the lemma-instantiation and rewrite cascade conflators suggest that historical variation phenomena go beyond those which can be easily captured by simple deterministic techniques. The direct comparison between the manually constructed rewrite rule-set and a simple Levenshtein editor indicates that the latter general-purpose metric is too coarse to function as an accurate predictor of etymological relations.

By optimizing the path probability through the space of canonicalization hypotheses returned by the type-wise conflators, the Hidden Markov Model disambiguator was able to eliminate a large number of false positive conflations, providing a dramatic improvement in precision with only minimal loss of recall. This supports the intuition that sentential context provides useful information about the lexical disposition of unfamiliar words, a well-known phenomenon in the domain of psycholinguistics. Moreover, the fact that the disambiguator's transition probabilities were trained on contemporary text supports the hypothesis that the language's syntagmatic properties are considerably more stable than either its phonetic or graphematic properties.

Document Conventions

The rest of this document is organized as follows: The four submitted articles described above appear in chronological order as the individual chapters 1 through 4. Appendix A contains explicit characterizations of the rule-based finite-state components used in the rest of the work, appendix B provides a brief overview of selected software developed in the course of the work described here, and appendix C contains details on the various corpora involved.

The articles appearing here as chapters 1 through 4 have been reformatted for inclusion in this document. Most notably, inconsistent typographical conventions have been unified and local bibliographic references have been extracted to a global bibliography beginning on page 129. In some cases, variable names have been changed to be more consistent with those used in

the rest of this document. Typographical errors appearing in the publications have been corrected here when they were discovered, and occurrences of first person plural pronouns as used in anonymous submissions were replaced with the corresponding singular pronouns. In some cases, comments have been added to the articles as presented here which did not appear in the original publications, for example cross-references within this document. Such comments appear as footnotes explicitly marked with the prefix "EDIT".

Part I

Submitted Publications

"... meddle first, understand later. You had to meddle a bit *before you had anything to try to understand. And the thing was never, ever to go back and hide in the Lavatory of Unreason. You have to try to get your mind around the Universe before you can give it a twist."*

Terry Pratchett, *Interesting Times*

ORIGINALLY APPEARED AS:
Bryan Jurish. Finding canonical forms for historical German text. In A. Storrer, A. Geyken, A. Siebert, and K.-M. Würzner, editors. *Text Resources and Lexical Knowledge*, pages 27–37. Mouton de Gruyter, 2008.

Chapter 1

Finding Canonical Forms

1.1 Introduction

Historical text presents numerous challenges for contemporary natural language processing techniques. In particular, the absence of consistent orthographic conventions in historical text presents difficulties for any technique or system requiring reference to a fixed lexicon accessed by orthographic form, such as document indexing systems (e.g. Sokirko, 2003), part-of-speech taggers (e.g. DeRose, 1988; Brill, 1992; Schmid, 1994; Jurish, 2003), simple word stemmers (e.g. Lovins, 1968; Porter, 1980), or more sophisticated morphological analyzers (e.g. Geyken and Hanneforth, 2006). When adopting historical text into such a system, one of the most important tasks is the discovery of one or more *canonical extant forms* for each word of the input text: synchronically active text types which best represent the historical input form.[1]

The process of collecting variant forms into equivalence classes represented by one or more canonical extant types is commonly referred to as *conflation*, and the equivalence classes themselves are referred to as *conflation sets*. Given a high-coverage analysis function for extant forms, an unknown (historical) form w can then be analyzed as the disjunction of analyses over (the extant

[1] As an anonymous reviewer pointed out, the absence of consistent orthographic conventions is not restricted to corpora of historical text. Various other types of text corpora – including transcriptions of spoken language, corpora containing transcription errors, and corpora for languages with non-standard orthography – might also benefit from a canonicalization strategy such as those presented here.

members of) its conflation set $[w]$:

$$\text{analyses}(w) \quad := \quad \bigcup_{v \in [w]} \text{analyses}(v)$$

This paper describes two methods for finding conflation sets in a corpus of *circa* 5.5 million words of historical German verse extracted from quotation evidence in the digital edition of the *Deutsches Wörterbuch* (*DWB*, Bartz et al., 2004), and indexed with the TAXI document indexing system.[2] The conflation methods were implemented on the entire corpus as a TAXI plug-in module (TAXI/Grimm), and evaluated with respect to coverage by the TAGH morphology.

The rest of this paper is organized as follows: section 1.2 describes the first conflation strategy, based on identity of phonetic forms. The second strategy making use of *a priori* assumptions regarding corpus structure and permitting "fuzzy" matching via phonetic edit distance is presented in section 1.3. Finally, section 1.4 contains a brief summary of the preceding sections and a sketch of the ongoing development process.

1.2 Conflation by Phonetic Form

Although the lack of consistent orthographic conventions for middle high German and early new high German texts led to great diversity in surface graphemic forms, we may assume that graphemic forms were constructed to reflect phonetic forms.[3] Under this assumption, together with the assumption that the phonetic system of German is diachronically more stable than the graphematic system, the phonetic form of a word type should provide a better clue to the extant lemma of a historical word than its graphemic form. This insight is the essence of the "conflation by phonetic form" strategy as implemented in the TAXI/Grimm index module.

In order to map graphemic forms to phonetic forms, we may avail ourselves of previous work in the realm of text-to-speech synthesis, a domain in which the discovery of phonetic forms for arbitrary text is a well-known and often-studied problem (cf. Allen et al., 1987; Liberman and Church, 1992; Dutoit, 1997), the so-called "grapheme-to-phoneme", "grapheme-to-phone", or "letter-to-sound" (LTS) conversion problem. Use of a full-fledged LTS conversion module to estimate phonetic forms provides a more flexible and finer-grained approach to canonicalization by phonetic form than strategies using language-specific

[2] EDIT: See appendix C.1

[3] EDIT: This assumption is made explicit in Keller (1978).

phonetically motivated digest codes such as those described in Robertson and Willett (1993). The grapheme-to-phone conversion module in the TAXI/Grimm system uses the LTS rule-set distributed with the IMS German Festival package (Möhler et al., 2001), a German language module for the Festival text-to-speech system (Black and Taylor, 1997; Taylor et al., 1998).

1.2.1 Implementation

As a first step, the IMS German Festival letter-to-sound (LTS) rule-set was adapted to better accommodate both historical and contemporary forms; assumedly at the expense of precision for both historical and contemporary forms. In particular, the following changes were made:[4]

1. By default, the grapheme "h" is ignored (considered silent).
2. A single additional rule maps the grapheme sequence "sz" to voiceless [s].
3. Vowel-length estimates output by the IMS German rule-set are ignored; thus [e] and [eː] are both mapped to the canonical phonetic form [e].
4. Schwas ([ə]) predicted by the IMS German rule-set are replaced by [e] in the canonical phonetic form.
5. Adjacent occurrences of any single vowel predicted by the IMS German rule-set are replaced by a single occurrence, thus [aa], [aaa], and [aaaa] are all mapped to [a].

The adapted rule-set was converted to a *deterministic finite-state transducer* (Aho and Ullman, 1972; Roche and Schabes, 1997) using the GFSM finite-state machine utility library. Formally, the finite-state transducer (FST) used by the TAXI/Grimm LTS module is defined as the machine M_{pho} arising from the composition of two *Aho-Corasick pattern matchers* (Aho and Corasick, 1975) M_L, M_R and an additional *output filter* M_O:

$$M_{\text{pho}} = (M_L \circ M_R \circ M_O) : \mathcal{A}^* \rightarrow \mathcal{P}^* \tag{1.1}$$

where $\mathcal{A}$ is the finite *grapheme alphabet* and $\mathcal{P}$ is the finite *phone alphabet*. To define the individual component machines, let R be the (IMS German)

[4] EDIT: Although the changes described here accurately reflect the phonetization rule-set which was used to acquire the coverage results in section 1.2.3, the rule-set was further modified in the course of later work, so that the description given here does not encompass all the changes made. A full list of the changes to the phonetization rule-set can be found in appendix A.2.

Festival LTS rule-set source, a finite set of rules of the form $(\alpha[\beta]\gamma \to \pi) \in \mathcal{A}^* \times \mathcal{A}^+ \times \mathcal{A}^* \times \mathcal{P}^*$, read as: the *source grapheme string* β is to be mapped to the *target phonetic string* π if β occurs with *left graphemic context* α and *right graphemic context* γ; let $\prec$ be a linear *precedence order* on R which prevents multiple rules from applying to the same source substring (only the $\prec$-minimal rule is applied at each source position, proceeding from left to right); for a nonempty rule subset $S \subseteq R$, let $(\alpha_S[\beta_S]\gamma_S \to \pi_S) = \min_{\prec} S$; let $\text{AhoCorasick}(P) : \mathcal{A}^* \to \wp(P)^*$ be the Aho-Corasick pattern matcher for a set P of string patterns from a finite alphabet $\mathcal{A}$; let $|\cdot|$ denote string length or set cardinality, depending on context; let $\text{Reverse}(\cdot)$ denote the transducer or string reversal operation, and let $\text{Concat}(\cdots)$ denote the string concatenation operation, then:

$$\begin{aligned} M_L &\approx \text{AhoCorasick}\left(\{\alpha : (\alpha[\beta]\gamma \to \pi) \in R\}\right) \\ &: \mathcal{A}^* \to (\mathcal{A} \times \wp(R))^* \\ &: w \mapsto \text{Concat}_{i=0}^{|w|} \left\langle w_i, \left\{(\alpha[\beta]\gamma \to \pi) \in R \mid w_{(i-|\alpha|)..i} = \alpha\right\}\right\rangle \end{aligned} \tag{1.2}$$

$$\begin{aligned} M_R &\approx \text{Reverse}\left(\text{AhoCorasick}\left(\{\text{Reverse}(\beta\gamma) : (\alpha[\beta]\gamma \to \pi) \in R\}\right)\right) \\ &: (\mathcal{A} \times \wp(R))^* \to \wp(R)^* \\ &: \langle w_i, S_i\rangle_I \mapsto \text{Concat}_{i\in I}\left(S_{i-1} \cap \left\{(\alpha[\beta]\gamma \to \pi) \in R : w_{i..(i+|\beta\gamma|)} = \beta\gamma\right\}\right) \end{aligned} \tag{1.3}$$

A similar construction also using a pair of Aho-Corasick pattern matchers (analogous to M_L and M_R) is employed by Laporte (1997) for compiling a single bimachine from a set of conflict-free hand-written phonetic conversion rules. Since `festival` LTS rule-sets are not conflict-free, Laporte's technique cannot be applied directly here, and the choice of which rule to apply must be delayed until application of the filter transducer M_O:

$$\begin{aligned} M_O &\approx \left(\bigcup_{S\in\wp(R)} \left[(S : \pi_S)\left(\wp(R) : \varepsilon\right)^{|\beta_S|-1}\right]\right)^* \\ &: \wp(R)^* \to \mathcal{P}^* \end{aligned} \tag{1.4}$$

In the interest of efficiency, the rule subsets $S \in \wp(R)$ on the lower tape of the filter transducer M_O can be restricted to those which actually occur on the upper tape of the right-context transducer M_R: such a restriction represents a considerable efficiency gain with respect to the "brute force" powerset construction given in Equation 1.4. Figure 1.1 shows an example of how the various machine components work together to map the graphemic form "sache" to the phonetic form [zaxə].

Input	#	s	a	c	h	e	#
$\mathbf{M_L}$ $\longrightarrow$	$\emptyset$	$[a]ch\rightarrow a$ $[a]\rightarrow a\text{ː}$, $[c]\rightarrow k$, $[e]\rightarrow ə$, $\#[s]a\rightarrow z$, $[s]\rightarrow s$	$[a]ch\rightarrow a$, $[a]\rightarrow a\text{ː}$, $[c]\rightarrow k$, $[e]\rightarrow ə$, $[s]\rightarrow s$	$[\,a\,]ch\rightarrow a$, $[\,a\,]\rightarrow a\text{ː}$, $a[ch]\rightarrow x$, $[\,c\,]\rightarrow k$, $[\,e\,]\rightarrow ə$, $[\,s\,]\rightarrow s$	$\emptyset$	$[a]ch\rightarrow a$, $[a]\rightarrow a\text{ː}$, $[c]\rightarrow k$, $[e]\rightarrow ə$, $[s]\rightarrow s$	$\emptyset$
$\mathbf{M_R}$ $\longleftarrow$	$\emptyset$	$\#[s]a\rightarrow z$, $[s]\rightarrow s$	$[a]ch\rightarrow a$, $[a]\rightarrow a\text{ː}$	$a[ch]\rightarrow x$, $[\,c\,]\rightarrow k$	$\emptyset$	$[e]\rightarrow ə$	$\emptyset$
$\mathbf{M_O}$ $\longrightarrow$	ε	**z**	**a**	**x**	ε	**ə**	ε

Figure 1.1: Example Letter-to-Sound transduction from "sache" to [zaxə]. Here, italic "ε" indicates the empty (phonetic) string.

Finally, phonetic forms are used to conflate graphemic variants $w \in \mathcal{W}$ as equivalence classes $[w]_{\text{pho}}$ with respect to the *phonetic equivalence relation* $\sim_{\text{pho}}$ on the corpus word-type alphabet $\mathcal{W} \subseteq \mathcal{A}^*$:

$$w \sim_{\text{pho}} v \quad :\Leftrightarrow \quad M_{\text{pho}}(w) = M_{\text{pho}}(v) \tag{1.5}$$

$$[w]_{\text{pho}} \quad = \quad \{v \in \mathcal{W} : w \sim_{\text{pho}} v\} \tag{1.6}$$

Note that the equivalence class generating function $[\cdot]_{\text{pho}} : \mathcal{W} \rightarrow \wp(\mathcal{W})$ can itself be characterized as a finite-state transducer, defined as the composition of the LTS transducer with its inverse, and restricted to the alphabet $\mathcal{W}$ of actually occurring corpus word-types:

$$[\cdot]_{\text{pho}} \quad := \quad M_{\text{pho}} \circ M_{\text{pho}}^{-1} \circ \text{Id}\,(\mathcal{W}) \tag{1.7}$$

1.2.2 Performance

A finite-state LTS transducer M_{pho} was compiled from the 396 rules of the adapted IMS German Festival rule-set using the procedure sketched above. The resulting transducer contained 131,440 arcs and 1,037 states, of which 292 were final states. The compilation lasted less than 30 seconds on a workstation with a 1.8GHz dual-core processor. Performance results for the transducer representation of the LTS rule-set and for two methods using `festival`

LTS Method	Throughput (tok/sec)	Relative
`festival` (TCP)	28.53	−4875.57 %
`festival` (pipe)	1391.45	± 0.00 %
FST (`libgfsm`)	9124.69	+ 555.77 %

Table 1.1: Performance results for LTS FST compared to direct communication with a `festival` process

directly are given in Table 1.1. As expected, the transducer implementation was considerably faster than either of the methods communicating directly with a `festival` process.

1.2.3 Coverage

The phonetic conflation strategy was tested on the full corpus of the verse quotation evidence extracted from the DWB, consisting of 6,581,501 tokens of 322,271 distinct graphemic word types. A preprocessing stage removed punctuation marks, numerals, and known foreign-language material from the corpus. Additionally, a rule-based graphemic normalization filter was applied which maps UTF-8 characters not occurring in contemporary German orthography onto the ISO-8859-1 (Latin-1) character set (e.g. *œ*, $\hat{o}^{e}$, and $\hat{o}$ are mapped to *oe,* *ö*, and *o*, respectively).[5] After preprocessing and filtering, the corpus contained 5,491,982 tokens of 318,383 distinct ISO-8859-1 encoded graphemic types.

Of these 318,383 Latin-1 word types occurring in the corpus, 135,070 (42.42%) were known to the TAGH morphology (Geyken and Hanneforth, 2006), representing a total coverage of 4,596,962 tokens (83.70%). By conflating those word types which share a phonetic form according to the LTS module, coverage was extended to a total of 173,877 (54.61%) types, representing 5,028,999 tokens (91.57%). Thus, conflation by phonetic form can be seen to provide a reduction of 21.17% in type-wise coverage errors, and of 48.27% in token-wise coverage errors. Some examples of word types conflated by the phonetic canonicalization strategy are given in Figure 1.2.

[5] EDIT: The "graphemic normalization filter" used here was an early version of the conservative transliterator formally introduced in chapter 4 and explicitly defined in appendix A.1.

Extant Form w	**Phonetic Equivalence Class** $[w]_{\text{pho}}$
fröhlich	*frölich, fröhlich, vrœlich, frœlich, frölich, fröhlich, vrölich, fröhlig, frölig, ...*
Herzenleid	*hertzenleid, herzenleid, herzenleit, hertzenleyd, hertzenleidt, herzenlaid, hertzenlaid, hertzenlaidt, hertzenlaydt, herzenleyd, ...*
Hochzeit	*hochtzeit, hochzeit, hochzeyt, hochzît, hôchzît, hochzeid, ...*
Schäfer	*schäfer, schäffer, scheffer, scheppher, schepher, schäfer, schäffer, schähffer, ...*

Figure 1.2: Some words conflated by identity of phonetic form

1.3 Conflation by Lemma Instantiation Heuristics

Despite its encouragingly high coverage, conflation by identity of phonetic form is in many cases too strict a criterion for lemma-based canonicalization – many word pairs which intuitively should be considered instances of the same lemma are assigned to distinct phonetic equivalence classes. Examples of such desired conflations undiscovered by the phonetic conflation strategy include the pairs *(abbrechen, abprechen)*, *(geschickt, geschicket)*, *(gut, guot)*, *(Licht, liecht)*, *(Teufel, tiuvel)*, *(umgehen, umbgehn)*, *(voll, vol)*, and *(wollen, wolln)*. In an attempt to address these shortcomings of the phonetic conflation method, additional conflation heuristics were developed which make use of the dictionary structure of the TAXI/Grimm corpus in order to estimate and maximize a *lemma instantiation likelihood* function.

1.3.1 Implementation

The TAXI/Grimm corpus is comprised of *verse quotation evidence* drawn from a dictionary corpus (Bartz et al., 2004). It is plausible to assume that each of the quotations occurring in an article for a particular dictionary lemma contain some variant of that lemma – otherwise there would not be much sense including the quotation as "evidence" for the lemma in question.

Working from this assumption that each quotation contains at least one variant of the dictionary lemma for which that quotation appears as evidence, a lemma instantiation conflation heuristic has been developed which does not require strict identity of phonetic forms – instead, *string edit distance*

(Levenshtein, 1966; Wagner and Fischer, 1974; Navarro, 2001) on phonetic forms is used to estimate similarity between each word in the corpus and each of the dictionary lemmata under which it occurs. Further, inspired by previous work in unsupervised approximation of semantics and morphology (Church and Hanks, 1990; Yarowsky and Wicentowski, 2000; Baroni et al., 2002), *pointwise mutual information* (McGill, 1955; Cover and Thomas, 1991; Manning and Schütze, 1999) between dictionary lemmata and their candidate instances is employed to detect and filter out "chance" similarities between rare lemmata and high-frequency words.

Formally, the lemma instantiation heuristics attempt to determine for each quotation q which phonetic type i occurring in q best instantiates the dictionary lemma ℓ associated with the article containing q. For $\mathcal{W}$ the set of all word types occurring in the corpus, $\mathcal{L} \subseteq \mathcal{W}$ the set of all dictionary lemmata, and $\mathcal{Q} \subseteq \wp(\mathcal{W}^*)$ the set of all quotations:

$$\begin{aligned} \text{bestInstance}(\cdot) &: \mathcal{Q} \to \mathcal{W} \\ &: q \mapsto \arg\max_{w \in q} \mathrm{L}(M_{\text{pho}}(w), M_{\text{pho}}(\text{lemma}(q))) \end{aligned} \tag{1.8}$$

where the probabilities $\mathrm{P}(\ell, i), \mathrm{P}(\ell)$, and $\mathrm{P}(i)$ used to compute pointwise mutual information are first instantiated by their maximum likelihood estimates over the entire corpus:

$$\mathrm{P}(\ell, i) = \frac{\sum_{w_i \in M^{-1}_{\text{pho}}(i)} \sum_{w_\ell \in M^{-1}_{\text{pho}}(\ell)} f(Token = w_i, Lemma = w_\ell)}{|Corpus|} \tag{1.9}$$

$$\mathrm{P}(\ell) = \sum_i \mathrm{P}(\ell, i) \tag{1.10}$$

$$\mathrm{P}(i) = \sum_\ell \mathrm{P}(\ell, i) \tag{1.11}$$

Raw bit-length pointwise mutual information values $\tilde{\mathrm{I}}(\ell, i)$ are computed and normalized to the unit interval $[0, 1]$ for each lemma and candidate instance, defining $\tilde{\mathrm{I}}(i|\ell)$ and $\tilde{\mathrm{I}}(\ell|i)$ respectively:

$$\tilde{\mathrm{I}}(\ell, i) = \log_2 \frac{\mathrm{P}(\ell, i)}{\mathrm{P}(\ell)\,\mathrm{P}(i)} \tag{1.12}$$

$$\tilde{\mathrm{I}}(i|\ell) = \frac{\tilde{\mathrm{I}}(\ell, i) - \min \tilde{\mathrm{I}}(\ell, \mathcal{W})}{\max \tilde{\mathrm{I}}(\ell, \mathcal{W}) - \min \tilde{\mathrm{I}}(\ell, \mathcal{W})} \tag{1.13}$$

$$\tilde{\mathrm{I}}(\ell|i) = \frac{\tilde{\mathrm{I}}(\ell, i) - \min \tilde{\mathrm{I}}(\mathcal{L}, i)}{\max \tilde{\mathrm{I}}(\mathcal{L}, i) - \min \tilde{\mathrm{I}}(\mathcal{L}, i)} \tag{1.14}$$

The user-specified function $d_{\max}(\ell, i)$ serves a dual purpose: first as a normalization factor for the fuzzy phonetic similarity estimate $\mathrm{sim}(\ell, i)$, and second

as a cutoff threshold for absolute phonetic edit distances $\mathrm{d}_{\mathrm{edit}}(\ell, i)$, blocking instantiation hypotheses when phonetic dissimilarity grows "too large":

$$d_{\max}(\ell, i) \quad = \quad \min\{|\ell|, |i|\} - 1 \tag{1.15}$$

The lemma instantiation likelihood function $\mathrm{L}(i, \ell)$ is defined as the product of the normalized phonetic similarity and the arithmetic average component-normalized mutual information score:

$$\mathrm{sim}(\ell, i) \quad = \quad \begin{cases} \frac{d_{\max}(\ell,i) - \mathrm{d}_{\mathrm{edit}}(\ell,i)}{d_{\max}(\ell,i)} & \text{if } \mathrm{d}_{\mathrm{edit}}(\ell, i) \leq d_{\max}(\ell, i) \\ 0 & \text{otherwise} \end{cases} \tag{1.16}$$

$$\mathrm{L}(i, \ell) \quad = \quad \frac{\mathrm{sim}(\ell, i) \times (\tilde{\mathrm{I}}(\ell|i) + \tilde{\mathrm{I}}(i|\ell))}{2} \tag{1.17}$$

Finally, the edit-distance lemma instantiation heuristic conflates those word pairs which either share a phonetic form or appear as best instances of some common dictionary lemma:[6]

$$\begin{aligned} w \sim_{\mathrm{li}} v \quad :\Leftrightarrow \quad & (w \sim_{\mathrm{pho}} v) \text{ or} \\ & (\mathrm{lemma}(\mathrm{bestInstance}^{-1}(w)) \cap \mathrm{lemma}(\mathrm{bestInstance}^{-1}(v)) \neq \emptyset) \end{aligned} \tag{1.18}$$

1.3.2 Performance

A major advantage of this approach arises from the relatively small number of edit distance comparisons which must be performed. Since the Wagner-Fischer algorithm (Wagner and Fischer, 1974) used to compute phonetic edit distances has quadratic running time, $\mathcal{O}(\mathrm{d}_{\mathrm{edit}}(w, v)) = \mathcal{O}(|w||v|)$, the number of edit distance comparisons comprises the bulk of the heuristic's running time, and should be kept as small as possible. Restricting the comparisons to those pairs (ℓ, i) of dictionary lemmata and phonetic types occurring in quotation evidence for those lemmata requires that approximately 3.38 million comparisons be made during analysis of the entire TAXI/Grimm quotation corpus. If instead every possible unordered pair of phonetic types were to be compared – as required by some morphology induction techniques – a total of *circa* 340 billion comparisons would be required, over ten thousand times as many! With restriction of comparisons to dictionary lemmata, the heuristic analysis completes in 28 minutes on a 1.8GHz dual-core processor workstation, which corresponds to a projected running time of about 5.35 years for a method comparing all unordered word pairs, which is clearly unacceptable.

[6]Note that $\sim_{\mathrm{li}}$ is not an equivalence relation in the strict sense, since although it is reflexive and symmetric, it is not transitive.

1.3.3 Coverage

Using the verse quotation evidence corpus described above in section 1.2.3, the lemma instantiation conflation heuristics discovered conflations with extant forms known to the TAGH morphology for 29,248 additional word types not discovered by phonetic conflation, including all of the example word pairs given in the introduction to this section. Additionally, 9,415 word types were identified as "best instances" for DWB lemmata unknown to the TAGH morphology. Together with phonetic conflation, the lemma instantiation heuristics achieve a total coverage of 212,540 types (66.76%), representing 5,185,858 tokens (94.43%). Thus, the lemma instantiation heuristic conflation method provides a reduction of 26.76% in type-wise coverage errors and of 33.88% in token-wise coverage errors with respect to the phonetic identity conflation method alone, resulting in a total reduction of 42.26% in type-wise coverage errors and of 65.80% in token-wise coverage errors with respect to the literal TAGH morphology.

1.4 Summary & Outlook

Two strategies were presented for discovering synchronically active canonical forms for unknown historical text forms. Together, the two methods achieve TAGH morphological analyses for 94.43% of tokens, reducing the number of unknown tokens by 65.8% in a corpus of *circa* 5.5 million words of historical German verse. In the interest of generalizing these strategies to arbitrary input texts, a robust system for lazy online best-path lookup operations in weighted finite-state transducer cascades (such as phonetic equivalence classes or best-alignments with a target language in the form of a finite-state acceptor) is currently under development.

While the high coverage rate of the conflation strategies presented here is encouraging, a number of important questions remain. Chief among these is the question of the canonicalization strategies' reliability: how many of the discovered extant "canonical" forms are in fact morphologically related to the source forms? Conversely, were all valid canonical forms for each covered source word indeed found, or were some missed? A small gold standard test corpus is currently under construction which should enable quantitative answers to these questions in terms of the information retrieval notions of *precision* and *recall*.

ORIGINALLY APPEARED AS:
Bryan Jurish. Efficient online *k*-best lookup in weighted finite-state cascades. In T. Hanneforth and G. Fanselow, editors, *Language and Logos: Studies in Theoretical and Computational Linguistics*, pages 313-327. Akademie Verlag, 2010a.

Chapter 2

Efficient Online *k*-best Lookup

2.1 Introduction

Weighted finite-state transducers (WFSTs) have proved to be powerful and efficient aids for a variety of natural-language processing tasks, including automatic phonetization and phonological rule systems (Kaplan and Kay, 1994; Laporte, 1997), morphological analysis (Geyken and Hanneforth, 2006), and shallow syntactic parsing (Roche, 1997). In particular, *cascades* arising from the composition of two or more WFSTs can be used to model processing "pipelines", each component of which is itself a (weighted) finite-state transducer.[1] Typically, the input to such a pipeline is a simple string, corresponding to a *lookup* operation for the input string in the processing cascade.

Unfortunately, an exhaustive "offline" compilation of the processing cascade turns out in many cases to be infeasible, due to memory restrictions and the combinatorial properties of the composition operation itself. Even for simple lookup operations in "dense" cascades,[2] the resulting WFST may in fact be several times larger than the processing pipeline itself. In many such cases – particularly in optimization and error-correction problems – the output WFST itself serves only as an intermediate processing datum, however: we are not interested in an exhaustive representation of the lookup output, but rather only in a small finite subset of its language, such as the *k-best paths*.

This paper presents a novel algorithm for efficient k-best search in a

[1] EDIT: cf. Pereira and Riley (1997)

[2] Informally, the "density" of a cascade C corresponds to the cardinality of the underlying rational relation $|[\![C]\!]| \leq |\mathcal{A}^* \times \mathcal{B}^*|$; the densest cascades containing at least one valid path for every pair of in- and output-strings $(s,t) \in \mathcal{A}^* \times \mathcal{B}^*$.

subclass of weighted finite-state lookup cascades which avoids the combinatorial explosion associated with "dense" cascade relations by means of *online computation*:[3] dynamic construction of only those states and arcs required for a k-best search of the lookup output. Use of a *greedy termination* clause together with an additional *cutoff parameter* helps to ensure speedy completion and simultaneously prune unwanted results from the output.

2.1.1 Example Application

As an example application, consider the task of *orthographic standardization* of historical text, which must precede any adequate treatment of historical corpora by conventional NLP tools, due to the lack of consistent orthographic conventions in such corpora (Jurish, 2008). In this scenario, the processing cascade consists of at least:

- a weighted *edit transducer* M_Δ which robustly models (potential) diachronic change likelihood as a (dense) weighted rational relation, and
- a *target acceptor* A_L representing the synchronically active lexicon of extant word forms.

The processing cascade $C_{\Delta L} = M_\Delta \circ A_L$ thus models all potential diachronic changes resulting in some extant word form.[4] A lookup cascade $C_{\vec{w}\Delta L} = (\mathrm{Id}\,(\vec{w}) \circ C_{\Delta L}) = (\mathrm{Id}\,(\vec{w}) \circ M_\Delta \circ A_L)$ for a historical text form $\vec{w}$ in the cascade $C_{\Delta L}$ represents the set of all extant forms $\vec{v}$ into which $\vec{w}$ may have evolved, weighted by the likelihood of a direct etymological relation $\vec{w} \rightsquigarrow \vec{v}$. The k-best output strings of the lookup cascade are then simply the k extant word forms considered most likely to be directly related to the historical form $\vec{w}$. Restricting the admissible output paths by applying an external cutoff threshold $c_{\max}$ is equivalent to imposing an *a priori* upper bound on the likelihood of acceptable diachronic derivations, which is especially important in the case of a dense editor M_Δ.

[3] Also sometimes referred to as "lazy evaluation" or "on-the-fly computation".

[4] EDIT: The Levenshtein and heuristic rewrite canonicalization cascades used in chapters 3 and 4 are instances of just this sort of processing cascade. The Levenshtein cascade is defined as $C_{\mathrm{Lev}} = M_{\mathrm{Lev}} \circ A_{\mathrm{Lex}_{\mathrm{Lev}}}$ and the heuristic rewrite cascade as $C_{\mathrm{rw}} = M_{\mathrm{rw}} \circ A_{\mathrm{Lex}_{\mathrm{rw}}}$, where M_{Lev} is a WFST representation of the Levenshtein string edit distance (Schulz and Mihov, 2002), M_{rw} is the rewrite editor transducer formally specified in appendix A.5, and $A_{\mathrm{Lex}_{\mathrm{Lev}}}, A_{\mathrm{Lex}_{\mathrm{rw}}}$ are weighted acceptors representing the contemporary lexicon extracted from the TAGH morphology as described in appendix A.3.

2.1.2 Desiderata

In light of the preceding example, a number of important properties for a candidate solution may be identified:

- **Online computation:** states and transitions of intermediate processing stages should be computed "on-the-fly" and discarded when no longer needed, to avoid the combinatorial explosion associated with dense cascades.
- **Type-wise input:** the algorithm should function efficiently for type-wise input, for maximal flexibility.
- ***k*-best strings:** output of the algorithm should be an enumeration of the k best strings of the lookup output for a user-specified natural number k, thus allowing the user some control over the maximum degree of ambiguity returned.
- **Arbitrary cascade depth:** the algorithm should not itself impose any upper bound on the depth of the processing cascade. In particular, pair-wise "lazy evaluation" of sub-cascades is to be avoided, since such methods – although elegant and formally correct – tend to introduce nontrivial amounts of runtime and memory overhead.[5]
- **Arbitrary regular weighting function:** the algorithm should function correctly for arbitrary regular weighting functions, i.e. for arbitrary cascades of weighted finite-state transducers. In particular, no assumption should be made about the cascade architecture regarding the presence, placement, content, or disposition of an "editor WFST" such as M_Δ.[6]
- **Cutoff threshold:** the algorithm should accept as an additional parameter a cutoff threshold which serves to further restrict the set of acceptable output paths.
- **Greedy termination:** the algorithm should terminate and return as quickly as possible in the average case; i.e. as soon as the k best paths

[5]Preliminary tests with the OpenFst library (Allauzen et al., 2007) supported these intuitions.

[6]This desideratum is considered a critical feature of any candidate solution, eliminating specialized techniques such as those described in Oflazer and Güzey (1994); Oflazer (1996), relying as these do on an implicit *edit distance* weighting function in the style of Levenshtein (1966); Wagner and Fischer (1974), rather than the weighting function arising from an arbitrary WFST cascade.

have been discovered, or it has been determined that no further paths are to be found below the cutoff threshold.

2.2 Formal Background

Definition 2.1 (Semiring)**.** *A structure* $\mathcal{K} = \langle \mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1} \rangle$ *is a semiring if*

1. $\langle \mathbb{K}, \oplus, \overline{0} \rangle$ *is a commutative monoid with* $\overline{0}$ *as the identity element for* $\oplus$*,*
2. $\langle \mathbb{K}, \otimes, \overline{1} \rangle$ *is a monoid with* $\overline{1}$ *as the identity element for* $\otimes$*,*
3. $\otimes$ *distributes over* $\oplus$*, and*
4. $\overline{0}$ *is an annihilator for* $\otimes$*:* $\forall a \in \mathbb{K}, a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$*.*

$\mathcal{K}$ is *commutative* if $\forall a, b \in \mathbb{K}, a \otimes b = b \otimes a$, and $\mathcal{K}$ is *idempotent* if $\forall a \in \mathbb{K}, a \oplus a = a$. For an idempotent semiring $\mathcal{K}$, the *natural order* over $\mathbb{K}$ is the partial order $\leq_{\mathbb{K}}$ defined by $\forall a, b \in \mathbb{K}, (a \leq_{\mathbb{K}} b) :\Leftrightarrow ((a \oplus b) = a)$. The natural order is both *negative* (i.e. $\overline{1} \leq_{\mathbb{K}} \overline{0}$) and *monotonic*, $\forall a, b, c \in \mathbb{K}, (a \leq_{\mathbb{K}} b)$ implies $(a \oplus c) \leq_{\mathbb{K}} (b \oplus c)$, $(a \otimes c) \leq_{\mathbb{K}} (b \otimes c)$, and $(c \otimes a) \leq_{\mathbb{K}} (c \otimes b)$. $\mathcal{K}$ is said to be *bounded* if $\overline{1}$ is an annihilator for $\oplus$: $\forall a \in \mathbb{K}, \overline{1} \oplus a = \overline{1}$. In a bounded semiring, $\overline{1} \leq_{\mathbb{K}} a \leq_{\mathbb{K}} \overline{0}$ for all $a \in \mathbb{K}$. Every bounded semiring is also idempotent (Mohri, 2002, Lemma 3). For current purposes, I will restrict my attention to bounded semirings.

Definition 2.2 (WFST)**.** *A weighted finite-state transducer over a semiring* $\mathcal{K}$ *is a 6-tuple* $M = \langle \mathcal{A}, \mathcal{B}, Q, q_0, F, E \rangle$ *with:*[7]

1. $\mathcal{A}$ *a finite input (or "lower") alphabet,*
2. $\mathcal{B}$ *a finite output (or "upper") alphabet,*
3. Q *a finite set of states,*
4. $q_0 \in Q$ *the designated* initial state*,*
5. $F \subseteq Q$ *the set of final states, and*
6. $E \subseteq Q \times Q \times (\mathcal{A} \cup \{\varepsilon\}) \times (\mathcal{B} \cup \{\varepsilon\}) \times \mathbb{K}$*, a finite set of transitions.*

[7] WFSTs are sometimes defined with an additional *final weight* function $\rho : Q \to \mathbb{K}$, and/or a non-deterministic *initial weight function* $\alpha : Q \to \mathbb{K}$ in place of q_0. I ignore these extensions here in the interest of clarity.

For a transition $e = (q_1, q_2, a, b, c) \in E$, I denote by $p[e]$ its source state q_1, by $n[e]$ its destination state q_2, by $i[e]$ its input label a, by $o[e]$ its output label b, and by $c[e]$ its weight (or "cost") c. A weighted finite-state acceptor (WFSA) can be regarded as a WFST with $\mathcal{A} = \mathcal{B}$ and $i[e] = o[e]$ for all $e \in E$.

Definition 2.3 (String Transducer)**.** *For a string $\vec{w} = w_1 \cdots w_n \in \mathcal{A}^*$ over an alphabet $\mathcal{A}$, the* string transducer *for $\vec{w}$ is the WFSA* $\mathrm{Id}(\vec{w}) = \langle \mathcal{A}, \mathcal{A}, Q_{\vec{w}}, 0, \{n\}, E_{\vec{w}} \rangle$ *with* $Q_{\vec{w}} = \{i \in \mathbb{N} : i \leq n\}$ *and* $E_{\vec{w}} = \bigcup_{i=1}^{n} \{(i-1, i, w_i, w_i, \bar{1})\}$.

A path π is a finite sequence $e_1 e_2 \ldots e_{|\pi|}$ of $|\pi|$ transitions such that $n[e_i] = p[e_{i+1}]$ for $1 \leq i < |\pi|$. Extending the notation for transitions, I define the source and sink states of a path as $p[\pi] = p[e_1]$ and $n[\pi] = n[e_{|\pi|}]$, respectively. The input label string $i[\pi]$ yielded by a path π is the concatenation of the input labels of its transitions: $i[\pi] = i[e_1]i[e_2] \ldots i[e_{|\pi|}]$; the output label string $o[\pi]$ is defined analogously. The weight $c[\pi]$ of a path π is the $\otimes$-product of its transitions: $w[\pi] = \bigotimes_{i=1}^{|\pi|} c[e_i]$.

If $p[\pi] = q_0$ and $n[\pi] \in F$, π is called *successful*. A *cycle* is a path π with $p[\pi] = n[\pi]$. A string transducer $\mathrm{Id}(\vec{w})$ contains exactly one successful path $\pi_{\vec{w}}$ with $i[\pi_{\vec{w}}] = o[\pi_{\vec{w}}] = \vec{w}$. A state $r \in Q$ is said to be *accessible from* a state $q \in Q$ if there exists a path π with $p[\pi] = q$ and $n[\pi] = r$; r is *accessible* if it is accessible from q_0. For $q \in Q$, $\vec{w} \in \mathcal{A}^*$, $\vec{v} \in \mathcal{B}^*$, and $R \subseteq Q$, $\Pi(q, \vec{w}, \vec{v}, R)$ denotes the set of paths from q to some $r \in R$ with input string $\vec{w}$ and output string $\vec{v}$, and $\Pi(q, R) = \bigcup_{\vec{w} \in \mathcal{A}^*, \vec{v} \in \mathcal{B}^*} \Pi(q, \vec{w}, \vec{v}, R)$ denotes the set of paths originating at q and ending at some $r \in R$.

Definition 2.4 (Transducer Weight)**.** *The weight assigned by a WFST M to a pair of strings $(\vec{w}, \vec{v}) \in \mathcal{A}^* \times \mathcal{B}^*$ is defined as*

$$[\![M]\!](\vec{w}, \vec{v}) = \bigoplus_{\pi \in \Pi(q_0, \vec{w}, \vec{v}, F)} c[\pi]$$

Definition 2.5 (Composition of WFSTs)**.** *Given WFSTs $M_1 = \langle \mathcal{A}, \mathcal{B}, Q_1, q_{0_1}, F_1, E_1 \rangle$ and $M_2 = \langle \mathcal{B}, \mathcal{C}, Q_2, q_{0_2}, F_2, E_2 \rangle$ over a commutative and complete*[8] *semiring $\mathcal{K}$, the composition of M_1 and M_2 is written $M_1 \circ M_2$, and is itself a WFST such that for all $\vec{w} \in \mathcal{A}^*$, $\vec{v} \in \mathcal{C}^*$:*

$$[\![M_1 \circ M_2]\!](\vec{w}, \vec{v}) = \bigoplus_{\vec{u} \in \mathcal{B}^*} [\![M_1]\!](\vec{w}, \vec{u}) \otimes [\![M_2]\!](\vec{u}, \vec{v})$$

Further, $M_3 = \langle \mathcal{A}, \mathcal{C}, (Q_1 \times Q_2), E_3, (q_{0_1}, q_{0_2}), (F_1 \times F_2) \rangle$ is such a WFST,

[8] cf. Ésik and Kuich (2004)

$[\![M_3]\!] = [\![M_1 \circ M_2]\!]$, *where:*[9]

$$\begin{aligned} \widetilde{Q} &= \{(q, q, \varepsilon, \varepsilon, \bar{1}) : q \in Q\} \\ E_3 &= \bigcup_{\substack{(q_1,r_1,a_1,a_2,c_1)\in E_1\cup\widetilde{Q_1} \\ (q_2,r_2,a_2,a_3,c_2)\in E_2\cup\widetilde{Q_2}}} \{((q_1, q_2), (r_1, r_2), a_1, a_3, c_1 \otimes c_2)\} \end{aligned}$$

2.3 Algorithms

This section develops an algorithm for discovering the k-best label paths in a dense cascade of weighted finite state transducers over a bounded semiring, attempting to fulfill the desiderata from section 2.1.2. I begin with a brief review of the well-known algorithm which serves as the basis for the current approach, and consider its generalization to abstract semiring weights in section 2.3.1. Section 2.3.2 extends the algorithm to online lookup operations in weighted finite-state cascades. Section 2.3.3 explores the implications of a greedy k-best termination strategy, and section 2.3.4 extends the discussion to include a user-specified cutoff threshold. Finally, section 2.3.5 addresses the problem of extending the algorithm to return output label strings.

The current approach is best understood as a variant of the well-known *Dijkstra Algorithm* (Dijkstra, 1959; Cormen et al., 2001), presented here as Algorithm 2.1. Using a Fibonacci heap (Fredman and Tarjan, 1987) to implement the processing queue S, and assuming constant time access to outgoing edges for a vertex, Algorithm 2.1 has running time $\mathcal{O}(\textsc{Dijkstra}) = \mathcal{O}\left(|E| + |V| \log |V|\right)$.

2.3.1 Semiring Weights

In its original form, Dijkstra's algorithm assumes a graph $G = \langle V, E \rangle$ with non-negative real-valued weighted edges $E \subseteq (V \times V \times \mathbb{R}_+)$, ordered by the natural linear order $<$, thus implicitly equating "best" with "$<$-minimal". The first adaptation to be undertaken is a straightforward generalization of the Dijkstra algorithm to an abstract semiring $\mathcal{K} = \langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1} \rangle$, using minimality with respect to a partial order $\leq_{\mathbb{K}}$ to define "best" weights.

First, the initialization of the best-distance vector $d[\cdot]$ must be adapted to use the relevant semiring constants $\bar{0}$ and $\bar{1}$:

2: $d[\cdot] := \{v \mapsto \bar{0} : v \in V\}$ /* Initialize $d[\cdot] : V \to \mathbb{K}$ */
3: $d[v_0] := \bar{1}$

[9] The construction given here for E_3 is only valid for idempotent semirings; cf. Mohri et al. (1996) for a generalization to non-idempotent semirings.

Algorithm 2.1 Dijkstra (1959)

```
 1: function DIJKSTRA(V, E, v_0)
 2:     d[·] := {v ↦ ∞ : v ∈ V}                          /* Initialize */
 3:     d[v_0] := 0
 4:     S := V
 5:     while S ≠ ∅                                      /* Main loop */
 6:         u := arg min_{u∈S} d[u]                      /* Best-first search */
 7:         S := S\{u}
 8:         foreach e ∈ E : p[e] = u                     /* Expand */
 9:             d' := d[u] + c[e]                        /* Accumulate */
10:             if d' < d[v] then                        /* Relax */
11:                 d[v] := d'
12:     return d[·]
```

Next, the best-first order of extraction from the vertex-queue S must be adapted to use the partial order $\leq_{\mathbb{K}}$:

```
6: u := arg min_{u∈S,<_K} d[u]        /* Best-first search using <_K */
```

Finally, the RELAX step is adapted to use the semiring multiplication operation $\otimes$ for accumulating the characteristic weight of a path, as well as the semiring order for the "better-path" check of line 10:

```
 9: d' := d[u] ⊗ c[e]                          /* Accumulate using ⊗ */
10: if d' <_K d[v] then                        /* Relax */
11:     d[v] := d'
```

Dijkstra's original algorithm emerges as an instance of the generalized algorithm using the non-negative tropical semiring $\langle \mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0 \rangle$ (Simon, 1987). Important to note is that the generalization to abstract semirings has implications for the correctness of the algorithm. In particular, graph cycles with a net weight $c <_{\mathbb{K}} \bar{1}$ will cause the algorithm never to terminate at all, inducing an infinite series of $<_{\mathbb{K}}$-decreasing weights $d[u]$ for the cycle root vertex u, leading to an infinite loop of RELAX steps. Further, the relaxability check of line 10 is not meaningful for all partial orders $\leq_{\mathbb{K}}$: a non-monotonic order may cause a partial path to be disregarded here which would lead to a better path for some subsequent vertex. I therefore restrict my attention for current purposes to *bounded* (idempotent) semirings using the monotonic natural semiring order, $\forall a, b \in \mathbb{K}$:

$$\begin{array}{rcll} (\bar{1} \oplus a) & = & \bar{1} & \text{(Boundedness)} \\ (a \oplus a) & = & a & \text{(Idempotence)} \\ (a \leq_{\mathbb{K}} b) & \Leftrightarrow & ((a \oplus b) = a) & \text{(Natural Order)} \end{array}$$

2.3.2 Online Cascade Lookup

The next task is to extend the algorithm to operate on lookup cascades $C = (\mathrm{Id}\,(\vec{w}) \circ M_2 \circ \cdots \circ M_{|C|})$ for $\vec{w} \in \mathcal{A}_2^*$. Suppose the standard construction for composition of WFSTs given in Definition 2.5 yields for C the WFST $M = \langle \mathcal{A}, \mathcal{B}, Q, q_0, F, E \rangle$. Clearly, $\langle Q, E \rangle$ can be treated as an edge-labelled weighted graph. By assumption however, M is too large to be computed offline, so that in particular the composition of transitions E must be performed at runtime. The resulting algorithm is presented here together with some auxiliary subroutines as Algorithm 2.2.

Algorithm 2.2 Dijkstra's algorithm for online lookup cascades

```
1:  function DIJKSTRA-CASCADE(w, C)
2:      S := {q_0}                                              /* Initialize */
3:      d[·] := {q_0 ↦ 1̄}
4:      while S ≠ ∅                                             /* Main loop */
5:          q := arg min_{q∈S,<_K} d[q]                         /* Best-first search */
6:          S := S\{q}
7:          foreach e ∈ ARCS(w, C, q)                           /* Expand outgoing arcs */
8:              d' := d[q] ⊗ c[e]                               /* Accumulate */
9:              if d' <_K COST(d[·], n[e]) then                 /* Relax */
10:                 d[n[e]] := d'
11:                 S := S ∪ {n[e]}                             /* Enqueue */
12:     return d[·]
13: function ARCS(w⃗, C, q)
14:     return EXPAND-ARCS(C, q, 1, w⃗[q[1]]) ∪ EXPAND-ARCS(C, q, 1, ε)
15: function EXPAND-ARCS(C, q, i, a)
16:     A := ∅
17:     foreach e ∈ E_i ∪ {(q[i], q[i], ε, ε, 1̄)} : p[e] = q[i] & i[e] = a
18:         if i = |C| then
19:             A := A ∪ {e}
20:         else
21:             foreach e' ∈ EXPAND-ARCS(C, q, i + 1, o[e])
22:                 A := A ∪ {(⟨p[e], p[e']⟩, ⟨n[e], n[e']⟩, i[e], o[e'], c[e] ⊗ c[e'])}
23:     return A
24: function COST(d[·], q)
25:     if d[q] defined then return d[q]
26:     return 0̄
```

The online expansion of outgoing transitions from a state $q = \langle q_{\vec{w}}, q_2, \ldots,$

$q_{|C|}\rangle \in Q$ is performed by the auxiliary function ARCS given in Algorithm 2.2.[10] ARCS is implemented as a pair of calls to the function EXPAND-ARCS, which recursively descends the cascade, linking together transitions from adjacent components with matching out- rsp. input labels in accordance with Definition 2.5.

The only other change made to the core algorithm DIJKSTRA-CASCADE is a move to sparse administrative structures: rather than initialize the queue S with the set of all cascade states Q, which would entail explicitly representing such states and thus pre-compiling them, Algorithm 2.2 instead uses a dynamic queue S which at any given point in the computation holds only those states which need to be (re-)investigated. Similarly, the map $d[\cdot]$ of best weights is implemented as a sparse partial map, and the default case $d[q] = \bar{0}$ is handled by the auxiliary function COST. For Algorithm 2.2, the use of sparse structures has few consequences – states unreachable from q_0 will no longer be processed, but the algorithm otherwise proceeds exactly as in Algorithm 2.1, with running time growing by a factor of the cascade depth $|C|$ to allow for online expansion of transitions. Since cascade depth is expected to be a small constant, I ignore it in the sequel.

2.3.3 *k*-Best Final States

Dijkstra's algorithm solves the *single source shortest distances* problem, returning a map $d : Q \to \mathbb{K}$ which associates each state with the best net weight of any path to that state from the designated initial state q_0. In the current problem context, we are not interested in an exhaustive enumeration $d[\cdot]$ of net weights for all cascade states, but rather only for the *final states* of the lookup output: $d_F : F \to \mathbb{K}$. Even more specifically, we are interested only in the k-best mappings for some final state, a partial function $d_{F,k} : F \xrightarrow{\text{partial}} \mathbb{K}$ such that the following hold:

$$\begin{array}{rcl} d_{F,k} & \subseteq & d_F \subseteq d \\ |d_{F,k}| & \leq & k \\ \forall q, r \in F \,.\, d[q] <_{\mathbb{K}} d[r] \;\&\; r \in \mathrm{dom}(d_{F,k}) & \Rightarrow & q \in \mathrm{dom}(d_{F,k}) \end{array}$$

Clearly, $d_F = (d \upharpoonright F)$ is simply the restriction of the map $d[\cdot]$ to the subset F of final states, and $d_{F,k}$ can be generated from d_F by extraction

[10] The function ARCS takes advantage of the facts that in a lookup cascade, the initial component $\mathrm{Id}\,(\vec{w})$ has at most one outgoing (non-ε) arc e, and that $i[e] = o[e] = \vec{w}[q[1]]$. A generalization to arbitrary WFST cascades would involve iterating over all outgoing arcs of the initial cascade component here.

of the $j \leq k$ $<_{\mathbb{K}}$-minimal elements. Not only does such an extraction add additional runtime complexity,[11] it requires that Algorithm 2.2 first run in its entirety, which as noted above is unacceptable for dense cascades. Instead, the algorithm can be optimized for the task of discovering $d_{F,k}[\cdot]$ as given in Algorithm 2.3.[12]

Algorithm 2.3 Dijkstra's algorithm adapted for k-best net weights to final states

```
1:  function DIJKSTRA-KBEST(w, C, k)
2:      S := {q_0}                                        /* Initialize */
3:      d[·] := {q_0 ↦ 1̄}
4:      d_{F,k}[·] := ∅
5:      while S ≠ ∅                                       /* Main loop */
6:          q := arg min_{q∈S,<_K} d[q]                   /* Best-first search */
7:          S := S\{q}
8:          if q ∈ F then                                 /* Finality check */
9:              d_{F,k}[q] := d[q]
10:             if |d_{F,k}| = k then break               /* Greedy termination */
11:         foreach e ∈ ARCS(w, C, q)                     /* Expand outgoing arcs */
12:             d' := d[q] ⊗ c[e]                         /* Accumulate */
13:             if d' <_K COST(d[·], n[e]) then           /* Relax */
14:                 d[n[e]] := d'
15:                 S := S ∪ {n[e]}                       /* Enqueue */
16:     return d_{F,k}[·]
```

The best-first queue management and relaxation strategy of Algorithms 2.1 and 2.2 remains unchanged in the function DIJKSTRA-KBEST of Algorithm 2.3, thus DIJKSTRA-KBEST terminates whenever DIJKSTRA does, and the correctness conditions are unaffected – termination is guaranteed for bounded semirings. The additional statements in lines 8-10 can all be implemented as (amortized) constant-time operations, so the worst-case running time also remains unchanged: $\mathcal{O}(\text{DIJKSTRA-KBEST}) = \mathcal{O}(\text{DIJKSTRA}) = \mathcal{O}(|E| + |Q| \log |Q|)$. Memory use grows in the worst case by at most $\mathcal{O}(|F|)$ for storage of the partial output map $d_{F,k}[\cdot]$.

More important for the current problem context are the average case time and space complexity for DIJKSTRA-KBEST versus the original DIJKSTRA-

[11] $\mathcal{O}(|F|k)$, using a standard implementation for small k.

[12] As originally presented by Dijkstra (1959), the algorithm includes a single designated sink vertex parameter as well as a greedy termination clause, which is extended here to a set of designated final states.

CASCADE function. Whereas DIJKSTRA-CASCADE must always compute the net weight of at least one complete path to each state, DIJKSTRA-KBEST need only compute weights for at most k paths ending in final states. That the first such weights computed are indeed the k best weights sought follows from the correctness of the best-first search order, which in turn follows from the boundedness of the semiring $\mathcal{K}$. Since immediately upon discovery of the k^{th} best weight to a final state at line 10, Algorithm 2.3 breaks out of the queue-processing loop and returns the partial map $d_{F,k}$, its time complexity can be more precisely specified by (2.1),

$$\mathcal{O}\left(\text{DIJKSTRA-KBEST}\right) = \mathcal{O}\left(|E_{F,k}| + |Q_{F,k}| \log |Q_{F,k}|\right) \tag{2.1}$$

where:

$$\begin{aligned} Q_{F,k} &= \{q \in Q : d[q] \leq_{\mathbb{K}} \max\left(\text{rng}\left(d_{F,k}\right)\right)\} \\ E_{F,k} &= \{e \in E : p[e] \in Q_{F,k}\} \end{aligned}$$

Assuming that k best final weights were indeed found (which will always be the case if $k \leq |F|$), $Q_{F,k}$ is the set of states to which at least one path exists with a net weight less than or equal to some k-best final weight in $d_{F,k}$, and $E_{F,k}$ is the set of all transitions leaving any state in $Q_{F,k}$. By the correctness of the best-first search order for bounded semirings, $Q_{F,k}$ contains all and only those states q which may be extracted from the queue at line 6 before discovery of the k^{th} best net weight to a final state at line 8 and consequent termination at line 10. It follows that $E_{F,k}$ is the set of transitions which must be expanded (and possibly relaxed) by the loop of lines 11-15.

In many interesting cases, $Q_{F,k}$ and $E_{F,k}$ will be much smaller than Q and E respectively, so that the reduced time complexity of Equation (2.1) represents a major improvement over a brute force approach using Algorithm 2.2 directly. Consider for example a simple error-correction cascade similar to that described in section 2.1.1, and let p_c be the average probability over all states $q \in Q_{\vec{w}\Delta L}$ that a path exists from the initial state $q_{0_{\vec{w}\Delta L}}$ to q with net weight $c' \leq_{\mathbb{K}} c$. If $c \in \mathbb{K}$ is the maximum weight to a k-best final state, then the expected size of $Q_{F,k}$ is $\text{E}(|Q_{F,k}|) = \text{E}_{p_c}(\mathbb{1}_{Q_{\vec{w}\Delta L}}) = \sum_{q \in Q_{\vec{w}\Delta L}} p_c = p_c |Q_{\vec{w}\Delta L}|$. The number of states which must be expanded for a k-best search with maximum net path weight c thus depends crucially on p_c, which can be understood as the probability of the existence of a "neighbor" path with edit cost $c' \leq_{\mathbb{K}} c$. It is therefore of paramount importance for purposes of runtime efficiency both (a) to ensure that M_Δ models the phenomena it is intended to represent as accurately as possible, effectively minimizing p_c globally for all $c \in \mathbb{K}$, and (b) to minimize p_c locally by preventing $c = \max(\text{rng}(d_{F,k}))$ from growing too large, since $c \leq c'$ implies $p_c \leq p_{c'}$.

2.3.4 Cutoff Threshold

An unsubtle but effective method for local minimization of the maximum path weight returned by Algorithm 2.3 is the explicit specification of a user-specified cutoff threshold $c_{\max} \in \mathbb{K}$ on path weights as an input parameter. Intuitively, such a parameter represents an *a priori* upper bound on the cost of "acceptable" paths. For the example application from section 2.1.1, a parameter $c_{\max}$ can limit the algorithm's running time even when the input word $\vec{w}$ represents an extinct lexeme not explicitly accounted for by a dense M_Δ, in which case its k nearest neighbors according to $[\![M_\Delta]\!]$ would be randomly distributed in L, and their inclusion as "best" paths for $\vec{w}$ would only introduce noise (both precision and recall errors) into the host application. Implementing the parameter $c_{\max}$ for Algorithm 2.3 requires only the insertion of a simple check after line 7:

if $d[q] >_{\mathbb{K}} c_{\max}$ **then break** /* Cost upper-bound exceeded */

Whenever $c_{\max}$ is exceeded for the minimum-cost state in the queue, it must also be exceeded for every other queued state as well. Since $\leq_{\mathbb{K}}$ is monotonic, queue processing can cease as soon as any any state with a minimum net path weight exceeding $c_{\max}$ is extracted from the queue. Note that while it is possible in the case of the example cascade architecture from section 2.1.1 to incorporate $c_{\max}$ into the edit transducer by modifying M_Δ such that for all $\vec{w} \in \mathcal{A}_\Delta^*, \vec{v} \in \mathcal{B}_\Delta^*$, $\Pi(q_{0_\Delta}, \vec{w}, \vec{v}, F_\Delta) \neq \emptyset$ implies $[\![M_\Delta]\!](\vec{w}, \vec{v}) \leq_{\mathbb{K}} c_{\max}$, such a construction not only introduces additional storage requirements by introducing new states into M_Δ, but is not in general possible if the processing cascade (which by assumption is too large to be computed and stored offline in its entirety) contains multiple independent weighted components. Implementation of the upper bound as a parameter does not increase run time complexity or storage requirements for the algorithm, and allows additional flexibility: the user may for example choose to instantiate $c_{\max}$ as a function of input word length, representing the upper bound in terms of average cost per character rather than a global cost for all words.

2.3.5 Label Strings

Extending the algorithm to return the k-best (output) label strings rather than the k-best net path weights is not as trivial a task as it may at first appear. The traditional method (Cormen et al., 2001) of maintaining a backtrace vector $p[\cdot] : Q \to Q$ mapping states to their best predecessors causes the number of returned paths $|d_{F,k}|$ be bounded above by the number of final states $|F|$, and does not correctly compute the k-best paths if these

are defined to include labels in addition to states. Extending the semiring $\mathcal{K}$ to a k-best semiring $\mathcal{K}_k$ as described by Mohri (2002) not only yields a non-idempotent semiring, but also entails additional modifications for direct storage of path backtraces in the semiring itself.

The current approach instead extends the processing queue S to store state-string pairs $\langle q, s\rangle \in Q \times \mathcal{B}^*$ such that s is the output label string of some path from q_0 to q. The best-weight vector is then re-defined as a (sparse) map $d[\cdot] : Q \times \mathcal{B}^* \to \mathbb{K}$ such that $d[q, s]$ represents the net weight associated with the best path from q_0 to q with output label string s, and the output buffer $d_{F,k}$ is similarly extended to a buffer $d_{\Pi,k}$. An additional kludge[13] parameter $x_{\max} \in \mathbb{N}$ limits the number of allowable queue extractions. The resulting algorithm is presented here as Algorithm 2.4.

Algorithm 2.4 Adaptation of Dijkstra's algorithm for k-best output label strings

```
 1: function DIJKSTRA-STRINGS(w⃗, C, k, c_max, x_max)
 2:     S := {⟨q_0, ε⟩}                                   /* Initialize */
 3:     d[·] := {⟨q_0, ε⟩ ↦ 1̄}
 4:     d_{Π,k}[·] := ∅
 5:     while S ≠ ∅                                       /* Main loop */
 6:         ⟨q, s⟩ := arg min_{⟨q,s⟩∈S,<_K} d[q, s]       /* Best-first search */
 7:         S := S\{⟨q, s⟩}
 8:         if x_max = 0 then break                       /* Too many extractions */
 9:         x_max := x_max − 1
10:         if d[q, s] >_K c_max then break               /* Cost upper-bound exceeded */
11:         if q ∈ F then                                 /* Finality check */
12:             d_{Π,k}[q, s] := d[q, s]
13:             if |d_{Π,k}| = k then break               /* Greedy termination */
14:         foreach e ∈ ARCS(w, C, q)                     /* Expand outgoing arcs */
15:             d′ := d[q, s] ⊗ c[e]                      /* Accumulate */
16:             s′ := s⌢o[e]                              /* Append */
17:             if d′ <_K COST(d[·], ⟨n[e], s′⟩) then     /* Relax */
18:                 d[n[e], s′] := d′
19:                 S := S ∪ {⟨n[e], s′⟩}
20:     return d_{Π,k}[·]
```

Assume for the moment that the kludge parameter is vacuous, e.g. $x_{\max} = -1$. If the cascade contains an instance of a certain type of "degenerate"

[13] cf. Raymond (2010)

cycle, then Algorithm 2.4 may never terminate at all. A degenerate cycle in this sense can be operationally defined as one which may lead to an infinite series of RELAX steps for increasingly long label strings at lines 17 through 19. Formally, I call a cycle π degenerate if (2.2)-(2.5) hold for some $\pi' \in E^*$ and for all $n \in \mathbb{N}$.

$$\begin{aligned} \pi'\pi &\in \Pi(q_0, Q) && (2.2)\\ i[\pi] &= \varepsilon && (2.3)\\ o[\pi] &\neq \varepsilon && (2.4)\\ c\left[\pi^n\right] &\leq_{\mathbb{K}} c_{\max} && (2.5) \end{aligned}$$

Condition (2.2) requires that degenerate cycles are accessible. A non-accessible cycle can never induce an infinite loop, since only accessible states are ever inserted into the queue at line 19. Condition (2.3) states that only paths with an empty input string can degenerate. This follows from the fact that C is a lookup cascade with initial component $\mathrm{Id}\,(\vec{w})$, so that the maximum number of iterations for a cycle with $i[\pi] = s \neq \varepsilon$ is $\frac{|\vec{w}|}{|s|}$. Condition (2.4) states that degenerate cycles must have non-empty output strings, since a cycle with $o[\pi] = \varepsilon$ generates at most one index configuration $\langle q, s\rangle$ for its source state $q = p[\pi]$, and this configuration will fail the relaxability check at line 17 after the first iteration. Finally, condition (2.5) captures the intuition that a degenerate cycle may be iterated arbitrarily many times without its weight exceeding the bound parameter[14] $c_{\max}$, $c[\pi^n] = c[\pi]^n = \bigotimes_{i=1}^{n} c[\pi] \leq_{\mathbb{K}} c_{\max}$, and thus will never be pruned by the check at line 10. In particular, this condition attains for $c[\pi] = \bar{1} <_{\mathbb{K}} c_{\max}$, since $\bar{1}^n = \bar{1}$ for all $n \in \mathbb{N}$. I denote by $\widetilde{\Pi}(C)$ the set of paths for which the weight-independent criteria (2.2)-(2.4) hold:

$$\widetilde{\Pi}(C) = \{\pi \in E^* : \exists \pi' \in E^* : \pi'\pi \in \Pi(q_0, Q)\ \&\ p[\pi] = n[\pi]\ \&\ i[\pi] = \varepsilon \neq o[\pi]\}$$

That Algorithm 2.4 finds the k best label strings in the absence of degenerate cycles whenever at least k distinctly labelled successful paths exist follows from the correctness of Algorithm 2.3. Note in particular that paths with distinct label strings ending in the same state are treated as distinct objects, as are paths with identically labelled strings ending in distinct states, analogous to the trellis construction used in the well-known *Viterbi algorithm* (Viterbi, 1967).

[14] For purposes of defining path degeneracy without a greedy termination clause, the upper bound variable $c_{\max}$ may be defined in terms of the maximum target weight, $c_{\max} = \max(\mathrm{rng}(d_{\Pi,k}))$.

Rather than rely on an expensive cycle check to detect degenerate cycles, I introduce a kludge parameter $x_{\max}$ which places an upper bound on the number of queue extractions performed and limits the running time of Algorithm 2.4 to $\mathcal{O}(x_{\max}(\max(\deg(Q)) + \log x_{\max}))$, where $\max(\deg(Q))$ is the maximum output degree of any state in Q, $\deg(q \in Q) = |\{e \in E : p[e] = q\}|$. Despite its inelegance, this technique can be useful in both development and production environments – in the former to detect and report potential errors in the cascade architecture, and in the latter to place a hard limit on the computational resources consumed.

When $x_{\max}$ is finite but the break at line 8 is not responsible for its termination, Algorithm 2.4 has the running time specified in (2.6),

$$\mathcal{O}\left(\textsc{Dijkstra-Strings}\right) = \mathcal{O}\left(|E_{\Pi,k}| + |V_{\Pi,k}| \log |V_{\Pi,k}|\right) \tag{2.6}$$

where:

$$\begin{aligned} V_{\Pi,k} &= \{\langle q, s\rangle \in Q \times \mathcal{B}^* : d[q,s] \leq_{\mathbb{K}} \max\left(\operatorname{rng}\left(d_{\Pi,k}\right)\right)\} \\ E_{\Pi,k} &= \{\langle\langle q, s\rangle, e\rangle \in V_{\Pi,k} \times E : p[e] = q\} \end{aligned}$$

Since $x_{\max}$ is finite, $V_{\Pi,k}$ and $E_{\Pi,k}$ are as well, since at most finitely many extractions have been performed and each extraction relaxes only finitely many transitions. $V_{\Pi,k}$ and $E_{\Pi,k}$ are further limited by the cutoff threshold $c_{\max} \in \mathbb{K}$ as described in section 2.3.4. In particular, whenever C contains no degenerate cycles, $x_{\max}$ may be set to $kn|Q| \leq kn|\vec{w}| \prod_{i=1}^{|C|} |Q_i|$, where $n = \min\{n \in \mathbb{N} : \forall \pi \in \widetilde{\Pi}(C) : c[\pi^n] >_{\mathbb{K}} c_{\max}\}$ to guarantee both termination and discovery of the $j \leq k$ best paths, although this quantity is considered too large to be of practical use in the dense cascades for which the algorithm was developed, for which the explicit enumeration and storage of Q itself would incur unacceptable computational overhead.

2.4 Summary

I have presented an algorithm for discovery of the k best output label strings for weighted finite state transducer lookup cascades of arbitrary depth which computes cascade structure online and is therefore suitable for use with "dense" cascades which cannot be pre-compiled. The correctness conditions and running time of the algorithm were discussed for both worst- and average-case scenarios, as were the implications of a greedy termination strategy and an external cutoff threshold. Some properties of degenerate cascades were identified, and the subclass of lookup cascades for which the algorithm is expected to terminate was restricted accordingly. The algorithm as presented

here has been implemented in the `gfsmxl` C library,[15] and is being successfully used to implement a robust orthographic standardization cascade for historical German text.

[15] `http://www.ling.uni-potsdam.de/~moocow/projects/gfsm/#gfsmxl`

ORIGINALLY APPEARED AS:
Bryan Jurish. Comparing canonicalizations of historical German text. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology* (SIGMORPHON), pages 72-77, 2010b.

Chapter 3

Comparing Canonicalizations

3.1 Introduction

Historical text presents numerous challenges for contemporary natural language processing techniques. In particular, the absence of consistent orthographic conventions in historical text presents difficulties for any system requiring reference to a fixed lexicon accessed by orthographic form, such as document indexing systems (Sokirko, 2003; Cafarella and Cutting, 2004), part-of-speech taggers (DeRose, 1988; Brill, 1992; Schmid, 1994), simple word stemmers (Lovins, 1968; Porter, 1980), or more sophisticated morphological analyzers (Geyken and Hanneforth, 2006; Clematide, 2008).

When adopting historical text into such a system, one of the most crucial tasks is the association of one or more *extant equivalents* with each word of the input text: synchronically active types which best represent the relevant features of the input word. Which features are considered "relevant" here depends on the application in question: for a lemmatization task only the root lexeme is relevant, whereas syntactic parsing may require additional morphosyntactic features. For current purposes, extant equivalents are to be understood as *canonical cognates*, preserving both the root(s) and morphosyntactic features of the associated historical form(s), which should suffice (modulo major grammatical and/or lexical semantic shifts) for most natural language processing tasks.

In this paper, I present three methods for automatic discovery of extant canonical cognates for historical German text, and evaluate their performance on an information retrieval task over a small gold-standard corpus.

3.2 Canonicalization Methods

In this section, I present three methods for automatic discovery of extant canonical cognates for historical German input: *phonetic conflation* (Pho), *Levenshtein edit distance* (Lev), and a heuristic *rewrite transducer* (rw). The various methods are presented individually below, and characterized in terms of the linguistic resources required for their application. Formally, each canonicalization method R is defined by a characteristic *conflation relation* $\sim_R$, a binary relation on the set $\mathcal{A}^*$ of all strings over the finite grapheme alphabet $\mathcal{A}$. Prototypically, $\sim_R$ will be a true equivalence relation, inducing a partitioning of $\mathcal{A}^*$ into equivalence classes or "conflation sets" $[w]_R = \{v \in \mathcal{A}^* : v \sim_R w\}$.

3.2.1 Phonetic Conflation

If we assume despite the lack of consistent orthographic conventions that historical graphemic forms were constructed to reflect phonetic forms, and if the phonetic system of the target language is diachronically more stable than the graphematic system, then the phonetic form of a word should provide a better clue to its extant cognates (if any) than a historical graphemic form alone. Taken together, these assumptions lead to the canonicalization technique referred to here as *phonetic conflation*.

In order to map graphemic forms to phonetic forms, we may avail ourselves of previous work in the realm of text-to-speech synthesis, a domain in which the discovery of phonetic forms for arbitrary text is an often-studied problem (Allen et al., 1987; Dutoit, 1997), the so-called "letter-to-sound" (LTS) conversion problem. The phonetic conversion module used here was adapted from the LTS rule-set distributed with the IMS German Festival package (Möhler et al., 2001), and compiled as a finite-state transducer (Jurish, 2008).[1]

In general, the phonetic conflation strategy maps each (historical or extant) input word $w \in \mathcal{A}^*$ to a unique phonetic form $\text{pho}(w)$ by means of a computable function $\text{pho} : \mathcal{A}^* \to \mathcal{P}^*$,[2] conflating those strings which share a common phonetic form:

$$w \sim_{\text{Pho}} v :\Leftrightarrow \text{pho}(w) = \text{pho}(v) \tag{3.1}$$

[1] EDIT: See section 1.2.1 for the construction of the phonetization transducer from a `festival` LTS rule-set, and see appendix A.2 for details on the rule-set used here.

[2] $\mathcal{P}$ is a finite phonetic alphabet.

3.2.2 Levenshtein Edit Distance

Although the phonetic conflation technique described in the previous section is capable of successfully identifying a number of common historical graphemic variation patterns[3] such as *ey/ei, æ/ö, th/t,* and *tz/z*, it fails to conflate historical forms with any extant equivalent whenever the graphemic variation leads to non-identity of the respective phonetic forms, as determined by the LTS rule-set employed. In particular, whenever a historical variation would effect a pronunciation difference in synchronic forms, that variation will remain uncaptured by a phonetic conflation technique. Examples of such phonetically salient variations with respect to the simplified IMS German Festival rule-set include *guot/gut* "good", *liecht/licht* "light", *tiuvel/teufel* "devil", and *wolln/wollen* "want".

In order to accommodate graphematic variation phenomena beyond those for which strict phonetic identity of the variant forms obtains, we may employ an approximate search strategy based on the simple *Levenshtein edit distance* (Levenshtein, 1966; Navarro, 2001). Formally, let $\text{Lex} \subseteq \mathcal{A}^*$ be the lexicon of all extant forms, and let $d_{\text{Lev}} : \mathcal{A}^* \times \mathcal{A}^* \to \mathbb{N}$ represent the Levenshtein distance over grapheme strings, then define for every input word $w \in \mathcal{A}^*$ the "best" synchronic equivalent $\text{best}_{\text{Lev}}(w)$ as the unique extant word $v \in \text{Lex}$ with minimal edit-distance to the input word:[4]

$$\text{best}_{\text{Lev}}(w) = \arg\min_{v \in \text{Lex}} d_{\text{Lev}}(w, v) \tag{3.2}$$

Ideally, the image of a word w under best_{Lev} will itself be the canonical cognate sought,[5] leading to conflation of all strings which share a common image under best_{Lev}:

$$w \sim_{\text{Lev}} v :\Leftrightarrow \text{best}_{\text{Lev}}(w) = \text{best}_{\text{Lev}}(v) \tag{3.3}$$

The function $\text{best}_{\text{Lev}}(w) : \mathcal{A}^* \to \text{Lex}$ can be computed using a variant of the Dijkstra algorithm (Dijkstra, 1959) even when the lexicon is infinite (as in the case of productive nominal composition in German) whenever the set Lex can be represented by a finite-state acceptor (Mohri, 2002; Allauzen and Mohri, 2009; Jurish, 2010a). For current purposes, I used the (infinite) input

[3] EDIT: Pilz et al. (2006) refer to such variation patterns as "allographs", based on the intuition that the distinct surface forms are but variant realizations of a single lexical/phonological unit.

[4] I assume that whenever multiple extant minimal-distance candidate forms exist, one is chosen randomly.

[5] Note here that every extant form is its own "best" equivalent: $w \in \text{Lex}$ implies $\text{best}_{\text{Lev}}(w) = w$, since $d_{\text{Lev}}(w, w) = 0 < d_{\text{Lev}}(w, v)$ for all $v \neq w$.

w	$\text{best}_{\text{Lev}}(w)$		Extant Equivalent	
aug	*aus*	"out"	*auge*	"eye"
faszt	*fast*	"almost"	*fasst*	"grabs"
ouch	*buch*	"book"	*auch*	"also"
ram	*rat*	"advice"	*rahm*	"cream"
vol	*volk*	"people"	*voll*	"full"

Figure 3.1: Example spurious Levenshtein distance conflations

language of the TAGH morphology transducer (Geyken and Hanneforth, 2006) stripped of proper names, abbreviations, and foreign-language material to approximate Lex.[6]

3.2.3 Rewrite Transducer

While the simple edit distance conflation technique from the previous section is quite powerful and requires for its implementation only a lexicon of extant forms, the Levenshtein distance itself appears in many cases too coarse to function as a reliable predictor of etymological relations, since each edit operation (deletion, insertion, or substitution) is assigned a cost independent of the characters operated on and of the immediate context in the strings under consideration. This operand-independence of the traditional Levenshtein distance results in a number of spurious conflations such as those given in Figure 3.1.

In order to achieve a finer-grained and thus more precise mapping from historical forms to extant canonical cognates while preserving some degree of the robustness provided by the relaxation of the strict identity criterion implicit in the edit-distance conflation technique, a non-deterministic weighted finite-state "rewrite" transducer was developed to replace the simple Levenshtein metric. The rewrite transducer was compiled from a heuristic two-level rule-set (Karttunen et al., 1987; Kaplan and Kay, 1994; Laporte, 1997) whose 306 rules were manually constructed to reflect linguistically plausible patterns of diachronic variation as observed in the lemma-instance pairs automatically extracted from the full 5.5 million word DWB verse corpus (Jurish, 2008).[7]

[6] EDIT: See appendix A.3 for a precise specification of the modified TAGH lexical acceptor.

[7] EDIT: The method used for automatic extraction of lemma-instance pairs is described in section 1.3. Although some spurious pairs were extracted by this method, manual inspection of the most frequent corresponding string alignments was used to eliminate most "noise" from the heuristic rule-set. Preliminary experiments comparing an early version of the manual rule-set to a weighted editor FST supporting only context-insensitive

From → To /	Left __ Right	⟨Cost⟩	Example(s)	
$\varepsilon \to e$ /	$(\mathcal{A}\backslash\{e\})$ __ #	⟨ 5 ⟩	*aug* ⇝ *auge*	"eye"
$z \to s$ /	s __	⟨ 1 ⟩	*faszt* ⇝ *fasst*	"grabs"
$o \to a$ /	__ u	⟨ 1 ⟩	*ouch* ⇝ *auch*	"also"
$\varepsilon \to h$ /	V __ C	⟨ 5 ⟩	*ram* ⇝ *rahm*	"cream"
$l \to ll$ /	__	⟨ 8 ⟩	*vol* ⇝ *voll*	"full"

Figure 3.2: Some example heuristics used by the rewrite transducer. Here, ε represents the empty string, # represents a word boundary, and $\mathrm{V}, \mathrm{C} \subset \mathcal{A}$ are sets of vowel-like and consonant-like characters, respectively.

In particular, phonetic phenomena such as *schwa deletion*, *vowel shift*, *voicing alternation*, and *articulatory location shift* are easily captured by such rules.

Of the 306 heuristic rewrite rules, 131 manipulate consonant-like strings, 115 deal with vowel-like strings, and 14 operate directly on syllable-like units. The remaining 46 rules define expansions for explicitly marked elisions and unrecognized input. Some examples of rules used by the rewrite transducer are given in Figure 3.2.

Formally, the rewrite transducer M_{rw} defines a pseudo-metric $[\![M_{\mathrm{rw}}]\!] : \mathcal{A}^* \times \mathcal{A}^* \to \mathbb{R}_\infty$ on all string pairs (Mohri, 2009). Assuming the non-negative tropical semiring (Simon, 1987) is used to represent transducer weights, analogous to the transducer representation of the Levenshtein metric (Allauzen and Mohri, 2009), the rewrite pseudo-metric can be used as a drop-in replacement for the Levenshtein distance in Equations (3.2) and (3.3), yielding Equations (3.4) and (3.5):[8]

$$\mathrm{best}_{\mathrm{rw}}(w) = \arg\min_{v \in \mathrm{Lex}} [\![M_{\mathrm{rw}}]\!](w, v) \tag{3.4}$$

$$w \sim_{\mathrm{rw}} v :\Leftrightarrow \mathrm{best}_{\mathrm{rw}}(w) = \mathrm{best}_{\mathrm{rw}}(v) \tag{3.5}$$

3.3 Evaluation

3.3.1 Test Corpus

The conflation techniques described above were tested on a corpus of historical German verse extracted from the quotation evidence in a single volume of

single-character edit operations whose edit costs were automatically trained from the full set of automatically extracted lemma-instance pairs showed that the manual heuristics outperformed the automatically trained editor in precision, recall, and processing speed.

[8] EDIT: See appendix A.5 for a precise specification of the rewrite editor M_{rw}.

the digital first edition of the dictionary *Deutsches Wörterbuch* ("DWB", Bartz et al., 2004).[9] The test corpus contained 11,242 tokens of 4157 distinct word types, discounting non-alphabetic types such as punctuation. Each corpus type was manually assigned one or more extant equivalents based on inspection of its occurrences in the whole 5.5 million word DWB verse corpus in addition to secondary sources.[10] Only extinct roots, proper names, foreign and other non-lexical material were not explicitly assigned any extant equivalent at all; such types were flagged and treated as their own canonical cognates, i.e. identical to their respective "extant" equivalents. In all other cases, equivalence was determined by direct etymological relation of the root in addition to matching morphosyntactic features. Problematic types were marked as such and subjected to expert review. 296 test corpus types representing 585 tokens were ambiguously associated with more than one canonical cognate. In a second annotation pass, these remaining ambiguities were resolved on a per-token basis.

3.3.2 Evaluation Measures

The three conflation strategies from section 3.2 were evaluated using the gold-standard test corpus to simulate a document indexing and query scenario. Formally, let $G \subset \mathcal{A}^* \times \mathcal{A}^*$ represent the finite set of all gold-standard pairs $(w, \tilde{w})$ with $\tilde{w}$ the manually determined canonical cognate for the corpus type w, and let $Q = \{\tilde{w} : \exists (w, \tilde{w}) \in G\}$ be the set of all canonical cognates represented in the corpus. Then define for a binary conflation relation $\sim_R$ on $\mathcal{A}^*$ and a query string $q \in Q$ the sets $\text{relevant}(q), \text{retrieved}_R(q) \subseteq G$ of *relevant* and *retrieved* gold-standard pairs as:

$$\text{relevant}(q) = \{(w, \tilde{w}) \in G : \tilde{w} = q\} \tag{3.6}$$

$$\text{retrieved}_R(q) = \{(w, \tilde{w}) \in G : w \sim_R q\} \tag{3.7}$$

Type-wise precision and recall can then be defined directly as:

$$\text{pr}_{\text{typ}} = \frac{\left|\bigcup_{q \in Q} \text{retrieved}_R(q) \cap \text{relevant}(q)\right|}{\left|\bigcup_{q \in Q} \text{retrieved}_R(q)\right|} \tag{3.8}$$

$$\text{rc}_{\text{typ}} = \frac{\left|\bigcup_{q \in Q} \text{retrieved}_R(q) \cap \text{relevant}(q)\right|}{\left|\bigcup_{q \in Q} \text{relevant}(q)\right|} \tag{3.9}$$

[9] EDIT: See appendix C.1.1

[10] EDIT: In particular, Lexer (1992); Benecke et al. (1986); Kluge and Seebold (1989); Hennig (2001) were often consulted, in addition to the DWB itself.

Method	Time	Throughput
Pho	1.82 sec	7322 tok/sec
Lev	278.03 sec	48 tok/sec
rw	7.02 sec	1898 tok/sec

Table 3.1: Processing time for elementary canonicalization functions

If $\mathrm{tp}_R(q) = \mathrm{retrieved}_R(q) \cap \mathrm{relevant}(q)$ represents the set of *true positives* for a query q, then token-wise precision and recall are defined in terms of the gold-standard frequency function $f_G : G \to \mathbb{N}$ as:

$$\mathrm{pr}_{\mathrm{tok}} = \frac{\sum_{q \in Q, g \in \mathrm{tp}_R(q)} f_G(g)}{\sum_{q \in Q, g \in \mathrm{retrieved}_R(q)} f_G(g)} \tag{3.10}$$

$$\mathrm{rc}_{\mathrm{tok}} = \frac{\sum_{q \in Q, g \in \mathrm{tp}_R(q)} f_G(g)}{\sum_{q \in Q, g \in \mathrm{relevant}(q)} f_G(g)} \tag{3.11}$$

I use the unweighted harmonic precision-recall average F (van Rijsbergen, 1979) as a composite measure for both type- and token-wise evaluation modes:

$$\mathrm{F}(pr, rc) = \frac{2 \cdot pr \cdot rc}{pr + rc} \tag{3.12}$$

3.3.3 Results

The elementary canonicalization function for each of the conflation techniques[11] was applied to the entire test corpus to simulate a corpus indexing run. Running times for the various methods on a 1.8GHz Linux workstation using the `gfsmxl` C library are given in Table 3.1. The Levenshtein edit-distance technique is at a clear disadvantage here, roughly 150 times slower than the phonetic technique and 40 times slower than the specialized heuristic rewrite transducer. This effect is assumedly due to the density of the search space (which is maximal for an unrestricted Levenshtein editor), since the `gfsmxl` greedy k-best search of a Levenshtein transducer cascade generates at least $|\mathcal{A}|$ configurations per character, and a single backtracking step requires an additional $3|\mathcal{A}|$ heap extractions (Jurish, 2010a). Use of specialized lookup algorithms (Oflazer, 1996) might ameliorate such problems.

Quantitative results for several conflation techniques with respect to the DWB verse test corpus are given in Table 3.2. An additional conflation

[11] $\mathrm{pho}, \mathrm{best}_{\mathrm{Lev}}$ and $\mathrm{best}_{\mathrm{rw}}$ for the phonetic, Levenshtein, and heuristic rewrite transducer methods respectively

	Type-wise %			Token-wise %		
R	$\mathrm{pr}_{\mathrm{typ}}$	$\mathrm{rc}_{\mathrm{typ}}$	$\mathrm{F}_{\mathrm{typ}}$	$\mathrm{pr}_{\mathrm{tok}}$	$\mathrm{rc}_{\mathrm{tok}}$	$\mathrm{F}_{\mathrm{tok}}$
id	**99.9**	70.8	82.9	**99.1**	83.7	90.7
Pho	96.7	80.1	87.6	92.7	89.6	91.1
Lev	96.6	78.9	86.9	97.2	87.8	92.2
rw	98.5	88.4	**93.2**	98.2	93.4	**95.8**
Pho | Lev	94.1	84.3	88.9	91.3	91.6	91.5
Pho | rw	96.1	**89.8**	92.8	92.5	**94.5**	93.5

Table 3.2: Evaluation results for various conflation techniques on the DWB verse corpus. The maximum value in each column appears in boldface type.

relation "id" using strict identity of grapheme strings ($w \sim_{\mathrm{id}} v :\Leftrightarrow w = v$) was tested to provide a baseline for the methods described in section 3.2. As expected, the strict identity baseline relation was the most precise of all methods tested, achieving 99.9% type-wise and 99.1% token-wise precision. This is unsurprising, since the id method yields false positives only when a historical form is indistinguishable from a non-equivalent extant form, as in the case of the mapping *wider* $\rightsquigarrow$ *wieder* ("again") and the non-equivalent extant form *wider* ("against"). Despite its excellent precision, the baseline method's recall was the lowest of any tested method, which supports the claim that a synchronically-oriented lexicon cannot adequately account for a corpus of historical text. Type-wise recall was particularly low (70.8%), indicating that diachronic variation was more common in low-frequency types.

Surprisingly, the phonetic and Levenshtein edit-distance methods performed similarly for all measures except token-wise precision, in which Lev incurred 61.6% fewer errors than Pho. Given their near-identical type-wise precision, this difference can be attributed to a small number of phonetic misconflations involving high-frequency types, such as *wider* $\sim$ *wieder* ("against" $\sim$ "again"), *statt* $\sim$ *stadt*, ("instead" $\sim$ "city"), and *in* $\sim$ *ihn* ("in" $\sim$ "him"). Contrary to expectations, Lev did not yield any recall improvements over Pho, although the union of the two underlying conflation relations ($\sim_{\mathrm{Pho}\,|\,\mathrm{Lev}} = \sim_{\mathrm{Pho}} \cup \sim_{\mathrm{Lev}}$) achieved a type-wise recall of 84.3% (token-wise recall 91.6%), which suggests that these two methods complement one another when both an LTS module and a high-coverage lexicon of extant types are available.

Of the methods described in section 3.2, the heuristic rewrite transducer M_{rw} performed best overall, with a type-wise harmonic mean F of 93.2% and a token-wise F of 95.8%. While M_{rw} incurred some additional precision

errors compared to the naïve graphemic identity method id, these were not as devastating as those incurred by the phonetic or Levenshtein distance methods, which supports the claim from section 3.2.3 that a fine-grained context-sensitive pseudo-metric incorporating linguistic knowledge can more accurately model diachronic processes than an all-purpose metric like the Levenshtein distance.

Recall was highest for the composite phonetic-rewrite relation $\sim_{\mathrm{Pho}\,|\,\mathrm{rw}} = \sim_{\mathrm{Pho}} \cup \sim_{\mathrm{rw}}$, although the precision errors induced by the phonetic component outweighed the comparatively small gain in recall. The best overall performance is achieved by the heuristic rewrite transducer M_{rw} on its own, yielding a reduction of 60.3% in type-wise recall errors and of 59.5% in token-wise recall errors, while minimizing the number of newly introduced precision errors.

3.4 Conclusion & Outlook

I have presented three different methods for associating unknown historical word forms with synchronically active canonical cognates. The heuristic mapping of unknown forms to extant equivalents by means of linguistically motivated context-sensitive rewrite rules yielded the best results in an information retrieval task on a corpus of historical German verse, reducing type-wise recall errors by over 60% compared to a naïve text-matching strategy. Depending on the availability of linguistic resources (e.g. phonetization rule-sets, lexica), use of phonetic canonicalization and/or Levenshtein edit distance may provide a more immediately accessible route to improved recall for other languages or applications, at the expense of some additional loss of precision.

I am interested in verifying these results using larger corpora than the small test corpus used here, as well as extending the techniques described here to other languages and domains. In particular, I am interested in comparing the performance of the domain-specific rewrite transducer used here to other linguistically motivated language-independent metrics such as those described by Covington (1996); Kondrak (2000).

ORIGINALLY APPEARED AS:
Bryan Jurish. More than words: Using token context to improve canonicalization of historical German. *Journal for Language Technology and Computational Linguistics*, 25(1):23-40, 2010c.

Chapter 4

More Than Words

4.1 Introduction

Historical text presents numerous challenges for contemporary natural language processing techniques. In particular, the absence of consistent orthographic conventions in historical text presents difficulties for any system requiring reference to a fixed lexicon accessed by orthographic form, such as information retrieval systems (Sokirko, 2003; Cafarella and Cutting, 2004), part-of-speech taggers (DeRose, 1988; Brill, 1992; Schmid, 1994), simple word stemmers (Lovins, 1968; Porter, 1980), or more sophisticated morphological analyzers (Geyken and Hanneforth, 2006; Zielinski et al., 2009).[1]

Traditional approaches to the problems arising from an attempt to incorporate historical text into such a system rely on the use of additional specialized (often application-specific) lexical resources to explicitly encode known historical variants. Such specialized lexica are not only costly and time-consuming to create, but also – in their simplest form of static finite word lists – necessarily incomplete in the case of a morphologically productive language like German, since a simple finite lexicon cannot account for highly productive morphological processes such as nominal composition (cf. Kempken et al., 2006).

To facilitate the extension of synchronically-oriented natural language pro-

[1] While neither information retrieval (IR) systems nor stemmers use a *static* fixed lexicon in the usual sense, the effective lexicon of an IR system is fixed at indexing time as the set of all actually occurring word forms. Similarly, the lexicon of a traditional stemmer has a static portion (hard-coded inflection rules) as well as a dynamic portion (set of stems) determined by the actual input. In both cases, historical spelling variants will be treated as distinct lexemes rather than associated with an equivalent contemporary cognate unless additional measures such as those described here are taken.

cessing techniques to historical text while minimizing the need for specialized lexical resources, one may first attempt an automatic *canonicalization* of the input text. Canonicalization approaches (Jurish, 2008, 2010b; Gotscharek et al., 2009b) treat orthographic variation phenomena in historical text as instances of an error-correction problem (Shannon, 1948; Kukich, 1992; Brill and Moore, 2000), seeking to map each (unknown) word of the input text to one or more extant *canonical cognates*: synchronically active types which preserve both the root and morphosyntactic features of the associated historical form(s). To the extent that the canonicalization was successful, application-specific processing can then proceed normally using the returned canonical forms as input, without any need for additional modifications to the application lexicon.

I distinguish between *type-wise* canonicalization techniques which process each input word independently and *token-wise* techniques which make use of the context in which a given instance of a word occurs. In this paper, I present a token-wise canonicalization method which functions as a disambiguator for sets of hypothesized canonical forms as returned by one or more subordinated type-wise techniques. Section 4.2 provides a brief review of the type-wise canonicalizers used to generate hypotheses, while section 4.3 is dedicated to the formal characterization of the disambiguator itself. Section 4.4 contains a quantitative evaluation of the disambiguator's performance on an information retrieval task over a manually annotated corpus of historical German. Finally, section 4.5 provides a brief summary and conclusion.

4.2 Type-wise Conflation

Type-wise conflation techniques are those which process each input word in isolation, independently of its surrounding context. Such a type-wise treatment allows efficient processing of large documents and corpora (since each input type need only be processed once), but disregards potentially useful context information. Formally, a type-wise conflator r is fully specified by a characteristic *conflation relation* $\sim_r$, a binary relation on the set $\mathcal{A}^*$ of all strings over the finite grapheme alphabet $\mathcal{A}$. Prototypically, $\sim_r$ will be a true equivalence relation, inducing a partitioning of the set $\mathcal{A}^*$ of possible word types into equivalence classes or "conflation sets" $[w]_r = \{v \in \mathcal{A}^* : v \sim_r w\}$ induced by some type $w \in \mathcal{A}^*$. Where appropriate, I distinguish between the full conflation set $[w]_r$ containing all strings conflated by r with w and a conflator-specific finite subset $\downarrow[w]_r \subseteq [w]_r$ representing the *canonicalization hypotheses* provided by r for w: the former sets will be used to characterize the retrieval function for r used to define the evaluation measures precision

and recall in section 4.4.2, while the latter will be used in the definition of the token-wise disambiguator in section 4.3.3. Unless otherwise specified, I assume $\downarrow[w]_r = [w]_r$. In the sequel, I will use the terms "conflation" and "type-wise canonicalization" interchangeably where no ambiguity will result, and the term "conflator" will be used to refer to a specific type-wise canonicalization method.

4.2.1 String Identity

The simplest of all possible conflators is raw identity of surface strings. The conflation relation $\sim_{\text{id}}$ is in this case nothing more or less than the string identity relation itself:

$$w \sim_{\text{id}} v :\Leftrightarrow w = v \tag{4.1}$$

String identity is the easiest conflator to implement (no additional programming effort or resources are required) and provides a high degree of precision, "false friends" being limited to historical homographs such as the historical form *wider* when it occurs as a variant of the contemporary form *wieder* ("again") rather than the lexically distinct contemporary homograph *wider* ("against"). Since its coverage is restricted to valid contemporary forms, string identity cannot account for any spelling variation at all, resulting in very poor recall – many relevant types are not retrieved in response to a query in current orthography. Nonetheless, its inclusion as a conflator ensures that the set of candidate hypotheses $[w]$ for a given input word w is non-empty,[2] and it provides a baseline with respect to which the relative utility of more sophisticated conflators can be evaluated.

As an example, consider the historical form *Abftände*, a variant of the contemporary cognate *Abstände* ("distances"). The conflation set $[\textit{Abftände}]_{\text{id}} = \{\textit{Abftände}\}$ is non-empty, but does not contain the desired contemporary cognate ($\textit{Abstände} \notin [\textit{Abftände}]_{\text{id}}$), so Equation (4.20) from section 4.4.2 dictates that no instances of the historical variant *Abftände* will be retrieved via string identity for a query of the contemporary form *Abstände*.

4.2.2 Transliteration

A slightly less naïve family of conflation methods are those which employ a simple deterministic transliteration function to replace input characters

[2]Since $[w]_{\text{id}} = \{w\}$, $[w]_{\text{id}} \subseteq [w]$ implies $w \in [w]$, and thus $[w] \neq \emptyset$. Since the more reliable transliterating conflator described in section 4.2.2 also ensures a non-empty set of conflation hypotheses, the identity conflator itself was not used to generate hypotheses for the disambiguator in the current experiments.

which do not occur in contemporary orthography with extant equivalents. Formally, a transliteration conflator is defined in terms of a character transliteration function $\text{xlit} : \mathcal{A} \to \widetilde{\mathcal{A}}^*$, where $\mathcal{A}$ is as before a "universal" grapheme alphabet (e.g. the set of all Unicode[3] characters) and $\widetilde{\mathcal{A}} \subseteq \mathcal{A}$ is that subset of the universal alphabet allowed by contemporary orthographic conventions. The elementary character transliteration function is extended to a string transliteration function $\text{xlit}^* : \mathcal{A}^* \to \widetilde{\mathcal{A}}^*$ in the usual manner by iteratively applying xlit to each character of the input string in turn (Equation 4.2), canonicalization hypotheses are limited to the transliterator output (Equation 4.3), and the characteristic conflation relation $\sim_{\text{xlit}}$ is defined as identity of transliterated strings (Equation 4.4):

$$\text{xlit}^*(a_1 a_2 \ldots a_n) := \text{xlit}(a_1)\,\text{xlit}(a_2)\ldots\text{xlit}(a_n) \tag{4.2}$$

$$\downarrow[w]_{\text{xlit}} := \{\text{xlit}^*(w)\} \tag{4.3}$$

$$w \sim_{\text{xlit}} v :\Leftrightarrow \text{xlit}^*(w) = \text{xlit}^*(v) \tag{4.4}$$

In the case of historical German, deterministic transliteration is especially useful for its ability to account for typographical phenomena, e.g. by mapping 'ſ' (long 's', as commonly appeared in texts typeset in fraktur) to a conventional round 's', and mapping superscript 'e' to the conventional *Umlaut* diacritic '¨', as in the transliteration *Abftäͤnde* ↦ *Abstände* ("distances"). Given this transliteration, a query for the contemporary form *Abstände* will successfully retrieve all instances of the historical form *Abftäͤnde*: $\text{xlit}^*(\textit{Abstände}) = \textit{Abstände} = \text{xlit}^*(\textit{Abftäͤnde})$, so $\textit{Abstände} \in [\textit{Abftäͤnde}]_{\text{xlit}}$.

The current work makes use of a conservative transliteration function based on the `Text::Unidecode` Perl module.[4,5] Due to the fact that the underlying character transliteration table is comparatively small and can be implemented as an in-memory array, transliteration is a very efficient conflation method, with $\mathcal{O}(\text{xlit}) = \mathcal{O}(1)$ and therefore $\mathcal{O}(\text{xlit}^*) = \mathcal{O}(n)$. In terms of expressive power, since xlit is finite, it can be represented by a finite state transducer, and therefore so can its reflexive and transitive closure xlit^*.

Despite its efficiency, and although it outdoes even string identity in terms of its precision, deterministic transliteration suffers from its inability to account for spelling variation phenomena involving extant characters such as the *th/t* and *ey/ei* allographs common in historical German. As an example, consider an instance of the historical form *Theyl* corresponding to the contemporary cognate *Teil* ("part"). Both historical and contemporary forms will be transliterated to themselves, since both strings contain only

[3] Unicode Consortium (2011), `http://www.unicode.org/`

[4] `http://search.cpan.org/~sburke/Text-Unidecode-0.04/`

[5] EDIT: See appendix A.1 for a precise specification of the transliterator used here.

extant characters, but the historical form will not be retrieved by a query for the contemporary form: $\text{xlit}^*(Teil) = Teil \neq Theyl = \text{xlit}^*(Theyl)$ implies $Teil \not\sim_{\text{xlit}} Theyl$ and therefore $Teil \notin [Theyl]_{\text{xlit}}$.

4.2.3 Phonetization

A more powerful family of conflation methods is based on the dual intuitions that graphemic forms in historical text were constructed to reflect phonetic forms[6] and that the phonetic system of the target language is diachronically more stable than its graphematic system. Phonetic conflators map each (historical or extant) word $w \in \mathcal{A}^*$ to a unique phonetic form $\text{pho}(w)$ by means of a computable function $\text{pho} : \mathcal{A}^* \to \mathcal{P}^*$,[7] conflating those strings which share a common phonetic form:

$$w \sim_{\text{pho}} v :\Leftrightarrow \text{pho}(w) = \text{pho}(v) \tag{4.5}$$

Since $[w]_{\text{pho}}$ may be infinite – if for example $\text{pho}(\cdot)$ maps any substring of one or more instances of a single character (e.g. 'a') to a single phone (e.g. [a]) – additional care must be taken to ensure a finite set of canonicalization hypotheses $\downarrow[w]_{\text{pho}}$. A straightforward way to ensure a finite hypothesis set is simply to restrict $[w]_{\text{pho}}$ to some finite set of pre-defined target strings $T \subset \mathcal{A}^*$, setting $\downarrow[w]_{\text{pho}} = \downarrow_T[w]_{\text{pho}} = [w]_{\text{pho}} \cap T$. If pho can be represented as a finite-state transducer M_{pho} and the target lexicon can be represented as a finite-state acceptor A_{Lex}, a more robust alternative is to use a k-best string lookup algorithm such as that described in Jurish (2010a) on the cascade $\mathcal{C}_{\text{pho}}(w) = \text{Id}(w) \circ M_{\text{pho}} \circ M_{\text{pho}}^{-1} \circ A_{\text{Lex}}$, defining $\downarrow[w]_{\text{pho}} = \downarrow_{\mathcal{C},k}[w]_{\text{pho}} = \text{kbest}(k, \mathcal{C}_{\text{pho}}(w))$ for some finite upper bound k on the number of admissible hypotheses, assuming an appropriate weighting scheme on A_{Lex}.

The phonetic conversion module used here was adapted from the phonetization rule-set distributed with the IMS German Festival package (Möhler et al., 2001), a German language module for the Festival text-to-speech system (Black and Taylor, 1997) and compiled as a finite-state transducer (Jurish, 2008).[8,9] Phonetic conflation offers a substantial improvement in recall over

[6] Keller (1978) codified this intuition as the imperative "write as you speak" governing historical spelling conventions.

[7] $\mathcal{P}$ is a finite phonetic alphabet.

[8] In the absence of a language-specific phonetization function, a general-purpose phonetic digest algorithm such as SOUNDEX (Russell, 1918), the *Kölner Phonetik* (Postel, 1969), PHONIX (Gadd, 1988, 1990), or Metaphone (Philips, 1990, 2000) may be employed instead (Robertson and Willett, 1993; Kempken, 2005).

[9] EDIT: See section 1.2.1 for the construction of the phonetization transducer from a `festival` LTS rule-set, and see appendix A.2 for details on the rule-set used here.

conservative methods such as transliteration or string identity: variation phenomena such as the *th/t* and *ey/ei* allographs mentioned above are correctly captured by the phonetization transducer: $\text{pho}(Theyl) = [\text{taɪl}] = \text{pho}(Teil)$ which implies $Teil \in [Theyl]_{\text{pho}}$. Unfortunately, these improvements often come at the expense of precision: in particular, many high-frequency types are misconflated by the simplified phonetization rule-set, including **in* ∼ *ihn* ("in" ∼ "him"), **statt* ∼ *Stadt*, ("instead" ∼ "city"), and **wider* ∼ *wieder* ("against" ∼ "again"). While such high-frequency cases might be easily handled in a mature system by a small exception lexicon, the underlying tendency of strict phonetic conflation either to over- or to under-generalize – depending on the granularity of the phonetization function – is likely to remain, expressing itself in information retrieval tasks as reduced precision or reduced recall, respectively.

4.2.4 Rewrite Transduction

Despite its comparatively high recall, the phonetic conflator fails to relate unknown historical forms with any extant equivalent whenever the graphemic variation leads to non-identity of the respective phonetic forms (e.g. $\text{pho}(umb) = [\text{ʔʊmp}] \neq [\text{ʔʊm}] = \text{pho}(um)$ for the historical variant *umb* of the preposition *um* ("around")), suggesting that recall might be further improved by relaxing the strict identity criterion on the right hand side of Equation (4.5). Moreover, a fine-grained and appropriately parameterized conflator should be less susceptible to precision errors than an "all-or-nothing" (phonetic) identity condition (Kondrak, 2000, 2002). A technique which fulfills both of the above desiderata is *rewrite transduction*, which can be understood as a generalization of the well-known *string edit distance* (Damerau, 1964; Levenshtein, 1966).

Formally, let $\text{Lex} \subseteq \mathcal{A}^*$ be the (possibly infinite) lexicon of all extant forms encoded as a finite-state acceptor A_{Lex}, and let M_{rw} be a weighted finite-state transducer over a bounded semiring $\mathcal{K}$ which models (potential) diachronic change likelihood as a weighted rational relation. Then define for every input type $w \in \mathcal{A}^*$ the "best" extant equivalent $\text{best}_{\text{rw}}(w)$ as the unique extant type $v \in \text{Lex}$ with minimal edit-distance to the input word:

$$\text{best}_{\text{rw}}(w) = \arg\min_{v \in \mathcal{A}^*} [\![M_{\text{rw}} \circ A_{\text{Lex}}]\!](w, v) \tag{4.6}$$

Ideally, the image of a word w under best_{rw} will itself be the canonical cognate sought, leading to conflation of all strings which share a common image under best_{rw}:

$$w \sim_{\text{rw}} v :\Leftrightarrow \text{best}_{\text{rw}}(w) = \text{best}_{\text{rw}}(v) \tag{4.7}$$

The current experiments were performed using the heuristic rewrite transducer described in Jurish (2010b), compiled from 306 manually constructed two-level rules, while the lexical target acceptor A_{Lex} was extracted from the TAGH morphology transducer (Geyken and Hanneforth, 2006).[10] The native TAGH weights were scaled for compatibility and used to provide a prior cost distribution over target word forms based on their derivational complexity. Best-path lookup was performed using a specialized variant of the well-known *Dijkstra algorithm* (Dijkstra, 1959) as described in Jurish (2010a). Related approaches to historical variant detection include Kempken (2005); Rayson et al. (2005); Ernst-Gerlach and Fuhr (2006); Gotscharek et al. (2009b).

Although this rewrite cascade does indeed improve both precision and recall with respect to the phonetic conflator, these improvements are of comparatively small magnitude, precision in particular remaining well below the level of conservative conflators such as naïve string identity or transliteration, due largely to interference from "false friends" such as the valid contemporary compound *Rockermehl* ("rocker-flour") for the historical variant *Rockermel* of the contemporary form *Rockärmel* ("coat-sleeve") as appearing in Figure 4.1.

4.3 Token-wise Disambiguation

In an effort to recover some degree of the precision offered by conservative conflation techniques such as transliteration while still benefiting from the flexibility and improved recall provided by more ambitious techniques such as phonetization or rewrite transduction, I have developed a method for disambiguating type-wise conflation sets which operates on the token level, using sentential context to determine a unique "best" canonical form for each input token. Specifically, the disambiguator employs a Hidden Markov Model (HMM) whose lexical probability matrix is dynamically re-computed for each input sentence from the conflation sets returned by one or more subordinated type-wise conflators, and whose transition probabilities are given by a static word k-gram model of the target language, in this case contemporary German adhering to current orthographic conventions. Similar approaches for traditional spell-checking applications using strictly local context for language modelling have been described by Kernighan et al. (1990); Church and Gale (1991); Brill and Moore (2000); Verberne (2002). Most closely related to the current proposal is the approach of Mays et al. (1991), who use a word trigram model to disambiguate unweighted confusion

[10] EDIT: See appendix A.5 for a precise specification of the rewrite heuristics, and see appendix A.3 for details on the extraction of the rewrite target lexicon from the TAGH morphology transducer.

	Dete	ſammlete	Steyne	im	Rockermel
id	*Dete*	*ſammlete*	*Steyne*	*im*	*Rockermel*
xlit	*Dete*	*sammlete*	*Steyne*	*im*	*Rockermel*
pho	$\emptyset$	$\emptyset$	{*Steine*}	{*im*, *ihm*}	{*Rockärmel*}
rw	*Tete*⟨1⟩	*sammelte*⟨5⟩	*Steine*⟨1⟩	*im*⟨0⟩	*Rockermehl*⟨10⟩
hmm	*Dete*	*sammelte*	*Steine*	*im*	*Rockärmel*

Figure 4.1: Example of the proposed conflator disambiguation architecture for the input sentence *"Dete ſammlete Steyne im Rockermel"* ("Dete gathered rocks in the coat-sleeve"). Costs assigned by the rewrite transducer appear in angled brackets, and the conflation hypotheses selected by the HMM disambiguator are underlined.

sets returned by a traditional approximate Damerau-Levenshtein matcher analogous to the rewrite cascade from section 4.2.4. An example of the proposed disambiguation architecture for the conflators described in section 4.2 is given in Figure 4.1.

4.3.1 Basic Model

Formally, let $\mathcal{W} \subset \widetilde{\mathcal{A}}^*$ be a finite set of known extant words, let $\mathbf{u} \notin \mathcal{W}$ be a designated symbol representing an unknown word, let $S = \langle w_1, \ldots, w_{n_S} \rangle$ be an input sentence of n_S (historical) words with $w_i \in \mathcal{A}^*$ for $1 \leq i \leq n_S$, and let $R = \{r_1, \ldots, r_{n_R}\}$ be a finite set of (opaque) type-wise conflators. Then, the disambiguator HMM is defined in the usual way (Rabiner, 1989; Charniak et al., 1993; Manning and Schütze, 1999) as the 5-tuple $D = \langle \mathcal{Q}, \mathcal{O}_S, \Pi, A, B_S \rangle$, where:

1. $\mathcal{Q} = (\mathcal{W} \cup \{\mathbf{u}\}) \times R$ is a finite set of model *states*, where each state $q \in \mathcal{Q}$ is a pair $\langle \tilde{w}_q, r_q \rangle$ composed of an extant word form $\tilde{w}_q$ and a conflator r_q;

2. $\mathcal{O}_S = \bigcup_{i=1}^{n_S} \{w_i\}$ is the set of *observations* for the input sentence S;

3. $\Pi : \mathcal{Q} \to [0, 1] : q \mapsto p(Q_1 = q)$ is a static probability distribution over $\mathcal{Q}$ representing the model's *initial state probabilities*;

4. $A : \mathcal{Q}^k \to [0, 1] : \langle q_1, \ldots, q_k \rangle \mapsto p(Q_i = q_k | Q_{i-k+1} = q_1, \ldots, Q_{i-1} = q_{k-1})$ is a static conditional probability distribution over state k-grams representing the model's *state transition probabilities*; and

5. $B_S : \mathcal{Q} \times \mathcal{O}_S \to [0,1] : \langle q, o \rangle \mapsto p(O = o | Q = q)$ is a dynamic probability distribution over observations conditioned on states representing the model's *lexical probabilities*.

Using the shorthand notation w_i^{i+j} for the string $w_i w_{i+1} \ldots w_{i+j}$, the model D computes sentential probability as the sum of path probabilities over all possible generating state sequences:

$$p(S = w_1^{n_S}) = \sum_{q_1^{n_S} \in \mathcal{Q}^{n_S}} p(S = w_1^{n_S}, Q = q_1^{n_S}) \tag{4.8}$$

Assuming suitable boundary handling for negative indices, joint path probabilities themselves are computed as:

$$p(S = w_1^{n_S}, Q = q_1^{n_S}) = \prod_{i=1}^{n_S} p(q_i | q_{i-k+1}^{i-1}) p(w_i | q_i) \tag{4.9}$$

Underlying these equations are the following Markov assumptions:

$$p(q_i | q_1^{i-1}, w_1^{i-1}) = p(q_i | q_{i-k+1}^{i-1}) \tag{4.10}$$

$$p(w_i | q_1^{i}, w_1^{i-1}) = p(w_i | q_i) \tag{4.11}$$

Equation (4.10) asserts that state transition probabilities depend on at most the preceding $k-1$ states. Equation (4.11) asserts the independence of observed surface forms (historical spellings) from all but the model's current state. Taken together, these assumptions will lead to the use of a k-gram distribution over contemporary word forms to model both syntactic and (local) semantic constraints of the target language as operating on conflator-dependent type-wise canonicalization hypotheses for historical input forms. Crucially, the product of these two component distributions as used in the path probability computation from Equation (4.9) will allow linguistic context constraints (insofar as they are captured by the k-gram transition probabilities) to override prior type-wise estimates of a conflation's reliability (and vice versa), leading to a disambiguator dependent on both token context and prior estimates of conflation likelihood.

4.3.2 Transition Probabilities

The finite target lexicon $\mathcal{W}$ can easily be extracted from a corpus of contemporary text. For estimating the static distributions Π and A, we first make the following assumptions:

$$p(Q = \langle \tilde{w}_q, r_q \rangle) = p(W = \tilde{w}_q) p(R = r_q) \tag{4.12}$$

$$p(R = r) = \frac{1}{n_R} \tag{4.13}$$

Equation (4.12) asserts the independence of extant forms and conflators, while Equation (4.13) assumes a uniform distribution over conflators. Given these assumptions, the static state distributions Π and A can be estimated as:

$$\Pi(q) \;:\approx\; p\left(W_1 = \tilde{w}_q\right)/n_R \tag{4.14}$$

$$A(q_1, \ldots, q_k) \;:\approx\; p\left(W_i = \tilde{w}_{q_k} | W_{i-k+1}^{i-1} = \tilde{w}_{q_1} \ldots \tilde{w}_{q_{k-1}}\right)/n_R \tag{4.15}$$

Equations (4.14) and (4.15) are nothing more or less than a word k-gram model over extant forms, scaled by the constant $\frac{1}{n_R}$. One can therefore use standard maximum likelihood techniques to estimate Π and A from a corpus of contemporary text (Bahl et al., 1983; Manning and Schütze, 1999).

For the current experiments, a word trigram model ($k = 3$) was trained on the TIGER corpus of contemporary German (Brants et al., 2002). Probabilities for the "unknown" form **u** were computed using the simple smoothing technique of assigning **u** a pseudo-frequency of $\frac{1}{2}$ (Lidstone, 1920; Manning and Schütze, 1999). To account for unseen trigrams, the resulting trigram model was smoothed by linear interpolation of uni-, bi-, and trigrams (Jelinek and Mercer, 1980, 1985), using the method described by Brants (2000) to estimate the interpolation coefficients.

4.3.3 Lexical Probabilities

In the absence of a representative corpus of conflator-specific manually annotated training data, simple maximum likelihood techniques cannot be used to estimate the model's lexical probabilities B_S. Instead, lexical probabilities are instantiated as a Maxwell-Boltzmann distribution for a set d_r of conflator-specific distance functions (Jaynes, 1983):

$$B\Big(\langle \tilde{w}, r\rangle, w\Big) :\approx \frac{b^{\beta d_r(w,\tilde{w})}}{\sum_{r' \in R} \sum_{\tilde{w}' \in \downarrow[w]_{r'}} b^{\beta d_{r'}(w,\tilde{w}')}} \tag{4.16}$$

Here, $b, \beta \in \mathbb{R}$ are free model parameters with $b \geq 1$ and $\beta \leq 0$. For a conflator $r \in R$, the function $d_r : \mathcal{A}^* \times \mathcal{W} \to \mathbb{R}_+$ is a pseudo-metric used to estimate the reliability of the conflator's association of an input word w with the extant form $\tilde{w}$, and the set $\downarrow[w]_r \subseteq [w]_r \subseteq \mathcal{A}^*$ is a finite set of canonicalization hypotheses provided by r for w, as described in section 4.2.

It should be explicitly noted that the denominator of the right-hand side of Equation (4.16) is a sum over all model states (canonicalization hypotheses) $\langle \tilde{w}', r'\rangle$ actually associated with the observation argument w by the type-wise conflation stage, and *not* a sum over observations w' associable with the state argument $\langle \tilde{w}, r\rangle$. This latter sum (if it could be efficiently computed) would

adhere to the traditional form $\Big(\text{sim}(o,q)/\sum_{o'}\text{sim}(o',q)\Big)$ for estimating a probability distribution $p(O|Q)$ over *observations* conditioned on model states such as the HMM lexical probability matrix B_S is defined to represent; whereas the estimator in Equation (4.16) is of the form $\Big(\text{sim}(o,q)/\sum_{q'}\text{sim}(o,q')\Big)$, which corresponds more closely to a distribution $p(Q|O)$ over *states* conditioned on observations.[11]

From a practical standpoint, it should be clear that Equation (4.16) is much more efficient to compute than an estimator summing globally over potential observations, since all the data needed to compute Equation (4.16) are provided by the type-wise preprocessing of the input sentence S itself, whereas a theoretically pure global estimator would require a whole arsenal of *inverse* conflators as well as a mechanism for restricting their outputs to some tractable set of admissible historical forms, and hence would be of little practical use. From a formal standpoint, I believe that Equation (4.16) as used in the run-time disambiguator can be shown to be equivalent to a global estimator, provided that the conflator pseudo-metrics d_r are symmetric and the languages of both historical and extant forms have identical and uniform density with respect to the d_r, but a proof of this conjecture is beyond the scope of this paper.

It was noted above in section 4.2.3 that for the phonetic conflator in particular, the equivalence class $[w]_{\text{pho}} = \{v \in \mathcal{A}^* : w \sim_{\text{pho}} v\}$ may not be finite. In order to ensure the computational tractability of Equation (4.16) therefore, the phonetic conflation hypotheses considered were implicitly restricted to the finite set $\mathcal{W}$ of known extant forms used to define the model's states, $\downarrow[w]_{\text{pho}} = \downarrow_{\mathcal{W}}[w]_{\text{pho}} = [w]_{\text{pho}} \cap \mathcal{W}$. Transliterations and rewrite targets which were not also known extant forms were implicitly mapped to the designated symbol **u** for purposes of estimating transition probabilities for previously unseen extant word types.

For the current experiments, the following model parameters were used:

$$\begin{aligned}
b &= 2 \\
\beta &= -1 \\
R &= \{\text{xlit}, \text{pho}, \text{rw}\} \\
d_{\text{xlit}}(w,\tilde{w}) &= 2/|w| && \text{if } \tilde{w} = \text{xlit}^*(w) \\
d_{\text{pho}}(w,\tilde{w}) &= 1/|w| && \text{if } \tilde{w} \in \downarrow[w]_{\text{pho}} \\
d_{\text{rw}}(w,\tilde{w}) &= [\![M_{\text{rw}} \circ A_{\text{Lex}}]\!](w,\tilde{w})/|w| && \text{if } \tilde{w} = \text{best}_{\text{rw}}(w)
\end{aligned}$$

In all other cases, $d_r(w,\tilde{w})$ is undefined and $B(\langle\tilde{w}, r\rangle, w) = 0$. Note that all

[11] See the discussion surrounding Equation 20 in Charniak et al. (1993) for a more detailed look at these two sorts of lexical probability estimators and their effects on HMM part-of-speech taggers.

conflator distance functions are scaled by inverse input word length $\frac{1}{|w|}$, thus expressing an average distance per input character as opposed to an absolute distance for the input word. Defining distance functions in terms of (inverse) word length in this manner captures the intuition that a conflator is less likely to discover a false positive conflation for a longer input word than for a short one, natural language lexica tending to be maximally dense for short (usually closed-class) words.[12] The transliteration and phonetic conflators are constants given input word length, whereas the rewrite conflator makes use of the cost $[\![M_{\text{rw}} \circ A_{\text{Lex}}]\!](w, \tilde{w})$ assigned to the conflation pair by the rewrite cascade itself.

4.3.4 Runtime Disambiguation

Having defined the disambiguator model D, it can be used to determine a unique "best" canonical form for each input sentence S by application of the well-known *Viterbi algorithm* (Viterbi, 1967). Formally, the Viterbi algorithm computes the state path with maximal probability for the observed sentence:

$$\textsc{Viterbi}(S, D) = \underset{\langle q_1,\ldots,q_{n_S}\rangle \in \mathcal{Q}^{n_S}}{\arg\max}\ p(q_1, \ldots, q_{n_S}, S|D) \tag{4.17}$$

Extracting the disambiguated canonical forms $\hat{S} = \langle \hat{w}_1, \ldots, \hat{w}_{n_S} \rangle \in (\mathcal{A}^*)^{n_S}$ from the state sequence $\hat{Q} = \langle \hat{q}_1, \ldots, \hat{q}_{n_S} \rangle = \textsc{Viterbi}(S, D)$ returned by the Viterbi algorithm is a simple matter of projecting the extant word components of the HMM state structures, taking care to map the designated symbol $\mathbf{u}$ onto an appropriate output string. Let witness : $\wp(\mathcal{A}^*) \to \mathcal{A}^*$ be a choice function over conflation hypotheses,[13] witness$(\downarrow[w]_r) \in \downarrow[w]_r$ for all $w \in \mathcal{A}^*, r \in R$ with $\downarrow[w]_r \neq \emptyset$, and for $1 \leq i \leq n_S$, define:

$$\hat{w}_i := \begin{cases} \text{witness}\left(\downarrow[w]_{r_{\hat{q}_i}}\right) & \text{if } \tilde{w}_{\hat{q}_i} = \mathbf{u} \\ \tilde{w}_{\hat{q}_i} & \text{otherwise} \end{cases} \tag{4.18}$$

Following the equivalence class notation for type-wise conflators, I write $[w_i]_{\text{hmm},D}$ to denote the singleton set $\{\hat{w}_i\}$ containing the unique canonical

[12] Despite this tendency of natural languages, the combinatorial properties of concatenative monoids dictate that the number of potential "false friends" grows exponentially with input string length if for example arbitrary substitutions are allowed, suggesting an increased likelihood of false positive conflations for *longer* input words. In this context, note that the use of per-character distances results in higher-entropy probability distributions (Shannon, 1948) for longer input strings, effectively treating the d_r distance estimates as increasingly unreliable as input string length grows.

[13] Since conflation hypothesis sets $\downarrow[w]_r$ are finite, the axiom of choice is not strictly required here.

form returned by the HMM disambiguator D for an input token w_i in sentential context S, omitting the model subscript D where no ambiguity will result.

4.3.5 Expressive Power

It was noted in section 4.2 above that each of the type-wise conflators used in the current approach have representations as (weighted) finite-state transducers (WFSTs). Since the union of WFSTs is itself a WFST, as is the concatenation of WFSTs (Mohri, 2009), the type-wise analysis stage which generates canonicalization hypotheses for the disambiguator can be expressed by an extended rational algebraic expression, assuming specialized functions such as the k-best lookup used by the rewrite transducer are included in the inventory of admissible operations. Hidden Markov Models have been shown to be equivalent to the sub-family of WFSTs called probabilistic finite-state automata (PFSAs) by Vidal et al. (2005). Pereira and Riley (1997) advocate a decomposition of HMM component distributions into dedicated WFSTs which may then be cascaded (composed) to simulate the original HMM for use in speech processing applications. Hanneforth and Würzner (2009) present a technique for creating n-gram language models using only the algebra of weighted rational languages which can in principle be extended to implement the disambiguator's dynamic lexical probability distribution given by Equation (4.16) as just such a dedicated WFST component. Finally, since the Viterbi algorithm can be applied directly to PFSAs (Vidal et al., 2005) and with minimal adaptation to appropriately weighted WFSTs (Mohri, 2002; Jurish, 2010a), the entire proposed canonicalization architecture does not exceed the expressive power of the weighted rational relations.

4.4 Evaluation

4.4.1 Test Corpus

The conflation and disambiguation techniques described above were tested on a manually annotated corpus of historical German drawn from the *Deutsches Textarchiv*.[14,15] The test corpus was comprised of the full body text from 13 volumes published between 1780 and 1880, and contained 152,776 tokens of 17,417 distinct types in 9,079 sentences, discounting non-alphabetic types such as punctuation. To assign an extant canonical equivalent to each token of the test corpus, the text of each volume was automatically aligned

[14] `http://www.deutschestextarchiv.de`

[15] EDIT: See appendix C.2.1

token-wise with a contemporary edition of the same volume. Automatically discovered non-identity alignment pair types were presented to a human annotator for confirmation. In a second annotation pass, all tokens lacking an identical or manually confirmed alignment target were inspected in context and manually assigned a canonical form. Whenever they were presented to a human annotator, proper names and extinct lexemes were treated as their own canonical forms. In all other cases, equivalence was determined by direct etymological relation of the root in addition to matching morphosyntactic features. Problematic tokens were marked as such and subjected to expert review. Marginalia, front and back matter, speaker and stage directions, and tokenization errors were excluded from the final evaluation corpus.

4.4.2 Evaluation Measures

The canonicalization methods from sections 4.2 and 4.3 were evaluated using the gold-standard test corpus to simulate an information retrieval task Formally, let $C = \{c_1, \ldots, c_{n_C}\}$ be a finite set of canonicalizers, and let $G = \langle g_1, \ldots, g_{n_G} \rangle$ represent the test corpus, where each token g_i is a $(2 + n_C)$-tuple $g_i = \langle w_i, \tilde{w}_i, [w_i]_{c_1}, \ldots, [w_i]_{c_{n_C}} \rangle \in \mathcal{A}^* \times \mathcal{A}^* \times \wp(\mathcal{A}^*)^{n_C}$, for $1 \leq i \leq n_G$. Here, w_i represents the literal token text as appearing in the historical corpus, $\tilde{w}_i$ is its gold-standard canonical cognate, and $[w_i]_{c_j}$ is the set of canonical forms assigned to the token by the canonicalizer c_j, for $1 \leq j \leq n_C$. Let $Q = \bigcup_{i=1}^{n_G} \{\tilde{w}_i\}$ be the set of all canonical cognates represented in the corpus, and define for each canonicalizer $c \in C$ and query string $q \in Q$ the sets $\text{relevant}(q), \text{retrieved}_c(q) \subset \mathbb{N}$ of *relevant* and *retrieved* corpus tokens as:

$$\text{relevant}(q) = \{i \in \mathbb{N} : q = \tilde{w}_i\} \tag{4.19}$$

$$\text{retrieved}_c(q) = \{i \in \mathbb{N} : q \in [w_i]_c\} \tag{4.20}$$

Token-wise precision ($\text{pr}_{\text{tok},c}$) and recall ($\text{rc}_{\text{tok},c}$) for the canonicalizer c can then be defined as:

$$\text{pr}_{\text{tok},c} = \frac{\left|\bigcup_{q \in Q} \text{retrieved}_c(q) \cap \text{relevant}(q)\right|}{\left|\bigcup_{q \in Q} \text{retrieved}_c(q)\right|} \tag{4.21}$$

$$\text{rc}_{\text{tok},c} = \frac{\left|\bigcup_{q \in Q} \text{retrieved}_c(q) \cap \text{relevant}(q)\right|}{\left|\bigcup_{q \in Q} \text{relevant}(q)\right|} \tag{4.22}$$

Type-wise measures $\text{pr}_{\text{typ},c}$ and $\text{rc}_{\text{typ},c}$ are defined analogously, by mapping the token index sets of Equations (4.19) and (4.20) to corpus types before

	% Types			% Tokens		
c	$\mathrm{pr}_{\mathrm{typ}}$	$\mathrm{rc}_{\mathrm{typ}}$	$\mathrm{F}_{\mathrm{typ}}$	$\mathrm{pr}_{\mathrm{tok}}$	$\mathrm{rc}_{\mathrm{tok}}$	$\mathrm{F}_{\mathrm{tok}}$
id	99.0	59.2	74.1	99.8	79.3	88.4
xlit	**99.1**	89.5	94.1	**99.8**	96.8	98.3
pho	97.1	96.1	96.6	91.4	99.2	95.1
rw	97.6	**96.5**	**97.0**	94.3	**99.3**	96.7
hmm	98.6	95.3	96.9	99.7	99.1	**99.4**

Table 4.1: Evaluation data for various canonicalization techniques with respect to the *Deutsches Textarchiv* evaluation subset. The maximum value in each column appears in boldface type.

applying Equations (4.21) and (4.22). I use the unweighted harmonic precision-recall average F (van Rijsbergen, 1979) as a composite measure for both type- and token-wise evaluation modes:

$$\mathrm{F}(pr, rc) = \frac{2 \cdot pr \cdot rc}{pr + rc} \tag{4.23}$$

I follow Charniak et al. (1993) in using *relative error reduction rates* rather than absolute differences when comparing the performance of different canonicalizers. The general form for the (relative) error reduction in evaluation mode x provided by a method c_2 over method c_1 is: $\frac{x_{c_2} - x_{c_1}}{1 - x_{c_1}}$, assuming $0 \leq x_{c_1} \leq x_{c_2} \leq 1$. For example, given the data in Table 4.1, the error reduction in type-wise recall $x = \mathrm{rc}_{\mathrm{typ}}$ provided by $c_2 = \mathrm{rw}$ with respect to $c_1 = \mathrm{xlit}$ is $\frac{\mathrm{rc}_{\mathrm{typ,rw}} - \mathrm{rc}_{\mathrm{typ,xlit}}}{1 - \mathrm{rc}_{\mathrm{typ,xlit}}} = \frac{.965 - .895}{1 - .895} \approx 0.67 = 67\%$.

4.4.3 Results

Evaluation results for the canonicalization techniques described in sections 4.2 and 4.3 with respect to the test corpus are given in Table 4.1 and graphically depicted in Figure 4.2. Immediately apparent from the data is the typical precision–recall trade-off pattern discussed above: conservative conflators such as string identity (id) and transliteration (xlit) have near-perfect precision ($\geq 99\%$ both type- and token-wise), but relatively poor recall. On the other hand, ambitious conflators such as phonetic identity (pho) or the heuristic rewrite transducer (rw) reduce type-wise recall errors by over 66% and token-wise recall errors by over 75% with respect to transliteration, but these recall gains come at the expense of precision.

As hoped, the HMM disambiguator (hmm) presented in section 4.3 does indeed recover a large degree of the precision lost by the ambitious type-wise

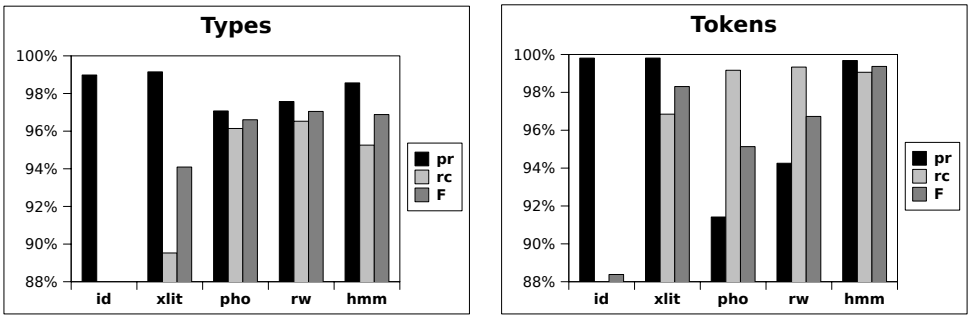

Figure 4.2: Evaluation data for various canonicalization techniques: visualization

conflators, achieving a reduction of over 41% of type-wise precision errors and of over 94% of token-wise precision errors with respect to the heuristic rewrite conflator. While some additional recall errors are made by the HMM, there are comparatively few of these, so that the type-wise harmonic average F falls by a mere 0.1% in absolute magnitude (3% relative error introduction) with respect to the highest-recall method (rw). Indeed, the token-wise composite measure F is substantially higher for the HMM disambiguator (99.4%, vs. 96.7% for the rewrite method), with an error reduction rate of over 64% compared to its closest competitor, deterministic transliteration (xlit).

The most surprising aspect of these results is the recall performance of the conservative transliterator xlit with $\mathrm{rc}_{\mathrm{tok}} = 96.8\%$, reducing token-wise recall errors by over 84% compared to the naïve string identity method. While such performance combined with the ease of implementation and computational efficiency of the transliteration method makes it very attractive at first glance, note that the test corpus was drawn from a comparatively recent text sample, whereas diachronically more heterogeneous corpora have been shown to be less amenable to such simple techniques (Gotscharek et al., 2009c; Jurish, 2010b).

4.5 Conclusion

I have identified a typical precision–recall trade-off pattern exhibited by several type-wise conflation techniques used to automatically discover extant canonical forms for historical German text. Conservative conflators such as string identity and transliteration return very precise results, but fail to associate many historical spelling variants with any appropriate contemporary cognate at all. More ambitious techniques such as conflation by phonetic

form or heuristic rewrite transduction show a marked improvement in recall, but disappointingly poor precision. To address these problems, I proposed a method for disambiguating canonicalization hypotheses at the token level using sentential context to optimize the path probability of candidate canonical forms given the observed historical forms. The disambiguator uses a Hidden Markov Model whose lexical probabilities are dynamically re-computed for every input sentence based on the canonicalization hypotheses returned by a set of subordinated type-wise conflators, the entire canonicalization cascade remaining within the domain of weighted rational transductions.

The proposed disambiguation architecture was evaluated on an information retrieval task over a gold standard corpus of manually confirmed canonicalizations of historical German text drawn from the *Deutsches Textarchiv*. Use of the token-wise disambiguator provided a relative precision error reduction of over 94% with respect to the best recall method, and a relative recall error reduction of over 71% with respect to the most precise method. Overall, the proposed disambiguation method performed best at the token level, achieving a token-wise harmonic precision-recall average $\mathrm{F} = 99.4\%$.

I am interested in verifying these results using larger and less homogeneous corpora than the test corpus used here, as well as extending the techniques described here to other languages and domains. In particular, I am interested in comparing the performance of the manually constructed rewrite transducer used here with a linguistically motivated language-independent conflator (Covington, 1996; Kondrak, 2000) on the one hand, and with conflators induced from a training sample by machine learning techniques (Ristad and Yianilos, 1998; Kempken et al., 2006; Ernst-Gerlach and Fuhr, 2006) on the other. Future work on the disambiguator itself should involve a systematic investigation of the effects of the various model parameters as well as more sophisticated smoothing techniques for handling previously unseen extant types and sparse training data.

Part II

Appendices

"There are more things in heaven and earth, Horatio,
Than are dreamt of in your philosophy."

Shakespeare, *Hamlet*, Act I, Scene V

Appendix A

Finite-State Components

Due to length constraints on the original articles which appear here as chapters 1 through 4, a complete and precise formal specification of the rule-based finite-state components used in the canonicalization experiments was not possible, despite the fact that the relative success or failure of the corresponding canonicalization methods depends in no small part on the concrete definition of just these components. For this reason, I present here formal specifications for all of the rule-based finite-state components used in the preceding chapters. Section A.1 defines the conservative transliteration rules, section A.2 defines the construction of the phonetization transducer, section A.3 defines the extraction of the rewrite target lexicon from the TAGH morphology transducer, section A.4 defines some implicit morphological security heuristics, and section A.5 defines the construction of the heuristic rewrite editor.

As Hamlet's admonition of his friend in the quotation beginning this appendix suggests, there are numerous phenomena which remain uncaptured by any of the rule-based components described here. Improvement might be expected from joint work with a German-language historian e.g. to fine-tune the symbolic portion of the rewrite rules for specific periods or dialects of origin, or to define appropriate exception lexica (Gotscharek et al., 2009c). Broader coverage might be achieved through the use of a language-independent rewrite editor (Kondrak, 2003), while the use of machine-learning techniques (Ristad and Yianilos, 1998) might provide better resolution of elementary rewrite operation costs. Despite their many shortcomings and over-simplifications, the canonicalization components described here represent more than simple proof-of-concept "toy" rule-sets, as demonstrated in the preceding chapters by the degree to which their use improved recall for information retrieval tasks over real historical corpora.

A.1 Transliteration Rules

This section documents the most common character transliteration rules used by the xlit canonicalizer (section 4.2.2), and implicitly used to pre-process input forms for all other non-trivial canonicalization methods discussed above. Input was assumed to be encoded in UTF-8 (Unicode Consortium, 2011). The following table contains transliteration rules for the Unicode Latin-1 Supplement, which is codepoint-identical with the fixed-width 8-bit ISO-8859-1 (Latin-1) encoding. Identity mappings ($\text{xlit}(a) = a$) were omitted from the table.

Table A.1: Transliteration rules

$a \mapsto \text{xlit}(a)$	**Unicode Codepoint**
¡ ↦ !	U+00A1: INVERTED EXCLAMATION MARK
ª ↦ a	U+00AA: FEMININE ORDINAL INDICATOR
« ↦ "	U+00AB: LEFT-POINTING DOUBLE ANGLE QUOTATION MARK
± ↦ +/-	U+00B1: PLUS-MINUS SIGN
² ↦ 2	U+00B2: SUPERSCRIPT TWO
³ ↦ 3	U+00B3: SUPERSCRIPT THREE
¹ ↦ 1	U+00B9: SUPERSCRIPT ONE
º ↦ o	U+00BA: MASCULINE ORDINAL INDICATOR
» ↦ "	U+00BB: RIGHT-POINTING DOUBLE ANGLE QUOTATION MARK
¼ ↦ 1/4	U+00BC: VULGAR FRACTION ONE QUARTER
½ ↦ 1/2	U+00BD: VULGAR FRACTION ONE HALF
¾ ↦ 3/4	U+00BE: VULGAR FRACTION THREE QUARTERS
¿ ↦ ?	U+00BF: INVERTED QUESTION MARK
À ↦ A	U+00C0: CAPITAL LETTER A WITH GRAVE
Á ↦ A	U+00C1: CAPITAL LETTER A WITH ACUTE
Â ↦ A	U+00C2: CAPITAL LETTER A WITH CIRCUMFLEX
Ã ↦ A	U+00C3: CAPITAL LETTER A WITH TILDE
Å ↦ A	U+00C5: CAPITAL LETTER A WITH RING ABOVE
Æ ↦ AE	U+00C6: CAPITAL LETTER AE
Ç ↦ C	U+00C7: CAPITAL LETTER C WITH CEDILLA
È ↦ E	U+00C8: CAPITAL LETTER E WITH GRAVE
É ↦ E	U+00C9: CAPITAL LETTER E WITH ACUTE
Ê ↦ E	U+00CA: CAPITAL LETTER E WITH CIRCUMFLEX
Ë ↦ E	U+00CB: CAPITAL LETTER E WITH DIAERESIS
Ì ↦ I	U+00CC: CAPITAL LETTER I WITH GRAVE
Í ↦ I	U+00CD: CAPITAL LETTER I WITH ACUTE

(continued on following page)

Transliteration rules (continued from previous page)

$a \mapsto \mathrm{xlit}(a)$	**Unicode Codepoint**
Î ↦ EI	U+00CE: CAPITAL LETTER I WITH CIRCUMFLEX
Ï ↦ I	U+00CF: CAPITAL LETTER I WITH DIAERESIS
Đ ↦ D	U+00D0: CAPITAL LETTER ETH
Ñ ↦ N	U+00D1: CAPITAL LETTER N WITH TILDE
Ò ↦ O	U+00D2: CAPITAL LETTER O WITH GRAVE
Ó ↦ O	U+00D3: CAPITAL LETTER O WITH ACUTE
Ô ↦ O	U+00D4: CAPITAL LETTER O WITH CIRCUMFLEX
Õ ↦ O	U+00D5: CAPITAL LETTER O WITH TILDE
× ↦ x	U+00D7: MULTIPLICATION SIGN
Ø ↦ Ö	U+00D8: CAPITAL LETTER O WITH STROKE
Ù ↦ U	U+00D9: CAPITAL LETTER U WITH GRAVE
Ú ↦ U	U+00DA: CAPITAL LETTER U WITH ACUTE
Û ↦ AU	U+00DB: CAPITAL LETTER U WITH CIRCUMFLEX
Ý ↦ Y	U+00DD: CAPITAL LETTER Y WITH ACUTE
Þ ↦ TH	U+00DE: CAPITAL LETTER THORN
à ↦ a	U+00E0: SMALL LETTER A WITH GRAVE
á ↦ a	U+00E1: SMALL LETTER A WITH ACUTE
â ↦ a	U+00E2: SMALL LETTER A WITH CIRCUMFLEX
ã ↦ a	U+00E3: SMALL LETTER A WITH TILDE
å ↦ a	U+00E5: SMALL LETTER A WITH RING ABOVE
æ ↦ ae	U+00E6: SMALL LETTER AE
ç ↦ c	U+00E7: SMALL LETTER C WITH CEDILLA
è ↦ e	U+00E8: SMALL LETTER E WITH GRAVE
é ↦ e	U+00E9: SMALL LETTER E WITH ACUTE
ê ↦ e	U+00EA: SMALL LETTER E WITH CIRCUMFLEX
ë ↦ e	U+00EB: SMALL LETTER E WITH DIAERESIS
ì ↦ i	U+00EC: SMALL LETTER I WITH GRAVE
í ↦ i	U+00ED: SMALL LETTER I WITH ACUTE
î ↦ ei	U+00EE: SMALL LETTER I WITH CIRCUMFLEX
ï ↦ i	U+00EF: SMALL LETTER I WITH DIAERESIS
ð ↦ d	U+00F0: SMALL LETTER ETH
ñ ↦ n	U+00F1: SMALL LETTER N WITH TILDE
ò ↦ o	U+00F2: SMALL LETTER O WITH GRAVE
ó ↦ o	U+00F3: SMALL LETTER O WITH ACUTE
ô ↦ o	U+00F4: SMALL LETTER O WITH CIRCUMFLEX
õ ↦ o	U+00F5: SMALL LETTER O WITH TILDE
ø ↦ ö	U+00F8: SMALL LETTER O WITH STROKE

(continued on following page)

Transliteration rules (continued from previous page)

$a \mapsto \text{xlit}(a)$	**Unicode Codepoint**
ù ↦ u	U+00F9: SMALL LETTER U WITH GRAVE
ú ↦ u	U+00FA: SMALL LETTER U WITH ACUTE
û ↦ au	U+00FB: SMALL LETTER U WITH CIRCUMFLEX
ý ↦ y	U+00FD: SMALL LETTER Y WITH ACUTE
þ ↦ th	U+00FE: SMALL LETTER THORN
ÿ ↦ y	U+00FF: SMALL LETTER Y WITH DIAERESIS

Most of these mappings adhere to the following general schema:

- remove those diacritics which do not occur in contemporary German orthography (e.g. by mapping $\grave{A}$, $\acute{A}$, $\hat{A}$, $\tilde{A}$, and $\mathring{A}$ to an unadorned A);
- decompose multi-character ligatures into their component symbols (e.g. by mapping $Æ$ to AE);
- map punctuation and foreign letters to graphical or phonetic approximations, respectively (e.g. $\frac{1}{4} \mapsto$ *1/4*, $Þ \mapsto TH$).

Notable exceptions to the general schema are the two mappings ($\hat{\imath} \mapsto ei$) and ($\hat{u} \mapsto au$) and the corresponding upper-case mappings ($\hat{I} \mapsto EI$) and ($\hat{U} \mapsto AU$). These are diacritic-sensitive non-identity mappings, each of which incorporates a well-known phonetic shift in historical German[1] and the extinction of the circumflex diacritic into a single phonetically motivated character replacement heuristic.

The most frequently used replacement heuristics for historical German are the mappings from the Unicode combining diacritic superscript 'e' at codepoint U+0364 (COMBINING LATIN SMALL LETTER E) to a conventional diaeresis (*Umlaut*), and the mapping from the long 's' character ('ſ') at codepoint U+017F (LATIN SMALL LETTER LONG S) to a conventional round 's':

$a \mapsto \text{xlit}(a)$	**Unicode Codepoints**
$\overset{e}{A} \mapsto$ Ä	U+0041 U+0364 ↦ U+00C4
$\overset{e}{O} \mapsto$ Ö	U+004F U+0364 ↦ U+00D6
$\overset{e}{U} \mapsto$ Ü	U+0055 U+0364 ↦ U+00DC
$\overset{e}{a} \mapsto$ ä	U+0061 U+0364 ↦ U+00E4
$\overset{e}{o} \mapsto$ ö	U+006F U+0364 ↦ U+00F6
$\overset{e}{u} \mapsto$ ü	U+0075 U+0364 ↦ U+00FC
ſ ↦ s	U+017F ↦ U+0073

[1] The diphthongizations [iː] ⇝ [aɪ] and [uː] ⇝ [aʊ], respectively.

While the mappings from superscript 'e' to a conventional *Umlaut* violate the strict interpretation of xlit(·) as a *character* transliteration function (cf. section 4.2.2) if input characters are defined as Unicode codepoints (since simple character replacements cannot account for string-to-character mappings such as that from the codepoint string `[U+0061 U+0364]` '$\overset{e}{a}$' to the single codepoint `[U+00E4]` 'ä'), no conflict need result if the alphabet $\mathcal{A}$ is understood in terms of logical characters, i.e. $\left\{\overset{e}{A}, \overset{e}{O}, \overset{e}{U}, \overset{e}{a}, \overset{e}{o}, \overset{e}{u}\right\} \subset \mathcal{A}$. Since the C library transliterator implementation[2] does not make use of any abstract logical alphabet beyond that defined by the Unicode standard, a deterministic local preprocessor was used to handle these cases in practice.

Otherwise, characters were mapped according to the transliteration tables of the `Text::Unidecode` Perl module[3] whenever such a mapping was defined, e.g. the UTF-8 input string 'Πλάτωνν' was transliterated to 'Platon' using the `Text::Unidecode` tables. All remaining Unicode codepoints were mapped to the empty string.

A.2 Phonetization Rules

This section presents the finite-state implementation of the modified IMS German Festival phonetization module used in chapters 1, 3, and 4.

A.2.1 Pre-processing Filters

A.2.1.1 Transliteration Filter

The transliterator M_{xlit} described in section A.1 was used to pre-process input for the phonetization transducer.

A.2.1.2 Lower-Case Filter

All input to the phonetization transducer was converted to lower-case before being passed into the core phonetization components. Let $\{\mathcal{A}_{\text{uc}}, \mathcal{A}_{\text{lc}}, \mathcal{A}_{\text{nc}}\}$ be a partitioning of the input alphabet $\mathcal{A}$ into upper-case, lower-case, and caseless characters respectively, and let $\text{lc}(a) \in \mathcal{A}_{\text{lc}}$ represent the unique lower-case variant of an upper-case character $a \in \mathcal{A}_{\text{uc}}$. Then, M_{lc} is a deterministic finite-state transducer mapping any upper-case character a to its lower-case

[2] See appendix B.1

[3] `http://search.cpan.org/~sburke/Text-Unidecode-0.04/`

variant $\mathrm{lc}(a)$, and passing other characters through unchanged:[4]

$$M_{\mathrm{lc}} := \mathrm{Id}(\mathcal{A}_{\mathrm{lc}} \cup \mathcal{A}_{\mathrm{nc}}) \cup \{a \mapsto \mathrm{lc}(a) : a \in \mathcal{A}_{\mathrm{uc}}\}$$

A.2.1.3 Word Boundary Insertion

The designated symbol '`#`' was used in the phonetization transducer to represent a word boundary. This symbol was inserted at the beginning and end of input strings by a simple transducer $M_{\#}$.

$$M_{\#} := (\{\varepsilon\} \times \{\texttt{\#}\})(\mathcal{A}\backslash\{\texttt{\#}\})^*(\{\varepsilon\} \times \{\texttt{\#}\})$$

A.2.2 Character Classes

The `festival` LTS rule syntax does not directly support regular operations such as union or difference. Instead, rule-sets may declare a finite number of *named character classes* to represent salient subsets of the input alphabet for use in rule contexts.[5] The IMS German Festival rule-set defined a total 13 character classes. Of these, the following three classes were used in my modifications:

$$\begin{aligned}
\mathcal{A}_{\mathrm{pho}} &= \left\{\begin{matrix}\texttt{a},\texttt{b},\texttt{c},\texttt{d},\texttt{e},\texttt{f},\texttt{g},\texttt{h},\texttt{i},\texttt{j},\texttt{k},\texttt{l},\texttt{m},\texttt{n},\texttt{o},\texttt{p},\texttt{q},\texttt{r},\texttt{s},\texttt{t},\texttt{u},\texttt{v},\texttt{w},\texttt{x},\texttt{y},\texttt{z},\\ \texttt{ä},\texttt{ö},\texttt{ü},\texttt{ß},\texttt{\#},\texttt{??}\end{matrix}\right\} \\
L &= \mathcal{A}_{\mathrm{pho}}\backslash\{\texttt{\#},\texttt{??}\} \\
C &= \{\texttt{b},\texttt{c},\texttt{d},\texttt{f},\texttt{g},\texttt{h},\texttt{j},\texttt{k},\texttt{l},\texttt{m},\texttt{n},\texttt{p},\texttt{q},\texttt{r},\texttt{s},\texttt{t},\texttt{v},\texttt{w},\texttt{x},\texttt{z},\texttt{ß}\} \\
V &= \{\texttt{a},\texttt{e},\texttt{i},\texttt{o},\texttt{u},\texttt{ä},\texttt{ö},\texttt{ü}\}
\end{aligned}$$

These classes appear in italics when they occur in the rules presented below.

A.2.3 Core LTS Rules

According to the strict licensing conditions of the IMS German Festival package (Möhler et al., 2001) from which the phonetization transducer used in the current work was derived (cf. section 1.2), I am forbidden to reproduce the modified phonetization rule-set in its entirety. Instead, I present here a

[4] For efficiency reasons, lower-casing was performed in practice by system calls to a constant-time C library function. The characterization here as a finite-state transducer is meant for expositional purposes only.

[5] As the name suggests, character classes are sets of input symbols rather than strings or other complex expressions, analogous to the UNIX character classes used by programs such as `egrep`.

list of the 157 rules from the original rule-set[6] which were deleted from the simplified phonetization transducer, together with those 19 new rules which were added. The special symbols '`#`' representing a word boundary and '`??`' representing unrecognized input were passed through the core LTS transducer unchanged using a special feature of the LTS rule compiler. For maximal compliance with the IMS German Festival license, target phonetic strings have been replaced by an ellipsis (...) in deleted rules. For ease of reading, deleted rules appear in gray text, while inserted rules appear in boldface black text. Phonetic strings are given in SAMPA[7] notation.

Table A.2: Phonetizer modifications

OP :	α	[	β	]	γ	$\rightarrow$	π
DEL:		[	a u h	]		→	...
DEL:		[	ä ä h	]		→	...
DEL:		[	ä h	]		→	...
DEL:		[	a e a e h	]		→	...
DEL:		[	a e h	]		→	...
DEL:		[	a a h	]		→	...
DEL:		[	b	]	b a r	→	...
DEL:		[	b	]	c h e n	→	...
DEL:		[	b	]	h a f t	→	...
DEL:		[	b	]	h e i t	→	...
DEL:		[	b	]	l e i n	→	...
DEL:		[	b	]	l i n g	→	...
DEL:		[	b	]	l i n g s	→	...
DEL:		[	b	]	l o s	→	...
DEL:		[	b	]	n i s	→	...
DEL:		[	b	]	s a l	→	...
DEL:		[	b	]	s a m	→	...
DEL:		[	b	]	s c h a f t	→	...
DEL:		[	b	]	s e l	→	...
DEL:		[	b	]	t u m	→	...
DEL:		[	b	]	w ä r t s	→	...
DEL:		[	b	]	w a e r t s	→	...
DEL:		[	b b	]	b a r	→	...
DEL:		[	b b	]	c h e n	→	...
DEL:		[	b b	]	h a f t	→	...

(continued on following page)

[6] As contained in the file `festival/lib/german/ims_german_lts.scm` extracted from the distribution archive `ims_german_1.2c-os.tgz` downloaded October 18, 2005 from `http://www.ims.uni-stuttgart.de/phonetik/synthesis/festival_opensource.html`.

[7] `http://www.phon.ucl.ac.uk/home/sampa/home.htm`

Phonetizer modifications (continued from previous page)

OP :	α	[	β	]	γ	$\rightarrow$	π
DEL:		[	b b	]	h e i t	→	...
DEL:		[	b b	]	l e i n	→	...
DEL:		[	b b	]	l i n g	→	...
DEL:		[	b b	]	l i n g s	→	...
DEL:		[	b b	]	l o s	→	...
DEL:		[	b b	]	n i s	→	...
DEL:		[	b b	]	s a l	→	...
DEL:		[	b b	]	s a m	→	...
DEL:		[	b b	]	s c h a f t	→	...
DEL:		[	b b	]	s e l	→	...
DEL:		[	b b	]	t u m	→	...
DEL:		[	b b	]	w ä r t s	→	...
DEL:		[	b b	]	w a e r t s	→	...
DEL:		[	b b	]	*NL*	→	...
INS:	**#**	[	**d a r**	]	*C L*	→	**d a:**
DEL:		[	d d	]	#	→	...
DEL:		[	d d s	]	#	→	...
DEL:		[	d	]	b a r	→	...
DEL:		[	d	]	c h e n	→	...
DEL:		[	d	]	h a f t	→	...
DEL:		[	d	]	h e i t	→	...
DEL:		[	d	]	l e i n	→	...
DEL:		[	d	]	l i n g	→	...
DEL:		[	d	]	l i n g s	→	...
DEL:		[	d	]	l o s	→	...
DEL:		[	d	]	n i s	→	...
DEL:		[	d	]	s a l	→	...
DEL:		[	d	]	s a m	→	...
DEL:		[	d	]	s c h a f t	→	...
DEL:		[	d	]	s e l	→	...
DEL:		[	d	]	t u m	→	...
DEL:		[	d	]	w ä r t s	→	...
DEL:		[	d	]	w a e r t s	→	...
DEL:		[	d d	]	b a r	→	...
DEL:		[	d d	]	c h e n	→	...
DEL:		[	d d	]	h a f t	→	...
DEL:		[	d d	]	h e i t	→	...
DEL:		[	d d	]	l e i n	→	...
DEL:		[	d d	]	l i n g	→	...
DEL:		[	d d	]	l i n g s	→	...

(continued on following page)

Phonetizer modifications (continued from previous page)

OP :	α	[	β	]	γ	$\rightarrow$	π
DEL:		[	d d	]	l o s	→	...
DEL:		[	d d	]	n i s	→	...
DEL:		[	d d	]	s a l	→	...
DEL:		[	d d	]	s a m	→	...
DEL:		[	d d	]	s c h a f t	→	...
DEL:		[	d d	]	s e l	→	...
DEL:		[	d d	]	t u m	→	...
DEL:		[	d d	]	w ä r t s	→	...
DEL:		[	d d	]	w a e r t s	→	...
DEL:		[	d d s	]	t	→	...
DEL:		[	d d	]	NL	→	...
DEL:		[	e i h	]		→	...
DEL:		[	e e h	]		→	...
DEL:		[	g	]	b a r	→	...
DEL:		[	g	]	c h e n	→	...
DEL:		[	g	]	h a f t	→	...
DEL:		[	g	]	h e i t	→	...
DEL:		[	g	]	l e i n	→	...
DEL:		[	g	]	l i n g	→	...
DEL:		[	g	]	l i n g s	→	...
DEL:		[	g	]	l o s	→	...
DEL:		[	g	]	n i s	→	...
DEL:		[	g	]	s a l	→	...
DEL:		[	g	]	s a m	→	...
DEL:		[	g	]	s c h a f t	→	...
DEL:		[	g	]	s e l	→	...
DEL:		[	g	]	t u m	→	...
DEL:		[	g	]	w ä r t s	→	...
DEL:		[	g	]	w a e r t s	→	...
DEL:		[	g g	]	b a r	→	...
DEL:		[	g g	]	c h e n	→	...
DEL:		[	g g	]	h a f t	→	...
DEL:		[	g g	]	h e i t	→	...
DEL:		[	g g	]	l e i n	→	...
DEL:		[	g g	]	l i n g	→	...
DEL:		[	g g	]	l i n g s	→	...
DEL:		[	g g	]	l o s	→	...
DEL:		[	g g	]	n i s	→	...
DEL:		[	g g	]	s a l	→	...
DEL:		[	g g	]	s a m	→	...

(continued on following page)

Phonetizer modifications (continued from previous page)

OP :	α	[	β	]	γ	→	π
DEL:		[	g g	]	s c h a f t	→	...
DEL:		[	g g	]	s e l	→	...
DEL:		[	g g	]	t u m	→	...
DEL:		[	g g	]	w ä r t s	→	...
DEL:		[	g g	]	w a e r t s	→	...
DEL:		[	g g	]	*Cu*	→	...
DEL:		[	g g	]	*NL*	→	...
DEL:	#	[	h	]		→	...
DEL:		[	h	]	a u	→	...
DEL:		[	h	]	a f t	→	...
DEL:		[	h	]	e i t	→	...
DEL:		[	h	]	a f t i g	→	...
DEL:		[	h	]	a n g	→	...
DEL:	e r	[	h	]	*V*	→	...
DEL:	e l	[	h	]	*V*	→	...
DEL:	g e	[	h	]		→	...
DEL:	z u	[	h	]		→	...
DEL:	a n	[	h	]		→	...
DEL:	a b	[	h	]		→	...
DEL:	h i n	[	h	]		→	...
DEL:	v e r	[	h	]		→	...
DEL:		[	h	]		→	...
INS:		[	**h**	]	*C*	→	
INS:		[	**h**	]	**#**	→	
INS:		[	**h**	]		→	**h**
DEL:		[	i e h	]		→	...
DEL:		[	i h	]		→	...
INS:		[	**i u w**	]		→	**OY**
INS:		[	**i u**	]		→	**OY**
INS:		[	**i w**	]		→	**OY**
INS:		[	**m p**	]		→	**m**
INS:		[	**m b**	]		→	**m**
DEL:		[	ö ö h	]		→	...
DEL:		[	ö h	]		→	...
DEL:		[	o e o e h	]		→	...
DEL:		[	o e o e	]		→	...
DEL:		[	o e h	]		→	...
DEL:		[	o o h	]		→	...
INS:		[	**o u w**	]		→	**aU**
INS:		[	**o u**	]		→	**aU**

(continued on following page)

Phonetizer modifications (continued from previous page)

OP :	α	[β]	γ	$\rightarrow$	π
INS:		[**o w**]		→	**aU**
DEL:		[r h]		→	...
DEL:		[r r h]		→	...
INS:	**#**	[**s**]	**l**	→	**S**
INS:	**#**	[**s**]	**m**	→	**S**
INS:	**#**	[**s**]	**w**	→	**S**
INS:	**#**	[**s**]	**n**	→	**S**
INS:	**#**	[**s**]	**r**	→	**S**
DEL:		[ü ü h]		→	...
DEL:		[ü h]		→	...
DEL:		[u e u e h]		→	...
DEL:		[u e h]		→	...
DEL:		[u u h]		→	...
INS:		[**u o**]		→	**u:**
INS:		[**u a**]		→	**u:**
DEL:		[w]	b a r	→	...
DEL:		[w]	c h e n	→	...
DEL:		[w]	h a f t	→	...
DEL:		[w]	h e i t	→	...
DEL:		[w]	l e i n	→	...
DEL:		[w]	l i n g	→	...
DEL:		[w]	l i n g s	→	...
DEL:		[w]	l o s	→	...
DEL:		[w]	n i s	→	...
DEL:		[w]	s a l	→	...
DEL:		[w]	s a m	→	...
DEL:		[w]	s c h a f t	→	...
DEL:		[w]	s e l	→	...
DEL:		[w]	t u m	→	...
DEL:		[w]	w ä r t s	→	...
DEL:		[w]	w a e r t s	→	...

Most of the deleted rules were approximations of morpheme-final allophony phenomena which used a short list of common suffix surface forms in lieu of a full morphological decomposition of the input forms. Since the suffix list is by no means complete, particularly with regard to historical variants, such specialized rules can be expected to harm rather than help a phonetic conflator for historical input.

The next largest category of deleted rules are those which define mappings for (or sensitive to) the letter *h*. These were replaced by three simple rules

which define *h* as silent before a consonant or word boundary, and otherwise map it to a glottal fricative. The most typical case of historical variation with respect to *h* in German – namely, the historical pattern ($th \rightsquigarrow t$) – was handled by an original IMS German Festival rule (R133: `[ t h ]` → `t`).

In a few cases, specialized rules were added to the LTS rule-set to account for observed historical variation phenomena. In particular, the rules inserted for *dar, iuw, iu, mp, mb, ouw, ou, ow, sl, sm, sw, sn, sr, uo* and *ua* do not properly represent either contemporary or historical phonological rules. Rather, these rules are specifically designed for canonicalization of historical German input, mapping historical grapheme substrings unlikely to occur in contemporary words to plausible contemporary phonetic realizations, similar to the transliteration mapping ($\hat{\imath} \mapsto ei$) discussed above in section A.1.

A.2.4 Post-processing Filters

Several high-level phonetization operations in the IMS German Festival package are realized by post-processing SCHEME functions. For the phonetization transducer used in the current work, I implemented similar functions as a finite-state transducer which was composed onto the upper (phonetic) tape of the core LTS transducer compiled from the raw `festival` LTS rules by the construction given in section 1.2.1. Unlike the core LTS transducer, the post-processing FST was compiled using a more traditional SPE-style two-level rule compiler (Mohri and Sproat, 1996) for obligatory rules with feeding and bleeding. The rules given below are presented in `lextools`[8] syntax. Sections A.2.4.1 through A.2.4.4 describe the finite-state implementations of the IMS German Festival post-processing routines, and sections A.2.4.5 through A.2.4.9 present additional post-processing rules applied to the modified phonetizer used in the current work.

A.2.4.1 Affricates

The following rules were applied to join plosive-fricative substrings into the appropriate affricates:

`p f` → `[pf]` / __
`t s` → `[ts]` / __
`t S` → `[tS]` / __
`t Z` → `[tZ]` / __

[8] `http://www.research.att.com/sw/tools/lextools/`

A.2.4.2 Post-Vocalic R

The following rules were applied to alter the pronunciation of post-vocalic 'r' from a uvular fricative ('R') to central near-open vowel ('6'):

```
R → 6 /    a __
R → 6 /    E __
R → 6 /    I __
R → 6 /    O __
R → 6 /    U __
R → 6 /    Y __
R → 6 /    9 __
R → 6 / [2:] __
R → 6 / [E:] __
R → 6 / [OY] __
R → 6 / [a:] __
R → 6 / [aI] __
R → 6 / [aU] __
R → 6 / [e:] __
R → 6 / [eI] __
R → 6 / [o:] __
R → 6 / [u:] __
R → 6 / [y:] __
```

A.2.4.3 Palatal vs. Velar Fricatives

A simple vowel-sensitive rule was applied to implement the allophony of palatal ('C') and velar ('x') fricatives depending on the immediately preceding vowel:

```
C → x / ([a:]|a|O|[u:]|U|[aU]) __
```

A.2.4.4 Glottal Stop

A simple rule inserted a glottal stop ('?') for words beginning with a vowel. Another rule inserted a glottal stop after common prefixes, as an approximation of a more abstract phonological rule which inserts glottal stops between morpheme boundaries.[9]

[9]The square brackets around the symbol '?' are required here by `lextools` syntax in order to distinguish the SAMPA symbol for a glottal stop '?' from the optionality operator '*?*'.

$\varepsilon \rightarrow$ [?] / # __ V
$\varepsilon \rightarrow$ [?] / #(Ent|Um|[aU]s|dIs|dEs|Un|mIs|fER|In|g@|k[o:]|b@|ap) __ V

The phonetizer might be additionally simplified and the general goal of a *phonemic* representation (as opposed to a synthesis-oriented *phonetic* representation such as the IMS German Festival system provides) more closely approximated by removal of some or all of the preceding post-processing rules, but preliminary experiments showed that for the phonetic conflator on its own, removing the above rules did not have any consequences. In an early attempt (chapter 1) to broaden the phonetic conflator's coverage, the following post-processing rules were developed to approach a more abstract representation.

A.2.4.5 Unrecognized Input

Any unrecognized input (represented by the symbol '??') was mapped to a schwa ('@'):

[??] $\rightarrow$ @ / __

A.2.4.6 Schwa Removal

Schwa ('@') was mapped to a close-mid front unrounded vowel ('e') whenever it occurred:

[@] $\rightarrow$ e / __

A.2.4.7 Vowel Shortening

Vowel length estimates from the LTS rule-set were discarded. Manual inspection of later evaluation data (chapters 3 and 4) suggests that these rules in particular are responsible for many of the phonetic conflator's precision

errors.

$$\texttt{[2:]} \rightarrow \texttt{2} \;/\; \_\_$$
$$\texttt{[a:]} \rightarrow \texttt{a} \;/\; \_\_$$
$$\texttt{[A:]} \rightarrow \texttt{A} \;/\; \_\_$$
$$\texttt{[e:]} \rightarrow \texttt{e} \;/\; \_\_$$
$$\texttt{[E:]} \rightarrow \texttt{E} \;/\; \_\_$$
$$\texttt{[i:]} \rightarrow \texttt{i} \;/\; \_\_$$
$$\texttt{[I:]} \rightarrow \texttt{I} \;/\; \_\_$$
$$\texttt{[o:]} \rightarrow \texttt{o} \;/\; \_\_$$
$$\texttt{[O:]} \rightarrow \texttt{O} \;/\; \_\_$$
$$\texttt{[u:]} \rightarrow \texttt{u} \;/\; \_\_$$
$$\texttt{[U:]} \rightarrow \texttt{U} \;/\; \_\_$$
$$\texttt{[y:]} \rightarrow \texttt{y} \;/\; \_\_$$
$$\texttt{[Y:]} \rightarrow \texttt{y} \;/\; \_\_$$

A.2.4.8 Redundancy Trimming

Repeated occurrences of the same phone were deleted. For each phone p, a rule was applied of the form:

$$pp^* \rightarrow p \;/\; \_\_$$

A.2.4.9 Word Boundary Removal

Finally, the word boundary symbol '`#`' was removed from the output string by application of the single obligatory rule:

$$\texttt{[\#]} \rightarrow \varepsilon \;/\; \_\_$$

A.2.5 Phonetization FST

The final phonetization FST M_{pho} was defined by composition of the pre-processing filters, the core LTS rule transducer and the post-processing filters. If $M_{\text{pho}_{\text{CORE}}}$ represents the core LTS rule transducer compiled from the modified LTS rule-set of section A.2.3 and F_{pho} represents the transducer compiled from all post-processing filters described in section A.2.4, then the final phonetization FST M_{pho} can be defined as:

$$M_{\text{pho}} := M_{\text{xlit}} \circ M_{\text{lc}} \circ M_{\#} \circ M_{\text{pho}_{\text{CORE}}} \circ F_{\text{pho}}$$

A.3 TAGH Filters

In chapters 3 and 4, it was noted that the lexicon Lex used to define the Levenshtein and heuristic rewrite conflation cascades was extracted from the input language of the TAGH morphology transducer (Geyken and Hanneforth, 2006).[10] This section documents the target lexicon extraction process in terms of weighted finite-state filters applied to the upper (abstract) tape of the raw morphology transducer M_{TAGH}. The filters were constructed as weighted acceptors (identity transducers), and followed the general strategy of assigning a cost penalty to TAGH constructs deemed "unsafe" for use in a rewrite cascade target lexicon. Together with an appropriate cost upper bound parameter to the cascade search function (see section 2.3.4), these penalties effectively prevent such constructs from being considered valid rewrite targets.

The filters presented in this section were developed in the course of the work described above by manual inspection of proposed rewrite canonicalizations. In most cases, spurious canonicalizations indicated errors in the rewrite transducer itself. In other cases, errors were due to the presence of a "false friend" in the target lexicon. Occasionally, modifications were made to both the rewrite transducer and the target lexicon. The penalization rules presented below should not be construed as comprising a complete list of dangerous target constructs for a rewrite cascade; rather, they represent a sample of the most frequent such constructs encountered in the course of this work. Nonetheless, these filters demonstrate what I consider one of the major benefits of the finite-state canonicalization approach when applied to a high quality lexicon such as TAGH provides; namely the ability to concisely and accurately express linguistically salient constraints in terms of linguistically meaningful criteria such as morpheme boundaries, word category, lexical stem category, syntactic features, *etc.*, rather than being restricted to simple string operations on raw surface forms.

A.3.1 Syntactic Category Filters

The following penalties were assigned based on the STTS part of speech tag (Schiller et al., 1995) assigned by TAGH to the input word on its lower tape:

[10] The coverage data in chapter 1 was acquired using TAGH version 1.1.4. All other mentions of the TAGH morphology refer to TAGH version 2.0.1.

STTS Tag	Penalty	Description
`$.`	100	period
`$,`	100	comma
`$(`	100	other punctuation
`CARD`	100	cardinal number or other digit string
`FM`	100	foreign language material
`ITJ`	100	interjection
`NE`	100	proper name
`XY`	100	abbreviation or non-linguistic material

Similar penalties were applied to lexical stems of the corresponding categories occurring within a complex derivation:

TAGH Code	Penalty	Description
`/NE`	100	proper name (backwards-compatible)
`/PN`	100	given name
`/LN`	100	surname
`/GN`	100	geographic name
`/ON`	100	organization name
`/X`	100	abbreviation or acronym

In most cases it should be clear that the surface forms associated with these tags are dangerous if allowed as potential outputs of a rewrite cascade, particularly if the cascade editor is insensitive to its operands and/or their context, as in the case of a Levenshtein editor. Regarding the exclusion of proper names from the target lexicon, note that many historical variants of extant German lexemes – in particular adjectives and common nouns – still survive as surnames or geographic names, for example (*Aehnlich*/`NE.lastname` ∼ *ähnlich*/`ADJD`), (*Schmidt*/`NE.lastname` ∼ *Schmied*/`NN`), or (*Thür*/`NE.geoname` ∼ *Tür*/`NN`). If these analyses were allowed into the rewrite target lexicon without any penalty, their presence would prevent the rewrite and Levenshtein canonicalization cascades from discovering the correct extant equivalent (*ähnlich*, *Schmied*, and *Tür* in the above examples), since in both cases identity transductions are always cost-minimal.

A.3.2 Lexical Stem Filters

The following table documents the lexical stems which were penalized whenever they occurred in complex derivations. Here, a single dot (.) represents an arbitrary morpheme- or word-boundary, and the dollar sign ($) represents a word-boundary. Since the TAGH transducer differentiates between various types of morpheme boundaries, the actual regular expressions used were in

fact somewhat more complex then the schematic patterns presented below. The simple dot-and-dollar notation is used here for purposes of clarity.

Table A.3: TAGH Lexical stem filters

TAGH Stem	**Penalty**	**Example**
.äs/V.	20	*(*armîsen* ↦ *Arm.äs.en*)
.Au/N.	10	*(*rîchiu* ↦ *Reich.Au*)
.Bus/N.	10	*(*absentibus* ↦ *Absent.Ei.Bus*)
.dau/V.	20	*(*dauszen* ↦ *Dau.Szene*)
.Ehe/N.	50	*(*gangen* ↦ *Gang.Ehen*)
.Eid/N.	50	*(*Rächerkleide* ↦ *räch.ekl.Eid.e*)
.Ei/N.	50	*(*Hiebei* ↦ *Hieb.Ei*)
.Eis/N.	50	*(*abdominalis* ↦ *abdominal.Eis*)
.eng/A.	20	*(*rachtung* ↦ *Rache.tu.eng*)
.Enge/N.	10	*(*schm$\mathring{e}$egen* ↦ *Schmu.Enge.n*)
.Ente/N.	10	*(*Tausent* ↦ *Tau.s.Ente*)
.gay/A.	50	*(*papagays* ↦ *Papa.gay.s*)
.geh/V.	20	*(*gelung* ↦ *geh.Lunge*)
.gei/V.	20	*(*gyt* ↦ *gei.t*)
.gell/V.	20	*(*gelung* ↦ *gell.ung*)
.Gel/N.	50	*(*blutge* ↦ *Blut.Gel*)
.gel/V.	20	*(*gelung* ↦ *gel.ung*)
.Gen/N.	50	*(*giengen* ↦ *gien.Gen*)
.Gens/N.	50	*(*g$\mathring{u}$nstiger* ↦ *Gens.Tiger*)
.Heu/N.	10	*(*rîchiu* ↦ *reiz.Heu*)
.iah/V.	20	*(*jaren* ↦ *iah.en*)
.Ich/N.	50	*(*küniges* ↦ *kühn.Ich.s*)
.Ire/N.	50	*(*anticipiren* ↦ *anti.giep.Ire.n*)
.Lehn/N$	10	*(*versamlen* ↦ *versam.Lehn*)
.Leiche/N$	10	*(*etslîche* ↦ *ätsch.Leiche*)
.Lunge/N$	50	*(*gelung* ↦ *geh.Lunge*)
.Mahl/N.	10	*(*schmahl* ↦ *Schi.Mahl*)
.Neige/N.	10	*(*mähnige* ↦ *mäh.Neige*)
.Öl/N.	10	*(*herrligkeit* ↦ *Herr.Öl.ig.keit*)
.öl/V.	20	*(*betöbern* ↦ *bet.öl.er.n*)
.penn/V.	20	*(*staupenschlag* ↦ *stau.penn.Schlag*)
.Po/N.	10	*(*posz* ↦ *Po.s*)
.Reh/N.	10	*(*innrer* ↦ *Inn.Reh*)
.Reis/N.	10	*(*hausereys* ↦ *haus.e.Reis*)
.Riege/N$	10	*(*rostrig* ↦ *Rost.Riege*)

(continued on following page)

Lexical stem filters (continued from previous page)

TAGH Stem	**Penalty**	**Example**
`.Schi/N.`	50	*(*$schmahl \mapsto Schi.Mahl$*)
`.Szene/N.`	10	*(*$dauszen \mapsto Dau.Szene$*)
`.Tee/N.`	50	*(*$Schwerte \mapsto schwer.Tee$*)
`.Teig/N.`	10	*(*$abschlächtig \mapsto Abschlag.Teig$*)
`.Term/N$`	10	*(*$sonderm \mapsto sonn.Term$*)
`.te/V.`	20	*(*$teüfels \mapsto te.Fels$*)
`.tu/V.`	20	*(*$rachtung \mapsto Rache.tu.eng$*)
`.Zen/N.`	50	*(*$aufspreizzen \mapsto auf.spreiz.Zen$*)

Finally, the following stems were penalized in both mono- and multimorphemic words:

TAGH Stem	**Penalty**	**Description**
`.-.`	100	hypenated compound
`.aal.mehlig/A.`	100	unlikely derivation
`.alm.mehlig/A.`	100	unlikely derivation
`.alm.ehig/A.`	100	unlikely derivation
`$helf.en/VVFIN.subjII`	20	unlikely form *(hülfe)*
`$geheimniss.en/VVIMP`	20	unlikely form
`.Proceß/N.`	100	extinct variant
`.Vortheil/N.`	100	extinct variant
`.Teen/N.`	100	neologism

A.3.3 Target Lexica

The literal filter patterns listed above were compiled into a weighted acceptor F. A weighted identity transducer M_F was compiled from the finite pattern inventory in F by setting:

$$M_F \quad := \quad \mathrm{Id}\left(\left((\overline{\mathcal{A}^* F \mathcal{A}^*})F\right)^*(\overline{\mathcal{A}^* F \mathcal{A}^*})\right)$$

The resulting identity transducer was then composed with the raw morphology transducer M_{TAGH} to produce the basic target lexicon A_{Lex}:

$$A_{\text{Lex}} \quad := \quad \mathrm{Proj1}(M_{\text{TAGH}} \circ M_F)$$

Finally, an editor-dependent linear scaling factor $z \in \mathbb{R}$ was applied to the weights of A_{Lex} to make TAGH's heuristic weighting function compatible with

that of the rewrite editor. Let $A_{\mathrm{Lex}\,/z}$ be that acceptor which comes from A_{Lex} by dividing[11] each weight in A_{Lex} by the constant z. The target lexica for the Levenshtein and heuristic rewrite editor cascades were then computed as:

$$\begin{aligned} A_{\mathrm{Lex}_{\mathrm{Lev}}} &:= A_{\mathrm{Lex}\,/20} \\ A_{\mathrm{Lex}_{\mathrm{rw}}} &:= A_{\mathrm{Lex}\,/2} \end{aligned}$$

The choice of scaling factors can be understood by relating the edit costs assigned by the corresponding editor transducers to the rule-based derivational complexity weighting scheme of the TAGH transducer itself, which assigns e.g. a cost of 10 to strong morpheme boundaries (`#`) such as between noun stems in a multimorphemic compound. Such boundaries will be treated by the Levenshtein cascade as if they had a cost of $\frac{1}{2}$. Since for a Levenshtein editor, all non-identity transductions have a fixed cost of 1, a compound target word with two or more stems (e.g. *rächekleid : räch/V#ekl/A#Eid/N*) may cause a less expensive non-identity path to become available, if that path leads to a sufficiently simpler derivation (e.g. *rächerkleid : Räch~er/N#Kleid/N*, which is not simple enough to override the identity path, since it still contains one strong morpheme boundary). For the heuristic rewrite cascade, strong morpheme boundaries are assigned a cost of 5, putting them on a par with transductions such as $(c \rightarrow z)$, $(d \rightarrow t)$, or $(iu \rightarrow ie)$, but notably more costly than common transductions such as $(th \rightarrow t)$ or $(ey \rightarrow ei)$ and less costly than uncommon transductions like $(f \rightarrow s)$ or $(a \rightarrow o)$.[12]

A.4 Morphological Security

It was noted in chapter 3 that for the Levenshtein-distance conflation relation, each extant word type is its own "best" equivalent. More generally, any reflexive conflation relation has the property that an extant form can serve as its own canonicalization. Even for the general case of a weighted rewrite transducer, if null-cost identity transductions are included (as they were in the heuristic rewrite transducer used in the current work; see appendix A.5.2), then the cost-minimal path for any valid extant input word will be

[11] I refer here to traditional division of weights regarded as real numbers, rather than to any operation specific to the semiring over which A_{Lex} is defined. Assumedly, the scaling arithmetic could be carried out on-the-fly by a suitable specialized semiring, but the current off-line procedure eliminates both unnecessary complexity and run-time overhead.

[12] See appendix A.5 for a complete specification of the heuristic rewrite transducer.

the word itself.[13] In historical context however, some of these strings are more likely to represent historical variants of non-identical forms than the homographic extant forms. Common cases for such "false friends" are proper names, abbreviations, and acronyms, as described above in section A.3.

For reasons of runtime efficiency, the implemented system did not attempt to canonicalize valid modern forms which were considered "safe", implicitly treating such words as their own canonical forms. An input string was considered "safe" if it contained only non-alphabetic characters (punctuation etc.), or if it was a word covered by the TAGH morphology with at least one "safe" analysis. Safety of morphological analyses was defined using the following heuristic patterns to detect unsafe analyses:[14]

```
$FM                 .Ei/N.          .öl/V.
$XY                 .Eis/N.         .penn/V.
$ITJ                .gell/V.        .Reh/N.
$NE.lastname        .Gel/N.         .Szene/N.
$NE.orgname         .gel/V.         .Tee/N.
$NE.productname     .Gen/N.         .Teig/N.
.äs/V.              .Heu/N.         .te/V.
.Bus/N.             .Loo/NE.        .Thür/NE.
.dau/V.             .Öl/N.          .Zen/N.
```

If M_{TAGH} is the raw morphology transducer and U represents the unweighted acceptor for unsafe morphological analyses, then a morphology variant $M_{\text{TAGH}_{\text{safe}}}$ which produces only "safe" analyses can be defined as:

$$M_{\text{TAGH}_{\text{safe}}} := M_{\text{TAGH}} \circ \text{Id}\left(\overline{\mathcal{A}^* U \mathcal{A}^*}\right)$$

In practice, the morphological security heuristics were implemented as Perl regular expressions operating on the string representations of morphological analysis paths, rather than compiled into the morphology transducer itself. This choice was made based on the desire to retain the original TAGH analyses whenever available on the one hand and to minimize redundant computational effort on the other. In a pure finite-state architecture, the functionality of the morphological safety heuristics could be reduced entirely to target lexicon filters.

[13] This claim assumes that no alternative path involving non-identity transductions leads to an extant form with substantially reduced morphological complexity, as discussed above in appendix A.3.3.

[14] The morphological safety patterns are presented in the same notation as that used to specify the TAGH filters in section A.3.

A.5 Heuristic Rewrite Rules

This appendix contains the heuristic rule-set used to implement the editor WFST for the rewrite cascades described in chapters 3 and 4. The core rewrite editor $M_{\mathrm{rw}_{\mathrm{CORE}}}$ was compiled with a weighted non-deterministic variant of the compiler described above for `festival` LTS rule-sets using the basic construction given in section 1.2.1, where the compiler's output filter M_O was modified to allow multiple weighted outputs. The weights assigned to the rewrite rules were interpreted as (non-negative) costs by the tropical semiring (Simon, 1987). Sections A.5.1 through A.5.7 present the core heuristic rules, sorted by the phonological properties of the operands.[15] The final rewrite editor M_{rw} was constructed by composing specialized filters onto each side of the core transducer compiled from the basic rule-set. The filters and their roles in the construction of M_{rw} are described in section A.5.8.

A.5.1 Character Classes

The following character classes were defined and used to facilitate definition of rule contexts:

$$
\begin{aligned}
\mathcal{A}_{\mathrm{rw}} &= \left\{ \begin{matrix} \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}, \mathtt{h}, \mathtt{i}, \mathtt{j}, \mathtt{k}, \mathtt{l}, \mathtt{m}, \mathtt{n}, \mathtt{o}, \mathtt{p}, \mathtt{q}, \mathtt{r}, \mathtt{s}, \mathtt{t}, \mathtt{u}, \mathtt{v}, \mathtt{w}, \mathtt{x}, \mathtt{y}, \mathtt{z}, \\ \mathtt{ä}, \mathtt{ö}, \mathtt{ü}, \mathtt{ß}, \mathtt{'}, \mathtt{\#}, \mathtt{??} \end{matrix} \right\} \\
L &= \mathcal{A}_{\mathrm{rw}} \backslash \{ \mathtt{'}, \mathtt{\#}, \mathtt{[??]} \} \\
C &:= \{ \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{f}, \mathtt{g}, \mathtt{h}, \mathtt{j}, \mathtt{k}, \mathtt{l}, \mathtt{m}, \mathtt{n}, \mathtt{p}, \mathtt{q}, \mathtt{r}, \mathtt{s}, \mathtt{t}, \mathtt{v}, \mathtt{w}, \mathtt{x}, \mathtt{z} \} \\
OBST &:= \{ \mathtt{c}, \mathtt{d}, \mathtt{f}, \mathtt{g}, \mathtt{h}, \mathtt{k}, \mathtt{p}, \mathtt{q}, \mathtt{s}, \mathtt{t}, \mathtt{v}, \mathtt{x}, \mathtt{z}, \mathtt{ß}, \mathtt{\#} \} \\
V &:= \{ \mathtt{a}, \mathtt{e}, \mathtt{i}, \mathtt{o}, \mathtt{u}, \mathtt{ä}, \mathtt{ö}, \mathtt{ü} \}
\end{aligned}
$$

For implementation-specific reasons, the rule-set definition file contained a total of 49 character class definitions. For the most part, these classes represented simple disjunctions or complements with respect to the input alphabet $\mathcal{A}_{\mathrm{rw}}$ which were defined as character classes only because the LTS rule compiler does not support arbitrary regular expressions. For ease of reading, such superfluous classes have been replaced by more traditional notation using parentheses () and curly brackets {} in the tables below.

[15] The astute reader will note that the categories below do not coincide completely with those used in chapter 3 (Jurish, 2010b). The coarse categories used for the rule counts in chapter 3 were further subdivided for this appendix to facilitate legibility. In the course of the rule-set re-factorization, 4 redundant rules were discovered which were omitted from the rule-set presented here.

A.5.2 Identity Rules

Each valid input character $a \in \mathcal{A}_{\text{rw}}$ was associated with an implicit null-cost identity rule:

$$[a] \rightarrow a \ \langle 0 \rangle$$

A.5.3 Consonant Rules

A.5.3.1 Plosives

In addition to conversion rules between common homophonic allographs such as $(c \leftrightarrow k)$ or $(th \leftrightarrow t)$, the plosive-directed rewrite rules included a number of voicing alternations such as $(\{c, k\} \rightarrow g)$ and $(\{d, dt, t\} \leftrightarrow \{d, dt, t\})$, as realized by the variants *(Egk ∼ Eck)*, *(sanc ∼ sang)*, *(Gewandheit ∼ Gewandtheit)*, and *(Mundt ∼ Mund)*:

Table A.4: Rewrite rules: plosives

α	[β]	γ	$\rightarrow$	π_{rw}	$\langle cost \rangle$
	[b]		→	p	⟨ 8 ⟩
	[b b]		→	p p	⟨ 8 ⟩
	[c]		→	g	⟨ 6 ⟩
	[c]	#	→	g	⟨ 1 ⟩
	[c]	$(\mathcal{A}_{\text{rw}} \backslash \{\text{h}, \text{k}\})$	→	k	⟨ 5 ⟩
	[c k]		→	k	⟨ 1 ⟩
$(\mathcal{A}_{\text{rw}} \backslash \{\text{d}, \text{t}\})$	[d]		→	d t	⟨ 7 ⟩
	[d]		→	t	⟨ 5 ⟩
	[d t]		→	d	⟨ 5 ⟩
	[d t]	#	→	d	⟨ 1 ⟩
	[d t]		→	t	⟨ 1 ⟩
	[g k]		→	c k	⟨ 5 ⟩
	[k]		→	c k	⟨ 1 ⟩
	[k]	#	→	g	⟨ 1 ⟩
	[p]		→	b	⟨ 5 ⟩
	[t]		→	d	⟨ 7 ⟩
	[t]	#	→	d	⟨ 4 ⟩
	[t]		→	d t	⟨ 5 ⟩
	[t]	$(V \cup \{\text{r}\})$	→	t h	⟨ 5 ⟩
	[t]		→	t t	⟨ 8 ⟩
$(\mathcal{A}_{\text{rw}} \backslash \{\text{d}, \text{t}\})$	[t]	$(\mathcal{A}_{\text{rw}} \backslash \{\text{d}, \text{t}\})$	→	t t	⟨ 2 ⟩
	[t h]		→	d	⟨ 7 ⟩
	[t h]		→	t	⟨ 2 ⟩
	[t h]	#	→	t	⟨ 1 ⟩

(continued on following page)

Rewrite rules: plosives (continued from previous page)

α	[β]	γ	$\rightarrow$	π_{rw}	$\langle cost \rangle$
	[t h]	$(V \cup \{\texttt{r}, \texttt{l}, \texttt{w}\})$	$\rightarrow$	t	$\langle$ 1 $\rangle$
	[t t]		$\rightarrow$	t	$\langle$ 2 $\rangle$
	[w]		$\rightarrow$	b	$\langle$10 $\rangle$

A.5.3.2 Fricatives

Fricative rules included workarounds mapping between *s* and *f* for common OCR errors involving long 's' ('f'), as well as a deletion rule for *w* following a back vowel as in *(frauwen ∼ Frauen)*. Location shift rules were included for pre-sonorant *s (swert ∼ Schwert)* and for post-vocalic *h (nähsten ∼ nächsten)*. Fricative-oriented rewrites were otherwise limited to common cases of homophonic allography such as $\{ch, g\}$, $\{ph, f\}$, and $\{s, ss, ß\}$:

Table A.5: Rewrite rules: fricatives

α	[β]	γ	$\rightarrow$	π_{rw}	$\langle cost \rangle$
	[c h]		$\rightarrow$	g	$\langle$10 $\rangle$
	[f]		$\rightarrow$	f f	$\langle$ 4 $\rangle$
	[f]		$\rightarrow$	s	$\langle$10 $\rangle$
	[f f]		$\rightarrow$	f	$\langle$ 4 $\rangle$
	[f f]		$\rightarrow$	s s	$\langle$10 $\rangle$
	[f f]		$\rightarrow$	ß	$\langle$10 $\rangle$
	[g]	$(V \cup \{\#\})$	$\rightarrow$	c h	$\langle$10 $\rangle$
	[g]	e $\{\texttt{n}, \texttt{r}, \texttt{s}, \#\}$	$\rightarrow$	c h	$\langle$10 $\rangle$
r	[h]	$\{\texttt{d}, \texttt{t}\}$	$\rightarrow$		$\langle$ 1 $\rangle$
V	[h]		$\rightarrow$	c h	$\langle$10 $\rangle$
	[p h]		$\rightarrow$	f	$\langle$ 1 $\rangle$
	[r]		$\rightarrow$	r r	$\langle$ 5 $\rangle$
	[s]	#	$\rightarrow$		$\langle$ 6 $\rangle$
$(\mathcal{A}_{\text{rw}} \backslash \{\#\})$	[s]		$\rightarrow$	f	$\langle$10 $\rangle$
	[s]	$\{\texttt{l}, \texttt{r}, \texttt{m}, \texttt{n}, \texttt{w}\}$	$\rightarrow$	s c h	$\langle$ 1 $\rangle$
$(\mathcal{A}_{\text{rw}} \backslash \{\#\})$	[s]		$\rightarrow$	s s	$\langle$ 4 $\rangle$
$(\mathcal{A}_{\text{rw}} \backslash \{\#\})$	[s]	#	$\rightarrow$	s s	$\langle$ 3 $\rangle$
$(\mathcal{A}_{\text{rw}} \backslash \{\#\})$	[s]		$\rightarrow$	ß	$\langle$ 8 $\rangle$
$(\mathcal{A}_{\text{rw}} \backslash \{\#\})$	[s]	#	$\rightarrow$	ß	$\langle$ 5 $\rangle$
#	[s c h]		$\rightarrow$	c h	$\langle$10 $\rangle$
$(\mathcal{A}_{\text{rw}} \backslash \{\#\})$	[s s]		$\rightarrow$	s	$\langle$10 $\rangle$
$(\mathcal{A}_{\text{rw}} \backslash \{\#\})$	[s s]	#	$\rightarrow$	s	$\langle$ 4 $\rangle$
$(\mathcal{A}_{\text{rw}} \backslash \{\#\})$	[s s]		$\rightarrow$	ß	$\langle$ 1 $\rangle$
	[s z]		$\rightarrow$	s	$\langle$ 5 $\rangle$

(continued on following page)

Rewrite rules: fricatives (continued from previous page)

α	[β]	γ	$\rightarrow$	π_{rw}	$\langle cost \rangle$
$(\mathcal{A}_{\text{rw}} \backslash \{\#\})$	[s z]		$\rightarrow$	s s	$\langle$ 1 $\rangle$
$(\mathcal{A}_{\text{rw}} \backslash \{\#\})$	[s z]		$\rightarrow$	ß	$\langle$ 1 $\rangle$
	[v]		$\rightarrow$	f	$\langle$ 5 $\rangle$
{a, o, u}	[w]	e	$\rightarrow$		$\langle$ 1 $\rangle$
	[z]		$\rightarrow$	ß	$\langle$10 $\rangle$
	[z z]		$\rightarrow$	ß	$\langle$10 $\rangle$
	[ß]	ß	$\rightarrow$		$\langle$ 0.1$\rangle$
	[ß]		$\rightarrow$	s	$\langle$10 $\rangle$
	[ß]		$\rightarrow$	s s	$\langle$ 5 $\rangle$

A.5.3.3 Affricates

Affricate-oriented rewrite heuristics were limited to the historical allographs {*c*, *cz*, *sc*, *ts*, *tz*, *z*} as appearing in the conflations *(seciren ∼ sezieren)*, *(selczamen ∼ seltsamen)*, *(transscendenten ∼ transzendenten)*, and *(reitzen ∼ reizen)*:

Table A.6: Rewrite rules: affricates

α	[β]	γ	$\rightarrow$	π_{rw}	$\langle cost \rangle$
	[c]		$\rightarrow$	z	$\langle$ 5 $\rangle$
	[c z]		$\rightarrow$	t s	$\langle$ 5 $\rangle$
	[c z]		$\rightarrow$	t z	$\langle$ 5 $\rangle$
	[c z]		$\rightarrow$	z	$\langle$ 1 $\rangle$
$(\mathcal{A}_{\text{rw}} \backslash \{\#\})$	[s c]		$\rightarrow$	z	$\langle$ 5 $\rangle$
	[t]	z	$\rightarrow$		$\langle$ 5 $\rangle$
	[z]		$\rightarrow$	t z	$\langle$10 $\rangle$
$(\mathcal{A}_{\text{rw}} \backslash \{\text{r}\})$	[z]	$(\mathcal{A}_{\text{rw}} \backslash \{\text{e}\})$	$\rightarrow$	t z	$\langle$ 5 $\rangle$

A.5.3.4 Sonorants

Sonorant-oriented rewrite heuristics included the assimilations ($m\{b, p\} \rightarrow m$) as realized by the conflations *(umbgehen ∼ umgehen)*, *(kompt ∼ kommt)*, and *(nimp ∼ nimm)*. A seldom-invoked deletion rule ($en \rightarrow e$) was used to account for variation in functional morphemes as in *(reimenmacher ∼ Reimemacher)*.

Table A.7: Rewrite rules: sonorant consonants

α	[β]	γ	$\rightarrow$	π_{rw}	$\langle cost \rangle$
m	[b]	*OBST*	$\rightarrow$		⟨ 2 ⟩
m	[b]	{d, t, #}	$\rightarrow$	m	⟨ 1 ⟩
	[l]		$\rightarrow$	l l	⟨ 8 ⟩
	[l l]		$\rightarrow$	l	⟨ 8 ⟩
	[m]		$\rightarrow$	m m	⟨ 5 ⟩
	[m m]		$\rightarrow$	m	⟨ 5 ⟩
e	[n]		$\rightarrow$		⟨10 ⟩
	[n]		$\rightarrow$	n n	⟨ 5 ⟩
	[n n]		$\rightarrow$	n	⟨ 5 ⟩
	[n t]	p f	$\rightarrow$	m	⟨ 5 ⟩
m	[p]		$\rightarrow$		⟨ 5 ⟩
m	[p]	{d, t, #}	$\rightarrow$	m	⟨ 1 ⟩
#	[y]		$\rightarrow$	j	⟨ 1 ⟩

A.5.4 Vowel Rules

A.5.4.1 Front Vowels

Operations targeting front vowels included a deletion rule ($e\# \rightarrow \#$) for word-final e, often invoked for the historical dative $-e$ suffix as in *(korbe ∼ Korb)* and *(schwerte ∼ Schwert)*. Despite its clear motivation and obvious usefulness, this rule and its n-inserting pendant ($e\# \rightarrow en\#$) were assigned quite high costs, due to the continued activity of $-e$ and $-en$ as inflectional suffixes. Vowel shift rules included the unroundings ($iu \rightarrow ie$) and ($\ddot{u} \rightarrow i$), as exemplified by the conflations *(biut ∼ biete)* and *(würken ∼ wirken)*. Height- and backness shifting rules such as ($i \rightarrow e$), ($o \rightarrow \ddot{o}$), and ($u \rightarrow \ddot{u}$) were used to achieve conflations such as *(schricken ∼ schrecken)*, *(ofters ∼ öfters)*, and *(Thur ∼ Tür)*; although the latter two of these rules may be primarily of graphematic rather than phonological origin. Insertion rules for the characters e and i were used to account for conflations such as *(tück ∼ Tücke)* and *(kürbs ∼ Kürbis)*; these rules were also often invoked to account for vowels in the DWB verse corpus which were elided apparently for reasons of metric foot. The following table also includes a generic post-vocalic h-insertion rule[16] ($\varepsilon \rightarrow h$) to account for vowel lengthening conflations such as *(jaren ∼ Jahren)* and *(füren ∼ führen)*, as well as a corresponding deletion rule for vowel shortening conflations such as *(roht ∼ rot)*.

[16] See section A.5.8 for the general mechanics used to define insertion rules for the rewrite editor.

Table A.8: Rewrite rules: front vowels

α	[β]	γ	$\rightarrow$	π_{rw}	$\langle cost \rangle$
$(\mathcal{A}_{\text{rw}} \backslash \{\texttt{e}\})$	[<eps>]	#	$\rightarrow$	e	$\langle$ 5 $\rangle$
C	[<eps>]	$(C \cup \{\#\})$	$\rightarrow$	e	$\langle$10 $\rangle$
V	[<eps>]	$(C \backslash \{\texttt{d}, \texttt{h}, \texttt{j}, \texttt{q}, \texttt{s}, \texttt{w}, \texttt{x}\})$	$\rightarrow$	h	$\langle$ 5 $\rangle$
C	[<eps>]	C	$\rightarrow$	i	$\langle$10 $\rangle$
	[a e]		$\rightarrow$	ä	$\langle$ 1 $\rangle$
	[e]		$\rightarrow$		$\langle$15 $\rangle$
	[e]	#	$\rightarrow$		$\langle$ 9 $\rangle$
	[e]	#	$\rightarrow$	e n	$\langle$10 $\rangle$
	[e]		$\rightarrow$	ä	$\langle$ 5 $\rangle$
	[e e]		$\rightarrow$	e	$\langle$ 8 $\rangle$
	[e e]		$\rightarrow$	e h	$\langle$ 8 $\rangle$
	[e e]		$\rightarrow$	ä h	$\langle$ 8 $\rangle$
	[e h]		$\rightarrow$	e e	$\langle$ 5 $\rangle$
V	[h]	$(\mathcal{A}_{\text{rw}} \backslash V)$	$\rightarrow$		$\langle$ 5 $\rangle$
$(\mathcal{A}_{\text{rw}} \backslash V)$	[i]	$(\mathcal{A}_{\text{rw}} \backslash V)$	$\rightarrow$	e	$\langle$ 5 $\rangle$
$(\mathcal{A}_{\text{rw}} \backslash V)$	[i]	$(\mathcal{A}_{\text{rw}} \backslash V)$	$\rightarrow$	i e	$\langle$ 5 $\rangle$
	[i e]		$\rightarrow$	i	$\langle$ 5 $\rangle$
	[i u]		$\rightarrow$	i e	$\langle$ 5 $\rangle$
#	[j]		$\rightarrow$	i	$\langle$ 1 $\rangle$
	[o]		$\rightarrow$	ö	$\langle$ 9 $\rangle$
	[o e]		$\rightarrow$	ö	$\langle$ 1 $\rangle$
	[u]		$\rightarrow$	ü	$\langle$ 5 $\rangle$
	[u e]		$\rightarrow$	ü	$\langle$ 1 $\rangle$
	[y]		$\rightarrow$	i	$\langle$ 5 $\rangle$
$(\mathcal{A}_{\text{rw}} \backslash V)$	[y]	$(\mathcal{A}_{\text{rw}} \backslash V)$	$\rightarrow$	i e	$\langle$ 5 $\rangle$
	[ä]		$\rightarrow$	e	$\langle$ 5 $\rangle$
	[ö]		$\rightarrow$	ü	$\langle$10 $\rangle$
	[ü]		$\rightarrow$	i	$\langle$ 5 $\rangle$
	[ü]		$\rightarrow$	ö	$\langle$10 $\rangle$
	[ü e]		$\rightarrow$	ü	$\langle$ 1 $\rangle$

A.5.4.2 Back Vowels

Rewrite rules targeting back vowels included the bidirectional shifts $(a \leftrightarrow o)$ and $(o \leftrightarrow u)$, as in *(aufgehaben ∼ aufgehoben)*, *(Toback ∼ Tabak)*, *(rond ∼ rund)*, and *(sunen ∼ sonnen)*. The rule $(v \rightarrow u)$ accounted for variants such as *(vnd ∼ und)* common in very old source material. Also possibly of graphematic origin are the rewrites $(\ddot{u} \rightarrow u)$ and $(\ddot{o} \rightarrow o)$ used in the conflations *(darümb ∼ darum)* and *(kömpt ∼ kommt)*.

Table A.9: Rewrite rules: back vowels

α	[β]	γ	$\rightarrow$	π_{rw}	$\langle cost \rangle$
	[a]		$\rightarrow$	o	⟨10 ⟩
	[a a]	$(\mathcal{A}_{\text{rw}} \backslash (V \cup \{\text{y}\}))$	$\rightarrow$	a	⟨ 5 ⟩
	[o]		$\rightarrow$	a	⟨ 9 ⟩
	[o]		$\rightarrow$	u	⟨15 ⟩
	[o o]		$\rightarrow$	o	⟨ 9 ⟩
	[o u]		$\rightarrow$	o	⟨ 5 ⟩
	[o u]		$\rightarrow$	u	⟨ 5 ⟩
	[u]		$\rightarrow$	a	⟨10 ⟩
	[u]		$\rightarrow$	o	⟨10 ⟩
	[u e]		$\rightarrow$	u	⟨10 ⟩
	[v]		$\rightarrow$	u	⟨10 ⟩
	[w]		$\rightarrow$	u	⟨10 ⟩
	[ä]		$\rightarrow$	a	⟨10 ⟩
	[ö]		$\rightarrow$	o	⟨ 5 ⟩
	[ü]		$\rightarrow$	u	⟨10 ⟩

A.5.4.3 Diphthongs

Diphthong-oriented rules included the diphthongizations ($ee \rightarrow ei$), ($\{iu, u\} \rightarrow au$), ($y \rightarrow ei$), and ($\{iu, öi, öü, ü\} \rightarrow eu$) as occurring in the conflations *(schleer ∼ Schleier)*, *(riuschent ∼ rauschend)*, *(suber ∼ sauber)*, *(lyb ∼ Leib)*, *(liute ∼ Leute)*, *({fröide, vröude} ∼ Freude)*, and *(üch ∼ euch)*. The monophthongization rules ($ei \rightarrow ie$) and ($ua \rightarrow u$) accounted for conflations such as *(phantasei ∼ Phantasie)* and *(sluag ∼ schlug)*, while the diphthong shifts ($\{ou, uw\} \rightarrow au$) accounted for examples such as *(troum ∼ Traum)*, *(gruwen ∼ grauen)*. The rules ($i \rightarrow j$) and ($i \rightarrow je$) were used to implement the historical allographs $\{i, j\}$ as occurring in the conflations *(ie ∼ je)* and *(itzt ∼ jetzt)*.

Table A.10: Rewrite rules: diphthongs

α	[β]	γ	$\rightarrow$	π_{rw}	$\langle cost \rangle$
	[a]	i	$\rightarrow$	e	⟨ 5 ⟩
	[a i]		$\rightarrow$	e i	⟨ 5 ⟩
	[e e]		$\rightarrow$	e i	⟨10 ⟩
	[e i]		$\rightarrow$	i e	⟨ 5 ⟩
	[e w]		$\rightarrow$	e u	⟨ 5 ⟩
	[e y]		$\rightarrow$	e i	⟨ 1 ⟩
	[e ü]		$\rightarrow$	e u	⟨ 1 ⟩
$(\mathcal{A}_{\text{rw}} \backslash V)$	[i]	$(\mathcal{A}_{\text{rw}} \backslash V)$	$\rightarrow$	e i	⟨10 ⟩

(continued on following page)

Rewrite rules: diphthongs (continued from previous page)

α	[β]	γ	$\rightarrow$	π_{rw}	$\langle cost \rangle$
$(\mathcal{A}_{\text{rw}} \backslash V)$	[i]	$(\mathcal{A}_{\text{rw}} \backslash V)$	$\rightarrow$	i e	$\langle$ 5 $\rangle$
$(\mathcal{A}_{\text{rw}} \backslash V)$	[i]	V	$\rightarrow$	j	$\langle$ 5 $\rangle$
$(\mathcal{A}_{\text{rw}} \backslash V)$	[i]	$(\mathcal{A}_{\text{rw}} \backslash V)$	$\rightarrow$	j e	$\langle$15 $\rangle$
	[i u]		$\rightarrow$	a u	$\langle$10 $\rangle$
	[i u]		$\rightarrow$	e u	$\langle$ 1 $\rangle$
	[i w]		$\rightarrow$	e u	$\langle$ 5 $\rangle$
	[o u]		$\rightarrow$	a u	$\langle$ 1 $\rangle$
	[o w]		$\rightarrow$	o u	$\langle$ 1 $\rangle$
	[u]		$\rightarrow$	a u	$\langle$10 $\rangle$
	[u a]		$\rightarrow$	u	$\langle$10 $\rangle$
	[u o]		$\rightarrow$	u	$\langle$ 1 $\rangle$
	[u w]		$\rightarrow$	a u	$\langle$ 5 $\rangle$
a	[w]		$\rightarrow$	u	$\langle$ 5 $\rangle$
	[y]		$\rightarrow$	e i	$\langle$ 5 $\rangle$
	[ö i]		$\rightarrow$	e u	$\langle$ 1 $\rangle$
	[ö u]		$\rightarrow$	e u	$\langle$ 1 $\rangle$
	[ü]		$\rightarrow$	e u	$\langle$ 5 $\rangle$

A.5.5 Explicit Elision Rules

Elisions explicitly marked with an apostrophe were handled by a small number of substitution heuristics accounting for common cases such as *(in's* $\sim$ *ins)*, *(sag'* $\sim$ *sage)*, *(red'st* $\sim$ *redest)*, and *(ew'ge* $\sim$ *ewige)*. Preposition-determiner contractions such as *(auf'm* $\sim$ *auf_dem)* triggered a workaround rule mapping $'m$ to s, thus generating the (literally spurious but morpho-syntactically correct modulo grammatical case) conflation *(auf'm* $\sim$ *aufs)*. The most common remaining explicit elisions targeted the e of the third person neutral pronoun *es*: $'s$ suffixes of some common pronouns were deleted, which – although by no means correct – at least retains one of the two extant word types in the corresponding multi-word tokens. In general, string-internal apostrophes indicating elided word boundaries such as *('s* $\sim$ *_es)* were considered tokenization errors and ignored in the evaluations of chapters 3 and 4.

Table A.11: Rewrite rules: elisions

α	[β]	γ	$\rightarrow$	π_{rw}	$\langle cost \rangle$
	[’]		$\rightarrow$		$\langle$0.9 $\rangle$
	[’]	#	$\rightarrow$		$\langle$0.1 $\rangle$
	[’]	s #	$\rightarrow$		$\langle$0.15$\rangle$

(continued on following page)

Rewrite rules: elisions (continued from previous page)

α	[β]	γ	$\rightarrow$	π_{rw}	$\langle cost \rangle$
	[’]	#	$\rightarrow$	e	$\langle 0.2 \rangle$
	[’]	s t #	$\rightarrow$	e	$\langle 0.8 \rangle$
	[’]	t #	$\rightarrow$	e	$\langle 0.5 \rangle$
	[’]	g	$\rightarrow$	i	$\langle 0.5 \rangle$
	[’ m]	#	$\rightarrow$	s	$\langle 0.2 \rangle$
	[’ n]	#	$\rightarrow$	s	$\langle 0.2 \rangle$
	[’ s]	#	$\rightarrow$		$\langle 0.2 \rangle$
# d u	[’ s]	#	$\rightarrow$		$\langle 0.1 \rangle$
# e r	[’ s]	#	$\rightarrow$		$\langle 0.1 \rangle$
# e u c h	[’ s]	#	$\rightarrow$		$\langle 0.1 \rangle$
# i c h	[’ s]	#	$\rightarrow$		$\langle 0.1 \rangle$
# s i e	[’ s]	#	$\rightarrow$		$\langle 0.1 \rangle$
# w i r	[’ s]	#	$\rightarrow$		$\langle 0.1 \rangle$

A.5.6 Unrecognized Input Rules

30 additional rules for handling unrecognized input in the DTA corpus[17] were added to the rewrite transducer. Unrecognized input was encoded as the special symbol ‘??’ and assumed to represent a single lower-case letter active in contemporary German orthography, i.e. an element of the character class L. A unigram probability distribution $p(L)$ over the character class L was estimated from the TIGER corpus (Brants et al., 2002), and for each of the 30 target letters $l \in L$, a rule was added mapping the symbol ‘??’ to l whose cost was computed as the unit-normalized pointwise Shannon entropy of $p(L = l)$:[18]

$$\texttt{??} \rightarrow l \quad \left\langle \frac{\log p(L = l)}{\sum_{l' \in L} \log p(L = l')} \right\rangle$$

More sophisticated character-level language models can be expected to improve rewrite-based canonicalization of unrecognized input. Since unrecognized input represented less than 0.001% of the corpus data,[19] implementation of such models was deemed unnecessary in the context of the current work.

[17] In the case of the DTA, unrecognized input was usually due to illegibility of the source material due to damage, decay, etc.

[18] i.e. the ratio of the length under an optimal encoding for a message with probability $p(L = l)$ to the sum of the lengths of all messages recognized by the encoding; cf. Shannon (1948); Bell et al. (1990); Cover and Thomas (1991).

[19] Only 174 of 62,649,796 tokens in the *Deutsches Textarchiv* included unrecognized input.

A.5.7 Miscellaneous Rules

The remaining rules are not easily classified with respect to simple phonetic or surface string properties, for the most part operating on syllables or character substrings and/or making strict requirements on the string context in which they may be applied. Typical syllable-level operations include the *e*-deletion rules ($est\# \rightarrow st\#$) and ($et\# \rightarrow t\#$) which accounted for conflations such as *(liebest ∼ liebst)* and *(geliebet ∼ geliebt)*,[20] as well as insertion rules such as ($\#g \rightarrow \#ge$), ($g \rightarrow ig$), and ($n \rightarrow en$) used in conflations such as *(gricht ∼ Gericht)*, *(ewge ∼ ewige)*, and *(verzeihn ∼ verzeihen)*. The transposition rules ($le \rightarrow el$) and ($re \rightarrow er$) accounted for conflations such as *(ufwieglen ∼ aufwiegeln)* and *(trauren ∼ trauern)*.

Morpheme-level operations included the rules ($aller \rightarrow all$), ($dar \rightarrow da$), ($für \rightarrow vor$), ($ism \rightarrow ismus$), and ($san \rightarrow sam$), as used in the conflations *(allerseitig ∼ allseitig)*, *(darmit ∼ damit)*, *(fürnehmen ∼ vornehmen)*, *(Dogmatism ∼ Dogmatismus)*, and *(gleichsan ∼ gleichsam)*.

Some common exceptions to the general rule-set were encoded as rules in their own right, e.g. ($\#eh \rightarrow \#ehe$) for *(ehmals ∼ ehemals)*, ($maur \rightarrow mauer$) for *(kirchhofmaur ∼ Kirchhofmauer)*, ($scharm \rightarrow charm$) for *(scharmant ∼ charmant)*, ($ümb \rightarrow um$) for *(darümb ∼ darum)*, ($niß\# \rightarrow nis\#$) for *(Erlebniß ∼ Erlebnis)*, ($wol \rightarrow wohl$) for *(gleichwol ∼ gleichwohl)*, and ($foder \rightarrow forder$) for *(gefodert ∼ gefordert)*.

Table A.12: Rewrite rules: miscellaneous

α	$[\ \beta\ ]$	γ	$\rightarrow$	π_{rw}	$\langle cost \rangle$
i	[a l]		→	e l l	⟨10 ⟩
	[d a c h t]		→	d e n k	⟨10 ⟩
#	[d a r]	$(\mathcal{A}_{\text{rw}} \backslash V)$	→	d a	⟨ 1 ⟩
	[d i e n s t]		→	d i e n	⟨10 ⟩
g	[e]	{l, n, w}	→		⟨ 5 ⟩
$(\mathcal{A}_{\text{rw}} \backslash \{\text{d}, \text{t}\})$	[e]	s t #	→		⟨ 1 ⟩
$(\mathcal{A}_{\text{rw}} \backslash \{\text{d}, \text{t}\})$	[e]	s t e #	→		⟨ 1 ⟩
$(\mathcal{A}_{\text{rw}} \backslash \{\text{d}, \text{t}\})$	[e]	t #	→		⟨ 1 ⟩
$(\mathcal{A}_{\text{rw}} \backslash \{\text{d}, \text{t}\})$	[e]	t e #	→		⟨ 1 ⟩
# s	[e]	l	→	e e	⟨ 4 ⟩
h i	[e]	$(\mathcal{A}_{\text{rw}} \backslash \{\text{r}\})$	→	e r	⟨ 5 ⟩
# z w e	[e]		→	i	⟨ 5 ⟩
# z	[e]	$(\mathcal{A}_{\text{rw}} \backslash \{\text{r}\})$	→	r	⟨10 ⟩
#	[e n]		→	e n t	⟨10 ⟩

(continued on following page)

[20] *e*-deletions were particularly useful on the DWB verse corpus, where they also served to reverse numerous insertions apparently performed for reasons of metric foot.

Rewrite rules: miscellaneous (continued from previous page)

α	[β]	γ	$\rightarrow$	π_{rw}	$\langle cost \rangle$
	[e n]		$\rightarrow$	e r n	$\langle 15 \rangle$
# a l l	[e r]		$\rightarrow$		$\langle 10 \rangle$
	[e r]	{m, n, s, #}	$\rightarrow$	e r e	$\langle 8 \rangle$
	[e r]	i g	$\rightarrow$	r	$\langle 10 \rangle$
	[e r]		$\rightarrow$	r e	$\langle 20 \rangle$
	[f ü r]		$\rightarrow$	v o r	$\langle 9 \rangle$
# m a n	[g]		$\rightarrow$	c h	$\langle 4 \rangle$
#	[g]	{l, n, r}	$\rightarrow$	g e	$\langle 5 \rangle$
#	[g]	$(\mathcal{A}_{\text{rw}} \backslash \{e, i, l, n, r\})$	$\rightarrow$	g e	$\langle 1 \rangle$
C	[g]		$\rightarrow$	i g	$\langle 5 \rangle$
	[g]	e {n, r, s, #}	$\rightarrow$	i g	$\langle 1 \rangle$
#	[g e]		$\rightarrow$	g e g e	$\langle 15 \rangle$
# e	[h]	$(\mathcal{A}_{\text{rw}} \backslash V)$	$\rightarrow$	h e	$\langle 5 \rangle$
# m a n	[i g]		$\rightarrow$	c h	$\langle 4 \rangle$
	[i g]	e {n, r, s, #}	$\rightarrow$	c h	$\langle 5 \rangle$
	[i g]	k e i t	$\rightarrow$	i c h	$\langle 10 \rangle$
	[i s m]	#	$\rightarrow$	i s m u s	$\langle 5 \rangle$
	[i s m]	s #	$\rightarrow$	i s m u s	$\langle 5 \rangle$
$(\mathcal{A}_{\text{rw}} \backslash \{e\})$	[l]	n #	$\rightarrow$	e l	$\langle 5 \rangle$
	[l e]	{e, n, s, t, #}	$\rightarrow$	e l	$\langle 5 \rangle$
i g	[l i c h]	#	$\rightarrow$		$\langle 1 \rangle$
	[m a u r]		$\rightarrow$	m a u e r	$\langle 1 \rangle$
i	[n]	g #	$\rightarrow$		$\langle 1 \rangle$
$(\mathcal{A}_{\text{rw}} \backslash \{e\})$	[n]		$\rightarrow$	e n	$\langle 5 \rangle$
s a	[n]	#	$\rightarrow$	m	$\langle 10 \rangle$
w	[o]	l	$\rightarrow$	o h	$\langle 0.5 \rangle$
f	[o]	d e r	$\rightarrow$	o r	$\langle 0.5 \rangle$
e	[r]		$\rightarrow$		$\langle 5 \rangle$
a	[r]	#	$\rightarrow$		$\langle 10 \rangle$
a u	[r]		$\rightarrow$	e r	$\langle 10 \rangle$
e i	[r]		$\rightarrow$	e r	$\langle 10 \rangle$
e u	[r]		$\rightarrow$	e r	$\langle 10 \rangle$
ä u	[r]		$\rightarrow$	e r	$\langle 10 \rangle$
	[r]	n #	$\rightarrow$	e r	$\langle 5 \rangle$
$(\mathcal{A}_{\text{rw}} \backslash \{\#\})$	[r e]	{e, n, s, t}	$\rightarrow$	e r	$\langle 5 \rangle$
#	[s]	c h a r m	$\rightarrow$		$\langle 2 \rangle$
C	[s]	t	$\rightarrow$	e s	$\langle 2 \rangle$
{d, t}	[s]	t #	$\rightarrow$	e s	$\langle 1 \rangle$
d	[t]	#	$\rightarrow$	e t	$\langle 1 \rangle$
#	[u]	f	$\rightarrow$	a u	$\langle 2 \rangle$

(continued on following page)

Rewrite rules: miscellaneous (continued from previous page)

α	[	β	]	γ	$\rightarrow$	π_{rw}	$\langle cost \rangle$
#	[	u	]	b e r	$\rightarrow$	ü	⟨ 1 ⟩
ü e	[	z	]		$\rightarrow$	ß	⟨ 1 ⟩
n i	[	ß	]	#	$\rightarrow$	s	⟨ 5 ⟩
	[	ü m b	]		$\rightarrow$	u m	⟨ 5 ⟩

A.5.8 Rewrite Filters

A.5.8.1 Transliteration Filter

The transliterator M_{xlit} described in appendix A.1 was used to pre-process input for the rewrite cascade.

A.5.8.2 Epsilon Insertion Filter

The `festival` LTS rule compiler does not support ε moves on its upper tape, thus no generic insertion rules can be defined in the strict `festival` LTS rule syntax. While this restriction is justifiable in the case of letter-to-sound rules, it presents difficulties for a generic rewrite editor. In order to simulate generic insertion rules, a filter M_{ε} was applied to the lower tape of the core rewrite editor which optionally inserted a single instance of the special symbol `<eps>` at every input string position. Generic insertions were then simulated in the core rewrite rule-set as substitutions on `<eps>`.

$$M_{\varepsilon} := \Big((\{\varepsilon\} \times \{\texttt{<eps>}\}) \cup \mathcal{A}_{\text{rw}} \Big)^*$$

A.5.8.3 Word Boundary Filters

The designated word boundary symbol '`#`' was inserted by the transducer $M_{\#}$ as described above in section A.2.1.3, and a transducer $M_{\not\#}$ was responsible for deleting '`#`' from the editor's upper tape:

$$M_{\not\#} := \Big((\{\texttt{\#}\} \times \{\varepsilon\}) \cup (\mathcal{A} \backslash \{\texttt{\#}\}) \Big)^*$$

A.5.8.4 Graphemic Case Filters

The core rewrite rule-set ignored graphemic case distinctions (upper- vs. lower-case characters). In recent German texts, graphemic case can provide useful hints regarding the canonical cognate: since common nouns and proper names are always upper-cased in contemporary German orthography as well as in

recent historical texts such as the DTA subset used in chapter 4, lower-case input can be considered an indicator that the canonical form is not to be found among words of these syntactic categories. Older source material may not adhere to these graphemic case conventions – indeed, the DWB verse corpus used in chapters 1 and 3 contained exclusively lower-case characters.[21] In order to enable both the case-independent formulation of the core rewrite rule-set and re-use of the TAGH target acceptor which implements the strict graphemic case conventions of contemporary orthography, graphemic case handling was implemented by external filters.

Let $\{\mathcal{A}_{\text{uc}}, \mathcal{A}_{\text{lc}}, \mathcal{A}_{\text{nc}}\}$ and M_{lc} be defined as in section A.2.1.2. Additionally, define the upper-casing transducer M_{uc} as the dual of M_{lc}:

$$M_{\text{uc}} := \text{Id}(\mathcal{A}_{\text{uc}} \cup \mathcal{A}_{\text{nc}}) \cup \{a \mapsto \text{uc}(a) : a \in \mathcal{A}_{\text{lc}}\}$$

where $\text{uc}(a) \in \mathcal{A}_{\text{uc}}$ represents the upper-case variant of the lower-case character $a \in \mathcal{A}_{\text{lc}}$.

For the DWB verse corpus, all input was treated as if it were written in upper-case. Since sentence-initial words are also upper-cased in contemporary German, this strategy only serves to increase the number of possible conflations, allowing improved recall but potentially lowering precision. The forced-case input filter $M_{\text{ul}*}$ was implemented as the unweighted deterministic transducer compiled from the regular expression:

$$M_{\text{ul}*} := \{\varepsilon\} \cup \left(M_{\text{uc}}\ M_{\text{lc}}^*\right)$$

For the DTA corpus which contained case distinctions, pre- and post-processing case filters were applied which used the designated symbols `<UC>` and `<LC>` to mark upper- and lower-case words, respectively. These symbols were passed through the core rewrite transducer unchanged. The case preprocessing filter was defined as the deterministic transducer:

$$M_{\text{case},0} := \{\varepsilon\} \cup \left((M'_{\text{lc}} \cup M'_{\text{uc}})M_{\text{lc}}^*\right)$$

where:

$$\begin{aligned} M'_{\text{lc}} &:= (\{\varepsilon\} \times \{\texttt{<LC>}\})\,\text{Id}(\mathcal{A}_{\text{lc}} \cup \mathcal{A}_{\text{nc}}) \\ M'_{\text{uc}} &:= (\{\varepsilon\} \times \{\texttt{<UC>}\})(\mathcal{A}_{\text{uc}} \circ M_{\text{lc}}) \end{aligned}$$

Changes in graphemic case were allowed by a post-processing filter M_{xc} optionally converting between the case-markers `<LC>` and `<UC>` with cost=1:

$$M_{\text{xc}} := \left((\{\texttt{<LC>}\} \times \{\texttt{<UC>}\}\langle 1\rangle) \cup (\{\texttt{<UC>}\} \times \{\texttt{<LC>}\}\langle 1\rangle) \cup \text{Id}(\mathcal{A})\right)^*$$

[21] This property of the DWB verse corpus is symptomatic of the orthographic ambitions of its original authors, the brothers Jacob and Wilhelm Grimm.

The case-marker flags were applied to the immediately following letter and simultaneously removed from the output string by the output filter $M_{\text{case,1}}$:

$$M_{\text{case,1}} := \Big(\big(\{\texttt{<LC>}\} \times \{\varepsilon\}\big) \cup \Big(\big(\{\texttt{<UC>}\} \times \{\varepsilon\}\big) M_{\text{uc}}\Big) \cup \overline{\{\texttt{<LC>}, \texttt{<UC>}\}}\Big)^*$$

A.5.8.5 Rewrite Editor Variants

Finally, the rewrite editor $M_{\text{rw}_{\text{DWB}}}$ used on the DWB verse corpus as described in chapter 3 and the editor $M_{\text{rw}_{\text{DTA}}}$ used on DTA corpus as described in chapter 4 were defined as:

$$\begin{array}{rcl} M_{\text{rw}_{\text{DWB}}} & := & M_{\text{xlit}} \circ M_{\#} \circ M_{\text{ul}*} \circ M_{\varepsilon} \circ M_{\text{rw}_{\text{CORE}}} \circ M_{\not\#} \\ M_{\text{rw}_{\text{DTA}}} & := & M_{\text{xlit}} \circ M_{\#} \circ M_{\text{case,0}} \circ M_{\varepsilon} \circ M_{\text{rw}_{\text{CORE}}} \circ M_{\text{xc}} \circ M_{\text{case,1}} \circ M_{\not\#} \end{array}$$

The canonicalization cascades passed to the online k-best lookup algorithm from chapter 2 to compute the functions $\text{best}_{\text{Lev}}(\cdot)$ and $\text{best}_{\text{rw}}(\cdot)$ were defined in terms of the adjusted TAGH lexica described in appendix A.3 as:

$$\begin{array}{rcl} C_{\text{Lev}} & = & M_{\text{Lev}} \circ A_{\text{Lex}_{\text{Lev}}} \\ C_{\text{rw}} & = & M_{\text{rw}} \circ A_{\text{Lex}_{\text{rw}}} \end{array}$$

Appendix B

Selected Software

This appendix contains a brief introduction to the various software libraries, modules, and utilities developed by me in the course of the work described above.

B.1 `unicruft`: Transliteration

`http://odo.dwds.de/~moocow/software/unicruft/`

`unicruft` is a stand-alone C library which implements the transliteration function xlit(·) as described in section 4.2.2 and formally defined in appendix A.1, using a global array for constant-time lookup of character transliteration target strings. Additional heuristics for handling multi-codepoint characters such as combining diacritics were implemented by a simple GNU `flex` scanner.[1] Supported encodings include the variable-width Unicode encoding UTF-8, the Latin-1 subset of UTF-8, the 8-bit fixed-width Latin-1 encoding ISO-8859-1, the contemporary German subset of Latin-1, and 7-bit fixed-width ASCII. The `unicruft` distribution includes a simple command-line conversion utility as well as external API bindings for the high level programming language Perl.[2]

B.2 `gfsm` & `gfsmxl`: Finite-State Operations

`http://www.ling.uni-potsdam.de/~moocow/projects/gfsm`

`gfsm` is a stand-alone C library for representation and manipulation of

[1] `http://www.gnu.org/software/flex/`
[2] `http://www.perl.org`

weighted finite-state transducers, using `glib`[3] for low-level data structures, and providing both command-line utilities for common high-level operations as well as Perl language API bindings. The library was used in the work described above for run-time operations on finite-state machines, including phonetization and morphology lookup; as well as in the implementation of the `festival` LTS rule compiler (see appendix B.3).

`gfsmxl` is a `gfsm` extension library for online k-best string lookup operations in weighted finite-state transducer cascades which implements Algorithm 2.4 from chapter 2. It was used in the current work to compute the elementary rewrite canonicalization functions $\text{best}_{\text{Lev}}(\cdot)$ and $\text{best}_{\text{rw}}(\cdot)$ in chapters 3 and 4.

B.3 `Lingua::LTS`: LTS Rule Compiler

`http://odo.dwds.de/~moocow/software/Lingua-LTS/`

`Lingua::LTS` is a Perl module providing an object-oriented compiler and interpreter for deterministic letter-to-sound rules with `festival`-like syntax and semantics which implements the FST construction for `festival` LTS rule-sets from section 1.2.1 using the `gfsm` library (see appendix B.2). A straightforward extension to the construction for precedence-ordered deterministic LTS rules as used by `festival` enables compilation of weighted parallel rule-sets into non-deterministic weighted finite state transducers. This latter feature was used to compile the heuristic rewrite transducer used in chapters 3 and 4 and formally defined in appendix A.5.

B.4 `Taxi`: Structured Text Indices

`http://odo.dwds.de/~moocow/software/Taxi-Mysql/`

`Taxi` *("**T**ext **a**nd **X**ML **I**ndex")* is a command-line and client/server suite implemented in Perl for flexible indexing and intuitive query of structured linguistic data extracted from arbitrary XML documents using the `MySQL` relational database as a back-end. The `Taxi` system was developed in the course of the work described in chapter 1 for storage and retrieval of the DWB verse quotation evidence corpus. Type-wise manual annotation of the evaluation portion of the DWB verse corpus used in chapter 3 was also performed using a specialized `Taxi` plug-in module.

[3] `http://directory.fsf.org/project/glib/`

B.5 `moot`: HMM Tagging/Disambiguation

`http://www.ling.uni-potsdam.de/~moocow/projects/moot`

`moot` is a C++ library and program suite for Hidden Markov Model decoding using the Viterbi algorithm. Originally designed for part-of-speech tagging in the presence of a high-coverage morphological component using ambiguity classes to improve performance for unknown words (Jurish, 2003), `moot` was extended in the course of the work described above to allow dynamic re-computation of the underlying model parameters at run-time, based on the sentence to be decoded.[4] The dynamic HMM disambiguator described in chapter 4 made use of this latter feature to implement the Maxwell-Boltzmann estimation of canonicalization hypotheses' lexical probabilities.

B.6 `dta-tokwrap`: XML/TEI Serialization

`http://odo.dwds.de/~moocow/software/dta-tokwrap/`

`dta-tokwrap` is a suite of C utilities together with a high-level object-oriented Perl module for linguistically salient serialization of arbitrary XML documents encoded according to the Text Encoding Initiative (TEI) P5 Guidelines.[5] As their name suggests, the `dta-tokwrap` utilities were developed and used to serialize the XML/TEI documents of the *Deutsches Textarchiv* corpus – including the evaluation portion used in chapter 4 – before invoking an external stream-oriented tokenizer for estimation of sentence- and token-boundaries.

B.7 `DTA::EvalCorpus`: Alignment and Annotation

Construction of the DTA evaluation subset used in chapter 4 involved the automatic alignment of historical texts with contemporary editions. The automatic alignment was performed using GNU `diff`[6] on the respective tokenized texts. The type-wise confirmation phase was implemented by my colleague M. Drotschmann. The token-wise annotation of unconfirmed alignments was performed using the specialized graphical user interface module `DTA::EvalCorpus::GUI`.

[4] The dynamic HMM decoder is implemented by the program `dmoot`.

[5] `http://www.tei-c.org/Guidelines/P5/`

[6] `http://www.gnu.org/software/diffutils/`

B.8 DTA::CAB: Canonicalization

http://odo.dwds.de/~moocow/software/DTA-CAB/

DTA::CAB *("**C**ascaded **A**nalysis **B**roker")* is a top-level set of command-line programs and client/server suite for robust and reliable canonicalization of historical input text, written in Perl. This is the highest-level run-time canonicalization package described here, using `unicruft` for transliteration, `gfsm` for phonetizer and morphology lookup, `gfsmxl` for rewrite conflation, `dmoot` for canonicalization hypothesis disambiguation, and a static `moot` model for part-of-speech tagging. The canonicalizer-specific evaluation data in chapters 3 and 4 was acquired using DTA::CAB, and a dedicated DTA::CAB server[7] is responsible for expanding user queries (i.e. inverse canonicalization) based on a preceding canonicalization for the *Deutsches Textarchiv*.

[7] Publicly accessible demo version: http://www.deutschestextarchiv.de/cab/

Appendix C

Corpora

C.1 DWB Verse Corpus

The corpus of historical German verse used in chapters 1 and 3 was comprised of the quotation evidence in the digital edition of the *Deutsches Wörterbuch* (DWB, Bartz et al., 2004), also known as "the Grimm". Legacy SGML sources for the 622 volumes of the DWB were provided by the University of Trier. In the context of an exploratory project at the *Berlin-Brandenburgische Akademie der Wissenschaften* (BBAW, "Berlin-Brandenburg Academy of Sciences"),[1] I converted these sources to well-formed XML using the UTF-8[2] character encoding and indexed them with the `Taxi` suite (appendix B.4).

The SGML sources contained a total of 1156 entities, of which only 50 were assigned appropriate expansions by the provided DTDs, and a further 52 were defined by the ISO 8879:1986 standard. The remaining SGML entities were automatically parsed into (*BaseLetter DiacriticMark*$^+$) strings and assigned appropriate definitions using Unicode combining diacritic marks.[3] After manual addition of 47 letter and diacritic aliases, 361 of the remaining entities were exhaustively decomposed in this manner, including all of the entities occurring in the verse quotation evidence used for the corpus. The remaining 693 entities represented low-level typographical control functions and foreign-language characters (mostly Greek and Hebrew); these were assigned partial expansions based on their decompositions.

Verse quotation evidence was extracted from the UTF-8 XML sources by

[1] `http://www.bbaw.de`

[2] cf. RFC 3629 (`http://tools.ietf.org/html/rfc3629`), ISO/IEC 10646:2003 Annex D (`http://www.iso.org/iso/rss.xml?csnumber=39921&rss=detail`), Unicode Consortium (2011, Chapter 3).

[3] `http://www.unicode.org/charts/PDF/U1DC0.pdf`

the simple XPath[4] expression `//add//q//*` and heuristically tokenized using a dedicated Perl script. The full DWB verse quotation evidence corpus as indexed by `Taxi` contained 6,581,501 tokens of 322,271 distinct graphemic types, including punctuation and numerals. Of these, 5,492,076 tokens of 320,310 distinct types contained only alphabetic characters and diacritic marks and were thus treated as "word-like". The corpus was divided by XPath expressions into 917,362 lines of verse (`//add//q`) in 382,766 distinct quotations (`//add`) each of which was associated with exactly one of 297,613 distinct dictionary entries (`//add/ancestor::entry[1]`). Bibliographic meta-data was associated with each quotation by means of further XPath expressions: each quotation was assigned to one of 77,901 distinct citation identifiers (`//add/title/bibl/ref`) and one of 8,303 distinct author identifiers (`//add/title/bibl/author`), but the inconsistent use of abbreviations and omissions in the original print-oriented SGML sources suggest that these values should be treated with caution.

Cited authors included Otfrid von Weissenburg (ca. 790–875), Hartmann von Aue (ca. 1170–1210), Walther von der Vogelweide (ca. 1170–1230), Konrad von Würzburg (ca. 1220–1287), Heinrich Wittenwiler (ca. 1370–1420), Oswald von Wolkenstein (ca. 1377–1445), Sebastian Brant (1457–1521), Hans Sachs (1494–1576), Johann Baptist Fischart (1546–1591), Andreas Gryphius (1616–1664), Daniel Caspar von Lohenstein (1635–1683), Gotthold Ephraim Lessing (1729–1781), Friedrich Schiller (1759–1805), and Johann Wolfgang von Goethe (1749–1832), indicating a high degree of heterogeneity in the corpus data, since the works of these authors are distributed among all of the Old, Middle, Early-New, and New High German periods and various regional dialects.

C.1.1 DWB Evaluation Subcorpus

The evaluation in chapter 3 was performed on a manually annotated subset of the DWB verse corpus, which is referred to here as the DWB evaluation corpus. The evaluation subcorpus contained all the verse quotations from a single volume (`gr01.xml`) of the DWB. Specifically, the evaluation corpus was comprised of 13,327 tokens of 4,170 distinct string types of which 11,242 tokens of 4,157 distinct types were "word-like", and was organized into 1915 lines of verse in 778 quotations distributed over 652 dictionary entries.

Each evaluation corpus type was manually assigned one or more extant equivalents by a native German speaker based on inspection of its occurrences in the full DWB verse corpus in addition to secondary sources, using a specialized `Taxi` plug-in module. Extinct roots, proper names, foreign and

[4] `http://www.w3.org/TR/xpath`

other non-lexical material were not explicitly assigned any extant equivalent at all, but rather flagged and treated as if identical with their respective canonical cognates. In all other cases, equivalence was determined by direct etymological relation of the root in addition to matching morphosyntactic features. Problematic types were marked as such and subjected to expert review, which was performed by the author together with a language historical expert. 585 evaluation corpus tokens of 296 distinct types were ambiguously associated with more than one canonical cognate. In a second annotation pass, these remaining ambiguities were resolved on a per-token basis.

Availability

At the time of this writing (January, 2011), neither the full `Taxi` DWB verse corpus nor the evaluation subcorpus is publicly available. A publicly accessible print-oriented electronic version of the DWB can be found at `http://www.woerterbuchnetz.de/`. The `Taxi` DWB verse corpus and/or the evaluation subset used in the current work may be available for research purposes by special agreement with the Berlin-Brandenburg Academy of Sciences.

C.2 DTA Corpus

The *Deutsches Textarchiv* (DTA, "German Text Archive")[5] is a project at the *Berlin-Brandenburgische Akademie der Wissenschaften* (BBAW, "Berlin-Brandenburg Academy of Sciences"),[6] funded by the *Deutsche Forschungsgemeinschaft* (DFG, "German Research Foundation"). The goal of the DTA is the high-quality digitization of a core set of German-language texts from various disciplines originally published between ca. 1650 and 1900, and their release as a freely available linguistically annotated electronic text corpus. Currently (5th January, 2011), 532 texts are available online from the DTA website, which implements an inverse-canonicalizing search function similar to that described in the evaluations from chapters 3 and 4 using the `DTA::CAB` web service (appendix B.8). The ongoing DTA project provided much of the motivation for the development of the methods described in the current work, since the serialization and canonicalization of "raw" historical texts must precede any search of those texts using contemporary query terms according

[5] `http://www.deutschestextarchiv.de`
[6] `http://www.bbaw.de`

to the proposed canonicalization architecture.[7]

The DTA texts subjected to the canonicalization methods described here comprised 390 titles from 312 authors in 649 volumes published between 1778 and 1903. The electronic text of each volume was encoded according to the Text Encoding Initiative (TEI) P5 Guidelines,[8] serialized using the `dta-tokwrap` package (appendix B.6), and tokenized using the `ToMaSoTaTh` tokenizer developed by the *Digitales Wörterbuch der deutschen Sprache des 20. Jahrhunderts* (DWDS, "Digital Dictionary of 20th Century German")[9] project with a modified abbreviation lexicon and without multi-word expressions. The full tokenized corpus contained 62,649,970 tokens of 1,583,347 distinct graphemic types, of which 51,288,749 tokens of 1,334,366 types contained only alphabetic characters, diacritic marks, and/or hyphens and were thus treated as "word-like".

C.2.1 DTA Evaluation Subcorpus

The evaluation in chapter 4 was performed on a manually annotated subset of the DTA corpus, which is referred to here as the DTA evaluation corpus. The DTA evaluation corpus was extracted from the main body text from 13 volumes published between 1780 and 1880, namely:

1. Brentano, Clemens: *Geschichte vom braven Kasperl und dem schönen Annerl*. Berlin: Vereinsbuchhandlung, 1838.
2. Busch, Wilhelm: *Max und Moritz*. München: Braun & Schneider, 1865.
3. Goethe, Johann Wolfgang von: *Iphigenie auf Tauris*. Leipzig: Göschen, 1787.
4. Goethe, Johann Wolfgang von: *Wilhelm Meisters Lehrjahre*. Bd. 1. Berlin: Unger, 1795.
5. Goethe, Johann Wolfgang von: *Wilhelm Meisters Lehrjahre*. Bd. 2. Berlin: Unger, 1795.
6. Goethe, Johann Wolfgang von: *Wilhelm Meisters Lehrjahre*. Bd. 3. Berlin: Unger, 1795.

[7] Pilz et al. (2006, Sec. 4.3) argue in contrast that "documents cannot be indexed before the search", which seems to stem chiefly from a desire to maintain the independence of their (inverse-) canonicalization rule-set from the indexed corpus, e.g. in order to allow users to alter the rule-set on a per-search basis.

[8] `http://www.tei-c.org/Guidelines/P5/`

[9] `http://www.dwds.de`

7. Goethe, Johann Wolfgang von: *Wilhelm Meisters Lehrjahre*. Bd. 4. Berlin: Unger, 1796.
8. Goethe, Johann Wolfgang von: *Torquato Tasso*. Leipzig: Göschen, 1790.
9. Kant, Immanuel: Beantwortung der Frage Was ist Aufklärung? In: *Berlinische Monatsschrift*, 1784, H. 12, S. 481-494.
10. Lessing, Gotthold Ephraim: *Die Erziehung des Menschengeschlechts*. Berlin: Voss, 1780.
11. Schiller, Friedrich: *Kabale und Liebe*. Mannheim: Schwan, 1784.
12. Spyri, Johanna: *Heidi's Lehr- und Wanderjahre*. Gotha: Perthes, 1880.
13. Storm, Theodor: *Immensee*. Berlin: Duncker, 1852.

The corpus contained a total of 191,265 tokens of 19,221 distinct types, of which 152,776 tokens of 17,417 types were "word-like", divided into 9,079 sentences. To assign an extant canonical equivalent to each token of the test corpus, the text of each volume was automatically aligned token-wise with a contemporary edition of the same volume (appendix B.7). Automatically discovered non-identity alignment pair types were presented to a native speaker of German for confirmation.

In a second annotation pass, all tokens lacking an identical or manually confirmed alignment target were inspected in context and manually assigned a canonical form. Whenever they were presented to a human annotator, proper names and extinct lexemes were treated as their own canonical forms. Alignments spanning token boundaries such as (*bey_Seite* $\sim$ *beiseite*) or (*destomehr* $\sim$ *desto_mehr*) were explicitly marked, and the historical tokens were additionally associated with compositional or concatenated equivalents which were used for evaluation purposes. In all other cases, equivalence was determined by direct etymological relation of the root in addition to matching morphosyntactic features. Problematic tokens were marked as such and subjected to expert review. Marginalia, front and back matter, speaker and stage directions, and tokenization errors were excluded from the final evaluation corpus.

Construction of the DTA evaluation corpus is an ongoing project. At the time of this writing (January, 2011), an additional 117 text volumes containing a total of 5,467,703 tokens have been automatically aligned with contemporary editions, of which 65 volumes containing a total of 2,092,632 tokens have passed through the token-level annotation phase and are awaiting expert review.

Availability

At the time of this writing (January, 2011), the TEI-P5 XML sources of all of the 532 online DTA texts are available for non-commercial use under the terms of the Creative Commons "*by-nc*" license[10] from the DTA web site, `http://www.deutschestextarchiv.de`. The contemporary editions used in the ongoing construction of the DTA evaluation subcorpus were made available to the DTA project by special agreement with an external provider, the terms of which preclude public distribution of this corpus at the current time. Efforts to ensure the availability of the DTA evaluation subcorpus for research and education purposes are being made; contact the DTA project (`info@deutschestextarchiv.de`) for details.

[10] `http://creativecommons.org/licenses/by-nc/3.0/`

Glossary

allograph One of multiple alternative written realizations of some common linguistic (often phonetic) feature or features; e.g. *t* and *th* are allographs in historical German associated with the common phonetic properties "voiceless alveolar plosive". See also: *grapheme*.

ambiguity In general, any linguistic construct admitting more than one interpretation; used in chapter 4 to denote a word type conflation set with more than one extant element, i.e. a multiplicity of candidate canonical forms. See also: *disambiguation*.

canonicalization *"A process for converting data ... into a 'standard', 'normal', or canonical form."*[1] In the context of the current work, the data to be converted are historical spelling variants, the desired canonical forms for which are lexically equivalent extant cognates, insofar as these exist. I informally refer to any method for associating historical forms with potential extant cognates as a **canonicalization method**, using the term **canonicalization function** for a mathematical function (i.e. a left-total and right-unique binary relation) associating each input string with exactly one canonical object. The range of a canonicalization function need not be restricted to extant forms; in particular a phonetization function mapping arbitrary input strings to unique phonetic forms can be considered a canonicalization function in this sense. See also *conflation*.

cascade See *weighted finite-state cascade*.

cognate *"A word either descended from the same base word of the same ancestor language as the given word, or strongly believed to be a regular reflex of the same reconstructed root of proto-language as the given word."*[2] In particular, I use the term **canonical cognate** to denote

[1] Source: `http://en.wikipedia.org/wiki/Canonicalization`
[2] Source: `http://en.wiktionary.org/wiki/cognate`

an extant lexical equivalent of a given historical word, e.g. *Tür* is (usually) the canonical cognate of the historical form *Thür*. See also: *canonicalization*, sections 3.1, 4.1.

conflation Literally, a 'blowing' or 'fusing together' of distinct items into a single (composite) entity. In the context of the current work, the items to be combined are character strings representing both historical and modern spelling variants (word surface types), to be merged into a **conflation set** which ideally should be co-extensional with a unique characteristic "deep" word type. I refer to any method for inducing a conflation set from arbitrary (historical) input strings as a **conflator**, formally characterizing these in terms of a binary **conflation relation** on strings (cf. sections 1.1, 3.2, and 4.2). In particular, any canonicalization function operating on word types defines a conflator as the composition of that function with its inverse, inducing a conflation relation which is also a true equivalence relation (i.e. reflexive, symmetric, and transitive). See also: *canonicalization*.

coverage See *lexical coverage*.

Damerau-Levenshtein distance An edit distance function defined as the minimum number of character insertions, deletions, substitutions, and/or transpositions required to transform one argument string into the other. See also: *edit distance*, *Levenshtein distance*.

diachronic Of or relating to a span of time; usually used in linguistics to refer to changes undergone by a given language over the course of time. Contrast: *synchronic*.

disambiguation Elimination of all but one of a set of multiple admissible interpretations; used in chapter 4 to denote the selection of a unique canonical form for each token of a historical input sentence from a (potentially ambiguous) set of conflation hypotheses. See also: *ambiguity*.

distance function A mathematical function $d(\cdot, \cdot)$ from pairs of objects to non-negative real numbers representing the degree of the arguments' dissimilarity. If for all x, y, z it also holds that $d(x, y) = d(y, x)$ [symmetry], $d(x, z) \leq d(x, y) + d(y, z)$ [triangle inequality], and $d(x, y) = 0 \Rightarrow x = y$ [identity of indiscernibles], then d is a true **metric**.

edit distance A distance function over strings, usually refers to the Levenshtein distance. The **generalized edit distance** is an edit distance

function additionally parameterized by an *editor* or *error model* specifying the admissible edit operations together with their costs. See also: *Damerau-Levenshtein distance*, *Levenshtein distance*, *editor*.

edit transducer An *editor* implemented as a weighted finite-state transducer. See also: *editor*, *rewrite transducer*, *finite-state machine*, and section 2.1.1.

editor An inventory of elementary **edit operations** (string transformations), each of which is associated with a characteristic **edit cost**. See also: *edit distance*, *error model*.

error model An *editor* whose edit operations represent errors introduced by transmission over a noisy channel, the costs of which represent the respective errors' independent inverse likelihood. Can be treated for most purposes as synonymous with 'editor', the only difference between the two notions being the noisy-channel assumptions connoted by 'error model'. See also: *edit distance*, *editor*.

error reduction rate In empirical evaluations, the (relative) error reduction rate for an evaluated method A with respect to a method B is the proportion of the absolute difference in negative results ('errors') for B versus A to the total negative results for B, $(\overline{B} - \overline{A})/\overline{B}$, informally understandable as the proportion of errors made by B which are not incurred by A. Error reduction rates are particularly useful for comparing different methods when 'errors' are restricted to a fixed interval, as precision and recall errors are restricted to the interval $[0, 1]$, since absolute differences necessarily grow smaller as the number of errors approaches zero. See section 4.4.2 for an explicit definition of relative precision and recall error reduction rates as used in the current work.

extant *"Still existing; not destroyed or lost"* (Source: Porter, 1913). Used here to describe synchronically active lexemes; i.e. elements of the contemporary lexicon, at least one instance of which belongs to the extension of a contemporary speaker's competence. By contrast, **extinct** lexemes are not retrievable via the contemporary lexicon, and their instances thus have no canonical cognate. See also: *canonicalization*, *cognate*, *lexeme*, *lexicon*, *synchronic.*.

finite-state machine (FSM) An abstract machine representing a set of strings (a **finite-state acceptor (FSA)**) or string pairs (a **finite-state transducer (FST)**), optionally associating each element with a unique

weight or cost drawn from a given semiring (a **weighted finite-state acceptor (WFSA)** or **weighted finite-state transducer (WFST)**, respectively). In the current work, I treat all of the above sorts of finite-state machine as weighted transducers, which are formally defined in section 2.2.

gold standard A manually annotated data-set, assumed for purposes of training or empirical evaluation to represent the truth, the whole truth, and nothing but the truth. As with all empirical generalizations, the second of these assumptions ('the whole truth') is only valid to the extent that the data constitute a representative sample of the population under consideration.

grapheme An elementary unit of written language, e.g. a letter, character, or glyph. In adjectival form, I use **graphemic** to describe historical or contemporary synchronic character- or character-string related phenomena, and **graphematic** to describe phenomena arising from or pertaining to the relation between graphemic forms and the diachronic lexicon. See also: *allograph*, *phone*.

hidden Markov model (HMM) A stochastic language model which approximates sentence probability as a chain of locally state-dependent transitions between 'hidden' model states interleaved with state-dependent lexical insertions. Can be implemented as a weighted finite-state machine. See also: section 4.3.1 and the references cited there.

language model In general, any formal model of a string language, i.e. a well-defined finite specification of a (possibly infinite) set of strings. In practice, usually denotes a **stochastic language model**, a mathematical model defining a probability distribution over strings. Sometimes used in error-correction applications to add some degree of context dependence to a character- or word-local error model. See also: *error model*, section 4.3.

lemma A conventional "unmarked" representation of a lexeme. In the context of a dictionary corpus such as used in chapter 1, 'lemma' also denotes the keyword associated with a given dictionary entry. Assuming the dictionary's structure accurately reflects that of the mental lexicon, no ambiguity need result. See also: *lexeme*.

letter-to-sound (LTS) conversion See *phonetic digest*.

Levenshtein distance An edit distance function defined as the minimum number of character insertions, deletions, and/or substitutions required to transform one argument string into the other. See also: *Damerau-Levenshtein distance*, *edit distance*, section 3.2.2.

lexeme The elementary unit of lexical organization, In generative linguistics, a single lexeme is characterized by a unique combination of semantic, syntactic, and morphological properties, and may have multiple surface word-type realizations; e.g. *love*, *loves*, *loved*, and *loving* are all potential realizations of a single lexeme whose lemma is LOVE. In computer science, 'lexeme' is often used synonymously with 'word type', most likely due to the fact that the vast majority of programs requiring any explicit lexicon at all do in fact use surface types as the basic (and often only) units of lexical organization. In the current work, 'lexeme' denotes whatever elementary unit of organization is appropriate to a given lexicon; if unspecified, the linguistic usage (i.e. the 'mental' lexicon) is assumed. See also: *lemma*, *lexicon*.

lexical coverage In general, the ratio of the number of words (types or tokens) in a given corpus which are accounted for by a given lexical analysis function to the total size of the corpus. As used in chapter 1 as an evaluator for canonicalization functions, denotes the fraction of corpus words which can be conflated with at least one extant form using a given canonicalization function.

lexicon In the generative linguistic tradition, the 'mental lexicon' is the vocabulary of lexemes associated with a given language or speaker's competence. In the theory of formal languages, a lexicon is usually characterized as a finite set of discrete symbols, and the (weak) extension of a language is defined as a set of strings over these lexical symbols. An ambiguity arises when attempting to construct a formal model of the mental lexicon: the simplest formal models require that lexical symbols be valid surface word-forms, while linguistic tradition posits more abstract units of lexical organization (lexemes). In the current work, I use 'lexicon' primarily in just this problematic sense of a formal model of the mental lexicon, where the units of lexical organization and access depend on the application or model in question. See also: *lexeme*.

path A sequence of adjacent transitions in a finite-state machine or hidden Markov model. See also: *finite-state machine*, *hidden Markov model*, and section 2.2.

phone An elementary unit of spoken language. Each uttered phone is an instance of some **phoneme**, the latter being an abstract 'mental' unit of speech sound possibly instantiated by multiple phones. In adjectival form, I speak of a **phonetic** representation (e.g. of a word) whenever the representation has an immediate interpretation as a string of phones. See also: *grapheme*.

phonetic digest Any hash value or digest code acting as a canonical form for an input word string whose computation relies on assumed phonetic properties of the input characters. I distinguish phonetic **digest algorithms** such as SOUNDEX or PHONIX commonly used for information retrieval which return an abstract code with no immediate interpretation as phonetic or phonemic strings from true phonetization or **letter-to-sound (LTS)** algorithms commonly used in text-to-speech synthesis systems which directly return a phonetic string. In the case of variable-length digest codes such as those computed by the *Kölner Phonetik* or *Metaphone* which can be trivially converted to an (underspecified) phonetic representation and the simplified phonetization function from sections 1.2, 3.2.1, and 4.2.3, this distinction is connotative only.

pointwise mutual information (PMI) In information theory, the difference between the sum of the code-lengths for a pair of stochastic events considered independently and the length of their joint code under an optimal encoding scheme. Informally, PMI can be understood as quantifying the degree of association between two stochastic events. I use PMI in section 1.3 to detect and filter out "false friends" during estimation of the dictionary lemma instantiation relation.

precision In information retrieval, *"the proportion of retrieved material that is actually relevant"* [to a given search request] (Source: van Rijsbergen, 1979), i.e. the empirical probability of relevance given retrieval. As used for the evaluations in chapters 3 and 4, 'relevance' is given by a manually annotated canonical-cognate relation, a 'search request' is a contemporary word form, 'retrieval' is performed by an inverse canonicalization function, and the items to be retrieved are either word types or tokens. See also: *recall*, *precision-recall harmonic average*, sections 3.3.2 and 4.4.2.

precision-recall harmonic average (F) In information retrieval, a composite measure of a retrieval function's effectiveness defined in terms of its precision and recall, usually attributed to van Rijsbergen (1979,

chapter 7, where it appears as E). See also: *precision*, *recall*, sections 3.3.2 and 4.4.2.

pseudo-metric In mathematics, a distance function for which the identity of indiscernibles need not hold. Used here to denote any arbitrary distance function. See also: *distance function*.

recall In information retrieval, *"the proportion of relevant material actually retrieved in answer to a search request"* (Source: van Rijsbergen, 1979), i.e. the empirical probability of retrieval given relevance. As used for the evaluations in chapters 3 and 4, 'relevance' is given by a manually annotated canonical-cognate relation, a 'search request' is a contemporary word form, 'retrieval' is performed by an inverse canonicalization function, and the items to be retrieved are either word types or tokens. See also: *precision*, *precision-recall harmonic average*, sections 3.3.2 and 4.4.2.

rewrite transducer An edit transducer used in the current work to heuristically model diachronic spelling variation. See: *edit transducer*, sections 3.2.3, 4.2.4, and appendix A.5.

semiring An algebraic structure with two binary operations $\otimes$ and $\oplus$ used in the context of the current work to interpret weights of a weighted finite-state machine along a single path and over multiple paths, respectively. See definition 2.1 in section 2.2 and the references cited there.

string edit distance See *edit distance*.

synchronic Of or relating to a particular point in time; usually used in linguistics to refer to properties of a given language at the current moment, e.g. the synchronic lexicon is the lexicon of currently active lexemes. Contrast: *diachronic*.

token A single actually occurring instance of a *type*. Here, I am only concerned with those token-specific properties relevant to the successful canonicalization of historical word types. In section 4.3, I use sentential context for this purpose, effectively defining a canonicalization function on sentence types. I nonetheless refer to sentence-type canonicalization as a 'token'-level function, to distinguish it from other methods operating directly on surface word types. See also: *type*.

transliteration *"To express or represent in the characters of another alphabet"* (Source: Porter, 1913). Used in chapter 4 to denote any canonicalization function defined as the reflexive and transitive closure of a map from (possibly extinct) single characters to strings of extant characters. See also: *canonicalization*, *conflation*, and section 4.2.2.

type A class of phenomena considered as a single abstract whole, as opposed to a collection of individual instances or *tokens*. In the current work, the types with which I am primarily concerned are word forms as given by a surface character string. See also: *token*.

weighted finite-state cascade (WFSC) A weighted relation arising from the iterated composition of two or more finite-state machines, in theory itself representable as a weighted finite-state transducer. See also: *finite-state machine*, section 2.2.

weighted finite-state transducer (WFST) See *finite-state machine*.

Acronyms and Abbreviations

DL Damerau-Levenshtein (distance).

DTA *Deutsches Textarchiv*; see appendix C.2.

DWB *Deutsches Wörterbuch*; see appendix C.1.

FSA finite-state acceptor; see *finite-state machine*.

FSM finite-state machine.

FST finite-state transducer; see *finite-state machine*.

HMM hidden Markov model.

LTS letter-to-sound.

PMI pointwise mutual information.

WFSA weighted finite-state acceptor; see *finite-state machine*.

WFSC weighted finite-state cascade.

WFSM weighted finite-state machine; see *finite-state machine*.

WFST weighted finite-state transducer; see *finite-state machine*.

Symbols and Notation

$[\![M]\!]$ Weight assigned by a WFST M to an argument string or pair, see definition 2.4 from section 2.2.

$[w]$ Standard notation for an equivalence class induced by w. Subscripted with a conflation relation r, $[w]_r$ represents the conflation-set of the word w provided by the conflator r. Square brackets are also used in chapter 2 to indicate array-indexing operations. See $f[i]$, sections 1.1, 3.2, and 4.2.

$\sim$ Operator infix notation for a conflation relation; subscripted where appropriate to indicate the associated conflation method. See also $\tilde{w}$ and $\widetilde{Q}$ for other uses of the tilde symbol.

π Variable for a phonetic string (chapter 1, appendix A.2), a rewrite target string (appendix A.5), or for a *(W)FST path* (chapter 2).

$\mathcal{A}$ A finite alphabet; subscripted where appropriate.

F Precision-recall harmonic average; subscripted to indicate the evaluated canonicalization method and evaluation unit (types or tokens).

$f[i]$ `C`-like array indexing notation used in chapter 2 for the image of the object i under the (usually finite) function f. Not to be confused with a conflation set $[w]$.

hmm Subscript for the HMM conflation disambiguator from chapter 4.

Id$(\cdot)$ Identity function, used to explicitly convert a string or acceptor to a *(W)FST*.

id Subscript indicating the identity conflation relation, a trivial canonicalization function used as a baseline for evaluations.

$\mathcal{K}$ Cariable for a semiring (structure).

$\mathbb{K}$ Carrier for a semiring; a set of admissible weights.

Lev Subscript indicating the Levenshtein distance or its implementation as an edit transducer.

Lex Target language *lexicon*, a set of surface word types (character strings).

M Variable for a *WFST*, subscripted where appropriate.

$\mathbb{N}$ Set of all natural numbers.

$\mathcal{P}$ A finite phonetic alphabet.

pho Phonetization function or subscript indicating the associated conflation relation.

pr Precision; subscripted to indicate the evaluated canonicalization method and evaluation unit (types or tokens).

$\widetilde{Q}$ Used in chapter 2 to denote a characteristic set of pseudo-transitions for a set Q of states used in the constructive definition of weighted finite-state transducer composition. Not to be confused with a conflation relation $\sim$ or a canonical cognate $\tilde{w}$.

$\mathbb{R}$ Set of all real numbers.

$\mathbb{R}_\infty$ Set of all real numbers or ∞.

$\mathbb{R}_+$ Set of all non-negative real numbers.

rc Recall; subscripted to indicate the evaluated canonicalization method and evaluation unit (types or tokens).

rw Subscript for the heuristic rewrite transducer or its associated conflation relation.

S Variable for a set of LTS rules (chapter 1) or for a historical input sentence (chapter 4).

$\mathcal{W}$ A finite word alphabet, the set of surface types actually occurring in a given corpus.

w Variable for a word, a string of characters.

$\tilde{w}$ The canonical cognate of the word w.

$\vec{w}$ Vector notation for a word used in chapter 2 for positional subscripting convenience.

xlit Transliteration function or subscript indicating the associated conflation relation.

Bibliography

A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/360825.360855.

A. V. Aho and J. D. Ullman. *The Theory of Parsing, Translation and Compiling, Volume I: Parsing*. Prentice-Hall, Englewood Cliffs, N.J., 1972.

C. Allauzen and M. Mohri. Linear-space computation of the edit-distance between a string and a finite automaton. In *London Algorithmics 2008: Theory and Practice*. College Publications, 2009.

C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri. OpenFst: A general and efficient weighted finite-state transducer library. In J. Holub and J. Zdárek, editors, *Implementation and Application of Automata, 12th International Conference, CIAA 2007, Prague, Czech Republic, July 16-18, 2007, Revised Selected Papers*, pages 11–23, Berlin, 2007. Springer. doi: 0.1007/978-3-540-76336-9_3.

J. Allen, M. S. Hunnicutt, and D. Klatt. *From Text to Speech: the MITalk system*. Cambridge University Press, 1987.

L. R. Bahl, F. Jelinek, and R. L. Mercer. A Maximum Likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2):179–190, 1983.

A. Baron and P. Rayson. VARD2: A tool for dealing with spelling variation in historical corpora. In *Proceedings of the Postgraduate Conference in Corpus Linguistics*, Aston University, Birmingham, UK, 22nd May 2008.

M. Baroni, J. Matiasek, and H. Trost. Unsupervised discovery of morphologically related words based on orthographic and semantic similarity. In *Proceedings of the Workshop on Morphological and Phonological Learning of ACL-2002*, pages 48–57, 2002.

H.-W. Bartz, T. Burch, R. Christmann, K. Gärtner, V. Hildenbrandt, T. Schares, and K. Wegge, editors. *Der Digitale Grimm. Deutsches Wörterbuch von Jacob und Wilhelm Grimm.* Zweitausendeins, Frankfurt am Main, 2004. URL `http://www.dwb.uni-trier.de`.

T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression.* Prentice Hall, 1990.

G. F. Benecke, W. Müller, and F. Zarncke. *Mittelhochdeutsches Wörterbuch.* Leipzig 1854-1866, 3. Nachdruckauflage: Olms, Hildesheim, 1986.

A. W. Black and P. Taylor. Festival speech synthesis system. Technical Report HCRC/TR-83, University of Edinburgh, Centre for Speech Technology Research, 1997. URL `http://www.cstr.ed.ac.uk/projects/festival`.

J. Bortz. *Statistik für Sozialwissenschaftler.* Springer, Berlin, 4th edition, 1993.

S. Brants, S. Dipper, S. Hansen, W. Lezius, and G. Smith. The TIGER treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories*, Sozopol, 2002.

T. Brants. TnT – a statistical part-of-speech tagger. In *Proceedings of ANLP-2000*, 2000.

E. Brill. A simple rule-based part-of-speech tagger. In *Proceedings of ANLP-92, 3rd Conference on Applied Natural Language Processing*, pages 152–155, 1992.

E. Brill and R. C. Moore. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, 2000.

J. Byun, S.-W. Lee, Y.-I. Song, and H.-C. Rim. Two phase model for sms text messages refinement. In *Proceedings 2008 AAAI Workshop on Enhanced Messaging*, Menlo Park, California, 2008. AAAI Press.

M. Cafarella and D. Cutting. Building Nutch: Open source search. *Queue*, 2(2):54–61, 2004. ISSN 1542-7730. doi: http://doi.acm.org/10.1145/988392.988408.

E. Charniak, C. Hendrickson, N. Jacobson, and M. Perkowitz. Equations for part-of-speech tagging. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 784–789, 1993.

K. W. Church and W. A. Gale. Probability scoring for spelling correction. *Statistics and Computing*, 1:93–103, 1991.

K. W. Church and P. Hanks. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29, 1990.

S. Clematide. An OLIF-based open inflection resource and yet another morphological system for German. In Storrer et al. (2008), pages 183–194.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, Cambridge, MA, 2nd edition, 2001.

T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, New York, 1991.

M. A. Covington. An algorithm to align words for historical comparison. *Computational Linguistics*, 22:481–496, 1996.

D. Cutting, J. Kupiec, J. Pedersen, and P. Sibun. A practical part-of-speech tagger. In *Proceedings of ANLP-1992*, 1992.

F. J. Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7:171–176, March 1964. doi: 10.1145/363958.363994.

A. P. Dempster, N. M. Laird., and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society Series B*, 39:1–38, 1977.

S. DeRose. Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14(1):31–39, 1988.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

S. Dipper and B. Schrader. Comparing distance and relatedness of medieval text variants from German. In Storrer et al. (2008), pages 39–51.

T. Dutoit. *An Introduction to Text-to-Speech Synthesis*. Kluwer, Dordrecht, 1997.

A. Ernst-Gerlach and N. Fuhr. Generating search term variants for text collections with historic spellings. In M. Lalmas, A. MacFarlane, S. Rüger, A. Tombros, T. Tsikrika, and A. Yavlinsky, editors, *Advances in Information Retrieval*, volume 3936 of *Lecture Notes in Computer Science*, pages 49–60. Springer, Berlin, 2006. doi: 10.1007/11735106_6.

A. Ernst-Gerlach and N. Fuhr. Retrieval in text collections with historic spelling using linguistic and spelling variants. In *Proceedings of the 7th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL '07)*, pages 333–341, New York, 2007. ACM. doi: 10.1145/1255175.1255242.

A. Ernst-Gerlach and N. Fuhr. Semiautomatische Konstruktion von Trainingsdaten für historische Dokumente. In *Proceedings of the Workshop "Information Retrieval 2010" at LWA 2010*, 2010.

Z. Ésik and W. Kuich. Equational Axioms for a Theory of Automata. In C. M. Vide, V. Mitrana, and G. Păun, editors, *Formal Languages and Applications*, volume 148 of *Studies in Fuzziness and Soft Computing*, chapter 10, pages 183–196. Springer, Berlin, 2004.

R. Fano. *Transmission of information: A statistical theory of communications*. MIT Press, New York, 1961.

M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

T. N. Gadd. 'Fisching fore werds': phonetic retrieval of written text in information systems. *Program*, 22(3):222–237, 1988. doi: 10.1108/eb046999.

T. N. Gadd. PHONIX: The algorithm. *Program*, 24(4):363–366, 1990. doi: 10.1108/eb047069.

A. Geyken and T. Hanneforth. TAGH: A complete morphology for German based on weighted finite state automata. In *Finite State Methods and Natural Language Processing, 5th International Workshop, FSMNLP 2005, Revised Papers*, volume 4002 of *Lecture Notes in Computer Science*, pages 55–66, Berlin, 2006. Springer. doi: 10.1007/11780885_7.

D. Gildea and D. Jurafsky. Learning bias and phonological-rule induction. *Computational Linguistics*, 22(4):497–530, 1996.

A. Gotscharek, A. Neumann, U. Reffle, C. Ringlstetter, and K. U. Schulz. Constructing a lexicon from a historical corpus. In *Proceedings of the Conference of the American Association for Corpus Linguistics (AACL09)*, Edmonton, 2009a.

A. Gotscharek, A. Neumann, U. Reffle, C. Ringlstetter, and K. U. Schulz. Enabling information retrieval on historical document collections: the role of matching procedures and special lexica. In *Proceedings of The Third Workshop on Analytics for Noisy Unstructured Text Data*, AND '09, pages 69–76, New York, 2009b. ACM. doi: 1568296.1568309.

A. Gotscharek, U. Reffle, C. Ringlstetter, and K. U. Schulz. On lexical resources for digitization of historical documents. In *Proceedings of the 9th ACM symposium on Document Engineering*, DocEng '09, pages 193–200, New York, 2009c. ACM. doi: 1600193.1600236.

T. Hanneforth and K.-M. Würzner. Statistical language models within the algebra of weighted rational languages. *Acta Cybernetica*, 19(2):313–356, 2009.

A. Hauser, M. Heller, E. Leiss, K. U. Schulz, and C. Wanzeck. Information access to historical documents from the early new high german period. In *Proceedings of IJCAI-07 Workshop on Analytics for Noisy Unstructured Text Data (AND-07)*, pages 147–154, 2007.

M. Heller. Fachspezifische Indexierung von historischen Dokumenten. ein Framework zur approximativen Indexierung semistrukturierter Dokumente. In *Tagungsband "Forschung in der digitalen Welt"*, Hamburg, 10-11 April 2006.

M. Heller. Approximative Indexierungstechnik für historische deutsche Textvarianten. *Abhandlungen der Arbeitsgemeinschaft Geschichte und EDV* (in print), 2010.

M. Heller and G. Vogeler. Modern information retrieval technology for historical documents. In *Conference Proceedings of the Association of History and Computing*, pages 143–148, Amsterdam, 14-17 September 2005.

B. Hennig. *Kleines Mittelhochdeutsches Wörterbuch, 4. Auflage.* Max Niemeyer Verlag, Tübingen, 2001.

M. Hulden. Fast approximate string matching with finite automata. *Procesamiento del Lenguaje Natural*, 43:57–64, September 2009.

E. T. Jaynes. Brandeis lectures. In *E. T. Jaynes: Papers on Probability, Statistics and Statistical Physics*, pages 40–76. D. Reidel, Dordrecht, 1983.

F. Jelinek and R. L. Mercer. Interpolated estimation of Markov source parameters from sparse data. In E. S. Gelsema and L. N. Kanal, editors, *Pattern Recognition in Practice*, pages 381–397. North-Holland Publishing Company, Amsterdam, 1980.

F. Jelinek and R. L. Mercer. Probability distribution estimation from sparse data. *IBM Technical Disclosure Bulletin*, 28:2591–2594, 1985.

D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2nd edition, 2009.

B. Jurish. A hybrid approach to part-of-speech tagging. Technical report, Project "Kollokationen im Wörterbuch", Berlin-Brandenburg Academy of Sciences, Berlin, 2003. URL `http://www.ling.uni-potsdam.de/~moocow/pubs/dwdst-report.pdf`.

B. Jurish. Finding canonical forms for historical German text. In Storrer et al. (2008), pages 27–37. (Article 1).

B. Jurish. Efficient online k-best lookup in weighted finite-state cascades. In T. Hanneforth and G. Fanselow, editors, *Language and Logos: Studies in Theoretical and Computational Linguistics*, volume 72 of *Studia grammatica*, pages 313–327. Akademie Verlag, Berlin, 2010a. (Article 2).

B. Jurish. Comparing canonicalizations of historical German text. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology* (SIGMORPHON), pages 72–77, 2010b. (Article 3).

B. Jurish. More than words: Using token context to improve canonicalization of historical German. *Journal for Language Technology and Computational Linguistics*, 25(1):23–40, 2010c. (Article 4).

R. M. Kaplan and M. Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.

L. Karttunen, R. M. Kay, and K. Koskenniemi. A compiler for two-level phonological rules. In M. Dalrymple, R. Kaplan, L. Karttunen, K. Koskenniemi, S. Shaio, and M. Wescoat, editors, *Tools for Morphological Analysis*, volume 87-108 of *CSLI Reports*, pages 1–61. CSLI, Stanford University, Palo Alto, CA, 1987.

R. E. Keller. *The German Language*. Faber & Faber, London, 1978.

S. Kempken. *Bewertung von historischen und regionalen Schreibvarianten mit Hilfe von Abstandsmaßen*. Diploma thesis, Universität Duisburg-Essen, 2005.

S. Kempken, W. Luther, and T. Pilz. Comparison of distance measures for historical spelling variants. In M. Bramer, editor, *Artificial Intelligence in Theory and Practice*, pages 295–304. Springer, Boston, 2006. doi: 10.1007/978-0-387-34747-9_31.

M. D. Kernighan, K. W. Church, and W. A. Gale. A spelling correction program based on a noisy channel model. In *Proceedings COLING-1990*, volume 2, pages 205–210, 1990.

F. Kluge and E. Seebold. *Etymologisches Worterbuch der deutschen Sprache*. de Gruyter, Berlin, 1989.

D. Knuth. *The Art of Computer Programming, Volume 3: Sorting And Searching. Second Edition*. Addison-Wesley, Reading, MA, 1998.

G. Kondrak. A new algorithm for the alignment of phonetic sequences. In *Proceedings NAACL*, pages 288–295, 2000.

G. Kondrak. Identifying cognates by phonetic and semantic similarity. In *Proceedings NAACL*, pages 103–110, Pittsburgh, June 2001.

G. Kondrak. *Algorithms for Language Reconstruction*. PhD thesis, University of Toronto, July 2002.

G. Kondrak. Phonetic alignment and similarity. *Computers and the Humanities*, 37(3):273–291, August 2003.

B. Krenn and C. Samuelsson. *The Linguist's Guide to Statistics*. Unpublished manuscript, 1997. URL `http://www.ofai.at/~brigitte.krenn/papers/stat_nlp.ps.gz`.

W. Kuich and A. Salomaa. *Semirings, Automata, Languages*, volume 5 of *EATCS Monographs on Theoretical Computer Science*. Springer, Berlin, 1986.

K. Kukich. Techniques for automatically correcting words in texts. *ACM Computing Surveys*, 24(4):377–439, 1992.

É. Laporte. Rational transductions for phonetic conversion and phonology. In Roche and Schabes (1997).

V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(1966):707–710, 1966.

M. Lexer. *Mittelhochdeutsches Handwörterbuch*. Leipzig 1872-1878, Nachdruck: S. Hirzel Verlag, Stuttgart, 1992.

M. J. Liberman and K. W. Church. Text analysis and word pronunciation in text-to-speech synthesis. In S. Furui and M. M. Sondhi, editors, *Advances in Speech Signal Processing*. Dekker, New York, 1992.

G. J. Lidstone. Note on the general case of the Bayes-Laplace formula for inductive or *a priori* probabilities. *Transactions of the Faculty of Actuaries*, 8:182–192, 1920.

J. B. Lovins. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11:22–31, 1968.

C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, 1999.

E. Mays, F. J. Damerau, and R. L. Mercer. Context based spelling correction. *Information Processing & Management*, 27(5):517–522, 1991. doi: 10.1016/0306-4573(91)90066-U.

W. J. McGill. Multivariate information transmission. *IEEE Trans. Inf. Theory*, 4(4):93–111, 1955.

G. Möhler, A. Schweitzer, and M. Breitenbücher. *IMS German Festival manual, version 1.2*. Institute for Natural Language Processing, University of Stuttgart, 2001. URL `http://www.ims.uni-stuttgart.de/phonetik/synthesis`.

M. Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.

M. Mohri. Weighted automata algorithms. In *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science, pages 213–254. Springer, Berlin, 2009.

M. Mohri and R. Sproat. An efficient compiler for weighted rewrite rules. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 231–238, June 1996. doi: 10.3115/981863.981894.

M. Mohri, F. C. N. Pereira, and M. Riley. Weighted automata in text and speech processing. In *Proceedings of the 12th biennial European Conference on Artificial Intelligence (ECAI-96), Workshop on Extended Finite State Models of Language*, Chichester, 1996. John Wiley and Sons.

M. Mohri, F. C. N. Pereira, and M. Riley. Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16(1):69–88, 2002.

S. Mori, D. Takuma, and G. Kurata. Phoneme-to-text transcription system with an infinite vocabulary. In *Proceedings COLING-2006*, pages 729–736,

Sydney, Australia, July 2006. Association for Computational Linguistics. doi: 10.3115/1220175.1220267.

G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

K. Oflazer. Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics*, 22 (1):73–89, 1996.

K. Oflazer and C. Güzey. Spelling correction in agglutinative languages. In *Proceedings ANLP-94*, pages 194–195, 1994.

F. C. N. Pereira and M. D. Riley. Speech recognition by composition of weighted finite automata. In Roche and Schabes (1997), pages 431–453.

L. Philips. Hanging on the metaphone. *Computer Language*, 7(12):39, December 1990.

L. Philips. The double metaphone search algorithm. *C/C++ Users Journal*, June 2000, June 2000.

T. Pilz, W. Luther, N. Fuhr, and U. Ammon. Rule-based search in text databases with nonstandard orthography. *Literary and Linguistic Computing*, 21(3):179–186, 2006. doi: 10.1093/llc/fql020.

T. Pilz, A. Ernst-Gerlach, S. Kempken, P. Rayson, and D. Archer. The identification of spelling variants in English and German historical texts: manual or automatic? *Literary and Linguistic Computing*, 23(1), 2008. doi: 10.1093/llc/fqm044.

T. Pirinen and K. Lindén. Finite-state spell-checking with weighted language and error models. To appear, 2010. URL `http://www.helsinki.fi/~tapirine/publications/Pirinen-lrec-2010.pdf`.

J. J. Pollock and A. Zamora. Automatic spelling correction in scientific and scholarly text. *Communications of the ACM*, 27:358–368, 1984.

M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

N. Porter, editor. *Webster's Revised Unabridged Dictionary*. G. & C. Merriam Co., 1913. URL `http://machaut.uchicago.edu/websters`.

H. J. Postel. Die Kölner Phonetik. Ein Verfahren zur Identifizierung von Personennamen auf der Grundlage der Gestaltanalyse. *IBM-Nachrichten*, 19:925–931, 1969.

L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

E. S. Raymond, editor. *Jargon File version 4.4.7*. `http://catb.org/jargon/html/`, 2010.

P. Rayson, D. Archer, and N. Smith. VARD versus Word: A comparsion of the UCREL variant detector and modern spell checkers on English historical corpora. In *Proceedings of the Corpus Linguistics 2005 conference*, Birmingham, UK, July 14-17 2005.

U. Reffle, A. Gotscharek, C. Ringlstetter, and K. U. Schulz. Successfully detecting and correcting false friends using channel profiles. *Int. J. Doc. Anal. Recognit.*, 12:165–174, October 2009. doi: 10.1007/s10032-009-0091-y.

E. S. Ristad and P. N. Yianilos. Learning string edit distance. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 20(5):522–532, May 1998.

B. Roark and R. Sproat. *Computational Approaches to Morphology and Syntax*. Oxford University Press, 2007.

A. M. Robertson and P. Willett. A comparison of spelling-correction methods for the identification of word forms in historical text databases. *Literary and Linguistic Computing*, 8(3):143–152, 1993.

E. Roche. Parsing with finite-state transducers. In Roche and Schabes (1997).

E. Roche and Y. Schabes, editors. *Finite-State Language Processing*. MIT Press, Cambridge, MA, 1997.

R. C. Russell. Soundex coding system. *United States Patent* 1,261,167, 1918.

R. C. Russell. Soundex coding system. *United States Patent* 1,435,663, 1922.

A. Schiller, S. Teufel, and C. Thielen. Guidelines fur das Tagging deutscher Textcorpora mit STTS. Technical report, University of Stuttgart, Institut für maschinelle Sprachverarbeitung and University of Tübingen, Seminar für Sprachwissenschaft, 1995.

H. Schmid. Probabilistic part-of-speech tagging using decision trees. In *International Conference on New Methods in Language Processing*, pages 44–49, Manchester, UK, 1994.

K. U. Schulz and S. Mihov. Fast string correction with Levenshtein-automata. *International Journal of Document Analysis and Recognition*, 5:67–85, 2002.

M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.

M. P. Schützenberger. On the algebraic theory of automata. In W. A. Kalenich, editor, *Information Processing 65: Proceedings of IFIP Congress 1965*, pages 27–29. Spartan Books, 1965.

C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

I. Simon. The nondeterministic complexity of finite automata. Technical Report RT-MAP-8073, Instituto de Matemática e Estatística da Universidade de São Paulo, 1987.

A. Sokirko. A technical overview of DWDS/dialing concordance. Talk delivered at the meeting *Computational linguistics and intellectual technologies*, Protvino, Russia, 2003. URL `http://www.aot.ru/docs/OverviewOfConcordance.htm`.

H. Speer and A. Deutsch, editors. *Deutsches Rechtswörterbuch*. Verlag Hermann Böhlaus Nachfolger, Weimar, 2010.

A. Storrer, A. Geyken, A. Siebert, and K.-M. Würzner, editors. *Text Resources and Lexical Knowledge*. Mouton de Gruyter, Berlin, 2008.

P. Taylor, A. W. Black, and R. J. Caley. The architecture of the Festival speech synthesis system. In *Proceedings of the Third International Workshop on Speech Synthesis*, 1998.

E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64(1-3):100–118, 1985. doi: 10.1016/S0019-9958(85)80046-2.

E. Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, 92(1):191–211, 1992. doi: 10.1016/0304-3975(92)90143-4.

Unicode Consortium. *The Unicode Standard*. The Unicode Consortium, Mountain View, CA, 2011. ISBN 978-1-936213-01-6. URL `http://www.unicode.org/versions/Unicode6.0.0/`.

C. J. van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, 1979.

S. Verberne. *Context-sensitive spell checking based on word trigram probabilities*. Masters thesis, University of Nijmegen, 2002.

E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco. Probabilistic finite-state machines – Part II. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:1026–1039, 2005. doi: 10.1109/TPAMI.2005.148.

A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, pages 260–269, April 1967.

R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

D. Yarowsky and R. Wicentowski. Minimally supervised morphological analysis by multimodal alignment. In K. Vijay-Shanker and C.-N. Huang, editors, *Proceedings of the 38th Meeting of the Association for Computational Linguistics*, pages 207–216, Hong Kong, October 2000.

E. M. Zamora, J. J. Pollock, and A. Zamora. The use of trigram analysis for spelling error detection. In *Information Processing and Management*, pages 305–316, 1981.

A. Zeldes. *Data-Based Methods for Historical Grammar and Lexicon Extraction in a Diachronic Corpus*. M.A. thesis, Humboldt Universität zu Berlin, 2007.

A. Zielinski, C. Simon, and T. Wittl. Morphisto: Service-oriented open source morphology for German. In C. Mahlow and M. Piotrowski, editors, *State of the Art in Computational Morphology*, pages 64–75. Springer, Berlin, 2009. doi: 10.1007/978-3-642-04131-0_5.

J. Zobel and P. Dart. Finding approximate matches in large lexicons. *Software – Practice and Experience*, 25(3):331–345, 1995.

J. Zobel and P. Dart. Phonetic string matching: lessons from information retrieval. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 166–172, New York, 1996. ACM. doi: 10.1145/243199.243258.

J. Zobel, A. Moffat, and R. Sacks-Davis. Searching large lexicons for partially specified terms using compressed inverted files. In *Proc. International Conference on Very Large Databases*, pages 290–301. Morgan Kaufmann, 1993.