PRONUNCIATION MODELLING AND BOOTSTRAPPING

MARELIE HATTINGH DAVEL

PRONUNCIATION MODELLING AND BOOTSTRAPPING

By

Marelie Hattingh Davel

Submitted in partial fulfilment of the requirements for the degree

Philosophiae Doctor (Electronic Engineering)

in the

Faculty of Engineering, the Built Environment and Information Technology

at the

UNIVERSITY OF PRETORIA

Advisor: Professor E. Barnard

August 2005

PRONUNCIATION MODELLING AND BOOTSTRAPPING

Bootstrapping techniques have the potential to accelerate the development of language technology resources. This is of specific importance in the developing world where language technology resources are scarce and linguistic diversity is high. In this thesis we analyse the pronunciation modelling task within a bootstrapping framework, as a case study in the bootstrapping of language technology resources.

We analyse the grapheme-to-phoneme conversion task in the search for a grapheme-to-phoneme conversion algorithm that can be utilised during bootstrapping. We experiment with enhancements to the Dynamically Expanding Context algorithm and develop a new algorithm for grapheme-to-phoneme rule extraction (*Default&Refine*) that utilises the concept of a 'default phoneme' to create a cascade of increasingly specialised rules. This algorithm displays a number of attractive properties including rapid learning, language independence, good asymptotic accuracy, robustness to noise, and the production of a compact rule set. In order to have greater flexibility with regard to the various heuristic choices made during rewrite rule extraction, we define a new theoretical framework for analysing instance-based learning of rewrite rule sets. We define the concept of *minimal representation graphs*, and discuss the utility of these graphs in obtaining the smallest possible rule set describing a given set of discrete training data.

We develop an approach for the interactive creation of pronunciation models via bootstrapping, and implement this approach in a system that integrates various of the analysed grapheme-to-phoneme alignment and conversion algorithms. The focus of this work is on combining machine learning and human intervention in such a way as to minimise the amount of human effort required during bootstrapping, and a generic framework for the analysis of this process is defined. Practical tools that support the bootstrapping process are developed and the efficiency of the process is analysed from both a machine learning and a human factors perspective. We find that even linguistically untrained users can use the system to create electronic pronunciation dictionaries accurately, in a fraction of the time the traditional approach requires. We create new dictionaries in a number of languages (isiZulu, Afrikaans and Sepedi) and demonstrate the utility of these dictionaries by incorporating them in speech technology systems.

Keywords: bootstrapping, grapheme-to-phoneme conversion, grapheme-to-phoneme alignment, letter-to-sound, pronunciation modelling, pronunciation prediction, pronunciation rules, pronunciation dictionary, language technology resource development.

UITSPRAAKMODELLERING EN SELFSTEUN

Selfsteuntegnieke beloof om die ontwikkeling van taalhulpbronne vir tegnologiese toepassings te versnel. Hierdie belofte is veral belangrik in die onwikkelende wêreld, waar sulke hulpbronne skaars is, en beduidende taalverskeidenheid voorkom. In hierdie tesis ontleed ons die uitspraakvoorspellingstaak binne ’n selfsteunraamwerk, as ’n gevallestudie van selfsteunontwikkeling van taalhulpbronne.

Ons ontleed grafeem-na-foneemomskakeling, op soek na ’n algoritme wat vir selfsteundoeleindes gebruik kan word. Ons ondersoek verbeteringe aan die “Dinamiese Konteksuitbreiding” (DEC) algoritme, en ontwikkel ’n nuwe algoritme vir die onttrekking van grafeem-na-foneemreëls (*Verstek&Verfyn*) wat die begrip van ’n ‘verstekfoneem’ gebruik om ’n rits van toenemend afgestemde reëls te skep. Hierdie algoritme vertoon ’n aantal aantreklike eienskappe, insluitende kort leertye, taalonafhanklikheid, goeie uitloopakkuraatheid, ruisbestandheid, en die skep van klein reëlstelle. Om groter plooibaarheid in ’n aantal heuristiese keuses te verkry, stel ons ’n nuwe teoretiese raamwerk vir die ontleed van geval-gebasseerde leerprosesse van herskryfreëls voor. Ons stel die begrip van *kleinste voorstellende grafieke* voor, en bespreek die nut van sulke grafieke in die onttrek van die kleinste moontlike reëlstel wat gegewe leervoorbeelde beskryf.

Ons ontwikkel ’n benadering tot die wisselwerkende skep van uitspraakmodelle deur selfsteun, en verwerklik hierdie benadering in ’n stelsel wat verskeie van die ontlede algoritmes vir belyning en reëlonttrekking saamvat. Ons gee aandag aan die saamvoeg van masjienleer en menslike ingrype om die hoeveelheid menslike inset tydens selfsteun so klein moontlik te hou, en ontwikkel ’n algemene raamwerk vir die ontleding van hierdie proses. Verder ontwikkel ons praktiese gereedskap ter ondersteuning van selfsteun, en ontleed die doeltreffendheid daarvan uit die oogpunte van masjienleer en menslike bruikbaarheid. Ons bevind dat selfs gebruikers sonder taalkundige opleiding akkurate woordeboeke sodoende kan skep, in ’n breukdeel van die tyd wat die gebruiklike benadering vereis. Ons skep nuwe woordeboeke vir verskeie tale (isiZulu, Afrikaans en Sepedi), en toon die nuttigheid van hierdie woordeboeke in spraaktegnologietoepassings.

Sleutelterme: selfsteun, grafeem-na-foneem omsetting, grafeem-na-foneem belyning, letter-na-klank, uitspraakmodellering, uitspraakvoorspelling, uitspraakwoordeboek, uitspraakreëls, hulpbronontwikkeling vir taaltegnologie.

ACKNOWLEDGEMENTS

This research was performed in the Human Language Technologies (HLT) Research Group of the Meraka Institute. It was guided by Etienne Barnard, for the past three years my PhD advisor, colleague and the ideal co-explorer (a rare privilege!)

The HLT Research group grew in parallel with this thesis, and I am grateful to all the group members who assisted in one way or another: Louis Joubert, Francois Aucamp and others, who assisted with the development of ’System B’; Aby Louw who assisted in integrating some of the newly created dictionaries in Text-to-Speech systems; and the many other HLT researchers, developers and students who provided a supportive research environment.

Much of the collected data relied on the patience of the various dictionary developers. I am especially grateful to Nadia Barnard, who assisted with both kindness and skill.

I would also like to thank:

- Johan Eksteen, my manager at the CSIR at the time, who initially supported my decision to make the jump from project manager to researcher, and who has supported my work ever since.
- Liesbeth Botha, who guided my initial explorations in speech-related research.
- Rich Stern and the Carnegie Mellon Speech Group, who hosted me at Carnegie Mellon in Pittsburgh for a very enjoyable year.

Finally, I would like to thank my friends and family for putting up with me as a mostly absent, part-time PhD student; and of course, MC, without whose support this whole endeavour would have been quite impossible.

TABLE OF CONTENTS

CHAPTER ONE - INTRODUCTION 2

1.1 HLT in the developing world 2

1.2 Bootstrapping of HLT resources 3

1.3 Pronunciation modelling within a bootstrapping framework 4

1.4 Overview of thesis 4

CHAPTER TWO - BACKGROUND 6

2.1 Introduction 6

2.2 Pronunciation Modelling 6

2.2.1 Manual development of pronunciation models 7

2.2.1.1 Pronunciation dictionaries 7

2.2.1.2 Pronunciation rules 8

2.2.2 Data-driven approaches to g-to-p rule extraction 9

2.2.2.1 Neural networks and decision trees 9

2.2.2.2 Pronunciation by Analogy 10

2.2.2.3 Instance-based learning 10

2.2.2.4 Alternative approaches 12

2.2.3 Grapheme-to-phoneme alignment 12

2.2.4 Grapheme-based systems 13

2.3 Bootstrapping of HLT resources 14

2.4 The automated generation of pronunciation dictionaries 15

2.5 Conclusion 16

CHAPTER THREE - BOOTSTRAPPING MODEL 17

3.1 Introduction 17

3.2 Model description 17

3.2.1 Components 18

3.2.2 Process 19

3.2.3 Examples 19

3.3 Efficiency of bootstrapping process 20

3.3.1 Human factors . . . 22
3.3.2 Machine learning factors . . . 23
3.3.3 System analysis . . . 24
3.4 Bootstrapping pronunciation models . . . 25
3.4.1 Algorithmic requirements . . . 25
3.5 Conclusion . . . 25

CHAPTER FOUR - GRAPHEME-TO-PHONEME CONVERSION 26
4.1 Introduction . . . 26
4.2 Baseline algorithm . . . 26
4.3 Experimental data and approach . . . 27
4.4 Grapheme-to-phoneme alignment . . . 28
4.4.1 Pre-processing of graphemic nulls . . . 29
4.4.2 Utilising the phonemic character of null-phonemes . . . 29
4.5 DEC-based grapheme-to-phoneme prediction . . . 31
4.5.1 Standard DEC . . . 31
4.5.2 Shifting windows . . . 32
4.5.3 Rule pairs . . . 35
4.5.4 Conflict resolution . . . 36
4.5.5 Default rules . . . 36
4.6 A default-and-refinement approach to g-to-p prediction . . . 37
4.6.1 Asymptotic performance . . . 40
4.6.1.1 Regular spelling systems . . . 40
4.6.1.2 Less regular spelling systems . . . 41
4.6.2 Learning efficiency . . . 42
4.6.3 Size of the rule set . . . 43
4.6.4 Continuous learning . . . 43
4.7 Bootstrapping analysis . . . 47
4.7.1 Predictive ability . . . 47
4.7.2 Conversion accuracy . . . 48
4.7.3 Computational cost . . . 49
4.7.4 Robustness to noise . . . 52
4.8 Conclusion . . . 54

CHAPTER FIVE - MINIMAL REPRESENTATION GRAPHS 55
5.1 Introduction . . . 55
5.2 Conceptual approach . . . 55
5.3 Theoretical framework . . . 70

5.3.1 Rule format 70
5.3.2 Rule set analysis 73
5.3.2.1 Training data, word patterns and sub-patterns 73
5.3.2.2 Conflict rules and conflict resolution 75
5.3.2.3 Complete, accurate, minimal and possibly_minimal rule sets 77
5.3.2.4 Allowed states and allowed operations 78
5.3.2.5 Matchwords, possible_words, rulewords and shared_words 80
5.3.2.6 Complementing rules: containpat, mincomp and supercomp 83
5.3.2.7 Z_m as a subset of $Z_{combined}$ 86
5.3.3 Rule ordering 87
5.3.4 Characteristics of an allowed state 90
5.3.5 Initial allowed state 92
5.3.6 Allowed operations 92
5.3.6.1 Decreasing rule set size 93
5.3.6.2 Removing unnecessary edges 93
5.3.6.3 Identifying required rules 94
5.3.6.4 Resolving conflict rules 94
5.3.7 Breaking ties 95
5.3.8 Optimising generalisation ability 95
5.4 Alternative algorithms as specialisation of general framework 96
5.5 Extensions 96
5.6 Conclusion 97

CHAPTER SIX - BOOTSTRAPPING PRONUNCIATION MODELS 98
6.1 Introduction 98
6.2 Bootstrapping system 98
6.2.1 User perspective 99
6.2.2 System perspective 100
6.2.3 Algorithmic choices 101
6.2.4 System configuration 102
6.3 Experiment A: Validation of concept 102
6.3.1 Experimental protocol 103
6.3.2 Human factors 104
6.3.2.1 User learning curve 104
6.3.2.2 Effect of linguistic expertise 105
6.3.2.3 The cost of using audio assistance 106
6.3.2.4 The cost of phoneme corrections 107
6.3.2.5 Related factors 107

6.3.3 Machine learning factors . . . 108
6.3.3.1 System continuity . . . 108
6.3.3.2 Predictive accuracy . . . 108
6.3.3.3 Validity of base data . . . 109
6.3.4 System analysis . . . 109
6.4 Experiment B: Semi-automatic detection of verifier errors . . . 110
6.5 Experiment C: Building a medium-sized dictionary . . . 112
6.5.1 Experimental protocol . . . 112
6.5.2 Human factors analysis . . . 112
6.5.3 Analysis of machine learning factors . . . 114
6.5.4 System analysis . . . 115
6.6 Building systems that utilise bootstrapped dictionaries . . . 118
6.6.1 isiZulu Text-to-Speech . . . 118
6.6.2 Sepedi Speech Recognition . . . 118
6.6.3 Afrikaans Text-to-Speech . . . 118
6.6.4 Other systems . . . 119
6.7 Conclusion . . . 120

CHAPTER SEVEN - CONCLUSION 121
7.1 Introduction . . . 121
7.2 Summary of contribution . . . 121
7.3 Further application and future work . . . 122
7.4 Conclusion . . . 124

APPENDIX A - THE ARPABET PHONE SET 125

APPENDIX B - SOME THEOREMS REGARDING MINIMAL REPRESENTATION GRAPHS 126
B.1 Word sets . . . 126
B.2 Characteristics of Z_m . . . 129
B.3 Z_m as a subset of $Z_{combined}$ . . . 131
B.4 Rule ordering in Z_m . . . 134
B.5 Rule ordering in Z_m as a subset of $Z_{combined}$ . . . 137
B.6 Characteristics of an allowed state . . . 139
B.7 Initial allowed state . . . 141

REFERENCES 144

CHAPTER ONE

INTRODUCTION

1.1 HLT IN THE DEVELOPING WORLD

Human language technologies (HLT) hold much promise for the developing world, especially for user communities that have a low literacy rate, speak a minority language, or reside in areas where access to conventional information infrastructure is limited. For example, information systems that provide speech-enabled services via a telephone can serve a user in his or her language of choice in a remote location, without requiring additional expertise from the user, or a sophisticated Internet infrastructure. In South Africa, while Internet penetration is still low, more than 90% of the population has access to a telephone; and the potential of voice services for improving access to information is receiving increasing attention [1].

The development of various forms of human language technology - such as speech recognition, speech synthesis or multilingual information retrieval systems - requires the availability of extensive language resources. The development of these resources involves significant effort, and can be a prohibitively expensive task when such technologies are developed for a new language. The development of an accurate automatic speech recognition system, for example, requires access to an electronic pronunciation dictionary, a large annotated speech corpus from various speakers, and an extensive textual corpus. Such resources are freely available for only a small subset of the world's languages. This presents a significant obstacle to the development of HLT in the developing world, where few electronic resources are available for local languages, skilled computational linguists are scarce, and linguistic diversity is high. (India, for example, recognises nineteen official languages and South Africa eleven. In countries such as Indonesia and Nigeria, several hundred languages are widely spoken [2].)

In order to realise the potential benefit of HLT in the developing world, the language resource barrier must first be overcome: techniques are required that support the fast and cost-effective development of language resources in a new language, without extensive reliance on assistance from skilled computational linguists or access to prior language resources. Techniques that can support or accelerate the language resource development effort include the cross-language re-use of information [3, 4], statistical approaches to automated resource generation [5, 6], and bootstrapping, the focus of this thesis.

1.2 BOOTSTRAPPING OF HLT RESOURCES

The popular saying "to pull oneself up by one's bootstraps" is typically used to describe the process of "improving one's position by one's own efforts" [7]. In computer terminology this term was originally used to describe the process of iteratively loading a computer operating system from a few initial instructions, but soon came to describe any process where "a simple system activates a more complicated system" [8]. We use the term to describe an iterative process whereby a model of some form is improved via a controlled series of increments, at each stage utilising the previous model to generate the next one[1].

This generic technique has been applied successfully to the language resource development problem previously, especially in the creation of automatic speech recognition systems [3, 9–11]. When acoustic models are developed for a new target language, an automatic speech recognition system can be initialised with pre-developed models from an acoustically similar source language, and these initial models improved through an iterative process whereby audio data in the target language is automatically segmented using the current set of acoustic models, the models retrained and the target data re-segmented via a set of incremental updates. The potential saving in resource requirements achieved through such a process was well demonstrated by Schultz and Waibel [12], among others.

When considering resource bootstrapping approaches in more detail (as discussed in Chapter 3) it becomes apparent that these approaches rely on an automated mechanism that converts between various representations of the data considered. Each representation provides some specific advantage – making the data more amenable to a particular form of analysis – which can be utilised in improving or increasing the resource itself. In the above example, two representations are utilised: annotated audio data and acoustic models; and the mechanisms to move from one representation to the other are well defined through the phone alignment and acoustic modelling tasks, respectively.

The bootstrapping process has been applied successfully to a variety of additional language resource development tasks, including the development of parallel corpora [13], morphological dictionaries [14], morphological analysers [15] and linguistically tagged corpora [16]. We are specifically interested in the use of bootstrapping for the development of pronunciation models in new languages.

[1]The term 'bootstrapping' is discussed in more detail in Section 2.3.

1.3 PRONUNCIATION MODELLING WITHIN A BOOTSTRAPPING FRAMEWORK

A pronunciation model for a specific language describes the process of letter-to-sound conversion: given the orthography of a word, it provides a prediction of the phonemic realisation of that word. This is a component required by many speech processing tasks – including general domain speech synthesis and large vocabulary speech recognition – and is often one of the first resources required when developing speech technology in a new language.

The letter-to-sound relationship is typically modelled through explicit pronunciation dictionaries [17–19], but can also be represented according to various abstract letter-to-sound formalisms. Grapheme-to-phoneme (g-to-p) rule sets can either be hand-crafted, or a letter-to-sound representation can be obtained from a given training dictionary, using approaches such as neural networks, instance-based learning, decision trees, or pronunciation by analogy models [20–25]. In effect, these data-driven letter-to-sound formalisms provide a second representation of the training dictionary, by converting the training dictionary to a set of base elements and operators of some form, which we will refer to in general as g-to-p rule sets. These letter-to-sound formalisms have been studied over the past twenty years (as discussed in more detail in Section 2.2), resulting in a number of efficient representation techniques. Since efficient techniques exist to analyse the same pronunciation data according to more than one representation, it should be possible to utilise these representations during bootstrapping.

A letter-to-sound conversion mechanism is valuable, not only in the absence of explicit pronunciation dictionaries, but also in order to accommodate speech technology in memory constrained environments, or to deal with out-of-vocabulary words in speech systems. Such applications require a balance between the need for small rule sets, fast computation and optimal accuracy, and various approaches to pronunciation modelling have been defined to meet these requirements. Bootstrapping introduces an additional requirement: the ability to obtain a high level of generalisation given a very small training set. If such a g-to-p mechanism can be obtained, it seems probable that a bootstrapping approach will be beneficial in improving the speed and accuracy with which pronunciation models can be developed in a new language.

1.4 OVERVIEW OF THESIS

The aim of this thesis is to analyse the pronunciation modelling task within a bootstrapping framework. The goals are two-fold: (a) to obtain a mechanism for pronunciation modelling that is well suited to bootstrapping; and (b) to analyse the bootstrapping of pronunciation models from a theoretical and a practical perspective, as a case study in the bootstrapping of HLT resources.

The thesis is structured as follows:

- In Chapter 2 we provide background information with regard to the pronunciation modelling task and the use of bootstrapping for the development of HLT resources in general.
- In Chapter 3 we sketch a framework for analysing the bootstrapping process. This framework provides the context for subsequent chapters, and describes the requirements for a conversion algorithm suitable to bootstrapping.
- In Chapter 4 we analyse the grapheme-to-phoneme conversion task in the search for an appropriate conversion algorithm. This leads to the definition of *Default & Refine*, a novel algorithm for grapheme-to-phoneme rule extraction that is well suited to bootstrapping.
- In Chapter 5 we utilise the characteristics of the pronunciation modelling task analysed in the prior chapter in order to define a new framework for grapheme-to-phoneme prediction. We define the concept of *minimal representation graphs*, and demonstrate the utility of these graphs in obtaining a minimal rule set describing a given set of training data.
- In Chapter 6 we apply the new grapheme-to-phoneme algorithms in the bootstrapping of pronunciation models. We experiment with a number of options, and analyse the efficiency of this process according to the framework defined in Chapter 3. We develop bootstrapped pronunciation models in three languages (isiZulu, Afrikaans and Sepedi) and integrate the bootstrapped dictionaries in speech technology systems.
- In Chapter 7 we summarise the contribution of this thesis, and discuss further applications and future work.

CHAPTER TWO

BACKGROUND

2.1 INTRODUCTION

This chapter provides background information with regard to the main topics discussed in subsequent chapters:

- Section 2.2 provides an overview of various approaches to pronunciation modelling;
- Section 2.3 describes the use of bootstrapping for the development of HLT resources in general; and
- Section 2.4 discusses current approaches to the creation of pronunciation dictionaries in a semi-automated fashion.

In this chapter, as in the remainder of this thesis, we use the ARPAbet symbol set (included in Appendix A) to demonstrate phonemic concepts.

2.2 PRONUNCIATION MODELLING

A pronunciation model for a specific language provides an accurate mechanism for letter-to-sound conversion, also referred to as grapheme-to-phoneme (g-to-p) conversion. Given the orthography of a word, grapheme-to-phoneme conversion provides a prediction of the phonemic realisation of that word. Where additional pronunciation characteristics such as stress or tone are predicted, this process is referred to as grapheme-to-phoneme conversion with stress and/or tone assignment. This can be the first of a two-phase process in pronunciation prediction: the first task being grapheme-to-phoneme conversion, the second phoneme-to-allophone conversion. The rules utilised in the latter phase are typically referred to as phonological rules, and are not always required explicitly, depending on the specific type of speech technology that will be utilising the dictionary. For example, a speech recognition system may either model phonological effects explicitly, or utilise a phonemic lexicon and rely on the context-dependent acoustic models to capture many of the phonological effects [26]. As the distinction between phonemes and phones is often blurred, we approach this differentiation in a pragmatic fashion in this thesis.

Pronunciations can be idiosyncratic, and not all pronunciation phenomena are regular to the extent of being predictable. Also, letter-to-sound conversion does not only depend on orthography: the phonemic outcome can (and does) depend on other linguistic features such as word part-of-speech, word morphology or word etymology. From a bootstrapping perspective, we are interested in approaches to the pronunciation prediction problem where additional linguistic resources are not available (or can be bootstrapped easily), and therefore we focus our attention on grapheme-to-phoneme conversion based mainly on orthography.

The remainder of this section provides an overview of current approaches to pronunciation modelling: Section 2.2.1 describes the manual development of pronunciation models, both the development of explicit pronunciation dictionaries and the handcrafting of grapheme-to-phoneme conversion rules, and Section 2.2.2 provides an overview of different approaches to the data-driven extraction of grapheme-to-phoneme conversion rules. As many of the data-driven approaches require grapheme-to-phoneme alignment prior to grapheme-to-phoneme rule extraction, approaches to grapheme-to-phoneme alignment are discussed separately in Section 2.2.3. Section 2.2.4 discusses an alternative speech processing approach that circumvents the need for explicit pronunciation modelling.

2.2.1 MANUAL DEVELOPMENT OF PRONUNCIATION MODELS

2.2.1.1 PRONUNCIATION DICTIONARIES

Many electronic pronunciation dictionaries (such as NETtalk [20] or OALD [18]) were created as digital versions of similar printed dictionaries. Classical printed pronunciation dictionaries typically only list word base forms, and for each word base form its ‘standard’ pronunciation. Pronunciation variants are only included when more than one distinct pronunciation exists for a single word (e.g. the past tense and present tense variants of the English word ‘read’: *r iy d* and *r eh d*). Electronic dictionaries that are frequently utilised in speech applications (such as CMUdict [17]) soon grow to include additional word forms (plurals and other derivatives), and multiple pronunciation variants, as required by the applications utilising the dictionary. Pronunciation variants can be generated automatically using phonological rule sets[1] or added according to a manual process.

Task-designed electronic pronunciation dictionaries, such as *FONILEX*, developed by Mertens and Vercammen [19], include systematic mechanisms to derive word variants from base forms. *FONILEX* specifically is a full-form lexicon (it lists the various word base forms separately) and

[1]The automatic extraction of phonological rules utilise techniques similar to those applied during grapheme-to-phoneme rule extraction, as described in Section 2.2.2.

provides an ‘abstract’ representation of each word, as well as three ‘concrete’ pronunciations representing three different speaking styles. The concrete pronunciations are derived automatically from the abstract representation via a set of phonological rewrite rules. In this way, regular variants are captured via phonological rules, rather than additional dictionary entries. Irregular variants are included as additional dictionary entries. A related approach, followed independently by Allen *et al* [27] and Coker *et al* [28], utilises morphemes as the stored unit, and obtains dictionary entries by combining these morphemes using a set of morphological rules. Here morphological rules are used to generate the word base form itself, which is not stored individually. It is interesting to note that *FONILEX* was compiled semi-automatically using grapheme-to-phoneme conversion, and verified manually – an approach that is related to the bootstrapping process investigated in this thesis.

2.2.1.2 PRONUNCIATION RULES

Manual pronunciation rules are typically developed according to the two-stage process described in Section 2.2; that is, two rule sets are created: one set of grapheme-to-phoneme rules, and a second set of phonological rules that generate the appropriate allophone (or allophones) per phoneme. Both rule sets are often augmented by a set of exceptions. These rule sets can be described according to different formalisms, a general formalism for a multi-level rewrite rule being:

$$\{a\}^*g\{b\}^* \rightarrow \{c\}^*p\{d\}^* \tag{2.1}$$

which, more typically, is simplified as: $\{a\}^*g\{b\}^* \rightarrow p$, where g indicates the grapheme being considered and p the specific phonemic realisation of g. $\{a\}^*$ and $\{b\}^*$ represent zero or more contextual elements to the left and the right of the grapheme (respectively) of words that this rule can be applied to, and $\{c\}^*$ and $\{d\}^*$ indicate how the word is amended (or not) during the application of this specific rule. Depending on the exact formalism, the left and/or right contexts of the left-hand side can either consist of graphemes only, or a combination of graphemes and phonemes, and similarly, the right-hand side can either be defined in terms of phonemes only, or a combination of graphemes and phonemes. A null (or empty) phoneme or grapheme may be utilised explicitly within the formalism. Furthermore, a single contextual element can also be used to represent a class of such graphemes or phonemes. Formalisms differ based on the order in which rules are applied, the direction in which rules are parsed, and whether a single rule or a sequence of all matching rules are applied when predicting a single word. Manually developed rewrite rules exist for a number of languages, including languages as diverse as English [29], Arabic [30] and isiXhosa [31].

Typically, the more modern the writing system of a language, the stronger the connection between the spoken and written form of a language, and the more regular the spelling system of the language[2]. Languages with a fairly recent spelling system (such as Swahili) have an almost direct correspondence between the orthography and the pronunciation of a word, while a language such as English or French

[2]As discussed further in Section 4.6.

includes significant historical ‘baggage’ in its spelling system. For languages with highly regular spelling systems, the manual development of a set of pronunciation rules can be a manageable task for a skilled linguist. For languages with less regular spelling systems this task becomes particularly arduous, with the set of words that that can be predicted correctly using the manually developed rule set only achieving larger sizes if amended by a sizeable exceptions dictionary. For example, the rule set developed by Elovitz *et al* [29], consisting of 329 rules for English, achieved only 25.7% word accuracy when evaluated by Damper *et al* [25] and 19.3% word accuracy when a modified version was evaluated by Bagshaw [32] (using different corpora). Semi-manually developed finite state transducer systems can achieve better performance [26], but require significant expertise to develop.

2.2.2 DATA-DRIVEN APPROACHES TO G-TO-P RULE EXTRACTION

Data-driven approaches to grapheme-to-phoneme rule extraction can be used to generalise from existing pronunciation dictionaries when handling out-of-vocabulary words in speech systems, and to compress information when requiring a pronunciation model in a memory-constrained environment. Such applications require a balance between the need for small rule sets, fast computation and optimal accuracy, and various approaches to pronunciation modelling have been defined to meet these requirements. Approaches include the application of neural networks [20, 33], decision trees [22–24, 34], Pronunciation by Analogy (PbA) models [32,35–38], instance-based learning algorithms such as Dynamically Expanding Context (DEC) [21,36] and IB1-IG [24], finite state transducers [39], Bayesian networks [40], and the combination of methods and additional information sources through meta-classifiers [41]. Many of these algorithms require grapheme-to-phoneme alignment prior to rule extraction, as discussed in Section 2.2.3.

Benchmarking these pronunciation prediction algorithms is difficult: There are few standardised pronunciation prediction tasks that are widely used, and the task itself is very sensitive to training/test set distributions. A strict evaluation of three of the data-driven approaches (a neural network, IB1-IG and PbA) can be found in [25]. Results obtained when applying different algorithms are discussed in further detail in Section 4.6.1; the remainder of this section provides an overview of the various approaches to grapheme-to-phoneme rule extraction mentioned above.

The automatic extraction of phonological rules utilise similiar techniques as those described here. Such rule sets are used to generate an allophonic representation for a phonemic pronunciation, as demonstrated by Ellison [42], Tajchman *et al* [43] and others, or to assign additional pronunciation characteristics such as stress to the pronunciation of the word [44]. The application of data driven techniques for the development of phonological rule sets is not discussed further: we rather focus our attention on the grapheme-to-phoneme conversion process specifically.

2.2.2.1 NEURAL NETWORKS AND DECISION TREES

A neural network was one of the first data-driven approaches to grapheme-to-phoneme rule set extraction demonstrated. A neural network was trained by Sejnowski and Rosenberg [20] using the

English NETtalk corpus, and later re-implemented by McCulloch and others as the NETspeak [33] system. Words were windowed with a fixed number of graphemes (between 3 and 11 graphemes) per window, and a feed-forward neural network was trained to associate each letter, surrounded by its graphemic window, with a specific phoneme outcome. A similar system was later evaluated by Damper *et al* [25].

Various decision tree based approaches have been demonstrated, including systems developed by Andersen *et al* [22, 45], Black *et al* [23] and Hakkinen *et al* [34], obtaining comparable results. The detail implementations differed based on various aspects, including the type of questions generated, the pruning method, the splitting criteria and detailed parameter choices. The algorithms were applied to different languages and corpora, and different evaluation processes used. Andersen *et al* compared a binary decision tree with Trie structures using both an English (NETtalk and CMUdict) and a French (ONOMASTICA) database [22]. Black *et al* utilised Classification and Regression Trees (CART) and English (OALD and CMUdict), French (BRULEX) and German (CELEX) dictionaries [23]. Hakkinen *et al* explicitly compared the performance of neural networks and decision trees for the English CMUdict task. Hakkinen *et al* found that neural networks provide better generalisation than decision trees when limited training data is available, and perform more consistently across mismatched test sets, while decision trees typically outperform neural networks where training and test data are closely matched [34].

2.2.2.2 PRONUNCIATION BY ANALOGY

Pronunciation by Analogy (PbA) models predict the pronunciation of a new word by searching through known words for matching sub-word parts. This set of algorithms was designed specifically for the task of grapheme-to-phoneme prediction. Originally suggested by Dedina and Nusbaum [46], the approach was further developed by Sullivan and Damper [35], Yvon [36, 47], Damper and Eastmond [37], Bagshaw [32], and Marchand and Damper [38].

Languages with irregular spelling systems such as English and French perform well within analogy-based frameworks, and for English, the best asymptotic results to date have been achieved with PbA [25]. Unfortunately, current versions of these algorithms can be ‘slow learners’, only approaching asymptotic accuracy for larger training dictionary sizes, as discussed further in Section 4.2. Depending on the amount of prior manipulation of the training data employed by PbA algorithms, these algorithms can be seen as a form of instance-based learning.

2.2.2.3 INSTANCE-BASED LEARNING

We use the term *instance-based learning* as used by Aha *et al* [48] to describe algorithms that generate classification predictions using specific instances from a set of training data, rather than using a generalised abstraction created from the training set, and do not differentiate among instance-based learning, memory-based learning or case-based reasoning. These algorithms all utilise ‘lazy learning’: rather than generalising from a training set, the entire training set is typically retained (in some form or another) and predictions are based on reasoning about these retained exemplars, analogous to the process of nearest neighbour classification[3].

In [24], Daelemans *et al* provide a strong argument for the utility of memory-based approaches for language processing tasks, noting that in many of these tasks exceptions tend to occur in 'groups or pockets in instance space'. As it is difficult to differentiate between actual noise inherent to language data[4] and small regular families of exceptions (that provide useful predictive information), Daelemans argues that exceptions should preferably be retained, as is inherent to standard instance-based learning. Two specific approaches that have been applied successfully to grapheme-to-phoneme conversion are (1) variations of *IB1-IG* [49], as developed and applied to the grapheme-to-phoneme task by Daelemans *et al* [24]; and (2) Kohonen's Dynamically Expanding Context (DEC) [50], initially applied by Torkkola [21] to the grapheme-to-phoneme task:

1. **IB1-IG**

   IB1-IG [24, 49] is in essence a k-nearest neighbour classifier that utilises as distance measure a weighted version of graphemic context overlap. Appropriate weighting of the graphemic context is an important aspect of the algorithm, and is attained through information gain techniques. Given a grapheme-to-phoneme aligned training dictionary, words are windowed, and a learning instance is generated per window (each instance focussing on a specific letter within the context of the rest of the window) and associated with a specific phonemic classification of that letter. Weights are associated with each feature based on a normalised measure of the amount of information the specific feature contributes to knowledge about the specific phonemic class (over the entire instance base). New words are predicted by finding the instances that are closest to the target word, using the weighted distance measure. Ties are resolved by considering frequency of outcome, and frequency of occurrence of the specific feature (where a feature defines both a letter and its position).

   Daelemans *et al* [24] evaluated this algorithm on the task of grapheme-to-phoneme conversion with stress assignment, using the CELEX database (as one of a set of language learning tasks considered), and found comparable accuracy rates between IB1-IG and a decision tree approach. The IB1-IG algorithm performed better than the C5.0 decision tree used for comparison: the difference in performance was slight (but significant) if the number of instances required for a decision tree node to be retained was chosen as 1 (similar to the IB1-IG approach); a larger number of required instances caused greater pruning of the decision tree, and decreased its performance. Damper *et al* [25] found that IB1-IG obtained higher accuracy than a neural network, but not the same level of asymptotic accuracy as PbA. Further results are provided by Hoste *et al* [41] in an evaluation of meta-classification techniques.

[3]It should be noted that the differentiation among techniques described in this section is not strict: for example, a decision tree learning algorithm that does not allow any pruning can also be seen as a form of instance-based learning.

[4]For example, as caused by true exceptions, or discrepancies in the way in which the lexicon was developed.

2. **DEC**

   Kohonen's Dynamically Expanding Context (DEC) [50], initially applied by Torkkola to the grapheme-to-phoneme problem [21], is another instance-based learning algorithm that predicts phoneme realisation based solely on graphemic context. In DEC, each rule specifies a mapping of a single grapheme to a single phoneme for a given left and right graphemic context, i.e is of the form: *(left-context,grapheme,right-context)* $\rightarrow$ *phoneme*.

   Rules are extracted by finding the smallest context that provides a unique mapping of grapheme to phoneme. If an $n-$letter context is not sufficient, the context is expanded to either the right or the left. This 'specificity order' influences the performance of the algorithm. The set of extracted rules are stored as a hierarchical tree, with more general rules at the root, and more specific rules at the leaves. The tree is traversed from the root to the leaves, and the rule at the first matching leaf (the rule describing the largest matching context) is used to predict the specific grapheme-to-phoneme realisation. If no leaf is matched, the most probable outcome of the last matching leaf is used, as can be estimated from the training data. If the extracted 'rule set' is allowed to contain contexts of an arbitrary size, no training words are discarded, and the tree structure is simply used to arrange the set of all training instances in an efficient structure.

2.2.2.4 ALTERNATIVE APPROACHES

A number of further approaches to pronuncation modelling exist, including:

1. Finite state transduction, as demonstrated by Luk and Damper [39], and more recently by Hazen *et al* [26]. Finite state transduction as used in [26] requires significant linguistic specification, while Luk and Damper's approach requires less linguistic input but makes a number of (restrictive) assumptions in order to create a trainable system.

2. The application of Bayesian networks for grapheme-to-phoneme conversion [40]. Bayesian networks are more typically used for pronunciation variation modelling, rather than phonemic base form generation.

3. The use of hierarchical systems of meta-classifiers, and even meta-meta-classifiers as investigated by Hoste *et al* [41].

Various of these approaches can be utilised during bootstrapping, as discussed further in Section 4.2.

2.2.3 GRAPHEME-TO-PHONEME ALIGNMENT

The majority of data-driven approaches to grapheme-to-phoneme rule extraction first require that the training dictionary be aligned on a grapheme-to-phoneme basis. For languages with alphabetic writing systems[5], each grapheme is mapped to its corresponding phoneme, and phonemic or graphemic

[5]For ideographic, pictographic, syllabic or even moraic languages, a more complex process is required – see for example [51] for a comparison of alignment approaches for Japanese.

nulls inserted where required: A phonemic null is inserted where a single phoneme is produced from more than one grapheme; a graphemic null where a single grapheme results in more than one phoneme. In languages where graphemic nulls are rare, graphemic exceptions that can map to more that one phoneme (such as $x \rightarrow k\ s$) can be replaced with two pseudo-graphemes (e.g. replacing x with $x\ X$) and only phonemic nulls inserted. This technique, suggested by Pagel *et al* [52], results in fewer alignment errors.

Initial data sets used for grapheme-to-phoneme benchmarking (such as *NETtalk* [20]) were hand aligned. Dalsgaard, Andersen and others [53, 54] applied forced Viterbi alignment [55] to create automatic grapheme-to-phoneme alignments, based on the probabilities *P(grapheme i | phoneme j)*. Initial probabilities were obtained from words and pronunciations that have equal length. This approach provides fairly accurate alignments: when benchmarked against the *NETtalk* hand alignments, Andersen *et al* achieved a word alignment accuracy of 83.7% and a phoneme alignment accuracy of 93.2% [22]. It should be noted that the *NETtalk* hand alignments may not be the ideal benchmark to use for measuring alignment accuracy, as discussed in more detail in Section 4.4. Black *et al* [23] used a similar alignment approach but defined a candidate set to restrict misalignments. In Black's approach the possible grapheme-to-phoneme mappings are specified prior to alignment, and used to restrict the alignment options during Viterbi alignment.

2.2.4 GRAPHEME-BASED SYSTEMS

The discussion up to this point has assumed that a pronunciation model is a required component for a variety of speech processing systems, including automatic speech recognition systems. Schillo *et al* [56] demonstrated an alternative approach by introducing the concept of grapheme-based speech recognition: rather than using a pronunciation dictionary, graphemes are used directly as basis for the acoustic sub-units modelled. This grapheme-based approach results in surprisingly accurate systems. Since the perplexity of the language model has a significant effect on the accuracy of the system, a strong language model compensates well for an inaccurate pronunciation model. The results obtained by Schillo *et al* were independently confirmed by Kanthak and Ney [57] and Killer [58]. While grapheme-based systems are conceptually less complete than system that incorporate an explicit pronunciation dictionary, grapheme-based systems for languages with fairly regular spelling systems (such as Italian, Spanish or Dutch) do not seem to be significantly less accurate than phoneme-based systems, especially in the presence of a strong language model, exhibiting a less than 2% relative decrease in accuracy in [57]. For languages with less regular systems, the decrease in accuracy becomes more noticeable: In [57] a 25.7% relative decrease in accuracy was observed for an English system with a word trigram perplexity of 124.5.

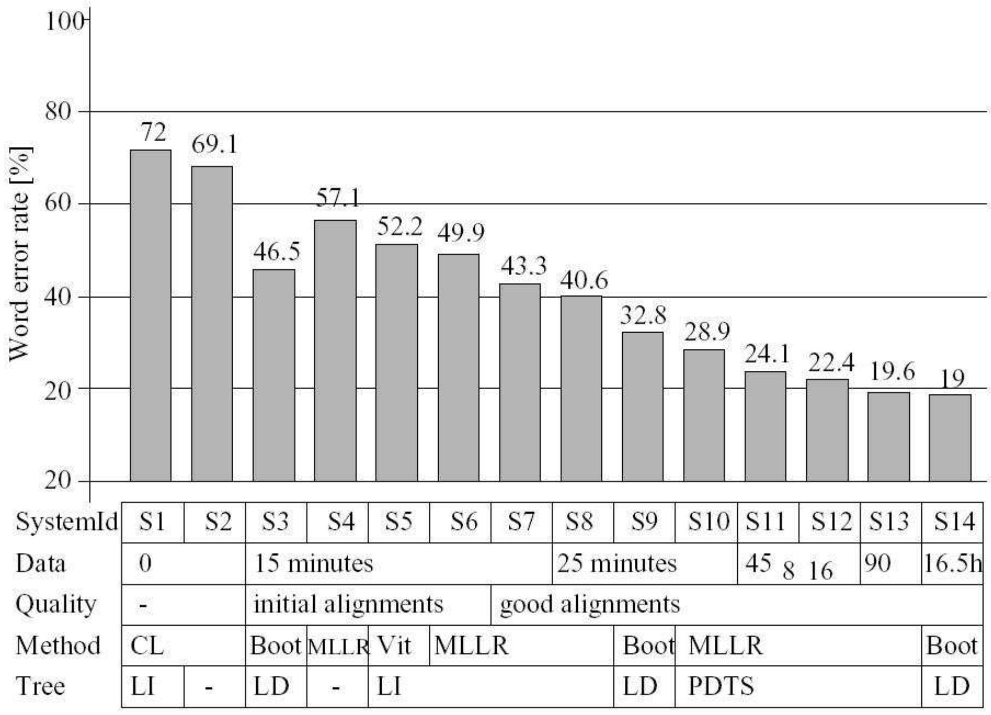
2.3 BOOTSTRAPPING OF HLT RESOURCES

We use the term ‘bootstrapping’ to describe an iterative process whereby a model of some form is improved via a controlled series of increments, at each stage utilising the previous model to generate the next one. This is a broader definition than often employed in machine learning, where bootstrapping typically indicates a semi-supervised approach to learning, where a small set of labelled instances is used to seed a classifier, label unclassified data, and retrain the classifier [59, 60]. Both the above interpretations should not be confused with the use of this term in the field of Statistics, where it can also indicate a statistical method for estimating the sampling distribution of an estimator by resampling with replacement from the original sample [61].

Bootstrapping can be a useful technique during language resource development, and has been used extensively in the creation of resources required by automatic speech recognition systems [3, 9–11]. In speech recognition, a bootstrapping technique is often combined with some form of cross-language information sharing. For example, when acoustic models are developed for a new target language, an automatic speech recognition system can be initialised with pre-developed models from an acoustically similar source language, and these initial models improved through an iterative process whereby audio data in the target language is automatically segmented using the current set of acoustic models, the models retrained and the target data re-segmented via a set of incremental updates.

The potential saving in resource requirements achieved through such a process was well demonstrated by Schultz and Waibel [12]. For example, in a set of experiments conducted on a Portuguese system, Schultz and Waibel obtained near-equal performance using either a fairly large amount (16.5 hours) of target data, or adapting multilingual models through a combination of bootstrapping and adaptation, using 90 minutes of target data. The increase in performance using different techniques is illustrated in Figure 2.1. Here, *Data* refers to the amount of target language data used and *Quality* refers to the quality of the alignments: *initial* alignments are generated by the multilingual system, while *good* alignments are updated based on improving systems. *Method* refers to the adaptation method used: using the unadapted initial system in a cross-language transfer approach (CL), Viterbi training using the alignments from the initial system (Vit), Maximum Likelihood Linear Regression adaptation of the initial system using the target data (MLLR), or bootstrapping (Boot). Bootstrapping consists of the following phases per bootstrapping cycle: creating initial alignments, Viterbi training, model clustering, retraining and writing improved alignments. *Tree* refers to the decision tree used for clustering: the original multi-lingual language independent tree (LI), a Portuguese language dependent tree (LD) or a tree built using the Polyphone Decision Tree Specialisation (PDTS) process[6] [12]. This example illustrates both the cross-language re-use of information – seeding the acoustic models using a related language – and the essence of a bootstrapping approach: iteratively improving acous-

[6]A standard context modelling technique is to cluster models using a CART-based clustering technique and a splitting criterion based on maximum entropy gain. The Polyphone Decision Tree Specialisation (PDTS) technique was proposed by Schultz as a mechanism to adapt the context modelling based on the target data, by restarting the decision tree growing process according to the target data available, resulting in significant improvements [62].



SystemId	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14
Data	0		15 minutes					25 minutes			45	8 16	90	16.5h
Quality	-		initial alignments				good alignments							
Method	CL		Boot	MLLR	Vit	MLLR			Boot	MLLR				Boot
Tree	LI	-	LD	-	LI				LD	PDTS				LD

Figure 2.1: *Experimental results when applying cross-language re-use of acoustic information techniques in the bootstrapping of a Portuguese system, from [12].*

tic models by utilising the models developed during the previous bootstrapping cycle to re-align the data, and retrain the models.

Additional language resource development tasks that have been shown to benefit from some form of bootstrapping include the development of parallel corpora [13], morphological dictionaries [14], text categorization [63], automatic audio alignments [64], grammar parsers [65], morphological analysers [15], linguistically tagged corpora [16], and the development of pronunciation lexicons, as discussed in Section 2.4.

2.4 THE AUTOMATED GENERATION OF PRONUNCIATION DICTIONARIES

In this section, we consider automated and semi-automated approaches to the generation of pronunciation dictionaries in a new language, referring to two types of approaches: Stuker [66] investigated ways in which existing phoneme recognisers can be used to generate a pronunciation dictionary for a new language, utilising audio data and word-level transcriptions in the target language. Using nine mono-lingual and a multi-lingual phoneme recogniser, phoneme recognition of the audio data is performed, and different voting and normalisation techniques are used to obtain a hypothesized pronunciation (or pronunciations) per words. This technique does not currently result in usable dictionaries, but further work is in progress.

A demonstrably successful approach to the semi-automated generation of pronunciation dictionaries, is the use of bootstrapping within the Festival Text-to-Speech System [67]. This system includes a rule extraction component based on Classification and Regression Trees, which can be used to generate letter-to-sound rules from a small lexicon. This lexicon is then grown iteratively by submitting additional words to the system, and having a human verify the correctness of the predictions. This process was recently demonstrated by Maskey *et al* [68], utilising an approach that is analogous to the approach used in this thesis. Maskey *et al* developed a Nepali pronunciation dictionary by iteratively extracting a grapheme-to-phoneme rule set, predicting a set of additional dictionary entries (varying from 100 words per cycle initially to 5000 words per cycle later in the process), identifying a subset of these words based on a calculated confidence score, and having these corrected by a Nepali speaker. In a related approach, the *FONILEX* dictionary was compiled semi-automatically using grapheme-to-phoneme conversion, and verified manually [19].

2.5 CONCLUSION

This chapter provided background on the pronunciation modelling task, and described various approaches to pronunciation modelling, focussing on data-driven techniques. The pronunciation modelling topic is addressed further in Chapter 4, where we define a grapheme-to-phoneme rule extraction mechanism suitable to bootstrapping. The current chapter also provided a brief overview of prior work related to the bootstrapping of HLT resources; this discussion continues in Chapter 3 with the definition of a general model for the bootstrapping of HLT resources.

CHAPTER THREE

BOOTSTRAPPING MODEL

3.1 INTRODUCTION

In this chapter we sketch a basic framework for the analysis of the bootstrapping process. We describe the bootstrapping model in Section 3.2, and discuss the factors to consider when evaluating the efficiency of the bootstrapping process in Section 3.3. In Section 3.4 we show how this model applies to the pronunciation modelling task in particular.

3.2 MODEL DESCRIPTION

As introduced in Section 2.3, we use the term ‘bootstrapping’ to describe *an iterative process whereby a model is improved via a controlled series of increments, at each stage utilising the previous model to generate the next one*. During bootstrapping the model is grown systematically, becoming increasingly accurate from one increment to the next. When analysing the bootstrapping process, it soon becomes apparent that the process relies on an automated or semi-automated mechanism to convert among various representations of the model considered. Each representation describes the same task in a format that provides a specific benefit: either because the representation is amenable to automated modelling and analysis, or because it describes the current model in a way that is convenient for a human to verify and improve. The remainder of this section contains a definition of the various components of a bootstrapping system, a description of the bootstrapping process, and examples of bootstrapping applications.

3.2.1 COMPONENTS

The general bootstrapping concept utilising two model representations is depicted in Figure 3.1. The number of representations is limited to two for the sake of simplicity – three or more representations can also be included in the model.

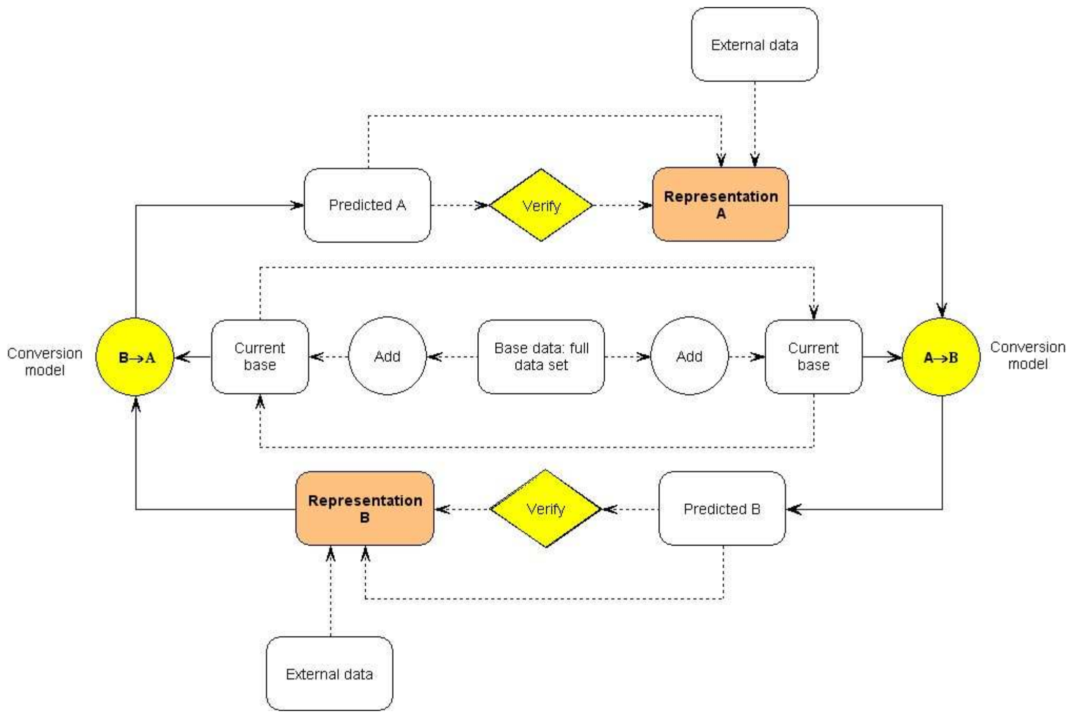


Figure 3.1: *General bootstrapping concept, utilising two model representations.*

The following components play a role during bootstrapping:

- *Alternative representations:* Two or more representations of the same model lie at the heart of the bootstrapping process. In the Fig. 3.1 these are indicated as *A* and *B*.
- *Conversion mechanisms:* Each conversion mechanism (indicated as $A \rightarrow B$ and $B \rightarrow A$) provides an automated or semi-automated means to convert data from one representation to another.
- *Verification mechanisms:* Once converted to a specific representation, the model can be improved via automated or human (manual) verification, indicated in the figure by the *Verify* components.
- *Base data:* This term is used to refer to the domain of the model. The *current base* indicates the domain that has been used in training the current model, and consists of a subset of, or the

full base data set. The current base data is implicitly or explicitly included in each of the two representations.

- *Increment mechanisms:* The *Add* components are used to increase the current base during bootstrapping. At the one extreme, all model instances can be included in a single increment; at the other, a single instance can be added per bootstrapping cycle. The increment mechanisms may utilise active learning techniques [69, 70] in order to select an appropriate set of instances to add.

- *External data:* This term refers to additional data sources that are utilised during bootstrapping. Typically, external data is used to initialise a bootstrapping system with models that were developed on a related task.

3.2.2 PROCESS

Prior to bootstrapping, the various representations are initialised in preparation for the first iteration. Typically only a single representation requires initialisation (A in this instance). External data may be included in this process, or the bootstrapping process starts without any initial knowledge of the task not included in the base data. The increment mechanism chooses the first base set to use. Once initialised, the bootstrapping process consists of the following steps, many of which are optional, as indicated:

1. The current base, as well as the current representation A is used to generate the next representation B.

2. B is verified, either manually or automatically. (Optional)

3. Based on the current state of the bootstrapping system, the increment mechanism increases the current base set. (Optional if (6) is not)

4. The current base, as well as the current representation B is used to generate representation A.

5. A is verified, either manually or automatically. (Optional)

6. Based on the current state of the bootstrapping system, the increment mechanism increases the current base set. (Optional if (3) is not)

This cycle is repeated until a sufficiently accurate and/or comprehensive model is obtained.

3.2.3 EXAMPLES

Two typical examples of bootstrapping are illustrated in Figures 3.2 and 3.3. The first example (Fig. 3.2) illustrates the automated bootstrapping scenario described in Section 1.2. For this task, the base data consists of audio data and phonemic transcriptions (initially not aligned with the audio

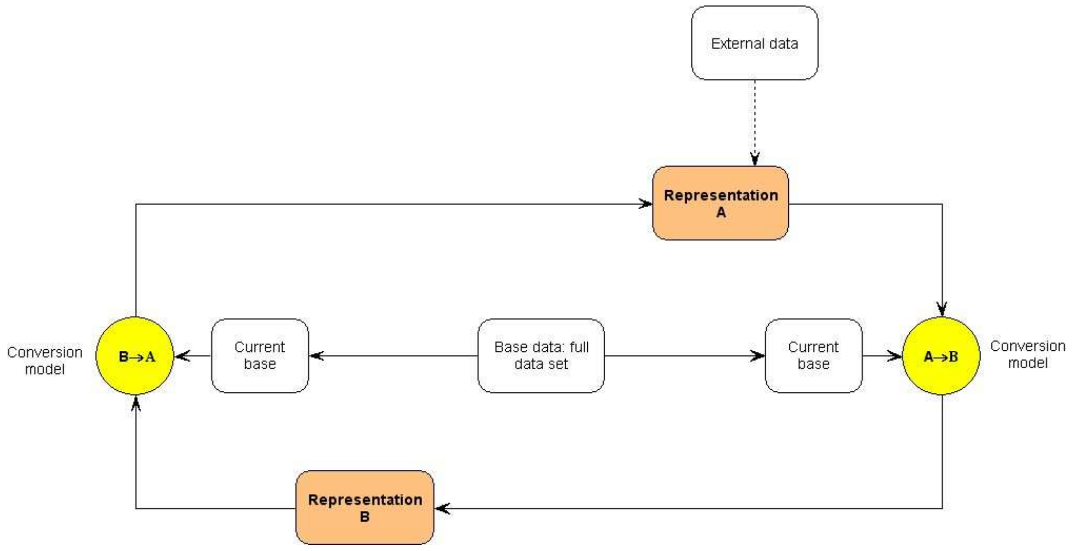


Figure 3.2: *An example of automated bootstrapping.*

data). A represents the phonemic segmentation of the audio data, and B the acoustic models derived from the segmentations. The focus is on the refinement of the acoustic models: the segmentations themselves are only important to the extent that they influence the quality of the acoustic models. The $A \rightarrow B$ mechanism consists of the training, re-clustering, and re-training of acoustic models, and the $B \rightarrow A$ mechanism of automatic Viterbi alignment of the phonemic transcriptions, utilising the current acoustic models.

The second example (Fig. 3.3) illustrates a simple bootstrapping scenario where machine learning and human intervention are combined, as would be the case, for example, when bootstrapping audio segmentations for Text-to-Speech purposes. The base data again consists of audio data and phonemic transcriptions; A represents the human-readable segmentation of the audio data, and B the acoustic models derived from the segmentations. The $A \rightarrow B$ mechanism consists of acoustic model training, and the $B \rightarrow A$ mechanism of automatic alignment. Here the focus is on achieving optimal segmentations and these are hand-verified until the acoustic models are stable enough to support accurate alignments (and possibly even after that, if high quality segmentations are required).

3.3 EFFICIENCY OF BOOTSTRAPPING PROCESS

The main aim of a bootstrapping system is to obtain as accurate a model as possible from available data. When human intervention is used to supplement or create the training data itself, the aim shifts towards *minimising the amount of human effort required during the process*. This is the focus of our

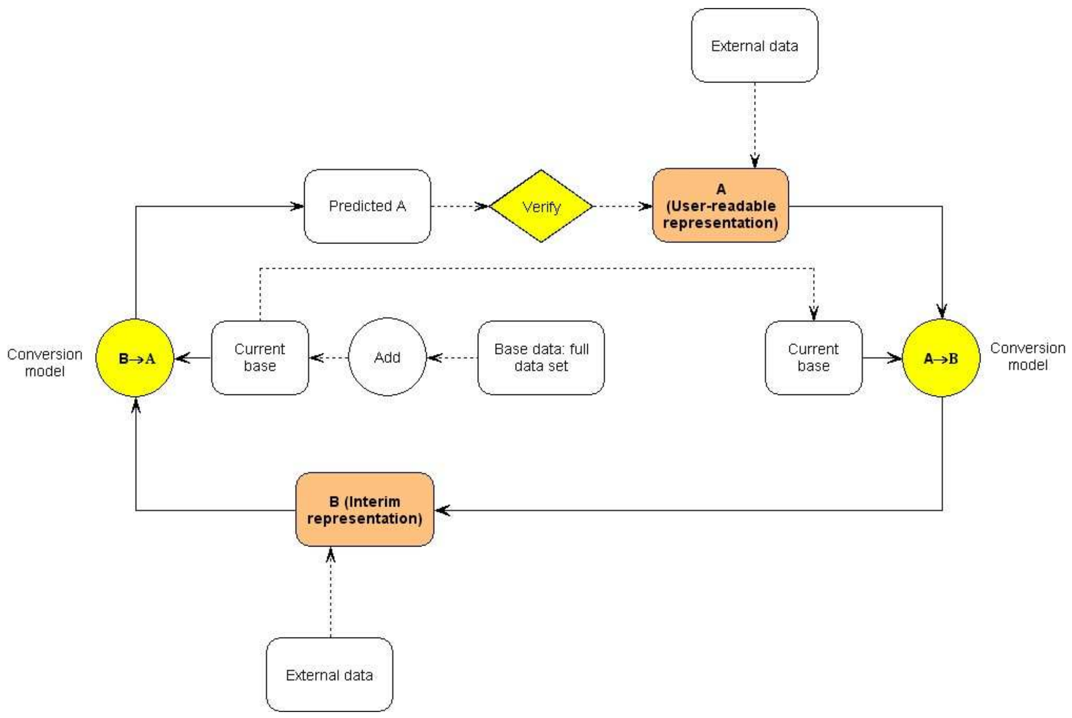


Figure 3.3: *An example of bootstrapping where machine learning and human intervention are combined.*

analysis, and we therefore measure bootstrapping efficiency as a function of model accuracy:

$$efficiency(a) = \frac{t_{bootstrap}(a)}{t_{manual}(a)} \tag{3.1}$$

where a is the accuracy of the current model as measured against an independent test set and $t_{bootstrap}(a)$ and $t_{manual}(a)$ specify the time (measured according to amount of human intervention) required to develop a model of accuracy a with and without bootstrapping respectively.

Bootstrapping is analysed according to bootstrapping cycles. While bootstrapping, all base instances do not result in valid data that can be included in the model training process. Of the instances that define valid base data, some will be correctly represented by the initial representation (B), and others will contain errors. We define a number of variables to assist us in the analysis of these instances: At the start of cycle x of the bootstrapping process, we define $n(x)$ as the number of instances included in the current base, $n_{invalid}(x)$ as the number of instances that are invalid, $n_{correct}(x)$ as the number of instances that are valid and correct, and $n_{error}(x)$ as the number of instances that are valid and incorrect. For these variables, the following will always hold:

$$\begin{aligned} n(x) &= n_{invalid}(x) + n_{valid}(x) \\ n_{valid}(x) &= n_{correct}(x) + n_{error}(x) \end{aligned} \tag{3.2}$$

Related incremental variables are used to represent the increase during cycle x, namely $inc\_n(x)$, $inc\_n_{invalid}(x)$, $inc\_n_{valid}(x)$, $inc\_n_{correct}(x)$ and $inc\_n_{error}(x)$. The same intervention mechanism may have different cost implications based on the $status$ of the instance. In the simplest case, the status of an instance may simply be correct, incorrect or invalid, but subtler differences are possible, e.g. the number of changes required to move from an incorrect to a correct version. The expected status of a newly predicted instance changes as the system becomes more accurate. Prior to human intervention at stage x of the bootstrapping process, the number of instances of each status within the current increment is given by:

$$inc\_n(x) = \sum_{s \in status} inc\_n(s, x) \tag{3.3}$$

Combining machine learning and human intervention in a way that minimises the amount of human effort required during the process can be achieved in two ways: (a) by minimising the effort required by the human verifier to identify errors accurately, and (b) by optimising the speed and accuracy with which the system learns from the human input. This section describes the various factors that influence the efficiency of the bootstrapping process from both these perspectives.

3.3.1 HUMAN FACTORS

The first human factor that impacts on the efficiency of the bootstrapping process relates to *required user expertise:* whether the task requires expert skills, or whether a limited amount of task-directed training is sufficient. If is assumed that the user has the skills required, the following measurements provide an indication of the efficiency of the bootstrapping process for a specific user:

- *User learning curve:* The time it takes for a specific user to become fully proficient using the bootstrapping system. Measured as t_{train}, initial training data is assumed to be discarded.
- *Cost of intervention:* The average amount of user time required per intervention i when an instance is in status s, for a fully trained user using the bootstrapping system. Measured as $t_{verify}(i, s)$ a different average cost may be associated with different types of interventions. If more than one intervention is used to generate a single instance during one cycle of bootstrapping, the combination of mechanisms is modelled as an additional (single) mechanism. Depending on the bootstrapping process, it may be more realistic to measure this value for a set of instances.
- *Task difficulty:* The average number of errors for a fully trained user using the bootstrapping system. Indicated by $error\_rate_{bootstrap}(i, s)$, this is measured in percentage as the average number of errors per 100 instances generated using intervention mechanism i to verify an instance initially in state s.

- *Quality and cost of user verification mechanisms:* Implicit in the above two measurements are the cost and effect on error-rate of additional assistance provided during user intervention. Rather than modelling additional user assistance provided during existing interventions separately, the combined intervention is again modelled as an additional type of intervention. In the same way, automated verification mechanisms are modelled as additional interventions.
- *Difficulty of manual task:* The average number of errors for a fully trained user developing instances manually. Indicated by $error\_rate_{manual}$, this is measured in percentage as the average number of errors per 100 manual instances developed, where each manual instance can be associated with an individual base data instance in the bootstrapped system.
- *Manual development speed:* The average amount of time per instance development for a fully trained user performing this task manually, measured as $t_{develop}$; this value can also be analysed separately per types of instance development as $t_{develop}(s)$, if so required.
- *Initial set-up cost:* The time it takes for a user to prepare the initial system for manual development or bootstrapping; measured in time as t_{setup_manual} and $t_{setup\_bootstrap}$ respectively.

3.3.2 MACHINE LEARNING FACTORS

The faster a system learns between verification cycles, the fewer corrections are required from a human verifier, and the more efficient the bootstrapping process becomes. From a machine learning perspective, learning speed and accuracy are directly influenced by:

- *Predictive accuracy of current base:* modelled as the expected number of instances of each status at a specific cycle of the bootstrapping process, and indicated by $E(inc\_n(s, x))$. Implicit to this measurement are four factors:
  - *Accuracy of representations:* The ability of the chosen representations to model the specific task.
  - *Set sampling ability:* The ability to identify the the next ‘best’ instance or instances to add to the knowledge base, possibly utilising active learning techniques.
  - *System continuity:* The speed at which the system updates its knowledge base. This has a significant effect on system responsiveness, especially during the initial stages of bootstrapping.
  - *Robustness to human error:* The stability of the conversion mechanisms and chosen representations in the presence of noise introduced by human error.
- *On-line conversion speed:* Any additional time costs introduced when computation is performed while a human verifier is required to be present (but idle while waiting for the computation to complete); measured as an average per number of valid instances developed and indicated by $t_{idle}(n)$.

- *Quality and cost of verification mechanisms:* The average amount of time required to utilise additional assistance mechanism j – from a computational perspective – when an instance is in status s, measured as $t_{auto}(j, s)$.
- *Validity of base data:* Using invalid data slows the bootstrapping process, especially if human intervention is required to verify the validity of base data; measured in % of base data, this is indicated by $valid\_ratio$.

Two additional factors that are not included explicitly in the general model, but can be included based on the requirements of the specific bootstrapping task, are:

- *Conversion accuracy:* The ability of the conversion to model convert between representations without loss of accuracy.
- *Effect of incorporating additional data sources:* The ability of the system to boost accuracy by incorporating external data sources at appropriate times.

3.3.3 SYSTEM ANALYSIS

The combined effect of the machine learning factors and human factors provide an indication of the expected cost of using the bootstrapping system. The time to develop a bootstrapping model via N cycles of bootstrapping, utilising a set of interventions I, is given by:

$$\begin{aligned} t_{bootstrap}(N, I) &= t_{setup\_bootstrap} + t_{train} + t_{iterate}(N, I) \\ &= t_{setup\_bootstrap} + t_{train} \\ &\quad + \sum_{x=1}^{N-1} \Bigg(\sum_{i \in I} \sum_{s \in status} (t_{verify}(s, i) + t_{auto}(s, i)) * inc\_n(s, x) \\ &\quad + t_{idle} * inc\_n_{valid}(x+1) \Bigg) \end{aligned} \tag{3.4}$$

where $t_{iterate}(N, I)$ combines the cost of the various iterations, excluding the cost associated with system setup and user training. The expected value of $inc\_n(s, x)$ depends on the specific conversion mechanism, and is influenced by $valid\_ratio$ and $error\_rate_{bootstrap}(i, s)$.

This cost of bootstrapping can be compared to the expected cost of developing n_{manual} instances via a manual process:

$$t_{manual} = t_{setup\_manual} + t_{develop} * n_{manual} \tag{3.5}$$

If $n_{bootstrap}$ and n_{manual} are chosen such that

$$E[inc\_n(correct, n_{bootstrap})] = E[inc\_n(correct, n_{manual})] \tag{3.6}$$

where the number of valid instances generated during bootstrapping is given by:

$$n_{bootstrap} = \sum_{x=1}^{N-1} inc\_n(valid, x) \tag{3.7}$$

the accuracy of each of the two systems is approximately equivalent, and the values of eq. 3.4 and 3.5 can be combined according to eq. 3.1 in order to obtain a measure of the expected efficiency of the bootstrapping process. We use this measure to analyse a specific bootstrapping system in Chapter 6.

3.4 BOOTSTRAPPING PRONUNCIATION MODELS

The scenario depicted in Fig. 3.3 can be applied to the bootstrapping of pronunciation models. In this case, the base data consists of a word list; A represents an explicit pronunciation dictionary, each instance consisting of a word and pronunciation pair; and B represents a set of grapheme-to-phoneme rules. The $A \rightarrow B$ mechanism represents grapheme-to-phoneme rule extraction, and the $B \rightarrow A$ mechanism grapheme-to-phoneme conversion. Additional verification assistance that can be provided include automated error detection, and audio support during verification.

3.4.1 ALGORITHMIC REQUIREMENTS

An appropriate grapheme-to-phoneme rule extraction and conversion mechanism lies at the heart of the bootstrapping process. From the discussion in 3.3.2 it follows that the following are the most important requirements for a grapheme-to-phoneme formalism to be used in bootstrapping:

1. It should have high predictive ability, even for very small training set sizes.
2. It should be able to represent the word/pronunciation data exactly (in order to prevent conversion loss when switching between representations).
3. It should allow continuous model updating at a low computational cost.
4. Pronunciation prediction should be fast.
5. It should be robust to noise in the training data.

3.5 CONCLUSION

In this chapter we defined a framework and terminology for the analysis of a bootstrapping system. We showed how this model applies to the bootstrapping of pronunciation models and defined the requirements for a grapheme-to-phoneme conversion mechanism suitable for bootstrapping. These requirements are taken into account in the next chapter (Chapter 4) in the search for such a mechanism. The bootstrapping topic itself is revisited in Chapter 6.

CHAPTER FOUR

GRAPHEME-TO-PHONEME CONVERSION

4.1 INTRODUCTION

In this chapter we analyse the grapheme-to-phoneme (g-to-p) conversion task through a number of experiments. Our aim is obtain a pronunciation modelling mechanism that is well suited to bootstrapping. We choose an instance based learning approach, with Dynamically Expanding Context (DEC) as the baseline algorithm, for reasons discussed in Section 4.2. We utilise the pronunciation dictionaries described in Section 4.3 to analyse various aspects of the task, and to benchmark our results. As DEC is sensitive to alignment errors, we first analyse grapheme-to-phoneme alignment accuracy (in Section 4.4), and define the alignment approach we utilise in subsequent experiments. We then proceed to analyse a number of variations of DEC, and suggest small adaptations to the standard algorithm (Section 4.5). These variations lead to the definition of a new grapheme-to-phoneme conversion algorithm described in Section 4.6. This algorithm – *Default & Refine* – has a number of attractive properties that makes it suitable for bootstrapping.

4.2 BASELINE ALGORITHM

As discussed in Section 3.4.1, the ideal grapheme-to-phoneme conversion mechanism will have the following characteristics:

1. High predictive ability, even for very small training set sizes.
2. Exact representation of training data.
3. Low computational cost (both for rule extraction and pronunciation prediction).
4. Robustness to noise in the training data.

Of the approaches discussed in Section 2.2.2, we exclude any that require linguistic input (such as finite state transduction) or extensive computational resources (such as meta-classifiers). Of the remaining approaches, most exhibit comparable asymptotic performance, with the best results currently achieved by pronunciation by analogy (PbA) approaches and instance-based learning methods, as described earlier.

Both PbA approaches and instance-based learning methods can provide exact representation after conversion, as required. Also, the computational complexity of examples within both of these classes of algorithms are within acceptable limits, with PbA approaches providing some advantage with regard to computational cost [25]. As bootstrapping is typically not the aim of grapheme-to-phoneme approaches, little information is available with regard to robustness to noise. The first requirement then becomes the deciding factor for choice of algorithm: how well does the algorithm generalise from very small data sets. Again, explicit information is not available, but it seems from the results provided by Damper *et al* in [25] that the PbA algorithm only starts to generalise well when the training dictionary is of sufficient size[1]. We therefore choose an example of instance-based learning as the basis for our initial experimentation. Specifically, we choose Dynamically Expanding Context (DEC), an algorithm that is simple to implement, and generalises fairly well from a small training set.

4.3 EXPERIMENTAL DATA AND APPROACH

We utilise the following databases during experiments:

- *NETtalk*, a publicly available 20,008-word English pronunciation dictionary [20], derived from Miriam Webster's pocket dictionary (1974). Hand-crafted grapheme-to-phoneme alignments are included in the electronic version.

- *FONILEX*, a publicly available pronunciation dictionary of Dutch words as spoken in the Flemish part of Belgium [19]. We obtained the exact 173,873-word pre-aligned version of the dictionary as used by Hoste [41].

- *OALD*, a publicly available English pronunciation dictionary [18]. We obtained the exact 60,399-word pre-aligned version of the dictionary as used by Black [23].

- *Afrikaans A*, a 5,013-word Afrikaans pronunciation dictionary, built using the bootstrapping system and developed as part of this thesis. This dictionary was transcribed by a linguistically sophisticated first-language Afrikaans speaker and manually verified by the author. Of the 5,013 words, 90 words are invalid: the remaining 4,923 words are all valid and distinct.

- *Afrikaans B*, a 8,053-word Afrikaans pronunciation dictionary, built using the bootstrapping system and developed as part of this thesis. This dictionary was bootstrapped from *Afrikaans*

[1]When trained on the (American English) Teachers' Word Book (TWB), the PbA algorithm that was evaluated achieved approximately 40% word accuracy after 2000 words [25]

A and transcribed by a linguistically sophisticated first-language Afrikaans speaker, but not exhaustively verified. (Some verification was performed, as described in Section 6.5.) Of the 8,053 words, 271 words are invalid: the remaining 7,782 words are all valid and distinct.

Where any of the above databases include pronunciation variants (one word associated with two or more valid pronunciations), all but the first pronunciation variant are removed from the database, prior to dividing the database into training and test sets. When we report on results, we use the term *phoneme correctness* to specify the percentage of phonemes identified correctly, *phoneme accuracy* as the number of correct phonemes minus the number of insertions, divided by the total number of phonemes in a correct pronunciation, and *word accuracy* to specify the percentage of words completely correct. While we typically report on phoneme accuracy only, phoneme correctness is sometimes included in order to provide a comparative measure with results from other sources. Unless otherwise stated, we perform 10-fold cross-validation. During 10-fold cross-validation we subdivide the entire corpus randomly into 10 distinct sub-sections, and then perform 10 training/testing experiments, training on nine of the sub-sections and testing on the tenth. For the different measurements (word accuracy, phoneme accuracy, phoneme correctness) we report on the standard deviation of the mean of each of these measurements, indicated by σ_{10}[2]. Where there is uncertainty with regard to the measure used in a benchmark result, word accuracy provides the least ambiguous comparison.

As in previous sections, we use the format

$$(x_1..x_m, g, y_1..y_n) \rightarrow p \tag{4.1}$$

to specify extracted grapheme-to-phoneme rules. Here g indicates the focal grapheme, x_i and y_j the graphemic context, and p the phonemic realisation of the grapheme g. We also use a more compact representation:

$$x_1..x_m - g - y_1..y_n \rightarrow p \tag{4.2}$$

to indicate the same rule. Note that each grapheme specifies a separate element, even though these separate elements are written next to each other (without spaces or other indicators of element boundary.)

4.4 GRAPHEME-TO-PHONEME ALIGNMENT

Errors in grapheme-to-phoneme alignment do not affect different rule extraction techniques to the same extent. DEC-based rule extraction mechanisms are sensitive to alignment accuracy. For example, the correct DEC extraction rule for the grapheme-pair ‘ee’ in English is $-e-e \rightarrow iy$ and $e-e- \rightarrow \phi$ where ϕ indicates the null phoneme. If the system incorrectly aligns the words “keen”

[2]If the mean of a random variable is estimated with n independent measurements, and the standard deviation of those measurements is σ, the standard deviation of the mean is $\sigma_n = \frac{\sigma}{\sqrt{n}}$.

and “seen” as follows: $k\ e\ e\ n \rightarrow k\ iy\ \phi\ n$ and $s\ e\ e\ n \rightarrow s\ \phi\ iy\ n$, DEC will not be able to extract the fairly simple rule specified above, as the two words provide conflicting evidence with regard to the pronunciation of the grapheme pair ‘ee’. Note that the linguistic accuracy of the position of the null phoneme is not important, as long as the choice of position is consistent across the set of training instances. As DEC is sensitive to alignment accuracy, we optimise the grapheme-to-phoneme alignment process before analysing the grapheme-to-phoneme conversion process.

4.4.1 PRE-PROCESSING OF GRAPHEMIC NULLS

Many languages require few or no graphemic nulls and the additional variability introduced by catering for graphemic nulls result in miss-alignments. For our base algorithm (*Align v1*) we use forced Viterbi alignment based on the probabilities *P(grapheme i* | *phoneme j)*; and initialise probabilities from words and pronunciations that have equal length, as described by Andersen *et al* [54]. However, we insert graphemic and phonemic nulls in two separate steps. In a pre-processing phase, graphemic null generator pairs (two graphemes that result in more than two phonemes) are identified by Viterbi alignment of all word-pairs where pronunciation length is longer than word length. Phonemic nulls are inserted in a second phase of Viterbi alignment. (Where the first phase introduces unnecessary graphemic nulls, these are typically mapped to phonemic nulls during the second phase.) In both phases the alignment process is repeated until no further likelihood improvement is observed.

Alignment accuracy on the *NETtalk* corpus using this implementation (*Align v1*) is higher than the results reported by Andersen *et al* [22], as compared in Table 4.1. This improvement is due to an implementation difference rather than a conceptual difference: The algorithms are similar, apart from the different handling of graphemic nulls, and graphemic nulls do not occur in the *NETtalk* corpus[3].

4.4.2 UTILISING THE PHONEMIC CHARACTER OF NULL-PHONEMES

An additional improvement can be obtained if the transcription convention used by *NETtalk* is adapted. In *NETtalk*, null phonemes are used to identify graphemes that are “deleted” during pronunciation, for example the word *writer* is transcribed as $w\ r\ i\ t\ e\ r \rightarrow \phi\ r\ ay\ t\ \phi\ axr$. An alternative convention would be to use null phonemes simply to identify instances where two or more graphemes give rise to a single phoneme (without identifying a particular grapheme as deleted), by aligning the first grapheme in such a group with a non-null phoneme, and subsequent graphemes with nulls. Using this convention, the word *writer* is transcribed as $w\ r\ i\ t\ e\ r \rightarrow r\ \phi\ ay\ t\ axr\ \phi$. A null phoneme then simply indicates that the phonemic realisation remains the same for more than one grapheme.

Using a set of about 40 rewrite rules, the *NETtalk* dictionary can be rewritten using either the one convention or the other. Using the second convention, the dictionary responds better to data-

[3]In earlier work, when adding graphemic nulls by hand, we found that the use of pseudo-phonemes can complement the use of pseudo-graphemes. Pagel *et al* [52] suggested the use of pseudo-graphemes (e.g. creating two graphemes Xx to represent the k and s phonemes that originate from x separately). We found that, when a more natural choice, the use of pseudo-phonemes (e.g. creating a ks phoneme to represent the k and s combination) can improve alignment accuracy.

driven alignment and the second version of our Viterbi algorithm (*Align v2*). This algorithm explicitly calculates the *probability that a specific grapheme is realised as a null phoneme, given the previous non-null phonemic realisation of the preceding grapheme or graphemes,* and provides a significant performance improvement, as shown in Table 4.1.

Table 4.1: *Phoneme and word alignment accuracy obtained on the NETtalk corpus.*

Database	Type	Phoneme	Word
NETtalk-original	Iterative Viterbi [22]	93.2	83.7
NETtalk-original	Align v1	96.5	87.3
NETtalk-rewritten	Align v2	98.7	95.4

The effect of the improvement in alignment accuracy on rule extraction accuracy is depicted in Fig. 4.1. The *Align v1* and *Align v2* algorithms are used prior to *DEC-min*[4] rule extraction on a 10,000-word subset of the *FONILEX* database, and grapheme-to-phoneme prediction accuracy measured against a 5000-word test set.

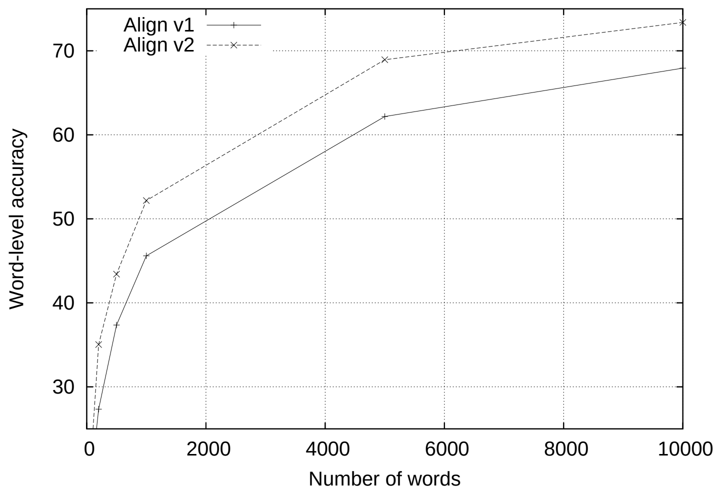


Figure 4.1: *Effect of different alignment algorithms on word-level pronunciation prediction accuracy of DEC-min, as measured on a 10,000-word subset of FONILEX.*

In order to verify that this effect is not corpus-specific, we perform a further evaluation using the *OALD* corpus. We analyse the effect of the two different alignment algorithms (*Align v1 and Align v2*) when extracting both *DEC-grow* and *DEC-min* rules using training sets of increasing size. For each training set of a specific size, 10 distinct training sets are generated. All training sets are tested against a non-overlapping 5970-word test set (10% of the full data set). A similiar trend is observed as on the *FONILEX* corpus, as depicted in Fig. 4.2. For example, the mean phoneme accuracy for *DEC-grow* rules trained on a 5000-word training set is 86.83% (with $\sigma_{10} = 0.07$) when aligned according to *Align v1*, and 87.54% (with $\sigma_{10} = 0.06$) when aligned according to *Align v2*. During the earlier

[4]The *DEC-min* algorithm is described in Section 4.5.2.

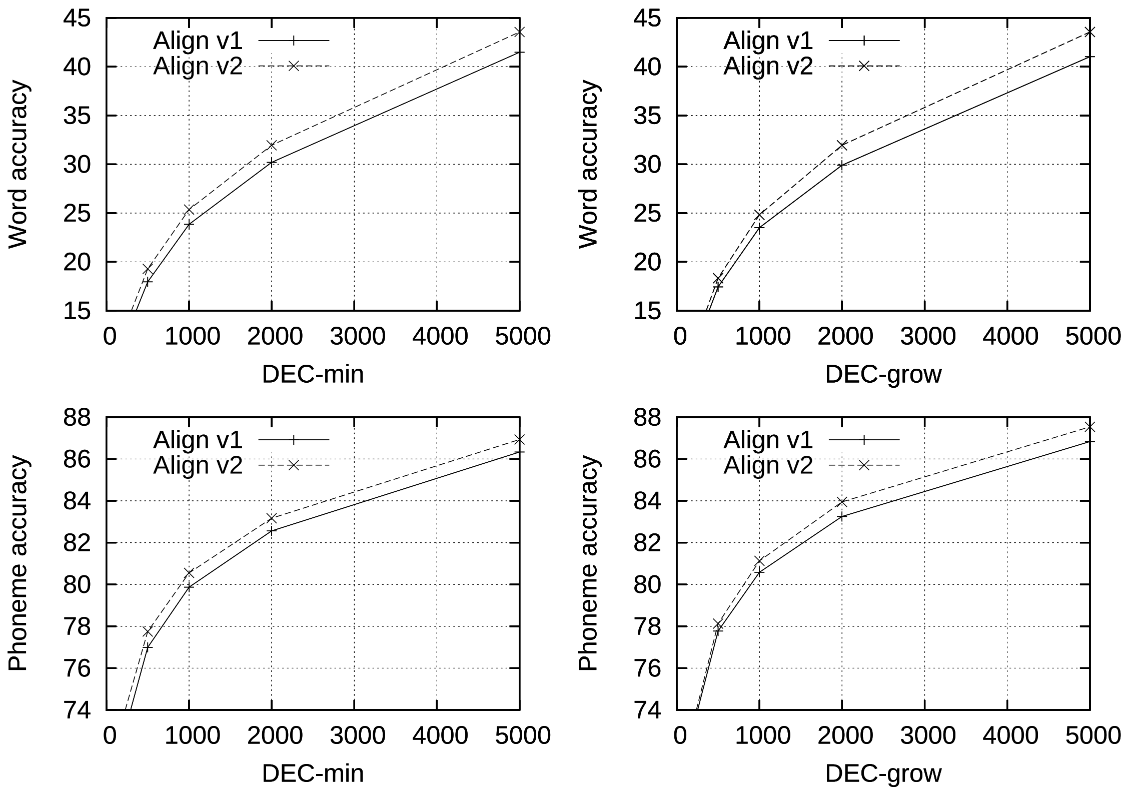


Figure 4.2: *Effect of different alignment algorithms on prediction accuracy of DEC-grow and DEC-min, as measured using the OALD corpus.*

stages of the rule extraction process (when alignment probabilities are still unstable) this provides a signficant advantage.

4.5 DEC-BASED GRAPHEME-TO-PHONEME PREDICTION

4.5.1 STANDARD DEC

A conceptual description of DEC as applied to the grapheme-to-phoneme problem by Torkkola [21] is provided in Section 2.2.2.3. In this section, we discuss the approach in further detail: Each DEC rule specifies a mapping of a single grapheme to a single phoneme for a given left and right graphemic context, i.e is of the form: *(left-context,grapheme,right-context)* $\rightarrow$ *phoneme*. Each word in the training dictionary is aligned with its pronunciation on a per-grapheme basis, as illustrated in Table 4.2. Rules are extracted by finding the smallest context that provides a unique mapping of grapheme to phoneme. If an $n-$letter context is not sufficient, the context is expanded to either the right or the left. This 'specificity order' influences the performance of the algorithm. Different orderings are illustrated in Table 4.3 as applied to grapheme *'s'* in the word *'interesting'*. Context 1 is expanded symmetrically on a right-grapheme-first basis, context 2 is expanded symmetrically on a left-grapheme-first basis, and context 3 favours the right context on a 2:1 basis. The set of extracted rules are stored as a hierarchical tree, with more general rules at the root, and more specific rules at the leaves. The tree is traversed from the root to the leaves, and the rule at the first matching leaf (the rule describing the

Table 4.2: *Word alignment and rule extraction in standard DEC.*

Alignment examples	r o s e → r ow z ϕ	
	r o w s → r ow ϕ z	
	r o o t → r uw ϕ t	
Rule examples for -o-	in context -o:	-o-o → uw
	in context -se:	-o-se → ow
	in context o-:	o-o- → ϕ

Table 4.3: *Different examples of context expansion order in DEC.*

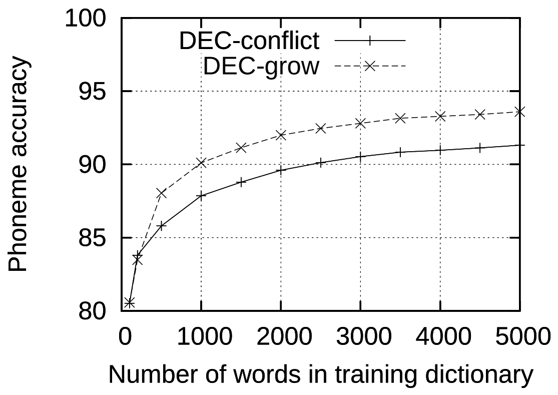
size	context 1	context 2	context 3
0	s	s	s
1	st	es	st
2	est	est	sti
3	esti	rest	esti
4	resti	erest	estin

largest matching context) is used to predict the specific grapheme-to-phoneme realisation. If no leaf is matched, the most probable outcome of the last matching leaf is used, as can be estimated from the training data. In our implementation of DEC, we do not explicitly order the rules in a tree structure, but number them according to the order in which they are extracted (corresponding to a topological sort of all rules that can apply to a single word). We then search via reverse rule order rather than tree traversal. This variation does not change the algorithm functionally.

If DEC is not allowed to grow an asymmetric context when it reaches a word boundary and conflicting rules are ignored (*DEC-conflict*) the performance of the algorithm degrades for larger training corpora, especially if rules regarding the context surrounding a grapheme early or late in a word are of predictive importance. In order to remove this effect, the version of DEC (*DEC-grow*) that was implemented as baseline algorithm allows a context to grow towards the opposite side if a word boundary is encountered. This effect is illustrated in Fig. 4.3 where we plot the results for *DEC-conflict* and *DEC-grow* during the initial stages of learning (using the *FONILEX* corpus).

4.5.2 SHIFTING WINDOWS

DEC, as applied by Torkkola [21] expands the context of a grapheme one letter at a time, either favouring the right- or left-hand side explicitly. We analyse the implications of using a sliding window rather than a strict expanding context. We define a sliding window that first considers all possible contexts of size *n*, before continuing to consider contexts of size *n*+1, which prevents rules with unnecessarily large contexts from being extracted. In contrast to the DEC context expansion of Table 4.3, a sliding window applied to grapheme *'s'* in the word *'interesting'* would result in the context ordering indicated in Table 4.4. Since multiple rules of the same context size may apply to a single grapheme-to-phoneme mapping (such as *re,s,ti* → *s* and *ere,s,t* → *s*), contexts that are already served



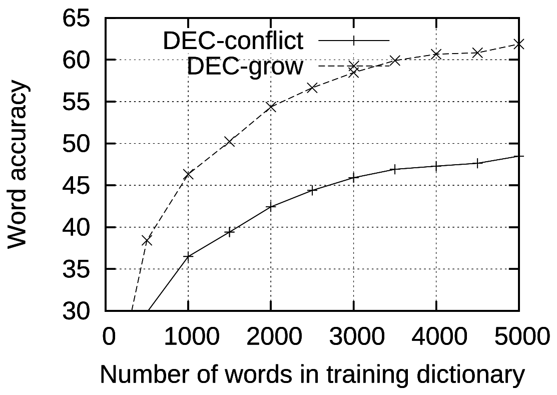


Figure 4.3: *Comparing DEC-conflict and DEC-grow during initial learning stage (first 5000 words of FONILEX). DEC-grow is chosen as baseline algorithm.*

Table 4.4: *Context expansion order in shifted DEC.*

order	size	context	order	size	context
1	0	s	2	1	st
3	1	es	4	2	est
5	2	sti	6	2	res
7	3	esti	8	3	rest
9	3	stin	10	3	eres

by existing rules can be removed to prevent over-specialisation. Because all contexts of each size are considered, the order in which contexts are expanded (for a specific context-level) becomes less significant than in standard DEC.

Figures 4.4 and 4.5 compare the performance of different DEC variations. In all experiments, a symmetric right-first expansion scheme is used[5] (as also in Table 4.4). The size of the maximum context allowed when extracting rules is not restricted, and the same word training order (random selection from corpus) is used. In order to compare with previous results, we use the exact alignments as used in [41]. Where word variants occur, we only use the first variant – both for training and testing purposes.

Three shifted window versions of DEC are implemented: extracting the first valid rule encountered (*DEC-win*) extracting the maximum number of valid rules (*DEC-max*) and pruning this system to obtain the minimum number of rules that still provide full coverage for the training corpus (*DEC-min*). When a shifting window is used, more than one conflicting rule of the same size may apply to a word. Various conflict resolution strategies can be implemented: in the set of experiments reported below, the most frequently observed rule is favoured. For the training set sizes analysed, the pruned, shifted window version of DEC (*DEC-min*) provides a small but consistent performance improvement in word accuracy[6]. *DEC-win* is not shown, but results in a learning curve similar to *DEC-grow*, both

[5]A symmetric, right-first expansion scheme is used when rule options are generated for consideration prior to selection of the actual rule – actual rules are generated according to a shifting window, and do not exhibit strict right-first behaviour.

[6]Note that phoneme accuracy initially follows a different trend for this corpus.

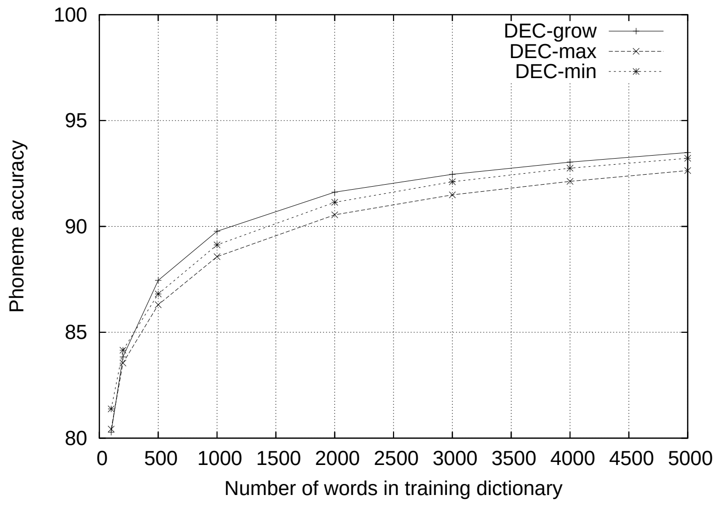


Figure 4.4: *Word-level accuracy of different DEC variations during initial learning stage, as measured using the first 5000 words of FONILEX.*

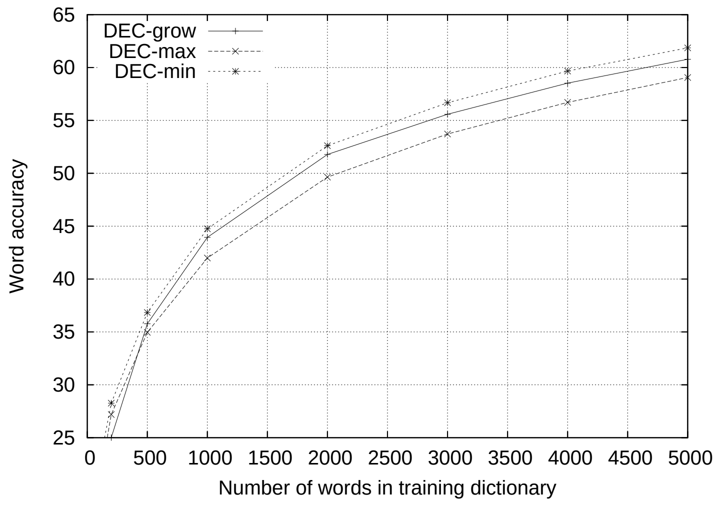


Figure 4.5: *Phoneme-level accuracy of different DEC variations during initial learning stage, as measured using the first 5000 words of FONILEX.*

with regard to word and phoneme accuracy. Asymptotic performance is only approached for larger training sets, as compared in Table 4.5. *DEC-min* continues to perform better than *DEC-grow*, with a small margin. The improvements during the initial learning stages are small, and introduce additional overhead during computation. Of more interest is that the new DEC variation (*DEC-min*) forms the basis for further algorithmic improvement, as discussed in the next sections.

As can be expected, the extracted rule sets grow in different ways with regard to rule number and rule length, as the size of the training dictionary increases. An analysis of the different types of rule sets extracted from the same training dictionary is provided in Table 4.6. The numbers of rules of each size (the size of the context that specifies the rule) are compared, as extracted from different sized training dictionaries using *DEC-grow*, *DEC-max* and *DEC-min*. Note that *DEC-max* tends to extract more rules than *DEC-grow* but that these rules tend to be shorter. *DEC-min* reduces

Table 4.5: *Phoneme correctness, phoneme accuracy and word accuracy comparison for different DEC variations, as measured using the FONILEX corpus.*

	phoneme correctness		phoneme accuracy		word accuracy	
		$\pm\sigma_{10}$		$\pm\sigma_{10}$		$\pm\sigma_{10}$
DEC-max	98.44	0.01	98.28	0.01	88.71	0.06
DEC-grow	98.50	0.01	98.32	0.04	88.60	0.07
DEC-win	98.57	0.01	98.40	0.01	89.53	0.05
DEC-min	98.58	0.01	98.41	0.01	89.58	0.06

the number of rules significantly (in comparison with *DEC-max*). *DEC-min* extracts slightly more rules than *DEC-win*, but as can be expected, these are much shorter (more general).

Table 4.6: *Number and size of rules: DEC-grow, DEC-max and DEC-min*

Rule type:	DEC-grow			DEC-max			DEC-min		
Dict size:	100	1,000	10,000	100	1,000	10,000	100	1,000	10,000
1	27	27	27	27	27	27	27	27	27
2	65	92	103	108	105	103	86	104	102
3	127	545	1259	256	1224	2375	102	705	1661
4	19	323	1996	24	926	7031	9	375	3469
5	7	131	1845	-	78	3081	-	33	1381
6	-	33	712	-	7	341	-	3	178
7	-	8	280	-	-	27	-	-	18
8	-	-	71	-	-	5	-	-	3
9	-	-	32	-	-	1	-	-	1
10	-	-	4	-	-	1	-	-	1
11	-	-	5	-	-	-	-	-	-
12	-	-	1	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-	-
14	-	-	-	-	-	-	-	-	-
15	-	-	1	-	-	-	-	-	-
Total	245	1,160	6,337	415	2,367	12,992	224	1,247	6,841

4.5.3 RULE PAIRS

When analysing the specific errors made by these DEC variations, it becomes apparent that some rules occur in 'rule pairs', i.e. two rules always occur as companions in the training data. These rule pairs are sometimes not applied as companions in the test data, causing errors. For example, during rule extraction a rule $-e - en \rightarrow iy$ is typically followed by a second rule rule $e - e - n \rightarrow \phi$ or $e - e- \rightarrow \phi$, and is a better rule to apply when predicting the instance $k - e - en$ than the otherwise equally likely rule $\#k - e-$. We experiment with the implication of forcing such rule pairs to occur in tandem. First, we identify rule pairs that always occur together in the training data and exhibit a context overlap of at least the two focal graphemes. Then we restrict our rule application to only use one of the rules in such a pair if the second rule in the pair is also applicable to the same training instance. However, constraining rule pairs in this way does not have a significant effect on predictive accuracy: In some instances the rule pair approach does correct a second phoneme that would otherwise have been wrong, but in a comparable number of cases this approach causes a second phoneme to be wrong, which would otherwise have been correct. We therefore do not continue with further experimentation along this route.

4.5.4 CONFLICT RESOLUTION

In standard DEC, the largest matching rule is always unique[7]. When a shifting window is used, more than one conflicting rule of the same size may apply to a word. If we use $num(r, p)$ to specify the number of training instances that match the context of a specific rule r and specific outcome as p, we calculate the 'accuracy' of the rule r as:

$$accuracy(r) = \frac{num(r, outcome(r))}{\sum_x num(r, x) + 1} \text{ for all possible outcomes } x. \tag{4.3}$$

In the experiments described above, if more than one candidate rule (of the same size) is applicable to the current word being predicted, we choose the rule for which $accuracy(r)$ is highest. This is a fairly simple conflict resolution strategy, and various alternative options are possible. We experiment with a number of these, including (1) voting among possible rules (choosing the outcome that most of the candidate rules agree upon), (2) applying the smaller (fall-back) context rather than any of the larger conflicting rules, and (3) simply choosing any of the rules at random (in practice whichever of the candidate rules was generated first during rule extraction), and find no consistent improvement using any of the alternative conflict resolution strategies. We continue to use the initial conflict resolution strategy (highest $accuracy(r)$) for further experimentation.

4.5.5 DEFAULT RULES

The question of how to best resolve conflict is closely linked to the question of how to best define default rules. One of the consequences of DEC rule extraction is that there exists only a single rule of any given length that can potentially apply to a specific word (where this length lies between one and the total length of the word being predicted). If the word being predicted is of length n, and no matching rule of length n exists, then a single rule of size $n - 1$ may potentially apply. In effect the latter rule acts as 'back-off value' for the rule of length n. If a rule of length $n - 1$ does not exist either, the (unique) matching rule of length $n - 2$ becomes the next possible candidate[8]. When using shifting windows, there is no longer a unique rule of any given length that can potentially apply when predicting a word – more than one candidate may exist. We therefore consider the effect of adding default rules explicitly: for any set of rules of context size n with one or more internal disagreements

[7]This rule may be conflicted (i.e. not a leaf node in Torkkola's original implementation) in which case the most frequently observed outcome across the training data is generated, but no conflicting rules of the same size can exist.

[8]From a conceptual perspective – this is not the process that is followed in practice during DEC prediction.

and no 'default rule' of size $n - 1$, we add an explicit rule of context size $n - 1$ with an outcome p such that $num(r, p)$ is the maximum over all possible outcomes. Interestingly, adding this additional information decreases rule accuracy. An error analysis indicates that inappropriate 'default rules' are extracted: while these rules correctly 'fill the gaps' among the rules extracted from the training data, the 'default rules' are forced to specific value by the previously extracted *DEC-min* rules, and do not generalise well. This leads us to the definition of a default-and-refinement approach to grapheme-to-phoneme prediction, as discussed in the next section (Section 4.6). This approach utilises a similar rule definition format as DEC, but the rule extraction process is more distant from original DEC than the variations studied up to this point.

4.6 A DEFAULT-AND-REFINEMENT APPROACH TO G-TO-P PREDICTION

Grapheme-to-phoneme prediction algorithms rely on the connection between the spoken and written form of a language. It is expected that, the more modern the writing system of a language, the stronger this connection, and the more regular the spelling system of the language [71]. This may not always hold in practice, for example, when a language with mainly (or only) an oral tradition is transcribed for the first time, and the variability introduced through the initial transcription process has not yet stabilised through usage or an education system that utilises the written form. While alternative outcomes are possible, the languages studied as part of this thesis all exhibit a combination of a fairly modern writing system associated with a fairly to highly regular spelling system.

The more regular the spelling system of the language, the stronger the concept of a 'default phoneme': a grapheme that is realised as a single phoneme significantly more often than as any other phoneme. Figure 4.6 and Figure 4.7 illustrate this phenomenon for Flemish. When counting the number of times a specific grapheme is realised as a specific phoneme, most graphemes follow the trend depicted in Figure 4.6. Here, *y* is realised as a single phoneme more than 60% of the time, with the next two phonemic candidates occurring only 24% and 4% of the time, respectively. For graphemes that exhibit 'conflicted default phoneme' behaviour, such as (*h,j,n,u*), the trend is less strong, but also clearly discernible, as depicted in Figure 4.7. Similar trends are observable for languages with less regular spelling systems, with a larger proportion of graphemes of these languages displaying the behaviour depicted in Figure 4.7.

We use this information to define an algorithm that utilises greedy search to find the most general rule at any given stage of the rule extraction process, and explicitly orders these rules according to the reverse rule extraction order[9]. Explicitly ordering the rules provides flexibility during rule extraction, and ensures that the default pattern acts as a back-off mechanism for the more specialized rules. The framework we use is similar to that used in previous sections: Each grapheme-to-phoneme rule

[9]It is interesting to note that, while the rule application order of DEC is ordered by context size (largest rule first), our reverse rule extraction order automatically reverts to context size ordering in the case of DEC-based rule extraction.

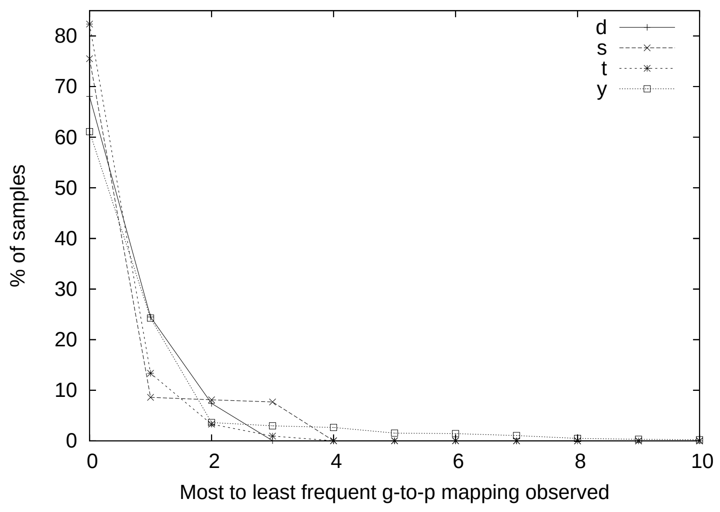


Figure 4.6: *Default phoneme behaviour of graphemes d,s,t and j in Flemish. Only the first 10 phonemic candidates are displayed.*

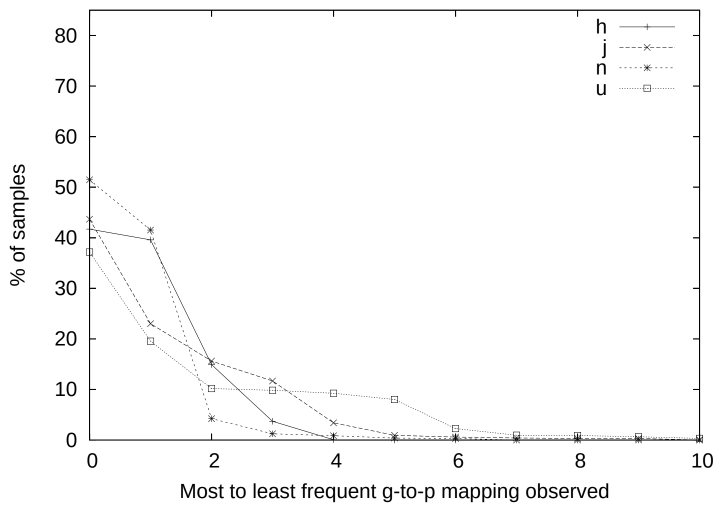


Figure 4.7: *Conflict phoneme behaviour of graphemes h,j,n,u in Flemish. Only the first 10 phonemic candidates are displayed.*

consists of a pattern

$$g_{left} - g - g_{right} \rightarrow p \tag{4.4}$$

where g indicates the grapheme being considered, g_{left} and g_{right} are the graphemic left and right contexts of the rule, and p the specific phonemic realisation of g. The pronunciation for a word is generated one grapheme at a time. Each grapheme and its left and right context as found in the target word are compared with each rule in the ordered rule set; and the first matching rule is applied.

Prior to rule extraction, grapheme-to-phoneme alignment is performed according to the Viterbi-based alignment process described in Section 4.4. Pronunciation variants are currently not allowed: if a word has more than one possible pronunciation, only the first is kept. Each aligned word-pronunciation pair is used to generate a set of possible rules by extracting the sub-pattern of each word pattern; an example of such a process is shown in Table 4.7.

Table 4.7: *The relationship between a word (test) and, for one of its graphemes (e), the word pattern and sub-patterns that are generated during rule extraction.*

Word	test
Word pattern	#t-e-st# → eh
Sub-patterns	-e- → eh,-e-s → eh,t-e- → eh,t-e-s → eh t-e-st → eh, #t-e-s → eh,-e-st# → eh #t-e-st → eh,t-e-st# → eh,#t-e-st# → eh

Once all possible rules have been generated in this way, rules are extracted on a per-grapheme basis, one rule at a time. For any specific grapheme, applicable words are split into two sets based on whether the current rule set (initially empty) predicts the pronunciation of that grapheme accurately (*Completed* words) or not (*New* words). These two large word sets are used to keep track of status, but further manipulation utilises two sets of sub-patterns: the *Possible* sub-patterns, indicating all possible new rules, and consisting of all the sub-patterns of each word pattern in *New*, excluding all for which the left-hand side is an existing rule; and the *Caught* set of sub-patterns, indicating all the sub-patterns covered by the current rule set irrespective of whether the outcome of the rule matches that of the word or not. Both the *Possible* and *Caught* sets of sub-patterns count the number of times, per possible outcome, that a matching word pattern is observed in the relevant word sets.

The next rule is chosen by finding the pattern for which the matching count in *Possible* minus the conflicting count in *Caught* is highest. (The conflicting count is the number of times a matching left-hand pattern is observed with a conflicting right-hand phoneme.) Definition of a new rule moves words from the *New* to the *Completed* set. Any words that are currently in the *Completed* set and conflict with the new rule, are moved back to the *New* set. This process is repeated until all words have been moved from the *New* to the *Completed* set. The algorithm ensures that the next rule chosen is the one that will cause the most net words to be moved from the *New* to the *Completed* set, irrespective of context size. As this number (net words processed) is always positive[10], the algorithm cannot enter an infinite loop. The stronger the default behaviour exhibited by a specific grapheme described by a new rule, the more words are processed during the extraction of that specific rule. Conflict is only resolved in the *Completed* set: new rules are allowed to conflict with words still in *New*, which ensures that the rule set is built for the default pattern(s) first.

In order to ensure computational efficiency when trained on larger dictionaries, we use the following techniques during implementation:

- Words are pre-processed and the word patterns relevant to a single grapheme extracted and written to file. All further manipulation considers a single grapheme (and the corresponding set of word patterns) at a time.

- The context size of the sub-patterns considered is grown systematically: only sub-patterns up to

[10]A rule based on a full word pattern can only apply to that single word, and will result in a 'net move count' of 1. Since the maximum of all these 'net move counts' is selected, this value will always be positive.

size $max + win$ are evaluated, where max indicates the current largest rule, and win is defined to ensure that any larger contexts that may be applicable are considered, without requiring all patterns to be searched.

- Whenever a sub-pattern in *Possible* or *Caught* reaches a count of zero, the sub-pattern is deleted and not considered further, unless re-added based on an inter-set move of a related word.

While these techniques ensure that a fairly large dictionary (200,000 words) can be trained in an acceptable amount of time when using the process in a non-interactive fashion, the process to train a sizeable dictionary becomes too slow for interactive bootstrapping. This issue is addressed further in Section 4.6.4. In the remainder of this thesis we refer to the algorithm described above as '*Default&Refine*'.

4.6.1 ASYMPTOTIC PERFORMANCE

In order to evaluate the asymptotic behaviour of *Default&Refine,* we compare our results on a fairly large corpus with published results for a number of alternative algorithms. As the *Default&Refine* algorithm is motivated by 'default behaviour', we first evaluate the algorithm on a language with a fairly regular spelling system (Flemish), before testing it on a language with an irregular spelling system (English).

4.6.1.1 REGULAR SPELLING SYSTEMS

We evaluate the accuracy of the *Default&Refine* algorithm when trained on the full *FONILEX* training set, and compare its performance with that of alternative algorithms in Table 4.8: the *IB1-IG* result utilises an instance-based learning algorithm and is as reported in [41]; the *DEC-grow* and *DEC-min* results are calculated using the algorithms described in Section 4.5.2; and the *D&R* result reports the *Default&Refine* values. The *DEC* and *Default&Refine* experiments utilise the same alignments as used in [41].

Table 4.8: *Phoneme correctness, phoneme accuracy and word accuracy comparison for different algorithms using the FONILEX corpus*

	phon correct		phon accuracy		word accuracy	
		$\pm\sigma_{10}$		$\pm\sigma_{10}$		$\pm\sigma_{10}$
IB1-IG	98.18	-	-	-	86.37	-
DEC-grow	98.50	0.01	98.32	0.04	88.60	0.07
DEC-min	98.58	0.01	98.41	0.01	89.58	0.06
D&R	98.87	0.01	98.78	0.01	92.03	0.06

The focus of [41] was to investigate the effect of cascading two classifiers – one trained on *FONILEX* and one on *CELEX* – a Dutch variant corpus, and creating meta-classifiers using C5.0 (decision tree learning), IB1-IG (instance-based learning as described in Section 2.2.2.3), IGTREE (an algorithm that induces decision trees utilising information gain) and MACCENT (a maximum entropy-based algorithm). The highest accuracy reported was for such a meta-classifier system: 91.55% word accuracy for a single meta-classifier; and 92.25% word accuracy for a meta-meta-classifier of all meta-classifiers. (These systems all utilised the $CELEX$ data as an additional data source.) We find that *Default&Refine* has good asymptotic accuracy, and performs better than the comparative (single) classifiers.

4.6.1.2 LESS REGULAR SPELLING SYSTEMS

As the algorithm is motivated by 'default behaviour' we were interested in the extent in which the algorithm would fail for a language such as English, with a less regular spelling system. We therefore evaluate the asymptotic performance of the algorithm against benchmark results available for both the *NETtalk* and the *OALD* corpus. It is reassuring to find that the algorithm again performs well, as shown in Tables 4.9 and 4.10.

Table 4.9: *Phoneme accuracy, phoneme correctness and word accuracy comparison for different algorithms using the NETtalk corpus*

	phon correct		phon accuracy		word accuracy	
		$\pm\sigma_{10}$		$\pm\sigma_{10}$		$\pm\sigma_{10}$
Trie	-	-	89.8	-	51.7	-
DTree	-	-	89.9	-	53.0	-
DEC-T	-	-	90.8	-	-	-
DEC-Y	-	-	92.21	-	56.67	-
D&R	91.37	0.08	90.50	0.1	58.66	0.21
SMPA	-	-	93.19	-	63.96	-

In Table 4.9 we compare the performance of a number of algorithms on the *NETtalk* corpus. We list the results obtained by Andersen *et al* [22] using Trie structures (*Trie*) and decision trees (*DTree*) respectively; by both Torkkola [21] and Yvon [36] using Dynamically Expanding Context (*DEC-T* and *DEC-Y*); by Yvon [36] using SMPA, a pronunciation-by-analogy algorithm; and the results of *Default&Refine* (*D&R*) using own alignments. The phoneme correctness reported in [36] for DEC seems anomalously high, in relation to our own experiments, those obtained in [21], and the reported word accuracy. The *SMPA* algorithm employs a pronunciation by analogy approach, and is less suitable for training on very small data sets. The latter results only pertain to words that could be pronounced – about 0.5% of words were not pronounceable with SMPA when fully trained. Note also that the SMPA results score the accuracy of variants in the test set differently to the approach employed in this thesis[11].

In Table 4.10 we compare the performance of *Default&Refine* (*D&R*) with the results obtained by

[11]In the SMPA experiments all variants but one are removed from the training set, but all variants are retained in the test set – if any of the possible variants are generated during testing, the prediction is marked as accurate. This is different to the scoring approach used in this thesis, as described in Section 4.3

Black *et al* [23] using Classification and Regression Trees (*CART*) for two data sets: one including stress assignment (SA) and one without. We use the exact alignments, and the same single training set and test set as used by Black[12]. The CART trees were generated taking part-of-speech information into account – which *Default&Refine* does not use. Without POS information, the CART result (with stress assignment) decreases to 95.32% phoneme correctness and 71.28% word accuracy .

Table 4.10: *Phoneme accuracy, phoneme correctness and word accuracy comparison for CART and Default&Refine using the OALD corpus (SA indicates stress alignment)*

	phon correct	phon accuracy	word accuracy
Incl. SA:			
CART	95.80	-	74.56
D&R	97.12	96.87	83.76
Excl. SA:			
CART	96.36	-	76.92
D&R	97.80	97.56	87.40

4.6.2 LEARNING EFFICIENCY

In order to use this algorithm for the bootstrapping of pronunciation dictionaries, we are specifically interested in the performance of the algorithm when trained on very small training sets. We therefore evaluate word and phoneme accuracy for different training dictionaries of sizes smaller than 3,000 words, using subsets from *FONILEX*. Figure 4.8 demonstrates the phoneme accuracy learning curve for *Default&Refine* in comparison with *DEC-grow*. Each rule set is evaluated against the full 17,387-word test set.

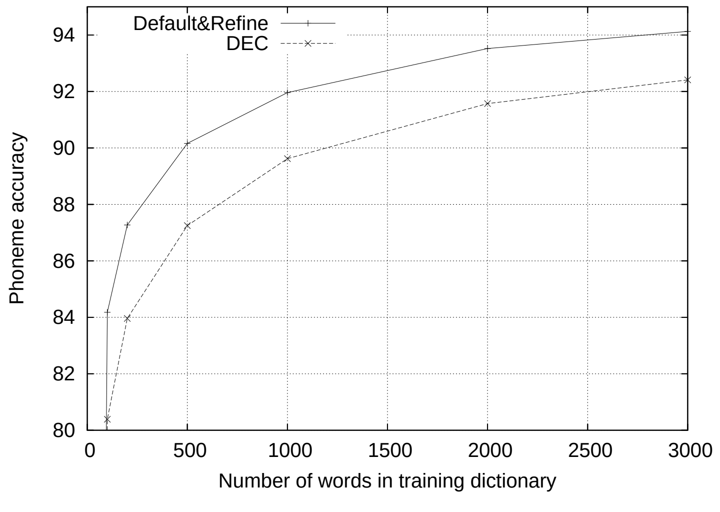


Figure 4.8: *Phoneme accuracy during initial 3000 training words, as measured using the FONILEX corpus.*

[12]When 10-fold cross-validation is performed using different subsets of this data set, a slightly lower cross-validated accuracy is obtained: 96.62% phoneme accuracy and 82.37% word accuracy when stress assignment is included, and 97.66% phoneme accuracy and 86.41% word accuracy without stress assignment.

4.6.3 SIZE OF THE RULE SET

While the size of the rule set is typically not a concern during grapheme-to-phoneme bootstrapping, it can be important for other applications (such as dictionary compression). We therefore analyse the size of the rule set, and find that the rule set extracted by *Default&Refine* is significantly smaller that extracted by *DEC-grow*, as shown in Figures 4.9 and 4.10. *Default&Refine* provides both a more accurate and more compact prediction model: the 156,486-word training dictionary is represented with 100% accuracy by 15,053 rules.

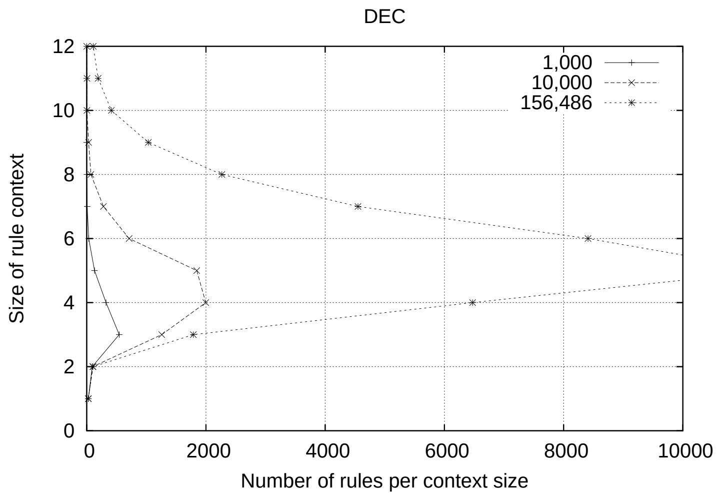


Figure 4.9: *Number of rules per context size extracted by DEC-grow from training dictionaries of three different sizes.*

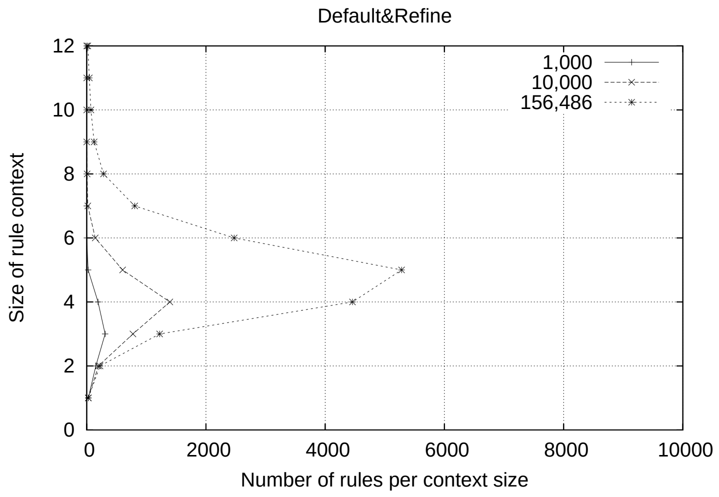


Figure 4.10: *Number of rules per context size extracted by Default&Refine from training dictionaries of three different sizes.*

4.6.4 CONTINUOUS LEARNING

The ideal bootstrapping system will be able to update the rule set after every correction by the verifier, immediately incorporating further learning in the bootstrapping knowledge base. The time taken for such updates is therefore of crucial importance. The update speed is influenced by two factors: the alignment speed and the rule extraction speed. If n represents the number of words in the training dictionary, then the complexity of the alignment process and that of the rule extraction process is both approximately $O(n)$, if it is assumed for the sake of simplicity that all words are more or less of equal length[13]. This is typical of various of the rule extraction techniques that are appropriate for grapheme-to-phoneme bootstrapping.

If the entire set of training words is processed after every correction, the update time becomes a limiting factor as the dictionary grows. In our implementation, continuous updating becomes unwieldy when the number of words with known pronunciations exceeds approximately 2000. On the other hand, by performing batch updates at specific times that suit the verifier (e.g. at the end of a verification session), the update time does not become a constraint, but the learning obtained during the session is not utilised to refine models until after the end of the session. In order to obtain an algorithm that allows for continuous model updating while keeping the update time within acceptable limits, an incremental version of the *Default&Refine* algorithm was developed.

While the original algorithm creates a set of graphemic rule trees (one tree per grapheme) from the training set by considering all the training words simultaneously, the incremental version utilises the trees constructed during the previous (batch mode) update, and adds the new refinements as leaves to these trees: for each grapheme in the new word, if the realised phoneme is predicted accurately by the current graphemic tree, no update occurs; otherwise the smallest rule is extracted that will describe the new word without affecting any of the existing predictions. This version has $O(d)$ complexity where d represents the average depth of the various graphemic rule trees (which is approximately equivalent to the average context size of the graphemic rule set). Using this incremental process, additional learning can be obtained from the new words added without causing discernible delay, even for large training dictionaries.

In practice, the bootstrapping process operates in two phases: during the first phase a batch update occurs for every word; during the second phase a batch update occurs at synchronisation events only, and incremental updates are performed in between synchronisation events. The interval between synchronisation events is based on a set number of “update words”, i.e. words that have been corrected by the verifier (words that were correctly predicted prior to verification do not contribute to this count). At the end of this interval, a synchronisation event occurs: the complete training dictionary is re-aligned, and new rules are extracted in batch mode. During the update interval, the Viterbi probabilities calculated at the previous synchronisation event are used per word to perform a fast alignment (the probabilities are used in the standard way, but not updated) and incremental *Default&Refine* is used to extract additional rules from the single aligned word-pronunciation pair. Phase 2 is initiated well before the time required by the full update event becomes noticeable. (For our current system we progress from phase 1 to phase 2 when 1500 valid words have been processed.)

As can be expected, the new algorithm is an approximation of standard *Default&Refine*, and

[13]See section 4.7.3 for a further discussion of the computational complexity of *Default&Refine*.

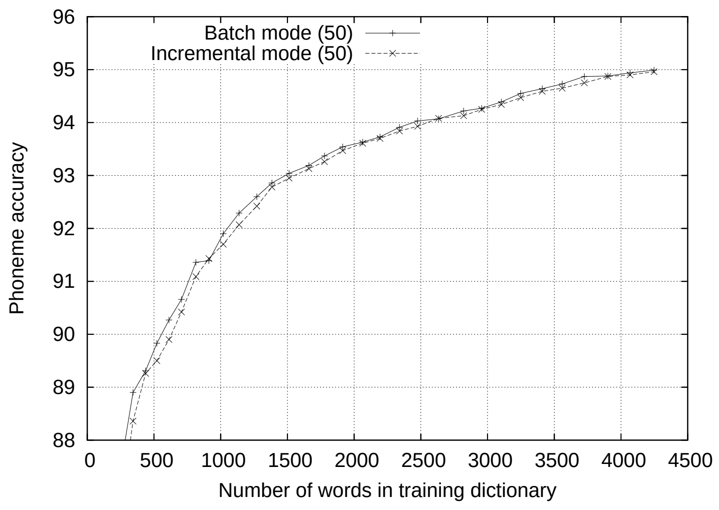


Figure 4.11: *Phoneme accuracy comparison for incremental and batch mode at an update interval of 50, measured using the FONILEX corpus.*

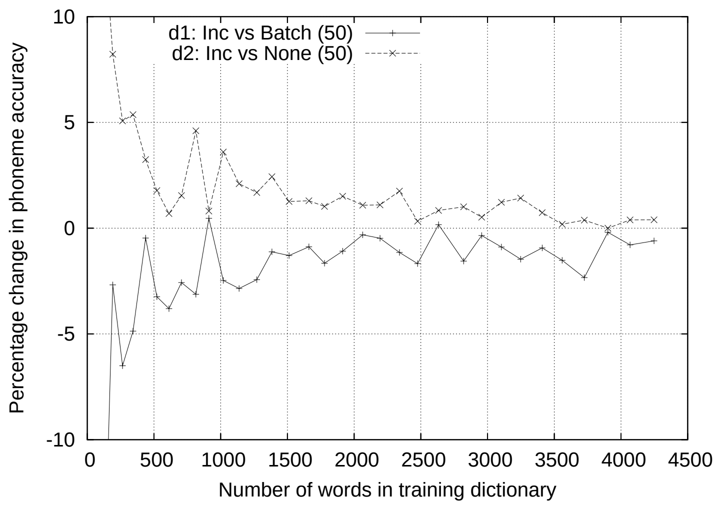


Figure 4.12: *Relative change in phoneme accuracy when comparing incremental with batch mode (δ_1), and incremental mode vs no updating between synchronisation events (δ_2); both at update interval 50.*

therefore somewhat less accurate than the original. We evaluate the performance of the system using an existing pronunciation dictionary (*FONILEX*), and perform 10-fold cross-validation on all our results. In order to determine the efficiency of the incremental approach, we first compare the two rule extraction processes (incremental mode and batch or standard mode) without taking changes in alignment into account. We utilise the same set of alignments[14] for both types of rule extraction, and measure phoneme accuracy on the same training set using the two different algorithms. We find that the decrease in accuracy is slight once the graphemic trees are of sufficient size, as demonstrated in Fig. 4.11 for a synchronisation interval of 50. The difference in accuracy can be analysed in further detail by calculating two values: δ_1, the relative increase in phoneme error rate when utilising the

[14]The alignments used were obtained from a 173,873-word training dictionary.

incremental mode compared to the batch mode, and δ_2, the relative decrease in phoneme error rate when utilising the incremental mode, in comparison with only performing updates at synchronisation events and not updating the models in between; that is,

$$\delta_1(x) = \frac{inc(x) - batch(x)}{1 - batch(x)} * 100 \tag{4.5}$$

$$\delta_2(x) = \frac{inc(x) - batch(x-1)}{1 - batch(x-1)} * 100 \tag{4.6}$$

and where $batch(x)$ indicates the phoneme accuracy using batch rule extraction, and $inc(x)$ the phoneme accuracy using incremental rule extraction, both at synchronisation point x. Fig. 4.12 illustrates the trends for the δ_1 and δ_2 values for an update interval of 50 (still utilising ideal alignments), providing an additional perspective on the same data as displayed in Fig. 4.11.

The effect on rule set accuracy is strongly influenced by the length of the update interval. We therefore compare the performance of the two algorithms for different update intervals, and find that the average δ_1 and δ_2 values are both fairly linear in relation to the update interval: the longer the interval, the less accurate incremental updating becomes when compared with batch updating, and the more value is provided by incremental updating vs performing no updates in between synchronisation events. In Fig 4.13 we plot the δ_1 and δ_2 values for update intervals of length $50, 100, 150$ and 200 during the first 4500 words of bootstrapping. These trends continue for larger update intervals.

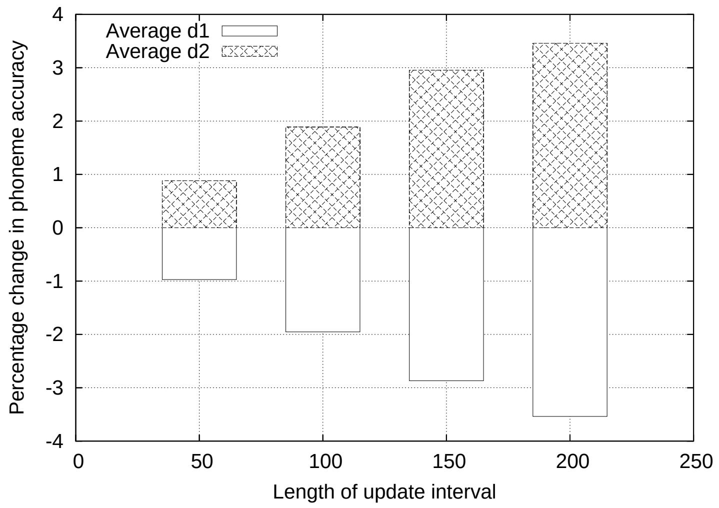


Figure 4.13: *Average δ_1 and δ_2 values for update intervals of length 50,100,150 and 200.*

Finally, in order to ensure that the fast alignment process does not introduce a noticeable loss in accuracy, we compare the two algorithms (batch and incremental rule extraction), applying the alignment process as it would be used in practise: performing a full alignment during synchronisation events and using the fast alignment process in between. We find that while there is a greater variance in the effect on phoneme accuracy when using the fast alignment process during the first phase of bootstrapping, this effect becomes negligible during the second phase of bootstrapping. (In practice, fast alignment is only used during the second stage of bootstrapping.) In Fig 4.14 we plot the δ_1

values for an update interval of 50, when using ideal alignments and actual alignments.

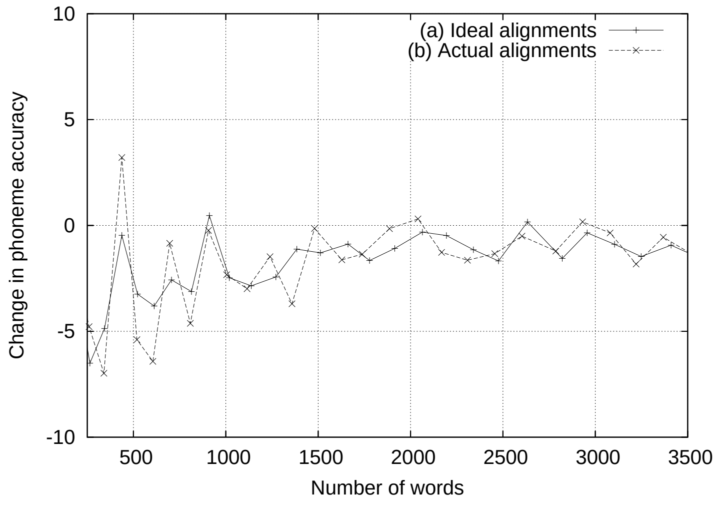


Figure 4.14: *Change in phoneme accuracy (δ_1) when comparing incremental with batch mode when (a) ideal alignments are used, and (b) when actual alignments are used.*

The above results indicate that incremental *Default&Refine* provides an effective way of increasing system responsiveness. As there is a clear trade-off between the length of an update interval and learning efficiency, the update interval can be chosen in a way that is suitable for the specific dictionary developer: longer continuous sessions (requiring slightly more corrections), or shorter sessions with frequent breaks. As the dictionary size increases and the rule set approaches asymptotic accuracy, the number of words considered between synchronisation events increases automatically[15]. For large dictionaries, the batch update process can become a daily event, rather than an hourly event, as would be the case for relatively small dictionaries. Also, as the user interface requires little processing capacity, the batch update may be scheduled to occur in the background during incremental verification, transparent to the user[16].

4.7 BOOTSTRAPPING ANALYSIS

In this section we summarise the characteristics of *DEC-grow, DEC-min* and *Default&Refine* according to the four main requirements for bootstrapping, as described in Section 4.2: predictive ability, conversion accuracy, computational cost and robustness to noise.

4.7.1 PREDICTIVE ABILITY

In Fig. 4.15 we compare the accuracy of the three algorithms for small training sets, using the *FONILEX* corpus. The *Default&Refine* algorithm performs particularly well, achieving 90%

[15]For example, using an update interval of 50, approximately 200 training words are considered per session when just past the 4000-word mark. (See Fig. 4.12.)

[16]This approach was not implemented.

phoneme accuracy prior to the 500 word-mark. *DEC-grow* requires an additional 800 words before the same level of accuracy is reached. Since the correction of incorrectly predicted phonemes is the most labour-intensive aspect of bootstrapping pronunciation dictionaries (as discussed in Section 6.3.2.4) this represents a significant improvement to the process.

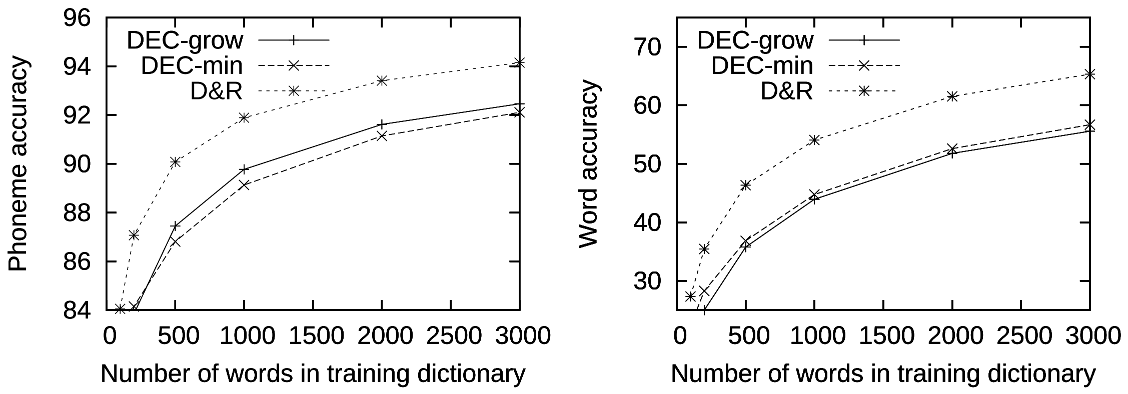


Figure 4.15: *Accuracy comparison during initial 5000 words of training, as measured using the FONILEX corpus.*

From a bootstrapping perspective, asymptotic accuracy is not as important, unless very large dictionaries are built. Asymptotic accuracies for different languages are compared for the dictionaries listed in Table 4.11. Per dictionary the number of words in total (*size*) and number of distinct words (*distinct*) are indicated. Word accuracy is listed in Table 4.12 and phoneme accuracy in Table 4.13, as analysed during 10-fold cross-validation of the dictionaries.

Table 4.11: *Dictionaries used for accuracy analysis*

Language	Dictionary	Size	Distinct
Afrikaans	Afrikaans B	7,782	7,782
English	NETtalk	20,008	19,802
English	OALD (no SA)	60,399	59,835
Flemish	FONILEX	173,873	163,526

Table 4.12: *Word accuracy of g-to-p algorithms for larger dictionaries in different languages.*

Dictionary	DEC-grow		DEC-min		Default&Refine	
		$\pm\sigma_{10}$		$\pm\sigma_{10}$		$\pm\sigma_{10}$
Afrikaans B	79.08	0.44	79.90	0.51	84.82	0.29
NETtalk	47.82	0.41	47.61	0.35	58.66	0.21
OALD (excl SA)	77.62	0.17	79.98	0.17	86.41	0.15
FONILEX	88.60	0.07	89.58	0.06	92.03	0.06

4.7.2 CONVERSION ACCURACY

All the studied algorithms are memory-based and provide complete retrieval of training data: the entire training dictionary can be reconstructed from the grapheme-to-phoneme rule set without any

Table 4.13: *Phoneme accuracy of g-to-p algorithms for larger dictionaries in different languages.*

Dictionary	DEC-grow		DEC-min		Default&Refine	
		$\pm\sigma_{10}$		$\pm\sigma_{10}$		$\pm\sigma_{10}$
Afrikaans B	95.96	0.09	95.98	0.14	97.08	0.08
NETtalk	87.82	0.11	87.20	0.08	90.50	0.10
OALD (no SA)	95.85	0.04	96.08	0.04	97.41	0.03
FONILEX	98.32	0.04	98.41	0.01	98.78	0.01

loss of accuracy.

4.7.3 COMPUTATIONAL COST

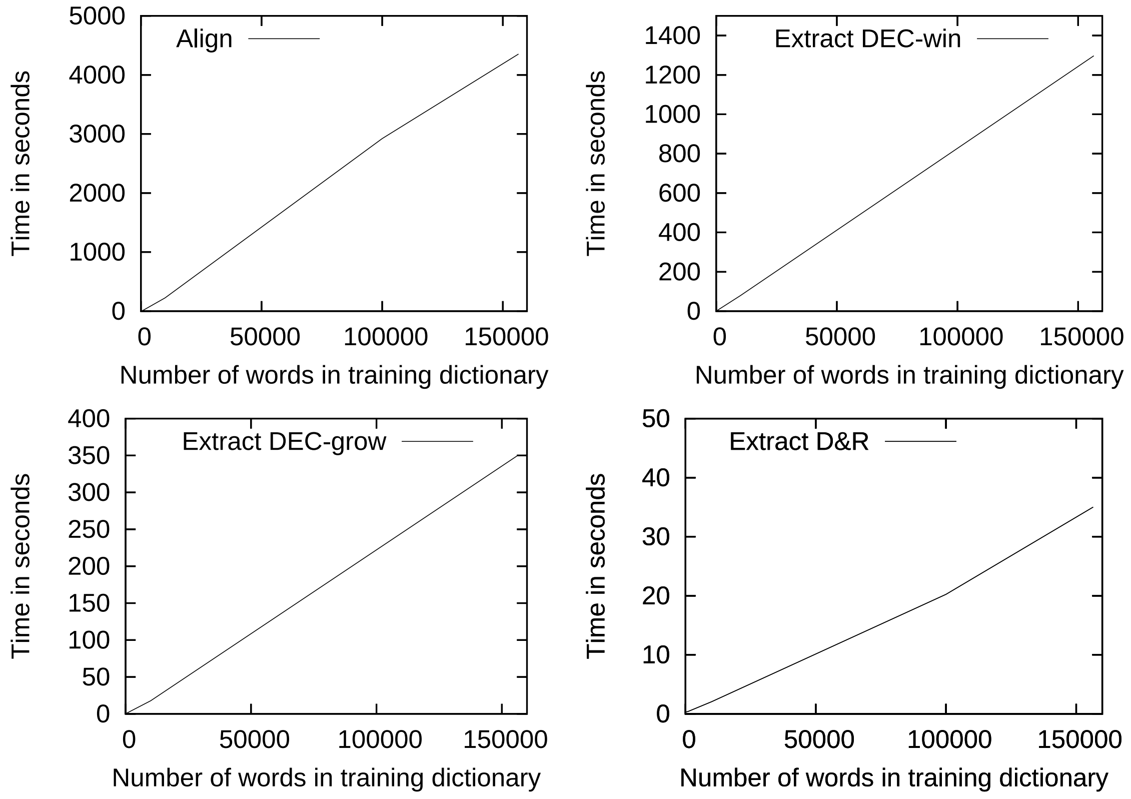


Figure 4.16: *Time required for alignment and extraction of initial patterns from different sized training dictionaries, measured using the FONILEX corpus.*

The computational cost of the various algorithms results from four separate processes:

1. *Grapheme-to-phoneme alignment:* Aligning words on a grapheme-to-phoneme basis. An identical grapheme-to-phoneme alignment process is used for all of the algorithms. The computational cost of alignment is influenced by the number of times the full dictionary is processed before the alignment probabilities stabilise. As this is typically a small number, alignment is approximately $O(n)$ where n indicates the number of words in the training dictionary. As the probabilities stabilise more quickly when more training data is available, alignment can exhibit better than linear dependency in practice.

2. *Extracting initial patterns prior to rule set extraction:* In the current implementation, the dictionary is read once, all required patterns are extracted and separated according to grapheme. Further rule extraction utilises the per-grapheme pattern sets as input. This process is again $O(n)$ for all the algorithms.

3. *Rule extraction:* Extracting a specific rule set from a grapheme-specific pattern set. For the implementations of *DEC-grow* and *Default&Refine*, rule extraction may require as many as

$$n + (n-1) + (n-2)... \sim \frac{n(n-1)}{2} \tag{4.7}$$

steps, which results in $O(n^2)$ behaviour. This would be the case for a dictionary that is conflicted to the extent that every single word gives rise to a separate rule. However, in practise, the number of steps required is closer to

$$n + k.n + k^2.n + ... \sim \frac{n}{1-k} \tag{4.8}$$

where $0 \leq k < 1$ provides some indication of the pronunciation conflict for the specific language (and dictionary) being considered. The more exceptions in the dictionary, the higher k, and the higher the complexity of rule extraction. In practice, rule extraction therefore displays $O(n)$ behaviour.

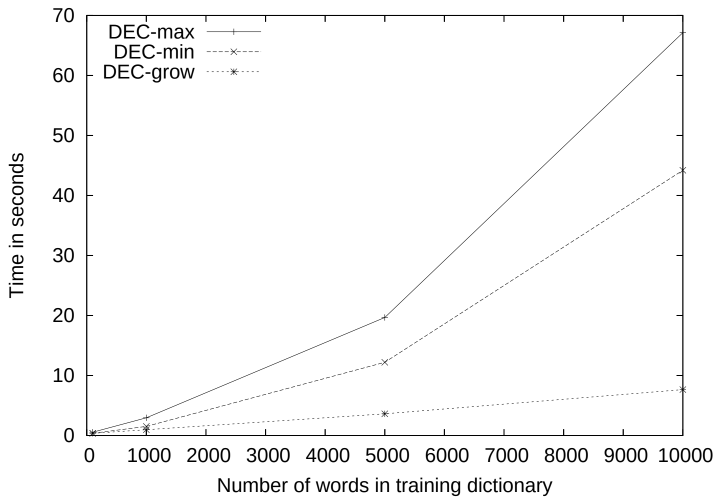


Figure 4.17: *Time required to extract DEC-grow, DEC-max and DEC-min rules from different sized training dictionaries, measured using the FONILEX corpus.*

4. *Pronunciation prediction:* Predicting the pronunciation of a single word based on an existing rule set. Pronunciation prediction is efficient for all the algorithms studied. For each type of rule extraction, the ensuing rule set can be arranged in an efficient tree structure. Pronunciation prediction is of $O(d.l)$ where l indicates the length of the word predicted, and d again represents the average depth of the various graphemic rule trees (which is approximately equivalent

to the average context size of the graphemic rule set, as described in Section 4.6.4). Our implementation of *DEC-max* and *DEC-min* exhibit worse than linear dependency, as depicted in Fig. 4.17.

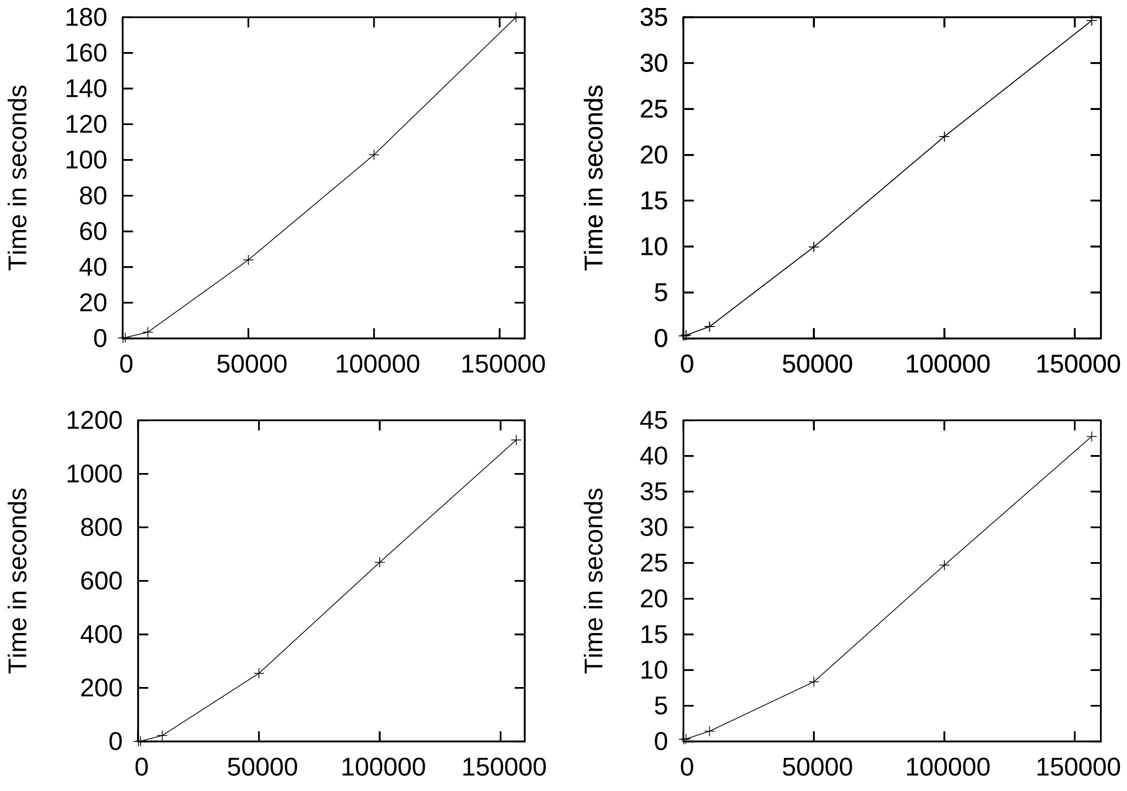


Figure 4.18: *Time required to extract Default&Refine rules for different phonemes from different sized training dictionaries, measured using the FONILEX corpus.*

These trends are further illustrated in Fig. 4.16 4.18 and 4.17. Execution time for alignment, pattern extraction and rule extraction is plotted for a training dictionary as it increases in size. These values were measured on a 1600 MHz Intel Pentium 4 personal computer with 1 GB memory, using the initial *Perl* prototype used during experimentation (System A). In comparison, equivalent algorithms are much faster as implemented in System B, a more robust version of the initial prototype[17], as listed in Table 4.14.

[17]These systems are described in more detail in Chapter 6. System A was developed in *Perl* by the author, and used during algorithm development and experimentation; System B was re-implemented in *Java* (without any algorithmic changes) by members of the CSIR HLT Research Group. The second system was used to build a medium-sized dictionary, as described in Section 6.5.

Table 4.14: *A comparison of computation times in seconds for alignment and Default&Refine rule extraction for two different implementations of the bootstrapping system.*

Task	System A	System B
Alignment: 10,000 words	185.32	8.15
Alignment: 50,000 words	793.23	49.08
Default&Refine: 10,000 words	272.40	26.13
Default&Refine: 50,000 words	1335.36	320.06

4.7.4 ROBUSTNESS TO NOISE

In order to analyse the effect of errors on predictive accuracy, we conduct a number of simulation experiments, using *Afrikaans A*, one of a set of Afrikaans bootstrapped dictionaries, as described in Section 4.3. Based on earlier experience with dictionary developers who are more error prone (see Section 6.3.2.2), we artificially corrupt a fraction of these transcriptions and then measure the predictive accuracy of *Default&Refine* on the corrupted databases.

We introduce two types of corruptions into the transcriptions:

- *Systematic corruptions* reflect the fact that users are prone to making certain transcription errors - for example, in the ARPAbet phone set, *ay* is often used where *ey* is intended. We allow a number of such substitutions, to reflect observed confusions by Afrikaans transcribers.
- *Random corruptions* simulate the less systematic errors that also occur in practice; in our simulations, random insertions, substitutions and deletions of phonemes are introduced.

We generate four corrupted data sets (systematic substitutions and random insertions, substitutions and deletions), where 1%, 2%, 5% and 10% of the words are randomly selected for corruption. We generate *Default&Refine* and *DEC-grow* rule sets with 90% of the words of each (corrupted) dictionary and measure the accuracy of the rules against the remaining 10% (using the original uncorrupted dictionary), and perform 10-fold cross-validation.

The effect of the simulated errors on predictive accuracy is depicted below. In Figure 4.19 the average word accuracy and phoneme accuracy are plotted against the percentage of corrupted words for *DEC-grow* and *Default&Refine*. Note that the most significant effect is due to insertions, as unnecessary insertions cause superfluous graphemic nulls, which introduce alignment errors. This effect is visible for both *DEC-grow* and *Default&Refine*, as both rely on accurate pre-alignments. Figure 4.20 provides a more detailed analysis: the change in average word accuracy and phoneme accuracy is plotted in the same way as above. Here it can be seen that deletions and substitutions affect the predictive accuracy to a similar extent, whether random or systematic. This behaviour is quite different to the behaviour observed later (see Section 6.4), when the position of rules in the extracted rule set is used to predict errors in the training data. As no rules are discarded during standard *Default&Refine*, rule set position does not affect predictive accuracy. Both rule extraction techniques perform well in the presence of low levels of noise, with *Default&Refine* providing a slight advantage over *DEC-grow*.

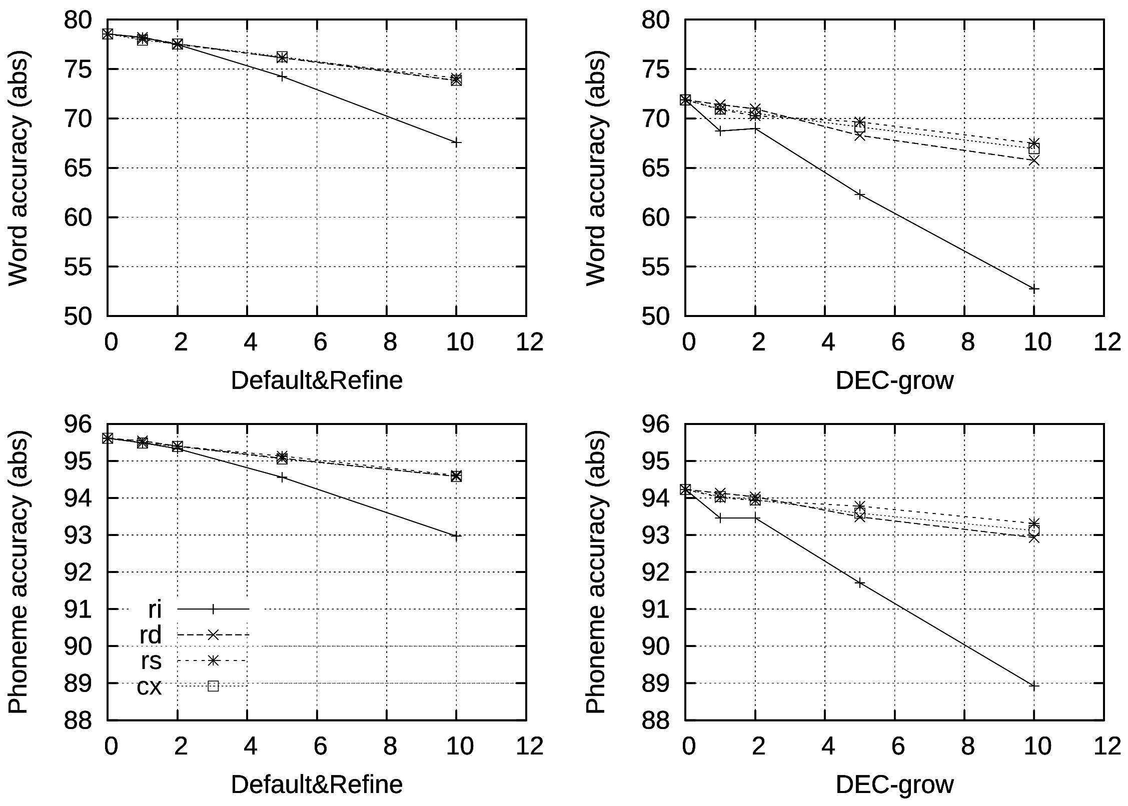


Figure 4.19: *Effect of noise on average phoneme and word accuracy when extracting rules from a corrupted version of the Afrikaans A database. Databases are corrupted with random insertions(ri), random deletions (rd), random substitutions (rs) and systematic substitutions (cx).*

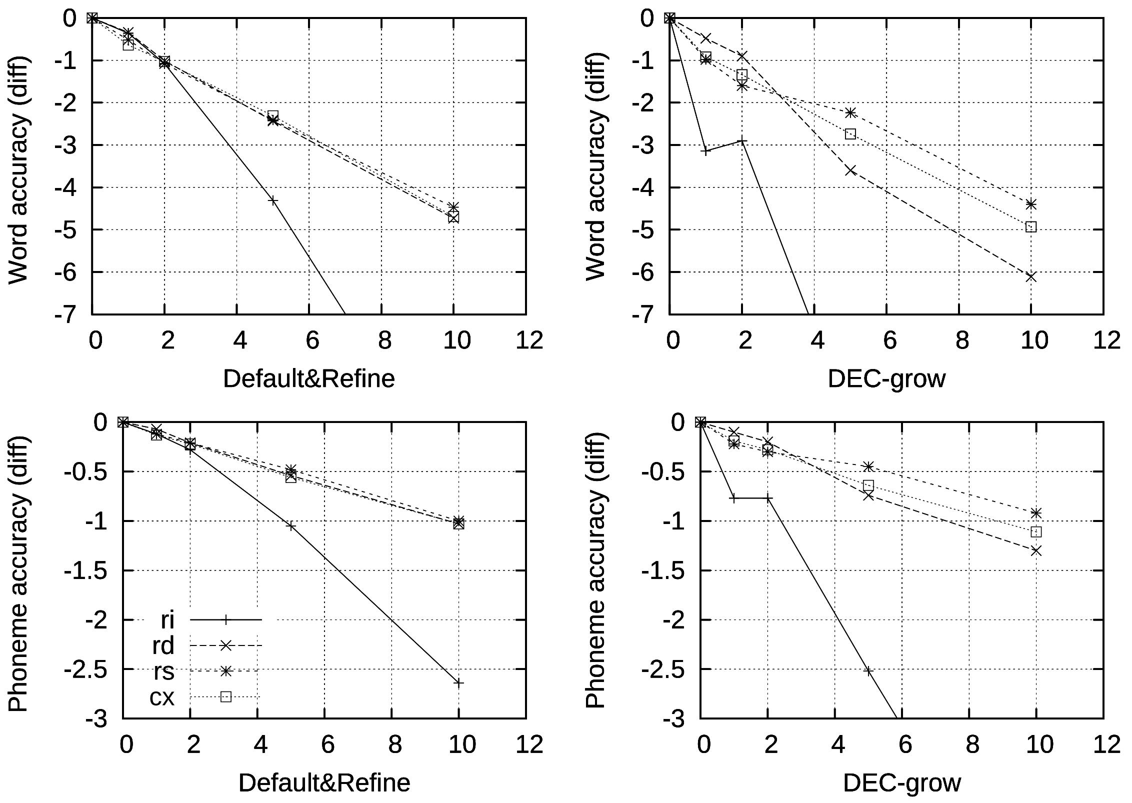


Figure 4.20: *Effect of noise on change in average phoneme and word accuracy when extracting rules from a corrupted version of the Afrikaans A database. Databases are corrupted with random insertions(ri), random deletions (rd), random substitutions (rs) and systematic substitutions (cx).*

4.8 CONCLUSION

In this chapter we analysed the grapheme-to-phoneme conversion task through a set of experiments based on variations of Dynamically Expanding Context (DEC). We proposed an enhancement to the standard approach for grapheme-to-phoneme alignment and defined a new grapheme-to-phoneme conversion algorithm (*Default & Refine*). This algorithm utilises the concept of a default phoneme to extract a cascade of increasingly more specialised rules, and has a number of attractive properties including language independence, rapid learning, good asymptotic accuracy, robustness to noise, and the production of compact rule sets. In subsequent chapters, we utilise both *DEC-min* and *Default&Refine* as grapheme-to-phoneme conversion mechanism during bootstrapping.

Table 4.6 and Figures 4.9 and 4.10 depict an interesting trend: as the rule sets that fully describe the training data become smaller and smaller, the generalisation accuracy of the rule set increases. This raises an interesting theoretical question: What is the smallest possible rule set within a rewrite rule based framework that can fully reconstruct a given set of training data with 100 % accuracy? In the next chapter (Chapter 5) we explore this question further.

CHAPTER FIVE

MINIMAL REPRESENTATION GRAPHS

5.1 INTRODUCTION

In Chapter 4 we analysed the grapheme-to-phoneme conversion task and developed an algorithm suitable for bootstrapping. During the development of this algorithm (*Default&Refine*) an interesting trend was observed: if different rule sets that all provide complete recovery of a set of training data are extracted, the smaller rule sets tend to generalise better on an unseen test set. This is not an atypical situation when addressing machine learning problems, but leads us to an interesting theoretical question: is it possible to define an algorithm that extracts the smallest possible rule set within the rewrite rule framework studied in the previous chapter, from any given set of training data? All the algorithms discussed in Chapter 4 use heuristic information to attempt to obtain such a rule set; we are interested in understanding the exact options available when attempting to obtain a minimal rule set given a set of training data.

In Section 5.2 we describe a conceptual approach that allows us to analyse the interdependencies among words in the training data in a rigorous fashion. This framework provides us both with a basis for analysing current rule extraction algorithms, and points towards a method for the extraction of a provably minimal rule set. In Section 5.3 we define the discussed framework in more detail, and demonstrate how this framework can be used to extract rule sets. In Section 5.6 we discuss the implications of our results.

5.2 CONCEPTUAL APPROACH

In this section we provide a conceptual overview of the suggested approach, referred to as *minimal representation graphs* in the remainder of this thesis. We use the same rewrite rule formalism as utilised in Chapter 4; that is, each rule describes the mapping of a single grapheme to a single phoneme using the format:

$$x_1..x_m - g - y_1..y_n \rightarrow p \tag{5.1}$$

Here g indicates the focal grapheme, x_i and y_j the graphemic context, and p the phonemic realisation of the grapheme g. The rule set is accompanied by an explicit rule application order. A pronunciation prediction for any specific word is generated one focal grapheme at a time, by applying the first matching rule found when searching through the rule set according to the rule application order. Initially we focus on a training data set that does not contain any variants, that is, every word is associated with a single unique pronunciation[1].

The goal of the approach is to obtain the smallest possible rule set that describes a set of training data completely, as an indirect approach to obtaining optimal accuracy on an unseen test set. In order to better analyse the options available when attempting to extract such a rule set, we define a framework that relies on four main observations:

1. If, for every training word, we extract all the sub-patterns of that word (as illustrated in Table 5.1), we obtain a list of all the rules that can possibly be extracted from the training data. Some of these rules will conflict with one another with regard to phonemic outcome, and we refer to these rules as *conflicted* rules. By choosing any subset of the full set of rules, and assigning a specific outcome to each rule, all possible rule sets can be generated, whether accurate in predicting the training data, or not.

Table 5.1: *The relationship between a word and its sub-pattern rules.*

Example	grapheme e to phoneme E in word 'test'
Word pattern	#t-e-st# → E
Sub-patterns	-e- → E,-e-s → E,t-e- → E,t-e-s → E t-e-st → E, #t-e-s → E,-e-st# → E #t-e-st → E,t-e-st# → E,#t-e-st# → E

2. If all the orderings among the full set of possible rules (say Z) that may be required by a subset of Z to be accurate in predicting the training data can be defined, then it becomes possible to construct a rule graph of the full rule set according to all the orderings possible, and to define appropriate operations that can manipulate this rule graph in well defined ways. During graph manipulation, specific outcomes can be assigned to rules and rules identified as *required* or superfluous. Superfluous rules can consequently be deleted, until only a minimal rule set is retained.

3. During rule prediction, the relative rule application order of two rules that occur in an extracted rule set is only of importance if the two rules conflict with regard to outcome, and if both can

[1]We discuss options for dealing with pronunciation variants in Section 5.5

apply to a single word. During rule extraction, the order in which two rules occur in an interim rule set is only of importance if both can apply to a single word in the training data, and that word has not yet been 'caught' by any required rule occurring earlier in the rule set. For each rule, we refer to the latter set of words as the *possible words* associated with that rule.

4. The full set of possible rules Z cannot occur in any order. It is possible to restrict the allowable orderings between any two rules for two reasons: (1) if one rule is more specific than another, the first rule must occur earlier in the rule set than the second in any minimal rule set. If not, the second (more general) rule will always be invoked when predicting a word that applies to both rules, and the first rule will be redundant (which is impossible if the rule set is minimal); and (2) if two rules are applicable to the same word in the training data but conflict with regard to outcome. For such rules the words shared in the *possible words* sets of each rule dictate the orderings that are valid.

Using the above observations, we can analyse a set of training data in order to understand the interdependencies among words in the training data, and the options for extracting a minimal rule set. We illustrate the concepts using a simple 3-word example, consisting of the words 'test','ten' and 'tea' and consider the steps required to extract a rule set for the letter 'e'. As the software that we developed to implement this approach uses a single character representation of each grapheme and phoneme, we do the same in this example.

Prior to rule extraction, a *word pattern* is generated from each aligned word-pronunciation pair in the training data, as shown in Table 5.2. Hashes denote word boundaries.

Table 5.2: *Word patterns associated with the words 'test','ten' and 'tea'.*

	aligned ARPAbet example	single character representation
Words	t e s t $\rightarrow$ t eh s t t e n $\rightarrow$ t eh n t e a $\rightarrow$ t iy ϕ	t e s t $\rightarrow$ t e s t t e n $\rightarrow$ t e n t e a $\rightarrow$ t i ϕ
Word patterns	#t-e-st# $\rightarrow$ eh #t-e-n# $\rightarrow$ eh #t-e-a# $\rightarrow$ iy	#t-e-st# $\rightarrow$ e #t-e-n# $\rightarrow$ e #t-e-a# $\rightarrow$ i

For each of the word patterns, we generate a set of sub-patterns (as listed in Table 5.1 for the word pattern #t-e-st# $\rightarrow$ e). These sub-patterns are arranged in a graph structure according to specificity, with the more general rules later in the graph (closer to the root), and more specialised rule earlier (higher up in the graph). Initially, an ordering is only added between two rules where the context of one rule contains the context of another, and we refer to these orderings as *contain pattern* relationships. A topological sort of this graph will result in a rule set that is accurate, but contains a large number of superfluous rules. From the outset, the process assumes that any of the rules may be deleted in future. As it becomes clear that certain rules are required in order to retain accuracy over the training data (irrespective of further allowed changes to the rule set), these rules are marked as *required* rules.

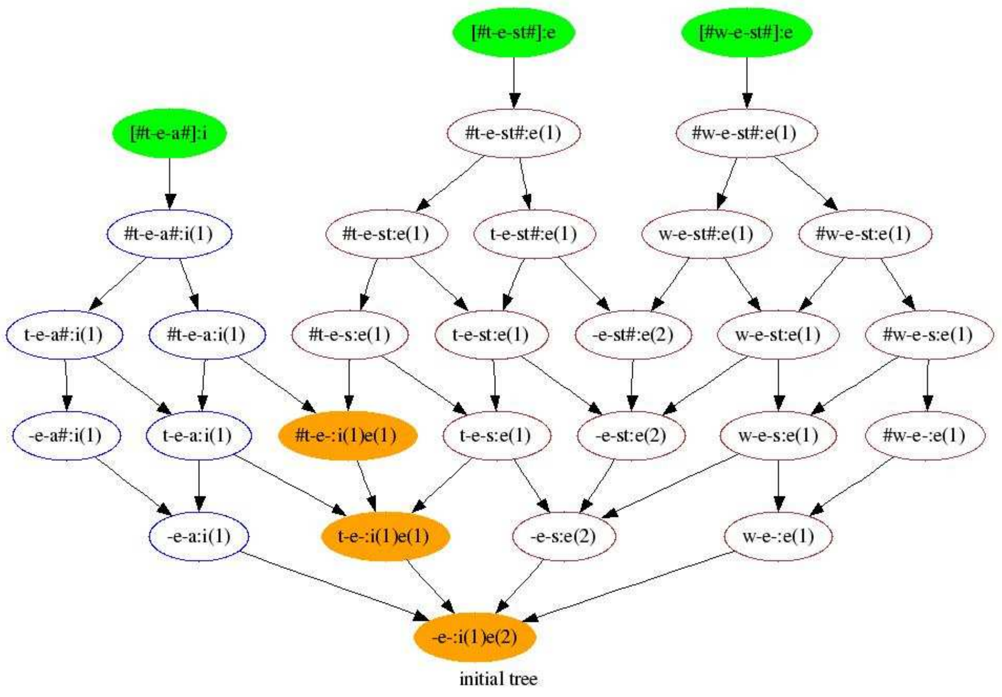


Figure 5.1: *An example rule graph, corresponding to the word patterns in Table 5.2*

This process is illustrated in Fig. 5.1. Word nodes (one per word pattern) are indicated in green. Clear nodes indicate rule nodes that can only predict a single outcome. For these nodes, different coloured outlines indicate different outcomes. Orange nodes are associated with more than one possible outcome: different choices with regard to outcome will result in different rule sets. Black edges indicate that an ordering between two rules is required, irrespective of further rule graph manipulation. In the initial graph these edges represent *contain pattern* relationships. Currently no rules are marked as required; if there were, these would be marked in yellow.

Orderings are transitive. If all the orderings implied by the current set of edges are considered, then the only additional orderings that can possibly occur in the full rule set are between rules that share a word in their respective *possible words* set, and have not already been assigned a fixed ordering. We refer to these rules as *minimal complements*. These *minimal complement* relationships are added and utilised during rule extraction. We do not indicate them explicitly on all the graphs used to illustrate the current example, as the addition of minimal complement relationships results in visually complex graphs. For illustration, we mark the minimal complements related to a single rule '-e-st' for the initial graph of Fig. 5.1 and display the result in Fig. 5.2. Minimal complement relationships are marked as orange edges.

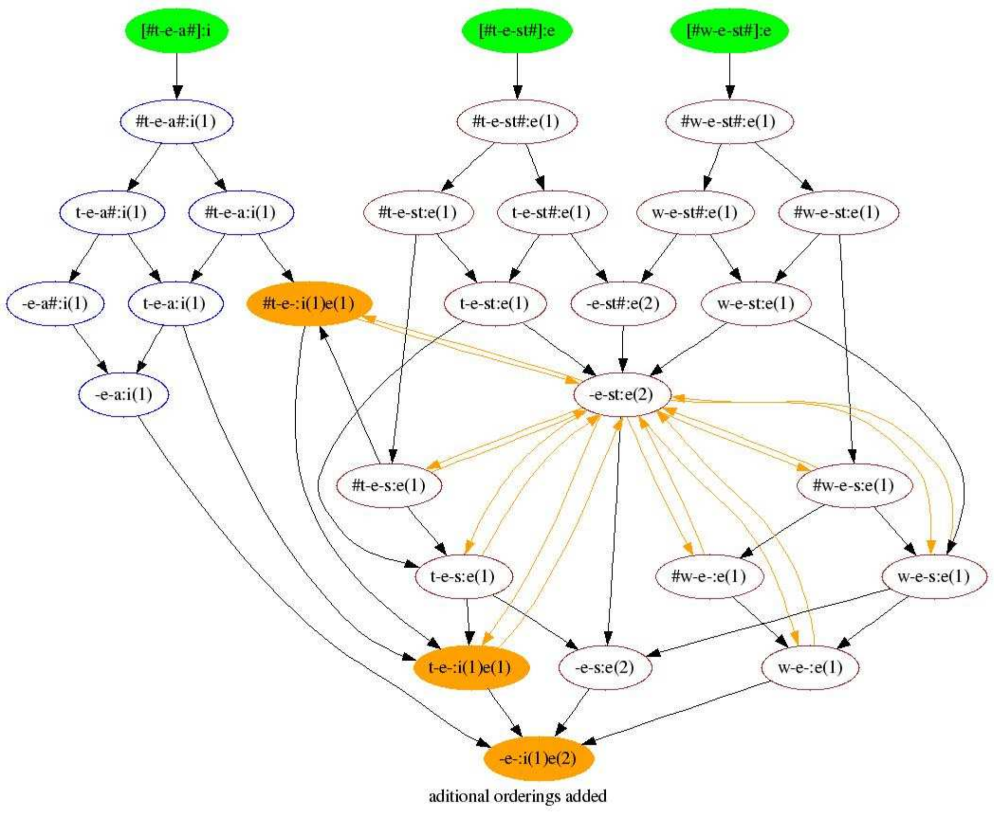


Figure 5.2: *Marking the minimal complement relationships associated with the rule '-e-st' for the rule graph of Fig. 5.1.*

Note that the minimal complements associated with any rule r can only occur in a restricted range: the context of the earliest rule may not contain rule r, and the context of the latest rule may not be contained by r itself. As this range is restricted, the number of additional orderings that may be required is similarly restricted. Each additional minimal complement pair added to the graph introduces two possible orderings. This increases the number of options to consider when making any single decision (whether to resolve a conflicted node to a single outcome, or whether a specific rule is required or can be deleted.) We would like to remove as many of the 'double orderings' as possible, and replace these with orderings that indicate a single direction. In some cases additional information is available to choose one of the orderings and discard the other:

- If the possible words associated with a rule r is a subset of the possible words of a second rule s, rule r must always occur earlier in the rule extraction order than s. The reasoning is similar to that followed when adding the initial contain pattern orderings, but now holds for minimal complements that are not necessarily in a contain pattern relationship. We refer to these relationships as *super complements*. While contain pattern relationships can be added to the graph from the outset, *super complements* emerge as the rule set extraction process progresses. As more rules are marked as $required$, the possible words sets of later rules decrease, and super complement relationships start to emerge. Once an ordering is added between two super complements, this relationship is not changed at a later stage during rule manipulation[2]

- If a rule r predicts a single outcome, and accurately matches all the words in the intersection of the possible words of rule r and the possible words of another rule s, and there is at least one word in this set that s will mispredict given any of its allowed outcomes, then rule r has to occur before rule s for the rule set to be accurate. We refer to these relationships as *order required* relationships. If neither of the two rules matches the full set of shared words, the relationship is still inconclusive. As with super complement relationships, order required relationships also emerge as the rule set extraction process progresses.

In Fig. 5.3 we identify and add additional super complement relationships. The current rule graph does not have any order required relationships among nodes.

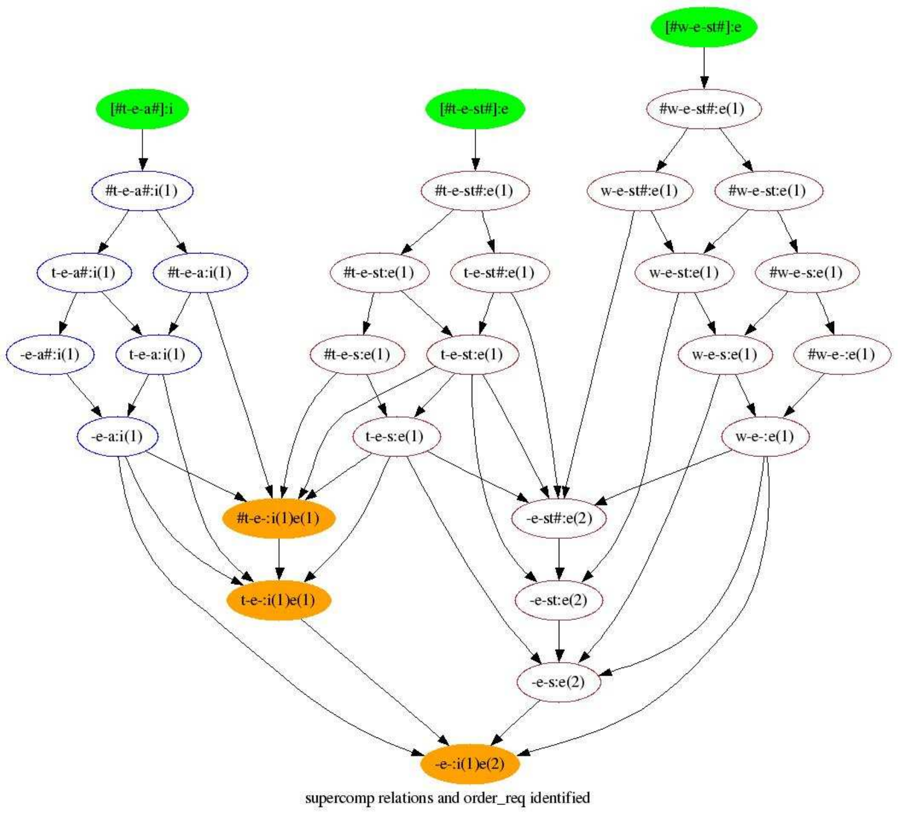


Figure 5.3: *Adding super complements to the rule graph of Fig. 5.1. (Minimal complements are not shown.)*

Since orderings are transitive, we can remove any definite orderings that are already implied by others. For example, in Fig. 5.3 the relationship between rules ‘t-e-st’ and ‘#t-e- is already implied

[2]As more rules are marked as required, the possible words sets of all other rules become smaller. If a set of possible words associated with a rule r is the subset of the possible words associated with a rule s, this relationship will be maintained unless both sets become equal. In the latter case, one of the two rules are redundant and will be deleted during rule extraction, as discussed later. Since either r or s will be deleted, the ordering between these two rules become insignificant, and the prior ordering based on their previous super complement relationship may be retained without restricting rule graph manipulation options.

by the relationships between rules ‘t-e-st’ and ‘t-e-s’, and between rules ‘t-e-s’ and ‘#t-e-’. Such redundant edges can be removed without losing any information currently captured in the rule graph.

This process is illustrated in Fig. 5.4. Note how the relationships become simpler and the graph more loosely connected from Fig. 5.1 to Fig. 5.4.

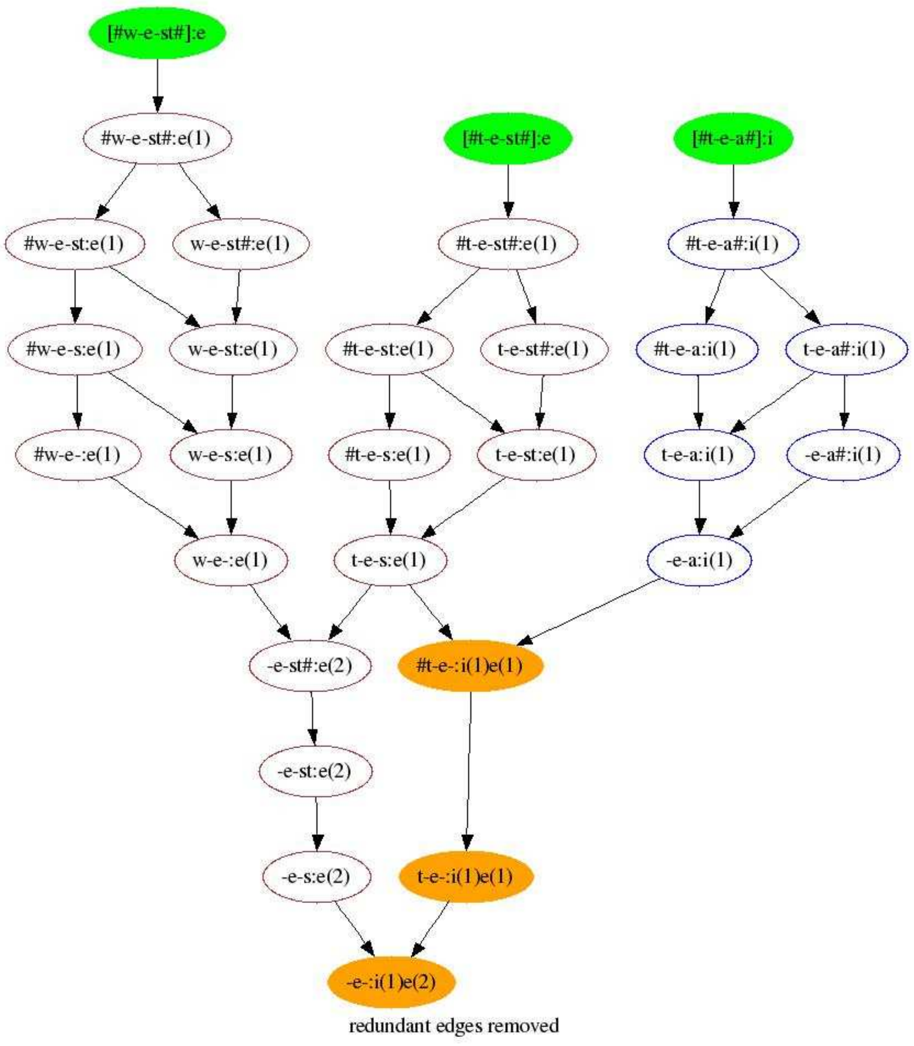


Figure 5.4: *Removing unnecessary edges from the rule graph of Fig. 5.3. (Minimal complements are not shown.)*

If we can be sure that we have added all the necessary orderings (caused by contain pattern, super complement or order required relationships) and we keep track of all minimal complement relationships that still have an uncertain ordering, we now have a rule graph that both contains all possible rules, and specifies all possible orderings that may be required to define a valid rule application order.

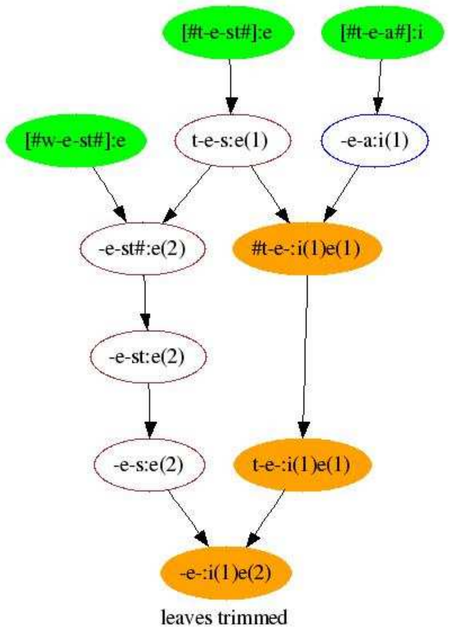


Figure 5.5: *Removing unnecessary rules from the rule graph of Fig. 5.4.(Minimal complements are not shown.)*

We can now use this rule graph as basis to make decisions about which outcome to select where a rule is conflicted (has more than one outcome), or even decide when a rule can be deleted or not.

When rules are deleted, it is possible that one of the rules required by a minimal rule set is deleted unintentionally, and in order to compensate for this deletion, two or more additional rules may have to be kept to retain accuracy over the training data. The final rule set will then have more rules than strictly required. To prevent this from happening, rules are eliminated by deleting redundant rules, identifying required rules and resolving conflict rules via a small set of allowed operations. The $state$ of rule extraction can always be described by a triple consisting of the possible rules that can be included in the rule set (Z'), the rules that have been marked as required (Z_e), and the orderings that are definite ($oset(Z')$, the black edges in the graph). Additional orderings that are possible can automatically be generated from such a state. Each allowed operation changes the state of rule extraction, from one *allowed state* to another, with the initial allowed state as depicted in Fig. 5.1.

One example of such an allowed deletion operation can be illustrated as follows: The rule graph in Fig. 5.4 clearly contains a number of superfluous rules. Whenever a rule r exists such that (1) it is not conflicted, and (2) all the possible words associated with rule r can be caught by one or more immediate successors that agree with rule r with regard to outcome, and (3) rule r does not have any immediate successors that can potentially disagree with regard to outcome, then rule r can safely be deleted from the rule graph. All rules that meet these conditions, can be deleted from the rule graph, as illustrated in Fig. 5.5. Since the rule graph is now significantly simpler, we start displaying the remaining minimal complements from Fig. 5.6 onwards.

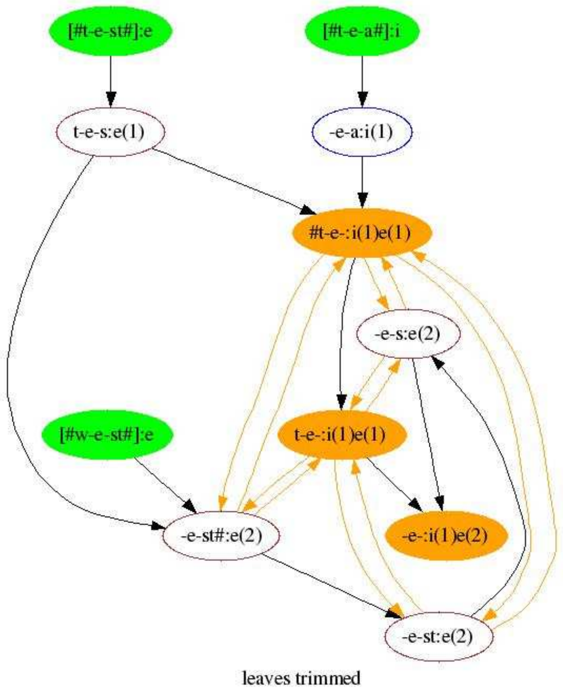


Figure 5.6: *Removing unnecessary rules from the rule graph of Fig. 5.4.(Minimal complements are shown.)*

Where the possible words associated with rule r are exactly the same as the possible words of any one of its successors s, rule r and rule s are deemed *rule variants*. Either of two rule variants can be generated at the same point in the rule extraction order, without influencing the number of rules in the final rule set. The process keeps track of all deleted rules that are variants of retained rules. In this way, while a rule node is physically deleted, the rules are in effect merged, and either of the two rules may be utilised in the final rule set, as discussed later.

Additional deletion operations identify rules that have an empty set of possible words, and rules that are true variants of another, that is, two rules that are both resolved to a single outcome, and have identical relationships with identical predecessors and successors. While these deletion operations create a rule graph that is significantly simpler, we have not yet made any decisions with regard to the best choice of outcome for any of the conflicted nodes. Prior to rule resolution, we first identify any rule r as *single* where – given the current state of rule extraction – at least one word can only be predicted by either rule r or by another rule directly in the path of r. In the remaining figures, these single rules are marked '*S'.

There are various conditions under which a conflicted rule can be resolved, one of which we illustrate here. Conflicted nodes can be thought of as 'default' or 'fallback' nodes. During pronunciation prediction, a fallback node will only be invoked if a more specialised rule is not available that matches the word being predicted. These nodes therefore only need to be retained if, in some way or another, the rule can generalise from its immediate predecessors. This requires that at least two predecessors should predict a similar outcome. If this is not the case, the fallback node does not provide any further advantage, and can be removed from the rule graph without constraining the rule set in a way that does not allow final minimisation[3]. This process is illustrated in Fig. 5.7 and Fig. 5.8.

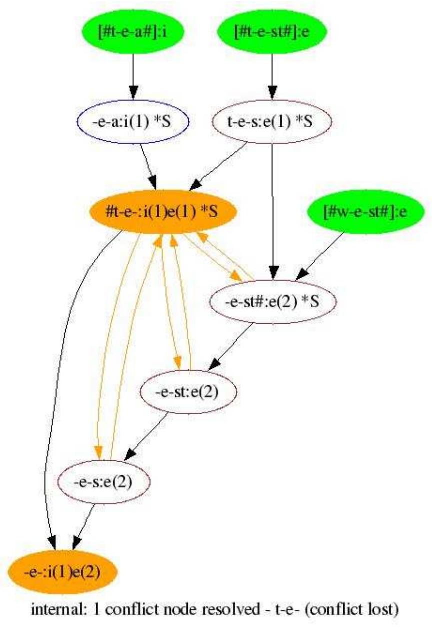


Figure 5.7: *Resolving conflicted rule 't-e-'.*

[3]This does not apply to the root node. The root node is handled as a special case, as discussed below.

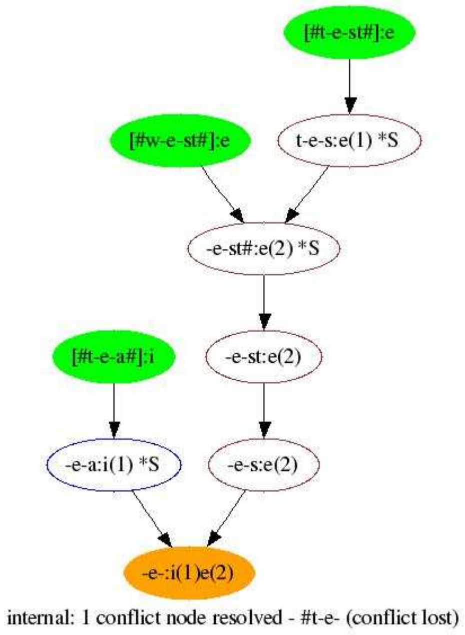


Figure 5.8: *Resolving conflicted rule '#t-e-'.*

Note that in the Fig. 5.8, none of the minimal complement relationships have been retained. Additional resolution operations analyse the definite and possible predecessors and select a specific outcome based on this analysis. When resolving a conflicted rule to a specific outcome, it is required that at least one of the predecessors that has an outcome that matches the outcome selected for resolution must be marked as a *single* rule. If such a single rule exists, this implies that some rule with the selected outcome will be generated at this point in the rule extraction order. While there is not certainty that such a rule is required, the conflicted rule may not yet be resolved.

Applying the same deletion operator discussed earlier, three additional rule nodes can be deleted, as illustrated in Fig. 5.9.

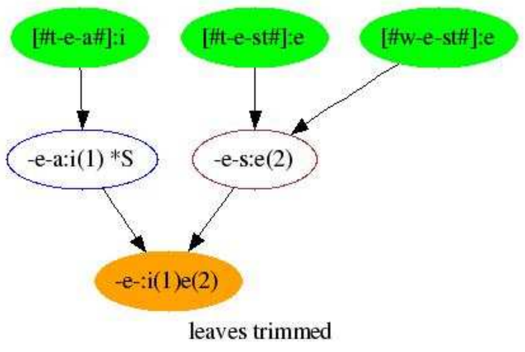


Figure 5.9: *Removing unnecessary rules '-e-st', '-e-st#' and 't-e-s'.*

If the resolution operator discussed previously were to be applied to the root node, the rule set would remain valid. However, this would result in the root node being deleted, and it is easier in practise to manipulate the graph assuming a single root node. Also, we would like to generate some ‘default rule’ that can be used to predict any word pattern not previously seen. Therefore the root node is always resolved to a single outcome, once all its predecessors are resolved (and not deleted, as would be the case if the standard resolution operator were applied). Resolving the root node to a single outcome when standard application of a deletion operator indicated that it should have been deleted, is similar to choosing one variant of a rule above another variant of the same rule. As all variants are retained during rule extraction, and the final choice with regard to which variant to choose is postponed until after graph minimisation, manipulating the root node as a special case does not restrict the rule extraction process in any way. In Fig. 5.10 the root node is resolved to one of its possible outcomes.

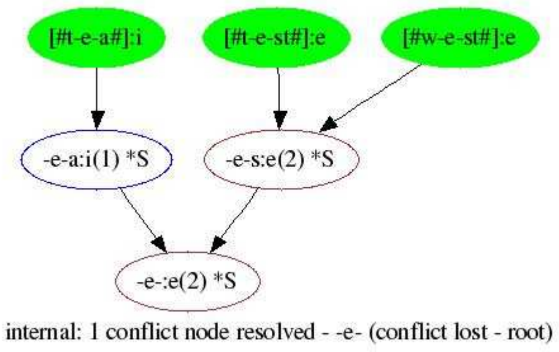


Figure 5.10: *Resolving conflicted rule ‘-e-’.*

If for at least one word pattern w in the possible words set of a rule r, there exists no other rule than can possibly predict word pattern w correctly, given the current state of rule extraction (the remaining rule set, the required rule set and the decided orderings); then rule r is a $required$ rule and can be marked as such. When a rule is identified as a required rule, all words in the possible words set of rule r are removed from the possible words sets of rules occurring later in the rule graph. In Fig 5.11 two rules are marked as required, with required nodes indicated in yellow. One final deletion (using the standard deletion operator) and the minimal rule set is obtained, as depicted in Fig. 5.12.

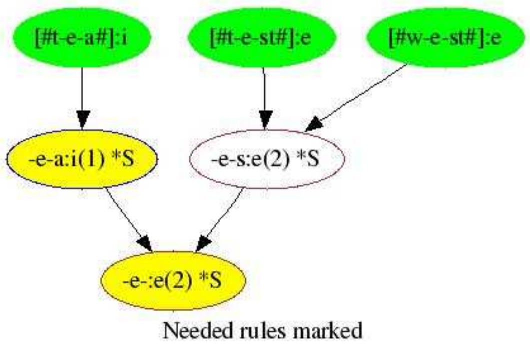


Figure 5.11: *Identifying required rules '-e-a' and '-e-'.*

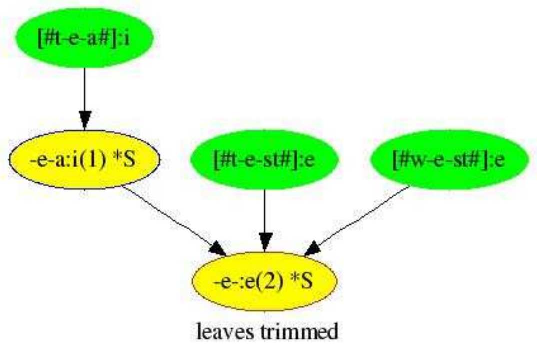


Figure 5.12: *The final (minimal) rule graph.*

The rule set that can now be extracted from the rule graph by performing a topological graph traversal. This results in the rule set listed in Table 5.3. For each extracted rule, a number of possible variants are listed. A rule can be replaced by any of its variants without affecting the accuracy of the rule set, or requiring the inclusion of additional rules. Note that for any single word that gives rise to a single rule (such as the word pattern #t-e-a# in this example), all word sub-patterns that have not been identified as currently part of the rule set are included as variants.

Table 5.3: *The final rule set generated from the words in Table 5.2, including possible variants.*

Rule number	Extracted rule	Possible variants
1	-e-a $\rightarrow$ i	#t-e-a #t-e-a# -e-a# t-e-a# t-e-a
2	-e- $\rightarrow$ e	-e-st# -e-s -e-st

At this stage, heuristic choices related to characteristics such as rule context size, rule context symmetry, or variance with regard to the training data can be utilised to choose the most appropriate rule set. In larger rule sets, many rules do not have variants, but a relatively large proportion of rules retain one or more variants. The ability to make heuristic choices late in the rule extraction process, provides significant flexibility in obtaining the appropriate rule set.

5.3 THEORETICAL FRAMEWORK

In this section we describe the above framework in more detail, and provide a more rigorous definition of the terminology used[4]. We provide proofs for the key statements in Appendix B. When we refer to a specific statement in the text, we are referring to the statement as found in Appendix B.

Firstly, we define the rule format and the various terms used during rule set analysis. We then proceed to show how a relationship between two rules in a minimal rule set translates to a specific relationship between the same two rules in a larger rule set, and describe the conditions and implications of a rule ordering between two rules occurring in either of these types of rule sets. Using these conditions, we provide a formal definition of an allowed state of rule extraction. We analyse the characteristics of an allowed state and define an initial state that can be shown to be allowable in these terms. We then define the various allowed operations that, when applied, progress the rule graph from one allowed state to another. In contrast to the overall framework, the set of allowed operations are still somewhat experimental, as discussed in section 5.3.6. Finally, we discuss the minimality of the extracted rule set and describe additional options for the improvement of generalisation ability.

5.3.1 RULE FORMAT

As discussed in Section 5.2, we use a set of rewrite rules to describe the mapping of a single grapheme to a single phoneme.

If G is the set of possible graphemes and H the set of possible phonemes; the i^{th} rule for grapheme g is formulated as

$$\begin{aligned} rule(g,i) &= (x_1..x_m, g, y_1..y_n) \rightarrow z; \\ x_1..x_m, g, y_1..y_n &\in G; z \in H; \\ &\text{alternatively written as:} \\ x_1..x_m - g &- y_1..y_n \rightarrow z \end{aligned} \tag{5.2}$$

where $x_1..x_m$ defines the m-grapheme left context of g, $y_1..y_n$ defines the n-grapheme right context of g, and z is the predicted phonemic realisation of grapheme g when found within the given left and right word contexts. G includes ϕ_G, the null grapheme and $\#$ the word boundary marker (with always $g \neq \#$). H includes ϕ_H, the null phoneme.

[4]Terms and definitions are presented in definition boxes, interspersed among more general comments.

The $outcome(r)$ function describes the phonemic outcome of the rule r:

$$outcome(rule(g,i)) = outcome(x_1..x_m - g - y_1..y_n \rightarrow z) = z. \tag{5.3}$$

The $context(r)$ function describes the application context of the rule[a] r directly:

$$context(rule(g,i)) = context(x_1..x_m - g - y_1..y_n \rightarrow z) = x_1..x_m - g - y_1..y_n. \tag{5.4}$$

[a] $context(.)$ can also be applied to word patterns, as defined in eq. 5.9

The rule application order $rule\_order(Z', r, s)$ specifies the order in which any two rules r and s occurring in a rule set Z' are applied, where

$$\forall r, s \in Z' : rule\_order(Z', r, s) = 1 \implies rulenum(r) < rulenum(s) \tag{5.5}$$

and the $rulenum(r)$ function describes the rule number of a specific rule r directly:

$$rulenum(rule(g,i)) = i. \tag{5.6}$$

The $oset(Z')$ for a rule set Z' consists of the entire set of orderings specified by $rule\_order(.)$, i.e:

$$oset(Z') = closure(Z', rule\_order(.)) \tag{5.7}$$

where $rule\_order(.)$ defines the current set of orderings over Z' and $closure(.)$ consists of the transitive closure of the set of rule pairs for which a specific relation is defined, i.e.:

$$\begin{aligned} closure(Z', relation(.)) = \cup_i (r,s) \forall r, s \in Z' : \\ relation(Z', r, s) = 1 \text{ or } \exists t \in Z' : relation(Z', r, t) = 1, \\ relation(Z', t, s) = 1. \end{aligned} \tag{5.8}$$

The $rule\_order(.)$ relation restricts the $rulenum(.)$ function to a set of options, and does not necessarily specify an ordering between every two rules. If rules are applied according to the $rule\_order(.)$ relation and an ordering between two rules that both match a word is indeterminate, either of the rules can potentially be invoked. It is possible to convert from an implicit $rule\_order(.)$ to an explicit rule numbering via the $assign(.)$ function:

Let the set $assign(Z', oset(Z'))$ define all the possible rule number assignments that are valid given the specified rule set Z' and rule orderings $oset(Z')$. Per assignment, a single rule number is assigned to every rule, consistent with $oset(Z')$.

Note that for a specific value of $assign(Z', oset(Z'))$, $rulenum(r) < rulenum(s)$ does not imply that $rule\_order(Z', r, s) = 1$.

A word w consists of a sequence of graphemes in G. During pronunciation prediction of a word of length n (also counting word boundaries), we create n word patterns that each focus on a specific grapheme in the word. When focusing on grapheme i, the word pattern is described as:

$$\forall w = x_1..x_n; x_j \in G\ ; n \geq 1; 1 \leq i \leq n :$$
$$word\_pattern(w, i) = x_1..x_{i-1} - x_i - x_{i+1}..x_n. \tag{5.9}$$
$$\tag{5.10}$$

The $context(w)$ function can also be applied to word patterns, where the $context$ of a word pattern w is simply the word pattern itself.

The $match(w, r)$ function indicates that a rule r occurring in a rule set Z' can be applied to predict a word pattern w:

$$\forall r \in Z' : match(w, r) = 1 \iff context(w) \supseteq context(r). \tag{5.11}$$

The $winningrule(w, g)$ relation describes the first matching rule(s) found in rule set Z' for word pattern w with regard to grapheme g, i.e

$$\forall r \in Z' : r \in winningrule(Z', oset(Z'), w, g) \iff match(w, r) = 1,$$
$$\nexists s : match(w, s) = 1, (s, r) \in oset(Z'). \tag{5.12}$$

Rules with equivalent contexts and different outcomes are not allowed in the final rule set, i.e:

$$\forall g \in G, i, j \in N : context(rule(g,i)) = context(rule(g,j)) \implies$$
$$outcome(rule(g,i)) = outcome(rule(g,j)). \tag{5.13}$$

Conflicting rules will however exist during the interim steps of rule extraction, as discussed below.

5.3.2 RULE SET ANALYSIS

5.3.2.1 TRAINING DATA, WORD PATTERNS AND SUB-PATTERNS

The rule set is derived from a set of training data. As in the previous chapters, a data set consisting of aligned word-pronunciation pairs is used as input during rule extraction. Word patterns and word sub-patterns are extracted from this set, and form the basis for further rule set construction.

Each word-pronunciation pair consists of two sequences $x_1..x_n$ and $y_1..y_n$, where $n \geq 1, x_i \in G$ and $y_i \in H$. Let $TD(g)$ be the set of all word patterns in the training data that describe a specific grapheme g, associated with a specific phonemic outcome per word pattern. Then:

$$\forall g \in G : w \in TD(g) \iff w = x_1..x_{i-1} - g - x_{i+1}..x_n \rightarrow y_i,$$
$$\text{where } x_1..x_n \text{ and } y_1..y_n \text{ an aligned word-pronunciation pair.} \tag{5.14}$$

In the remainder of this section, assume g to simplify notation (for example let $rule(i)$ be equivalent to $rule(g,i)$ for the specific g being considered). TD does not contain word variants (multiple pronunciations of a single word), that is:

$$\nexists w_1, w_2 \in TD : context(w_1) = context(w_2) \implies$$
$$outcome(w_1) \neq outcome(w_2). \tag{5.15}$$

A word pattern is in effect the largest possible rule that describes the grapheme-to-phoneme mapping accurately. The combined left and right contexts of the word pattern therefore contains the full word, including word boundaries. For each word pattern, a set of sub-pattern rules – describing all possible sub-contexts of the word pattern – can be generated, as previously shown in Table 5.1.

Let Z be the set of all possible word patterns and sub-patterns associated with the word patterns in TD.

For any two rule sets, Z_A and Z_B, let $Z_A \subseteq Z_B$ indicate that one set is equal to or a subset of the other, both with regard to the context and outcome of rules. More specifically:

$$\begin{gathered} Z_A \subseteq Z_B \iff r \in Z_A \implies r' \in Z_B, \\ context(r) = context(r'), outcome(r) \subseteq outcome(r'). \end{gathered} \tag{5.16}$$

Let $|Z'|$ indicate the number of rules in any rule set Z', where $Z' \subseteq Z$.

Let $allset(Z')$ consist of all possible orderings in a rule set Z', whether contradictory or not:

$$\forall Z' \subseteq Z_{combined} : allset(Z') = \cup_{i,j}(v_i, v_j) \forall v_i, v_j \in Z', i \neq j. \tag{5.17}$$

A word pattern w can be referred to either as a word pattern $w \in TD$ or as a rule $w \in Z$. The set Z then consists of all possible rules that can potentially apply to the word patterns in TD.

5.3.2.2 CONFLICT RULES AND CONFLICT RESOLUTION

As the set Z consists of all possible rules that can potentially apply to the word patterns in TD, it may include a number of conflicting rules. Under certain conditions, these rules can be resolved to a specific outcome. Until a rule is resolved to a single outcome, a set of allowable outcomes is retained per rule.

Let $Z_{conflict}$ consist of all the conflicting rules in Z, that is, rules that contradict eq. 5.13. Let $Z_{no-conflict}$ be the set of remaining rules, when all conflicting rules in $Z_{conflict}$ are removed from Z, i.e

$$\begin{gathered} Z_{conflict} \cup Z_{no-conflict} = Z. \\ Z_{conflict} \cap Z_{no-conflict} = \phi. \end{gathered} \tag{5.18}$$

Define the $conflictrule(r_{\alpha 1}, r_{\alpha 2}, .., r_{\alpha n})$ for all n rules $r_{\alpha i} \in Z_{conflict}$ with equivalent contexts as one rule with one of n alternative outcomes, i.e:

$$\begin{aligned} \forall r_{\alpha i} \in Z_{conflict}, context(r_{\alpha i}) = context(r_\alpha) \forall i = 1...n : \\ conflictrule(r_{\alpha 1}, r_{\alpha 2}, .., r_{\alpha n}) = context(r_\alpha) \rightarrow z_1 \| z_2 \| ... \| z_n, \\ z_j = outcome(r_{\alpha j}) \forall j = 1...n, \\ \text{where } z_j || z_k \text{ indicate that either } z_j \text{ or } z_k \text{ is a possible } outcome. \end{aligned} \tag{5.19}$$

Define the resolution of a $conflictrule$ as a specific-outcome version of the rule, i.e let:

$$\begin{gathered} \forall r_\alpha \in Z_{conflict}, z_x \in \cup_{r_{\alpha i}} outcome(rule(r_{\alpha i})) : \\ resolve(conflictrule(r_{\alpha 1}, r_{\alpha 2}, .., r_{\alpha n}), z_x) = context(r_\alpha) \rightarrow z_x. \end{gathered} \tag{5.20}$$

If a rule r with the same context is referred to with regard to different rule sets in which different resolved versions of the rule may occur, let $outcome(r|Z')$ indicate $outcome(r)$ where $r \in Z'$.

For each subset of all elements in $Z_{conflict}$ with equivalent contexts, it is possible to generate a single $conflictrule$. Let the set $Z_{conflict-combined}$ consist of all the conflict rules generated from $Z_{conflict}$ according to eq. 5.19, which have not been resolved. Let the set $Z_{conflict-resolved}$ consist of all the resolved conflict rules, where a conflict rule will move from $Z_{conflict-combined}$ to $Z_{conflict-resolved}$ upon resolution (according to eq. 5.20). Let $Z_{combined}$ consist of all elements in $Z_{no-conflict}$ combined with the elements in $Z_{conflict-combined}$ and $Z_{conflict-resolved}$, where

$$\begin{aligned}
&Z_{no-conflict} \cup Z_{conflict-resolved} \cup Z_{conflict-combined} = Z_{combined} \\
&Z_{conflict-combined} \cap Z_{no-conflict} = \phi \\
&Z_{conflict-resolved} \cap Z_{conflict-combined} = \phi \\
&Z_{conflict-resolved} \cap Z_{no-conflict} = \phi \qquad (5.21)
\end{aligned}$$

and let $$Z_{single} = Z_{conflict-resolved} \cup Z_{no-conflict} \qquad (5.22)$$

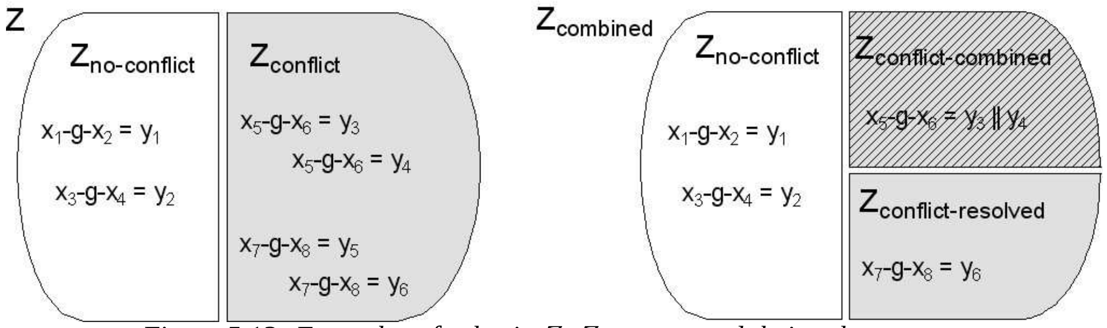


Figure 5.13: *Examples of rules in Z, $Z_{combined}$ and their subsets.*

The relationships among the different sets are depicted in Fig. 5.13.

5.3.2.3 COMPLETE, ACCURATE, MINIMAL AND POSSIBLY_MINIMAL RULE SETS

Any subset of $Z_{combined}$, ordered according to a specific rule ordering $rule\_order(.)$ will describe the training data with a certain degree of accuracy. The ideal rule set will be one that is not only complete but also accurate, and not only accurate but also minimal, as defined below:

A $complete$ rule set can predict all the words in the training data:

$$\begin{aligned}&\forall Z' \subseteq Z_{combined} : complete(Z') = 1 \iff \\ &\quad \forall w \in TD, \exists r \in Z' : match(w, r) = 1.\end{aligned} \tag{5.23}$$

An $accurate$ rule set predicts all words in the training data accurately:

$$\begin{aligned}\forall Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z') :\\ accurate(Z', oset(Z')) = 1 \iff complete(Z') = 1,\\ \forall w \in TD, \forall r \in winningrule(Z', oset(Z'), w) : outcome(w) = outcome(r).\end{aligned} \tag{5.24}$$

A $minimal$ rule set is an accurate rule set that contains the fewest rules possible:

$$\begin{aligned}\forall Z' \subseteq Z_{combined}, oset(Z') \subseteq allset(Z') :\\ minimal(Z', oset(Z')) = 1 \iff accurate(Z', oset(Z')) = 1,\\ \nexists Z'' \subseteq Z_{combined}, oset(Z'') \subseteq allset(Z'') :\\ accurate(Z'', oset(Z'')) = 1, |Z''| < |Z'|.\end{aligned} \tag{5.25}$$

A $possibly\_minimal$ rule set is a set of rules that can be minimal, if ordered correctly:

$$\begin{aligned}&\forall Z' \subseteq Z_{combined} : possibly\_minimal(Z') = 1 \iff \\ &\exists oset(Z') \subseteq allset(Z') : minimal(Z', oset(Z')) = 1.\end{aligned} \tag{5.26}$$

5.3.2.4 ALLOWED STATES AND ALLOWED OPERATIONS

The full set of rules in $Z_{combined}$ consists of all possible rules and is therefore a superset of one or more $minimal$ rule sets Z_m[5]. We would like to delete the unnecessary rules until only a $minimal$ rule set is retained. When rules are deleted, it is possible that one of the rules required by Z_m is deleted unintentionally, and in order to compensate for this deletion, two or more additional rules may have to be kept to retain accuracy over TD. The final rule set Z' will then have a number of rules $|Z'| > |Z_m|$. To prevent this from happening, rules are eliminated by adding orderings, deleting redundant rules, identifying required rules and resolving conflict rules via a set of allowed operations. The $state$ of rule extraction can always be described by the triple $Z', Z_e, oset(Z')$, where Z' indicates the possible rules that can still be included in the final rule set, Z_e indicates required rules that have to be included in the final rule set, and $oset(Z')$ identifies some of the required rule orderings among elements of Z'. Each allowed operation changes the $state$ of rule extraction, from one $allowed\_state$ to another, with $allowed\_state$ as defined below (in eq 5.29).

Let the $order\_subset(oset_A(Z_A), oset_B(Z_B))$ relation be true if a set of rule orderings $oset_A(Z_A)$ is equal to or a subset of another set of rule orderings $oset_B(Z_B)$ (possibly defined on a different rule set) when the two sets of rule orderings are compared on their rule set intersection. More specifically:

$$\begin{aligned} &\forall Z_A \subseteq Z_{combined}, Z_B \subseteq Z_{combined}, \\ &\forall oset_A(Z_A) \subseteq allset(Z_A), oset_B(Z_B) \subseteq allset(Z_B) : \\ &order\_subset(oset_A(Z_A), oset_B(Z_B)) = 1 \iff \\ &\forall r, s \in Z_A \cap Z_B : (r, s) \in oset(Z_A) \implies (r, s) \in oset(Z_B). \end{aligned} \tag{5.27}$$

Let $minrules(Z', Z_e, oset(Z'))$ identify all the $minimal$ rule set and rule ordering set pairs that can be derived from Z', given the set of orderings $oset(Z')$ and a required rule subset Z_e. More specifically:

$$\begin{aligned} &\forall Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z') : \\ &(Z_m, oset_m(Z_m)) \in minrules(Z', Z_e, oset(Z')) \iff \\ &minimal(Z_m, oset_m(Z_m)) = 1, Z_e \subseteq Z_m \subseteq Z', \\ &order\_subset(oset(Z'), oset_m(Z_m)) = 1. \end{aligned} \tag{5.28}$$

[5]By definition, at least one $miminal$ rule set will always exist.

Let $allowed\_state(Z', Z_e, oset(Z'))$ indicate that for a given required subset Z_e and required set of orderings $oset(Z')$, there exists a $minimal$ rule set Z_m contained within Z':

$$\begin{aligned} &\forall Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z') : \\ &allowed\_state(Z', Z_e, oset(Z')) = 1 \iff \\ &\exists Z_m, oset_m(Z_m) : (Z_m, oset_m(Z_m)) \in minrules(Z', Z_e, oset(Z')). \end{aligned} \tag{5.29}$$

Define an $allowed\_op$ as any operation that, when applied to any possible $allowed\_state$ of a rule set and rule ordering set, will result in another $allowed\_state$.

Let each element in $minset(Z_m, oset(Z_m))$ consist of all and only those orderings required for a $possibly\_minimal$ rule set Z_m to be minimal, given some prior set of orderings $oset(Z_m)$:

$$\begin{aligned} &\forall Z_m \subseteq Z_{combined}, possibly\_minimal(Z_m) = 1, \\ &\forall oset(Z_m) \subseteq allset(Z_m) : \\ &oset_m(Z_m) \in minset(Z_m, oset(Z_m)) \iff \\ &oset(Z_m) \subseteq oset_m(Z_m), minimal(Z_m, oset_m(Z_m)) = 1. \end{aligned} \tag{5.30}$$

It follows directly from the definition of $minrules$ (eq. 5.28) and $minset$ (eq. 5.30) that:

$$\begin{aligned} &\forall Z_m \subseteq Z_{combined}, possibly\_minimal(Z_m) = 1, \forall oset(Z_m) \subseteq allset(Z_m) : \\ &oset_m(Z_m) \in minset(Z_m, oset(Z_m)) \iff \\ &(Z_m, oset_m(Z_m)) \in minrules(Z_m, Z_m, oset_m(Z_m)). \end{aligned} \tag{5.31}$$

Such a $minset(.)$ ordering does not exist for all prior orderings $oset(.)$. The $valid(Z_m, oset(Z_m))$ relation indicates that a specific $oset(Z_m)$ defined with regard to a $possibly\_minimal$ rule set Z_m consists of a subset of the restrictions required by at least one $minset(Z_m, oset(Z_m))$. Specifically:

$$\begin{aligned} &\forall Z_m \subseteq Z_{combined}, possibly\_minimal(Z_m) = 1, \\ &\forall oset(Z_m) \subseteq allset(Z_m) : valid(Z_m, oset(Z_m)) = 1 \iff \\ &\exists oset_m(Z_m) \subseteq allset(Z_m) : oset_m(Z_m) \in minset(Z_m, oset(Z_m)). \end{aligned} \tag{5.32}$$

It follows directly from the definition of $allowed\_state$ (eq. 5.29) and $valid$ (eq. 5.32) that:

$$\begin{aligned} &\forall Z_m \subseteq Z_{combined}, possibly\_minimal(Z_m) = 1, \forall oset(Z_m) \subseteq allset(Z_m) : \\ &valid(Z_m, oset(Z_m)) = 1 \iff allowed\_state(Z_m, Z_m, oset(Z_m)) = 1. \end{aligned} \tag{5.33}$$

If Z_m is a minimal rule set describing the training data TD, then some of the rules in Z_m will each be a single unique rule, while other rules will each be one of a set of possible options – any one of which could have been generated at a specific point in the rule application order without influencing the number of rules required to predict the training set accurately and completely. Such a combination of rules is referred to as a $rule\_variant$ set.

5.3.2.5 MATCHWORDS, POSSIBLE_WORDS, RULEWORDS AND SHARED_WORDS

Throughout rule extraction, we keep track of the set of words that may influence our decisions with regard to a specific rule. In this way we identify words that match a specific rule ($matchwords$), words that will invoke a specific rule during prediction ($rulewords$), and the set of possible words that may result in rulewords in the final ordering ($possible\_words$). We also identify the possible words that any two rules share ($shared\_words$).

Let the set $matchwords(r)$ consist of all words matched by a specific rule r:

$$\begin{aligned}&\forall r \in Z_{combined}, w \in TD : \\ w \in matchwords(r) &\iff match(w,r) = 1. \end{aligned} \tag{5.34}$$

Let the set $rulewords(Z', oset(Z'), r)$ consist of all words that can cause a specific rule r to be invoked (where the actual rule invoked will depend on the actual rule number assignment), given the current set of rule orderings $oset(Z')$:

$$\begin{aligned}&\forall r \in Z', Z' \subseteq Z_{combined}, w \in TD : \\ &w \in rulewords(Z', oset(Z'), r) \iff \\ &\quad r \in winningrule(Z', oset(Z'), w)\end{aligned} \tag{5.35}$$

Not all rules can necessarily be invoked when predicting the words in TD - for rules that cannot be invoked given the current rule set, the set of $rulewords(.)$ is empty. Note also that the actual words that will invoke rule r in the final ordered rule set consists of the set $rulewords(Z_m, minset(oset(Z_m)), r)$ not the set $rulewords(Z_m, oset(Z_m), r)$.

As the rule set is manipulated, additional rules are added to the required subset Z_e, which can affect the $possible\_words$ sets of all rules later in the rule graph. When comparing two rules r and s, it is possible that rule set extraction has progressed further in a section of the rule graph leading up to one rule than in the section of the rule graph leading up to the other. In order to be able to obtain a clear comparison of the two rules, we choose a shared point in the rule graph (rule v in the definition below) and only allow rules defined prior to this point to influence the $possible\_words(.)$ sets of both rules, resulting in a stable basis for rule comparison.

Let the set $possible\_words(Z', Z_e, oset(Z'), v, r)$ consist of all words that match a specific rule r, and have not yet been caught by rule v or an earlier rule than v, where v in Z_e, the required subset of rule set Z' when rule extraction is in state $Z', Z_e, oset(Z')$:

$$\begin{aligned} \forall r, s \in Z', v \in Z_e, Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z') :& \\ w \in possible\_words(Z', Z_e, oset(Z'), v, r) \iff& \\ v = r \text{ or } (v, r) \in oset(Z'), match(w, r) = 1,& \\ \nexists s \in Z_e : match(w, s) = 1, s = v \text{ or } (s, v) \in oset(Z')^{a}.& \end{aligned} \tag{5.36}$$

[a]Note that if $v \neq r$ and $(v, r) \notin oset(Z')$ then $possible\_words(Z', Z_e, oset(Z'), v, r) = \phi$

Let v_0 be an imaginary rule that matches no words, is always the first rule to occur in any rule set, and does not contribute to the rule count of a rule set. The rule v_0 has the following characteristics:

$$v_0 \in Z_{combined}. \tag{5.37}$$

$$|\{v_0\}| = 0. \tag{5.38}$$

$$\forall w \in TD : match(w, v_0) = 0. \tag{5.39}$$

$$\begin{aligned} \forall v_i \in Z', Z' \subseteq Z_{combined}, v_i \neq v_0, \forall oset(Z') \subseteq allset(Z') :& \\ (v_0, v_i) \in oset(Z').& \end{aligned} \tag{5.40}$$

Since v_0 does not affect further rule set orderings directly, and cannot affect any word-rule relationship, such a rule can be added without causing any side effects in the rule set. We use rule v_0 as a stable point for rule comparison when two rules do not share identical predecessors in Z_e when evaluating $possible\_words$ sets with regard to some stable point v, as defined in eq. 5.36. An alternative stable point that can be used is the last shared parent of the two rules in Z_e, as defined below.

Let $last\_parent(Z', Z_e, oset(Z'), r, s)$ be the latest possible rule or rules in Z_e that occur earlier than both rules r and s according to $oset(Z')$, or the earliest of r and s, if $r, s \in Z_e$:

$$\begin{aligned} &\forall r, s \in Z', v \in Z_e, Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z') : \\ &\qquad v \in last\_parent(Z', Z_e, oset(Z'), r, s) = 1 \iff \\ &\{(v,r),(v,s)\} \in oset(Z');\ \nexists t \in Z_e : \{(v,t),(t,r),(t,s)\} \in oset(Z'). \end{aligned} \tag{5.41}$$

Let $shared\_words(Z', Z_e, oset(Z'), r, s, v)$ identify those words that are in the $possible\_words$ sets of two different rules r and s with regard to some rule v:

$$\begin{aligned} &\forall r, s \in Z', v \in Z_e, Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset'(Z') \subseteq allset(Z') : \\ &\qquad shared\_words(Z', Z_e, oset(Z'), v, r, s) \equiv \\ &possible\_words(Z', Z_e, oset(Z'), v, r) \cap possible\_words(Z', Z_e, oset(Z'), v, s). \end{aligned} \tag{5.42}$$

5.3.2.6 *COMPLEMENTING RULES: CONTAINPAT, MINCOMP AND SUPERCOMP*

We now introduce a number of relationships that may exist between pairs of rules. These relationships are crucial in understanding how rules may substitute for one another, and therefore form the foundation for the derivation of minimal rule sets.

Let $complement(Z', Z_e, oset(Z'), v, r, s)$ indicate that rules r and s have overlapping $possible\_words$ sets, i.e.

$$\begin{aligned} &\forall r, s \in Z', r \neq s, v \in Z_e, Z_e \subseteq Z' \subseteq Z_{combined} : \\ &complement(Z', Z_e, oset(Z'), v, r, s) = 1 \iff \\ &\quad \exists w \in shared\_words(Z', Z_e, oset(Z'), v, r, s). \end{aligned} \tag{5.43}$$

Let the $path(relation(r, s))$ indicate that a path of relations of a specific type exists between rules r and s in a rule set Z', i.e

$$\forall r, s \in Z' : path(relation(r, s)) = 1 \iff \exists\, x_1 = r, x_2, ..., x_n = s \in Z' : \\ relation(x_1, x_2) = relation(x_2, x_3) = ... = relation(x_{n-1}, x_n) = 1. \quad (5.44)$$

$$\forall r, s \in Z' : path(relation(r, s)) = -1 \iff \exists\, x_1 = r, x_2, ..., x_n = s \in Z' : \\ relation(x_1, x_2) = relation(x_2, x_3) = ... = relation(x_{n-1}, x_n) = -1. \quad (5.45)$$

$$\forall r, s \in Z' : path(relation(r, s)) = 0 \iff \nexists\, x_1 = r, x_2, ..., x_n = s \in Z' : \\ relation(x_1, x_2) = relation(x_2, x_3) = ... = relation(x_{n-1}, x_n) \neq 0. \quad (5.46)$$

where r, s and x_i in the domain of the specific relation.

Let the $path(relation_1/relation_2(r, s)) = -1|0|1$ indicate that a path exists between r and s as defined above, but with edges of either type $relation_1(.)$ or type $relation_2(.)$.

Let the $path(relation_1\&relation_2(r, s)) = -1|0|1$ indicate that a path exists between r and s as defined above, but with edges such that both $relation_1(.)$ and $relation_2(.)$ hold.

Let the relation $containpat(Z', r, s)$ indicate that rule r is a rule with the smallest possible context that contains the context of rule s, i.e.:

$$\forall r, s \in Z', Z' \subseteq Z_{combined} : \\ containpat(Z', r, s) = 1 \iff context(r) \supset context(s) \\ \text{and } \nexists t \in Z' : context(r) \supset context(t) \supset context(s). \quad (5.47)$$

Let $containpat(Z', r, s) = -1$ if and only if $containpat(Z', s, r) = 1$; and let $containpat(Z', r, s) = 0$ if and only if no $containpat(.)$ relationship exists between r and s in Z'.

From the definition of $containpat$ it follows immediately that

$$\forall r, s \in Z', Z', \subseteq Z_{combined} : \\ path(containpat(Z', r, s)) = 1 \iff context(r) \supset context(s). \quad (5.48)$$

Let the bidirectional relation $mincomp(Z', Z_e, oset(Z'), v, r, s)$ be true for all rules r and s that are minimal complements of each other, i.e.

$$\begin{aligned} \forall r, s \in Z', v \in Z_e, Z_e \subseteq Z' \subseteq Z_{combined} : \\ mincomp(Z', Z_e, oset(Z'), v, r, s) = 1 \iff \\ complement(Z', Z_e, oset(Z'), v, r, s) = 1, \\ path(containpat(Z', r, s)) = 0. \end{aligned} \tag{5.49}$$

Let the bidirectional relation $direct(.)$ be true for rules that have either a direct $mincomp(.)$ or a direct $containpat(.)$ relationship, i.e.

$$\begin{aligned} \forall r, s \in Z', v \in Z_e, Z_e \subseteq Z' \subseteq Z_{combined} : \\ direct(Z', Z_e, oset(Z'), r, s) = 1 \iff \\ containpat(Z', r, s) = \pm 1 \text{ or } mincomp(Z', Z_e, oset(Z'), r, s) = 1. \end{aligned} \tag{5.50}$$

Let the $subset(Z', Z_e, oset(Z'), v, r, s)$ relation indicate that the $possible\_words$ that can be caught by a rule r is a strict subset of the $possible\_words$ that can be caught by another rule s, with respect to a rule v that occurs earlier in the rule set than either r or s, for a given rule extraction state $Z', Z_e, oset(Z')$:

$$\begin{aligned} \forall r, s \in Z', Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z'), \\ v \in Z_e, \{(v, r), (v, s)\} \in oset(Z') : \\ subset(Z', Z_e, oset(Z'), v, r, s) = 1 \iff \\ shared\_words(Z', Z_e, oset(Z'), v, r, s) \neq \phi, \\ possible\_words(Z', Z_e, oset(Z'), v, r) \subset possible\_words(Z', Z_e, oset(Z'), v, s). \end{aligned} \tag{5.51}$$

Let the $supercomp(Z', Z_e, oset(Z'), v, r, s)$ relation be true when two rules r and s are both in a $subset(Z', Z_e, oset(Z'), v, r, s)$ and a $mincomp(Z', Z_e, oset(Z'), v, r, s)$ relation:

$$\forall r, s \in Z', v \in Z_e, Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z'),$$

$$supercomp(Z', Z_e, oset(Z'), v, r, s) = 1 \iff \tag{5.52}$$

$$mincomp(Z', Z_e, oset(Z'), v, r, s) = 1,$$

$$subset(Z', Z_e, oset(Z'), v, r, s) = 1. \tag{5.53}$$

For all $r, s \in Z', Z' \subseteq Z_{combined}$, $oset'(Z') \subseteq allset(Z')$: Let $anyset(Z', oset(Z'), r, s)$ be an alternative naming convention for $anyset(Z', Z', oset(Z'), r, s)$, where $anyset$ can be the $subset$ (eq. 5.51), $possible\_words$ (eq. 5.36), or $order\_req$ (eq. 5.58).

5.3.2.7 Z_M AS A SUBSET OF $Z_{COMBINED}$

As mentioned in section 5.3.2.4, the full set of rules in $Z_{combined}$ consists of all possible rules and is therefore a superset of one or more $minimal$ rule sets Z_m. During rule extraction each $allowed$ rule state is defined by a triple $Z', Z_e, oset(Z')$, and each allowed state can can give rise to one or more minimal rule sets $Z_m, oset(Z_m)$, where $(Z_m, oset_m(Z_m)) \in minrules(Z', Z_e, oset(Z'))$.

If any two rules r and s in Z_m have a specific relationship in one such state, this implies further relationships in prior and ensuing states (as shown in statement 15). For any two rules r and s in Z_m it holds that if r and s have a $containpat$ relationship with regard to an appropriate[6] node v when the rule extraction process is in state $Z', Z_e, oset(Z')$, rules r and s will have a similar relationship when rule extraction reaches the state $Z_m, Z_m, oset_m(Z_m)$. It can also be shown that, for any two rules r and s in Z_m, it holds that if r and s have a $complement$ relationship with regard to an appropriate node v when the rule extraction process is in the final state $Z_m, Z_m, oset_m(Z_m)$, then rules r and s will have a similar relationship for each $allowed\_state$ $Z', Z_e, oset(Z')$ leading up to the final state. For $containpat$ path relations, the statement is stronger: Any two rules r and s in Z_m will have a $path(containpat(.))$ relationship when the rule extraction process is in state $Z', Z_e, oset(Z')$ if and only if rules r and s will have a similar relationship when rule extraction reaches the state $Z_m, Z_m, oset_m(Z_m)$.

[6]This v acts as the stable comparison point previously described. Any v such that both (v, r) and (v, s) are included in the set of established orderings would be valid.

5.3.3 RULE ORDERING

In this section, we analyse the conditions and implications of an ordering relationship between two rules in a rule set Z' where $Z' \subseteq Z_{combined}$. We first define what we mean by ordering requirements, and then show how these requirements can be translated to the relationships defined in section 5.3.2.6.

Let $order_{acc}(Z', Z_e, oset(Z_m), r, s)$ indicate that if r does not occur before s, no state $Z', Z_e, oset'(Z')$ can result in an $allowed\_state$, where $oset'(Z')$ is a superset of the $oset(Z')$ orderings:

$$\begin{aligned} &\forall r, s \in Z', Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z') : \\ &\qquad order_{acc}(Z', Z_e, oset(Z'), r, s) = 1 \iff \\ &\nexists oset'(Z') \supseteq oset(Z') \cup (s, r) : allowed\_state(Z', Z_e, oset'(Z')) = 1. \end{aligned} \tag{5.54}$$

Let $order_{red}(Z', oset(Z'), r, s)$ indicate that if rule s occurs before rule r, at least one rule in the rule set Z' will become redundant, irrespective of any further orderings added:

$$\begin{aligned} &\forall r, s \in Z', Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z') : \\ &\qquad order_{red}(Z', Z_e, oset(Z'), r, s) = 1 \iff \\ &\text{at least one rule } t \in Z' \text{ becomes redundant given any state} \\ &(Z', Z'_e, oset'(Z')), oset'(Z') \supseteq oset(Z') \cup (s, r), Z'_e \supseteq Z_e. \end{aligned} \tag{5.55}$$

Let $order(Z', Z_e, oset(Z'), r, s)$ indicate that either an $order_{acc}(Z', Z_e, oset(Z'), r, s)$ or an $order_{red}(Z', Z_e, oset(Z'), r, s)$ relationship holds between any two rules r and s:

$$\begin{aligned} &\forall r, s \in Z', Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z') : \\ &order(Z', Z_e, oset(Z'), r, s) = 1 \iff order_{acc}(Z', Z_e, oset(Z'), r, s) = 1, \\ &\text{or } order_{red}(Z', Z_e, oset(Z'), r, s) = 1. \end{aligned} \tag{5.56}$$

Let $direct\_order(Z', Z_e, oset(Z'), r, s) = 1$ indicate that a direct ordering requirement exists between rules r and s:

$$\begin{aligned} &\forall r, s \in Z', Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z') : \\ &direct\_order(Z', Z_e, oset(Z'), r, s) = 1 \iff order(Z', Z_e, oset(Z'), r, s) = 1; \\ &\nexists t : direct\_order(Z', Z_e, oset(Z'), r, t) = 1, \\ &direct\_order(Z', Z_e, oset(Z'), t, s) = 1. \end{aligned} \tag{5.57}$$

As before, for each of the $order^*$ relations ($order$, $order_{acc}$, $order_{red}$ and $direct\_order$), let $order^*(Z', Z_e, oset(Z'), r, s) = -1$ indicate that $order^*(Z', Z_e, oset(Z'), s, r) = 1$, and let $order^*(Z', Z_e, oset(Z'), r, s) = 0$ indicate that $order^*(Z', Z_e, oset(Z'), r, s) \neq \pm 1$.

Using the above definitions it can be shown that if an $order_{acc}$ relationship exists between any two rules r and s in Z_m, then a path of $complement$ and $direct\_order$ relations exist between these two rules (statement 19). That is, under appropriate conditions, $order_{acc}(Z_m, , Z_m, oset(Z_m), r, s) = 1 \implies path(direct\_order\&complement(Z_m, Z_m, oset(Z_m), v, r, s)) = 1$, where v is a rule earlier than r or s (and in practice typically the $last\_parent$ of these two rules). This means that any two rules in Z_m can only have an accuracy ordering requirement if there exists a path of such complementing direct orderings from one to the other.

Once a $minimal$ rule set Z_m has been obtained, then, if $order_{red}(Z_m, Z_m, oset(Z_m), r, s) = 1$ it also follows that $order_{acc}(Z_m, Z_m, oset(Z_m), r, s) = 1$ (statement 16). The redundancy ordering requirement therefore does not introduce any additional ordering requirements in the final rule graph, but does provide a way to restrict the set of rule orderings early on in the rule extraction process. If any two rules Z' are in a $subset$ relationship with regard to some rule v as above, then it can be shown that these two rules are also in an $order_{red}(Z_m, Z_m, oset(Z_m), r, s)$ relationship with regard to any $Z_m, oset_m(Z_m)$ pair that can be reached from the current $allowed\_state$ (statement 21). Identified $subset$ relationships can therefore be used to define initial orderings prior to further graph manipulation.

The process of applying allowed operations (as discussed in section 5.3.6) leads to further ordering requirements becoming visible. Below we define the conditions under which an $order\_req$ relationship can be enforced between two rules. It can be shown that if two rules r and s are in an $order\_req(Z', Z_e, oset(Z'), v, r, s)$ relationship with regard to a rule v as above, then these two rules are also in an $order_{acc}(Z', Z_e, oset(Z'), r, s)$ relationship (statement 26). The $order\_req$ relationships therefore provide an indication of accuracy ordering requirements that emerge during rule set extraction.

Let the $order\_req(Z', Z_e, oset(Z'), v, r, s)$ relation be true if two rules r and s disagree with regard to outcome, and have a non-empty set of $shared\_words$, all of which agree with rule r with regard to outcome, and at least one of which disagrees with rule s with regard to outcome:

$$\begin{aligned}
\forall r \in Z_{single}, s \in Z', v \in Z_e, Z_e \subseteq Z_m \subseteq Z' \subseteq Z_{combined},\\
\forall oset'(Z') \subseteq allset(Z') :\\
order\_req(Z', Z_e, oset(Z'), v, r, s) = 1 \iff\\
direct\_order(Z', Z_e, oset(Z'), r, s) = 1,\\
shared\_words(Z', Z_e, oset(Z'), v, r, s) \neq \phi,\\
\forall w_i \in shared\_words(Z', Z_e, oset(Z'), v, r, s) : outcome(w_i) = outcome(r),\\
\exists w' \in shared\_words(Z', Z_e, oset(Z'), v, r, s) : outcome(w') \notin outcome(s).
\end{aligned} \tag{5.58}$$

5.3.4 CHARACTERISTICS OF AN ALLOWED STATE

Let $order\_decided(Z', Z_e, oset(Z'), r, s)$ indicate that the direction of rule ordering between two directly related rules r and s in a rule set Z' has been established based on an identified $containpat$ or $supercomp$ relationship, given some required subset of rules Z_e and some prior set of orderings $oset(Z')$. More specifically:

$$\begin{aligned}
&\forall r, s \in Z', Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset'(Z') \subseteq allset(Z') : \\
&\qquad order\_decided(Z', Z_e, oset'(Z'), r, s) = 1 \iff \\
&\qquad\qquad containpat(Z', r, s) = 1; \\
&\qquad \text{or } \exists v \in Z_e : supercomp(Z', Z_e, oset'(Z'), v, r, s) = 1; \\
&\qquad \text{or } \exists oset(Z') \subseteq allset(Z') : order\_subset(oset(Z'), oset'(Z')) = 1, \\
&\qquad\qquad order\_decided(Z', Z_e, oset(Z'), r, s) = 1.
\end{aligned} \tag{5.59}$$

Let $decided\_set(Z', Z_e, oset(Z'))$ take any rule set Z', required subset Z_e and set of rule orderings $oset(Z')$, and generate a set of all the current $order\_decided$ relationships:

$$\begin{aligned}
&\forall r, s \in Z', Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z') : \\
&\qquad (r, s) \in decided\_set(Z', Z_e, oset(Z')) \iff \\
&\qquad path(order\_decided(Z', Z_e, oset(Z'), r, s)) = 1.
\end{aligned} \tag{5.60}$$

Note that an $order\_decided(Z', Z_e, oset(Z'), r, s)$ relationship does not imply that rule r and s will either or both be retained in Z_m, but only that if both were retained, r would be ordered prior to s.

Let $order\_possible1(Z', Z_e, oset(Z'), r, s)$ indicate that, even though it has not yet been established whether a rule ordering is required between two rules r and s: if a rule ordering is required, the direction of such an ordering will be from rule r to rule s because of the existence of a $order\_req$ relationship between these two rules. More specifically:

$$\begin{aligned} &\forall r, s \in Z', Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z') : \\ &order\_possible1(Z', Z_e, oset(Z'), r, s) = 1 \iff \\ &\exists v \in Z_e, \{(v, r), (v, s)\} \in oset(Z') : order\_req(Z', Z_e, oset'(Z'), v, r, s) = 1. \end{aligned} \tag{5.61}$$

Let $order\_possible(Z', Z_e, oset(Z'), r, s)$ indicate that the direction of rule ordering (if any) has not yet been established between two minimal complements r and s that have remaining $shared\_words$ when rule extraction is in state $Z', Z_e, oset(Z')$. More specifically:

$$\begin{aligned} &\forall r, s \in Z', Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z') : \\ &order\_possible(Z', Z_e, oset(Z'), r, s) = 1 \iff mincomp(Z', r, s) = 1, \\ &shared\_words(Z', Z_e, oset(Z'), r, s) \neq \phi, \\ &order\_decided(Z', Z_e, oset(Z'), r, s) = 0. \\ &order\_possible1(Z', Z_e, oset(Z'), r, s) \neq -1. \end{aligned} \tag{5.62}$$

Let $possible\_set(Z', Z_e, oset(Z'))$ take any rule set Z', required subset Z_e and set of rule orderings $oset(Z')$, and generate a set of all the $decided$ and $possible$ rule orderings. More specifically:

$$\begin{aligned} &\forall r, s \in Z', Z_e \subseteq Z' \subseteq Z_{combined}, oset(Z') \subseteq allset(Z') : \\ &(r, s) \in possible\_set(Z', Z_e, oset(Z')) \iff \\ &path(order\_possible/order\_decided(Z', Z_e, oset(Z'), r, s)) = 1. \end{aligned} \tag{5.63}$$

Consider any $Z', Z_e, oset(Z')$ combination such that $allowed\_state(Z', Z_e, oset(Z')) = 1$. By definition (eq. 5.28 and eq. 5.29) it will always hold that:

$$Z_e \subseteq Z_m \subseteq Z_{combined}. \tag{5.64}$$

Furthermore, it can be shown (statement 28 and 27) that:

$$order\_subset(decided\_set(Z', Z_e, oset(Z')), oset_m(Z_m))) = 1. \tag{5.65}$$

$$order\_subset(oset_m(Z_m), possible\_set(Z', Z_e, oset(Z'))) = 1. \tag{5.66}$$

Now consider any two rules r and s that will be retained in the minimal rule set Z_m. From the above it follows that if rules r and s are ordered according to $decided\_set(Z', Z_e, oset(Z'))$ then these two rules will retain this ordering in the minimal rule set ordering $oset_m(Z_m)$. Also, any rule ordering that will eventually be required in $oset_m(Z_m)$ is currently listed in $possible\_set(Z', Z_e, oset(Z')$. For any given state $Z', Z_e, oset(Z')$, the definite and possible rule orderings can therefore be generated automatically, and used to reason about further graph manipulation options.

5.3.5 INITIAL ALLOWED STATE

The rules in $Z_{combined}$ describe the training data completely, but not necessarily accurately. Since all possible rules are included in $Z_{combined}$, it follows that $\phi \subseteq Z_m \subseteq Z_{combined}$ for all $minimal$ rule sets Z_m. Furthermore, it can been shown that if the rules in $Z_{combined}$ are ordered according to $containpat$ and $supercomp$ relations, then no overly restrictive orderings will be added. If the rule set $Z_{combined}$ is ordered according to the rule set orderings generated by $decided\_set(Z_{combined}, \phi, \phi)$, then the rule set is in an $allowed\_state$ (statement 33). This state is used as the initial state prior to application of the various $allowed\_ops$, as described below.

5.3.6 ALLOWED OPERATIONS

Each $allowed\_op$ as defined in section 5.3.2.4 changes the state of the rule set, required rule subset and rule ordering set, from one $allowed\_state$ to another, with the initial $allowed\_state$ defined in section 5.3.5. These operations are not unique, and both stronger and weaker versions can be constructed. While the framework up to this point has been defined rigorously, we now discuss a number of possible operations in order to demonstrate how this framework can be used during rule extraction. We describe a number of operations that we have implemented and tested in our rule extraction system. Specifically, we describe allowed operations that can be applied to (1) delete rules, (2) remove unnecessary edges, (3) mark rules as required, and (4) resolve conflicted rules.

When applying any of these operations it is assumed that the rule graph is in an $allowed\_state$ defined by the triple $Z', Z_e, oset(Z')$. Prior to discussing these operations in further detail, it should be noted that the rule graph edges added according to the $decided\_set$ and $possible\_set$ orderings have two functions: On the one hand, these edges define the order in which rules will occur, as discussed up to this point. Secondly, these edges link any rule to *all possible rules* that may potentially replace or be replaced by the current rule. *Direct successors* or *predecessors* are identified by following both $decided\_set$ and $possible\_set$ orderings and identifying rules that either have to or can occur directly before or after the rule being considered. From these sets of rules it is possible to define the total number of rules that (1) will definitely and (2) may potentially (based on decisions made in other sections of the rule graph) be deleted if a current rule is associated with a specific outcome.

5.3.6.1 DECREASING RULE SET SIZE

A rule r can only be deleted if it can be shown that the rule has become redundant, and will remain redundant. This can occur for two reasons: (1) because of the position of the specific rule r in the rule graph, all words that match this rule are already caught by required rules (correctly identified as such) that occur earlier in the rule set; or (2) because the rule can be 'merged' with a second rule occurring at the same point in the rule extraction order. We define three different allowed operations with regard to rule deletion:

1. A rule r may be deleted if some rule s exists such that:
   - Rules r and s are resolved ($r, s \in Z_{single}$) and agree with regard to outcome.
   - Rules r and s have identical relationships with identical predecessors (both possible and definite).
   - Rules r and s have identical relationships with identical successors (both possible and definite).
2. A rule r may be deleted if a set of rules v_i exists such that:
   - The v_i constitute all the direct successors of rule r.
   - Rule r and all the v_i are resolved ($r, v_i \in Z_{single}$) and agree with regard to outcome. (No rule t with a potentially conflicting outcome can have an ordering that allows it to occur between rule r and any v_i.)
3. A rule r may be deleted if for some allowed v, the $possible\_words(Z', Z_e, oset(Z'), v, r) = \phi$.

5.3.6.2 REMOVING UNNECESSARY EDGES

Since $decided$ orderings are transitive, it is possible to remove any explicit ordering $rule\_order(Z', r, s)$ if it already holds that $rule\_order(Z', r, t) = 1$ and $rule\_order(Z', t, s) = 1$ for some $t \in Z'$. Note that this does not remove (r, s) from $oset(Z')$.

5.3.6.3 IDENTIFYING REQUIRED RULES

When a rule is identified as *required*, it is moved from the possible rule set Z' to the set of required rules Z_e. This is an $allowed\_op$ for a rule r if the rule r itself has already been resolved (that is, it can only predict a single outcome), and no further rules exist that may potentially agree with regard to outcome.

5.3.6.4 RESOLVING CONFLICT RULES

We define four different operations that can be used to resolve conflicted rules. All of these operations utilise the concept of a $single$ rule and a $conflict\_count$. A rule s is identified as a $single$ rule if there exists at least one word w such that $(w, s) \in oset(Z')$ and no t exists such that $match(w, t) = 1$, unless $(s, t) \in oset(Z')$. This means that word w can only be predicted by rule s, or by a later rule that has a decided path from word w via rule s. For each possible outcome we count the number of predecessors that will definitely be deleted if rule r is resolved to that specific outcome ($definite\_count$), as well as the number of predecessors that may possibly be deleted ($possible\_count$). These counts are not calculated per predecessor, but rather per word set that a predecessor represents. For a predecessor s to contribute to a ($definite\_count$), the predecessor must be resolved (that is, $s \in Z_{single}$), be identified as a $single$ rule, and have only one successor (the conflicted rule r itself).

1. If it is clear that rule r will provide an advantage if resolved to a specific outcome, resolution is performed, and the conflicted rule is replaced with a normal rule with the selected outcome. A rule may only be resolved in this way, if the $definite\_count$ for a specific outcome dominates the sum of the $definite\_count$ and $possible\_count$ for all alternative outcomes. It is also required that at least one of the predecessors with a outcome matching the outcome selected for rule resolution be a $single$ rule. This prevents an unnecessary rule from being generated at this point in the rule application order. (In a later step, the resolved conflict node will merge with the $single$ predecessor.)

2. If a conflicted rule r has no predecessors that can potentially agree with each other with regard to a specific outcome, the rule r may be deleted. (We refer to this process as a *lost conflict*). A rule may be resolved in this way if the sum of the $definite\_count$ and $possible\_count$ is less than or equal to one, for all possible outcomes.

3. If for any of the outcomes the sum of the $definite\_count$ and $possible\_count$ is less than or equal to one, that outcome may be removed from the possible outcomes of the conflicted rule, even though the rule remains a conflicted rule. If all except one outcome are removed in this way, then at least one predecessor must be a $single$ rule that agrees with regard to the final outcome, as discussed with regard to the previous operator.

4. The root node is not allowed to be resolved via operator (2) or (3). If operator (1) is not applicable, the root node is resolved to the outcome that occurs most in the training data. This operation is only allowed when all predecessors have been resolved.

5.3.7 BREAKING TIES

Once all the possible operations have been applied to the rule graph, and no further simplification is possible, it does not necessarily mean that all conflicted rules have been resolved. The extent to which the rule graph is resolved, depends on the strength of the various allowed operations defined. Note that the set difference between the increasing rule set Z_e, and the decreasing rule set Z' provides a clear indication of the extent to which the current solution still falls short of the minimal rule set Z_m. If Z_e equals Z', a minimal rule set has been obtained.

The remaining conflicted nodes can be solved by viewing the rule graph as a constraint satisfaction problem (CSP). By assigning the various remaining (node-specific) outcome values to each of the remaining conflicted nodes, and searching through the resulting search space, the final solution can be obtained. During the CSP search process, all conflicted nodes are solved simultaneously, and the rule minimisation process proceeds rapidly using the various deletion operations. By searching through all the remaining allowed rule sets, the smallest possible set can be obtained.

The magnitude of this CSP is determined by the coverage of the operations employed. If only trivial operations were employed, and all conflicts were left to the CSP to resolve, a huge CSP would result for even very small problems. The stronger the allowed operations defined, the smaller the CSP to solve. Our current implementation has been used to solve small tasks of $n = 100$ words, and we have been able to extract rule sets that are smaller than that extracted by *Default&Refine*. Various CSP-specific techniques can be applied to improve the computational tractability of the task. However, this is not the focus of the current chapter, which aims to define a solid theoretical basis for further experimentation: computational tractability will be addressed in future work.

5.3.8 OPTIMISING GENERALISATION ABILITY

Once all conflicted rules have been solved, and the minimal rule set obtained, it is possible to refine the rule set by choosing the best rule option among the various variants available. Possible selection criteria include smallest rule context, most symmetric rule context, best coverage of the training data, best fallback given the following set of rules in the specificity hierarchy, and various others. Since the choice of variant does not influence the number of rules generated, this provides flexibility in the construction of the final rule set. When heuristics are employed during rule extraction, choices are limited earlier on: this framework allows heuristic choices to be postponed as late as possible during rule extraction, and makes those choices explicit.

5.4 ALTERNATIVE ALGORITHMS AS SPECIALISATION OF GENERAL FRAMEWORK

It is interesting to note that the rule extraction algorithms discussed in Chapter 4 can be viewed within the *minimal representation graph* framework. In the case of DEC the full set of rules in Z is not constructed: only the rules that match the DEC format are generated. The rule graph is not constructed prior to rule extraction but is grown during rule extraction according to $containpat$ relationships. Each additional conflicting word results in another leaf being added to the graph.

Default&Refine also grows the graph from the root outwards, ordering rules according to rule extraction order. At each level, a decision is made with regard to the rule and associated outcome that best predicts the set of words that must be caught at that level in the rule graph (that will not be predicted correctly by a later rule). This is conceptually similar to generating a full rule graph prior to rule extraction, and resolving rules strictly from the root outwards according to a greedy heuristic at each level. Since neither algorithm proceeds with allowed operations from an allowed initial state, both will in general produce larger-than-minimal rule sets.

5.5 EXTENSIONS

The current framework provides a solid theoretical base for reasoning about the choices made during grapheme-to-phoneme rule extraction. We are interested in how this framework can be extended to incorporate additional techniques, and this will be addressed in future work. Specific extensions that may fit well within this framework include:

- Pronunciation variants: currently pronunciation variants are not allowed (See eq. 5.13). If a single pseudo-phoneme is generated for each alternating sound, the same framework can be used to process pronunciation variants, with variants expected to drift towards the top of the graph, unless clearly systematic. Further choices ensue with regard to resolving conflict between a single phoneme and a matching pseudo-phoneme, and extensions to the current framework may assist in resolving such issues.

- Class-based groupings: It is clear from rule set analysis that groups of graphemes tend to influence neighbouring graphemes in systematic ways. It should be possible to accelerate the learning process by extracting such graphemic groups during rule set extraction. This may require the interlinking of a number of minimal memory graphs in a single structure.

- Combining phonemic and graphemic context: The same rule set can be generated in terms of either graphemic or phonemic context. We are interested in the advantages and disadvantages of combining both approaches in a single rule set.

- Graphemic chunks: As all possible word sub-components are generated during rule set extraction, the extent to which the rule graph is manipulated brings this approach closer or removes it further from pronunciation-by-analogy techniques. We are interested in the similarities and differences between these two approaches.

5.6 CONCLUSION

In this chapter we described a theoretical framework that can be used to analyse the grapheme-to-phoneme prediction problem in a rigorous fashion. Using this framework, it is possible to define a number of ‘allowed operations’ that attempt to extract the smallest possible rule set from any given set of training data. By making the various options available at each stage of rule extraction explicit, we obtain a better understanding of the grapheme-to-phoneme prediction task itself. Furthermore, the new framework provides a solid foundation for further research in pronunciation prediction, including the potential incorporation of pronunciation variants, class-based groupings and/or graphemic chunks within a rewrite-rule based framework.

CHAPTER SIX

BOOTSTRAPPING PRONUNCIATION MODELS

6.1 INTRODUCTION

In this chapter we apply the grapheme-to-phoneme rule extraction mechanisms developed earlier in order to bootstrap pronunciation models. We analyse the bootstrapping process by developing pronunciation models in Afrikaans, a Germanic language with a fairly regular grapheme-to-phoneme relationship, and describe a number of experiments conducted to evaluate specific aspects of the bootstrapping process. In Section 6.5.4 we analyse the efficiency of the bootstrapping process according to the framework defined in Chapter 3. The completed system has since been used for the development of dictionaries in a number of additional languages (isiZulu, Sepedi and Setswana[1]) and these dictionaries integrated in speech technology systems, as described in Section 6.6.

6.2 BOOTSTRAPPING SYSTEM

Two bootstrapping systems were developed:

- System A: The bootstrapping approach as described in Section 3.4 was implemented in Perl, to run within a Web browser [72]. This prototype provided an experimental platform for the evaluation of the various algorithms described in Chapter 4 and allowed initial measurements with regard to developer efficiency and accuracy. The experiments described in Sections 6.3 and 6.4 utilised this system.
- System B: Components of System A were re-implemented in Java in order to provide more user-friendly interaction. The new system does not implement all the algorithms evaluated in

[1]Three more of South Africa's official languages, from the Bantu family.

this thesis, but provides a more robust platform for dictionary development[2]. System B was used in the experiment described in Section 6.5.

Both systems implement the bootstrapping approach described in Section 3.4, as described in more detail from both the user and system perspective in the next two sections.

6.2.1 USER PERSPECTIVE

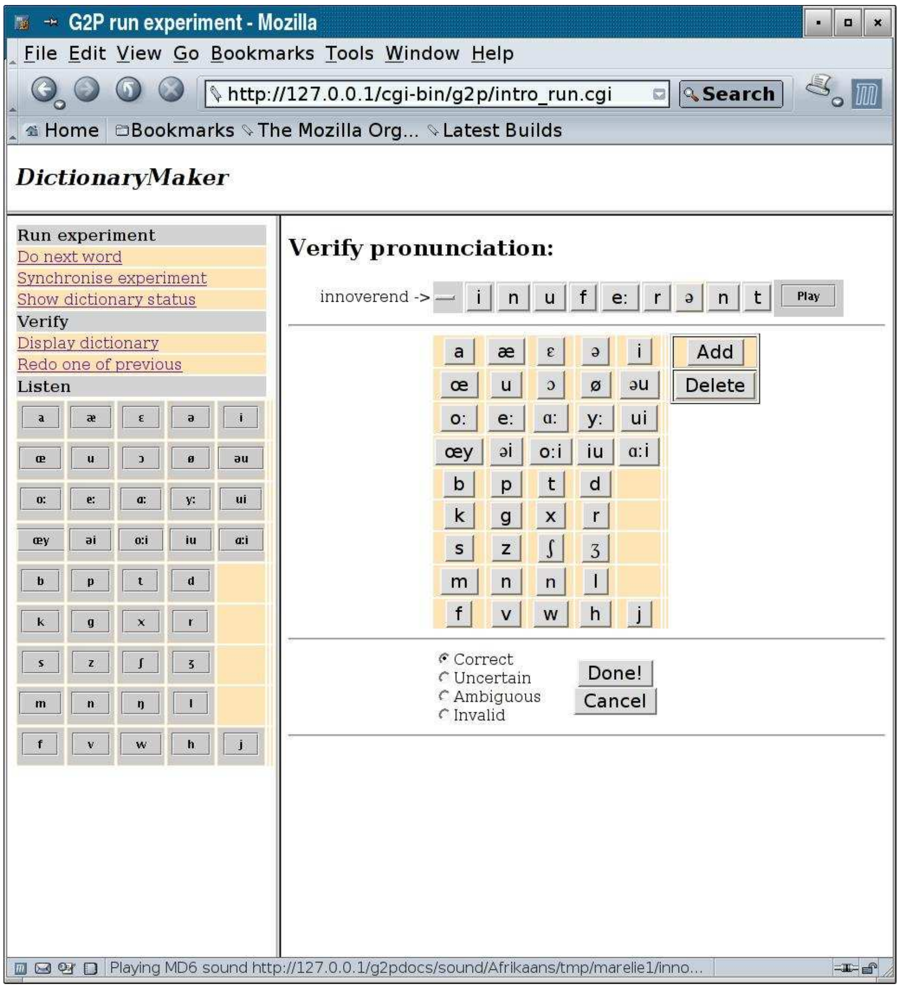


Figure 6.1: *Correcting the predicted pronunciations (System A).*

The dictionary development task as presented to the *verifier* is depicted in Fig. 6.1. The verifier is presented with each word/pronunciation pair in turn, and asked to provide a verdict of pronunciation accuracy. The verifier is required to verify all new predictions – none are assumed to be correct[3].

[2]This system will be released as Open Source Software in the near future – see http://www.csir.co.za/hlt for more information.

[3]In an alternative approach, Maskey *et al* [68] utilised a confidence metric to assume the correctness of some of the words. We preferred to verify all new predictions, given the unpredictability of some exceptions in pronunciation prediction tasks.

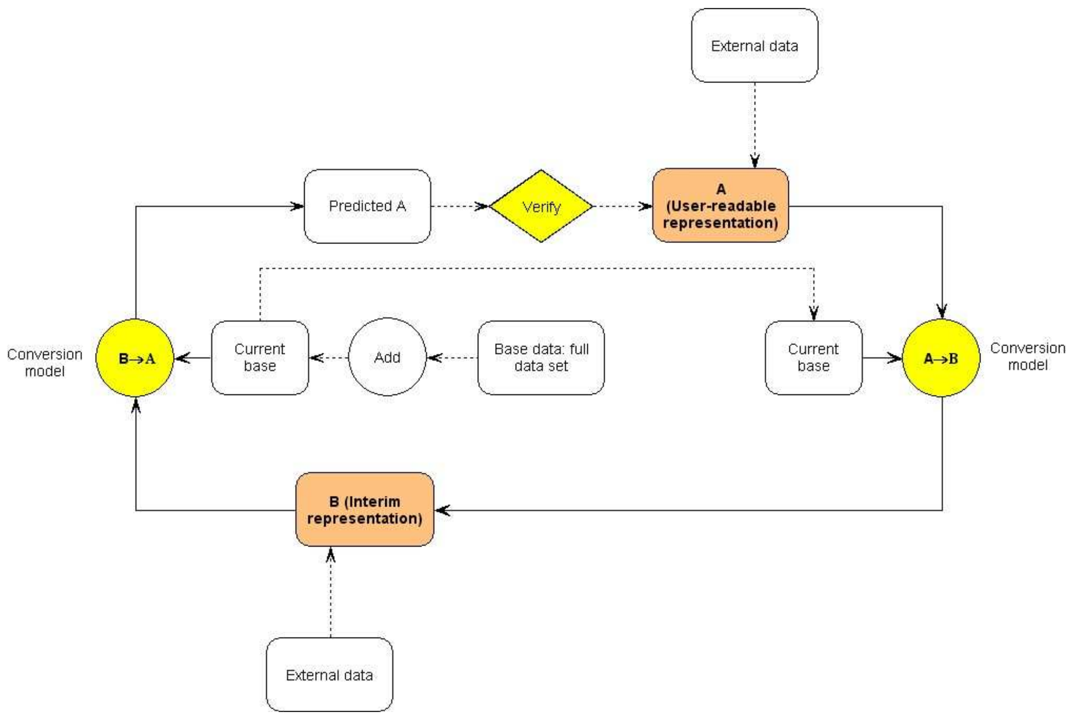


Figure 6.2: *The bootstrapping system concept.*

Once the word list and phoneme set have been loaded and the system prepared, no further expertise is required from the verifier apart from being able to differentiate between correct and incorrect pronunciations.

The verifier is presented with two representations of the pronunciation, namely a visual transcription and an audio version. The audio version is created by concatenating pre-recorded samples of each phoneme (i.e. the word is 'sounded' rather than synthesised). The verifier specifies a verdict: whether the pronunciation is *correct* as predicted, whether the word itself is *invalid*, *ambiguous* depending on context, or whether the verifier is *uncertain* about the status of the word. If the pronunciation is wrong, the verifier specifies the correct pronunciation by removing, adding or replacing phonemes in the presented pronunciation. Once the verifier is certain of the accuracy of a specific pronunciation, he or she is encouraged to listen to the audio version of the final pronunciation, and so identify potential errors. At any stage the verifier can *Redo* a word, in order to correct a previous mistake. The verifier can also *List possible errors* which provides a list of exceptional pronunciations, as discussed in more detail in Section 6.4.

6.2.2 SYSTEM PERSPECTIVE

Fig. 6.2 illustrates the bootstrapping concept from a system perspective. The bootstrapping system is initialised with a grapheme and phoneme set, and a large word list (containing no pronunciation information). Each phoneme is associated with a pre-recorded audio sample. The system can be primed with an existing rule set or dictionary, if available. If neither is available, the system will predict empty pronunciations initially, which, when corrected, form the basis for further bootstrapping.

Bootstrapping occurs in two phases. During the initial phase, the grapheme-to-phoneme models are updated whenever a word is verified as correct. In the second phase, a complete update (referred to as a synchronisation event) only occurs after a set of words has been verified as correct. In between synchronisation events, learning can either be ceased, or continued using an incremental algorithm[4]. The dictionary developer chooses the number of words at which the system progresses from the first to the second phase, as well as the size of the set corrected before models are synchronised with the new training data during the second phase. Once initialised, the following steps are repeated:

1. The system analyses its current understanding of the task and generates the next word to consider, as described in Section 6.2.3.

2. For the chosen word, the system generates a new pronunciation using its current grapheme-to-phoneme rule set.

3. The system creates a 'sounded' version of each word using the predicted pronunciation and associated sound samples, and records the verifier's final response.

4. If a word has been verified as correct, the system increases its update synchronisation counter. If an update event is due, the system updates its grapheme-to-phoneme rule set based on the new set of pronunciations.

This process is repeated (with increasingly accurate predictions) until a pronunciation dictionary of sufficient size is obtained.

6.2.3 ALGORITHMIC CHOICES

In the experiments conducted here we either use *DEC-min* or *Default&Refine* for rule extraction, as stated per experiment. We also state whether incremental learning is utilised between synchronisation events or not. A further algorithmic choice concerns the mechanism whereby the next 'best' word to add to the knowledge base is selected, as this can influence the speed at which the system learns. We utilise three different techniques in our experiments, as referred to in the various experiment descriptions:

- *Evenly selected from corpus:*
  Here we order the available word list alphabetically, and select every n^{th} word in order to obtain a subset of the required size.

[4]Such as incremental Default&Refine, described in Section 4.6.4

- *Systematic growth in context:*
  The system grows its understanding of pronunciations-in-context systematically. Contexts of varying sizes are ordered according to occurrence frequency in general text, creating a list of ‘contexts in question’. A continuous process predicts the next best word to verify based on the current state of the system: the shortest word is chosen that contains the next context in question. If so required, the system will attempt to obtain certainty on as many contexts of size *n* as possible, before continuing to a context of size *n+1*.

- *Random:*
  A subset is chosen at random.

An alternative approach is suggested in [68], where words are ordered according to frequency in general text, and the most frequent words are processed first. This provides the advantage that more frequent words are automatically included in the dictionary but can also decrease learning performance if the more frequent words tend to have irregular pronunciations, as is possible, depending on the specific language being considered.

6.2.4 SYSTEM CONFIGURATION

Fig. 6.3 depicts the options presented to the user preparing the dictionary development process. Displaying the current status, as shown here, is one task within an experimental environment that allows a user to manipulate and generate the various resources involved (the rule set, word list and pronunciation dictionary) as required. For each experiment, the system logs the history of all activities and archives the intermediary data resources for further analysis.

6.3 EXPERIMENT A: VALIDATION OF CONCEPT

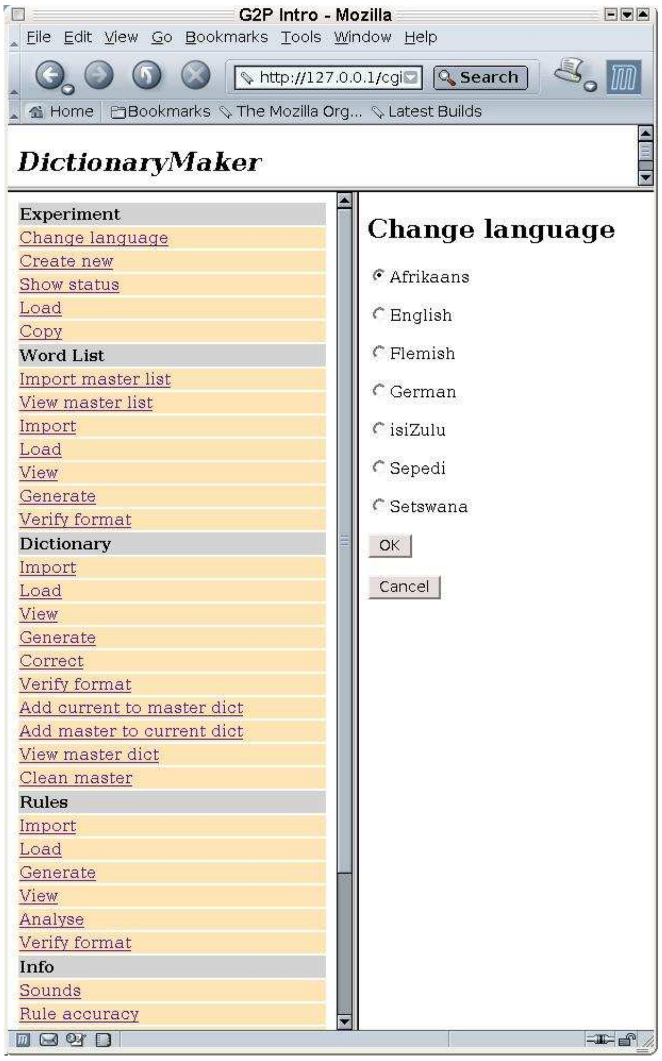
In this section we report on a series of experiments conducted in order to analyse the bootstrapping approach. The experiments are aimed at understanding a number of issues, including the following:

1. Can the bootstrapping approach be used to develop pronunciation dictionaries more quickly than conventional transcription?

2. How important is the linguistic background of the dictionary developer? Is it possible for a first language speaker without any phonetic training to develop an accurate pronunciation dictionary? (As mentioned in Section 1.1, this is highly significant in the developing world.)

3. How long does it take for a developer to become proficient with the bootstrapping system?

4. What are the practical issues that affect the speed and accuracy of dictionary development using the bootstrapping approach?



Change language

[✓ Afrikaans] English Flemish German isiZulu Sepedi Setswana

Figure 6.3: *Preparing the bootstrapping system (System A).*

In Section 6.3.1 we describe the experimental protocol followed. Utilising the framework defined in Chapter 3, we analyse the bootstrapping process from both a human factors perspective (Section 6.3.2) and a machine learning perspective (Section 6.3.3). In Section 6.3.4 we analyse the efficiency of the overall system, and compare expected and measured values.

6.3.1 EXPERIMENTAL PROTOCOL

The first set of experiments involved three dictionary developers who created pronunciation dictionaries for Afrikaans. All three developers are first-language Afrikaans speakers; and in informal interviews all three were found to employ a broadly similar dialect of "standard" Afrikaans. Two of the developers (whom we will refer to as A and B) have no formal linguistic training, whereas developer C has significant linguistic expertise, and has previous experience in the creation of pronunciation dictionaries.

The following protocol was used for all three developers:

1. A brief tutorial on the bootstrapping system, as well as the chosen phonetic representation, was presented by one of the experimenters.

2. A training set of 1000 words was drawn from a corpus of Afrikaans words, and the developers were given the opportunity to familiarise themselves with the system (and the phoneme set) by developing pronunciation rules for a subset of these words using the bootstrapping system. The process continued until the developers were satisfied that they were comfortable with the software and phoneme set.

3. A new set of 1000 words was selected, and the developers were asked to produce the most accurate rules they could, by listening to the sounded version produced by the system, correcting it if necessary, and repeating these two steps until satisfied with the pronunciation.

4. Further sets of 1000 words were used to experiment with various other factors, such as the effect of giving developers the option not to use audio assistance.

Each set of 1000 words was selected according to the 'systematic growth in context' word selection technique[5] from an independent 40,000-word subset of the full Afrikaans word list. The *DEC-min* algorithm was used for rule extraction, and all experiments were conducted in *phase 1* operation, that is, the rule set was updated after every corrected word. During these experiments we measured several relevant variables, including: the time taken to complete each verification; the number of phonemes changed per word verified; whether the developer chose to use the audio assistance; whether a developer returned to a word to re-correct it at a later stage; and the amount of idle (resting) time between sets of verifications.

6.3.2 HUMAN FACTORS

6.3.2.1 USER LEARNING CURVE

To measure a developer's facility in using the bootstrapping software, it is useful to obtain separate measurements of how long it takes (on average) to verify words in which no corrections are made, words where one correction is made, words where two corrections are made, etc. This eliminates the confounding effect of the system becoming more accurate as it learns more rules (thus accelerating apparent developer performance). By this measure, all three developers reached a satisfactory level of performance within approximately 400 words. For example, Fig. 6.4 depicts how the times for developer C to correct zero through four errors converge to their stable values; similar tendencies were seen for the other developers as well.

[5]as described in Section 6.2.3

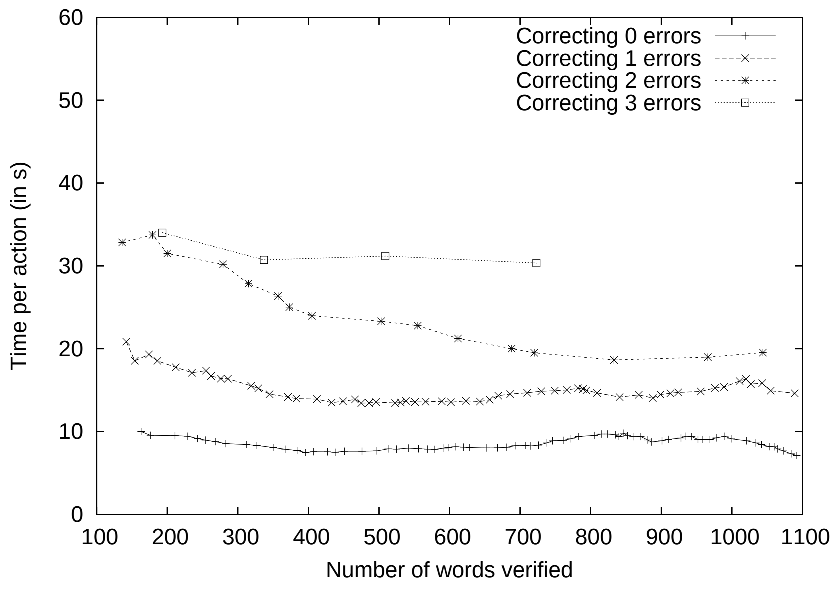


Figure 6.4: *Average time taken by developer C to verify words requiring zero, one, two or three corrections, as a function of the number of words verified. The averages were computed for blocks of 50 words each.*

This is highly encouraging, since the initial 400 words were completed in less than two hours in every case. Even linguistically untrained users can therefore become proficient at using bootstrapping within this length of time.

6.3.2.2 EFFECT OF LINGUISTIC EXPERTISE

The ability of linguistically untrained users to become proficient at using the bootstrapping system does not necessarily imply that the users were using the system accurately. It is an interesting question whether it is at all possible for a first language speaker without any phonetic training to develop an accurate pronunciation dictionary.

In order to analyse the effect of linguistic sophistication, the performance of developers A and B (who have had no linguistic training) was compared with that of developer C along the dimensions of speed and accuracy. Because there is unavoidable ambiguity in defining "correct" pronunciations (even within a particular dialect), we measured accuracy by manually comparing all cases where any pair of developers chose different transcriptions for a word. In those cases, a transcription was flagged as erroneous if (in the opinion of the author) it did not represent an accurate transcription of the word.

Table 6.1: *Estimated transcription accuracies of three developers on a set of 1000 words.*

Developer	Transcription experience	Word accuracy
A	None	83.6%
B	None	98.0%
C	Substantial	99.0%

Table 6.1 summarises the accuracies of the three developers, as estimated using this process. Only words marked as "valid" by a developer were included in the evaluation. As expected, developer C

was found to be highly accurate. Interestingly, developer B was only slightly less accurate, whereas developer A made significantly more errors than either of the others. During analysis it was revealed that developer A had not adhered to the protocol defined in Section 6.3.1: when confident of the accuracy of a pronunciation, developer A had accepted pronunciations without utilising the audio assistance provided by the system. Two conclusions are suggested by these measurements:

- It is possible for a linguistically inexperienced developer to use the bootstrapping system to attain levels of speed and accuracy comparable to those of a highly proficient dictionary developer.
- Developers with limited linguistic experience should be required to listen to every transcription, since it is easy to become over-confident about one's ability to read phonetic transcriptions.

6.3.2.3 THE COST OF USING AUDIO ASSISTANCE

Since we found that the developer who did not sound words out made many more errors than those who did, it is important to investigate how much this sub-process delays the process of verification. To this end, we asked developer C to verify an additional set of 200 words, only choosing to sound out those words where she considered it useful. In Fig. 6.5 the time taken to verify words with various numbers of corrections is compared with the times when the use of audio assistance was compulsory.

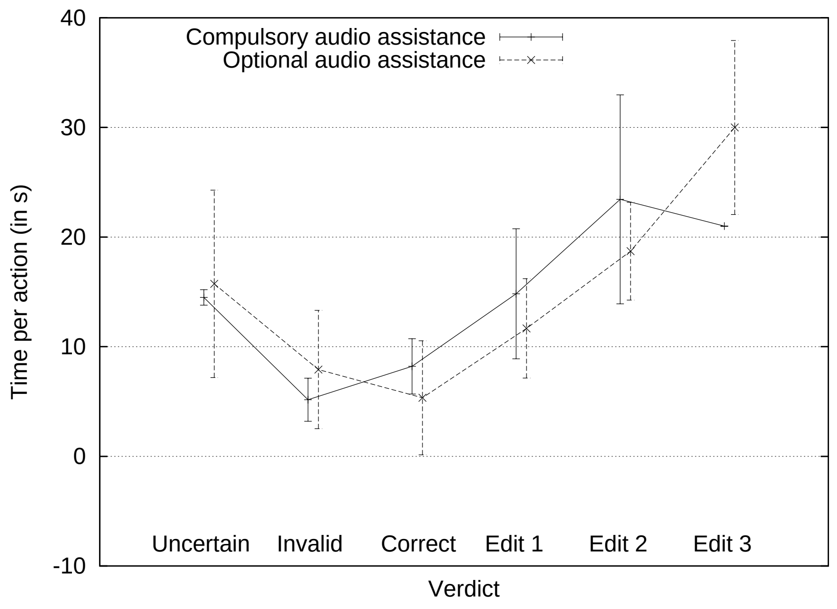


Figure 6.5: *Average time taken by developer C to verify words, with and without compulsory use of audio assistance.*

We found that this choice did not cause the developer to commit any errors; however, the reduction in verification time was also relatively small (3.6 seconds on average). This confirms the suggestion in Section 6.3.2.2 that it is generally better not to make the use of audio assistance optional.

6.3.2.4 *THE COST OF PHONEME CORRECTIONS*

The number of phoneme corrections required is the dominant factor in determining verification time. For example, analysis shows that the length of the words to be verified correlates with the verification time if no corrections are required, but not if one correction is required, and that word length is the less important of these two factors. (Word length similarly does not predict verification time if two or more corrections are required.) Developers take comparable durations to perform their verifications, as shown in Fig. 6.6.

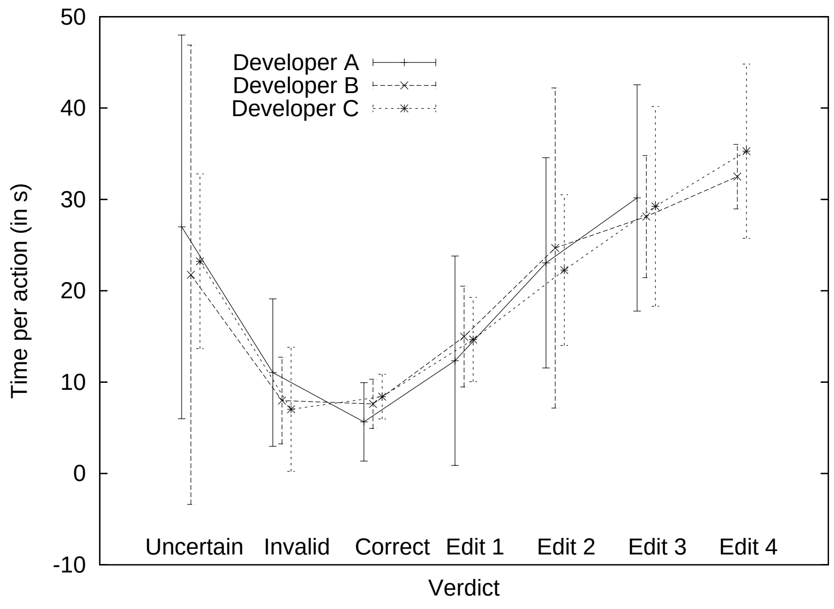


Figure 6.6: *Average time taken by three developers to verify words requiring different numbers of corrections (or to mark words as invalid or ambiguous/uncertain). The averages were computed for the same set of 1000 words as above.*

6.3.2.5 *RELATED FACTORS*

Our experiments have underlined a number of practical factors that need to be taken into account when developing pronunciation dictionaries using bootstrapping:

- Relatively informal instruction of the developers is sufficient, if they are given the opportunity to learn by using the system.
- The appropriate definition and usage of the phoneme set requires some care. When a new language is being developed, it is advisable to do this in an iterative fashion: developers develop a small dictionary, and their comments as well as transcriptions are reviewed to determine whether any phonemes are absent from the set being used, and also to determine what conventions are required to ensure consistency of the dictionary.
- For a linguistically inexperienced dictionary developer, the audio samples used should ideally match the developer's regional accent.

- When developers have limited linguistic experience, they should be required to listen to every word prior to final acceptance of a transcription.

6.3.3 MACHINE LEARNING FACTORS

6.3.3.1 SYSTEM CONTINUITY

The faster the system learns, the fewer corrections are required of the human verifier, and the more efficient the bootstrapping process becomes. The most important aspect that influences the speed at which the system learns relates to the continuity with which the system updates its knowledge base. A continuous process was chosen, whereby the system regenerates its prediction models after every single word verified. This has a significant effect on system training responsiveness, especially during the initial stages of dictionary development when the system has access to very little information on which to base its predictions.

6.3.3.2 PREDICTIVE ACCURACY

The increasing likelihood that the system will correctly predict pronunciations as more words are verified is depicted in Fig. 6.7, which shows the average number of phoneme corrections required as a function of the number of words verified by developer B. The number of corrections decreases steadily as more words are verified, producing an increasingly accurate dictionary and enabling the developer to process subsequent words more rapidly.

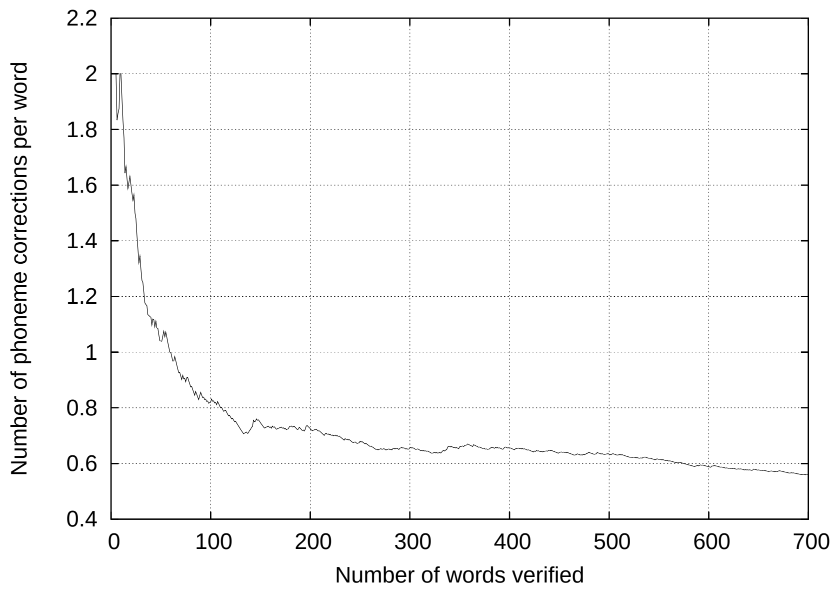


Figure 6.7: *Expected number of phonemes that required correction by developer B as a function of the number of words verified.*

6.3.3.3 VALIDITY OF BASE DATA

A final factor that influences the speed of dictionary development concerns the validity of the initial word lists. In this set of experiments word lists were obtained from Internet text and contained up to 15% invalid words.

6.3.4 SYSTEM ANALYSIS

We can combine the information in Figs. 6.7 and 6.6 to derive a model of how long it will take system users such as developers B and C to create pronunciation dictionaries of various sizes. To do this, we fit an exponential curve through the smooth part of the graph in Fig. 6.7 (i.e., for 100 or more words verified), and estimate a linear model for the expected verification time as a function of the required number of corrections. Fig. 6.8 shows how machine learning produces slower-than-linear growth in development time, and that a fairly sizeable dictionary can be created in fewer than 20 hours of developer time. The bootstrapping approach is compared to manual verification at 19.2s and 30s per word. (19.2s was the fastest average time observed in our laboratory using a proficient phonetic transcriber, and represents an optimistic time estimate.)

Also note that the model of expected development time, which was based on measurements of the time taken by Developer B, predicts Developer C's measurements with reasonable accuracy.

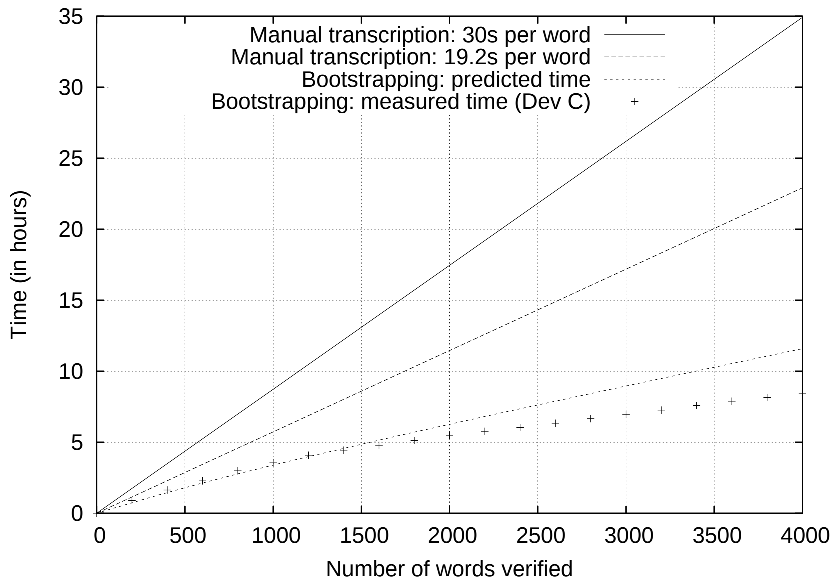


Figure 6.8: *Expected time (in hours) required to compile an Afrikaans pronunciation dictionary, as a function of dictionary size.*

From this set of experiments we conclude that a bootstrapping approach can be used to generate pronunciation dictionaries efficiently. Encouragingly, similar estimates are found for an experienced creator of pronunciation dictionaries (with significant linguistic training), and a developer with no prior exposure to formal linguistics.

6.4 EXPERIMENT B: SEMI-AUTOMATIC DETECTION OF VERIFIER ERRORS

Dictionary developers are typically required to enter phonemic predictions for several thousand words in order to develop dictionaries of sufficient accuracy. Although our interface attempts to assist developers in this task (e.g. by audibly sounding out the chosen pronunciations and by providing automatic predictions for every word), it is inevitable that errors will arise from time to time.

Fortunately, the *Default&Refine* approach is well suited to assist in the detection of such errors. Since every rule in the hierarchy is selected to describe a particular set of words, and errors are likely to result in rules that are applicable to few words besides the erroneous one, one expects that erroneous transcriptions will tend to show up as rules which support few words. Of course, there may also be valid pronunciation rules which are not supported by many examples; it therefore is an experimental issue to determine how useful this guideline is in practically detecting transcription errors. Different languages will differ in this regard – a highly "regular" language such as Spanish [6] will generally have many examples of each valid rule, whereas the idiosyncrasies of English pronunciation will produce a large number of valid special cases. As a consequence, our approach is expected to be more successful for languages such as Spanish.

To investigate the utility of the proposed method for detecting transcription errors, we conduct a number of simulation experiments with Afrikaans. Heuristically, we expect Afrikaans to lie somewhere in the middle of the continuum between regular and irregular languages. Our experiments use a verified dictionary with 4 923 valid words (*Afrikaans A*). Based on earlier experience with dictionary developers who are error prone (see Section 6.3.2.2), we artificially corrupt a fraction of these transcriptions and then measure the efficiency of the number-of-words guideline to indicate the words with corrupted transcriptions. This is the similar to the process followed in Section 4.7.4 where we evaluated the effect of noise on the predictive ability of the *Default&Refine* and *DEC-grow* algorithms.

As in Section 4.7.4 we introduce two types of corruptions into the transcriptions:

- *Systematic corruptions* reflect the fact that users are prone to making certain transcription errors - for example, in the DARPA phone set, *ay* is often used where *ey* is intended. We allow a number of such substitutions, to reflect observed confusions by Afrikaans transcribers.

- *Random corruptions* simulate the less systematic errors that also occur in practice; in our simulations, random insertions, substitutions and deletions of phonemes are introduced.

We generate four corrupted data sets (systematic substitutions; random insertions, substitutions and deletions), where 1% of the words are randomly selected for corruption. *Default&Refine* rule sets are then generated for each case, and the percentage of erroneous words that are matched by the most specific rules are determined[7]. In Fig. 6.9 we show the fraction of errors that remain undetected

[6] That is, a language with a very regular mapping between phonemes and graphemes.

[7] Since *Default&Refine* always applies rules in the order most to least specific, the rule ordering used for prediction was

against the fraction of words examined, as this threshold of specificity is adjusted. Note that this depiction is closely related, but not identical, to that in the well known Detection Error Tradeoff (DET) curves [73].

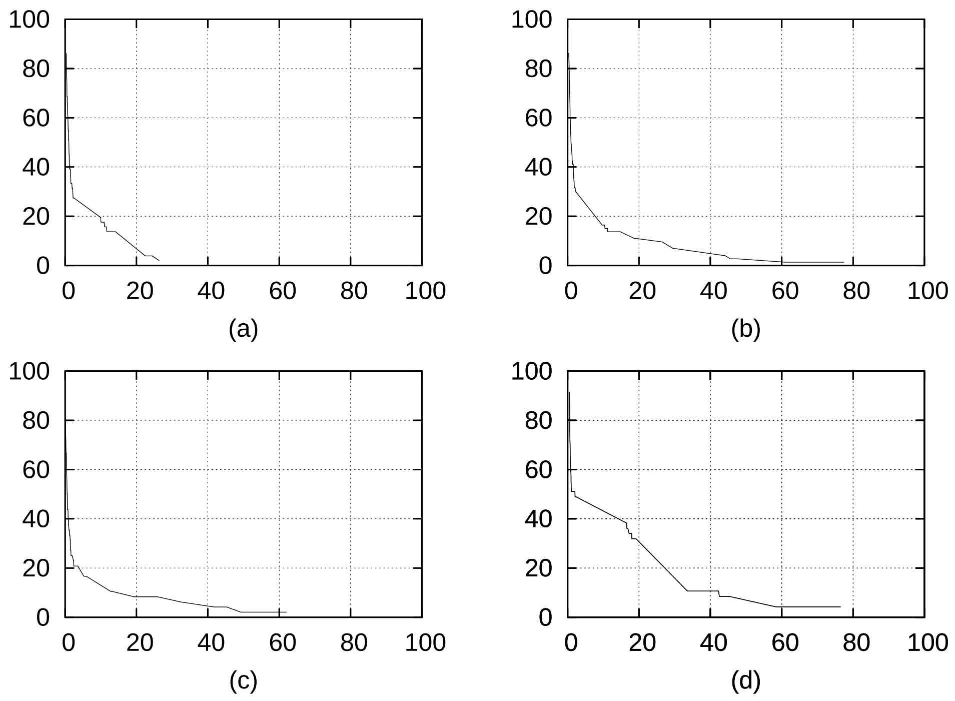


Figure 6.9: *Fraction of erroneous words that are not detected as a function of the fraction of all words examined, when words are examined in the order of their most specific rules, for various types of corruptions: (a) random substitutions (b) random insertions (c) random deletions and (d) systematic substitutions.*

These results suggest that this method has significant use in accelerating the process of error detection. For all three types of random errors, more than 90% of the errors can be identified after inspecting fewer than 20% of the transcriptions. As far as the systematic errors are concerned, about half the errors occur in the first 5% of the words inspected; by that time, the systematic patterns are obvious, and can be used to select other candidate words where these same errors may have occurred.

In practice, the error-detection process can be combined with the synchronisation event, with possible errors flagged by the bootstrapping system and corrected where necessary by a human verifier, prior to continuing with the next session. This then becomes a simple and efficient way of identifying errors during bootstrapping. Alternatively, the error-detection process can be used as a stand-alone technique, in order to identify possible errors in a pronunciation dictionary developed via different means.

used as measure of specificity. The specificity of a word is taken as the specificity of its most specific grapheme, since a transcription error may result in one or more rules becoming highly specific to that word.

6.5 EXPERIMENT C: BUILDING A MEDIUM-SIZED DICTIONARY

In the final controlled experiment we build a medium-sized Afrikaans dictionary utilising the new techniques developed in this thesis. In section 6.5.1 we define our experimental protocol and in the remainder of this section we analyse the efficiency of the process according to the framework defined in Section 3.

6.5.1 EXPERIMENTAL PROTOCOL

Up to this point, the various dictionaries developed during experimentation were fairly small (approximately 1000 to 2000 words). In this experiment, we verify the effectiveness of the various techniques when building a medium-sized dictionary in a continuous process. Since we are growing the dictionary from a previous baseline we are specifically interested in the extent to which the bootstrapping process supports the extension of an existing dictionary.

We utilise one of the developers (Developer C) who has previous experience in using the bootstrapping system. We perform bootstrapping using System B, and initialise the bootstrapping system using the dictionary *Afrikaans A*[8]. We use incremental *Default&Refine* for active learning in between synchronisation sessions, and standard *Default&Refine* during synchronisation. We set the update interval (number of words modified in between synchronisations) to 50, and order words randomly (in the list of new words to be predicted).

At the end of the bootstrapping session we perform error detection. (No additional error detection is performed during bootstrapping.) We first extract the list of graphemic nulls, and identify possible word errors from the graphemic null generators. We then extract *Default&Refine* rules from the full dictionary with the purpose of utilising these rules to identify errors, similar to the process described in Section 6.4. We list all words from word sets that result in a new rule and contain fewer than five words as possible errors, and verify these words manually[9].

6.5.2 HUMAN FACTORS ANALYSIS

We measure the time taken by the verifier (developer C) to perform each verification action, and analyse the effectiveness of the verification process from a human factors perspective. Fig. 6.10 illustrates the verification process as the dictionary grows from 5500 to 7000 words. We plot the time taken to verify each valid word, indicating whether 0,1,2, or 3 corrections are required, for each word as it is added to the dictinary. (The number of training words on the x-axis includes both valid and invalid words.)

We note the following:

[8]As described in Section 4.3, we create the *Afrikaans A* dictionary by cross-analysing the dictionaries from the various experiments run to date and manually verify discrepancies.

[9]A word set associated with a rule tends to have either only one or two words associated with it, or a large set of words: within an acceptable range, the error detection process is not sensitive with regard to the exact cut-off point selected.

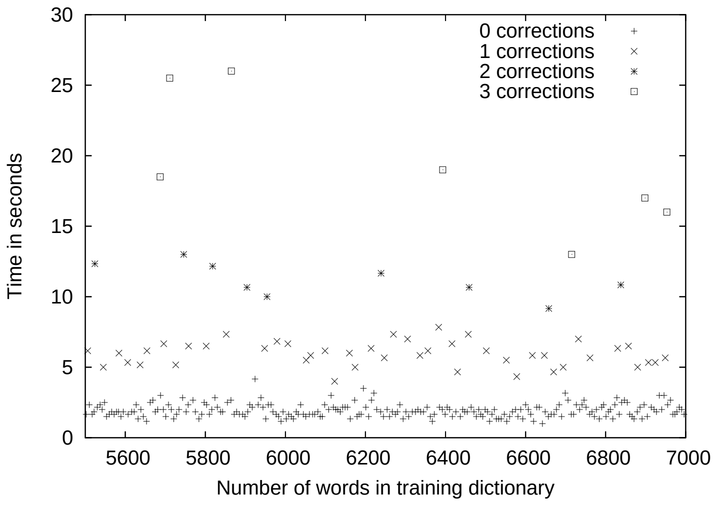


Figure 6.10: *Time taken to verify words requiring zero, one, two or three corrections, as a function of the number of words verified. For the first three measures, the averages were computed for blocks of 5 words each.*

- *User learning curve:* Developer C was proficient in using the system prior to the current bootstrapping session, and further training was not required[10].

- *Cost of intervention:* In this experiment we utilised two intervention mechanisms: verifying predictions, and verifying the list of possible errors. Table 6.2 provides the average verification times observed for Developer C where the intervention mechanism is a single verification of a prediction ($t_{verify(single,s)}$) for words that are in different states s prior to verification. Verification of the list of possible errors took approximately 27 minutes (for approximately 3000 words).

Table 6.2: *Statistics of the time taken to verify words requiring 0,1,2 or 3 errors, or to identify a word as invalid or ambiguous (μ is the mean, and σ the standard deviation.).*

Verdict	Time in seconds	
	μ	σ
correct	1.95	1.35
1 error	5.79	2.30
2 errors	10.74	3.19
3 errors	17.91	6.12
invalid	3.39	4.71
ambiguous	8.92	5.08

- *Task difficulty:* During the bootstrapping process, 3019 words were added to the dictionary, of which 181 were invalid or ambiguous. During error detection, 9 errors were found in the remaining 2838 valid words. Given our analysis in Section 6.4 we estimate that this represents

[10]The value of t_{train} during the initial session was $< 120\ min$.

at least 50% of the errors, and therefore estimate the actual error rate to be 0.6%[11]. It is interesting to note that, while our error detection protocol resulted in a re-verification of 3.3% of the full dictionary (1832 grapheme-specific patterns, or about 300 words), the average position of each error in the ordered error prediction list was at 0.67% of the full training dictionary, with the majority of errors found in the first 0.1% of words, i.e. the first or second pattern on the per-grapheme list of potential errors.

- *Difficulty of manual task:* $error\_rate_{manual}$ is assumed to be $< 0.5\%$, which is an optimistic estimate for the range of manual development speeds evaluated.
- *Manual development speed:* Different values of $t_{develop}$ are used for comparison, ranging from $19.2s$, again an optimistic estimate.
- *Initial set-up cost:* As this is an extension of an existing system, no further set-up cost was incurred[12].

6.5.3 ANALYSIS OF MACHINE LEARNING FACTORS

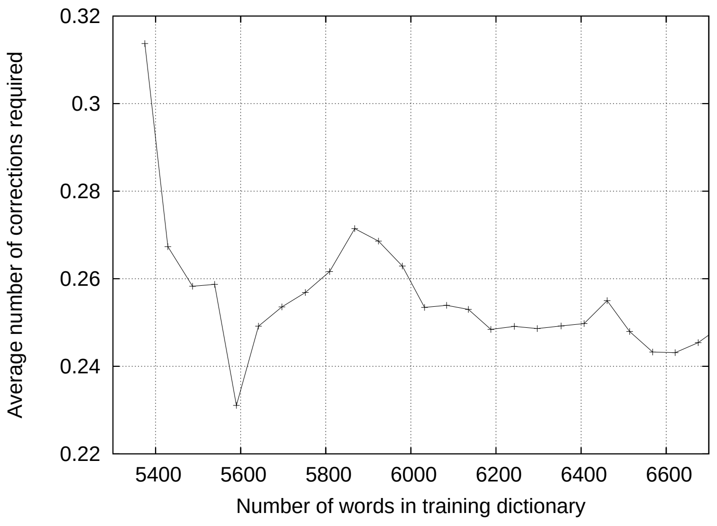


Figure 6.11: *The average number of corrections required as a function of the number of words verified. Averages were computed for blocks of 50 words each.*

From a machine learning perspective, the following is observed:

- *Predictive accuracy of current base:* Measured directly during experimentation, the number of corrections required per word added to the dictionary $(inc\_n(s, n))$ is depicted in Fig. 6.11. We plot the running average (per blocks of 50 words) of the number of corrections as a function of the number of words verified.

[11] 18 errors in 2838 valid words.
[12] In the previous experiment $t_{setup\_bootstrap}$ - $t_{setup\_manual} < 60\ min.$

- *On-line conversion speed:* The average time taken for a synchronisation event was 50.15 seconds ($\sigma = 7.72s$). This value increased gradually from $35s$ during the initial cycle, to $56s$ in the final cycle.
- *Quality and cost of verification mechanisms:* The computational times required for both verification mechanisms are included in the verification times. No additional processing is required.
- *Validity of base data:* $valid\_ratio = 94\%$.

6.5.4 SYSTEM ANALYSIS

Based on our observations during this experiment, we can assign approximate values to the different costs and efficiencies involved during bootstrapping of an Afrikaans dictionary up to 10,000 words. We list these values in Table 6.3.

Table 6.3: *Typical observed values for various bootstrapping parameters.*

Bootstrapping parameter		Estimated value
Training cost	t_{train}	$<$ 120 min
Verification cost for single words, with x corrections required for a word in state s:	$t_{verify(single,s)}$	$(2 + 4.5x)$ sec
Verification cost during error detection (per 1000 words):	$t_{verify(error-det)}$	$<$ 10 min
Verification cost during error detection (per 400 words):	$t_{verify(error-det)}$	$<$ 3 min
Task difficulty - bootstrapping, no error detection	$error\_rate_{bootstrap}$	$0\% - 1\%$
Task difficulty - bootstrapping, error detection	$error\_rate_{bootstrap}$	$0\% - 0.5\%$
Task difficulty - manual	$error\_rate_{manual}$	$0 - 0.5\%$
Manual development speed	$t_{develop}$	$19.2 - 30$ sec
Initial set-up cost	$t_{setup\_bootstrap}$ - t_{setup_manual}	$<$ 60 min

We use eq. 3.4 to analyse our results, and for the single word verifier we combine the values of $t_{auto(s,single)}$ with $t_{verify(s,single)}$ as a single measurement, as discussed in the previous section. We also combine the value of t_{idle} with $t_{verify(error-det)}$, as these two events both occur during synchronisation. We then obtain the following expected cost of N cycles of bootstrapping:

$$E[t_{bootstrap}(N)] = E[t_{setup\_bootstrap}] + E[t_{train}] + E[t_{iterate}(N)] \tag{6.1}$$

$$E[t_{iterate}(N)] = \sum_{x=1}^{N-1}\left(\sum_{s\in status}(E(t_{verify}(s, single)).E(inc\_n(s,x))\right)$$
$$+\sum_{x=1}^{N-1}\left(t_{idle}(inc\_n(valid,x+1)) + t_{verify(error-det)}(inc\_n(valid,x+1))\right) \tag{6.2}$$

We assume an update event after every 100 errors (approximately 400 words verified.) As t_{idle} is dominated by $t_{verify(error-det)}$ during the initial 10,000 words, we keep this value constant as the number of words in the training dictionary increases[13], and estimate it at:

$$t_{verify(error-det)}(400) + t_{idle}(400) = 180 \text{ seconds} \tag{6.3}$$

From Table 6.3 we estimate $E(t_{verify}(s, single))$ as $t_0 + t_e x$ seconds, where x is an indication of the number of corrections required, $t_0 = 2$ and $t_e = 4.5$. In order to estimate $\sum_{x=1}^{N-1} E(t_{verify}(s, single))E(inc\_n(s, x))$ for different states (different numbers of corrections per word) we smooth the number of errors across the training data – as if a word could only have one error – and fit an exponential curve through the accuracy measurements depicted in Fig. 6.11. That is, we assume the probability that the system will predict an error when the training dictionary is of size d is given by $p_e(d)$, where:

$$\begin{aligned} p_e(d) &= P_0 e^{-\frac{d}{k}} \\ \text{i.e. } \log p_e(d) &= \log P_0 - \frac{d}{k} \end{aligned} \tag{6.4}$$

and P_0 and k are parameters to be estimated. The time required for d corrections $T(d)$ (excluding synchronisation events) is then given by:

$$\begin{aligned} T(d) &= \sum_{i=0}^{d-1} (t_0 + t_e P_0) \\ &= dt_0 + t_e P_0 \sum_{i=0}^{d-1} e^{-\frac{d}{k}} \\ &= dt_0 + t_e P_0 \frac{1 - e^{-\frac{d}{k}}}{1 - e^{-\frac{1}{k}}} \end{aligned} \tag{6.5}$$

For the specific data depicted in Fig. 6.11 we obtain the estimates:

$$\begin{aligned} logP_0 = -1.274 \\ -\frac{1}{k} = -3.49 * 10^{-5} \end{aligned} \tag{6.6}$$

We can combine eq. 6.2 and eq. 6.5 in order to estimate the value of $E[t_{iterate}(d/400)]$ for various values of total dictionary size d:

$$\begin{aligned} E[t_{iterate}(d/400)] &= dt_0 + t_e P_0 \frac{1 - e^{-\frac{d}{k}}}{1 - e^{-\frac{1}{d}}} \\ &\quad + \frac{d}{400} * (t_{verify(error-det)}(400) + t_{idle}(400)) \end{aligned} \tag{6.7}$$

[13]This value is influenced by the number of words corrected per cycle – a number that remains constant per cycle.

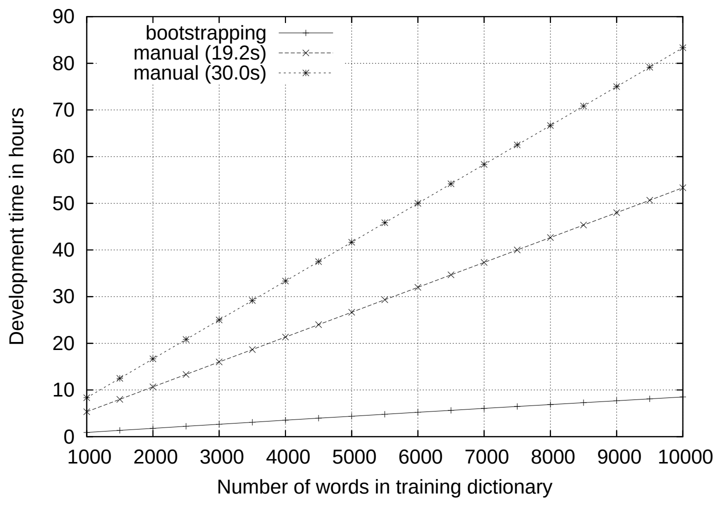


Figure 6.12: *Time estimates for creating different sized dictionaries. Manual development is illustrated for values of $t_{develop}(1)$ of 19.2 and 30 seconds, respectively.*

In Fig. 6.12 we plot eq. 6.7 for different values of d, using the estimates from eq. 6.3 and eq. 6.6. On the same graph we plot the cost of manual dictionary development (again excluding setup cost) using eq. 3.5 and estimates for $t_{develop}(d)$ of 19.2 and 30 seconds, both optimistic estimates. For these estimates we assume that the same base data (or at least data with a similiar validity ratio) is used for both approaches. We also assume that the error rates for the bootstrapping system with error detection and the manual process are approximately equal. In Fig. 6.13 we plot the efficiency estimates of the bootstrapping process as compared to a manual dictionary development process for the same values as Fig. 6.12.

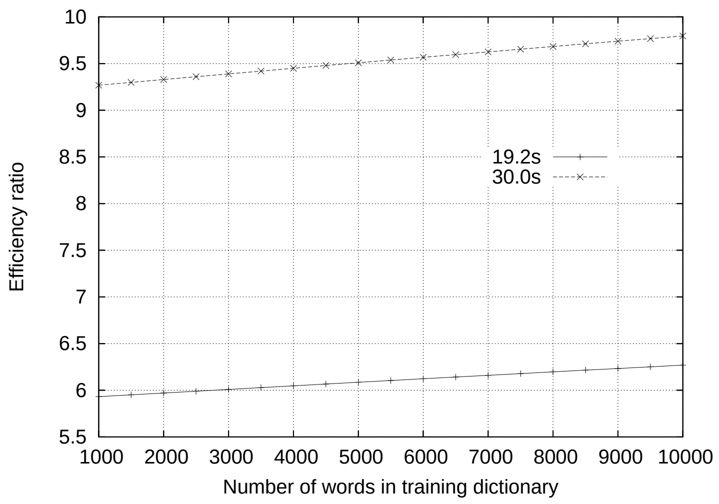


Figure 6.13: *Estimates of the efficiency of bootstrapping, as compared with manual development for values of $t_{develop}(1)$ of 19.2 and 30 seconds, respectively.*

6.6 BUILDING SYSTEMS THAT UTILISE BOOTSTRAPPED DICTIONARIES

In the work up to this point we have verified the bootstrapping process through (1) simulated experiments in which an actual pronunciation dictionary existed, and was utilised as a pseudo-verifier, and (2) by creating multiple dictionaries using different human verifiers and comparing the results. In this section we describe a number of speech technology systems that were developed using the bootstrapped dictionaries.

6.6.1 ISIZULU TEXT-TO-SPEECH

The first system developed using a bootstrapped dictionary was a general purpose text-to-speech (TTS) system developed in the Festival [74] framework as part of the Local Language Speech Technology Initiative (LLSTI) [75], a collaborative project that aims to support the development of speech technology systems in local languages. A small grapheme-to-phoneme rule set was generated using the bootstrapping system and converted to the Festival letter-to-sound format. (The *DictionaryMaker* prototype can automatically export a developed dictionary as either a Festival-formatted lexicon or Festival-formatted letter-to-sound rules.)

The TTS system used the *Multisyn* approach to synthesis and is described in more detail in [76] and [77]. The completed system was evaluated for intelligibility and naturalness by both technologically sophisticated and technologically unsophisticated users, as described in [78].

Table 6.4: *Parameters of the isiZulu text-to-speech dictionary*

Number of graphemes in orthography	26
Number of phonemes in phoneme set	50
Number of words in dictionary	855
Number of derived rules (*DEC-min*)	84

6.6.2 SEPEDI SPEECH RECOGNITION

During 2004, the University of Limpopo collected a first corpus of Sepedi (Northern Sotho) speech with the purpose of creating an automatic speech recognition (ASR) system, and required a pronunciation dictionary in order to proceed with further development. In collaboration with partners from the University of Limpopo, a bootstrapped dictionary was created. Again a fairly small number of words were bootstrapped in order to develop a concise set of letter-to-sound rules. These were then used to develop a speech recognition system using the HTK [79] framework, as described in [80].

6.6.3 AFRIKAANS TEXT-TO-SPEECH

Much of the initial experimentation with the bootstrapping approach was performed for Afrikaans, as described in previous sections of this thesis. The Afrikaans dictionary was used to develop a

Table 6.5: *Parameters of the Sepedi speech recognition dictionary*

Number of graphemes in orthography	27
Number of phonemes in phoneme set	41
Number of words in dictionary	2827
Number of derived rules (*DEC-min*)	90

Afrikaans TTS system for the South African Centre for Public Service Innovation (CPSI), who are using the voice to pilot a system that will allow citizens to interact with a governmental service that deals with passport applications via a number of interaction mechanisms not previously available. One of the mechanisms tested includes the use of cellphone based Short Message Service (SMS) to communicate, and converting such SMSs to voice when a user prefers a voice-based service – mainly in order to ensure accessibility to all citizens, including illiterate system users, and system users with specific disabilities. This system is currently being piloted.

Table 6.6: *Parameters of the Afrikaans text-to-speech dictionary*

Number of graphemes in orthography	40
Number of phonemes in phoneme set	43
Number of words in dictionary	7782
Number of derived rules (*Default&Refine*)	1471

6.6.4 OTHER SYSTEMS

The CPSI pilot project described above aims to provide services in four languages: English, Afrikaans, isiZulu and Sepedi; a Sepedi voice similar to those described in Sections 6.6.1 and 6.6.3 was therefore developed, using the dictionary built as described in Section 6.6.2. Further development on the Sepedi voice is currently under way, specifically aimed at improving the intonation contours of the current voice.

Furthermore, an initial isiZulu ASR system and an Afrikaans ASR system were developed, with further optimisation currently in progress. A first Setswana dictionary was developed, and will be refined and integrated in similar systems as part of the OpenPhone [81] project, a project sponsored by the International Development Research Centre (IDRC) and the Open Society Initiative (OSI), which aims to make telephony services more accessible to information service providers in the developing world.

6.7 CONCLUSION

In this section we demonstrated the practical application of the bootstrapping system, evaluating the efficiency of the approach from both a human factors and a machine learning perspective. We found that, even with optimistic estimates for the time required to develop a single instance of a pronunciation dictionary manually, the bootstrapping process provides a significant cost saving, as illustrated in Fig. 6.12. We also described a number of speech technology systems developed using newly bootstrapped dictionaries. In the next chapter (Chapter 7) we discuss the implications of our results.

Chapter Seven

Conclusion

7.1 INTRODUCTION

As initially discussed in Section 1.4, the aim of this thesis was two-fold: (a) to obtain a mechanism for pronunciation modelling that is well suited to bootstrapping; and (b) to analyse the bootstrapping of pronunciation models from a theoretical and a practical perspective, as a case study in the bootstrapping of HLT resources. In this chapter we evaluate the extent in which we were able to reach these goals. We summarise the contribution of this thesis, and discuss future work.

7.2 SUMMARY OF CONTRIBUTION

This thesis was able to demonstrate conclusively that the proposed bootstrapping approach is a practical and cost-efficient way to develop pronunciation dictionaries in new languages. The specific contributions made in the course of this research are the following:

- A demonstration of a fully interactive (on-line) bootstrapping approach to the development of pronunciation dictionaries, in Section 6.5 [82].
- Development and evaluation of a practical system that allows users (without specialist linguistic expertise) to develop such pronunciation dictionaries, and an analysis of the factors that influence this process, in Section 6.3 [83, 84].
- The development of *Default&Refine*, a new algorithm for grapheme-to-phoneme prediction, in Section 4.6 [85]. This algorithm has a number of desirable features, including language independence, rapid generalisation from small data sets, good asymptotic accuracy, robustness to human error, and the production of compact rule sets.

- A number of algorithmic refinements to ensure a practical bootstrapping system, including optimised alignment and an incremental (on-line learning) version of the g-to-p algorithm used during bootstrapping, in Sections 4.4.2 and 4.6.4 [84, 86].
- The development and evaluation of a novel error-detection tool that can assist in the verification of pronunciation dictionaries – both during bootstrapping and in support of alternative dictionary development approaches, in Section 6.4 [86].
- Definition of a conceptual framework that can be used to describe the bootstrapping process in general, and the bootstrapping of pronunciation dictionaries in particular, in Chapter 3.
- Development of usable pronunciation dictionaries in a number of South African languages (isiZulu, Sepedi, Afrikaans and Setswana), and the integration of these dictionaries in actual speech technology (speech recognition and speech synthesis) systems, in Section 6.6.
- The development of *minimal representation graphs*: a theoretical framework that supports the rigorous analysis of instance-based learning of rewrite rule sets, in Section 5. This framework aims to derive the smallest possible rule set describing a given set of discrete training data.

7.3 FURTHER APPLICATION AND FUTURE WORK

The current thesis forms the basis for three main directions of future research, related to (1) the process of bootstrapping pronunciation dictionaries, (2) grapheme-to-phoneme conversion, and (3) further refinement of the *minimal representation graph* framework.

The current bootstrapping process provides an effective platform for the development of pronunciation dictionaries but further gains are likely to arise from future improvements. Specific issues that we would like to address in future include:

- Active learning during bootstrapping: determining optimal ways in which to choose the next instance or set of instances to utilise during bootstrapping.
- An evaluation of the implications of different initialisation mechanisms, for example when a limited rule set is known prior to dictionary creation, or when a pronunciation dictionary exists in a phonologically similar language.
- Further analysis of the ways in which algorithmic requirements change for different phases of the bootstrapping process.
- Practical support for phone set manipulation during bootstrapping, including re-bootstrapping of appropriate sections of the dictionary after phone set manipulation.
- Support for the bootstrapping of other linguistic entities such as intonation, stress or hyphenation.

G-to-p conversion algorithms in general have been well studied, especially with regard to asymptotic accuracy and computational complexity. However, little work has been published to date in evaluating and improving initial learning efficiency (accuracy when trained on very small data sets) and robustness to noise (transcription errors occurring in the training dictionaries) – two aspects that are of importance during bootstrapping. We are interested whether further improvements may be obtained from the following sources:

- Adapting the algorithm (or its parameters) according to the specific grapheme extracted. As all rule extraction and rule application occurs on a per-grapheme basis, it should be possible to introduce further algorithmic refinements suitable to the characteristics of the specific grapheme being considered. We would like to analyse the current graphemic behaviour in further detail.
- Utilising this algorithm within a framework that includes additional data sources (such as part-of-speech tags).
- Learning from and predicting multiple pronunciations (incorporating word-level pronunciation variants).
- Incorporating class-based learning in the current algorithm: combining graphemes according to predictive behaviour in such a way that learning is accelerated.
- Investigating the threshold for valuable exceptions. In Section 6.4 it was clearly shown that the effect of errors in the training data tend to accumulate in the last $10 - 20\%$ of rules extracted. For *Default&Refine* specifically (and for noisy training sets) all exceptions may not contribute to predictive accuracy.

Some of the above questions related to grapheme-to-phoneme conversion may be better analysed in terms of the *minimal representation graph* framework. The current framework provides a theoretical basis for understanding the task of instance-based learning of rewrite rules. Further work related to this framework specifically include:

- Further development of the set of allowed operators utilising the framework, as well as a rigorous analysis of the legality and optimality of the set of operators.
- The application of established techniques related to the solution of constraint satisfaction problem, in order to improve the computational tractability of the current graph solution process. This will be required before a rigorous evaluation of the extracted rule sets on larger training dictionaries will become possible.

Additionally, the *Default&Refine* algorithm provides an interesting perspective on the grapheme-to-phoneme conversion task, viewing pronunciation as a hierarchy of regularity – with systematic instances and exceptions occuring in a continuum of regularity. We are interested in applying the same algorithm to other natural language processing tasks that exhibit similar behaviour.

7.4 CONCLUSION

This thesis has developed a number of tools in support of the bootstrapping process, and has demonstrated the value of this approach for the practical and cost-effective development of pronunciation dictionaries. Human language technologies have great potential value in the developing world, and bootstrapping will undoubtedly play a significant role in accelerating the development of such technologies. We therefore hope that theoretical interest and practical importance will continue to drive developments in this area.

APPENDIX A

THE ARPABET PHONE SET

The ARPAbet phone set was developed as part of the ARPA Speech Understanding project (1971-1976), and is included with the TIMIT speech corpus [87].

Table A.1: *ARPAbet phone set [87]*

no		example	no		example	no		example
1	iy	beat	22	r	red	43	zh	measure
2	ih	bit	23	y	yet	44	sh	shoe
3	eh	bet	24	w	wet	45	v	very
4	ae	bat	25	m	mom	46	f	fief
5	ix	roses	26	em	buttom	47	dh	they
6	ax	the	27	n	non	48	th	thief
7	ah	butt	28	nx	flapped n	49	hh	hay
8	uw	boot	29	en	button	50	hv	Leheigh
9	uh	book	30	ng	sing	51	dcl	d closure
10	ao	about	31	eng	Washington	52	bcl	b closure
11	aa	cot	32	ch	church	53	gcl	g closure
12	er	bird	33	jh	judge	54	tcl	t closure
13	axr	diner	34	b	bob	55	pcl	p closure
14	ey	bait	35	p	pop	56	kcl	k closure
15	ay	bite	36	d	dad	57	q	glottal stop
16	oy	boy	37	dx	butter	58	epi	epinthetic closure
17	aw	bought	38	t	tot	59	qcl	d closure
18	ow	boat	39	g	gag	60	h#	begin silence
19	ux	beauty	40	k	kick	61	#h	end silence
20	l	led	41	z	zoo	62	pau	between silence
21	el	bottle	42	s	sis			

APPENDIX B

SOME THEOREMS REGARDING MINIMAL REPRESENTATION GRAPHS

This appendix contains a number of proofs supporting the arguments in Chapter 5.

B.1 WORD SETS

Statement 1 (context_implies_matchwords)

$$\begin{aligned}\forall r, s \in Z', Z' \subseteq Z_{combined} :\\ context(r) \supset context(s) \implies\\ matchwords(r) \subseteq matchwords(s).\end{aligned} \tag{B.1}$$

Consider any word $w \in matchwords(r)$, then $match(w, r) = 1$ (eq. 5.34), and then $context(w) \supseteq context(r)$ (eq. 5.11). Now also $context(w) \supseteq context(r) \supset context(s)$, so $match(w, s) = 1$ (eq. 5.34), and then $w \in matchwords(s)$. Since this holds for all $w \in matchwords(r)$, $matchwords(r) \subseteq matchwords(s)$.

Statement 2 (subset_transitive)

$$\begin{aligned}\forall r, s \in Z', v \in Z_e, Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z') :\\ path(subset(Z', Z_e, oset(Z'), v, r, s)) = 1 \implies\\ subset(Z', Z_e, oset(Z'), v, r, s) = 1.\end{aligned} \tag{B.2}$$

If $path(subset(Z', Z_e, oset(Z'), v, r, s)) = 1$, then there exists $n \geq 2$ rules $t_1 = r, t_2, ..., t_n = s$ such that for each t_i, t_{i+1} pair it holds that $subset(Z', Z_e, oset(Z'), v, t_i, t_{i+1}) = 1$. Then $possible\_words(Z', Z_e, oset(Z'), v, t_i = r) \subset ... \subset possible\_words(Z', Z_e, oset(Z'), v, t_i) \subset possible\_words(Z', Z_e, oset(Z'), v, t_{i+1}) \subset ... \subset possible\_words(Z', Z_e, oset(Z'), v, t_n = s)$, and then $subset(Z', Z_e, oset(Z'), v, r, s) = 1$ (from eq, 5.51).

Statement 3 (possible_words_rulewords)

$$\begin{aligned} \forall r, s \in Z_e, Z_e \subseteq Z_{combined}, \forall oset(Z_e) \subseteq allset(Z_e) : \\ w \in possible\_words(Z_e, Z_e, oset(Z_e), r, r) \iff \\ w \in rulewords(Z_e, oset(Z_e), r). \end{aligned} \tag{B.3}$$

If $w \in possible\_words(Z_e, Z_e, oset(Z_e), r, r)$, then $match(w, r) = 1$ and there exists no rule $s \in Z_e$ such that $match(w, s) = 1$ and $(s, r) \in oset(Z_e)$ (eq. 5.36). Then $r \in winningrule(Z', oset(Z'), w)$ (eq. 5.12), and then $w \in rulewords(Z_e, oset(Z_e), r)$ (eq. 5.35); and vice versa.

Statement 4 (words_relations)

$$\begin{aligned} \forall r \in Z', Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z'), \\ \forall v \in Z_e, v = r \text{ or } (v, r) \in oset(Z') : \\ rulewords(Z', oset(Z'), r) \subseteq \\ possible\_words(Z', Z_e, oset(Z'), v, r) \subseteq \\ matchwords(r). \end{aligned} \tag{B.4}$$

If $w \in rulewords(Z', oset(Z'), r)$, then $match(w, r) = 1$ and there exists no $s \in Z'$ such that $match(w, s) = 1$ and $(s, r) \in oset(Z')$ (eq. 5.35 and eq. 5.12). Then, since $Z_e \subseteq Z'$, there also exists no such $s \in Z_e$, and then $w \in possible\_words(Z', Z_e, oset(Z'), v, r)$ with $v = r$ the only valid value for v (eq. 5.36). For any w' in $possible\_words(Z', Z_e, oset(Z'), v, r)$ it also holds by definition that $match(w', r) = 1$ (eq. 5.36), so w' in $matchwords(r)$ (eq. 5.34).

Statement 5 (complement_directpath)

$$\begin{aligned} \forall r, s \in Z', Z_e \subseteq Z' \subseteq Z_{combined}, \\ \forall oset(Z') \subseteq allset(Z') : \\ complement(Z', Z_e, oset(Z'), r, s) = 1 \implies \\ mincomp(Z', Z_e, oset(Z'), r, s)) = 1 \\ \text{or } path(containpat(Z', r, s)) = \pm 1. \end{aligned} \tag{B.5}$$

Let $r, s \in Z'$ be any two rules such that $complement(Z', Z_e, oset(Z'), r, s) = 1$; and consider all the options for a $containpat$ path between r and s. If $path(containpat(Z', r, s)) = \pm 1$ the statement holds. If neither $path(containpat(Z', r, s)) = 1$ nor $path(containpat(Z', r, s)) = -1$, then, since $complement(Z', Z_e, oset(Z'), r, s) = 1$, it follows that $mincomp(Z', Z_e, oset(Z'), r, s) = 1$ by definition (eq. 5.49).

Statement 6 (rulewords_sub_rulewords)

$$\begin{aligned} \forall r \in Z', Z' \subseteq Z_{combined}, \\ \forall oset(Z') \subseteq oset'(Z') \subseteq allset(Z') : \\ rulewords(Z', oset'(Z'), r) \subseteq rulewords(Z', oset(Z'), r). \end{aligned} \tag{B.6}$$

Let w be any word such that $w \in rulewords(Z', oset'(Z'), r)$. By definition (eq 5.35 and eq. 5.12) it follows that $match(w, r) = 1$ and there exists no s such that $match(w, s) = 1$ and $(s, r) \in oset'(Z')$. Now $oset(Z') \subseteq oset'(Z')$, which means that $oset(Z')$ has fewer restrictions than $oset'(Z')$ and if $(s, r) \notin oset'(Z')$ for an s as above, then also $(s, r) \notin oset(Z')$, and then $w \in rulewords(Z', oset(Z'), r)$. Since this holds for any $w \in rulewords(Z', oset'(Z'), r)$, $rulewords(Z', oset'(Z'), r) \subseteq rulewords(Z', oset(Z'), r)$.

Statement 7 (rulewords_redundant)

$$\begin{aligned} \forall r \in Z', Z' \subseteq Z_{combined}, \forall oset'(Z') \subseteq allset(Z') : \\ \exists oset(Z') \subseteq oset'(Z') : rulewords(Z', oset(Z'), r) = \phi \iff \\ \text{r is a redundant rule in } Z', oset'(Z'). \end{aligned} \tag{B.7}$$

For any $oset'(Z') \supseteq oset(Z')$, it follows directly from statement 6 that if $rulewords(Z', oset(Z'), r) = \phi$ also $rulewords(Z', oset'(Z'), r) = \phi$ (since $rulewords(Z', oset'(Z'), r) \subseteq rulewords(Z', oset'(Z'), r)$). In all orderings that include $oset(Z')$, rule r will never be invoked to predict a word. Also, if r is a redundant rule in $oset'(Z')$ then $rulewords(Z', oset'(Z'), r) = \phi$. For at least $oset(Z') = oset'(Z')$, but possibly also for other sets of orderings, it then holds that $rulewords(Z', oset(Z'), r) = \phi$.

B.2 CHARACTERISTICS OF Z_M

Statement 8 (possibly_minimal_single)

$$\begin{aligned} &\forall r \in Z_m, Z_m \subseteq Z_{combined} : \\ &possibly\_minimal(Z_m) = 1 \implies r \in Z_{single} \end{aligned} \tag{B.8}$$

Consider any $r \in Z_m \subseteq Z_{combined}$. Then $r \in Z_{conflict-resolved} \cup Z_{no-conflict} \cup Z_{conflict-combined}$ (eq. 5.21). Since $possibly\_minimal(Z_m) = 1$, there exists an ordering $oset(Z_m)$ such that $minimal(Z_m, oset(Z_m)) = 1$ (eq. 5.26). According to this ordering, rule r will be invoked by at least one word w in TD, otherwise r would be a redundant rule in a $minimal$ rule set, which is impossible. If $r \in Z_{conflict-combined}$, r would predict w with at least two different outcomes depending on which of the specific alternative outcomes are selected (eq. 5.19). Since this is not possible in an $accurate$ set of rule orderings, $minimal(Z_m, oset(Z_m)) \neq 1$, and then it is not possible to find the required $oset(Z_m)$, and Z_m cannot be a $possibly\_minimal$ rule set. For $possibly\_minimal(Z_m) = 1$ to hold, $r \notin Z_{conflict-combined}$, and then $r \in Z_{conflict-resolved} \cup Z_{no-conflict}$, which is the same as stating that $r \in Z_{single}$ (eq. 5.22). Note that any $possibly\_minimal$ rule set Z_m can therefore be assumed to be a subset of Z_{single}.

Statement 9 (rulewords_eq_invalid)

$$\begin{aligned} &\forall r, s \in Z_m, Z_m \subseteq Z_{single}, possibly\_minimal(Z_m) = 1, \\ &\forall oset(Z_m) \subseteq allset(Z_m) : valid(oset(Z_m)) = 1 \implies \\ &rulewords(Z_m, oset(Z_m), r) \neq rulewords(Z_m, oset(Z_m), s). \end{aligned} \tag{B.9}$$

Consider any rules r and s ordered according to the $valid$ ordering $oset(Z_m)$, and let $oset'(Z_m)$ be the final ordering used during word prediction, where $oset'(Z_m) \supseteq oset(Z_m)$. Irrespective of whether there is a $rule\_order(.)$ relation between rules r and s or not, or the existence of any additional rules; in the final rule numbering assignment based on $oset'(Z_m)$, either $rulenum(r) < rulenum(s)$ or $rulenum(s) < rulenum(r)$. Choose r to be the rule such that $rulenum(r) < rulenum(s)$, and let $rulewords(Z_m, oset(Z_m), r) = rulewords(Z_m, oset(Z_m), s)$. Let w be any word pattern in TD such that $w \in rulewords(Z_m, oset'(Z_m), s)$. At least one such a word pattern must exist, otherwise (from statement 7) s is a redundant rule in a $possibly\_minimal$ rule set. Since, from statement 6, $rulewords(Z_m, oset'(Z_m), s) \subseteq rulewords(Z_m, oset(Z_m), s) = rulewords(Z_m, oset(Z_m), r)$, it follows that $match(w, r) = 1 = match(w, s)$ (eq. 5.35). For any additional t such that $match(w, t) = 1$, one of the following situations can occur: (1) no such t exists, (2) $(t, r) \notin oset'(Z_m), (t, s) \notin oset'(Z_m)$, (3) $\{(t, r), (t, s)\} \in oset'(Z_m)$, (4) $(t, r) \in oset'(Z_m)), (t, s) \notin oset(Z_m)$, or (5) $(t, r) \notin oset(Z_m), (t, s) \in oset(Z_m)$. If (1) or (2) occurs, then $w \in rulewords(Z_m, oset'(Z_m), r)$ and $w \in rulewords(Z_m, oset'(Z_m), s)$ (eq.

5.35), but since $rulenum(r) < rulenum(s)$, rule s will never be invoked to predict word pattern w. If (3) occurs, then rule t will always be invoked to predict word pattern w, and neither rule r or s will be invoked for this purpose. If (4) occurs, then either t or s can be invoked, but since $rulenum(t) < rulenum(r) < rulenum(s)$ only t will be invoked. If (5) occurs, either rule t or r can be invoked, but again rule s will not be invoked to predict word pattern w. Rule s will therefore never be invoked to predict word pattern w, irrespective of the option that occurs. Since this holds for all $w \in rulewords(Z_m, oset'(Z_m), s)$, rule s is a redundant rule in a $possibly\_minimal$ rule set, so $valid(oset'(Z_m)) \neq 1$. But $oset'(Z_m)$ was not restricted in any other way than by requiring that $oset'(Z_m) \supseteq oset(Z_m)$ where $rulewords(Z_m, oset(Z_m), r) = rulewords(Z_m, oset(Z_m), s)$. It is therefore not possible that $rulewords(Z_m, oset(Z_m), r) = rulewords(Z_m, oset(Z_m), s)$. If $rulewords(Z_m, oset(Z_m), r) \neq rulewords(Z_m, oset(Z_m), s)$ then it is possible that a word $w' \in rulewords(Z_m, oset(Z_m), s)$ exists such that $match(w', s) = 1, match(w', r) \neq 1$, and that s can be invoked to predict w' irrespective of whether $rulenum(r) < rulenum(s)$; and a similar contradiction does not occur. Exactly the same argument holds if s is chosen to be the rule such that $rulenum(s) < rulenum(r)$.

Statement 10 (poswords_redundant)

$$\begin{aligned}
&\forall r, s \in Z_m, r \neq s, Z_m \subseteq Z_{combined}, \forall oset(Z_m) \subseteq allset(Z_m) : \\
&\qquad minimal(Z_m, oset_m(Z_m)) = 1 \implies \\
&\forall s \in Z_m : possible\_words(Z_m, Z_m, oset_m(Z_m), r, r) \not\equiv \\
&\qquad possible\_words(Z_m, Z_m, oset_m(Z_m), r, s).
\end{aligned} \tag{B.10}$$

From statement 3 and 7 it follows that if $minimal(Z_m, oset_m(Z_m)) = 1$, then $possible\_words(Z_m, Z_m, oset_m(Z_m), r, r) = rulewords(Z_m, oset_m(Z_m), r) \neq \phi$. If $(r, s) \notin oset_m(Z_m)$, then (since $r \neq s$) $possible\_words(Z_m, Z_m, oset_m(Z_m), r, s) = \phi$ (eq. 5.36), and then the statement holds. Now consider the situation if $(r, s) \in oset_m(Z_m)$: $possible\_words(Z_m, Z_m, oset_m(Z_m), r, r) \equiv possible\_words(Z_m, Z_m, oset_m(Z_m), r, s)$ implies that for each word pattern $w \in possible\_words(Z_m, Z_m, oset_m(Z_m), r, s)$ also $match(w, r) = 1$. Then, if $(r, s) \in oset_m(Z_m)$, $possible\_words(Z_m, Z_m, oset_m(Z_m), s, s) = \phi$ (eq. 5.36). But then $rulewords(Z_m, oset_m(Z_m), s) = \phi$ (statement 3) and then s becomes a redundant rule (statement 7), which is impossible since $s \in Z_m$. Then $possible\_words(Z_m, Z_m, oset_m(Z_m), r, r) \not\equiv possible\_words(Z_m, Z_m, oset_m(Z_m), r, s)$, and the statement again holds.

B.3 Z_M AS A SUBSET OF $Z_{COMBINED}$

Statement 11 (poswords_sub_poswords)

$$\begin{aligned}
&\forall r \in Z_B, v \in Z_b, \forall Z_a \subseteq Z_b \subseteq Z_B \subseteq Z_A \subseteq Z_{combined},\\
&\forall oset_A(Z_A) \subseteq allset(Z_A), \forall oset_B(Z_B) \subseteq allset(Z_B),\\
&order\_subset(oset_A(Z_A), oset_B(Z_B)) = 1 :\\
&possible\_words(Z_B, Z_b, oset_B(Z_B), v, r) \subseteq\\
&possible\_words(Z_A, Z_a, oset_A(Z_A), v, r).
\end{aligned} \tag{B.11}$$

Let w be any word such that $w \in possible\_words(Z_B, Z_b, oset_B(Z_B), v, r)$. By definition (eq 5.36) it follows that $match(w, r) = 1$ and that there exists no $s \in Z_b$ such that $match(w, s) = 1$ and $\{(s, r), (s, v)\} \in oset_B(Z_B)$. Now $order\_subset(oset_A(Z_A), oset_B(Z_B)) = 1$, which means that $oset_A(Z_A)$ has fewer restrictions than $oset_B(Z_B)$ and, if for no such matching $s \in Z_b$ it holds that $(s, r) \in oset_B(Z_B)$, there also exists no such $s \in Z_b$ such that $\{(s, r), (s, v)\} \in oset_A(Z_A)$ (eq. 5.27). And since $Z_a \subseteq Z_b$, there also exists no such $s \in Z_a$, and then $w \in possible\_words(Z_A, Z_a, oset_A(Z_A), v, r)$ (eq. 5.36). Since this holds for all w in $possible\_words(Z_B, Z_b, oset_B(Z_B), v, r)$, it follows that $possible\_words(Z_B, Z_b, oset_B(Z_B), v, r) \subseteq possible\_words(Z_A, Z_a, oset_A(Z_A), v, r)$.

Statement 12 (sharedwords_sub_sharedwords)

$$\begin{aligned}
&\forall r \in Z_B, v \in Z_b, \forall Z_a \subseteq Z_b \subseteq Z_B \subseteq Z_A \subseteq Z_{combined},\\
&\forall oset_A(Z_A) \subseteq allset(Z_A), \forall oset_B(Z_B) \subseteq allset(Z_B),\\
&order\_subset(oset_A(Z_A), oset_B(Z_B)) = 1 :\\
&shared\_words(Z_B, Z_b, oset_B(Z_B), v, r, s) \subseteq\\
&shared\_words(Z_A, Z_a, oset_A(Z_A), v, r, s).
\end{aligned} \tag{B.12}$$

Let w be any word pattern such that $w \in shared\_words(Z_B, Z_b, oset_B(Z_B), v, r, s)$. Then $w \in possible\_words(Z_B, Z_b, oset_B(Z_B), v, r)$ and $w \in possible\_words(Z_B, Z_b, oset_B(Z_B), v, s)$ (eq. 5.42). Since $order\_subset(oset_A(Z_A), oset_B(Z_B)) = 1$, it follows from statement 11 that also $w \in possible\_words(Z_A, Z_a, oset_A(Z_A), v, r)$ and $w \in possible\_words(Z_A, Z_a, oset_A(Z_A), v, s)$, and then $w \in shared\_words(Z_A, Z_a, oset_A(Z_A), v, r, s)$ (eq. 5.42). As this holds for all $w \in shared\_words(Z_B, Z_b, oset_B(Z_B), v, r, s)$, $shared\_words(Z_B, Z_b, oset_B(Z_B), v, r, s) \subseteq shared\_words(Z_A, Z_a, oset_A(Z_A), v, r, s)$.

Statement 13 (poswords_sub_poswords_later_v)

$$\begin{aligned}\forall r \in Z', Z_a \subseteq Z_b \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z'),\\ \forall v \in Z_a, v = r \text{ or } (v,r) \in oset(Z'),\\ \forall t \in Z_b, t = v \text{ or } \{(v,t),(t,r)\} \in oset'(Z') :\\ possible\_words(Z', Z_b, oset(Z'), t, r) \subseteq\\ possible\_words(Z', Z_a, oset(Z'), v, r).\end{aligned} \tag{B.13}$$

Consider any word pattern $w \in possible\_words(Z', Z_b, oset(Z'), t, r)$. Then $match(w,r) = 1$ and there exists no $q \in Z_b$ such that $match(w,q) = 1$ and $q = t$ or $(q,t) \in oset(Z')$. Since $t = v$ or $(v,t) \in oset(Z')$, there therefore also exists no such matching $q \in Z_b$ such that $q = v$ or $(q,v) \in oset(Z')$. Since $Z_a \subseteq Z_b$, there also exists no such $q \in Z_a$. By definition then $w \in possible\_words(Z', Z_a, oset(Z'), v, r)$ (eq. 5.36). Since this holds for all word patterns $w \in possible\_words(Z', Z_b, oset(Z'), t, r)$, it follows that $possible\_words(Z', Z_a, oset(Z'), t, r) \subseteq possible\_words(Z', Z_b, oset'(Z'), v, r)$.

Statement 14 (poswords_sub_poswords_later_all)

$$\begin{aligned}\forall r \in Z_B, v \in Z_b, \forall Z_a \subseteq Z_b \subseteq Z_B \subseteq Z_A \subseteq Z_{combined},\\ \forall oset_A(Z_A) \subseteq allset(Z_A), \forall oset_B(Z_B) \subseteq allset(Z_B),\\ order\_subset(oset_A(Z_A), oset_B(Z_B)) = 1,\\ \forall v \in Z_a, v = r \text{ or } (v,r) \in oset(Z'),\\ \forall t \in Z_b, t = v \text{ or } \{(v,t),(t,r)\} \in oset'(Z') :\\ possible\_words(Z_B, Z_b, oset_B(Z_B), t, r) \subseteq\\ possible\_words(Z_A, Z_a, oset_A(Z_A), v, r).\end{aligned} \tag{B.14}$$

Since $possible\_words(Z_B, Z_b, oset_B(Z_B), t, r) \subseteq possible\_words(Z_A, Z_a, oset_A(Z_A), t, r)$ (statement 11), and $possible\_words(Z_A, Z_a, oset_A(Z_A), t, r) \subseteq possible\_words(Z_A, Z_a, oset_A(Z_A), v, r)$ (statement 13, choosing $Z_a \equiv Z_b$), it follows that $possible\_words(Z_B, Z_b, oset_B(Z_B), t, r) \subseteq possible\_words(Z_A, Z_a, oset_A(Z_A), v, r)$.

Statement 15 (relations_superset_minset)

$$\forall r,s,v \in Z_m, Z_e \subseteq Z_m \subseteq Z' \subseteq Z_{combined},$$
$$\forall oset(Z') \subseteq allset(Z'), oset_m(Z_m) \subseteq allset(Z_m),$$
$$allowed\_state(Z', Z_e, oset(Z')) = 1$$
$$\forall (Z_m, oset_m(Z_m)) \in minrules(Z', Z_e, oset(Z')) :$$
$$containpat(Z', r, s) = 1 \implies containpat(Z_m, r, s) = 1. \tag{B.15}$$
$$containpat(Z', r, s) = -1 \implies containpat(Z_m, r, s) = -1. \tag{B.16}$$
$$path(containpat(Z_m, r, s)) = 1 \iff path(containpat(Z', r, s)) = 1. \tag{B.17}$$
$$complement(Z_m, Z_m, oset(Z_m), v, r, s) = 1 \implies$$
$$complement(Z', Z_e, oset(Z'), v, r, s) = 1. \tag{B.18}$$
$$mincomp(Z_m, Z_m, oset(Z_m), v, r, s) = 1 \implies$$
$$mincomp(Z', Z_e, oset(Z'), v, r, s) = 1. \tag{B.19}$$

Consider each relationship separately, and note that this relates specifically to $r, s, v \in Z_m$:

- Eq. B.15: If $containpat(Z', r, s) = 1$, then $context(r) \supset context(s)$ and no $t \in Z'$ exists such that $context(r) \supset context(t) \supset context(s)$ (eq. 5.47). Since $Z_e \subseteq Z_m \subseteq Z'$, no such t can exist in Z_m either, and then $containpat(Z_m, r, s) = 1$.

- Eq. B.16: If $containpat(Z', r, s) = -1$, then $containpat(Z', s, r) = 1$. Then $containpat(Z_m, s, r) = 1$ (from eq. B.15), and then $containpat(Z_m, r, s) = -1$.

- Eq. B.17: If $path(containpat(Z_m, r, s)) = 1$, then $context(r) \supset context(s)$ (eq. 5.48) and then $path(containpat(Z', r, s)) = 1$; and vice versa.

- Eq. B.18: If $complement(Z_m, Z_m, oset(Z_m), v, r, s) = 1$ then there exists a rule $v \in Z_m$ and word w such that $w \in shared\_words(Z_m, Z_m, oset(Z_m), v, r, s)$ (eq 5.43). Since $order\_subset(oset(Z'), oset(Z_m)) = 1$ and $Z_e \subseteq Z_m \subseteq Z'$ by definition (eq. 5.29), it follows from statement 12 that also $w \in shared\_words(Z', Z_e, oset(Z'), v, r, s)$, and then $complement(Z', Z_e, oset(Z'), v, r, s) = 1$ (eq 5.43).

- Eq. B.19: If $mincomp(Z_m, Z_m, oset(Z_m), v, r, s) = 1$ then $complement(Z_m, Z_m, oset(Z_m), v, r, s) = 1$ and $path(containpat(Z_m, r, s)) = 0$ (eq. 5.49). Then also $complement(Z', Z_e, oset(Z'), v, r, s) = 1$ (eq. B.18) and $path(containpat(Z', r, s)) = 0$ (eq. B.17), and then $mincomp(Z', Z_e, oset(Z'), r, s) = 1$.

B.4 RULE ORDERING IN Z_M

Statement 16 (red_implies_acc)

$$\begin{aligned}\forall r, s \in Z_m, Z_m \subseteq Z_{combined}, possibly\_minimal(Z_m) = 1,\\ \forall oset(Z_m) \subseteq allset(Z_m), valid(oset(Z_m)) = 1 :\\ order_{red}(Z_m, oset(Z_m), r, s) = 1 \implies order_{acc}(Z_m, oset(Z_m), r, s) = 1.\end{aligned} \tag{B.20}$$

If $order_{red}(Z_m, oset(Z_m), r, s) = 1$ then requiring s to occur before r causes some rule t to become redundant given any state $(Z_m, Z_m, oset'(Z_m))$, where $oset'(Z_m) \supseteq oset(Z_m) \cup (s, r)$ (eq. 5.55). If it were possible that for any such $oset'(Z_m)$ it could hold that $accurate(Z_m, oset'(Z_m)) = 1$, then rule t could be removed from the rule set and the new rule set would still be accurate. However, then the new rule set would have fewer rules than Z_m, which is impossible, given that Z_m is a $possibly\_minimal$ rule set (eq. 5.26 and eq. 5.25). This means that if both r and s are in Z_m and $order_{red}(Z_m, oset(Z_m), r, s) = 1$, then also $order_{acc}(Z_m, oset(Z_m), r, s) = 1$.

Statement 17 (order_implies_acc)

$$\begin{aligned}\forall r, s \in Z_m, Z_m \subseteq Z_{combined}, possibly\_minimal(Z_m) = 1,\\ \forall oset(Z_m) \subseteq allset(Z_m), valid(oset(Z_m)) = 1 :\\ order(Z_m, oset(Z_m), r, s) = 1 \implies order_{acc}(Z_m, oset(Z_m), r, s) = 1.\end{aligned} \tag{B.21}$$

This follows directly from statement 16 and eq. 5.56.

Statement 18 (direct_order_implies_complement)

$$\begin{aligned}\forall r, s \in Z_m, Z_m, \subseteq Z_{combined}, possibly\_minimal(Z_m) = 1,\\ \forall oset(Z_m) \subseteq allset(Z_m), valid(oset(Z_m)) = 1 :\\ direct\_order(Z_m, oset(Z_m), r, s)) = 1 \implies\\ \forall v \in Z_m, (v, r), (v, s) \in oset(Z_m) : complement(Z_m, Z_m, oset(Z_m), v, r, s) = 1.\end{aligned} \tag{B.22}$$

Since Z_m is a $possibly\_minimal$ rule set, $order(Z_m, oset(Z_m), r, s) = 1$ implies that $order_{acc}(Z_m, oset(Z_m), r, s) = 1$ (statement 17). The accuracy ordering between any two rules r and s can be caused in two ways: (1) A direct ordering requirement arises from one or more word patterns that each create the need for such an ordering independently. (2) An indirect ordering requirement is caused by a set of word patterns in TD predicted one after the other, each prediction an independent event. Such a set of word patterns may require rule r to occur earlier than another rule v, and again require rule v to to occur earlier than rule s, creating an indirect ordering requirement from rule r to rule s. If $direct\_order(Z_m,\ oset'(Z_m), r, s) = 1$, then by definition there exists no rule t such that $order(Z_m, oset'(Z_m), r, t) = 1$ and $order(Z_m, oset'(Z_m), t, s) = 1$, even though $order(Z_m, oset'(Z_m), r, s) = 1$. It therefore follows that the ordering between r and s is a direct ordering (as in (1) above) caused by at least one (single) word pattern w. Such a word pattern w will be predicted incorrectly if $rulenum(s) < rulenum(r)$ (since $order_{acc}(Z_m, oset(Z_m), r, s) = 1$) and predicted correctly for at least one ordering which requires that $rulenum(r) < rulenum(s)$ (since $valid(oset(Z_m)) = 1$). This is only possible if the word pattern w is predicted accurately by rule r and incorrectly by rule s. Then $match(w, r) = 1$ and $match(w, s) = 1$, and no $v \in Z_m$ exists earlier in the rule set than r such that $match(w, v) = 1$. Since such a w exists, it follows that $complement(Z_m, Z_m, oset(Z_m), v, r, s) = 1$ for all v such that $\{(v, r), (v, s)\} \in oset(Z_m)$.

Statement 19 (order_complement_order)

$$\begin{aligned} &\forall r, s \in Z_m, Z_m \subseteq Z_{single}, possibly\_minimal(Z_m) = 1, \\ &\forall oset(Z_m) \subseteq allset(Z_m) : \\ &order_{acc}(Z_m, Z_m, oset(Z_m), r, s) = 1 \implies \forall v : \{(v, r), (v, s)\} \in oset(Z_m) : \\ &path(complement\&direct\_order(Z_m, Z_m, oset(Z_m), v, r, s)) = 1. \end{aligned} \tag{B.23}$$

If $order_{acc}(Z_m, oset(Z_m), r, s) = 1$ then by definition, $order(Z_m, oset(Z_m), r, s) = 1$ (eq. 5.56) and $path(direct\_order(Z_m, oset(Z_m), r, s)) = 1$ (eq. 5.57). Now consider any rules t_i, t_{i+1} along the path from r to s. Since for each t_i, t_{i+1} pair it holds that $direct\_order(Z_m, oset(Z_m), t_i, t_{i+1}) = 1$, it follows from statement 18 that $complement(Z_m, oset(Z_m), v, t_i, t_{i+1}) = 1$ for a valid v. Since this holds for all t_i along this path, it follows that $path(complement\&direct\_order(Z_m, Z_m, oset(Z_m), v, r, s)) = 1$.

Statement 20 (subset_implies_red_min)

$$\begin{aligned} &\forall r, s, v \in Z_m, Z_m \subseteq Z_{single}, possibly\_minimal(Z_m) = 1, \\ &\forall oset(Z_m) \subseteq allset(Z_m), \\ &subset(Z_m, oset(Z_m), r, r, s) = 1 \implies \\ &order_{red}(Z_m, oset(Z_m), r, s) = 1. \end{aligned} \tag{B.24}$$

If $subset(Z_m, oset(Z_m), r, r, s) = 1$ it follows by definition that $possible\_words(Z_m, Z_m, oset(Z_m), r, r) \subset possible\_words(Z_m, Z_m, oset(Z_m), r, s)$ (eq. 5.51), and then it will hold for any word pattern $w \in possible\_words(Z_m, Z_m, oset(Z_m), r, r)$ that also $match(w, s) = 1$ (eq. 5.36). Since $possible\_words(Z_m, Z_m, oset(Z_m), r, r) = rulewords(Z_m, oset(Z_m), r)$ (statement 3), this set provides a list of all the words in TD that may possibly invoke rule r during pronunciation prediction (eq. 5.35). It therefore follows that, if s occurred before r, all words that could possibly match r would first be matched against s and r would never be invoked. Rule r will then be a redundant rule within a $possibly\_minimal$ rule set, which contradicts the definition of a $possibly\_minimal$ rule set. Since rule r becomes redundant if rule s occurs before rule r when enforcing all the restrictions required by $oset(Z_m)$, it follows that $order_{red}(Z_m, oset(Z_m), r, s) = 1$ (eq. 5.55).

Statement 21 (subset_implies_red)

$$\begin{aligned}\forall r, s \in Z_m, v \in Z_e, Z_e \subseteq Z_m \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z'),\\ \forall (Z_m, oset_m(Z_m) \in minrules(Z', Z_e, oset(Z')) :\\ subset(Z', Z_e, oset(Z'), v, r, s) = 1 \implies\\ order_{red}(Z_m, oset(Z_m), r, s) = 1.\end{aligned} \tag{B.25}$$

Choose any $v \in Z_e$ and $r, s \in Z_m$ such that $subset(Z', Z_e, oset(Z'), v, r, s) = 1$. Then $v = r$ or $\{(v, r), (v, s)\} \in oset(Z')$ and $possible\_words(Z', Z_e, oset(Z'), v, r) \subset possible\_words(Z', Z_e, oset(Z'), v, s)$ (eq. 5.51). Now consider any word pattern $w \in possible\_words(Z_m, Z_m, oset(Z_m), r, r)$. Since $order\_subset(oset(Z'), oset_m(Z_m)) = 1$ and $Z_e \subseteq Z_m$ (eq. 5.29) it follows from statement 14 that also $w \in possible\_words(Z', Z_e, oset(Z'), v, r)$. Since $possible\_words(Z', Z_e, oset(Z'), v, r) \subset possible\_words(Z', Z_e, oset(Z'), v, s)$ (as given above), it follows that also $w \in possible\_words(Z', Z_e, oset(Z'), v, s)$. This implies that $match(w, s) = 1$ and no $q \in Z_e$ exists such that $match(w, q) = 1$ and $q = v$ or $(q, v) \in oset(Z')$. Now since $v = r$ or $\{(v, r), (v, s)\} \in oset(Z')$ this means no such matching $q \in Z_e$ exists such that $q = r$ or $\{(q, r), (q, s)\} \in oset(Z')$. Then no $q \in Z_m$ exists such that $q = r$ or $\{(q, r), (q, s)\} \in oset_m(Z_m)$ (since $Z_e \subseteq Z_m$ and $order\_subset(oset(Z'), oset_m(Z_m)) = 1$), and then $w \in possible\_words(Z_m, Z_m, oset_m(Z_m), r, s)$. Since this holds for all $w \in possible\_words(Z_m, Z_m, oset(Z_m), r, r)$, it follows that $possible\_words(Z_m, Z_m, oset_m(Z_m), r, r) \subseteq possible\_words(Z_m, Z_m, oset_m(Z_m), r, s)$. But from statement 10 it is not possible that $possible\_words(Z_m, Z_m, oset_m(Z_m), r, r) \equiv possible\_words(Z_m, Z_m, oset_m(Z_m), r, s)$, so $possible\_words(Z_m, Z_m, oset_m(Z_m), r, r) \subset possible\_words(Z_m, Z_m, oset_m(Z_m), r, s)$, and then $subset(Z_m, Z_m, oset_m(Z_m), r, r, s) = 1$ (eq. 5.51). Then it follows from statement 20 that $order_{red}(Z_m, oset(Z_m), r, s) = 1$.

Statement 22 (containpat_implies_order_min)

$$\begin{aligned}\forall r, s \in Z_m, Z_m \subseteq Z_{combined}, possibly\_minimal(Z_m) = 1, valid(oset(Z_m)) = 1\\ path(containpat(Z_m, r, s)) = 1 \implies order_{red}(Z_m, oset(Z_m), r, s) = 1.\end{aligned} \tag{B.26}$$

If $path(containpat(Z_m, r, s)) = 1$ then $context(r) \supset context(s)$ (eq. 5.47) and then $matchwords(r) \subseteq matchwords(s)$ (statement 1). Let $oset'(Z_m)$ be any ordering that includes $(s, r) \cup oset(Z_m)$. Then for every word pattern w such that $match(w, r) = 1$ also $match(w, s) = 1$, and since $(s, r) \in oset'(Z_m)$ and both s and r in Z_m, $possible\_words(Z_m, Z_m, oset'(Z_m), s, r) = \phi$

(eq. 5.36). Since $rulewords(Z_m, oset'(Z_m), r) \subseteq possible\_words(Z_m, Z_m, oset'(Z_m), s, r)$ (statement 4) also $rulewords(Z_m, oset'(Z_m), r) = \phi$, which means that r has become a redundant rule (statement 7). Since this holds for any $oset'(Z_m) \supseteq oset(Z_m) \cup (s,r)$ it follows that $order_{red}(Z_m, oset(Z_m), r, s) = 1$ (eq. 5.55).

B.5 RULE ORDERING IN Z_M AS A SUBSET OF $Z_{COMBINED}$

Statement 23 (path_order_min)

$$\begin{aligned} &\forall r, s \in Z_m, Z_e \subseteq Z_m \subseteq Z' \subseteq Z_{combined}, \\ &\forall oset(Z') \subseteq allset(Z'), allowed\_state(Z', Z_e, oset(Z')) = 1, \\ &\forall (Z_m, oset_m(Z_m) \in minrules(Z', Z_e, oset(Z')) : \\ &containpat/supercomp/rule\_order(Z', Z_e, oset(Z'), r, s) = 1 \implies \\ &(r,s) \in oset_m(Z_m). \end{aligned} \tag{B.27}$$

Choose any $r, s \in Z_m$:

- If $rule\_order(Z', Z_e, oset(Z'), r, s) = 1$, then $(r,s) \in oset(Z')$ (eq. 5.7). Since $allowed\_state(Z', Z_e, oset(Z')) = 1$, $order\_subset(oset(Z'), oset_m(Z_m)) = 1$ (eq. 5.29) and then $(r,s) \in oset_m(Z_m)$.
- If $supercomp(Z', Z_e, oset(Z'), r, s) = 1$, then $subset(Z', Z_e, oset(Z'), r, s) = 1$ by definition (eq. 5.52) and then, from statement 21 it follows that $order_{red}(Z_m, oset_m(Z_m), r, s) = 1$, and again $(r,s) \in oset_m(Z_m)$.
- If $containpat(Z', r, s) = 1$ then $containpat(Z_m, r, s) = 1$ (eq. B.15) and then it follows from statement 22 that $order_{red}(Z_m, oset_m(Z_m), r, s) = 1$. Again $(r,s) \in oset_m(Z_m)$.

Statement 24 (order_not_order)

$$\begin{aligned} &\forall r, s \in Z', Z_e \subseteq Z' \subseteq Z_{combined}, \\ &\forall oset(Z') \subseteq allset(Z'), allowed\_state(Z', Z_e, oset(Z')) = 1 : \\ &order(Z', Z_e, oset(Z'), r, s) = 1 \implies order(Z', Z_e, oset(Z'), s, r) \neq 1. \end{aligned} \tag{B.28}$$

Consider any $r, s \in Z'$ and first consider the $order_{acc}$ relation specifically. If $order_{acc}(Z', Z_e, oset(Z'), r, s) = 1$, then no $oset'(Z') \supseteq oset(Z') \cup (s,r)$ exists such that $allowed\_state(Z', Z_e, oset'(Z')) = 1$ (eq. 5.54). But since $allowed\_state(Z', Z_e, oset(Z')) = 1$, at least one $Z_m, oset_m(Z_m)$ pair exists such that $(Z_m, oset_m(Z_m)) \in minrules(Z', Z_e, oset(Z'))$ (eq. 5.29), such that $minimal(Z_m,\ oset_m(Z_m)) = 1$ (eq. 5.28). Clearly $(s,r) \notin oset_m(Z_m)$, which means that either (1) $(r,s) \in oset_m(Z_m)$ or (2) that the relationship between r and s is indeterminate, that is, that either $rulenum(r) < rulenum(s)$ or $rulenum(s) < rulenum(r)$ is allowed without affecting the value of $accurate(Z_m, oset_m(Z_m))$, or that (3) either or both of r and s are not in Z_m. In all the above cases, (r, s) can be added to $oset_m(Z_m)$, and it will still hold that $minimal(Z_m, oset_m(Z_m)) = 1$. Then $order_{acc}(Z', Z_e, oset(Z'), s, r) \neq 1$. The same can be shown to hold with regard to $order_{red}(Z', Z_e, oset(Z'), r, s)$ using eq. 5.55 in the same way as above, and therefore this statement also holds with regard to the $order$ relation in general.

Statement 25 (order_complement_options)

$$\begin{aligned} &\forall r, s \in Z', Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z) : \\ &complement\&order(Z', Z_e, oset(Z'), r, s) = 1 \implies \\ &path(containpat(Z', Z_e, oset(Z'), r, s))) = 1 \\ &\text{or } mincomp(Z', Z_e, oset(Z'), r, s) = 1. \end{aligned} \tag{B.29}$$

If $complement\&order(Z', Z_e, oset(Z'), r, s) = 1$ then both $complement(Z', Z_e, oset(Z'), r, s) = 1$ and $order(Z', Z_e, oset(Z'), r, s) = 1$ (eq. 5.44). Since $complement(Z', Z_e, oset(Z'), r, s) = 1$ it follows from statement 5 that either $mincomp(Z', Z_e, oset(Z'), r, s)) = 1$ or that $path(containpat(Z', r, s)) = \pm 1$. However, if $path(containpat(Z', r, s)) = -1$, then $path(containpat(Z', s, r)) = 1$ (eq. 5.47 and eq. 5.44) and then $order_{red}(Z', Z_e, oset(Z'), s, r) = 1$ (statement 22). Since it is not possible that both $order(Z', Z_e, oset(Z'), s, r) = 1$ and $order(Z', Z_e, oset(Z'), r, s) = 1$ (statement 24), it is therefore impossible that $path(containpat(Z', r, s)) = -1$. So either $mincomp(Z', Z_e, oset(Z'), r, s)) = 1$ or $path(containpat(Z', r, s)) = 1$.

Statement 26 (oreq_implies_acc)

$$\begin{aligned} &\forall r, s \in Z', Z_e \subseteq Z' \subseteq Z_{combined}, \\ &\forall oset(Z') \subseteq allset(Z'), allowed\_state(Z', Z_e, oset(Z')) = 1 : \\ &\exists v \in Z_e : order\_req(Z', Z_e, oset(Z'), v, r, s) = 1, \\ &order_{red}(Z', Z_e, oset(Z'), r, s) = 0 \implies \\ &direct\_order(Z', Z_e, oset(Z'), r, s) \neq -1. \end{aligned} \tag{B.30}$$

If $order\_req(Z', Z_e, oset(Z'), v, r, s) = 1$, then by definition (eq. 5.58) it follows that $direct\_order(Z', Z_e, oset(Z'), r, s) = 1$, $shared\_words(Z', Z_e, oset(Z'), v, r, s) \neq \phi$, and for all $w_i \in shared\_words(Z', Z_e, oset(Z'), v, r, s)$ it holds that $outcome(w_i) = outcome(r)$. Furthermore, there exists at least one w' in $shared\_words(Z', Z_e, oset(Z'), v, r, s)$ such that $outcome(w') \notin outcome(s)$. The value of $direct\_order(Z', Z_e, oset(Z'), r, s)$ can be $0, 1$ or -1. Since $order_{red}(Z', Z_e, oset(Z'), r, s) = 0$ the value of $direct\_order(Z', Z_e, oset(Z'), r, s)$

depends on the value of $order_{acc}(Z', Z_e, oset(Z'), r, s)$, which also can be $0, 1$ or -1 (eq. 5.57). Now consider any $(Z_m, oset_m(Z_m)) \in minrules(Z', Z_e, oset(Z'))$. Since $allowed\_state(Z', Z_e, oset(Z')) = 1$, at least one such a $Z_m, oset_m(Z_m)$ pair exists. Since $shared\_words(Z_m, Z_m, oset_m(Z_m), v, r, s) \subseteq shared\_words(Z', Z_e, oset(Z'), v, r, s)$ (statement 12) it still holds that for all the $x_i \in shared\_words(Z_m, Z_m, oset_m(Z_m), v, r, s)$ $outcome(x_i) = outcome(r)$. If it were possible that $direct\_order/order_{acc}(Z', Z_e, oset(Z'), s, r) = 1$, then there would exist at least one word pattern y such that s would predict y accurately, and r would predict y incorrectly. But since for all the x_i above $outcome(x_i) = outcome(r)$, no such y can exist, and therefore $direct\_order/order_{acc}(Z', Z_e, oset(Z'), s, r) \neq 1$ which implies that $direct\_order(Z', Z_e, oset(Z'), r, s) \neq -1$.

B.6 CHARACTERISTICS OF AN ALLOWED STATE

Statement 27 (allowed_state_decided)

$$\begin{aligned} \forall Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z'), \\ allowed\_state(Z', Z_e, oset(Z')) = 1, \\ \forall (Z_m, oset_m(Z_m)) \in minrules(Z', Z_e, oset(Z')) : \\ order\_subset(decided\_set(Z', Z_e, oset(Z')), oset_m(Z_m)) = 1. \end{aligned} \tag{B.31}$$

Choose any $(Z_m, oset_m(Z_m)) \in minrules(Z', Z_e, oset(Z'))$. Then $minimal(Z_m, oset_m(Z_m)) = 1$, $Z_e \subseteq Z_m \subseteq Z'$ and $order\_subset(oset'(Z'), oset_m(Z_m)) = 1$, by definition (eq. 5.28). Now consider any $r, s \in Z_m \cap Z'$ such that $(r, s) \in decided\_set(Z', Z_e, oset'(Z'))$. Then $r, s \in Z_m$ and $path(order\_decided(Z', Z_e, oset'(Z'), r, s)) = 1$ (eq. 5.60). From the definition of $order\_decided$, this implies that there exists a path $v_1 = r, v_2, ..., v_n = s$, all $v_i \in Z'$, such that either (1) $containpat(Z', v_i, v_{i+1}) = 1$, or (2) $supercomp(Z', Z_e, oset(Z'), v_i, v_{i+1}) = 1$ or (3) $(v_i, v_{i+1}) \in oset(Z')$. Now let t_i be only those v_j such that $v_j \in Z_m$. Then there exists a path $t_1 = r, t_2, ..., t_n = s$ such that for each t_i, $path(containpat/supercomp/rule\_order(Z', Z_e, oset(Z'), t_i, t_{i+1})) = 1$, with all $t_i \in Z_m$. From statement 23 it then follows that for each t_i, t_{i+1} pair it holds that $order(Z', Z_e, oset(Z'), t_i, t_{i+1}) = 1$, and then $(t_i, t_{i+1}) \in oset_m(Z_m)$. Since these t_i form a path from r to s, it follows that also $(r, s) \in oset_m(Z_m)$ (eq. 5.7). Since it holds for any $r, s \in Z_m \cap Z'$ that if $(r, s) \in decided\_set(Z', Z_e, oset'(Z'))$ then also $(r, s) \in oset_m(Z_m)$, it follows (from eq. 5.27) that $order\_subset(decided\_set(Z', Z_e, oset(Z')), oset_m(Z_m)) = 1$.

Statement 28 (allowed_state_possible)

$$\begin{aligned} &\forall Z_e \subseteq Z' \subseteq Z_{combined}, \forall oset(Z') \subseteq allset(Z'), \\ &allowed\_state(Z', Z_e, oset(Z')) = 1 \implies \\ &\forall (Z_m, oset_m) \in minrules(Z', Z_e, oset(Z')) : \\ &order\_subset(oset_m(Z_m), possible\_set(Z', Z_e, oset(Z'))) = 1. \end{aligned} \tag{B.32}$$

Choose any $(Z_m, oset_m) \in minrules(Z', Z_e, oset(Z'))$, and any $r, s \in Z_m$ such that $(r, s) \in oset_m(Z_m)$. Since $(r, s) \in oset_m(Z_m)$, it follows that $order_{acc}(Z_m, Z_m, oset(Z_m), r, s) = 1$ (statement 17) and then $path(direct\_order\&complement(Z_m, Z_m, oset_m(Z_m), v, r, s)) = 1$ for all v such that $\{(v, r), (v, s)\} \in oset_m(Z_m)$ (statement 19). This means that a path of $n \geq 2$ rules exist with $t_1 = r, t_2, t_3, \ldots, t_n = s$, all the $t_i \in Z_m$. For each of these t_i, t_{i+1} pairs both $direct\_order(Z_m, Z_m, oset_m(Z_m), t_i, t_{i+1}) = 1$ and $complement(Z_m, Z_m, oset_m(Z_m), v, t_i, t_{i+1}) = 1$. Since $complement(Z_m, Z_m, oset_m(Z_m), v, t_i, t_{i+1}) = 1$, it follows that $complement(Z', Z_e, oset(Z'), v, t_i, t_{i+1}) = 1$ (eq. B.18), and then either (1) $path(containpat(Z', t_i, t_{i+1}) = 1$ or (2) $mincomp(Z', Z_e, oset(Z'), v, t_i, t_{i+1}) = 1$ (statement 25). In both cases $shared\_words(Z', Z_e, oset(Z'), v, t_i, t_{i+1}) \neq \phi$ (eq. 5.43).

- If (1) $path(containpat(Z', t_i, t_{i+1})) = 1$, then $path(order\_decided(Z', Z_e, oset(Z'), t_i, t_{i+1})) = 1$ (eq. 5.59), and then $(t_i, t_{i+1}) \in possible\_set(Z', Z_e, oset(Z'))$ (eq. 5.63).

- If (2) $mincomp(Z', Z_e, oset(Z'), v, t_i, t_{i+1}) = 1$ then (since $shared\_words(Z', Z_e, oset(Z'), v, t_i, t_{i+1}) \neq \phi$) it follows that the value of $order\_possible(Z', Z_e, oset(Z'), v, t_i, t_{i+1})$, depends on the values of (a) $order\_decided(Z', Z_e, oset(Z'), t_i, t_{i+1})$, and (b) $order\_possible1(Z', Z_e, oset(Z'), t_i, t_{i+1})$ (eq. 5.62).

  First consider the possible values for (a), and assume $order\_possible1(Z', Z_e, oset(Z'), t_i, t_{i+1}) \neq -1$, the least restrictive choice. If $order\_decided(Z', Z_e, oset(Z'), t_i, t_{i+1}) = 0$, then the value of $order\_possible(Z', Z_e, oset(Z'), t_i, t_{i+1}) = 1$ (eq. 5.62), and then $(t_i, t_{i+1}) \in possible\_set(Z', Z_e, oset(Z'))$. If $order\_decided(Z', Z_e, oset(Z'), t_{i+1}, t_i) = 1$, then also $(t_i, t_{i+1}) \in possible\_set(Z', Z_e, oset(Z'))$ (eq. 5.63). Since $(t_i, t_{i+1}) \in Z_m$ and $direct\_order(Z_m, Z_m, oset_m(Z_m), t_i, t_{i+1}) = 1$ it follows from statement 27 that $direct\_order(Z', Z_e, oset'(Z'), t_i, t_{i+1}) \neq -1$. For the two valid options it then holds that $(t_i, t_{i+1}) \in possible\_set(Z', Z_e, oset(Z'))$.

  Now consider the possible values for (b), and assume that $order\_decided(Z', Z_e, oset(Z'), t_i, t_{i+1}) = 0$, again the least restrictive choice. If $order\_possible1(Z', Z_e, oset(Z'), t_i, t_{i+1}) \neq -1$, then $order\_possible(Z', Z_e, oset(Z'), t_i, t_{i+1}) = 1$. If it were possible that $order\_possible1(Z', Z_e, oset(Z'), t_i, t_{i+1}) = -1$, then $order\_possible(Z', Z_e, oset(Z'), t_{i+1}, t_i) = 1$ (eq. 5.62) and then it would hold that $order\_req(Z', Z_e, oset(Z'), v, t_{i+1}, t_i) = 1$ for a valid v (eq. 5.61), and then $direct\_order(Z', Z_e, oset(Z'), t_{i+1}, t_i) \neq -1$ (statement 26), or stated differently, $direct\_order(Z', Z_e, oset(Z'), t_i, t_{i+1}) \neq 1$. But since $direct\_order(Z_m, Z_m, oset_m(Z_m), t_i, t_{i+1}) = 1$, this causes a contradiction. Then it must hold that $order\_possible1(Z', Z_e, \ oset(Z'), t_i, t_{i+1}) \neq -1$, and then $(t_i, t_{i+1}) \in possible\_set(Z', Z_e, oset(Z'))$.

Since it holds for all (t_i, t_{i+1}) along a path from r to s that $(t_i, t_{i+1}) \in possible\_set(Z', Z_e, oset(Z'))$, it follows that $(r, s) \in possible\_set(Z', Z_e, oset(Z'))$ (eq. 5.63). Since this holds for any $(r, s) \in oset_m(Z_m)$, it follows that $order\_subset(oset_m(Z_m), possible\_set(Z', Z_e, \ oset(Z')) = 1$ (eq. 5.27).

B.7 INITIAL ALLOWED STATE

Statement 29

$$\forall w \in TD, r, w' \in Z', w' \text{ a word pattern matching word } w, Z' \subseteq Z_{combined} :$$
$$match(w, r) = 1 \iff path(containpat(w', r)) = 1. \tag{B.33}$$

Let w be any word pattern in $Z_{combined}$ and w' its associated word pattern in TD. If $match(w, r) = 1$, then, $context(w) \supseteq context(r)$ by definition (eq. 5.11). Given the construction of $Z_{combined}$, the only rule s that can exist such that $context(w) = context(s)$ is that s which is the word pattern w', so $context(w) = context(w') \supset context(r)$. Since $context(w') \supset context(r)$ it follows from eq. 5.48 that $path(containpat(Z', w', r)) = 1$. Similarly, if $path(containpat(Z', w', r)) = 1$, then (again from eq. 5.48) $context(w') \supset context(r)$, and then $context(w) = context(w') \supset context(r)$, and then $match(w, r) = 1$ (eq. 5.11).

Statement 30 (only_word_patterns) *If, prior to any rule resolution, all rules in $Z_{combined}$ are ordered according to the set of relationships $decided\_set(Z_{combined}, \phi, \phi)$, then predicting any word $w \in TD$ will only invoke the word pattern $w' \in Z_{no-conflict}$.*

Since $oset(Z_{combined}) = \phi$ does not contain any orderings whatsoever, it follows directly from the definition of $rulewords$ (eq. 5.35 and eq. 5.12) and $matchwords$ (eq. 5.34) that $matchwords(r) = rulewords(Z_{combined}, \phi, r)$. Let $w \in TD$ be any word to be predicted, and $w' \in Z_{combined}$ be each associated word pattern. Given the way in which $Z_{combined}$ has been constructed, w' exists for each word w, $match(w, w') = 1$, and furthermore there may exist a set of rules $\{r_i\}$ such that also $match(w', r_i) = 1$. From statement 29 it follows that for each of these r_i there exists a $path(containpat(Z_{combined}, w', r_i)) = 1$. Since $path(containpat(Z_{combined}, w', r_i)) = 1$, then also $path(order\_decided(Z_{combined}, \phi, \phi, w', r_i)) = 1$ (eq.5.59) and even if such r_i exist, none will be invoked – only w'. Since word variants are not allowed, $w' \in Z_{no-conflict}$.

Statement 31 (complete) *The set of rules $Z_{no-conflict}$ describes the training data accurately and completely.*

Consider any training word pattern w. If any sub-patterns existed in Z that matched both this word pattern and a conflicting one, it would have been removed from $Z_{no-conflict}$. Therefore, if a rule in $Z_{no-conflict}$ is applicable, it will be accurate. There are no word variants in TD; therefore, for each grapheme in each word pattern there exists at least one sub-pattern (the word pattern itself) that describes the grapheme in a way that does not conflict with any other pattern, implying that an applicable rule will always be found.

Statement 32 (initial_superpath_implies_subset)

$$\begin{aligned}\forall r, s \in Z_m, Z_m \subseteq Z' \subseteq Z_{combined}, possibly\_minimal(Z_m) = 1 :\\ path(containpat/supercomp(Z', \phi, \phi, v_0, r, s)) = 1 \implies\\ subset(Z_m, Z_m, \phi, v_0, r, s) = 1.\end{aligned} \tag{B.34}$$

Consider any $w \in possible\_words(Z', \phi, \phi, v_0, r)$, $Z_m \subseteq Z'$. Since ϕ contains no orderings whatsoever, then any set $possible\_words(A, B, \phi, v_0, x)$ will consist of all the words in TD matched by x, irrespective of the constitution of A or B, except that A and B should meet the requirements specified by eq. 5.36 for $possible\_words(A, B, \phi, v_0, x)$ to be defined. Then $matchwords(x) = possible\_words(A, B, \phi, v_0, x)$ for all valid values of A, B and x; and then it also holds that for all $y \in Z_m$: $matchwords(y) \equiv possible\_words(Z', \phi, \phi, v_0, y) \equiv possible\_words(Z_m, Z_m, \phi, v_0, y) \equiv rulewords(Z_m, \phi, y)$. For any $r, s \in Z_m$, if $path(containpat/supercomp(Z', \phi, \phi, r, s)) = 1$ then there exists a set of rules $v_1 = r, v_2, .., v_n = s$ such that for each (v_i, v_{i+1}), either (1) $containpat(Z', v_i, v_{i+1}) = 1$ or (2) $supercomp(Z', \phi, \phi, v_0, v_i, v_{i+1}) = 1$. If (1) $containpat(Z', v_i, v_{i+1}) = 1$ then $context(v_i) \supset context(v_{i+1})$ (eq. 5.47) and then $matchwords(v_i) \subseteq matchwords(v_{i+1})$ (statement 1); and then $rulewords(Z_m, \phi, v_i) \subseteq rulewords(Z_m, \phi, v_{i+1})$. If (2) $supercomp(Z', \phi, \phi, , v_0, v_i, v_{i+1}) = 1$, then $possible\_words(Z', \phi, \phi, v_i) \subset possible\_words(Z', \phi, \phi, v_{i+1})$ by definition (eq. 5.51), and then also $rulewords(Z_m, \phi, v_i) \subset rulewords(Z_m, \phi, v_{i+1})$. Then it holds for all v_i that $rulewords(Z_m, \phi, v_1 = r) \subseteq rulewords(Z_m, \phi, v_2) \subseteq \ldots \subseteq rulewords(Z_m, \phi, v_n = s)$; and then $rulewords(Z_m, \phi, r) \subseteq rulewords(Z_m, \phi, s)$. But since both r and s in Z_m, and since $valid(\phi) = 1$, it is not possible that $rulewords(Z_m, \phi, r) = rulewords(Z_m, \phi, s)$ (statement 9). So $rulewords(Z_m, \phi, r) \subset rulewords(Z_m, \phi, s)$, and then $possible\_words(Z', \phi, \phi, r) \subset possible\_words(Z', \phi, \phi, s)$, and then $subset(Z', \phi, \phi, r, s) = 1$ (eq. 5.51).

Statement 33 (Initial allowed state) *If the rule set $Z_{combined}$ is ordered according to the rule set orderings generated by $decided\_set(Z_{combined}, \phi, \phi)$, then the rule set is accurate, complete and in an $allowed\_state$, i.e:*

$$\begin{gathered} Z' \equiv Z_{combined}, Z_e = \phi, oset(Z') = decided\_set(Z', \phi, \phi) \implies \\ accurate(Z', oset(Z')) = 1, allowed\_state(Z', Z_e, oset(Z')) = 1. \end{gathered} \tag{B.35}$$

Let w be any word in TD and w' its associated word pattern in $Z_{no-conflict}$. It follows from statement 30 that predicting word w according to $Z', oset(Z')$ will always invoke the word pattern $w' \in Z_{no-conflict}$, which always exists. Since there is always such a word pattern, the new rule set will be complete. Since only rules in $Z_{no-conflict}$ will be invoked, and the rule set $Z_{no-conflict}$ is accurate (from statement 31), the new rule set will also be accurate; and then $accurate(Z', oset(Z')) = 1$. From the definition of $minimal$ (eq. 5.25), if a rule set can be accurate and complete, at least one rule set and rule ordering set will always exist such that $minimal(Z_m, oset_m(Z_m)) = 1$. Since $Z_{combined}$ consists of all possible rules, all such Z_m will be a subset of $Z_{combined}$, and then $\phi = Z_e \subseteq Z_m \subseteq Z' \equiv Z_{combined}$. Now consider any $r, s \in Z_m$ such that also $(r, s) \in oset(Z')$, i.e. $(r, s) \in decided\_set(Z', \phi, \phi)$. Then $path(order\_decided(Z', \phi, \phi, r, s)) = 1$ (eq. 5.60), and then $path(containpat/supercomp/rule\_order(Z', \phi, \phi, v_0, r, s)) = 1$ (eq. 5.59). Since ϕ contains no orderings whatsoever, this is only possible if $path(containpat/supercomp(Z', \phi, \phi, r, s)) = 1$. Then $subset(Z', \phi, \phi, r, s) = 1$ (statement 32), $order_{red}(Z_m, Z_m, oset_m(Z_m), r, s) = 1$ (statement 21), and then $(r, s) \in oset_m(Z_m)$ (eq. 5.56 and eq. 5.7). Since this holds for all $(r, s) \in Z_m$ it follows that $order\_subset(decided\_set(Z', \phi, \phi), oset_m(Z_m)) = 1$ (eq. 5.27), and then $order\_subset(oset(Z'), oset_m(Z_m)) = 1$, and then $allowed\_state(Z', Z_e, oset(Z')) = 1$ (eq. 5.29).

REFERENCES

[1] E.Barnard, J.P.L Cloete, and H. Patel, "Language and technology literacy barriers to accessing government services," *Lecture notes in Computer Science*, vol. 2739, pp. 37–42, 2003.

[2] *Ethnologue, Languages of the World*, http://www.ethnologue.com, 1 April 2005.

[3] B. Wheatley, K. Kondo, W. Anderson, and Y. Muthusumy, "An evaluation of cross-language adaptation for rapid HMM development in a new language," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Adelaide, 1994, pp. 237–240.

[4] A. Constantinescu and G. Chollet, "On cross-language experiments and data-driven units for ALISP," in *Proceedings Automatic Speech Recognition and Understanding*, 1997, pp. 606–613.

[5] Rita Singh, Bhiksha Raj, and Richard M. Stern, "Automatic generation of phone sets and lexical transcriptions," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Istanbul, Turkey, 2000, pp. 1691–1694.

[6] Rita Singh, Bhiksha Raj, and Richard M. Stern, "Automatic clustering and generation of contextual questions for tied states in hidden markov models," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Phoenix, Arizona, March 1999, vol. 1, pp. 117–120.

[7] Catherine Soanes, *Compact Oxford English Dictionary of Current English*, Oxford University Press, 2003.

[8] "Wikipedia Public Encyclopaedia," 14 April 2005, `http://en.wikipedia.org/wiki/Bootstrapping`.

[9] L. Osterholtz, A. McNair, I. Rogina, H. Saito, T. Sloboda, J. Tebelskis, A. Waibel, and M. Woszczyna, "Testing generality in JANUS: A multi-lingual speech to speech translation system," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1992, vol. 1, pp. 209–212.

[10] J. Glass, G. Flammia, D. Goodine, M. Phillips, J. Polifroni, S. Sakai, S. Seneff, and V. Zue, "Multilingual spoken-language understanding in the MIT Voyager system," *Speech Communication*, vol. 17, pp. 1–18, 1995.

[11] Tanja Schultz and Alex Waibel, "Fast bootstrapping of LVCSR systems with multilingual phoneme sets," in *Proceedings Eurospeech*, Rhodes, Greece, 1997, pp. 371–374.

[12] T. Schultz and A. Waibel, "Language-independent and language-adaptive acoustic modeling for speech recognition," *Speech Communication*, vol. 35, pp. 31–51, Aug. 2001.

[13] C. Callison-Burch and M. Osborne, "Bootstrapping parallel corpora," in *North American Chapter of the Association for Computational Linguists (NAACL) Workshop, Building and using parallel texts: data driven machine translation and beyond*, Edmonton, Canada, 2003.

[14] Jan Daciuk, "Computer-assisted enlargement of morphological dictionaries," in *Finite State Methods in Natural Language Processing, Workshop at 13th European Summer School in Logic, Language and Information*, Helsinki, Finland, August 2001.

[15] Kemal Oflazer and Sergei Nirenberg, "Practical bootstrapping of morphological analyzers," in *Proceedings of Computational Natural Language Learning (CoNLL) Workshop at the Annual Meeting of the European Chapter of the Association for Computational Linguists (EACL)*, Bergen, Norway, June 1999.

[16] J. Zavrel and W. Daelemans, "Bootstrapping a tagged corpus through combination of existing heterogeneous taggers," in *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC-2000)*, Athens, Greece, 2000, pp. 17–20.

[17] "The CMU pronunciation dictionary," 1998, `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`.

[18] R. Mitten, "Computer-usable version of Oxford Advanced Learner's Dictionary of Current English," Tech. Rep., Oxford Text Archive, 1992.

[19] P. Mertens and F. Vercammen, "Fonilex manual," Tech. Rep., K.U.Leuven CCL, 1998.

[20] T.J. Sejnowski and C.R. Rosenberg, "Parallel networks that learn to pronounce English text," *Complex Systems*, pp. 145–168, 1987.

[21] K. Torkkola, "An efficient way to learn English grapheme-to-phoneme rules automatically," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Minneapolis, USA, April 1993, vol. 2, pp. 199–202.

[22] O. Andersen, R. Kuhn, A. Lazarides, P. Dalsgaard, J. Haas, and E. Noth, "Comparison of two tree-structured approaches for grapheme-to-phoneme conversion," in *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Philadelphia, USA, 1996, vol. 3, pp. 1700–1703.

[23] A. Black, K. Lenzo, and V. Pagel, "Issues in building general letter to sound rules," in *3rd ESCA Workshop on Speech Synthesis*, Jenolan Caves, Australia, November 1998, pp. 77–80.

[24] W. Daelemans, A. van den Bosch, and J. Zavrel, “Forgetting exceptions is harmful in language learning,” *Machine Learning*, vol. 34, no. 1-3, pp. 11–41, 1999.

[25] R.I. Damper, Y. March, M.J. Adamson, and K. Gustafson, “Evaluating the pronunciation component of text-to-speech systems for English: a performance comparison of different approaches,” *Computer Speech and Language*, vol. 13, pp. 155–176, April 1999.

[26] Timothy J. Hazen, I.Lee Hetherington, Han Shu, and Karen Livescu, “Pronunciation modelling using a finite-state transducer representation,” *Speech Communication*, vol. (article in press), 2005.

[27] J. Allen, M.S. Hunnicut, and D. Klatt, *From Text to Speech: The MITalk system*, Cambridge University Press, Cambridge, 1987.

[28] Cecil H. Coker, Kenneth W. Church, and Mark Y. Liberman, “Morphology and rhyming: two powerful alternatives to letter-to-sound rules for speech synthesis,” in *Proceedings of ESCA Workshop on Speech Synthesis*, Autrans, France, 1990.

[29] H.S. Elovitz, R. Johnson, A. McHugh, and J.E.Shore, “Letter-to-sound rules for automatic translation of English text to phonetics,” *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 24, pp. 446–459, December 1976.

[30] Yousif A. El-Imam, “Phonetization of Arabic: rules and algorithms,” *Computer Speech and Language*, vol. 18, pp. 339–373, October 2004.

[31] J.C. Roux, “Grapheme-to-phoneme conversion in Xhosa,” *South African Journal of African Languages*, vol. 9, pp. 74–78, 1989.

[32] P.C. Bagshaw, “Phonemic transcription by analogy in text-to-speech synthesis: novel word pronunciation and lexicon compression,” *Computer Speech and Language*, vol. 12, pp. 119–142, April 1990.

[33] Neil McCulloch, Mark Bedworth, and John Bridle, “NETspeak: A re-implementation of NETtalk,” *Computer Speech and Language*, vol. 2, pp. 289–302, 1987.

[34] J. Hakkinen, J. Suontausta, S. Riis, and K Jensen, “Accessing text-to-phoneme mapping strategies in speaker independent isolated word recognition,” *Speech Communication*, vol. 41, pp. 455–467, 2003.

[35] K.P.H. Sullivan and R.I. Damper, “Novel-word pronunciation: a cross-language study,” *Speech Communication*, vol. 13, pp. 441–452, 1993.

[36] F. Yvon, “Grapheme-to-phoneme conversion using multiple unbounded overlapping chunks,” in *Proceedings of Conference on New Methods in Natural Language Processing (NeMLaP)*, Ankara, Turkey, 1996, pp. 218–228.

[37] R.I. Damper and J.F.G. Eastmond, "Pronunciation by analogy: impact of implementational choices on performance," *Language and Speech*, vol. 40, pp. 1–23, 1997.

[38] Y. Marchand and R.I. Damper, "A multi-strategy apporach to improving pronunciation by analogy," *Computational Linguistics*, vol. 26, pp. 195–219, 2000.

[39] R. Luk and R. Damper, "Stochastic phonographic transduction for English," *Computer Speech and Language*, vol. 10, pp. 133–153, 1996.

[40] C.X. Ma and M.A. Randolph, "An approach to automatic phonetic baseform generation based on bayesian networks," in *Proceedings of Eurospeech*, Aalborg, Denmark, September 2001, pp. 1453–1456.

[41] V. Hoste, W. Daelemans, E.T.K. Sang, and S. Gillis, "Meta-learning for phonemic annotation of corpora," in *Proceedings of the International Conference on Machine Learning (ICML-2000)*, Stanford University, USA, 2000.

[42] T. Mark Ellison, *The machine learning of phonological structure*, Ph.D. thesis, University of Western Australia, 1992.

[43] Gary Tajchman, Eric Fosler, and Daniel Jurafsky, "Building multiple pronunciation models for novel words using exploratory computational phonology," in *Proceedings of Eurospeech*, Madrid, Spain, September 1995.

[44] Walter Daelemans, Steven Gillis, and Gert Durieux, "The acquisition of stress: a data-oriented approach," *Computational Linguistics*, vol. 208, pp. 421–451, 1994.

[45] Ove Andersen and Paul Dalsgaard, "Multi-lingual testing of a self-learning approach to phoneme transcription of orthography," in *Proceedings of Eurospeech*, Madrid, Spain, September 1995.

[46] M.J. Dedina and H.C. Nusbaum, "PRONOUNCE: A program for pronunciation by analogy," *Computer Speech and Language*, vol. 5, pp. 55–64, 1991.

[47] F. Yvon, "Self-learning techniques for grapheme-to-phoneme conversion," 1994, `http://citeseer.ist.psu.edu/yvon94selflearning.html`.

[48] D.W. Aha, D. Kibler, and M.K. Albert, "Instance-Based Learning algorithms," *Machine Learning*, vol. 6, pp. 437–66, 1991.

[49] W. Daelemeans, A. van den Bosch, and T. Weijters, "IGTree: using trees for compression and classification in lazy learning algorithms," *Artificial Intelligence Review*, vol. 11, pp. 407–423, 1997.

[50] T. Kohonen, “Dynamically expanding context, with application to the correction of symbol strings in the recognition of speech,” in *Proceedings of the 8th International Conference on Pattern Recognition (8th ICPR)*, Paris, France, Oct 1986, pp. 1148–1151.

[51] Timothy Baldwin and Hozumi Tanaka, “A comparative study of unsupervised grapheme-phoneme alignment methohds,” in *The 22nd Annual Meeting of the Cognitive Science Society (CogSci2000*, Philadelphia, 2000, pp. 597–602.

[52] V. Pagel, K. Lenzo, and A. Black, “Letter to sound rules for accented lexicon compressoin,” in *International Conference on Spoken Language Processing (ICSLP)*, Sidney, Australia, 1998, vol. 5, pp. 2015–2018.

[53] P. Dalsgaard, O. Andersen, and A.V. Hanser, “Theory and application of two approaches to grapheme-to-phoneme conversion,” Tech. Rep., ONOMASTICA project, May 1995.

[54] O. Andersen and P. Dalsgaard, “Multilingual testing of a self-learning approach to phonemic transcription of orthography,” in *Proceedings of Eurospeech*, Madrid, Spain, September 1995, vol. 2, pp. 1117–1120.

[55] A.J. Viterbi, “Error bounds for convolutional codes and a asymptotically optimum decoding algorithm,” *IEEE Transactions on Information Theory*, vol. 13, pp. 260–269, 1967.

[56] C. Schillo, G.A. Fink, and F. Kummert, “Grapheme based recognition for large vocabularies,” in *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Beijing, China, October 2000, pp. 129–132.

[57] Stephen Kanthak and Hermann Ney, “Context-dependent acoustic modelling using graphemes for large vocabulary speech recognition,” in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Orlando,Florida, 2002, pp. 845–848.

[58] Mirjam Killer, “Grapheme based speech recognition,” Tech. Rep., Interactive Systems Laboratory: Carnegie Mellon University / Swiss Federal Institute of Technology, Pittsburgh, USA, March 2003.

[59] M.A. Hearst, “Noun homograph disambiguation,” in *Proceedings of the 7th Annual Conference of the University of Waterloo Centre for the New OED and Text Research*, Oxford, 1991, pp. 1–19.

[60] D. Yarowsky, “Unsupervised word sense disambiguation rivaling supervised methods,” in *ACL-95*, Cambridge, 1995, pp. 88–95.

[61] B. Efron, “Bootstrap methods: another look at the jackknife,” *The Annals of Statistics*, vol. 7, pp. 1–26, 1979.

[62] T. Schultz and A. Waibel, “Polyphone decision tree specialization for language adaptation,” in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Istanbul,Turkey, June 2000, vol. 3, pp. 1707–1710.

[63] Alberto Lavelli, Bernardo Magnini, and Fabrizio Sebastiani, “Building thematic lexical resources by bootstrapping and machine learning,” in *Proceedings of the LREC Workshop on Linguistic Knowledge Acquisition and Representation: Bootstrapping Annotated Language Data*, Las Palmas, Spain, 2002.

[64] Pedro J. Moreno, Chris Joerg, Jean-Manuel van Thong, and Oren Glickman, “A recursive algorithm for the forced alignment of very long audio segments,” in *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, 1998.

[65] I. Aldezabal, K Gojemola, and K. Sarasola, “A bootstrapping approach to parser development,” in *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, Trento, 2000.

[66] Sebastian Stuker, “Automatic creation of pronunciation dictionaries,” Tech. Rep., Interactive Systems Laboratory: Carnegie Mellon University / Universitat Karlsruhe, Pittsburgh, USA, April 2002.

[67] Alan W. Black and Kevin A. Lenzo, “Building synthetic voices,” Tech. Rep., Language Technology Institute, Carnegie Mellon University, Pittsburgh, USA, January 2003.

[68] S. Maskey, L. Tomokiyo, and A.Black, “Bootstrapping phonetic lexicons for new languages,” in *Proceedings of Interspeech*, Jeju, Korea, October 2004, pp. 69–72.

[69] David Cohen, Zoubin Ghahramani, and Michael Jordan, “Active learning with statistical models,” *Journal of Artificial Intelligence Research*, vol. 4, pp. 129–145, 1990.

[70] Matthias Seeger, “Learning with labeled and unlabeled data,” Tech. Rep., Institute for Adaptive and Neural Computation, University of Edinburgh, Edinburgh, December 19 2002.

[71] Richard Sproat, Branimir Boquraev, Steven Bird, Don Hindle, Martin Kay, David McDonald, Hans Uszkoreit, and Yorick Wilks, *A Computational Theory of Writing Systems*, Cambridge University Press, Cambridge, 2000.

[72] M. Davel, “DictionaryMaker v1.0 manual,” Tech. Rep., CSIR, Pretoria, South Africa, 2004.

[73] A. Martin, G. Doddington, T. Kamm, M. Ordowski, and M. Przybocki, “The DET curve in assessment of detection task performance,” in *Proceedings of the European Conference on Speech Communication and Technology*, 1997, pp. 1895–1898.

[74] A. Black, P. Taylor, and R. Caley, “The festival speech synthesis system,” 1999, `http://festvox.org/festival/`.

[75] "Local language speech technology initiative (LLSTI)," 1 April 2005, `http://www.llsti.org`.

[76] M. Davel and E. Barnard, "LLSTI isiZulu TTS project report," Tech. Rep., CSIR, Pretoria, South Africa, November 2004.

[77] J.A. Louw, M. Davel, and E. Barnard, "A general purpose isiZulu TTS system," in *South African Journal of African Languages (submitted for publication)*, 2005.

[78] M. Davel and E. Barnard, "LLSTI isiZulu TTS evaluation report," Tech. Rep., CSIR, Pretoria, South Africa, September 2004.

[79] S. Young, D. Kershaw, J. Odell, D. Ollason, V. Valtchev, and P. Woodland, "The htk book. revised for htk version 3.0," July 2000, `http://htk.eng.cam.ac.uk/`.

[80] Tebogo M. Modiba, "Aspects of automatic speech recognition with respect to Northern Sotho," M.S. thesis, University of the North, South Africa, 2004.

[81] "The openphone project," 1 April 2005, `http://www.meraka.org.za/hlt/`.

[82] M. Davel and E. Barnard, "Bootstrapping for language resource generation," in *Proceedings of the Symposium of the Pattern Recognition Association of South Africa*, South Africa, 2003, pp. 97–100.

[83] M. Davel and E. Barnard, "The efficient creation of pronunication dictionaries: human factors in bootstrapping," in *Proceedings of Interspeech*, Jeju, Korea, October 2004, pp. 2797–2800.

[84] M. Davel and E. Barnard, "The efficient creation of pronunciation dictionaries: machine learning factors in bootstrapping," in *Proceedings of Interspeech*, Jeju, Korea, October 2004, pp. 2781–2784.

[85] M. Davel and E.Barnard, "A default-and-refinement approach to pronunciation prediction," in *Proceedings of the Symposium of the Pattern Recognition Association of South Africa*, South Africa, November 2004, pp. 119–123.

[86] M. Davel and E. Barnard, "Bootstrapping pronunciation dictionaries: practical issues," in *Proceedings of Interspeech (accepted for publication)*, Lisboa, Portugal, September 2005.

[87] J. S. Garofolo, Lori F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren, "The DARPA TIMIT acoustic-phonetic continuous speech corpus, NIST order number PB91-100354," February 1993.