A Relational Method for the Automatic Analysis of Highly-Inflectional Agglutinative Morphologies

J D Riding

British and Foreign Bible Society
Machine Assisted Translation Team

and

Oxford Brookes University
Pattern Recognition Group

M.Phil. Thesis

October 2007

Acknowledgments

I am grateful to the British and Foreign Bible Society for funding this work. Thanks are also due to the United Bible Societies for providing access to many vernacular corpora. Particular thanks must go to Dr Kees de Blois of the United Bible Societies for his assistance in examining the Bantu analysis and to many colleagues too numerous to mention by name for providing helpful criticism and comment. Lastly, thanks are also due to St Deniol's Residential Library, Hawarden, for the award of the Lawton Scholarship in support of this work which enabled much of this thesis to be compiled in an environment free from distraction.

Abstract

This thesis presents a method for analysing word forms in highly-inflectional and agglutinative natural languages. The method described is language independent and can be run either automatically or as a semi-supervised process. The wider context of this research in providing authoring tools for developing world vernacular languages is discussed. Other work in the field of automatic morpheme analysis is reviewed and its relevance to this method and context evaluated.

Morpheme identification is made by means of establishing the relationship of each hypothesised morpheme with its corresponding stem lemmata. A list of discovered morphemes is generated together with a list of their stems. Inflection paradigms are identified where stems share the same morpheme structures. The process is described in detailed and three example analyses are shown drawn from disparate language groups. The language independent nature of the process is demonstrated. The benefits and limitations of the method are discussed and areas for further work identified.

Contents

List of Tables

Chapter 1

Computers and Highly-Inflected Language

The diversity inherent in the almost seven thousand natural languages presently listed by the Ethnologue [20] presents enormous challenges to an increasingly connected world. The homogenising effect of the Internet is at once at odds with this diversity. This conflict emphasises the need for text processing systems that are able to work with the greatest possible number of natural languages. Those languages coming under increasing pressure include many, indeed the majority, of vernacular languages in the developing world. Many of these languages are highly inflectional in nature. This characteristic poses particular problems for computer based text processing. The sheer number of languages militates against the traditional, knowledge-based approach to analysis.

1.1 Aims of this research

This research seeks to demonstrate that progress can be made towards computer-based, truly language independent methods for analysing form in natural language. The particular context of this work is the field of biblical translation. Within this context there is great scope for computer-based tools to assist translators in their work. Amongst the greatest needs is help in building dictionaries, accidence grammars, and from these, spelling checks and hyphenation tables. The accurate analysis of word form is a pre-requisite for all these tasks. Traditionally such analyses are prepared by human linguists. The information produced is then encoded as a knowledge-base which can be used by programs designed to help the translator.

An alternative approach is to provide a measure of automatic word-form analysis to assist the translator. Some systems exist but are typically focused upon major *lingua franca* where commercial returns from the work can be reasonably expected. This excludes most of the work of the Bible translator. Most Bible translation projects are aimed at a vernacular language. Such languages

rarely have the resources available to develop knowledge-based systems. Likewise, systems focused on a particular language or language group have little general impact. There is a clear need for language independent processing which can assist linguists and translators across the broadest possible spectrum of language.

1.2 Outcomes from this work

This paper describes a process for the analysis of word-forms which is, within certain key limitations, wholly language independent. Most natural languages construct forms of words by means of prefixes and suffixes, often described as affixes, or by changing or adding morphemes within a word, usually termed infixes. Some use a combination of all three. This process seeks to provide automatic identification of morphemes and morpheme structures for any natural language where the principal mechanism for word formation is affixal. Thus any language which modifies words to create surface forms by the addition of one or more prefix or suffix morphemes is a candidate. This research also recognises that the complexities of word formation in natural language are such that solutions which are 100% correct are unlikely. It seeks, therefore, to identify characteristics which commonly present problems for the analysis and to assess the limitations these place on the process as a whole. Notwithstanding these limitations this research aims to generate from its analysis data-sets which can be of help to linguists and translators and which might be used to construct knowledge based systems for a broad set of natural languages.

An alternative approach to the largely statistical basis of most other recent work is proposed. In addition to the initial identification of possible morphemes (hypomorphs), which remains statistical, the concept of the inflection paradigm is introduced and the identification of hypomorphs by reference to their corresponding stems is demonstrated. This approach makes possible the identification of relatively poorly attested hypomorphs. It is also more generally accessible to linguists than the rather more abstract methods such as MDL algorithms.

A secondary aim of this research was to explore how effectively object-oriented (OO) design techniques might be applied to this kind of processing. Most implementations of this sort have hitherto been largely procedural. The opportunity to develop an OO model, the individual components of which might have wider application in linguistic processing, was of particular interest.

1.3 Structure of this thesis

The remainder of this thesis is divided into five sections and a short appendix detailing the principal OO classes developed to build the prototype system. In the Second Chapter I give a brief introduction to the particular context of this research and then present a review of previous work in the area of word form

analysis. This review also includes some of the earlier history of automatic morpheme discovery which has shaped much of the subsequent work in this field. The Third Chapter describes in detail the process which I have developed in the particular context of my own work. The language independent nature of the process is stressed. A description is also given of related work on the assessment of the main type of inflection preferred by a particular language. The concept of the hypomorph is introduced as is its complement, the stem and stem set and finally the idea of the inflection paradigm is explored. It is then shown how these concepts can be used to identify and validate hypomorph structures in any affixal language.

Chapter Four presents the results obtained from the process on three languages. The example languages are drawn from two disparate linguistic groups, two from the Indo-European family and one from the African Bantu group. An example analysis is presented first of all for English to ensure maximum accessibility for these results. Latin was chosen as a much more highly inflected language, sufficiently closely related to English and other Western European languages as to be generally accessible. The Latin results are validated against one of the many more traditional grammars of Latin which exist. Lastly, results for Kiswahili are described. Whilst it is an East African *lingua franca* Kiswahili shares many of the characteristics of strongly affixal and agglutinative vernacular languages worldwide and particularly in sub-Saharan Africa. As a member of the Bantu group it is remote from both English and Latin.

A brief account of the prototype system design is given in chapter 5 including a description of the OO classes developed to provide the necessary behaviours. Chapter 6 summarises the results and identifies the limitations of the process as demonstrated by the three test languages. Current and future applications for this processing are discussed. Lastly, potentially fruitful areas for further research are noted.

Chapter 2

Computers and natural language

2.1 Venturing beyond the developed world

The Ethnologue Catalogue of Living Languages for 2005 [20] lists a total of 6,912 distinct natural languages. Three percent of these (230) are classified as European. Analyses of the natural languages of the world wide web suggest that between 75-85% of all web pages are written in English and about 60% of all search requests to Google in January 2002 were placed using English. Of the remainder only requests placed in German (12%), Japanese (7%), Spanish (6%), French (5%), Chinese (3%) and Italian (2%) amounted to more than 1% of the search traffic at Google[16]. The remaining 8% is attributed to the other 6,905 world languages in the Ethnologue lists. Early developments in IT which were based largely in North America and Western Europe promoted a relatively small number of European languages as *de facto* standards for human interaction with computers. Early systems in effect hard-coded the Latin alphabet as used by modern English onto their keyboards and displays. Given that much of the early use of the Internet was by the academic community, amongst whom this very restricted set of languages form the *lingua franca* for international discussion and exchange, it is not surprising that as IT systems developed they relied heavily on these few core languages.

An examination of the growth of the non-English Internet population is instructive. Whereas the increase in use by Western European language speakers has been steady but slow, the growth in use by Asiatic, Chinese, Japanese and Korean users has been much faster. Outstripping even the Asiatics, however is the growth of usage by speakers of other languages. Sub-Saharan Africa alone accounts for approximately 1,500 of these other languages.

Personal and office computing has become by far the most common context in which the majority of people throughout the world encounter and interact with computer systems. From the first IBM PCs, Apple Macs and Apricots of

the early 1980s with their limited set of applications the growth of PC usage to the present day has been spectacular. In the intervening years one improvement in particular can be identified as having contributed enormously to the widening use of PCs, the development of the Graphical User Interface (GUI). The GUI has shaped the scope and manner of PC use more than any other single hardware or software development. Prior to the advent of the GUI PCs remained the domain of those with the time and interest to learn how to interact with them via their primitive Command Line Interfaces (CLIs). Anyone with any doubt as to how unhelpful and arcane CLIs can be need look no further than current Linux implementations. The CLI limited PC use in two ways. Firstly, the complicated command sets which had to be learned did not encourage non-computing specialists to explore the possible benefits of the PC; and secondly the character-based interface which the CLI enforced severely restricted the type of programs which could be written for the platform. It was certainly possible to write complicated graphical programs (such as Auto-CAD) but since each program writer had to build an interface by which the user could interact at a graphical level with the program, not only did this discourage many programmers from even beginning, those that did inevitably created a whole raft of programs each with its own idiosyncratic way of presenting its own graphical interface to the user.

The advent of GUI systems such as Gem, Mac and finally Windows opened up far wider possibilities for the general use of PCs by providing a common look and feel to all applications irrespective of their tasks. In particular the GUI opened the way for the WYSIWYG concept which has had far-reaching implications for the capture, storage and processing of text in particular. In 1980 type-setting was a skilled process, carried out by professional typographers and compositors on dedicated type-setting engines. By 1990 it was clear that what remained of the industry was doomed and by 2000 virtually all typesetting was being done by computer, usually some form of PCs and almost always via a GUI. Word-processing has become the single most common use for PCs and the processing of text in one form or another consumes more CPU cycles than any other computing activity in the world today.

As text processing has developed authoring software has gradually offered more and more help to the author. For the major *lingua franca* any credible word processor must offer spelling checks, hyphenation and grammar critique as well as a host of relative trivia such as automatic capitalisation of sentence initial words. Providing such assistance for a small group of well-defined *lingua franca* is relatively straightforward. Offering similar help for local vernacular languages is far more challenging. The single biggest difficulty which vernaculars tend to present is the complexity of their morphologies. Complex morphology makes it extremely difficult to build authoring systems using traditional methods such as supplied dictionaries and tables. Typically such knowledge bases do not exist. If systems are to be developed for natural languages beyond those that are currently commercially viable ways must be found to build dictionaries and tables automatically. The two key characteristics required by such a system are:

1. Largely automatic morpheme recognition.
2. Language independence.

The process described in this paper seeks to address these two requirements.

2.2 Recent work

During the last fifty years there has been some considerable interest in the identification of morphemes in natural languages and the generation of word forms. The increasing use of IT in non-English cultures has quickened this interest. The success of many of these approaches has led some to conclude that "the analysis and generation of word forms is no longer an academic research topic"[29, 111]. Nevertheless the continuing interest and progress in this field suggests that there is still much to be done. Where it is concluded that no further progress can be made it seems often to be the case that this is due to limitations of scope rather than to the solution of all the problems associated with the task. This review reports upon the more recent work in morpheme analysis and attempts to identify the scope of the various solutions proposed and where further work is needed.

The last fifty years have seen a continuing interest in attempts to automatically identify components of words in natural language. Some of this work has been geared to syllable recognition, for example the work of Andrew Black at SIL [6], whilst others have pursued the search for morphemes and morpheme structures. These two enquiries are in fact closely related as demonstrated by Zelig Harris as early as 1955 [23]. Much work was done during the 1950s towards the development of a general linguistic theory and in particular a theory for language which would be independent of any particular language or language group. Noam Chomsky stands pre-eminent here [8] but note should also be taken of the work of Charles Hockett, in particular his categorisation of competing theories of grammar. Hockett [25], identifies what were then the two competing theories of grammar, Item and Process (IP) and Item and Arrangement (IA). In fact Hockett also identifies a third system of rather older provenance, Word and Paradigm (WP) [25, 210]. He notes that whilst considerable effort was being put into defining both IP and IA grammars, the traditional WP grammar still needed to be taken seriously. I shall return to this observation below. Hockett's characterisation of grammars as IP or IA is helpful insofar as it can provide a basic categorisation for many of the more recent attempts to identify morpheme structures in natural languages. On closer examination we find that some of these attempts are focused on the identification of the 'process' of deriving forms of words whereas others are geared more toward the recognition and categorisation of the 'arrangements' of words.

In the intervening period since the pioneering work of Hockett, Harris and Chomsky there have been many attempts to create processes for morpheme identification. The last twenty years in particular have seen the development of three, possibly four methods aimed at morpheme discovery. These methods

appear in various guises but in practice most of the attempts to solve the problem of morpheme recognition fall into one or more of these categories. All these methods are aimed at identifying the evidence inherent in a text which testifies to the existence of different forms of a word, typically where each form represents a particular function or functional context. All of these methods rely on one fundamental observation of language: where a language is inflected, we may expect to find that a significant number of those inflections will be expressed as morphemes which will inevitably occur more frequently as N-grams in the text than the corresponding N-grams of their stems. This frequency of occurrence represents a signal which we can identify from the background noise of the text as a whole and which provides a way for us to begin to analyse the components of the words in the text. Any significant body of text will demonstrate this principle and even in a language such as English which has a very low rate of inflection, it is not difficult to construct a single sentence with clear evidence for a common morpheme structure:

> *I was sitting at a table in the window of a cafe, drinking a cup of coffee, talking to my companion and watching the people walking past outside.*

In this example a brief analysis of the N-grams which represent the words generates a strong signal for a final *-ing* morpheme. It is this fundamental characteristic that makes possible the identification of morphemes in a text. We should also note that our use of the word 'text' is significant. Whilst recognising that language is primarily constructed as utterances, this research is focused upon language in the context of text processing software. As text in the vast majority of languages is almost invariably held on computer systems as alphabetic character strings this is the context for this work. Throughout this discussion, unless noted otherwise, only affixal morphologies are considered. Infixal morphologies such as Hebrew or Arabic and some circumfixal morphologies such as Dutch or German are not considered by this analysis.

2.2.1 Methods for morpheme identification

The problem of identifying and categorising morphemes from natural language is traditionally viewed as the establishment of lemmata and their associated word forms. Such identifications and categories must be stored in some way that enables them to be retrieved reasonably efficiently. If this can be done with a reasonable degree of accuracy it is valid to suggest that the information which has been discovered and stored represents a viable model for the morphology of the language under analysis. Recent developments in the field of modeling such systems are often heavily dependent on the work of Jorma Rissanen and his development of the principle of Minimum Description Length [41]. Rissanen has demonstrated that when a dataset can be analysed and that analysis stored, the encoded length of the analysis, provided that the analysis adequately describes the stochastic complexity of the data, is directly related to

the efficiency of the analysis. In other words, the shorter the MDL for a particular model, the more closely that model is considered to represent the reality of the dataset. This principle has been applied in recent years by many working in the field of morpheme identification in natural language. It represents the most common method for encoding and evaluating morphemic information derived from natural language.

An alternative approach is to employ statistics to identify candidate N-grams as morphemes. This approach has fallen out of common use in recent years but examples can still be found. In practice it can be argued that the statistical approach and the MDL approach are in fact different representations of the same characteristics. Both are geared to the identification of the stem/morpheme boundary and whilst the two processes describe their results in different ways they are fundamentally similar. This is noted most clearly by Snover, Jarosz and Brent [43] who comment:

> The use of probabilistic models is equivalent to minimum description length models. Searching for the most probable hypothesis is just as compelling as searching for the smallest hypothesis and a model formulated in one framework can, through some mathematical manipulation, be reformulated into the other framework.

We may, therefore conclude that MDL based methods are largely analogous to statistical methods.

Two other methods remain which I describe fully in the next section. Memory-based algorithms have been developed which perform well but typically require not only supervision of the learning process but also the provision of a predefined lexicon from which the system can learn to recognise the various transformations it may encounter. For example the work of van den Bosch and Daelmans on memory based morphological analysis [7].

Lastly, I identify a category of analysis which has yet to be generally recognised and for which I suggest the label "relational". Two forms of this can be found. Latent Semantic Analysis (LSA) [15] [42] has demonstrated an ability to rival the performance of MDL systems. Most recently good results have been demonstrated by systems which utilise the relationships implied by hypothesised morpheme structures and their corresponding stem lemmata. In such analyses the identification of morphemes is not solely dependent on statistical information or on the neatness of a particular MDL encoding, rather the identification also depends on the relationships which the identification establishes between hypothesised morphemes and their companion N-grams within the word list or text stream. The recognition and evaluation of these relationships gives a very helpful confirmation of the initial hypothesis. I term this technique 'Paradigm Based Analysis' (PBA).

Most of the recent attempts at morpheme identification employ one or more of these techniques.

2.2.2 Common limitations

The breadth of difficulties encountered by many in the pursuit of a truly generic solution to the problem of identifying morpheme boundaries in natural language has encouraged some researchers to adopt an incremental approach. This is most commonly expressed by limiting the analysis to a particular language or language group or to a particular vocabulary within a given language or even a single part of speech. Such restrictions allow real progress to be made in a limited context. Examples quoted below include systems to identify technical terminology or perhaps solely adjectival forms and systems geared to single languages such as Dutch or German.

An alternative to limiting the scope of enquiry for a system is to bootstrap the process by providing some form of lexical database which supplies examples of morphological forms in the language under analysis. Such systems are typically geared to extending the analysis provided in the input lexicon by using that information to enable the words of a wider corpus to be parsed into stem/morpheme pairs. As in the case of limiting the task to specific vocabularies or parts of speech this technique allows significant progress to be made.

Both these problem limitation techniques share the same constraint in that they require significant input from the operator, either via supplied lexica or via the encoding of example rules which can then be used to generate a model for the analysis. For the languages of the developed world this is less of a limitation than for the languages of the developing world.

Many of the systems which have been developed to address the problem of unsupervised learning of morphology are in practice a hybrid between some heuristic ability which is given a 'kick-start' in the form of supplied tables or vocabularies. The system uses this as a basis for its work thus ensuring a foundation for learning which is considered reliable by the designers. The value of this approach is considerable as it removes much of the uncertainty involved in trying to discover morpheme structures without any supplied information. Morpheme identification 'from cold' invariably encounters problems in the area of setting acceptance thresholds for discovered structures. It is generally the case that high thresholds generate limited morpheme sets whereas low thresholds are prone to corrupting the morpheme set with too many poor identifications. The problem of setting acceptance thresholds is fundamental to the success of unsupervised learning of morphemes and it is indicative of the amount of work that remains to be done in this area that so many morpheme discovery systems choose to avoid this problem by making use of pre-defined databases.

2.3 Some examples of successful attempts at morpheme analysis

The examples cited are divided into two broad categories. First I consider systems that are dependent upon some degree of information other than the dataset to be processed. These include systems aimed at technical term acqui-

sition, limited scope analysis (for example attempts to identify particular forms which correspond to certain parts of speech), analysis which relies upon supplied rule sets such as transformational rules and supervised or memory-based systems. It is fair to suggest that all these systems have in common a degree of supervision in their processing whether this is provided by an extensive lexicon of examples, pre-defined rules or a combination of these. Indeed some systems demonstrate a dependence on pre-existing lexica or rule-sets and are geared to providing a limited solution which is effective solely in a particular context of vocabulary or part of speech.

2.3.1 Morpheme identification dependent upon existing rules or lexica

Limiting the scope of the task is clearly a useful tactic to allow the researcher to focus on particular parts of what may otherwise be a very complex problem. The solution of difficulties which create significant problems for the processing can be postponed to a later date and much useful experience can be gained in addressing more tractable issues. There are two contexts in which we see this tactic commonly employed. Language processing of texts dealing with technical vocabularies is a common field for such applications. Such contexts have the advantage that much technical vocabulary is created and used fairly systematically and as such offers a rich harvest for processing, yielding good results as a consequence of consistent usage. Examples include Grabar and Zweigenbaum[21], Paulo, Correia, Mamede and Hagege [39] and Daille [13]. The second context is that of limiting the process not by restricting the input data to a particular context but by focusing the output to a particular category, typically a single grammatical category. Here, information gleaned from a wide input data-set can be focused upon a clearly defined problem and inconsistencies in the input data which have no bearing on the output context can be disregarded. Recent work in this context includes Nakov [38], Theron and Cloete [45] and Yarowsky and Wicentowski [49].

2.3.1.1 Automatic acquisition of technical terms

Limiting the scope of learning to technical vocabularies has the dual benefit of being able to address the problem of word form identification in complex vocabularies which are not well served by standard lexica and at the same time limiting the analysis to a vocabulary which, whilst complex, is often artificial. Much technical vocabulary is constructed from loan words. In the case of western European languages these are most often borrowed from Greek and Latin. The way European languages handle the import of loan words is typically well defined and it is the nature of technical authors who generate these terms to be consistent in the way they handle such words. We can observe that whilst such terms often seem highly complex, in practice their transformation is often highly regular. This plays to the strengths of machine based analysis.

An example of this approach can be found in the work of Grabar and Zweigenbaum [21]. The context for their work is the need to construct dictionaries of medical terms. They recognise and capitalise on the tendency for technical vocabulary to follow fairly consistent constructs by preparing their method with existing vocabulary lists in which related forms are identified and from which common stems and affixes can be derived. They also identify three different forms of morphological change viz. inflection (where variant forms of the same word are created as in the English plural form), derivation (where affixes are added to a root to generate a functionally different form as in forming an adjective from a noun) and lastly composition (in which several roots and possible affixes combine to form a single term as in *haemoglobin*). Whilst human linguists may readily categorise inflection into such categories it is not always clear how these distinctions assist a machine analysis. For instance, the practical differences between derivation and inflection are often difficult to perceive. Where some would suggest that stem modification takes place at the stem morpheme boundary in the case of derivation and not for inflection on closer inspection it often seems more likely that such mutation is for phonetic reasons rather than driven by a change of grammatical category. The identification of composition is however helpful insofar as it identifies a type of inflection which can be extremely difficult for machine analysis to handle. Languages which are truly agglutinative can present difficult problems for automatic analysis. Most linguists regard any language which forms words by the accretion of affix morphemes into structures containing multiple morphemes as agglutinative. I prefer to distinguish these languages (Kiswahili would be a good example) from languages I term as truly agglutinative. Languages of this type have a habit of forming words not simply by stacking affixes around a stem but also by stacking stems themselves together. Consider the German *unterseebooten* where one can identify *'unter'* as a preposition morpheme and '*-en*' as a plural morpheme but it is difficult to regard '*-seeboot-*' as anything other than an agglutination of two nouns '*see*' and '*boot*'. In the case of German this rarely presents too many problems for machine analysis as in common with most Western European languages German has a low rate of inflection and such composites rarely generate forms with medial morpheme accretion. Some other languages however (thankfully relatively few) are prone to this phenomenon to a much greater extent, for example Innuktitut and similar Inuit languages. Such mutation of form is quite simply a nightmare for an analysis engine. Grabar and Zweigenbaum make the assumption that all stems are word initial, reasonable enough for Western European languages. They prepare their process with pre-existing vocabularies which identify related forms of the same term and limit their analysis to the identification of nominal forms. As their principal interest is dictionary building, this makes a lot of sense as citation forms in Western European languages are almost invariably nominal and, indeed, nominative. In reviewing their results they note that the lists of morphemes they derive could probably be normalised to reduce the size of the lexicon.

Paulo, Correia, Mamede and Hagege [39] address a similar requirement. Their Automatic Term Acquisition (ATA) system is designed to identify tech-

nical terms from a text. They define *term* as a noun or noun phrase and so the scope of their system in some ways goes a little beyond simple form recognition in that it attempts to deal with phrases as well as single words. ATA begins by lemmatising the text using prepared external lexica. The lemmatised text is then passed through a rule-based processor that generates candidate terms on the basis of statistical significance and conformity to supplied morphological and syntactical rules. This part of the process is able to identify phrases as well as single nouns. High frequency terms are typically identified statistically whilst the supplied rule-sets serve to help the system identify low frequency terms. ATA is clearly a useful tool for index and dictionary building but its usefulness in the task of language independent processing or unsupervised learning of morphology is very limited.

A further example not only of limiting processing to the acquisition of technical terms in a text but also limiting that acquisition to a single part-of-speech (PoS) in a single language can be found in the work of Beatrice Daille at Nantes [13]. Her process sets out to identify French relative adjectives in technical vocabulary closely related to French agricultural terms. She begins with a supplied corpus fully tagged for lemmata and PoS. Rules are then derived to govern the generation of relative adjectives from cognate nouns in a similar manner to the work of Theron and Cloete [45] (below).

2.3.1.2 Limited solutions

Nakov [38] limits his processing to a single language – in this case German – and within this scope to nominals for which he seeks gender, case and number. He further limits the task by providing a morphologically tagged corpus as training data for his system. This is supplemented by a lexicon of known stems which his process expands into a set of declension tables as forms are identified from a wider corpus. Similarly, Daelemans, Berck and Gillis [12] restrict their processing to deriving phonological categories for Dutch.

More ambitious is the work of Theron and Cloete (Stellenbosch) [45] who, whilst focusing on a particular transformation, demonstrate that their process is clearly language independent. Fuller details of this work appear below in the context of systems which use transformational rules to identify morphemes. Their focus on the identification of plural forms from a list of singular citation forms, based upon the provision of an example set of similar singular/plural transformations is nevertheless a good example of the value of restricting the scope of a process to allow a tight focus on a particular problem. In other ways, however, Theron and Cloete broaden the application of their work by demonstrating good results for a number of languages drawn from different language groups and in generating the transformations they apply from example word form pairs. Nevertheless the transformations they generate are derived from lists of example word pair transformations.

The validity of the limited approach is further justified in the attempt to identify supposedly irregular forms of inflection as in Yarowsky and Wicentowski's [49] work on irregular forms of inflection as, for example, in make/made

as opposed to walk/walked. Clearly the task of identifying related forms becomes very much more complex where stem mutation occurs. In the majority of languages such mutations occur at morpheme boundaries, most often at stem/morpheme boundaries as in the example above. Stem mutation creates huge problems for stem pattern recognition and requires particular techniques to identify the apparently irregular forms created. Such circumstances require the use of supplied tables to enable the system to identify correctly irregular forms based upon common phonemic shift, semantic similarity, ready lemmatised word lists and like patternings from a related language. Yarowsky and Wicentowski apply these techniques with success to identify irregular forms which are then weighted for validity on the basis of the vowel and consonant transformations necessary to agree a match. Thus, not only are irregular forms identified, the transformation rules which govern their mutations are also found. This is an effective solution to a particularly difficult part of the problem of morpheme identification and necessarily relies heavily on supplied knowledge of parts of speech, a pre-lemmatised dictionary, a phonemically tagged alphabet and seed data of similar mutations in a closely related language. Results are good but the heavy dependence on supplied knowledge makes this an expensive solution to apply generally. The authors' description of their method sums up the approach well:

> the *nearly* unsupervised induction of inflectional morphological analysers, with a focus on highly irregular forms. (My italics).

Such approaches allow, indeed rely, on the encoding of a certain amount of knowledge upon which the process can rely to make its decisions. This is a perfectly valid principle and is particularly effective in contexts where there is a large body of information which describes the accidence of the language in question. Where little or no information is available as a pre-requisite the approach fails.

2.3.1.3 Morpheme identification or generation via transformational rules

There is a strong trend in recent research to utilise the work of Koskenniemi [33] and Sproat [44] by providing two-level morphology rules which describe the changes from base to surface forms of words. Such rules provide a reliable means of identifying relationships between different forms of a word as found in text. Many processors for two-level morphology rules have been built, notably PC-KIMMO [2] and KGEN [35] by SIL. These systems typically require predefined rules which describe the transformations that occur within the words in a text. As such they are at one and the same time extremely useful in providing guaranteed rules which can be applied to pairs of words to identify valid relationships of form. Nevertheless they require an existing rule base which can be applied to the task. For those seeking truly unsupervised learning this is a disappointment. For those seeking language independent systems an existing knowledge base for each language is a pre-requisite.

The value of related form identification via rules of transformation is undisputed, and the work of Theron and Cloete [45] (see above) has demonstrated that such rule sets can not only be utilised but also be generated, at least in part, automatically. This work, based at Stellenbosch University, is noteworthy as it demonstrates automatic processing of a non-European language – in this case Xhosa – in addition to Afrikaans. Whilst the emphasis is on the discovery of transformation rules to describe the word formation changes for secondary forms of words (in this case *citation* ⇒ *target form* transformations), such processing is potentially of great benefit to those focusing on morpheme and stem identification. Although the initial partitioning of words into stem/morpheme pairs can be informed by the addition of transformation rules, the degree of human intervention required is such that the task becomes impractical as anything more than an exercise in the context of single language processing. Theron and Cloete demonstrate that given an initial identification of stem/citation form and target form it is possible to construct a system to map the transformations which occur in rewriting the source form as the target. This mapping can then be applied to other words in the language to predict similar transformations. Such processing offers the possibility of building upon an initial identification of stems and morphemes and extending that analysis into terms which are less well attested in the dataset but which nevertheless exhibit valid transformations. It is clear that for rule induction to be a reliable process in this context, a set of word pairs which exhibit the target transformations must be provided. Theron and Cloete provide this data from machine readable dictionaries and hand-encoding of suitable *source* ⇒*target* word pairs. As one might expect, the performance of the system improves the more examples it is given from which to learn. More than 93% of transformation predictions were successful in Afrikaans, given a 4:1 ratio of learning data to output data. The basic technique used by the system is to construct a character by character edit sequence to record the correspondences and changes between the source and target letter forms of each word pair. An acyclic finite state automaton (AFSA) is then constructed to process the edit sequences. This AFSA is viewed as a directed acyclic graph (DAG) in which insert operations common to a significant number of the input edit sequences are likely to identify stem/morpheme boundaries. Boundary mutations are identified as such by their lower occurrence threshold and the necessary rewrite rules are constructed to describe them.

The principal benefits of this work to morpheme identification are two-fold. Firstly it demonstrates that common principles exist which can be applied to languages from greatly dissimilar language groups.[1] Secondly the ability to extend the analysis to incorporate terms of statistically low significance is essential to provide the fullest possible analysis. There is scope for further development of this work in the induction of word formation rules derived from automatically generated training data.

Other systems which aim to identify transformation rules for inflections in-

[1] Afrikaans is part of the Indo-European group whereas Xhosa belongs the Bantu group, most particularly 'Click' Bantu. Source - Ethnologue http://www.ethnologue.com/show_language.asp?code=XOS - 10:03 11th April 2005.

clude Daille [13] and Yarowsky and Wicentowski [49] (above) and examples can be found of systems which aim to encode known transformation rules and apply them to wider corpora to identify word components. These include Antworth [2] and Miles [35]. The work of George Kiraz [30] [31] [32], whilst not directly relevant to a discussion on affixal language morphology deserves mention here. His SEMHE system demonstrates clearly the ability to parse Syriac and Arabic words with a high level of accuracy based upon the application of transformational rules. So far as I have been able to discover this work represents the only sustained attempt to handle non-linear morphology since the work of Beesley on Arabic [4] [9] in the early 1990s.

2.3.1.4 Memory based parsers

The final category of processes that can be considered in the overall context of systems dependent on existing rules or lexica are memory based parsers. The work of van den Bosch and Daelemans at Tilburg [7] gives a good example of this approach. Memory based morphological analysis (MBMA) is clearly heavily dependent on supplied knowledge. In this case the authors, working only in Dutch, use the CELEX lexical database as a source of example data. This database contains a full morphological tagging for Dutch and provides many instances of word form transformation as the result of morphological change. Using this data MBMA can partition new word forms and assign them to the correct syntactical category as defined by CELEX with a high degree of accuracy.

The analysis and application of transformational forms which underpins many of the systems discussed above suggests that it is valid to consider such solutions as part of the IP (Item and Process) category of grammars identified by Hockett [25].

2.3.2 Morpheme identification without the use of existing lexica or rule-sets

The work reviewed to date has in common the decision to manage the process of morpheme identification by utilising lexica or other pre-existing rule sets. It is clear that this approach is effective and generates useful results. It is equally clear that such processing is dependent upon the availability of linguistic databases from which the necessary information can be drawn to allow such systems to acquire the knowledge needed to successfully identify morphemes and morpheme structures. There are however, systems which require no prior knowledge of the data to be processed. These include work of Goldsmith [19], Creutz and Lagus [11] both of whom utilise Minimum Description Length processing, that of Schone and Jurafsky [42], Monson et al., [36] [37] and my own work described below.

2.3.2.1 Minimum Description Length parsers

Foremost amongst those adopting this approach is Goldsmith's *Linguistica* system [19] the scope of which has thus far been geared to morpheme identification in European Languages with lower inflection rates. This excludes languages such as Hungarian, Finnish, Kiswahili and other highly inflectional languages. Despite this restriction Goldsmith declares his longer term goal to be "the treatment of unrestricted natural languages" [19, 153]. The only pre-requisite for *Linguistica* is a corpus of text in the target language. Using fewer than 5,000 words seems to limit the analysis significantly although we should be wary of identifying such minimum thresholds as it seems likely they are ultimately dependent upon the rate of inflection exhibited by the target language. Thus, unlike the examples discussed above *Linguistica* takes no input from lexica or from morphological transformation rules. The goal to which *Linguistica* aspires is "to produce output that matches as closely as possible the analysis that would be given by a human morphologist". Longer term intentions for the project are to provide data for a grammar acquisition system [19, 154].

The basis for the discovery of possible morphemes is the use of minimum description length (MDL) as a tool to identify the description of the morphological components present in the corpus which exhibits the least redundancy. An MDL is often the tool of choice for such systems and gives good results. However, Goldsmith claims that *Linguistica* goes beyond this analysis in its ability to identify "signatures" (the term he uses to describe inflection paradigms) and in its success in modeling the process followed by a human morphologist. These goals are achieved by a process that falls fundamentally into two phases. In the first phase of the processing an attempt is made to partition each word in the corpus into stem+morpheme.[2] Each possible partition is tested against the MDL which is continually refined as more examples are parsed. The second stage of the process is to group the detected morphemes into *signatures* by virtue of common stem association. This stage is, so far as I have been able to discover, common only to *Linguistica* and my own work in this area. Goldsmith's results and my own for English for example, are very similar even to the type of errors which are generated when the process fails to partition correctly. In his introduction Goldsmith helpfully identifies four systems for discovering morphology including his own. He first cites Zelig Harris who proposed morpheme identification by marking stem/morpheme boundaries[23, 24]. Identification by *tri*-gram and *di*-gram analysis forms the second approach [19, 156]. The third is characterised by the work of Jacquemin and of Gaussier. Gaussier [17] sets out to acquire derivational rules from a supplied analytical lexicon in a process which appears to be related to Theron and Cloete above. Jacquemin [26] uses proximity within text stream to hypothesize related forms of a single stem. This has been further developed by Schone and Jurafsky [42], see below.

[2] It should be noted that complex and agglutinative morphologies are not well served by this approach and that the model also admits the possibility of a stem+0 partition. Both these issues are indicative of the degree to which *Linguistica* is, possibly inadvertently, tailored to Western Indo-European languages.

Unlike the majority of methods discussed in this review, Goldsmith not only suggests that his system should give good results across a range of natural language, he demonstrates it with analyses of English, French, Spanish, Italian and Latin corpora. This illustrates both the flexibility of his approach and its limitation, at least currently, to western European languages and particularly to the Romance group. Of all the systems reviewed in this chapter, *Linguistica* gives the most generally useful analysis within the limitations of its target language group.

Recognising the limitations of Goldsmith to Western European languages, others have sought to extend his work into more complex morphologies. Noteworthy amongst these are Creutz and Lagus [11] who have sought to apply these principles to Finnish – a notoriously complex morphology, at least by European standards. In their work on Finnish Creutz and Lagus use the same basic technique as does Goldsmith together with their own implementation of Sequential Segmentation and Maximum Likelihood (ML). Creutz and Lagus set three criteria of evaluation: (1) correspondence with linguistic morphemes, (2) efficiency of data compression and (3) computational efficiency. Their results for English are very close to *Linguistica* and their MDL implementation performs almost identically although *Linguistica* outperforms the others in terms of time taken. The size of the codebooks generated by the various systems is, however, interesting. *Linguistica* generated a codebook between two and three times the size of that generated by the Creutz and Lagus MDL/ML systems. Given the closeness of the results one might conclude that codebook cost is directly related to computing time. Put simply, *Linguistica* trades size of codebook for computing efficiency. For Finnish, the ML method returned better results (just) than either of the other two. Creutz and Lagus suggest that this is a valid measure of the system's ability to 'generalize the analysis of unseen word forms'. Both the Creutz and Lagus methods are prone to excessive word splitting whereas *Linguistica* is more likely to miss valid partitions.

The emphasis upon discovering and cataloguing the arrangement of stem and morpheme is reminiscent of Hockett's IA (Item and Arrangement) grammar class.

2.3.2.2 Relational parsers

Within the context of unsupervised morphology analysis there remains a subcategory which has not yet been discussed. I term these systems 'relational' parsers as they share an ability to identify morphemes by noting the relationships between words indicated by the patterns generated by morphemes. The work of Schone and Jurafsky [42] seems to me to fall into this category although it differs from other relational approaches by Monson, Lavie, Carbonell and Levin [?] and myself. Whilst it might be argued that relational parsers fall into the IA grammar category as proposed by Hockett I would like to suggest that in fact they are more akin to the traditional Word Paradigm (WP) type of grammar. Their use of paradigm membership as a strong indicator of relationship goes beyond the recognition of components within a single word form.

As such these systems are likely to have more scope for expansion into general descriptions of morphologies.

Whereas the various attempts at unsupervised learning listed above are all based largely on their ability to recognise and, at least to some degree, categorise contiguous *N*-grams of characters, Schone and Jurafsky have adopted a different approach to the task. They note that systems such as those discussed previously, which rely almost wholly on statistical means to identify stems and affixes, are prone to counter-intuitive errors such as partitioning 'ally' into 'all-y' and failing to partition 'dirty' as 'dirt-y'. This, they suggest is due to the absence of any form of semantic context with which to inform the analysis. Following the work of Deerwester and Landauer et al. [34, 15], they demonstrate that the results which can be gained using Latent Semantic Analysis (LSA) alone can be as successful as a statistically based system such as *Linguistica*. The method has four components: (1) Suffix inflection is assumed so words are inserted in a trie[3] and the significant branch positions noted. These branches are assumed to be stem/morpheme boundaries. Schone and Jurafsky set an arbitrary threshold to select only the best attested affixes. If prefix data is required this is found by reversing the dataset and re-running the process. (2) Having identified sets of inflections (signatures in Goldsmith's terminology) a set of morphological rewrite rules is derived for each pair of word forms sharing the same stem. (3) Each pair of words is now tested for shared semantic context. This is achieved on the basis of frequency within the corpus and position offset from one to the other within the text. Word pairs which share similar 'Semantic Vectors' are flagged as probably related. Dissimilar vectors indicate a lower likelihood that there is a semantic relationship between the two words. This data can be used to adjust the results from statistical analyses to avoid some of the otherwise glaring errors which occur. Schone and Jurafsky consider LSA to be complementary to purely statistical approaches.

Most recently Monson, Lavie, Carbonell and Levin [**?**] have proposed a new approach to the problem of identifying morphology by inflection class. Their work is geared to the vernacular languages of South America in pursuit of the goal to develop MT systems to translate between pairs of languages where one is resource rich (for example Spanish) and the other is what they term low-density i.e. resource limited. They base their approach on the attempt to identify inflection classes as evidenced by a target corpus. They cite the work of Harris [23], Hafer and Weiss [22] as foundational to their method but note their process is limited to suffix based languages. Their method partitions every word at every possible boundary point (i.e. between all letters) and selects the boundary that gives the best result. At present this selection is tuned by hand until the best results are achieved. Their results for Spanish are encouraging and they hope to apply their method to more languages to expand the system's capabilities.

The last example of a relational approach to morpheme identification is my

[3]trie: A data-structure used to represent a hierarchic structure within which each node, unlike a binary tree, may have more than two children.

own work. I describe in detail the process I have developed in the next chapter.

Chapter 3

A relational approach to morphology

3.1 Pre-requisites

The two principle requirements for this process were that pre-requisites should be kept to an absolute minimum and secondly, that the process developed should be applicable to the widest possible set of languages. The principal limitation of scope is the restriction to languages which create word forms by the addition of one or more prefixes or suffixes. The context for which the outcomes from this research are intended is to provide assistance to the Translation Consultant charged with overseeing the work of translation teams who are translating into a language which, more likely than not, the consultant does not know. Thus the preferred scenario is that nothing need be known about the target language other than that it conforms to the general profile associated with this processing. The single critical requirement is that a sufficient body of text exists which will generate a word list large enough to display most of the different types of inflection present in the language. This research has been carried out using wordlists which have been generated from Bible translations. A corpus of text of the extent of the Bible can generate a wordlist of anything from 12,000 unique words to in excess of 100,000 words, dependent upon the rate of inflection for that language and, to a lesser degree, the nature of the translation. It might be imagined that the nature of a translation of a corpus as extensive as the Bible would have little effect on the size of the word list for that translation. This is not so. Modern translations which follow the principle of dynamic equivalence can often use a surprisingly small vocabulary set, particularly where the translators have consciously set out to generate a text which is accessible to the widest possible constituency. The inevitable consequence of this generally laudable desire is the use of a vocabulary set which represents the lowest common linguistic denominator for the target constituency.

3.1.1 Deriving a wordlist

Modern vernacular translations of the Bible are created by translation teams representative of the main Christian denominations amongst the target constituency. Typically these teams will be drawn from priests, ministers and theologians who are mother-tongue speakers of the target language. In most cases a translation is sponsored by the local churches working with and through the national Bible society. The local translators are supported by translation consultants attached to the local Bible society who provide expertise in Bible languages, biblical studies and formal linguistics. The consequence of this model is that these translations are well grounded in modern linguistics and principles of translation as well as the perhaps more traditional strengths of sound theology and acceptable churchmanship. At a practical level, new translations are created with the aid of an authoring system named Paratext [47], which provides a text processing environment, specifically geared to Biblical text, within which the translators can work. Amongst the many benefits this brings is the ability to generate a wordlist from a translation very easily using Paratext.

3.1.1.1 Biblical text as the corpus

Using a corpus such as the Bible as a source of representative wordlists has both advantages and disadvantages. The existence of a common authoring platform avoids many of the problems associated with extracting text from non-standard corpora. Some of the languages at which this process is aimed do not yet benefit from agreed character sets. If they do have an internationally recognised character set it is possible that there is no agreed encoding for those characters. Whilst the Unicode definition has provided internationally agreed encoding for many languages there are as yet few vernaculars for which collation sequences and even punctuation and case shifting conventions have been agreed. Whilst such issues are in some ways trivial, a practical language independent solution must deal with the variants it encounters. The wide use of Paratext by the Bible translation community makes this much easier to achieve.

More critical is the concern that any corpus geared to a particular specialism or cultural sub-group may not be representative of the general use of language for that wider constituency. This is a valid concern. Biblical translation has changed greatly during the last fifty years. In the first half of the twentieth century and before the emphasis was often on the creation of a 'formal' and 'dignified' translation of a sacred text. Such translations often strove to represent as closely as possible the words of the original text, sometimes at the expense of understanding.[1] This is now much less common. For modern translators the emphasis is usually on 'dynamic' or 'functional' equivalence [48]. This approach requires the translators to try and find ways of expressing the intent

[1] For an example see the sixteenth century English translation of the book of Psalms for Henry VIII's Great Bible by Miles Coverdale, Bishop of Exeter which contains the phrase, "I am become like a bottle in the smoke". As a fairly close word for word translation of the Hebrew this is good. As a means of conveying the image "I feel dried up like a smoked wine-skin" it perhaps leaves something to be desired.

of the original text in a manner that is accessible to their constituency. As a consequence of this, modern translations tend to use far fewer 'technical' religious terms, preferring instead to express these concepts in language as close to everyday usage as they can. In practice it is likely that a modern translation will offer fewer distortions of vocabulary or syntactical form. There is, however, one area where care must be taken to remove, so far as is possible, a potential source of confusion.

Proper names do not typically translate but are more often transliterated. For example the Hebrew 'Abimelech' is simply reconstructed phonemically rather than translated as "my father is King". Over the extent of a corpus the size of the Bible this generates several thousand words which are likely to carry potentially misleading morphological characteristics. If, for example, we were to process an English text we would find that amongst the strong signals for English morphemes like *-ed* and *-ing* there would be a similarly strong signal for *-ah*. This is generated from the many Hebrew names which, in their transliterated English form, end with this phoneme.[2] To minimise this confusion I remove as many proper-names as possible from the wordlist prior to processing.

3.1.1.2 Other corpora

Since the context of this work has been primarily the field of biblical translation I have had no need to seek alternative sources. There is, however, no reason at all for this processing to be limited to word lists derived from biblical text. Alternatives might be newspaper or magazine archives and possibly records of judicial or parliamentary proceedings. Clearly, some of these might also raise

[2] As in Isaiah, Jonah, Hezekiah, Judah etc... Proper-names generate a particular area of confusion for the analyst of word-forms. Given the narrative nature of much of the Bible, proper-names, whether of people or place, are common within these narratives. Translators will in general strive to cast the original text into the form and vocabulary of the target language. This cannot usually be achieved with proper-names. In the vast majority of cases translators will render names into their language by simple transliteration or by the adoption of an equivalent existing form. An example of the former would be the name *Abram* (Gen 12). In the 1960 Swahili Union Version of the Bible this is rendered into Kiswahili as *Abramu*. More or less a straight copy of the name as rendered in the English Authorized and Revised Standard Versions of the Bible, both of which were a major influence on the translators. Other than the addition of the final *u* to conform to Kiswahili's preference for open final syllables the name is unchanged. By contrast an example of the latter principle in action is found just a few chapters later in the treatment of the name *Abraham*. In this case *Abraham* becomes *Ibrahimu*. The translators recognized the strong Arabic/Quranic influence in East Africa and opted to preserve the common identity of the Patriarch in both Christianity and Islam by creating a Kiswahili form of the Arabic *Ibrahim*.
In addition to these inconsistencies there remains the vexed question of whether a proper-name should decline as do other regular nouns in the language. This question has received various answers since the earliest translations of the Hebrew scripture into Greek and in the rendering of Hebrew names in the Greek texts of the New Testament. Generally speaking, number is observed as in *Amalekite* $\Rightarrow$*Amaleki* and *Amalekites* $\Rightarrow$*Waamaleki* from English to Kiswahili. Other functional inflections of case are less regularly observed. In short, it is difficult to identify a common standard for the usage of transliterated proper-names. In order to remove a source of possible confusion from the analysis, proper-names are removed from the generated word list prior to processing.

the question of non-standard technical vocabulary and usage but provided a sufficient corpus exists the process should perform equally well.

3.1.2 The Prototype System

Just as translators are seeking the widest accessibility within their target constituency, so the prototype implementation has been designed to be operable within the widest possible platform set and using the minimum of hardware and software resources. The potential users of this are often working with outdated hardware and have no funds to spare to purchase state of the art software platforms. To address these issues I have written the prototype software in Java 5. In addition to the Sun Java 5 Run Time Environment it also utilises the IBM International Components for Unicode library and the XStream Java object to XML serialiser from codehaus.org. All of these systems are offered freely under public licences by the Open Source community. The IDE used to create the code was NetBeans 5.5, an Open Source IDE sponsored by Sun and also available under the GNU public licence.

A major element of prototype development was the creation of an object framework for the principal software components of the system. Whilst this framework remains a work in progress much of it is now well-defined. Details of the system objects can be found at Appendix A. The development work was carried out on a P4 2.8 GHz system with 1 GB of memory running Windows XP. Further testing was carried out on a similar system under Linux (Red Hat Fedora Core 5). The cross-platform nature of the Java code ensures the system is equally at home running under Windows, OSX, Linux, Unix or any other operating system that can provide a Java 5 JRE. It is hoped that the system may be made available in the future as a web service.

Java proved well-suited to the needs of the project; in particular the close affinity to Unicode ensured that working with non-Roman scripts has not presented any real problems other than the difficulties of presenting the results via LaTeX. The maturity of Java ensured that where support was needed it was readily available from the Java community.

3.1.3 Process parameters

There are a handful of parameters which must be decided before the main part of the process can run. These are: Locale selection, a minimum stem length, initial morpheme evidence threshold θ_{morf} and resource allocation.

1. Locale selection. The ISO defines a number of 'locales' which are supported by most operating systems. Locales include information about preferred number formatting, date and currency display and the preferred collation sequence for that locale. It should be noted that such data is locality rather than language dependent. There are for instance a number of different French locales dependent upon locality - fr-FR, fr-CA, fr-BE etc. In the case of many vernacular languages locale tables will not exist

but it may be that a locale has been defined for another language with similar characteristics. In this case it may well be helpful, particularly when displaying results, to ensure that the data appears in a form and order which the user can readily interpret.

2. Minimum stem-length. Any attempt in an affixal language to partition a word in search of a morpheme structure will hypothesize not only a morpheme structure but also a stem. To minimise the processing required it should be possible to ignore words which are considered so short as to be unlikely to generate any useful morpheme hypotheses. I do this by setting a minimum stem-length. Words which have fewer characters than this are less likely to offer much useful data and are disregarded.

3. Initial morpheme evidence threshold. When seeking character strings which may be considered to be morphemes or morpheme structures it is necessary to set a threshold of evidence below which such strings are discarded. I have spent much time experimenting with this threshold which I call θ_{morf} and which I describe in detail below. For a wordlist derived from a complete Bible text experiment has led me to set this threshold at 128 occurrences within the wordlist.

4. Resource allocation. Partitioning lists of N words with an average length of c characters where the minimum stem is set to four characters and the minimum morpheme length is set to two characters generates $N(c-5)$ partitions. For a wordlist of 100,000 entries this can result in approaching 1,000,000 objects. Many of these will be discarded or combined at later stages of the process but they must be created initially. Experiment has demonstrated that this system, coded in Java, requires 256 Megabytes of memory for its VM when processing highly inflectional languages. In practice this means that machines with less than $1/2$ Gigabyte of memory will struggle to manage the process.

3.1.4 Assessing the inflection rate

One of the primary goals for an analysis of word-formation in a language must be to establish first into which of the principal inflection categories it falls and to determine its general rate of inflection. The three types of inflection category: prefixal, suffixal and affixal are described in detail below. Establishing the rate of inflection present in a language can be difficult, not least because of the subjective nature of many measures. An objective assessment is required, based upon a common set of evidence for all languages. Finding a common evidence set for something as diverse as natural language does not sound easy but, happily, the overall context of this work supplies one. The corpus of scripture which comprises the Christian Bible is an ideal basis from which to assess the inflection rate of a language. Clearly some form of base measure must be established as a yardstick for comparison. In order to reflect the relative rates of inflection this

key measure should be both widely accessible and have itself a relatively low rate of inflection. The obvious candidate is English.

Listing all the words found in a modern English translation of the Bible[3] generates a table of roughly 12,500 unique surface forms, disregarding proper-names. Taking an English word list drawn from scripture as a notional 'standard' with regard to inflection rates offers the possibility of comparing other languages via word lists drawn from the corresponding translations of scripture. There is a temptation to try and complicate matters by defining a set of 'core' grammatical categories which we might expect to find and then compare the number of surface forms required by the target language to generate those items. This has two drawbacks. Firstly, defining a language independent group of categories is not straightforward. Intuition might expect most languages to share basic abilities but this is by no means guaranteed. For example, English mother-tongue speakers are often surprised to discover that other languages can denote not just 'one' and 'many' but perhaps 'one', 'two' and 'many'.[4] Conversely they may be taken aback on finding that Ancient Coptic has no direct equivalent of the passive voice. Such differences make it difficult to construct a set of universal 'core' items which might be used as a basis for comparison. Secondly, any sub-setting of the word list generated for a target language requires knowledge of that language. As stated above, an important pre-requisite for this work is the need for language independence in method. Without extensive prior knowledge of the target language the word list cannot easily be subset to identify 'core' categories.

Given this limitation, the assessment of inflection rate is reduced to monitoring the size of the word list generated for the target language. The validity of this measure is good. Since the list is drawn from a parallel corpus of text to that of the English base, we may expect that both texts will express the same semantics via the mechanisms of their respective syntax and morphologies. Clearly there may be differences in the size of the stem lemmata lists for different languages but generally the premise that a larger word list will reflect a more highly inflected language is likely to be a good guide. Differences in the number of homonyms and synonyms are likely to cancel themselves out. If we apply this measure to a set of test languages we discover the following relative inflection rates:

Table 3.1: Example inflection rates for some European and African languages.

Language	Language Type	Size of WL	Rate
English	European	12825	1.00
Kalenjin	Nilotic	26241	2.04
Kinyarwanda	Bantu	70006	5.46
Kiswahili	Bantu	40790	3.18
Latin	European	46659	3.64
Kikamba	Bantu	47604	3.71

[3] For the purposes of this work the corpus of the Bible is defined as the sixty-six books of the Old and New Testaments and the seventeen books of the Deuterocanon commonly present in the Catholic, Greek and Russian Orthodox and Protestant Canons viz: Tobit, Judith, Greek Esther, Wisdom, Sirach (Ecclesiasticus), Baruch, The Letter or Jeremiah, The Song of the Three Young Men (Benedicite), Susanna, Bel and the Dragon, 1st and 2nd Maccabees, 1st and 2nd Esdras, Manasseh and Psalm 151.

[4] For example both Archaic Greek and Hebrew can make the distinction between one, many and the 'dual' form of a word.

Since much of the processing is highly iterative an indication of the inflection rate relative to a known base is helpful in predicting the time and resources needed for the process.

3.1.5 Calculating the primary inflection vector

For the purposes of this study the scope of investigation has been limited to languages which utilise affixal modification of form as their principal means of morphing stem lemmata into the surface forms required by a particular context. Implicit in this limitation is that no attempt is made to identify any form of infixal change, modification due to vocalic or consonantal mutation, crasis, elision or any other form of word-internal reshaping. Given the context of a linear language stream it is clear that only modifications formed by the addition of characters at word-initial or word-final positions fall within the scope of this study. Word-initial modification is generally termed 'prefixal', word-final modification is termed 'suffixal'.

For languages which are primarily affixal there are three possibilities as to how surface forms of words are formed. Some languages tend to prefer creating forms by adding or changing prefixes,[5] some prefer to do this using suffixes,[6] others use suffix and prefix transformation equally[7]. I term this preference the Primary Inflection Vector or *PIV*. In order to give the system the best chance of discovering useful information it is better to first process at the *PIV* and when that analysis is complete to then move to the secondary vector. Where there is a clear preference indicated by the wordlist data it is possible to detect this and set the *PIV* automatically. Languages which do not exhibit a clear preference may require user confirmation of the *PIV* although in practice it is probably unnecessary to do this. An arbitrary decision will suffice. Natural languages that prefer to utilise affixal strategies for word modification can be cast into three types:

1. Primarily Suffixal Languages. This category includes the majority of European languages. English is an example of a primarily suffixal language albeit with a generally low inflection rate. Other European languages which can be categorized similarly include the Romance group (French, Italian, Spanish, Portuguese and Romanian) amongst whom the inflection-rate is significantly higher but where the preferred mode of inflection is strongly suffixal. The common ancestor of this group, Latin, has identical characteristics but exhibits a much higher rate of inflection.

2. Primarily Prefixal Languages. Whilst it is true to say that most European languages have some evidence of prefixal inflection (probably the most general example is the alpha-privative negation prefix in its various forms -

[5] Examples include Kiswahili.

[6] as does English and most other European languages including Latin.

[7] This is less usual in Indo-European languages but very common elsewhere. For example Kinyarwanda from the Bantu group.

a-, i-, im-, in-, un-, etc...) other fairly common forms of prefixal inflection in European languages include tense modifiers as in German *ge-* (e.g. *geboren*) or the less commonly used English forms *be-* (e.g. *bethought*) and *en-* (e.g. *encapitaled*). Classical Greek also uses common verbal prefixes, most notably in the tense marker ε and $\_\varepsilon$ where _ represents consonantal reduplication in the perfect form of the verb (e.g. $\mu\varepsilon\mu o\rho\phi\omega\mu\alpha\iota$).
To find strongly prefixal languages it is necessary to look beyond the European group. It might be argued that the Semitic languages exhibit this tendency, particularly Hebrew, but the strength of the infixal modification present excludes them from the immediate field of study. One of the strongest candidate groups which demonstrates primarily prefixal inflection is the Bantu group of languages in sub-Saharan Africa. Originating in West Africa this group accounts for much of the diversity in vernacular language in sub-Saharan Africa. The Bantu migration has carried these languages across a broad swathe of the continent from their origins in West Africa through Central and East Africa as far South as Matabeleland and even the Northern Veldt. Probably the most easily recognized example of the group is Kiswahili, the *de facto lingua franca* of much of East and East Central Africa. Just as Latin exhibits a strong preference for suffixal inflection, Kiswahili exhibits a similarly strong preference for the prefixal form. In this Kiswahili is typical of many Bantu languages and whilst it appears to share a rate of inflection roughly equivalent to Latin it should be noted that there are many other Bantu languages with significantly higher inflection rates. The apparent equivalence of Latin with Kiswahili was a surprise and suggests that a comparison of lemmata lists might show that the Latin text used a wider general vocabulary than the Kiswahili.

3. Generally Affixal Languages. In addition to affixal languages with strong preferences for either prefixal or suffixal change a third category can be found that, whilst strongly inflectional, shows no particular preference for either prefix or suffix but uses both methods equally in constructing new forms. Languages with such morphologies present a particular challenge to the analyst as they tend to generate large numbers of forms by comparison with languages which exhibit a tendency to be primarily prefixal or suffixal. It is not easy to find such languages in the European group but many can be found amongst the Bantus. A good example would be Kikamba, a language spoken in South Eastern Kenya. Kikamba uses both prefixes and suffixes to generate surface forms of words. In comparison with a simple suffixal language such as English it is many times more complex in the number of word-forms which it can generate.

To process a word list most effectively the machine should be given the best chance of identifying morpheme structures from their owning stem-lemmata. If the language prefers a particular affix type significantly then a search for morphemes in that position will bear more fruit than in the other. It will also remove a greater degree of noise from the weaker signal which indicates the

secondary inflection vector. In simple terms this means that a language like English will yield better results when suffix morphemes are sought first rather than prefix morphemes. The converse is true of Kiswahili. Finding the optimum processing vector or *PIV* is the first significant task for the machine.

In the case of a language which prefers to construct forms using suffixes we may expect to find more *prima facie* evidence for suffix morpheme structures than for prefix structures. For affixal morphologies such structures are expressed as common final N-grams in the word list. A simple method can be constructed to investigate the final and initial N-grams in a word list as follows:

Experiment has determined that the optimum basis for assessing the *PIV* of a language is to review the occurrence count for all initial and final N-grams of length two, three, four and five characters. From the analysis, eight lists are created, four of initial N-grams and four of final N-grams. These lists are sorted on the basis of the occurrence of each N-gram such that those occurring most frequently are found at the top of the lists. The best five are then taken from each initial list, their occurrences summed and these results summed to generate a single value which represents the strength of the signal for prefixal inflection. Similarly the best five are taken from each of the four final lists and summed to generate a single value which represents the signal for suffixal inflection.

Let p represent the prefixal score and s the suffixal score. Let an N-gram list be represented in a table G_{li} where G_l is a list of N-grams, ordered by frequency, so that G_{l1} is the most frequently occuring, G_{l2} is the next most frequently ocuring, etc. Let P be a list of prefix N-grams and S be the list of suffix N-grams:

$$p = \sum_{l=2}^{5} \sum_{i=1}^{5} P_{li}$$

$$s = \sum_{l=2}^{5} \sum_{i=1}^{5} S_{li}$$

Intuition might encourage us to expect that this measure would form a sound basis for assessing the primary inflection vector since a generally prefixal or suffixal type of inflection should declare itself in these figures. Generally speaking this is so. There are, however, some circumstances in which the assessment breaks down. The two most common are distortions in the data due to phonemic preferences in syllable construction. In the case of a language such as Kiswahili which is known to be strongly prefixal in character we do not find this is confirmed by this measure. In fact a strongly suffixal signal is returned. This is generated by two characteristics of Kiswahili which can also occur in other languages. Kiswahili exhibits a small number of extremely common suffix morphemes, typically two characters in length. The evidence for these overwhelms what would otherwise be a very strong signal for prefix morphology. In addition to these morphemes, the preference shown by Kiswahili for open syllables generates further distortion through the limited number of common final syllables,

not necessarily morphemic, which occur. A similar distortion, driven by a preference for a particular syllabic form, occurs when a language notates opening glottals in onset position within syllables. Since these glottals are typically few in number and as they are equally typically followed by a vocalic nucleus they generate a strong signal for a limited number of spurious two or occasionally three character prefix morphemes.[8] Such distortions can be corrected by moving the basis of measure for prefix and suffix from the purely cumulative to a more comparative approach. This achieved as follows:

The occurrence counts for initial and final N-grams are summed. The total initial and final counts for each set of results are then used to generate a measure of prefix and suffix preference:

let p_l be the sum of the best five prefix occurrences of length l where $2 \geq l \geq 5$.

let s_l be the sum of the best five suffix occurrences of length l where $2 \geq l \geq 5$.

A reasonable assessment of the degree to which this language inflects by means of prefix rather than by means of suffix can be derived by summing the count for the best five morphemes in each list of initial N-grams and dividing that by the sum of the best five final N-grams:

$$PPIV = \frac{\sum_{l=2}^{5} p_l}{\sum_{l=2}^{5} s_l}$$

which can be simplified to $PPIV = \frac{p}{s}$. Likewise the preference for suffixal inflection is demonstrated by:

$$SPIV = \frac{\sum_{l=2}^{5} s_l}{\sum_{l=2}^{5} p_l}$$

which in turn can be simplied to $SPIV = \frac{s}{p}$. The strength of this preference can be expressed as the scale of the difference between the prefixal and suffixal preferences:

$$|PPIV - SPIV| = \left| \frac{\sum_{l=2}^{5} p_l}{\sum_{l=2}^{5} s_l} - \frac{\sum_{l=2}^{5} s_l}{\sum_{l=2}^{5} p_l} \right|$$

[8] The terminology *onset, nucleus* and *coda* is used to define the three components of syllables. The *onset* of a syllable is typically a consonant or consonantal cluster. *Nuclei* are typically vowels or perhaps glides and *codae* are usually consonants or consonant clusters. Most languages prefer to maximise onsets [6][40].

This refinement deals with the vast majority of syllabic distortions in *PIV* assessment. It can be improved still further by including in the assessment the best five N-grams taken from the initial+1 position and the final-1 position in words from the word list. This assessment is made, as before, for N-grams of two, three, four and five characters in length. Once again we calculate p/s and s/p for this data. Where this measure and the measure derived from N-grams in word boundary positions agree we can proceed to assign the *PIV* with confidence. Should the two results conflict we have the choice of preferring the stronger or perhaps referring the question to the user for confirmation. Equality of score between suffixal and prefixal is arbitrarily assigned to suffix. Where the scores are close the choice of *PIV* is unlikely to affect the processing but where a significant preference is indicated processing for that *PIV* first can improve the overall result.

3.2 Partitioning the wordlist

Having identified and set the *PIV* for the word list, each word in the list is now partitioned into all possible pairs of stem and suffix.[9] An example partition might be

morphemic ⇒{*(morphem, ic), (morphe, mic), (morph, emic), (morp, hemic), (mor, phemic)*}

This set of partitions expressed as ordered pairs of strings represents all the valid divisions which may be made within the word *morphemic*. The first item of each pair represents a putative stem and the second a possible morpheme or morpheme structure. Two constraints are set upon the partitioning process.

1. No attempt is made to identify morpheme structures of fewer than two characters in length.

2. A minimum length for stems is set at three or four characters. These constraints allow the process to avoid some areas of possible confusion which would otherwise be encountered. We have already noted the misleading effect of common final vowels, a minimum suffix length avoids this problem. Stems of fewer than three characters are often irregular in their formation of surface forms and are best sought as the complement to a morpheme structure which has already been identified and validated.

Taking the wordlist each word is examined to see if it contains at least five characters. This number is set by the minimum stem length plus two. Thus a minimum stem length of three characters ensures that only words of five characters or more will be partitioned. For each word of length equal to or greater

[9] It should be noted that henceforward this process description will discuss only suffix morpheme discovery in detail. The process for prefix morpheme discovery is identical and is achieved simply by reversing the word list data and processing as for suffixes. The results are then reversed once again to return them to their original form before they are reported.

than the minimum set for this processing run, a set of ordered pairs is generated where the first item in each pair represents a possible stem and the second item a possible morpheme structure, see example above. A table is maintained of all possible morpheme structures generated by this partitioning together with their occurrence. A word list of 50,000 words of average length nine characters will generate a partition set of 250,000 partitions. If the premise that common morphemes are found with many stems is valid, we may expect that the same morpheme structures will be found across many of the sets of partitions. This proves to be the case and forms the basis for our initial hypothesis for morpheme structures.

We now construct the set of all hypothesized morphemes (hypomorphs) from the union of all the morphemes found in the partition sets. Thus each morpheme in the set of all hypothesized morphemes is found in at least one of the partition sets generated from the wordlist. For each hypomorph we construct the set of all the corresponding stems found with the hypomorph in the partition sets where it can be found. The cardinality of this stem set is a measure of the validity of our hypothesis for this morpheme structure and, if the hypomorph is finally confirmed then this set comprises the putative stems attested by that hypomorph.

Clearly this process can generate very large sets of data. In order to restrict the number of hypothesized morphemes a threshold of occurrences, θ_{hypo}, is set below which possible morphemes are discarded. Experiment has suggested that for wordlists generated by a corpus of the size of the Bible, setting $\theta_{hypo}$to 128 for the initial pass is an acceptable filter for most languages. Hypomorphs with fewer than 128 occurrences are discarded.

It is the nature of any definition that the broader it is made the larger the number of entities which may be considered to share its common identity. This is clearly so in the current context. Consider, at this stage in the process of evaluating an English word list a strong signal is present for the suffix morpheme '*-ing'*. This is good. Less good is an even stronger signal for the morpheme '*-ng'*. The signal for '*-ng'* incorporates the evidence for '*-ing'* in addition to spurious signals derived from partitions such as *'bri-ng', 'flu-ng', 'stri-ng', 'doi-ng'* etc... To gain a clearer picture of the value of '*-ng'* we must first remove that portion of the '*-ng'* evidence which is dependent on '*-ing'*. This follows for all such related morpheme structures where the smaller structure is a final subset of the larger. In making these adjustments there is of course the possibility of losing some valid hypomorphs in favour of spurious, longer structures. This also occurs in English where the evidence for '*-ed'* is significantly reduced by the removal of that portion which is dependent on '*-red'* and '*-ted'*. Neither of these longer structures has any validity but the preference of English to close the final syllables of stems and the general frequency of '*r'* and '*t'* in such positions generates a strong signal for both these 3-gram suffixes. At this stage of the process, however, the benefits of this adjustment outweigh the disadvantages.

The set of hypomorphs remaining, each with a validity value at or above θ_{hypo} represents the best set of possible morpheme structures. From this set it is possible to proceed to the identification of putative stem-lemmata.

3.3 Initial stem identification

Possible stems can be found as the first items in the ordered pairs that are members of the sets of partitions generated from the word list. Only pairs where the hypomorph has an evidence value greater than θ_{hypo} are considered. For each of these hypomorphs a stem list is constructed containing each stem found with that morpheme in the word list. Each stem is now taken in turn and used to generate the set of morphemes found with that stem. Thus an English word list from which had been derived the hypomorph '*-ing'* at a validity $\geq\theta_{hypo}$ will now generate a large list of stems amongst which might be *pant, work* etc... From these example stems morpheme sets *(MS)* can be constructed e.g. *pant-{-ing, -ed}* and perhaps *work-{-ing, -ings, -ed, -er, -ers}* which contain all the hypomorphs associated with these stems in the partitions generated from the wordlist. In addition to the morphemes associated with these stems we may, in these cases but not necessarily all, add a further item to the morpheme set for a stem, representing the null morpheme since both *pant* and *work* may be found as surface forms in their own right. The morpheme sets for these stems now become *pant-{-ing, -ed, ∅}* and *work-{-ing, -ings, -ed, -er, -ers, ∅}*. These morpheme sets are the principal means by which we can assess the validity of the morpheme structures they contain. To evaluate them it is necessary to develop a means by which their relative merits can be assessed and those with the strongest claim to validity used to establish an initial set of validated morpheme structures.

The generated stem list provides an opportunity to extend the morpheme sets for each stem beyond those morphemes with validity$\geq\theta_{hypo}$ to include other possible morphemes which are attested by the full word list. The list of morphemes generated for each stem are likely to include morpheme structures that themselves fall below the threshold set in the initial phase of processing but which are attested by their partner stems and the other morphemes that are present in the generated morpheme set for those stems. In these circumstances both our example morpheme sets would be extended by the addition of an '*s'* morpheme in each case giving *pant-{-ing, -ed, s, ∅}* and *work-{-ing, -ings, -ed, -er, -ers, s, ∅}*. Note that possible single character morphemes, excluded from the initial search are recovered by this method. In view of the difficulty attributing a morphemic identity to a single character in various different contexts the occurrence value associated with such hypomorphs is set to 1.

Morpheme sets generated in this way represent significant evidence for both their individual hypomorph members and for the stems with which those members are found. The validity of these sets can be expressed as a function of the value associated with each individual morpheme (taken from the occurrence count for that hypomorph in the word list, adjusted for dependencies as discussed above) and the length of the hypomorph structure in characters. For each hypomorph we can state an evidence value thus $EV = c^l$ where c represents the adjusted occurrence count for that hypomorph raised to the power of the length of that hypomorph structure. This has the effect of weighting the value of a hypomorph strongly in favour of longer structures. For some highly

agglutinative languages where average word lengths can reach more than twenty characters this can generate quite large numbers. To ensure that these values remain computable at later stages the log of c is taken rather than the value itself thus: $EV = \log(c)^l$.

Having generated an EV for each morpheme in a morpheme set we calculate the morpheme set evidence value *(MSEV)* as the sum of the EV for each morpheme structure in the morpheme set thus: Let MS be a list of morphemes, where $MS_i$is the i-th morpheme in the list.

$$MSEV = \sum_{i=1}^{|MS|} EV = \sum_{i=1}^{|MS|} \log(c)^{length(MS_i)}$$

This value can be calculated for each morpheme set. At this stage each morpheme set is associated with a single stem for which it represents the range of items in that stem partition of the word list. It is likely that some of these stems will be in conflict. For example, in an analysis of an English word list the results at this stage of the process are likely to include entries such as *conflic- {-t, -ts, -ting, -ted}* and *conflict-{-s, -ing, -ed, ∅}*. Such conflicts may be resolved by reference to the *MSEV* for each morpheme set. Other possible means of deciding between conflicting stems include the presence or absence of the null morpheme. In this example both null morpheme and *MSEV* references prefer *conflict* to *conflic* and the latter stem is removed from the putative stem list. A further refinement of this part of the process would be to decrement the occurrence value for each morpheme in the morpheme set of the discarded stem by 1. This reflects the falling value of the evidence found for this morpheme. I have yet to thoroughly test this enhancement. It may well be that the effect of including it is negligible.

3.4 Building inflection paradigms

The remaining list of stems represent the best which can be identified from the initial morpheme structure hypothesis. These stems are now collated into hypothesized Inflection Paradigms *(IPs)*. Stems which share identical morpheme sets are placed together within the same inflection paradigm. Each *IP* consists of the set of morphemes which represents the *IP* for example *{ed, ing, s,∅}*. Also stored with the *IP* is the list of stems which exhibit all of the morphemes within the *IP* for example (*play, work*). Having constructed all possible *IPs* and discarded those which have only one morpheme the set of constructed *IPs* is reviewed and the lists of stems associated with each *IP* adjusted as follows. The set of *IPs* thus generated is reviewed and where an *IP* has a morpheme set which is a superset of another *IP* the stems for the first *IP* are added to the stem set of the second *IP* which has a morpheme set which is a proper subset of that of the first *IP*.

$$IP_1\{-ed, -ing, -s, -\oslash\}(play-, work-)$$

$$IP_2\{-ed, -ing\}(peal-)$$

$$IP_2\{-ed, -ing\}(peal-, play-, work-)$$

In this example the stems *play* and *work*, associated with IP_1 are also valid members of the stem set associated with IP_2 since they are found with both the two morphemes of IP_2. We may therefore add them to the stem set for IP_2. This is important as it will affect the weight given to IP_2 in terms of its overall validity as an *IP*.

The newly constructed *IPs* are now evaluated for evidence of extended morpheme structures. The stem set associated with each *IP* is examined for common final N-grams. This is often found particularly in stem sets derived from highly agglutinative languages. Experiment has confirmed that only common final N-grams of two or more characters should be noted. Common single final characters tend to mislead as they are often no more than common *codae* markers in the case of closed syllables or vocalic *nuclei* terminating open syllables. Latin, for example, will generate a strong signal for an *IP {-em, -es, -ibus}*. This is a partial expression of the third declension of Latin nouns. Closer examination of the stem set associated with this paradigm reveals a common final N-gram *-ent-* which encourages us to revise the morpheme structures to *{-entem, -entes, -entibus}*, removing the corresponding three character N-gram from each member of the stem set for that paradigm. In these circumstances it is often the case that the revised morpheme set is identical to another existing set. If this is so the stems are added to the stem set of that existing set and the original *IP* is deleted.

In practice the identification of sets of stems with common final N-grams is less straight-forward. In most cases some, but not all of the stems associated with an *IP* may share a common final N-gram. A more flexible means of identifying such commonalities is required which does not require all members of the stem set to share the common N-gram but which can identify a significant proportion of the stem set with common finals. At present this threshold θ_{stem} is achieved with a sliding scale of decision threshold calculated as follows:

For any stem set SS we set the threshold for the number of stems with common final N-grams to 3 or $1 + (|SS|\, 0.25)$, whichever is the greater. This provides us with a sliding scale that allows us to adjust a significant subset of stems within *IPs* with large stem sets where a significant number share common final N-grams. Stem sets with fewer than θ_{stem} members sharing a common final N-gram remain unchanged. The list of *IPs* is reviewed once more. Those with only one stem are removed as are *IPs* with only a single hypomorph in their morpheme sets. Lastly, in this iteration a Paradigm Evidence Value is calculated for each *IP*. The *PEV* is the product of the *MSEV* and the size of the associated stem set SS: $PEV_n = (MSEV_n\, |SS_n|)$ where: $MSEV = \sum_{i=1}^{|MS|} EV$ and $EV = \log{(c)}^l$.

IPs are now ranked in order of *PEV*. A threshold $\theta_{paradigm}$ is set at and above which *IPs* are retained, the remainder are discarded. Initial experiments have suggested that for most languages setting $\theta_{paradigm}$ to 128 is generally effective. From the remaining set of *IPs* a set of core morphemes are extracted comprising the union of all the morpheme sets associated with each remaining *IP*. These morphemes are considered well attested by virtue of their relationship to one another via their shared membership of the morpheme sets and by the number of stems with which they have been found. They form the basis for the remainder of this process which now seeks to extend this set of well-attested morphemes and identify their corresponding stems.

3.5 Extending the morpheme set

Before this list of well-attested morphemes is used to start the next stage of the process it is first ordered such that the longest morpheme structures appear first and within each group of morphemes of length l the members of that group are ordered on the basis of their individual occurrence values. Well-attested hypomorphs which will be used as a basis for further processing I term 'bankers'. For an English analysis this might generate a banker morpheme list ordered thus: *(-ations[43], -ation[129], ness[130], ings[83], ated[62], ..., ing[1032], ...)*. Each morpheme is now taken in turn and a set of stems is constructed which represent the stems found in all possible partitions of the word list where that morpheme is found. Those entries in the word list which generate these partitions are removed from the the working copy of the word list. This guards against the possibility that an entry such as *blessedness* might generate two conflicting partitions: *(bless, -edness)* and *(blessed, -ness)*. Further work is needed to assess whether or not better results can be obtained by permitting conflicting partitions at this stage and accepting the extra processing required to identify larger numbers of stem sets where the members share common final N-grams.

Full morpheme sets are now constructed from the word list for the stems identified from the banker morpheme set. For example, an English banker morpheme *'-ing'* might identify a stem *'prais-'* which in turn generates the morpheme set *{-e, -ed, -es, -ing}*.[10] The ensuing stems and their morpheme sets are collated into inflection paradigms as before. The resulting *IPs* are checked for common final N-grams. Once again, *IPs* where the morpheme set is a subset of another *IP* have their stem sets extended by the addition of the stems attested by the superset. A global morpheme set is constructed from

[10] It will be noted that a common outcome of this process is the generation of stems which do not always conform to that which linguists expect. In many cases the stem of a word is considered to be its citation form. In the case of this example *praise*. The true stem, or perhaps more accurately, the logical radix, is, as stated *prais*. Many linguists will argue that the 'e' morpheme listed for *prais* is spurious and serves only to voice the final consonant of the word. This may be so but if we are to limit the pre-requisites for this process as suggested above we must accept this sort of result. Rather than exposing a weakness in the process it serves to highlight the difficulties of orthographies which have developed over many hundreds of years.

those *IPs* where the *PEV* exceeds $\theta_{paradigm}$, now reduced to 64.

A third iteration of the process follows producing a further extended set of banker morphemes from the resulting *IPs*. This final banker morpheme set is selected on the basis of the *PEV* of the parent IPs, the number of morphemes in the morpheme set for that *IP* and the number of those morphemes which have a value $>$ than 1 (thus discounting the evidence of single character morphemes for whom it is difficult to derive a useful value). The standard values for these parameters have been set by experiment as follows. Valid *IPs* must have *PEVs* at or above 128 and morpheme sets of two or more members where at least two of the members of the morpheme set have occurrence values of more than 1. Some languages may give better results with different thresholds but this remains a matter for experiment. The global morpheme set resulting from this iteration is used to generate a global stem list by which the word list is partitioned for the last time and final *IPs* are built. Tables of morphemes, stems and *IPs* are constructed and the results reported for review. A full set of results for this processing run against English, Latin and Kiswahili wordlists can be found below.

3.6 Process summary

The basis for the identification of morpheme structures and their associated stem lemmata is the relationship between these entities. Following the initial stage of morpheme identification, which is based solely upon the relative occurrence rate of N-grams at word boundary positions, subsequent identification depends upon the stems implied by the initial morpheme set and their relationship both to the morpheme structures they attest individually and the structures which are attested in common with other stems. This relational approach expresses well the dependencies between entities and is also a more intuitive presentation for traditional linguists of the accidence of a language.

As noted above there are a number of areas in this work that would benefit from further experiment. Reference has been made to a number of thresholds which are at present to levels at which they give generally good results across all the test languages. It is expected that results would improve if these thresholds were set optimally per language. An interesting area of work remains in discovering the characteristics of individual languages which govern the optimal threshold settings for those languages and calculating optimal threshold settings automatically from the input data in the word list. Questions also remain about the benefit of permitting conflicting stem set generation at during the extension of the initial morpheme sets. As the prototype system moves into wider field trials I expect to address these and other issues and refine the method accordingly.

Chapter 4

Experimental results

The broad aim of this research has been to develop a process which might provide significant help to linguists by the unsupervised discovery of affixal morpheme structures across a wide spread of natural languages. Implicit in this is the recognition that the identification of morpheme structures enables, and is in fact dependent upon, the discovery of stem-lemmata. Whilst the focus of discovery has been limited to affixal languages a key constraint has been the need to generate results for the widest possible set of natural languages that falls within this group. It was, therefore, important that the process generated results for a disparate set of test languages drawn from more that one of the main linguistic groups. For the greatest accessibility results generated from English have also been included. Although it is also drawn from the same group, Latin was included in the set of test languages as an example of a highly inflected suffixal language. The final example is drawn from the one of the Bantu languages of sub-Saharan Africa, Kiswahili.

When the scope of testing is widened in this way it is inevitable that compromises must be made to accommodate the breadth of the characteristics of the target languages. The identification of the scope and nature of these compromises remains a continuing focus for this work. It is likely that in this broader context some issues of analysis may remain unresolved for some languages that, in the context of processing tailored specifically to that language might be satisfactorily dealt with. The general context for this work is, therefore, broad and these results should be interpreted through that context. Nevertheless, it is believed that that useful analysis has been achieved for each of the test languages. The very breadth of context itself necessitated the development of methods for assessing the inflection rate and the primary, or preferred, inflection vector of a language.

Results are presented for each language as follows: The inflection rate and vector are given as defined above. Examples of identified morphemes and morpheme structures are listed as are stem-lemmata and some of the many inflection paradigms identified.

4.1 English

As an appropriate candidate for this process, English leaves much to be desired. The nature of the relational approach, dependent as it is on corroboration from other members of the same set, tends to work best when the information inherent in the dataset is broad. Put simply, this process favours the more highly inflected languages. Secondly, English is a member of the Germanic group as a consequence of its Anglo-Saxon origins. One of the characteristics of these languages is the use of infixal inflection. This occurs in both nominal and verbal vocabulary. Examples include: *man, men*; *sink, sank, sunk*; and *ring, rang, rung*. Lastly, the length of time during which English has been both a written language and a language under rapid development is such that a significant number of orthographic complexities have developed. These include the presence, or absence, of the *e-schwa* to vocalise stem final syllables as in *burned* or *heard*. There can be little doubt that *burned* should parse as *burn-ed* and a moment's thought suggests *hear-d* as a contraction of *hear-ed*. As for *agreed*, should this parse as *agree-d* or *agre-ed*? At first sight *agre-ed* seems logical but the existence of the forms *agree-ment* and *agree-ing* might argue otherwise. We shall see that there is strong evidence for an English suffix morpheme *-ness*. This looks plausible as evidenced by words such as *bright-ness*. Some might, however argue that we should rewrite this as *bright-/e/n-ess*, supplying the *schwa* implicit within *brighten*. Such transformations can be explained by reference to the rules defined in transformational grammars of English [46]. It is, however, a principle of this process that no other information than the word list is used to generate the results. Nevertheless, even without the use of existing word-formation rules, useful results can be obtained.

The English word list used for this experiment was taken from the Revised Standard Version Bible. Proper names were removed from the list which left a total of 12,825 words to be processed.

4.1.1 Inflection rate and PIV

For the purposes of this experiment English has been regarded as the datum against which the inflection rates of other languages can be measured. The assessment of the PIV for English detected a clear suffixal bias in the dataset.

Table 4.1: English: Word initial and word final N-gram assessment.

Initial 2-grams	Occurrence	Final 2-grams	Occurrence
co-	507	*-ed*	1721
re-	493	*-ng*	1145
de-	340	*-es*	890
in-	311	*-er*	486
pr-	310	*-ly*	423
$\frac{p}{s} = \frac{1961}{4670} = 0.42$	1961	$\frac{s}{p} = \frac{4670}{1961} = 2.38$	4670

Initial 3-grams	Occurrence	Final 3-grams	Occurrence
con-	242	*-ing*	1112
dis-	183	*-ted*	345
pro-	153	*-ion*	307
for-	120	*-ers*	298
com-	109	*-ess*	227
$\frac{p}{s} = \frac{807}{2289} = 0.68$	807	$\frac{s}{p} = \frac{2289}{807} = 2.83$	2289

Initial 4-grams	Occurrence	Final 4-grams	Occurrence
cons-	62	*-tion*	237
over-	59	*-ting*	180
comp-	58	*-ness*	140
cont-	55	*-ring*	130
inte-	47	*-ions*	123
$\frac{p}{s} = \frac{281}{810} = 0.35$	281	$\frac{s}{p} = \frac{810}{281} = 2.88$	810

Initial 5-grams	Occurrence	Final 5-grams	Occurrence
inter-	30	*-ation*	138
beth–	29	*-tions*	84
thirt-	21	*-ering*	59
trans-	21	*-ously*	41
contr-	21	*-nding*	40
$\frac{p}{s} = \frac{122}{362} = 0.34$	122	$\frac{s}{p} = \frac{362}{122} = 2.97$	362

The sum of the prefix indicators is found to be 2.45 whereas that for suffix indicators amounts to 12.06. The initial assessment of boundary N-grams thus returns a strong preference for suffix as the preferred inflection vector for English. To confirm this the same calculation is made for N-grams in initial+1 and final-1 positions.

Table 4.6: English: Word initial+1 and word final-1 N-gram assessment.

Initial+1 2-grams	Occurrence	Final-1 2-grams	Occurrence
_*on*-	331	-*in*_	1247
_*ar*-	322	-*te*_	544
_*or*-	315	-*es*_	399
_*ea*-	306	-*er*_	378
_*ro*-	293	-*re*_	325
$\frac{p}{s} = \frac{1567}{2893} = 0.54$	1567	$\frac{s}{p} = \frac{2893}{1567} = 1.85$	2893

Initial+1 3-grams	Occurrence	Final-1 3-grams	Occurrence
_*est*-	82	-*tio*_	237
_*ver*-	77	-*tin*_	184
_*ear*-	75	-*nes*_	152
_*rea*-	75	-*rin*_	141
_*ort*-	73	-*ion*_	123
$\frac{p}{s} = \frac{382}{837} = 0.46$	382	$\frac{s}{p} = \frac{837}{382} = 2.19$	837

Initial+1 4-grams	Occurrence	Final-1 4-grams	Occurrence
_*ight*-	54	-*atio*_	138
_*nter*-	36	-*tion*_	84
_*eth*–	29	-*erin*_	59
_*ound*-	28	-*esse*_	51
_*esti*-	27	-*ishe*_	50
$\frac{p}{s} = \frac{174}{382} = 0.46$	174	$\frac{s}{p} = \frac{382}{174} = 2.20$	382

Initial+1 5-grams	Occurrence	Final-1 5-grams	Occurrence
_*ytrih*-	18	-*ation*_	48
_*wenty*-	17	-*ratio*_	28
_*event*-	16	-*resse*_	27
_*orty*–	16	-*tatio*_	24
_*ighte*-	14	-*endin*_	20
$\frac{p}{s} = \frac{81}{147} = 0.55$	81	$\frac{s}{p} = \frac{147}{81} = 1.81$	147

Once again these results show a strong preference for suffixal inflection. The sum of the prefix indicators is 3.00 whereas that for the suffixal indicators is 9.05. The PIV is therefore established as suffixal.

4.1.2 Initial Partitioning

The initial partition of the word list generates 7,942 possible suffix morphemes of which 2,521 occur more than once in the list. Setting the evidence threshold to 128 occurrences reduces this list to just fourteen hypomorphs:

Table 4.11: English initial hypomorph table - occurence $>$ 128.

morph	occurs	morph	occurs
-es	531	*-ts*	196
-ed	468	*-en*	151
-ing	425	*-ned*	142
-er	418	*-ting*	141
-ly	396	*-ness*	130
-ted	299	*-al*	130
-ers	255	*-ation*	129

At first sight this does not look promising. Of the thirty initial hypomorphs fifteen seem to be invalid. Such a limited dataset may seem disappointing but it is quite reasonable given the low inflection rate of English. In fact, as we shall see, it is a good base from which to proceed. Recognising that some of these may be subsets of longer structures we now adjust the occurrence scores to reflect this, subtracting the occurrence of the longer structure from the score for the shorter. When this is done our table of initial hypomorphs with occurrence values of 128 or more rewrites thus:

Table 4.13: English initial hypomorph table hypomorphs adjusted for common sub-structures.

morph	occurs	morph	occurs	morph	occurs
-ed	1614	*-ion*	298	*-ce*	156
-ng	1105	*-ers*	255	*-en*	151
-ing	1032	*-ns*	248	*-ent*	147
-es	759	*-ss*	232	*-te*	143
-er	418	*-tion*	227	*-ned*	142
-ly	396	*-nt*	221	*-ons*	141
-rs	376	*-ess*	218	*-ting*	141
-on	360	*-le*	218	*-al*	130
-ted	299	*-st*	199	*-ness*	130
-ts	299	*-red*	165	*-ation*	129

4.1.3 Validating hypomorphs

Taking the default of 128 as the threshold for morpheme validity a stem list is constructed from these morphemes. The morpheme list for each stem is extended across the word list and stem conflicts resolved, generating a set of 4,074 putative stems. Collating these stems into Inflection Paradigms where each paradigm is attested by more than one stem produces a paradigm set of 283 members. Having adjusted these paradigms to account for common final *N*-grams amongst the stems the following list of banker morphemes can be extracted:

Table 4.15: English well attested (banker) morpheme set.

morph	occurs	morph	occurs
-es	531	*-ings*	83
-ed	468	*-ate*	77
-ing	425	*-or*	76
-er	418	*-able*	73
-ly	396	*-ment*	71
-ers	255	*-ion*	65
-en	151	*-ful*	64
-al	130	*-ated*	62
-ness	130	*-ors*	47
-ation	129	*-ations*	43
-ies	114	*-ure*	38
-ions	114	*-cation*	17
-est	105	*-eless*	9

All bar three of these appear to be valid English suffix morphemes. Of these three, *-ure*, *-cation* and *-eless*, we can find valid partitions based upon them: *creat-ure, purifi-cation, sens-eless*. In each of these cases the process is correctly identifying the stem of the word: *creat-, purifi-* and *sens-*. For the first and last the confusion arises because of the *e* added to the stem to vocalise the final stem consonant. In the case of *purifi-* the stem final *y* rewrites as an *i*. The process now moves from extending a well-attested set of bankers to using these morphemes to parse the wordlist as a whole. Taking each morpheme in turn, longest structures first, the corresponding stems found with that morpheme are identified. These stems are then used to parse other words found with the same morpheme, thus generating a list of stems each of which shares a common set of morphemes. From this new set of inflection paradigms a new morpheme set is extracted:

Table 4.17: Extended banker morpheme set for English.

morph	morph	morph	morph
-age	*-edly*	*-ines*	*-ous*
-able	*-d*	*-ied*	*-ness*
-ably	*-e*	*-ies*	*-or*
-acy	*-ed*	*-ine*	*-ors*
-al	*-eless*	*-ing*	*-ously*
-ally	*-ely*	*-ingly*	*-r*
-ance	*-en*	*-ings*	*-re*
-ances	*-ens*	*-ion*	*-res*
-ants	*-er*	*-ional*	*-s*
-at	*-er-in-law*	*-ioned*	*-st*
-ate	*-ered*	*-ions*	*-t*
-ated	*-erer*	*-ior*	*-ted*
-ately	*-ering*	*-iors*	*-ting*
-ates	*-ers*	*-ished*	*-ung*
-ating	*-ery*	*-ive*	*-ure*
-ation	*-es*	*-ly*	*-ured*
-ations	*-ess*	*-ment*	*-ures*
-ator	*-est*	*-ments*	*-y*
-cation	*-ful*	*-n*	

This expansion has generated a number of morphemes which some might question. The vast majority of these are the consequence of morpho-phonemic changes at the stem-morpheme boundary. An example of this is found in *-ting* which is generated as a consequence of gemination of the closing consonant of the stem between the stem-final syllable and the suffix *-ing* as in *admitting, begetting, committing, flitting, forgetting, omitting, plotting, spitting* and *strutting*. It can be argued that in these cases, the *-ting* morpheme is in fact valid although we might prefer to rewrite it *-t-ing*. Others, as noted above, are the consequence of the presence or absence of the vocalising *e-schwa*, for example *-d, -n, -st, -t*.

These morphemes are now used to parse the wordlist once more. After generating new inflection paradigms and refining stems a 'bootstrap' morpheme set is generated which is used to prepare the final parse of the wordlist:

Table 4.19: Final English morpheme table.

morph	morph	morph
-able	*-eless*	*-ions*
-ably	*-ely*	*-ior*
-acy	*-en*	*-iors*
-age	*-ens*	*-ive*
-al	*-er*	*-ly*
-ally	*-er-in-law*	*-ment*
-ance	*-ered*	*-ments*
-ances	*-erer*	*-n*
-ants	*-ering*	*-ness*
-at	*-ers*	*-or*
-ate	*-ery*	*-ors*
-ated	*-es*	*-ous*
-ately	*-ess*	*-ously*
-ates	*-est*	*-r*
-ating	*-ful*	*-s*
-ation	*-ied*	*-st*
-ations	*-ies*	*-t*
-ator	*-ing*	*-ting*
-cation	*-ingly*	*-ung*
-d	*-ings*	*-ure*
-e	*-ion*	*-ured*
-ed	*-ional*	*-ures*
-edly	*-ioned*	*-y*

Six morphemes have been removed from the list generated at the previous stage: *-ine, -ines, -shed, -re, -res* and *-ted*. Of the remaining sixty-nine all are supported by reasonable evidence from the wordlist. Some, inevitably, will generate invalid parses: *-at, -cation, -or, -ors, -r, -st, -ting* and *-y*. The total invalid parses generated by these morphemes is 212. The total number of valid parses produced by the process is 6,611. For a language such as English with a low rate of inflection a success rate of 96.89% represents a very encouraging result.

4.1.4 Example stem sets and paradigm sets

Examples of inflection paradigms generated by this process include:

Table 4.21: Example English inflection paradigms.

blasphem	*-e*	command	*-ed*
	-ed		*-er*
	-er		*-ers*
	-ers		*-ing*
	-es		*-ment*
	-ies		*-ments*
	-ing		*-s*
	-ous		*-Ø*
	-y	divin	*-ation*
oppress	*-ed*		*-ations*
	-es		*-e*
	-ing		*-ed*
	-ion		*-er*
	-ions		*-ers*
	-ive		*-es*
	-or		*-ing*
	-ors	transgress	*-ed*
accus	*-ation*		*-es*
	-ations		*-ing*
	-e		*-ion*
	-ed		*-ions*
	-er		*-or*
	-ers		*-ors*
	-es		*-Ø*
	-ing		

Table 4.23: English nominal/adjectival paradigms.

ambassad	*-or*	bank	Ø
	-ors		*-s*
arch	*-er*		*-ers*
	-ers	bunch	*-Ø*
bald	*-Ø*		*-es*
	-ness		

The degree of ambiguity inherent in English morphology is at once clear. It is particularly common in languages with very low rates of inflection to find morphemes used in different contexts to indicate different things. In just these

few examples such ambiguity can be found in *bank-s* and *bunch-es* where it is impossible to be certain whether these forms are nominal or verbal. In these cases we may find that in some instances the form represents a verbal 3rd person singular yet in others it may represent the nominal plural.

Apparently verbal paradigms can also be identified:

Table 4.25: English verbal paradigms.

accomplish	-Ø	know	-Ø
	-ed		*-est*
	-es		*-ing*
	-ing		*-s*
account	-Ø	loath	-Ø
	-ed		*-ed*
	-ing		*-es*
	-s		*-ing*

with the same caveat as before for *account-s*.

The 212 invalid parsings can be accounted for as follows:

Table 4.27: Invalid partitions from the English analysis.

morph	bad parses
-at	*comb-at, defe-at, elishaph-at, entre-at,*
	ill-tre-at, maltre-at, repe-at, retre-at,
	sheb-at, somewh-at, thre-at, thro-at, winev-at
-cation	*communi-cation, confis-cation, convo-cation,*
	dedi-cation, edifi-cation, forni-cation,
	invo-cation, provo-cation, suppli-cation,
	vindi-cation
-d	*amaze-d, arrange-d, atone-d, base-d,*
	chastise-d, complete-d, defile-d, disfigure-d,
	encourage-d
-er	*badg-er*
-ers	*badg-ers*
-or	*anch-or, auth-or, baal-haz-or, cast-or, clam-or,*
	harb-or, horr-or, mirr-or, terr-or, trait-or
-ors	*anch-ors, harb-ors, indo-ors, past-ors*
	terr-ors, trait-ors
-r	*fierce-r, idle-r, large-r, pure-r*
-st	*agha-st, almo-st, ... evangeli-st, fewe-st,*
(86)	*fierce-st, ... fine-st, fore-st etc...*
-ting	*arres-ting, crea-ting, ... flee-ting, fron-ting,*
(68)	*predic-ting, projec-ting, ... trea-ting, visi-ting*
-y	*adversit-y, appl-y, iniquit-y,*
	marr-y, nullif-y

4.1.5 Summary

Most of the errors occur at the final stage of the process and are the result of populating the final morpheme set indiscriminately throughout the wordlist. A few are arguable. Consider *edify* > *edifi-cation* and *fierce* > *fierce-st* where the $y \Rightarrow i$ shift and the vocalising *schwa* has confused the analysis. Despite these errors, given the nature of English and its unsuitability to this type of process the overall result is pleasing.

4.2 Latin

The choice of Latin as test data for this process may seem odd insofar as it represents a language no longer in any kind of general use and without any present form of vernacular context. There are, however, a number of benefits to this choice. The Latin orthography with its absence of diacritical marks has

become, via the agency of English, the preferred orthography for the control languages of most modern computer systems. There are in consequence few problems associated with the encoding and display of Latin text on modern systems. Secondly, Latin is probably the most studied language in the Western world. Innumerable Latin grammars have been written.[1] Its place until recently in the curriculum of most Western European schooling, together with the large number of speakers of Latin-based Romance languages worldwide offers the hope of wide accessibility for these results. Lastly, and perhaps most significantly for this work, Latin exhibits a high rate of inflection by comparison to most modern European languages. Reference to the table of relative inflection rates above indicates that Latin has an inflection rate comparable to Kiswahili or Kikamba. As such it represents an accessible yet complex data set for this process.

4.2.1 Inflection rate and PIV

Applying the measure detailed above this can be assessed as follows:

The number of words found in a wordlist derived from the Latin Vulgate Bible, disregarding proper names, is 46,659. The 'control' value for the equivalent English word list is 12,825. This gives a relative inflection rate of approximately 1:3.64 for Latin Vulgata. This is, interestingly, higher than that of Kiswahili (see below) a fact not many, including this author, might have imagined. Such a result may suggest that other factors such as the number of lemmata found in the wordlist have distorted the figure.

The assessment of Primary Inflection Vector generated a strong suffixal signal from the following wordlist:

[1] These results were validated against the tables published in Jones and Sitwell [28].

Table 4.29: Initial and final N-gram assessment for Latin.

Initial 2-grams	Occurrence	Final 2-grams	Occurrence
co-	3680	*-is*	4238
in-	2909	*-nt*	3823
pr-	1857	*-us*	3414
re-	1705	*-um*	3227
de-	1657	*-ur*	2899
$\frac{p}{s} = \frac{11808}{17601} = 0.67$	11,808	$\frac{s}{p} = \frac{17601}{11808} = 1.49$	17,601

Initial 3-grams	Occurrence	Final 3-grams	Occurrence
con-	2572	*-tur*	2710
pro-	929	*-tis*	1725
per-	826	*-que*	1636
pra-	722	*-unt*	1454
int-	516	*-ant*	1104
$\frac{p}{s} = \frac{5565}{8629} = 0.64$	5565	$\frac{s}{p} = \frac{8629}{5565} = 1.55$	8629

Initial 4-grams	Occurrence	Final 4-grams	Occurrence
prae-	693	*-ntur*	1096
cons-	614	*-runt*	908
cont-	442	*-ibus*	804
conf-	332	*-ntes*	737
inte-	322	*-etur*	590
$\frac{p}{s} = \frac{2403}{4135} = 0.58$	2403	$\frac{s}{p} = \frac{4135}{2403} = 1.72$	4135

Initial 5-grams	Occurrence	Final 5-grams	Occurrence
trans-	240	*-erunt*	1096
inter-	221	*-entes*	449
circu-	212	*-entur*	420
super-	203	*-antur*	411
praec-	180	*-tibus*	355
$\frac{p}{s} = \frac{1056}{2526} = 0.41$	1056	$\frac{s}{p} = \frac{2526}{1056} = 2.39$	2526

The sum of the prefix indicators is found to be 2.3 whereas that for suffix indicators amounts to 7.15. An assessment of boundary N-grams thus returns a strong preference for suffix as the preferred inflection vector for Latin. To confirm this the same calculation is made for N-grams in initial+1 and final-1 positions.

Table 4.34: Inital+1 and final-1 N-gram assessment for Latin.

Initial+1 2-grams	Occurrence	Final-1 2-grams	Occurrence
_*on*-	2949	-*tu*_	4199
_*er*-	1777	-*ti*_	2335
_*ra*-	1498	-*ri*_	1687
_*es*-	1045	-*qu*_	1679
_*ro*-	984	-*en*_	1664
$\frac{p}{s} = \frac{8253}{11564} = 0.71$	8253	$\frac{s}{p} = \frac{11564}{8253} = 1.40$	11564

Initial+1 3-grams	Occurrence	Final-1 3-grams	Occurrence
_*rae*-	699	-*unt*_	1200
_*ons*-	637	-*uta*_	1191
_*ont*-	470	-*etn*_	1096
_*est*-	363	-*ire*_	1091
_*nte*-	345	-*nur*_	909
$\frac{p}{s} = \frac{2514}{5487} = 0.46$	2514	$\frac{s}{p} = \frac{5487}{2514} = 2.18$	5487

Initial+1 4-grams	Occurrence	Final-1 4-grams	Occurrence
_*rans*-	244	-*erun*_	892
_*esti*-	237	-*ente*_	664
_*nter*-	222	-*ione*_	518
_*ircu*-	215	-*entu*_	514
_*uper*-	206	-*antu*_	420
$\frac{p}{s} = \frac{3008}{1124} = 0.37$	1124	$\frac{s}{p} = \frac{1124}{3008} = 2.68$	3008

Initial+1 5-grams	Occurrence	Final-1 5-grams	Occurrence
_*ircum*-	186	-*tione*_	427
_*onsum*-	97	-*verun*_	350
_*abita*-	89	-*ntibu*_	256
_*isper*-	86	-*abitu*_	253
_*ancti*-	86	-*averi*_	207
$\frac{p}{s} = \frac{544}{1493} = 0.36$	544	$\frac{s}{p} = \frac{1493}{544} = 2.75$	1493

As for the boundary N-grams, each of these results shows a strong preference for suffixal inflection. The PIV is therefore establish as suffixal and the process attempts first to identify morphemes and morpheme structures in suffix position.

4.2.2 Initial partitioning

The initial partitioning of the word list derived from the Latin Vulgate generates a table of 178,363 stem-suffix partitions, where the minimum stem length has been set to four characters. Of these, 25,908 are non-unique suffixes and represent the raw data from which the process hopes to identify morphemes and morpheme structures. Setting an evidence threshold of one hundred and twenty-eight occurrences reduces the list to just one hundred and ninety five entries. The best one hundred of these are reproduced in the table below:

Table 4.39: Latin initial hypomorph table - occurence > 128.

Morph	Occ.	Morph	Occ.	Morph	Occ.	Morph	Occ.
is	4018	*tes*	840	*tem*	463	*verunt*	333
nt	3760	*ns*	839	*ium*	457	*atus*	330
us	3239	*te*	838	*ens*	448	*itis*	321
um	3043	*erunt*	807	*eris*	439	*ionem*	319
ur	2865	*bus*	806	*ere*	435	*tibus*	314
tur	2651	*rum*	794	*ri*	428	*ione*	310
it	2281	*ris*	782	*bit*	418	*tio*	307
es	1921	*ibus*	740	*rent*	417	*set*	302
ue	1619	*ntes*	713	*bat*	414	*res*	301
tis	1607	*ta*	701	*avit*	410	*are*	299
que	1548	*tus*	674	*ni*	407	*to*	299
em	1517	*os*	591	*imus*	406	*ans*	296
unt	1387	*tum*	589	*itur*	406	*ntem*	292
at	1377	*etur*	543	*nem*	406	*sset*	291
et	1352	*io*	541	*orum*	404	*dum*	289
am	1342	*vit*	540	*erit*	402	*ini*	284
ti	1197	*atur*	519	*entur*	387	*averunt*	278
as	1073	*tque*	500	*antur*	385	*rem*	276
ntur	1069	*ia*	480	*entes*	376	*abit*	272
ant	1038	*ret*	471	*nis*	370	*bitur*	271
re	1037	*rit*	470	*or*	363	*ati*	268
ent	975	*sque*	470	*atis*	361	*bunt*	268
mus	892	*sti*	467	*rat*	345	*ate*	267
runt	872	*bant*	465	*onem*	343	*erat*	265
ae	862	*ne*	463	*one*	336	*ata*	261

It is possible at once to identify a large number of succesful identifications but in addition to these there are many misleading suggestions. Of the first ten hypomorphs, five represent valid morphemes: *{-is, -us, -um, -it, -es}* although, interestingly, not all the occurrences noted will represent valid partitions and the other five members of the first ten *{-nt, -ur, -tur, -ue, -tis}* represent invalid

identifications and partitions. On closer inspection it becomes clear that in most cases strong signals for valid morphemes have become swamped by noise from part-morpheme partitions. For example the hypomorph *nt*[2] is a combination of *{-ant, -ent, -unt, -iunt, -erunt, -averunt, etc...}*. Many of these valid signals are also present in the list of the strongest one hundred. To better assess the value of each hypomorph a search must be made for other hypomorphs in the list which are reasonably well attested and which share a common final N-gram with the hypomorph under review. At present *reasonably well attested* is defined as $\frac{\theta^{morph}}{2}$ i.e. in this case sixty four. Since the value attributed to *-nt* is inflated by the scores of longer hypomorphs we can adjust the result by subtracting the score of any longer hypomorph for which *-nt* is a common final from the score for *-nt* itself. The result of this can be seen in the second table:

Table 4.41: Latin initial hypomorphs adjusted for common sub-structures.

Morph	Value	Morph	Value	Morph	Value	Morph	Value
ae	449	*ati*	268	*arum*	197	*uit*	148
os	421	*erit*	267	*erat*	192	*ationem*	147
as	418	*ata*	261	*amus*	189	*ta*	147
es	371	*et*	257	*ite*	186	*lis*	146
is	346	*orum*	250	*erent*	183	*abis*	145
ere	345	*atis*	247	*abunt*	180	*atione*	144
erunt	344	*ebant*	245	*li*	180	*abant*	143
am	341	*isti*	245	*ari*	178	*rem*	142
atur	337	*ti*	244	*atio*	176	*entem*	141
us	336	*antes*	237	*isset*	176	*sus*	141
ant	332	*ri*	230	*re*	176	*iae*	138
at	332	*abitur*	226	*rentur*	175	*abat*	137
atus	330	*um*	226	*mur*	170	*itur*	134
ens	330	*it*	223	*avi*	168	*tae*	134
avit	319	*to*	221	*erant*	164	*aret*	133
etur	312	*unt*	221	*abo*	163	*ii*	133
imus	311	*antur*	215	*iis*	160	*torum*	132
are	299	*em*	213	*erint*	157	*ate*	131
ans	296	*que*	213	*tam*	157	*iam*	130
eris	287	*ra*	213	*itque*	154	*num*	130
entes	284	*ebat*	206	*etis*	153	*ius*	129
averunt	278	*entur*	205	*ium*	153	*mini*	129
tum	278	*atum*	204	*asti*	150	*ndi*	128
ent	277	*eret*	200	*tos*	149	*si*	128
abit	272	*emus*	198	*io*	148	*tate*	128

which displays the best one hundred hypomorphs following this adjustment.

[2]It is interesting to note that a modern Latin primer in common use[1] instructs its students that -nt is indeed a valid Latin verbal ending, corresponding to the 3rd person plural.

We now have to search to position twenty three before finding an invalid hypomorph *-tum*. Of the first fifty hypomorphs only five are invalid. At this stage a 90% success rate for the best fifty hypomorphs is encouraging. As we move down the list the rate falls as we might expect yet of the best one hundred hypomorphs, seventy nine are valid. Of the twenty one failures thirteen are a consequence of Latin's tendency to close nominal stem-lemmata with *t* or *r*. The remainder are fairly random but one is worthy of note: *-uit* is the result of inconsistencies in the rendering of Latin orthography which have crept in during the last two thousand years or so. Such inconsistencies tend to be less frequent in orthographies which have been defined in more recent years.

4.2.3 Validating hypomorphs

Taking the default of 128 as the threshold for morpheme validity a stem list constructed from these morphemes with the morpheme list for each stem extended across the word list and stem conflicts resolved generates a set of 19,774 putative stems. Collating these stems into Inflection Paradigms where each paradigm is attested by more than one stem generates a paradigm set of 599 members. Having adjusted these paradigms to account for common final *N*-grams in stems the following list of banker morphemes can be extracted:

Table 4.43: Latin hypomorphs after 1st iteration.

*re**	*erat*	*entis*	*ationе*
it	*ente*	*entes*	*aretur*
is	*ebat*	*entem*	*issemus*
as	*ebam*	*ebant*	*entibus*
am	*avit*	*rentur**	*ebantur*
ae	*arum*	*itatis*	*avitque*
*ret**	*itque*	*itatem*	*averunt*
ere	*itate*	*issent*	*averint*
avi	*istis*	*eretur*	*ationum*
ari	*isset*	*entium*	*ationes*
are	*issem*	*ebatur*	*ationem*
itas	*erunt*	*avimus*	*eruntque*
isti	*erent*	*averit*	*averitis*
imus	*erant*	*averis*	*ationibus*
erit	*entur*	*averat*	*averuntque*

Of these sixty well attested morphemes only three are invalid (marked *), giving a partition accuracy of 95% for this stage of the process. Interestingly the three morphemes in question are related insofar as they all share an initial *-re-*. A second iteration of stem identification based upon this set of morphemes refines the list and the new set of banker morphemes becomes:

Table 4.45: Latin hypomorphs after 2nd iteration.

ae	*em*	*es*	*itatem*
am	*endi*	*et*	*itatis*
ant	*ent*	*ete*	*ite*
arum	*entes*	*etis*	*itis*
as	*entur*	*etur*	*itque*
at	*erant*	*ibus*	*itur*
ate	*erat*	*imus*	*ium*
atem	*ere*	*ione*	*orum*
atis	*erent*	*is*	*os*
averis	*erint*	*isque*	*que*
averit	*eris*	*isset*	*um*
averunt	*erit*	*isti*	*ur*
avi	*eritis*	*it*	*us*
avit	*erunt*	*itas*	*ionem*
ebat	*eruntque*	*itate*	*ionis*

These 60 morphemes are all accurately identified. A comparison of the first iteration with the second iteration morpheme lists shows that the three invalid morphemes have been removed. Following the third and final iteration of stem set and paradigm construction the final morpheme set of 123 members is extracted:

Table 4.47: Latin final morpheme set.

abant	*ati*	*ent*	*ia*
abat	*ato*	*ente*	*ibus*
abit	*atione*	*entem*	*imus*
abitur	*ationem*	*entes*	*ione*
abo	*ationes*	*entia*	*ionem*
abunt	*atis*	*entiam*	*ionis*
ae	*atum*	*entibus*	*is*
am	*atur*	*entis*	*isque*
amus	*aturus*	*entium*	*isse*
andum	*atus*	*entur*	*issent*
ans	*averat*	*erant*	*isset*
ant	*averint*	*erat*	*isti*
ante	*averis*	*ere*	*istis*
antem	*averit*	*erent*	*it*
antes	*averunt*	*erentur*	*itas*
antur	*averuntque*	*eret*	*itate*
ar	*avi*	*erim*	*itatem*
are	*avimus*	*erimus*	*itatis*
arent	*avit*	*erint*	*ite*
aret	*ebant*	*eris*	*itis*
ari	*ebantur*	*erit*	*itque*
arum	*ebat*	*eritis*	*itur*
as	*ebatur*	*ero*	*ium*
asset	*ebit*	*erunt*	*orum*
asti	*em*	*eruntque*	*os*
astis	*emus*	*es*	*que*
at	*endam*	*et*	*um*
ata	*endi*	*ete*	*unt*
atae	*endum*	*etis*	*untur*
ate	*ens*	*etur*	*ur*
atem	*ensque*		*us*

These results may be evaluated against a traditional analysis of Latin accidence with the following results:

Morphemes which are used to construct surface forms of verbs include structures which reflect all of the moods and voices commonly attributed to Latin and many of the tenses. Clearly, the completeness of the results will be dependent on the breadth of the input data set. Classifications which can be made include:

Table 4.49: Latin verbal conjugations found in the final morpheme set.

Verbal Conjugations	Related Morpheme Set
Present Indicative Active (1st Conjugation)	*-as, -at, -amus, -atis, -ant*
Present Indicative Active (2nd Conjugation)	*-es, -et, -emus, -etis, -ent*
Present Indicative Active (3rd Conjugation)	*-is, -it, -imus, -itis, -unt*
Present Indicative Active (4th Conjugation)	*-is, -it, -imus, -itis*
Present Participle Active (1st Conjugation)	*-ans, -antem, -antis, -ante, -antes*
Present Participle Active (2nd and 3rd Conjugation)	*-ens, -entem, -entis, -ente, -entes, -entium*
Present Infinitive Active	*-are, -ere, -ire*
Gerund	*are, -andum, -ere, -endum, -endi, -ire*
Present Imperative Active	*-ate, -ete, -ite*
Present Subjunctive Active	*-es, -et*
Future Indicative Active	*-abo, -abit, -abunt, -ebit, -es, -et, -emus, -etis, -ent*
Future Participle Active	*-ur, -us, -a, -um*
Imperfect Indicative Active	*-abat, -abant, -ebat, -ebant, -ebit*
Imperfect Subjunctive Active	*-aret, -arent, -eret, -erent*
Perfect Indicative Active (1st Conjugation)	*-avi, -asti, -avit, -avimus, -astis, -averunt*
Perfect Indicative Active (3rd Conjugation)	*-isti, -it, -imus, -istis, -erunt*
Perfect Infinitive Active	*-isse*
Perfect Subjunctive Active	*-erim, -eris, -erit, -erimus, -eritis, -erint*
Pluperfect Indicative Active	*-erat, -erant*
Pluperfect Subjunctive Active	*-isset, -issent*
Future Perfect Active	*-ero, -eris, -erit, -erint*
Supine	*-atum*
Present Indicative Passive	*-atur, -antur, -etur, -entur, -untur*
Present Infinitive Passive	*-ari*
Present Imperative Passive	*-are, -ere, -ire*
Gerundive	*-andum, -endum, -endam, -endi*
Present Subjunctive Passive	*-ar, -atur, -antur, -eris, -etur, -entur*
Future Indicative Passive	*-eris, -etur, -entur*
Imperfect Indicative Passive	*-ebatur, -ebantur*

Imperfect Subjunctive Passive	*-erentur*
Perfect Participle Passive	*-atus, -itus, -us, -um*
Present Indicative Deponent	*-atur, -etur, -entur, -itur, -untur*
Present Participle Deponent	*-ans, -antem, -antis, -ante, -antes, -ens,*
	-entem, -entis, -ente, -entes, -entium
Present Infinitive Deponent	*-ari*
Present Subjunctive Deponent	*-ar, -etur, -entur*
Future Indicative Deponent	*-ar, -etur, -entur*
Imperfect Indicative Deponent	*-ebatur, -ebantur*
Future Indicative Semi-Deponent	*-abo*

Such a list demonstrates clearly the difficulties associated with trying to categorise these morphemes automatically. Many are found in more than one grammatical context and coupled with the limitations of a randomly incomplete data set, irrespective of its size, the problem becomes insuperable.

It is the nature of nominals to exhibit lower degrees of inflection. Nevertheless we can discern groups of morphemes in this analysis which can be categorised following traditional Latin grammar, others which perhaps cannot but are nevertheless valid. Nominal morpheme structures generally recognised by grammarians include:

Table 4.52: Latin nominal declensions found in the final morpeheme set.

Declension	Related Morpheme Set
First Declension	*-am, -ae, -as, -arum, -is*
Second Declension	*-us, -um, -os, -orum, -is*
Third Declension	*-is, -em, -es, -um, -ium, -ibus*
Fourth Declension	*-us, -um, -ibus*
Fifth Declension	*-es, -em*

It is interesting that our analysis also generates more Nominal morpheme structures which do not fall within traditional grammatical categories but are none the less valid. These include: *-atione, ationem, -ationes, ia, ione, ionem, ionis, itas, itate, itatem, itatis.* These endings represent types of nouns which tend to have a more abstract meaning. One might argue that the first three structures on this list should be re-written, *-ation-e, -ation-em, -ation-es* or perhaps *-atio-n-e, -atio-n-em, -atio-n-es* since the *n* is present for phonological reasons to allow the *io* syllable to move easily into the next syllable by means of an inserted onset. In any event we can establish their validity by recognising that the forms they construct are cognate with other related forms constructed with the more limited set of morphemes from the main set of nominals. Forms created from morphemes in the *-itas* group also tend to be related to cognate nouns constructed from the principal nominal morpheme set. The *-em* and *-is*

endings are sufficiently well attested to encourage us to rewrite this group as *-ita-s, -ita-t-e, -ita-t-em* and *ita-t-is*. Once again the last three forms show an insert, in this case a *t*, for reasons of pronunciation. Such adjustment is beyond the ability of the processing described here yet the analysis sheds a broader light on many Latin semantic features identifying relationships which are not always clear from more traditional grammars.

In the case of both verbs and nouns morphemes represented by a single character are missing from the list. A review of the stems generated and their associated paradigms, reveals that these morphemes are present in that context.

4.2.4 Example stem sets and paradigm sets

Using these morpheme structures we can now build final *IP* sets and stem sets. Of the items from the original wordlist for which a parse was attempted we find that 99.53% are valid. This is encouraging. Example stem sets and IPs are listed below:

The first example shows three large groupings where cognate verbs and nouns are both present:

Table 4.54: Example Latin stem sets with both nominal and verbal forms.

Stem	Morpheme	Stem	Morpheme	Stem	Morpheme
aedific-	*-a*	*congreg-*	*-a*	*iudic-*	*-a*
	-abant		*-aberis*		*-abant*
	-abat		*-abis*		*-abat*
	-aberis		*-abit*		*-abis*
	-abis		*-abitur*		*-abit*
	-abit		*-abo*		*-abitur*
	-abitis		*-abunt*		*-abo*
	-abitur		*-abuntur*		*-abunt*
	-abo		*-amini*		*-ando*
	-abunt		*-ans*		*-andum*
	-abuntur		*-ant*		*-ans*
	-amus		*-antes*		*-ant*
	-ando		*-antur*		*-antes*
	-andum		*-are*		*-are*
	-ans		*-arentur*		*-arent*
	-ant		*-aret*		*-aret*
	-antes		*-ari*		*-as*
	-antibus		*-as*		*-asti*
	-antium		*-assent*		*-astis*
	-are		*-asset*		*-at*
	-arent		*-asti*		*-ata*
	-aret		*-astis*		*-ate*
	-ari		*-at*		*-ati*
	-as		*-ata*		*-atis*
	-asti		*-atae*		*-atum*
	-astis		*-ate*		*-aturus*
	-at		*-ati*		*-atus*
	-ata		*-atione*		*-averint*
	-atae		*-ationem*		*-averis*
	-ate		*-ationis*		*-averit*
	-ati		*-atis*		*-averunt*
	-atio		*-atus*		*-avi*
	-atione		*-averint*		*-avit*
	-ationem		*-averunt*		*-e*
	-ationes		*-avi*		*-em*
	-ationibus		*-avit*		*-emus*
	-ationis		*-entur*		*-ent*
	-atis		*-et*		*-entur*

aedific-	*-ator*	*congreg-*	*-o*	*iudic-*	*-es*
	-atum				*-et*
	-atur				*-i*
	-aturus				*-ibus*
	-atus				*-is*
	-averat				*-ium*
	-averint				*-o*
	-averis				*-um*
	-averit				
	-averunt				
	-avi				
	-avimus				
	-avit				
	-em				
	-emus				
	-ent				
	-entur				
	-es				
	-et				
	-etur				
	-ium				
	-o				

Table 4.57: Latin verbal paradigms.

Stem	Morpheme	Stem	Morpheme	Stem	Morpheme
ador-	*-a*	*descend-*	*-am*	*mitt-*	*-am*
	-abant		*-amus*		*-amus*
	-abat		*-ant*		*-ant*
	-abis		*-as*		*-antur*
	-abit		*-at*		*-as*
	-abitis		*-atis*		*-at*
	-abo		*-e*		*-atis*
	-abunt		*-ebant*		*-e*
	-amus		*-ebat*		*-ebant*
	-ando		*-ens*		*-ebat*
	-andum		*-ent*		*-endum*
	-ans		*-ente*		*-ens*
	-ant		*-entem*		*-ent*
	-ante		*-entes*		*-entem*
	-antes		*-entibus*		*-entes*
	-are		*-entis*		*-entis*
	-arent		*-entium*		*-entur*
	-aret		*-erant*		*-ere*
	-as		*-erat*		*-erent*
	-asset		*-ere*		*-eret*
	-asti		*-erent*		*-et*
	-at		*-eret*		*-etur*
	-ate		*-erint*		*-i*
	-atis		*-erit*		*-imus*
	-aturus		*-ero*		*-it*
	-averint		*-erunt*		*-ite*
	-averis		*-et*		*-itur*
	-averit		*-i*		*-unt*
	-averunt		*-imus*		*-untur*
	-avi		*-issent*		
	-avit		*-isset*		
	-em		*-isti*		
	-emus		*-it*		
	-ent		*-ite*		
	-es		*-o*		
	-et		*-unt*		
	-etis				
	-o				

Table 4.59: Latin example stem sets with just nominal forms.

Stem	Morpheme	Stem	Morpheme	Stem	Morpheme	Stem	Morpheme
famili-	*-a*	*camel-*	*-i*	*host-*	*-e*	*asin-*	*-a*
	-ae		*-is*		*-em*		*-ae*
	-am		*-o*		*-es*		*-am*
	-arum		*-orum*		*-i*		*-arum*
	-as		*-os*		*-ibus*		*-i*
	-is		*-um*		*-is*		*-is*
			-us		*-ium*		*-o*
							-orum
							-os
							-um
							-us

Of the 10,504 partitions attempted only 52 are invalid:

Table 4.61: Invalid Latin partitions.

aperi-re	*capi-te*	*faci-t*	*nesci-re*	*defer-e*	*adduc-tus*	*pollu-tus*
aperi-t	*consenti-re*	*deveni-re*	*nesci-t*	*defer-ri*	*constitu-tus*	*reduc-tus*
audi-re	*consenti-t*	*deveni-t*	*obaudi-re*	*infer-re*	*cooper-tus*	*sculp-tus*
audi-ri	*cupi-t*	*esuri-re*	*obaudi-t*	*infer-t*	*fini-tus*	*sopi-tus*
audi-t	*custodi-re*	*esuri-t*	*obedi-re*	*perfer-re*	*indu-tus*	
audi-te	*custodi-ri*	*exaudi-tus*	*obedi-t*	*perfer-ri*	*liber-tus*	
audi-tus	*custodi-t*	*incipi-t*	*obici-t*	*offer-re*	*menti-tus*	
capi-t	*custodi-te*	*inspici-t*	*refici-te*	*offer-t*	*oper-tus*	

Analysis of these failures reveals that all can be ascribed to one of three circumstances. The first group represents failure to correctly partition a fourth or third/fourth conjugation verb. These verbs are known to Latinists as *-io* verbs from their first person singular present indicative active ending. Such verbs often generate situations in which the *-i* of the ending is juxtaposed with a second *-i* which begins a suffix morpheme. In these circumstance the two identical vowels elide and rewrite as a single *i*. This introduces an irregularity in the conjugation pattern which the process fails to recognise. Modern Latin grammars represent the elision of the two *i* characters by an *i*-macron. Since the data from which the Latin word list used here was generated is taken from the Latin Vulgate Bible there are no macrons present.

The second group of failures are all in the context of a particularly badly behaved Latin verb - *fero*. Generations of schoolboys have laboured under the burden of correctly identifying and constructing the conjugation of *fero* and its

cognates. It is not surprising that this process after fewer than twenty minutes learning time also fails to handle *fero* correctly.

It might be argued that the third group is really part of the first since every member of this group is derived from a third/fourth conjugation verb. Each of these forms is in fact a past participle form of a verb. As in group one the confusion arises as consequence of elision of adjacent vowels and the subsequent loss of the elided vowel in the rewrite of the past participle form. Whilst this behaviour is in fact fairly regular and consistent the current processing is not able to identify it. It would certainly be possible to code for this kind of irregularity but that would prejudice the analysis for other languages.

Elsewhere in the analysis examples can be found of partitions which are valid but which linguists would prefer to categorise differently. Where particular forms of a word are constructed by use of suppletives we do not expect this analysis to recognise these irregularities. An English example would be the verb *go* which forms its past tense via the suppletive *went* which properly belongs to the verb *wend*. More general is the possibility that morpho-phonemic changes occur at the stem-morpheme boundary, much as discussed above in the case of fourth conjugation verbs. An example of this phenomenon can be found in the analysis of the verb *educo* - lead out:

Table 4.63: An example of Latin morpho-phonological changes.

Stem	Morpheme	Stem	Morpheme
educ-	*-a*	edux-	*-erat*
	-abit		*-erim*
	-am		*-eris*
	-as		*-erit*
	-at		*-ero*
	-avit		*-erunt*
	-ens		*-eruntque*
	-ent		*-i*
	-entem		*-issent*
	-entes		*-isset*
	-entium		*-isti*
	-entur		*-istis*
	-ere		*-it*
	-erent		*-itque*
	-eret		
	-et		
	-i		
	-is		
	-it		
	-ite		
	-unt		

In the first two columns can be seen the analysis for the *educ-* stem and then in the third and fourth column the various perfectives found which form their perfect by stem mutation. The mutation in this case is the insertion of an s after the c at the end of the stem. The s and c then combine in crasis and rewrite as x giving the related stem form *edux-*. It seems likely that Latin verbs that behave this way are either displaying a common ancestry with Greek verbs that form their aorist by σ insertion between stem and ending or perhaps have been influenced by this Greek form over time. Whilst linguists can recognise that these two stems are closely related and indeed form different parts of the same conjugation this process cannot make such identifications.

4.2.5 Summary

Of the 10,504 parsings generated by this process 52 proved invalid. This represents an accuracy of 99.53%. Such a result is pleasing for an unsupervised process.

4.3 Kiswahili

As a relatively well-known member of the Bantu language group which uses only standard Latin orthography to represent its characters Kiswahili is a practical choice as a demonstration language for this process. It is also a good example of the highly inflected, agglutinative morphology common to many Bantu languages. Since the primary context for which this process was designed is the analysis of highly-inflected, agglutinative vernacular languages Kiswahili is an excellent test for the process, combining as it does, the main linguistic characteristics of the target languages and relatively wide use as the principal *lingua franca* in much of East Africa. Kiswahili has been much studied with the result that more traditional grammars [3] and dictionaries [27] are available as a basis for reviewing the analysis. As with the Latin and English examples this report focuses on just the principal inflection vector, in the case of Kiswahili this is prefixal.

4.3.1 Inflection rate and PIV

Using the same assessment technique as described above the following results are returned for a Kiswahili word list of 40,790 words. By comparison to the equivalent English word list extracted from the equivalent corpus of text and having removed proper-names from both lists this represents an inflection rate of 1:3.18 for Kiswahili. That this figure should be lower than that for Latin is, as noted above, surprising and may call into question the accuracy of a corpus based assessment of inflection rate. Given the highly inflected nature of Kiswahili it may indicate that the vocabulary set used in the Kiswahili text is smaller than that for the equivalent Latin text. Alternatively it may indeed be a valid assessment, reflecting perhaps the more highly inflected nature of Latin nominals in comparison to Kiswahili nominals. Nevertheless the main purpose of the inflection assessment process is to determine the primary inflection vector the results of which appear below:

A strong signal was generated for a Prefixal PIV:

Table 4.65: Kiswahili initial and final N-gram assessment.

Initial 2-grams	Occurrence	Final 2-grams	Occurrence
wa-	6086	*-ia*	4176
ku-	3223	*-wa*	3761
ni-	2984	*-ka*	2898
ha-	2213	*-ha*	2072
al-	2058	*-za*	2021
$\frac{p}{s} = \frac{16564}{14928} = 1.11$	16564	$\frac{s}{p} = \frac{14928}{16564} = 0.90$	14928

Initial 3-grams	Occurrence	Final 3-grams	Occurrence
ali-	2038	*-sha*	1711
aka-	1515	*-iwa*	1207
wak-	1514	*-ika*	1140
ata-	1357	*-nda*	845
wal-	1312	*-lia*	840
$\frac{p}{s} = \frac{7736}{5743} = 1.35$	7736	$\frac{s}{p} = \frac{5743}{7736} = 0.74$	5743

Inital 4-grams	Occurrence	Final 4-grams	Occurrence
wali-	1287	*-isha*	1072
waka-	1078	*-enda*	536
nita-	995	*-anya*	536
wata-	886	*-ikia*	491
aliy-	688	*-liwa*	444
$\frac{p}{s} = \frac{4934}{3079} = 1.60$	4934	$\frac{s}{p} = \frac{3079}{4934} = 0.62$	3079

Initial 5-grams	Occurrence	Final 5-grams	Occurrence
aliye-	504	*-fanya*	370
ataka-	480	*-tenda*	259
walio-	447	*-uliwa*	242
nitak-	331	*-ambia*	238
watak-	323	*-ishwa*	229
$\frac{p}{s} = \frac{2085}{1338} = 1.56$	2085	$\frac{s}{p} = \frac{1338}{2085} = 0.64$	1338

The sum of the prefix indicators is found to be 6.6 whereas that for suffix indicators amounts to 3.9. An assessment of boundary N-grams thus returns a strong preference for prefix as the preferred inflection vector for Kiswahili. To confirm this the same calculation is made for N-grams in initial+1 and final-1 positions.

Table 4.70: Kiswahili initial+1 and final-1 N-gram assessment.

Initial+1 2-grams	Occurrence	Final-1 2-grams	Occurrence
_*li*-	3344	-*sh*_	2099
_*ta*-	2903	-*ik*_	1612
_*ka*-	2610	-*iw*_	1386
_*ak*-	2140	-*an*_	1248
_*al*-	1909	-*nd*_	1057
$\frac{p}{s} = \frac{12906}{7402} = 1.74$	12906	$\frac{s}{p} = \frac{7402}{12906} = 0.57$	7402

Initial+1 3-grams	Occurrence	Final-1 3-grams	Occurrence
_*ali*-	1851	-*ish*_	1346
_*ita*-	1540	-*end*_	645
_*ata*-	1312	-*any*_	640
_*aka*-	226	-*iki*_	595
_*tak*-	1142	-*liw*_	487
$\frac{p}{s} = \frac{7071}{3713} = 1.90$	7071	$\frac{s}{p} = \frac{3713}{7071} = 0.52$	3713

Initial+1 4-grams	Occurrence	Final-1 4-grams	Occurrence
_*taka*-	949	-*fany*_	439
_*liye*-	582	-*tend*_	311
_*itak*-	474	-*ambi*_	278
_*alio*-	457	-*siki*_	265
_*liyo*-	442	-*ishw*_	265
$\frac{p}{s} = \frac{2904}{1558} = 1.86$	2904	$\frac{s}{p} = \frac{1558}{2904} = 0.53$	1558

Initial+1 5-grams	Occurrence	Final-1 5-grams	Occurrence
_*takay*-	379	-*chuku*_	193
_*livyo*-	332	-*ifany*_	174
_*ataka*-	307	-*fanyi*_	157
_*takap*-	296	-*tafut*_	140
_*itaka*-	282	-*unguk*_	36
$\frac{p}{s} = \frac{1596}{800} = 0.36$	1596	$\frac{s}{p} = \frac{800}{1596} = 2.75$	800

As for the boundary N-grams, each of these results shows a strong preference for a primarily prefixal inflection. The sum of the prefix indicators is 8.50 whereas that for suffixes is just 3.14.

4.3.2 Initial partitioning

A total of 42,857 unique affixes are generated by the initial partition. As for Latin and English a minimum stem length of four characters is set. The initial evidence reduces this to 174 affixes which occur more than the threshold 128 times. The 100 best attested are reproduced in the table below:

Table 4.75: Initial hypomorph table for Kiswahili.

Morph	Occ.	Morph	Occ.	Morph	Occ.	Morph	Occ.
wa	6010	*uk*	741	*na*	441	*nika*	303
ku	3153	*li*	706	*nil*	439	*ita*	296
ni	2959	*aliy*	682	*mta*	433	*hat*	295
ha	2153	*ul*	644	*walio*	428	*mka*	289
al	2040	*uli*	622	*nili*	428	*atakay*	287
ak	2020	*ki*	612	*us*	426	*ik*	285
ali	2008	*wam*	594	*an*	422	*ml*	282
wak	1485	*zi*	584	*asi*	412	*alip*	280
aka	1470	*atak*	564	*aki*	406	*akam*	277
at	1429	*ma*	560	*nime*	400	*haku*	269
ata	1337	*um*	558	*wana*	383	*msi*	268
wal	1291	*uka*	516	*usi*	379	*wasi*	267
wali	1252	*wan*	510	*il*	375	*mli*	267
hu	1074	*mt*	507	*si*	368	*mn*	265
waka	1035	*aliye*	487	*ili*	363	*ana*	261
tu	1033	*nik*	480	*it*	344	*kut*	258
nit	1003	*haw*	479	*kum*	336	*kuk*	255
nita	984	*mk*	473	*waki*	327	*utak*	253
wat	964	*as*	472	*ms*	326	*alipo*	252
ya	963	*hawa*	472	*hak*	325	*kui*	245
am	953	*ataka*	472	*was*	322	*atakaye*	245
wata	864	*nim*	460	*un*	321	*wataka*	243
ut	854	*ume*	460	*nitak*	316	*im*	242
uta	758	*vi*	447	*watak*	310	*ime*	221
ame	746	*wame*	443	*kuwa*	309	*mw*	220

As in the previous examples there are many accurate assessments here but there are also many potentially misleading suggestions. Of the ten most highly scored hypomorphs six are valid *{wa, ku, ni, ha, ali, aka}* and four *{al, ak, wak,*

at} are incorrect. The four invalid morphs represent part of common morpheme structures thus: *'al'* is a truncation of *'ali'*, *'ak'* is drawn largely from *'aka'* and *'aku'*, *'wak'* from *'waka'* and *'waku'* and *'at'* from *'ata'* and *'ati'*. These results are not dissimilar to the Latin results.

Adjusting the scores for all these hypomorphs by removing from shorter morphemes the score of other longer morphemes with common final n-grams, and which are themselves achieving scores in excess of $\frac{\theta^m}{2}$, allows us to rewrite the initial analysis thus:

Table 4.77: Kiswahili initial hypomorphs adjusted for sub-structures.

Morph	Occ.	Morph	Occ.	Morph	Occ.	Morph	Occ.
waka	595	*waki*	327	*mna*	207	*mtaka*	154
ma	560	*kuwa*	309	*akawa*	205	*nili*	151
wata	544	*nika*	303	*alivyo*	201	*iliyo*	151
ata	524	*asi*	302	*ika*	200	*wanao*	149
wa	506	*ku*	291	*ili*	198	*uki*	147
hu	505	*mka*	289	*am*	194	*kuni*	146
uta	493	*akam*	277	*wakam*	193	*mi*	145
ame	477	*vi*	274	*nali*	193	*ya*	140
ume	460	*uli*	274	*ni*	191	*kuku*	139
wame	443	*haku*	269	*li*	189	*ulio*	138
wali	442	*msi*	268	*ha*	188	*awa*	137
aka	429	*wasi*	267	*ana*	187	*wakawa*	137
walio	428	*mli*	267	*ita*	183	*tuta*	136
aki	406	*mta*	257	*una*	182	*hai*	136
aliye	405	*alipo*	252	*yata*	176	*hau*	136
nime	400	*kui*	245	*hata*	176	*wam*	134
ki	398	*atakaye*	245	*tuli*	171	*zi*	134
usi	379	*na*	241	*nitawa*	169	*ham*	133
ali	376	*wana*	229	*kuu*	163	*sita*	132
nita	367	*ime*	221	*walipo*	159	*kuli*	131
uka	363	*tu*	219	*aliyo*	158	*mki*	130
kum	336	*hawaku*	210	*mme*	158		

The readjustment of morpheme value reduces the number of hypomorphs to 87 but all of these can be considered as valid Kiswahili morphemes or morpheme stacks. To have accurately identified 87 morphemes or morpheme stacks at this stage of the process is pleasing. Since the process is unaware of the success rate it proceeds to validate the list of hypomorphs by generating stem lists and inflection paradigms which in turn allow the valid morpheme set to be further extended.

4.3.3 Validating hypomorphs

These morphemes generate a stem list, once the morpheme list for each stem is extended across the word list and stem conflicts are resolved, of 4,180 putative stems. This is far fewer than were found at this stage in the Latin analysis. The highly inflected nature of Kiswahili is probably responsible for this perhaps in conjunction with a rather more limited lemmata set by comparison to Latin. In addition to the high rate of prefixal inflection the level of suffixal inflection is such that at this stage in the process accurate stem identification is much more difficult. Conversely, this same high rate of inflection allows a much larger set of Inflection Paradigms to be constructed of 2,540 members. Having adjusted these paradigms to account for common final N-grams in stems the following list of 94 banker morphemes can be extracted:

Table 4.79: Kiswahili banker morpheme set - 1st iteration.

aka	*hata*	*kuni*	*nili*	*ulio*	*walio*
akai	*hawaku*	*kuzi*	*nilipo*	*ulipo*	*walipo*
akam	*hawata*	*lililo*	*nime*	*ume*	*walivyo*
aki	*hu*	*lita*	*nita*	*umeni*	*wame*
ali	*huku*	*mka*	*nitai*	*usi*	*wana*
alipo	*huta*	*mki*	*nitaka*	*uta*	*wasi*
aliye	*ika*	*mli*	*nitakapo*	*utaka*	*wasio*
aliyo	*iliyo*	*mme*	*palipo*	*utakapo*	*wata*
ame	*ime*	*mna*	*siku*	*utam*	*wataka*
ana	*isiyo*	*mta*	*sita*	*vilivyo*	*watakao*
asipo	*ita*	*mtaka*	*tuka*	*wa*	*watakapo*
asiye	*kilicho*	*mtakapo*	*tuli*	*waka*	*yali*
ata	*ku*	*na*	*tuta*	*wakai*	*yata*
ataka	*kui*	*nali*	*uka*	*wakam*	*zilipo*
atakapo	*kuli*	*nika*	*uki*	*waki*	
haku	*kum*	*niki*	*uli*	*wali*	

Once again this set is 100% valid. A second iteration of the stem and paradigm building process extends this set to 148 members:

Table 4.81: Kiswahili banker morpheme set - 2nd iteration.

a	*ataku*	*kum*	*nili*	*uliyo*	*wata*
aka	*atam*	*kuni*	*nilipo*	*ume*	*wataka*
akai	*haita*	*kuwa*	*niliye*	*umeni*	*watakao*
akam	*haku*	*kuya*	*niliyo*	*una*	*watakapo*
akau	*haliku*	*kuzi*	*nime*	*usi*	*ya*
akawa	*hamku*	*lililo*	*nina*	*uta*	*yali*
aki	*hamta*	*lime*	*nita*	*utaka*	*yaliyo*
ali	*hata*	*lina*	*nitai*	*utakapo*	*yame*
alilo	*hatuku*	*lita*	*nitaka*	*utam*	*yata*
alio	*hawaku*	*m*	*nitakapo*	*utawa*	*zilipo*
alipo	*hawata*	*mka*	*palipo*	*vika*	*zilizo*
alivyo	*hu*	*mki*	*siku*	*vilivyo*	*zita*
aliye	*huku*	*mli*	*sita*	*wa*	
aliyo	*huta*	*mlio*	*tu*	*waka*	
ame	*ika*	*mlivyo*	*tuka*	*wakai*	
ameni	*ili*	*mme*	*tuki*	*wakam*	
ana	*ilipo*	*mna*	*tuli*	*waki*	
ange	*iliyo*	*mnao*	*tume*	*wali*	
asi	*ime*	*msi*	*tuta*	*walio*	
asipo	*ina*	*mta*	*twa*	*walipo*	
asiye	*isiyo*	*mtaka*	*u*	*walivyo*	
ata	*ita*	*mtakapo*	*uka*	*waliyo*	
ataka	*itakapo*	*na*	*uki*	*wame*	
atakapo	*kilicho*	*nali*	*uli*	*wana*	
atakavyo	*ku*	*ni*	*ulio*	*wanao*	
atakaye	*kui*	*nika*	*ulipo*	*wasi*	
atakayo	*kuli*	*niki*	*ulivyo*	*wasio*	

These 148 valid morphs represent the final banker set which will be used to generate the final set of stems and inflection paradigms. The final set of stems generated from these morphemes numbers 14,602 partitions of which 40 were invalid (99.99% validity). The failing partitions are:

Table 4.83: Failing partitions generated by the Kiswahili analysis.

Stem	Invalid Partition	Stem	Invalid Partitions
agia	*kuf-agia*	*ishika*	*kuz-ishika*
amka	*kut-amka*		*nimez-ishika*
andika	*kut-andika*	*isikia*	*wal-isikia*
iasi	*k-iasi*		*hun-isikia*
	kun-iasi		*kun-isikia*
	wamen-iasi	*isimulia*	*kuz-isimulia*
ichague	*n-ichague*		*nimez-isimulia*
ichimbia	*wamen-ichimbia*	*itamani*	*ak-itamani*
idharau	*hun-idharau*	*itengenezea*	*kun-itengenezea*
ifichia	*wamen-ifichia*	*itia*	*wamen-itia*
	kun-ifichia		*hun-itia*
igawa	*wal-igawa*		*kun-itia*
iharibu	*kuz-uharibu*	*ngazeni*	*ta-ngazeni*
	nimez-iharibu	*ubadili*	*h-ubadili*
	ak-iharibu		*k-ubadili*
	n-iharibu	*unywea*	*k-unywea*
ihesabu	*ak-ihesabu*	*uonee*	*ak-uonee*
imwaga	*wal-imwaga*	*usamehe*	*kut-usamehe*
inuke	*n-inuke*	*weka*	*kut-weka*
isha	*kuf-isha*	*zameni*	*ta-zameni*

In the vast majority of cases these failures are the result of the process not being able to identify a morpheme structure with a single character vocalic morpheme in stack final position. Those structures not found are: {*atakayei-, usii-, wamei-, aliyei-, atakavyoi-, waliyoi-, wasii-, watakapoi-, hamkui-, waliu-, usiu-, nitau-*}. The remaining errors are generated by stems where one stem is a subset of another stem where the longer stem begins with an extra character as in *kuf-agia*, generated by *ku-agia*, *kuf-isha,* generated by *ilipo-isha, kut-amka* by *ku-tamka*, *kut-andika* by *ku-tandika, kut-weka by ku-tweka* , *ta-ngazeni* by *lita-ngazeni* and *ta-zameni* by *nita-zameni.* Whilst these errors are disappointing, placed in the context of 14,562 valid partitions the results as a whole are encouraging.

The final morpheme set is derived from these inflection paradigms and contains 326 morphemes or morpheme structures of which 315 (96.63%) are valid. Invalid morphemes are marked so: *.

Table 4.85: Kiswahili final morpheme set.

a	*atakavyo*	*k**	*mnayo*	*tulipo*	*wakam*
ai	*atakaye*	*ka*	*mnayoya*	*tume*	*wakawa*
*ak**	*atakayo*	*ki*	*msi*	*tusi*	*wakazi*
aka	*ataku*	*kika*	*msii*	*tuta*	*waki*
akai	*atam*	*kili*	*mta*	*tutakavyo*	*wakim*
akaji	*atau*	*kilicho*	*mtaka*	*twa*	*wakuo*
akaki	*atawa*	*kime*	*mtakapo*	*u*	*wal**
akali	*ataya*	*kisicho*	*mtakavyo*	*ui*	*wali*
akam	*au*	*kita*	*mtakayo*	*uka*	*walii*
akamw	*h**	*kitakacho*	*mtazi*	*ukani*	*waliko*
akani	*ha*	*ku*	*mwa*	*ukawa*	*walio*
akau	*hai*	*kuf**	*mwana*	*uki*	*walipo*
akavi	*haija*	*kui*	*n**	*uli*	*walivyo*
akawa	*haita*	*kuki*	*na*	*ulini*	*waliyo*
akaya	*haja*	*kuku*	*nali*	*ulio*	*walizo*
akazi	*haku*	*kuli*	*nawa*	*ulipo*	*wam*
aki	*hali*	*kum*	*ni*	*ulivyo*	*wame*
akim	*haliku*	*kume*	*nika*	*uliye*	*wamen**
aku	*hamku*	*kun**	*nikam*	*uliyo*	*wamewa*
ali	*hamta*	*kuni*	*niki*	*ulizo*	*wameya*
alilo	*hata*	*kupa*	*nili*	*ulizowa*	*wamw*
alim	*hatuku*	*kut**	*nililo*	*ume*	*wana*
alio	*hauku*	*kuu*	*nilio*	*umem*	*wanao*
aliou	*hauta*	*kuvi*	*niliowa*	*umeni*	*wange*
aliowa	*hawa*	*kuwa*	*nilipo*	*umeya*	*wani*
alipo	*hawaku*	*kuya*	*nilivyo*	*una*	*wasi*
alipom	*hawata*	*kuz**	*niliye*	*unao*	*wasio*
alitu	*hayata*	*kuzi*	*niliyo*	*unge*	*wasipo*
alivi	*hazita*	*kwa*	*nime*	*uni*	*wata*
alivyo	*hu*	*li*	*nimeku*	*usi*	*wataka*
aliwa	*hui*	*lika*	*nimeya*	*usii*	*watakao*
aliye	*huja*	*lil*	*nimez**	*usio*	*watakapo*
aliyo	*huku*	*lililo*	*nina*	*usipo*	*wataku*
alizo	*hum*	*lime*	*ninge*	*uta*	*watam*
ame	*hun**	*lina*	*nisi*	*utai*	*wazi*
amei	*huta*	*lisi*	*nisipo*	*utaka*	*ya*
ameku	*huwa*	*lisilo*	*nita*	*utakapo*	*yaka*
ameli	*huzi*	*lita*	*nitai*	*utakavyo*	*yaki*
amem	*i*	*litakapo*	*nitaka*	*utakayo*	*yali*

ameni	*ika*	*m*	*nitakapo*	*utaku*	*yaliyo*
ametu	*ikam*	*ma*	*nitaku*	*utam*	*yame*
amewa	*ili*	*mka*	*nitam*	*utavi*	*yasi*
ana	*ilipo*	*mki*	*nitawa*	*utawa*	*yasiyo*
anam	*ilivyo*	*mkii*	*paka*	*uu*	*yata*
anavyo	*iliyo*	*mli*	*palipo*	*uwa*	*yatakapo*
anaye	*ime*	*mlio*	*pame*	*vi*	*yua*
ange	*ina*	*mlivyo*	*pata*	*vika*	*za*
asi	*inayo*	*mliyo*	*siku*	*vilivyo*	*zi*
asingali	*isi*	*mme*	*sita*	*vime*	*zili*
asipo	*isiyo*	*mna*	*ta**	*vya*	*zilivyo*
asiye	*ita*	*mnalo*	*tu*	*wa*	*zilizo*
ata	*itakapo*	*mnao*	*tuka*	*waka*	*zina*
ataka	*itakavyo*	*mnaowa*	*tuki*	*wakai*	*zisi*
atakapo	*itakayo*	*mnaya*	*tuli*	*wakali*	*zita*

For a highly inflectional and agglutinative language such as Kiswahili this is an excellent result.

4.3.4 Example stem sets and paradigm sets

Taking the 148 morphemes derived from the first part of the process new stems sets are built. For Kiswahili this generates 2,297 stems. Example stems with their associated prefixal inflections include:

Table 4.88: Inflection paradigm for Kiswahili verb *fanya*.

a	*ange*	*isiyo*	*mna*	*tuki*	*wakali*
ai	*asi*	*ita*	*mnalo*	*tuli*	*wakam*
aka	*asiye*	*itakapo*	*mnaya*	*tume*	*wakawa*
akai	*ata*	*itakavyo*	*msi*	*tuta*	*wakazi*
akaji	*atakapo*	*itakayo*	*msii*	*tutakavyo*	*waki*
akaki	*atakavyo*	*ka*	*mta*	*twa*	*wali*
akali	*atakaye*	*ki*	*mtakavyo*	*ui*	*walii*
akam	*atakayo*	*kilicho*	*mtakayo*	*uka*	*walio*
akau	*ataku*	*kime*	*mtazi*	*ukani*	*walipo*
akavi	*atam*	*kita*	*mwa*	*ukawa*	*walivyo*
akawa	*atau*	*ku*	*na*	*uki*	*wam*
akaya	*atawa*	*kui*	*nali*	*uli*	*wame*
akazi	*haita*	*kuki*	*ni*	*ulio*	*wamewa*
aki	*haja*	*kuku*	*nika*	*ulivyo*	*wameya*
aku	*haku*	*kuli*	*nikam*	*uliye*	*wana*
ali	*haliku*	*kum*	*niki*	*ume*	*wanao*
alilo	*hamta*	*kuni*	*nili*	*umem*	*wani*
alim	*hata*	*kuu*	*nililo*	*umeya*	*wasi*
aliou	*hatuku*	*kuvi*	*niliowa*	*una*	*wasio*
aliowa	*hawa*	*kuwa*	*nilivyo*	*unao*	*wata*
alipo	*hawaku*	*kuya*	*niliye*	*uni*	*watakapo*
alivi	*hayata*	*kuzi*	*nime*	*usi*	*watam*
alivyo	*hu*	*li*	*nimeku*	*usipo*	*wazi*
aliwa	*hui*	*lika*	*nimeya*	*uta*	*ya*
aliye	*hum*	*lililo*	*nisi*	*utai*	*yaka*
aliyo	*huta*	*liliyo*	*nita*	*utakapo*	*yali*
alizo	*huwa*	*lime*	*nitai*	*utakayo*	*yaliyo*
ame	*huzi*	*lisi*	*nitakapo*	*utavi*	*yame*
amei	*ika*	*lisilo*	*nitaku*	*utawa*	*yasi*
ameku	*ili*	*lita*	*nitam*	*uu*	*yata*
ameli	*ilipo*	*m*	*nitawa*	*uwa*	*yatakapo*
amem	*iliyo*	*mka*	*palipo*	*vi*	*zi*
ameni	*ime*	*mki*	*siku*	*vika*	*zili*
ametu	*ina*	*mli*	*sita*	*vilivyo*	*zilivyo*
amewa	*inayo*	*mlivyo*	*tu*	*wa*	*zilizo*
ana	*isi*	*mme*	*tuka*	*waka*	*zime*
				wakai	*zita*

A full analysis of this table would run to many pages, some example conjugations drawn from the analysis include:

Table 4.90: Example conjugations from the *fanya* analysis.

Conjugation / Tense	Members
Direct Imperative	*0 (null)*
Polite Imperative	*u-, m-*
Imperative	*usi-, msi-.*
Present (Affirmative)	*una-, mna-, wana-.*
Present (Negative)	*hu-, hawa-.*
Present Perfect (Affirmative)	*nime-, ume-, ame-, tume-, mme-, wame-.*
Subjunctive (Affirmative)	*ni-, a-, tu-, m-, wa-.*
Subjunctive (Negative)	*nisi-, usi-, asi-, msi- wasi-.*
Simple Past (Affirmative)	*nili-, uli-, ali-, tuli-, mli-, wali-.*
Simple Past (Negative)	*haku-, hatuku-, hawaku-.*
Future (Affirmative)	*nita-, uta-, ata-, tuta-, mta-, wata-.*
Future (Negative)	*huta-, hata-, hamta-.*
Present Conditional (Affirmative)	*ange-.*
KI - Affirmative	*niki-, uki-, aki-, yuki-, waki-.*
KI - Negative	*usipo-.*
KA - Affirmative	*nika-, uka-, aka-, tuka-, mka-, waka-.*
Present Relative (Affirmative)	*wanao-.*
Past Relative (Affirmative)	*niliye-, uliye-, aliye-, walio-.*
Future Relative (Affirmative)	*atakaye-.*
Negative Relative	*asiye-, wasio-.*
General Relative	*ni-, u-, a-, tu-, m-, wa-.*
Infinitive	*ku-.*

Nominal inflections can also be found representing various classes of nouns in Kiswahili.

Table 4.92: Kiswahili nominal paradigms.

	Lemma	Forms
M/WA Gender	*dudu*	*m-dudu, wa-dudu*
	kulima	*wa-kulima*
	pishi	*wa-pishi*
	vulana	*wa-vulana*
JI/MA	*gunia*	*ma-gunia*
	jembe	*ma-jembe*
	neno	*ma-neno*
	jumba	*ma-jumba*
KI/VI	*kapu*	*ki-kapu, vi-kapu*
	kombe	*ki-kombe, vi-kombe*
	pande	*ki-pande, vi-pande*
	tabu	*ki-tabu, vi-tabu*
	kula	*vya-kula*
	banda	*ki-banda, vi-banda*
U/MA	*gonjwa*	*ma-gonjwa*
	kati	*wa-kati*

The relatively low rate of nominal inflection in Kiswahili combined with the tendency for single character prefixes to mutate for morpho-phonological reasons conspire to limit the completeness of the results. Little conclusive evidence can be found for the M/MI and N classes of nouns. Nevertheless, those nominal morphemes which have been identified allow us to identify the other classes within the results. Conversely, the tendency of Kiswahili adjectives to demonstrate agreement with the noun provides sufficient evidence for the process to identify significant numbers of concordant adjectives:

Table 4.94: Kiswahili adjectival paradigms.

Adjective	Forms
baya	*ki-, vi-, m-, wa-, ma-, u-, ku..*
chungu	*ki-, vi, ma-.*
dogo	*ki-, vi-, wa-, ma-.*
kali	*ki-, vi-, wa-, ma-.*
kubwa	*ki-, vi-, wa-, ma-.*
tupu	*ki-, vi-, ma-.*
vivu	*wa-.*
zima	*ki-, ku..*
zuri	*ki-, ma-, vi-, wa, ku-.*

These represent five, possibly all six, Kiswahili Noun classes. Both singular and plural forms of the KI/VI and M/WA groups are present as is the singular of U/N and the plural from JI/MA. The infinitive marker *ku* is also present.

Using these morphemes and morpheme structures to parse the original word list of 40,790 words generates 35,195 partitions. 508 invalid partitions are generated by the 11 invalid morphemes listed above. In most cases these morphemes were generated as a result of an uncommon form where the prefix stack was not wholly identified as noted above. An exception is the (invalid) *ta-* morpheme. This seems to have been the consequence of a significant group of Arabic loan words within Kiswahili, many of which begin with the characters *ta-* as in *tafadhali*. The output also contains words which have been partitioned by the process at positions which are valid but not optimal. For example: *[ki]fanya:*

Table 4.96: Example Kiswahili sub-optimal partition.

alivyo-kifanya	*nilivyo-kifanya*	*uta-kifanya*
aliye-kifanya	*nita-kifanya*	*utakavyo-kifanya*
ame-kifanya	*nitakapo-kifanya*	*waka-kifanye*

In these cases all these partitions have failed to identify the final *ki* morpheme in the prefix stack. Nevertheless the partitions are all placed on valid morpheme boundaries and so are considered valid but not optimal. Such sub-optimal partitions occur when valid morpheme stacks are masked by the overwhelming strength of a subset of the stack as a morpheme stack in its own right. Conversely, the agglutinative nature of Kiswahili contributes to invalid partitions where the opening syllable of a stem is identical to a valid prefix morpheme. In these cases the stem is incorrectly divided by the process and the first characters of the stem attributed to the morpheme stack as in *kanyageni* and *kimbia*, incorrectly partitioned as *ka-nyageni* and *ki-mbia*. Whilst these partitions are not valid the identification of *ka-* and *ki-* as valid prefix morphemes stands.

4.3.5 Summary

Of the 14,602 primary partitions generated by the process only 40 (<00.01%) were invalid. Such a result suggests that this processing has a good chance of handling many of the morphological complexities of highly agglutinative languages such as Kiswahili. Such errors as occur can be ascribed to low occurrence morpheme structures the evidence for which is very limited or to a confusion in stem identification between two very similar stems.

Chapter 5

System design

Constructing the prototype system described in this report offered an opportunity to investigate how suitable Object Oriented (OO) techniques are for designing systems for linguistic processing. Early models of the process were procedural in their design and whilst few problems were experienced with the procedural approach the benefits of an OO approach seemed significant. The ability of classes to evolve as the system took shape was of great help and enabled the development of a suite of objects which encapsulate the behaviour of the principal linguistic components of the process. Although developed initially in the context of this research these objects form a library with much wider application in other areas of linguistic processing and with the potential for further development. The objects required to encode the required behaviours may be considered in two groups. The first group make the initial assessments of the type of inflection present in the word list and extract the initial set of hypomorphs. The second group take these hypomorphs and extend them by reference to the stems and paradigms they imply. The extended set of hypomorphs is then used to generate an implied stem list which may then be used to partition the remainder of the word list if desired.

The current implementation is in Java 5 and was developed using the NetBeans 5.5 IDE. Java is well suited to this task. It is a mature, full-featured language which provides all the standard features of traditional languages such as C and C++ but packaged within a run-time environment which removes much of the pain from developing applications. Its Object Orientated history allows existing classes to be extended as required and its Unicode compatibility (at least for 16-bit Unicode characters) enables most character based languages to be processed without the need to handle different code pages and custom font mappings. The cross-platform abilities of Java ensure wide accessibility for the system on Unix based operating systems, Windows platforms and even some mainframes. In short, any platform that can run the Sun Java JRE can mount this process.

5.1 Initial assessments and partitioning

A goal of this work was to determine what parameters might be set for the process automatically and what could not. The initial processing stage covers the task of acquiring a word list to process and examining the data within that word list for linguistic characteristics that may be used to identify the type of inflection most prevalent in the data - prefixal or suffixal. This is a key parameter for the main part of the analysis as experiment confirmed that the best results were obtained by focusing first upon the class of affix most commonly used. A managing class was designed to handle this initial phase by gathering the parameters for the main process either directly from the input data wherever possible or directly from the user where necessary. Parameters required from the user are initially limited to the source text file from which the word list is to be loaded and the locale to be used by the process. An early decision was made to support only Unicode encoded text [10]. The use of Unicode allows the process to take advantage of the pre-defined collation sequences in the Unicode Collation Algorithm (UCA) [14]. Unicode also provides character information which can be used to identify non-word forming characters such as punctuation. Non-Roman Scripts can pose particular problems often presenting complex orthographies where not all the characters in a language are represented as single glyphs. This scenario is further complicated within Unicode by the presence of legacy encodings for some characters, typically in the extended Latin set. For example a lower case e with an acute accent may be encoded as a single Unicode code point: u00E9 and alternatively as u0065 + u0301. These two types of encoding are often referred to as 'composed' and 'decomposed'. The UCA defines these two variants as equal. This, at least in theory, allows data originating from different sources to be compared. In practice, implementation of the UCA by software manufacturers is patchy, tending to focus upon languages where the greatest commercial return is thought to be. Providentially a comprehensive set of routines to handle normalisation and comparison issues exist in the International Components for Unicode package (ICU4J), this open source project is sponsored by IBM.

Having loaded and normalised the data set as composed characters the data is now transformed to lower-case and where possible proper names are removed. Names can only be removed for languages where they are consistently identified by a leading capital. The identification of proper names is significant in that it removes a significant source of confusion from the analysis. Proper names are not usually translated across languages, more often they are transliterated and bring with them elements of the morphology of their original language. This is of particular significance where a large body of proper names may be present all sharing a common linguistic heritage. If they are not removed from the analysis at this stage, misleading results may be generated. In the particular context of this work the word lists were all generated from biblical texts. This contains a set of proper names derived largely from Hebrew and exhibiting some of the characteristics of Hebrew morphology. As noted above these characteristics will, for example, generate a strong signal in an English word list for a suffix

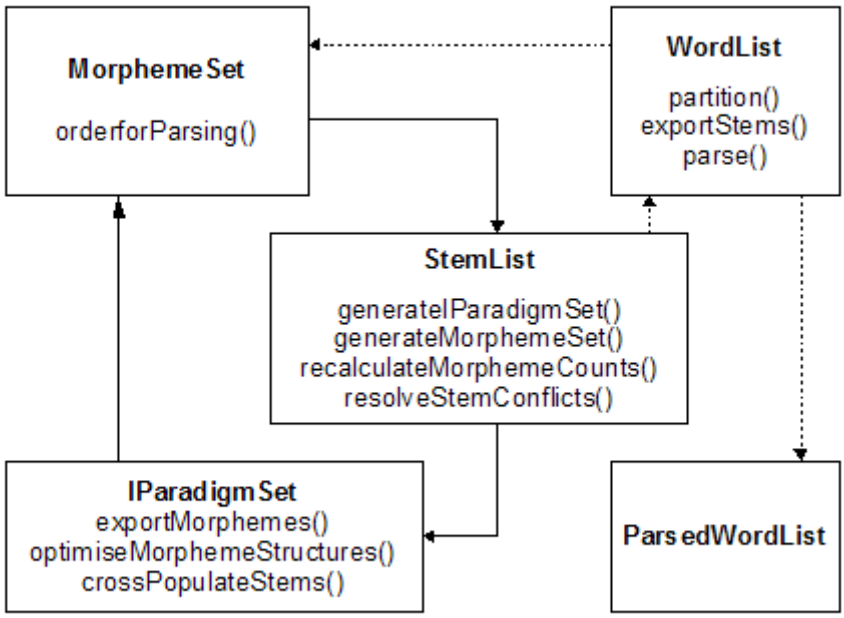
morpheme *ah* with the consequence that *-ah* hypomorph is proposed which can generate spurious partitions.

The loaded word list is also used to generate a character set for the language and an assessment of PIV is made as described above. The final part of the pre-process is the generation of the initial set of hypomorphs from the list of partitions of each word. The user is invited to set a minimum stem length before partitioning. In the current prototype the partitioning of the words in the word list is handled by a static class which provides the necessary methods for this task. Future revisions of the system are likely to embody a 'word' class which extends the Java StringBuffer() class and incorporates the partition methods. Partitioning a 70,000 word list where the average word length is 10 characters and minimum stem length 4 characters has the potential for generating more than 350,000 partitions. Whilst modern PCs are well able to handle this work it was found necessary to increase the memory available to the Java Runtime Environment to 256Mbytes to avoid out of memory errors. Having identified a set of hypomorphs with an occurrence greater than the initial discovery threshold the first part of the process concludes by adjusting the occurrence scores to reflect the influence of longer morpheme stacks upon shorter high occurrence hypomorphs. This generates the initial hypomorph set which is the basis for the second stage of the process.

5.2 Validating and extending the hypomorph set

There are four main objects which handle the second stage of processing. These objects are used to construct an iterative process which extends and validates the initial set of hypomorphs. The basic pattern of each iteration is the same: a MorphemeSet generates a StemList from the WordList; the StemList then generates an IParadigmSet from which in turn a new MorphemeSet is generated ready for the next iteration. The relationships between these classes and their key methods are shown in the diagram below:

Figure 5.1: Principal Classes for Generating the Morpheme Set.



The dotted lines indicate secondary relationships between the classes with the primary relationships indicated by the unbroken links. The methods which provide the functionality for the primary links are listed within each class.

The WordList object holds the base data set and supplies methods which identify stems implied by the hypomorph set or alternatively morphemes attested by a supplied list of stems.

WordList objects can:

- generate partitions of their members thus providing the raw dataset from which the initial set of hypomorphs can be derived.
- At the end of the process the WordList can generate a ParsedWordList in which the final MorphemeSet is used to generate a partition of each word in the WordList.
- Given a MorphemeSet the WordList can generate the corresponding StemList.

MorphemeSet objects are found within Stem and IParadigm objects which are in turn members of instances of the StemList object and IParadigmSet object

described below. In addition to managing the objects which are members of the set MorphemeSet can also:

- calculate the Morpheme Set Evidence Value (MSEV) for that instance
- generate an ArrayList object containing the members of the instance ordered in the right priority for use in parsing a word list to generate a Stemlist.

The StemList class extends the Java ArrayList class. Early models were based upon the TreeSet class but it was found that performance was poor sorting the object and retrieving members. The Arraylist object enabled a retrieval mechanism to be implemented which was better suited to the needs of the process. StemList objects are generated from the original word list by means of a supplied set of hypomorphs. Their members are Stem objects each of which contains the stem itself and the MorphemeSet associated with that stem. StemList objects can:

- generate a set of inflection paradigms from the stems they contain.
- resolve stem conflicts amongst members by reference to their associated MorphemeSets.
- generate the MorphemeSet implied by the member Stems of the StemList.

Validation of the hypomorphs implied by the individual stems is by reference to their relationship to morphemes throughout the word list. The IParadigmSet class handles this. Based upon the Java TreeSet class it holds a set of IParadigm objects. Each IParadigm object holds a MorphemeSet and a set of Stems which can be found with each of those morphemes. IParadigmSets can:

- calculate Paradigm Evidence Values (PEVs) for their members,
- optimise their member's MorphemeSets to include common characters shared by a high proportion of the stems associated with a paradigm,
- assign stems to other paradigms which exhibit a subset of the MorphemeSet of another paradigm.
- generate validated MorphemeSets from their members paradigms.

In addition to these methods each object implements standard 'housekeeping' such as methods to add and remove members, comparators to allow the various lists and sets to be sorted and their members compared as required.

5.3 Reporting and final parsing

Once the final set of hypomorphs has been built and the corresponding stem set found a final set of inflection paradigms is generated and the stem set rebuilt from the inflection paradigms. The final morpheme set which represents the principal output from the process can now be written out as can the final list of stems and inflection paradigms. If desired, a full parse of the input word list can be made using the stem list to identify the best partition for each stem. Stem and morpheme pairs directly attested by the final stem and morpheme lists are generated first and then the remaining members of the word list are partitioned using the final stem list to set the partition point. These partitions are considered secondary since they are derived from the primary partitions which are validated directly from the final morpheme set and its associated stems. The fully parsed word list is stored in a ParsedWordList object. Where a second processing run is required for the Secondary Inflection Vector, the input word list for this run is constructed from the stems listed in the parsed word list.

Chapter 6

Conclusions

The method described above has proved to be successful for discovering morpheme structures without the aid of dictionaries, grammars or example datasets. The identification of morphemes without recourse to dictionaries is made possible by one fundamental characteristic of language: in any list of words taken from an inflected language, the character strings that represent morpheme structures will be found to occur with greater frequency than the character strings that represent stem lemmata. This is the fundamental characteristic that underpins all automatic morpheme analysis. Its strength is its universality and its weakness the lack of any semantic features to identify unambiguously these instances. Whilst the presence of significant numbers of loan words, most typically borrowed technical vocabulary or proper names, can unsettle the analysis, overall the process is robust and can handle significant inaccuracies in the input data. The process is particularly effective in identifying forms of words which are cognate with other parts of speech but which are often considered as separate vocabulary items with their own citation forms by traditional grammars.

6.1 Benefits

The work described here has already made a significant contribution within the context of the ParaText Glossing Tool, particularly amongst those working with vernacular languages in the developing world, see 3.1.1 and below. To date it has been demonstrated that useful analysis can be generated for any natural language which falls within the general classification of affixal morphology. The full list of languages where good results have been obtained is not known but it is known to include South American languages, African languages, languages from South East Asia and the Indian sub-continent and even Inuit languages. Much, inevitably, still remains to be done and I look forward to pursuing further outcomes as time permits.

6.2 Limitations

The restriction to affixal languages only is clearly a limitation of this approach. Nevertheless, most natural language (with the notable exception of the Semitic group and those with similar characteristics) favour affixal inflection sufficiently to allow useful results to be obtained by this process [1]. This limitation is, however, common to all automatic morpheme discovery processes at present. Given the lack of any confirmatory semantic features it is inevitable that any automatic process will make errors. Anticipating an entirely correct solution is therefore unrealistic. Nevertheless the process is able to provide valid and useful analysis within the broad context of affixal and agglutinative languages.

Where a language is highly agglutinative the frequency of complex morpheme stacks is likely to be fewer than that of less complex structures. This can sometimes result in valid but sub-optimal parses. For the purposes of this work a sub-optimal parse which falls upon a valid morpheme boundary was counted as valid.

The difficulty of interpreting morpho-phonological changes at stem and affix boundaries has been detailed above and remains a weakness. It can be argued that this problem is largely orthographic as it is often the consequence of crasis as in $cs \Rightarrow x$. There are, however, many other examples of inflection driven mutation. Welsh exhibits a tendency to mutate leading consonants as in *Cymru* (Wales) which becomes *Croeso y Gymru* - Welcome to Wales. Such mutation might well lead this process to conclude that there exists a stem *_ymru* with a pair of prefix morphemes {*C-, G-*}. Without prior knowledge of such rewrites or extensive general phonemic tables it is difficult to see how the process could handle this kind of obfuscation.

Another area of weakness is the degree of confusion created by common stem boundary characters. There are two common circumstances in which this can occur. Morpho-phonological preferences can result in stem final syllables terminating with one of a very small number of consonants. This can become statistically significant such that hypomorphs are proposed which incorporate a common stem final character as an addition to an otherwise valid morpheme stack. The second scenario concerns failures where the stem boundary characters are also, coincidentally, valid morphemes at that position. This might be detected by recognising a duplication of a morpheme within a stack but unless the evidence for this is present in the dataset it is more likely that these characters will be split from the stem and attributed to the morphemes associated with the stem. This is a particular problem with more highly agglutinative languages. In these circumstances it can be quite impossible to mark the stem-morpheme boundary correctly without prior knowledge.

[1] Identifying languages as predominantly infixal or predominantly affixal is fraught with difficulty. In the context of this processing some languages which might be considered infixal may return good results whereas others, considered affixal, return poor results as a consequence of stem mutation. The clearest categorisation to date is probably that made by Bickel and Nichols who suggest that barely 25% of natural languages show evidence of isolating or non-linear formatives (their term for infix). This would suggest that helpful results from this process might be hoped for from at least 75% of natural languages [5, 86].

6.3 Possible applications of this work

Understanding how the words of a language are themselves constructed is crucial for much text processing. Highly inflected and agglutinative languages pose particular problems for dictionary building, hyphenation and spell-checking. In the context of the translation of the Bible into vernacular languages this work has already found an application as a pre-process to the glossing engine which powers the UBS Paratext Glossing Tool which is itself based upon earlier work by the author and his colleague David Robinson of the British and Foreign Bible Society. Whilst the morphological analysis as implemented by the Glossing Tool is primitive by comparison to that detailed in this work it significantly improves the performance of the glosser in more highly inflected and agglutinative languages.

Dictionary building is clearly an area where an ability to lemmatise a word list is essential in the context of inflected languages. Although the presence of sub-optimal parses in the output from this process means that the stem list generated must be reviewed, the large number of accurate stem-morpheme partitions and the relative ease with which sub-optimal parses can be identified from the stem lists can shortcut the time taken to review a stem list very considerably.

Hyphenating vernacular texts is a particular problem for typesetters. Syllable based hyphenation systems do not always perform well and can generate unhelpful partitions [40, 8]. The ability to mark morpheme boundaries as well as syllable boundaries can significantly improve the effectiveness of auto-hyphenation software. This is most clearly demonstrated when a syllable boundary used as a hyphenation point partitions the stem semantic of a word.[2] Syllable-based parsing alone cannot identify this but input from a morphology analyser can often identify such conflicts.

Dictionary based spell-checking is not often available to vernacular languages and can be of limited value when only common forms are listed and a significant proportion of a text may be *hapax legomena*. The morpheme structures identified by this process represent an important step toward the development of form based spelling checks which can identify words by reference to tables of forms without the need to store hundreds of often rarely encountered lexemes.

An application closely related to dictionary and grammar building is that of supplying data for knowledge based systems. Whilst developed world languages are well served with comprehensive databases of word formation rules and tables this is rarely the case of developing world vernaculars. The output from this process should provide significant help to those constructing knowledge bases

[2] Consider the sentence: *the Sparrow chirped for joy and the Raven crow-* and at this point the text reaches right margin and our syllable based hyphenator inserts a hyphen. Experiment has shown that more than 90% of readers are expecting the word to be completed as *crow-ed*. Sadly, *the Sparrow chirped for joy and the Raven crowned the Eagle King*. The problem of interpretation arises as a consequence of the stem being partitioned by the syllable boundary. An analysis of English morphology might conclude that a better hyphenation point might have been *crown-ed*, preserving the stem intact and removing a significant ambiguity for the reader.

for these languages.

6.4 Areas for further investigation

The process as it now stands is very well suited for adaptation to a semi-supervised environment. After each major processing stage the user could be invited to critique the results, deselecting invalid members of the hypomorph set and selecting valid members which have fallen below the automatic aceptance threshold. This would allow more comprehensive results and help to reduce the number of invalid partitions.

At least three other areas can be identified which would enhance this process significantly. A pre-process might be developed to address the problem of morpho-phonological changes. The limitation to affixal languages is clearly significant. Lastly, it is likely that far more can be done with the output from this process in assessing the inflection paradigms produced and collecting them into related sets. Such analysis should then allow predictions to be made of lexemes not found within the input word list but which are, nonetheless potentially valid surface forms.

6.4.1 Mapping morpho-phonological changes

The analysis of the Latin results given above clearly indicated the problems which arise when changes occur within stems. The example given in 4.2.4 of *duco* derived stems highlights this clearly. The coincidence of any pair of phonemes which can be represented in that language by a single character will, inevitably, introduce this kind of confusion. There is in principle at least no reason why such collisions could not involve more than just two characters although examples of this are rarer. It is more usual for this to occur in uniquely vocalic or consonantal contexts. Thus, vowels may lengthen as a consequence of elision and not all languages will represent this in the helpful manner of Dutch by doubling the character as in $a + a \Rightarrow aa$. Greek will, for example, in some circumstances rewrite a lengthened vowel by replacing it with a different character entirely as in $\varepsilon + \alpha \Rightarrow \eta$ or $\varepsilon + o \Rightarrow \omega$. Other examples might be cited but the common consequence of such behaviour is a mutation of either the stem or the morpheme. In the case of the Latin example given at 4.2.4, Latinists usually describe the transformation as a stem change. Whether this description is helpful is open to question. It can be argued that a better assessment of the change might be $duc + s \Rightarrow dux$ with the s recognised as a past or aorist tense marker. The Greek examples given above might transform either a stem or a morpheme structure. In many cases it seems likely that such transformations can be predicted by general phonological rules. Further research could explore how generally such predictions can accurately interpret these transformations in practice. Thereafter an assessment of to what degree it is possible for language specific rules to be derived from a word list might also be a fruitful area of work.

6.4.2 Non-contiguous pattern matching

Whilst the majority of inflected natural languages tend to exhibit inflection by prefix or suffix there remain a significant number that use the infix as a way of constructing surface forms. Examples of this kind of inflection can be found in languages from most of the principal groupings. Within Europe those languages which include the Teutonic group in their history tend to display some measure of infixal inflection. In common with most infixal languages the infix changes tend to be vocalic as in $man \Rightarrow men$ and $sink \Rightarrow sank \Rightarrow sunk$. More complex patterns are found in Semitic languages such as Arabic and Hebrew. The qal imperfective in Hebrew for example uses both infix and affix to construct its forms (Hebrew characters have been transliterated into English equivalents largely following the table given in Gesenius [18, 26] with stem letters encapitaled for clarity and *schwa* represented as subscripts): $yiQ_eToL \Rightarrow tiQ_eToL \Rightarrow tiQ_eT_eLyi \Rightarrow {}'eQ_eToL \Rightarrow yiQ_eT_eLw \Rightarrow tiQ_eToL_enah \Rightarrow tiQ_eT_eLw \Rightarrow niQ_eToL$. Many of these changes can be traced to the rules governing syllable construction and concatenation in Hebrew. The consequence for a morphology analyser is the need to identify the infixes from the stem, typically of three letters in Hebrew, which is embedded within the various affixes and infixes. Experiments using this process have demonstrated that the various affixes can be identified but a reliable means of extracting the stem remains to be developed. Two possible approaches suggest themselves. Using glossing technology it is possible to extract cognate forms from a text. These can then be examined for common features. Alternatively an attempt might be made to develop a pattern matching algorithm which can make use of the difference in frequency between stems and morphemes in a word list as does this process. This represents a major area of research, complementary to this work.

6.4.3 Constructing inflection tables

The main goal of this process is the identification of morpheme structures. As a consequence of the method employed a secondary output is a list of inflection paradigms. Such paradigms may form complete conjugations or declensions, it is more likely they will not. In either case the process as it stands has no way of identifying a complete paradigm from an incomplete paradigm. Given that an analysis of a highly inflected language such as Kiswahili can generate many hundreds, perhaps thousands of inflection paradigms this represents a large body of information from which it may be possible to construct a set of generic paradigms. Stems might then be assigned, at least tentatively, to these paradigms thus offering the prospect that lexemes not present in the initial data set might be hypothesized. Such a process would be particularly helpful in the context of form-based spelling checks.

6.4.4 Morpheme Stack Analysis

In the results presented here only one language has displayed a high degree of both inflection and agglutination. The rate of inflection in English is low and although some agglutination can be found as in for example *-ed-ness* it is infrequent and presents few problems for the linguist. In Latin the inflection rate is much higher yet whilst agglutination does occur it is rarely found to involve more than two components as in *-ab-emus* or *-at-us*. In the case of Kiswahili, however, agglutination is more common, producing many complex stacks of morphemes: *wa-li-po, ni-taka-po, a-taka-vyo-i*. Identifying the individual components within such stacks together with the rules which govern their construction would offer great benefits to dictionary builders and grammar writers. The ability to mark individual morpheme boundaries would likewise benefit hyphenation systems which are traditionally limited to syllable-based parsings. This remains a goal for future work.

J D Riding
October 2007

Bibliography

[1] Cambridge School Classics Project 2002. *Cambridge Latin Course*. Cambridge University Press, 1998.

[2] E L Antworth. *PC-Kimmo: A Two-level Processor for Morphological Analysis*. SIL, 1990.

[3] EO Ashton. *A Swahili Grammar*. Longman, 1944.

[4] K Beesley. Finite-state description of arabic morphology. In *Proceedings of the Second Cambridge Conference: Bilingual Computing in Arabic and English*, 1990.

[5] B Bickel and J Nichols. *The World Atlas of Language Structures*, chapter 20, pages 86–90. Oxford University Press, 2005.

[6] H Andrew Black. Requirements for a syllable parser. Technical report, SIL, March 1999.

[7] Antal van den Bosch and Walter Daelmans. Memory-based morphological analysis. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, 1999.

[8] Noam Chomsky. *The Logical Structure of Linguistic Theory*. Plenum Press, New York & London, 1975.

[9] B Comrie and M Eid, editors. *Computer analysis of Arabic morphology*, Amsterdam, 1991. Benjamins.

[10] The Unicode Consortium. *The Unicode Standard, Version 5.0*. Addison Wesley, 2006.

[11] M Creutz and K Lagus. Unsupervised discovery of morphemes. 2002.

[12] Walter Daelemans, Peter Berck, and Steven Gillis. Unsupervised discovery of phonological categories through supervised learning of morphological rules. In *Proceedings of COLING 1996, Copenhagen*, pages 95–100, 1996.

[13] Beatrice Daille. Morphological rule induction for terminology acquistion. In *Proceedings of the 18th conference on Computational linguistics - Volume 1*, 2000.

[14] M Davis and K Whistler. Unicode collation algorithm. Technical report, The Unicode Consortium, 2006.

[15] S Deerwester, S T Dumais, G W Furnas, T K Landauer, and R Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 1990.

[16] Martin Ebbertz. *Das Internet spricht Englisch ... und neuerdings auch Deutsch*. http://www.netz-tipp.de/sprachen.html, 2002.

[17] Eric Gaussier. Unsupervised learning of derivation morphology from inflectional lexicons. In *Proceedings of the Workshop on Unsupervised Learning in Natural Language Processing*, pages 24–30. Association for Computational Linguistics, 1999.

[18] H F W Gesenius. *Gesenius' Hebrew Grammar*. Oxford Clarendon Press, 2nd edition, 1910.

[19] John Goldsmith. Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, 27(2):153–196, 2001.

[20] Raymond G Gordon. *Ethnologue*. SIL International, 15th edition, 2005.

[21] N Grabar and P Sweigenbaum. Language-independent automatic acquisition of morphological knowledge from synonym pairs. 1999.

[22] Margaret A Hafer and Stephen F Weiss. Word segmentation by letter successor varieties. *Information storage and retrieval*, 10:371–385, 1974.

[23] Zellig Harris. From phoneme to morpheme. *Language*, 31:190–222, 1955.

[24] Zellig Harris. Morpheme boundaries within words: Report on a computer test. In *Transformations and Discourse Analysis Papers*, number 73. Department of Linguistics, University of Pennsylvania., 1967.

[25] Charles F Hockett. Two models of grammatical description. *Word*, 10(2-3):210–234, August-December 1954.

[26] Christian Jacquemin. Guessing morphology from terms and corpora. In *Proceedings of SIGIR 97*, pages 156–165. ACM, Philadelphia, 1997.

[27] Frederick Johnson. *A Standard Swahili-English Dictionary*. Oxford University Press, 1939.

[28] PV Jones and KC Sidwell. *Reading Latin, Grammar, Vocabulary and Exercises*. Cambridge University Press, 1986.

[29] F Karlsson and L Karttunen. Sub-sentencial processing. In R Cole, editor, *Survey of the State of the Art in Human Language Technology*. Giardini Editori e Stampatori, Italy, 1997.

[30] George A Kiraz. Semhe: A generalised two-level system. In *Proceedings of the 34th Meeting of the Association for Computational Linguistics*, pages 159–166. Association for Computational Linguistics, Association for Computational Linguistics, 1996a.

[31] George A Kiraz. Computing prosodic morphology. In *Proceedings of the 16th Conference on Computational linguistics - Volume 2*, pages 664–669. International Conference On Computational Linguistics, Association for Computational Linguistics, 1996b.

[32] George A Kiraz. Multitiered nonlinear morphology using multitape finite automata: a case study on syriac and arabic. *Computational Linguistics Volume 26 , Issue 1 (March 2000)*, pages 77–105, 2000.

[33] K Koskenniemi. *Two-level morphology: A general computational model for word form recognition and production.* Publication no. 11, Department of General Linguistics, University of Helsinki., 1983.

[34] T K Landauer, P W Foltz, and D Laham. Introduction to latent semantic analysis. In *Discourse Processes*, pages 259–284. 1998.

[35] Nathan Miles. Automatic generation of two-level fsm tables. [description of the kgen rule compiler.]. Master's thesis, Ohio State University, 1991.

[36] Christian Monson. A framework for unsupervised natural language morphology induction. In *Proceedings of the Student Workshop at ACL-04*, 2004a.

[37] Christian Monson, Alon Lavie, Jaime Carbonell, and Lori Levin. Unsupervised induction of natural language morphology inflection classes. In *Proceedings of the Workshop of the ACL Special Interest Group on Computational Phonology (SIGPHON)*. Association for Computations Linguistics, 2004b.

[38] Preslav Nakov. *Recognition and Morphological Classification of Unknown Words for German.* PhD thesis, Faculty of Mathematics and Informatics, Sofia University, 2001.

[39] J L Paulo, M Correia, N J Mamede, and C Hagege. Using morphological, syntactical and statistical information for automatic term acquisition. 2002.

[40] J D Riding. First experiments in automatic hyphenation. Technical report, British & Foreign Bible Society, 2005.

[41] Jorma Rissanen. *Stochastic Complexity in Statistical Enquiry.* World Scientific, 1989.

[42] P Schone and D Jurafsky. Knowledge-free induction of morphology using latent semantic analysis. In *Proceedings of CoNLL-2000 and LLL-2000*, pages 67–72, Lisbon, 2000.

[43] Matthew G Snover, Gaja E Jarosz, and Michael R Brent. Unsupervised learning of morphology using a novel directed search algorithm: Taking the first step. In *Proceedings of the 6th Workshop of the ACL Special Interest Group in Computational Phonology, Philadelphia, July 2002*, pages 11–20, 2002.

[44] Richard Sproat. *Morphology and Computation*. The MIT Press, Cambridge, England, 1992.

[45] Pieter Thoeren and Ian Cloete. Automatic acquisition of two-level morphological rules. pages 103–110, 1997.

[46] O Thomas. *Transformational Grammar and the Teaching of English*. Holt Reinhart Winston, 1965.

[47] UBS. Paratext software. http://paratext.ubs-translations.org/, March 2006.

[48] Jan de Waard and Eugene A Nida. *From One Language to Another, Functional Equivalence in Bible Translating*. Nelson, 1986.

[49] D Yarowsky and R Wicentowski. Minimally supervised morphological analysis by multimodal alignment. 2001.

Appendix A

Principal Java Classes

Details of the methods and properties of the main java classes required for this process follow:

1. WordList
2. MorphemeSet
3. StemList
4. IParadigmSet
5. ParsedWordList

mat.bartdevelopment

Class WordList

```
java.lang.Object
  └java.util.AbstractCollection<E>
      └java.util.AbstractList<E>
          └java.util.ArrayList
              └mat.bartdevelopment.WordList
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Cloneable, java.lang.Iterable, java.util.Collection, java.util.List, java.util.RandomAccess

---

```
public class WordList
extends java.util.ArrayList
```

WordList holds a list of words, each one held as a String object. It extends ArrayList with a number of methods to manipulate a word list.

See Also:

Serialized Form

Field Summary

Fields inherited from class java.util.AbstractList
modCount

Constructor Summary

WordList() Creates a new instance of WordList

Method Summary

boolean	**add**(java.lang.Object o) Override for ArrayList.add().
boolean	**add**(java.lang.String s) Override for ArrayList().add().
StemList	**exportStems**(mat.components.MorphemeSet morphemes, int minStemLen) Derives a list of stems from the word list as defined by the supplied morphemeSet.
WordList	**generateReversedWordList**() For each String held in the WordList, generates a corresponding String where the character order is reversed and stores it in a new word list object which is returned to the caller.

int	**getFirstIndex**(java.lang.String prefix) Returns the index of the first element in the WordList which starts with the supplied prefix.
WordList	**getPrefixRange**(java.lang.String prefix) Generates a WordList object containing the character strings found as stem/suffixes to the given prefix within this instance of WordList.
WordList	**getStartsWith**(java.lang.String prefix) Returns an ArrayList containing all the entries in this WordList which begin with the supplied prefix String.
WordList	**getSuffixDomain**(java.lang.String suffix) Returns a WordList object containing all the stems found associated with the supplied suffix in the current instance of WordList.
WordList	**getSuffixWords**(java.lang.String suffix) Returns a WordList object containing all the words found which end with the supplied suffix in the current instance of WordList.
boolean	**isSorted**() Getter for 'sorted'.
ParsedWordList	**parseWordList**(StemList stemlist, int minStemLength) **Deprecated.**
ParsedWordList	**parseWordList**(StemList stemlist, int minStemLength, int vector) Parse Wordlist.
void	**setSorted**(boolean sorted) Setter to allow the sort indicator to be set directly.
void	**sort**() Uses Collections.sort to sort the contents of the WordList.
void	**sort**(java.text.Collator c) Uses Collections.sort to sort the contents of the WordList using the supplied Collator.

Methods inherited from class java.util.ArrayList
add, addAll, addAll, clear, clone, contains, ensureCapacity, get, indexOf, isEmpty, lastIndexOf, remove, remove, removeRange, set, size, toArray, toArray, trimToSize

Methods inherited from class java.util.AbstractList
equals, hashCode, iterator, listIterator, listIterator, subList

Methods inherited from class java.util.AbstractCollection
containsAll, removeAll, retainAll, toString

Methods inherited from class java.lang.Object
finalize, getClass, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.util.List
containsAll, equals, hashCode, iterator, listIterator, listIterator, removeAll, retainAll, subList

Constructor Detail

WordList

```
public WordList()
```

Creates a new instance of WordList

Method Detail

add

```
public boolean add(java.lang.String s)
```

Override for ArrayList().add().

Parameters:
`s` - The String object containing the word to be added.
Returns:
Returns true if the add succeeds, otherwise false.

---

add

```
public boolean add(java.lang.Object o)
```

Override for ArrayList.add(). Only String objects are accepted as members of the WordList(). Attempts to add other classes of objects fail, returning false.

Specified by:
`add` in interface `java.util.Collection`
Specified by:
`add` in interface `java.util.List`
Overrides:
`add` in class `java.util.ArrayList`
Parameters:
`o`
- The object to be added to the WordList. The object should be a String object. This override ensures that any other type of object is rejected.
Returns:
Returns true if the add succeeds, otherwise false. An attempt to add anything other than a String object to the WordList will return false.

---

sort

```
public void sort()
```

Uses Collections.sort to sort the contents of the WordList. N.B. Collections.sort will obey the preferred UCA for the currently selected locale.

---

sort

```
public void sort(java.text.Collator c)
```

Uses Collections.sort to sort the contents of the WordList using the supplied Collator.

Parameters:
`c` - The Collator to be used to order the sort.

getStartsWith

```
public WordList getStartsWith(java.lang.String prefix)
```

Returns an ArrayList containing all the entries in this WordList which begin with the supplied prefix String. If the prefix is not found, returns null;

Parameters:

`prefix`
- A String object containing the string of characters to be sought as a prefix within the list.

Returns:

An ArrayList object containing all the elements of the word list which begin with the supplied prefix string.

---

getFirstIndex

```
public int getFirstIndex(java.lang.String prefix)
```

Returns the index of the first element in the WordList which starts with the supplied prefix. If the prefix is not found -1 is returned. N.B. This method requires the list to be sorted. If the list is unsorted this call returns -1.

Parameters:

`prefix`
- A String object containing the string of characters the which is to be sought as a prefix within the list.

Returns:

The index of the first element encountered in the list which begins with the supplied prefix string.

---

generateReversedWordList

```
public WordList generateReversedWordList()
```

For each String held in the WordList, generates a corresponding String where the character order is reversed and stores it in a new word list object which is returned to the caller. N.B. the reversed list is returned in unsorted form. In this 'raw' state each entry will share the index of its corresponding entry in the word list from which it was derived. To generate a sorted list use this method and then call instance.sort() on the newly created list.

Returns:

An unsorted WordList object containing the reversed form of each String held in this word list.

---

getPrefixRange

```
public WordList getPrefixRange(java.lang.String prefix)
```

Generates a WordList object containing the character strings found as stem/suffixes to the given prefix within this instance of WordList. The returned WordList is sorted according to the current Locale.

Parameters:

`prefix` - The prefix for which the 'range' items should be generated from this instance.

Returns:

A WordList object containing the range items found with the supplied prefix in this instance. N.B. The list of range items returned may include a null string "" if the supplied prefix is present in the WordList as a word in its own right.

---

getSuffixDomain

```
public WordList getSuffixDomain(java.lang.String suffix)
```

Returns a WordList object containing all the stems found associated with the supplied suffix in the current instance of WordList.

Parameters:

`suffix` - The suffix whose stems are to be sought.

Returns:

WordList containing each of the stems found with the supplied suffix.

---

getSuffixWords

```
public WordList getSuffixWords(java.lang.String suffix)
```

Returns a WordList object containing all the words found which end with the supplied suffix in the current instance of WordList.

Parameters:

`suffix` - The suffix whose words are to be sought.

Returns:

WordList containing each of the stems found with the supplied suffix.

---

isSorted

```
public boolean isSorted()
```

Getter for 'sorted'. Indicates whether the list has been sorted or not.

Returns:

boolean true = sorted WordList, false = unsorted WordList

---

setSorted

```
public void setSorted(boolean sorted)
```

Setter to allow the sort indicator to be set directly. Typically used only when the list has been populated with elements which are supplied ready sorted.

Parameters:

`sorted` - Value for 'sorted'.

---

exportStems

```
public StemList exportStems(mat.components.MorphemeSet morphemes,
                            int minStemLen)
```

Derives a list of stems from the word list as defined by the supplied morphemeSet. Assumes that the morphemes supplied are suffix Morphemes.

Parameters:

`minStemLen` - The minimum length of stem for this run.
`morphemes` - A set of valid suffix morphemes.

Returns:

StemList object of derived Stem objects and their associated morpheme sets.

---

parseWordList

```
public ParsedWordList parseWordList(StemList stemlist,
                                    int minStemLength)
```

Deprecated.

Takes a StemList which has been derived from this word list. Parses all words found in the StemList according to the StemList parsing. Then generates a MorphemeSet from the StemList. Orders the MorphemeSet and parses all unparsed words remaining on the word list using the morphemes in parse order one by one.

Parameters:

`stemlist` - The StemList to be used as the basis for the parse.
`minStemLength` - The minimum length of stem permitted.

Returns:

A set of ParsedWord objects representing the words in the wordlist as parsed by this process.

---

parseWordList

```
public ParsedWordList parseWordList(StemList stemlist,
                                    int minStemLength,
                                    int vector)
```

Parse Wordlist. Takes a StemList which has been derived from this word list. Parses all words found in the StemList according to the StemList parsing. Then generates a MorphemeSet from the StemList. Orders the MorphemeSet and parses all unparsed words remaining on the word list using the morphemes in parse order one by one.

Parameters:

`stemlist` - The StemList to be used as the basis for the parse.
`minStemLength` - The minimum length of stem permitted.
`vector`
- Indicates whether the StemList is the result of a prefixal or suffixal processing run. Permitted values: Processparams.PREFIXAL || Processparams.SUFFIXAL .

Returns:

A set of ParsedWord objects representing the words in the wordlist as parsed by this process.

---

mat.components

Class MorphemeSet

```
java.lang.Object
  └java.util.AbstractCollection<E>
      └java.util.AbstractSet<E>
          └java.util.TreeSet
              └mat.components.LogicalSet
                  └mat.components.MorphemeSet
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Cloneable, java.lang.Comparable, java.lang.Iterable, java.util.Collection, java.util.NavigableSet, java.util.Set, java.util.SortedSet

Direct Known Subclasses:

IParadigm

```
public class MorphemeSet
extends LogicalSet
implements java.lang.Comparable, java.io.Serializable
```

MorphemeSet objects store a set morphemes. Each morpheme is stored as a CStringBuffer object with item and count fields. MorphemeSet extends LogicalSet which in turn extends TreeSet. In addition to holding the set of morphemes MS can calculate a MorphemeSet Evidence Value (MSEV) for the current state of the set. This is calculated as the sum of the log10 of the count field for each morpheme, raised to the power of the length of the morpheme in characters: Sum((log10(me)^^|m|).

Author:

J D Riding

See Also:

Serialized Form

Constructor Summary

MorphemeSet()	Creates a new instance of MorphemeSet

Method Summary

boolean	**add**(java.lang.Object o) Override for .add method of TreeSet which is inherited by LogicalSet.
int	**compareTo**(java.lang.Object o) Implements compareTo() for Comparable by testing the String representation of the two instances of MorphemeSet being compared.
void	**dumpMS**(java.lang.String filename) Dumps the morpheme set as a list of strings in the format: morpheme [tab][tab] count.

void	**dumpMSRev**(java.lang.String filename, int vector) Dumps the morpheme set as a list of strings in the format: morpheme [tab][tab] count.
double	**getMSEV**() Calculates the MorphemeSet Evidence Value (MSEV) for the current state of the set.
int	**getValue**() This method sums the count values for each object in the set and returns the sum as the set value.
java.lang.String	**morphemesToString**() Returns a string representation of the morphemes in the set in the format {mmm, mmm, mmm...}
java.util.ArrayList	**orderForParsing**() Returns an ArrayList of CStringBuffer() objects, ordered on length of morpheme and morpheme value.
MorphemeSet	**reverseItems**() Reverses the order of characters in the item field of each member of this instance.
java.lang.String	**toString**() Returns a String representation of the MorphemeSet in the format: {mmm #000, mmm #000...}.
java.lang.String	**toString**(int vector) Retunrs a String representation of the MorphemeSet in the format: {mmm #000, mmm #000...}.
void	**toXMLFile**(java.lang.String fileName, java.lang.String headers, java.lang.String styleSheet) Writes out the MorphemeSet as an XML file.

Methods inherited from class mat.components.LogicalSet
complement, intersect, isMember, union

Methods inherited from class java.util.TreeSet
addAll, ceiling, clear, clone, comparator, contains, descendingIterator, descendingSet, first, floor, headSet, headSet, higher, isEmpty, iterator, last, lower, pollFirst, pollLast, remove, size, subSet, subSet, tailSet, tailSet

Methods inherited from class java.util.AbstractSet
equals, hashCode, removeAll

Methods inherited from class java.util.AbstractCollection
containsAll, retainAll, toArray, toArray

Methods inherited from class java.lang.Object
finalize, getClass, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.util.Set
containsAll, equals, hashCode, removeAll, retainAll, toArray, toArray

Constructor Detail

MorphemeSet

```
public MorphemeSet()
```

Creates a new instance of MorphemeSet

Method Detail

add

```
public boolean add(java.lang.Object o)
```

Override for .add method of TreeSet which is inherited by LogicalSet. A MorphemeSet can only hold CStringBuffer class elements. This method rejects objects of other classes.

Specified by:
`add` in interface `java.util.Collection`
Specified by:
`add` in interface `java.util.Set`
Overrides:
`add` in class `java.util.TreeSet`
Parameters:
`o` - - the instance to be added to the set.
Returns:
- true if the instance is succesfully added, otherwise false, meaning either that the instance is already a member of the set or that the instance is not of class CStringBuffer.

---

getValue

```
public int getValue()
```

This method sums the count values for each object in the set and returns the sum as the set value.

Returns:
The sum of the counts for each item in the set.

---

toString

```
public java.lang.String toString()
```

Returns a String representation of the MorphemeSet in the format: {mmm #000, mmm #000...}.

Overrides:
`toString` in class `java.util.AbstractCollection`
Returns:
String containing the text representation of the MorphemeSet.

---

toString

```
public java.lang.String toString(int vector)
```

Retunrs a String representation of the MorphemeSet in the format: {mmm #000, mmm #000...}. If vector is set to prefixal, reverses the morpheme strings first so that they display correctly.

Parameters:

`vector` - Indicates if this morphemeset is prefixal or suffixal. 0 = Prefixal, 1 = Suffixal.

Returns:

String containing the text representation of the MorphemeSet.

---

morphemesToString

```
public java.lang.String morphemesToString()
```

Returns a string representation of the morphemes in the set in the format {mmm, mmm, mmm...}

Returns:

String representing the Morphemes in the Morpheme set.

---

getMSEV

```
public double getMSEV()
```

Calculates the MorphemeSet Evidence Value (MSEV) for the current state of the set. This is calculated as the sum of the log10 of the count field for each morpheme, raised to the power of the length of the morpheme in characters: Sum((log10(me)^^|m|).

Returns:

Float containing the MSEV as calculated for the current state of the set.

---

compareTo

```
public int compareTo(java.lang.Object o)
```

Implements compareTo() for Comparable by testing the String representation of the two instances of MorphemeSet being compared.

Specified by:

`compareTo` in interface `java.lang.Comparable`

Parameters:

`o` - A Lemma object to be compared with this instance.

Returns:

returns <0, 0, >0 as for super.compareTo();

---

orderForParsing

```
public java.util.ArrayList orderForParsing()
```

Returns an ArrayList of CStringBuffer() objects, ordered on length of morpheme and morpheme value. This list is in the correct order for use as a parsing list for Stem recognition in the wordlist.

Returns:

ArrayList of ordered CStringBuffer() objects.

---

dumpMS

```
public void dumpMS(java.lang.String filename)
```

Dumps the morpheme set as a list of strings in the format: morpheme [tab][tab] count.

Parameters:

`filename`
- The name of the file to which the data is to be written. If filename = "" the data is written to System.out.

---

dumpMSRev

```
public void dumpMSRev(java.lang.String filename,
                      int vector)
```

Dumps the morpheme set as a list of strings in the format: morpheme [tab][tab] count.

Parameters:

`filename` - The name of the file to which the data is to be written.

`vector`
- The direction of inflection currently being processed. If filename = "" the data is written to System.out.

---

toXMLFile

```
public void toXMLFile(java.lang.String fileName,
                      java.lang.String headers,
                      java.lang.String styleSheet)
```

Writes out the MorphemeSet as an XML file.

Parameters:

`fileName` - "" = construct from set e.g. _i_is_o_orum_os_um_us.xml else as supplied.
`headers` - Any xml comment data to be written to the head of the file.
`styleSheet` - Name of xsl file to be used to transform the data.

---

reverseItems

```
public MorphemeSet reverseItems()
```

Reverses the order of characters in the item field of each member of this instance. Used to reset the order of character data following a prefixal processing run.

Returns:

MorphemeSet().

---

mat.bartdevelopment

Class StemList

```
java.lang.Object
  └java.util.AbstractCollection<E>
      └java.util.AbstractList<E>
          └java.util.ArrayList
              └mat.bartdevelopment.StemList
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Cloneable, java.lang.Iterable, java.util.Collection, java.util.List, java.util.RandomAccess

```
public class StemList
extends java.util.ArrayList
```

An extension of ArrayList() to store and manipulate Stem objects.

See Also:

Serialized Form

Field Summary

Fields inherited from class java.util.AbstractList
modCount

Constructor Summary

StemList() Creates a new instance of StemList

Method Summary

void	**dumpList**(java.lang.String filename, int minStemLen) Output this list as .toString to a flat text file.
void	**dumpList**(java.lang.String filename, int minStemLen, int vect Output this list as .toString to a flat text file.
java.util.TreeSet	**generateIParadigms**(int theta) Generate inflection paradigms from the Stems in this list.
IParadigmSet	**generateIParadigmSet**(int theta) Generate inflection paradigms from the Stems in this list.
mat.components.MorphemeSet	**generateMorphemeSet**(int minStemLength) Creates a MorphemeSet containing all the morphemes found in thi stemlist.
java.util.Comparator	**getComparator**() Getter for the comparator field.

java.util.TreeSet	**getStemSet**() Returns a TreeSet containing the data held in this List sorted in Stem.sortByLemma order.
void	**recalculateMorphemeCounts**(mat.components.MorphemeSet morphem Recalculates the morpheme occurence values for morphemes foun this list according to their representation within the list.
void	**removeStemsLessThanTheta**(int theta) Removes Stems from the list which have fewer than theta morphe associated with them.
StemList	**resolveStemConflicts**() The stems and morpheme sets generated by extending the initial morpheme evidence by reference to the stems as they appear in the word will contain many stems which are in conflict with one another.
StemList	**reverseDataSet**() Reverses the order of characters in the data for this instance.
void	**setComparator**(java.util.Comparator comparator) Set the comparator for this list to that supplied.
void	**sort**() Sorts the list using the currently assigned comparator.
void	**sort**(java.util.Comparator comparator) Sorts the list using the supplied comparator.
void	**stemNamesToXML**(java.lang.String fileName, java.lang.String headers, java.lang.String stemStylesheet) Writes out an XML file containing an XML element within which placed a element for each Stem in the list set to the value returned by this.getStem().
void	**stemsToCSV**(java.lang.String fileName, int minStemLength) Export the stem objects from the stem list to a csv based table.

Methods inherited from class java.util.ArrayList
add, add, addAll, addAll, clear, clone, contains, ensureCapacity, get, indexOf, isEmpty, lastIndexOf, remove, remove, removeRange, set, size, toArray, toArray, trimToSize

Methods inherited from class java.util.AbstractList
equals, hashCode, iterator, listIterator, listIterator, subList

Methods inherited from class java.util.AbstractCollection
containsAll, removeAll, retainAll, toString

Methods inherited from class java.lang.Object
finalize, getClass, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.util.List
containsAll, equals, hashCode, iterator, listIterator, listIterator, removeAll, retainAll, subList

Constructor Detail

StemList

```
public StemList()
```

Creates a new instance of StemList

Method Detail

getComparator

```
public java.util.Comparator getComparator()
```

Getter for the comparator field.

Returns:
The comparator currently being used to sort this list.

---

setComparator

```
public void setComparator(java.util.Comparator comparator)
                  throws InvalidCollatorException
```

Set the comparator for this list to that supplied. If the comparator supplied differs from that previously assigned.

Parameters:
`comparator` - The Collator to be used for this list.
Throws:
`InvalidCollatorException` - If an attempt is made to set the Collator to anything other than Stem.sortByLemma or Stem.sortByParadigmValue.

---

sort

```
public void sort()
          throws InvalidCollatorException
```

Sorts the list using the currently assigned comparator.

Throws:
`InvalidCollatorException` - If an attempt is made to sort with the Collator set to anything other than Stem.sortByLemma or Stem.sortByParadigmValue.

---

sort

```
public void sort(java.util.Comparator comparator)
          throws InvalidCollatorException
```

Sorts the list using the supplied comparator.

Parameters:
`comparator`
- The comparator to be used for this sort. N.B. only Stem.sortByLemma and Stem.sortByParadigmValue are valid collators for this class.
Throws:
`InvalidCollatorException`
- If an attempt is made to assign a comparator other than Stem.sortByLemma or

Stem.sortByParadigmValue.

---

getStemSet

```
public java.util.TreeSet getStemSet()
```

Returns a TreeSet containing the data held in this List sorted in Stem.sortByLemma order.

Returns:
A TreeSet containing the list elements from this instance.

---

removeStemsLessThanTheta

```
public void removeStemsLessThanTheta(int theta)
                              throws InvalidCollatorException
```

Removes Stems from the list which have fewer than theta morphemes associated with them.

Parameters:
`theta`
- The number of morphemes below which the stem should be removed from the list.

Throws:
`InvalidCollatorException`
- if the current instance is sorted by anything other than Stem.sortByParadigmValue.

---

generateIParadigmSet

```
public IParadigmSet generateIParadigmSet(int theta)
```

Generate inflection paradigms from the Stems in this list. An inflection paradigm is defined as a morpheme set which can be found for each of the stems associated with it.

Parameters:
`theta` - The minimum number of stems required to validate this paradigm.

Returns:
an IParadigmSet object.

---

generateIParadigms

```
public java.util.TreeSet generateIParadigms(int theta)
```

Generate inflection paradigms from the Stems in this list. An inflection paradigm is defined as a morpheme set which can be found for each of the stems associated with it.

Parameters:
`theta` - The minimum number of stems required to validate this paradigm.

Returns:
a TreeSet object of IPs.

---

dumpList

```
public void dumpList(java.lang.String filename,
                     int minStemLen)
```

Output this list as .toString to a flat text file. Filter on minimum stem length.

Parameters:

`filename`
- Name of the file to which the data should be written. If filename = "" the output strings are written to the console.
`minStemLen` - Minimum permitted Stem length.

---

dumpList

```
public void dumpList(java.lang.String filename,
                     int minStemLen,
                     int vector)
```

Output this list as .toString to a flat text file. Filter on minimum stem length.

Parameters:

`filename`
- Name of the file to which the data should be written. If filename = "" the output strings are written to the console.
`minStemLen` - Minimum permitted Stem length.

---

generateMorphemeSet

```
public mat.components.MorphemeSet generateMorphemeSet(int minStemLength)
```

Creates a MorphemeSet containing all the morphemes found in this stemlist. Recalculates morpheme occurence stats based on the evidence of this stem list. New stats are reported in the returned MS but not written back to this list.

Parameters:

`minStemLength` - Only Stems of this length or longer will be included in the analysis.

Returns:

A MorphemeSet containing the morphemes found in the list.

---

recalculateMorphemeCounts

```
public void recalculateMorphemeCounts(mat.components.MorphemeSet morphemes)
```

Recalculates the morpheme occurence values for morphemes found in this list according to their representation within the list.

Parameters:

`morphemes`
- An existing copy of the MS created by this.generateMorphemeSet() or null. If null, calls this.generateMorphemeSet to create MS.

---

stemNamesToXML

```
public void stemNamesToXML(java.lang.String fileName,
                           java.lang.String headers,
                           java.lang.String stemStylesheet)
```

Writes out an XML file containing an XML element within which is placed a element for each Stem in the list set to the value returned by this.getStem(). For each Stem in the list a separate XML file is created which is stord in the same directory as the stemNames file. This method is

typically used to write a stem index file which is then transformed by an XSLT such that

Parameters:

`fileName`
- The name of the file to which the list id to be dumped. N.B. To control the location of this file and the Stem files which are written out, ensure this name is specified as an absolute path by the caller.
`headers`
- Any XML headers (other than xml version which is always written) which are to be written to the file.
`stemStylesheet` - The name of the xslt file used to transform the Stem object xml files.

---

resolveStemConflicts

```
public StemList resolveStemConflicts()
```

The stems and morpheme sets generated by extending the initial morpheme evidence by reference to the stems as they appear in the word list will contain many stems which are in conflict with one another. These conflicts are resolved as follows: For any pair of stems s1 & s2 where (s1_initial-subset_s2) remove the stem with the lowest morpheme set value.

---

stemsToCSV

```
public void stemsToCSV(java.lang.String fileName,
                       int minStemLength)
```

Export the stem objects from the stem list to a csv based table. Tuple format: PEVStemMorphemeSet MorphemeSet format: mmm,mmm,mmm..,mmm or StemPEVMorpheme ...

Parameters:

`minStemLength`
- The minimum length of stem required for this report. Stems with fewer characters will be ignored.
`fileName` - The name of the file to which the csv data should be written.

---

reverseDataSet

```
public StemList reverseDataSet()
```

Reverses the order of characters in the data for this instance. Used to reset the order of character data following a prefixal processing run.

Returns:

StemList().

---

mat.bartdevelopment

Class IParadigmSet

```
java.lang.Object
  └java.util.AbstractCollection<E>
      └java.util.AbstractSet<E>
          └java.util.TreeSet
              └mat.bartdevelopment.IParadigmSet
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Cloneable, java.lang.Iterable, java.util.Collection, java.util.NavigableSet, java.util.Set, java.util.SortedSet

---

```
public class IParadigmSet
extends java.util.TreeSet
```

IParadigmSet provides methods for manipulating and assessing a set of IParadigm objects. These include the ability to combine common stem sets, to optimise all morpheme structures in member IPs to reflect the maximum viable length of morphemes, to combine stem sets where one is a superset of another and to export a set of validated morphemes.

See Also:

Serialized Form

Constructor Summary

IParadigmSet() Creates a new instance of IParadigmSet

Method Summary

boolean	**add**(java.lang.Object o) Adds an IParadigm object to the set.
boolean	**addAll**(java.util.Collection c) Adds a collection of IParadigm objects to the set.
mat.components.IParadigm	**crossPopulateStems**(mat.components.IParadigm ip) When an IParadigm is modified or a new IP is added to the IP Set a check must be made for existing IPs of which the new or modified IP is a proper subset.
void	**dumpIPSet**(java.lang.String fname) Dumps a string representation of the IP set to a .csv file with space as delimiter.
void	**dumpIPSet**(java.lang.String fname, int vector) Dumps a string representation of the IP set to a .csv file with space as delimiter.

mat.components.MorphemeSet	**exportMorphemes**(java.util.ArrayList morphemes, int pevTheta, int msetSizeTheta, int valuedMorphsTheta) **Deprecated.** *DEPRECATED from 9/8/06 replaced by overload above*
mat.components.MorphemeSet	**exportMorphemes**(mat.components.MorphemeSet bankers, int pevTheta, int msetSizeTheta) Returns a list a validated morphemes derived from this instance of the ParadigmSet.
mat.components.IParadigm	**get**(mat.components.IParadigm ip) Returns the object found at the location in the set occupied by the supplied key.
boolean	**mergeStemSets**(mat.components.IParadigm ip) Merges the stemset of the current instance of this object in the set with that supplied.
void	**optimiseMorphemeStructures**(java.util.ArrayList morphemes, int minStemLength, int minNumOfStems, double factor) Rebuilds each IP in the set with the optimal Morpheme Set for that IP.
IParadigmSet	**removeIPs**(int numStems) Removes IPs from this set which have stemsets with equal to or less than the number of stems given in numStems.
boolean	**replaceStemSet**(mat.components.IParadigm ip) Replaces the stemset of the current instance of this object in the set with that supplied.
mat.components.MorphemeSet	**reportMorphemes**(int theta) Generates a list of morphemes derived from this IPS for all morphemes found in IPS with PEVs at or above theta.
StemList	**reportStems**(int theta) Generates a list of stems attested by this IPS together with their morpheme sets.

Methods inherited from class java.util.TreeSet
ceiling, clear, clone, comparator, contains, descendingIterator, descendingSet, first, floor, headSet, headSet, higher, isEmpty, iterator, last, lower, pollFirst, pollLast, remove, size, subSet, subSet, tailSet, tailSet

Methods inherited from class java.util.AbstractSet
equals, hashCode, removeAll

Methods inherited from class java.util.AbstractCollection
containsAll, retainAll, toArray, toArray, toString

Methods inherited from class java.lang.Object
finalize, getClass, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.util.Set
containsAll, equals, hashCode, removeAll, retainAll, toArray, toArray

Constructor Detail

IParadigmSet

```
public IParadigmSet()
```

Creates a new instance of IParadigmSet

Method Detail

addAll

```
public boolean addAll(java.util.Collection c)
               throws java.lang.ClassCastException
```

Adds a collection of IParadigm objects to the set.

Specified by:
`addAll` in interface `java.util.Collection`
Specified by:
`addAll` in interface `java.util.Set`
Overrides:
`addAll` in class `java.util.TreeSet`
Parameters:
`c` - The collection of IParadigm objects to be added.
Returns:
true = the set has been modified false = the set has not been modified
Throws:
`java.lang.ClassCastException` - if an attempt is made to add objects of class other than IParadigm().

---

add

```
public boolean add(java.lang.Object o)
            throws java.lang.ClassCastException
```

Adds an IParadigm object to the set.

Specified by:
`add` in interface `java.util.Collection`
Specified by:
`add` in interface `java.util.Set`
Overrides:
`add` in class `java.util.TreeSet`
Parameters:
`o` - The IParadigm object to be added.
Returns:
true = the set has been modified false = the set has not been modified
Throws:
`java.lang.ClassCastException` - if an attempt is made to add an object of class other than IParadigm().

---

replaceStemSet

```
public boolean replaceStemSet(mat.components.IParadigm ip)
```

Replaces the stemset of the current instance of this object in the set with that supplied.

Parameters:

`ip` - An IP object containing the new stemset.

Returns:

true = the stem set has been modified false = the stem set has not been modified

---

mergeStemSets

```
public boolean mergeStemSets(mat.components.IParadigm ip)
```

Merges the stemset of the current instance of this object in the set with that supplied.

Parameters:

`ip` - An IP object containing the additional stemset.

Returns:

true = the stem set has been modified, false = the stem set has not been modified

---

get

```
public mat.components.IParadigm get(mat.components.IParadigm ip)
```

Returns the object found at the location in the set occupied by the supplied key.

Parameters:

`ip` - the IP to be returned, if present in the set.

Returns:

an IParadigm object if the required ip is present in the set, otherwise null;

---

optimiseMorphemeStructures

```
public void optimiseMorphemeStructures(java.util.ArrayList morphemes,
                                       int minStemLength,
                                       int minNumOfStems,
                                       double factor)
```

Rebuilds each IP in the set with the optimal Morpheme Set for that IP.

Parameters:

`morphemes`
- The set of all possible morphemes. Used as a reference data to assess the validity of extended morpheme structures.

---

crossPopulateStems

```
public mat.components.IParadigm crossPopulateStems(mat.components.IParadigm ip)
```

When an IParadigm is modified or a new IP is added to the IP Set a check must be made for existing IPs of which the new or modified IP is a proper subset. Where this is the case the stem sets of these supersets of the new IP should be added to the stemset of the new IP.

Parameters:

`ip` - The IP to be merged into the IP set.

Returns:

The merged IP.

---

exportMorphemes

```
public mat.components.MorphemeSet exportMorphemes(mat.components.MorphemeSet bankers,
                                                  int pevTheta,
                                                  int msetSizeTheta)
```

Returns a list a validated morphemes derived from this instance of the ParadigmSet. Morphemes are extracted from IPs where: 1. |IP(ms)| >= 3 and 2. IP.pev > pevTheta 3. |IP(ms) intersect bankers| >= 1

Parameters:

`bankers` - The set of previously established valid morphemes.
`pevTheta` - The PEV threshold below which the IP is discounted.
`msetSizeTheta` - The morpheme set size below which the IP is disregarded.

Returns:

A morpheme set of all morphemes validated by this IP set.

---

exportMorphemes

```
public mat.components.MorphemeSet exportMorphemes(java.util.ArrayList morphemes,
                                                  int pevTheta,
                                                  int msetSizeTheta,
                                                  int valuedMorphsTheta)
```

Deprecated. *DEPRECATED from 9/8/06 replaced by overload above*

Returns a list of valid morphemes from the IPS. Validity is determined by size of MS (sets with fewer than three morphemes are discarded) and by PEV - sets with a PEV lower than the supplied threshold are discarded.

Parameters:

`morphemes` - An ArrayList of Morpheme() objects as derived from the original wordlist.
`pevTheta`
- The PEV threshold below which paradigms are disregarded as sources of valid morphemes.
`msetSizeTheta`
- MorphemeSets with fewer members than this threshold are disregarded as sources of valid morphemes.
`valuedMorphsTheta`
- Sets the minimum number of morphemes required to be listed in 'morphemes'. Only sets with more listed morphemes than this theshold can be used as sources of valid morphemes.

Returns:

A MorphemeSet of CStringBuffer objects representing attested morphemes found in this IPS.

---

reportMorphemes

```
public mat.components.MorphemeSet reportMorphemes(int theta)
```

Generates a list of morphemes derived from this IPS for all morphemes found in IPS with PEVs at or above theta.

Parameters:

`theta` - The threshold above which IPs are included in this report.

Returns:

The set of morphemes found in this IP set.

---

reportStems

```
public StemList reportStems(int theta)
```

Generates a list of stems attested by this IPS together with their morpheme sets.

Parameters:
`theta` - The threshold above which IPs are included in this report.
Returns:
The set of stems which can be derived from this IP set.

---

dumpIPSet

```
public void dumpIPSet(java.lang.String fname)
```

Dumps a string representation of the IP set to a .csv file with space as delimiter.

Parameters:
`fname` - The name of the file to be written.

---

dumpIPSet

```
public void dumpIPSet(java.lang.String fname,
                      int vector)
```

Dumps a string representation of the IP set to a .csv file with space as delimiter.

Parameters:
`fname` - The name of the file to be written.
`vector` - The processing vector for this pass.

---

removeIPs

```
public IParadigmSet removeIPs(int numStems)
```

Removes IPs from this set which have stemsets with equal to or less than the number of stems given in numStems. e.g. numStems = 1 removes IPs with singleton stemsets.

Parameters:
`numStems` - The threshold of number of stems at which IPs are to be discarded.
Returns:
The truncated IP Set.

---

mat.bartdevelopment

Class ParsedWordList

```
java.lang.Object
  └java.util.AbstractCollection<E>
      └java.util.AbstractSet<E>
          └java.util.TreeSet
              └mat.bartdevelopment.ParsedWordList
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Cloneable, java.lang.Iterable, java.util.Collection, java.util.NavigableSet, java.util.Set, java.util.SortedSet

---

```
public class ParsedWordList
extends java.util.TreeSet
```

ParsedWordList provides a reporting structure for the outputs from the morpheme discovery process. Each word from the wordlist for which a parse is attempted is listed with prefixes, suffixes, stem, citation form and lexeme.

See Also:

Serialized Form

Constructor Summary

ParsedWordList() Creates a new instance of ParsedWordList

Method Summary

void	**dumpPWList**(java.lang.String fileName) Dumps a PWList to csv file.

Methods inherited from class java.util.TreeSet
add, addAll, ceiling, clear, clone, comparator, contains, descendingIterator, descendingSet, first, floor, headSet, headSet, higher, isEmpty, iterator, last, lower, pollFirst, pollLast, remove, size, subSet, subSet, tailSet, tailSet

Methods inherited from class java.util.AbstractSet
equals, hashCode, removeAll

Methods inherited from class java.util.AbstractCollection
containsAll, retainAll, toArray, toArray, toString

Methods inherited from class java.lang.Object
finalize, getClass, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.util.Set
containsAll, equals, hashCode, removeAll, retainAll, toArray, toArray

Constructor Detail

ParsedWordList

```
public ParsedWordList()
```

Creates a new instance of ParsedWordList

Method Detail

dumpPWList

```
public void dumpPWList(java.lang.String fileName)
```

Dumps a PWList to csv file.

Parameters:

`fileName` - The name of the utf-8 file to the data should be written.