Department of Modern Languages
2014

Weighted Finite-State Methods for Spell-Checking and Correction

Tommi A Pirinen

Academic dissertation to be publicly discussed, by due permission of the Faculty of Arts at the University of Helsinki in auditorium XII (in lecture room PIII), on the 28^{th} of February at 12 o'clock.

University of Helsinki
Finland

Supervisor
Krister Lindén, Helsingin yliopisto, Finland

Pre-examiners
Prof. Mikel L. Forcada, Universitat d'Alacant, Spain
Prof. Lars Borin, Göteborgs universitet, Sweden

Opponent
Prof. Lars Borin, Göteborgs universitet, Sweden

Custos
Prof. Lauri Carlson, Helsingin yliopisto, Finland

Contact information

Department of Modern Languages
P.O. Box 24 (Unioninkatu 40)
FI-00014 University of Helsinki
Finland

Email address: postmaster@helsinki.fi
URL: http://www.helsinki.fi/
Telephone: +358 9 1911, telefax: +358 9 191 51120

ISBN 978-952-10-9694-5 (printed)
ISBN 978-952-10-9695-2 (PDF)
Computing Reviews (1998) Classification: F.1.1, I.2.7, I.7.1
Helsinki 2014
Picaset Oy

Weighted Finite-State Methods for Spell-Checking and Correction

Tommi A Pirinen

Department of Modern Languages
P.O. Box 24, FI-00014 University of Helsinki, Finland
tommi.pirinen@helsinki.fi
http://www.helsinki.fi/%7etapirine

PhD Thesis, Series of Publications, 2014
Helsinki, January 2014, 94 pages
ISBN 978-952-10-9694-5 (printed)
ISBN 978-952-10-9695-2 (PDF)

Abstract

This dissertation is a large-scale study of spell-checking and correction using finite-state technology. Finite-state spell-checking is a key method for handling morphologically complex languages in a computationally efficient manner. This dissertation discusses the technological and practical considerations that are required for finite-state spell-checkers to be at the same level as state-of-the-art non-finite-state spell-checkers.

Three aspects of spell-checking are considered in the thesis: modelling of correctly written words and word-forms with finite-state language models, applying statistical information to finite-state language models with a specific focus on morphologically complex languages, and modelling misspellings and typing errors using finite-state automata-based error models.

The usability of finite-state spell-checkers as a viable alternative to traditional non-finite-state solutions is demonstrated in a large-scale evaluation of spell-checking speed and the quality using languages with morphologically different natures. The selected languages display a full range of typological complexity, from isolating English to polysynthetic Greenlandic with agglutinative Finnish and the Saami languages somewhere in between.

Tiivistelmä

Tässä väitöskirjassa tutkin äärellistilaisten menetelmien käyttöä oikaisuluvussa. Äärellistilaiset menetelmät mahdollistavat sananmuodostukseltaan monimutkaisempien kielten, kuten suomen tai grönlannin, sanaston sujuvan käsittelyn oikaisulukusovelluksis-

sa. Käsittelen tutkielmassani tieteellisiä ja käytännöllisiä toteutuksia, jotka ovat tarpeen, jotta tällaisia sananmuodostukseltaan monimutkallisempia kieliä voisi käsitellä oikaisuluvussa yhtä tehokkaasti kuin yksinkertaisempia kieliä, kuten englantia tai muita indoeurooppalaisia kieliä nyt käsitellään.

Tutkielmassa esitellään kolme keskeistä tutkimusongelmaa, jotka koskevat oikaisuluvun toteuttamista sanarakenteeltaan monimutkaisemmille kielille: miten mallintaa oikeinkirjoitetut sanamuodot äärellistilaisin mallein, miten soveltaa tilastollista mallinnusta monimutkaisiin sanarakenteisiin kuten yhdyssanoihin, ja miten mallintaa kirjoitusvirheitä äärellistilaisin mentelmin.

Tutkielman tuloksena esitän äärellistilaisia oikaisulukumenetelmiä soveltuvana vaihtoehtona nykyisille oikaisulukimille, tämän todisteena esitän mittaustuloksia, jotka näyttävät, että käyttämäni menetelmät toimivat niin rakenteellisesti yksinkertaisille kielille kuten englannille yhtä hyvin kuin nykyiset menetelmät että rakenteellisesti monimutkaisemmille kielille kuten suomelle, saamelle ja jopa grönlannille riittävän hyvin tullakseen käytetyksi tyypillisissä oikaisulukimissa.

Computing Reviews (1998) Categories and Subject Descriptors:
F.1.1 Finite-State Automata
I.3.1 Natural Language Processing
I.7.1 Spelling

General Terms:
thesis, finite-state, spell-checking, language model, morphology

Additional Key Words and Phrases:
statistical language models, morphologically complex languages

Preface

Spell-checkers are very basic-level, commonly used practical programs. They ar ubiquitous enough that nowadays they are in everyday use for most of us, whether in office software suites, Facebook text fields or a mobile phone autocorrect utility. The practical motivation for this thesis comes first and foremost from the disappointment in contemporary applications for smaller, more complex languages than English, such as my native language, Finnish. Why does my telephone not know *my Finnish*? Even I could implement this better! And so I did.

The scientific background of the thesis builds on the research of the computational linguistics scene at the University of Helsinki, starting from my master's thesis work, which was a finite-state model of the Finnish language. Using this language model as part of a practical piece of end-user software, and advancing towards the scientific treatment of problems when doing so, was a very obvious and motivating incentive for a doctoral dissertation. While the hard science behind the thesis is, in my opinion, delightfully simple, I believe the findings are interesting for anyone working in computational linguistics, or even natural language engineering, especially when it concerns languages different from English, in terms of their complexity, availability of resources, and so on.

The improvements on, and methods for, spell-checking and correction presented in the thesis should provide a good base for more efficient spell-checking in the future, but they are not yet included in popular end-user applications – because I believe the productification and social engineering required to make it happen is not part of the research work. The current approaches have been tested on colleagues, Helsinki students, and a select few brave alpha-testers from Greenland, whose native language is well-known to be trickier to handle than most languages. As the results of the testing were encouraging, we can hope that the future brings better spell-checkers and autocorrections for those of us with slightly more tricky native languages.

Acknowledgements

Official thanks go to the FIN-CLARIN project for funding my research, and to Yliopistotutkintolautakunta (University Exmaination Board), for providing spelling error ma-

terials.

On the academic level, there is an endless number of colleagues and acquaintances met in numerous conferences, workshops and other academic events throughout the world who have contributed to the thesis by giving ideas, challenging my theories and writing supporting or rivalling software and methods. I am grateful to each and everyone. What follows is a list of specific contributions I can recall at the moment, by no means an exhaustive list of thanks, and in no particular order. The first obvious contribution for the whole thesis project comes from the research group and other colleagues at the University of Helsinki, the HFST project for working on software, and the FinnTreeBank and Finnish Wordnet project for providing that extra bit of help with Finnish language model building. Specifically I would like to thank my original master's thesis advisor Kimmo Koskenniemi for introducing me to this line of work, and my PhD thesis advisor and HFST project leader Krister Lindén for keeping the goal in sight throughout the thesis project and helping me manage my time and ideas. In the HFST team I owe gratitude to Sam Hardwick for doing the majority of the software work in the `hfstospell` branch of the project, Miikka Silfverberg for providing the statistical and context-based finite-state models that go beyond my expertise, and Erik Axelson for maintaining the whole software behemoth. I am very grateful to Per Langgård for picking up my contribution in LREC 2011 as a way to get the next version of the Greenlandic speller to people and thus introducing me to the whole intriguing problematic field of the computational modelling of poly-synthetic languages. The thesis and its focus would be quite different without it. I thank Jack Rueter for numerous fruitful discussions on all topics related to Uralic languages, amongst others. I thank the whole of the Divvun / Giellatekno project at the University of Tromsø for the computational morphologies, which we have been slowly turning into spell-checkers, especially Sjur Moshagen and Trond Trosterud for co-operation in this project. I also thank the Apertium project and contributors for another fine source of linguistic materials to turn into spell-checkers, and Francis Tyers for all his help with Apertium-related issues, proofreading, and numerous fruitful discussions. I thank Don Killian for proofreading, and all the discussions on linguistic aspects of my studies. Finally, I would like to thank the pre-examiners for their insightful feedback.

On a personal level I am thankful to my family, my friends at the university of Helsinki, especially those in student organisations like Aspekti and Hyrmy, and the one deity who shares the initials with finite-state machines, FSM – without them, none of this would have been possible.

Original Papers

This thesis consists of an introduction and the following peer-reviewed publications, which are referred to as Papers I–X in the text. These publications are reproduced at the end of the print version of the thesis.

I Krister Lindén and Tommi Pirinen. Weighted finite-state morphological analysis of Finnish compounding with HFST-LEXC. In *Nodalida*, Odense, Denmark, 2009.

II Krister Lindén and Tommi Pirinen. Weighting finite-state morphological analyzers using HFST tools. In *Finite-State Methods in Natural Language Processing*, Pretoria, South Africa, 2009.

III Tommi A Pirinen and Krister Lindén. Finite-state spell-checking with weighted language and error models. In *Proceedings of the Seventh SaLTMiL workshop on creation and use of basic lexical resources for less-resourced languages*, Valletta, Malta, 2010.

IV Tommi A Pirinen and Krister Lindén. Building and using existing Hunspell dictionaries and TEX hyphenators as finite-state automata. In *Proccedings of Computational Linguistics - Applications, 2010*, Wisła, Poland, 2010.

V Tommi A Pirinen and Krister Lindén. Creating and weighting Hunspell dictionaries as finite-state automata. *Investigationes Linguisticæ*, 21, 2010.

VI Tommi A Pirinen. Modularisation of Finnish finite-state language description—towards wide collaboration in open-source development of morphological analyser. In *Proceedings of Nodalida*, volume 18 of *NEALT proceedings*, Rīga, Latvia, 2011.

VII Tommi A Pirinen and Francis M. Tyers. Compiling apertium morphological dictionaries with HFST and using them in HFST applications. *Language Technology for Normalisation of Less-Resourced Languages*, 2012.

VIII Tommi A Pirinen, Miikka Silfverberg, and Krister Lindén. Improving finite-state spell-checker suggestions with part of speech n-grams. In *Computational Linguistics and Intelligent Text Processing* 13^{th} *International Conference*, 2012.

IX Tommi A Pirinen and Sam Hardwick. Effect of language and error models on efficiency of finite-state spell-checking. In *Proceedings of FSMNLP*, Donostia–San Sebastían, Spain, 2012.

X Tommi A Pirinen. Quality and Speed Trade-Offs in Finite-State Spell-Checking and Correction, *forthcoming*

The implementations and data for reproducing the results are available in the version control system of the HFST project.[1]

[1] `http://svn.code.sf.net/p/hfst/code/trunk/articles/`

Contents

List of Tables

List of Figures

Glossary

computational linguistics is a branch of language technology leaning towards the linguistic aspects of the computational handling of language.

error model is any software implementation or formal language capable of enumerating a sorted list of correctly spelled word forms given a potentially misspelled word form.

finite-state acceptor is a finite-state machine, where symbols are single characters of a language, used in the context of this dissertation to encode dictionaries.

finite-state automaton is a graph-based data structure used in computational linguistics to encode a number of linguistic structures efficiently.

formal language is a language in terms of mathematics, that is, an arbitrary set of sequences of symbols.

formalism is used by many computational linguists including myself to refer to scripting languages and data formats used by a natural language processing system. For example, lexc, twolc and xfst together make a Xerox formalism for Finite-state morphologies.

grammar-checker is software capable of detecting grammar errors in word forms and their combinations, and their relation to whole sentences.

language model is any software implementation or formal language capable of describing whether a given word form *s* is correct in terms of spell-checking or not. This usage differs slightly from established usage in natural language engineering, where the language model refers to a statistical n-gram model, which is just one sub-class of my usage.

morph is a minimal indivisible segment split from a word form carrying meaning or purpose for a computational linguistic description; a morph can be segmented out of a word form in a text.

morpheme is an abstract concept grouping related morphs together by means of semantic, phonological or otherwise practical similarity in a manner that is usable for a computational linguistic description; a morpheme is not a concrete segment of a running text.

morphological complexity is used in this dissertation in a very limited sense of complexity of a language that is relevant to spell-checking and correction and manifested in terms of the number and predictability of word forms the language contains.

morphology is a branch of linguistics studying word formation, the structure of word forms.

morphotactics is a set of rules governing what combinations of morphs form correctly spelled word forms; also: morphotax.

natural language is a language spoken, written or otherwise used by people as means of communication.

natural language engineering is a branch of language technology leaning towards the software engineering aspects of the computational handling of language.

non-word error a spelling error where the mistyped string is not a valid word form in a dictionary of the language.

real-word error a spelling error where the mistyped string is another valid word form in a dictionary of the language.

scraping is a crude technical method of obtaining data from text corpora for computational linguistic systems, usually harvesting all potential linguistic data without human intervention.

spell-checker is software capable of detecting and correcting spelling errors in word forms.

spell-checking is the task of verifying that the word forms of the text are correctly written word forms in the language of the spell-checker. Spell-checker can, however, refer to software capable of both spell-checking and correction.

spelling correction is the task of correcting misspelled word forms in the text by correct ones, or suggesting alternatives in an interactive application.

tropical semi-ring is a mathematical, algebraic structure used in weighted finite-state automata to encode probabilities or penalty weights.

word form is a string in text separated by white space or punctuation that is not part of a word in a given language. This definition stems from how spell-checking libraries actually work.

$\mathcal{D}$ is a collection of data, such as a set of word forms, morphs, typing errors. I use subscript indices to describe the collection. E.g., $\mathcal{D}_{\text{wordforms}}$ for all word forms in a text corpus, $\mathcal{D}_{\text{errors}}$ for all typing errors in an error corpus, $\mathcal{D}_{\text{morphs}}$ for all morphs in a natural language.

$\mathcal{M}$ is an arbitrary finite-state machine. I use subscript indices to describe the type of automaton. E.g., $\mathcal{M}_{\text{L}}$ for language model, $\mathcal{M}_{\text{E}}$ for error model, $\mathcal{M}_{\text{things}}$ for an automaton encoding the disjunction of all things.

Acronyms

FLOSS free and libre open-source software.

FSA finite-state automaton.

FST finite-state transducer.

HFST Helsinki finite-state technology.

WFST weighted finite-state transducer.

Chapter 1

Introduction

Spell-checking and *correction* are among the more practical, well-understood subjects in *computational linguistics*, and earlier, computer science. The task of detecting spelling mistakes in different texts – written, scanned, or otherwise – is a very simple concept to grasp. Moreover, the computational handling of the problem has been the subject of research for almost as long as computers have been capable of processing texts, with the first influential academic works published in the 1960's, e.g. Damerau (1964), and the field has developed greatly since.

The purpose of this thesis is to research one specific approach to spell-checking – that of finite-state technology. Finite-state technology has its roots in the mathematical theory of *formal languages*, which I will refer to as *string sets* throughout this thesis to avoid confusion with the more common meaning of the term, *natural languages*. The theoretical beauty of this approach will be recapped more closely in the later subsections. The practical circumstances of this approach are the following: it gained popularity among computational linguists interested in *morphologically complex* languages around the 1980's and onwards, and it is thought by some of us to be the only way to handle more morphologically complex languages. This sets the direction of this thesis to *implement spell-checking for languages of varying morphological complexity* efficiently.

Finite-state approaches have traditionally been popular for the computational handling of Finnish, cf. Koskenniemi (1983). Finnish is my native language and for practical reasons a recurring theme in this thesis. Whilst concentrating on scientific contributions made over the years, this thesis is in a way also a book which describes the past years of the development of, language independent, *spell-checker* software[1] which is also used in the writing of this very thesis. So, it should be clear that the contribution of this thesis is meant to be a very concrete, should I even say a final – albeit beta-quality – system for spell-checking and correction. There are some practical implications when using one's native language to evaluate and develop new and improved methods, and the motivation

[1]The software is available as free and open source, like all good scientific research products.

for spending immeasurable amounts of spare time is there.

One important practical feature of this thesis is that the end product the scientific methods are built for is usable and available as an end-user spell-checker at some point in the future. For this reason I aim to ground the theoretical advances so that they are usable for a larger audience of end users and this includes limiting resources required to build the systems experimented with to freely-available and open-source materials and techniques. Indeed, if a spell-checker sub-system is mentioned in this thesis, you should at least be able to download a functional prototype. All our software and data is available in our version management system.[2] [3] I also strongly believe that this approach entailing not only *free and libre open-source software (FLOSS)* for the scientific software and results, but also properly conducted open scientific research will fulfil the basic requirement of reproducibility.

Another perspective in the thesis is the quality of the spell-checking for *morphologically complex* languages. For English, the vast majority of research on the topic has already shown impressive figures for the quality of spell-checking and corrections with statistical approaches. It almost seems like a scientific consensus on the solution. The common intuition, however, raises some suspicion that the morphologically more complex languages may not be so favourable to simple statistical approaches. The contribution of this thesis is also in showing the limits of these approaches for morphologically complex, and resource-poor languages, exploring possible modifications that can be used to salvage the situation.

The remainder of this first chapter is dedicated to an informal introduction to the thesis topic. It consists of closer definitions of the practical and scientific concepts central to building a spell-checking system as a research problem, some very down-to-earth rationals for this approach, and the background details that motivate the research of finite-state methods for spell-checking even when spell-checking itself has already been almost researched to death. Finally, as this is an article-based thesis, I give an overview of the articles of the thesis, and describe how the thesis is structured.

1.1 Components of Spell-Checking and Correction

Spell-checking and correction, as the title suggests, can already be separated into two components. Furthermore both of the components can be divided, classified and analysed into many separate pieces. In this section I try to cover different existing and practical conceptualisations, and introduce the relevant terminology in a neat and systematic manner.

The primary division of labour that is present throughout this thesis, and is shown in the title, is the division into spell-checking and correction. *Spell-checking* is the task of

[2] `http://svn.code.sf.net/p/hfst`

[3] A mono-spaced font is used for URLs and program code segments.

detecting errors in texts, whereas *spelling correction* is the task of *suggesting* the most likely correct *word forms* to rpelace the error. It is of great importance to treat these two tasks as separate, as the implementations made for each of them can be quite freely mixed and matched. While there are systems that use the same methods and approaches for both, this is not by any means a necessity. A practical system giving suggestions usually does not have access to the context of the word form it is correcting and to the criteria by which the word form was chosen. These are relevant constraints in some of the approaches I present in the thesis and for some of the concepts I present in this section.

The methods for detecting errors can be divided according to the data they use for evidence of the word being *misspelt*. The basic division is whether the system looks only at the *word form* itself, in an *isolated* spelling error detection, or if it actually looks at the *context* as well. Traditionally, the isolated error detection has been based on lookup from word form lists to check if the word form is valid. The only additional notion I have in terms of this thesis, talking about finite-state spell-checking, is that our *dictionaries* are systems which are capable of representing an infinite number of word forms, i.e., they contain not only the dictionary words, but also the derivations and compounds necessary to reasonably spell-check a morphologically complex language. Sometimes, this spell-checking is backed up with statistical information about words: the words that are in a dictionary, but are very rare, might be marked as errors if they are one typing error away from a very common word, such as lave,[4] [5] instead of *leave* (Kukich, 1992b). The spelling errors of this kind, where the resulting word is an existing word in the dictionary, are called *real-word errors*. The shortcoming of isolated approaches to spell-checking is that they will basically only recognise those spelling errors which result in a word form that is not in the dictionary, so-called *non-word errors*. Real-word spelling errors will almost always require the use of a context to obtain some evidence that something is wrong. Here again, the simplest approach is to use statistics; if the word we are inspecting does not usually appear in the current context of two to three words, it may be an error. A more elaborate context-based method for detecting errors is to use a full-fledged natural language processing system that can parse morphology, syntax or other features of the running text. Such a system can usually recognise sentences or phrases that are unusual, or even grammatically wrong; these systems are usually called grammar-checkers,[6] rather than spelling checkers, and are not a central topic of this

[4] I will use this special emphasis for misspellings throughout the thesis as most of the users of contemporary spell-checkers should be familiar with it.

[5] *lave* in case you were as unfamiliar with this rare word as I am, it is defined in Wiktionary as: "1. (obsolete) To pour or throw out, as water; lade out; bail; bail out. 2. To draw, as water; drink in. 3. To give bountifully; lavish. 4. To run down or gutter, as a candle. 5. (dialectal) To hang or flap down. 6. (archaic) To wash." `http://en.wiktionary.org/w/index.php?title=lave&oldid=21286981`

[6] A word of warning about this terminology: some practical systems, such as office software, will call any spelling checker that uses any context-based approach, as opposed to an isolated approach, a grammar

thesis, although I do try to clarify what kind of extensions could be made to turn my spell-checkers into such grammar-checkers.

The error-correction task is divided most naturally by the types of errors that are made. Many different classifications with different names exist, but most will agree that there is a category of errors that is based on unintentional mistakes, such as mistyping at the keyboard. This is the type of spelling error that is the main target for correction suggestion systems, and the common research results have claimed that between 80 and 95% of all spelling errors can be attributed to having *a single* typing error in a word form (Kukich, 1992b). The rest of the spelling errors are more or less based on *competence*. Such errors can be caused by not knowing how to spell a word form, which is especially common in orthographies like English, or not knowing the correct way to inflect a word, which is common when writing in a non-native language. Increasingly common are errors caused by limited input methods, such as dropping accents or otherwise transcribing text to match e.g. a virtual keyboard of a mobile phone – generally these errors are indistinguishable from typos.

1.2 Morphological Complexity

Morphology is the subfield of linguistics that deals with the structure of a word form. A word form, for the purposes of this dissertation, is defined as a segment of a text that is separated by whitespace symbols or other similar orthographic conventions. While from a linguistic point o -view this definition is not without problems, it is one that today's spell-checking software has to cope with. In fact, many spell-checkers today are restricted to handling word forms that some other software has decided are word forms like this. To give one example, in English 'cat' is a word form, as is 'dogs', or 'involuntarily'. The structure of a word form is composed of one or more morphs. A morph is a segment of a word that has been defined by linguistic theories as a minimal unit carrying meaning, or more vaguely, purpose. In 'cat' there is one morph, the root word meaning cat, in 'dogs' there is the root morph 'dog' and another morph 's' marking plurality in English. In the word 'foxes', there is a root morph 'fox' and a plural morph 'es'. Sometimes when describing morphology, abstractions called *morphemes* are used to group similar morphs together, for example in English it could be said based on these examples that the plural morpheme is '(e)s'.

Complexity in morphological features, is studied in the field of linguistic *typology*. There are no commonly agreed measures of morphological complexity, and this dissertation does not intend to be a contribution to typological studies. Rather, I try, as a practical issue, to define morphological complexity as it relates to spell-checking and

checker. This terminological confusion is not carried over to Microsoft Word's recent incarnations, but at the time of writing Apache OpenOffice.Org and LibreOffice still follow the confused terminology.

correction. In this section I will thus attempt to explain my understanding of the issue, partially because I have used terms such as *morphologically complex* in the thesis articles. It is therefore necessary to remember to read the word pair *morphologically complex* in this thesis as a concept which refers to aspects of languages' writing systems, dictionary formation and such features described in this section, rather than as contested definitions in linguistics at large.

One theme of my thesis was to bring effective and high-quality spell-checking and correction to languages that are of varying *morphological complexity*. To substantiate this claim I will first try to define what I mean by morphological complexity. There is a measure of complexity that affects the building and use of correct word form recognisers in terms of computational and practical resources, and there are only a few easily measurable factors in this playing field. In linguistics studies began with Greenberg (1960) and continued up to the contemporary endeavour known as the world atlas of language structures (Haspelmath et al., 2005), which has been widely used for measurements of similarity, complexity, and so on. The morphological complexity of a computational spell-checker is based on, e.g., the complexity of the word forms in terms of *morphs*. This is measurable as the average count of morphs in the word or the *morph-to-word* ratio. The morphs I use here refer to the smallest meaningful unit in the word that is used so regularly that it can be practically used in the language description – that is, a productive morph. Another factor of morphological complexity that is important is the variation that creates a set of related morphs. This variation can be realised in terms of the number of *morphemes* per language or *morphs-per-morpheme*. Since a spell-checker's main task is to predict all correct word forms, both the number of potential word forms created by morph chains and the amount of variation within these morphs are good indicators to estimate the computational complexity of that particular spell-checker.

These measures lend themselves nicely to a view of spell-checking software where the languages that have significantly higher values include those that have their special implementations in the world of open-source spell-checkers: Finnish and Voikko,[7] Hungarian and Hunspell,[8] Turkish and Zemberek,[9] and so forth. This tendency suggests that people trying to build spell-checkers with the limitations of almost only word-list or statistics-based systems will not be able to reach satisfying results, and other approaches are in fact necessary.

Perhaps the most important form of morphological complexity relevant to the topic of this thesis, and one that is often ignored, is the rate of productive derivation and compounding that produces new *ad hoc* words and word forms that are difficult to predict using finite word lists or even simple finite combinations of words and morphs pasted together. For a language where these processes are common, spell-checking greatly

[7] `http://voikko.puimula.org`
[8] `http://hunspell.sf.net`
[9] `http://code.google.com/p/zemberek/`

benefits from a dictionary capable of predicting a large number of words which have not been seen before. Surprisingly many of both existing systems, and those described in recent research, e.g. in Hassan et al. (2008); Watson (2003), simply do not acknowledge the need. Some gloss over the fact as a possible extension to the system, some ignore it altogether, but usually without any example of an implementation and an evaluation strictly based on English or a few other popular Indo-European languages with similar features. For many languages an easy way to produce dictionaries with reasonable coverage is a system which predicts infinite amounts of compound and derivation combinations.

To illustrate the practical issues with morphologically complex languages, we use the *de facto* open-source spell-checking system, Hunspell. Of the aforementioned morphological features, Hunspell supports up to 2 affixes per word, or 3 morphemes. For languages going beyond that, including Hungarian, the creator of a dictionary will at least need to combine multiple morphemes into one to create a working system. The morpheme count in Hunspell is realised as affix sets, of which Hunspell supports up to 65,000 (the exact number supported may vary somewhat in different versions, but it seems to be less than 65,535, which would have been expected for the two-byte coding of suffix flags that was used in the system). Hunspell does, however, include support for some compounding.

As a concrete example of limitations with e.g. Hunspell, there has been at least one failed attempt to build a spelling-checker for Finnish using Hunspell (Pitkänen, 2006), whereas we know from the early days of the finite-state methods in computational linguistics that Finnish is implementable, in fact it has often been suggested that the initial presentation of the automatic morphological analysis of Finnish by Koskenniemi (1983) is one of the main factors in the popularity of finite-state methods among many related languages. Many of the languages with similar morphological features, however, have been also implemented in the Hunspell *formalism*, indeed as even the name suggests, the formalism originated as a set of extensions to older *spell[10] formalisms to support Hungarian. For North Saami, a language from Uralic family I commonly use for test cases to compare with Finnish, an implementation of a spell-checker for Hunspell exists.[11] The languages that fall into the more complex side of the morphological complexity are usually noted as very hard to properly implement with the current Hunspell formalism and its limitations, though often the results are found tolerable. E.g. Gīkūyū Chege et al. (2010) used 19,000 words with extensive affix rules as a spell-checker, for which they say they attained a recall of 84%. The same does not seem to apply for Greenlandic, where it was found that a collection of 350,000 word forms only covered 25% of the word forms in newspaper texts.[12]

[10] Specifically, myspell, c.f. `http://www.openoffice.org/lingucomponent/dictionary.html`

[11] `http://divvun.no`

[12] `http://oqaaserpassualeriffik.org/a-bit-of-history/`

The sizes of wordlists lead to the final, perhaps most practical metric of measurement in the morphological complexity of languages in the context of computational applications. That is, the size of the dictionary when it is in the memory, whatever may be the method of encoding it or compressing it; this is easily measurable and a very important factor for any practical application. For example for a word-list approach, even uncompressed the wordlist of a million words with an average word length of N is just N megabytes, whereas storing the list as a finite-state automaton, suffix trie or hash is considerably less. Even though most wordlist-based software do store the data in one of the more advanced data structures, it is the limitations of Hunspell such as the two-affix limit, that will force major parts of wordform lists to be included in an inefficient format on disk, and will greatly hinder the speed of Hunspell operation in practice. With combinations of derivation and compounding the memory requirements tend to go up rapidly, even with finite-state approaches, so there is a practical correlation measure between the complexity of the language defined in theoretical terms earlier; in a worst case scenario this is obviously a limiting factor, e.g. some versions of my Greenlandic experiment with finite-state automata took up almost 3 gigabytes of memory, which is clearly unacceptable for a spelling checker in an average 2013 end-user computer system.

1.2.1 On the Infinity of a Dictionary

While I have tried my best to avoid hotly debated aspects of *morphological complexity*, there is one central differentiating factor between the finite-state approaches and wordlist approaches to language that I cannot avoid, and that is the expression power capable of predicting an infinite number of word forms. The fact that cyclic automata encode infinite lexicons is mentioned numerous times in the articles of my thesis and a few of them specifically deal with the problems arising. I provide here some rationale why I consider infinity a necessary feature.

Now, the concept of infinity in itself, as an abstraction is quite hard to grasp in many cases, especially if you have not studied mathematics of transfinite numbers. The basic example used in school mathematics is a set of numbers, for example, there is an infinite number of non-negative integers $\mathbb{N}_+ = 0, 1, 2, 3, \ldots$, and this is uncontested. Intuitively, it is also somewhat easy to grasp, for example, for any given number you can read out loud, there is always going to be larger numbers, which you can create by adding 1 to it, or any other positive integer for that matter (but not 0, for adding zero will not make a larger number). Now, in Finnish, numbers are written as compounds, that is, without spaces in between the number words. For example, 3 is ‘kolme’, 33 is ‘kolmekymmentäkolme’, and 33333 is ‘kolmekymmentäkolmetuhattakolmesataakolmekymmentäkolme’. So, we have a mapping from non-negative integers to words that belong to a dictionary of the Finnish language, and therefore the size of a Finnish language dictionary is at least equal to the size of the non-negative number set.

In practice, this is still debated, even when using the definition of words and word

forms that only includes space-separated strings, whether this infinity is really needed. The main argument here is that, even if there was a theoretical infinity of these words, we can encode a certain finite subset of them, and always get a virtual 100% coverage of all words that actual people use in real-world texts. And this is a very good argument, and one with which I fully agree. However, especially considering our number example, what is the subset of numbers we can select and be sure that no one will pick a number outside that set and write that as a word? Style guides will usually suggest something along the lines of writing words that have up to five compound parts. However, this is also an infinite set, since only the digits that are non-zero are read as numbers in Finnish, e.g., 3,000,000,000,000,000,000,003 is 'kolmetriljardiakolme'. After that, we can probably trust non-malicious[13] writers to stick with reasonably small numbers, say under one centillion, that is a number that has one, and six hundred zeroes after it.[14] Even with these, quite reasonable limitations, the number of common words to list gets quite large, and we have not even touched the tricky topic that Finnish numeral compounds have case agreement in inflection (Karttunen, 2006). This is one of the places where cyclic, infinite finite-state lexicons really show their power. If you encode these number words as a *finite-state automaton* and allow compounds of up to five components, you need to create five copies of the network, making it five times the size that all the numeral words as individual components need. However, if you make that network cyclic, allowing any number of combinations of number words, even infinite, the size of the network is almost exactly the same as the network that allows only one number word without compounding.[15]

Numerals are not very interesting as a linguistic example, and perhaps also somewhat suspicious, as it is easy to come up with new numbers. However, non-number word compounding is not very different in practice. We have a set of words that can be compounded together, and we have to at least predict which combinations the user might use. Speakers of language come up with new *ad hoc* compounds every day, and making good guesses or limits to this creativity would be a very difficult task. As a good experimental example, in my implementation of Finnish morphology,[16] I have opted to encode all compounds that are attested as "known" compounds. If today I pick up the first article on a Finnish news site,[17] I still get one compound that has not been encoded as such, even af-

[13] I say non-malicious here because certainly someone, such as myself, could write a number consisting of more parts, for example 'kolmesataakolmekymmentäkolmemiljoonaakolmesataakolmekymmentäkolmetuhattakolmesataakolmekymmentäkolme' in a written and published text, such as this one, just to prove a point.

[14] Finnish uses long scale system where systemic large numerals are made of the latin prefix, *mi*, *bi*, *tri*, …, *centi*, …, and either *-(i)llion* or *-(i)lliard* suffix to it.

[15] There is one arc added for the cycle, but if you consider Finnish specifically, some constraints are also needed to control the inflection of compounds; these are detailed in Karttunen (2006).

[16] Open Source Morphology of Finnish, `http://code.google.com/p/omorfi/`

[17] `http://yle.fi/uutiset/miksi_keva_kokousti_kuusi_tuntia_en_kerro_vaikka_kuinka_tenttaat/6950229`

ter *scraping* large corpora like Wikipedia for compounds: “toimitus+johtaja+sopimus” (CEO’s contract). The logic of trying to predict such forms is not very easy and statistics will not help either. We could again make some approximations to cover most of the expected cases, e.g. five total words and perhaps encode some semantics. This is a good approach, however, it requires a lot of manual work and the resulting dictionary will be massive. Including all combinations of five arbitrary nouns requires N^5, where N is the size of the noun dictionary (analogously to amount of digit combinations in five-digit numbers 10^5), and even as a finite-state automaton this is 5 times the size of finite-state automaton of nouns, or, indeed, a nominal finite-state automaton with infinitely long compounds. So all in all, the decision to support infinitely long compounds is based on the fact that it does have better predictive power and takes up less space, and is easier to implement in terms of work required. It is not maximally ideal for spelling correction, a problem we will deal with later in this thesis.

One argument against the infinity of the lexicon is that it is an artefact of the writing system. And this I wholeheartedly agree with. The Finnish example even shows it explicitly. Considering the agreement of inflection with numerals, we have inflection patterns like: “kolme+sataa+kolme” : “kolme-lle+sada-lle+kolme-lle” : “kolme-ksi+sada-ksi+kolme-ksi”, (303, singular nominative, allative and translative forms). This is not different from noun phrases in Finnish, which are written with spaces: “vihreä uni” : “vihreä-lle une-lle” : “vihreä-ksi une-ksi” (green dream, same forms as before). Here we have exactly same agreement features and structure of word combinations, but a space is added between the words. While we cannot change the quirks of writing system conventions very easily, it would make sense to handle both cases in the same way, and by extension, all morphology in the same way regardless of the writing system, which may or may not use whitespace and punctuation to delimit morphological units. This is also a feature I argue for in some of my articles included in this thesis as well. However, the fact remains that contemporary software does not let us work on units of text like that, and especially interfaces for spell-checking in popular real-world software rely on spell-checking to deal with space and punctuation-separated strings. As it is something that cannot be changed by a doctoral dissertation. I am left with the task of dealing with the problems that arise from infinite compounds and from recurring derivation that arise from the writing systems not using spaces or punctuation to separate morphological units. I have, for the most part, attempted to formulate the systems in a manner that will be usable if in the future we have systems and frameworks that let us handle texts as a succession of linguistic units rather than as chunks separated by whitespace.

As a side-note, it is not only compounding as the form of copying and concatenating certain word forms, it is also derivation that can cause loops that go towards infinity. What it means is that the parts of words that attach, e.g., to the beginning or the end of the word, can have long, unpredictable combinations. This is true for example for Greenlandic, where derivation is the contributing factor to the average word length and

thus to *morphological complexity* as I define it in this thesis. Another aspect of derivation is that it interfaces with compounding in a way that makes it clear that it is not just pasting all words together a few times in certain forms that is sufficient to predict all new words. Nor is it pasting together all the morphs in succession that is sufficient to predict new words. What is really needed is rather advanced combinations of morphs and compounds that form new words. For example in Finnish compounds should be made of certain forms of nouns, so one might naively expect to be able to predict all compounds by concatenating those nouns together. However, it is also all the verbs derived into nouns that can appear as parts of compounds, e.g., there is a word *meren-käynti* (seafaring, from *meri* + *käydä* + *-nti* sea + go + suffix for making nouns from verbs). For a more recent example of how productive and creative this is, I analysed some of my text logs from an internet relay chat service called IRC to find multipart compounds with verbs in them, and I found this particular example rather illuminating: *känny-irkkailu* (habitual mobile IRCing, from *känny* + *irkki* + *-ata* + *-illa* + *-u*, literally mobile phone + IRC + suffix to verb nouns with + suffix marking frequent or habitual doing + suffix for making nouns from verbs). This example exhibited a noun neologism derived into a verb, derived into another verb derived into a noun. This kind of creativity is not easily predicted by just concatenating word forms or morphs.

To summarise, I have opted to enable the possibility of infinite compounds and derivations in word forms of dictionaries used in spell-checking in this dissertation for the following reasons. Firstly, contemporary spelling checkers are mainly forced to work with the orthographic concept of space-separated tokens as word forms. Secondly, in finite-state structures it is typically more efficient to work with unbounded combinations. Thirdly, the art of picking and choosing the finite good combinations of compound parts and derivations is a lifetime's work in lexicography (cf. the Greenlandic lexicon's history mentioned earlier in Footnote 12) to reach a sufficient coverage for a spell-checker. For these reasons, and because it seemed sensible for my own work with my Finnish dictionary, I chose to investigate the intriguing problematics of infinite lexicons in finite-state spell-checkers as one major part of my dissertation. I believe I have managed to formulate the research in a manner that will be sustainable even if the world of spell-checking and language technology change to support different definitions of a word and its parts.

1.3 Finite-State Technology in Natural Language Processing

After considering the limitations and problems of wordlists and other simple approaches, we come to the second part of the thesis topic: finite-state methods. In this dissertation I have selected to use *finite-state automata* as a general framework for handling

spell-checking dictionaries because of its success in handling morphologically varied languages (Beesley and Karttunen, 2003). This approach is sufficient to cover the infinite dictionaries of the kind mentioned in the previous section. It is commonly thought that the extents of finite-state technologies are sufficient to cover most of the natural languages' word forms found in the running text, that is, a spell-checker's dictionary implemented with these technologies could know all correctly written word forms without technology being the limiting factor. There are some counter-arguments to this, e.g., Culy (1987) presents features of Bambara vocabulary that would require expressive powers beyond regular and even context-free grammars to predict Bambara's word forms. It seems to me that there are two limitations to this specification that allow finite-state approximations of Bambaran vocabulary: firstly it appears that Culy argues that despite the fact that these lexical elements are written with spaces in between them, they should be handled on the vocabulary side. This already relieves finite-state spell-checking system from the burden of recognising such lexical elements due to the fact that finite-state spell-checkers are forced to operate on space-separated strings. This definition does allow for Bambara to have ungrammatical forms in a spell-checked text, such as *wulunyina o wulufilela* (dog searcher dog watcher) mentioned in the text. This, however is akin to a spell-checker allowing the English phrase *I sees he* or Finnish *punaiselle kukasta* (to red from flower), something that in contemporary software is relayed from spell-checking to grammar checking. The second saving aspect for finite-state technology is its capability to describe all finite subsets of languages higher in the hierarchy. What this means is that, should there be a language having a productive word-formation like Bambara without intervening white spaces, it is still possible to capture the word forms that have less than infinite components in them matched in a manner that would require greater expressive powers. For example, one could create an *finite-state automaton (FSA)* that counts that if there are two or five or thirty-three matching components that fulfil the requirements, it will always be possible to select a number high enough that covers a large enough part of the words used in a real-world context that the spell-checker of such form would be enjoyable to use. There have already been studies on syntax that show that the expressive power can be limited to a rather small number of such dependencies that would otherwise be beyond finite-state technology (Karlsson, 2007).

The latest developments in the finite-state approach to natural language processing has been the concept of weights in finite-state automata. I have experimented with some approaches to bring the expressive power of statistical language models to traditional rule-based dictionaries. One significant part of the contributions in this thesis studies the notions of applying statistical approaches in conjunction with morphologically complex languages using finite-state methods.

The concept of modelling typing errors is not so widely accepted as part of finite-state technology. While for example in speech applications, trying to map a representation of

the spoken audio signal into written language is quite common, it is relatively new in the mapping of typing errors into correctly typed text. The contribution to finite-state methods in this thesis provides further research on the application of finite-state models of typing errors as a practical component in a full spelling correction system.

1.4 Overview of Thesis Articles

This dissertation is a collection of articles I have written during the development of different parts of our finite-state spell-checking system. It covers a wide range of topics, all tied together by the goal of getting things to work nicely, and investigates the quality and speed of spell-checkers for languages like English. My articles, in chronological order are presented in *Original Papers* on page 3.

1.4.1 Chronological Order

The development shown in the articles is easy to follow in **chronological order**, almost tells its own story. In the beginning we have a Finnish morphological analyser (Pirinen, 2008) which cannot yet be turned into a statistical dictionary as it contains too rich a morphological productivity. In articles (I; II) we explore some advanced weighting options of Finnish morphology that could provide better treatment of compounds and derived word forms in morphologically complex languages. In (III) we try to tackle the problem of scarcity of corpus resources especially in conjunction with morphologically complex languages, using a crowd-sourcing option provided by Wikipedia and measuring how smaller text corpora will still improve the spelling correction results. In (IV; V) I have studied the compilation and reuse of existing non-finite-state language descriptions as finite-state automata-based language models of a finite-state spell-checking system. In (V) I attempt to reimplement much of the Hunspell spell-checking system, including their error correction methods, in finite-state form. In (VI) I research the topic of how to maintain finite-state *language models*, and try to show that finite-state language models are feasible for long-term maintenance for a vast array of applications using just one language description instead of one description per application. I extend this theme in (VII) by showing how to convert language models outside of morphological analysers and spell-checkers into a suitable finite-state automaton. In (VIII) I tackle the problematic issue of context-aware spell-checkers for morphologically complex and resource-poor languages, showing some practical limitations and what can be done to get some benefits from such an implementation. In (IX; X) I have already reached the point of full-fledged, workable systems for spell-checking, and test the full power of those systems, first for speed and efficiency in (IX), and then in a larger scale survey (X), the quality of spelling suggestions, as well as extensions to some of the speed measurements. The chronological order is also visible in the schematic diagramme of Figure 1.1. The finite-

state language models (in the leftmost column) were conceptualised after the statistical weighting schemes of morphologically complex language (in the middle column) and the resulting combinations were evaluated on a larger scale (in the rightmost column).

1.4.2 Division of Labour

The **division of labour** within these articles also follows quite naturally from the chronological development. In (I; II), as I was the author of the Finnish finite-state implementation of the morphological analyser we were extending with weight structures, I did much of the groundwork in implementing the practical software for weighting the analyser. The result of this work is the weight support in such parts of our finite-state tool chain as `hfst-lexc` and `hfst-strings2fst` – that is, in practice the compilers for morphemes and other string sets. In these early articles, the evaluation scheme and major parts of the written article are the work of Krister Lindén. In (III) I already take into account that previous experiments were performed using commercial corpora unavailable for scientific, free and open-source work, which led me to devise a statistical training as well as an evaluation scheme based on freely-available Wikipedia data. The evaluation and statistical training setup that was built in this article has then evolved throughout the thesis, in the form of collected `make` recipes, `awk` scripts, `python` scripts and other pieces of `bash` command-line programming. The engineering work on Hunspell language models in (IV; V) originated from the compilation formulas for morphologies, documented e.g. in Lindén et al. (2009), and were constructed by Krister Lindén and myself. The same technique was applied to the building of Apertium system's language models in (VII) with technical help from one of the Apertium system's main contributors, Francis Tyers. The context-based weighted finite-state methods of (VIII) originated from Silfverberg and Lindén (2010), with whom I also co-operated in the porting of the system to finite-state spelling checker use. The evaluation scheme was slightly modified from an earlier one used in (IV). The writing of the article (VIII) was for the main part my own work, whereas the mathematical formulas were built in co-operation with the article's other authors. In (IX), the evaluation setup and article was written by myself, with the major part of the technical implementations done by Sam Hardwick, who is also the main author behind the finite-state spell-checking component of our current software, as is further detailed in Lindén et al. (2011). In the articles (VI; X) the majority of the test setup, article structure, engineering and other tasks were performed by myself, which is reflected in the fact that they are published under my name only. It is, however, true that all of the work within this thesis was carried out as a member in the *Helsinki finite-state technology (HFST)* research group, and there is no doubt that there is an abundance of minor ideas regarding engineering of the software, evaluation setup and article writeup that are the creation of collective brainstorming as well.

1.4.3 My Contributions

My contributions to the co-authored articles can be broken down as follow: In article (I), I implemented a proof-of-concept version of the finite-state lexicon compiler with weighting support and extended the original Finnish description by adding weights of word forms and their boundaries whereas the first author designed the scheme and evaluation system. In (II) we extended this scheme with a similar division of labour, i.e., I finalised the software implementation and worked with language descriptions and the first author designed and evaluated; the writing was shared with the first author mainly doing the overview and conclusion sections. In (III), I made the training corpora and extended the previous evaluation suites to the task; for the spell-checking automata and application procedure both authors contributed as well as the research team. The initial article writing was done by myself including the introductory part and conclusion, and the final version written jointly by both authors. Articles (IV) and (V) were based on finite-state formulations of `hfst-lexc` by the second author and refined for this purpose by both authors with the assistance of the research group. Most of the evaluation scheme was carried over from earlier projects, and was extended and rewritten by myself. The article was structured with the help of the second author, while writing was mainly done by myself, including the new formulations of the algorithms. In article (VII), the original idea was discussed jointly with the authors with myself doing the implementation based on previous work on different compilation algorithms in other tools in earlier articles. The evaluation scheme was taken from previous articles with minor modifications by myself. In article (VIII), we used the second author's algorithms from earlier work on POS tagging combined with my earlier evaluation scheme for spell-checking and the automata used for it. The second author wrote most of the parts concerning mathematics and the description of finite-state n-gram models, and I wrote the evaluation, introduction and conclusion sections. In article (IX), on the effects of weighted finite-state language and error models on speed and efficiency of finite-state spell-checking, I wrote most of the evaluation based on previous work, and the second author implemented the software including the optimisation schemes. The optimisation schemes were constructed jointly with both authors plus the help of the research group in a few project meeting sessions. In practice all articles have been discussed in project group meetings ,and the project has contributed ideas, software pieces, and linguistic descriptions as further detailed in the articles themselves. For exact changelogs one can also read SVN log[18] and for newer articles also git commitlog.[19]

[18] `http://sourceforge.net/p/hfst/code/HEAD/tree/trunk/articles/`
[19] `https://github.com/flammie/purplemonkeydishwasher`

1.4.4 Research Questions

Another way to conceptualise the thesis is under the following **research questions**: How to implement reasonable statistical language models for finite-state spell-checking, how to implement a finite-state equivalent of state-of-the-art non-finite-state spell-checking, how to develop and maintain a language model for finite-state spell-checking, what is the quality of finite-state spell-checking compared with a string-algorithm approach, what is the speed of the finite-state spell-checking compared with another software based-approach, what are the limitations of the finite-state-based approaches compared with string-algorithm spell-checking – and doing all of this with morphologically complex languages that are lesser-resourced.

1.4.5 Chapter Structure

The rest of the thesis is **organised** by topic **into chapters** as follows: In Chapter 2 I summarise prior research on spell-checking and correction, and describe the development of actual in-use spell-checkers. I then survey the previous research specifically on finite-state approaches to spell-checking, and describe the theory of finite-state spell-checking in terms of our specific implementation. Chapters 3—6 contain the research papers of the thesis sorted under four headings that match the original goals of this thesis. In Chapter 3, I go through the existing spell-checking and other language models and their use as part of the spell-checking system, and introduce the finite-state point of view to the language models that were initially non-finite-state. In Chapter 4, I describe some weighted finite-state methods to introduce statistics into the language and error models. In Chapter 5 I study the finite-state formulations of error modelling and contrast my implementations with others that have been used. In Chapter 6, I fully evaluate these systems in terms of both speed and quality to verify their usability in practical and real-world applications. Most of the articles I have written fall neatly under one of these headings, but I do revisit a few of them in more than one of the chapters. In Chapter 7, I summarise the thesis and lay out possible future work in the field.

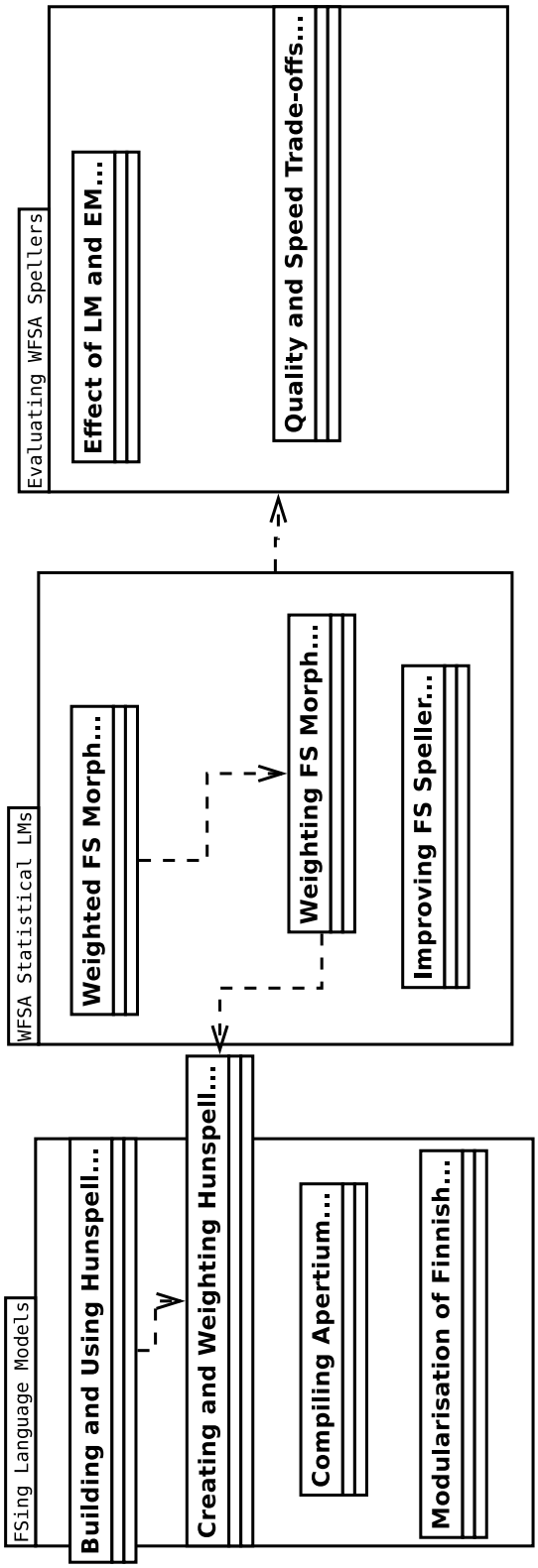

Figure 1.1: Articles and build order

Chapter 2

Background

Spell-checking and correction by computer is a topic that is already over half a century old, and one that has been researched periodically throughout that time. In this chapter, I attempt to walk through the long history of spell-checking with an eye on both the scientific study of spell-checking, as well as the practical end-user systems. While I attempt to cover as much as possible of the topic, there are bound to be many omissions and for a fuller picture I recommend reading some of the previous surveys on spell-checking and correction, such as Kukich (1992a) and Mitton (2009). Kukich (1992a) presents the history of spell-checking, both scientific improvements and some real-world applications are very well summed up until the publication time of the early 1990s. Mitton (2009) presents some more recent advances on non-word spelling correction. Kukich (1992a) also refers to spell-checking as a *perennial topic* in computational linguistics. I find this characterisation quite accurate as it has been a prevalent and recurring theme throughout the history of computational linguistics and earlier in computer science.

This background chapter is organised as follows: In Section 2.1, I go through the history of spell-checking and correction, especially the scientific contributions that relate to my research, and the practical software systems I have used as a baseline for my target functionality. After dealing with the history of spell-checking and correction I go through the closely related scientific fields in Section 2.2, whose results are relevant to my research. Finally, I go through the current situation in Section 2.3, and specifically the finite-state approaches to spell-checking and correction, and describe my particular system. I also detail much of the notation and terminology in these chapters, and contrast them with other similar approaches.

2.1 Brief History and Prior Work

The history of spell-checking and correction by computer is usually said to have begun somewhere around the 1960s, with inventions like Levenshtein's and Damerau's mea-

sures of distances between strings (Levenshtein, 1966; Damerau, 1964), or the Damerau-Levenshtein *edit distance*, which even today is the core of practically all spelling correction algorithms. This measure defines the distance between two strings of characters, in terms of editing operations performed on the string to match it to the other string. The editing operations defined were the following:

- *deletion* of a letter
- *addition* of a letter
- *changing* a letter to another, and
- *swapping* transposing adjacent letters (missing in Levenshtein's formulation, central to spelling correction for e.g., keyboards)

For example, the distance between *cat* and ca is 1 (deletion of t), the distance between *cat* and catt is 1 (addition of t), and so forth. Formally, these operations create a metric between two strings, and for spell-checking applications it is typical to find the smallest distance between strings, or the least amount of editing operations when correcting. It is also possible to use values differing from 1 for the operations to obtain better suitable distance metrics for specific spelling correction tasks.

It is easy to see from the definitions of the operations how useful they are for spelling correction; these operation provide the model for detecting and correcting the basic typos or slips of the finger on a keyboard. Damerau (1964) is often cited as pointing out that 95% of the errors are covered by distance 1, i.e. one application of this edit algorithm would fix 95% of misspellings in a text. While this is indeed the claim made in Damerau's article, citing it in the context of a modern spelling-checker may neglect the difference between input mechanisms (and data transmission, and storage) of computers of the time. It is likely that spelling errors from cursive hand writing, punch cards, and keyboards are different, though this variation is dealt with in the article (Damerau, 1964). Further studies on different forms of computerised texts (Kukich, 1992b) have shown that the range of single edit errors is around 70%–95% in various setups (e.g. OCR, dictating and typing, normal running text).

Much of the early research on error detection concentrated on efficiently looking up words from finite wordlists and building data structures to work as *language models*. During this time, the first statistical approaches, as in Raviv (1967), drawing from basic foundations of mathematical theory of information from as early as Shannon (1948),[1] were devised. In Raviv's research statistical theories are applied to characters in English legal texts, recognising also names and other terms. The input mode of these early applications seem to be more towards OCR than keyboard input. Their approach took the

[1] Referred to in Liberman (2012)

letter n-grams and assumed that words containing unlikely letter combinations were not spelled correctly.

For the very first spell-checking software in common use – there are a few disputes and claims to it – but it is commonly attributed to SPELL program (Gorin, 1971). Some of the earlier work has been used mainly by one research group for their own purposes only.[2] The first predecessors of SPELL according to Les Earnest were systems for recognising hand-written cursive text as auxiliary parts of larger software. Following this SPELL was possibly among the first stand-alone software for spell-checking.

Much of the following research on error detection consisted of more elaborate data structures for fast lookup and efficient encoding of large dictionaries. While interesting as a computer science and data structures topic, it is not particularly relevant to this thesis, so I will merely summarise that it consisted of hashing, suffix-trees and binary search trees, partitioning of dictionary by frequency (Knuth, 1973) and most importantly, finite-state automata (Aho and Corasick, 1975). Although Aho and Corasick's article is about constructing more specific finite-state automata for efficient keyword matching it constitutes an early implementation of finite-state spell-checking.

SPELL program's direct descendant is the current international ispell (Gorin, 1971), where the additional i originates from the name `ITS SPELL`. It is still commonly in use in many Unix-based systems. According to its documentation, the feature of suffix stripping based on classification was added by Bill Ackerman in 1978, e.g. it would only attempt to strip the plural suffix *-es* for words that were identified as having this plural suffix. The concept of affix flags is still used in all of ispell's successors as well.

At the turn of the 1990s there was a lot of new research on improving error correction capabilities, possibly partially due to rapid growth in the popularity of home computers. Most popularly the use of statistics from large text corpora and large datasets of real errors that could be processed to learn probabilities of word forms and error types was applied and extensively tested (Kernighan et al., 1990; Church and Gale, 1991). These simple and popular methods are still considered to be useful for modern spell-checkers, which can be seen as common revisions of techniques, such as in Brill and Moore (2000).

One of the digressions was to improve *error models* for English competence errors. The initial work for this is the often cited Soundex algorithm, originally meant for cataloguing names in a manner that allowed similarly pronounced names to be easily found (Russell and Odell, 1918). It can also be used to match common words, since it is a *similarity key* that maps multiple words into one code and back, for example squer and *square* have the same code `S140` and can be matched. What soundex similarity basically does is save the first letter and assign the remaining non-adjacent non-vowels a number.[3] There have been some schemes to elaborate and make this work for foreign names and

[2] `http://www.stanford.edu/~learnest/legacies.pdf`

[3] For details, see e.g. `http://en.wikipedia.org/wiki/Soundex` or the finite-state formulation in the thesis article X

more words, most notably Metaphones by Philips (1990, 2000).[4]

As computational power increased it became increasingly practical to look again at the problem of real-word spelling error detection. Mays et al. (1991) suggested that with English context-based approaches it is possible to detect 76% of the spelling errors, and are able to correct 73%. Context-aware models are a requirement for the discovery of real-word spelling errors, but independently of that, they can also be used to improve error correction.

Al-Mubaid and Truemper (2006) show that it is possible to uncover common real-word errors directly from the text using correctly spelled reference texts to calculate context factors for the words, and seeking words that deviate sufficiently enough from the factors in the other texts.

In the world of context-aware models, simple word-form n-grams are not the only form of context that has been used – especially in the context of languages other than English, it has often been noted that using sequences of morphological analyses instead of the surface word forms is a more important factor in detecting errors and improving the results (Otero et al., 2007, for Spanish). I have performed an experiment with this approach in (VIII) for Finnish.

The problems of implementing Hungarian with ispell, aspell and the like lead to Hunspell, with multiple affix stripping and compounding added. Similarly for Turkish, various computational methods have been used. Among those, Oflazer (1996) demonstrates one of the first finite-state spell-checking with a full-fledged finite-state language model. There is an earlier finite-state spelling correction system described by Aho and Corasick (1975), where both the dictionary and the texts are limited to keyword search from indexes. This method used a specialised search algorithm for error tolerance when traversing the finite-state network. Savary (2002) extends these finite-state methods to cover the error-model as well. Their work showed a practical implementation of finite-state edit distance, and finally in (III), I have shown that an approach by an extension to weighted finite-state automata for both language and error models for spell-checking and correction is plausible for morphologically complex languages.

In Table 2.1 I have first summarised the practical end-user applications of spell-checking. In the first column I give the related software and academic reference if available, in the second column is the year of the publication of the software or research finding. In the third column is a short note about the error models the system uses, the abbreviation ED is used for edit distance algorithm and FSA for finite-state automata's expressive power. In the fourth column is a characterisation of the dictionary or language model format, e.g., the wordlist means finite list of words and affixes which generally refer to algorithms that can remove add specific suffixes to some words of the wordlist (or remove them, depending on how you view the implementation). In the first set of

[4]The third is a commercial product without publicly available documentation: `http://amorphics.com/buy_metaphone3.html`

entries I give the history from the original SPELL software up until its contemporary descendant and *de facto* standard of *FLOSS*: Hunspell. The dictionaries have not changed much in this branch, practically all use word-lists with some affix capabilities based on classifying the words. In error correction, all use at least some form of edit distance algorithm, with varying weights or orderings of the edit operations, including simple soundslike weighting in GNU aspell[5] and Metaphone variants for English in kspell and its successors. More details on error models are provided in Chapter 5. In the second part of Table, I summarise some of the main academic research on specific practical and statistical advances as originally presented by Al-Mubaid and Truemper (2006) that I have used during my research. For example, the prominent `correct` spell-checker re-ranker by Church and Gale (1991), which basically re-orders the results of the above-mentioned spell programs by probabilities, has influenced some of my spell-checker designs so that the re-ranking approach has been designed into the original system. The further statistical approaches that are relevant are the Bayesian approach (Golding, 1995) and Winnow (Golding and Roth, 1999). The main differences between these approaches lie in statistical formulations. In the final section of Table, I show the main history of contemporary finite-state spell-checking, starting from Oflazer (1996). For this section the noteworthy differences are whether the error correction uses basic edit distance measures in the finite-state graph, a more advanced search, or finite-state algebra with the full expressive power of regular grammars. Other differences are in the implementation and efficiency of the spelling correction or error-tolerant search.

Name (Authors)	**Year**	**Error Models**	**Language Models**
SPELL – Unix – FLOSS branch (*spell)			
SPELL, (Gorin, 1971)	1971	ED1 (for partial wordlist)	Word-list, English affixes
ITS SPELL, (Bill Ackerman)	1978	ED1 unrestricted	Affix rules
international ispell (Geoff Kuenning)	1988		Non-English support
kspell, GNU aspell (Kevin Atkins)	1998 2002	Metaphone 1 Rule-weighted ED soundslike	Affix rules Compounding (dropped)
myspell	2000	weighted ED	2 affixes
Hunspell	2005	weighted ED Confusables	2 affixes Compounds

Continued on next page

[5] `http://aspell.net/man-html/The-Simple-Soundslike.html`

Table 2.1 – continued from previous page

Name	Year	Error Models	Language Models
Academic projects etc.			
correct (Church and Gale, 1991)	1991	Probabilistic ED	Probalistic wordlist
bayspell (Golding, 1995)	1995	Bayesian	Bayesian
WinSpell (Golding and Roth, 1999)	1999		Winnow
Finite-State			
(Oflazer, 1996)	1996	ED using error-tolerant search	FSA
(Savary, 2002)	2002	ED using FSA algebra	FSA
(Schulz and Mihov, 2002)	2002	ED using pre-composed FSA	FSA
(Mohri, 2003)	2003	ED using WFSA algebra	WFSA
(Otero et al., 2007)	2007	FSA	FSA
(Huldén, 2009)	2009	FSA using search algorithm	FSA
(Pirinen and Lindén, 2010b)	2010	WFSA	WFSA

Table 2.1: History of spell-checker applications. Abbreviation ED stands for edit distance and (W)FSA for weighted finite-state automata

2.2 Related Subfields and Research Results

There are a number of other research questions in the field of computational linguistics that use the same methodology and face the same problems. A large set of research problems within computational linguistics can be formulated in some frames that are directly relevant to the issues of spell-checking and correction, as most of them require a basic component from a language model and very often another one from an error model. The language model of spell-checking predicts how correct a word form is, and in a linguistic analyser it predicts the analysis and its likelihood. The error model of a spelling corrector predicts what is meant based on what is written assuming an error somewhere in the process of typing. Some of the error sources, such as cognitive errors, overlap in speech recognition and information extraction alike. Some of the error models for other

applications like diacritic restoration for information retrieval and noise cancellation for speech recognition may solve different problems but the approaches used are still applicable. Realising all this, at one point of the research, made me persistently require a modular design from our spelling correction systems – including a strict separation of the spelling detection task and the correction task – to be able to mix and match the approaches found in various sources to the spelling correction. The rest of this subsection I use to briefly introduce the specific approaches and algorithms from outside the spelling detection and correction domain that I have re-used or tried to re-use in my experiments on spell-checking and correction.

The relevant research on the finite-state approaches to *predictive text entry* is referred to e.g. in Silfverberg and Lindén (2010), and the results of these applications have been applied without modifications to our language models where applicable.

In many practical systems, a spelling corrector or its weak equivalent is used for pre-processing. This phase of processing is often called e.g. *normalisation*, *diacritic restoration*, and so forth. Normalisation and diacritic restoration are both basically spell-checking tasks with very limited alphabets and error models, e.g. in diacritic restorations only allowed corrections are diacritical additions to a base character (e.g. changing *a* to *à*, *á*, *ã*, *ą*, etc.), similarly normalisation restricts correction to e.g., case changes and typographic variants. Many normalisation approaches are directly relevant to spelling correction, and parts of the spell-checking system I have developed have been inspired by the development of systems such as mobile text message normalisation to standard written language (Kobus et al., 2008). Especially recent approaches in the context of social networking message normalisation are nearly identical to the finite-state spell-checking presented in this dissertation, e.g. Huldén (2013).

The field of *historical linguistics* also uses finite-state and edit-distance techniques, e.g., for mapping related words and word forms (Porta et al., 2013). Furthermore, the statistical approaches used in recent works in the field, such as Petterson et al. (2013), could well be used in the context of spelling correction.

2.3 Theory of Finite-State Models

Finite-state models for spell-checking and especially correction are relatively new, beginning from (Oflazer, 1996), and definitions and interpretations have not been standardised yet, so in this chapter, I will go through the various definitions and explain my solution and how I ended up with it. To begin with, I will use a few paragraphs to recap the formal definitions of finite-state systems and my selected notations. For further information on finite-state theory, see e.g. Aho et al. (2007); Mohri (1997).

The *FSAs* are conventionally marked in computer science and mathematics as systems, such as n-tuple $(Q, \Sigma, \delta, Q_i, Q_f, \rho)$, where Q is the set of the states in the automaton, Σ is the alphabet in transitions, $\delta : Q \times \Sigma \times W \rightarrow Q$ is a deterministic transition

mapping from a state to a state with an alphabet and a weight, and $Q_i \subset Q, Q_f \subset Q$ the subsets of states for initial states and final states, $\rho : Q_f \to W$ the final weight mapping and W is the structure of weights. The Σ set in the automata of a spell-checking system is almost always just some – usually language-specific for optimisation reasons – subset of the Unicode set of symbols (Unicode Consortium, 2013) for writing natural languages, with the addition of the following special symbols, which have specific meaning in automata theory: the empty symbol epsilon ε that matches zero-length strings on application of the automata, and the wild-card symbol $?$ that matches any one symbol of Σ during any application of the automaton. When talking of transducers, it merely means that the alphabet is of the form Σ^2. Practically this can be thought of so that the *finite-state acceptor* accepts e.g., correct wordforms and the *finite-state transducer (FST)*, e.g., rewrites misspelled word forms into one or more corrected forms. The weighted *FSAs* discussed throughout the thesis are using the *tropical semi-ring* (Mohri, 1997) weight structure $(\mathbb{R}_+ \cup \infty, \min, +)$; this is the so-called penalty weight structure, which practically means that on application and weight combination the smallest one is used, on combination of the weights they are added together. The set of final states is extended with a final weight function $\rho : W \to Q$ that specifies an additional weight for the paths in an automaton in each end-state. For the rest of the finite-state algebra I use the standard notations which to my knowledge do not have any variation that requires documenting in this introduction (e.g. $\cup$ for union, $\cap$ for intersection and so forth). I will furthermore use notation $\mathcal{M}$, often with a subscript: $\mathcal{M}_{\text{name}}$ for finite-state machine, where the name is a descriptive name for an automaton or an abbreviation.

An *FSA*, with a Σ set drawn from the set of natural language alphabets, such as letters A through Z, digits 0 through 9 and the punctuation marks hyphen and apostrophe, can accurately encode most words of the English language in the accepting paths of the automaton – excepting such edge cases as *naïve*. This kind of acceptors which recognise the words of a language are used both in the process of detecting spelling errors of non-word type in a running text, and matching the misspelt word forms to the correct word forms. The use of such automata as language models is documented very extensively in natural language processing. In the context of morphologically complex languages more relevant to the topics of this thesis see, e.g., Finite-State Morphology (Beesley and Karttunen, 2003; Beesley, 2004).

I use *FSAs* also in error correction. In this case, *FSTs* encode the relations from misspellings to corrections in their accepting paths. There are few influential works in the field of finite-state methods for error modelling. The initial work was laid out by Oflazer (1996), who uses an algorithmic approach on language model traversal as a limited form of error modelling; this has been extended and improved by among others Huldén (2009). An approach to weighted finite-state systems was described by Mohri (2003), which includes a good mathematical basis for finite-state error-modelling using weighted edit-distance automata and weighted automata algorithms.

Savary (2002) describes the finite-state error model as an automaton. In that paper, the research concentrates on the concept of expanding the language-model automata by a Levenshtein rule automaton, creating a language model automaton that can recognise word forms containing a given number of Levenshtein type errors, i.e. all word forms at a given Levenshtein-Damerau distance. My definition diverges here by considering the error model as a separate, arbitrary transducer; this allows operating on either the language model or the misspelt string with the error producing or removing the functionality of the model, and provides an opportunity to test which variation is the most effective.

Now we can also make the generalisation of considering the strings of the language that we are correcting as a single-path acceptor consisting of just one word, so we can formally define a finite-state spelling correction system as the composition $(\mathcal{M}_{\text{word}} \circ \mathcal{M}_{\text{E}} \circ \mathcal{M}_{\text{L}})_2$, where $\mathcal{M}_{\text{word}}$ is the word to correct, $\mathcal{M}_{\text{E}}$ the automaton encoding the error model, and $\mathcal{M}_{\text{L}}$ the automaton encoding the language model, and $_2$ is the projection selecting the results from the second tape of the final composition, i.e., the one on the language model side.

The weights in weighted finite-state automata are used to encode the preference in spelling correction, and sometimes also acceptability in the spell-checking function. A path that has a larger collected weight gets demoted in the suggestion list and those with smaller weights are promoted. The weight can be amended by the error model, in which the weight expresses the preference on errors corrected when mapping the incorrect string to correct. One of my contributions throughout the thesis is a theory and methodology for creating, acquiring and combining these weights in a way that is optimal for speed, and that is usable for morphologically complex languages with limited resources.

The baseline for acquisition of the weights for language and error models is simply calculating and encoding probabilities –this is what most of the comparable products do in spell-checking and correction, and what has been researched in statistical language models. The key formula for conceiving any statistical process in the tropical semiring giving the probabilistic distribution P as a part of a finite-state automaton is $-\log P$, i.e. the smaller the probability the bigger the weight. The probability here is not straightforward, but relatively simple. For most things we calculate

$$P(x) = \frac{f(x)}{\sum_{z \in \mathcal{D}} f(z)}, \tag{2.1}$$

where x is the event, $f()$ is the frequency of the event, and $\mathcal{D}$ the collection of all events, so the probability of x is counted as the proportion of the event x in all events z of $\mathcal{D}$. For example, for a simple language model, x could be a word form, and $\mathcal{D}_{\text{wordforms}}$ a corpus of running text turned into word forms, then we would expect that $P('\text{is}') > P('\text{quantitatively}')$, for any reasonable corpus of the English language. Similarly for

error modelling, x might be a typo of 'a' for 's' denoted a : s and $\mathcal{D}_{\text{errors}}$ an error corpus of typos written with a qwerty keyboard layout, then we would expect $P(\text{a} : \text{s}) > P(\text{a} : \text{l})$.

An important factor that *morphologically-complex* languages bring to the concept of statistical language models is that the amount of different plausible word forms is greater, the data is more sparse, and essentially language models turn from finite word-form lists to infinite language models. There is a good mathematical-lexicographical description of this problem in Kornai (2002). This means that for simple statistical training models, the number of unseen words rises, and unseen words in naive models mean a probability of 0, which would pose problems for simple language models, e.g. given the above weight formula $-\log(0) = \infty$ for any given non-zero-size corpus. A practical finite-state implementation will regard infinite weight as a non-accepting path even if it were otherwise a valid path in the automaton. The traditional approach to dealing with this is well known; assuming a probability distribution we can estimate the likelihoods of the tokens that were not seen, discount some of the probability mass of the seen tokens, or otherwise increase the probability mass and distribute it among the parts of the language model that would have been unseen. With language models generating infinitely many word forms, the models made using arbitrary weights may not be strict probability distributions at all. In practical applications this does not necessarily matter as the preference relation still works. Some of this background on not following strict statistical distributions in NLP context has been given by Google in their statistical machine translation work, e.g. by Brants et al. (2007).

The basic forms of estimating and distributing the probability mass to account for unseen events has been extensively researched. The basic logic that I have used in many of my research papers is the simplest known additive discounting: here the estimated probability for an event x is seen as

$$P(\hat{x}) = \frac{f(x) + \alpha}{\sum_{z \in \mathcal{D}} (f(z) + \alpha)} \tag{2.2}$$

that is, each frequency is incremented by α and this mass of increments is added to the divisor to keep the distribution in probabilistic bounds – this has the effect that all unseen tokens are considered to have been seen α times and all others α more times than they have been seen in the training data. For values of α in my practical applications, I have used the range 0.5–1, based on empirical testing and values found in the literature, such as Manning and Schütze (1999). There are other, more elegant methods for discounting as well (Chen and Goodman, 1999). However, the additive discounting being a few lines of code is reasonably easy to implement and understand without errors.

Chapter 3

Language Models

In this chapter, I will go through the various kinds of *language models*[1] that are used for the task of spell checking and correction. Depending on the system, these two tasks can either share a common language model, or use separate and different language models, or even different approaches to apply the language models. This separation is an important one and I will try to make it clear whenever I refer to language models in various systems and aspects of finite-state spell-checking. The primary purpose of a language model in both of these tasks is similar: to tell whether a word form is suitable, and, ideally, how suitable it is.

The purpose of this chapter is to present the development through traditional simple word-list spell-checking models to *complex morphological* analysers with compounding and derivation, and their finite-state formulations. This follows from my initial goal to not only present new language models and spell-checkers in my thesis, but to use finite-state technology to repeat the existing results of non-finite-state spell-checkers as a baseline. In the later chapters, I will show that the finite-state formulations are also as good as, or better than, their non-finite-state counterparts, and the statistical methods we devise to support morphologically complex languages can be applied to these finite-state automata.

The rest of the chapter is organised as follows: first in Section 3.1, I introduce some generic formulas for finite-state morphologies we have recognised when developing compilers for quite a few dictionary formats. Based on this, I show in Section 3.2 the finite-state compilation of Hunspell dictionaries, the *de facto* standard of open-source spell-checking. Then I show the conversion of the rule-based machine translation dictionaries as spell-checkers in Section 3.3. In section 3.4, I describe other language models

[1] As a terminological note, in *natural language engineering* the term language model is used narrowly, referring only to systems that are purely statistical n-gram models. In computational linguistics a language model is any method capable of telling whether a given word form is in a language, and can potentially provide further data about it.

that can be used in finite-state spell-checking. Finally, in Section 3.5, I introduce a brief general treatment on the management of linguistic data so that it applies to a multitude of projects in an article where I try to tie together the different approaches to compiling and using finite-state dictionaries and computational linguistic models at large.

3.1 Sketch of Generic Finite-State Formula for Morphology

Before delving further into the intricacies of compiling existing and new *formalisms* into finite-state automata, I would like to draw the readers' attention to one underlying formula of computational morphology that is central to all these approaches. This formula is the conceptualisation that all systems are regular combinations of sets of morphemes, combined with rules of *morphotactics*[2] and possibly morphophonology. The basic starting point is an arbitrary combination of morphs as an automaton $\mathcal{M}_{\text{morphs}} = (\bigcup_{\text{m}\in\mathcal{D}_{\text{morphs}}}(\text{m}))^\star$, where $\mathcal{D}_{\text{morphs}}$ is a collection of morphs of the language where, that is, a disjunction of strings that are the morphs of a language. This kind of a *bag of morphs* approach creates an automaton consisting of arbitrary combinations of the morphs of a language, e.g. for English it would have combinations like 'cat', 'cats', 'dog', 'dogs', …but also 'catdog', 'sdog', or 'sssdogsdogsscatcat'. To restrict these, we usually define classifications of these morphs, say, 'cat' and 'dog' are `[nouns]`, and 's' is a `[plural]`, English morphotax would define that a plural appears after nouns. To realise morphotactic rules over the classifications of morphs, we can usually create a set of helper symbols, e.g. `[nouns]`, `[plural]` $\in \Gamma, \Gamma \cap \Sigma = \emptyset$ and then have morphotax as an automaton construed over $(\Gamma \cup \Sigma)^\star$. In this example we might define something like `[nouns]` $\Sigma^\star$ `[plural]` $\Sigma^\star$, if those special symbols are prefixed to the relevant morphs.

3.2 Compiling Hunspell Language Models

Main Article: *Building and Using Existing Hunspell Dictionaries and TEX Hyphenators as Finite-State Automata*, by Tommi A Pirinen and Krister Lindén. In this article we present finite-state formulations of existing spell-checking language and error models.

3.2.1 Motivation

The **motivation** for this piece of engineering is far-reaching both on the technical and the theoretical level. We set out to prove that the existing methods for spell-checking are

[2] Morphotactics is used to refer to the rules governing legal combinations of morphs within word forms of a language. For example, one such rule would be defining that the English plural suffix form '-s' follows a certain subset of noun stems.

truly a subset of finite-state methods. In practice, we also tried to show that typically, if not always, the finite-state version should be faster than the non-finite-state version, e.g. for affix stripping, but also for error modelling. The practical motivation for the work is that the chances for any new spell-checking system surviving are rather limited, unless it is capable of benefiting from the existing Hunspell dictionaries without major difficulties.

3.2.2 Related Works

In **related works**, there has, to my knowledge, only been, one attempt to use Hunspell data as automata, namely a master's thesis project partially guided by Hunspell maintainers (Greenfield and Judd, 2010). The original Hunspell has been dealt with in an academic context by Trón et al. (2005).

3.2.3 Results

There is a number of meaningful **results** in our article. It is a central part of the thesis in that it provides the empirical proof that finite-state spell-checking is a proper superset of Hunspell's algorithmic approach in terms of expressive power, and the empirical results also suggest an improvement in speed across the board. The results on speed are detailed in my later articles and also in Chapter 6 of this thesis, so here I will first concentrate on enumerating the parts of the first result – the expressiveness and faithfulness of the finite-state formulation of the Hunspell systems.

Hunspell's language model writing formalism clearly shows that it is based on the main branch of word-list-based ispell spell-checkers. Words are assigned flags that combine them with affixes, where the concept of affixes is extended with context restrictions and deletions, although with the same affix logic as its predecessors. The additional features brought to Hunspell to set it apart from previous versions were the possibility of having more than one affix per root – two for most versions, and then a number of various extra features based on the same flags as used for affixation, such as compounding, offensive suggestion pruning,[3] and limited circumfixation. A majority of these features come from the basic formula we use for a bag of morphs and filters style finite-state morphology as described by Lindén et al. (2009). The practical side of the main result shows that all of the language models are around the same size as finite-state dictionaries and morphologies made with other methods, as shown in Table 3.1 reproduced from (IV).

Language	Dictionary	Roots	Affixes
Portugese (Brazil)	14 MiB	307,199	25,434

Continued on next page

[3] E.g. English Hunspell dictionary will recognise, but not suggest such word forms as *asshole, bugger, shitty, etc.*

Table 3.1 – continued from previous page

Language	Dictionary	Roots	Affixes
Polish	14 MiB	277,964	6,909
Czech	12 MiB	302,542	2,492
Hungarian	9.7 MiB	86,230	22,991
Northern Sámi	8.1 MiB	527,474	370,982
Slovak	7.1 MiB	175,465	2,223
Dutch	6.7 MiB	158,874	90
Gascon	5.1 MiB	2,098,768	110
Afrikaans	5.0 MiB	125,473	48
Icelandic	5.0 MiB	222087	0
Greek	4.3 MiB	574,961	126
Italian	3.8 MiB	95,194	2,687
Gujarati	3.7 MiB	168,956	0
Lithuanian	3.6 MiB	95,944	4,024
English (Great Britain)	3.5 MiB	46,304	1,011
German	3.3 MiB	70,862	348
Croatian	3.3 MiB	215,917	64
Spanish	3.2 MiB	76,441	6,773
Catalan	3.2 MiB	94,868	996
Slovenian	2.9 MiB	246,857	484
Faroese	2.8 MiB	108,632	0
French	2.8 MiB	91,582	507
Swedish	2.5 MiB	64,475	330
English (U.S.)	2.5 MiB	62,135	41
Estonian	2.4 MiB	282,174	9,242
Portuguese (Portugal)	2 MiB	40.811	913
Irish	1.8 MiB	91,106	240
Friulian	1.7 MiB	36,321	664
Nepalese	1.7 MiB	39,925	502
Thai	1.7 MiB	38,870	0
Esperanto	1.5 MiB	19,343	2,338
Hebrew	1.4 MiB	329237	0
Bengali	1.3 MiB	110,751	0
Frisian	1.2 MiB	24,973	73
Interlingua	1.1 MiB	26850	54
Persian	791 KiB	332,555	0
Indonesian	765 KiB	23,419	17
Azerbaijani	489 KiB	19,132	0
Hindi	484 KiB	15,991	0

Continued on next page

Table 3.1 – continued from previous page

Language	Dictionary	Roots	Affixes
Amharic	333 KiB	13,741	4
Chichewa	209 KiB	5,779	0
Kashubian	191 KiB	5,111	0

Table 3.1: Compiled Hunspell automata sizes in HFST. Reproduced from (IV)

When talking about Hunspell it is impossible to avoid the motivation behind the development of a new system for spell-checking – as my thesis is also about a new system for spell-checking. Hunspell was made on the basis that the existing spell-checking dictionary formalisms are insufficient to write a Hungarian dictionary efficiently. Similar systems were written for many of the more morphologically complex languages.

As the implementation framework for context restrictions and deletions for the Hunspell affix morphotactics, we used *twol* (two-level) rules (Karttunen et al., 1992). Twol rules are a generic way of defining valid contexts of character pairs in a transducer. Hunspell on the other hand, defines deletions, and combinations of morphs. In practice this means that when Hunspell formulates that a suffix follows a word root in a certain class if it ends in a given regular expression, our twol representation was made to allow the morph boundary symbol in that regular expression context while also deleting the context as defined by another expression. With the context restrictions and its combinatorics twol rules are well within reason, as restriction is the central rule type in the design of twol, whereas the deletions are not very easy to implement in that formalism. As an afterthought, a better and more efficient representation might have been to combine morphotactic filters with replace style rules (Karttunen, 1995), although neither the compilation time nor the clarity of the algorithms are a key issue for one-shot format conversions like this.

One of the main stumbling blocks for some of the finite-state systems with full language models is that – while the running of finite-state automata is known to be fast – the building of finite-state automata can be lengthy. This did not appear to be an issue for the practical systems we tried.

3.3 Using Rule-Based Machine Translation Dictionaries

Main Article: *Compiling Apertium morphological dictionaries with HFST and using them in HFST applications* by Tommi A Pirinen and Francis M. Tyers. In this engineering-oriented article, we experiment with the existing language models used for machine translation as part of the finite-state spell-checking system.

3.3.1 Motivation

The main **motivation** for doing this experiment was to see how well we can reuse the vast numbers of existing dictionaries from other projects as finite-state automata in completely unrelated language technology systems. This is especially interesting from my computer science background, since the reusability of code and data is something that I feel is lacking in computational linguistics even now in the 2010s.

3.3.2 Related Works

As this experiment was built on an existing, partially finite-state-based system, the **related works** already consist of a specific algorithm for building the automata from XML dictionaries (Ortíz-Rojas et al., 2005) for fast and accurate tokenising and analysing. To that end, our experiment does not provide any significant improvements, the final automata are nearly the same and the tokenisation algorithm used to evaluate the result closely imitates the one described in the article.

The compilation formula presented in the article was merely a simplification of the ones used for Hunspell and lexc, making it relatively straightforward and short as a technical contribution. This straightforward approach is notable however, as the computational side is based on a solid standardised XML format for dictionary representation and interchange, and this makes it optimal for future experiments.

3.3.3 Results

The main **results** shown in the paper are the improvement on text processing time as shown in Table 3.2. This is in line with previous results (Silfverberg and Lindén, 2009). More importantly to the topic of this thesis, results showing the reasonably fast processing time of the finite-state spelling checkers built in this manner; this is not totally unexpected as the *formalism* does not support compounding or recurring derivation, and thus the final automata are bound to be acyclic, except potential regular expression recognisers in language models, which we chose to exclude from the spell-checking models.

The main result that we tried to highlight in the paper was that we could already provide a rudimentary spell-checker based on lexical data built solely for the purpose of limited range machine translation, and we could do that for a range of languages, including some that at the time of writing lacked spell-checkers altogether. A list of the languages tested here is provided in Table 3.2, reproduced from (VII). The task of writing the compiler for this existing format based on other morphology *formalisms*, that we had made, and turning the compiled language model into a baseline spell-checker, was not more than a few days' work. This point concerning the generic usefulness of finite-state systems is often easily overlooked with assumption that all new systems require months of development time.

Language	HFST Rate	Apertium lookup	HFST lookup
Basque	7,900	35.7 s	160.0 s
Norwegian	9,200	6.6 s	200.2 s
Manx	4,700	0.8 s	11.2 s

Table 3.2: Efficiency of spelling correction in an artificial test setup, average over three runs. Rate given in words per second. Speed of HFST-based system against original in compilation and speed of HFST-based system against original in corpus analysis (as seconds in user time). Reformatted from (VII).

A side result, which I partially failed to communicate in this paper, is the development towards a generalised compilation formula for finite-state dictionaries. The compilation formula in this article is a variation of the earlier formulas I have presented (Lindén et al., 2009) and (IV) – I propose that this has two implications: for the engineering side, we can use the same algorithms to compile all the different dictionaries, which means that the automatic improvements and optimisations that can be applied to the structure of the automata will improve all applications. More importantly, it means that the different application formalisms for writing dictionaries use the same linguistic models and the same kind of abstraction i, or lack of abstraction, is present. This suggests that there is a potential for generalisations such that different applications could converge using a single, linguistically motivated dictionary format.

3.4 Other Possibilities for Generic Morphological Formula

There is an abundance of formalisms for writing dictionaries, analysers and language technology tools available, and I have only covered the most prominent for use in finite-state spell-checking. For example, I have given a compilation formula for the Hunspell formalism. The aspell and ispell formalisms are simplifications of this, and the formula can easily be simplified for them by removing the multiple suffix consideration and compounding loop in the bag-of-morphs construction: $\mathcal{M}_L = \mathcal{M}_{\text{prefix}}\mathcal{M}_{\text{roots}}\mathcal{M}_{\text{suffix}}$, where $\mathcal{M}_{\text{prefix}}$, $\mathcal{M}_{\text{roots}}$, and $\mathcal{M}_{\text{suffix}}$ are subsets of $\mathcal{M}_{\text{morphs}}$, that is, disjunctions of strings that make up the morphs of a language.

It should also be noted that the common baseline approaches using word-form lists or such spell-checking corpora, such as Norvig (2010), are even easier to formulate as the whole language model is just a disjunction of the surface word forms.

3.5 Maintenance of the Language Models

Main Article: *Modularisation of Finnish finite-state language description—towards wide collaboration in open-source development of morphological analyser* by Tommi A Pirinen. This short paper is mainly an opinion piece article, but it is included here as it ties together the aspect of maintainable language models for spell-checking applications.

3.5.1 Motivation

One of the main **motivations** for writing this article was the realisation after some years of work on the finite-state language model compilation and harvesting different computational dictionaries and descriptions of morphologies that there is a common pattern in all the data – as one generalised finite-state algebraic compilation algorithm works for all of them. An opposite but equally strong motivation was that for lesser-resourced languages a single computational language model is a very precious resource that needs to be reused for all possible applications, as there are no human resources to rewrite these descriptions separately for each application. This should motivate computational linguists in general to work towards reusable, well-written resources, with proper abstraction levels and basic knowledge representation.

Another motivation is that the language descriptions, whether for finite-state morphological analysis or Hunspell spell-checking are typically written by one linguist, and ignored when no one can read them any longer. This is a not a useful approach in the crowd-sourcing world of today, and it could surely be alleviated by simple improvements in formalisms and practices. In this branch of my research I argue for building more crowd-sourceable systems, in the form of linguistically motivated data that is understandable for anyone fluent in language. While producing easy access to achievable goals in dictionary building, the research of crowd-sourcing lexicography is still a new topic in science and very much a work in progress.

3.5.2 Related Works

Looking into the **related works**, I soon realised that I was not totally, though admittedly mostly, alone in my feeling that finite-state and other computational language descriptions need a some thought to be maintainable and re-usable. The three notable computational linguists writing on the same topic that I immediately came across when researching the topic were Maxwell and David (2008), and Wintner (2008). Similar concerns have been addressed in non-finite-state computational linguistics in the Grammatical Framework project, of which, e.g., the concept of smart paradigms (Ranta, 2008) was in part an inspiration for me to start thinking of better systems for managing finite-state language descriptions. The common features of all our concerns include the lack

of abstraction, and therefore very application-specific hacks – such as encoding optimisations of graph structure into the dictionary data as by Karttunen (2006) – in language descriptions that could ideally be very generic. This also leads me to suggest that some related works that are very under-used in the field of *computational* linguistics are the basic schoolbook materials on computer science and software engineering- Knuth's discussion of literate programming (Knuth, 1984) and the creation of and documented programs is useful for that is what computational language models really are. They are not merely data. Indeed, the whole object-oriented programming movement, involving the encapsulation, abstraction and management of data for re-usability and other principled purposes, is valuable.

3.5.3 Results

The main **results** of this paper are seen in the continuity of the Finnish language model and its relatively good amount of repurposing over the years. Beyond that, the common movement towards maintainable and reusable language models in computational linguistics, and morphology specifically, is still very much work in progress.

3.6 Conclusions

In this chapter I have shown that the vast majority of the different contemporary approaches to spell-checking can be formulated as finite-state automata, and they can even be compiled into one using one simple generic formula. These results, with the addition of all those existing finite-state language models that no one has been able to turn into e.g. Hunspell dictionaries should provide a basis for evidence that finite-state spell-checking is a plausible alternative for traditional spell-checking approaches at least as far as the modelling of correctly spelled language is concerned. Finally, I've described a way forward with finite-state morphologies as a maintainable form of electronic dictionaries for use with a number of applications including spell-checking.

Chapter 4

Statistical Language Models

One of the goals of my thesis was to bring the statistical *language models* closer to usability for *morphologically complex* languages with fewer resources. For this purpose I tried out the most typical traditional statistical approaches turning them into a finite-state form using well-studied algorithms of weighted finite-state automata. With the more morphologically complex languages, the improvement gained by bringing statistical data to the language models in the system is less encouraging than it is with English. I then experimented with various ways of improving the results by more efficient use of linguistic data and simply fine-tuning the parameters.

The problem that arises from the combination of morphologically complex languages and the lesser-resourced languages is challenging, since the amount and quality of the training data is smaller and the requirements for it are greater. I go through two separate approaches to tackle this. The first approach is to gather more linguistically motivated pieces of information from the data as the primary training feature rather than the plain old surface word forms. This is not free of problems, as many good pieces of information can only be extracted from analysed, disambiguated language resources. This requires manual labour, whereas surface word forms come without extra effort. So instead of requiring manually annotated data I have studied how to salvage what can be easily used from running texts without the need of analysis – for Finnish this meant word forms and roots for compound parts. The other approach I took was to try and to see if even a l imited resource of lesser quality – that happens to be available under a free and open licence – can be used combined with well-formed, rule-based material to greatly improve the results in error correction.

In this chapter I will describe the statistical methods I needed to introduce to get good statistical language models for morphologically complex languages such as Finnish, though most approaches could be applied to other languages with a few simple modifications. In Section 4.1, I will introduce the general format of all statistical finite-state models. In Section 4.2, I discuss the basic findings of the compound boundary

identification problem, and in Section 4.3, I extend this approach towards generic statistical training of language models with compounding features. In Section 4.4, I use the same statistical models with the Hunspell dictionaries turned into finite-state automata. Finally in Section 4.5, I study the common problem with the combination of fewer resources and a more complex morphology which actually requires more resources, and propose some possible solutions.

4.1 Theory of Statistical Finite-State Models

A basic statistical language model is a simple structure that can be *trained* using a corpus of texts that contains word forms. There is ample existing research for statistical and trained language models for various purposes, and I strongly recommend basic works like Manning and Schütze (1999) as a good reference for the methods and mathematical background in this chapter. For spelling correction, a simple logic of suggesting a word form that is more common before one that is less, is usually the best approach in terms of precision, and in the majority of cases this leads to a correct suggestion. This is formalised in weighted finite-state form by a simple probability formula: $w(x) = -\log P(x)$, where w is a weight in the tropical semi-ring structure and $P(x)$ is the probability of word form x. Morphologically complex languages' models are thought to be more difficult to learn because the number of the different word forms is much larger than for morphologically simple languages, which leads to decreased discriminative capabilities for the probability calculations, as the number of word forms with exactly the same number of appearances increases. In my research, I have studied some approaches to deal with this situation. What makes this situation even more difficult is the fact that morphologically complex languages often have fewer resources. Many of the contributions I have provided here concentrate on using the scarce resources in a way that will give nearly the same advantage from the statistical training as one would expect for morphologically poor languages with more resources.

The more complex statistical language models, such as those that use morphological, syntactic or other analyses, require a large amount of manually verified data that is not in general available for the majority of the world's languages. In my research, I present some methods to make use of the data that is available, even if it is unverified, to improve the language models when possible. In general, I try to show that the significance of analysed data is not essential to typical spell-checking systems, but more significant for the fine-tuning of the final product and the building of the grammar-checking systems from spell-checkers.

Another significant piece of information in statistical models is *context*. The basic statistical probabilities I described above work on simple frequencies of word forms in isolation. This kind of model is called e.g. a *unigram* or context-agnostic language model. The common language models for applications like part-of-speech guessing, au-

tomatic translation and even spell-checking for real word errors typically use the probabilities of e.g. word forms in context, taking the likelihood of the word following its preceding word forms as the probability value. These are called n-*gram* language models. The requirement for large amounts of training data is even steeper in models like these than it is with context-ignorant models, so I will only briefly study the use of such models in conjunction with finite-state spelling correction. One has to be wary though, when reading the terms in this chapter: I will commonly apply methods of n-gram language models to make unigram language models of complex languages work better, practically treating morphs as components of a unigram model the way one would treat word forms for n-gram word models of English. I will commonly refer to the resulting models as unigram while the underlying models are morph n-gram models.

The main theme to follow for the rest of the chapter is the concept of workable statistical models for morphologically complex languages with only small amounts of the training data available. The improvements are provided as ways to cleverly reuse the existing scarce data as much as possible while otherwise sticking to the well-established algorithms of the field otherwise.

4.2 Language Models for Languages with Compounding

Main article: *Weighted Finite-State Morphological Analysis of Finnish Inflection and Compounding* (I) by Krister Lindén and Tommi Pirinen. This article introduced the statistical models of a morphologically complex language with an infinite lexicon.

4.2.1 Motivation

The original **motivation** for this research was to implement the compound segmentation for Finnish given the ambiguous compounding of the previous implementation (Pirinen, 2008). In the article, we established that Finnish as a morphologically complex language requires special statistical handling for its language models to be considered similar to simpler statistical models for morphologically simpler languages.

4.2.2 Related Works

In **related works**, the first approaches to the compound segmentation problem that have been used are non-lexical, grapheme n-gram approaches (Kokkinakis, 2008, referred to in) to Swedish. Later solutions include systematically selecting the word forms with fewest compound boundaries (Karlsson, 1992), and using statistics about words for German (Schiller, 2006). In the article we showed a combination with both approaches, where the statistics are learned from the surface word forms in running text, instead

of a preanalysed and disambiguated corpus and a manual selection routine. Showing that this method worked added two important contributions to the language models of morphologically complex languages: firstly, information can be derived from running unanalysed texts, which is crucial, because the lack of good materials for most of the lesser-resourced languages, and secondly, the application of basic statistical formulas to morphologically complex languages, was an important generalisation.

4.2.3 Results

The **results**, as stated in the motivation, are that the experiment presented in the article was built on disambiguation of the compound boundaries, rather than building language models for general applications. The weighting mechanism however was applied to the language model of Finnish for all the future research in a wider set of topics.

In the article, I study the precision and recall of the compound segmentation of Finnish compounds using the statistical language modelling scheme. The results shown in the article are rather promising; nearly all compound segmentations are correctly found even with the baseline approaches of giving rule-based weights to morphological complexity. There is a slight improvement when using the statistically formed system that gives a higher granularity in a few cases. There is one caveat in the material used in the paper, since Finnish lacks a gold standard material for any analysed text corpora, we used a commercial corpus made by another combination of automatic analysers.[1] This means that the precision and recall values we obtained were gained by measuring how closely we imitated the referent black box system, rather than real world compound segmentations.

4.3 The Statistical Training of Compounding Language Models

Main Article: *Weighting Finite-State Morphological Analyzers using HFST tools* by Krister Lindén and Tommi Pirinen. In this article we set out to provide a generic formulation of statistical training of finite-state language models regardless of complexity.

4.3.1 Motivation

The **motivation** of this piece of research was to streamline and generalise the process of creating the statistical finite-state language models, as the experiment in the previous article (I) was constructed in a very *ad hoc* manner. The intention of the new method for training language models is to get all the existing language models trainable regardless

[1] `http://www.csc.fi/english/research/software/ftc`

of how they are constructed; this makes the process of training a language model very similar to what it has been in all non-finite-state software to date.

4.3.2 Related Works

Related works constitute a whole field of literature of statistical language models. The concept of having a good language model and using data from corpora to train it provides the basis of any form of statistical natural language engineering. For common established software in the field, see e.g. SRILM (Stolcke et al., 2002), and IRSTLM (Federico et al., 2008) and so forth. The existing work on training infinite dictionaries is more rare, as it is not needed for English. For German, this is dealt with by Schiller (2006), who also uses finite-state technology in its implementation.

4.3.3 Results

As an initial **result** in the article, I replicated the earlier compound segmentation task from (I) to show that the generalised formula works. Furthermore, I experimented with the traditional POS tagging task to see how compound ambiguity actually affects the quality of POS tagging. The results show an improvement which proves that statistical word-based compound modelling is a useful part of a statistically trained morphological analyser. This has the practical implication that it is indeed better to use a compound-based model as a baseline statistical trained analyser of a language, rather than a simple word-form-based statistical model.

In general, the unigram language model with the addition of word-form-based training for compounds shows good promise for training morphologically complex languages from the morphologically relevant parts instead of from plain word forms. Based on these results I moved towards an understanding that the statistical method of weighting morphologically complex language models based on the word constituents of compound forms is a way forward for statistically training morphologically complex language models.

4.3.4 Future Research

As conceivable **future research**, there is a generalisation to this method I have not yet experimented with. The constituents that I used for Finnish were word forms, and this is based on the fact that Finnish compounds are of a relatively simple form. The initial parts are the same as surface forms of regular singular nominatives or genitives of any plurality and in some cases any form. This is similar to some other languages, such as German and Swedish. There are some compound word forms with a *compositive* marker that is not available in isolated surface forms, and there are a few of those in Finnish as well, e.g. words ending in a ‘-nen’ final derivation will compound with the ‘-s’ form

which is not seen outside compounds. In addition, some archaic compositives exist for a few words, and some typically non-compounding cases are attested as well. This is, however, not a productive phenomenon and therefore it is handled in the lexicon by making those rare forms as new entries in the dictionary. In languages like Greenlandic, compounding does not heavily contribute to the complexity, though recurring inflection and derivation does. The generalisation that might be needed is to train the language model based on *morphs* instead of the uncompounded word forms. I believe this could make statistics on languages of varying morphological complexity more uniform, since the variation in the number of morphs between languages is far more limited than the number of word forms. There is a previous study on such language models based not on morphs, but on letter n-graphs, or statistically likely affixes, by Creutz et al. (2005).

4.4 Weighting Hunspell as Finite-State Automata

Main Article: *Creating and Weighting Hunspell Dictionaries as Finite-State Automata* by Tommi A. Pirinen and Krister Lindén. In this article we present weighted versions of finite-state Hunspell automata. This is also the main article introducing for the first time the weighted finite-state spell-checking on a larger scale.

4.4.1 Motivation

The main **motivation** for this experiment was to work on extending the previously verified results for weighting finite-state language models to newly created formulations of Hunspell language models as finite-state automata. The functionality of this approach could be counted as further motivation for moving from a Hunspell's non-finite-state approach to weighted finite-state automata with the added expressive power and efficiency.

4.4.2 Related Works

In **related works** describing probabilistic or otherwise preferential language models for spell-checking, the main source for the current version of context-insensitive spelling correction using statistical models comes from Church and Gale (1991).

4.4.3 Results

The main **results** in the article were measured from a large range of Hunspell languages for precision, showing improvement in quality across the board with a simple unigram training method and the Hunspell automata. A methodological detail about the evaluation should be noted; namely, that the errors were constructed using automatic methods

in the vein of (Bigert et al., 2003) and (Bigert, 2005). In this setup, a statistical approach to language models of spell-checking improves the overall quality.

The implication of these results for the larger picture of this research is to show that the finite-state formulation of Hunspell language models is a reliable way forward for Hunspell spell-checkers, providing the additional benefit of statistical language modelling to Hunspell's well-engineered rule-based models. This is important for practical development of the next generation of spell-checking systems, since the creation of dictionaries and language models requires some expert knowledge, which is not widely available for most of the lesser-resourced languages.

4.5 Lesser-Resourced Languages in Statistical Spelling Correction

Main Article: *Finite-State Spell-Checking with Weighted Language and Error Models* by Tommi A. Pirinen and Krister Lindén. In this article we first researched the problem from the point of view of less-resourced languages. Even though Finnish does not have significantly more freely usable resources this is the first explicit mention of the problem in my research. Furthermore, I used North Saami for the first time to show the limitedness of freely-available open-source language resources for spell-checking for a real lesser-resourced non-national language.

4.5.1 Motivation

The **motivation** for this article was to write a publication that explicitly explores the difficulties that morphologically complex and lesser-resourced languages face when trying to pursue statistical language models and corpus-based training. The article fits well into contemporary research on natural language engineering, such research often considers corpus-training as a kind of solution for all problems and ignores both the scarcity of resources and the much higher requirement of unannotated resources for morphologically complex languages.

4.5.2 Related Works

In **related works**, the concept of lesser-resourced languages in computational linguistics has come into focus in recent years. In particular, I followed Anssi Yli-Jyrä's recent work on African languages (Yli-Jyrä et al., 2005) and the work presented in the previous years at the Speech and Language Technology for Minority Languages workshop, where I delivered this paper. The concept of using Wikipedia as a basis for a statistical language model was likewise a new common theme in the LREC conferences.

4.5.3 Results

The most central **results** of this work show that the Wikipedias for small language communities will give some gains in spelling correction, when properly coupled with a rule-based language model. This is despite the fact that Wikipedias of lesser-resourced languages are comparably small and of not very high quality, especially when related to the quality of correct spelling. I have hypothesised that the underlying reason for this is that in events where simple spell-checking like edit distance or confusion sets produce ambiguous results, in the majority of the cases it is beneficial to simply opt for the most common word form from these results.

As an additional result in this article, I show how to use language models to automatically generate the baseline error models. This finding is expanded in Section 6.

In general, the concept of freely available and open-source, and crowd-sourced, data should be a common focus in the research of computational linguistics in lesser-resourced languages. This is one of the main sources of large amounts of free data that most communities are willing to produce, and it does not require advanced algorithms or an expert proofreader to make the data usable for many of the applications of computational linguistics.

4.6 Conclusions

In this chapter, I have discussed different approaches to creating statistical language models for finite-state spell-checking. I have shown in this chapter that it is possible to take arbitrary language models for morphologically complex languages and train them with the same approaches regardless of the method and theory according to which the original model was built. I have extended the basic word-form unigram training with methods from n-gram training to be able to cope with the infinite compounding of some morphologically-complex languages and have thus laid out a path forward for all morphologically-complex languages to be trained with limited corpora.

Chapter 5

Error Models

The error modelling for spelling correction is an important part of a spell-checking system and deserves to be treated as a separate topic. The weighted finite-state formulation of *error models* in a finite-state spell-checking system in particular is a relatively novel idea. There are some mentions of finite-state error modelling in the literature (Agirre et al., 1992; Van Noord and Gerdemann, 2001; Savary, 2002; Mohri, 2003). Error modelling, however, has been a central topic for non-finite-state solutions to spell-checking, cf. Kukich (1992a); Mitton (2009); Deorowicz and Ciura (2005), and I believe that a pure and well-formed finite-state solution is called for.

One of the typical arguments for not using finite-state error models is that specific non-finite-state algorithms are faster and more efficient than building and especially applying finite-state models. In the articles, where I construct finite-state models for spell-checking, I have run some evaluations on the resulting system to prove that the finite-state system is at usable speed and memory consumption levels. A thorough evaluation of this is given in Chapter 6.

The theoretical and practical motivation for suggesting finite-state models for error correction is that the expressive power of a *weighted finite-state transducer (WFST)* is an ideal format for specifying the combinations of the spelling errors that can be made and corrected, and it is possible to build more complex statistical models consisting of different types of errors by combining them from building blocks. The fact that traditional non-finite-state spelling correctors use a number of different algorithms starting from all the variants and modifications of the edit distance to arbitrary context-based string rewriting to phonemic folding schemes provides a good motivation to have such arbitrarily combinable systems to create these error models outside the application development environment, i.e. within end users' systems.

The rest of this chapter is laid out as follows: In Section 5.1, I describe how the finite-state application of error modelling is implemented in our spelling correction system, and in Section 5.2, I show how our variant of finite-state edit distance style error models are

constructed. In Section 5.3, I go through the existing error models used in the popular spelling software Hunspell, and show their finite-state formulations. In Section 5.4, I show how to build and apply the phonemic key type of error models in finite-state form. In Section 5.5, I discuss context-based language models in finite-state spelling correction for morphologically complex languages with limited resources that is, mostly the limitations and necessary modifications. Finally in Section 5.6, I discuss some other error correcting models and their finite-state formulations.

5.1 The Application of Finite-State Error Models

There are many approaches to spelling correction with finite-state automata. It is possible to use various generic graph algorithms to work on error-tolerant traversal of the finite-state graph, as is done by Huldén (2009). It is also possible to create a finite-state network that already includes potential error mapping, as is shown by Schulz and Mihov (2002). The approach that I use is to apply basic finite-state algebra with a transducer working as the error model. This is a slight variation of a model originally presented by Mohri (2003). We remove the errors from misspelled strings by two compositions, performed serially in a single operation: $\mathcal{M}_{\text{suggestions}} = (\mathcal{M}_{\text{wordform}} \circ \mathcal{M}_{\text{error}} \circ \mathcal{M}_{\text{language}})_2$, where $\mathcal{M}_{\text{suggestions}}$ is automaton encoding suggestions, ($\mathcal{M}_{\text{wordform}}$ is automaton containing misspelt word form, $\mathcal{M}_{\text{E}}$ is error model, and $\mathcal{M}_{\text{L}}$) is language model. This specific form of finite-state error correction was introduced in (III), but the underlying technology and optimisations were more widely documented by Lindén et al. (2011).

The implications of using weighted finite-state algebra are two-fold. On the one hand, we have the full expressive power of the weighted finite-state systems. On the other hand, the specialised algorithms are known to be faster, e.g. the finite-state traversal shown by Huldén (2009) is both fast and has the option of having regular languages as contexts, but no true weights. So there is a trade-off to consider when choosing between these methods, but considering the popularity of statistical approaches in the field of natural language engineering the time trade-off may be palatable for most end-users developing language models.

The implication of the fully weighted finite-state framework for spell-checking is that we can e.g. apply statistics to all parts of the process, i.e., the probability of the input, the conditional probability of a specific combination of errors and the probability of the resulting word form in context.

5.2 Edit Distance Measures

The baseline error model for spelling correction since its inception in the 1960s has been the Damerau-Levenshtein edit distance (Damerau, 1964; Levenshtein, 1966). There

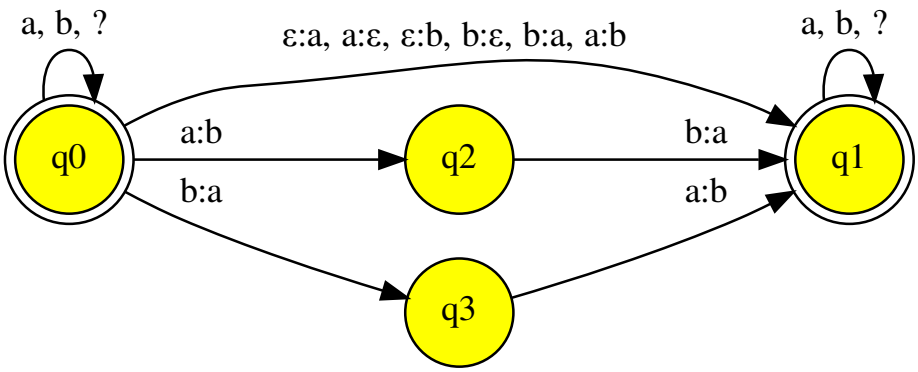

Figure 5.1: A non-deterministic unweighted transducer implementing edit distance 1, where $\Sigma = \mathrm{a}, \mathrm{b}, \varepsilon$. The deletion, addition and change is shown in the edits from q_0 to q_1, and the states q_2 and q_3 are used as a memory for the swaps of *ab* to *ba* and *ba* to *ab* respectively.

have been multiple formulations of the finite-state edit distance, but in this section we describe the one used by our systems. This formulation of the edit distance as a transducer was first presented by Schulz and Mihov (2002).

Finite-state edit distance is a rather simple concept; each of the error types except swapping is represented by a single transition of the form $\varepsilon : x$ (for insertion), $x : \varepsilon$ (for deletion) or $x : y$ (for change). Swaps requires an additional state in the automaton as a memory for the symbols to be swapped at the price of one edit that is a path $\pi_{x:yy:x}$. This unweighted edit distance 1 error model is shown in Figure 5.1.

There are numerous possibilities to formalise the edit-distance in weighted finite-state automata. The simplest form is a cyclic automaton that is capable of measuring arbitrary distances, using the weights as a distance measure. This weighted automaton for measuring edit distances is shown in Figure 5.2. This form is created by drawing arcs of the form $\varepsilon : x :: w$, $x : \varepsilon :: w$ and $x : y :: w$, from the initial state to itself, where w is the weight of a given operation. If the measure includes the swap of adjacent characters, the extra states and paths of the form $\pi_{x:yy:x::w}$ need to be defined. It is possible to use this automaton to help automatic harvesting or in training of an error model. The practical weighted finite-state error models used in real applications are of the same form as the unweighted ones defined earlier, with limited maximum distance, as the application of an infinite edit measure is expectedly slow.

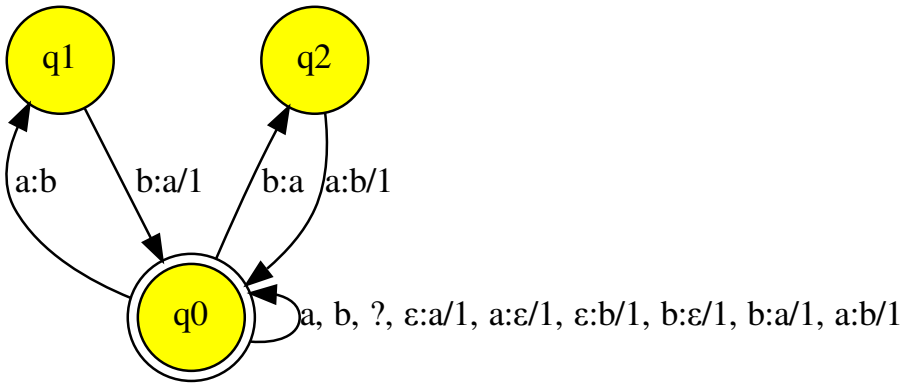

Figure 5.2: A Weighted automaton for measuring arbitrary edit distance where $\Sigma = \mathrm{a}, \mathrm{b}, \varepsilon$ and uniform weights of 1 per edit. The states q_1 and q_2 are used as a memory for the swaps of *ab* to *ba* and *ba* to *ab* to keep them at the weight of 1.

Mohri (2003) describes the mathematical background of implementing a weighted edit distance measure for finite-state automata. It differs slightly from the formulation we use in that it describes edit distance of two weighted automata exactly, whereas our definition is suitable for measuring the edit distance of a string or path automaton and the target language model. The practical differences between Mohri's formulations and the automata I present seem to be minimal, e.g. placing weights into the final state instead of on the arcs.

5.3 Hunspell Error Correction

Main Article: *Building and Using Existing Hunspell Dictionaries and TEX Hyphenators as Finite-State Automata* by Tommi A. Pirinen and Krister Lindén. In this article we have formulated the Hunspell's string-algorithm algorithms for error correction and their specific optimisations in terms of finite-state automata.

5.3.1 Motivation

The main **motivation** for this research was to reimplement Hunspell's spell-checking in a finite-state form. In terms of error modelling it basically consists of modifications of the edit distance for speed gains. One modification is to take the keyboard layout into

account when considering the replacement type of errors. Another addition is to sort and limit the alphabet used for other error types. Finally, Hunspell allows the writer of the spell-checking dictionary to define specific string-to-string mutations for common errors.

5.3.2 Results

The finite-state formulation of these errors is a combination of limitations and modifications to the edit distance automaton, disjuncted with a simple string-to-string mapping automaton. The key aspect of Hunspell's modified edit distance formulations are optimising the speed of the error lookup using configurable, language specific, limitations of the edit distance algorithm's search space. The other correctional facilities are mainly meant to cover the type of errors that cannot be reached with regular edit distance modifications. The effects of these optimisations are further detailed in Chapter 6.

The settings that Hunspell gives to a user are the following: A `KEY` setting to limit characters that can be *replaced* in the edit distance algorithm to optimise the replacements to adjacent keys on a typical native language keyboard. A `TRY` setting to limit the possible characters that are used in the insertion and deletion part of the edit distance algorithm. A `REP` setting that can be used for arbitrary string-to-string confusion sets, and a parallel setting called `MAP` used for encoding differences rather than linguistic confusions. Finally, for the English language, there is a `PHONE` setting that implements a variation of the double metaphone algorithm (Philips, 2000).

5.4 Phonemic Key Corrections

Phonemic keys are commonly used for languages where orthography does not have a very obvious mapping to pronunciation, such as English. In the phonemic keying methods the strings of a language are intended to connect with a phonemic or phonemically motivated description. This technique was originally used for cataloguing family names in archives (Russell and Odell, 1918). These systems have been successfully adapted to spell-checking and are in use for English, for example in aspell as the double metaphone algorithm (Philips, 2000). Since these algorithms are a mapping from strings to strings, it is trivially modifiable into a finite-state spelling correction model. Considering a finite-state automaton $\mathcal{M}_{phon}$ that maps all strings of a language into a single string called a phonetic key, the spelling correction can be performed with mapping $\mathcal{M}_{phon} \circ \mathcal{M}_{phon}^{-1}$. For example, a soundex automaton can map strings like squer and *square* into string `S140`, and the composition of S140 to the inverse soundex mapping would produce, among others, *square*, squer, sqr, sqrrr, and so forth. And this set can be applied to the language model collecting only the good suggestions, such as *square*.

5.5 Context-Aware Spelling Correction

Main article: *Improving finite-state spell-checker suggestions with part of speech n-grams* by Tommi A. Pirinen, Miikka Silfverberg, and Krister Lindén. In this article I have attempted to cover spelling correction of morphologically more complex languages with finite-state technology.

5.5.1 Motivation

The **motivation** for this article was to show that a finite-state version of spell-checking is also usable for a context-aware form, and secondarily to explore the limitations and required changes for the context-based language models that are needed for more morphologically complex languages to get similar results as are attained with simple statistical context models for morphologically poor languages.

5.5.2 Related Works

The **related works** include the well-known results on English for the same task implemented in non-finite-state approaches. In general, the baseline of the context-based spelling correction was probably established by Mays et al. (1991) and been revisited many times, e.g. by Wilcox-O'Hearn et al. (2008), and has usually been found to be beneficial for the languages that have been studied. The examples I have found are all of the morphologically poor languages such as English or Spanish. In my research I have tried to replicate these results for Finnish, but the results have not been as successful as with English or Spanish . The basic word form n-grams were not seen as an efficient method for Spanish by e.g. Otero et al. (2007), instead the method was extended by studying also the part-of-speech analyses to rank the suggestion lists for a spell-checker. In my article (VIII), we reimplemented a finite-state system very similar to the one used for Spanish, originally unaware of these parallel results.

5.5.3 Results

Results – given in Table 5.1 reproduced from (VIII) – show that the regular n-grams, which are used for English to successfully improve spelling correction, are not so effective for Finnish. Furthermore, the improvement that is gained for Spanish when applying the POS n-grams in this task does not give equally promising results for Finnish, but do improve the quality of suggestions, by around 5% points.

One of the negative results of the paper is that the time and memory consumption of the POS-based or word-form-based spelling correction is not yet justifiable for general consumer products as a good tradeoff. There have been many recent articles exploring the optimisation side of the context-based spell-checking that should be seen as the future

Algorithm	1	2	3	4	5	1—10
English						
Edit distance 2 (baseline)	25.9%	2.4%	2.4%	1.2%	3.5%	94.1%
Edit distance 2 with Unigrams	28.2%	5.9%	29.4%	3.5%	28.2%	97.6%
Edit distance 2 with Word N-grams	29.4%	10.6%	34.1%	5.9%	14.1%	97.7%
Edit distance 2 with POS N-grams	68.2%	18.8%	3.5%	2.4%	0.0%	92.9%
Finnish						
Edit distance 2 (baseline)	66.5%	8.7%	4.0%	4.7%	1.9%	89.8%
Edit distance 2 with Unigrams	61.2%	13.4%	1.6%	3.1%	3.4%	88.2%
Edit distance 2 with Word N-grams	65.0%	14.4%	3.8%	3.1%	2.2%	90.6%
Edit distance 2 with POS N-grams	71.4%	9.3%	1.2%	3.4%	0.3%	85.7%

Table 5.1: Precision of suggestion algorithms with real spelling errors. Reproduced from (VIII)

work for context-based finite-state spelling correction. The optimisation of the n-gram models has been brought up many times in non-finite-state solutions too, e.g. Church et al. (2007). While mainly an engineering problem, a lot is left to be desired from this development before practical end-user applications for contextual spell-checking can be considered.

5.6 Other Finite-State Error Models

Deorowicz and Ciura (2005) present an extensive study on error modelling for spelling correction. Some of the models have been introduced earlier in this chapter.

One of the optimisations used in some practical spell-checking systems, but not in Hunspell, is to avoid modifying the first character of the word in the error modelling step. The gained improvement in speed is notable. The rationale for this is the assumption that it is rarer to make mistakes in the first character of the word – or perhaps it is easier to notice and fix such mistakes. There are some measurements on this phenomenon by Bhagat (2007), but the probabilities of having the first-letter error in the studies they refer to vary from 1% up to 15%, so it is still open to debate. In measurements I made for morphologically complex languages in (VIII), I also found a slight tendency towards the effect that the errors are more likely in non-initial parts of the words, and these account for word-length effects.

The concept of creating an error model from a (possibly annotated) error corpus is plausible for finite-state spelling-correction as well. The modification of the training logic as presented by Church and Gale (1991) into finite-state form is mostly trivial. E.g. if we consider the building of a finite-state language model from a corpus of word

forms, the corpus of errors is the same set of string pairs with weights of

$$w(x:y) = -\log \frac{f(x:y) + \alpha}{\mathcal{D}_{\text{errors}} + \alpha}, \tag{5.1}$$

where $x, y \subset \Sigma^\star$ are error string pairs $x : y$ in a collection of all errors $\mathcal{D}_{\text{errors}}$.

5.7 Conclusions

In this chapter, I have enumerated some of the most important finite-state formulations of common error models found in spell-checkers and shown that they are implementable and combinable under basic finite-state algebra. I have re-implemented a full set of error models used by the current *de facto* standard spelling corrector Hunspell in a finite state form demonstrating that it is possible to replace string-algorithm correction algorithms by a sound finite-state error model.

Chapter 6

Efficiency of Finite-State Spell-Checking

One of the motivations for finite-state technology is that the time complexity of the application of finite-state automata is known to be linear with regard to the size of the input as long as finite-state automata can be optimised by operations like determinisation. In practice, however, the speed and size requirements of the finite-state automata required for language models, error models and their combinations are not easy to predict, and the worst case scenarios of these specific combinatorics are still theoretically exponential, because the error models and language models are kept separated and composed during run time due to space restrictions. This is especially important for error modelling in spell-checking, since it easily goes towards the worst case scenario, i.e. the theoretically ideal naive error model that can rewrite any string into any other string with some probabilities attached is not easily computable.

Towards the final parts of my research on finite-state spell-checking, I set out to write a few larger-scale evaluation articles to demonstrate the efficiency and usability of finite-state spell-checking. For larger scale testing, I also sought to show how the linguistically common but technologically almost undealt with concept of *morphological complexity* practically relates to the finite-state language models.

The evaluation of finite-state spelling correction needs to be contrasted with the existing comparisons and evaluations of spell-checking systems. Luckily, spell-checking has been a popular enough topic so that a number of recent evaluations are available. Also the informative introduction to spell-checking by Norvig (2010) has been a valuable source of practical frameworks for a test setup of the evaluation of spelling checkers as well as the ultimate baseline comparison for English language spell-checking.

The evaluation of finite-state spelling checkers in this chapter consists of two different goals: when working with English, we merely aim to reproduce the results that are published in the literature and are well known. When working with more *morphologi-*

cally complex languages, we aim to show that the more complex the languages we select, the more work and optimisations are needed to attain reasonable performance for baseline spell-checking and correction. Throughout the chapter, we have tried to follow the linguistically motivated selection of languages, i.e. selecting at least one from moderately rich and one from poly-synthetic languages; suitable languages for these purposes with open-source, freely available implementations have been acquired from the University of Tromsø's server.[1] From there we have selected North Saami as a moderately rich and Greenlandic as an example of a poly-synthetic language.

The efficiency evaluation is divided into two sections. In Section 6.1, I evaluate the speed efficiency and optimisation schemes of finite-state spelling checkers together with Sam Hardwick who did most of the groundwork on the finite-state spell-checking implementation in the article by e.g., Lindén et al. (2011). Then in Section 6.2, I measure the effects these optimisations have on the quality of spell-checking.

6.1 Speed of Finite-State Spell-Checking

Main article: *Effect of Language and Error Models on Efficiency of Finite-State Spell-Checking and Correction* by Tommi A. Pirinen and Sam Hardwick. In this article we set out to prove that finite-state spell-checking is an efficient solution and document how builders of spelling checkers can tune the parameters of their models.

6.1.1 Motivation

The **motivation** for the article was practical, in discussions with people building alternative spelling checkers, as well as with alpha testers of our more complex spell-checkers it was commonly noted that the speed might be an issue for finite-state spell-checking systems to gain ground from the traditional solutions. It is noteworthy, that the formulation of the system was from the beginning built so that the developers of the language and error models can freely make these optimisations as needed.

When discussing such practical measurements as the speed of a spell-checker, care must be taken in evaluation metrics. The most common use of a spell-checker is still inter-active, where a misspelt word is underlined when the user makes a mistake, and the system is capable of suggesting corrections when a user requests one. This sets the limits that a system should be capable of checking text as it is typed, and responding with suggestions as demanded. It is well known that as acceptors and language recognisers, finite-state language models can answer the question of whether the string is in a given language in linear time. In practical measurements, this amounts to spell-checking times up to 100,000 strings per second e.g. by Silfverberg and Lindén (2009). The central

[1] `http://giellatekno.uit.no`

Error model	English	Finnish	Greenlandic
Edit distance 1	0.26	6.78	4.79
Edit distance 2	7.55	220.42	568.36
No firsts ed 1	0.44	3.19	3.52
No firsts ed 2	1.38	61.88	386.06

Table 6.1: Effect of language and error models on speed (time in seconds per 10,000 word forms). Reproduced from (IX).

evaluation and optimisation was therefore performed on the error correction part, which varies both with the language model and the error model selected.

6.1.2 Related Works

In the article, we study different **related** optimisation schemes from non-finite-state spell-checking and evaluate how different error models and parameters affect the speed of finite-state spell-checking. We also ran the tests on a scale of morphologically different languages to see if the concept of morphological complexity is reflected in the speed of spell-checking.

The optimisation of speed in spell-checking software has been a central topic in my research since the beginning. There are numerous solutions that we ported into finite-state form to test their efficiency, and a few methods related to problems with the finite-state formulation of the error models (i.e. the generated duplicate correction paths in the edit distance models larger than 1 edits long, and the limitation of the result set size during traversal).

6.1.3 Results

The **results** of the evaluation – reproduced in Table 6.1 from the article (IX) – are enlightening. Basically the main finding of the article is that disallowing the change of the initial letter (*no firsts ed*) in the error model provides a significant boost of performance.

Ignoring errors in the first-position of the word is known from many practical systems, but is little documented. Some of the research, such as Bhagat (2007), provides some statistics on the plausibility of a method: it is more likely to make a mistake in other parts of the words than the very first letter, but does not offer hypotheses why this happens. Practically, we replicated these results for our set of morphologically correct languages as well, giving further evidence that it may be used as an optimisation scheme with a reasonable quality trade-off.

It is likewise interesting that even though we tried to devise methods to prevent the larger error models from doing unnecessary things like removing and adding precisely

the same characters in succession, this did not provide a significant addition to the search speed of the corrections.

The comparative evaluation of the different language models is relatively rarely found in literature. This could be due to the fact that scientific writing in spell-checking is still more oriented towards algorithmics research, where the language models of related systems do not have an influence on the asymptotic speed of the process and are thus considered uninteresting. Most of the studies are focused on measuring overall systems of single languages, such as Arabic (Attia et al., 2012), Indian languages (Chaudhuri, 2002),[2] Hungarian (Trón et al., 2005) or Spanish (Otero et al., 2007), as well as all the research on English (Mitton, 1987).

6.2 Precision of Finite-State Spell-Checking

Main article: *Speed and Quality Trade-Offs in Weighted Finite-State Spell-Checking*. In this article, I take a broad look at the evaluation of all the language and error models and testing approaches we have developed in the past few years of research and compare them with existing results in real systems.

6.2.1 Motivation

The **motivation** for this article was to perform a larger survey of all the finite-state spelling methods that have been implemented in my thesis project. In the paper I aim to show that the current state-of-the-art for finite-state spell-checking is starting to be a viable option for many common end-user applications. This article is a practical continuation of the speed optimisation evaluation, where I studied the concept of speeding up finite-state spell-checking systems by manipulating error and language models. In order for these optimisations to be usable for end-users, I needed to show that the quality degradation caused by the optimisation schemes that remove a part of the search space is not disproportionately large for the error models that are effective enough to be used in real-world systems.

6.2.2 Results

When measuring the quality of spelling error correction, the basic finding is that the simple edit-distance measure at length one will usually cover the majority of errors in typical typing error situations. For English, the original statistics were 95% (Damerau, 1964),

[2]The wording “Indian language(s)” is from the article, and while not a linguistically coherent group, is commonly used in the field of Indian natural language engineering and e.g., in linguistic conferences such as COLING 2012, CICLING 2012 organised in India, as the name of the topic focus. The specific examples in the article seem to be about Bangla.

Rank: **Language and error models**	1^{st}	2^{nd}	3^{rd}	4^{th}	5^{th}	rest
English Hunspell	59.3	5.8	3.5	2.3	0.0	0.0
English aspell	55.7	5.7	8.0	2.2	0.0	0.0
English w/ 1 error	66.7	7.0	5.2	1.8	1.8	1.8
English w/ 1 non-first error	66.7	8.8	7.0	0.0	0.0	1.8
English w/ 1 Hunspell error	45.6	8.7	0.0	0.0	0.0	0.0
English w/ 2 errors	71.9	14.0	0.0	3.5	3.5	3.5
English w/ 2 non-first errors	71.3	17.5	0.0	1.8	3.5	1.8
English w/ 2 Hunspell errors	73.7	12.3	1.8	0.0	0.0	3.5
English w/ 3 errors	73.7	14.0	0.0	3.5	3.5	5.3
English w/ 3 non-first errors	73.7	17.5	0.0	1.8	3.5	3.5
English w/ 3 Hunspell errors	73.7	12.3	1.8	0.0	0.0	8.8
Finnish aspell	21.1	5.8	3.8	1.9	0.0	0.0
Finnish w/ 1 errors	54.8	19.0	7.1	0.0	0.0	0.0
Finnish w/ 2 errors	54.8	19.0	7.1	2.4	0.0	7.1
Finnish w/ 1 non-first error	54.8	21.4	4.8	0.0	0.0	0.0
Finnish w/ 2 non-first errors	54.8	21.4	4.8	0.0	0.0	7.1
Greenlandic w/ 1 error	13.3	2.2	6.7	2.2	0.0	8.9
Greenlandic w/ 1 nonfirst error	13.3	2.2	6.7	2.2	0.0	8.9
Greenlandic w/ 2 errors	13.3	6.7	4.4	0.0	0.0	35.6
Greenlandic w/ 2 nonfirst errors	13.3	6.7	4.4	0.0	0.0	35.6

Table 6.2: Effect of different language and error models on correction quality (in%). Reproduced from (X).

with various newer studies giving similar findings between 79% and 95%, depending on the type of corpora (Kukich, 1992a). The numbers for morphologically complex languages are not in the same range. Rather the edit distance of one gains around 50–70% coverage. One reason for this may be that the length of a word is a factor in the typing error process. Thus it is more likely to type two or more typos and leave them uncorrected with a language where the average word length is 16 compared with one where it is 6. Intuitively this would make sense, and the figures obtained from my tests in article (X) – reproduced in Table 6.2 – provide some evidence to that effect.

6.3 Conclusions

In this chapter, I have studied the aspects of finite-state spell-checking that would ensure a plausible end-user spell-checker in a practical application. I have shown that decrease in speed is negligible when moving from optimised specific non-finite-state methods

to correct strings, or traverse a finite-state network to an actual finite-state algebra. The speed of finite-state solutions for correcting morphologically complex languages is faster than that of current software solutions such as Hunspell.

I have tested a set of morphologically different languages and large corpora to show the usability of finite-state language and error-models for practical spell-checking applications. I have also shown that it is possible to tune with little effort the systems for interactive spell-checking using the finite-state algebra.

The practical findings show the requirement of different speeds and quality factors for different applications of spelling checkers. The research presented here shows, that by varying both the automata that make up the language model and the error model of the spelling checker, it is possible to create suitable variants of spelling checkers for various tasks by simple fine-tuning the parameters of the finite-state automaton.

Chapter 7

Conclusion

In this thesis, I have researched the plausibility of making finite-state language and error models for real-world, finite-state spell-checking and correction applications, and presented approaches to formulate traditional spell-checking solutions for finite-state use. The contributions of this work are divided into two separate sections: Section 7.1 summarises the scientific contributions of the work that is in my thesis, and Section 7.2 recounts the results of this thesis in terms of real-world spelling checkers that can be used in end-user systems. Finally, in Section 7.3, I go through the related research questions and practical ideas that the results of this thesis lead into.

7.1 Scholarly Contributions

One of the main contributions, I believe, is the improvement and verification of the statistical language modelling methods in the context of weighted finite-state automata. I believe that throughout the thesis, by continuously using the initial finding of the word-form-based compound word training approach introduced in (I), I have shown that *morphologically complex* languages can be statistically trained as weighted finite-state automata and I maintain that this approach should be carried out in the future finite-state language models.

In the spelling correction mechanics, I have shown that weighted finite-state transducers implement a feasible error model. According to my experiments, it can be reasonably used in interactive spelling correction software in stead of any traditional spelling correction solution.

In my contribution to the practical software engineering side of language modelling, I have shown that a generalised finite-state formula based on a knowledge of morphology can be used to compile the most common existing language models into automata.

Finally, to tie in the scientific evaluation of finite-state spell-checking as a viable alternative, I have performed a large-scale testing on languages with a wide range of

morphological complexity, with different resources, and shown that all are plausibly usable for typical spelling correction situations.

7.2 Practical Contributions

The spell-checking software that is the topic of this thesis is a very practical real-world application that, despite this being an academic dissertation, cannot be ignored when talking about the contributions of this thesis. One of the important contributions, I believe, is the alpha testing version of the Greenlandic spelling correction for OpenOffice, providing a kind of a proof that finite-state spell-checking of Greenlandic is implementable.

Real-world software, consisting of components such as spell-checking libraries libvoikko [1] and enchant,[2] GNOME desktop environment,[3] office suites OpenOffice.org and Libreoffice,[4] and Firefox browser,[5] have been a good testing ground for other morphologically complex languages, including Finnish and North Saami, but also a range of forthcoming Uralic languages currently being implemented at the University of Helsinki,[6] most of which have never had a spell-checking system at all.

The statistical and weighted methods used to improve language models have been directly ported into classical tool chains and work flows of the finite-state morphology, and are now supported in the HFST tools including `hfst-lexc` for traditional finite-state morphologies.

7.3 Future Work

In terms of error correcting, the systems are deeply rooted in the basic models of typing errors, such as the Levenshtein-Damerau edit distance. It would be interesting to try to implement more accurate models of errors based on findings of cognitive-psychological studies on the kinds of errors that are actually made – this research path I believe has barely scratched the surface with optimisations like *avoid typing mistakes at the beginning of the word*. Furthermore, it should be possible to make adaptive error models using a simple feedback system for finite-state error-models.

During the time I spent writing this thesis, the world of spell-checking has changed considerably with the mass market of a variety of input methods in the new touch screen

[1] `http://voikko.sf.net`
[2] `http://abiword.com/enchant`
[3] `http://gnome.org`
[4] `http://libreoffice.org`
[5] `http://firefox.org`
[6] `http://www.ling.helsinki.fi/~rueter/UrjSpell/testaus_fi.shtml` (temporary URL for testing phase)

user interfaces. When I started my thesis work, by far the most common methods to input text were regular 100+ key PC keyboards and key pads on mobile phones with the T9 input method.[7] This is no longer the case, as input methods like Swype, XT9, and so on, are gaining popularity. A revision of the methods introduced in this thesis is required, especially since the input methods of these systems are more heavily reliant on high-quality, automatic spelling correction than before.[8] This point will slightly affect the considerations of the practical evaluations of the speed of spell-checking; new spelling correctors coupled with predictive entry methods need to be invoked after each input event instead of only when correction is required.

On the engineering and software management side, one practical, even foreseeable, development would be to see these methods being adopted into use in real-world spell-checkers. The actual implementation has been done with the help of spell-checking systems like voikko, and enchant. The morphologically complex languages that are represented in my thesis lack basic language technology support partially due to the dominance of too limited non-finite state forms of language modelling.

The concept of context-aware spell-checking with finite-state systems was merely touched upon by article (VIII). However, the results showed some promise and pinpointed some problems, and with some work it may be possible to gain as sizable improvements for morphologically complex languages as for morphologically simple ones. One possible solution to be drawn from the actual results of the other articles in the thesis is to use morph-based statistics for all languages to get more robust statistics and confirmatory results. In order to get the speed and memory efficiency to a level that is usable in everyday applications, I suspect there is room for much improvement through basic engineering decisions and optimisations. This has also been a topic in much of the recent spell-checking research for non-finite-state systems, see e.g. Carlson et al. (2001).

The concept of context-based error-detection, real-word error detection, and other forms of grammar correction has not been dealt with at a very large-scale in this thesis. That has been done partly on purpose, but in fact the basic context-aware real-word error detection task does not really differ in practice from its non-finite-state versions, and the conversion is even more trivial than in the case of the error correction part discussed in article (VIII). The main limiting factor for rapidly testing such a system is the lack of error corpora with real-word spelling errors.

As a theoretically interesting development, it would be nice to have formal proof that the finite-state spell-checking and correction models are a proper superset of the Hunspell spell-checking methods, as my quantitative evaluation and existing conversion algorithm for the current implementation shows that this is plausible and even likely.

[7] `http://www.t9.com`

[8] As a case in point, there are collections of humorous automatic spelling corrections caused by these systems, on the internet, see e.g., `http://www.damnyouautocorrect.com/`.

References

Agirre, E., Alegria, I., Arregi, X., Artola, X., de Ilarraza, A. D., Maritxalar, M., Sarasola, K., and Urkia, M. (1992). Xuxen: A spelling checker/corrector for Basque based on two-level morphology. In *Proceedings of The Third Conference on Applied Natural Language Processing*, pages 119–125. Association for Computational Linguistics.

Aho, A. and Corasick, M. (1975). Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340.

Aho, A., Lam, M., Sethi, R., and Ullman, J. (2007). *Compilers: principles, techniques, and tools*. Pearson/Addison Wesley, second edition.

Al-Mubaid, H. and Truemper, K. (2006). Learning to find context based spelling errors. *Data Mining and Knowledge Discovery Approaches Based on Rule Induction Techniques*, pages 597–627.

Attia, M., Pecina, P., Samih, Y., Shaalan, K., and van Genabith, J. (2012). Improved spelling error detection and correction for Arabic. pages 103–112.

Beesley, K. (2004). Morphological analysis and generation: A first step in natural language processing. In *First Steps in Language Documentation for Minority Languages: Computational Linguistic Tools for Morphology, Lexicon and Corpus Compilation, Proceedings of the SALTMIL Workshop at LREC*, pages 1–8.

Beesley, K. R. and Karttunen, L. (2003). *Finite State Morphology*. CSLI publications.

Bhagat, M. (2007). Spelling error pattern analysis of Punjabi typed text. Master's thesis, Thapar University.

Bigert, J. (2005). *Automatic and unsupervised methods in natural language processing*. PhD thesis, Royal Institute of Technology (KTH).

Bigert, J., Ericson, L., and Solis, A. (2003). AutoEval and Missplel: Two generic tools for automatic evaluation. In *Nodalida 2003*, page 7, Reykjavik, Iceland.

Brants, T., Popat, A. C., Xu, P., Och, F. J., and Dean, J. (2007). Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 858–867, Prague, Czech Republic. Association for Computational Linguistics.

Brill, E. and Moore, R. C. (2000). An improved error model for noisy channel spelling correction. In *ACL 2000: Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 286–293, Morristown, NJ, USA. Association for Computational Linguistics.

Carlson, A., Rosen, J., and Roth, D. (2001). Scaling up context-sensitive text correction. In *Proceedings of the Thirteenth Conference on Innovative Applications of Artificial Intelligence Conference*, pages 45–50. AAAI Press.

Chaudhuri, B. (2002). Towards Indian language spell-checker design. In *Language Engineering Conference, 2002. Proceedings*, pages 139–146. IEEE.

Chege, K., Wagacha, P., De Pauw, G., Muchemi, L., and Ng'ang'a, W. (2010). Developing an open source spell-checker for Gīkūyū. *AfLaT 2010*, pages 31–35.

Chen, S. and Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393.

Church, K. and Gale, W. (1991). Probability scoring for spelling correction. *Statistics and Computing*, 1(2):93–103.

Church, K., Hart, T., and Gao, J. (2007). Compressing trigram language models with Golomb coding. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 199–207.

Creutz, M., Lagus, K., Lindén, K., and Virpioja, S. (2005). Morfessor and Hutmegs: Unsupervised morpheme segmentation for highly-inflecting and compounding languages.

Culy, C. (1987). The complexity of the vocabulary of Bambara. In *The Formal Complexity of Natural Language*, pages 349–357. Springer.

Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176.

Deorowicz, S. and Ciura, M. (2005). Correcting spelling errors by modelling their causes. *International Journal of Applied Mathematics and Computer Science*, 15(2):275–285.

Federico, M., Bertoldi, N., and Cettolo, M. (2008). IRSTLM: an open source toolkit for handling large scale language models. In *Interspeech*, pages 1618–1621.

Golding, A. (1995). A Bayesian hybrid method for context-sensitive spelling correction. In *Proceedings of the Third Workshop on Very Large Corpora*, volume 3, pages 39–53. Massachusetts Institute of Technology Cambridge MA.

Golding, A. and Roth, D. (1999). A winnow-based approach to context-sensitive spelling correction. *Machine Learning*, 34(1):107–130.

Gorin, R. E. (1971). SPELL: A spelling checking and correction program. On-line manual; web page. Retrieved 1st of August 2013: `http://www.saildart.org/`, entries UP, DOC: SPELL.REG.

Greenberg, J. H. (1960). A quantitative approach to the morphological typology of language. *International Journal of American Linguistics*, 26(3):178–194.

Greenfield, K. and Judd, S. (2010). Open source natural language processing. Master's thesis, Worchester Polytechnic Institute.

Haspelmath, M., Bibiko, H.-J., Hagen, J., and Schmidt, C. (2005). *The World Atlas of Language Structures*, volume 125. Oxford University Press Oxford.

Hassan, A., Noeman, S., and Hassan, H. (2008). Language independent text correction using finite state automata. In *Proceedings of the Third International Joint Conference on Natural Language Processing*, volume 2, pages 913–918.

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

Huldén, M. (2013). Weighted and unweighted transducers for tweet normalization. In Alegria, I., Aranberri, N., Fresno, V., Gamallo, P., Padró, L., Vicente, I. S., Turmo, J., and Zubiaga, A., editors, *Tweet Normalisation Workshop at 29th Conference of the Spanish Society for Natural Language Processing (SEPLN 2013)*, pages 69–72.

Karlsson, F. (1992). Swetwol: A comprehensive morphological analyser for Swedish. 15:1–45.

Karlsson, F. (2007). Constraints on multiple center-embedding of clauses. *Journal of Linguistics*, 43(2):34.

Karttunen, L. (1995). The replace operator. In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics*, pages 16–23. Association for Computational Linguistics.

Karttunen, L. (2006). Numbers and Finnish numerals. *SKY Journal of Linguistics*, 19:407–421.

Karttunen, L., Kaplan, R. M., Zaenen, A., et al. (1992). Two-level morphology with composition. In *COLING 1992*, pages 141–148.

Kernighan, M., Church, K., and Gale, W. (1990). A spelling correction program based on a noisy channel model. In *Proceedings of the 13th conference on Computational Linguistics*, pages 205–210. Association for Computational Linguistics.

Knuth, D. (1973). *The Art of Computer Programming*, volume 3. Addison-Wesley, Reading, MA.

Knuth, D. E. (1984). Literate programming. *The Computer Journal*, 27(2):97–111.

Kobus, C., Yvon, F., and Damnati, G. (2008). Normalizing SMS: are two metaphors better than one. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING 2008)*, pages 441–448.

Kokkinakis, D. (2008). A semantically annotated swedish medical corpus. In Calzolari, N., Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., and Tapias, D., editors, *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2008/.

Kornai, A. (2002). How many words are there. *Glottometrics*, 4:61–86.

Koskenniemi, K. (1983). *Two-level Morphology: A General Computational Model for Word-Form Recognition and Production*. PhD thesis, University of Helsinki.

Kukich, K. (1992a). Spelling correction for the telecommunications network for the deaf. *Commun. ACM*, 35(5):80–90.

Kukich, K. (1992b). Techniques for automatically correcting words in text. *ACM Comput. Surv.*, 24(4):377–439.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics—Doklady 10, 707–710. Translated from Doklady Akademii Nauk SSSR*, pages 845–848. Untranslated version 1965.

Liberman, M. (2012). Noisily channeling Claude Shannon. Blog post in Language Log. Retrieved 1st of August 2013: `http://languagelog.ldc.upenn.edu/nll/?p=4115`.

Lindén, K., Axelson, E., Hardwick, S., Pirinen, T. A., and Silfverberg, M. (2011). Hfst—framework for compiling and applying morphologies. *Systems and Frameworks for Computational Morphology*, pages 67–85.

Lindén, K. and Pirinen, T. (2009a). Weighted finite-state morphological analysis of Finnish compounds with HFST-LEXC. In Jokinen, K. and Bick, E., editors, *Nodalida 2009*, volume 4 of *NEALT Proceedings*, pages 89–95.

Lindén, K. and Pirinen, T. (2009b). Weighting finite-state morphological analyzers using HFST tools. In Watson, B., Courie, D., Cleophas, L., and Rautenbach, P., editors, *FSMNLP 2009*, pages 1–12.

Lindén, K., Silfverberg, M., and Pirinen, T. (2009). Hfst tools for morphology—an efficient open-source package for construction of morphological analyzers. In Mahlow, C. and Piotrowski, M., editors, *sfcm 2009*, volume 41 of *Lecture Notes in Computer Science*, pages 28—47. Springer.

Manning, C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

Maxwell, M. and David, A. (2008). Joint grammar development by linguists and computer scientists. In *Workshop on NLP for Less Privileged Languages, Third International Joint Conference on Natural Language Processing*, pages 27–34, Hyderabad, India.

Mays, E., Damerau, F. J., and Mercer, R. L. (1991). Context based spelling correction. *Inf. Process. Manage.*, 27(5):517–522.

Mitton, R. (1987). Spelling checkers, spelling correctors and the misspellings of poor spellers. *Inf. Process. Manage.*, 23(5):495–505.

Mitton, R. (2009). Ordering the suggestions of a spellchecker without using context. *Nat. Lang. Eng.*, 15(2):173–192.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Comp. Linguistics*, 23:269–311.

Mohri, M. (2003). Edit-distance of weighted automata: general definitions and algorithms. *International Journal of Foundations of Computer Science*, 14(6):957–982.

Norvig, P. (2010). How to write a spelling corrector. Web page. retrieved first of November 2011, available `http://norvig.com/spell-correct.html`.

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

Ortíz-Rojas, S., Forcada, M. L., and Sánchez, G. R. (2005). Construcción y minimización eficiente de transductores de letras a partir de diccionarios con paradigmas. *Procesamiento del lenguaje natural*, 35:51–57.

Otero, J., Grana, J., and Vilares, M. (2007). Contextual spelling correction. *Computer Aided Systems Theory–EUROCAST 2007*, pages 290–296.

Petterson, E., Megyesi, B., and Tiedemann, J. (2013). An SMT approach to automatic annotation of historical text. In Þórhallur Eyþórsson, Borin, L., Haug, D., and Rögnvaldsson, E., editors, *Proceedings of the Workshop on Computational Historical Linguistics at NODALIDA 2013*, Linköping Electronic Conference Proceedings. Linköping University Electronic Press.

Philips, L. (1990). Hanging on the metaphone. *Computer Language*, 7(12):39–44.

Philips, L. (2000). The double metaphone search algorithm. *C/C++ Users Journal*, 18(6):38–43.

Pirinen, T. (2008). Suomen kielen äärellistilainen automaattinen morfologinen analyysi avoimen lähdekoodin menetelmin. Master's thesis, Helsingin yliopisto.

Pirinen, T., Lindén, K., et al. (2010). Creating and weighting Hunspell dictionaries as finite-state automata. *Investigationes Linguisticae*, 21:1–16.

Pirinen, T., Silfverberg, M., and Lindén, K. (2012). Improving finite-state spell-checker suggestions with part of speech n-grams. In *Computational Linguistics and Intelligent Text Processing 13th International Conference, CICLing 2012*.

Pirinen, T. and Tyers, F. (2012). Compiling Apertium morphological dictionaries with HFST and using them in HFST applications. *Language Technology for Normalisation of Less-Resourced Languages*, pages 25–29.

Pirinen, T. A. (2011). Modularisation of Finnish finite-state language description—towards wide collaboration in open source development of morphological analyser. In *Proceedings of Nodalida*, volume 18 of *NEALT proceedings*, pages 299–302.

Pirinen, T. A. (forthcoming). Quality and speed trade-offs in weighted finite-state spell-checking. —. submitted for review.

Pirinen, T. A. and Hardwick, S. (2012). Effect of language and error models on efficiency of finite-state spell-checking and correction. In *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*, pages 1–9, Donostia–San Sebastián. Association for Computational Linguistics.

Pirinen, T. A. and Lindén, K. (2010a). Building and using existing Hunspell dictionaries and TEX hyphenators as finite-state automata. In *Proccedings of Computational Linguistics - Applications, 2010*, pages 25–32, Wisła, Poland.

Pirinen, T. A. and Lindén, K. (2010b). Finite-state spell-checking with weighted language and error models. In *Proceedings of the Seventh SaLTMiL Workshop on Creation and Use of Basic Lexical Resources for Less-resourced Languages*, pages 13–18, Valletta, Malta.

Pitkänen, H. (2006). Hunspell-in kesäkoodi 2006: Final report. Technical report, COSS.fi.

Porta, J., Sancho, J.-L., and Gómez, J. (2013). Edit transducers for spelling variation in Old Spanish. In Þórhallur Eyþórsson, Borin, L., Haug, D., and Rögnvaldsson, E., editors, *Proceedings of the workshop on computational historical linguistics at NODALIDA 2013*, Linköping Electronic Conference Proceedings. Linköping University Electronic Press.

Ranta, A. (2008). How predictable is Finnish morphology? An experiment on lexicon construction. *Resourceful Language Technology: Festschrift in Honor of Anna Sågvall Hein*, pages 130–148.

Raviv, J. (1967). Decision making in Markov chains applied to the problem of pattern recognition. *Information Theory, IEEE Transactions on*, 13(4):536–551.

Russell, R. and Odell, M. (1918). Soundex. *US Patent*, 1261167.

Savary, A. (2002). Typographical nearest-neighbor search in a finite-state lexicon and its application to spelling correction. In *CIAA 2001: Revised Papers from the 6th International Conference on Implementation and Application of Automata*, pages 251–260, London, UK. Springer-Verlag.

Schiller, A. (2006). German compound analysis with wfsc. *Finite-State Methods and Natural Language Processing*, pages 239–246.

Schulz, K. and Mihov, S. (2002). Fast string correction with Levenshtein-automata. *International Journal of Document Analysis and Recognition*, 5:67–85.

Shannon, C. (1948). A mathematical theory of communications, I and II. *Bell Syst. Tech. J*, 27:379–423.

Silfverberg, M. and Lindén, K. (2009). HFST runtime format—a compacted transducer format allowing for fast lookup. In Watson, B., Courie, D., Cleophas, L., and Rautenbach, P., editors, *FSMNLP 2009*.

Silfverberg, M. and Lindén, K. (2010). Part-of-speech tagging using parallel weighted finite-state transducers. In Loftsson, H., Rögnvaldsson, E., and Helgadóttir, S., editors, *Advances in Natural Language Processing — Proceedigns of the 7th International Conference on NLP, IceTAL 2010*, volume 6233, pages 369–381, Reykjavik, Iceland.

Stolcke, A. et al. (2002). SRILM-an extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing*, volume 2, pages 901–904.

Trón, V., Kornai, A., Gyepesi, G., Németh, L., Halácsy, P., and Varga, D. (2005). Hunmorph: open source word analysis. In *Proceedings of the Workshop on Software*, pages 77–85. Association for Computational Linguistics.

Unicode Consortium (2013). The Unicode standard. WWW page. Retrieved 1st of August 2013 (unversioned), refered version 6.2.0; `http://www.unicode.org/versions/latest/`.

Van Noord, G. and Gerdemann, D. (2001). An extendible regular expression compiler for finite-state approaches in natural language processing. In *Automata Implementation*, pages 122–139. Springer.

Watson, B. (2003). A new algorithm for the construction of minimal acyclic DFAs. *Science of Computer Programming*, 48(2):81–97.

Wilcox-O'Hearn, L. A., Hirst, G., and Budanitsky, A. (2008). Real-word spelling correction with trigrams: A reconsideration of the Mays, Damerau, and Mercer model. In Gelbukh, A. F., editor, *Proceedings of CICLing 2008*, pages 605–616. Springer.

Wintner, S. (2008). Strengths and weaknesses of finite-state technology: A case study in morphological grammar development. *Nat. Lang. Eng.*, 14:457–469.

Yli-Jyrä, A. et al. (2005). Toward a widely usable finite-state morphology workbench for less studied languages, 1: Desiderata. *Nordic Journal of African Studies*, 14(4):479–491.

Chapter A

Appendices

A.1 Copyrights of Introduction and Articles

This introduction is published under *Creative Commons Attribution–NonCommercial-NoDerivs 3.0 Unported* licence, which can be found in Appendix A.3.

A.2 Original Articles

This electronic publication does not contain reproductions of the original articles. The viewers of this electronic version of the dissertation should obtain the articles refered from my homepage[1] or search for the articles using Google Scholar.[2]

A.3 Licence

This text is *Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported*.[3]

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN ”AS-IS” BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE. License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE (”CCPL” OR ”LICENSE”). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW.

[1] `http://www.helsinki.fi/%7etapirine/publications`

[2] `http://scholar.google.com`

[3] `http://creativecommons.org/licenses/by-nc-nd/3.0/legalcode`

ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

   (a) **"Adaptation"** means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

   (b) **"Collection"** means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

   (c) **"Distribute"** means to make available to the public the original and copies of the Work through sale or other transfer of ownership.

   (d) **"Licensor"** means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

   (e) **"Original Author"** means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram

the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

(f) **"Work"** means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

(g) **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

(h) **"Publicly Perform"** means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

(i) **"Reproduce"** means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that

are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

   (a) to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections; and,

   (b) to Distribute and Publicly Perform the Work including as incorporated in Collections.

   The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Adaptations. Subject to 8(f), all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Section 4(d).

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

   (a) You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested.

(b) You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.

(c) If You Distribute, or Publicly Perform the Work or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Collection, at a minimum such credit will appear, if a credit for all contributing authors of Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

(d) For the avoidance of doubt:

i. **Non-waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

ii. **Waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to

collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,

iii. **Voluntary License Schemes.** The Licensor reserves the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License that is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b).

(e) Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation.

5. Representations, Warranties and Disclaimer

   UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

   (a) This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities

who have received Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

(b) Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

(a) Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

(b) If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

(c) No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

(d) This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

(e) The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are

sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark ”Creative Commons” or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons’ then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.

Creative Commons may be contacted at `http://creativecommons.org/`.