

Automatic Acquisition of Two-Level Morphological Rules

DISSERTATION PRESENTED FOR THE DEGREE OF

Doctor of Philosophy

AT

The University of Stellenbosch

South Africa



By

Pieter Zacharias Theron

17 February 1999

Promoter: Professor Ian Cloete

Declaration

I the undersigned hereby declare that the work contained in this dissertation is my own original work and has not previously in its entirety or in part been submitted at any university for a degree.

Pieter Zacharias Theron

6 November 1999

Summary

There are numerous applications for computational systems with a natural language processing capability. All these applications, which include free-text information retrieval, machine-translation and computer-assisted language learning, require a detailed and correctly structured database (or lexicon) of language information on all the levels of language analysis (phonology, morphology, syntax, semantics, etc.). To hand-code this information can be time-consuming and error prone. An alternative approach is to attempt the automation of the lexicon construction process. The contribution of this thesis is to present a method to automatically construct rule sets for the morphological and phonological levels of language analysis. The particular computational morphological framework used is that of two-level morphology. The lexicon, which contains the language specific information of two-level analyzers/generators, consists of two components: (1) A morphotactic description of the words to be processed, as well as (2) a set of two-level phonological (or spelling) rules. The input to the acquisition process is source-target word pairs, where the target is an inflected form of the source word. It is assumed that the target word is formed from the source through the optional addition of a prefix and/or a suffix. There are two phases in the acquisition process: (1) segmentation of the target into morphemes and (2) determination of the optimal two-level rule set with minimal discerning contexts. In phase one, an acyclic deterministic finite state automaton (ADFSA) is constructed from string edit sequences of the input pairs. Segmentation of the words into morphemes is achieved through viewing the ADFSA as a directed acyclic graph (DAG) and applying heuristics using properties of the DAG as well as the elementary string edit operations. For phase two, the determination of the optimal rule set is made possible with a novel representation of rule contexts, with morpheme boundaries added, in a new DAG. We introduce the notion of a delimiter edge. Delimiter edges are used

to select the correct two-level rule type as well as to extract minimal discerning rule contexts from the DAG. To illustrate the language independence of an acquired rule set, results are presented for English adjectives, Xhosa noun locatives, Afrikaans noun plurals and Spanish adjectives. Furthermore, it is shown how rules are acquired from thousands of input source-target word pairs. Finally, the excellent generalization of an acquired rule set is shown by applying a slightly manually modified rule set to previously unseen words. The recognition accuracy on unseen words was 98.9% while the generation accuracy was 97.8%.

Opsomming

Daar is baie toepassings vir rekenaarstelsels met 'n natuurlike-taal verwerkingsvermoë. Al hierdie toepassings, wat vrye teks inligtingonttrekking, masjien vertaling en rekenaargesteurde taalonderrig insluit, benodig 'n gedetailleerde en korrek gestruktureerde databasis (of leksikon) van taalinligting oor al die vlakke van taalanalise (fonologie, morfologie, sintaks, semantiek, ens.). Om hierdie taalinligting per hand te kodeer kan tydrowend wees en foute kan maklik gemaak word. 'n Alternatiewe benadering is om die leksikon konstruksie proses te probeer outomatiseer. Die bydrae wat hierdie tesis maak is om 'n metode te beskryf vir die outomatiese aanleer van reëls vir die morfologiese en fonologiese vlakke van taalanalise. Die spesifieke rekenaarlinguistiese raamwerk wat gebruik is, is dié van twee-vlak morfologie. Die leksikon, waar die taalspesifieke inligting van twee-vlak analiseerders/genereerders gestoor word, bestaan uit twee komponente: (1) 'n Morfotaktiese beskrywing van die woorde wat verwerk sal word en (2) 'n stel van twee-vlak fonologiese (of spel) reëls. Die invoer van die aanleerproses is bron-teiken woordpare, waar die teikenwoord 'n infleksie van die bronwoord is. Dit word aanvaar dat die teikenwoord gevorm word deur die opsionele byvoeging van 'n voorvoegsel en/of 'n agtervoegsel by die bronwoord. Twee fases kan onderskei word in die aanleerproses: (1) Segmentasie van die teikenwoord in die morfeme waaruit dit bestaan en (2) die bepaling van die optimale stel twee-vlak reëls met die kortste moontlike onderskeidende kontekste. In fase een word 'n asikliese deterministiese eindige-toestand outomaat (ADETO) gekonstrueer van die string-redigeringsreeks (E. "string edit sequences") van die invoer woordpare. Die teikenwoorde word gesegmenteer in die morfeme waaruit dit bestaan deurdat die ADETO as 'n gerigte asikliese grafiek (GAG) beskou word en deur die toepassing van heuristiese reëls wat die eienskappe van die GAG benut sowel as die eienskappe van die

elementêre string-redigeringsreeks operasies. In fase twee word die vasstelling van die optimale stel twee-vlak reëls moontlik gemaak deur 'n unieke voorstelling van die reëlkontekste, met morfeemgrense bygevoeg, in 'n nuwe GAG. Ons skep die konsep “afbakeningsboog” (E. “delimiter edge”). Afbakeningsboë word gebruik om die korrekte twee-vlak reëlsoort te bepaal sowel as vir die onttrekking van die kortste onderskeidende kontekste vanuit die GAG. Om die taalonafhanklikheid van die leerproses te illustreer word resultate gegee vir Engelse byvoeglike naamwoorde, Xhosa selfstandige naamwoord lokatiewe, Afrikaanse selfstandige naamwoord meervoude en Spaanse byvoeglike naamwoorde. Verder word gewys hoe reëls geleer word vir duisende bron-teiken woordpare. Laastens word gewys hoe goed die aangeleerde reëls, met minimale veranderinge, veralgemeen om toegepas te word op woorde wat nie gesien is gedurende die leerproses nie. Die herkenningsakkuraatheid vir hierdie woorde was 98.9% en die genereringsakkuraatheid was 97.8%.

Acknowledgements

I would like to thank the following persons and institutions:

- Professor Ian Cloete my promotor.
- The staff and my fellow students at the Department of Computer Science, University of Stellenbosch.
- The Information Technology supporting staff of the University of Stellenbosch.
- The staff of ISSCO (Institut Dalle Molle pour les Etudes Sémantiques et Cognitives), University of Geneva, in particular Professor Margaret King and Professor Susan Armstrong, where I did research for a year.
- Dr. Lauri Karttunen and Dr. Kenneth Beesley of the Multi-Lingual Theory and Technology (MLTT) Group, Rank Xerox Research Center, Grenoble, for kindly providing their Xerox Finite State Tools.
- The examiners of the thesis: Professors Susan Armstrong, Ian Cloete, Justus Roux and Dr. Hendrik Boshoff.
- The following institutions provided bursaries: The Foundation for Research Development, the University of Stellenbosch, the Harry Crossley Fund and the Swiss Federal Government.
- My family and friends for their support.

Contents

1	Two-Level Morphological Rules	1
1.1	Introduction	1
1.2	Two-level Morphology	3
1.2.1	Morphotactic Description	6
1.2.2	Two-level Rule Formalism	9
1.3	Automatic Acquisition/Machine Learning	10
1.4	Other Work	13
1.5	Overview	14
2	Acquisition of Morphotactics	16
2.1	Introduction	16
2.2	Determining String Edit Sequences	17
2.2.1	Basic String Edit Algorithm	17
2.2.2	Normalized String Edit Distance	20
2.2.3	Morphology-Specific Heuristic	21
2.3	Merging Edit Sequences	25
2.4	Special Cases	28

CONTENTS

ii

3	Acquiring Two-Level Rules: Overview	31
3.1	Introduction	31
3.2	Acquiring a Rule from a Single Word Pair	32
3.3	Rule Set from Set of Lexical-Surface Representations	36
3.3.1	Minimal Discerning Rule Contexts	36
3.3.2	Left or Right Contexts	42
4	Acquiring Two-Level Rules:	
	Formal Analysis	45
4.1	Introduction	45
4.2	Two Questions in terms of full mixed-context sets	46
4.3	Two Questions in terms of shortened mixed-context sets	54
4.4	Two Questions in terms of shortened paths in a DAG	66
4.4.1	Preliminary Definitions	67
4.4.2	Back to the Two Questions	73
4.5	Left or Right Contexts	83
4.6	Insertion Rules	92
4.7	Summary	96
5	Results and Evaluation	98
5.1	Introduction	98
5.2	English Adjectives	98
5.3	Xhosa Noun Locatives	101
5.4	Spanish Adjectives	105
5.5	Afrikaans Noun Plurals	111
5.5.1	Unseen Words	118

<i>CONTENTS</i>	iii
6 Conclusion	122
6.1 Summary	122
6.2 Future Work	124
Bibliography	126
A Semantics of Two-Level Rules	131
Index	134

List of Figures

1.1	Components of the analyzer/generator PCKIMMO	5
2.1	Array used as workspace for dynamic programming algorithm.	19
2.2	Array for tracing backwards to establish the minimal string edit sequences.	22
2.3	ADFSA viewed as a DAG to extract prefixes.	26
2.4	Reversed-suffix ADFSA viewed as a DAG to extract suffixes.	27
3.1	ADFSA viewed as a DAG.	39
3.2	Right-context ADFSA viewed as a DAG.	43
4.1	$\mathcal{G} = G(M(C_i^{full}(\mathcal{W})))$	79
4.2	$\mathcal{G} = G(M(C_i^{right}))$	86
4.3	$\mathcal{G} = G(M(C_i^{left}))$	86
4.4	Mixed-context ADFSA subgraph for $0:i$	95

List of Tables

3.1	Truth table to select the correct rule type.	35
5.1	Number of rules acquired for each rule-set trained on four-fifths of the word pairs.	118
5.2	Modifications for perfect parsing to rule-sets trained on four-fifths of the word pairs.	119
5.3	Recognition errors on unseen one-fifth test word pairs. . . .	120
5.4	Generation errors on unseen one-fifth test word pairs. . . .	120
5.5	Recognition and generation accuracy on the unseen one-fifth test data (787 word pairs in each case).	121
A.1	Semantics of two-level rules	132
A.2	Truth table for the two-level rules	133

Chapter 1

Two-Level Morphological Rules

1.1 Introduction

There are numerous applications for computational systems with a natural language processing (NLP) capability. All these applications, which include free-text information retrieval, machine-translation and computer-assisted language learning, require a detailed and correctly structured database (or lexicon) of language information on all the levels of language analysis. Examples of the levels of language analysis are the phonological, morphological, syntactical, semantical and pragmatical levels. This thesis describes a method for automatically acquiring (or learning) morphological two-level rules¹, for use by morphological analyzers/generators.

The phonological level² is concerned with the sound (or spelling) changes

¹*Morphological* two-level rules is somewhat a misnomer, since two-level rules are phonological (or sound-changing) rules. However, traditionally two-level rules are called morphological two-level rules. This reference to *morphological* two-level rules probably refers to the fact that the phonological sound changes occur when morphological operations (such as plural formation) take place.

²When we are in particular concerned with how the sounds appear in normal written

which occur when words are formed. For example, when an *-s* is added to *fox*, the plural form *foxes* is formed. The sound change which occurred here is the addition of the *-e-* to the suffix *-s*.

The morphological level is concerned with the morphemes that constitute words and how they can be combined to form words. For example, *foxes* is formed through the combination of the root form *fox* and the suffix *-s*.

The syntactic level of analysis concerns the study of how words may be combined to form phrases. For example the phrase

[1]

The fox ate the hare.

consists of the main verb *ate* which is preceded by the subject *the fox* and followed by the object *the hare*. An example of the implicit semantic information in Example 1³ is that *the fox* is an *animal*.

An example of the pragmatic level of analysis is in the most probable reading of the following sentence:

[2]

The man took the sandwich to lunch.

Our pragmatic information, i.e. our information about the speaker's intentions, allow us to deduce that the man did not take the sandwich to lunch as a companion, but as food.

Up to now NLP systems were limited in their language coverage because the large volumes of language information required cannot be constructed text as letters, then we will talk about the *orthographic* level.

³We will refer to the examples by the number appearing in square brackets at the top right of each example.

manually. An alternative approach is to attempt the automation of the lexicon construction process. This research area is called Computational Lexicography and it employs techniques and knowledge from various disciplines such as Artificial Intelligence (Machine Learning, Knowledge Representation), Computational Linguistics, Databases and Lexicography.

In particular, the contribution of this study is to present a complete method to automatically construct rule sets for the phonological (or orthographic) level of language analysis. As a by-product some morphological analysis is done as well. A particular computational morphological framework is used, namely that of *two-level morphology*. This framework is selected since the two-level morphological model has been successfully applied to various languages (no other computational model is so language independent). Furthermore, the two-level model has benefited from one-and-a-half decades of research and is well enough defined to serve as a target formalism for the acquired morphological sound-changing (or spelling) rules.

1.2 Two-level Morphology

Computational systems based on the two-level model of morphology (Koskeniemi, 1983) have been remarkably successful for many languages (Sproat, 1992). Examples of these implementations are:

- Finnish (Koskeniemi, 1983)
- English (Karttunen and Wittenburg, 1983),
- Romanian (Kahn, 1983)
- Japanese (Alam, 1983)

- French (Lun, 1983)
- German (Morpholympics winner (Haapalainen et al., 1994))
- Turkish (Oflazer, 1994)
- Greek (Sgarbas et al., 1995)
- Basque (Alegria et al., 1996)
- Arabic (Beesley, 1996)
- Syriac (Kiraz, 1996)

Although not formally published, implementations exist for many of the remaining West-European languages (e.g. Dutch, Italian and Spanish), done by commercial companies (e.g. Xerox, Grenoble and Lingsoft, Finland) or research institutes (e.g. ISSCO⁴, Geneva).

The language specific information (i.e. the lexicon) of such a system is stored as

1. a morphotactic description of the words to be processed as well as
2. a set of two-level phonological (or spelling) rules.

The two-level formalism can handle either orthographic or phonological representations of words (see for example (Antworth, 1990))⁵. Exactly the same

⁴Institut Dalle Molle pour les Etudes Sémantiques et Cognitives

⁵Since we use to two-level model as our target formalism, the acquisition algorithm described in this thesis can learn as well from either the orthographic or phonological representations of words. However, we will provide examples only for the orthographic representation of words.

algorithms are used to process both these representations. The only constraint is that the words (or their phonological representation) must consist of letters (or phonetic symbols) of a finite alphabet.

The main components of the morphological analyzer/generator PCKIMMO (Antworth, 1990) are given in Figure 1.1. Both the two-level rule lexicon

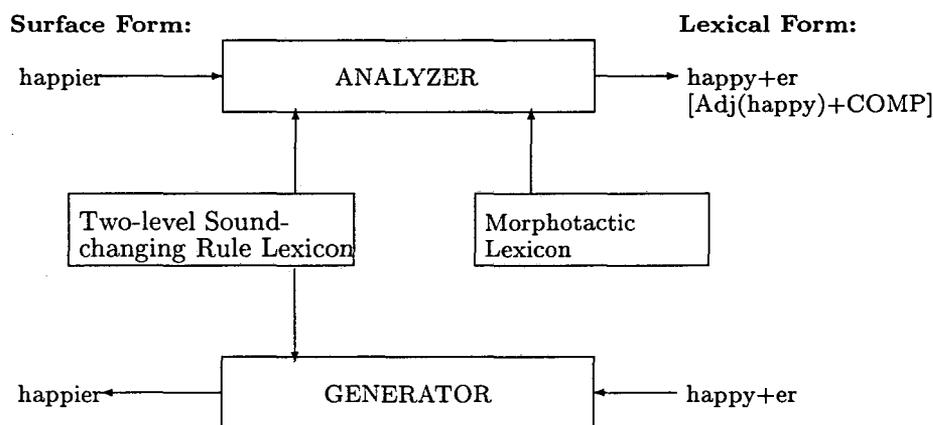


Figure 1.1: Components of the analyzer/generator PCKIMMO

and the morphotactic lexicon are used by the *analyzer* (see Figure 1.1). However, only the two-level rule lexicon is used by the *generator*.

Typical errors which occur for hand-coded lexicons for a morphological analyzer/generator are under- or overspecification of rules, which can cause overgeneration, overrecognition, failure to generate or failure to recognize.

An example of overgeneration is when, say, the forms **happir* and *happier* are generated from the lexical form *happy+er*.

An example of overrecognition is when the surface form *happier* is analyzed into both *happy+er [Adj(happy)+COMP]* and *red+er [Adj(red)+COMP]*.

Failure to generate or failure to recognize are simply when no form is generated or recognized, respectively.

More examples on these typical errors which might occur are given in Chapter 5.

Section 1.2.1 describes an example morphotactic part of the lexicon and Section 1.2.2 introduces the two-level rule formalism.

1.2.1 Morphotactic Description

The morphotactic part of the two-level lexicon lists the morphemes (e.g. prefixes and roots) and describes how they can be combined to form words. For example, given the following morphotactic formulas or segmentations (Antworth, 1990, p.107)

[3]

<u>Formed Word</u>	=	<u>Prefix</u> +	<u>Root</u>	+ <u>Suffix</u>
<i>bigger</i>	=		<i>big</i>	+ <i>er</i>
<i>biggest</i>	=		<i>big</i>	+ <i>est</i>
<i>unclear</i>	=	<i>un</i> +	<i>clear</i>	
<i>unclearly</i>	=	<i>un</i> +	<i>clear</i>	+ <i>ly</i>
<i>unhappy</i>	=	<i>un</i> +	<i>happy</i>	
<i>unhappier</i>	=	<i>un</i> +	<i>happy</i>	+ <i>er</i>
<i>unhappiest</i>	=	<i>un</i> +	<i>happy</i>	+ <i>est</i>
<i>unhappily</i>	=	<i>un</i> +	<i>happy</i>	+ <i>ly</i>
<i>unreal</i>	=	<i>un</i> +	<i>real</i>	
<i>cooler</i>	=		<i>cool</i>	+ <i>er</i>
<i>coolest</i>	=		<i>cool</i>	+ <i>est</i>
<i>coolly</i>	=		<i>cool</i>	+ <i>ly</i>

Two-Level Morphological Rules

7

<i>clearer</i>	=	<i>clear</i>	+ <i>er</i>
<i>clearest</i>	=	<i>clear</i>	+ <i>est</i>
<i>clearly</i>	=	<i>clear</i>	+ <i>ly</i>
<i>redder</i>	=	<i>red</i>	+ <i>er</i>
<i>reddest</i>	=	<i>red</i>	+ <i>est</i>
<i>really</i>	=	<i>real</i>	+ <i>ly</i>
<i>happier</i>	=	<i>happy</i>	+ <i>er</i>
<i>happiest</i>	=	<i>happy</i>	+ <i>est</i>
<i>happily</i>	=	<i>happy</i>	+ <i>ly</i>

the morphotactic lexicon of the morphological analyzer/generator PCKIMMO would be (Antworth, 1990, p.115):

```
ALTERNATION Begin      ADJ_PREFIX
ALTERNATION Adj_Prefix1 ADJ_ROOT1
ALTERNATION Adj_Prefix2 ADJ_ROOT1 ADJ_ROOT2
ALTERNATION Adj_Root1   ADJ_SUFFIX1 ADJ_SUFFIX2      (1)
ALTERNATION Adj_Root2   ADJ_SUFFIX2
ALTERNATION Adj_Suffix  End
```

LEXICON INITIAL

```
0          Begin      "["
```

LEXICON ADJ_PREFIX

```
un+       Adj_Prefix1 "NEG+"
0         Adj_Prefix2 ""
```

Two-Level Morphological Rules

8

LEXICON ADJ_ROOT1 (2)

clear	Adj_Root1	"Adj(clear)"
happy	Adj_Root1	"Adj(happy)"
real	Adj_Root1	"Adj(real)"

LEXICON ADJ_ROOT2

big	Adj_Root2	"Adj(big)"	(3)
cool	Adj_Root1	"Adj(cool)"	
red	Adj_Root2	"Adj(red)"	

LEXICON ADJ_SUFFIX1

+ly	Adj_Suffix	"+ADVBLZR"	(4)
-----	------------	------------	-----

LEXICON ADJ_SUFFIX2

+er	Adj_Suffix	"+COMP"	(5)
+est	Adj_Suffix	"+SUPERL"	(6)
0	Adj_Suffix	" "	

LEXICON End

0	#	"] "
---	---	-------

END

The *LEXICON* sections list the morphemes in their different categories. The first column in these sections indicates the morpheme strings (e.g. *clear*, *happy*). The third column in these sections is a tag indicating the

morphotactic category or function of the morpheme string in the first column. For example, the string *big* (see (3)) is an adjective root in LEXICON ADJ_ROOT2 and the string *+er* (see (5)) is an adjective suffix indicating the comparative case (*+COMP*). The *ALTERNATION* statements describe how the morphemes can be combined. For example, the fourth *ALTERNATION* statement (see (1)) says that adjective roots in LEXICON ADJ_ROOT1 (see (2)) can be followed by either the suffix *+ly* (see (4)) in the LEXICON ADJ_SUFFIX1, or the suffixes *+er* or *+est* in LEXICON ADJ_SUFFIX2 (see (5) and (6)).

In this thesis I show how a partial morphotactic description can be acquired automatically. The focus of this thesis is the acquisition of the phonological or orthographic rules in the next section.

1.2.2 Two-level Rule Formalism

Two-level rules are used in two-level computational morphological systems to model the sound changes which occur when morphemes combine to form words. These rules view a word as having a *lexical* and a *surface* representation, with a correspondence between them (Antworth, 1990), e.g.:

[4]

Lexical: *h a p p y + e r*

Surface: *h a p p i 0 e r*

Each pair of lexical and surface characters is called a *feasible pair*. A feasible pair can be written as *lexical-character:surface-character*. Such a pair is called a *default pair* when the lexical character and surface character are identical (e.g. *h:h*). When the lexical and surface character differ, it is

called a *special pair* (e.g. $y:i$). The null character (\emptyset) may appear as either a lexical character or a surface character (as in $+:0$), but not as both.

Two-level rules have the following syntax (Sproat, 1992, p.145):

[5]

$$CP \text{ op } LC - RC$$

CP (*correspondence part*), LC (*left context*) and RC (*right context*) are regular expressions over the alphabet of feasible pairs. In most, if not all, implementations based on the two-level model, the correspondence part consists of a single special pair. I also consider only single pair CPs in this thesis, since it is more commonly used. The operator **op** is one of four types:

1. Exclusion rule: $a:b \not\Leftarrow LC - RC$
2. Context restriction rule: $a:b \Rightarrow LC - RC$
3. Surface coercion rule: $a:b \Leftarrow LC - RC$
4. Composite rule: $a:b \Leftrightarrow LC - RC$

The exclusion rule ($\not\Leftarrow$) is used to prohibit the application of another, too general rule, in a particular subcontext. Since we will learn only from positive examples, we will consider only the \Rightarrow , \Leftarrow and \Leftrightarrow rule types. Appendix A summarizes the semantics of the four two-level rule types.

1.3 Automatic Acquisition/Machine Learning

Approaches to automatically acquire phonological rules can be grouped into three classes, based on the type of rules they learn:

1. Symbolic approach 1 which acquires ordered sequential *rewrite rules*,
2. Symbolic approach 2 which acquires unordered parallel *two-level rules* and
3. Connectionist approaches which encode rules as sets of *weights in an artificial neural network*.

Up to now most work has attempted to acquire rewrite rules, i.e. following approach (1). This thesis is the first publication to present an automatic method for the acquisition of two-level rules, i.e. following approach (2). The advantage of this second approach over the first is that the same acquired rule set can be used to *analyze* surface forms *into* their underlying morphemes as well as to *generate* surface forms *from* the underlying morphemes. This is in contrast to a set of rewrite rules which is one-directional, i.e. they can be used only to either analyze or generate. A further advantage of using the two-level model is that there already exist morphological analyzers/generators which interpret two-level rules (e.g. Antworth, 1990; Karttunen and Beesley, 1992).

The following example illustrates what is meant by *ordered* rules (approach (1)) and *unordered* rules (approach (2)): We need to write sound-changing rules to map the following two underlying forms to their surface forms:

[6]

Underlying form		Surface form
<i>Cay</i>	→	<i>Cby</i>
<i>Caye</i>	→	<i>Cbie</i>

We first write the ordered rewrite rules:

Two-Level Morphological Rules

12

[7]

$$1. a \rightarrow b / C _$$

$$2. y \rightarrow^* i / a _ e$$

$$3. y \rightarrow i / b _ e$$

Rule 1 above means that a string a is rewritten as a string b when a follows a C . When rule 1 executes, it creates the intermediate forms:

[8]

$$Cby$$

$$Cbye$$

This means that Rule 2 will not be able to execute if Rule 1 executes first, since the underlying a has already been changed into b . Thus we should either let Rule 2 execute first or rewrite it as Rule 3. If we choose Rule 3, then Rule 1 should always execute before Rule 3, since that would make the b available to match the b in Rule 3's context. Thus the *order* of execution of the rewrite rules is important.

The two-level rules for mapping the two underlying forms in Example 6 to their surface forms (and the inverse) are:

[9]

$$a:b \Leftrightarrow C:C _$$

$$y:i \Leftrightarrow a:b _ e:e$$

The first rule means that an a in the underlying form is changed to a b in the surface form, following a C in both the underlying form as well as

the surface form. The second rule means that *y* is changed into *i* following an *a* in the underlying form which is changed to a *b* in the surface form and preceding an *e* in both the underlying form as well as the surface form. These two rules can be viewed as constraints which are applied in parallel.

The advantage of the symbolic approaches over the connectionist approach is that their acquired rules can be understood by humans (and can thus be modified if required), while the meaning of the rules in connectionist systems are hidden in the “weights” on their connections.

1.4 Other Work

Other work on the automatic analysis and acquisition of morphology has been concentrated either on the learning of rewrite rules or the setting of weights in artificial neural networks.

Simons describes methods for studying morphophonemic alternations (using *annotated* interlinear text) (Simons, 1988) and Grimes presents a program for discovering affix positions and cooccurrence restrictions (Grimes, 1983). Koskenniemi provides a sketch of a discovery procedure for phonological two-level rules (Koskenniemi, 1990). Golding and Thompson (Golding and Thompson, 1985) and Wothke (Wothke, 1986) present systems to automatically calculate a set of word-formation rules. These rules are, however, ordered one-level rewrite rules and not unordered two-level rules, as in our system. Kuusik also acquires ordered rewrite rules, for stem sound changes in Estonian (Kuusik, 1996). Daelemans *et al.* use a general symbolic machine learning program to acquire a decision tree for matching Dutch nouns to their correct diminutive suffixes (Daelemans et al., 1996). The input to their process is the syllable structure of the nouns and a given set of five

suffix allomorphs⁶. They do not learn rules for possible sound changes. Our process automatically acquires the necessary two-level sound changing rules for prefix and suffix allomorphs, as well as the rules for stem sound changes. Connectionist work on the acquisition of morphology has been more concerned with implementing psychologically motivated models, than with acquisition of rules for a practical system ((Sproat, 1992, p.216) and (Gasser, 1994, 1996)).

1.5 Overview

The contribution of this study is to present a complete method for the automatic acquisition of an optimal set of two-level rules for source-target word pairs. It is assumed that the target word is formed from the source through the addition of a prefix and/or a suffix⁷. Furthermore, we show how a partial acquisition of the morphotactic lexicon results as a by-product of the rule-acquisition process. For example, in phase one the morphotactic description of the target word in the input pair

[10]

Source	Target
<i>happy</i>	<i>happier</i>

is computed as

⁶An allomorph is a different form of the same affix. For example, in the morphotactic equation $foxes = fox + es$, $-es$ is an allomorph of $-s$ in the morphotactic equation $foxes = fox + s$.

⁷Non-linear operations (such as infixation) are not considered here, since the basic two-level model deals with it in a round-about way. We can note that extensions to the basic two-level model have been proposed to handle non-linear morphology (Kiraz, 1996).

[11]

$$\textit{happier} = \textit{happy} + \textit{er}$$

In phase two, a lexical-surface representation is obtained by mapping the right-hand side of this morphotactic equation onto the left-hand side:

[12]

Lexical: *h a p p y + e r*

Surface: *h a p p i 0 e r*

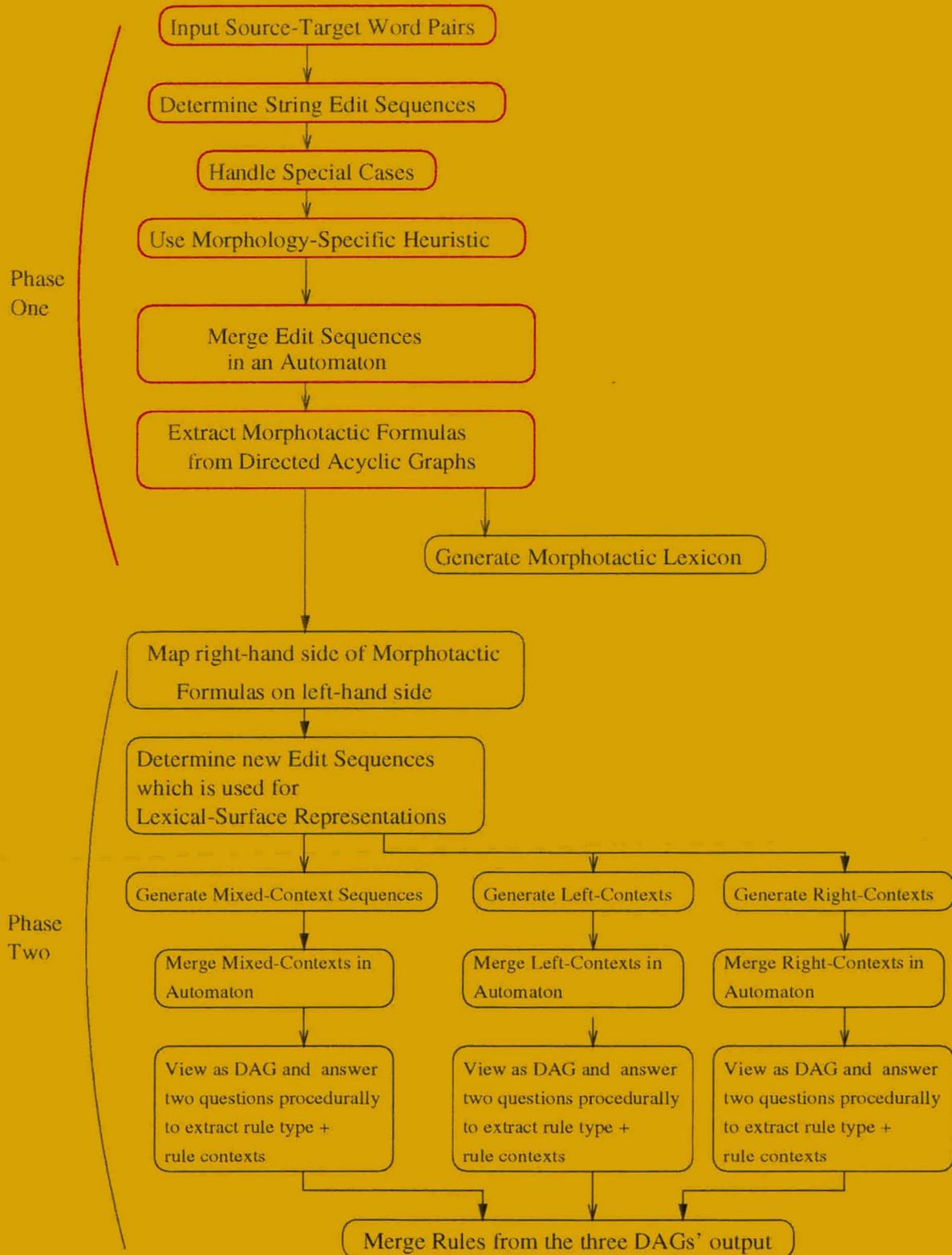
From this lexical-surface representation, the two-level rule

[13]

$$y:i \Leftrightarrow p:p -$$

can be derived in phase two. These processes are described in detail in the rest of the thesis: Chapter 2 describes the acquisition of morphotactics through segmentation and Chapter 3 provides an overview of the method for computing the optimal two-level rules. In Chapter 4 the method for computing the two-level rules is formally analyzed and described in detail. Chapter 5 evaluates the experimental results and Chapter 6 concludes the thesis.

Rule Acquisition Process Overview



Chapter 2

Acquisition of Morphotactics

2.1 Introduction

To acquire the morphotactics of a target word relative to a source word, one needs to know how the source word can be transformed into the target word. In general there are an infinite number of ways to change a source word into a target word. We are, however, interested in those changes which make the most sense linguistically. Our assumption is that the changes which make the most sense linguistically, are also the changes which require the least resources in time and space. For the acquisition of the morphotactic description for a set of source-target input words, we make use of two well known computational formalisms which provide constructive ways to determine the *minimal* resources required (relative to the given formalism). The first formalism or technique is *minimal string edit sequences* and the second is *acyclic deterministic finite state automata*.

The morphotactics of the input words are acquired by (1) computing the string edit difference between each source-target pair and (2) merging the edit sequences as an acyclic deterministic finite state automaton. The

automaton, viewed as a directed acyclic graph, is used to segment the target word into its constituent morphemes.

2.2 Determining String Edit Sequences

A string edit sequence is a sequence of *elementary operations* which change a source string into a target string (Sankoff and Kruskal, 1983, Chapter 1). The elementary operations used in this thesis are single character *deletion* (DELETE), *insertion* (INSERT) and *replacement* (REPLACE). We indicate the copying of a character by NOCHANGE. A cost is associated with each elementary operation. Typically, INSERT and DELETE have the same (positive) cost and NOCHANGE has a cost of zero. REPLACE could have the same or a higher cost than INSERT or DELETE. Edit sequences can be ranked by the sum of the costs of the elementary operations that appear in them. The interesting edit sequences are those with the lowest total cost. For most word pairs, there are more than one edit sequence (or mapping) possible which have the same minimal total cost.

2.2.1 Basic String Edit Algorithm

A basic dynamic programming algorithm to compute the minimum total cost string edit sequences for two finite strings, is given in this subsection. This algorithm is based on the one provided in (Sankoff and Kruskal, 1983, p.266).

Definitions Let Σ be a finite alphabet and Σ^* be the set of all finite strings over Σ . Let $\mathbf{a} = a_1 \dots a_m$ and $\mathbf{b} = b_1 \dots b_n$ be strings of Σ^* . Let λ be the null string with length zero. An elementary edit operation is a pair $x:y \neq \lambda:\lambda$,

where both x and y are strings of lengths 0 or 1. There are four types of elementary operations: insertions ($\lambda:y$), deletions ($x:\lambda$), replacements ($x:y$) and copying ($x:x$). Costs (or weights) are associated with each type of elementary operation and is indicated by $w(x:y)$. We define a string edit sequence S as a sequence of elementary operations: $x_1:y_1 \dots x_k:y_k$. Let $\mathbf{x} = x_1 \dots x_k$ and $\mathbf{y} = y_1 \dots y_k$. S is a string edit sequence between \mathbf{a} and \mathbf{b} if λ 's can be inserted in \mathbf{a} to make \mathbf{x} and in \mathbf{b} to make \mathbf{y} . The total cost of a string edit sequence S is $w(S) = \sum_{i=1}^k w(x_i:y_i)$. $d(\mathbf{a},\mathbf{b})$ is the minimum total cost of any string edit sequence between \mathbf{a} and \mathbf{b} . Let S^{min} be the set of string edit sequences between \mathbf{a} and \mathbf{b} which have the minimum total cost $d(\mathbf{a},\mathbf{b})$.

Problem Find S^{min} , the set of string edit sequences between \mathbf{a} and \mathbf{b} which have the minimum total cost $d(\mathbf{a},\mathbf{b})$. The minimum total cost $d(\mathbf{a},\mathbf{b})$ is often called the *edit distance* between the strings \mathbf{a} and \mathbf{b} .

Auxiliary Concepts Let $\mathbf{a}^i = a_1 \dots a_i$, $\mathbf{b}^j = b_1 \dots b_j$ and $d_{ij} = d(\mathbf{a}^i, \mathbf{b}^j)$.

Initialization $d_{00} = 0$, and $d_{ij} = \infty$ if either i or j is negative.

Recurrence

$$d_{ij} = \min \begin{cases} d_{i-1,j} + w(a_i:\lambda), & \text{(deletion of } a_i) \\ d_{i-1,j-1} + w(a_i:b_j), & \text{(replacement of } a_i \text{ by } b_j \text{ or} \\ & \text{copying if } a_i = b_j) \\ d_{i,j-1} + w(\lambda:b_j), & \text{(insertion of } b_j) \end{cases}$$

$$pointer_{i,j} = \left\{ \begin{array}{ll} (i-1, j) & \text{or} \\ (i-1, j-1) & \text{or} \\ (i, j-1) & \end{array} \right\} \begin{array}{l} \text{if corresponding} \\ \text{term is} \\ \text{minimum.} \end{array}$$

Solution $d(\mathbf{a}, \mathbf{b}) = d_{mn}$, and the set of minimum total cost edit sequences S^{min} can be found by using the array *pointer* to “backtrack” from (m, n) .

Workspace The array which serves as the workspace for the dynamic programming algorithm is given Figure 2.1. See Section 2.2.3 for an example

	0	1	2	.	.	.	n
0							
1							
2							
.							
.							
.							
m							

Figure 2.1: Array used as workspace for dynamic programming algorithm.

using the workspace.

2.2.2 Normalized String Edit Distance

Marzal and Vidal provided an extension to the basic string edit distance algorithm which *normalizes* the edit distances of the edit sequences (Marzal and Vidal, 1993). This normalization appropriately rates the cost of the edit sequences with respect to the sizes of the edit sequences which are compared. For example, two REPLACE operations in a comparison between sequences of length three are more important than three REPLACES in a comparison of sequences of length nine. This is done by dividing the total computed cost of a string edit sequence by its length (i.e. its number of elementary edit operations). The basic algorithm in Section 2.2.1 is thus modified as follows: Let $L(S)$ be the length (i.e. number of elementary edit operations) of the edit sequence S . Then the normalized total cost of S is:

[14]

$$\hat{w} = w(S)/L(S)$$

The minimal total cost (or edit distance) d_{ij} is then modified to be:

[15]

$$d_{ij} = \min \begin{cases} d_{i-1,j} + \hat{w}(a_i:\lambda), \\ d_{i-1,j-1} + \hat{w}(a_i:b_j), \\ d_{i,j-1} + \hat{w}(\lambda:b_j), \end{cases}$$

2.2.3 Morphology-Specific Heuristic

To select a single edit sequence which will most likely result in a correct segmentation, we added a morphology-specific heuristic to a normalized distance string edit algorithm (Marzal and Vidal, 1993). This heuristic always selects an edit sequence containing two subsequences which identify prefix-root and root-suffix boundaries. The heuristic depends on the elementary operations being limited only to INSERT, DELETE and NOCHANGE, i.e. no REPLACES are allowed. We assume that the target word contains more morphemes than the source word. It therefore follows that there are more INSERTS than DELETES in an edit sequence. Furthermore, the letters forming the morphemes of the target word appear only as the right-hand components of INSERT operations.

As noted above, more than one string edit sequence with the minimum total cost are possible for most input word pairs. For example, let $cost(INSERT) = 1$, $cost(DELETE) = 1$, $cost(REPLACE) = 1$ and $cost(NOCHANGE) = 0$. Then there are three possible minimal edit sequences to change *happy* into *unhappier*. Figure 2.2 is the already computed array for changing *happy* into *unhappier*. The three edit sequences are obtained by tracing the arrows back from the array element (5,9) to the array element (0,0). The arrows are the pictorial representation of the minimal edit sequences. A horizontal arrow in column j means that the letter heading column j is inserted. For example, the horizontal arrow in cell (5,9) means that the letter *r* is inserted. A diagonal arrow in cell (i,j) means that the letter heading row i is replaced (or copied if the two letters are the same) by the letter heading column j . For example, the diagonal arrow in cell (5,7) means that the letter *y* is replaced by the letter *i*. A vertical arrow in row i

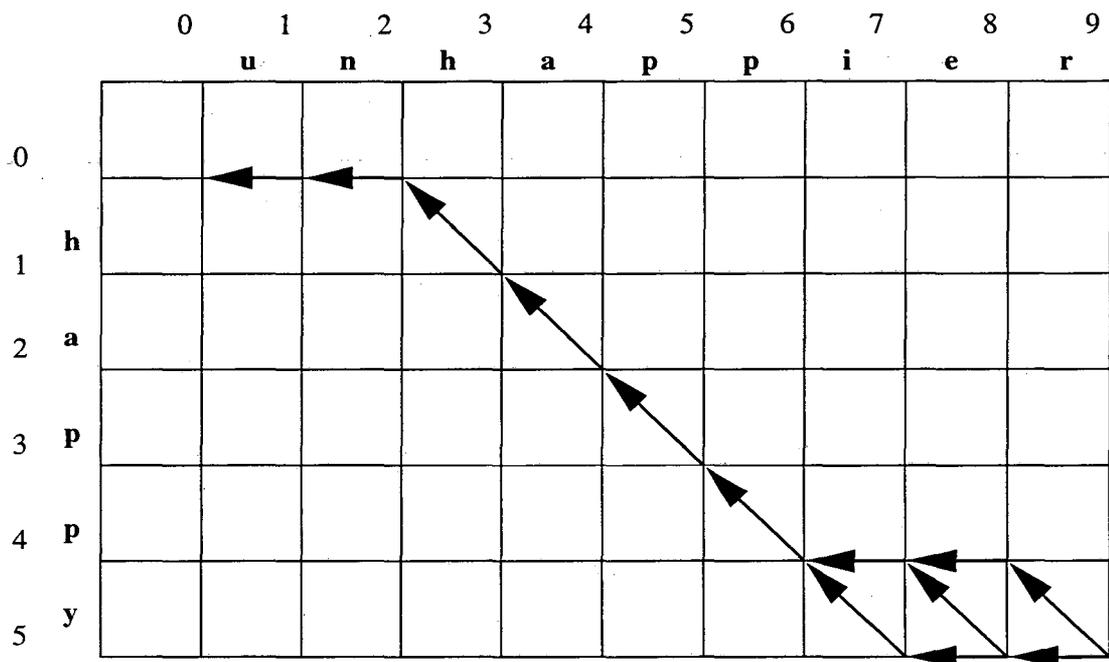


Figure 2.2: Array for tracing backwards to establish the minimal string edit sequences.

would mean that the letter heading row i is deleted. There are no vertical arrows in this example. The three edit sequences which can be extracted by following the arrows are:

[16]

$$0:u \ 0:n \ h:h \ a:a \ p:p \ p:p \ y:i \ 0:e \ 0:r$$

$$0:u \ 0:n \ h:h \ a:a \ p:p \ p:p \ 0:i \ y:e \ 0:r$$

$$0:u \ 0:n \ h:h \ a:a \ p:p \ p:p \ 0:i \ 0:e \ y:r$$

Remember that we want to use the edit sequences to segment the target word *unhappier* into its morphemes. Of the three edit sequences in Example 16, the first edit sequence makes the most sense linguistically. This is so since the letters forming the prefix *un-* of *unhappier* are present in the target side of the adjacent INSERT operations $0:u$ and $0:n$. Furthermore, for the suffix *-er* the letters are also present in the target side of the adjacent INSERT operations $0:e$ and $0:r$. In addition, y is replaced by i . This sound change also makes sense linguistically. However, we want a clear separation between the letters of the root form *happy* and the affix letters. For our purposes, we view the letter i to be part of the allomorphic suffix *-ier*. Thus we want to separate the REPLACE operation $y:i$ into a DELETE operation ($y:0$) and an INSERT operation ($0:i$). To this end we prohibit REPLACE operations, i.e. $\text{cost}(\text{REPLACE}) = \infty$. When this is done, then four minimal edit sequences are computed, this time with no REPLACE operations allowed:

[17]

$$0:u \ 0:n \ h:h \ a:a \ p:p \ p:p \ y:0 \ 0:i \ 0:e \ 0:r$$

$$0:u \ 0:n \ h:h \ a:a \ p:p \ p:p \ 0:i \ y:0 \ 0:e \ 0:r$$

0:u 0:n h:h a:a p:p p:p 0:i 0:e y:0 0:r

0:u 0:n h:h a:a p:p p:p 0:i 0:e 0:r y:0

Here, the first edit sequence again makes the most sense linguistically since the letters of the affixes are adjacent in the correct order and the *i* is separated from the *y*. Consider this first edit sequence (here written vertically) to change the string *happy* into the string *unhappier*:

[18]

0:u INSERT

0:n INSERT

h:h NOCHANGE

a:a NOCHANGE

p:p NOCHANGE

p:p NOCHANGE

y:0 DELETE

0:i INSERT

0:e INSERT

0:r INSERT

Note that the prefix *un-* as well as the suffix *-er* consist only of INSERTS. Furthermore, the prefix–root morpheme boundary is associated with an INSERT followed by a NOCHANGE and the root–suffix boundary by a NOCHANGE–DELETE–INSERT sequence. In general, the prefix–root boundary is just the reverse of the root–suffix boundary, i.e. INSERT–DELETE–NOCHANGE, with the DELETE operation being optional. The heuristic resulting from this observation is a bias giving highest precedence to INSERT operations, followed by DELETE and NOCHANGE, in the first half of the edit sequence. In the second half, the precedence is reversed.

2.3 Merging Edit Sequences

A single source-target edit sequence may contain spurious INSERTS which are not considered to form part of a morpheme. For example, the $0:i$ insertion in Example 18 should not contribute to the suffix *-er* to form *-ier*, since *-ier* is an allomorph of *-er*. We want to limit the number of allomorphic forms, since it is possible that each source-target word pair may introduce another allomorph and this will unnecessarily overpopulate the morphotactic part of the lexicon. The handling of the allomorphic forms should rather be done with the two-level sound-changing rules. To combat these spurious INSERTS, two acyclic deterministic finite state automata (ADFSA) are constructed: One from which the optimal prefixes will be extracted and one from which the optimal suffixes will be extracted. The first ADFSA accepts all and only the edit sequences for the set of source-target words. Since the automaton is deterministic, no state (or node, when viewed as a directed acyclic graph) can have more than one outgoing transition (or edge) labeled with the same elementary edit operation. Thus all the common prefixes of the edit sequences are merged in the automaton (see the first two edges labeled with $0:u$ and $0:n$ in Figure 2.3). The second ADFSA accepts all and only the *reversed* edit sequences for the set of source-target words. This automaton merges the reversed suffixes (see the first two edges labeled with $0:r$ and $0:e$ in Figure 2.4). These ADFSAs are then viewed as directed acyclic graphs (DAGs), with the elementary edit operations as edge labels. For example, the string-edit sequences computed with the morphology-specific heuristic for the three adjectively related word pairs *happy* \rightarrow *unhappier*, *big* \rightarrow *bigger* and *real* \rightarrow *unreal* are:

0:u 0:n h:h a:a p:p p:p y:0 0:i 0:e 0:r

b:b i:i g:g 0:g 0:e 0:r

0:u 0:n r:r e:e a:a l:l

To determine the optimal prefixes, these edit sequences are read into the ADFSA viewed as a DAG in Figure 2.3. For each edge a count is kept of the

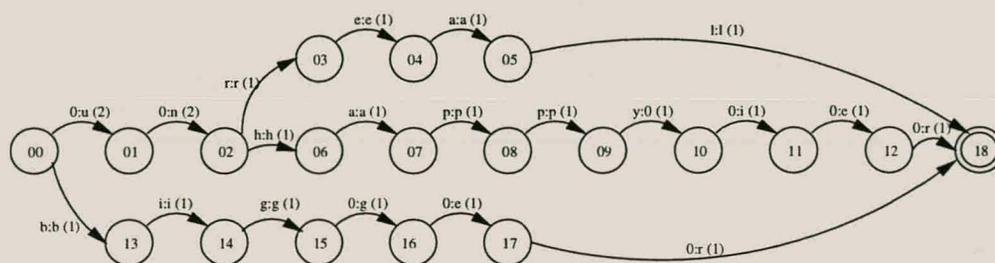


Figure 2.3: ADFSA viewed as a DAG to extract prefixes.

number of different edit sequences which pass through it (see the numbers between parenthesis in Figure 2.3). A path segment in the DAG consisting of one or more INSERT operations having a similar count, is then considered to be associated with a morpheme in the target word. Similar counts are measured relative to the INSERT operation which has the highest count in the contiguous sequence of INSERTS. A count is considered to be similar to this highest count, if that count varies with *less* than 50% from the highest count. The INSERTS in the sequence *0:u 0:n* have similar edge counts (two) and they are thus used to form the prefix *un-* in *un+real* and *un+happier*.

To determine the optimal suffixes, we first reverse the edit sequences in Example 19:

0:r 0:e 0:i y:0 p:p p:p a:a h:h 0:n 0:u
0:r 0:e 0:g g:g i:i b:b
l:l a:a e:e r:r 0:n 0:u

These reversed edit sequences are then read into the ADFSA, viewed as a DAG in Figure 2.4. The *0:r 0:e* INSERT sequence associated with the *-er*

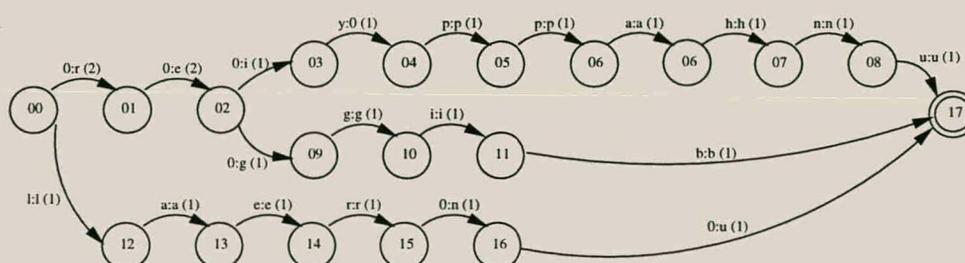


Figure 2.4: Reversed-suffix ADFSA viewed as a DAG to extract suffixes.

suffix appears two times more than the *0:r 0:e 0:i* INSERT sequence associated with the *-ier* suffix, even in this small set of three adjectively-related source-target pairs. The edges labeled with *0:r* and *0:e* have equal counts (two), which is also the highest count of the INSERT sequence. This means that there is a fall of 50% in the edge counts from the edge labeled with *0:e* to the edge labeled with *0:i* (count is one). Thus the *0:i* is considered to be a spurious INSERT which should not form part of the morpheme associated with the *0:r 0:e* INSERT sequence. To extract the morphemes of each target word, every path through the DAG is followed and only the target-sides of the elementary operations serving as edge labels, are written out. The null characters (*0*) on the target-side of DELETES are ignored (e.g. *y:0*) while the target-sides of INSERTS are only written if their frequency counts indi-

cate that they are *not* spurious allomorph INSERT operations (e.g. $0:i$ and $0:g$). The (reversed) strings obtained in this way are then un-reversed. The optimal prefixes are extracted in a similar way, from the DAG in Figure 2.3, constructed from the plain (i.e. normal order) edit sequences.

The following morphotactic descriptions result:

[21]

<u>Target Word</u>	=	<u>Prefix</u>	+	<u>Source</u>	+	<u>Suffix</u>
<i>unhappier</i>	=	<i>un</i>	+	<i>happy</i>	+	<i>er</i>
<i>bigger</i>	=			<i>big</i>	+	<i>er</i>
<i>unreal</i>	=	<i>un</i>	+	<i>real</i>		

Phase one can segment only one layer of affix additions at a time. However, once the morpheme boundary markers (+) have been inserted, the two-level rule acquisition process of phase two should be able to acquire the correct two-level rules for an arbitrary number of affix additions:

prefix1+prefix2+ ... +root+suffix1+suffix2+ ...

2.4 Special Cases

Sometimes a string edit sequence results which should be treated as a special case before it is merged in an acyclic deterministic finite state automaton. For example, the following string edit sequence results for the source-target word pair *happy* → *happily*:

[22]

h:h a:a p:p p:p 0:i 0:l y:y

Note the NOCHANGE operation $y:y$ at the end of this edit sequence. The source-side y is part of the root and should be moved towards the root. This is done as follows: The NOCHANGE operation $y:y$ is split into a DELETE and an INSERT operation, $y:0$ and $0:y$. The DELETE is then moved as the elementary operation following the second $p:p$:

[23]

$$h:h \ a:a \ p:p \ p:p \ y:0 \ 0:i \ 0:l \ 0:y$$

In this way the elementary NOCHANGE and DELETE operations of the root forms a contiguous block ($h:h \ a:a \ p:p \ p:p \ y:0$), followed by the INSERT operations of the suffix allomorph $0:i \ 0:l \ 0:y$. In general the above procedure is followed whenever a single NOCHANGE elementary operation is preceded (in the suffix) by at least two INSERT operations. Similarly, a NOCHANGE in the prefix must be followed by at least two INSERT operations, for this special case handling to be applied.

Another special case is when DELETE operations follow the isolated NOCHANGE identified in the first special case described above. For example, the following string edit sequence is computed for the Spanish adjective–superlative word pair *complementario* → *complementarísima*:

[24]

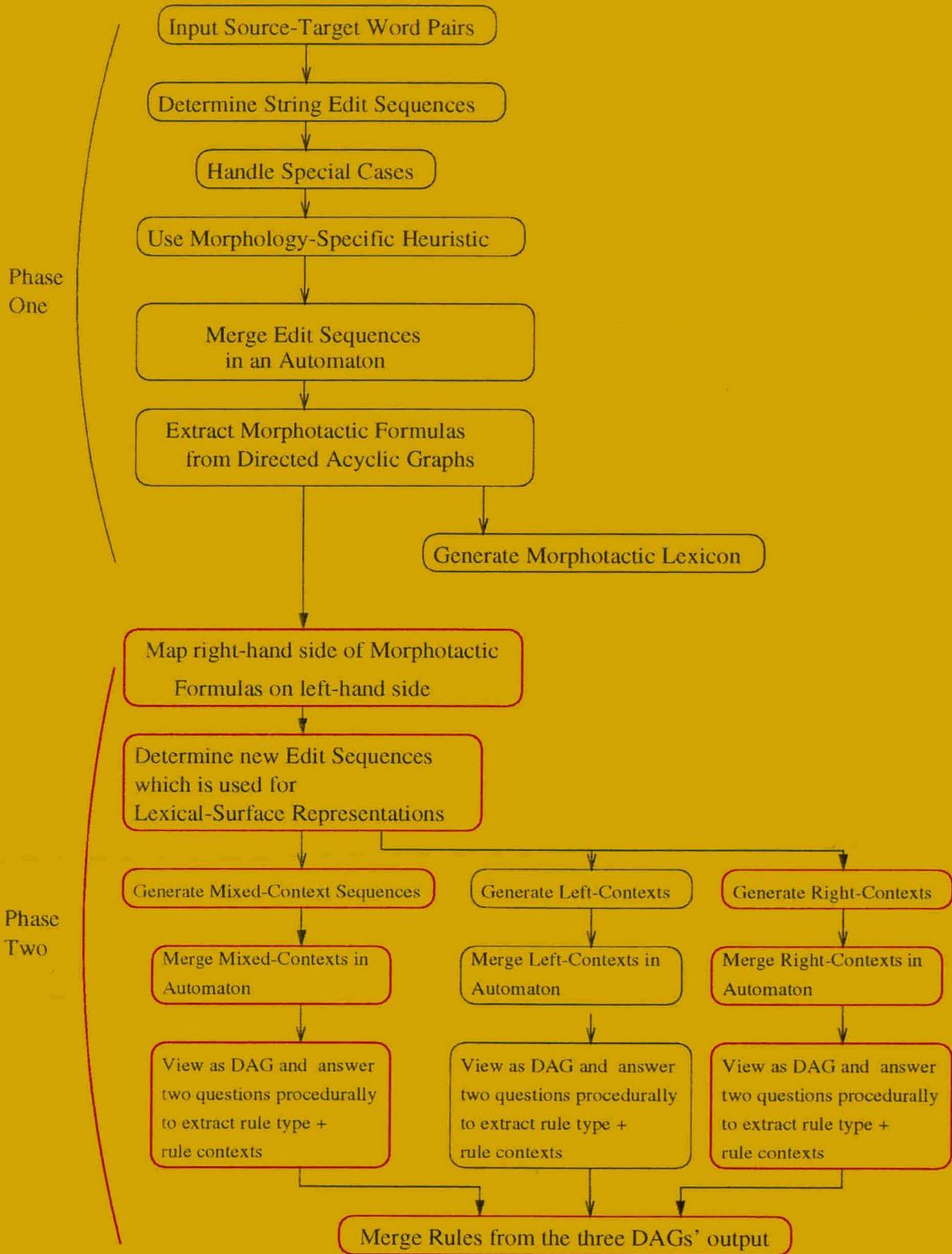
$$c:c \ o:o \ m:m \ p:p \ l:l \ e:e \ m:m \ e:e \ n:n \ t:t \ a:a \ r:r \ 0:i \ 0:s \ i:i \ o:0 \ 0:m \ 0:a$$

Here the NOCHANGE $i:i$ is handled as in the above special case. In addition the DELETE operation $o:0$ following this NOCHANGE is also moved towards the root. Thus the final edit sequence is:

c:c o:o m:m p:p l:l e:e m:m e:e n:n t:t a:a r:r i:0 o:0 0:í 0:s 0:i 0:m 0:a

Now all the root elementary operations (i.e. NOCHANGES and DELETES) are in a contiguous block as well as the suffix's INSERT operations. This edit sequence, with this special case handled, is now ready to be read into the reversed-suffix ADFSA, along with the other Spanish adjective word-pairs.

Rule Acquisition Process Overview



Chapter 3

Acquiring Two-Level Rules: Overview

3.1 Introduction

The method for acquiring the morphological two-level rules from the morphotactics computed in phase one is explained in this chapter. Phase one, described in the previous chapter, showed how the target words of the input source-target word pairs are segmented into their underlying morphemes.

The morphotactic formulas formed from these target words and their morphemes are used to calculate the lexical-surface representations of the target words. From these lexical-surface representations, *mixed-context sequences* are generated. The mixed-context sequences are then merged in an automaton viewed as a directed acyclic graph from which the two-level rules are extracted.

We start with the steps that need to be followed to acquire a rule from the morphotactic formula of a single word pair (Section 3.2). Next an overview is given of the acquisition of a set of rules from the morphotactics of a set of input word pairs (Section 3.3). Finally an overview is given of the acquisition of left and right contexts (Section 3.3:2). In the next chapter we will explain

formally and in detail how a set of rules is obtained from the morphotactics of a set of input word pairs.

3.2 Acquiring a Rule from a Single Word Pair

To acquire the optimal rules, we first determine the full length lexical-surface representation of each word pair. This representation is required for writing two-level rules (Section 1.2.2). The morphotactic descriptions from the previous chapter provide source-target input pairs from which *new* string edit sequences are computed: The right-hand side of the morphotactic description is used as the source and the left-hand side as the target string. For instance,

[26]

$$\begin{aligned} \text{Target Word} &= \text{Prefix} + \text{Source} + \text{Suffix} \\ \text{unhappier} &= \text{un} + \text{happy} + \text{er} \end{aligned}$$

is written as the new source-target word pair:

[27]

$$\begin{aligned} \text{Source: } & \text{un+happy+er} \\ \text{Target: } & \text{unhappier} \end{aligned}$$

A new edit sequence is computed from this source-target word pair:

[28]

$$u:u \ n:n \ +:0 \ h:h \ a:a \ p:p \ p:p \ y:i \ +:0 \ e:e \ r:r$$

This edit sequence maps the source into the target and provides the lexical and surface representation required by the two-level rules:

Lexical: *u n + h a p p y + e r*

Surface: *u n 0 h a p p i 0 e r*

The REPLACE elementary string edit operations (e.g. *y:i*) are now allowed since the morpheme boundary markers (+) are already present in the source string. REPLACES allow shorter edit sequences to be computed, since one REPLACE does the same work as an adjacent INSERT–DELETE pair. REPLACE, INSERT and DELETE have the same associated cost and NOCHANGE has a cost of zero. The morpheme boundary marker (+) is always mapped to the null character (0), which makes for linguistically more understandable mappings. Under these conditions, the selection of any minimal cost string edit mapping provides an acceptable lexical-surface representation¹.

To formulate a two-level rule for the source-target pair *happy-unhappier*, we need a correspondence part (CP) and a rule type (op), as well as a left context (LC) and a right context (RC) (see Section 1.2.2, page 9). Rules need only be coded for *special pairs*, i.e. INSERTS, DELETES or REPLACES. The only special pair in the above example is *y:i*, which will be the CP of the rule. Now the question arises as to *how large* the context of this rule must be? It should be large enough to uniquely specify the positions in the lexical-surface input stream where the rule is applied. On the other hand, the context should not be too large, resulting in an overspecified context which prohibits the application of the rule to unseen, but similar, words. Thus to make a rule as general as possible, its context (LC and RC) should

¹Our assumption is that such a minimal cost mapping will lead to an optimal rule set. In most (if not all) of the examples seen, a minimal mapping was also intuitively acceptable.

be as short as possible². By inspecting the edit sequence in Example 28, we see that y changes into i when y is preceded by a $p:p$, which serves as our first attempt at a (left) context for $y:i$. Now we can write the partially complete rule:

[30]

$$y:i \text{ op } p:p -$$

We still need to determine the rule type operator **op**. Determination of the correct rule type operator to use is done with the aid of two questions provided by (Antworth, 1990, p.53):

Question 1 Is E the only environment in which $L:S$ is allowed?

Question 2 Must L always be realized as S in E ?

The term *environment* (E) denotes the combined left (LC) and right (RC) contexts of a special pair. L is the lexical letter of the correspondence part $L:S$, and S is the surface letter. E , L and S are determined by inspecting the lexical-surface representation in Example 29. E in our example is $p:p-$, L is y and S is i . Thus the answer to question 1 is *true*, since $y:i$ only occurs after $p:p$ in our example. The answer to question 2 is also *true*, since y is always realized as i after a $p:p$ in the above edit sequence. Which rule type to use is gleaned from Table 3.1. Thus, to continue our example, we should use the composite rule type (\Leftrightarrow):

²If abstractions (e.g. *sets* such as V denoting vowels) over the regular pairs are introduced, it will not be so simple to determine what is “a more general context.” However, current implementations require abstractions to be explicitly instantiated during the compilation process ((Karttunen and Beesley, 1992, pp.19–21) and (Antworth, 1990, pp.49–50)). Thus, with the current state of the art, abstractions serve only to make the rules more readable.

Q1	Q2	op
false	false	none
true	false	\Rightarrow
false	true	\Leftarrow
true	true	\Leftrightarrow

Table 3.1: Truth table to select the correct rule type.

[31]

$$y:i \Leftrightarrow p:p -$$

This example shows how to go about coding the set of two-level rules for a *single* source-target pair. However, this soon becomes a tedious and error prone task when the number of source-target pairs increases, due to the complex interplay of rules and their contexts. As an alternative approach, I give in the next section an overview of how a rule set can be automatically acquired from larger numbers of source-target word pairs. To this end the two rule-type decision questions above will be rephrased from their declarative form to a procedural form. The reason for this is that in order to implement a computer program to find environments and special pairs for which the two questions are *true*, we need a detailed enough description of how to answer the two questions procedurally.

3.3 Rule Set from Set of Lexical-Surface Representations

This section gives an overview of how a rule set is acquired from a set of final edit sequences of input word pairs. It was shown in the previous section how these edit sequences are computed from the morphotactic formulas computed in phase one (see Chapter 2). These final string edit sequences serve as the lexical-surface representations of the word pairs from which we determine the *optimal two-level rule set* with minimal discerning contexts. An optimal two-level rule set is that rule set which has the least rules and rule contexts, while still parsing the input word pairs correctly.

An acyclic deterministic finite state automaton (ADFSA) is constructed from *mixed-context* representations of the edit sequences. The generation of mixed-context representations is described in Section 3.3.1 below. Determination of the *optimal rule set* is achieved by searching the ADFSA as a directed acyclic graph (DAG). We introduce the notion of an *edge-delimiter set*. Edge-delimiter sets are used to select the correct two-level *rule type* as well as to extract the associated minimal discerning contexts from the DAG.

3.3.1 Minimal Discerning Rule Contexts

It is important to acquire the minimal discerning context for each rule. This ensures that the rules are as general as possible (to work on unseen words as well) while limiting overgeneration and overrecognition. Recall that one should only code rules for the special pairs. Thus it is necessary to determine a rule type with associated minimal discerning context for each occurrence of a special pair in the final edit sequences. This is done by comparing all

the possible contiguous³ contexts of a special pair against all the possible contexts of all the other feasible pairs. To enable the computational comparison of the growing left and right contexts around a feasible pair, we developed a “mixed-context” representation discussed below.

We call the particular feasible pair for which a mixed-context is to be constructed, a *marker pair* (MP), to distinguish it from the feasible pairs in its context. The mixed-context representation is created by writing the first feasible pair to the left of the marker pair, then the first right-context pair, then the second left-context pair and so forth:

[32]

$$LC1, RC1, LC2, RC2, LC3, RC3, \dots, MP$$

The marker pair at the end serves as a label. Special symbols indicate the start (SOS) and end (EOS) of an edit sequence. If, say, the right-context of a MP is shorter than the left-context, an out-of-bounds symbol (OOB) is used to maintain the mixed-context format. For example the mixed-context of the special pair $y:i$ in the edit sequence in

[33]

$$h:h \ a:a \ p:p \ p:p \ y:i \ +:0 \ e:e \ r:r$$

is represented as:

[34]

$$p:p, +:0, p:p, e:e, a:a, r:r, h:h, EOS, SOS, y:i$$

³A two-level rule requires a contiguous context.

Acquiring Two-Level Rules: Overview

38

We can also write the mixed-context representation of the default pair $y:y$ in the edit sequence

[35]

$$u:u \ n:n \ +:0 \ h:h \ a:a \ p:p \ p:p \ y:y$$

computed from the following source-target pair:

[36]

Source: *un+happy*

Target: *unhappy*

The mixed-context representation for this default pair $y:y$ in Example 35 is

[37]

$$p:p, \text{EOS}, p:p, \text{OOB}, a:a, \text{OOB}, h:h, \text{OOB}, +:0, \text{OOB}, n:n, \text{OOB}, u:u, \\ \text{OOB}, \text{SOS}, \text{OOB}, y:y$$

The common prefixes of the mixed-contexts are merged in the acyclic deterministic finite state automaton which accepts all and only these mixed-context sequences. For example, the first $p:p$ of the mixed contexts in Example 34 and Example 37 is merged in the ADFSA in Figure 3.1. The transitions (or edges, when viewed as a DAG) of the ADFSA are labeled with the feasible pairs and special symbols in the mixed-context sequence. There is only one final state for this ADFSA. The start state (or root node, when viewed as a graph) is labeled 00 and the final state (or terminal node) is marked with a double circle. In Figure 3.1 the final state is labeled 28 . Note that all and only the terminal edges leading to this final state will be

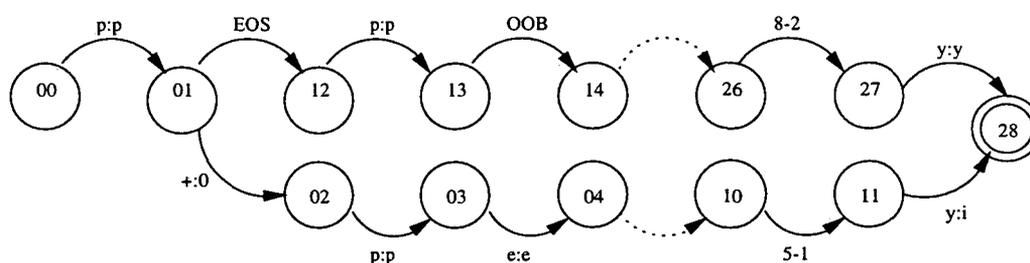


Figure 3.1: ADFSA viewed as a DAG.

labeled with the marker pairs, since they appear at the end of the mixed-context sequences. More than one terminal edge may be labeled with the same marker pair. In Figure 3.1 the two terminal edges are labeled with the marker pairs $y:y$ (a default pair) and $y:i$ (a special pair). Each mixed context is given a unique identification number (IDNO) as the edge label of the edge preceding the terminal edge. An IDNO is used to keep track of the mixed context, the left context and the right context rules which we will write for the same special pair (see Section 4.5).

In Figure 3.1 there are two paths. The path at the bottom, with the IDNO $5-1$, consists of edges labeled with the feasible pairs of the mixed context in Example 34. The path at the top, of which the edges are labeled with the feasible pairs of the mixed context in Example 37, has the IDNO $8-2$.

All the possible (mixed) contexts of a specific marker pair can be recovered by following every path from the root to the terminal edges labeled with that marker pair. If a path is traversed only up to an intermediate edge, a shortened context surrounding the marker pair can be extracted. We will call such an intermediate edge a *delimiter edge*, since it delimits a shortened context. For example, traversing the path which has the marker

pair $y:i$ in Figure 3.1 up to $+:0$ would result in the (unmixed) shortened context:

[38]

$$p:p - +:0$$

From this shortened context we can write a two-level rule with the marker pair $y:i$ as the correspondence part:

[39]

$$y:i \text{ op } p:p - +:0$$

This rule is more general than a rule using the full context:

[40]

$$y:i \text{ op } SOS \ h:h \ a:a \ p:p \ p:p - +:0 \ e:e \ r:r \ EOS$$

For each marker pair in the DAG which is also a special pair, we want to find those delimiter edges which produce the shortest contexts providing a *true* answer to at least one of the two rule type decision questions given on page 34. Note that those two questions (as stated by Antworth) do not provide a way to generate the environment (i.e. left- and right contexts) for which question 1 or question 2 will be *true*. They only allow us to test a given environment, but they do not tell us how to determine that environment procedurally for each special pair. The contribution of this thesis is to provide a procedural way to determine the environment E for the two questions. To this end the two questions are rephrased here in terms of the mixed-context ADFSA, viewed as a DAG⁴:

⁴The determination of this procedural formulation of the two questions will be described formally and in detail in the next chapter.

Question 1 Traverse all the paths in the DAG from the root to the terminal edges labeled with the marker pair $L:S$. Construct a set of edges \mathcal{E}_1 which contains a single edge for each of the paths traversed. Furthermore, no terminal edge labeled with the same L-component (i.e. lexical component) as the marker pair, and with a S-component (i.e. surface component) the same as the L-component, may be reachable from any edge in \mathcal{E}_1 . Then question 1 is *true* for the environment E constructed from the shortened mixed contexts associated with the path prefixes delimited by the edges in \mathcal{E}_1 .

Question 2 Traverse all the paths in the DAG from the root to the terminal edges labeled with the marker pair $L:S$. Construct a set of edges \mathcal{E}_2 which contains a single edge for each of the paths traversed. Furthermore, no terminal edge labeled with the same L-component as the marker pair, but with a different S-component, may be reachable from any edge in \mathcal{E}_2 . Then question 2 is *true* for the environment E constructed from the shortened mixed contexts associated with the path prefixes delimited by the edges in \mathcal{E}_2 .

For each marker pair which is a special pair, we traverse the DAG and mark the delimiter edges *nearest* to the root which allow a *true* answer to either question 1 or question 2. If both questions are *true* for a given edge, then that edge is marked *true* for both questions. This means that each path from the root to a terminal edge can have at most two marked delimiter edges: One delimiting a context for a \Rightarrow rule and one delimiting a context for a \Leftarrow rule. The marker pair used to answer the two questions serves as the correspondence part (CP) of the rule (Section 1.2.2).

To continue with the DAG in Figure 3.1, the edge labeled with $+:0$ (on

the path with the marker pair $y:i$) is the closest edge to the root which answers *true* to question 1. Thus the \Rightarrow rule is indicated (from Table 3.1):

[41]

$$y:i \Rightarrow p:p - +:0$$

Furthermore, the edge labeled with $+:0$ answers true to question 2 as well, thus a \Leftarrow rule is also extracted:

[42]

$$y:i \Leftarrow p:p - +:0$$

Note that the labels of the edges traversed from the root up to and including the delimiter edge, are used as the feasible pairs in the contexts of the extracted rules. For the rule in Example 42, the delimiter edge is the first edge labeled with $+:0$ on the path having $y:i$ as the terminal edge label.

3.3.2 Left or Right Contexts

The mixed-context representation has one obvious drawback: If an optimal rule has only a left or only a right context, it cannot be acquired. To solve this problem, two additional ADFSAs are constructed: One containing only the left context information for all the marker pairs and one containing only the right context information. The same process is then followed as with the mixed contexts. This will allow us to select the optimal rule (i.e. the rule with the shortest discerning context) from the output of one of the three different ADFSAs. The three possible rules for a particular special pair from the three ADFSAs are compared with each other and the optimal rule is selected. Thus, if the optimal rule for a particular special pair is in

the output for the right-context ADFSA for that special pair, then that rule is selected and the output of the other two ADFSA's is ignored.

To continue our example, the right-context information of the mixed contexts in Example 34 and Example 37 is presented in the DAG in Figure 3.2. This DAG is constructed from the two right-context sequences:

[43]

$$+:0, e:e, r:r, EOS, 5-1, y:i$$

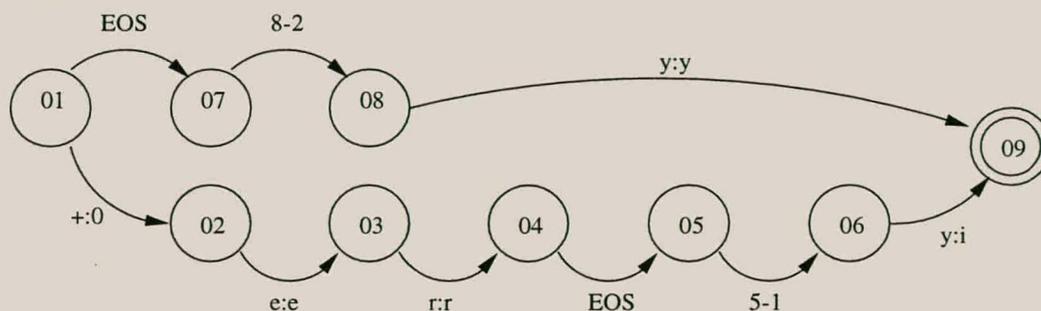
$$EOS, 8-2, y:y$$


Figure 3.2: Right-context ADFSA viewed as a DAG.

From the right-context DAG in Figure 3.2 we can extract another minimal \Rightarrow and \Leftarrow rule for the marker pair $y:i$. By traversing only the first edge labeled with $+:0$ we can extract the rules

[44]

$$y:i \Rightarrow - +:0$$

and

[45]

$$y:i \leftarrow - +:0$$

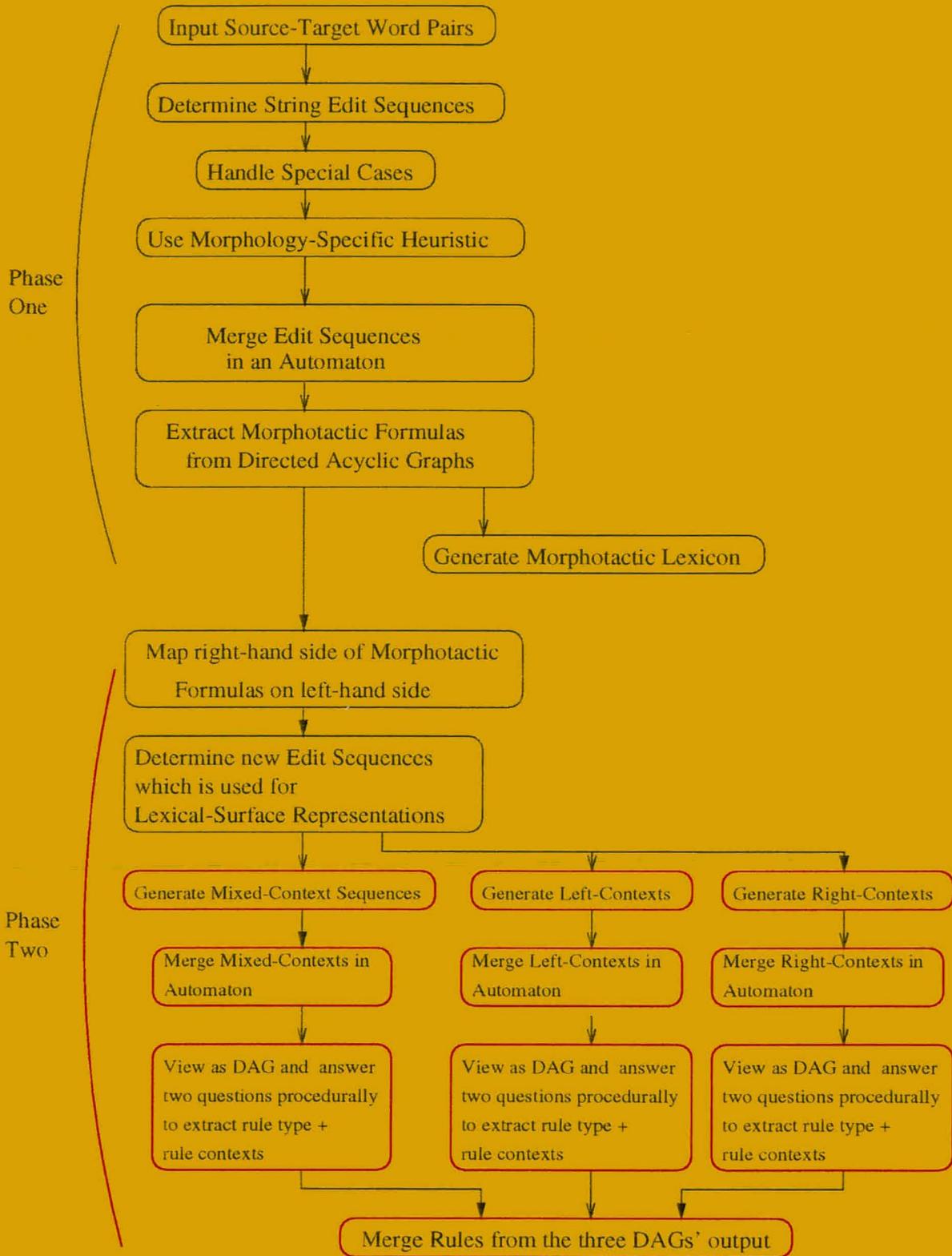
since both question 1 and question 2 are *true* for this edge labeled with $+:0$.

The final set of rules is selected from the output of all three the ADFSAs: For each special pair we select the \leftarrow and \Rightarrow rules which are the shortest and have the least ambiguity for each occurrence of the special pair. The rule which has the least ambiguity is the one which has a context which occurs the least (preferably not at all) as the context of rules of other special pairs. Thus the rule with the least ambiguity is the rule which has the maximal discerning context for that special pair, relative to the contexts of the rules of all other special pairs. For example, we select the rule in Example 44 instead of the rule in Example 41, since it has a shorter context. Furthermore, we select the rule in Example 45 above the rule in Example 42 since it also has a shorter context.

The rule set learned is *complete* since all possible combinations of marker pairs, rule types and contexts are considered by traversing all three DAGs. Furthermore, the rules in the set have the *shortest* possible contexts, since, for a given DAG, there is only one delimiter edge closest to the root for each path, marker pair and rule type combination.

In this chapter I have given an overview of how a set of rules is obtained from the morphotactics of a set of input word pairs. In the next chapter this will be explained formally and in detail.

Rule Acquisition Process Overview



Chapter 4

Acquiring Two-Level Rules: Formal Analysis

4.1 Introduction

In this chapter we explain formally and in detail how a set of two-level rules is acquired from the morphotactics of a set of input word pairs. The previous chapter introduced the two rule-type decision questions that need to be answered during the two-level rule acquisition process.

In the three sections, Section 4.2 to Section 4.4, in this chapter, the two rule-type decision questions are rephrased three times. This is done in order to explain formally how the procedural rephrasing (Chapter 3, page 40) of the two questions is reached. Each successive rephrasing is in more detail and builds on the concepts and reasoning used in the previous rephrasings: In Section 4.2 the two questions are rephrased in terms of full mixed contexts. Next, in Section 4.3, the conditions for the questions to be *true* are rephrased in terms of shortened mixed contexts. These first two sections below introduce the concepts and reasoning which are used in Section 4.4.

In Section 4.4 the conditions for the two questions to be *true* are rephrased in a procedural way, in terms of shortened paths in a DAG. The use of a DAG allows us enough procedural detail to implement a program which automatically computes minimal environments which provide *true* answers to either of the two rule-type decision questions.

The acquisition of left and right contexts are discussed in Section 4.5. Section 4.6 describes the special handling of insertion rules and Section 4.7 summarizes the chapter.

4.2 Two Questions in terms of full mixed-context sets

In this section we show more formally how the two questions can be rephrased in terms of sets of full mixed-context sequences. The concepts and reasoning in this section will be developed further in Section 4.3 and Section 4.4. This will be done until the two questions are rephrased in a detailed enough procedural way to implement a program to automatically calculate the environments to answer the two questions.

Throughout the next three sections we will use the set of edit sequences of the following four Xhosa noun-locative morphotactic equations, as an example:

[46]

<u>Locative</u>	=	<u>Prefix + Noun + Suffix</u>	<u>Glossary</u>
<i>ekhayeni</i>	=	<i>e + ikhaya + ni</i>	<i>at the house</i>
<i>ezinkomeni</i>	=	<i>e + iinkomo + ni</i>	<i>at/on the cows</i>
<i>ehasheni</i>	=	<i>e + ihashe + ni</i>	<i>at/on the horse</i>
<i>elangeni</i>	=	<i>e + ilanga + ni</i>	<i>at/in the sun</i>

\mathcal{W} denotes the set of full (i.e. not shortened) final input edit sequences:

[47]

$$\mathcal{W} = \{ w_1, w_2, w_3, w_4 \}$$

$w_1 =$	<i>e:e +:0 i:0 k:k h:h a:a y:y a:e +:0 n:n i:i,</i>
$w_2 =$	<i>e:e +:0 i:0 h:h a:a s:s h:h e:e +:0 n:n i:i,</i>
$w_3 =$	<i>e:e +:0 i:0 l:l a:a n:n g:g a:e +:0 n:n i:i,</i>
$w_4 =$	<i>e:e +:0 i:z i:i n:n k:k o:o m:m o:e +:0 n:n i:i</i>

These edit sequences serve as the lexical-surface representations of the input source-target word pairs.

We define $\mathcal{F} = F(\mathcal{W})$ to be the set of feasible pairs occurring in the input edit sequences in \mathcal{W} . For this example (i.e. Example 47) $\mathcal{F} = F(\mathcal{W}) = \{ a:a e:e g:g h:h i:i k:k l:l m:m n:n o:o a:e i:0 i:z o:e +:0 \}$.

\mathcal{F}^* denotes the set of possible-strings (or sequences) over the alphabet \mathcal{F} of feasible pairs. We have $\mathcal{F}^+ = \mathcal{F}^* - \{\lambda\}$, where λ is the empty string. Note that $\mathcal{W} \subset \mathcal{F}^+$.

Let $\mathcal{U} = U(\mathcal{F})$ be the union of the set of special symbols used in the mixed-context format, $\{SOS, EOS, OOB\}$, with the set of feasible pairs (\mathcal{F}). We have $\mathcal{U}^+ = \mathcal{U}^* - \{\lambda\}$, where λ is the empty string. If xy is a sequence in \mathcal{U}^+ , then x is a *prefix* and y a *suffix*.

$\mathcal{B} = B(\mathcal{W})$ is the set of special pairs occurring in the input edit sequences ($\mathcal{B} \subseteq \mathcal{F}$). In our running example we have $\mathcal{B} = B(\mathcal{W}) = \{ a:e \ i:0 \ i:z \ o:e \}$.

For now, we do not consider $+:0$ to be a special pair, since $+$ is always mapped to 0 , and the contexts of morpheme boundaries are specified in the morphotactic description. Thus, rules having $+:0$ as the correspondence part are considered redundant and will not be acquired in this thesis.

For the purpose of constructing the mixed contexts, we consider the feasible pairs in \mathcal{F} to be atomic. However, to answer the two questions, we need to be able to address the lexical (or L-) and surface (or S-) component of a marker pair (see question 1, page 40). Thus, we define $L(f)$ to be the L-component of feasible pair f , and $S(f)$ to be the S-component. We define $\mathcal{L} = L(\mathcal{B})$ to be the set of L-components of the special pairs occurring in the input edit sequences. In our example we have $\mathcal{L} = L(\mathcal{B}) = \{ a \ i \ o \}$.

We define $\mathcal{F}_{\mathcal{L}} \subseteq \mathcal{F}$ to be the set of feasible pairs which have an L-component which is in \mathcal{L} . Thus $t \in \mathcal{F}_{\mathcal{L}}$ if and only if $t \in \mathcal{F}$ and $L(t) \in \mathcal{L}$. Note that $B(\mathcal{W}) \subseteq \mathcal{F}_{\mathcal{L}} \subseteq F(\mathcal{W})$. For our example, we have $\mathcal{F}_{\mathcal{L}} = \mathcal{F}_{L(B(\mathcal{W}))} = \{ i:o \ a:a \ a:e \ o:e \ i:i \ o:o \ i:z \}$. Note that $\mathcal{F}_{\mathcal{L}}$ contains all the special pairs as well as those default pairs (e.g. $a:a$, $i:i$ and $o:o$) which have an L-component equal to one (or more) special pairs (e.g. $L(a:a) = "a" = L(a:e)$).

We define $\mathcal{C}_{\mathcal{L}}^{full} = \mathcal{C}_{\mathcal{L}}^{full}(\mathcal{W})$ to be the set of full mixed-context sequences written for every occurrence in the string-edit sequences in \mathcal{W} , of each feasible pair in $\mathcal{F}_{\mathcal{L}}$. For our example the set of full mixed-context sequences for each feasible pair in $\mathcal{F}_{\mathcal{L}}$ is:

[48]

$$\mathcal{C}_{\mathcal{L}}^{full} = \{ c_1, c_2, c_3, \dots, c_{16} \}$$

$c_1 =$	$+ :0 \text{ l:l e:e a:a SOS n:n OOB g:g OOB a:e OOB } + :0 \text{ OOB n:n OOB i:i}$ OOB EOS i:0,
$c_2 =$	$+ :0 \text{ k:k e:e h:h SOS a:a OOB y:y OOB a:e OOB } + :0 \text{ OOB n:n OOB i:i}$ OOB EOS i:0,
$c_3 =$	$+ :0 \text{ h:h e:e a:a SOS s:s OOB h:h OOB e:e OOB } + :0 \text{ OOB n:n OOB i:i}$ OOB EOS i:0,
$c_4 =$	$\text{h:h y:y k:k a:e i:0 } + :0 \text{ } + :0 \text{ n:n e:e i:i SOS EOS a:a,}$
$c_5 =$	$\text{l:l n:n i:0 g:g } + :0 \text{ a:e e:e } + :0 \text{ SOS n:n OOB i:i OOB EOS a:a,}$
$c_6 =$	$\text{h:h s:s i:0 h:h } + :0 \text{ e:e e:e } + :0 \text{ SOS n:n OOB i:i OOB EOS a:a,}$
$c_7 =$	$\text{y:y } + :0 \text{ a:a n:n h:h i:i k:k EOS i:0 OOB } + :0 \text{ OOB e:e OOB SOS OOB a:e,}$
$c_8 =$	$\text{g:g } + :0 \text{ n:n n:n a:a i:i l:l EOS i:0 OOB } + :0 \text{ OOB e:e OOB SOS OOB a:e,}$
$c_9 =$	$\text{m:m } + :0 \text{ o:o n:n k:k i:i n:n EOS i:i OOB i:z OOB } + :0 \text{ OOB e:e OOB SOS}$ OOB o:e,
$c_{10} =$	$\text{i:z n:n } + :0 \text{ k:k e:e o:o SOS m:m OOB o:e OOB } + :0 \text{ OOB n:n OOB i:i OOB}$ EOS i:i,
$c_{11} =$	$\text{n:n EOS } + :0 \text{ OOB e:e OOB h:h OOB s:s OOB a:a OOB h:h OOB i:0 OOB}$ $\text{ } + :0 \text{ OOB e:e OOB SOS OOB i:i,}$
$c_{12} =$	$\text{n:n EOS } + :0 \text{ OOB a:e OOB y:y OOB a:a OOB h:h OOB k:k OOB i:0 OOB}$ $\text{ } + :0 \text{ OOB e:e OOB SOS OOB i:i,}$
$c_{13} =$	$\text{n:n EOS } + :0 \text{ OOB a:e OOB g:g OOB n:n OOB a:a OOB l:l OOB i:0 OOB}$ $\text{ } + :0 \text{ OOB e:e OOB SOS OOB i:i,}$
$c_{14} =$	$\text{n:n EOS } + :0 \text{ OOB o:e OOB m:m OOB o:o OOB k:k OOB n:n OOB i:i OOB}$ $\text{ i:z OOB } + :0 \text{ OOB e:e OOB SOS OOB i:i,}$
$c_{15} =$	$\text{k:k m:m n:n o:e i:i } + :0 \text{ i:z n:n } + :0 \text{ i:i e:e EOS SOS OOB o:o,}$
$c_{16} =$	$\text{ } + :0 \text{ i:i e:e n:n SOS k:k OOB o:o OOB m:m OOB o:e OOB } + :0 \text{ OOB n:n OOB}$ i:i OOB EOS i:z

Note that more than one full mixed context can originate from the same edit sequence. For example, the mixed contexts which originate from the edit sequence $w_1 = "e:e +:0 i:0 k:k h:h a:a y:y a:e +:0 n:n i:i"$ are:

[49]

$$C_L^{full}(\{w_1\}) = \{c_2, c_4, c_7, c_{12}\}$$

$c_2 =$ +:0 k:k e:e h:h SOS a:a OOB y:y OOB a:e OOB +:0 OOB n:n OOB i:i OOB EOS i:0, $c_4 =$ h:h y:y k:k a:e i:0 +:0 +:0 n:n e:e i:i SOS EOS a:a, $c_7 =$ y:y +:0 a:a n:n h:h i:i k:k EOS i:0 OOB +:0 OOB e:e OOB SOS OOB a:e, $c_{12} =$ n:n EOS +:0 OOB a:e OOB y:y OOB a:a OOB h:h OOB k:k OOB i:0 OOB +:0 OOB e:e OOB SOS OOB i:i,

Similarly, we define \mathcal{C}_t^{full} , with $t \in \mathcal{F}_{\mathcal{L}}$, to be the set of all the full mixed-context sequences of which t is the marker pair. For example, if $t = "a:e"$ then the full mixed-context sequences of t is (by inspection of Example 48):

[50]

$$\mathcal{C}_{a:e}^{full} = \{ c_7, c_8 \}$$

$c_7 =$ y:y +:0 a:a n:n h:h i:i k:k EOS i:0 OOB +:0 OOB e:e OOB SOS OOB a:e, $c_8 =$ g:g +:0 n:n n:n a:a i:i l:l EOS i:0 OOB +:0 OOB e:e OOB SOS OOB a:e

\mathcal{C}_t^{full} fully specifies the environment of $t \in \mathcal{F}_{\mathcal{L}}$ in \mathcal{W} , since there is a one-to-one mapping of full contexts to mixed contexts.

In addition we define \mathcal{C}_l^{full} , with $l \in \mathcal{L}$ to be the set of all the full mixed-context sequences of which the marker pair has an L-component equal to l . For example,

[51]

$$\mathcal{C}_{L(a:e)}^{full} = \mathcal{C}_a^{full} = \{ c_4, c_5, c_6, c_7, c_8 \}$$

$c_4 =$ h:h y:y k:k a:e i:0 +:0 +:0 n:n e:e i:i SOS EOS a:a, $c_5 =$ l:l n:n i:0 g:g +:0 a:e e:e +:0 SOS n:n OOB i:i OOB EOS a:a, $c_6 =$ h:h s:s i:0 h:h +:0 e:e e:e +:0 SOS n:n OOB i:i OOB EOS a:a, $c_7 =$ y:y +:0 a:a n:n h:h i:i k:k EOS i:0 OOB +:0 OOB e:e OOB SOS OOB a:e, $c_8 =$ g:g +:0 n:n n:n a:a i:i l:l EOS i:0 OOB +:0 OOB e:e OOB SOS OOB a:e

We define the function $unmix(x)$ of a mixed context x to be the “unmixed” context, of which x is the mixed-context representation. The marker pair is indicated with bold type-face in the “unmixed” context. For example, $unmix(c_7) =$

$unmix(“y:y +:0 a:a n:n h:h i:i k:k EOS i:0 OOB +:0 OOB e:e OOB SOS OOB \mathbf{a:e}”)$ =

$“e:e +:0 i:0 k:k h:h a:a y:y \mathbf{a:e} +:0 n:n i:i”$. When we want to use the normal two-level rule context notation and when it is clear which feasible pair is the marker pair, we will write only an underscore (“_”) to indicate the position of the marker pair in the “unmixed” context. In this case, the above example would be written as

$“e:e +:0 i:0 k:k h:h a:a y:y _ +:0 n:n i:i”$.

$mix(y) = x$ is the inverse function of $unmix$.

We define the function $conc()$ (for “concatenate”) as follows:

Definition 1 Let \mathcal{X}_t be a set of mixed contexts for the marker pair t . Then $conc(\mathcal{X}_t) = unmix(c_1)|unmix(c_2)|\dots|unmix(c_{|\mathcal{X}_t|})$, where $c_i \in \mathcal{X}_t$ and the vertical bar “|” signifies the disjunction¹ of the contexts.

For example, $conc(C_{a:e}^{full}) = unmix(c_7)|unmix(c_8) =$

$“e:e +:0 i:0 k:k h:h a:a y:y _ +:0 n:n i:i” |$

$“e:e +:0 i:0 l:l a:a n:n g:g _ +:0 n:n i:i”$. Here it is clear that these two disjuncted contexts belong to the marker pair $a:e$, since they are concatenated from $C_{a:e}^{full}$. We can thus use the normal context notation with the underscore (“_”) as place holder for the marker pair.

$conc()$ will be used to write an environment E (i.e. contexts) from the

¹The term *disjunction* is used in the two-level literature and means the logical OR of two or more contexts.

mixed-context set of a special pair, in the rephrasing of the two questions below. Note that $\text{conc}(\mathcal{C}_{a:e}^{full})$ provides an environment E which fully specifies all the occurrences of “ $a:e$ ” in the input edit sequences in \mathcal{W} .

Let $v_j^k \in \mathcal{U}$ be the k 'th element of a mixed-context sequence c_j . The length of a mixed-context sequence is indicated by surrounding vertical bars, e.g. $|c_j|$. Then $v_j^{|c_j|}$ indicates the marker pair since it is the last element in the mixed-context sequence c_j .

Note that the two original rule-type decision questions do not provide procedures for determining either the environment E or the special pair $L:S$. They are stated in a declarative way as conditions which should hold for them to be *true*. As a first step towards a procedural answer of the two questions, the above definitions allow us to rephrase the two rule-type decision questions in terms of full mixed-context sets:

Question 1 Do all and only $c_j \in \mathcal{C}_s^{full}$ have s as a marker pair, i.e. is $v_j^{|c_j|} = s$ for every $c_j \in \mathcal{C}_s^{full}$? Then question 1 is *true* for the environment $E = \text{conc}(\mathcal{C}_s^{full}) = \text{unmix}(c_1)|\text{unmix}(c_2)|\dots|\text{unmix}(c_{|\mathcal{C}_s^{full}|})$.

Question 2 Must $L(s)$ always be realized as $S(s)$ for every $c_j \in \mathcal{C}_s^{full}$, i.e. is $S(v_j^{|c_j|}) = S(s)$ for every $c_j \in \mathcal{C}_s^{full}$? Then question 2 is *true* for the environment

$$E = \text{conc}(\mathcal{C}_s^{full}) = \text{unmix}(c_1)|\text{unmix}(c_2)|\dots|\text{unmix}(c_{|\mathcal{C}_s^{full}|})$$

By definition, this rephrased question 1 is always true, since \mathcal{C}_s^{full} contains all and only those full mixed contexts which have s as a marker pair.

To continue with our example, let s be the special pair $i:z$. Then $\mathcal{C}_{i:z}^{full} = \{c_{16}\} =$

$\{ +:0 \ i:i \ e:e \ n:n \ SOS \ k:k \ OOB \ o:o \ OOB \ m:m \ OOB \ o:e \ OOB \ +:0 \ OOB \ n:n \ OOB \ i:i \ OOB \ EOS \ i:z \}$.

Thus question 1 is *true* for $E = \text{conc}(\{c_{16}\}) = \text{unmix}(c_{16}) =$
 $SOS\ e:e\ +:0\ -\ i:i\ n:n\ k:k\ o:o\ m:m\ o:e\ +:0\ n:n\ i:i\ EOS.$

Since question 1 is *true* for this context, the \Rightarrow rule is indicated (from Table 3.1) for the marker pair $i:z$:

[52]

$$i:z \Rightarrow SOS\ e:e\ +:0\ -\ i:i\ n:n\ k:k\ o:o\ m:m\ o:e\ +:0\ n:n\ i:i\ EOS$$

Question 2 must be answered for the same special pair s as in question 1. With our example $L(i:z) = i$ and $S(i:z) = z$. Note that this i is not an index but the *letter* i . This rephrasal of question 2 is also true by definition, since every full mixed context $c_j \in \mathcal{C}_s^{full}$ has the same marker pair (i.e. $v_j^{|c_j|} = s$, for every $j \in [1..|\mathcal{C}_s^{full}|]$) and thus the same surface component of the marker pair, $S(v_j^{|c_j|}) = S(s)$, for every $j \in [1..|\mathcal{C}_s^{full}|]$.

To continue with our example, let s be the special pair $i:z$. Then $\mathcal{C}_{i:z}^{full} =$
 $\{c_{16}\} =$
 $\{ +:0\ i:i\ e:e\ n:n\ SOS\ k:k\ OOB\ o:o\ OOB\ m:m\ OOB\ o:e\ OOB\ +:0\ OOB$
 $n:n\ OOB\ i:i\ OOB\ EOS\ i:z \}.$

Thus question 2 is *true* for $E = \text{conc}(\{c_{16}\}) = \text{unmix}(c_{16}) =$
 $SOS\ e:e\ +:0\ -\ i:i\ n:n\ k:k\ o:o\ m:m\ o:e\ +:0\ n:n\ i:i\ EOS.$

Since question 2 is *true* for this context, the \Leftarrow rule is indicated (from Table 3.1) for the marker pair $i:z$:

[53]

$$i:z \Leftarrow SOS\ e:e\ +:0\ -\ i:i\ n:n\ k:k\ o:o\ m:m\ o:e\ +:0\ n:n\ i:i\ EOS$$

In this section I have rephrased the two rule-type decision questions in terms of full mixed-context sets. However, two-level rules acquired with this

rephrasal have full contexts and thus do not generalize to be successfully applied to previously unseen but similar words. In the next section I will rephrase the two questions in terms of shortened contexts, which allow the determination of rules with minimal discerning contexts. These general rules can then be successfully applied to previously unseen but similar words.

4.3 Two Questions in terms of shortened mixed-context sets

Now we will rephrase the two questions in terms of sets for the interesting case of *shortened* (mixed) contexts for a given special pair. Shortened contexts are interesting and useful since they allow us to determine the minimal discerning rule contexts (as opposed to the full contexts of the previous section) necessary to correctly analyze and generate a set of input source-target word pairs. As stated in Section 3.3.1 on page 36, it is important to acquire the minimal discerning context for each rule. This ensures that the rules are as general as possible (to work on unseen words as well) while limiting overgeneration and overrecognition.

The two questions are rephrased as functions on sets which provide conditions for them to be *true*. These conditions require the notions of a *discerning prefix-partitioner* as well as an *L-relative discerning prefix-partitioner* of a set of mixed-context sequences. A discerning prefix-partitioner is used to construct a context for a \Rightarrow rule and an L-relative discerning prefix-partitioner is used to construct a context for a \Leftarrow rule.

First we define a *prefix-partitioned full-context set* C_f^x of a feasible pair f , as follows:

Definition 2 A prefix-partitioned full-context set $C_f^x \subseteq C_f^{full}$ of feasible pair $f \in \mathcal{F}$ contains all and only the full mixed contexts $c_j \in C_f^{full}$ of which $x \in U^+$ is a prefix.

To continue from Example 48: We have the mixed-context set $C_{i:0}^{full} = \{c_1, c_2, c_3\}$ with

$c_1 = \quad +:0 \text{ l:l e:e a:a SOS n:n OOB g:g OOB a:e OOB } +:0 \text{ OOB n:n OOB i:i OOB EOS i:0,}$
$c_2 = \quad +:0 \text{ k:k e:e h:h SOS a:a OOB y:y OOB a:e OOB } +:0 \text{ OOB n:n OOB i:i OOB EOS i:0,}$
$c_3 = \quad +:0 \text{ h:h e:e a:a SOS s:s OOB h:h OOB e:e OOB } +:0 \text{ OOB n:n OOB i:i OOB EOS i:0,}$

Thus it follows that the prefix-partitioned full-context set $C_{i:0}^{“+:0 l:l”} = \{c_1\} =$

$\{“+:0 \text{ l:l e:e a:a SOS n:n OOB g:g OOB a:e OOB } +:0 \text{ OOB n:n OOB i:i OOB EOS i:0”\}$.

We define a *discerning prefix partitioner* of a mixed-context set as follows:

Definition 3 Let $\mathcal{H}_s = H(C_s^{full}) \subseteq U^+$ be a set which contains a prefix string for every mixed context $c_j \in C_s^{full}$, such that none of the prefixes in \mathcal{H}_s is the prefix of another element of \mathcal{H}_s . Furthermore, none of the prefixes in \mathcal{H}_s must be a prefix of a mixed context which has a default pair as the marker pair, with $L(f) = L(s)$. We call this $\mathcal{H}_s = H(C_s^{full})$ a discerning prefix partitioner of C_s^{full} .

The name discerning prefix partitioner comes from the fact that the set of prefixes in \mathcal{H}_s partitions the mixed-context set C_s^{full} into subsets, each of which contains mixed contexts which have another prefix in \mathcal{H}_s as a prefix. Furthermore, the prefixes in \mathcal{H}_s are only prefixes of mixed contexts which

are *discerned* (or *separated*) from those mixed contexts which have a default pair with the same L-component as s , as marker pair.

Note that one $x \in \mathcal{H}_s$ may be a prefix of more than one mixed context $c \in \mathcal{C}_s^{full}$. By inspecting the mixed-context set $\mathcal{C}_{i:0}^{full}$ above, we can see that “+:0” is a common prefix of all the elements c_1, c_2 and c_3 . Furthermore, “+:0” is not a prefix of any mixed context with the default pair $i:i$ as marker pair (by inspecting Example 48). Thus, $\mathcal{H}_{i:0}^1 = H(\mathcal{C}_{i:0}^{full}) = \{“+:0”\}$ is a discerning prefix partitioner. We use the superscript “1” to indicate that $\mathcal{H}_{i:0}^1$ is the first discerning prefix partitioner which we selected. Another discerning prefix partitioner is $\mathcal{H}_{i:0}^2 = \{“+:0 l:l”, “+:0 k:k”, “+:0 h:h”\}$. The superscript “2” in $\mathcal{H}_{i:0}^2$ is used to show this is the second discerning prefix partitioner which we have selected for $\mathcal{C}_{i:0}^{full}$.

$\mathcal{H}_{i:0}^* = \{“+:0”, “+:0 k:k”, “+:0 h:h”\}$ is *not* a discerning prefix partitioner, since “+:0” is a prefix of “+:0 k:k” and “+:0 h:h”.

$\mathcal{H}_s^{min} = H^{min}(\mathcal{C}_s^{full})$ denotes the *minimal discerning prefix partitioner* of the mixed-context set \mathcal{C}_s^{full} :

Definition 4 \mathcal{H}_s^{min} is a *minimal discerning prefix partitioner* of \mathcal{C}_s^{full} if no other discerning prefix partitioner \mathcal{H}'_s exists which contains an element y which is a true prefix of any element $x \in \mathcal{H}_s^{min}$, i.e. there does not exist a (non null) string $z \in \mathcal{U}^+$ such that $x = yz$, where $x \in \mathcal{H}_s^{min}$, $y \in \mathcal{H}'_s$.

Note that the discerning prefix partitioner $\mathcal{H}_{i:0}^1$ above is also the *minimal* discerning prefix partitioner $\mathcal{H}_{i:0}^{min}$, since the prefix cannot be made shorter, while still keeping $\mathcal{H}_{i:0}^1$ a discerning prefix partitioner.

To continue our example: From Example 48, we have the set $\mathcal{C}_{L(i:0)}^{full} = \mathcal{C}_i^{full} = \{c_1, c_2, c_3, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{16}\}$, with

$c_1 =$	$+:0$ l:l e:e a:a SOS n:n OOB g:g OOB a:e OOB $+:0$ OOB n:n OOB i:i OOB EOS $i:0$,
$c_2 =$	$+:0$ k:k e:e h:h SOS a:a OOB y:y OOB a:e OOB $+:0$ OOB n:n OOB i:i OOB EOS $i:0$,
$c_3 =$	$+:0$ h:h e:e a:a SOS s:s OOB h:h OOB e:e OOB $+:0$ OOB n:n OOB i:i OOB EOS $i:0$,
$c_{10} =$	i:z n:n $+:0$ k:k e:e o:o SOS m:m OOB o:e OOB $+:0$ OOB n:n OOB i:i OOB EOS $i:i$,
$c_{11} =$	n:n EOS $+:0$ OOB e:e OOB h:h OOB s:s OOB a:a OOB h:h OOB i:0 OOB $+:0$ OOB e:e OOB SOS OOB $i:i$,
$c_{12} =$	n:n EOS $+:0$ OOB a:e OOB y:y OOB a:a OOB h:h OOB k:k OOB i:0 OOB $+:0$ OOB e:e OOB SOS OOB $i:i$,
$c_{13} =$	n:n EOS $+:0$ OOB a:e OOB g:g OOB n:n OOB a:a OOB l:l OOB i:0 OOB $+:0$ OOB e:e OOB SOS OOB $i:i$,
$c_{14} =$	n:n EOS $+:0$ OOB o:e OOB m:m OOB o:o OOB k:k OOB n:n OOB i:i OOB i:z OOB $+:0$ OOB e:e OOB SOS OOB $i:i$,
$c_{16} =$	$+:0$ i:i e:e n:n SOS k:k OOB o:o OOB m:m OOB o:e OOB $+:0$ OOB n:n OOB i:i OOB EOS i:z

Note that the $i = L(i:0)$ is not an index but the *letter* i .

Let f be a feasible pair. Then $L(f):\neg S(f)$ is the standard notation for the set of feasible pairs which have the same L-component as f , but not the same S-component. This set of feasible pairs allows us to write the full mixed-context set for the feasible pairs which have the same L-component as f but not the same S-component. This full mixed-context set is used later in the definition of an L-relative discerning prefix partitioner below (Definition 5). The set

$$\text{Equation 1 } C_{L(f):\neg S(f)}^{full} = C_{L(f)}^{full} - C_f^{full}$$

denotes the set of mixed-context sequences with marker pairs which have the same L-component as the feasible pair f , but not the same S-component

as f .

$$\begin{aligned} \text{To continue the example, we have } C_{L(f):-S(f)}^{full} &= C_{L(f)}^{full} - C_f^{full} = \\ C_{i:-0}^{full} &= C_i^{full} - C_{i:0}^{full} = \\ \{c_1, c_2, c_3, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{16}\} - \{c_1, c_2, c_3\} &= \{c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{16}\}, \end{aligned}$$

with

$c_{10} =$	i:z n:n +:0 k:k e:e o:o SOS m:m OOB o:e OOB +:0 OOB n:n OOB i:i OOB EOS i:i,
$c_{11} =$	n:n EOS +:0 OOB e:e OOB h:h OOB s:s OOB a:a OOB h:h OOB i:0 OOB +:0 OOB e:e OOB SOS OOB i:i,
$c_{12} =$	n:n EOS +:0 OOB a:e OOB y:y OOB a:a OOB h:h OOB k:k OOB i:0 OOB +:0 OOB e:e OOB SOS OOB i:i,
$c_{13} =$	n:n EOS +:0 OOB a:e OOB g:g OOB n:n OOB a:a OOB l:l OOB i:0 OOB +:0 OOB e:e OOB SOS OOB i:i,
$c_{14} =$	n:n EOS +:0 OOB o:e OOB m:m OOB o:o OOB k:k OOB n:n OOB i:i OOB i:z OOB +:0 OOB e:e OOB SOS OOB i:i,
$c_{16} =$	+:0 i:i e:e n:n SOS k:k OOB o:o OOB m:m OOB o:e OOB +:0 OOB n:n OOB i:i OOB EOS i:z

Now we can define the L -relative discerning prefix partitioner \mathcal{H}_s^{Lrel} of a special pair s :

Definition 5 An L -relative discerning prefix partitioner $\mathcal{H}_s^{Lrel} =$

$H^{Lrel}(C_s^{full}, C_{L(s):-S(s)}^{full})$ of the special pair s is a discerning prefix partitioner of C_s^{full} such that there is a prefix $x \in \mathcal{H}_s^{Lrel}$ for every mixed context sequence $c_j \in C_s^{full}$ but no $x \in \mathcal{H}_s^{Lrel}$ is a prefix of any mixed context sequence $c_k \in C_{L(s):-S(s)}^{full}$.

The “ L -relative” part of the name refers to the fact that the two mixed-context sets C_s^{full} and $C_{L(s):-S(s)}^{full}$ are mixed-context sets for feasible pairs which have the same L -component (but different S -components).

By inspecting $C_{i:0}^{full}$ (page 55) and $C_{i:-0}^{full}$ (above), an L -relative discerning

prefix partitioner of the special pair $i:0$ is $\mathcal{H}_{i:0}^{Lrel} = H^{Lrel}(C_{i:0}^{full}, C_{i:-0}^{full}) = \{ "+:0 l:l", "+:0 k:k", "+:0 h:h" \}$. Thus, $\mathcal{H}_{i:0}^{Lrel}$ contains a prefix for every $c_j \in C_{i:0}^{full}$, but no prefix of $c_k \in C_{i:-0}^{full}$. Note that $\mathcal{H}_{i:0}^1 = H(C_{i:0}^{full}) = \{ "+:0" \}$ selected above as a discerning prefix partitioner, is *not* an L -relative discerning prefix partitioner, since $"+:0"$ is a prefix of $c_{16} \in C_{i:-0}^{full}$.

$\mathcal{H}_f^{minLrel} = H^{minLrel}(C_f^{full})$ denotes the *minimal L-relative discerning prefix partitioner* of C_f^{full} :

Definition 6 $\mathcal{H}_f^{minLrel}$ is a minimal L -relative discerning prefix partitioner of C_f^{full} if no other discerning prefix partitioner \mathcal{H}'_f^{Lrel} exists which contains an element y which is a true prefix of any element $x \in \mathcal{H}_f^{minLrel}$, i.e. there does not exist a (non null) string $z \in \mathcal{U}^+$ such that $x = yz$, where

$$x \in \mathcal{H}_f^{minLrel}, y \in \mathcal{H}'_f^{Lrel}.$$

The minimal L -relative discerning prefix partitioner will be used in question 2's rephrasal to establish the minimal rule environment (or context). As stated in Section 3.3.1 on page 36, it is important to determine the minimal contexts for which the two questions are *true*. The reason for this is that the shortest (minimal) context is the most general context and it can be best applied to previously unseen words.

Recall that we defined the function $unmix(x)$ (Section 4.2, page 51) of a mixed-context prefix x to be the "unmixed" context, of which x is the mixed-context representation, with "-" indicating the position of the marker pair if it is not included in the mixed-context prefix x . For example, we have $unmix("+:0 l:l e:e") = "e:e +:0 - l:l"$. $mix(y) = x$ is the inverse function of $unmix$. We define the subsumption of one context by another as follows:

Definition 7 A (possibly shortened) context y subsumes a (possibly shortened) context z if $mix(y)$ is a prefix of $mix(z)$.

We define the subsumption of one set of mixed-context prefixes by another:

Definition 8 *A set of mixed-context prefixes \mathcal{Y} subsumes another set of mixed-context prefixes \mathcal{Z} if for any $z \in \mathcal{Z}$ there exists a prefix $y \in \mathcal{Y}$.*

We can now define the subsumption of one environment of a special pair by another environment of the same special pair. This subsumption of environments will be used to identify a minimal environment of a special pair for which the two questions are *true*.

Definition 9 *An environment $E_{\mathcal{H}_s^1} = \text{conc}(\mathcal{H}_s^1)$ of the special pair s in \mathcal{W} subsumes another environment $E_{\mathcal{H}_s^2} = \text{conc}(\mathcal{H}_s^2)$ of s in \mathcal{W} if the discerning prefix partitioner \mathcal{H}_s^1 subsumes the discerning prefix partitioner \mathcal{H}_s^2 .*

Furthermore, since the minimal discerning prefix partitioner $\mathcal{H}_s^{\text{min}}$ is not subsumed by any other discerning prefix partitioner \mathcal{H}_s , it follows that the environment $E_{\mathcal{H}_s^{\text{min}}} = \text{conc}(\mathcal{H}_s^{\text{min}})$ of s in \mathcal{W} is not subsumed by any other environment $E_{\mathcal{H}_s} = \text{conc}(\mathcal{H}_s)$ of s in \mathcal{W} . We call the environment (of a special pair s) which is not subsumed by any other environment of s , the *minimal environment* of s .

We call the environment associated with an L-relative discerning prefix partitioner of a special pair s , the *L-relative environment* of s . Note that Definition 9 is also valid for an L-relative environment $E_{\mathcal{H}_s^{\text{Lrel}}} = \text{conc}(\mathcal{H}_s^{\text{Lrel}})$ of a special pair s , since an L-relative discerning prefix partitioner $\mathcal{H}_s^{\text{Lrel}}$ is also a discerning prefix partitioner. Thus we have the following corollary:

Corollary 1 *An L-relative environment $E_{\mathcal{H}_s^{\text{Lrel},1}} = \text{conc}(\mathcal{H}_s^{\text{Lrel},1})$ of the special pair s in \mathcal{W} subsumes another L-relative environment $E_{\mathcal{H}_s^{\text{Lrel},2}} = \text{conc}(\mathcal{H}_s^{\text{Lrel},2})$ of s in \mathcal{W} if the L-relative discerning prefix partitioner $\mathcal{H}_s^{\text{Lrel},1}$ subsumes the L-relative discerning prefix partitioner $\mathcal{H}_s^{\text{Lrel},2}$.*

We call the L-relative environment $E^{\text{minLrel}} = \text{conc}(\mathcal{H}_s^{\text{minLrel}})$ of a special pair s , which is not subsumed by any other L-relative environment of s , the *minimal L-relative environment* of s .

The two questions will now be rephrased (the second time), for each special pair $s \in \mathcal{B}$ of \mathcal{W} , as functions on sets to find all environment formulations E^i for which the questions will be *true*:

[54]

Question 1's Condition: Question 1 is *true* for a special pair s and any environment $E_{\mathcal{H}_s^i} = \text{conc}(\mathcal{H}_s^i)$.

If $\mathcal{H}_s = \mathcal{H}_s^{\text{min}}$ then $E_{\mathcal{H}_s}$ is a *minimal environment* which specifies all the contexts where s is allowed in \mathcal{W} .

Question 2's Condition: Question 2 is *true* for a special pair s and any environment $E_{\mathcal{H}_s^{\text{Lrel},j}} = \text{conc}(\mathcal{H}_s^{\text{Lrel},j})$.

If $\mathcal{H}_s^{\text{Lrel}} = \mathcal{H}_s^{\text{minLrel}}$ then $E_{\mathcal{H}_s^{\text{Lrel}}}$ is a *minimal environment* in which $L(s)$ is always realized as $S(s)$.

Question 1 is considered to be *false* when $\mathcal{H}_s^i = \emptyset$, since this implies an useless null environment. Similarly, question 2 is considered to be *false* when $\mathcal{H}_s^{\text{Lrel},i} = \emptyset$.

Note that since $\mathcal{H}_s^{\text{Lrel}}$ is also a discerning prefix partitioner of $\mathcal{C}_s^{\text{full}}$, question 1 is always true for the environments $E_{\mathcal{H}_s^{\text{Lrel},j}} = \text{conc}(\mathcal{H}_s^{\text{Lrel},j})$ as well (if $\mathcal{H}_s^{\text{Lrel}} \neq \emptyset$). Thus, if we can find an environment which provides a *true* answer to question 2, then we are guaranteed that at least this same environment will also provide a *true* answer to question one.

However, $\mathcal{H}_s^{\text{min}}$ may contain true prefixes of $\mathcal{H}_s^{\text{Lrel}}$, but not the other

way round. Thus, $E_{\mathcal{H}_s}^{min}$ may subsume the environment $E_{\mathcal{H}_s^{Lrel}}^{min}$, but not the other way round.

Furthermore, it is possible that an environment $E_{\mathcal{H}_s}^{min}$ exists (which provides a *true* answer for question 1) and contains subcontexts which do not have a corresponding subcontext in any environment $E_{\mathcal{H}_s^{Lrel}}$.

To construct an environment fitting the conditions for *question 1* for the special pair $i:0$ in our example, we need to find a discerning prefix partitioner $\mathcal{H}_s = H(C_{i:0}^{full})$. From Example 48, we have the full mixed-context set of $i:0$, $C_{i:0}^{full} = \{c_1, c_2, c_3\}$, with

$c_1 =$ +:0 l:l e:e a:a SOS n:n OOB g:g OOB a:e OOB +:0 OOB n:n OOB i:i
OOB EOS i:0,
 $c_2 =$ +:0 k:k e:e h:h SOS a:a OOB y:y OOB a:e OOB +:0 OOB n:n OOB i:i
OOB EOS i:0,
 $c_3 =$ +:0 h:h e:e a:a SOS s:s OOB h:h OOB e:e OOB +:0 OOB n:n OOB i:i
OOB EOS i:0,

By inspection we can see that “+:0” is a common prefix of all the elements c_1, c_2 and c_3 . Furthermore, “+:0” is not a prefix of any mixed context with the default pair $i:i$ as marker pair (from inspection of Example 48). Thus, $\mathcal{H}_{i:0}^1 = H(C_{i:0}^{full}) = \{“+:0”\}$ is a discerning prefix partitioner of $C_{i:0}^{full}$. We use the superscript “1” to indicate that $\mathcal{H}_{i:0}^1$ is the first discerning prefix partitioner which we selected. Note that $\mathcal{H}_{i:0}^1 = \mathcal{H}_{i:0}^{min}$ since the prefix cannot be made shorter, while still keeping $\mathcal{H}_{i:0}^1$ a discerning prefix partitioner. Thus the minimal environment is $E_{\mathcal{H}_{i:0}^1}^{min} = “+:0 - ”$.

To construct $\mathcal{H}_{i:0}^{Lrel} = H^{Lrel}(C_{i:0}^{full})$ for *question 2* we need to inspect $C_{i:0}^{full}$ and $C_{i:-0}^{full}$ to ensure that $\mathcal{H}_{i:0}^{Lrel}$ contains prefixes of $C_{i:0}^{full}$, but not of $C_{i:-0}^{full}$ (see Definition 5). From Equation 1 we know that $C_{i:-0}^{full} = C_i^{full} - C_{i:0}^{full}$. Note that the $i = L(i:0)$ is not an index but the *letter* i . From Example 48, we

have the set $C_i^{full} = \{c_1, c_2, c_3, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{16}\}$, with

$c_1 =$	+ :0 l:l e:e a:a SOS n:n OOB g:g OOB a:e OOB +:0 OOB n:n OOB i:i OOB EOS i:0,
$c_2 =$	+ :0 k:k e:e h:h SOS a:a OOB y:y OOB a:e OOB +:0 OOB n:n OOB i:i OOB EOS i:0,
$c_3 =$	+ :0 h:h e:e a:a SOS s:s OOB h:h OOB e:e OOB +:0 OOB n:n OOB i:i OOB EOS i:0,
$c_{10} =$	i:z n:n +:0 k:k e:e o:o SOS m:m OOB o:e OOB +:0 OOB n:n OOB i:i OOB EOS i:i,
$c_{11} =$	n:n EOS +:0 OOB e:e OOB h:h OOB s:s OOB a:a OOB h:h OOB i:0 OOB +:0 OOB e:e OOB SOS OOB i:i,
$c_{12} =$	n:n EOS +:0 OOB a:e OOB y:y OOB a:a OOB h:h OOB k:k OOB i:0 OOB +:0 OOB e:e OOB SOS OOB i:i,
$c_{13} =$	n:n EOS +:0 OOB a:e OOB g:g OOB n:n OOB a:a OOB l:l OOB i:0 OOB +:0 OOB e:e OOB SOS OOB i:i,
$c_{14} =$	n:n EOS +:0 OOB o:e OOB m:m OOB o:o OOB k:k OOB n:n OOB i:i OOB i:z OOB +:0 OOB e:e OOB SOS OOB i:i,
$c_{16} =$	+ :0 i:i e:e n:n SOS k:k OOB o:o OOB m:m OOB o:e OOB +:0 OOB n:n OOB i:i OOB EOS i:z

$$\text{Thus } C_{L(s):-S(s)}^{full} = C_{L(s)}^{full} - C_s^{full} = C_{i:-0}^{full} = C_i^{full} - C_{i:0}^{full} =$$

$$\{c_1, c_2, c_3, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{16}\} - \{c_1, c_2, c_3\} = \{c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{16}\},$$

with

$c_{10} =$	i:z n:n +:0 k:k e:e o:o SOS m:m OOB o:e OOB +:0 OOB n:n OOB i:i OOB EOS i:i,
$c_{11} =$	n:n EOS +:0 OOB e:e OOB h:h OOB s:s OOB a:a OOB h:h OOB i:0 OOB +:0 OOB e:e OOB SOS OOB i:i,
$c_{12} =$	n:n EOS +:0 OOB a:e OOB y:y OOB a:a OOB h:h OOB k:k OOB i:0 OOB

continued on next page

<i>continued from previous page</i>	
	+:0 OOB e:e OOB SOS OOB i:i,
$c_{13} =$	n:n EOS +:0 OOB a:e OOB g:g OOB n:n OOB a:a OOB l:l OOB i:0 OOB +:0 OOB e:e OOB SOS OOB i:i,
$c_{14} =$	n:n EOS +:0 OOB o:e OOB m:m OOB o:o OOB k:k OOB n:n OOB i:i OOB i:z OOB +:0 OOB e:e OOB SOS OOB i:i,
$c_{16} =$	+:0 i:i e:e n:n SOS k:k OOB o:o OOB m:m OOB o:e OOB +:0 OOB n:n OOB i:i OOB EOS i:z

As a first attempt at an L-relative discerning prefix partitioner of $i:0$ we construct, by inspection, the discerning prefix partitioner $\mathcal{H}_{i:0}^1 =$

$H(C_{i:0}^{full}, C_{i:-0}^{full}) = \{ "+:0" \}$. However, $\mathcal{H}_{i:0}^1 = \{ "+:0" \}$ contains a prefix of $c_{16} \in C_{i:-0}^{full}$. Thus, from Definition 5, $\mathcal{H}_{i:0}^1$ cannot be an L-relative discerning prefix partitioner of $i:0$.

Thus, question 1 is *true* and question 2 is *false* for the rule

[55]

$$i:0 \quad \text{op} \quad +:0 \quad _$$

with the special pair $i:0$ as the correspondence part and $+":0 _$ as the shortened context in \mathcal{W} . We therefore select the \Rightarrow rule type (Table 3.1):

[56]

$$i:0 \Rightarrow +:0 _$$

This is also the minimal environment for the \Rightarrow rule type and special pair $i:0$ relative to $C_{i:0}^{full}$ and $C_{i:i}^{full}$, since $\mathcal{H}_{i:0}^1$ is the minimal discerning prefix partitioner of $C_{i:0}^{full}$.

Now we have established a \Rightarrow rule with a minimal context by finding a discerning prefix partitioner for which question 1 answers *true*. Next we will

try to establish a \Leftarrow rule by finding an L-relative discerning prefix partitioner for which question 2 answers *true*.

The minimal discerning prefix partitioner $\mathcal{H}_{i:0}^1 = \{ "+:0" \}$, is not the only partitioner which allows a *true* answer for question 1. Another discerning prefix partitioner is $\mathcal{H}_{i:0}^2 = \{ "+:0 l:l", "+:0 k:k", "+:0 h:h" \}$. The superscript "2" in $\mathcal{H}_{i:0}^2$ is used to show this is the second discerning prefix partitioner which we have selected for $\mathcal{C}_{i:0}^{full}$. $\mathcal{H}_{i:0}^2$ is a discerning prefix partitioner, since it contains a prefix for each element of $\mathcal{C}_{i:0}^{full}$, i.e. for $c_1, c_2, c_3 \in \mathcal{C}_{i:0}^{full}$ and no prefix of $c_{10}, c_{11}, c_{12}, c_{13}, c_{14} \in \mathcal{C}_{i:i}^{full}$. In addition, $\mathcal{H}_{i:0}^2$ is an L-relative discerning prefix partitioner since $\mathcal{H}_{i:0}^2$ does not contain a prefix of any element of $\mathcal{C}_{i:-0}^{full}$ (see Definition 5). Furthermore, $\mathcal{H}_{i:0}^2 = \mathcal{H}_{i:0}^{minLrel}$ because none of the three prefixes can be made shorter, while still keeping $\mathcal{H}_{i:0}^2$ an L-relative discerning prefix partitioner.

Thus, question 2 is *true* with $\mathcal{H}_{i:0}^{Lrel} = \mathcal{H}_{i:0}^2$ and the rule context is $E_{\mathcal{H}_{i:0}^{Lrel}} = "+:0 - l:l \mid +:0 - k:k \mid +:0 - h:h"$. Therefore, for the special pair "*i:0*" and the environment $E_{\mathcal{H}_{i:0}^{Lrel}}$ we can write the following \Leftarrow rule (cf. Table 3.1):

[57]

$$i:0 \Leftarrow +:0 - l:l \mid +:0 - k:k \mid +:0 - h:h$$

This is also the minimal environment for the \Leftarrow rule type and the special pair *i:0* relative to $\mathcal{C}_{i:0}^{full}$ and $\mathcal{C}_{i:-0}^{full}$, since $\mathcal{H}_{i:0}^2$ is the minimal L-relative discerning prefix partitioner of *i:0*. Furthermore, since $\mathcal{H}_{i:0}^{Lrel}$ is also a discerning prefix partitioner from Definition 5, question 1 is true as well. Thus we can write another \Rightarrow rule:

[58]

$$i:0 \Rightarrow +:0 - l:l \mid +:0 - k:k \mid +:0 - h:h$$

The context of this \Rightarrow rule is however not minimal. The minimal context was already established in Example 56.

The minimal context for question 1 (i.e. for a \Rightarrow rule) is the context constructed from the minimal discerning prefix partitioner and the minimal context for question 2 (i.e. for a \Leftarrow rule) is the context constructed from the minimal L-relative discerning prefix partitioner.

In this section I have shown how the two questions' conditions can be met with functions on shortened mixed-context sets. To this end I defined the notions of a minimal discerning prefix partitioner and a minimal L-relative discerning prefix partitioner. This rephrasal of the two questions in terms of functions on sets is another step closer to a procedural answer to the two questions. However, it does not yet have sufficiently explicit steps to be implemented as a computer program. Thus we need to rephrase the two questions again. This (third and last) rephrasal is in the next section. It will build on the terms defined in this section, to produce explicit procedural steps to answer the two questions automatically.

4.4 Two Questions in terms of shortened paths in a DAG

We will now show how the two questions can be answered in terms of shortened paths in a *directed acyclic graph* (DAG). This will give us the explicit procedural steps to answer the two questions automatically (i.e. with an

implemented computer program). For completeness, we first define (Section 4.4.1) an *acyclic deterministic finite state automaton* (ADFSA) and its *underlying graph*. Then we show how an ADFSA is constructed for a set of mixed-context sequences. Since the automaton is deterministic, it merges common prefixes which provides a compact representation of the mixed-context sequences. When this automaton is then viewed as a DAG, it provides a way to compare all the mixed-context sequences with each other at the same time. This allows us to extract the minimal discerning contexts for the \Rightarrow and \Leftarrow rules.

In Section 4.4.2 we define an *edge-delimiter set* for a given special pair, relative to the ADFSA, and show how a discerning prefix partitioner can be derived from it. The edge-delimiter set is used to extract shortened \Rightarrow rule contexts from the ADFSA, viewed as a DAG. Finally, we define an *L-relative edge-delimiter set* for a given special pair, relative to the ADFSA, and show how an L-relative discerning prefix partitioner can be derived from it. The L-relative edge-delimiter set is used to extract shortened \Leftarrow rule contexts from the ADFSA, viewed as a DAG.

4.4.1 Preliminary Definitions

Based on the definition in (Revuz, 1992, p.182), we define an *acyclic deterministic finite state automaton* (ADFSA):

Definition 10 An acyclic deterministic finite state automaton (ADFSA) \mathcal{A} is a 5-tuple: $\mathcal{A} = (\mathcal{Q}, \mathcal{I}, R, \mathcal{Q}^T, q_0)$, where

- \mathcal{Q} is the finite set of states;
- \mathcal{I} is an finite alphabet;

- q^0 is the initial state;
- Q^T is the subset of terminal states of Q ;
- R is a function in $Q \times \mathcal{I}$ defining the transitions (arcs) of the automaton.
- The automaton is deterministic if, for every state q , every transition out of q has a different label $v \in \mathcal{I}$.
- An automaton is acyclic if the underlying graph is acyclic (compare Definition 11). A string or sequence w is accepted or recognized by the automaton if $q_0.w \in Q^T$.

Let the state reached by the transition labeled with $v \in \mathcal{I}$, from state q , be denoted by $q.v = R(q, v)$. The notation is transitive: If $w \subset \mathcal{I}^+$ and $v^i \in \mathcal{I}$ is the i 'th symbol in w , then $q.w$ denotes the state reached by following the transitions labelled by each successive alphabet symbol v^1, v^2, \dots, v^n in w . Acyclic automata recognize finite sets of finite words. We define the *path* p_w of a word w in \mathcal{A} , as the sequence of states followed between the transitions to accept w , i.e. $p_w = q_0, q_1, \dots, q_t$ where $q_t \in Q^T$. Since the automaton is deterministic and acyclic, there is only one path for every input word w .

We define the *underlying labeled directed graph* of an automaton \mathcal{A} :

Definition 11 An underlying labeled directed graph $\mathcal{G} = \mathcal{G}(\mathcal{A})$ of an automaton \mathcal{A} is a 7-tuple: $\mathcal{G} = (\mathcal{N}, \mathcal{I}, R, G^Q, \mathcal{E}, \mathcal{N}^T, n_0)$, where

- $G^Q(\mathcal{A})$ is a function doing a one-to-one mapping of the states of \mathcal{A} to nodes of \mathcal{G} ; $G^{q_i}(\mathcal{A})$ maps the state q_i to a node n_i .
- \mathcal{N} is the set of nodes which is equal to $G^Q(\mathcal{A})$,

- R is a function in $Q \times \mathcal{I}$ defining the transitions (arcs) of the automaton (see Definition 10) associated with the graph.
- \mathcal{E} is the set of directed edges connecting all the nodes in \mathcal{N} ; $\text{label}(e)$ is the label of edge $e \in \mathcal{E}$. A node $n_i = G^{q_i}(\mathcal{A})$ is connected to another node $n_j = G^{q_j}(\mathcal{A})$ with an edge e , if and only if there exists an $a \in \mathcal{I}$ such that $q_j = R(q_i, a)$. Then, $\text{label}(e) = a$.
- \mathcal{I} is the finite set of labels of the edges \mathcal{E} , which is equal to the set of input symbols of \mathcal{A} ;
- $\mathcal{N}^T = G^{\mathcal{Q}^T}(\mathcal{A})$ is the subset of nodes having no outgoing edges. We call these nodes terminal nodes and we call the edges which ends in them terminal edges.
- $n_0 = G^{q_0}(\mathcal{A})$ is the root node, which corresponds to the initial state of \mathcal{A} ;

A path p in \mathcal{G} is a sequence of zero or more edges connected by nodes in \mathcal{G} . Note that this definition of a path in terms of edges differs from the usual definition of a path in terms of nodes connected with edges. Let $\mathcal{P} = \{p_1 \dots p_{\|\mathcal{P}\|}\}$ be the set of all paths in \mathcal{G} . $|p|$ denotes the length of path p , i.e. the number of edges in p . A path $p_i \in \mathcal{P}$ can be written as $p_i = e_i^1 e_i^2 \dots e_i^{|p_i|}$, where e_i^j is the j 'th edge in path p_i . $\text{string}(p_i^k)$ denotes the string of concatenated labels of $e_i^1 \dots e_i^k$.

The directed graph \mathcal{G} is *acyclic* if, for every path $p_i \in \mathcal{P}$, $e_i^j \neq e_i^k$, where $1 \leq j, k \leq |p_i|$ and $j \neq k$. Such a graph is called a *directed acyclic graph* (DAG).

We define the notion of *reachability* as follows:

Definition 12 An edge e^n is reachable from an edge e^j in a DAG G , if there is a path segment from e^j to e^n , i.e. there exists a path segment e^j, e^{j+1}, \dots, e^n in the DAG.

We define this notion of reachability also for sets of edges:

Definition 13 A set of edges \mathcal{E}_2 is reachable from a set of edges \mathcal{E}_1 in a DAG G , if for every edge $e^n \in \mathcal{E}_2$ there exists an edge $e^j \in \mathcal{E}_1$ such that e^n is reachable from e^j .

We use $\mathcal{M} = M(\mathcal{C}^{full})$ to denote the acyclic deterministic finite state automaton constructed from the mixed-context set \mathcal{C}^{full} , i.e. \mathcal{M} recognizes all and only the strings in \mathcal{C}^{full} . The input alphabet is \mathcal{U} , the union of the feasible pairs and the special symbols in the mixed-context sequences (Section 4.2, page 47). Recall that the feasible pairs are considered to be atomic symbols.

I will now show how the full mixed-context set of the Section 4.3 is mapped onto a directed acyclic graph \mathcal{G} . This is done since the DAG \mathcal{G} will be traversed procedurally to extract the minimal contexts which will provide *true* answers to the two rule-type decision questions (see page 77).

Let $\mathcal{G} = G(M(\mathcal{C}^{full}))$ be the underlying labeled DAG of \mathcal{M} . Let p_{c_i} denote the path (i.e. sequence of edges) in \mathcal{G} which must be followed to recover all and only the symbols in the sequence $c_i \in \mathcal{C}^{full}$, i.e. $string(p_{c_i}) = c_i$. Since, $M(\mathcal{C}^{full})$ is deterministic and acyclic, there is a one-to-one mapping of mixed contexts to paths in \mathcal{G} , i.e. for every mixed context $c_i \in \mathcal{C}^{full}$ there is only one path $p_{c_i} \in \mathcal{P}$ such that $string(p_{c_i}) = c_i$, and *vice versa*. To indicate the mixed-context sequence recovered by following a path p_i , we use the notation c_{p_i} . Thus, $c_{p_i} = string(p_{c_i})$. Furthermore, if $c_i = v_i^1, v_i^2, \dots, v_i^{|c_i|}$, with $v_i^j \in \mathcal{U}$, and path $p_{c_i} = e_{c_i}^1 \dots e_{c_i}^{|c_i|}$, with $e_{c_i}^j \in \mathcal{E}$, then $label(e_{c_i}^j) = v_i^j$

for $1 \leq r \leq |c_i|$. Recall that $v_i^{|c_i|}$ is the marker pair of c_i . From Example 48, consider $c_8 = v_8^1, v_8^2, \dots, v_8^{17} =$

g:g +:0 n:n n:n a:a i:i l:l EOS i:0 OOB +:0 OOB e:e OOB SOS OOB

a:e.

Then we have the path $p_{c_8} = e_{c_8}^1 \dots e_{c_8}^{17}$ with

$label(e_{c_8}^1) = g:g,$
$label(e_{c_8}^2) = +:0,$
$label(e_{c_8}^3) = n:n,$
$label(e_{c_8}^4) = n:n,$
$label(e_{c_8}^5) = a:a,$
$label(e_{c_8}^6) = i:i,$
$label(e_{c_8}^7) = l:l,$
$label(e_{c_8}^8) = EOS,$
$label(e_{c_8}^9) = i:0,$
$label(e_{c_8}^{10}) = OOB,$
$label(e_{c_8}^{11}) = +:0,$
$label(e_{c_8}^{12}) = OOB,$
$label(e_{c_8}^{13}) = e:e,$
$label(e_{c_8}^{14}) = OOB,$
$label(e_{c_8}^{15}) = SOS,$
$label(e_{c_8}^{16}) = OOB,$
$label(e_{c_8}^{17}) = \mathbf{a:e}.$

The marker pair is $label(e_{c_8}^{17}) = v_8^{17} = \mathbf{a:e}$.

Let c_i^k , with $c_i \in \mathcal{C}^{full}$, denote the prefix of c_i up to the k 'th symbol, i.e. $c_i^k = v_i^1, v_i^2, \dots, v_i^k$, where $v_i^j \in \mathcal{U}$. Similarly, let p_i^k , denote the path prefix of path p_i , in the DAG $\mathcal{G} = G(M(\mathcal{C}^{full}))$, up to the k 'th edge in the full

path, i.e. $p_i^k = e_i^1, e_i^2, \dots, e_i^k$, where $e_i^j \in \mathcal{E}$. We define the *equality of two path prefixes* as follows:

Definition 14 *Prefix p_i^k of path p_i and prefix p_j^k of path p_j are equal (i.e. $p_i^k = p_j^k$) if and only if $e_i^r = e_j^r$, where $1 \leq r \leq k$.*

For instance, from Example 48 we see that $v_{c_1}^1 = v_{c_2}^1 = v_{c_3}^1 = v_{c_{16}}^1 = "+:0"$. Thus the paths p_{c_1} , p_{c_2} , p_{c_3} and $p_{c_{16}}$ share the same first edge, i.e. $e_{c_1}^1 = e_{c_2}^1 = e_{c_3}^1 = e_{c_{16}}^1$, since $\text{label}(e_{c_1}^1) = \text{label}(e_{c_2}^1) = \text{label}(e_{c_3}^1) = \text{label}(e_{c_{16}}^1) = "+:0"$ and since \mathcal{M} is deterministic. The reasons why these four edge identifications identify the *same* edge, is that they all have the same label ($+:0$) and they must all originate from the root node, since all four denote the first edge of a path. Furthermore, since \mathcal{M} is deterministic, there can be only one outgoing edge labeled $+:0$, from the root node. It follows that the four path prefixes of length one are equal, i.e. $p_{c_1}^1 = p_{c_2}^1 = p_{c_3}^1 = p_{c_{16}}^1$. In this example the common prefix has length one, but longer common prefixes are possible.

The common prefixes in an acyclic deterministic finite state automaton are always merged, since the automaton is deterministic — i.e. no two outgoing transitions from the same state can have the same label.

A DAG $\mathcal{G} = G(M(C^{full}(\mathcal{W})))$, which is the underlying graph of an acyclic deterministic finite state automaton $\mathcal{M} = M(C^{full}(\mathcal{W}))$, is sometimes called a *prefix-merged* directed acyclic graph when we want to focus attention on the fact that the prefixes are merged. An example of a prefix-merged DAG is given in Figure 4.1 on page 79.

4.4.2 Back to the Two Questions

To answer the two questions, in terms of a DAG, for *shortened* contexts we first construct the prefix-merged DAG $\mathcal{G} = G(M(C_{L(s)}^{full}(\mathcal{W})))$. We define below the notion of an *edge-delimiter set* and show its connection with a discerning prefix partitioner. This will allow us to rephrase the two questions in terms of edges in the underlying graph \mathcal{G} of the ADFSFA \mathcal{M} which accepts the mixed-context strings in $C_{L(s)}^{full}(\mathcal{W})$.

Let $\mathcal{P}_f = P_f(\mathcal{G})$ be the set of paths which have f as the label of the last edge. This can be rephrased more formally as follows: Let $p_{f,j}$ be the j 'th path in \mathcal{P}_f , then $label(e_j^{|p_{f,j}|}) = f$. Remember that the label of the last edge in the path p_j corresponds to a marker pair in the mixed-context sequence c_{p_j} , i.e. $label(e_j^{|p_j|}) = v_{c_{p_j}}$.

We now define an *edge-delimiter set* of a special pair s as follows:

Definition 15 Let $\mathcal{D}_s = D_s(G(M(C_{L(s)}^{full}(\mathcal{W})))) \subseteq \mathcal{E}$ be a set of edges in the DAG $\mathcal{G} = G(M(C_{L(s)}^{full}(\mathcal{W})))$ such that each path from the root to the terminal edges labeled with the special pair s , go through exactly one edge in \mathcal{D}_s .

1. Thus each edge in the set of terminal edges labeled with the marker pair s is reachable from exactly one edge e in the edge-delimiter set \mathcal{D}_s .
2. Furthermore, no terminal edge labeled with a default pair which has the same L -component as s , must be reachable from any edge e in the edge-delimiter set \mathcal{D}_s .

We call \mathcal{D}_s an *edge-delimiter set* of the terminal edges in \mathcal{G} which are labeled with the special pair s .

Recall that since the special pair s in \mathcal{D}_s is the label of a terminal edge in \mathcal{G} , it must be a marker pair of at least one mixed-context string which is recognized by the automaton $\mathcal{M} = M(C_{L(s)}^{full}(\mathcal{W}))$. From Definition 15 it follows that no two edges in \mathcal{D}_s are on the same path $p \in \mathcal{P}_s$. Two paths in \mathcal{P}_s could, however, share an edge $e \in \mathcal{D}_s$ if those two paths have a prefix in common and the edge e is part of that common prefix. Thus, due to the possible sharing of edges between paths in \mathcal{G} , it follows that $\|\mathcal{D}_s\| \leq \|\mathcal{P}_s\|$. There can be more than one edge-delimiter set for a given special pair s . We define the *minimal edge-delimiter set* of the special pair s as follows:

Definition 16 Let $\mathcal{D}_s^{min} = D_s^{min}(G(M(C_{L(s)}^{full}(\mathcal{W})))$ be the edge-delimiter set of f such that no other edge-delimiter set \mathcal{D}_s contains an edge e_j which is closer to the root of $\mathcal{G} = G(M(C_{L(s)}^{full}(\mathcal{W})))$ than an edge $e_i \in \mathcal{D}_s^{min}$. Thus, no element e_j , of any other \mathcal{D}_s , precedes an $e_i \in \mathcal{D}_s^{min}$ on any path $p_{s,r}$. We call \mathcal{D}_s^{min} the minimal edge-delimiter set of the special pair s .

Let \mathcal{P}_s^e be the set of path prefixes $\{p_{s,j}^e\}$ such that $p_{s,j}^e$ is the path prefix up to the edge e , of the j 'th path from the root to a terminal edge labeled with s . Since more than one path can pass through an edge e , we use the second subscript (here j) in $p_{s,j}^e$ to indicate to which path the prefix $p_{s,j}^e$ belongs. We define

Definition 17 the function $\text{pathprefixes}(\mathcal{D}, \mathcal{P})$ with two arguments,

1. an edge-delimiter set \mathcal{D} in a DAG and
2. a set of paths \mathcal{P} in the same DAG,

and it returns the set of (path) prefixes of the paths in \mathcal{P} which are delimited by the edge-delimiters in \mathcal{D} , i.e. $\text{pathprefixes}(\mathcal{D}, \mathcal{P}) = \{p_j^e | e \in \mathcal{D}, p_j \in \mathcal{P}\}$.

To obtain a short-hand notation, we define the set of path prefixes, as delimited by the edges in \mathcal{D}_s , of the paths with terminal edges labeled by s , as follows:

Definition 18 Let $\mathcal{P}_s^{\mathcal{D}_s} = \text{pathprefixes}(\mathcal{D}_s, \mathcal{P}_s)$ denote the set of path prefixes $\{p_{s,j}^e | e \in \mathcal{D}_s, p_{s,j} \in \mathcal{P}_s\}$. We call $\mathcal{P}_s^{\mathcal{D}_s}$ an edge-delimited path prefix set of the marker pair s .

Note that none of these prefixes in $\mathcal{P}_s^{\mathcal{D}_s}$ is the prefix of another element of $\mathcal{P}_s^{\mathcal{D}_s}$, since no two edges in \mathcal{D}_s are on the same path, and since no two edges leaving the root node (or any other node) have the same label (the automaton is deterministic). Thus,

Corollary 2 for every two path prefixes $x, y \in \mathcal{P}_s^{\mathcal{D}_s}$, $\text{string}(x)$ is not a (string) prefix of $\text{string}(y)$ where $x \neq y$.

This corollary is required since we want to rewrite the discerning prefix partitioner of Section 4.3 in terms of path prefixes in the DAG

$$\mathcal{G} = G(M(C_{L(s)}^{\text{full}}(\mathcal{W}))).$$

To this end we furthermore define the function $\text{stringset}(\mathcal{P}^{\text{pref}})$ of a set of path prefixes $\mathcal{P}^{\text{pref}}$ as follows:

Definition 19 $\text{stringset}(\mathcal{P}^{\text{pref}}) = \{\text{string}(x) | x \in \mathcal{P}^{\text{pref}}\}$

From Definition 3, Definition 15 and Corollary 2 it follows that:

Corollary 3 The set $\mathcal{H}_s = \mathcal{H}_{\mathcal{D}_s} = \text{stringset}(\text{pathprefixes}(\mathcal{D}_s, \mathcal{P}_s)) = \text{stringset}(\mathcal{P}_s^{\mathcal{D}_s}) = \{\text{string}(p_{s,j}^e) | e \in \mathcal{D}_s, p_{s,j} \in \mathcal{P}_s\}$ is a discerning prefix partitioner of $\mathcal{C}_s^{\text{full}}$. Thus, $\mathcal{H}_{\mathcal{D}_s} = H(\mathcal{C}_s^{\text{full}})$.

We have now written a discerning prefix partitioner of $\mathcal{C}_s^{\text{full}}$ in terms of an edge-delimiter set \mathcal{D}_s of the special pair s . This is useful since we have

already shown how to write the environment for question 1 to be *true*, once we have the discerning prefix partitioner of the mixed-context set C_s^{full} (Section 4.3, page 61). Now (in Corollary 3) we have a way to compute the discerning prefix partitioner from the edge-delimiter set.

We say that the edges in \mathcal{D}_s *delimit* the prefix-strings in $\mathcal{H}_{\mathcal{D}_s}$. From Definition 16 and Corollary 3 it follows that no $string(p_{s,j}^e)$, with $p_{s,j}^e \in \mathcal{P}_{\mathcal{D}_s}^{pref}$ can be a true prefix of any $string(p_{s,i}^e)$, with $p_{s,i}^e \in \mathcal{P}_{\mathcal{D}_s^{min}}^{pref}$. Thus, the minimal discerning prefix partitioner \mathcal{H}_s^{min} is equal to the set of prefix strings delimited by the edges in the minimal edge-delimiter set for the special pair s , \mathcal{D}_s^{min} . It thus follows that $\mathcal{H}_{\mathcal{D}_s^{min}} = \mathcal{H}_s^{min}$.

We define an *L-relative edge-delimiter set* \mathcal{D}_s^{Lrel} as follows:

Definition 20 An L-relative edge-delimiter set \mathcal{D}_s^{Lrel} of a special pair s is an edge-delimiter set of s such that

1. each terminal edge labeled with the special pair s is reachable from an edge $e \in \mathcal{D}_s^{Lrel}$,
2. but no terminal edge labeled with $L(s):\neg S(s)$ is reachable from any edge $e \in \mathcal{D}_s^{Lrel}$.

We can now also write an L-relative discerning prefix partitioner of C_s^{full} (see Definition 5) in terms of an L-relative edge-delimiter set \mathcal{D}_s^{Lrel} of the special pair s . This is useful since we have already shown how to write the environment for question 2 to be *true*, once we have the L-relative discerning prefix partitioner of the mixed-context set C_s^{full} (Section 4.3, page 61). Furthermore, since the edge-delimiter set \mathcal{D}_s^{Lrel} can be computed from the DAG $\mathcal{G} = G(M(C_{L(s)}^{full}(\mathcal{W})))$, we now have all the procedural steps to automatically determine the environment for which question 2 is *true*.

From Definition 5 and Definition 20, it follows that:

Corollary 4 *The set $\mathcal{H}_s^{Lrel} = \mathcal{H}_{\mathcal{D}_s^{Lrel}} = \text{stringset}(\text{pathprefixes}(\mathcal{D}_s^{Lrel}, \mathcal{P}_s)) = \{\text{string}(p_{s,j}^e) | e \in \mathcal{D}_s^{Lrel}, p_{s,j} \in \mathcal{P}_s\}$ is an L-relative discerning prefix partitioner for the special pair s .*

An L-relative edge-delimiter set is called the *minimal L-relative edge-delimiter set*, $\mathcal{D}_s^{minLrel}$, of C_s^{full} if no element e_j , of any other \mathcal{D}_s^{Lrel} , precedes an $e_i \in \mathcal{D}_s^{minLrel}$ on any path $p_{s,r}$. It follows that no $\text{string}(p_{s,j}^e)$, with

$p_{s,j}^e \in \mathcal{P}_{\mathcal{D}_s^{Lrel}}^{pref}$, can be a true prefix of any $\text{string}(p_{s,i}^e)$, with $p_{s,i}^e \in \mathcal{P}_{\mathcal{D}_s^{minLrel}}^{pref}$. Thus the minimal L-relative discerning prefix partitioner $\mathcal{H}_s^{minLrel}$ is equal to the set of prefix strings delimited by the edges in the minimal L-relative edge-delimiter set for the special pair s , $\mathcal{D}_s^{minLrel}$. It thus follows that $\mathcal{H}_{\mathcal{D}_s^{minLrel}} = \mathcal{H}_s^{minLrel} = \{\text{string}(p_{s,j}^e) | e \in \mathcal{D}_s^{minLrel}, p_{s,j} \in \mathcal{P}_s\}$.

The above definitions allow us to rephrase the two rule-type decision questions in a procedural way which show where the special pairs come from and how to construct an environment for them which results in a *true* answer for either or both of the questions. Now the two questions are rephrased, for each special pair $s \in \mathcal{B}$ in the input string edit sequences \mathcal{W} , in terms of delimiter edges and paths in the ADFSA DAG $\mathcal{G} = G(M(C_{L(s)}^{full}(\mathcal{W})))$:

[59]

Question 1's Condition: Question 1 is *true* for a special pair $s \in \mathcal{B} =$

$B(\mathcal{W})$ and any environment

$E_{\mathcal{D}_s^i} = \text{conc}(\text{stringset}(\text{pathprefixes}(D_s^i(\mathcal{G}), P_s(\mathcal{G}))),$ with
 $\mathcal{G} = G(M(C_{L(s)}^{full}(\mathcal{W}))).$

If $\mathcal{D}_s = \mathcal{D}_s^{min}$ then $E_{\mathcal{D}_s}$ is the *minimal environment* ($E_{\mathcal{D}_s}^{min}$) which specifies all the contexts where s is allowed.

Question 2's Condition: Question 2 is *true* for a special pair $s \in \mathcal{B} =$

$B(\mathcal{W})$ and any environment

$$E_{\mathcal{D}_s^{Lrel,j}} = \text{conc}(\text{stringset}(\text{pathprefixes}(D_s^{Lrel,j}(\mathcal{G}), P_s(\mathcal{G}))), \text{ with } \\ \mathcal{G} = G(M(C_{L(s)}^{full}(\mathcal{W}))).$$

If $\mathcal{D}_s^{Lrel} = \mathcal{D}_s^{minLrel}$ then $E_{\mathcal{D}_s^{Lrel}}$ is a *minimal environment* ($E_{\mathcal{D}_s^{Lrel}}^{min}$) in which $L(s)$ is always realized as $S(s)$.

Now I have rephrased the two questions in enough procedural detail for them to be answered by an automated algorithm. Below I will show, with our example started in Example 46 on page 47, how the environments (contexts) which provide *true* answers for the two questions, are obtained. This is done with the aid of a minimal edge-delimiter set in a DAG for question 1 and a minimal L-relative edge-delimiter set in a DAG for question 2. When question 1 is *true* for the environment extracted from the DAG, then that environment is for a \Rightarrow rule. When question 2 is *true* then that environment is for a \Leftarrow rule.

To answer *question 1* for the special pair $i:0$ in our example (i.e. from Example 48 on page 48), we need to find an edge-delimiter set $\mathcal{D}_{i:0}$. To find an edge-delimiter set $\mathcal{D}_{i:0}$ we inspect the DAG $\mathcal{G} = G(M(C_i^{full}(\mathcal{W})))$ in Figure 4.1. The root node (start state if viewed as an automaton) is labeled 00 and the terminal node (final state) has a double circle. In Figure 4.1 the terminal node is labeled 33 . To prevent a too cluttered figure, the edges of the DAG \mathcal{G} represented in Figure 4.1 are not explicitly numbered. However, each edge can be uniquely addressed by the tuple (*preceding_node_number, edge_label*), since different edges leaving a node always have different labels (the automaton is deterministic). We can thus write a path or path prefix as a sequence of these edge tuples (see the set of paths

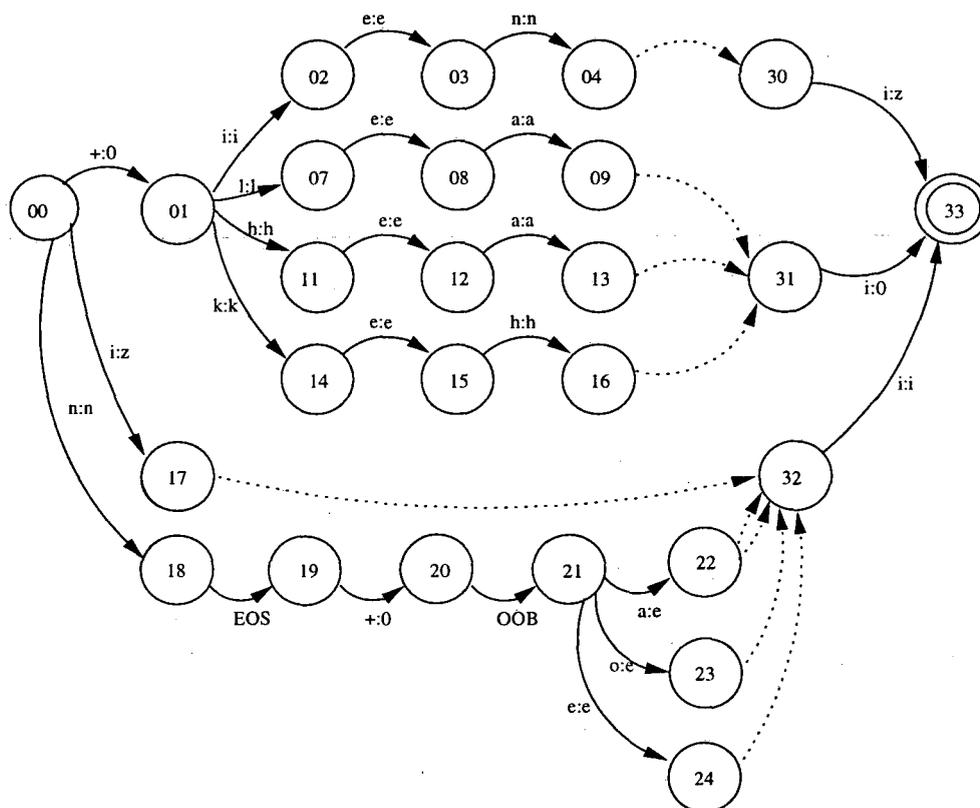


Figure 4.1: $\mathcal{G} = G(M(C_i^{full}(\mathcal{W})))$

$\mathcal{P}_{i:0}$ below). Also note that some of the nodes and edges in Figure 4.1 have been replaced by dotted edges, to enable the DAG to fit on one page. There are three terminal edges labeled with marker pairs: $(30, i:z)$, $(31, i:0)$ and $(32, i:i)$. Only two of these marker pairs are special pairs: $(30, i:z)$ and $(31, i:0)$.

To continue our example, we can now attempt to construct $\mathcal{D}_{i:0}$ by inspection of \mathcal{G} : The set of paths having the special pair “ $i:0$ ” as marker pair, is $\mathcal{P}_{i:0} = \{p_{i:0,1}, p_{i:0,2}, p_{i:0,3}\}$, with

[60]

$$\begin{aligned} p_{i:0,1} &= (00, +:0) (01, l:l) (07, e:e) (08, a:a) \dots (31, i:0), \\ p_{i:0,2} &= (00, +:0) (01, h:h) (11, e:e) (12, a:a) \dots (31, i:0), \\ p_{i:0,3} &= (00, +:0) (01, k:k) (14, e:e) (15, h:h) \dots (31, i:0) \end{aligned}$$

The first edge, $(00, +:0)$, is shared by all the paths $p_{i:0,1}, p_{i:0,2}, p_{i:0,3} \in \mathcal{P}_{i:0}$, and therefore they all go through this edge. Furthermore, no terminal edge labeled with a default pair is reachable from this edge. Thus, $\mathcal{D}_{i:0}^1 = \{ (00, +:0) \}$ is an edge-delimiter set of the marker pair $i:0$. The superscript “1” indicates that this is the first edge-delimiter set of $i:0$ under consideration. To reconstruct the (common) mixed-context prefix up to this edge $(00, +:0)$, we can traverse any of the path prefixes of the three paths $p_{i:0,1}, p_{i:0,2}$ or $p_{i:0,3}$, since the path prefixes delimited by this edge are equal in this case, i.e. $p_{i:0,1}^{(00,+:0)} = p_{i:0,3}^{(00,+:0)} = p_{i:0,3}^{(00,+:0)} = “(00, +:0)”$. Thus $\mathcal{P}_{i:0}^{\mathcal{D}_{i:0}^1} = \{p_{i:0,1}^{(00,+:0)}\} = \{ (00, +:0) \}$ and question 1 is true for the environment

$$\begin{aligned} E_{\mathcal{D}_{i:0}^1} &= \text{conc}(\text{stringset}(\text{pathprefixes}(\mathcal{D}_{i:0}^1, \mathcal{P}_{i:0}))) \\ &= \text{conc}(\text{stringset}(\text{pathprefixes}(\{ (00, +:0) \}, \{p_{i:0,1}, p_{i:0,2}, p_{i:0,3}\}))) \\ &= \text{conc}(\text{stringset}(\{p_{i:0,1}^{(00,+:0)}\})) \\ &= \text{conc}(\{ “+:0” \}) \end{aligned}$$

$$= \text{unmix}("+:0")$$

$$= "+:0 _".$$

To answer *question 2* for the special pair $i:0$, we need to find $\mathcal{D}_{i:0}^{Lrel}$. As a first attempt at an L-relative edge-delimiter set for the special pair $i:0$, let us consider the edge-delimiter set $\mathcal{D}_{i:0} = \{(00, +:0)\}$. We see that the terminal edge labeled with $i:0$ is reachable from the edge $(00, +:0) \in \mathcal{D}_{i:0}$. However, from inspecting the DAG \mathcal{G} , we see that the terminal edge labeled with $i:z$ is also reachable from the edge $(00, +:0)$, via the path $p_{i:z} = "(00,+:0) (01,i:i) (02,e:e) (03,n:n) \dots (30,i:z)"$. This marker pair " $i:z$ " is an element of the feasible pair set $L(i:0):\neg S(i:0) = "i:-0"$. Thus, from Definition 20, $\mathcal{D}_{i:0}$ cannot be an L-relative edge-delimiter set for the special pair $i:0$. Thus, question 1 is *true* and question 2 is *false* for the rule with the special pair $i:0$ as the correspondence part and " $+:0 _$ " as the context in \mathcal{W} . We must select the \Rightarrow rule type (Table 3.1):

[61]

$$i:0 \Rightarrow +:0 _$$

The environment $E_{\mathcal{D}_{i:0}^1} = "+:0 _"$ is also the minimal environment for the \Rightarrow rule type and special pair $i:0$ relative to $C_{i:0}^{full}$, since $\mathcal{D}_{i:0}^1 = \mathcal{D}_{i:0}^{min}$.

Example 61 is the same result as obtained with the analysis of the shortened mixed-context sets in Section 4.3 (see Example 56 on page 64). However, this time we have followed procedural steps which are detailed enough to be implemented as a computer program. This is also the case for the \Leftarrow rule which we determine below with the aid of an L-relative edge-delimiter set.

By inspection of the DAG \mathcal{G} in Figure 4.1, it is fairly straightforward to see that the set of edges $\mathcal{D}_{i:0}^2 = \{(01, h:h), (01, k:k), (01, l:l)\}$ forms an

edge-delimiter set for the special pair $i:0$, which is the label of the terminal edge $(31, i:0)$. The superscript “2” indicates that this is the second edge-delimiter set of $i:0$ under consideration. $\mathcal{D}_{i:0}^2$ is an edge-delimiter set, since each path up to the edge $(31, i:0)$ go through exactly one of these three edges. Note that none of these path prefixes delimited by $\mathcal{D}_{i:0}^2$ is a prefix of the path $p_{i:z} = “(00,+ :0) (01,i:i) (02,e:e) (03,n:n) \dots (30,i:z)”$, since the edge $(01, i:i)$ is not included in the edge-delimiter set $\mathcal{D}_{i:0}^2$. Therefore $\mathcal{D}_{i:0}^{Lrel} = \mathcal{D}_{i:0}^2$ is an L-relative edge-delimiter set of the special pair $i:0$ in \mathcal{G} . We have the set of path prefixes $\mathcal{P}_{i:0}^{\mathcal{D}_{i:0}^2} = \{p_{i:0,1}^{(01,h:h)}, p_{i:0,2}^{(01,k:k)}, p_{i:0,3}^{(01,l:l)}\} =$

$$\{ (00,+ :0) (01, h:h),$$

$$(00,+ :0) (01, k:k),$$

$$(00,+ :0) (01, l:l) \}$$

The environment for which question 2 is *true* is thus computed as follows:

$$\begin{aligned} E_{\mathcal{D}_{i:0}^2} &= \text{conc}(\text{stringset}(\text{pathprefixes}(\mathcal{D}_{i:0}^2, \mathcal{P}_{i:0}))) \\ &= \text{conc}(\text{stringset}(\text{pathprefixes}(\{ (01,h:h), (01,k:k), (01,l:l) \}, \\ &\quad \{p_{i:0,1}, p_{i:0,2}, p_{i:0,3}\}))) \\ &= \text{conc}(\text{stringset}(\{p_{i:0,1}^{(01,h:h)}, p_{i:0,1}^{(01,k:k)}, p_{i:0,1}^{(01,l:l)}\})) \\ &= \text{conc}(\{ “+ :0 h:h”, “+ :0 k:k”, “+ :0 l:l” \}) \\ &= \text{unmix}(\text{“+ :0 h:h”}) \mid \text{unmix}(\text{“+ :0 k:k”}) \mid \text{unmix}(\text{“+ :0 l:l”}) \\ &= + :0 _ h:h \mid + :0 _ k:k \mid + :0 _ l:l \end{aligned}$$

Thus question 2 is *true* for the environment $E = + :0 _ h:h \mid + :0 _ k:k \mid + :0 _ l:l$. We must therefore select the \Leftarrow rule from Table 3.1:

[62]

$$i:0 \Leftarrow + :0 _ l:l \mid + :0 _ k:k \mid + :0 _ h:h$$

This is also the minimal environment for the \Leftarrow rule type and special pair

$i:0$ relative to $\mathcal{C}_{i:0}^{full}$, since $\mathcal{D}_{i:0}^2 = \mathcal{D}_{i:0}^{minLrel}$ and therefore $\mathcal{H}_{\mathcal{D}_{i:0}^2}$ is the minimal L-relative discerning prefix partitioner of $\mathcal{C}_{i:0}^{full}$.

Example 62 is the same result as obtained with the analysis of the shortened mixed-context sets in Section 4.3 (see Example 57 on page 65). However, this time we have followed procedural steps which are detailed enough to be implemented as a computer program.

4.5 Left or Right Contexts

The mixed-context representation has one obvious drawback: If an optimal rule has only a left context, longer than one feasible pair, or only a right context, it cannot be acquired. To solve this problem, two additional ADF-SAs are constructed: One containing only the left context information for all the marker pairs and one containing only the right context information. The same process is then followed as with the mixed contexts.

A right-context sequence is created from a mixed-context sequence by removing all left-context and out-of-bounds (OOB) symbols. To continue our example (from Example 48), consider the mixed-context sequences of

[63]

$$\mathcal{C}_i^{full} = \{ c_1, c_2, c_3, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{16} \}$$

$c_1 =$	$+:0$ l:l e:e a:a SOS n:n OOB g:g OOB a:e OOB $+:0$ OOB n:n OOB i:i OOB EOS 3-3 i:0,
$c_2 =$	$+:0$ k:k e:e h:h SOS a:a OOB y:y OOB a:e OOB $+:0$ OOB n:n OOB i:i OOB EOS 3-0 i:0,
$c_3 =$	$+:0$ h:h e:e a:a SOS s:s OOB h:h OOB e:e OOB $+:0$ OOB n:n OOB i:i OOB EOS 3-2 i:0,

continued on next page

continued from previous page

$c_{10} =$ i:z n:n +:0 k:k e:e o:o SOS m:m OOB o:e OOB +:0 OOB n:n OOB i:i OOB
 EOS i:i,
 $c_{11} =$ n:n EOS +:0 OOB e:e OOB h:h OOB s:s OOB a:a OOB h:h OOB i:0 OOB
 +:0 OOB e:e OOB SOS OOB i:i,
 $c_{12} =$ n:n EOS +:0 OOB a:e OOB y:y OOB a:a OOB h:h OOB k:k OOB i:0 OOB
 +:0 OOB e:e OOB SOS OOB i:i,
 $c_{13} =$ n:n EOS +:0 OOB a:e OOB g:g OOB n:n OOB a:a OOB l:l OOB i:0 OOB
 +:0 OOB e:e OOB SOS OOB i:i,
 $c_{14} =$ n:n EOS +:0 OOB o:e OOB m:m OOB o:o OOB k:k OOB n:n OOB i:i OOB
 i:z OOB +:0 OOB e:e OOB SOS OOB i:i,
 $c_{16} =$ +:0 i:i e:e n:n SOS k:k OOB o:o OOB m:m OOB o:e OOB +:0 OOB n:n OOB
 i:i OOB EOS 3-1 i:z

Note that those mixed-context sequences written for *special pairs*, i.e. c_1 , c_2 , c_3 and c_{16} , are given an identification number (IDNO) preceding the marker pair. For example, the IDNO of c_1 is 3-3 and the IDNO of c_{16} is 3-1. An identification number is used to keep track of the mixed-context, the left-context and the right-context rules which we will write for the same special pair.

We write the following set of right-context sequences for these mixed contexts:

[64]

$$C_i^{right} = \{ r_1, r_2, r_3, r_4, r_5, r_6 \}$$

$r_1 =$ l:l a:a n:n g:g a:e +:0 n:n i:i EOS 3-3 i:0,
 $r_2 =$ k:k h:h a:a y:y a:e +:0 n:n i:i EOS 3-0 i:0,
 $r_3 =$ h:h a:a s:s h:h e:e +:0 n:n i:i EOS 3-2 i:0,
 $r_4 =$ i:i n:n k:k o:o m:m o:e +:0 n:n i:i EOS 3-1 i:z
 $r_5 =$ EOS i:i,
 $r_6 =$ n:n k:k o:o m:m o:e +:0 n:n i:i EOS i:i,

Note that the IDNOs are copied from the mixed contexts of the special pairs. Furthermore, note that the mixed contexts c_{11} , c_{12} , c_{13} and c_{14} all map to the same right context r_5 .

To create a left-context sequence, the right-context and OOB symbols are deleted from the mixed-context sequences. Thus the set of left contexts generated from $c_1, c_2, c_3, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}$ and c_{16} are

[65]

$$C_i^{left} = \{ l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8, l_9 \}$$

$l_1 = +:0 \text{ e:e SOS } 3-0 \text{ i:0,}$ $l_2 = +:0 \text{ e:e SOS } 3-2 \text{ i:0,}$ $l_3 = +:0 \text{ e:e SOS } 3-3 \text{ i:0,}$ $l_4 = i:z +:0 \text{ e:e SOS } i:i,$ $l_5 = n:n +:0 \text{ e:e h:h s:s a:a h:h i:0 +:0 e:e SOS } i:i,$ $l_6 = n:n +:0 \text{ a:e y:y a:a h:h k:k i:0 +:0 e:e SOS } i:i,$ $l_7 = n:n +:0 \text{ a:e g:g n:n a:a l:l i:0 +:0 e:e SOS } i:i,$ $l_8 = n:n +:0 \text{ o:e m:m o:o k:k n:n i:i i:z +:0 e:e SOS } i:i,$ $l_9 = +:0 \text{ e:e SOS } 3-1 \text{ i:z}$
--

Note that l_1, l_2, l_3 and l_9 are the left contexts of the special pairs, with the IDNOs copied from the mixed contexts c_1, c_2, c_3 and c_{16} , respectively.

The right contexts in C_i^{right} are used to construct the DAG $\mathcal{G} = G(M(C_i^{right}))$ in Figure 4.2. Similarly, the left contexts in C_i^{left} are used to construct the DAG $\mathcal{G} = G(M(C_i^{left}))$ in Figure 4.3.

The final set of rules is selected from the output of all three the ADFSAs, i.e. the ADFSAs for the mixed contexts, the right contexts and the left contexts. In Section 4.4.2 (Example 61 on page 81 and Example 62 on page 82) we have seen how the following two rules were extracted from the mixed-context ADFSAs viewed as the DAG in Figure 4.1 (page 79):

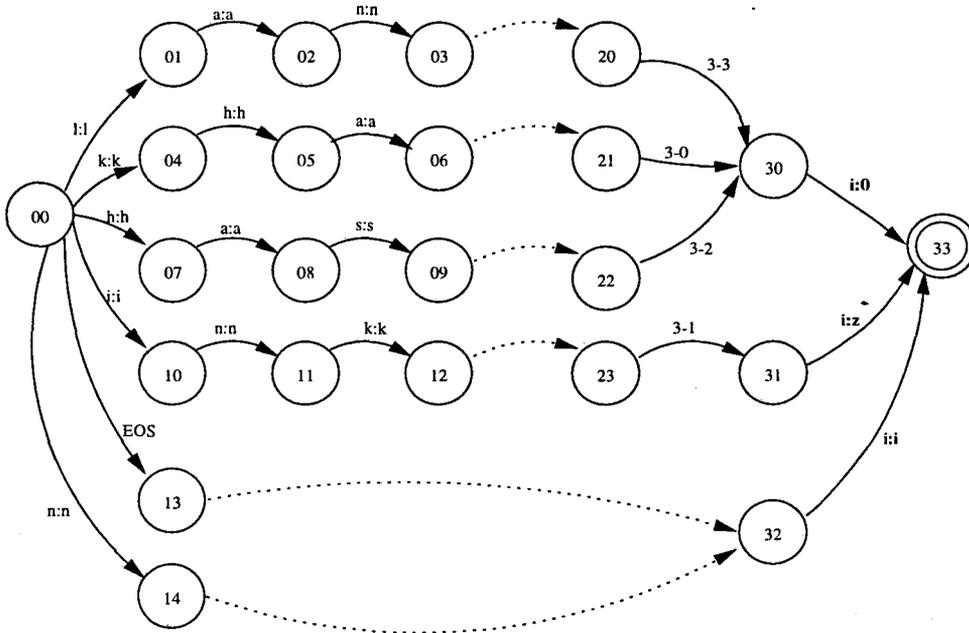


Figure 4.2: $\mathcal{G} = G(M(C_i^{right}))$

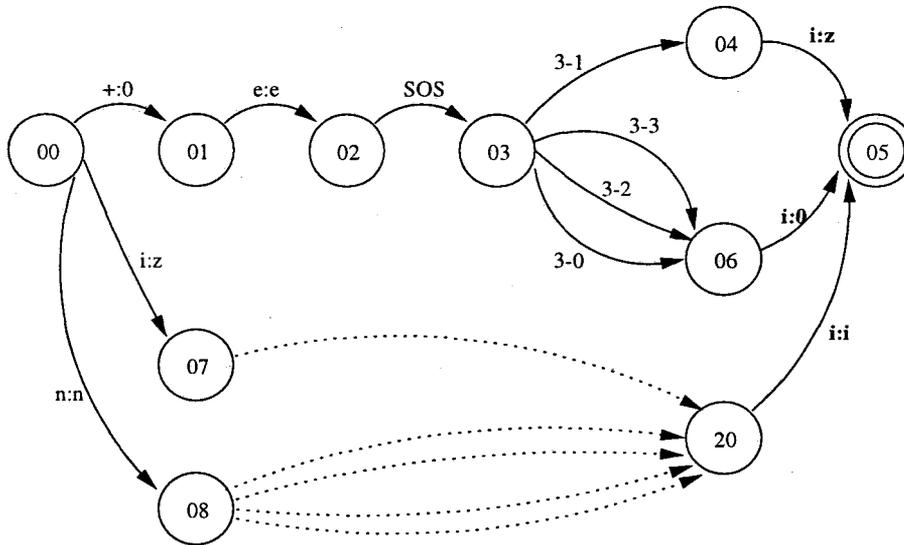


Figure 4.3: $\mathcal{G} = G(M(C_i^{left}))$

[66]

$$\begin{aligned}
 i:0 &\Rightarrow +:0 _ \\
 i:0 &\Leftarrow +:0 _ l:l \mid +:0 _ k:k \mid +:0 _ h:h
 \end{aligned}$$

The second rule above can be written as the following three simple rules (i.e. rules without disjuncted contexts):

[67]

$$\begin{aligned}
 i:0 &\Leftarrow +:0 _ l:l \\
 i:0 &\Leftarrow +:0 _ k:k \\
 i:0 &\Leftarrow +:0 _ h:h
 \end{aligned}$$

From the right-context DAG in Figure 4.2 we can extract the following simple rules:

[68]

$$\begin{aligned}
 i:0 &\Rightarrow _ l:l \\
 i:0 &\Rightarrow _ k:k \\
 i:0 &\Rightarrow _ h:h \\
 i:0 &\Leftarrow _ l:l \\
 i:0 &\Leftarrow _ k:k \\
 i:0 &\Leftarrow _ h:h \\
 \hline
 i:z &\Rightarrow _ i:i \\
 i:z &\Leftarrow _ i:i
 \end{aligned}$$

From the left-context DAG in Figure 4.3 we can extract the following simple rules:

[69]

$$i:0 \Rightarrow +:0 _$$

$$i:z \Rightarrow +:0 _$$

Note that no \Leftarrow rule can be extracted for the special pairs **i:0** and **i:z** in the left-context DAG in Figure 4.3. This is so since the two surface characters *0* and *z* are in free variation relative to the left-context information (compare (Antworth, 1990, p.89)). Two characters are in free variation when they are realized as surface characters from the same lexical character (in this case *i*) in exactly the same context. We can now list all the simple rules extracted from all three the DAGs together and group them according to their context identification numbers:

[70]

3-0

$$i:0 \Rightarrow _ k:k$$

$$i:0 \Rightarrow +:0 _$$

$$i:0 \Leftarrow _ k:k$$

$$i:0 \Leftarrow +:0 _ k:k$$

3-1

$$i:z \Rightarrow _ i:i$$

$$i:z \Rightarrow +:0 _$$

$$i:z \Leftarrow _ i:i$$

$$i:z \Leftarrow +:0 _ i:i$$

continued on next page

continued from previous page

3-2

$$i:0 \Rightarrow _ h:h$$

$$i:0 \Rightarrow +:0 _$$

$$i:0 \Leftarrow _ h:h$$

$$i:0 \Leftarrow +:0 _ h:h$$

3-3

$$i:0 \Rightarrow _ l:l$$

$$i:0 \Rightarrow +:0 _$$

$$i:0 \Leftarrow _ l:l$$

$$i:0 \Leftarrow +:0 _ l:l$$

From this list we select the rules with the contexts which have the lowest *ambiguity count*, which occurs the most as the context of the given special pair and rule type, and which is the shortest.

We start with the IDNO group of the CP that occurs in the least number of IDNO groups. Here, there are only two CPs: $i:0$ (that occurs in IDNO groups 3-0, 3-2 and 3-3) and $i:z$ (that occurs only in IDNO group 3-1). Thus we begin with the IDNO group 3-1 of the CP $i:z$. We select its first \Rightarrow rule, " $i:z \Rightarrow _ i:i$ ", above the " $i:z \Rightarrow +:0 _$ " rule. The reason for this is that the context " $+:0 _$ " also appears in three \Rightarrow rules for the special pair $i:0$ and thus has a higher ambiguity count than the context " $_ i:i$ " which occurs only as the context of the $i:z$ special pair. Thus the ambiguity count of a context for a specific rule type and special pair is counted as the number of other special pairs for which it also appears as the context of the same rule

type. For example, the ambiguity count for “ $_ i:i$ ” in “ $i:z \Rightarrow _ i:i$ ” is zero since it does not occur as the context of another special pair. Furthermore, the ambiguity count for “ $+:0 _$ ” in “ $i:z \Rightarrow +:0 _$ ” is three, since it also occurs in three rules with $i:0$ as CP: “ $i:0 \Rightarrow +:0 _$ ” in IDNO groups 3-0, 3-2 and 3-3.

We select the first \Leftarrow rule, “ $i:z \Leftarrow _ i:i$ ”, from the group with IDNO 3-1, since its context is shorter than the context of the second \Leftarrow rule.

Now we have selected a \Rightarrow and a \Leftarrow rule for the CP $i:z$ from IDNO group 3-1. Next we must select \Rightarrow and \Leftarrow rules for the CP $i:0$.

We start with the IDNO group 3-0: Both the “ $i:0 \Rightarrow _ k:k$ ” and the “ $i:0 \Rightarrow +:0 _$ ” rules have the same ambiguity count (0)². However, the “ $+:0 _$ ” context appears three times as the context of a $i:0 \Rightarrow$ rule (once each in IDNO groups 3-0, 3-2 and 3-3), while the “ $_ k:k$ ” context appears only once in a $i:0 \Rightarrow$ rule (in IDNO group 3-1). Thus we select the

“ $i:0 \Rightarrow +:0 _$ ” rule.

We follow this selection procedure for all the IDNO groups and in this way select the final simple rules:

²Note that the “ $+:0 _$ ” context of the “ $i:0 \Rightarrow +:0 _$ ” rule initially had an ambiguity count of one, since the “ $i:z \Rightarrow +:0 _$ ” rule was not yet eliminated as a possible candidate for the $i:z \Rightarrow$ rule.

3-0

 $i:0 \Rightarrow +:0 _$ $i:0 \Leftarrow _ k:k$

3-1

 $i:z \Rightarrow _ i:i$ $i:z \Leftarrow _ i:i$

3-2

 $i:0 \Rightarrow +:0 _$ $i:0 \Leftarrow _ h:h$

3-3

 $i:0 \Rightarrow +:0 _$ $i:0 \Leftarrow _ l:l$

We can then merge the \Rightarrow rules into a single \Rightarrow rule for each special pair, which gives us the final merged rule set for special pairs that have i as the lexical component:

[72]

 $i:0 \leftarrow _ k:k$ $i:0 \leftarrow _ l:l$ $i:0 \leftarrow _ h:h$ $i:0 \Rightarrow _ +:0 _$ $i:z \Rightarrow _ i:i$ $i:z \leftarrow _ i:i$

The rule set learned is *complete* since all possible combinations of marker pairs, rule types and contexts are considered by traversing all three DAGs. Furthermore, the rules in the set have the *shortest* possible contexts, since, for a given DAG, there is only one delimiter edge closest to the root for each path, marker pair and rule type combination.

4.6 Insertion Rules

Insertion rules (or epenthesis rules) are handled somewhat differently from the other rules, i.e. deletion and replacement. Different handling is necessitated since the correspondence part of an insertion rule has the null character on the lexical level. We need to obtain a discerning context for an insertion rule, relative to all the contexts of all the possible insertion rules.

For example, for the insertion correspondence $0:i$ we need its discerning context relative to the contexts of the correspondences $0:\neg i$. From a theoretical point of view, the correspondence $0:0$, i.e. the mapping of the null character to itself, is an element of the correspondences $0:\neg i$. The correspondence $0:0$ can appear between any two feasible pairs, of which none

is an insert correspondence. Thus we need to compare the mixed-context representation for $0:i$ with all the potential mixed contexts generated for the correspondences $0:\neg i$ which include the theoretical $0:0$ correspondence. For example, for the morphotactic formulas

[73]

Target	Prefix	+	Source	+	Suffix	
<i>endlini</i>	=	<i>e</i>	+	<i>indlu</i>	+	<i>ni</i>
<i>endlwini</i>	=	<i>e</i>	+	<i>indlu</i>	+	<i>ni</i>

we compute the final string-edit sequences $\mathcal{W} = \{w_1, w_2\}$, where

$$w_1 = e:e +:0 i:0 n:n d:d l:l u:w +:0 \mathbf{0:i} n:n i:i, \text{ and}$$

$$w_2 = e:e +:0 i:0 n:n d:d l:l u:i +:0 n:n i:i.$$

Note that in the sequence w_1 , $\mathbf{0:i}$ indicates the insertion of an i . The following mixed-context sequence set is computed for this insertion of the i :

$$C_{L(0:i)}^{full}(\mathcal{W}) = C_0^{full}(\mathcal{W}) = \{c_1, c_2, \dots, c_{22}\} \text{ where}$$

$$c_1 = +:0 n:n u:w i:i l:l \text{ EOS } d:d \text{ OOB } n:n \text{ OOB } i:0 \text{ OOB } +:0 \text{ OOB } e:e \text{ OOB } \text{ BOS } \text{ OOB } 9-1 \mathbf{0:i},$$

$$c_2 = d:d l:l n:n u:i i:0 +:0 +:0 n:n e:e i:i \text{ BOS } \text{ EOS } - \mathbf{0:0},$$

$$c_3 = d:d l:l n:n u:w i:0 +:0 +:0 \mathbf{0:i} e:e n:n \text{ BOS } i:i \text{ OOB } \text{ EOS } - \mathbf{0:0},$$

$$c_4 = n:n d:d i:0 l:l +:0 u:i e:e +:0 \text{ BOS } n:n \text{ OOB } i:i \text{ OOB } \text{ EOS } - \mathbf{0:0},$$

$$c_5 = i:0 n:n +:0 d:d e:e l:l \text{ BOS } u:i \text{ OOB } +:0 \text{ OOB } n:n \text{ OOB } i:i \text{ OOB } \text{ EOS } - \mathbf{0:0},$$

$$c_6 = +:0 i:0 e:e n:n \text{ BOS } d:d \text{ OOB } l:l \text{ OOB } u:i \text{ OOB } +:0 \text{ OOB } n:n \text{ OOB } i:i \text{ OOB } \text{ EOS } - \mathbf{0:0},$$

$$c_7 = e:e +:0 \text{ BOS } i:0 \text{ OOB } n:n \text{ OOB } d:d \text{ OOB } l:l \text{ OOB } u:i \text{ OOB } +:0 \text{ OOB } n:n \text{ OOB } i:i \text{ OOB } \text{ EOS } - \mathbf{0:0},$$

continued on next page

<i>continued from previous page</i>	
$c_8 =$	BOS e:e OOB +:0 OOB i:0 OOB n:n OOB d:d OOB l:l OOB u:i OOB +:0 OOB n:n OOB i:i OOB EOS - 0:0,
$c_9 =$	n:n d:d i:0 l:l +:0 u:w e:e +:0 BOS 0:i OOB n:n OOB i:i OOB EOS - 0:0,
$c_{10} =$	i:0 n:n +:0 d:d e:e l:l BOS u:w OOB +:0 OOB 0:i OOB n:n OOB i:i OOB EOS - 0:0,
$c_{11} =$	+:0 i:0 e:e n:n BOS d:d OOB l:l OOB u:w OOB +:0 OOB 0:i OOB n:n OOB i:i OOB EOS - 0:0,
$c_{12} =$	e:e +:0 BOS i:0 OOB n:n OOB d:d OOB l:l OOB u:w OOB +:0 OOB 0:i OOB n:n OOB i:i OOB EOS - 0:0,
$c_{13} =$	BOS e:e OOB +:0 OOB i:0 OOB n:n OOB d:d OOB l:l OOB u:w OOB +:0 OOB 0:i OOB n:n OOB i:i OOB EOS - 0:0,
$c_{14} =$	l:l u:w d:d +:0 n:n 0:i i:0 n:n +:0 i:i e:e EOS BOS OOB - 0:0,
$c_{15} =$	l:l u:i d:d +:0 n:n n:n i:0 i:i +:0 EOS e:e OOB BOS OOB - 0:0,
$c_{16} =$	u:w +:0 l:l 0:i d:d n:n n:n i:i i:0 EOS +:0 OOB e:e OOB BOS OOB - 0:0,
$c_{17} =$	u:i +:0 l:l n:n d:d i:i n:n EOS i:0 OOB +:0 OOB e:e OOB BOS OOB - 0:0,
$c_{18} =$	+:0 n:n u:i i:i l:l EOS d:d OOB n:n OOB i:0 OOB +:0 OOB e:e OOB BOS OOB - 0:0,
$c_{19} =$	n:n i:i +:0 EOS u:i OOB l:l OOB d:d OOB n:n OOB i:0 OOB +:0 OOB e:e OOB BOS OOB - 0:0,
$c_{20} =$	i:i EOS n:n OOB +:0 OOB u:i OOB l:l OOB d:d OOB n:n OOB i:0 OOB +:0 OOB e:e OOB BOS OOB - 0:0,
$c_{21} =$	n:n i:i 0:i EOS +:0 OOB u:w OOB l:l OOB d:d OOB n:n OOB i:0 OOB +:0 OOB e:e OOB BOS OOB - 0:0,
$c_{22} =$	i:i EOS n:n OOB 0:i OOB +:0 OOB u:w OOB l:l OOB d:d OOB n:n OOB i:0 OOB +:0 OOB e:e OOB BOS OOB - 0:0

Note that a mixed context is generated for each $0:0$ occurring between each feasible pair in \mathcal{W} , which is not an insert pair. These mixed contexts are then read into an ADFSA which accepts all and only these mixed context sequences. This ADFSA is then viewed as a DAG. This prefix-merged DAG concerning the marker pair $0:i$, is presented in Figure 4.4. Note that the

graph includes only explicit paths for c_1 , c_6 , c_{11} and c_{18} . The dotted arcs indicate the shortening of these paths to make the graph less cluttered. The paths for the eighteen other mixed contexts are collapsed into a single path indicated by a dashed arc. The following two rules can be extracted from

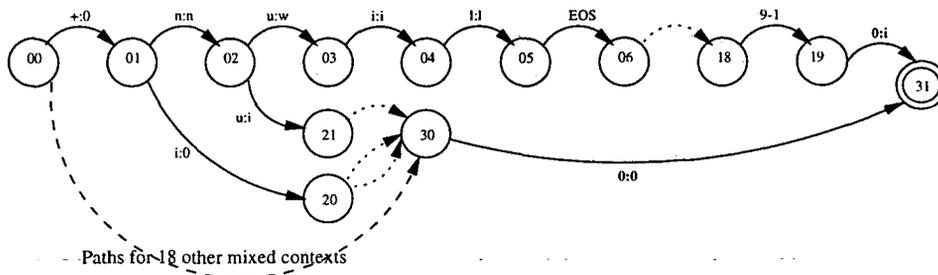


Figure 4.4: Mixed-context ADFSA subgraph for $0:i$

this subgraph in Figure 4.4:

[74]

$$0:i \Rightarrow u:w +:0 _ n:n$$

and

[75]

$$0:i \Leftarrow u:w +:0 _ n:n$$

The contexts of both rules are extracted after traversing from the root node to the edge labeled $u:w$, which ends in node 03 . This works for the first rule, since from this edge no terminal edge labeled with a default pair $(0:0)$ is reachable, while the terminal edge labeled with $0:i$ is reachable. Similarly, for the second rule no terminal edge labeled with a feasible pair $0:-i$ is reachable, while the terminal edge labeled with $0:i$ is reachable.

4.7 Summary

In the previous sections we have shown that to acquire the optimal rule set $\mathcal{R}_{\mathcal{W}}$ for \mathcal{W} , we need to construct the DAG $\mathcal{G} = G(M(C_{L(s)}^{full}(\mathcal{W})))$ for each special pair s appearing in \mathcal{W} and compute minimal edge-delimiter sets.

The original two rule-type decision questions provided by Antworth (Section 3.2, page 34) do not explain in an algorithmic form where the special pairs serving as the CPs of the rules come from. Neither do they explain in a procedural way where the environments (rule contexts) come from, for the two questions to be *true*. In Section 4.2 we rephrased the two questions first in terms of full mixed-context sets. In Section 4.3 we further developed the reasoning used in Section 4.2, to rephrase the two questions in terms of shortened mixed-context sets. The definitions and formulas developed in Section 4.4 then allowed us to rephrase the conditions for the questions to be *true* in enough procedural detail to be implemented as a computer program. This procedural explanation makes use of an automaton accepting mixed contexts, which is then viewed as the DAG $\mathcal{G} = G(M(C_{L(s)}^{full}(\mathcal{W})))$. From \mathcal{G} , two delimiter sets are extracted for each special pair s :

1. For the \Rightarrow rules we need to compute the minimal edge-delimiter set \mathcal{D}_s^{min} and
2. for the \Leftarrow rules we need the minimal L-relative edge-delimiter set $\mathcal{D}_s^{minLrel}$.

We defined $\mathcal{D}_s^{min} = D_s^{min}(\mathcal{G})$ and $\mathcal{D}_s^{minLrel} = D_s^{minLrel}(\mathcal{G})$.

Furthermore, we defined $\mathcal{P}_s = P_s(\mathcal{G})$ to be all the paths in the DAG \mathcal{G} from the root node to the terminal node labeled with the special pair s .

The associated minimal discerning prefix partitioner is

$\mathcal{H}_s^{min} = \text{stringset}(\text{pathprefixes}(\mathcal{D}_s^{min}, \mathcal{P}_s)$ and the minimal L-relative discerning prefix partitioner is

$$\mathcal{H}_s^{minLrel} = \text{stringset}(\text{pathprefixes}(\mathcal{D}_s^{minLrel}, \mathcal{P}_s).$$

In addition we defined the environment for question 1 to be *true*, associated with the minimal discerning prefix partitioner, as

$$E_{\mathcal{H}_s^{min}} = E(\mathcal{H}_s^{min}) = \text{unmix}(x_1)|\text{unmix}(x_2)|\dots|\text{unmix}(x_n),$$

where $x_i \in \mathcal{H}_s^{min}$. We also defined the environment for question 2 to be *true*, associated with the minimal L-relative discerning prefix partitioner, to be $E_{\mathcal{H}_s^{minLrel}} = E(\mathcal{H}_s^{minLrel}) = \text{unmix}(x_1)|\text{unmix}(x_2)|\dots|\text{unmix}(x_n),$

$$\text{where } x_i \in \mathcal{H}_s^{minLrel}.$$

The optimal rule set for each special pair $s \in S$ in \mathcal{W} is $\mathcal{R}_{\mathcal{W},s} =$

$$\{ "s \Rightarrow E_{\mathcal{H}_s^{min}} " \} \cup \{ "s \Leftarrow E_{\mathcal{H}_s^{minLrel}} " \}.$$

In addition, we have shown how the best rules extracted from the mixed-context DAG, the right-context DAG and the left-context DAG are merged into the final rule set. The “best” rules are the rules with the least ambiguity and the shortest context. The less the ambiguity, the less the possible overgeneration, and the shorter the context the more general the rule.

Finally, the somewhat different generation of mixed contexts for insertion rules has been described.

Chapter 5

Results and Evaluation

5.1 Introduction

In this chapter two-level rule acquisition results are presented for example source-target word sets from four different languages: English adjectives, Xhosa noun locatives, Spanish adjectives and Afrikaans noun plurals. The examples from these four different languages serve to illustrate the language independence of the rule acquisition process. Furthermore, it is shown how the rule acquisition process can be scaled up to acquire a two-level rule set for thousands of words. Finally, the chapter concludes by illustrating the accuracy of an acquired rule set on previously *unseen* words. The unseen words are words which were not in the set of word pairs from which the rule set was acquired.

5.2 English Adjectives

Consider the example English adjective pairs given by (Antworth, 1990, p.106):

<u>Source</u>	→	<u>Target</u>
<i>big</i>	→	<i>bigger</i>
<i>big</i>	→	<i>biggest</i>
<i>clear</i>	→	<i>unclear</i>
<i>clear</i>	→	<i>unclearly</i>
<i>happy</i>	→	<i>unhappy</i>
<i>happy</i>	→	<i>unhappier</i>
<i>happy</i>	→	<i>unhappiest</i>
<i>happy</i>	→	<i>unhappily</i>
<i>real</i>	→	<i>unreal</i>
<i>cool</i>	→	<i>cooler</i>
<i>cool</i>	→	<i>coolest</i>
<i>cool</i>	→	<i>coolly</i>
<i>clear</i>	→	<i>clearer</i>
<i>clear</i>	→	<i>clearest</i>
<i>clear</i>	→	<i>clearly</i>
<i>red</i>	→	<i>redder</i>
<i>red</i>	→	<i>reddest</i>
<i>real</i>	→	<i>really</i>
<i>happy</i>	→	<i>happier</i>
<i>happy</i>	→	<i>happiest</i>
<i>happy</i>	→	<i>happily</i>

In phase one the acquisition process correctly acquires the segmentation for these twenty-one adjective pairs:

<u>Target</u>	=	<u>Prefix</u>	+	<u>Source</u>	+	<u>Suffix</u>
<i>bigger</i>	=			<i>big</i>	+	<i>er</i>
<i>biggest</i>	=			<i>big</i>	+	<i>est</i>
<i>unclear</i>	=	<i>un</i>	+	<i>clear</i>		
<i>unclearly</i>	=	<i>un</i>	+	<i>clear</i>	+	<i>ly</i>
<i>unhappy</i>	=	<i>un</i>	+	<i>happy</i>		
<i>unhappier</i>	=	<i>un</i>	+	<i>happy</i>	+	<i>er</i>
<i>unhappiest</i>	=	<i>un</i>	+	<i>happy</i>	+	<i>est</i>
<i>unhappily</i>	=	<i>un</i>	+	<i>happy</i>	+	<i>ly</i>
<i>unreal</i>	=	<i>un</i>	+	<i>real</i>		
<i>cooler</i>	=			<i>cool</i>	+	<i>er</i>
<i>coolest</i>	=			<i>cool</i>	+	<i>est</i>
<i>coolly</i>	=			<i>cool</i>	+	<i>ly</i>
<i>clearer</i>	=			<i>clear</i>	+	<i>er</i>
<i>clearest</i>	=			<i>clear</i>	+	<i>est</i>
<i>clearly</i>	=			<i>clear</i>	+	<i>ly</i>
<i>redder</i>	=			<i>red</i>	+	<i>er</i>
<i>reddest</i>	=			<i>red</i>	+	<i>est</i>
<i>really</i>	=			<i>real</i>	+	<i>ly</i>
<i>happier</i>	=			<i>happy</i>	+	<i>er</i>
<i>happiest</i>	=			<i>happy</i>	+	<i>est</i>
<i>happily</i>	=			<i>happy</i>	+	<i>ly</i>

From these segmentations, the morphotactic component (Section 1.2.1, page 6) required by the morphological analyzer/generator is generated with uncomplicated text-processing routines. Six simple rules are acquired in phase

two¹:

[78]

$$0:d \Leftarrow d:d _ +:0$$

$$0:d \Rightarrow d:d _ +:0$$

$$0:g \Leftarrow g:g _ +:0$$

$$0:g \Rightarrow g:g _ +:0$$

$$y:i \Leftarrow _ +:0$$

$$y:i \Rightarrow _ +:0$$

Note that these six simple rules can be merged into three correct \Leftrightarrow rules which do the same work, but are more readable:

[79]

$$0:d \Leftrightarrow d:d _ +:0$$

$$0:g \Leftrightarrow g:g _ +:0$$

$$y:i \Leftrightarrow _ +:0$$

5.3 Xhosa Noun Locatives

To better illustrate the complexity of the rules that can be learned automatically by our process, consider the following set of fourteen Xhosa noun-locative pairs:

¹The results in this thesis were verified on either the two-level processor PC-KIMMO (Antworth, 1990) or the Xerox Finite State Tools. The two-level rule compiler KGEN (developed by Nathan Miles) was used to compile the acquired rules into the state tables required by PC-KIMMO. Both PC-KIMMO and KGEN are available from the Summer Institute of Linguistics (<http://www.sil.org/>). The Xerox Finite State Tools were kindly provided by the Multi-Lingual Theory and Technology (MLTT) Group, Rank Xerox Research Center, Grenoble.

<u>Source Word</u>	→	<u>Target Word</u>	<u>Glossary</u>
<i>inkosi</i>	→	<i>enkosini</i>	<i>at the captain</i>
<i>iinkosi</i>	→	<i>ezinkosini</i>	<i>at the captains</i>
<i>ihashe</i>	→	<i>ehasheni</i>	<i>on/at the horse</i>
<i>imbewu</i>	→	<i>embewini</i>	<i>in/at the seed</i>
<i>amanzi</i>	→	<i>emanzini</i>	<i>in/at the water</i>
<i>ubuchopho</i>	→	<i>ebucotsheni</i>	<i>in the brain</i>
<i>ilizwe</i>	→	<i>elizweni</i>	<i>in the country</i>
<i>ilanga</i>	→	<i>elangeni</i>	<i>in/at the sun</i>
<i>ingubo</i>	→	<i>engubeni</i>	<i>on the cloth</i>
<i>ingubo</i>	→	<i>engutyeni</i>	<i>on the cloth</i>
<i>indlu</i>	→	<i>endlini</i>	<i>in the house</i>
<i>indlu</i>	→	<i>endlwini</i>	<i>in the house</i>
<i>ikhaya</i>	→	<i>ekhayeni</i>	<i>at the house</i>
<i>ikhaya</i>	→	<i>ekhaya</i>	<i>at the house</i>

Note that this set contains ambiguity: The locative of *ingubo* is either *engubeni* or *engutyeni*. Our process must learn the necessary two-level rules to map *ingubo* to *engubeni* and *engutyeni*, as well as to map both *engubeni* and *engutyeni* in the other direction, i.e. to *ingubo*. Similarly, *indlu* and *ikhaya* each have two different locative forms. Furthermore, the two source words *inkosi* and *iinkosi* (the plural of *inkosi*) differ only by a prefixed *i*, but they have different locative forms. This small difference between source words provides an indication of the sensitivity required of the acquisition process to provide the necessary discerning information to a two-level morphological processor. At the same time, our process needs to

cope with possibly radical modifications between source and target words. Consider the mapping between *ubuchopho* and its locative *ebucotsheni*. Here, the only segments which stay the same from the source to the target word are the three letters *-buc-*, the letter *-o-* (the deletion of the first *-h-* is correct) and the second *-h-*.

The target words are correctly segmented during phase one as:

[81]

<u>Target</u>	=	<u>Prefix</u> +	<u>Source</u>	+ <u>Suffix</u>
<i>enkosini</i>	=	<i>e +</i>	<i>inkosi</i>	+ <i>ni</i>
<i>ezinkosini</i>	=	<i>e +</i>	<i>inkosi</i>	+ <i>ni</i>
<i>ehasheni</i>	=	<i>e +</i>	<i>ihashe</i>	+ <i>ni</i>
<i>embewini</i>	=	<i>e +</i>	<i>imbewu</i>	+ <i>ni</i>
<i>emanzini</i>	=	<i>e +</i>	<i>amanzi</i>	+ <i>ni</i>
<i>ebucotsheni</i>	=	<i>e +</i>	<i>ubuchopho</i>	+ <i>ni</i>
<i>elizweni</i>	=	<i>e +</i>	<i>ilizwe</i>	+ <i>ni</i>
<i>elangeni</i>	=	<i>e +</i>	<i>ilanga</i>	+ <i>ni</i>
<i>engubeni</i>	=	<i>e +</i>	<i>ingubo</i>	+ <i>ni</i>
<i>engutyeni</i>	=	<i>e +</i>	<i>ingubo</i>	+ <i>ni</i>
<i>endlini</i>	=	<i>e +</i>	<i>indlu</i>	+ <i>ni</i>
<i>endlwini</i>	=	<i>e +</i>	<i>indlu</i>	+ <i>ni</i>
<i>ekhayeni</i>	=	<i>e +</i>	<i>ikhaya</i>	+ <i>ni</i>
<i>ekhaya</i>	=	<i>e +</i>	<i>ikhaya</i>	

Note that the prefix *e+* is computed for all the input target words, while all but *ekhaya* (a correct alternative of *ekhayeni*) have *+ni* as a suffix.

From this segmented data, phase two computes 34 minimal context rules. These rules perfectly analyze and generate the 14 source-target word pairs:

$$0:e \leftarrow o:y +:0 - n:n$$

$$0:e \Rightarrow o:y +:0 -$$

$$0:i \leftarrow u:w +:0 - n:n$$

$$0:i \Rightarrow u:w +:0 -$$

$$0:s \leftarrow p:t - h:h$$

$$0:s \Rightarrow p:t -$$

$$a:0 \leftarrow +:0 -$$

$$a:0 \Rightarrow +:0 -$$

$$a:e \leftarrow - +:0$$

$$a:e \Rightarrow - +:0$$

$$b:t \leftarrow - o:y$$

$$b:t \Rightarrow - o:y$$

$$h:0 \leftarrow c:c -$$

$$h:0 \Rightarrow c:c -$$

$$i:0 \leftarrow +:0 - n:n$$

$$i:0 \leftarrow - k:k$$

$$i:0 \leftarrow - l:l$$

$$i:0 \leftarrow - h:h$$

$$i:0 \leftarrow - m:m$$

$$i:0 \Rightarrow +:0 -$$

$$i:z \leftarrow - i:i$$

$$i:z \Rightarrow - i:i$$

continued on next page

continued from previous page

$$o:e \leftarrow _ +:0 \ n:n$$

$$o:e \Rightarrow _ +:0$$

$$o:y \leftarrow b:t _$$

$$o:y \Rightarrow b:t _$$

$$p:t \leftarrow o:o _$$

$$p:t \Rightarrow o:o _$$

$$u:0 \leftarrow +:0 _$$

$$u:0 \Rightarrow +:0 _$$

$$u:i \leftarrow _ +:0 \ n:n$$

$$u:i \Rightarrow _ +:0$$

$$u:w \leftarrow _ +:0 \ 0:i \ n:n$$

$$u:w \Rightarrow l:l _$$

The vertical bar (“|”) is the traditional two-level notation which indicate the disjunction of two (or more) contexts. As with the rules acquired in Section 5.2, the \leftarrow and \Rightarrow rules of a special pair can be merged into a single \Leftrightarrow rule, if required. For example the two rules above for the special pair $i:z$ can be merged into

[83]

$$i:z \Leftrightarrow _ i:i$$

since this \Leftrightarrow does the same work as the \leftarrow and \Rightarrow rules together.

5.4 Spanish Adjectives

Consider the following fifty Spanish feminine adjectives and their superlatives: These fifty adjective pairs were selected randomly from a set of 643

adjective pairs².

The first phase correctly computed the morphotactic formulas:

[84]

<u>Target</u>	=	<u>Source</u>	+ <u>Suffix</u>
<i>acerrimas</i>	=	<i>acre</i>	+ <i>imas</i>
<i>admirativísimas</i>	=	<i>admirativo</i>	+ <i>ísimas</i>
<i>afirmativísimas</i>	=	<i>afirmativo</i>	+ <i>ísimas</i>
<i>alajuelensísimas</i>	=	<i>alajuelense</i>	+ <i>ísimas</i>
<i>alardosísimas</i>	=	<i>alardoso</i>	+ <i>ísimas</i>
<i>alavensísimas</i>	=	<i>alavense</i>	+ <i>ísimas</i>
<i>alcoyanísimas</i>	=	<i>alcoyano</i>	+ <i>ísimas</i>
<i>alicucísimas</i>	=	<i>alicuz</i>	+ <i>ísimas</i>
<i>altísimas</i>	=	<i>alto</i>	+ <i>ísimas</i>
<i>ambiciosísimas</i>	=	<i>ambicioso</i>	+ <i>ísimas</i>
<i>aragonesísimas</i>	=	<i>aragonés</i>	+ <i>ísimas</i>
<i>arterísimas</i>	=	<i>artero</i>	+ <i>ísimas</i>
<i>artistiquísimas</i>	=	<i>artístico</i>	+ <i>ísimas</i>
<i>asalariadísimas</i>	=	<i>asalariado</i>	+ <i>ísimas</i>
<i>atentísimas</i>	=	<i>atento</i>	+ <i>ísimas</i>
<i>australianísimas</i>	=	<i>australiano</i>	+ <i>ísimas</i>
<i>avarísimas</i>	=	<i>avaro</i>	+ <i>ísimas</i>
<i>avariciosísimas</i>	=	<i>avaricioso</i>	+ <i>ísimas</i>
<i>baladorísimas</i>	=	<i>balador</i>	+ <i>ísimas</i>

continued on next page

²These Spanish feminine adjectives were kindly provided by the MLTT group at Xerox, Grenoble.

continued from previous page		
------------------------------	--	--

<i>basiquísimas</i>	=	<i>básico</i>	+ <i>ísimas</i>
<i>bastitanísimas</i>	=	<i>bastitano</i>	+ <i>ísimas</i>
<i>bayamonesísimas</i>	=	<i>bayamonés</i>	+ <i>ísimas</i>
<i>benevolísimas</i>	=	<i>benévolo</i>	+ <i>ísimas</i>
<i>biobienquísimas</i>	=	<i>biobiense</i>	+ <i>ísimas</i>
<i>bizantinísimas</i>	=	<i>bizantino</i>	+ <i>ísimas</i>
<i>bobatiquísimas</i>	=	<i>bobático</i>	+ <i>ísimas</i>
<i>bogotanísimas</i>	=	<i>bogotano</i>	+ <i>ísimas</i>
<i>borgoñonísimas</i>	=	<i>borgoñón</i>	+ <i>ísimas</i>
<i>brasilerísimas</i>	=	<i>brasileiro</i>	+ <i>ísimas</i>
<i>burgalesísimas</i>	=	<i>burgalés</i>	+ <i>ísimas</i>
<i>caballeresquísimas</i>	=	<i>caballeresco</i>	+ <i>ísimas</i>
<i>calidísimas</i>	=	<i>cálido</i>	+ <i>ísimas</i>
<i>campechanísimas</i>	=	<i>campechano</i>	+ <i>ísimas</i>
<i>canoniquísimas</i>	=	<i>canónico</i>	+ <i>ísimas</i>
<i>capitalistísimas</i>	=	<i>capitalista</i>	+ <i>ísimas</i>
<i>caspolinísimas</i>	=	<i>caspolino</i>	+ <i>ísimas</i>
<i>chalaquísimas</i>	=	<i>chalaco</i>	+ <i>ísimas</i>
<i>chiricanísimas</i>	=	<i>chiricano</i>	+ <i>ísimas</i>
<i>chorreantísimas</i>	=	<i>chorreante</i>	+ <i>ísimas</i>
<i>clericalísimas</i>	=	<i>clerical</i>	+ <i>ísimas</i>
<i>compatibilísimas</i>	=	<i>compatible</i>	+ <i>ísimas</i>
<i>competitivísimas</i>	=	<i>competitivo</i>	+ <i>ísimas</i>

continued on next page

continued from previous page

<i>compostelanísimas</i>	=	<i>compostelano</i>	+ <i>ísimas</i>
<i>convinentísimas</i>	=	<i>convincente</i>	+ <i>ísimas</i>
<i>critiquísimas</i>	=	<i>crítico</i>	+ <i>ísimas</i>
<i>crudísimas</i>	=	<i>crudo</i>	+ <i>ísimas</i>
<i>cruentísimas</i>	=	<i>cruento</i>	+ <i>ísimas</i>
<i>cubiertísimas</i>	=	<i>cubierto</i>	+ <i>ísimas</i>
<i>cumanagotísimas</i>	=	<i>cumanagoto</i>	+ <i>ísimas</i>
<i>cuzqueñísimas</i>	=	<i>cuzqueño</i>	+ <i>ísimas</i>

The second phase acquired the following 36 two-level sound-changing rules:

[85]

$o:0$	\Leftarrow	$n:n$	_
$o:0$	\Leftarrow	$t:t$	_
$o:0$	\Leftarrow	$d:d$	_ +:0
$o:0$	\Leftarrow	$r:r$	_
$o:0$	\Leftarrow	$s:s$	_
$o:0$	\Leftarrow	$v:v$	_ +:0
$o:0$	\Leftarrow	$\tilde{n}:\tilde{n}$	_
$o:0$	\Leftarrow	$l:l$	_
$o:0$	\Rightarrow	$n:n$	_ $t:t$ _ $d:d$ _ +:0 $r:r$ _
		$s:s$	_ $v:v$ _ +:0 $\tilde{n}:\tilde{n}$ _ $l:l$ _
$o:u$	\Leftarrow	$c:q$	_ +:0

continued on next page

continued from previous page

 $o:u \Rightarrow c:q - +:0$ $z:c \Leftarrow - +:0$ $z:c \Rightarrow - +:0$ $0:e \Leftarrow \# a:a c:c - r:r$ $0:e \Rightarrow \# a:a c:c -$ $0:i \Leftarrow i:i b:b - l:l$ $0:i \Rightarrow i:i b:b -$ $\acute{a}:a \Leftarrow b:b -$ $\acute{a}:a \Leftarrow c:c -$ $\acute{a}:a \Rightarrow b:b - | c:c -$ $\acute{e}:e \Leftarrow n:n -$ $\acute{e}:e \Leftarrow - s:s$ $\acute{e}:e \Rightarrow n:n - | - s:s$ $\acute{i}:i \Leftarrow r:r -$ $\acute{i}:i \Leftarrow t:t -$ $\acute{i}:i \Rightarrow r:r - | t:t -$ $\acute{o}:o \Leftarrow - n:n$ $\acute{o}:o \Rightarrow - n:n$ $a:0 \Leftarrow - +:0$ $a:0 \Rightarrow - +:0$ $c:q \Leftarrow - o:u +:0$ $c:q \Rightarrow - o:u +:0$ $e:0 \Leftarrow - +:0 \acute{i}:i$

continued on next page

continued from previous page

$$e:0 \Rightarrow s:s _ | t:t _ +:0 | l:l _ +:0$$

$$e:r \Leftarrow _ +:0 i:i$$

$$e:r \Rightarrow r:r _ +:0$$

The hashes (#) in the contexts of the $0:e$ rules are the normal notation to indicate the beginning or end of a word. These 36 rules correctly analyze the 50 word pairs, but overgenerated in the case of seven word pairs:

[86]

<u>Source</u>	<u>Correct Target</u>	<u>Overgenerated Non-word</u>
<i>artístico</i>	<i>artistiquísimas</i>	<i>artísticoísimas</i>
<i>básico</i>	<i>basiquísimas</i>	<i>basicoísimas</i>
<i>bobático</i>	<i>bobatiquísimas</i>	<i>bobáticoísimas</i>
<i>caballeresco</i>	<i>caballeresquísimas</i>	<i>caballerescoísimas</i>
<i>canónico</i>	<i>canoniquísimas</i>	<i>canónicoísimas</i>
<i>chalaco</i>	<i>chalaquísimas</i>	<i>chalacoísimas</i>
<i>crítico</i>	<i>critiquísimas</i>	<i>críticoísimas</i>

The reason for these overgenerations is that the automatic acquisition cannot acquire only the lexical or the surface component of a feasible pair in the contexts. Thus the automatic algorithm sometimes acquires slightly overspecified rules. This overspecification of the rules sometimes causes overgeneration³ (compare (Antworth, 1990, p.39)). We need to modify the $o:u \Leftarrow c:q _ +:0$ rule manually into:

³Overspecification in general may also cause rule conflicts (compare (Antworth, 1990, p.39)). However, rules acquired with our automatic algorithm never caused unresolvable rule conflicts in the tested examples.

$$o:u \Leftarrow c: - +:0$$

Notice that the $c:q$ in the context has been changed to “ $c:$ ”. This new rule means that a lexical o corresponds to a surface u always following a c on the lexical level and preceding a morpheme boundary. This c on the lexical level may correspond to any letter in the alphabet on the surface level. With this single modification the 36 rules perfectly analyze and generate the 50 adjectively related word pairs.

5.5 Afrikaans Noun Plurals

To test the acquisition process on Afrikaans noun plurals, we selected 57 singular-plural pairs from an Afrikaans dictionary. The first phase correctly computed the following morphotactic formulas for the 57 pairs:

<u>Target</u>	=	<u>Source</u>	+ <u>Suffix</u>
<i>alveolare</i>	=	<i>alveolaar</i>	+ <i>e</i>
<i>ampse</i>	=	<i>ampseed</i>	+ <i>e</i>
<i>asjasse</i>	=	<i>asjas</i>	+ <i>e</i>
<i>barbarismes</i>	=	<i>barbarisme</i>	+ <i>s</i>
<i>beddens</i>	=	<i>bed</i>	+ <i>s</i>
<i>bedinge</i>	=	<i>beding</i>	+ <i>e</i>

<i>continued on next page</i>

continued from previous page		
------------------------------	--	--

<i>brandstroke</i>	=	<i>brandstrook</i>	+ e
<i>dekane</i>	=	<i>dekaan</i>	+ e
<i>depressies</i>	=	<i>depressie</i>	+ s
<i>elande</i>	=	<i>eland</i>	+ e
<i>emetika</i>	=	<i>emetikum</i>	+ a
<i>emetikums</i>	=	<i>emetikum</i>	+ s
<i>floras</i>	=	<i>flora</i>	+ s
<i>gewelfhoeke</i>	=	<i>gewelfhoek</i>	+ e
<i>goggas</i>	=	<i>gogga</i>	+ s
<i>gooiringe</i>	=	<i>gooring</i>	+ e
<i>grille</i>	=	<i>gril</i>	+ e
<i>inkomelinge</i>	=	<i>inkomeling</i>	+ e
<i>kajaks</i>	=	<i>kajak</i>	+ s
<i>kandelas</i>	=	<i>kandela</i>	+ s
<i>kasrekenings</i>	=	<i>kasrekening</i>	+ s
<i>kaste</i>	=	<i>kas</i>	+ e
<i>katte</i>	=	<i>kat</i>	+ e
<i>kraagstene</i>	=	<i>kraagsteen</i>	+ e
<i>kreasies</i>	=	<i>kreasie</i>	+ s
<i>kwekelinge</i>	=	<i>kwekeling</i>	+ e
<i>lesers</i>	=	<i>leser</i>	+ s
<i>liefies</i>	=	<i>liefie</i>	+ s
<i>lowwe</i>	=	<i>loof</i>	+ e

continued on next page

continued from previous page		
------------------------------	--	--

<i>mededaders</i>	=	<i>mededader</i>	+ s
<i>nadroejakkalse</i>	=	<i>nadroejakkals</i>	+ e
<i>nekrologieë</i>	=	<i>nekrologie</i>	+ ë
<i>ohms</i>	=	<i>ohm</i>	+ s
<i>outeurs</i>	=	<i>outeur</i>	+ s
<i>palankyne</i>	=	<i>palankyn</i>	+ e
<i>paljasse</i>	=	<i>paljas</i>	+ e
<i>parias</i>	=	<i>paria</i>	+ s
<i>persgesprekke</i>	=	<i>persgesprek</i>	+ e
<i>pietse</i>	=	<i>piets</i>	+ e
<i>polsstokke</i>	=	<i>polsstok</i>	+ e
<i>redakteurs</i>	=	<i>redakteur</i>	+ s
<i>reisigers</i>	=	<i>reisiger</i>	+ s
<i>relatiewe</i>	=	<i>relatief</i>	+ e
<i>sarsies</i>	=	<i>sarsie</i>	+ s
<i>selfaansitters</i>	=	<i>selfaansitter</i>	+ s
<i>sinekures</i>	=	<i>sinekure</i>	+ s
<i>skeepsagente</i>	=	<i>skeepsagent</i>	+ e
<i>skeppings</i>	=	<i>skepping</i>	+ s
<i>strokiesfilms</i>	=	<i>strokiesfilm</i>	+ s
<i>stronke</i>	=	<i>stronk</i>	+ e
<i>suffikse</i>	=	<i>suffiks</i>	+ e
<i>swartjies</i>	=	<i>swartjie</i>	+ s

continued on next page

<i>continued from previous page</i>

swartkunste = *swartkuns* + *e*

tertvulsels = *tertvulsel* + *s*

uitgrawings = *uitgrawing* + *s*

vampiere = *vampier* + *e*

verswerings = *verswering* + *s*

Afrikaans plurals are almost always derived with the addition of a suffix (mostly *-e* or *-s*) to the singular form. Different sound changes may occur during this process. For example⁴, gemination, which indicates the shortening of a preceding vowel, occurs frequently (e.g. *kat* → *katte*), as well as consonant insertion (e.g. *kas* → *kaste*) and elision (e.g. *ampseed* → *ampsede*). Several sound changes may occur in the same word. For example, elision, consonant replacement and gemination occurs in *loof* → *lowwe*. Afrikaans (a Germanic language) has borrowed a few words from Latin. Some of these words have two plural forms, which introduce ambiguity in the word mappings: One plural is formed with a Latin suffix (*-a*) (e.g. *emetikum* → *emetika*) and one with an indigenous suffix (*-s*) (e.g. *emetikum* → *emetikums*). Allomorphs occur as well, for example *-ens* is an allomorph of the suffix *-s* in *bed* + *s* → *beddens*. Phase two acquired the following 30 sound-changing rules:

[89]

$$0:d \Rightarrow d:d +:0 - 0:e 0:n s:s$$

$$0:e \Rightarrow d:d +:0 0:d - 0:n s:s$$

<i>continued on next page</i>

⁴All examples come from the 57 input word pairs. Fifty word pairs were randomly selected and these seven examples, each of which illustrates an aspect, were added.

continued from previous page

$0:k \leftarrow r:r \ e:e \ k:k \ +:0 \ - \ e:e$
 $0:k \leftarrow t:t \ o:o \ k:k \ +:0 \ - \ e:e$
 $0:k \Rightarrow r:r \ e:e \ k:k \ +:0 \ - \mid t:t \ o:o \ k:k \ +:0 \ -$
 $0:l \leftarrow l:l \ +:0 \ - \ e:e$
 $0:l \Rightarrow l:l \ +:0 \ - \ e:e$
 $0:n \Rightarrow d:d \ +:0 \ 0:d \ 0:e \ - \ s:s$
 $0:s \leftarrow j:j \ a:a \ s:s \ +:0 \ - \ e:e$
 $0:s \Rightarrow j:j \ a:a \ s:s \ +:0 \ -$
 $0:t \leftarrow a:a \ t:t \ +:0 \ - \ e:e$
 $0:t \leftarrow k:k \ a:a \ s:s \ +:0 \ - \ e:e$
 $0:t \leftarrow n:n \ s:s \ +:0 \ - \ e:e$
 $0:t \Rightarrow a:a \ t:t \ +:0 \ - \mid k:k \ a:a \ s:s \ +:0 \ - \mid n:n \ s:s \ +:0 \ -$
 $a:0 \leftarrow k:k \ a:a \ -$
 $a:0 \leftarrow l:l \ a:a \ -$
 $a:0 \Rightarrow k:k \ a:a \ - \mid l:l \ a:a \ -$
 $e:0 \leftarrow e:e \ - \ d:d$
 $e:0 \leftarrow e:e \ - \ n:n$
 $e:0 \Rightarrow e:e \ - \ d:d \mid e:e \ - \ n:n$
 $f:w \leftarrow _ \ +:0$
 $f:w \Rightarrow _ \ +:0$
 $m:0 \leftarrow _ \ +:0 \ a:a$
 $m:0 \Rightarrow _ \ +:0 \ a:a$
 $o:0 \leftarrow o:o \ - \ k:k$

continued on next page

continued from previous page

$$o:0 \Rightarrow o:o _ k:k$$

$$o:w \Leftarrow _ f:w$$

$$o:w \Rightarrow _ f:w$$

$$u:0 \Leftarrow _ m:0 +:0 a:a$$

$$u:0 \Rightarrow _ m:0 +:0 a:a$$

These two-level rules correctly analyze and generate the 57 input word pairs, except for an overgeneration on *bed* → *beddens*. This overgeneration is *bed* → **beds*. The only way to prevent this overgeneration, is to manually add the following exclusion rule:

[90]

$$s:s \ / \Leftarrow b:b e:e d:d +:0 _$$

The next step was to show the feasibility of automatically acquiring a minimal rule set for a wide-coverage parser. To get hundreds or even thousands of input pairs, we implemented routines to extract the lemmas (“head words”) and their inflected forms from a machine-readable dictionary (Theron and Cloete, 1992; Theron, 1993). In this way we extracted 3935 Afrikaans noun-plural pairs which could serve as the input to our process.

During phase one, all of the 3935 input word pairs were segmented correctly. This took less than two minutes on a Pentium-Pro running Linux and the peak memory usage was less than three megabytes.

To facilitate the evaluation of phase two, we define a *simple rule* as a rule which has an environment consisting of a single context. This is in contrast with an environment consisting of two or more contexts disjoined together.

Phase two acquired 1196 *simple rules* for 43 special pairs. This took less than six hours on a Pentium-Pro running Linux and the peak memory usage was less than twenty megabytes.

Of these 1196 simple rules, 593 are \Leftarrow rules and 603 are \Rightarrow rules. The average length of the simple rule contexts is 5.36 feasible pairs. Compare this with the average length of the 3935 final input edit sequences which is 12.6 feasible pairs. The 1196 simple rules can be reduced to 42 \Leftarrow rules and 43 \Rightarrow rules (i.e. one rule per special pair) with environments consisting of disjuncted contexts. This acquired set of 42 \Leftarrow rules and 43 \Rightarrow rules do not analyze and generate the 3935 word pairs 100% correctly — there is overgeneration on 680 (17.2%) of the source words and two overrecognitions. There are, however, no failures — the correct target words are always included in the lists of overgenerated forms.

The total number of feasible pairs in the 3935 final input edit strings is 49657. In the worst case, all these feasible pairs should be present in the rule contexts to accurately model the sound changes which might occur in the input pairs. However, the actual result is much better: Our process acquires a two-level rule set which models the sound changes with only 12.9% (6405) of the number of input feasible pairs. Since most feasible pairs are used twice in the rule set (once in the context of a \Leftarrow rule and once in a context of a \Rightarrow rule), the actual number of different feasible pairs used is closer to half the figure given above, i.e. 6.45% (3203) of the input feasible pairs.

To perfectly analyze and generate the 3935 word pairs, i.e. with no overgeneration or overrecognition, I manually added 17 exclusion ($/\Leftarrow$) rules with a total of 75 contexts. Note that since our automatic acquisition process cannot acquire exclusion rules, these exclusion rules should always be manually added if overgeneration occurs. In addition, the underspecified contexts of

Rule set	\Leftarrow	\Rightarrow	Total no. of rules	No. of \Leftarrow contexts	No. of \Rightarrow contexts	Total no. of FPs
1	42	43	85	513	521	5381
2	39	40	79	519	526	5566
3	40	41	81	493	501	5231
4	40	41	81	503	510	5289
5	40	41	81	502	509	5293
Average:	40.2	41.2	79.6	506	513.4	5352

Table 5.1: Number of rules acquired for each rule-set trained on four-fifths of the word pairs.

16 of the acquired rules were enlarged, mostly to add the morpheme boundary as part of the context. There were 24 underspecified contexts, which is only 2% of the total number of contexts. These two groups of modifications took less than two days to make, with the aid of inspecting the mixed contexts and the analyzer/generator output. With these manual modifications, the rule set perfectly analyze and generate the 3935 word pairs.

5.5.1 Unseen Words

To obtain a prediction of the recognition and generation accuracy over *unseen* words, we divided the 3935 input pairs into five equal sections. Each fifth was held out in turn as test data while a set of two-level rules was learned from the remaining four-fifths. To get an indication of the size of the acquired rule sets, see Table 5.1. Table 5.1 lists the number and type of rules and rule contexts acquired for each of the five rule sets, as well as the total number of feasible pairs (FPs) used in each rule set.

Rule set	No. of $/\Leftarrow$ rules added	No. of $/\Leftarrow$ rule contexts	No. of contexts modified	New total no. of rules
1	18	70	30	103
2	17	69	24	96
3	18	72	24	99
4	16	51	28	97
5	15	70	24	96
Average:	16.8	66.4	26	98.2

Table 5.2: Modifications for perfect parsing to rule-sets trained on four-fifths of the word pairs.

For each of the five rounds, the acquired rule set was manually edited until that rule set perfectly analyzed and generated the four-fifths of word pairs from which the rule set was acquired. The number of $/\Leftarrow$ rules added and the number of rules modified for each rule set, are given in Table 5.2. With these modifications, each of the five acquired rule sets perfectly parsed the four-fifths training word-pairs.

These five modified rule sets were then each tested on the unseen one-fifth test data (787 word pairs in each case). The number and type of recognition errors are listed in Table 5.3 and the generation errors are listed in Table 5.4.

Table 5.5 lists the recognition and generation accuracy for each of the five tests. The average recognition accuracy over the unseen test word pairs was 98.9% while the average generation accuracy was 97.8%⁵.

⁵These results are an improvement over those in (Theron and Cloete, 1997; Theron, 1997a,b,c). The reason for this is that we acquire only \Leftarrow and \Rightarrow rules, and not \Leftrightarrow rules.

Rule set	Target words with recognition errors	Target words with overrecognition	Total no. of forms overrecognized	Total recognition failure
1	6	0	0	6
2	8	0	0	8
3	14	1	1	13
4	8	1	1	7
5	6	0	0	6
Average:	8.4	0.4	0.4	8

Table 5.3: Recognition errors on unseen one-fifth test word pairs.

Rule set	Source words with generation errors	Source words with overgeneration	Total no. of forms overgenerated	Total generation failure
1	13	10	13	6
2	25	19	25	8
3	25	16	22	13
4	12	6	7	7
5	11	6	7	6
Average:	17.2	11.4	14.8	8

Table 5.4: Generation errors on unseen one-fifth test word pairs.

Rule set	Target words which correctly recognized	Source words which correctly generated	% target words which correctly recognized	% source words which correctly generated
1	781	774	99.2%	98.4%
2	779	762	99.0%	96.8%
3	773	762	98.2%	96.8%
4	779	775	99.0%	98.5%
5	781	776	99.2%	98.6%
Average:	778.6	769.8	98.9%	97.8%

Table 5.5: Recognition and generation accuracy on the unseen one-fifth test data (787 word pairs in each case).

To my knowledge, no other researcher has done similar tests on the generation and recognition accuracy of a set of rules on previously unseen words. In my opinion, the results achieved here are excellent.

Furthermore, the exclusion ($/\Leftarrow$) rules are manually added here.

Chapter 6

Conclusion

6.1 Summary

There are many applications for computational systems which can do natural language processing (NLP). Example applications where some form of NLP is required are free-text information retrieval, machine-translation and computer-assisted language learning. An NLP system needs information on the language(s) it processes. This language specific information is typically stored in a lexicon, which is a detailed structured database on the words of the target language(s). Traditionally, there are several levels of language information discerned, e.g. the phonological level, the morphotactic level, the syntactic level and the semantic level. Up to now NLP systems have been limited in their coverage of the languages that they process. The reason for this is to a large extent due to their limited lexicons, which is manually constructed. To manually construct a lexicon can be time-consuming and error-prone. An alternative is to attempt the automatic acquisition of the lexicon.

This thesis contributes an automated method for the acquisition of phono-

logical and morphological components of the lexicon. To this end, use is made of a particular computational morphological framework, namely two-level morphology. A two-level morphological analyzer/generator is used to both analyze a target word into its morphemes, as well as to generate a target word from its underlying morphemes. The lexicon of a two-level morphological analyzer/generator consists of two components: (1) A morphotactic description of the words to be processed, as well as (2) a set of two-level phonological (or spelling) rules. In this thesis I have shown how the second component above is automatically acquired from source-target word pairs, where the target is an inflected form of the source word. It is assumed that the target word is formed from the source through the optional addition of a prefix and/or a suffix. Furthermore, I have shown how the first component is acquired as a by-product of the rule-acquisition process.

Two phases can be discerned in the rule-acquisition process: (1) segmentation of the target words into morphemes and (2) determination of the optimal two-level rule set with minimal discerning contexts. In the first phase, an acyclic deterministic finite state automaton (ADFSFA) is constructed from string edit sequences of the input source-target word pairs. Segmentation of the target words into morphemes is achieved through viewing the ADFSFA as a directed acyclic graph (DAG) and applying heuristics using properties of the DAG as well as the elementary string edit operations.

In phase two, the morphotactic formulas computed in the first phase are used as the input: The right-hand side of each morphotactic formula is mapped onto the left-hand side. This mapping is then used to compute new string edit sequences which serve as the lexical-surface representations of the input target words. These lexical-surface representations are used to generate mixed contexts, as well as left and right contexts. The mixed

contexts were then read into an acyclic deterministic finite state automaton, which was viewed as a DAG. I introduced delimiter edges which were used to extract the two-level rule type as well as the minimal rule contexts from the DAG. The same process was followed for the left- and right contexts. The three resulting rule sets (one from the mixed contexts, one from the left contexts and one from the right contexts) were then merged into the final two-level sound-changing rule set. This use of delimiter edges in a DAG provides the first procedural way to answer the two rule-type decision questions provided by (Antworth, 1990, p.53).

There are several advantages of the rule-acquisition process described in this thesis: This is the first description available of a method for the automatic acquisition of two-level morphological rules (Theron and Cloete, 1997). Furthermore, the acquired rule set can be used by publicly available morphological analyzers/generators. In addition, I have shown that the rule acquisition process is portable between subsets of at least four different languages (English adjectives, Xhosa noun locatives, Afrikaans noun plurals and Spanish adjectives). Furthermore, the acquired rule set generalizes very well to previously unseen words (i.e. words not used during the acquisition process). Finally I have shown that two-level rule sets can be acquired for wide-coverage parsers, by using thousands of source-target words extracted from a machine-readable dictionary.

6.2 Future Work

The aim of this thesis was to automate the two-level morphological rule acquisition process as much as possible. This aim has been reached, thus it is not clear what other steps can be automated. I can, however, name two

steps that are worth investigating: The first is in phase one. It would be helpful if words with infixes could also be correctly segmented. An example of a word with infixation is the Afrikaans plural noun *mond+e+vol*. Currently phase one can only segment prefixes and suffixes. Note that infixation does not influence phase two: Once the target word has been correctly segmented, phase two will acquire the correct two-level rules for any number of segmentations in the target word.

The second step that would be helpful if it were further automated is the generation of the exclusion ($/\Leftarrow$) rules in phase two. The exclusion rules are used to eliminate overgeneration. It is not clear how this can be automated, since the special pair used as the correspondence part (CP) of the exclusion rule is often not the same as the CP of the rule which allowed the overgeneration. Currently these exclusion rules need to be added manually. Fortunately, even for the few thousand word pairs used for tests in this thesis, this took less than two days.

Finally, with the good results in mind, the automatic acquisition of two-level rule sets for wide-coverage morphological analyzers/generators can now, for the first time, be successfully attempted.

Bibliography

- Alam, Y. S., 1983. A Two-level Morphological Analysis of Japanese. *Texas Linguistic Forum* 22:229–252.
- Alegria, I., Artola, X., Sarasola, K., and Urkia, M., 1996. Automatic Morphological Analysis of Basque. *Literary & Linguistic Computing* 11, no. 4:193–204.
- Antworth, E. L., 1990. *PC-KIMMO: A Two-level Processor for Morphological Analysis*. Dallas, Texas: Summer Institute of Linguistics.
- Beesley, K. R., 1996. Arabic Finite-State Morphological Analysis and Generation. In *COLING-96: 16th International Conference on Computational Linguistics*, vol. 1, pp. 89–94. Center for Sprogteknologi, Copenhagen.
- Daelemans, W., Berck, P., and Gillis, S., 1996. Unsupervised Discovery of Phonological Categories through Supervised Learning of Morphological Rules. In *COLING-96: 16th International Conference on Computational Linguistics*, pp. 95–100. Copenhagen, Denmark.
- Gasser, M., 1994. Acquiring Receptive Morphology: A Connectionist Model. In *Proceedings of ACL-94*, pp. 279–286. Association of Computational Linguistics, Morristown, New Jersey.

- Gasser, M., 1996. Transfer in a connectionist model of the acquisition of morphology. In *Yearbook of Morphology*, pp. 97–115. Netherlands: Kluwer Academic Publishers.
- Golding, A. R. and Thompson, H. S., 1985. A morphology component for language programs. *Linguistics*, no. 23:263–284.
- Grimes, J. E., 1983. *Affix positions and cooccurrences: the PARADIGM program*, vol. 69 of *Publications in Linguistics*. Dallas, Texas: Dallas: Summer Institute of Linguistics and University of Texas at Arlington.
- Haapalainen, M., Silvonen, M., Lindin, K., Koskenniemi, K., and Karlsson, F., 1994. GERTWOL. *LDV-Forum* 11, no. 1:17–33.
- Kahn, R., 1983. A Two-level Morphological Analysis of Rumanian. *Texas Linguistic Forum* 22:253–270.
- Karttunen, L. and Beesley, K. R., 1992. Two-level Rule Compiler. Technical Report ISTL-92-2, Xerox Palo Alto Research Center.
- Karttunen, L. and Wittenburg, K., 1983. A Two-level Morphological Analysis of English. *Texas Linguistic Forum* 22:217–228.
- Kiraz, G. A., 1996. SEMHE: A generalized two-level System. In *Proceedings of ACL-96*, pp. 159–166. Association of Computational Linguistics, Santa Cruz, California.
- Koskenniemi, K., 1983. *Two-level Morphology: A General Computational Model for Word-Form Recognition and Production*, vol. Publications No. 11. Helsinki, Finland: University of Helsinki Department of General Linguistics.

- Koskeniemi, K., 1990. A discovery procedure for two-level phonology. In *Computational Lexicology and Lexicography: Special Issue dedicated to Bernard Quemada, Vol. I*, eds. L. Cignoni and C. Peters, pp. 451–465. Pisa: Linguistica Computazionale, Volume VI.
- Kuusik, E., 1996. Learning Morphology: Algorithms for the Identification of Stem Changes. In *COLING-96: 16th International Conference on Computational Linguistics*, pp. 1102–1105. Copenhagen, Denmark.
- Lun, S., 1983. A Two-level Morphological Analysis of French. *Texas Linguistic Forum* 22:271–278.
- Marzal, A. and Vidal, E., 1993. Computation of Normalized Edit Distance and Applications. *IEEE Trans. Pattern Analysis and Machine Intelligence* 15, no. 9:926–932.
- Oflazer, K., 1994. Two-level description of Turkish morphology. *Literary & Linguistic Computing* 9, no. 2:137–148.
- Revuz, D., 1992. Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science* 92:181–189.
- Sankoff, D. and Kruskal, J. B., 1983. *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison*. Massachusetts: Addison-Wesley.
- Sgarbas, K., Fakotakis, N., and Kokkinakis, G., 1995. A PC-KIMMO-Based Morphological Description of Modern Greek. *Literary & Linguistic Computing* 10, no. 3:189–201.
- Simons, G. F., 1988. Studying morphophonemic alternation in annotated

- text, parts one and two. *Notes on Linguistics* , no. 41,42:41:41–46; 42:27–38.
- Sproat, R., 1992. *Morphology and Computation*. Cambridge, Massachusetts: The MIT Press.
- Theron, P., 1993. *Towards an Automated Methodology for Building a Relational Lexicon from a Dictionary*. Master's thesis, University of Stellenbosch, Stellenbosch, South Africa.
- Theron, P., 1997a. Automatic Acquisition of Two-Level Morphological Lexicons. Invited Presentation: *Séminaire de Recherche en Linguistique Informatique*, Departement of Linguistics, University of Geneva, February 13, 1997.
- Theron, P., 1997b. Automatic Acquisition of Two-Level Morphological Rules. Invited Presentation: Multi-Lingual Theory and Technology Group, Rank Xerox Research Center, Grenoble, France, March 10, 1997.
- Theron, P., 1997c. Automatic Acquisition of Two-Level Morphological Rules. Invited Presentation: *Workshop of the Swiss Group for Artificial Intelligence and Cognitive Science (SGAICO): Special Interest Group 'Natural Language Processing'*, University of Zurich, May 20, 1997.
- Theron, P. and Cloete, I., 1992. Automatically linking words and concepts in an Afrikaans dictionary. *The Southern African Computer Journal* , no. 7:9–14.
- Theron, P. and Cloete, I., 1997. Automatic Acquisition of Two-Level Morphological Rules. In *Fifth Conference on Applied Natural Language Pro-*

cessing, ed. R. Grishman, pp. 103–110. Association for Computational Linguistics, Washington: Morgan Kaufmann Publishers.

Wothke, K., 1986. Machine learning of morphological rules by generalization and analogy. In *COLING-86: 11th International Conference on Computational Linguistics*, pp. 289–293. Bonn.

Appendix A

Semantics of Two-Level Rules

Table A.1 summarizes the semantics of two-level rules. In Table A.1, L indicates a character on the lexical level, S a character in the surface level and E indicates the environment or context of the two-level rule. The “ \neg ” symbol means *not*.

Table A.2 is a truth table for the two-level rules. These two tables appear in (Antworth, 1990) on a loose leaflet.

$L:S \Rightarrow E$	<p>“Only but not always.”</p> <p>L is realized as S only in E.</p> <p>L realized as S is not allowed in $\neg E$.</p> <p>If $L:S$, then it must be in E.</p> <p>Implies $L:\neg S$ in E is permitted.</p>
$L:S \Leftarrow E$	<p>“Always but not only.”</p> <p>L is always realized as S in E.</p> <p>L realized as $\neg S$ is not allowed in $\neg E$.</p> <p>If L is in E, then it must be $L:S$.</p> <p>Implies $L:S$ may occur elsewhere.</p>
$L:S \Leftrightarrow E$	<p>“Always and only.”</p> <p>L is realized as S only and always in E.</p> <p>Both $L:S \Rightarrow E$ and $L:S \Leftarrow E$.</p> <p>Implies $L:S$ is obligatory in E and occurs nowhere else.</p>
$L:S \not\Leftarrow E$	<p>“Never.”</p> <p>L is never realized as S in E.</p> <p>L realized as S is not allowed in E.</p> <p>If L is in E, then it must be $L:\neg S$.</p>

Table A.1: Semantics of two-level rules

There is an L.		Is the rule satisfied?			
Is it realized as S ?	Is it in E ?	$L:S \Rightarrow E$	$L:S \Leftarrow E$	$L:S \Leftrightarrow E$	$L:S \nLeftarrow E$
T	T	T	T	T	F
T	F	F	T	F	T
F	T	T	F	F	T
F	F	T	T	T	T

Table A.2: Truth table for the two-level rules

Index

- \Leftarrow rule, 53, 65, 82
- \Rightarrow rule, 53, 64, 81, 91
- feasible pair
 - set, 47
- abstractions, 34
- accuracy
 - generation, 118, 119
 - average, 119
 - recognition, 118, 119
 - average, 119
- acyclic deterministic finite state au-
tomaton (ADFSA), 16, 25–
28, 30, 67
 - additional, 42, 83
 - mixed-context, 85
- Afrikaans noun plurals, 111, 116
- allomorph, 23, 25, 29, 114
- ambiguity, 102, 114
 - count, 89
- analyzer, *see* morphological ana-
lyzer/generator
- analyzer/generator output, 118
- common prefixes, 25, 38, 72
- conc(), 51, 80, 82
- connectionist approach, 11, 13, 14
- consonant insertion, 114
- consonant replacement, 114
- context
 - \Leftarrow rule, 54
 - \Rightarrow rule, 54
 - contiguous, 37
 - growing, 37
 - minimal discerning, 36
 - shortened, 39
 - size, 33
- contexts
 - disjuncted, 51, 117
- correspondence part (CP), 10, 33,
41, 81
- default pair, 9
- default pairs, 48
- delimiter edge, 39

- marked, 41
- directed acyclic graph (DAG), 25–27, 66, 69
 - prefix-merged, 72, 94
 - traversing, 44
- discerning prefix partitioner, 54, 55, 62, 67, 73, 75
 - L-relative, 54, 58, 59, 64, 76, 77
 - minimal, 59, 65, 83
 - minimal, 56, 64, 76
- disjuncted contexts, 51, 117
- disjunction, 105
- dynamic programming
 - algorithm, 17
- edge
 - terminal, 74
- edge count, 26, 27
- edge label, 27
- edge-delimited path prefix set, 75
- edge-delimiter set, 67, 73–76, 80
 - L-relative, 67, 76, 81, 82
 - minimal, 77, 78
 - minimal, 74, 78
- edges, 38
- edit operation, *see* elementary operation
 - eration
- edit sequence, *see* string edit sequence, 49
- edit sequences
 - final, 36, 47
 - input, 47
- elementary edit operation, *see* elementary operation
- elementary operation, 17, 29
 - copying (NOCHANGE), 17, 21, 33
 - cost, 17, 21, 33
 - deletion (DELETE), 17, 21, 33
 - edge labels, 25
 - insertion (INSERT), 17, 21, 33
 - replacement (REPLACE), 17, 21, 33
 - root, 30
- elision, 114
- end of edit sequence (EOS), 37, 47
- English adjectives, 98
- environment (E), 34, 51
 - L-relative, 60, 61
 - minimal, 61
 - minimal, 60–62, 64, 77, 81, 82
- errors
 - generation, 119

- recognition, 119
 - typical, 5
- exclusion rule, *see* rule, exclusion
- failure, 5
- failures, 117
- feasible pair, 9, 10
 - set, 47, 57
- feasible pairs, 38, 47
 - total number, 117, 118
- gemination, 114
- generator, *see* morphological analyzer/generator
- identification number (IDNO), 39, 85
 - groups, 90
- intermediate form, 12
- L-relative discerning prefix partitioner, *see* discerning prefix partitioner
- L-relative prefix partitioner, *see* discerning prefix partitioner
- language analysis levels, 1
 - morphological, 2
 - orthographic, 2, 3
 - phonological, 1, 3
 - pragmatic, 2
 - syntactic, 2
- language independence, 98
- left context (LC), 10, 33, 42, 83, 85
- lemmas, 116
- lexical character, 9
- lexical component (L-component), 41, 48, 91, 110
- lexical letter, 34
- lexical-surface input stream, 33
- lexical-surface representation, 9, 15, 31–34, 36, 47
 - full length, 32
- lexicon, 1, 3, 4
 - morphotactic, 5–7, 14
 - two-level rule, 5
- machine-readable dictionary, 116
- marker pair (MP), 37, 39, 48
- memory usage, 116, 117
- minimal discerning contexts, 67
- minimal edge-delimiter set, 76
- minimal environment, *see* environment, minimal
- minimal L-relative environment, *see* environment, L-relative, min-

- imal
- mix(), 51, 59
- mixed context, 48, 55, 94, 118
- mixed-context prefix, 80
 - set, 60
- mixed-context representation, 36–
 - 38, 42, 51, 59, 83, 93
- mixed-context sequence, 31, 52, 67
 - full, 46
- mixed-context set, 52, 55, 70, 93
 - full, 50, 52, 57, 70
 - shortened, 81
- mixed-context string, 73, 74
- morpheme, 26
- morpheme boundary, 24, 48, 118
 - marker, 28, 33
- morphological analyzer/generator,
 - 1, 5, 7, 11, 100
- morphological level, 2
- morphological processor, 102
- morphology-specific heuristic, 21,
 - 24
- morphotactic component, 100
- morphotactic description, 32, 48
 - left-hand side, 32
 - right-hand side, 32
- morphotactic formula, 31, 106, 111
- morphotactics, 31
- node
 - root, 38, 78
 - terminal, 38, 78
- null character, 92
- operator
 - four types, 10
- optimal prefixes, 25, 26, 28
- optimal suffixes, 25, 26
- optimal two-level rule set, 36
- orthographic level, 2, 3
- out-of-bounds (OOB), 37, 47, 83
- overgeneration, 5, 36, 54, 110, 116,
 - 117
 - elimination, 125
- overrecognition, 5, 36, 54, 117
- path, 39, 69, 78
- path prefix set, 74
- pathprefixes(), 74, 80, 82
- paths
 - set of, 73
- phase one, 31
- phonological level, 1, 3
- pragmatic level, 2
- prefix, 47

- prefix partitioner, *see* discerning
 - prefix partitioner
- prefix-partitioned full-context set,
 - 54
- prefixes
 - optimal, 25
- procedural steps, 66
- reachability, 69
 - sets, 70
- right context (RC), 10, 33, 42, 83,
 - 84
- rule
 - composite (\Leftrightarrow), 10, 34
 - context restriction (\Rightarrow), 10
 - epenthesis, 92
 - exclusion ($/\Leftarrow$), 10, 116, 117,
 - 125
 - general, 33
 - insertion, 92
 - optimal, 33, 42
 - simple, 87, 100, 116, 117
 - surface coercion (\Leftarrow), 10
- rule conflicts, 110
- rule set
 - optimal, 96
- rule type, 33
- rule type operator, 34
- rules
 - final set, 85
 - general, 36
 - least ambiguity, 44
 - minimal context, 103
 - optimal, 32
 - ordered, 11
 - orthographic, 9
 - overspecified, 110
 - phonological, 4, 9, 10
 - redundant, 48
 - rewrite, 11, 13
 - sound-changing, 3, 11, 25, 108,
 - 114
 - spelling, 3, 4
 - two-level, 1, 9–15, 25, 116
 - unordered, 11
- simple rule, 87, 100, 116, 117
- sound changes, 114
 - model, 117
- source string, 33
- source-target word pair
 - single, 35
- Spanish feminine adjectives, 105
- special pair, 10

- set, 48
- special pairs, 33, 48
- special symbols, 37, 38, 47, 70
- spurious INSERTS, 25, 27, 28
- start of edit sequence (SOS), 37, 47
- state
 - final, 38, 78
 - start, 38, 78
- string edit sequence, 16, 29
 - cost, 18
 - edit distance, 18
 - minimal, 19
 - normalized edit distance, 20, 21
 - reversed, 25
 - source-target, 25
- strings
 - over alphabet, 47
- stringset(), 75, 80, 82
- subsumption, 59
 - of environments, 60
- suffix, 47
- suffixes
 - optimal, 25
- surface character, 9
- surface component (S-component), 41, 48, 110
- surface form, 11–13
- surface letter, 34
- symbolic approach, 13
- syntactic level, 2
- target string, 32
- thesis advantages, 124
- training word-pairs, 119
- transitions, 38
- two questions, 34, 52, 61
 - declarative form, 35
 - procedural, 35, 40, 77
 - rule-type decision, 45
- two-level morphology, 3
- typical errors, 5
- underlying form, 11–13
- underlying graph, 67, 73
 - labeled directed, 68
- underlying morpheme, 11, 31
- underscore (-), 51
- understandable mapping, 33
- unmix(), 51, 59, 81, 82
- unseen words, 36, 54, 98, 118
- vertical bar (“|”), 105
- wide-coverage parser, 116

Index

140

Xhosa noun locatives, 46, 101