Irish Grammar Checking
A Prototype Implementation

Brian Ó Raghallaigh
B.A. (Mod.) CSLL
Final Year Project, May 2003
Supervisor: Dr. Carl Vogel

May 13, 2003
Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

_________________________ May 13, 2003
Brian Ó Raghallaigh
Acknowledgements

There are a number of people without whom the completion of this project would have been a lot more difficult. First and foremost I would like to thank Dr. Carl Vogel for sharing his expertise with me and more importantly for his constant encouragement and belief in me. I would also like to thank my brother Eoghan Ó Raghallaigh for his assistance regarding the finer points of Irish Grammar. Thanks also to the rest of my family for their support and to all my classmates for their encouragement and assistance along the way.
'Who invented this language and what place was it invented in and what time was it invented? Not difficult. Fénius Farsaid invented it at Nimrod’s tower at the end of ten years after the dispersal from the tower... It is then that the language was cut out from the many languages and was assigned to one man of them so that it is his name that is on this language, so that Goídele (Gaelic/Irish) is thence from Goídel son of Aingen son of Glúinfind son of Láimfind son of Agnuman of the Greeks... It is there that this language was then regulated: what was best then of every language and what was broadest and what was finest, that was cut out into Irish' (Ahlqvist, 1983)
Contents

1 Introduction ............................ 1
  1.1 Project Goals .................................. 2
  1.2 Project Overview ................................ 2
  1.2.1 Chapter 2: Modern Irish ...................... 2
  1.2.2 Chapter 3: Proofing Tools for Irish ............ 2
  1.2.3 Chapter 4: Grammar Checking .................. 3
  1.2.4 Chapter 5: A Prototype Implementation .......... 3
  1.2.5 Chapter 6: A Description of the Code .......... 4
  1.2.6 Chapter 7: Interfacing with Ispell ............ 4
  1.3 Concluding Remarks and Future Work ............ 4

2 Modern Irish .............................. 5
  2.1 Introduction .................................. 6
  2.2 The Irish Language: A Brief History ............. 6
  2.3 Irish Orthography .............................. 9
  2.4 Irish Syntax .................................. 10
  2.5 Irish Morphology ............................... 10
     2.5.1 Mutation .................................. 11
     2.5.2 Mutation, Inflection, and the Present Implementation 13
  2.6 Conclusion .................................. 13

3 Irish & Proofing Tools ...................... 15
  3.1 Introduction .................................. 16
  3.2 Spell Checking Systems ......................... 16
  3.3 Irish Spell Checkers ........................... 17
4 Grammar Checking
   4.1 Introduction ........................................ 21
   4.2 Finite-State Grammar Checking .................. 21
   4.3 Limitations of Finite-State Technique ........ 22
   4.4 Scope of Implementation ........................ 23
      4.4.1 Introduction .................................. 23
      4.4.2 The Verb Phrase ............................... 23
      4.4.3 The Noun Phrase ................................ 28
   4.5 Conclusion .......................................... 31

5 The Implementation ...................................... 32
   5.1 Introduction ........................................ 33
   5.2 Grammatical Information ............................ 33
      5.2.1 Word Classes and Part-of-Speech Tagging .... 33
      5.2.2 The Morphosyntactic Tagset ................. 34
   5.3 The Tagger .......................................... 36
      5.3.1 Introduction .................................. 36
      5.3.2 Tagging Rules ................................. 36
      5.3.3 To Tag or Not To Tag? .......................... 39
   5.4 The Grammar Checker ............................... 40
      5.4.1 Introduction .................................. 40
      5.4.2 The Control Predicate ......................... 40
      5.4.3 The Grammar Rules ............................ 41
   5.5 Conclusion .......................................... 42

6 Algorithms, with Representative Discussion of Code .... 44
   6.1 Introduction ........................................ 45
   6.2 The Architecture of the Grammar Checker ....... 45
      6.2.1 Text Preprocessing ........................... 46
      6.2.2 Grammar Checking ............................. 48
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.3</td>
<td>Tagging</td>
<td>48</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Text Postprocessing</td>
<td>48</td>
</tr>
<tr>
<td>6.3</td>
<td>Running the Grammar checker</td>
<td>49</td>
</tr>
<tr>
<td>6.4</td>
<td>Conclusion</td>
<td>49</td>
</tr>
<tr>
<td>7</td>
<td>Ispell</td>
<td>52</td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>53</td>
</tr>
<tr>
<td>7.2</td>
<td>What is Ispell?</td>
<td>53</td>
</tr>
<tr>
<td>7.3</td>
<td>Components of Ispell</td>
<td>53</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Buildhash</td>
<td>53</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Munchlist</td>
<td>54</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Findaffix</td>
<td>55</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Tryaffix</td>
<td>55</td>
</tr>
<tr>
<td>7.3.5</td>
<td>Icombine</td>
<td>56</td>
</tr>
<tr>
<td>7.4</td>
<td>Work done so far with Ispell and Irish</td>
<td>56</td>
</tr>
<tr>
<td>7.5</td>
<td>Conclusion</td>
<td>56</td>
</tr>
<tr>
<td>8</td>
<td>Conclusion</td>
<td>59</td>
</tr>
<tr>
<td>8.1</td>
<td>Overview</td>
<td>60</td>
</tr>
<tr>
<td>8.2</td>
<td>Evaluating the Grammar Checking Tool</td>
<td>60</td>
</tr>
<tr>
<td>8.3</td>
<td>Future Work</td>
<td>61</td>
</tr>
<tr>
<td>8.4</td>
<td>Conclusion</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>62</td>
</tr>
<tr>
<td>A</td>
<td>Program Code</td>
<td>64</td>
</tr>
<tr>
<td>A.1</td>
<td>txt2pl.c</td>
<td>65</td>
</tr>
<tr>
<td>A.2</td>
<td>grammarcheck.pl</td>
<td>76</td>
</tr>
<tr>
<td>A.3</td>
<td>tagger.pl</td>
<td>86</td>
</tr>
<tr>
<td>A.4</td>
<td>utilities.pl</td>
<td>124</td>
</tr>
<tr>
<td>A.5</td>
<td>pl2txt.pl</td>
<td>126</td>
</tr>
<tr>
<td>B</td>
<td>Parole Morphosyntactic Tagset</td>
<td>129</td>
</tr>
<tr>
<td>B.1</td>
<td>Parole Common Morphosyntactic Tagset</td>
<td>130</td>
</tr>
</tbody>
</table>
# List of Figures

2.1 The Celtic Language Family Tree .................................................. 6
2.2 Irish-speaking Population between 1799 and 1851 .......................... 8
2.3 Eclipses of Consonants in Irish ...................................................... 12

5.1 Categories & Attributes of the Parole Tagset ................................. 35

6.1 Algorithm for txt2pl.c in Pseudocode ........................................... 50
6.2 Flow Chart of Grammar Check Program ......................................... 51

7.1 Input to Buildhash ............................................................................ 54
7.2 Creating Dict-File and Affix-File ................................................... 57
7.3 Architecture of Ispell ......................................................................... 58
Abstract

This project investigates the computational problem of grammar checking with respect to Modern Irish. A ‘first pass’ solution is provided with functionality limited to the checking of short distance grammatical dependencies. The grammar is implemented using finite state techniques as it is intended that the current implementation will ultimately be linked with a spell checker that uses lexical resources which have been optimised using finite state techniques. This potential is also investigated. The project also incorporates a program for tagging an Irish wordlist with grammatical information.
Chapter 1

Introduction
1.1 Project Goals

The first goal of this project is to investigate the computational problem of grammar checking with respect to Modern Irish, and to provide a ‘first pass’ solution to the problem with limited functionality, but with the potential for reuse or extension. The process of grammar checking should ultimately be linked with the process of spell checking, however for the purposes of this project, I’ve separated the two tasks, and approached the grammar checking part of proofing as an isolated task. The reason for this is that the problem of spell checking with specific regard for Modern Irish has already been addressed (McGettigan, 1999; Colleran, 2000; Burke, 2002), and an ideal solution would be to design a grammar checking tool that could ultimately be added to an existing spell checker. Thus the second goal of this project is to examine the existing spell checking tools available for use with Modern Irish, and also to look at how they might be used as a platform for grammar checking.

1.2 Project Overview

This section describes how the body of this dissertation is structured.

1.2.1 Chapter 2: Modern Irish

The purpose of this chapter is to give a brief history of the Irish Language, and also to give a general description of the Modern Irish Language for the benefit of those readers who may be unfamiliar with the appearance and structure of the Irish Language. Information about the orthography, syntax, and morphology of Irish is provided, as an understanding of this is necessary when reading the approach taken herein towards implementing a grammar checker for Irish.

1.2.2 Chapter 3: Proofing Tools for Irish

This chapter looks at previous work done in the area of computational proofing for the Irish Language. Much of the work has been done by previous CSLL students here at Trinity College Dublin. This work was focused at providing an Irish spell checker that would interface with Microsoft Office products. This work began with the undergraduate CSLL
students Úna McGettigan and Nicola Colleran, and culminated with an optimised efficient spelling checker with facility for both error detection and correction, produced by Michael Burke, a CSLL graduate, as his M.Sc. by research. All of this work was done under the supervision of Dr. Carl Vogel, as is this current project. Other work in the area has also been done, and this is also described in Chapter 3.

1.2.3 Chapter 4: Grammar Checking

Chapter 4 attempts to describe what it means to do ‘grammar checking’. As every language differs in so many ways from the next, grammar checking of any given language has to be done with specific regard for the constructs of that language. In effect, while a generic spell checking tool such as the UNIX tool Ispell, can exist for use with many differing languages, such a generic ‘grammar checker’ cannot exist. Thus what it is to do ‘grammar checking’ of Modern Irish must be defined. More specifically, I describe aspects of Irish syntax and morphology that are peculiar to Irish, or at the very least, differ from that of English.

An initial ambition was to integrate limited grammar checking capabilities into existing spell checkers, for example Ispell — see Chapter 7, or the Microsoft licensed spell checker of Burke (2002). However, as this means both prototyping a grammar checker from scratch and integrating it with existing systems, I focus on a grammar checker prototype. However it will become clear that I have also made steps towards integration.

1.2.4 Chapter 5: A Prototype Implementation

Chapter 5 describes my approach to Irish grammar checking as implemented in this project. One of the main challenges to overcome was to obtain an accurate and extensive wordlist with grammatical information associated with entries in the wordlist. Such a resource was not available thus an alternative solution had to be sought. The solution was to write a program that would somehow deduce from a word its grammatical category. This represents one of the main contributions of the work in that it provides a lexical resource that had not existed before. This could then be used to either tag the available wordlists ‘offline’, or else be used to tag the words of a text being checked ‘online’. The latter is the technique being employed in this particular implementation, and this process is explained in Chapter 5.
1.2.5 Chapter 6: A Description of the Code

Chapter 6 describes the code. The technique employed to implement the grammar checker, as described in Chapter 5, has been implemented primarily in the Prolog programming language. The algorithms used are looked at here with representative discussion of the code. Some text preprocessing is done on text files being checked by the grammar checking program to make them compatible for use with Prolog. This process is also described in Chapter 6. Files which have been checked are converted back into their original text format using a Perl program.

1.2.6 Chapter 7: Interfacing with Ispell

As stated above, one of the goals of this project is to examine the existing spell checking tools available for use with Modern Irish, and to look at how they might be used as a platform for grammar checking. Having already investigated the Irish spell checking resources in Chapter 3, Chapter 7 focuses on the UNIX interactive spell checking tool Ispell. As Ispell can be used as a spell checker for Irish and its source code is freely available to download and extend, it is an ideal candidate for use as a platform for introducing grammar checking capabilities to an existing spell checker. For this reason, I have decided to dedicate Chapter 7 of this project to documenting Ispell, as well as suggesting how it might be extensible for use as an Irish grammar checker. Ispell already has a user interface in the form of an interface with emacs, a UNIX editor.

1.3 Concluding Remarks and Future Work

The final chapter of this project is a summary of what has been achieved in this project. As well as discussing problems that were encountered and overcome, problems that are yet to be overcome are discussed. Interesting challenges yet to be tackled are also discussed under the guise of ‘future work’.
Chapter 2

Modern Irish
2.1 Introduction

The Irish language has a long and interesting history as both a written language and a spoken language, thus a brief overview of the history of Irish is given in this chapter. As this project is concerned with the provision of proofing tools for use with Irish, it is necessary to provide further information on Irish as it exists today in written form. This chapter provides an introduction to the features of Modern Irish orthography, morphology, and syntax, relevant to this project.

2.2 The Irish Language: A Brief History

The Modern Irish Language is a descendant of the hypothetical ancestor-language Indo-European. Irish belongs to the Celtic branch of the Indo-European language family and survives alongside Scottish Gaelic, Welsh and Breton as a spoken Celtic language. The form of Gaelic that has become what is known as Irish today, came to the island of Ireland by invading Gaels about 300 B.C., it is thought. It subsequently spread to the Isle of Man and to Scotland. The languages spoken in in the Isle of Man and in Scotland gradually diverged from that which was spoken in Ireland, and became languages in their own right known as Manx and Scottish Gaelic. The last native speaker of Manx, Ned Maddrell, died in 1974 at the age of 97. There are still between 20,000 and 30,000 native speakers of Scottish Gaelic residing mainly in the Western Highlands and Islands of Scotland.

![Celtic Language Family Tree](image)

Figure 2.1: The Celtic Language Family Tree

Today Irish is in every sense a minority language, surviving only as a community language in outlying and diminishing rural districts in the West of Ireland known as the
CHAPTER 2. MODERN IRISH

Gaeltacht. The population of the Gaeltacht areas is now at less than 20,000 with almost no monoglots remaining. What is interesting about Irish though is its sharp demise from a language that was spoken by the entire population of Ireland and which boasted an extensive literary culture, to its current status as a minority language, strictly subordinate to English in almost every part of Irish society.

In order to qualify the unprecedented decline of the Irish language, it is necessary to describe the heights from which the Irish language fell. Wardhaugh (1987) puts the status of the Irish language in Ireland and Europe between the sixth and ninth centuries in perspective.

‘Ireland was a Celtic domain outside the Roman Empire and after the Roman legions were withdrawn from England the Celts in Ireland found themselves in a strong enough position either to resist or to assimilate any other non-Celtic people in Ireland. Ireland’s ‘Golden Age’ is the period between the sixth and the ninth centuries. The Irish language flourished during this period and it was the only vernacular language of any western European country that came to rival Latin as a language suitable for education and writing. By the eighth century it had attained sufficient prestige so that it could even replace Latin in monasteries and for religious purposes and by the tenth century the Irish Celts had successfully asserted their hegemony over everyone else in Ireland — other Celts, Picts, Anglo-Saxons, Norse, and Danes alike. All had been successfully gaelicized.’ (Wardhaugh, 1987, p. 90)

How then could a language with such dominance, in its own domain at least, begin to decline? Irish continued to flourish into the second millennium A.D., but invading forces began slowly to make their mark on the country, and more importantly on the language of the country. When Anglo-Norman invaders turned their attentions to Ireland in the latter part of the twelfth century, Ireland is said to have been 100% Irish speaking. The Anglo-Norman invaders were initially assimilated into the Celtic community. The English language was at this point still only in its infancy, and was still being influenced heavily by Danish and by Norman French, while much of the islands of Britain and Ireland remained Celtic speaking. From this point on however, despite the number or Irish speakers growing in parallel with a growth in population on the island of Ireland, the percentage of the population that spoke Irish decreased with the introduction of English and Norman French
into towns on the east coast such as Dublin. Dramatic decline in the use of Irish took place in the seventeenth and eighteenth centuries for a variety of reasons. A combination of the fact that the English were trying to impose the English language on the people, and the fact that English was fast becoming the language of trade and government, leaving Irish to be seen as the language of the poor, lead many people to abandon the use of Irish. By 1731 about two thirds of the population spoke Irish decreasing to about 50% by 1799, and by 1851, after the famine which decimated mainly the poorer Irish speaking communities, only about 25% of the population still spoke Irish with only about a quarter of those being monoglots (Ó Ruairí, 1999). Some estimates of the Irish-speaking population between 1799 and 1851 are shown in Figure 2.2. The table shown in Figure 2.2 is based on figures found in (Hindley, 1990).

<table>
<thead>
<tr>
<th>Date</th>
<th>Total Irish-speaking</th>
<th>Irish only</th>
<th>bilingual</th>
<th>Total Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>1799</td>
<td>2,100,000</td>
<td>800,000</td>
<td>1,600,000</td>
<td>5,400,000</td>
</tr>
<tr>
<td>1812</td>
<td>3,000,000</td>
<td></td>
<td></td>
<td>5,937,856</td>
</tr>
<tr>
<td>1821</td>
<td>3,740,000</td>
<td></td>
<td></td>
<td>6,801,827</td>
</tr>
<tr>
<td>1835</td>
<td>4,000,000</td>
<td></td>
<td></td>
<td>7,767,401</td>
</tr>
<tr>
<td>1841</td>
<td>4,100,000</td>
<td></td>
<td></td>
<td>8,175,124</td>
</tr>
<tr>
<td>1851</td>
<td>1,524,286</td>
<td>319,602</td>
<td></td>
<td>6,552,365</td>
</tr>
</tbody>
</table>

Figure 2.2: Irish-speaking Population between 1799 and 1851

The number of Irish language speakers dwindled even further right into the twentieth century at an alarming rate, so much so that many had predicted the death of the Irish language before the close of the millennium. Thankfully these predictions have not become a reality, and despite the fact that its continuing survival as the native language of a portion of the Irish population is still in doubt, its survival as a language that will be in use from day to day in Ireland has become almost a certainty in the last decade. The current status of the Irish language is the culmination of a process of revival which began over a hundred years ago, shortly before the setting up of the Irish State.

Irish is now the medium of communication in 176 schools outside of the Irish speaking Gaeltacht areas in Ireland, and is compulsory as a subject in most all other schools. It is an option for study at all universities in Ireland as well as many other universities across the globe. There exist many Irish language publications, as well as an Irish language nationwide radio station and an Irish language television channel. Irish has also become the medium of
communication in many workplaces. The point of all of this in terms of this current study is that in this ‘information age’ dominated by computing and communications technology, it is increasingly evident that the availability of tools as simple as spell checkers which are assumed by users of dominant languages like English, are vital to bring people to use the Irish language in modern environments.

2.3 Irish Orthography

The written history of Irish has traditionally been divided into four distinct eras: Old Irish (c. A.D.600-900), Middle Irish (c. 900-1200), Early Modern Irish (c. 1200-1600) and Modern Irish. A written standard for Modern Irish was published in 1959. Since the remaining Irish speaking areas are separated from each other, three quite distinct dialects have developed, and it was deemed necessary to bring the written language into conformity with one centralised standard to aid learners of the language, amongst other things.

Today, most Irish that is written is written using the Roman alphabet which was inherited from Latin. However, this alphabet was only widely accepted for use with Irish in the latter part of the twentieth century. Previous to that, Irish had been written down using a uniquely Gaelic medieval variant of the Latin alphabet. This Gaelic alphabet has only 18 letters: 13 consonants and 5 vowels. The letters j,k,q,v,w,x,y, and z of the Roman alphabet were not used in this Gaelic alphabet as the sounds that might have been represented by those letters either don’t exist in Irish or are depicted differently orthographically. For example, the letter j is used in English orthography to represent a kind of palatalised alveolar consonant. The Irish language, like Russian, has a dual system of consonants whereby there exists both a palatalised and a velarised version of most consonants. The terms palatalisation and velarisation refer to the secondary articulation of consonants. The palatalised consonants are known as ‘slender’ consonants, or caol in Irish, and the velarised consonants are known as ‘broad’ consonants, or leathan in Irish. A slender and broad consonant pair is depicted orthographically using one consonant symbol and the vowels immediately surrounding it denote whether it is slender or broad, that is, a consonant with a slender vowel i or e on either side of it will be slender and a consonant with a broad vowel a, o or u on either side of it will be broad. Thus unlike English orthography for example, a palatalised alveolar consonant will be depicted using a d with slender vowels on either side of it, and not with a j.
CHAPTER 2. MODERN IRISH

With modernisation in mind however, and for simplicity in computing terms, the Roman alphabet has been adopted. All twenty-six letters of that alphabet are used in written Irish, however the letters that had been absent from the traditional Gaelic alphabet are only used in loan words from other languages, foreign place-names, scientific and mathematical terms, etc. In the case of the vowels, long vowels may occur and these are denoted using the vowel symbol with an acute accent above it.

2.4 Irish Syntax

Syntax pertains to the study of grammatical relations between words and other units within a sentence. Languages often tend to be described in terms of the basic or at least most common syntactic structure of the sentences of the language. Irish can be described as being a ‘Verb-Subject-Object’ language, or VSO language for short. Undoubtedly, for a proofing tool to do grammar checking on Irish, it would have to be programmed with an understanding of the Syntax of Irish. The syntax of the well-formed clauses of the language would have to be encoded using sophisticated syntactic formalisms. This type of grammar checking however is outside the scope of this project so an in depth description of Irish syntax would be excessive at this point. Ó Siadhail (1989) provides a more complete description of Irish syntax. The scope of the ‘Grammar Checking’ done by the prototype implementation of this current study is described in more detail in Chapter 4, however for the moment it suffices to say that a description of the morphology of Irish would be more relevant to the current study and thus the morphological system of Irish is described in the next section.

2.5 Irish Morphology

Morphology refers to the study of the grammatical structure of words and the categories realized by them. Modern Irish has a rich morphology. That is to say, much grammatical information is encoded into the words themselves. The morphology of English is usually described in terms of inflection and derivation, for example, *cats* is a morphological inflection of the noun *cat* as the addition of an *-s* signals a difference in grammatical form between singular and plural, and *driver* is a morphological derivation of the verb *drive*, the former being derived from the latter. Irish morphology cannot be described merely in
terms inflection and derivation. The morphological phenomenon of *mutation*, a significant feature of Celtic languages must be introduced.

### 2.5.1 Mutation

*Initial mutation*, as the name suggests, is a mutation of the beginning of a word. It does not constitute an inflection, as the form of the word remains the same. Changes in the form of a word, or inflection, is sometime referred to as ‘final mutation’, however the use of the term ‘mutation’ will be restricted here to alterations to the beginnings of words, and alterations to the ends of words will be described as inflections. The term mutation pertains specifically to a grammatically conditioned alteration of consonants, especially at the beginnings of words, however in orthographic terms with regard to Irish, the term mutation is used herein as an umbrella term when describing any type of orthographic change at the beginning of a word. Morphological rules in Irish can in one sense be seen as being related to phonological rules in that they they are sometimes determined by their phonetic environment. The alternation in English between ‘a’ and ‘an’ for example can be seen as being determined by the phonetic environment. However morphological rules in Irish are also determined by grammatical and lexical considerations. The two types of initial mutation known as *lenition* and *eclipses* are now discussed. The grammatical conditions under which lenition and eclipses take place are described using examples in Chapter 4.

**Lenition**

The phonetic manifestation of the process of lenition is that plosives become continuants, and that there is a loss of tension in laterals, vibrants and nasals. In more general terms, a sound that has been lenited can be said to have been ‘weakened’. The lenition of certain letters is not depicted orthographically, such as the lenition of *r* which is expressed by palatalisation. These such lenitions are not relevant to the present study. The cases in which lenition are depicted orthographically are on the other hand central to this study and are as follows. Lenition of the letters *c*, *g*, *t*, *d*, *p*, *b*, *s*, *f*, and *m* is depicted orthographically with the insertion of a *h* following the letter. There are cases however where lenition of the above letters should occur but it doesn’t. The letter *s* is not lenited if it is followed by *t*, *p*, *c* or *m*. The letters *d*, *t*, *s*, *l*, and *n* are generally not lenited if the preceding consonant
CHAPTER 2. MODERN IRISH

<table>
<thead>
<tr>
<th>Basic Consonant</th>
<th>Eclipsed Consonant</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>gc</td>
</tr>
<tr>
<td>g</td>
<td>ng</td>
</tr>
<tr>
<td>t</td>
<td>dt</td>
</tr>
<tr>
<td>d</td>
<td>nd</td>
</tr>
<tr>
<td>p</td>
<td>bp</td>
</tr>
<tr>
<td>b</td>
<td>mb</td>
</tr>
<tr>
<td>f</td>
<td>bfh</td>
</tr>
</tbody>
</table>

Figure 2.3: Eclipses of Consonants in Irish

is homorganic.

**Eclipses**

The phonetic manifestation of the process of eclipses is that voiced plosives become nasalised, and voiceless consonants become voiced. An example of the nasalisation of a plosive is the change from a $b$ sound to an $m$ sound. An example of a voiceless consonant sound in English is the sound depicted orthographically with an $f$. Its voiced counterpart would be depicted in English orthography with a $v$. The same opposition is depicted in Irish orthography with an $f$ (voiceless), and a $bhf$ (voiced). The way in which the letters $c$, $g$, $t$, $d$, $p$, $b$, and $f$ are eclipsed orthographically in Irish is shown in Figure 2.3. Vowels can also be eclipsed in Irish. Vowel eclipses is denoted orthographically by prefixing an $n$ which is appended to the beginning of the word with a hyphen. Eclipses of vowels in Irish encodes specific grammatical information such as number or case and thus is not comparable to the alternation in English between ‘a’ and ‘an’.

And what about the grammatical function of lenition and eclipses? Ó Siadhail (1989) gives an overview of the circumstances where lenition occurs under the three headings of proclitics, compounds, and attributive combinations. The grammatical functions of eclipses are slightly more limited than those of lenition with the triggers of eclipses being limited to either proclitics or numerals. As the grammar checker implemented in this project checks for the correct placement of lenition and eclipses under a limited set of circumstances, the focus is on the effect of proclitics and compounds upon the beginning of a following verb or noun. Examples of proclitics in Irish include preverbal particles, as shown in (1), or the use of prepositions before nouns, as shown in (2). Examples of compounds in Irish include
the employment of adjectives attributively before nouns, as shown in (3), the use of degree adverbs before adverbs and adjectives, as shown in (4) and (5) respectively, and the use of numerals before nouns, as shown in (6).

(1) *brisim* ‘I break’
    *Ní bhrisim* ‘I don’t break’

(2) *Meiriceá* ‘America’
    *dó Meiriceá* ‘from America’

(3) *fear* ‘a man’
    *seanfhear* ‘an old man’

(4) *gaofar* ‘windy’
    *an-gaofar* ‘very windy’

(5) *maith* ‘good’
    *an-mhaith* ‘very good’

(6) *bád* ‘a boat’
    *trí bhád* ‘three boats’

### 2.5.2 Mutation, Inflection, and the Present Implementation

The correct usage of mutation in written Irish is the main focus of the current implementation, as the grammar checking process only deals with short distance grammatical dependencies. It is for this reason that the inflectional system of Irish has not been looked at in detail here. Some grammar rules governing the inflection of words are looked at, but the limitations of the current implementation become apparent when trying to deal with the inflectional system of Irish. The focus of the current implementation thus remains on the detection of grammatical errors which can be determined based only on on the immediate context. More is said about the scope of the grammar checker implemented in this project in Section 4.2.

### 2.6 Conclusion

This chapter gave a brief history of the Irish language as well as short introduction to the orthography of Irish. As the prototype implementation being described in this thesis deals
only with the system of mutation in Irish, the main focus of this chapter has been on the morphology of initial mutation in Irish. Detailed description of Irish syntax as well as the inflectional system of Irish have been avoided as they do not feature strongly in the current implementation.
Chapter 3

Irish & Proofing Tools
3.1 Introduction

The implementation of a proofing tool with grammar checking capabilities is a task that is inextricably linked to the well known and previously tackled task of spell checking. An approach to providing a grammar checker must at some stage involve either the use of an existing spell checker as a resource for the grammar checking tool, or the creation of a new spell checking tool created as part of the grammar checking tool. After all, words must be spelt correctly before the relationship between those words can be attested.

More specifically however, both tasks rely on lexical resources and thus it makes sense to pool such resources. Grammar checking however requires more information about the words than spell checking does, thus integrating the two means adapting lexical data structures optimised for assumptions required for orthography alone to carry additional information relevant to grammar checking.

In the case of this project however, as the prototype implementation described herein is presented for possible use in conjunction with an existing Irish spell checking tool, it is necessary here to describe what resources are available in terms of Irish spell checkers and their capabilities. Firstly, however, I would like to give an introduction to spell checking in the more general sense, without specific regard for Irish language spell checkers.

3.2 Spell Checking Systems

It is worth noting at this point that there are two major areas in any type of proofing that are distinct in many senses. The first is the task of error detection, whereby errors are located and marked in some sense. The second is error correction whereby possible correct alternatives for errors are suggested, or in some cases, errors are corrected automatically. However, the use of the term Spell Checker can often be generalised, as is the case in this report, to mean both detection and correction. The same convention would apply to the use of the term Grammar Checker; however, the grammar checking implementation described in this report is limited to having only error detection capabilities.

The most basic way to implement a spell checker is to store the words of the language to be checked as a list in the computer memory, and to check each word in the file being checked against this list. If a word is not found in the list, the word is marked as incorrect. A basic algorithm for error correction would suggest possible alternatives to words not
CHAPTER 3. IRISH & PROOFING TOOLS

found in the dictionary. These suggestions could be described as 'near misses' in that they would be words contained in the dictionary that differed from the word not found in the dictionary in terms of just one letter.

Spell checkers have moved on in terms of sophistication in comparison with the type of spell checker described above, however in many ways, the basic ideas are the same. Users of Unix based operating systems will be familiar with the spell checking tool called Ispell. Ispell is a very useful spell checker in that it is designed in such a way that essentially it is extensible for use with any language. It is conceivable that any basic spell checker should work with any language if a dictionary for that language is available to the spell checking engine, however Ispell incorporates other features that are also extensible to any language. It provides facility for the reduction of dictionary size which can be used for any language, provided information is provided about the morphology of the language being added. Thus Ispell is both an efficient and a generic spell checker and should be an ideal platform to build grammar checking capabilities onto. This potential is explored in more detail in Chapter 7, along with a more thorough documentation of how Ispell works.

While Ispell does seem like an ideal spell checker, to achieve an optimal solution in terms of efficiency with a certain language, it is better to design a spell checker with the specific morphology of that language in mind. For this reason, the spell checkers that have been developed thus far for Irish, have been developed independently of any existing spell checker. The work done so far in this area is described in the next section, as these existing tools can also be looked upon as potential host platforms for any grammar checking tool being developed.

3.3 Irish Spell Checkers

Although the UNIX Spell Checking Tool Ispell can be used for spell checking Irish Language documents under UNIX, and is considered in Chapter 7 of this thesis as a potential platform for the implementation of Irish grammar checking capabilities, much of the work done so far on Irish Proofing tools has been focused on the provision of tools that interface with Microsoft Office Products. A summary of the work done so far is now given, and although it may seem like a somewhat biased account, much of this work has been carried out here at Trinity College also.
CHAPTER 3. IRISH & PROOFING TOOLS

3.3.1 A Microsoft Licensed Irish Spell Checker

There are many advantages to designing a spell checker that will work with Microsoft Office Products. As was mentioned in Chapter 2, many Irish students study Irish at all levels. It is safe to say that much of the Irish word processing that will be done by these students will be done using Microsoft Word. This will also be the case in other situations where Irish word processing tasks are a regular occurrence such as in governmental departments. Therefore, it is Almost imperative that a proofing tool for Irish work with Microsoft Office. From the programmer’s point of view, by interfacing the proofing tool with an existing word processor, the focus can be entirely on the implementation of the spell checker.

The task of designing an efficient Spell Checking Tool and interfacing it with Microsoft Office products has just recently been completed, and the add-on can be downloaded from http://office.microsoft.com/downloads/2002/ptk.aspx

This work was undertaken and completed by Michael Burke (Burke, 2002) under the supervision of Dr. Carl Vogel. His implementation inherited work done by both Úna McGettigan in her final year project entitled A Spell Checker for Irish (McGettigan, 1999), and Nicola Colleran in her final year project entitled Interfacing an Irish Spellchecker to Microsoft Word 2000 (Colleran, 2000).

The implementation of this particular spell checker involved the construction of sophisticated data structures motivated by finite state machines to efficiently store a huge wordlist that was compiled specifically for this spell checker. Such optimal data structures allow for very fast online access to the words in the wordlist and make for a very efficient spell checker. As was stated, is is hoped that the current implementation will eventually be used to extend such a spell checker as this, thus it makes sense to try and maintain the finite state machine in the design of the grammar checker. For this reason, the prototype presented in this thesis has been designed as a finite state grammar checker. More is said about this in Chapter 4.

3.3.2 Other Irish Spell Checkers

Other independent work has been carried out in designing an Irish spell checker which interfaces with Microsoft Office products. A program known as Gaelspell has been developed by John P. Mullen, of the New Mexico State University. The program is available to
download at http://www.gaelspell.com/.

3.4 Conclusion

This chapter has looked at the proofing task of spell checking and the way in which it is closely related to the task of grammar checking at hand in this project. As the current project is being designed in such a way that it may ultimately be extended to interface with an existing spell checker, the existing Irish language spell checkers were reviewed in this chapter. It was noted that the optimisation of spell checkers is closely related to finite state technology and that this is a motivation for the current prototype grammar checker to be implemented using a finite state grammar.
Chapter 4

Grammar Checking
CHAPTER 4. GRAMMAR CHECKING

4.1 Introduction

Grammar checking can in a sense be seen as an extension of spell checking which takes context and grammatical features of the word being examined into account. The question to be answered by the grammar checking part of a proofing tool will be, if a word is spelt correctly, is its form (in terms of inflection, mutation, derivation etc.) correct, as determined by its context. Those familiar with English grammar checking in Microsoft Word for example will think of grammar checking as an irritating and generally inaccurate tool which highlights what it deems to be inaccurate in terms of the overall structure of sentences. This type of grammar checking can be seen as a sort of stylistic checking which can be rather subjective. An example of such subjectivity would be the way in which prescriptive grammars and style guides might warn against sequences of prepositional phrases although in many cases such sequences are most appropriate. The problem also lies in the state of the technology for computationally encoding a grammar of natural language. What such a grammar checker for English will do accurately on the other hand is highlight discrepancies in short distance grammatical dependencies such as subject-verb agreement. It is with this area of short distance grammatical dependency that this current project is concerned. Short distance grammatical accuracy is an area of grammar checking that is much more significant in Irish than it is in English. The system of initial mutation in Irish was introduced in section 2.5. The processes of lenition and eclipses in Irish, which are part of this system of initial mutation, will be discussed further in this chapter. Firstly though, further description of the type of grammar checking being implemented in the current project is necessary.

4.2 Finite-State Grammar Checking

The extent to which the current implementation checks for grammatical errors is limited. A small grammar is built up as a set of rules pertaining to the way in which short distance grammatical dependencies should be adhered to in Irish. This grammar however is limited in the sense that it is a Finite-State Grammar for Irish. What this means basically is that when scanning a text for errors, given a certain state, that particular state may only be followed by states which are legal given the particular finite-state grammar defined. The motivation for this is that the lexical resources used by the Microsoft licensed Irish Spell
CHAPTER 4. GRAMMAR CHECKING

checker of (Burke, 2002) are optimised using finite state technology. If a grammar checker such as the current implementation were to ultimately draw on the same lexical resources, such a grammar checker should be designed in such a way as to maintain the efficiency of the data structures. The actual system presented in Chapter 5 is based on an equivalent notion of passing information through ‘registers’ constituting an N-token window on the context being checked. Using such a technique, adjacent grammatical dependencies, and relatively close distance grammatical dependencies may be dealt with.

4.3 Limitations of Finite-State Technique

Using this Finite-State technique certain limitations become apparent. Certain agreement requirements will be overlooked by the grammar checker. In example (8) below, an adverbial phrase will cause the finite-state grammar checker to loose track of the fact that ni in example (7), becomes nach because of the verb sílim which precedes the adverbial phrase in example (8), and as a result now acts as a complementiser as well as a negative particle and causes eclipses instead of lenition. The correct version of (8) is shown in (9).

(7)  Ní chónaíonn sé ann
doesn’t live he there
‘He doesn’t live there’

(8)  * Sílim ar an gcéad dul síos ní chónaíonn sé ann

(9)  Sílim  ar an gcéad dul síos nach gcónaíonn sé ann
think I in the first instance doesn’t live he there
‘First of all, I don’t think he lives there’

It is conceivable that any number of adverbials could be slotted in in the place of or along with ar an gcéad dul síos which would cause some sort of a loop, and would hence disrupt the grammar checking process.
4.4 Scope of Implementation

4.4.1 Introduction

In Section 2.5, the phenomenon of mutation in Irish was introduced, more specifically the occurrence of initial mutation in the form of \textit{lenition} and \textit{eclipses}. It was said that these features of Irish morphology would become central to the grammar checking implemented in this project as the grammatical function of mutation is dependent on immediate context or at least very close context. This section describes many of the ways in which mutation can occur in Irish and thus gives the reader an idea of the many grammatical functions of mutation. More specifically however, it describes in detail the types of grammatical errors which will be detected by the finite-state grammar being constructed herein for Irish.

The triggers of initial mutation can be categorised as either \textit{proclitics}, \textit{compounds}, \textit{attributive combinations}, and \textit{numerals}. Various proclitics, compounds, attributive combinations and numerals that cause mutation as well as the type of mutation they cause are now described. Firstly, the causes and manifestations of initial mutation in verb phrases are described by example, and secondly the causes and manifestations of initial mutation in noun phrases are described by example. With all examples given, examples illustrating incorrect initial mutation are shown first. Ungrammatical examples are preceded by an asterisk in every case.

4.4.2 The Verb Phrase

Preverbal Particles - Present/Future Tenses & Conditional Mood

The following preverbal particles cause certain initial mutations to occur with verbs occurring in the present tense, future tense, or conditional mood. These are documented below:

- NEGATIVE

  - Nf:
    Lenites following verb if it begins with a lenitable consonant.

  (10) *Ní cónaíonn mé ann

  (11) *Ní gcónaíonn mé ann
CHAPTER 4. GRAMMAR CHECKING

(12) Ní chónaíonn mé ann
don’t live I there
‘I don’t live there’

● INTERROGATIVE

– An:
Eclipses following verb if it begins with a eclipsable consonant.

(13) * An dúnann sé
(14) * An dhúnann sé
(15) An ndúnann sé
does close it
‘Does it close’

– Nach:
Eclipses following verb if it begins with a eclipsable consonant. Prefixes ‘n-’ to following verb if it begins with a vowel.

(16) * Nach déanann tú spórt
(17) * Nach dhéanann tú spórt
(18) Nach ndéanann tú spórt
don’t do you sport
‘Don’t you do sports’

(19) * Nach ólann tú fuisce
(20) Nach n-ólann tú fuisce
don’t drink you whiskey
‘Don’t you drink whiskey’

– Cé:
Lenites following verb if it begins with a consonant.
(21) *Cé dúnnann an doras
(22) *Cé ndúnnann an doras
(23) Cé dhúnnann an doras
    who closes the door
    ‘Who closes the door’

– Cad/Céard:
The particle ‘a’ is placed before the verb following verb is lenited if it begins
with a consonant.

(24) *Cad a seinneann tú
(25) Cad a sheinneann tú
    what PRT play you
    ‘What do you play’

– Cathairn/Cén uair:
The particle ‘a’ is placed before the verb following verb is lenited if it begins
with a lenitable consonant.

(26) *Cathairn a dúnnann an siopa
(27) *Cathairn a ndúnnann an siopa
(28) Cathairn a dhúnnann an siopa
    when PRT closes the shop
    ‘When does the shop close’

Preverbal Particles - Past Tense

The following preverbal particles cause certain initial mutations to occur with verbs occurring in the past tense. These are documented below:

• NEGATIVE
- Níor:
  Lenites following verb if it begins with a lenitable consonant.

(29) * Níor cónaigh mé ann
(30) * Níor gcónaigh mé ann
(31) Níor chónaigh mé ann
didn’t live       I there
‘I didn’t live there’

- INTERROGATIVE

- Ar:
  Lenites following verb if it begins with a lenitable consonant.

(32) * Ar dún sé
(33) * Ar ndún sé
(34) Ar dhún sé
did close it
‘Did it close’

- Nár:
  Lenites following verb if it begins with a lenitable consonant.

(35) * Nár bris tú do chos
(36) * Nár mbris tú do chos
(37) Nár brís tú do chos
didn’t break you your leg
‘Didn’t you break your leg’

(38) * Nár n-ól tú
(39) Nár ól tú
didn’t drink you
‘Didn’t you drink’

- Cár:
  Lenites following verb if it begins with a lenitable consonant.

  (40)  * Cár cuir tú d’airgead
  (41)  * Cár g’ceir tú d’airgead
  (42)  Cár chuair tú d’airgead

  where put you your money
  ‘Where did you put your money’

  (43)  * Cár n-inigh tú
  (44)  Cár inigh tú

  where go you
  ‘Where did you go’

- Cé:
  Lenites following verb if it begins with a lenitable consonant.

  (45)  * Cé dún an doras
  (46)  * Cé ndún an doras
  (47)  Cé dhún an doras

  who closed the door
  ‘Who closed the door’

- Cad/Céard:
  The particle ‘ar’ is placed before the verb following verb is lenited if it begins
  with a consonant.

  (48)  * Cad ar seinn tú
  (49)  Cad ar sheinn tú

  what PRT played you
  ‘What did you play’
CHAPTER 4. GRAMMAR CHECKING

– **Cathain/Cén uair:**

The particle ‘ar’ is placed before the verb following verb is lenited if it begins with a consonant.

(50) * Cathain ar dún an siopa
(51) * Cathain ar ndún an siopa
(52) Cathain ar dhún an siopa

when  PRT closed the shop
‘When did the shop close’

### 4.4.3 The Noun Phrase

**Dative Case**

A noun is in the dative case when it occurs following a preposition. Inflected dative case forms have largely died out in Irish except for a finite number of fossilised exceptions, an example of which is shown in (53). However, *initial mutations* do occur in the dative case.

(53)  

| **Éire** | ‘Ireland’ |
| **In Éirinn** | ‘In Ireland’ |

• Nouns following the simple prepositions ó ‘from’, de ‘of’, do ‘to/for’, tríd ‘through’, roimh ‘before’, mar ‘as’, um ‘towards’ and faoi ‘under’, following which the indefinite article is implied, are lenited.

• Nouns following the simple preposition i ‘in’ are eclipsed.

• Nouns following the simple preposition ar are lenited when ar is acting semantically as the preposition ‘on’ with an implied indefinite article following it. However it does not cause lenition when it forms part of a petrified phrase such as those shown in (54). The semantic opposition here is a source of ambiguity which puts the detection of an error regarding the use of lenition following ar beyond the scope of the current implementation.
CHAPTER 4. GRAMMAR CHECKING

ar bord  
  aboard
ar cairde  
  on credit
ar clé  
  to the left
ar crith  
  trembling
ar crochadh  
  hanging
ar deil  
  in proper working order
ar deis  
  to the right
ar díol  
  on sale
ar fiuchadh  
  boiling
ar foluain  
  hovering
ar lasadh  
  alight
ar leathshúil  
  having only one eye
ar maidin  
  in the morning
ar malairt  
  in exchange
ar meisce  
  drunk
ar mire  
  mad
ar seachrán  
  astray
ar tuarastal  
  for a salary

- Nouns following the simple prepositions ag ‘at’, as ‘from’, go ‘to’, le ‘with’, and chuig ‘to’ remain unchanged.

- The simple preposition go ‘to’ is followed by dtí when it precedes the article.

- The simple preposition le ‘with’ becomes leis when it precedes the article.

- When a simple preposition is followed by the definite article, the following noun can be either lenited or ellipsed, depending on dialect, except in the following cases.

  - When the simple preposition de ‘of’ is combined with the article an, it becomes den, and always lenites the following noun if that noun begins with a lenitable consonant.

  - When the simple preposition do ‘to/for’ is combined with the article an, it becomes den, and always lenites the following noun if that noun begins with a lenitable consonant.
– When the preposition *i* ‘in’ is combined with the article it becomes *sa*, and **always** lenites the following noun if that noun begins with a lenitable consonant. If the following noun begins with a vowel, *sa* becomes *san*.

- The simple prepositions *dí* ‘from’ and *faoi* ‘under’ are amalgamated with the article *an*, as they end with vowels.

\[
\begin{align*}
\hat{d} + \text{an} & \rightarrow \text{ún} \\
faoi + \text{an} & \rightarrow \text{faoín}
\end{align*}
\]

**Numbers**

The numbering system in Irish is a source of much difficulty to learners of the language as a result of the irregular way in which the numbers effect the words which they precede. These dependencies are described below. Mutation caused by numbers does not actually carry semantic information in the way that some of the other types of mutation looked at does, rather it is a manifestation of the phonetic environment.

- **Initial Mutation:**

  – Words following the numbers 1 - 6, and 11 - 16 are lenited.
  – Words following the numbers 7 - 10, and 17 - 19 are eclipsed.

- **Exceptions:**

  There are a finite number of nouns that have a historical plural form which is used following a number. When these particular nouns occur following the numbers 3 - 6, and 13 - 16, they are not lenited. However, a *h-* is prefixed to those examples that begin with a vowel. The words in question are shown in (56), with the plural form that is used with numbers shown between the brackets.
CHAPTER 4. GRAMMAR CHECKING

bliain [bliana] year
ceann [cinn] thing
cloch [clocha] stone
cloigeann [cloigne] head
fiche [fidhid] twenty
orlach [orlaf] inch
pingin [pingine] penny
seachtain [seachtaine] week
slat [slata] yard
troigh [troithe] foot (measurement)
uair [uaire] hour
ubh [ubhe] egg

4.5 Conclusion

In this chapter, the task of grammar checking was looked at with specific regard to the project at hand. It was concluded that a type of finite-state grammar checker would be optimal when implementing a grammar checker for Irish as it would be desirable to be able to maintain efficient finite-state data structures used in the optimisation of existing spell checkers should the current prototype ever be extended for use with such a spell checker.

The limitations of such a finite-state technique were looked into as well the types of short distance dependencies in Irish that can be determined as being grammatical or ungrammatical using a finite-state implementation such as this current prototype. Using examples, it was illustrated what sort of grammatical dependencies are examined by the current implementation, as well as some seemingly simple examples that unfortunately are outside the scope of the current implementation.
Chapter 5

The Implementation
CHAPTER 5. THE IMPLEMENTATION

5.1 Introduction

The purpose of this chapter is to describe the design and functionality of the prototype implementation mentioned in the title of this report. It is hoped that the grammar checker implemented as part of this project will be seen as a step in the right direction towards the goal of providing efficient and accurate proofing tools for use in any environment with Irish. This chapter aims to describe what this particular implementation does, as well as what it doesn’t do, and also to describe how it does what it does.

5.2 Grammatical Information

The implementation of a spell checker involves the compilation of an extensive and accurate word list. In order to add grammar checking capabilities to a spell checking engine, grammatical information must be appended to all of the entries in the spell checker’s dictionary. The most extensive and accurate Irish word list available is that which was compiled for use with Michael Burke’s Irish Spell checker (Burke, 2002). However, this list lacks grammatical information. If grammar checking capabilities were to be added to this particular spell checker, or any other Irish spell checker that used this dictionary, some automated method for tagging the roughly 250,000 entries in this wordlist with grammatical information would have to be sought, as it is unrealistic to actually hand-tag each word.

5.2.1 Word Classes and Part-of-Speech Tagging

The process of part-of-speech tagging (or just tagging for short) is the process of assigning part-of-speech or other lexical class marker to each word in a corpus. Tags are usually applied to punctuation markers; thus tagging for natural language is the same process as tokenization for computer languages(i.e. segmenting units and determining their category), although tags for natural language are much more ambiguous.

There are a number of different tagging algorithms. The input to a tagging algorithm is a string of words and a specified tagset. The output is a single best tag for each word. Taggers can be characterised as being either rule-based or stochastic. Rule-based taggers use hand-written rules to distinguish tag ambiguity. Stochastic taggers are either HMM-based (Hidden Markov Model), choosing the tag sequence which maximises the product of
word likelihood and tag sequence probability, or cue-based, using decision trees or maximum models to combine probabilistic features.

The Irish wordlist referred to above was compiled from various sources, including various corpora, however the final wordlist was not tagged. The problem with tagging a wordlist as opposed to a corpus is that it is not possible to eliminate tag ambiguity using any of the methods described above. Tagging must be done in an ambiguous manner, and the ambiguities must be dealt with at some other stage during the grammar checking process. Tag ambiguity elimination is outside the scope of the current project, however the initial process of ambiguously tagging the words automatically was completed.

5.2.2 The Morphosyntactic Tagset

Before going on to a description of how the tagging of the wordlist was done, the tagset used must be described. When deciding upon a tagset to use, a number of factors must be considered. The tagset must be sufficiently rich to encode all the necessary grammatical information to do grammar checking. However it is also useful to adopt a standard tagset, if it exists, for reasons of compatibility with other similar work being done. Fortunately, such a tagset existed for Irish that was sufficiently rich in its encoding of grammatical information, and which was in use as a standard tagset in 14 European countries.

This tagset is known as The Parole Common Morphosyntactic Tagset and incorporates categories and attributes that are common to the 14 participating languages (Belgian & Swiss French, Catalan, Danish, Dutch, English, French, Finnish, German, Greek, Irish, Italian, Portuguese, Spanish and Swedish). Institúid Teangeolaíocht Éireann (ITÉ) have been using this tagset in the linguistically annotated subset of the PAROLE Irish corpus, and as a result have already modified the tagset sufficiently for use with Irish. It is this tagset, as it is in use by ITÉ, that I have adopted for use in this project.

The Parole tagset has 15 categories, each with their own set of attributes. These are shown schematically in Figure 5.1.

The full tagset including possible values for attributes is given in Appendix B. Some of the attributes do not apply to Irish, and some have been added specifically for Irish. Information such as this is also illustrated in Appendix B, as well as some examples as to how ITÉ have been using the tagset to tag Irish Corpora.

A tag for a word will consist of a value for each of the attributes of an appropri-
CHAPTER 5. THE IMPLEMENTATION

<table>
<thead>
<tr>
<th>Category</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>noun</td>
<td>(POS, Type, Gender, Number, Case, Sem-Gender, Contrast, Usage)</td>
</tr>
<tr>
<td>verb</td>
<td>(POS, Type, Mood, Tense, Person, Number, Gender, Dependency, Neg)</td>
</tr>
<tr>
<td>adjective</td>
<td>(POS, Type, Degree, Gender, Number, Case)</td>
</tr>
<tr>
<td>pronoun</td>
<td>(POS, Type, Person, Gender, Number, Case, Possessor)</td>
</tr>
<tr>
<td>determiner</td>
<td>(POS, Type, Person, Gender, Number, Case, Possessor)</td>
</tr>
<tr>
<td>article</td>
<td>(POS, Type, Gender, Number, Case)</td>
</tr>
<tr>
<td>adverb</td>
<td>(POS, Type, Degree, Function, Wh-ness)</td>
</tr>
<tr>
<td>adposition</td>
<td>(POS, Type, Formation, Gender, Number, Person)</td>
</tr>
<tr>
<td>conjunction</td>
<td>(POS, Type, Ctype, Coord-pos)</td>
</tr>
<tr>
<td>numerals</td>
<td>(POS, Type, Gender, Number, Case)</td>
</tr>
<tr>
<td>interjection</td>
<td>(POS)</td>
</tr>
<tr>
<td>unique membership class</td>
<td>(POS, Particle Type, B-Function)</td>
</tr>
<tr>
<td>residuals</td>
<td>(POS, Type)</td>
</tr>
<tr>
<td>punctuation</td>
<td>(POS, Type)</td>
</tr>
<tr>
<td>abbreviation</td>
<td>(POS)</td>
</tr>
</tbody>
</table>

Figure 5.1: Categories & Attributes of the Parole Tagset

The tagging process then must associate the tag with the word somehow. Some of the attributes can be marked for non-applicability, or can be left underspecified. Non-applicability is denoted using an underscore and means that that particular attribute doesn’t apply in Irish for the category in question. Underspecification is denoted using a hyphen. An example follows.

- *fear ncmsn_n*

  The first example shows a possible tagging of the Irish word *fear*, meaning man. The tag stands for: “noun, common, masculine, singular, nominative, n/a, unmarked”. As the nominative singular and genitive plural forms of the noun *fear* are the same, this word can also be tagged as follows:

- *fear ncmprg_n*

  In this case the tag stands for: “noun, common, masculine, plural, genitive, n/a, unmarked”. An alternative to both of the above tags for the word *fear* would be to leave the attributes of number and case underspecified as follows:

- *fear ncm--_n*
- This way the word can be taken to be either singular or plural and either nominative or genitive.

5.3 The Tagger

5.3.1 Introduction

A spell checker needs a wordlist with all forms of all words of the language being checked by the spell checker. A dictionary will provide headwords and in some cases other possible forms of the headword, thus it can be used as a source when compiling such a wordlist. However the task of representing the whole lexeme in the wordlist used by a spell checker poses problems for inflecting languages such as Irish. In many case, the stem of an inflected form of a word does not correspond to the headword in a dictionary, and thus the complete paradigm of the lexical item cannot always by predicted from the dictionary entry.

Burke (2002) used techniques of computational morphology as well as corpora extraction to compile a comprehensive wordlist that contains root words as well as their various inflected and mutated forms, however as was stated earlier, this wordlist lacks an encoding of grammatical information pertaining to the wordlist entries. Automated tagging of the entries with grammatical information would require that the tagger program understand how the words will appear in both their root word form and their various inflected and mutated forms.

The method used here to do this tagging is based on the compilation of tagging rules which decide whether or not a word is mutated based on the beginning of the word, and predict the grammatical category of a word including its form, based on the end of the word. This section represents one of the most labour intensive parts of the project, given that it involved incremental refinement of rules based on direct examination of word forms. This process is now described.

5.3.2 Tagging Rules

The tagging rules can be classified in two distinct categories. The first set of rules determine the state of initial mutation of a word. These will be described henceforth as the *prefix rules*. The rest of the rules determine the form of the word, that is the form of the end of
the word, from which the grammatical category and all appropriate attribute values can be predicted. These rules will henceforth be described as suffix rules.

The Prefix Rules

The prefix rules need to determine firstly whether or not a word is mutated. If it is the case that the word is mutated, they then must also describe the type of mutation in question. Types of initial mutation include lenition and eclipses as discussed in Section 2.5, however there are other modifications which occur to the beginnings of words in Irish do not strictly constitute mutation, however as was stated in Section 2.5.1, the term mutation will be used herein to cover other types of alterations which occur to the beginnings of words. One such example is the prefixing of $h$ to an initial vowel following another vowel to avoid vowel hiatus. Ó Siadhail (1989) describes this as constituting ‘no change’ in the system of initial mutation in Irish, however, when grammar checking written Irish, it is necessary to acknowledge the presence of such a ‘prefixed’ $h$ if its use in the written language is to be marked as ‘correct’ or ‘incorrect’ by a program for automated grammatical error detection. Other examples include the prefixing of a $t$ following the article to uninflected feminine singular nouns and masculine singular nouns in the genitive case which begin with an $s$, the prefixing of a $t$ with a hyphen to uninflected masculine nouns which begin with a vowel following the article, and the prefixing of a $h$ to genitive singular feminine nouns which begin with a vowel. The checking for correct usage of such prefixes is outside the scope of the current implementation however, as it involves the taking into account of wider context.

If a word is lenited it is marked [l] and if it is eclipsed it is marked [e]. As was described in Section 2.5.1, in some cases words are not lenitable or eclipsable. This fact must be taken into account so as that a word that is not mutable is not marked as being ‘incorrect’ for not being mutated where mutation would normally apply. With this in mind, the rules were designed to tag words as either [lenitable], i.e. beginning with a consonant that is lenitable, and [eclipsable], i.e. beginning with a vowel or a consonant that is eclipsable. The ordering of these rules is important. The rules which tag a word as being lenited, [l], or eclipsed, [e], occur before the rules pertaining to mutability, thus if a word is tagged as being either [lenitable] or [eclipsable], it can be taken for granted that the word is neither lenited or eclipsed. This allowed for simpler rules to check mutability, needing only to look at the first character of a word, in the knowledge that the word is neither lenited or
CHAPTER 5. THE IMPLEMENTATION

eclipsed.

Regarding other types of ‘initial mutation’ as described above, the only error that is
checked for in the current implementation is the absence of a prefixed ‘h’ where it should
occur. Instead of tagging a word as having a prefixed ‘h’, words beginning with a vowel
are tagged as [vowel], and thus the absence of a prefixed ‘h’ is implicitly encoded.

The Suffix Rules

The suffix rules are used to determine the grammatical category of a word as well as to
predict the correct values for the attributes of the grammatical category in question. As
the initial goal was to tag an existing wordlist, and to do so without any of the tagging
techniques mentioned in Section 5.2.1, the method used was to compile a list of possible
suffixes for Irish words and to associate each suffix with a set of possible tags.

This list of possible suffixes was compiled with the aid of A Reverse Dictionary of
Modern Irish (Doyle & Gussmann, 1996). This dictionary provides the complete paradigm
for the main nominal and adjectival headwords in (Ó Dónaill, 1977) as well inflected
forms of verbs, pronouns and prepositions that are not predictable from the headword.
The dictionary only provides the stems for regular verb entries as the conjugations of
regular verbs are almost completely regular in Irish. Thus when compiling the list of
suffixes the various personal verb ending had to be included despite them not being in
the reverse dictionary. Some unusual or dialectal forms of words which are included in
the reverse dictionary were ignored, as the current implementation is in keeping with the

Certain word forms in Irish are very helpful as regards what they can tell you about the
word. For example, most words that end in a broad consonant tend to be masculine nouns
of the first declension. For more explanation on the declension system of Modern Irish, see
(Mac Giolla Phádraig, 1963). In brief though, nouns in the first declension are those which
are ‘slenderised’ to form the genitive case. What slenderisation means in orthographic
terms is that the slender vowel ‘i’ is inserted before the final consonant and the phonetic
manifestation is that the final consonant becomes palatalised. Palatalisation is where there
is secondary articulation of the consonant at the palat. Slenderisation of a final consonant
is also an indication of a nominative plural form in Irish. Some other examples of the sort
of information that can be deduced about a word based on its suffix are given below.
CHAPTER 5. THE IMPLEMENTATION

- -i: plural noun form
- -eog: Feminine noun form in nominative case
- -eöige: Feminine noun form in genitive case
- -im: 1st person singular present tense verb form
- -óimid: 1st person plural future tense verb form

Lexical Tagging

It became apparent during the process of compiling the tagging rules that words classes with only a small number of words should be manually tagged, thus ensuring accurate tagging of words belonging to those word classes at least. Prepositions for example could be manually tagged as there are only a finite number of them. Grammatical particles could also be tagged manually as again there is only a small number of such words. The advantage of this is that the way in which such words would be tagged could then be known and thus used in grammar rules involving those words. In the case of the current implementation this was extremely desirable as most grammar rules written involve the effect of grammatical particles, prepositions, and other such words which were manually tagged on the words which they proceed.

5.3.3 To Tag or Not To Tag?

Thus far, I have spoken about tagging which a view to tagging an existing wordlist that is in use by an existing Irish spell checker. The tagger program as it is implemented is capable of tagging such a wordlist, however the way in which the prototype grammar checker is currently implemented does not require the use of any wordlist. Instead it was decided to tag the words of a text being checked ‘online’. What this means essentially is that the grammar checker tags the words of a text as they are read in, the tags of adjacent words are then compared for grammatical compatibility and then the tags are discarded. This method of tagging is less efficient, however it is an adequate approach to achieve the goals stated at the outset of this paper. Ultimately, if the grammar checking tool was interfaced with a spell checker that accessed a wordlist, it would be desirable for such a wordlist to be tagged ‘offline’ for grammatical category on just one occasion, thus saving on the computing overhead of ‘online’ tagging every time a grammar checking process is called. One of the reasons for not taking this step was that the tagger, as it is currently
implemented, tags words in an ambiguous manner. It was thought that this tagging process should be optimised and checked for accuracy before a wordlist should be tagged ‘offline’ for subsequent use by the grammar checker.

5.4 The Grammar Checker

5.4.1 Introduction

This section looks at how the grammar checker program, written in Prolog, uses the information provided by the tagger program about the words of a text file being checked to check for accuracy with short distance grammatical dependencies. The program takes as input a file which has been formatted for use with Prolog by the program `txt2pl`. This program is described in more detail in Section 6.2.1.

5.4.2 The Control Predicate

The Prolog predicate `main/7` takes as arguments the input and output streams, as well as five ‘registers’ which constitute the ‘N-register’ window on the context being checked, as talked about is Section 4.2, and thus controls the iteration through the file being checked as well as the writing to the new ‘corrected’ version of the original file. The five registers constitute a three word window, R1, R3 and R5, with R2 and R4 storing space between words. Each time a word or an item of punctuation is read from the input stream, `tagit/2` from `tagger.pl` is called on the item and the fifth register is instantiated to the resulting set of tags for that item. At this point, the predicate `check/6` is called. The different instances of `check/6` constitute the grammar checking rules, and these are described in the next section. The sixth argument of `check/6` will be instantiated to a particular state of grammaticality or ungrammaticality. If it is the case that the current state is ungrammatical, the type of ungrammaticality is then written to the output stream and `main/7` is recursed upon, moving grammatical tags stored in R5 to R4, and putting grammatical information on the next item in the input stream into R5.
CHAPTER 5. THE IMPLEMENTATION

5.4.3 The Grammar Rules

The grammar rules are encoded in the various instances of the predicate check/6 contained in grammarcheck.pl, and as was described in the last section, check/6 is called from the control predicate main/7 which is also contained in grammarcheck.pl. If any instance of check/6 succeeds, excluding the last instance, the particular window being checked will be deemed as ungrammatical. The check rules defined look at two word and three word contexts. Words are deemed to be governed by a previous word only in the case where they are separated by a space, and not by any sort punctuation. Thus check rules will only succeed if words in the window being checked are each separated by a single space. In the body of the predicate, the predicate compatible/2 is used to check what type of word or punctuation is contained in a particular register. compatible/2 works much like the predicate member/2 except that it checks if a list is a member of a list of lists. So what it does is check if a particular tag-list is a member of the list of tag-lists contained in a particular register, succeeding only if the word represented by the list of tag-lists contained in the register matches the type of word being sought by the check rule.

The check rules basically search for certain types of words and then check if the subsequent words are in keeping with the effect the particular word being searched for has on a word it precedes. For example, the Irish first person singular possessive pronoun *mo*, meaning ‘my’, causes a noun following it to be lenited. The tagger program tags the word *mo* as follows:

\[
\text{mo }[[\text{lenitable}],[d,p,1, -,s,-A,-B]]
\]

The rule pertaining to the correct usage of *mo*, i.e. that a following word must be lenited, searches for a register that represents the word *mo* by querying compatibility as follows:

\[
\text{compatible}([d,p,1, -,s, - , _ ] , \text{Register})
\]

If the above query succeeds, then it checks if the following word is not lenited. In the case of ungrammaticality, the check rule needs to succeed, thus the above rule which pertains to the first person singular possessive pronoun *mo* succeeds if the word which
follows *mo* is tagged as [lenitable]. As was explained in Section 5.3.2, with the ordering of
the tagging rules, the tagging of a word as `[lenitable]` implies that the word has not been
lenited but that it could have been.

The rules as they have been defined thus far accurately check the usage of some possess-
itive pronouns, all prepositions, all prepositions followed by the definite article, and some
preverbal particles. The rules regarding possessive pronouns are limited to the first and the
second person, as the third person possessive pronoun singular and plural, masculine and
feminine is written as *a*, `his, her, its, their`, and thus is ambiguous. If *a* means `his/its`
it lenites a following consonant, if *a* means `her` it prefixes a `h` to a following vowel, and
if *a* means `their` it causes eclipses of following vowels or consonants. The current im-
plementation cannot disambiguate the sense of a word like *a*. The types of prepositions and
preverbal particles and their effects that are checked are shown by example in Section 4.4.

It was stated in Section 2.5.2 that providing rules that check if words in a text being
checked are inflected correctly is somewhat outside the scope of the current implementa-
tion. The problem is that the correct form of a word in Irish can sometimes be determined by
another word which can in some cases can be an arbitrarily long distance away.

Another rule which has been implemented relies on the tagger program providing ac-
curate information regarding the form of the word. The rule checks that the rule in Irish
which requires that adjectives following feminine nouns occurring in the nominative case
should be lenited if possible is fulfilled. This rule is less accurate than some of the other
rules because of the possibility that words that are not actually feminine nouns in the
nominative case are tagged as being so. Increasing the accuracy in an area such as this
depends directly on the optimisation of the tagging program.

### 5.5 Conclusion

In this chapter, the implementation was described. As the present implementation involves
the tagging of words for grammatical category as well as other information, the problem of
tagging was described. The Parole Morphsytactic tagset being used by the tagger program
of this implementation was also described. A description was then given on how the process
of grammar checking takes place, and the way in which the tagger program is called by the
grammar checking program to tag words `online`. The information associated with each
word by the tagger program is then compared by the grammar checker for grammatical
dependency if any using check rules.
Chapter 6

Algorithms, with Representative Discussion of Code
6.1 Introduction

The next section of this chapter gives a schematic overview of the overall architecture of the prototype grammar checker implementation presented in this project, showing the various components of the program and summarising what they do and how they interact with each other. Subsequent sections describe in more detail the function of the various components of the implementation. The last section describes how the program is run.

6.2 The Architecture of the Grammar Checker

- `grammarcheck.pl` the ‘main’ grammar checking program which is written in the Prolog Programming Language.
  - Interface: ‘grammarcheck(IrishTextFile,CheckedIrishTextFile).’
  - ‘grammarcheck/2’ calls the c program `txt2pl.c` to convert the input to a format more compatible with Prolog.
  - It then opens input/output streams and calls the main grammar checking function, i.e. the predicate `main/7`.
  - `main/7`
    * Arguments 1 and 2 are the input/output streams.
    * Arguments 3 - 7 are registers to contain grammatical information.
    * The registers are populated by a call to `tagit/2` passing it the current ‘item’ in the input stream. `tagit/2` is contained in `tagger.pl`.
    * `check/6` then matches with a series of rules to check for grammaticality within a three word range (5 registers are used to take account of space between words).
    * If a check rule succeeds, it’s last argument is passed to `mark/2` which writes an error, if applicable, to the output stream.
  - `grammarcheck/2` exits when recursion on main terminates at the end of the input stream.

- `tagger.pl`
CHAPTER 6. ALGORITHMS, WITH REPRESENTATIVE DISCUSSION OF CODE

- *tagger.pl* contains *tagit/2* which tags words for grammatical category based on their suffixes and is called from *main/2*.
- *tagit/2* also tags punctuation accordingly.
- Another important feature of tagit is that it tags for initial mutation using prefix rules. This information is used in the ‘check’ rules in *grammarcheck.pl* as most short distance grammatical errors in Irish involve errors at the beginning of the word.

* Input and Output

- *txt2pl.c*
  * Converts words from the input text file to prolog format, one per line: ‘word’ ⇒ word([w,o,r,d]). It does likewise with punctuation, e.g. ‘.’ ⇒ punct([hyphen]).
- *pl2txt.pl*
  * Converts the checked prolog file back to original text format with grammatical errors marked in. This is a perl program.

6.2.1 Text Preprocessing

As the current implementation deals with text conversion, attention must be payed to the *Coding System* being used when creating an Irish text document upon which one intends to run the Grammar Checker. As Irish texts contain acute accents, some sort of extended character set will be required. An ideal character set for use with Irish is *Latin-1*. If you are using Emacs, the following steps should be followed when creating a file that you wish to check using the Grammar Checker.

C-x C-f ~/Irish.txt //Create File
M-x iso-accents-mode //convenient emacs mode for displaying accents on the screen
//write stuff
M-x set-language-environment: Latin-1
C-x C-s //Save File
CHAPTER 6. ALGORITHMS, WITH REPRESENTATIVE DISCUSSION OF CODE

OR

C-x C-f ~/Irish.txt //Create File
M-x iso-accents-mode
//write stuff
C-x C-s //Save File
<you will be prompted to select coding system>
Select: iso-8859-1

txt2pl

As the Prolog grammar checker deals only with words and punctuation which are in the form of Prolog terms denoting words or punctuation, a text preprocessing program had to be written. As one of the goals of the project at the outset was to investigate the possibility of interfacing the grammar checker with the UNIX spell checker Ispell which is written in the C Programming language, it was decided to write the text preprocessor in C also. The program converts words in a text into Prolog terms denoting words, with the letters of a word forming a Prolog list which is an argument of the ‘word’ term constructor. Some examples of this are shown in (57). Punctuation has also to be dealt with accordingly. Any punctuation which has special function in Prolog is written by name and becomes the sole element of the list which is an argument of the punctuation term constructor ‘punct’. Other punctuations are simply reprinted as the sole element of the list which is an argument of the punctuation term ‘punct’. Control characters and spacing are also represented as punctuation for the Prolog grammar checker, however, the main reason for them being accounted for is so that the text file can be regenerated exactly as it appeared in the first instance except with the addition of flags marking grammatical errors if any. Examples of how punctuation is converted for use with Prolog is shown in (58). To avoid problems with how Prolog deals with accented characters, the text preprocessor also converts accented characters to Prolog terms denoting accented characters. Only acute accents occur in Irish. An acute accent is known as a ‘fada’ in Irish thus an accented character such as ø is converted to f(a), f standing for ‘fada’ and denoting the fact that the a is acutely accented. Another problem that had to be dealt with was uppercase characters in the text. As Prolog interprets anything beginning with an uppercase character as a variable, it was necessary to convert an uppercase character such as A to the Prolog term c(a), with the c standing
CHAPTER 6. ALGORITHMS, WITH REPRESENTATIVE DISCUSSION OF CODE

for ‘capital’ and denoting the fact that the \( a \) is uppercase. The program also has to decide whether or not hyphens and apostrophes are acting as punctuation or as part of a word. It interprets them to be part of a word if there is an alphabetical character on both sides of them. The basic algorithm can be described in pseudocode as is shown in Figure 6.1

(57) Brian is converted to word([\( c(b), r, i, a, n \)]).
    álaimh ‘beautiful’ is converted to word([\( f(a), l, a, i, n, n \)]).
    Éire ‘Ireland’ is converted to word([\( c(f(e), i, r, e) \)]).

(58) ’ is converted to punct([apostrophe]).
    \( \backslash n \) is converted to punct([newline]).

6.2.2 Grammar Checking

The grammar checking program is the main program in the current implementation. The grammar checking program is described in more detail in Section 5.4, and the full source code is shown in Appendix A.2. The program takes an Irish text file as input and writes a version of that text with grammatical errors marked to a new text file. The grammar checking program calls the text preprocessing and postprocessing programs before and after calling the main grammar checking routines. The main grammar checking program calls on \texttt{tagger.pl} to tag words during the grammar checking process. A schematic diagram showing the flow of control when the predicate \texttt{grammarcheck/2} from \texttt{grammarcheck.pl} is called is shown in Figure 6.2.

6.2.3 Tagging

The program which tags words for grammatical category is called \texttt{tagger.pl}. The tagging process is described in more detail in Section 5.3. The full source code for \texttt{tagger.pl} is shown in Appendix A.3. The way in which the tagger program is called during the grammar checking process is shown schematically in Figure 6.2.

6.2.4 Text Postprocessing

The postprocessing of the checked Irish text file involves reversing all of the text formatting done by the program \texttt{txt2pl}, and regenerating the original text file exactly as it was except
CHAPTER 6. ALGORITHMS, WITH REPRESENTATIVE DISCUSSION OF CODE

with grammatical errors marked in. The program which does this is called `pl2txt.pl`. This particular program is written in the Perl programming language and the source code can be found in Appendix A.5.

6.3 Running the Grammar checker

To run the grammar checker, run Prolog and consult the file `grammarcheck.pl`. This loads the other components of the implementation written in Prolog, i.e. `tagger.pl` and `utilities.pl`. To grammarcheck an Irish text file `Input.txt`, type the following command at the prolog prompt:

```
? - grammarcheck(Input,OutputFileName).
```

The Original file `Input.txt` will remain unchanged, and running the above command will result in the creation of two new files, namely `OutputFileName`, and `OutputFileName.txt`. The former is the file marked for grammatical errors before postprocessing, i.e. in Prolog format as defined for the current implementation, and the latter being the text file identical to the `Input` text file with grammatical errors marked in.

6.4 Conclusion

In this section, the implementation as described in 5 was described in terms of how the various components of the program have been pieced together. The overall architecture of the code, as well as the flow of control of the grammar checking program was illustrated using schematic representation. Details on how to run the code were also given.
open input file and new output file

read in file character by character

if first character is word start
    print ("'word(['");

while (not at End Of File) {
    if (current character is lowercase)
        print character;
    if (next character is part of word)
        print comma;
    else
        print ("']'.");

    if (current character is uppercase)
        convert to lowercase;
    print ("c(character)'");
    if (next character is part of word)
        print comma;
    else
        print ("']'.");

    if (current character is accented)
        convert to unaccented character;
    print ("f(character)'");
    if (next character is part of word)
        print comma;
    else
        print ("']'.");

    if (current character is punctuation)
        print ("punct([name of punctuation]).'")
}

Figure 6.1: Algorithm for txt2pl.c in Pseudocode
CHAPTER 6. ALGORITHMS, WITH REPRESENTATIVE DISCUSSION OF CODE

grammarcheck.pl

Interface: grammarcheck(Input,Output).

main(unchecked,checked,R1–R5)
  Tagit(CurrentItem,Tags)
  Check(R1–R5,Result)
  Mark(checked,Result)
  write(checked,Result)
  Write(checked,Item)
  R4 = R5
  R5 = Next Item

main(End_of_File,checked,R1–R5)
  close(InStream)
  close(OutStream)

pl2txt(checked,Output)

txt2pl.c
Tagger.pl
tagit/2
pl2txt.pl

Figure 6.2: Flow Chart of Grammar Check Program
Chapter 7

Ispell
CHAPTER 7. ISPELL

7.1 Introduction

As was stated in the introduction to this thesis, one of the goals of the project was to look at existing Irish spell checkers and to document them as potential platforms for hosting a grammar checking engine. Other Irish spell checkers which are available are discussed in 3. As was stated there, Ispell is another spell checker which can be used to spell check Irish language documents. The present chapter is intended to describe how Ispell works and to propose the future exploration of Ispell in terms of its data structures and algorithms as to how they might by used to incorporate grammar checking capabilities.

7.2 What is Ispell?

Ispell is an interactive spell-checking program for UNIX, for which an emacs interface is available as well as the standard command-line mode. The way it works is that it attempts to find near misses that might include the word you meant. Ispell can handle languages other than English. For Ispell to be able to do spell checking for Irish for example, all it needs is an Irish word list. In fact, all it needs is a raw word list. This is a word list made up only of root words which are tagged to indicate how they can be inflected, conjugated, mutated etc. You must provide Ispell with an affix file which indicates what these tags mean. From this Ispell automatically generates a hashed dictionary file. A hashed dictionary file is a very efficient indexed dictionary.

7.3 Components of Ispell

The various components used to prepare Ispell for spell checking with a new language are described in the subsequent subsections. A schematic representation of this entire offline process is given in 7.2. A diagram showing the components of the Ispell program for spell checking is shown in 7.3. This diagram illustrates the components of Ispell involved in the online spell checking process.

7.3.1 Buildhash

The buildhash program builds hashed dictionary files for later use by ispell. The raw word list (with affix flags) is given in dict-file, and the the affix flags are defined by affix-file.
The hashed output is written to hash-file. The formats of the two input files are described in `ispell(4)`. The -s (silent) option suppresses the usual status messages that are written to the standard error device. A schematic representation of the input to `Buildhash` is shown in 7.1.

![Diagram of input to Buildhash]

**Figure 7.1: Input to Buildhash**

### 7.3.2 Munchlist

The munchlist shell script is used to reduce the size of dictionary files, primarily personal dictionary files. It is also capable of combining dictionaries from various sources. The given files are read (standard input if no arguments are given), reduced to a minimal set of
roots and affixes that will match the same list of words, and written to standard output.

Input for munchlist contains of raw words (e.g. from your personal dictionary files) or root and affix combinations (probably generated in earlier munchlist runs). Each word or root/affix combination must be on a separate line.

7.3.3 Findaffix

The findaffix shell script is an aid to writers of new language descriptions in choosing affixes. The given dictionary files (standard input if none are given) are examined for possible prefixes (-p switch) or suffixes (-s switch, the default). Each commonly-occurring affix is presented along with a count of the number of times it appears and an estimate of the number of bytes that would be saved in a dictionary hash file if it were added to the language table. Only affixes that generate legal roots (found in the original input) are listed.

7.3.4 Tryaffix

The tryaffix shell script is used to estimate the effectiveness of a proposed prefix (-p switch) or suffix (-s switch, the default) with a given expanded-file. Only one affix can be tried with each execution of tryaffix, although multiple arguments can be used to describe varying forms of the same affix flag (e.g., the D flag for English can add either D or ED depending on whether a trailing E is already present). Each word in the expanded dictionary that ends (or begins) with the chosen suffix (or prefix) has that suffix (prefix) removed; the dictionary is then searched for root words that match the stripped word. Normally, all matching roots are written to standard output, but if the -c (count) flag is given, only a statistical summary of the results is written. The statistics given are a count of words the affix potentially applies to and an estimate of the number of dictionary bytes that a flag using the affix would save. The estimate will be high if the flag generates words that are currently generated by other affix flags (e.g., in English, batters can be generated by either bath/X or bather/S).
7.3.5 Icombine

The icombine program is a helper for munchlist. It reads a list of words in dictionary format (roots plus flags) from the standard input, and produces a reduced list on standard output which combines common roots found on adjacent entries. Identical roots which have differing flags will have their flags combined, and roots which have differing capitalisations will be combined in a way which only preserves important capitalisation information. The optional aff-file specifies a language file which defines the character sets used and the meanings of the various flags. The -T switch can be used to select among alternative string character types by giving a dummy suffix that can be found in an altstringtype statement.

7.4 Work done so far with Ispell and Irish

Some work has been done in the preparation of Ispell for use with Irish. Both Alastair McKinstry and Kevin P. Scannel have provided tagged raw word lists and affix files so that Ispell can be used to spell check Irish language documents. Kevin P. Scannel’s version can be downloaded from http://borel.slu.edu/~kps/ispell/. This however is as far as anyone has gone with regard to Irish in the context of Ispell. Ispell doesn’t take context into account and hence cannot report on grammatical accuracy.

7.5 Conclusion

It is hoped that this brief description of Ispell will provide a starting point for anyone wishing to investigate the data structures and algorithms of Ispell with the intention of using Ispell as a basis for implementing an Irish grammar checker. Whether or not Ispell would be a suitable candidate to host a grammar checker, such as the prototype described in this thesis, is as yet undecided.
An aid to writers of new language definitions in choosing affixes.

estimates the effectiveness of a proposed affix.

Reduces the size of the dictionary files. The given files are reduced to minimal set of roots and affixes.

Reads in a list of words in dictionary format (i.e. root plus flags) and reduces size of list.

Builds hashed dictionary files for later use by ispell.

Figure 7.2: Creating Dict-File and Affix-File
Figure 7.3: Architecture of Ispell
Chapter 8

Conclusion
CHAPTER 8. CONCLUSION

8.1 Overview

The main aim of this project was to look at the computational problem of grammar checking, and to develop a prototype grammar checker for Irish that would operate as a stand-alone application for checking Irish language text documents. As was stated in the introduction, the solution provided would only constitute a ‘first pass’ solution and thus it has been designed to detect only a restricted set of short distance grammatical dependencies. These grammatical dependencies pertain mainly to the correct employment of mutation in written Irish. Having designed such an application, it is intended here in the conclusion firstly to evaluate the application and secondly to look at what should happen next in the area of Irish proofing tools.

8.2 Evaluating the Grammar Checking Tool

The application that has been designed as part of this project will have limited success in checking any given Irish language text document for accuracy in terms of grammaticality. This however is because it has been designed in such a way as to only tackle a small subset of grammatical dependencies, and the primary goal was to look in more general terms as to how the problem of creating a grammar checking tool for Irish should be approached.

The implementation as it currently exists does not necessarily capture all short distance grammatical errors, however it is believed that it has been designed in such a way that grammar rules for cases of short distance grammatical dependency which have been overlooked can easily be added. The current state of the project could be described as ‘work in progress’ in the sense that rules can be added at any point. However the current implementation is complete in the sense that it has explored the possibility of checking for accuracy with short distance grammatical dependencies in Irish and their effect on the mutation of words. The cases in which mutation is conditioned merely by the phonetic environment and the cases where mutation carries semantic information are both treated equally here as the orthographic effect of an error is the same, and thus a grammar checker must correct both. Examples of both are covered by the current implementation.

As much of the work in this project focused on the creation of a tagging program which would tag either an existing wordlist or the words of a text being checked ‘on the fly’ so to speak, it is worth evaluating what was achieved in this regard. It is believed that the
tagger provided will be a valuable resource for future work in the area, as it is effective in what it does. Proper evaluation of the accuracy of the tagger has not yet been carried out, however, based on the meticulous manner in which the rules were compiled, it is believed that any word which is tagged by the tagger program will be tagged correctly. The only downside to this is that the program is almost certain to tag a word with some incorrect tags as well as the correct one. This is not the case however with important grammatical particles and word classes such as prepositions that only contain a small number of words. These have been manually tagged and thus will not be tagged ambiguously or incorrectly by the tagger program.

8.3 Future Work

There have been many allusions to future work throughout the course of this thesis. The main aspect of such future work which needs to be explored is the possibility of interfacing this current implementation to an existing spell checker for Irish. As optimal algorithms for spell checking Irish language documents exist, any further work done with this particular implementation would have to consider more seriously its compatibility with such an existing spell checker. There is also scope for continuing the work of refining the tagging program, as there are a number of the suffix rules which could be made more specific. Some of the rules might also be reassessed as they do not conform to standardised spelling and it might be more optimal just to stick to the official standard for Irish.

8.4 Conclusion

It is hoped that the work presented in this thesis constitutes a useful contribution to the task of providing computational proofing tools for Irish. Despite its continuing status as a minority language, Irish captivates the imagination of many with its eloquent grammatical structures. It is my belief that it will not be long before someone else takes up the task of trying to conquer this intriguing grammar and encoding it computationally.
Bibliography


Appendix A

Program Code
A.1  txt2pl.c

/*---------------------------------------------------------------------------------------------
 *
 * txt2pl.c
 * Author: Brian Ó Raghallaigh
 * Date: March 2003
 *
 * This program takes a text file as input and converts it into a list of
 * Prolog terms denoting the words and punctuation of the text file. The
 * new file will have one Prolog term per line followed by a full stop.
 *
 * For example:
 *
 * 'hello world.'
 *
 * Will be converted to:
 *
 * word([h,e,l,l,o]).
 * punct([space]).
 * word([w,o,r,l,d]).
 * punct([.]).
 *
 * As the example above illustrates, a word is converted into a prolog
 * list containing the letters of the word separated by commas and this
 * list becomes an argument of the term constructor 'word'. Punctuation,
 * including control characters and space, is denoted in the same way
 * except that the term constructor is 'punct' and the prolog list
 * contains just one element, i.e. the punctuation symbol, or else the
 * name of the punctuation in the case of control characters, space, or
 * punctuation that has special function in Prolog. Hyphens and
 * apostrophes can occur as part of a 'word' term if they occur within
 * the original word boundaries, e.g. 'passer-by' becomes
 * 'word([p,a,s,s,e,r,hyphen,b,y]).'
 *
 * The program also converts uppercase characters into prolog terms
 * denoting uppercase characters, e.g. 'C' becomes 'c(c)', with the term
 * constructor 'c' standing for 'capital'. The reason for this conversion
 * is that Prolog treats anything beginning with an uppercase character
 * as a variable. As the program is designed for use with Irish texts,
 * it does likewise with acutely accented characters, e.g. á becomes
 * f(a), with 'f' standing for 'fada', the Irish word for acute accent.
 * An 'Á' would be converted to cf(a). The reason for this conversion is
 * so that Prolog won't have to deal with character code conversions when
 * processing the output of this program.
 *
 *--------------------------------------------------------------------------------------------*/

#include <stdio.h>
#include <ctype.h>
APPENDIX A. PROGRAM CODE

```c
#define isaccented(c) (((c) >= 0xc0) && ((c) <= 0xff))
#define islexical(c) (isalpha(c) || isaccented(c) || (c == 0x27) || (c == 0x2d))
#define iswordstart(c) (isalpha(c) || isaccented(c))

int c;

extern FILE *fopen();

main(int argc, char *argv[])
{
    FILE *in,*out,*ifp,*ofp;
    void txt2pl(FILE *, FILE *);

    if (argc < 2) {
        printf("Usage: txt2pl infile > outfile\n");
        exit(1);
    }

    if((in=fopen(argv[1],"r")) == NULL){
        fprintf(stderr,"Error - cannot open %s \n",argv[1]);exit(2);
    }else{
        out=fopen(argv[2],"w");
        txt2pl(in,out);
        fclose(in);
        fclose(out);
    }
}

void txt2pl(FILE *ifp, FILE *ofp)
{
    int c;

    c = fgetc(ifp);
    if(!isspace(c) && !ispunct(c) && !iscntrl(c) && !isdigit(c)){
        fprintf(ofp,"word["");
    }
    ungetc(c,ifp);

    // Iterate through the text file character by character
    while((c = fgetc(ifp)) != EOF){
        // prints lowercase characters unchanged
        if(islower(c)){
            fputc(c,ofp);
            // if the next character in the input is lexical, a comma is printed
            if ((c = fgetc(ifp)) && islexical(c)){
```
APPENDIX A. PROGRAM CODE

```c
    fprintf(ofp,"\n");
} else{
    fprintf(ofp, "]\n");
} ungetc(c, ifp);

// The next to cases deal with apostrophes and hyphens
// occurring within a word

// Dec: 39; Hx: 27; Oct: 047; Html: &\#39
// converts ’’ to punct([apostrophe]).
else if(c == 0x27){
    fprintf(ofp, "apostrophe");
    if ((c = fgetc(ifp)) & islexical(c)){
        fprintf(ofp, ",";
    } else{
        fprintf(ofp, "]\n");
    }
    ungetc(c, ifp);
}

// Dec: 45; Hx: 2D; Oct: 055; Html: &\#45
// converts ’-’ to punct([hyphen]).
else if(c == 0x2D){
    fprintf(ofp, "hyphen");
    if ((c = fgetc(ifp)) & islexical(c)){
        fprintf(ofp, ",";
    } else{
        fprintf(ofp, "]\n");
    }
    ungetc(c, ifp);
}

else if(isupper(c)){
    c = tolower(c);
    fprintf(ofp, "c\n");
    fputc(c, ofp);
    fprintf(ofp, ")\n");
    if ((c = fgetc(ifp)) & islexical(c)){
        fprintf(ofp, ",";
    } else{
        fprintf(ofp, "]\n");
    }
    ungetc(c, ifp);
}

// converts ’á’ to ’f(a)’
else if(c == 0xe1){
    c = ’a’;
```
fprintf(ofp,"f(");
putc(c, ofp);
fprintf(ofp,")");
if ((c = fgetc(ifp)) && islexical(c)) {
    fprintf(ofp,";");
} else {
    fprintf(ofp,"]\.\n");
}
ungetc(c, ifp);

// converts 'é' to 'f(e)'
else if(c == 0xe9) {
    c = 'e';
    fprintf(ofp,"f(");
    putc(c, ofp);
    fprintf(ofp,")");
    if ((c = fgetc(ifp)) && islexical(c)) {
        fprintf(ofp,";");
    } else {
        fprintf(ofp,"]\.\n");
    }
    ungetc(c, ifp);
}

// converts 'i' to 'f(i)'
else if(c == 0xed) {
    c = 'i';
    fprintf(ofp,"f(");
    putc(c, ofp);
    fprintf(ofp,")");
    if ((c = fgetc(ifp)) && islexical(c)) {
        fprintf(ofp,";");
    } else {
        fprintf(ofp,"]\.\n");
    }
    ungetc(c, ifp);
}

// converts 'ó' to 'f(o)'
else if(c == 0xf3) {
    c = 'o';
    fprintf(ofp,"f(");
    putc(c, ofp);
    fprintf(ofp,")");
    if ((c = fgetc(ifp)) && islexical(c)) {
        fprintf(ofp,";");
    } else {
        fprintf(ofp,"]\.\n");
    }
ungetc(c, ifp);
}

// converts 'ú' to 'f(u)'
else if (c == 0xfa){
    c = 'u';
    fprintf(ofp,"f(");
    fputc(c, ofp);
    fprintf(ofp,")");
    if ((c = fgetc(ifp)) && islexical(c)){
        fprintf(ofp,"",");
    }else{
        fprintf(ofp,"[]\n");
    }
    ungetc(c, ifp);
}

// converts 'Á' to 'cf(a)'
else if(c == 0xc1){
    c = 'a';
    fprintf(ofp,"cf(");
    fputc(c, ofp);
    fprintf(ofp,")");
    if ((c = fgetc(ifp)) && islexical(c)){
        fprintf(ofp,"",");
    }else{
        fprintf(ofp,"[]\n");
    }
    ungetc(c, ifp);
}

// converts 'É' to 'cf(e)'
else if(c == 0xc9){
    c = 'e';
    fprintf(ofp,"cf(");
    fputc(c, ofp);
    fprintf(ofp,")");
    if ((c = fgetc(ifp)) && islexical(c)){
        fprintf(ofp,"",");
    }else{
        fprintf(ofp,"[]\n");
    }
    ungetc(c, ifp);
}

// converts 'í' to 'cf(i)'
else if(c == 0xcd){
    c = 'i';
    fprintf(ofp,"cf(");
    fputc(c, ofp);

APPENDIX A. PROGRAM CODE

```c
fprintf(ofp,"\n");
if ((c = fgetc(ifp)) && islexical(c)){
    fprintf(ofp,"\n");
}else{
    fprintf(ofp,"\n");
}
}

// converts '0' to 'cf(o)'
else if(c == 0x3){
    c = 'o';
    fprintf(ofp,\"cf(\n\n\n);
    fputc(c, ofp);
    fprintf(ofp,"\n");
    if ((c = fgetc(ifp)) && islexical(c)){
        fprintf(ofp,"\n");
    }else{
        fprintf(ofp,"\n");
    }
    ungetc(c,ifp);
}

// converts '0' to 'cf(u)'
else if (c == 0xda){
    c = 'u';
    fprintf(ofp,\"cf(\n\n\n);
    fputc(c, ofp);
    fprintf(ofp,"\n");
    if ((c = fgetc(ifp)) && islexical(c)){
        fprintf(ofp,"\n");
    }else{
        fprintf(ofp,"\n");
    }
    ungetc(c,ifp);
}

// converts ' ' to punct([space]).
else if (c == \')
    fprintf(ofp,"punct([space]).\n")
    if ((c = fgetc(ifp)) != EOF && iswordstart(c)){
        fprintf(ofp,"word([\n");
    }
    ungetc(c,ifp);
    if ((c = fgetc(ifp)) != EOF && ((c == 0x2D) || (c == 0x27))){
        fprintf(ofp,"punct([\n");
    }
    ungetc(c,ifp);
```
// converts '\n' to punct([newline]).
else if (c == '\n'){
    fprintf(ofp,"punct([newline]).\n");
    if ((c = fgetc(ifp)) != EOF && iswordstart(c)){
        fprintf(ofp,"word["");
    }
    ungetc(c,ifp);
    if ((c = fgetc(ifp)) != EOF && ((c == 0x2D) || (c == 0x27))){
        fprintf(ofp,"punct["");
    }
    ungetc(c,ifp);
}

// converts '\r' to punct([return]).
else if (c == '\r'){
    fprintf(ofp,"punct([return]).\n");
    if ((c = fgetc(ifp)) != EOF && iswordstart(c)){
        fprintf(ofp,"word["");
    }
    ungetc(c,ifp);
    if ((c = fgetc(ifp)) != EOF && ((c == 0x2D) || (c == 0x27))){
        fprintf(ofp,"punct["");
    }
    ungetc(c,ifp);
}

// converts '\t' to punct([tab]).
else if (c == '\t'){
    fprintf(ofp,"punct([tab]).\n");
    if ((c = fgetc(ifp)) != EOF && iswordstart(c)){
        fprintf(ofp,"word["");
    }
    ungetc(c,ifp);
    if ((c = fgetc(ifp)) != EOF && ((c == 0x2D) || (c == 0x27))){
        fprintf(ofp,"punct["");
    }
    ungetc(c,ifp);
}

// Dec: 32; Hx: 22; Oct: 042; Hml: &#34
// converts '"' to punct([quotes]).
else if (c == 0x22){
    fprintf(ofp,"punct([quotes]).\n");
    if ((c = fgetc(ifp)) != EOF && iswordstart(c)){
        fprintf(ofp,"word["");
    }
    ungetc(c,ifp);
    if ((c = fgetc(ifp)) != EOF && ((c == 0x2D) || (c == 0x27))){
        fprintf(ofp,"punct["");
    }
}
ungetc(c,ifp);
}

// Dec: 37; Hx: 25; Oct: 045; Html: &amp;#37
// converts '%%' to punct([percent]).
else if (c == 0x25){
  fprintf(ofp,"punct([percent]).\n");
  if (c = fgetc(ifp) != EOF &amp;& iswordstart(c)){
    fprintf(ofp,"word("");
  }
  ungetc(c,ifp);
  if (c = fgetc(ifp) != EOF &amp;& (c == 0x2D) || (c == 0x27)){
    fprintf(ofp,"punct(['");
  }
  ungetc(c,ifp);
}

// Dec: 39; Hx: 27; Oct: 047; Html: &amp;#39
// converts '' to punct([apostrophe]).
else if (c == 0x27){
  fprintf(ofp,"punct([apostrophe]).\n");
  if (c = fgetc(ifp) != EOF &amp;& iswordstart(c)){
    fprintf(ofp,"word(['");
  }
  ungetc(c,ifp);
  if (c = fgetc(ifp) != EOF &amp;& (c == 0x2D) || (c == 0x27)){
    fprintf(ofp,"punct(['");
  }
  ungetc(c,ifp);
}

// Dec: 40; Hx: 28; Oct: 050; Html: &amp;#40
// converts '!' to punct([openround]).
else if (c == 0x28){
  fprintf(ofp,"punct([openround]).\n");
  if (c = fgetc(ifp) != EOF &amp;& iswordstart(c)){
    fprintf(ofp,"word(['");
  }
  ungetc(c,ifp);
  if (c = fgetc(ifp) != EOF &amp;& (c == 0x2D) || (c == 0x27)){
    fprintf(ofp,"punct(['");
  }
  ungetc(c,ifp);
}

// Dec: 41; Hx: 29; Oct: 051; Html: &amp;#41
// converts ')') to punct([closeround]).
else if (c == 0x29){
  fprintf(ofp,"punct([closeround]).\n");
  if (c = fgetc(ifp) != EOF &amp;& iswordstart(c)){

APPENDIX A. PROGRAM CODE

```c
    fprintf(ofp,"word(["");
    }
    ungetc(c,ifp);
    if ((c = fgetc(ifp)) != EOF && ((c == 0x2D) || (c == 0x27))){
        fprintf(ofp,"punct(['");
    }
    ungetc(c,ifp);
}

// Dec: 45; Hx: 2D; Oct: 055; Html: &amp;#45
// converts '-' to punct([hyphen]).
else if (c == 0x2D){
    fprintf(ofp,"punct([hyphen]).\n");
    if ((c = fgetc(ifp)) != EOF && iswordstart(c)){
        fprintf(ofp,"word(["");
    }
    ungetc(c,ifp);
    if ((c = fgetc(ifp)) != EOF && ((c == 0x2D) || (c == 0x27))){
        fprintf(ofp,"punct(["");
    }
    ungetc(c,ifp);
}

// Dec: 91; Hx: 5B; Oct: 133; Html: &amp;#91
// converts ']' to punct([opensquare]).
else if (c == 0x5B){
    fprintf(ofp,"punct([opensquare]).\n");
    if ((c = fgetc(ifp)) != EOF && iswordstart(c)){
        fprintf(ofp,"word(["");
    }
    ungetc(c,ifp);
    if ((c = fgetc(ifp)) != EOF && ((c == 0x2D) || (c == 0x27))){
        fprintf(ofp,"punct(["");
    }
    ungetc(c,ifp);
}

// Dec: 93; Hx: 5D; Oct: 135; Html: &amp;#93
// converts ']]' to punct([closesquare]).
else if (c == 0x5D){
    fprintf(ofp,"punct([closesquare]).\n");
    if ((c = fgetc(ifp)) != EOF && iswordstart(c)){
        fprintf(ofp,"word(["");
    }
    ungetc(c,ifp);
    if ((c = fgetc(ifp)) != EOF && ((c == 0x2D) || (c == 0x27))){
        fprintf(ofp,"punct(["");
    }
    ungetc(c,ifp);
}
```
// Dec: 95; Hx: 5F; Oct: 137; Html: &lt;#95
// converts '__' to punct([underscore]).
else if (c == 0x5F) {
    fprintf(ofp,"punct([underscore]).\n");
    if ((c = fgetc(ifp)) != EOF && iswordstart(c)){
        fprintf(ofp,"word(['");
    }
    ungetc(c,ifp);
    if ((c = fgetc(ifp)) != EOF && (c == 0x2D) || (c == 0x27)) {
        fprintf(ofp, "punct(['");
    }
    ungetc(c,ifp);
}

// Dec: 123; Hx: 7B; Oct: 173; Html: &lt;#123
// converts '{' to punct([opencurly]).
else if (c == 0x7B) {
    fprintf(ofp,"punct([opencurly]).\n");
    if ((c = fgetc(ifp)) != EOF && iswordstart(c)) {
        fprintf(ofp,"word(['");
    }
    ungetc(c,ifp);
    if ((c = fgetc(ifp)) != EOF && (c == 0x2D) || (c == 0x27)) {
        fprintf(ofp,"punct(['");
    }
    ungetc(c,ifp);
}

// Dec: 125; Hx: 7D; Oct: 175; Html: &lt;#125
// converts '}' to punct([closecurly]).
else if (c == 0x7D) {
    fprintf(ofp,"punct([closecurly]).\n");
    if ((c = fgetc(ifp)) != EOF && iswordstart(c)) {
        fprintf(ofp,"word(['");
    }
    ungetc(c,ifp);
    if ((c = fgetc(ifp)) != EOF && (c == 0x2D) || (c == 0x27)) {
        fprintf(ofp,"punct(['");
    }
    ungetc(c,ifp);
}

// ispunct(c): printing character except space or letter or digit
// converts all other 'punctuation' to punct([punctuation]).
else if (ispunct(c)) {
    fprintf(ofp,"punct(['");
    fprintf(ofp,",ofp);
    fprintf(ofp,"]).\n");
    if ((c = fgetc(ifp)) != EOF && iswordstart(c)) {
APPENDIX A. PROGRAM CODE

```c
    fprintf(ofp,"word["");
 }
 ungetc(c,ifp);
 if ((c = fgetc(ifp)) != EOF && ((c == 0x2D) || (c == 0x27))){
     fprintf(ofp,"punct["");
 }
 ungetc(c,ifp);

// converts 'digit' to digit([digit]).
else if (isdigit(c)){
    fprintf(ofp,"digit["");
    fputc(c, ofp);
    fprintf(ofp,"]\n\n");
    if ((c = fgetc(ifp)) != EOF && iswordstart(c)){
        fprintf(ofp,"word["");
    }
    ungetc(c,ifp);
    if ((c = fgetc(ifp)) != EOF && ((c == 0x2D) || (c == 0x27))){
        fprintf(ofp,"punct["");
    }
    ungetc(c,ifp);
```
A.2 grammarcheck.pl

`grammarcheck.pl` reads an Irish language text file and calls the `txt2pl` program to format the file for use with Prolog. It then checks the file for short distance grammatical errors, marking the output with a description of the error, where it occurs.

```prolog
:- use_module(library(system)). % comment out for SWI-Prolog
:- ensure_loaded([tagger]).
:- ensure_loaded([utilities]).

% usage: systemcall(text2prolog,infile).
% calls buildcall/3

systemcall(Program,Args) :-
    buildcall(Program,Args,SC),
    system(SC,Status). % comment out for SWI-Prolog
    % shell(SC,Status). % include for SWI-Prolog

systemcall(Script,Program,Args) :-
    buildcall(Script,Program,Args,SC),
    system(SC,Status). % comment out for SWI-Prolog
    % shell(SC,Status). % include for SWI-Prolog

% checks a file that has been formatted for use with prolog
grammarcheck(FileIn,FileOut) :-
    open(FileIn,read,StreamIn),
    open(FileOut,write,StreamOut),
    main(StreamIn,StreamOut,R1,R2,R3,R4,R5).

% checks an unformatted Irish text file
% calling txt2pl to format the file
grcheck(FileIn,CFileOut) :-
    systemcall(txt2pl,FileIn),
    addextension(FileIn,PrologFileIn),
    open(PrologFileIn,read,StreamIn),
    open(CFileOut,write,StreamOut),
    main(StreamIn,StreamOut,R1,R2,R3,R4,R5),
    reformat(CFileOut,CFileOut).
```
reformat(CFileOut,CFileOut) :-
    syscall(per1.pl2txt,CFileOut).

% Control Predicate
main(StreamIn,StreamOut, R1,R2,R3,R4,R5) :-
    read(StreamIn,Item),
    Item \= end_of_file,
    tagit(Item,R5), % R5: FullTagSet of Item
    check(R1,R2,R3,R4,R5,Flag),
    mark(StreamOut,Flag),
    write(StreamOut,Item),
    nl(StreamOut),
    main(StreamIn,StreamOut, R2,R3,R4,R5,NR).

main(StreamIn,StreamOut, R1,R2,R3,R4,R5) :-
    at_end_of_stream(StreamIn),
    close(StreamIn),
    close(StreamOut).

main(_,_,_,_,_,_).

mark(StreamOut, ungrammatical01) :-
    write(StreamOut, 'LENITION OMITTED!'),nl(StreamOut).
mark(StreamOut, ungrammatical02) :-
    write(StreamOut, 'ECLIPSES OMITTED!'),nl(StreamOut).
mark(StreamOut, ungrammatical03) :-
    write(StreamOut, 'PREFIXED H OMITTED!'),nl(StreamOut).
mark(StreamOut, ungrammatical04) :-
    write(StreamOut, 'UNNECESSARY LENITION!'),nl(StreamOut).
mark(StreamOut, ungrammatical05) :-
    write(StreamOut, 'UNNECESSARY ECLIPSES!'),nl(StreamOut).
mark(StreamOut, ungrammatical06) :-
    write(StreamOut, 'MISSING CONTRACTION!'),nl(StreamOut).
mark(StreamOut, grammatical).

% possesive pronouns causing mutation

% noun following 1st person singular possesive pronoun
% noun must be lenited if possible
check(_,_,C3,C4,C5,ungrammatical01) :-
    nonvar(C3),
    nonvar(C4),
    compatible([d,p,1,-,s,_,_],C3),
    compatible([space],C4),
    % implies that it is not lenited and that it should have been
    compatible([lenitable],C5).

% noun following 2nd person singular possesive pronoun
% noun must be lenited if possible
check(_,_,C3,C4,C5,ungrammatical01) :-
nonvar(C3),
nonvar(C4),
compatible([d,p,2,−,s,−,−],C3),
compatible([space],C4),
% implies that it is not lenited and that it should have been
compatible([lenitable],C5).

% noun following 1st person plural possessive pronoun
% noun must be eclipsed if possible
check(_,C3,C4,C5,ungrammatical102) :-
    nonvar(C3),
nonvar(C4),
compatible([d,p,1,−,p,−,−],C3),
compatible([space],C4),
% implies that it is not eclipsed and that it should have been
compatible([eclipsable],C5).

% noun following 2nd person plural possessive pronoun
% noun must be eclipsed if possible
check(_,C3,C4,C5,ungrammatical102) :-
    nonvar(C3),
nonvar(C4),
compatible([d,p,2,−,p,−,−],C3),
compatible([space],C4),
% implies that it is not eclipsed and that it should have been
compatible([eclipsable],C5).

% adjective following feminine noun must be lenited
% if adjective is lenitable
check(_,C3,C4,C5,ungrammatical101) :-
    nonvar(C3),
nonvar(C4),
compatible([n,c,f,−,n,−,−],C3),
compatible([space],C4),
compatible([a,−,−,−,−],C5),
% implies that it is not lenited and that it should have been
compatible([lenitable],C5).

% if adjective following 'chomh' (=as) begins with a vowel
% then it should be prefixed with a 'h'
check(_,C3,C4,C5,ungrammatical103) :-
    nonvar(C3),
nonvar(C4),
compatible([c,h,o,m,h],C3),
compatible([space],C4),
% implies that there is no prefixed 'h' and that there should have been
compatible([vowel],C5).

% if adjective following 'chomh' (=as) begins with a consonant
% then it should be unmutated
check(_,__,C3,C4,C5,ungrammatical04) :-
nonvar(C3),
nonvar(C4),
compatible([c,h,o,m,h],C3),
compatible([space],C4),
compatible([l],C5).

check(_,__,C3,C4,C5,ungrammatical05) :-
nonvar(C3),
nonvar(C4),
compatible([c,h,o,m,h],C3),
compatible([space],C4),
compatible([e],C5).

% prepositions followed by article
% preposition + singular article = eclipses of noun
% if noun is eclipsable
check(C1,C2,C3,C4,C5,ungrammatical02) :-
nonvar(C1),
nonvar(C2),
nonvar(C3),
nonvar(C4),
compatible([s,p,s,n,n,0],C1),
compatible([space],C2),
compatible([t,d,_,s,_,],C3),
compatible([space],C4),
% implies that it is not eclipsed and that it should have been
compatible([eclipsable],C5),
compatible([n,_,_,_,_,_,_],C5).

% simple preposition 'Ã£o' + definite article = 'Ã£o'
% causes eclipses following noun
check(_,__,C3,C4,C5,ungrammatical02) :-
nonvar(C3),
nonvar(C4),
compatible([f(o),n],C3),
compatible([space],C4),
compatible([n,_,_,_,_,_,_],C5),
% implies that it is not eclipsed and that it should have been
compatible([eclipsed],C5).

% simple preposition 'de' + definite article = 'den'
% causes lenition following noun
check(_,__,C3,C4,C5,ungrammatical01) :-
nonvar(C3),
nonvar(C4),
compatible([d,e,n],C3),
compatible([space],C4),
compatible([n,_,_,_,_,_,_],C5),
%% implies that it is not lenited and that it should have been
compatible([lenitable],C5).

%% simple preposition 'do' + definite article = 'don'
%% causes lenition following noun
check(_,C3,C4,C5,ungrammatical101) :-
    nonvar(C3),
    nonvar(C4),
    compatible([d,o,n],C3),
    compatible([space],C4),
    compatible([n,_,_,_],C5),
    %% implies that it is not lenited and that it should have been
    compatible([lenitable],C5).

%% simple preposition 'faoi' + definite article = 'faoin'
%% causes eclipses following noun
check(_,C3,C4,C5,ungrammatical102) :-
    nonvar(C3),
    nonvar(C4),
    compatible([f,a,o,i,n],C3),
    compatible([space],C4),
    compatible([n,_,_,_],C5),
    %% implies that it is not eclipsed and that it should have been
    compatible([eclipsable],C5).

%% missing contractions, i.e. prepositions that end
%% in a vowel should be joined with the article 'an'

%% simple preposition 'i6' followed definite article 'an'
%% should be contracted to 'i6n'
check(_,C3,C4,C5,ungrammatical106) :-
    nonvar(C3),
    nonvar(C4),
    compatible([f(o)],C3),
    compatible([space],C4),
    compatible([t,d,_,s,\],C5).

%% simple preposition 'de' followed definite article 'an'
%% should be contracted to 'den'
check(_,C3,C4,C5,ungrammatical106) :-
    nonvar(C3),
    nonvar(C4),
    compatible([d,e],C3),
    compatible([space],C4),
    compatible([t,d,_,s,\],C5).

%% simple preposition 'do' followed definite article 'an'
%% should be contracted to 'don'
check(_,C3,C4,C5,ungrammatical106) :-
    nonvar(C3),
nonvar(C4),
compatible([d,o],C3),
compatible([space],C4),
compatible([t,d,_s,],C5).

% simple preposition 'faoi' followed definite article 'an'
% should be contracted to 'faoin'
check(_,_,C3,C4,C5,ungrammatical06) :-
  nonvar(C3),
  nonvar(C4),
  compatible([f,a,o,i],C3),
  compatible([space],C4),
  compatible([t,d,_s,],C5).

% preverbal particles that mutate following word

% ní: negative particle, present tense
check(_,_,C3,C4,C5,ungrammatical101) :-
  nonvar(C3),
  nonvar(C4),
  compatible([u,n],C3),
  compatible([space],C4),
  compatible([v,_,n,_,_,_,],C5),
% implies that it is not lenited and that it should have been
  compatible([lenitable],C5).

% an: interogative particle, present tense
check(_,_,C3,C4,C5,ungrammatical102) :-
  nonvar(C3),
  nonvar(C4),
  compatible([a,n],C3),
  compatible([space],C4),
  compatible([v,_,_,n,_,_,],C5),
% implies that it is not eclipsed and that it should have been
  compatible([eclipsable],C5).

% nach: negative interogative particle
check(_,_,C3,C4,C5,ungrammatical102) :-
  nonvar(C3),
  nonvar(C4),
  compatible([n,a,c,h],C3),
  compatible([space],C4),
  compatible([v,_,_,n,_,_,],C5),
% implies that it is not eclipsed and that it should have been
  compatible([eclipsable],C5).

% cíé/cad + a: interrogative determiner + relative particle
check(C1,C2,C3,C4,C5,ungrammatical101) :-
  nonvar(C1),
  nonvar(C2),
nonvar(C3),
nonvar(C4),
compatible([d,q,-,-,-,-,-],C1),
compatible([space],C2),
compatible([u,r],C3),
compatible([space],C4),
compatible([v,-,-,-,-,-],C5),
% implies that it is not lenited and that it should have been
compatible([lenitable],C5).

% cathain + a: interrogative adverb + relative particle
check(C1,C2,C3,C4,C5,ungrammatical101) :-
  nonvar(C1),
  nonvar(C2),
  nonvar(C3),
  nonvar(C4),
  compatible([r,q,-,-,-],C1),
  compatible([space],C2),
  compatible([u,r],C3),
  compatible([space],C4),
  compatible([v,-,-,-,-,-],C5),
  % implies that it is not lenited and that it should have been
  compatible([lenitable],C5).

% nífor: negative particle, past tense
check(_,_,C3,C4,C5,ungrammatical101) :-
  nonvar(C3),
  nonvar(C4),
  compatible([n,f(i),o,r],C3),
  compatible([space],C4),
  compatible([v,-,-,-,-,-],C5),
  % implies that it is not lenited and that it should have been
  compatible([lenitable],C5).

% ar: interrogative particle, past tense
check(_,_,C3,C4,C5,ungrammatical101) :-
  nonvar(C3),
  nonvar(C4),
  compatible([a,r],C3),
  compatible([space],C4),
  compatible([v,-,-,-,-,-],C5),
  % implies that it is not lenited and that it should have been
  compatible([lenitable],C5).

% níár: negative interrogative particle, past tense
check(_,_,C3,C4,C5,ungrammatical101) :-
  nonvar(C3),
  nonvar(C4),
  compatible([n,f(a),r],C3),
  compatible([space],C4),
compatible([v,__,s,__,__,__],C5),
% implies that it is not lenited and that it should have been
compatible([lenitable],C5).

% clár: interrogative determiner, past tense
check(__,C3,C4,C5,ungrammatical101) :-
  nonvar(C3),
  nonvar(C4),
  compatible([c,f(a),r],C3),
  compatible([space],C4),
  compatible([v,__,s,__,__,__],C5),
  % implies that it is not lenited and that it should have been
  compatible([lenitable],C5).

% simple prepositions that lenite following word

% simple preposition 'né'
% causes lenition of following noun
check(__,C3,C4,C5,ungrammatical102) :-
  nonvar(C3),
  nonvar(C4),
  compatible([f(o)],C3),
  compatible([space],C4),
  compatible([n,__,__,__,__,__],C5),
  % implies that it is not lenited and that it should have been
  compatible([lenitable],C5).

% simple preposition 'de'
% causes lenition of following noun
check(__,C3,C4,C5,ungrammatical102) :-
  nonvar(C3),
  nonvar(C4),
  compatible([d,e],C3),
  compatible([space],C4),
  compatible([n,__,__,__,__,__],C5),
  % implies that it is not lenited and that it should have been
  compatible([lenitable],C5).

% simple preposition 'do'
% causes lenition of following noun
check(__,C3,C4,C5,ungrammatical102) :-
  nonvar(C3),
  nonvar(C4),
  compatible([d,o],C3),
  compatible([space],C4),
  compatible([n,__,__,__,__,__],C5),
  % implies that it is not lenited and that it should have been
  compatible([lenitable],C5).

% simple preposition 'tríf'
APPENDIX A. PROGRAM CODE

% causes lenition of following noun
check(_,C3,C4,C5,ungrammatical02) :-
    nonvar(C3),
    nonvar(C4),
    compatible([t,r,f(i)],C3),
    compatible([space],C4),
    compatible([n,_,_,'n'-'n'],C5),
    % implies that it is not lenited and that it should have been
    compatible([lenitable],C5).

% simple preposition 'roimh'
% causes lenition of following noun
check(_,C3,C4,C5,ungrammatical02) :-
    nonvar(C3),
    nonvar(C4),
    compatible([r,o,i,m,h],C3),
    compatible([space],C4),
    compatible([n,_,_,'n'-'n'],C5),
    % implies that it is not lenited and that it should have been
    compatible([lenitable],C5).

% simple preposition 'mar'
% causes lenition of following noun
check(_,C3,C4,C5,ungrammatical02) :-
    nonvar(C3),
    nonvar(C4),
    compatible([m,a,r],C3),
    compatible([space],C4),
    compatible([n,_,_,'n'-'n'],C5),
    % implies that it is not lenited and that it should have been
    compatible([lenitable],C5).

% simple preposition 'um'
% causes lenition of following noun
check(_,C3,C4,C5,ungrammatical02) :-
    nonvar(C3),
    nonvar(C4),
    compatible([u,m],C3),
    compatible([space],C4),
    compatible([n,_,_,'n'-'n'],C5),
    % implies that it is not lenited and that it should have been
    compatible([lenitable],C5).

% simple preposition 'faoi'
% causes lenition of following noun
check(_,C3,C4,C5,ungrammatical02) :-
    nonvar(C3),
    nonvar(C4),
    compatible([f,a,o,i],C3),
    compatible([space],C4),
compatible([n,_,_,_,_],C5),
% implies that it is not lenited and that it should have been
cOMPATIBLE([lenitable],C5).

% simple preposition 'i'
% causes lenition of following noun
check(_,_,C3,C4,C5,ungrammatical02) :-
    nonvar(C3),
    nonvar(C4),
    compatible([f,a,o,i],C3),
    compatible([space],C4),
    compatible([n,_,_,_,_],C5),
% implies that it is not eclipsed and that it should have been
    compatible([eclipsable],C5).

% if not ungrammatical according to above cases,
% then taken to be grammatical
check(CategoryInfo1,CategoryInfo2,CategoryInfo3,CategoryInfo4,CategoryInfo5,grammatical1)
A.3  tagger.pl

% Generates a set of grammatical tags for Irish words that have been %
% passed to tagger.pl as prolog terms of the form 'word([a,b,c]).'
%
% The tags are generated based on the prefix and the suffix of the %
% word being tagged using a series of prefix and suffix matching %
% rules. For some words, where necessary, the word itself is given %
% as a tag. Prefix rules tag words with information about the %
% beginning of a word. The prefix of a word is either eclipsed, %
% lenited, a vowel, lenitable, or eclipsable. Suffix rules tag %
% words for grammatical category.

:- ensure_loaded([utilities]).

% usage: prefix(Prefix,Word).
% succeeds if Prefix is a prefix of Word

prefix([],[]).
prefix([H|Prefix],[H|T]) :-
  prefix(Prefix,T).

% usage: suffix(Word,Suffix).
% succeeds if Suffix is a suffix of Word

suffix(T,T).
suffix([H|T],Suffix) :-
  suffix(T,Suffix).

% tagit: case 1
% case 1 tags words that appear lexically in the suffix rules

tagit(word(Word),[InitialMutation|Cinfo]) :-
  category(InitialMutation,prefix(Prefix)),
  setof(CategoryInfo,category(CategoryInfo,lexical(Word)),Cinfo),!.

% tagit: case 2
% case 2 tags all other words if matches are found
% with both prefix and suffix rules

tagit(word(Word),[InitialMutation|Cinfo]) :-
APPENDIX A. PROGRAM CODE

category(InitialMutation, prefix(Prefix)),
prefix(Prefix,Word),
setof(CategoryInfo, category(CategoryInfo, [Head|Suffix]), Cinfo),
suffix(Word, [Head|Suffix]), !.

% case 3 should only occur when a word begins
% with a letter that is not in the Irish alphabet
% áábcdeéfghiilmnoóprstuú
% ÁÁBCDEÉFGHIÍLMNOÓPRSTUÚ

% tagit: case 3
% case 3 tags words that match only prefix rules
tagit(word(Word), [InitialMutation]) :-
category(InitialMutation, prefix(Prefix)),
prefix(Prefix, Word), !.

% tagit: case 4
% case 4 tags words that match only suffix rules
tagit(word(Word), [Cinfo]) :-
setof(CategoryInfo, category(CategoryInfo, [Head|Suffix]), Cinfo),
suffix(Word, [Head|Suffix]), !.

% tagit: case 5
% case 5 is a base case, and tags words that match
% neither prefix or suffix rules with the empty list
% and puts that into the list of tags
ntagit(word(Word), []).% tagit: case 6
% case 6 tags punctuation as the list containing just
% one element and puts that into the list of tags
tagit(punct([space]), [[space]]) :- !.
tagit(punct([newline]), [[space]]) :- !.
tagit(punct([return]), [[space]]) :- !.
tagit(punct([tab]), [[space]]) :- !.
tagit(punct(Punctuation), [[punctuation]]).
tagit(digit(Digit), [[digit]]).

% prefix tagging: eclipses [e]
category([e], prefix([m,b])). % eclipsed 'b'
category([e], prefix([m,c(b)])). % eclipsed uppercase 'b'
category([e], prefix([g,c])). % eclipsed 'c'
category([e], prefix([g,c(c)])). % eclipsed uppercase 'c'
APPENDIX A. PROGRAM CODE

category([e], prefix([n,d])). % eclipsed 'd'
category([e], prefix([n,c(d)])). % eclipsed uppercase 'd'
category([e], prefix([b,h,f])). % eclipsed 'f'
category([e], prefix([b,h,c(f)])). % eclipsed uppercase 'f'
category([e], prefix([n,g])). % eclipsed 'g'
category([e], prefix([n,c(g)])). % eclipsed uppercase 'g'
category([e], prefix([b,c(p)])). % eclipsed 'p'
category([e], prefix([d,t])]. % eclipsed 't'
category([e], prefix([d,c(t)])]. % eclipsed uppercase 't'

category([e], prefix([n,-,a])). % eclipsed 'a'
category([e], prefix([n,c(a)])]. % eclipsed uppercase 'a'
category([e], prefix([n,-,e])). % eclipsed 'e'
category([e], prefix([n,c(e)]). % eclipsed uppercase 'e'
category([e], prefix([n,-,i])). % eclipsed 'i'
category([e], prefix([n,c(i)]]. % eclipsed uppercase 'i'
category([e], prefix([n,-,o])]. % eclipsed 'o'
category([e], prefix([n,c(o)]). % eclipsed uppercase 'o'
category([e], prefix([n,-,u])]. % eclipsed 'u'
category([e], prefix([n,c(u)]]. % eclipsed uppercase 'u'
category([e], prefix([n,-,f(a)])]. % eclipsed 'á'
category([e], prefix([n,cf(a)])]. % eclipsed uppercase 'á'
category([e], prefix([n,-,f(e)]). % eclipsed 'é'
category([e], prefix([n,cf(e)])]. % eclipsed uppercase 'é'
category([e], prefix([n,-,f(i)]). % eclipsed 'í'
category([e], prefix([n,cf(i)]). % eclipsed uppercase 'í'
category([e], prefix([n,-,f(o)]). % eclipsed 'ó'
category([e], prefix([n,cf(o)]). % eclipsed uppercase 'ó'
category([e], prefix([n,-,f(ü)])]. % eclipsed 'ü'
category([e], prefix([n,cf(ü)]). % eclipsed uppercase 'ü'

% prefix tagging: lenition [1]
category([1], prefix([b,h])).
category([1], prefix([c(b),h])].
category([1], prefix([c,h]]).
category([1], prefix([c(c),h])].
category([1], prefix([d,h]]).
category([1], prefix([d,c,h])].
category([1], prefix([f,h])].
category([1], prefix([c(f),h)])
category([1], prefix([g,h])].
category([1], prefix([c(g),h])].
category([1], prefix([m,h])].
category([1], prefix([c(m),h])].
category([1], prefix([p,h])].
category([1], prefix([c(p),h])].
category([1], prefix([s,h])].
category([1], prefix([c(s),h])].
category([1], prefix([t,h)])]
category([1],prefix([c(t),h])).

category([lenitable],prefix(X)) :-
    lenitable(X).

category([eclipsable],prefix(X)) :-
    eclipsable(X).

% more prefix tagging:
% word with initial vowel is tagged [vowel]
% vowels are also eclipsable

category([vowel],prefix(X)) :-
    vowel(X).

% vowel(X): X are vowels, accutely accented or unaccented,
% lowercase or uppercase.

vowel(a).
vowel(e).
vowel(i).
vowel(o).
vowel(u).
vowel(f(a)).
vowel(f(e)).
vowel(f(i)).
vowel(f(o)).
vowel(f(u)).
vowel(c(a)).
vowel(c(e)).
vowel(c(i)).
vowel(c(o)).
vowel(c(u)).
vowel(cf(a)).
vowel(cf(e)).
vowel(cf(i)).
vowel(cf(o)).
vowel(cf(u)).

% eclipsable(X): X are the letters, both lowercase and uppercase,
% that can be eclipsed when they occur at the beginning of a word

eclipsable(X) :-
    vowel(X).
eclipsable(X) :-
    prefix([b],X).
eclipsable(X) :-
    prefix([c(b)],X).
eclipsable(X) :-
    prefix([c],X).
eclipsable(X) :-

prefix([c(c)],X).
eclipsable(X) :-
    prefix([d],X).
eclipsable(X) :-
    prefix([c(d)],X).
eclipsable(X) :-
    prefix([f],X).
eclipsable(X) :-
    prefix([c(f)],X).
eclipsable(X) :-
    prefix([g],X).
eclipsable(X) :-
    prefix([c(g)],X).
eclipsable(X) :-
    prefix([p],X).
eclipsable(X) :-
    prefix([c(p)],X).
eclipsable(X) :-
    prefix([t],X).
eclipsable(X) :-
    prefix([c(t)],X).

% lenitable(X): X are the letters, both lowercase and uppercase,
% that can be lenited but not eclipsed when they occur at the
% beginning of a word

lenitable(X) :-
    prefix([b],X).
lenitable(X) :-
    prefix([c(b)],X).
lenitable(X) :-
    prefix([c],X).
lenitable(X) :-
    prefix([c(c)],X).
lenitable(X) :-
    prefix([d],X).
lenitable(X) :-
    prefix([c(d)],X).
lenitable(X) :-
    prefix([f],X).
lenitable(X) :-
    prefix([c(f)],X).
lenitable(X) :-
    prefix([g],X).
lenitable(X) :-
    prefix([c(g)],X).
lenitable(X) :-
    prefix([t],X).
lenitable(X) :-
    prefix([c(t)],X).
APPENDIX A. PROGRAM CODE

lenitable(X) :-
    prefix([m], X).
lenitable(X) :-
    prefix([c(m)], X).
lenitable(X) :-
    prefix([p], X).
lenitable(X) :-
    prefix([c(p)], X).
lenitable(X) :-
    prefix([s], X).
lenitable(X) :-
    prefix([c(s)], X).
lenitable(X) :-
    prefix([t], X).
lenitable(X) :-
    prefix([c(t)], X).

% Suffix Rules
% category(CategoryInfo, Suffix).
% CategoryInfo: morphosyntactical description
% Tagset used: The Parole Common Morphosyntactical Tagset
% Variations from aforementioned tagset:-
% Uppercase tags given in lowercase

% category(CategoryInfo, [a]).

category([u,r], lexical([a])). % relative particle
category([u,r], lexical([c(a)])).
category([u,v], lexical([a])). % vocative particle
category([u,v], lexical([c(a)])).
category([d,p,3,_,_], lexical([a])). % possessive determiner(underspecified number)
category([d,p,3,_,_], lexical([c(a)])).
category([v,1,1,1,_,1,a], lexical([b,a])).
category([v,1,1,1,_,1,a], lexical([c(b), a])).
category([n,c,f,p,n,_,y], lexical([b,a])). % =cows
category([n,c,f,p,n,_,y], lexical([c(b), a])).
category([n,c,f,s,n,_,n], lexical([1, e,a,b,a])). % =bed
category([n,c,f,s,n,_,n], lexical([c(l), e,a,b,a])).
category([n,c,m,s,_,_,n], [b,a]). % roba=robe
category([n,c,-p,n,_,y], [b,a]). % cruba=claws/hooves; luba=loops
category([a,q,p,_,_,_,], lexical([f(e), a,s, c,a])).
category([n,c,-p,n,_,y], [c,a]). % muca=pigs
category([n,c,m,s,_,_,n], [c,a]). % peaca=sin
category([n,c,m,s,_,_,y], [c,a]).
category([n,c,m,s,_,_,n], [d,a]). % planda=plant
category([n,c,-p,n,_,y], [d,a]). % teada=strings
category([n,c,m,s,_,_,y], [d,a]). % ruda=thing-GEN
category([n,c,f,s,_,_,y], [d,a]). % troda=fight(ing)-GEN
category([a,q,p,_,_,_,], lexical([f, a,d,a])). % =long
APPENDIX A. PROGRAM CODE

92

category([a,q,p,-,-,-],[c(f),a,d,a])
category([a,q,p,-,-,-],[d,a])
category([n,c,m,s,-,-,n],[f,a])  \% anfa=tempest
category([n,c,f,s,-,-,n],[f,a])  \% céadfa=sense
category([n,c,-,-,p,n,-,-,n],[f,a])  \% buaf=toads
category([v,m,p,s,-,-,-],[f,a])  \% verb past participle; eg. scriofa=written
category([a,q,p,-,-,-],[f,a])  \% naofa=holy
category([a,q,f,-,-,-],[b,o,g,a])  \% soft-PL
category([a,q,p,-,-,-],[c(b),o,g,a])
category([i],[e],[i,e,o,g,a])  \% indeed
category([i],[e],[i,e,o,g,a])
category([n,c,f,p,n,-,-,y],[e,o,g,a])
category([a,q,p,-,-,-],[f(o),g,a])  \% young-PL
category([a,q,f,-,-,-],[c(f),o,g,a])
category([n,c,f,p,n,-,-,y],[f(o),g,a])
category([n,c,f,s,-,-,n],[c(1),o,r,g,a])  \% =staff/Shaft/shin

category([a,q,p,-,-,-],[r,g,a])  \% òrga=golden
category([a,q,p,-,-,-],[r,g,a])  \% dearga=red-PL
category([n,c,m,s,-,-,n],[u,g,a])  \% mug=mug

category([a,q,f,-,-,-],[b,h,a])  \% dubha=black-PL
category([n,c,m,s,-,-,n],[b,h,a])  \% gabha=smith

category([n,c,f,s,-,-,n],[b,h,a])  \% ealbha=flock/herd
category([n,c,-,-,p,n,-,-,y],[b,h,a])  \% griobha=griﬃons/mythological bird

category([a,q,f,-,-,-],[c(1),a,c,h,a])  \% duck

category([n,c,-,-,p,n,-,-,y],[c,h,a])  \% strong plurals; eg. paidreacha=prayers

category([n,c,-,-,p,n,-,-,y],[c,h,a])  \% weak plural; eg. beacha

category([a,q,p,-,-,-],[d,o,r,c,h,a])  \% dark

category([a,q,p,-,-,-],[c,d,o,r,c,h,a])  \% salacha=dirty-PL

category([n,c,-,-,s,g,-,-,y],[d,h,a])
category([n,c,m,s,-,-,n],[g,h,a])  \% rogha=choice
category([n,c,m,s,n,-,-,n],[m,h,a])  \% cumha=sadness

category([n,c,-,-,p,n,-,-,y],[m,h,a])  \% lamha=hands

category([a,q,p,-,-,-],[m,h,a])  \% fomh=mighty-PL
category([n,c,f,s,-,-,n],[b,e,a,t,h,a])  \% =life

category([a,q,f,-,-,-],[c(b),e,a,t,h,a])
category([n,c,m,s,g,-,-,y],[t,h,a])  \% ceatha=shower-GEN

category([v,m,p,-,-,-],[t,h,a])  \% scart=separated

category([a,q,c,-,-,-],[s,i,a])  \% farther

category([a,q,c,-,-,-],[c(s),i,a])  \% bia=food

category([n,c,m,s,-,-,n],[a,l,a])
category([n,c,f,s,g,-,-,y],[f(a),l,a])
category([a,q,p,-,-,-],[f(a),l,a])
category([n,c,m,s,-,-,n],[b,l,a])
category([n,c,m,p,-,-,-],[c,l,a])
category([n,c,f,s,g,-,-,y],[d,l,a])
category([n,c,m,s,-,-,n],[f,l,a])
category([n,c,f,s,g,_,y],[h,l,a]).
category([n,c,m,s,-,_,n],[l,1,a]).
category([n,c,m,p,n,_,n],lexical([f(u),1,1,a])).
category([n,c,m,p,n,_,n],lexical([c(o),1,1,a])).
category([n,c,f,s,n,_,n],[n,l,1,a]).
category([n,c,f,s,n,_,n],lexical([o,l,1,a])).
category([n,c,f,s,n,_,n],lexical([c(o),1,1,a])).
category([n,c,m,p,-,_,n],[o,l,1,a]).
category([n,c,f,s,g,_,y],[o,l,1,a]).
category([a,q,p,_,-,gl],[o,l,1,a]).
category([n,c,m,s,-,_,n],[p,l,1,a]).
category([n,c,m,s,-,_,n],[r,l,1,a]).
category([n,c,m,s,-,_,n],[s,l,1,a]).
category([n,c,m,s,-,_,n],[t,l,1,a]).
category([c,c,_,_],lexical([s,u,1,a])).
category([c,c,_,_],lexical([s,u,1,a])).
category([s,p,sn,n,0],lexical([s,u,1,a])).
category([s,p,sn,n,0],lexical([s,u,1,a])).
category([n,c,f,s,_,g,_,y],[f(u),1,a,1,a]).
category([a,q,p,_,-,l],[f(u),1,a,1,a]).
category([n,c,m,s,-,_,n],[m,a]).
category([a,q,p,_,-,l],[m,a]).
category([a,q,c,_,-,l],[m,a]).
category([a,q,p,_,-,l],[m,a]).
category([t,d,_,-,p,_,l],lexical([n,a])). % nominative plural article
category([t,d,_,-,p,_,l],lexical([c(n),a])).
category([t,d,_,f,_,-,m],lexical([n,a])). % feminine genitive article
category([t,d,_,f,_,-,m],lexical([c(n),a])).
category([n,c,f,s,_,g,_,y],lexical([b,1,i,a,n,a])).
category([n,c,f,s,_,g,_,y],lexical([c(b),1,i,a,n,a])).
% abnormal plural form used with numbers
category([n,c,f,p,n,_,y],lexical([b,1,i,a,n,a])).
category([n,c,f,p,n,_,y],lexical([c(b),1,i,a,n,a])).
category([n,c,m,s,n,_,n],[a,n,a]). % mana=motto
category([n,c,f,p,n,_,n],[a,n,a]). % comharsana=neighbours
category([n,c,m,s,g,_,y],[a,n,a]). % ceana=affection-GEN
category([n,c,f,s,g,_,y],[a,n,a]). % féachana
category([a,q,p,_,-,p,_,l],[a,n,a]). % tréana=strong-PL
category([n,c,m,p,n,_,n],[f(a),n,a]).
category([a,q,p,_,-,l],[f(a),n,a]).
category([n,c,f,s,_,-,n],[g,n,a]).
category([n,c,f,s,_,-,n],[h,n,a]).
category([n,c,m,s,n,_,n],[m,n,a]).
category([n,c,f,s,_,n],[m,n,a]).
category([n,c,m,p,_,-,n],[n,n,a]).
category([n,c,f,p,_,-,n],[n,n,a]).
category([n,c,m,p,n,_,y],[o,n,a]).
category([n,c,m,s,g,_,y],[o,n,a]).
category([a,q,_,-,_,-,l],[o,n,a]).
category([n,c,f,s,g,_,y],[f(o),n,a]).
APPENDIX A. PROGRAM CODE

```python
category([n, c, m, s, -,_n], lexical([t, r, f(a), t, h, n, f(o), n, a]))
category([n, c, m, s, -,_n], lexical([c(t), r, f(a), t, h, n, f(o), n, a]))
category([n, c, m, s, -,_y], [a, r, n, a])
category([n, c, f, p, n, -,_y], [a, r, n, a])
category([n, c, f, s, g, -,_y], [a, r, n, a])
category([n, c, m, p, n, -,_y], [o, r, n, a])
category([n, c, f, p, n, -,_y], [o, r, n, a])
category([n, c, m, s, -,_n], [s, n, a])
category([n, c, f, s, -,_n], [s, n, a])
category([c, c, -,_m], lexical([m, u, n, a]))
category([c, c, -,_m], lexical([c(m), u, n, a]))
category([n, c, m, s, -,_n], [f(u), n, a])
category([n, c, m, s, g, -,_y], [f(u), n, a])
category([n, c, f, s, g, -,_y], [f(u), n, a])
category([a, q, p, -,_g], [f(u), n, a])
category([n, c, m, s, -,_n], [a, p, a])
category([n, c, m, s, g, -,_y], [a, p, a])
category([n, c, m, s, -,_n], [p, f(a), p, a])
category([n, c, m, s, -,_n], [1, p, a])
category([n, c, m, s, -,_n], [m, p, a])
category([n, c, m, s, -,_n], [o, p, a])
category([n, c, m, s, -,_n], [f(o), p, a])
category([n, c, m, s, -,_n], [r, p, a])
category([n, c, m, s, -,_n], [s, p, a])
category([n, c, m, s, -,_n], [u, p, a])
category([n, c, m, s, n, -,_n], lexical([c, a, r, a]))
category([n, c, m, s, n, -,_n], lexical([c(c), a, r, a]))
category([n, c, m, s, n, -,_n], lexical([d, a, r, a]))
category([n, c, m, s, n, -,_n], lexical([c(d), a, r, a]))
category([a, q, p, -,_p], [e, a, r, a])
category([n, c, m, s, g, -,_y], [f(e), a, r, a])
category([a, q, p, -,_g], [f(e), a, r, a])
category([a, q, p, -,_g], [f(e), a, r, a])
category([a, q, c, -,_g], [f(e), a, r, a])
category([a, q, s, -,_g], [f(e), a, r, a])
category([a, q, p, -,_g], [m, h, a, r, a])
category([a, q, p, -,_g], [m, h, a, r, a])
category([n, c, m, s, -,_n], [c, r, a])
category([a, q, p, -,_p], [c, r, a])
category([a, q, c, -,_g], [c, r, a])
category([a, q, s, -,_g], [c, r, a])
category([n, c, m, s, -,_n], [d, r, a])
category([n, c, m, s, g, -,_y], [e, r, a])
category([a, q, p, -,_g], [e, r, a])
category([n, c, m, s, -,_n], [f, r, a])
category([n, c, m, s, -,_n], [g, r, a])
category([n, c, f, s, -,_n], [b, h, r, a])
category([n, c, m, s, -,_n], [c, h, r, a])
category([n, c, f, p, n, -,_y], [d, h, r, a])
category([n, c, m, -,_n], [g, h, r, a])
category([n, c, m, -,_n], [g, h, r, a])
```
APPENDIX A. PROGRAM CODE

```prolog
category([a,q,p,\-,p,\-,\-[m,h,r,a]]).
category([a,q,c,\-,\-,\-,\-[m,h,r,a]]).
category([a,q,s,\-,\-,\-,\-[m,h,r,a]]).
category([n,c,m,p,\-,\-,\-,\-[t,h,r,a]]).
category([n,c,m,s,\-,\-,\-,\-[l,r,a]]).
category([n,c,m,\--,\-,\-,\-[m,r,a]]).
category([n,c,m,\-,\-,\-,\-[n,r,a]]).
category([n,c,f,s,n,\-,\-,\-[c,a,o,r,a]]).
category([n,c,f,s,n,\-,\-,\-[c(c),a,o,r,a]]).
category([n,c,m,s,\-,\-,\-[e,o,r,a]]).
category([a,q,p,\-,\-,g],[e,o,r,a]).
category([n,c,m,s,\-,\-,\-[f(o),r,a]]).
category([a,q,p,\-,\-,g],[f(o),r,a])
category([n,c,m,\-,\-,\-,\-[r,r,a]]).
category([n,c,f,s,\-,\-,\-,\-[r,r,a]]).
category([n,c,m,\-,\-,\-,\-[n],[s,r,a]]).
category([n,c,f,s,n,\-,\-,\-[n],[t,r,a]]).
category([n,c,f,s,\-,\-,\-,\-[n],[t,r,a]]).
category([],lexical([g,u,r,a])). % is
category([],lexical([c(g),u,r,a])).
category([c,c,\-,\-,\-,\-,\-[m,u,r,a]]).
category([c,c,\-,\-,\-,\-[c(m),u,r,a]]).
category([n,c,m,s,\-,\-,\-,\-[f(u),r,a]]).
category([a,q,p,\-,\-,g],[f(u),r,a]).
% simple preposition 'i' +
% definite article occurring before a consonant
category([s,a],lexical([s,a])).
category([s,a],lexical([c(s),a])).
category([n,c,m,s,\-,\-,\-,\-[a,s,a]]).
category([n,c,f,s,\-,\-,\-,\-[a,s,a]]).
category([n,c,m,\-,\-,\-,\-[f(a),s,a]]).
category([n,c,m,\-,\-,\-,\-[c(s),a]]).
category([n,c,m,\-,\-,\-,\-[b,h,s,a]]).
category([n,c,m,\-,\-,\-,\-[m,m,h,s,a]]).
category([n,c,m,\-,\-,\-,\-[l,s,a]]).
category([a,q,p,\-,\-,\-,\-[l,s,a]]).
category([n,c,m,\-,\-,\-,\-[n],[m,s,a]]).
category([n,c,m,\-,\-,\-,\-[n],[n,s,a]]).
category([n,c,m,\-,\-,\-,\-[n],[o,s,a]]).
category([n,c,m,p,n,\-,\-,\-[f(o),s,a]]).
category([n,c,m,\-,\-,\-,\-[n],[a,t,a]]).
category([a,q,p,\-,\-,\-,\-[a,t,a]]).
category([n,c,m,\-,\-,\-,\-[f(a),t,a]]).
category([a,q,p,\-,\-,\-,\-[d,t,a]]).
category([n,c,m,\-,\-,\-,\-[f,t,a]]).
category([a,q,p,\-,\-,\-,\-[f,t,a]]).
category([n,c,m,\-,\-,\-,\-[a,b,h,t,a]]).
category([n,c,m,s,\-,\-,\-,\-[a,c,h,t,a]]).
category([n,c,f,s,\-,\-,\-,\-[a,c,h,t,a]]).
category([a,q,p,\-,\-,g],[a,c,h,t,a]).
```
category([n,c,m,s,g,_,y],[f(a),c,h,t,a]).
category([n,c,m,s,g,_,y],[o,c,h,t,a]).
category([n,c,f,s,g,_,y],[o,c,h,t,a]).
category([a,q,p,,-,-,g],[o,c,h,t,a]).
category([a,q,p,,-,-,],[o,c,h,t,a]).
category([a,q,p,,-,-,],[f(o),c,h,t,a]).
category([a,q,p,,-,-,],[g,h,t,a]).
category([a,q,p,,-,-,],[a,l,t,a]).
category([a,q,p,,-,-,],[f(a),l,t,a]).
category([a,q,p,,-,-,],[l,l,t,a]).
category([a,q,p,,-,-,],[o,l,t,a]).
category([a,q,p,,-,-,],[u,l,t,a]).
category([a,q,p,,-,-,],[a,n,t,a]).
category([a,q,p,,-,-,],[f(a),n,t,a]).
category([a,q,p,,-,-,],[f(a),n,t,a]).
category([n,c,m,p,,-,-,y],[n,n,t,a]).
category([n,c,m,p,,-,-,y],[o,n,t,a]).
category([n,c,m,p,,-,-,y],[f(o),n,t,a]).
category([a,q,p,,-,-,],[f(o),n,t,a]).
category([n,c,m,p,n,,-,y],[u,n,t,a]).
category([n,c,m,p,,-,-,y],[f(u),n,t,a]).
category([a,q,p,,-,-,],[f(u),n,t,a]).
category([n,c,m,s,,-,-,n],[o,t,a]).
category([n,c,m,s,,-,-,n],[f(o),t,a]).
category([n,c,m,s,,-,-,n],[r,t,a]).
category([n,c,m,s,,-,-,n],[r,t,a]).
category([a,q,p,,-,-,],[s,t,a]).
category([n,c,m,s,,-,-,n],[u,t,a]).
category([n,c,m,s,,-,-,n],[f(u),t,a]).
category([a,q,p,,-,-,],[u,a]).

% category(CategoryInfo, [a]).
category([d,p,3,,-,-,_,],lexical([f(a)])).
category([d,p,3,,-,-,_,],lexical([cf(a)])).
category([i,i],lexical([f(a)])).
category([i,i],lexical([cf(a)])).
category([n,c,m,s,n,_,n],lexical([b,f(a)])).
category([n,c,m,s,n,_,n],lexical([c(b),f(a)])).
category([n,c,f,s,n,_,n],lexical([b,f(a)])).
category([n,c,f,s,n,_,n],lexical([c(b),f(a)])).
category([d,i,,-,-,,-,-,c],lexical([c,f(a)])).
category([d,i,,-,-,,-,-,c],lexical([c,c,f(a)])).
category([a,q,p,,-,-,],lexical([c,f(a)])).
category([a,q,p,,-,-,],lexical([c,c,f(a)])).
category([r,q,,-,-,],lexical([c,c,f(a)])).
category([r,q,,-,-,],lexical([c,c,f(a)])).
category([], lexical([t, f(a)])).
category([], lexical([c(t), f(a)])).
category([], lexical([a, t, f(a)])).
category([], lexical([c(a), t, f(a)])).
category([n, c, m, s, n, _, n], [f(a)]).  % m/m4
ncategory([n, c, m, s, g, _, n], [f(a)]).
category([n, c, f, s, g, _, n], [f(a)]).  % f/f4
category([n, c, f, s, g, _, n], [f(a)]).
category([v, m, c, _, 2, s, _, _], [f(a)]).  % should be more specific -f(e)á
% should be more specific -t(e)á/-t(e)áiteá
category([v, m, i, h, 2, s, _, _], [f(a)]).

% category(CategoryInfo, [b]).
category([], lexical([a, b])).  % copula
category([], lexical([c(a), b)]).
category([n, c, m, s, n, _, n], lexical([a, b])).
category([n, c, m, s, n, _, n], lexical([c(a), b)]).
category([c, c, _, _], lexical([m, u, n, a, b])).
category([c, c, _, _], lexical([c(m), u, n, a, b)]).
category([], lexical([g, u, r, a, b])).  % copula
category([], lexical([c(g), u, r, a, b)]).
category([c, c, _, _], lexical([m, u, r, a, b])).
category([c, c, _, _], lexical([c(m), u, r, a, b)]).
category([], lexical([a, r, b])).
category([], lexical([d, a, r, b])).
category([], lexical([c(d), a, r, b])).
category([], lexical([c, f(e), r, b])).
category([], lexical([c(c), f(e), r, b)])).
category([], lexical([g, u, r, b])).  % copula
category([], lexical([c(g), u, r, b)]).
category([n, c, m, s, n, _, n], [a, b]).  % m
category([n, c, f, s, n, _, n], [d, h, b])].  % f2
category([n, c, f, s, n, _, n], [i, b]).  % f2
category([n, c, f, s, n, _, n], [f(i), b]).  % f2
category([n, c, m, s, n, _, n], [o, b])].  % m1
category([n, c, m, s, n, _, n], [o, r, b])].  % m1
category([n, c, m, s, g, _, y], [o, i, r, b]).  % m1

% category(CategoryInfo, [c]).
category([n, c, m, s, n, _, n], [a, c])].
category([n, c, f, s, n, _, n], [a, c])].
category([v, m, m, p, 2, s, _, _], [g, l, a, c])].
category([n, c, m, s, g, _, y], [a, i, c])].
category([n, c, f, s, n, _, n], [a, i, c])].
category([n, c, m, s, n, _, n], lexical([c, i, c])].  % m4
category([n, c, m, s, n, _, n], lexical([c(c), i, c)])].
category([n, c, m, s, g, _, n], lexical([c, i, c)])].  % m4
APPENDIX A. PROGRAM CODE

category([n,c,m,s,g,_,n],lexical([c(c), i, c])).
category([n,c,f,s,n,_,n],[e,i,c]).
category([v,m,m,p,2,s,_,n],[f,e,i,c]).
category([n,c,f,s,n,_,n],[f(e), i, c]).
category([n,c,m,s,g,_,n],[f(i), s, c]).
category([v,m,m,p,2,s,_,n],[f(u), s, c]).
category([n,c,f,s,n,_,n],[a,l,s,c]).
category([v,m,m,p,2,s,_,n],[a,l,s,c]).
category([n,c,m,s,g,_,n],[a,l,s,c]).
category([v,m,m,p,2,s,_,n],[f(u), s, c]).
category([n,c,m,s,g,_,n],[f(u), s, c]).
category([v,m,m,p,2,s,_,n],[o,s,c]).
category([n,c,m,s,n,_,n],[i,s,c]).
category([n,c,f,s,n,_,n],[f(u), s, c]).
category([v,m,m,p,2,s,_,n],[f(u), s, c]).
category([n,c,m,s,n,_,n],[e,a,d]).
category([n,c,f,s,n,_,n],[a,e,a,d]).
category([n,c,m,s,n,_,n],[f(a), d]).
category([n,c,m,s,n,_,n],[f(a), d]).
category([n,c,m,s,n,_,n],[f(a), d]).
category([n,c,m,s,n,_,n],[a,g,h,d]).
category([n,c,m,s,n,_,n],[a,g,h,d]).
category([n,c,m,s,n,_,n],[a,g,h,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([n,c,m,s,n,_,n],[a,r,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([a,q,p,_,_],[a,r,d]).
category([a,q,p,_,_],[a,r,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([a,q,p,_,_],[a,r,d]).
category([a,q,p,_,_],[a,r,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).
category([n,c,m,s,n,_,n],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([n,c,m,s,g,_,_],[i,d]).

% should be more specific -f(a) imid/-óimid/-eoidim

category([v,m,i,f,_,_,_],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([a,q,p,_,_],[a,r,d]).
category([a,q,p,_,_],[a,r,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([a,q,p,_,_],[a,r,d]).
category([a,q,p,_,_],[a,r,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([a,q,p,_,_],[a,r,d]).
category([a,q,p,_,_],[a,r,d]).
category([v,m,i,f,_,_,_],[i,d]).
category([v,m,i,p,1,p,_,_],[i,d]).
category([a,q,p,_,_],[a,r,d]).
category([a,q,p,_,_],[a,r,d]).
APPENDIX A. PROGRAM CODE

\[
\text{category}([n,c,m,s,g,\_], [i,r,d]). \ \% \text{ml; e.g. ceird}
\]
\[
\text{category}([n,c,m,s,n,\_], [o,r,d]). \ \% \text{ml}
\]
\[
\text{category}([v,m,m,p,2,s,\_], \text{lexical}([s,c,i,u,r,d])).
\]
\[
\text{category}([v,m,m,p,2,s,\_], \text{lexical}([c(s), c,i,u,r,d])).
\]
\%
\]
\[
\% \text{substantive; ar \sim + genitive}
\]
\[
\text{category}([n,s,\sim, s,\sim, \_], \text{lexical}([f,u,d])).
\]
\[
\text{category}([n,s,\sim, s,\sim, \_], \text{lexical}([c(f), u,d])).
\]
\[
\text{category}([d,d,\sim, \sim, \_], \text{lexical}([f(u), d])). \ \% \text{demonstrative adjective}
\]
\[
\text{category}([d,d,\sim, \sim, \_], \text{lexical}([c(f(u), d)]).
\]
\[
\text{category}([a,q,p,\sim, \sim, \_], \text{lexical}([f(u), d])).
\]
\[
\text{category}([a,q,p,\sim, \sim, \_], \text{lexical}([c(f(u), d)]).
\]
\[
\text{category}([n,c,m,s,n,\_], \text{lexical}([f(u), d])). \ \% \text{try (rugby)}
\]
\[
\text{category}([n,c,m,s,n,\_], \text{lexical}([c(f(u), d)]).
\]
\[
\text{category}([v,m,m,p,2,s,\_], [f(u), d]). \ \% \text{e.g. scrúd}
\]
\]
\[
\text{category}([n,c,m,s,n,\_], \text{lexical}([a,e])).
\]
\[
\text{category}([n,c,m,s,n,\_], \text{lexical}([c(a), e])).
\]
\[
\text{category}([n,c,m,s,\sim, \_], [a,e]).
\]
\[
\text{category}([n,c,m,s,\_], [a,e]).
\]
\[
\text{category}([n,c,f,s,g,\_], [a,e]).
\]
\[
\text{category}([n,c,f,s,g,\_], [b,e]).
\]
\[
\text{category}([s,p,c,m,s,3], \text{lexical}([d,e])).
\]
\[
\text{category}([s,p,c,m,s,3], \text{lexical}([c(d), e])).
\]
\[
\text{category}([d,e], \text{lexical}([d,e])). \ \% \text{simple preposition}
\]
\[
\text{category}([d,e], \text{lexical}([c(d), e])).
\]
\[
\text{category}([s,p,s,n,n,0], \text{lexical}([d,e])). \ \% \text{simple preposition}
\]
\[
\text{category}([s,p,s,n,n,0], \text{lexical}([c(d), e])).
\]
\[
\text{category}([n,c,f,s,g,\_], [d,e]).
\]
\[
\text{category}([n,c,f,s,g,\_], [f,e]).
\]
\[
\text{category}([n,c,f,s,g,\_], [g,e]).
\]
\[
\text{category}([n,c,f,s,g,\_], [b,h,e]).
\]
\[
\text{category}([n,c,m,s,n,\_], \text{lexical}([f,i,c,h,e])).
\]
\[
\text{category}([n,c,m,s,n,\_], \text{lexical}([c(f), i,c,h,e])).
\]
\[
\text{category}([m,c,\_], \text{lexical}([f,i,c,h,e])).
\]
\[
\text{category}([m,c,\_], \text{lexical}([c(f), i,c,h,e])).
\]
\[
\text{category}([n,c,m,s,n,\_], \text{lexical}([c,1,u,1,c,h,e])).
\]
\[
\text{category}([n,c,m,s,n,\_], \text{lexical}([c(c), 1,u,1,c,h,e])).
\]
\[
\text{category}([n,c,f,s,g,\_], [c,h,e]).
\]
\[
\text{category}([a,q,p,\sim, \sim, g], [d,h,e]).
\]
\[
\text{category}([n,c,f,s,g,\_], [g,h,e]).
\]
\[
\text{category}([n,c,f,s,g,\_], [m,h,e]).
\]
\[
\text{category}([a,q,p,\sim, g], [m,h,e]).
\]
\[
\text{category}([n,c,m,s,g,\_], [t,h,e]).
\]
\[
\text{category}([n,c,f,s,g,\_], [t,h,e]).
\]
\[
\text{category}([v,m,m,p,0,\_], [t,h,e]).
\]
\[
\text{category}([1,e], \text{lexical}([1,e])). \ \% \text{simple preposition}
\]
\[
\text{category}([1,e], \text{lexical}([c(1), e])).
\]
\[
\text{category}([s,p,s,n,n,0], \text{lexical}([1,e])). \ \% \text{simple preposition}
\]
APPENDIX A. PROGRAM CODE

```plaintext
category([s,p,s,n,n,0],lexical([c(1),e]))).
category([n,c,f,s,g,_,y],[l,e]).
category([n,c,m,s,_,n],[l,1,e]).
category([n,c,f,s,g,_,y],[l,1,e]).
category([n,c,f,s,g,_,y],[m,e]).
category([n,c,f,s,g,_,y],[n,e]).
category([a,q,c,_,_,n],[n,e]).
category([a,q,s,_,_,n],[n,e]).
category([n,c,f,s,g,_,y],[p,e]).
category([n,c,m,s,_,n],[r,e]).
category([n,c,f,s,g,_,y],[r,e]).
category([n,c,f,s,g,_,y],[s,e]).
category([v,m,p,s,0,_,n],[t,e]). % past participle

% category(CategoryInfo,[ê]).
category([n,c,m,s,n,_,n],[f(e)]). % m4
category([n,c,m,s,_,n],[f(e)]). % m4
category([n,c,f,s,_,n],[f(e)]). % e.g.
category([n,c,f,s,_,n],[f(e)]). % f4
category([n,c,f,s,_,n],[f(e)]). % f4
category([p,p,3,m,s,a,[]],lexical([f(e)])). % pronoun 3rd singular (acc)
category([p,p,3,m,s,a,[]],lexical([cf(e)])).
category([a,q,_,_,-,],lexical([c,i,b,f(e)])).
category([a,q,_,_,-,],lexical([c,c,i,b,f(e)])).
category([c,c,_,[]],lexical([c,f(e)])).
category([c,c,_,[]],lexical([c,c,f(e)])).
category([d,q,_,_,-,],lexical([c,c,f(e)])).
category([d,q,_,_,-,],lexical([c,c,f(e)])).
category([n,c,f,s,n,_,n],lexical([c,c,f(e)])).
category([n,c,f,s,_,n],lexical([c,c,f(e)])).
category([s,p,s,n,n,0],lexical([f,f(e)])).
category([s,p,s,n,n,0],lexical([c,f,f(e)])).
category([s,p,c,m,s,3],lexical([f,f(e)])).
category([s,p,c,m,s,3],lexical([c,f,f(e)])).
category([a,q,_,_,-,],lexical([i,n,n,f(e)])).
category([a,q,_,_,-,],lexical([c,i,n,n,f(e)])).
category([r,g,_,_,-],lexical([i,n,n,f(e)])).
category([r,g,_,_,-],lexical([c,i,n,n,f(e)])).
category([n,s,_,s,n,_,n],lexical([i,n,n,f(e)])).
category([n,s,_,s,n,_,n],lexical([c,i,n,n,f(e)])).
category([p,p,1,_,-,s,[]],lexical([m,f(e)])). % pronoun 1st singular (nom)
category([p,p,1,_,-,s,[]],lexical([c(m),f(e)])).
category([p,p,3,_,s,[]],lexical([s,f(e)])). % pronoun 3rd singular (nom)
category([p,p,3,_,s,[]],lexical([c(s),f(e)])).
category([p,p,3,_,s,[]],lexical([t,f(e)])). % indefinite personal pronoun
```
APPENDIX A. PROGRAM CODE

category([n, c, m, s, n, _, n], [a, f]). % ml; e.g. staf

category([n, c, m, s, n, _, n], [a, f]). % m3; e.g. geaf

category([n, c, f, s, n, _, n], [d, h, f]). % f2; e.g. faidhf

category([n, c, f, s, n, _, n], [i, f]). % f2; e.g. scainf

category([n, c, f, s, n, _, n], [f(i), f]). % f2; e.g. rinlf

category([n, c, m, s, g, _, n], [u, f]). % m4; e.g. triuf

category([n, c, f, s, n, _, n], [i, l, f]). % f4; e.g. seilf

category([n, c, f, s, g, _, n], [i, l, f]). % f4;

% category(CategoryInfo, [g]).

category([u, c], lexical([a, g])).
category([u, c], lexical([c(a), g)]).
category([a, g], lexical([a, g])]. % simple preposition
category([a, g], lexical([c(a), g)]).
category([s, p, s, n, n, 0], lexical([a, g])]. % simple preposition
category([s, p, s, n, n, 0], lexical([c(a), g)]).
category([n, c, m, s, n, _, n], [a, g]). % m1
category([n, c, m, s, g, _, y], [a, i, g]). % m1
category([n, c, m, s, g, _, y], [f(a), i, g]). % m1
category([n, c, f, s, n, _, n], [c, i, g])
category([n, c, f, s, n, _, n], [e, i, g]).
category([n, c, f, s, n, _, n], [f, i, g])
category([n, c, f, s, n, _, n], [l, i, g])
category([n, c, f, s, n, _, n], [m, i, g])
category([v, m, -, s, -, _, i], [n, i, g])
category([n, c, m, s, g, _, y], [o, i, g]).
category([n, c, f, p, n, _, y], [f(o), i, g])
category([c, h, u, i, g], lexical([c, h, u, i, g])]. % simple preposition
category([c, h, u, i, g], lexical([c(c), h, u, i, g)]).
category([s, p, s, n, n, 0], lexical([c(c), h, u, i, g])]. % simple preposition
category([s, p, s, n, n, 0], lexical([c(c), h, u, i, g)]).
category([v, m, s, p, 2, s, _, n], lexical([t, u, i, g])].
category([v, m, s, p, 2, s, _, n], lexical([c(t), u, i, g])].
category([n, c, f, s, n, _, n], [u, i, g])
category([m, c, _, _, _], lexical([c(f(u), i, g)])
category([m, c, _, _, _], lexical([c(c), f(u), i, g)])
category([n, c, m, s, n, _, n], lexical([c(c), f(u), i, g)])
category([n, c, m, s, n, _, n], lexical([c(c), f(u), i, g)])
category([n, c, f, s, n, _, n], [f(u), i, g])
category([n, c, f, s, n, _, n], [f(u), i, g])
category([v, m, m, p, 2, s, _, n], lexical([c(e, a, l, g)])
category([v, m, m, p, 2, s, _, n], lexical([c(c), e, a, l, g)])
category([n, c, f, s, n, _, n], [a, l, g])
category([n, c, m, p, n, _, y], [i, l, g])
category([n, c, m, p, n, _, y], [i, l, g])
category([n, c, m, s, n, _, n], [o, l, g])
category([n, c, m, s, n, _, n], [a, n, g]) % ml gs ~a
category([n, c, f, s, n, _, n], [a, n, g]) % f3 npl ~a
APPENDIX A. PROGRAM CODE

category([v,m,m,p,2,s,-,n],[a,n,g]). % verbal noun
category([v,m,m,p,2,s,-,n],[a,i,n,g]). % e.g. taraing
category([n,c,f,s,n,-,n],[a,i,n,g]).
category([n,c,f,s,n,-,n],[e,o,g]). % f2
category([a,q,p,-,-,],lexical([f(o),g])).
category([n,c,m,s,n,-,n],lexical([f(o),g])).
category([n,c,m,s,n,-,n],lexical([cf(o),g])).
category([n,c,m,s,n,-,n],[f(o),g]). % f2
category([n,c,m,s,n,-,n],[a,r,g]).
category([n,c,f,s,n,-,n],[a,r,g]).
category([a,q,p,-,-,],[a,r,g]).
category([v,m,m,p,2,s,-,n],[a,i,r,g]).
category([v,m,m,p,2,s,-,n],[f(a),i,r,g]).
category([n,c,m,s,g,-,y],[e,i,r,g]).
category([n,c,m,s,n,-,y],[o,i,r,g]).
category([n,c,m,s,g,-,y],[o,i,r,g]).
category([n,c,m,s,n,-,n],[o,r,g]). % m3
category([n,c,f,s,n,-,n],[f(u),g]).
category([v,m,m,p,2,s,-,n],[f(u),g]).

% category(CategoryInfo, [h]).
category([n,c,f,s,n,-,n],[a,b,h]).
category([v,m,m,p,2,s,-,n],[a,b,h]).
category([s,p,c,-,p,2],lexical([c,h,u,g,a,i,b,h])).
category([s,p,c,-,p,2],lexical([c(c),h,u,g,a,i,b,h])).
category([s,p,c,-,p,2],lexical([r,o,m,h,a,i,b,h])).
category([s,p,c,-,p,2],lexical([c(r),o,m,h,a,i,b,h])).
category([s,p,c,-,p,2],lexical([u,m,a,i,b,h])).
category([s,p,c,-,p,2],lexical([c(u),m,a,i,b,h])).
category([s,p,c,-,p,2],lexical([i,o,n,a,i,b,h])).
category([s,p,c,-,p,2],lexical([c(i),o,n,a,i,b,h])).
category([v,t,i,s,-,-,d],lexical([r,a,i,b,h])).
category([v,t,i,s,-,-,d],lexical([c(r),a,i,b,h])).
category([s,p,c,-,p,2],lexical([t,h,a,r,a,i,b,h])).
category([s,p,c,-,p,2],lexical([c(t),h,a,r,a,i,b,h])).
category([s,p,c,-,p,2],lexical([e,a,d,r,a,i,b,h])).
category([s,p,c,-,p,2],lexical([c(e),a,d,r,a,i,b,h])).
category([s,p,c,-,p,2],lexical([o,r,a,i,b,h])).
category([s,p,c,-,p,2],lexical([c(o),r,a,i,b,h])).
category([s,p,c,-,p,2],lexical([a,s,a,i,b,h])).
category([s,p,c,-,p,2],lexical([c(a),s,a,i,b,h])).
category([s,p,c,-,p,2],lexical([u,a,i,b,h])).
category([s,p,c,-,p,2],lexical([c(u),a,i,b,h])).
category([n,c,m,s,g,-,y],[f(a),i,b,h]).
category([n,c,m,s,g,-,y],[f(e),i,b,h]).
category([s,p,c,-,p,2],lexical([l,i,b,h])).
APPENDIX A. PROGRAM CODE

category([s,p,c,-,p,2],lexical([c(1),i,b,h])).
category([n,c,m,s,g,_,y],[o,i,b,h]).
category([s,p,c,-,p,2],lexical([d,a,o,i,b,h])).
category([s,p,c,-,p,2],lexical([c(d),a,o,i,b,h])).
category([s,p,c,-,p,3],lexical([d,f(o),i,b,h])).
category([s,p,c,-,p,3],lexical([c(d),f(o),i,b,h)]).
category([p,p,-,p,-,n],lexical([s,i,b,h])).
category([p,p,-,p,-,n],lexical([c(s),i,b,h])).
category([n,c,f,s,n,-,n],[u,i,b,h]).
category([a,q,p,-,-,g],[u,i,b,h]).
category([n,c,p,-,p,2],lexical([f,f(u),i,b,h])).
category([n,c,p,-,p,2],lexical([c(f),f(u),i,b,h)]).
category([n,c,p,-,p,2],lexical([d,f(i),b,h])).
category([n,c,p,-,p,2],lexical([c(d),f(i),b,h)]).
category([n,c,f,s,n,-,n],[a,l,b,h]).
category([a,q,p,-,-,l],[a,l,b,h]).
category([n,c,f,s,n,-,n],[i,l,b,h]).
category([a,q,p,-,-,g],[i,l,b,h]).
category([n,c,m,s,g,_,y],[a,n,b,h]).
category([n,c,m,s,g,_,y],[i,n,b,h]).
category([n,c,m,s,g,_,y],[o,b,h]).
category([v,m,m,p,2,s,-,n],[o,b,h]).
category([s,p,c,-,p,3],lexical([d,f(i),o,b,h])).
category([s,p,c,-,p,3],lexical([c(d),f(i),o,b,h)]).
category([n,c,m,s,n,-,n],[a,r,b,h]).
category([a,q,p,-,-,l],[a,r,b,h]).
category([],lexical([c,f(a),r,b,h)]).
category([],lexical([c(c),f(a),r,b,h)]).
category([],lexical([n,f(a),r,b,h)]).
category([],lexical([c(n),f(a),r,b,h)]).
category([],lexical([c(f,e),r,b,h)]).
category([],lexical([c(c),f(e),r,b,h)]).
category([n,c,m,s,g,_,y],[i,r,b,h]).
category([a,q,p,-,-,g],[i,r,b,h]).
category([],lexical([g,u,r,b,h])).
category([],lexical([c(g),u,r,b,h])).
category([],lexical([a,c,h])).
category([],lexical([c(a),c,h])).
category([s,p,s,n,n,0],lexical([a,c,h])).
category([s,p,s,n,n,0],lexical([c(a),c,h])).
category([n,c,m,s,n,-,n],[a,c,h]).
category([a,q,p,-,-,l],[a,c,h]).
category([u,n],lexical([n,a,c,h])).
category([n,a,c,h],lexical([n,a,c,h])).
category([u,n],lexical([c(n),a,c,h])).
category([n,a,c,h],lexical([c(n),a,c,h])).
category([a,q,p,-,-,l],[f(a),c,h]).
category([v,m,m,p,2,s,-,n],[i,c,h]).
category([a,q,p,-,-,l],[o,c,h]).
APPENDIX A. PROGRAM CODE

category([n,c,m,s,n,-,n],[f(o),c,h]).
category([n,c,m,s,n,-,n],[u,c,h]).
category([a,q,p,-,-,-],[u,c,h]).
category([n,c,f,s,n,-,n],[f(u),c,h]).
category([v,m,m,p,2,s,-,n],[f(u),c,h]).
category([n,c,m,s,n,-,n],[b,a,d,h]).
category([v,m,i,s,0,-,-,n],[b,a,d,h]).
category([v,m,i,h,-,-,n],[b,a,d,h]).
category([v,m,i,m,-,-,n],[b,a,d,h]).
category([n,c,m,s,n,-,n],[c,a,d,h]).
category([v,m,i,s,0,-,-,n],[c,a,d,h]).
category([v,m,i,h,-,-,n],[c,a,d,h]).
category([v,m,m,p,-,-,-],[c,a,d,h]).
category([n,c,m,s,n,-,n],[d,a,d,h]).
category([v,m,i,s,0,-,-,n],[d,a,d,h]).
category([v,m,i,h,-,-,n],[d,a,d,h]).
category([v,m,m,p,-,-,-],[d,a,d,h]).
category([n,c,m,s,n,-,n],[e,a,d,h]).
category([v,m,i,s,0,-,-,n],[e,a,d,h]).
category([v,m,c,-,-,-],[e,a,d,h]).
category([v,m,i,h,-,-,n],[e,a,d,h]).
category([v,m,m,p,-,-,-],[e,a,d,h]).
category([n,c,m,s,n,-,n],[f,a,d,h]).
category([v,m,i,s,0,-,-,n],[f,a,d,h]).
category([v,m,c,-,-,-],[f,a,d,h]).
category([v,m,i,h,-,-,n],[f,a,d,h]).
category([v,m,m,p,-,-,-],[f,a,d,h]).
category([n,c,m,s,n,-,n],[g,a,d,h]).
category([v,m,i,s,0,-,-,n],[g,a,d,h]).
category([v,m,i,h,-,-,n],[g,a,d,h]).
category([v,m,m,p,-,-,-],[g,a,d,h]).
category([n,c,m,s,n,-,n],[b,h,a,d,h]).
category([v,m,i,s,0,-,-,n],[b,h,a,d,h]).
category([v,m,i,h,-,-,n],[b,h,a,d,h]).
category([v,m,m,p,-,-,-],[b,h,a,d,h]).
category([n,c,m,s,n,-,n],[c,h,a,d,h]).
category([v,m,i,s,0,-,-,n],[c,h,a,d,h]).
category([v,m,i,h,-,-,n],[c,h,a,d,h]).
category([v,m,m,p,-,-,-],[c,h,a,d,h]).
category([n,c,m,s,n,-,n],[g,h,a,d,h]).
category([v,m,i,s,0,-,-,n],[g,h,a,d,h]).
category([v,m,i,h,-,-,n],[g,h,a,d,h]).
category([v,m,m,p,-,-,-],[g,h,a,d,h]).
category([n,c,m,s,n,-,n],[m,h,a,d,h]).
category([v,m,i,s,0,-,-,n],[m,h,a,d,h]).
category([v,m,i,h,-,-,n],[m,h,a,d,h]).
category([v,m,m,p,-,-,-],[m,h,a,d,h]).
category([n,c,m,s,n,-,n],[t,h,a,d,h]).
category([v,m,i,s,0,-,-,n],[t,h,a,d,h]).
category([v,m,i,h,-,-,n],[t,h,a,d,h]).
APPENDIX A. PROGRAM CODE

category([v,m,m,p,\text{\textbf{\textordfou}}\text{-}},\ldots,\text{	extbf{\textordfou}}}],[t,h,a,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[1,a,d,h]).
category([v,m,i,s,0,\text{-}},\ldots,\text{-}],[1,a,d,h]).
category([v,m,i,h,\text{-}},\ldots,\text{-}],[1,a,d,h]).
category([v,m,m,p,\text{-}},\ldots,\text{-}],[1,a,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[m,a,d,h]).
category([v,m,i,s,0,\text{-}},\ldots,\text{-}],[m,a,d,h]).
category([v,m,m,p,\text{-}},\ldots,\text{-}],[m,a,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[n,a,d,h]).
category([v,m,i,s,0,\text{-}},\ldots,\text{-}],[n,a,d,h]).
category([v,m,m,p,\text{-}},\ldots,\text{-}],[n,a,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[p,a,d,h]).
category([v,m,i,s,0,\text{-}},\ldots,\text{-}],[p,a,d,h]).
category([v,m,i,h,\text{-}},\ldots,\text{-}],[p,a,d,h]).
category([v,m,m,p,\text{-}},\ldots,\text{-}],[p,a,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[r,a,d,h]).
category([v,m,i,s,0,\text{-}},\ldots,\text{-}],[r,a,d,h]).
category([v,m,i,h,\text{-}},\ldots,\text{-}],[r,a,d,h]).
category([v,m,m,p,\text{-}},\ldots,\text{-}],[r,a,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[s,a,d,h]).
category([v,m,i,s,0,\text{-}},\ldots,\text{-}],[s,a,d,h]).
category([v,m,i,h,\text{-}},\ldots,\text{-}],[s,a,d,h]).
category([v,m,m,p,\text{-}},\ldots,\text{-}],[s,a,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[t,a,d,h]).
category([v,m,i,s,0,\text{-}},\ldots,\text{-}],[t,a,d,h]).
category([v,m,i,h,\text{-}},\ldots,\text{-}],[t,a,d,h]).
category([v,m,m,p,\text{-}},\ldots,\text{-}],[t,a,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[u,a,d,h]).
category([v,m,i,s,0,\text{-}},\ldots,\text{-}],[u,a,d,h]).
category([v,m,i,h,\text{-}},\ldots,\text{-}],[u,a,d,h]).
category([v,m,m,p,\text{-}},\ldots,\text{-}],[u,a,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[a,i,d,h]).
category([v,m,i,f,\text{-}},\ldots,\text{-}],[a,i,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[f(a),i,d,h]).
category([v,t,i,f,\text{-}},\ldots,\text{-}],[f(a),i,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[f(a),i,d,h]).
category([v,m,i,f,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([v,m,i,f,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([v,m,i,f,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([v,m,i,f,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([v,m,i,f,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([v,m,i,f,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([v,m,i,f,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([v,m,i,f,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([v,m,i,f,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([v,m,i,f,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([v,m,i,f,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([v,m,i,f,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([v,m,i,f,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([v,m,i,f,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([n,c,m,s,n,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).
category([v,m,i,f,\text{-}},\ldots,\text{-}],[f(e),i,d,h]).

\% tuillidh=more-GEN
\% cinnidh=decision-GEN
\% geimhridh=winter-GEN
\% actually -(a)iodh
APPENDIX A. PROGRAM CODE

category([v,m,c,_,_,-,_,_], [o,d,h]).
category([v,m,c,_,_,-,_,_], [f(o),d,h]).
category([n,c,m,p,n,_,_], [i,g,h]).
category([n,c,m,s,g,_,_], [i,g,h]).
category([v,m,m,p,2,s,_,n], [i,g,h]). % 2nd verbal conjugation
category([n,c,m,p,n,_,_], [f(i),g,h]). % gaiscigh=warriors/heroes
% cloigh=to stick to/to cleave/to subdue
category([v,m,m,p,2,s,_,n], [f(i),g,h]).
% =choose/select/elect
category([v,m,m,p,2,s,_,n], lexical([t,o,g,h])).
category([v,m,m,p,2,s,_,n], lexical([c(t),o,g,h])).
category([n,c,m,s,n,_,_], [o,g,h]).
category([n,c,m,s,n,_,_], [a,m,h]).
category([a,q,p,--,n], [a,m,h]).
category([a,q,p,--,p,g], [a,m,h]).
category([n,c,m,s,n,_,_], [f(a),m,h]).
category([a,q,p,--,n], [f(a),m,h]).
category([n,c,m,p,2,s,_,n], [i,m,h]).
category([n,c,m,m,_,_], [i,m,h]).
category([a,q,p,--,n], [i,m,h]).
category([r,o,i,m,h], lexical([r,o,i,m,h])) . % simple preposition
category([r,o,i,m,h], lexical([c(r),o,i,m,h])) .
category([s,p,s,n,n,0], lexical([r,o,i,m,h])) . % simple preposition
category([s,p,s,n,n,0], lexical([c(r),o,i,m,h])) .
category([n,c,m,s,g,_,_], [f(i),m,h]).
category([c,h,o,m,h], lexical([c,h,o,m,h])).
category([c,h,o,m,h], lexical([c(c),h,o,m,h)]).
category([n,c,m,s,n,_,_], [o,m,h]).
category([v,m,m,p,2,s,_,n], [o,m,h]).
category([n,c,m,s,n,_,_], [f(u),m,h]).
category([n,c,m,s,n,_,_], [a,t,h]).
category([v,m,m,p,2,s,_,n], lexical([i,t,h])) .
category([v,m,m,p,2,s,_,n], lexical([c(i),t,h])) .
category([n,c,m,s,n,_,_], [i,t,h]).
category([n,c,f,s,n,_,_], [i,t,h]).
category([v,m,m,p,2,s,_,n], [i,t,h]).
category([v,m,m,p,2,s,_,n], lexical([c,a,i,t,h])).
category([v,m,m,p,2,s,_,n], lexical([c(c),a,i,t,h])).
category([a,q,p,--,n], lexical([m,a,i,t,h])).
category([a,q,p,--,n], lexical([c(m),a,i,t,h])).
category([v,t,n,_,_,_,_,d], lexical([b,h,e,i,t,h])) . % ???
category([v,t,n,_,_,_,_,d], lexical([c(b),h,e,i,t,h])).
category([n,c,f,s,n,_,_], [f(i),t,h]).
category([n,c,m,s,n,_,_], [o,t,h]).
category([n,c,f,s,n,_,_], [o,t,h]).
category([n,c,m,s,n,_,_], [u,t,h]).
category([n,c,m,s,n,_,_], [f(u),t,h]).

% category(CategoryInfo, [i]).
APPENDIX A. PROGRAM CODE

category([i], lexical([i]))
category([i], lexical([c(i)]))
category([s,p,s,n,n,0], lexical([i]))
category([s,p,s,n,n,0], lexical([c(i)]))
category([s,p,c,f,s,3], lexical([a,i,c,i]))
category([s,p,c,f,s,3], lexical([c(a), i, c, i]))
category([s,p,c,f,s,3], lexical([c(h), u, i, c, i]))
category([s,p,c,f,s,3], lexical([c(c), h, u, i, c, i]))
category([s,p,c,f,s,3], lexical([d, i]))
category([s,p,c,f,s,3], lexical([c(d), i]))
category([s,p,c,f,s,3], lexical([l, f(e), i]))
category([s,p,c,f,s,3], lexical([c(l), f(e), i]))
category([s,p,c,f,s,3], [t, h, i])
category([f, a, o, i], lexical([f, a, o, i])). % simple preposition
category([f, a, o, i], lexical([c(f), a, o, i]))
category([s,p,s,n,n,0], lexical([f, a, o, i]))
category([s,p,s,n,n,0], lexical([c(f), a, o, i]))
category([s,p,c,m,s,3], lexical([f, a, o, i]))
category([s,p,c,m,s,3], lexical([c(f), a, o, i]))
category([m,c,[-,[-]], lexical([n, a, o, i])). % cardinal number 9
category([m,c,[-,[-]], lexical([c(n), a, o, i]))
category([n,c,m,s,n,[-,n], lexical([a, o, i]))
category([n,c,m,s,n,[-,n], lexical([c(a), o, i]))
category([n,c,m,s,n,[-,n], [a, o, i]). % e.g. draoi
category([n,c,f,s,n,[-,n], [a, o, i]). % e.g. laoi
category([s,p,c,f,s,3], [p, i])
category([s,p,c,f,s,3], [t, i])

% category(CategoryInfo, [i]).

% pronoun in the accusative case
category([p,p,3,f,s,a,[-]], lexical([f(i)]))
category([p,p,3,f,s,a,[-]], lexical([c(f)]))
category([n,c,m,s,n,[-,n], [f(i)]).
category([n,c,[-,p,[-,y], [f(i)].
category([n,c,f,s,g,[-,y], [f(i)]).
category([v,m,c,[-,0,[-,y], [f, a, f(i)].
category([v,m,i,h,0,[-,y], [t, a, f(i)].
category([v,m,c,[-,0,[-,y], [f, f(i)].
category([v,m,m,p,2,[-,y], [i, g, f(i)].
category([v,m,m,p,2,[-,y], [f(i), g, f(i)].
category([u,n], lexical([n, f(i)])). % negative verbal particle
category([n,f(i)], lexical([n, f(i)]))
category([u,n], lexical([c(n), f(i)]))
category([n,f(i)], lexical([c(n), f(i)]))
category([v,m,i,h,0,[-,y], [t, f(i)].
category([t,r,f(i)], lexical([t, r, f(i)])). % simple preposition
category([t,r,f(i)], lexical([c(t), r, f(i)])).
category([s,p,s,n,n,0], lexical([t, r, f(i)])). % simple preposition
APPENDIX A. PROGRAM CODE

```plaintext
category([s,p,s,n,n,0],lexical([c(t),r,f(i)])).
category([m,c,_,_,_],lexical([t,r,f(i)])).
category([m,c,_,_,_],lexical([c(t),r,f(i)])).
category([n,c,m,s,n,_,n],lexical([t,r,f(i)])).
category([n,c,m,s,n,_,n],lexical([c(t),r,f(i)])).

% category(CategoryInfo, [1]).

category([n,c,m,s,n,_,n], [a,1]).
category([a,q,p,_,_,_],lexical([g,e,a,1])).
category([a,q,p,_,_,_],lexical([c(g),e,a,1])).
category([n,c,m,s,n,_,n], [f(a),1]).
category([n,c,m,p,n,_,n], [a,i,1]).
category([n,c,m,s,g,_,y], [a,i,1]).
category([n,c,f,s,n,_,n], [f(a),i,1]).
category([n,c,f,s,n,_,n], [c,i,1]).
category([n,c,f,s,n,_,n], [d,i,1]).
category([n,c,f,s,n,_,n], [e,i,1]).
category([n,c,f,s,n,_,n], [f(e),i,1]).
category([n,c,m,s,g,_,y], [f(e),i,1]).
category([a,q,p,_,_,_], [f(u),i,1]).
category([n,c,f,s,n,_,n], [a,f(i),1]).
category([n,c,m,s,n,_,n], [a,l,1]).
category([n,c,f,s,n,_,n], [a,l,1]).
category([n,c,m,s,n,_,n], [o,l,1]).
category([n,c,m,s,n,_,n], [o,l]).
category([n,c,m,s,n,_,n], [f(o),1]).
category([n,c,m,s,n,_,n], [f(u),1]).

% category(CategoryInfo, [m]).

category([n,c,m,s,n,_,n], [a,m]).
category([a,q,p,_,_,_], [a,m]).
category([s,p,c,_,s,1],lexical([a,g,a,m])).
category([s,p,c,_,s,1],lexical([c(a),g,a,m])).
category([s,p,c,_,s,1],lexical([c,h,u,g,a,m])).
category([s,p,c,_,s,1],lexical([c(l),h,u,g,a,m])).
category([s,p,c,_,s,1],lexical([r,o,m,h,a,m])).
category([s,p,c,_,s,1],lexical([c(r),o,m,h,a,m])).
category([s,p,c,_,s,1],lexical([f,a,r,a,m])).
category([s,p,c,_,s,1],lexical([c(f),a,r,a,m])).
category([s,p,c,_,s,1],lexical([t,h,a,r,a,m])).
category([s,p,c,_,s,1],lexical([c(t),h,a,r,a,m])).
category([s,p,c,_,s,1],lexical([a,s,a,m])).
category([s,p,c,_,s,1],lexical([c(a),s,a,m])).
category([n,c,m,s,n,_,n], [f(a),m]).
category([v,m,m,p,2,s,n,n], [f(a),m]).
category([n,c,m,s,n,_,n], [a,d,h,m]).
```

APPENDIX A. PROGRAM CODE

category([n, c, m, s, g, _, y], [i, d, h, m]).
category([n, c, f, s, n, _, n], [i, d, h, m]).
category([n, c, f, s, n, _, n], lexical([i, m])).
category([n, c, f, s, n, _, n], lexical([c(i, m)]).
category([s, p, c, -, s, l], lexical([u, a, i, m])).
category([n, c, m, s, g, _, y], [a, i, m]).
category([v, m, i, p, 1, s, _, _], [a, i, m]).
category([v, m, m, p, 1, s, _, _], [a, i, m]).
category([v, m, i, p, 1, s, _, _], [f(a), i, m]).
category([v, m, m, p, 1, s, _, _], [f(a), i, m]).
category([v, m, i, p, 1, s, _, _], [c, i, m]).
category([v, m, i, p, 1, s, _, _], [d, i, m]).
category([v, m, m, p, 1, s, _, _], [d, i, m]).
category([n, c, m, s, n, _, n], [e, i, m]).
category([v, m, m, p, 1, s, _, _], [m, i, m]).
category([v, m, m, p, 2, s, _, n], [e, i, m]).
category([n, c, f, s, n, _, n], [f(e), i, m]).
category([v, m, m, p, 2, s, _, n], [f(e), i, m]).
category([v, m, m, p, 1, s, _, _], [f(e), i, m]).
category([v, m, i, p, 1, s, _, _], [h, i, m]).
category([v, m, m, p, 1, s, _, _], [h, i, m]).
category([v, m, i, p, 1, s, _, _], [l, i, m]).
category([v, m, m, p, 1, s, _, _], [l, i, m]).
category([n, c, m, s, n, _, n], [o, i, m]).
category([v, m, i, p, 1, s, _, _], [f(o), i, m]).
category([v, m, m, p, 1, s, _, _], [f(o), i, m]).
category([n, c, m, s, g, _, y], [r, i, m]).
category([v, m, i, p, 1, s, _, _], [r, i, m]).
category([v, m, m, p, 1, s, _, _], [r, i, m]).
category([v, m, m, p, 1, s, _, _], [s, i, m]).
category([v, m, m, p, 1, s, _, _], [s, i, m]).
category([v, m, m, p, 1, s, _], [s, i, m]).
category([a, q, p, -, -, -], lexical([t, i, m])).
category([a, q, p, -, -, -], lexical([c(t), i, m])).
category([n, c, f, s, n, _, n], [t, i, m]).
category([v, m, i, p, 1, s, _, _], [t, i, m]).
category([v, m, m, p, 1, s, _, _], [t, i, m]).
category([n, c, f, s, n, _, n], [u, i, m]).
category([v, m, i, p, 1, s, _, _], [f(u), i, m]).
category([v, m, m, p, 1, s, _, _], [f(u), i, m]).
category([n, c, f, s, n, _, n], [f(i), i, m]).
category([v, m, m, p, 1, s, _], [f(i), i, m]).
category([n, c, m, s, n, _, n], [a, l, m]).
category([n, c, m, s, g, _, y], [l, i, m]).
category([n, c, m, s, n, _, n], [o, l, m]).
category([n, c, m, s, n, _, n], lexical([a, i, n, m])).
category([c(a), i, n, m]).
category([n, c, f, s, n, _, n], lexical([c(s), e, i, n, m]))
category([n, c, f, s, n, _, n], lexical([c(s), e, i, n, m])).
category([n,c,m,s,n,_,n],[o,m]).
category([v,m,m,p,2,s,_,n],[o,m]).
category([v,m,i,p,1,s,_,n],[o,m]).
category([a,q,p,_,-,-,-],[o,m]).
category([s,p,c,_,-,s,1],lexical([d,o,m])).
category([s,p,c,_,-,s,1],lexical([c(d),o,m])).
category([s,p,c,_,-,s,1],lexical([1,i,o,m])).
category([s,p,c,_,-,s,1],lexical([c(l),i,o,m])).
category([s,p,c,_,-,s,1],lexical([d,f(i),o,m])).
category([s,p,c,_,-,s,1],lexical([c(d),f(i),o,m])).
category([s,p,c,_,-,s,1],lexical([t,r,f(i),o,m])).
category([s,p,c,_,-,s,1],lexical([c(t),r,f(i),o,m])).
category([n,c,m,s,n,_,n],[f(o),m]).
category([n,c,m,s,n,_,n],[a,r,m]).
category([n,c,m,s,g,_,n],[i,r,m]).
category([n,c,f,s,n,_,n],[i,r,m]).
category([s,p,c,_,-,s,1],lexical([o,r,m])).
category([s,p,c,_,-,s,1],lexical([c(o),r,m])).
category([a,q,p,_,-,-,-],[o,r,m]).
category([u,m],lexical([u,m])). % simple preposition
category([u,m],lexical([c(u),m])).
category([s,p,s,n,n,0],lexical([u,m])). % simple preposition
category([s,p,s,n,n,0],lexical([c(u),m])).
category([v,m,m,m,p,2,s,_,n],[u,m]).
category([n,c,m,s,n,_,n],[f(u),m]).

% category(CategoryInfo,[n]).
category([t,d,m,s,_,-],lexical([a,n])).
category([t,d,m,s,_,-],lexical([c(a),n])).
category([t,d,f,s,n],lexical([a,n])).
category([t,d,f,s,n],lexical([c(a),n])).
category([u,q],lexical([a,n])). % interrogative verbal particle
category([a,n],lexical([a,n])).
category([a,n],lexical([c(a),n])).
category([a,n],lexical([c(a),n])).
category([v,i,_,-,_,-,_,-,a],lexical([a,n])).
category([v,i,_,-,_,-,_,-,a],lexical([c(a),n])).
category([n,c,n,n,_,n],[a,n]).
category([n,c,f,_,-,g,_,y],lexical([p,e,a,r,s,a,n])).
category([n,c,f,_,-,g,_,y],lexical([c(p),e,a,r,s,a,n])).
category([n,c,f,_,-,g,_,y],lexical([c(o,m,h,a,r,s,a,n)]).
category([n,c,f,_,-,g,_,y],lexical([c(c),o,m,h,a,r,s,a,n)]).
category([n,c,m,s,n,_,n],[f(a),n]).
category([g,a,n],lexical([g,a,n])). % simple preposition
category([g,a,n],lexical([c(g),a,n])).
category([s,p,s,n,n,0],lexical([g,a,n])). % simple preposition
category([s,p,s,n,n,0],lexical([c(g),a,n])).
% simple preposition 'i' + definite article occurring before vowel or 'fh'
category([s,a,n],lexical([s,a,n])).
APPENDIX A. PROGRAM CODE

category([s,a,n], lexical([c(s), a, n]))

% simple preposition 'de' + definite article
category([d,e,n], lexical([d, e, n]))
category([d,e,n], lexical([c(d), e, n]))
category([n,c,m,s,n,...,n], [e,n])

% interrogative pronoun with article
category([t,d,m,s,-], lexical([c, f(e), n]))
category([t,d,m,s,-], lexical([c(c), f(e), n]))
% affects following word like nom. article
category([t,d,f,s,n], lexical([c, f(e), n]))
category([t,d,f,s,n], lexical([c(c), f(e), n]))
category([n,c,f,s,n,...,n], [d,h,n])
category([s,p,c,n,n,0], lexical([i,n]))
category([s,p,c,n,n,0], lexical([c(i), n]))
category([p,d,-,-,-,...,-], lexical([i,n]))
category([p,d,-,-,-,...,-], lexical([c(i), n]))
category([n,c,f,s,n,...,n], [a,i,n])
category([n,c,m,p,n,...,y], [a,i,n])
category([n,c,m,s,g,...,y], [a,i,n])
category([r,q,-,-,...,], lexical([c, a, t, h, a, i, n]))
% interrogative adverb
category([r,q,-,-,...,], lexical([c(c), a, t, h, a, i, n]))
category([n,c,m,p,n,...,y], [f(a), i, n])
category([n,c,m,s,g,...,y], [f(a), i, n])
category([a,q,p,-,-,-], [f(a), i, n])
category([n,c,f,s,n,...,n], [c,i,n])
category([n,c,f,s,n,...,n], [d,i,n])
category([n,c,m,s,g,...,y], [e,i,n])
category([n,c,m,s,g,...,y], [e,i,n])
category([p,x,-,-,-,...,-], lexical([f, f(e), i, n]))
category([p,x,-,-,-,...,-], lexical([c(f), f(e), i, n]))
category([a,q,-,-,-,-], lexical([f, f(e), i, n]))
category([a,q,-,-,-,-], lexical([c(f), f(e), i, n]))
category([n,c,m,s,g,...,y], [f(e), i, n])
category([n,c,m,s,g,...,y], [f(i), i, n])
category([n,c,f,s,n,...,n], [g,i,n])
category([a,q,p,m,s,g], [g,i,n])
category([n,c,f,s,n,...,n], [h,i,n])
category([a,q,p,m,s,g], [h,i,n])
category([n,c,f,s,n,...,n], [m,i,n])

% simple preposition 'faoi' + definite article
category([f,a,o,i,n], lexical([f, a, o, i, n]))
category([f,a,o,i,n], lexical([c(f), a, o, i, n]))
category([n,c,f,s,n,...,n], [o,i,n])
% sceoin=terror
category([v,m,m,p,2,s,...,n], [o,i,n])
% caoin=to lament
category([a,q,p,-,-,-], [o,i,n])
% diomhaoin=idle
category([n,c,m,p,n,...,y], [f(o), i, n])
category([n,c,m,s,g,...,y], [f(o), i, n])
category([n,c,m,s,g,...,y], [r,i,n])
category([d,d,-,-,-,...,-], lexical([s,i,n]))
category([d,d,-,-,-,...,-], lexical([c(s), i, n]))
category([a,q,p,-,-,-], lexical([s,i,n]))
APPENDIX A. PROGRAM CODE

```prolog
category([a,q,p,--,--],[lexical([c(s),i,n])]).
category([r,g,--,--],[lexical([a,n,s,i,n])]).
category([r,g,--,--],[lexical([c(a),n,s,i,n])]).
category([r,g,--,--],[lexical([f,r,e,i,s,i,n])]).
category([r,g,--,--],[lexical([c(f),r,e,i,s,i,n])]).
category([n,c,m,s,g,--],[s,i,n])], [n,c,f,s,n,--],[t,i,n])].
category([n,c,m,s,--],[u,i,n])].
category([n,c,m,s,--],[f(u),i,n])].
category([n,c,m,s,--],[f(i),n])].
category([n,c,f,s,n,--],[f(i),n])].
category([r,g,--,--],[lexical([a,n,n])]).
category([r,g,--,--],[lexical([c(a),n,n])]).
category([s,p,c,--],[lexical([a,n,n])]).
category([s,p,c,--],[lexical([c(a),n,n])]).
category([n,c,m,s,--],[lexical([c(a),n,n])]).
category([v,m,i,p,--],[i,n,n])].
category([v,m,i,p,--],[f(a),n,n])]. % crann=torments
category([n,c,s,n,--],[i,n,n])].
category([v,m,i,h,--],[i,n,n])]. % actually -ainn/-finn
category([s,p,c,--],[i,n,n])].
category([s,p,c,--],[f(i),n,n])].
category([v,m,i,h,--],[f(i),n,n])].
category([n,c,m,s,--],[o,n,n])].
category([a,q,p,--],[o,n,n])].
category([v,m,i,p,--],[f(a),n,n])]. % donn=burns
category([v,m,i,p,--],[f(u),n,n])]. % brdn=pushes
category([n,c,m,s,--],[lexical([a,o,n])]).
category([n,c,m,s,--],[lexical([c(a),o,n])]).
category([m,c,--],[lexical([a,o,n])]).
category([m,c,--],[lexical([c(a),o,n])]).
category([d,o,n],lexical([d,o,n]))].
category([d,o,n],lexical([c(d),o,n]))].
% m1/m3, more subdivision of suffixes???
category([n,c,m,s,--],[o,n])].
category([n,c,f,s,--],[o,n])]. % f2/f3
% simple preposition 'do' + definite article
category([f(o),n],lexical([f(o),n]))].
category([f(o),n],lexical([cf(o),n]))].
category([n,c,f,s,--],[f(o),n])].
category([n,c,f,--],[a,r])].
category([n,c,m,p,--],[a,r])].
category([n,c,m,p,--],[s,i,n])].
category([n,c,f,s,--],[s,i,n])].
category([n,c,m,p,--],[y],[i,r,n])].
```


% category(CategoryInfo, [o]).

category([d, o], lexical([d, o])). % simple preposition
category([d, o], lexical([c(d), o)]).
category([s, p, s, n, n, 0], lexical([d, o])). % simple preposition
category([s, p, s, n, n, 0], lexical([c(d), o)]).
category([d, d, -, -, -, -, ], lexical([s, e, o])). % determiner
category([d, d, -, -, -, -, ], lexical([c(s), e, o)]).
category([a, q, p, -, -, -, ], lexical([s, e, o])). % adjective
category([a, q, p, -, -, -, ], lexical([c(s), e, o)]).
category([p, d, -, -, -, -, ], lexical([s, e, o])). % demonstrative pronoun
category([p, d, -, -, -, -, ], lexical([c(s), e, o)]).
category([r, g, -, -, -], lexical([a, n, s, e, o])). % adverb
category([r, g, -, -, -], lexical([c(a), n, s, e, o)]).
category([u, a], lexical([g, o])).
category([u, a], lexical([c(g), o)]).
category([g, o], lexical([g, o])). % simple preposition
category([g, o], lexical([c(g), o)]).
category([s, p, s, n, n, 0], lexical([g, o])). % simple preposition
category([s, p, s, n, n, 0], lexical([c(g), o)]).
category([c, s, -, -], lexical([g, o])].
category([c, s, -, -], lexical([c(g), o)]).
category([u, s], lexical([g, o])].
category([u, s], lexical([c(g), o)]).
category([u, q], lexical([g, o])].
category([u, q], lexical([c(g), o)]).
category([d, p, 1, -, s, -], lexical([m, o])). % possessive pronoun
category([d, p, 1, -, s, -], lexical([c(m), o)]).
category([d, p, 2, -, s, -], lexical([d, o])). % possessive pronoun
category([d, p, 2, -, s, -], lexical([c(d), o)]).
category([n, c, m, s, -, -], [o]). % e.g. lao, ceo
category([a, q, p, -, -, -], [o]). % e.g. beo

% category(CategoryInfo, [o]).

category([f(o)], lexical([f(o)])]. % simple preposition
category([f(o)], lexical([c(f(o)])].
category([s, p, s, n, n, 0], lexical([f(o)])]. % simple preposition
category([s, p, s, n, n, 0], lexical([c(f(o)])].
category([u, p], lexical([f(o)])].
category([u, p], lexical([c(f(o)])].
category([i], lexical([c(f(o)])].
category([i], lexical([c(f(o)])].
% prepositional pronoun
category([s,p,c,m,s,3],lexical([d,f(o)])).
category([s,p,c,m,s,3],lexical([c(d),f(o)])).

% cardinal number 70
category([m,c,_,_,_],lexical([s,e,a,c,h,t,f(o)])).
category([m,c,_,_,_],lexical([c(s),e,a,c,h,t,f(o)])).

% cardinal number 80
category([m,c,_,_,_],lexical([o,c,h,t,f(o)])).
category([m,c,_,_,_],lexical([c(o),c,h,t,f(o)])).
category([a,q,c,_,_,_],lexical([m,f(o)])). % comparative
category([a,q,c,_,_,_],lexical([c(m),f(o)])).
category([a,q,c,_,_,_],lexical([m,f(o)])). % superlative
category([a,q,c,_,_,_],lexical([c(m),f(o)])).
category([n,c,m,s,_,_],f(o)). % noun 4th declension
category([n,c,f,s,_,_],f(o)). % noun 4th declension
category([i],lexical([h,f(o)])). % interjection
category([i],lexical([c(h),f(o)])).

category([n,c,m,s,n,_,n],[a,p]).
category([v,m,m,p,2,s,_,n],[a,p]).
category([n,c,m,s,n,_,n],[a,p]).
category([n,c,f,s,n,_,n],[i,d,h,p]). % e.g. faidhp
category([n,c,f,s,n,_,n],[i,p]).
category([n,c,f,s,n,_,n],[f(i),p]).
category([n,c,f,s,n,_,n],[i,l,p]).
category([v,m,m,p,2,s,_,n],[o,l,p]). % intransitive verb
category([n,c,m,s,n,_,n],[o,l,p]).
category([n,c,m,s,_,_],[a,m,p]). % e.g. tramp
category([n,c,f,s,n,_,n],[i,m,p]). % e.g. plimp
category([v,m,m,p,2,s,_,n],[o,m,p]). % intransitive verb
category([n,c,m,s,n,_,n],[u,m,p]). % e.g. tump-thump
category([n,c,m,s,n,_,n],[o,p]).
category([v,m,m,p,2,s,_,n],[o,p]).
category([n,c,m,p,n,_,y],[i,r,p]).
category([n,c,m,s,g,_,y],[i,r,p]).
category([n,c,m,s,n,_,n],[o,r,p]).

% category(CategoryInfo,[p]).
category([n,c,m,s,n,_,n],[a,p]).
category([v,m,m,p,2,s,_,n],[a,p]).
category([n,c,m,s,n,_,n],[a,p]).
category([n,c,f,s,n,_,n],[i,d,h,p]). % e.g. faidhp
category([n,c,f,s,n,_,n],[i,p]).
category([n,c,f,s,n,_,n],[f(i),p]).
category([n,c,f,s,n,_,n],[i,l,p]).
category([v,m,m,p,2,s,_,n],[o,l,p]). % intransitive verb
category([n,c,m,s,n,_,n],[o,l,p]).
category([n,c,m,s,_,_],[a,m,p]). % e.g. tramp
category([n,c,f,s,n,_,n],[i,m,p]). % e.g. plimp
category([v,m,m,p,2,s,_,n],[o,m,p]). % intransitive verb
category([n,c,m,s,n,_,n],[u,m,p]). % e.g. tump-thump
category([n,c,m,s,n,_,n],[o,p]).
category([v,m,m,p,2,s,_,n],[o,p]).
category([n,c,m,p,n,_,y],[i,r,p]).
category([n,c,m,s,g,_,y],[i,r,p]).
category([n,c,m,s,n,_,n],[o,r,p]).

% category(CategoryInfo,[r]).
category([a,r],lexical([a,r])). % simple preposition
category([a,r],lexical([c(a),r])).
category([s,p,s,n,n,0],lexical([a,r])). % simple preposition
category([s,p,s,n,n,0],lexical([c(a),r])).
category([u,b,p],lexical([a,r])).
category([u,b,p],lexical([c(a),r])).
category([u,r],lexical([a,r])).
category([u,r],lexical([c(a),r])).
category([u,q],lexical([a,r])).
APPENDIX A. PROGRAM CODE

```c
category([u,q], lexical([c(a),r)]).
category([a,r], lexical([a,r])).
category([a,r], lexical([c(a),r])).
category([v,y,i,-,-,-,-,-,d,a], lexical([a,r])).
category([v,y,i,-,-,-,-,-,d,a], lexical([c(a),r])).
category([n,c,m,s,n,_,n], [b,a,r]).
category([n,c,m,s,n,_,n], [c,a,r]).
category([r,q,-,-,___], lexical([c,f(a),r])). % interrogative adverb
category([r,q,-,-,___], lexical([c(c),f(a),r])).
category([c,f(a),r], lexical([c(f(a),r)]).
category([c,c,f(a),r], lexical([c(c),f(a),r)]).
category([n,c,m,s,n,_,n], [d,a,r]).
category([n,c,m,s,n,_,n], [e,a,r]).
category([v,m,i,p,0,-,-,-,___], [e,a,r]). % should be more specific - fear
category([v,m,i,f,0,-,-,-,___], [e,a,r]). % should be more specific - tear
category([v,m,m,p,0,-,-,-,___], [e,a,r]). % should be more specific - design
category([v,m,m,p,2,s,_,n], lexical([d,e,a,r])). % =design
category([v,m,m,p,2,s,_,n], lexical([c(d),e,a,r])).
category([n,c,m,s,n,_,n], lexical([c,f(u),i,g,e,a,r])).
category([n,c,m,s,n,_,n], lexical([c(c),f(u),i,g,e,a,r])).
category([m,p,___,___], lexical([c,f(u),i,g,e,a,r])).
category([m,p,___,___], lexical([c(c),f(u),i,g,e,a,r])).
category([n,c,m,s,n,_,n], lexical([s,e,i,s,e,a,r])).
category([n,c,m,s,n,_,n], lexical([c(s),e,i,s,e,a,r])).
category([m,p,___,___], lexical([s,e,i,s,e,a,r])).
category([m,p,___,___], lexical([c(s),e,i,s,e,a,r])).
category([n,c,m,s,n,_,n], [f(e),a,r]).
category([a,q,p,___,___], [f,a,r]).
category([v,m,i,f,0,-,-,-,___], [f,a,r]).
category([n,c,m,s,n,_,n], [g,a,r]).
category([n,c,m,s,n,_,n], [b,h,a,r]).
category([n,c,m,s,n,_,n], [c,h,a,r]).
category([n,c,m,s,n,_,n], [d,h,a,r]).
category([n,c,m,s,n,_,n], [g,h,a,r]).
category([a,q,p,___,___], [m,h,a,r]).
category([t,h,a,r], lexical([t,h,a,r])). % simple preposition
category([t,h,a,r], lexical([c(t),h,a,r])).
category([s,p,s,n,n,0], lexical([t,h,a,r])). % simple preposition
category([s,p,s,n,n,0], lexical([c(t),h,a,r])).
category([n,c,m,s,n,_,n], [t,h,a,r]).
category([n,c,m,s,g,_,y], [t,h,a,r]).
category([n,c,f,s,g,_,y], [t,h,a,r]).
category([v,m,m,p,2,s,_,n], [i,a,r]).
category([a,q,p,___,___], [i,a,r]).
category([n,c,m,s,n,_,n], [l,a,r]).
category([m,a,r], lexical([m,a,r])). % simple preposition
category([m,a,r], lexical([c(m),a,r])).
category([s,p,s,n,n,0], lexical([m,a,r])). % simple preposition
category([s,p,s,n,n,0], lexical([c(m),a,r])).
category([c,s,___,___], lexical([m,a,r])).
```
APPENDIX A. PROGRAM CODE

```c

category([c,s,_,_],lexical([c(m),a,r)]).
category([r,r,_,_],lexical([m,a,r])).
category([r,r,_,_],lexical([c(m),a,r])).
category([n,c,m,s,n,_,n],[m,a,r]).
category([v,m,i,s,i,p,_,_],[m,a,r]).
category([n,c,m,s,n,_,n],[n,a,r]).
category([u,n],lexical([n,f(a),r])). % negative interogative
category([n,f(a),r],lexical([n,f(a),r])).
category([c,s,_,_],lexical([m,u,n,a,r])).
category([c,s,_,_],lexical([c(m),u,n,a,r])).
category([n,c,m,s,n,_,n],[p,a,r]).
category([n,c,m,s,n,_,n],lexical([c,e,a,t,h,r,a,r])).
category([n,c,m,s,n,_,n],lexical([c(c),e,a,t,h,r,a,r])).
category([m,p,_,_],lexical([c(e),e,a,t,h,r,a,r])).
category([m,p,_,_],lexical([c(c),e,a,t,h,r,a,r])).
category([c,s,_,_],lexical([m,u,r,a,r])).
category([c,s,_,_],lexical([c(m),u,r,a,r])).
category([n,c,m,s,n,_,n],[s,a,r]).
category([v,m,m,p,2,s,_,n],lexical([t,a,r])).
category([v,m,m,p,2,s,_,n],lexical([c(t),a,r])).
category([n,c,m,s,n,_,n],[t,a,r]).
category([v,m,i,p,0,_,_],[t,a,r]).
category([v,m,i,p,0,_,_],[t,a,r]).
category([n,c,m,s,n,_,n],lexical([s,e,a,c,h,t,a,r])).
category([n,c,m,s,n,_,n],lexical([c(s),e,a,c,h,t,a,r])).
category([m,p,_,_],lexical([s,e,a,c,h,t,a,r])).
category([m,p,_,_],lexical([c(s),e,a,c,h,t,a,r])).
category([n,c,m,s,n,_,n],lexical([o,c,h,t,a,r])).
category([n,c,m,s,n,_,n],lexical([c(o),c,h,t,a,r])).
category([m,p,_,_],lexical([o,c,h,t,a,r])).
category([n,c,m,s,n,_,n],[c,u,a,r]).
category([v,m,m,p,2,s,_,n],[c,u,a,r]).
category([a,q,p,_,_],[c,u,a,r]).
category([d,p,1,_,_],[f(a),r]).
category([d,p,1,_,_],[f(a),r]).
category([n,c,m,s,n,_,n],[f(a),r]).
category([],lexical([n,f(a),r])).
category([],lexical([c(n),f(a),r])).
category([s,p,c,_,_],[t,r,f(i),n,f(a),r)]).
category([s,p,c,_,_],[t,r,f(i),n,f(a),r)]).
category([s,p,c,_,_],[f(o),n,f(a),r])).
category([s,p,c,_,_],[f(o),n,f(a),r])).
category([n,c,m,s,n,_,n],[a,e,r]).
category([],lexical([c,f(e),r])).
category([],lexical([c(f(e),r)]).
category([s,p,c,m,s,3],lexical([a,i,r])).
category([s,p,c,m,s,3],lexical([c(a),i,r])).
category([n,c,f,s,n,_,n],[a,i,r]).
category([n,c,f,s,n,_,n],[a,i,r]).
```
APPENDIX A. PROGRAM CODE

117

category([a,q,p,m,s,g], [a,i,r]).
category([a,q,p,-,p,n], [a,i,r]).
category([v,m,i,s, -, -, -, n], lexical([f,u,a,i,r])).
category([v,m,i,s, -, -, -, n], lexical([c(f), u, a, i, r])).
category([], lexical([n,u,a,i,r])).
category([], lexical([c(n), u, a, i, r])).
category([n,c,m,s,g, -, y], [f(a), i, r]).
category([v,m,m,p,2,s, -, n], [b,i,r]).
category([a,q,p,-, -, [c,i,r]]).
category([n,c,f,s,n, -, n], [d,i,r]).
category([a,q,p,-, -, [d,i,r]]).
category([i,d,i,r], lexical([i,d,i,r])).
category([i,d,i,r], lexical([c(i), d, i, r])).
category([e,p,s,n,n,0], lexical([i,d,i,r])).
category([e,p,s,n,n,0], lexical([c(i), d, i, r])).
category([n,c,m,s,n, -, n], [e,i,r]).
category([n,c,m,s,n, -, n], [f(e), i, r]).
category([n,c,m,s, g, -, y], [f(e), i, r]).
category([n,c,m,p,n, -, y], lexical([f,i,r])).
category([n,c,m,p,n, -, y], lexical([c(f), i, r])).
category([n,c,m,s,g, -, y], [f,i,r]).
category([n,c,m,s,g, -, y], [g,i,r]).
category([a,q,p,-, -, [b,h,i,r]]).
category([n,c,f,s,n, -, n], [c,h,i,r]).
category([n,c,f,s,n, -, n], [d,h,i,r]).
category([n,c,f,s,n, -, n], [g,h,i,r]).
category([n,c,f,s,n, -, n], [m,h,i,r]).
category([a,q,p,m,s,g], [m,h,i,r]).
category([n,c,m,s,g, -, y], [t,h,i,r]).
category([a,q,p,m,s,g], [t,h,i,r]).
category([n,c,m,s, g, -, y], [l,i,r]).
category([v,m,m,p,2,s, -, n], [m,i,r]).
category([a,q,p,-, -, [l,m,i,r]]).
category([n,c,f,s,n, -, n], [n,i,r]).
category([n,c,m,s,n, -, n], [o,i,r]).
category([n,c,m,s,n, -, n], [f(o), i, r]).
category([n,c,f,s,n, -, n], [p,i,r]).
category([n,c,m,s,g, -, y], [r,i,r]).
category([n,c,f,s,n, -, n], [s,i,r]).
category([n,c,m,s,g, -, y], [s,i,r]).
category([n,c,m,s,g, -, y], [t,i,r]).
category([n,c,f,s,n, -, n], [u,i,r]).
category([v,m,m,p,2,s, -, n], lexical([c,u,i,r])).
category([v,m,m,p,2,s, -, n], lexical([c(c), u, i, r])).
category([n,c,m,s,g, -, y], [f(u), i, r]).
category([n,c,f,s,n, -, n], [f(i), r]).
category([n,c,m,s,n, -, n], [o,r]).
category([a,q,p,-, -, [o,r]]).
category([u,n], lexical([n,f(i), o,r])).
category([n,f(i), o,r], lexical([n,f(i), o,r])).
APPENDIX A. PROGRAM CODE

category([u,n],lexical([c(n),f(i),o,r)]).
category([n,f(i),o,r],lexical([c(n),f(i),o,r])).
category([v,i,\neg,n,\neg,\neg],lexical([n,f(i),o,r])).
category([v,i,\neg,n,\neg,\neg],lexical([c(n),f(i),o,r])).
category([n,c,m,s,n,\neg,n],[f(o),r]).
category([n,c,m,s,n,\neg,n],[r,r])
category([a,q,p,\neg,\neg],[r,r])
category([],lexical([g,u,r])).
category([],lexical([c(g),u,r])).
category([d,p,2,\neg,p,\neg,\neg],lexical([b,h,u,r])).
category([d,p,2,\neg,p,\neg,\neg],lexical([c(b),h,u,r]))
category([n,c,m,s,n,\neg,n],[f(u),r])
category([m,p,\neg,\neg],lexical([d,e,i,c,h,n,i,f(u),r]))
category([m,p,\neg,\neg],lexical([c(d),e,i,c,h,n,i,f(u),r]))

% category(CategoryInfo, [s]).
category([a,s],lexical([a,s])) % simple preposition
category([a,s],lexical([c(a),s]))
category([s,p,s,n,n,0],lexical([s,a,s])) % simple preposition
category([s,p,s,n,n,0],lexical([c(a),s]))
category([s,p,s,m,s,3],lexical([s,a,s]))
category([s,p,s,m,s,3],lexical([c(a),s]))
category([r,p,b,\neg,\neg],lexical([a,s]))
category([r,p,b,\neg,\neg],lexical([c(a),s])) % simple preposition

category([s,e,a,c,h,a,s],lexical([s,e,a,c,h,a,s]))
category([s,e,a,c,h,a,s],lexical([c(s),e,a,c,h,a,s])) % simple preposition
category([s,p,s,n,n,0],lexical([s,e,a,c,h,a,s]))
category([s,p,s,n,n,0],lexical([c(s),e,a,c,h,a,s]))
category([n,c,m,s,n,\neg,n],[a,s])
category([a,q,p,\neg,\neg],[a,s])
category([],lexical([f,f(a),s]))
category([],lexical([c(f),f(a),s]))
category([n,c,m,s,n,\neg,n],[f(a),s])
category([n,c,m,\neg,\neg,n],[c,s])
category([v,i,\neg,g,\neg,\neg,\neg,\neg,i,a],lexical([i,s]))
category([v,i,\neg,g,\neg,\neg,\neg,\neg,i,a],lexical([c(i),s]))
category([n,c,f,s,n,\neg,n],[a,i,s])
category([n,c,m,s,g,\neg,y],[a,i,s])
category([n,c,m,s,g,\neg,y],[f(a),i,s]). % bás=death-GEN; ml
category([n,c,f,s,n,\neg,n],[f(a),i,s]). % cáis=cheese; f2
category([n,c,f,s,n,\neg,n],[c,i,s])
category([n,c,m,s,g,\neg,y],[c,i,s])
category([n,c,f,s,n,\neg,n],[d,i,s])
category([n,c,m,s,g,\neg,y],[d,i,s])
category([n,c,f,s,n,\neg,n],[e,i,s])
category([n,s,\neg,s,\neg,\neg,n],lexical([f(e),i,s]))
category([n,s,\neg,s,\neg,\neg,n],lexical([c(f,e),i,s]))
APPENDIX A. PROGRAM CODE

category([n,c,f,s,n,_,n],[f(e),i,s]).
category([n,c,m,s,g,_,y],[f(e),i,s]).
category([n,c,f,s,n,_,n],[f,i,s]).
category([n,c,m,s,g,_,y],[f,i,s]).
category([n,c,f,s,n,_,n],[g,i,s]).
category([n,c,f,s,n,_,n],[h,i,s]).
category([n,c,m,s,g,_,y],[h,i,s]).
category([n,c,f,s,n,_,n],[i,i,s]).
category([a,q,p,_,_,_],[l,i,s]).
category([n,c,m,s,g,_,y],[m,i,s]). % titimis=epilepsy-GEN
category([v,m,c,_,1,p,_,_],[m,i,s]).
category([v,m,i,h,1,p,_,_],[m,i,s]).
category([v,m,m,p,1,p,_,_],[m,i,s]).
category([n,c,f,s,n,_,n],[n,i,s]).
category([n,c,m,s,g,_,y],[n,i,s]).
category([],lexical([c,1,o,i,s])).
category([],lexical([c(c),1,o,i,s])).
category([n,c,f,s,n,_,n],[o,i,s]).
category([n,c,m,s,g,_,y],[f(o),i,s]).
category([n,c,f,s,n,_,n],[p,i,s]).
category([n,c,f,s,n,_,n],[r,i,s]).
category([n,c,m,p,n,_,y],[r,i,s]).
category([n,c,m,s,g,_,y],[r,i,s]).
category([n,c,f,s,n,_,n],[s,i,s]).
category([n,c,m,s,g,_,y],[s,i,s]).
category([n,c,f,s,n,_,n],[t,i,s]).
category([n,c,m,p,n,_,y],[u,i,s]).
category([n,c,m,s,g,_,y],[u,i,s]).
category([n,c,m,p,n,_,y],[f(u),i,s]).
category([n,c,m,s,g,_,y],[f(u),i,s]).
category([n,c,f,s,n,_,n],[f(i),s]).
category([v,m,c,_,2,p,_,_],[f(i),s]).
category([v,m,i,h,2,p,_,_],[f(i),s]).
category([v,m,m,p,2,p,_,_],[f(i),s]).
category([n,c,m,s,_,_,n],[n,s]).
category([s,p,s,n,n,0],lexical([o,s])).
category([s,p,s,n,n,0],lexical([c(o),s])).
category([n,c,m,s,n,_,n],[o,s]). % aos=age/agegroup
category([n,c,m,s,n,_,n],[f(o),s]).
category([n,c,m,s,_,_,n],[t,s]).
category([n,c,f,s,n,_,n],[t,s]).
category([],lexical([a,g,u,s])).
category([],lexical([c(a),g,u,s])).
category([n,c,m,s,n,_,n],[u,s]).
category([n,c,m,s,n,_,n],[f(u),s]).

%category(CategoryInfo,[t]).
category([s,p,c,−,s,2],lexical([l,e,a,tl])).
category([s,p,c,−,s,2],lexical([l,e,a,t])).
category([s,p,c,−,s,2],lexical([a,g,a,tl])).
category([s,p,c,−,s,2],lexical([a,g,a,t])).
category([s,p,c,−,s,2],lexical([c(a),g,a,tl])).
category([s,p,c,−,s,2],lexical([c(a),g,a,t])).
category([s,p,c,−,s,2],lexical([c,h,u,g,a,tl])).
category([s,p,c,−,s,2],lexical([c(h),u,g,a,t])).
category([s,p,c,−,s,2],lexical([r,o,m,h,a,tl])).
category([s,p,c,−,s,2],lexical([r,o,m,h,a,t])).
category([s,p,c,−,s,2],lexical([c(r),o,m,h,a,tl])).
category([s,p,c,−,s,2],lexical([c(r),o,m,h,a,t])).
category([s,p,c,−,s,2],lexical([i,o,n,a,tl])).
category([s,p,c,−,s,2],lexical([i,o,n,a,t])).
category([s,p,c,−,s,2],lexical([c(i),o,n,a,tl])).
category([s,p,c,−,s,2],lexical([c(i),o,n,a,t])).
category([s,p,c,−,s,2],lexical([f,a,r,a,tl])).
category([s,p,c,−,s,2],lexical([f,a,r,a,t])).
category([s,p,c,−,s,2],lexical([t,h,a,r,a,tl])).
category([s,p,c,−,s,2],lexical([t,h,a,r,a,t])).
category([s,p,c,−,s,2],lexical([a,s,a,tl])).
category([s,p,c,−,s,2],lexical([a,s,a,t])).
category([n,c,m,s,n,−,n],lexical([a,tl])).
category([n,c,m,s,n,−,n],lexical([a,t])).
category([v,m,m,p,2,s,−,n],lexical([a,tl])).
category([v,m,m,p,2,s,−,n],lexical([a,t])).
category([n,c,m,s,n,−,n],[a,t]).
category([n,c,f,s,n,−,n],[a,t]).
category([n,c,m,s,n,−,n],[f(a),t]).
category([n,c,m,s,n,−,n],[c,h,t]).
category([n,c,f,s,n,−,n],[c,h,t]).
category([n,c,m,s,n,−,n],lexical([s,e,a,c,h,tl])).
category([n,c,m,s,n,−,n],lexical([s,e,a,c,h,t])).
category([m,c,−,−,−],lexical([s,e,a,c,h,tl])).
category([m,c,−,−,−],lexical([s,e,a,c,h,t])).
category([n,c,m,s,n,−,n],[o,c,h,tl]).
category([n,c,m,s,n,−,n],[o,c,h,t]).
category([a,q,p,−,−,−],[c,h,tl]).
category([s,p,c,−,s,2],lexical([u,a,i,tl])).
category([s,p,c,−,s,2],lexical([u,a,i,t])).
category([s,p,c,−,s,2],lexical([d,u,i,tl])).
category([s,p,c,−,s,2],lexical([d,u,i,t])).
category([n,c,m,s,g,−,n],[i,tl]).
category([n,c,m,s,g,−,n],[i,t]).
category([v,m,m,p,2,s,−,n],[i,tl]).
category([v,m,m,p,2,s,−,n],[i,t]).
category([n,c,f,s,n,−,n],[f(i),t]).
category([n,c,m,s,n,−,n],[a,l,tl]).
category([n,c,m,s,n,−,n],[a,l,t]).
category([n,c,m,s,n,−,n],[f(a),i,tl]).
category([n,c,m,s,n,−,n],[f(a),i,t]).
category([n,c,m,p,n,−,y],[i,l,t]).
category([n,c,m,s,g,−,y],[i,l,t]).
category([n,c,f,s,n,−,n],[i,l,tl]).
category([n,c,f,s,n,−,n],[i,l,t]). % verbal noun
category([n,c,m,s,n,−,n],[a,n,tl]).
category([n,c,f,s,n,−,n],[i,n,tl]).
APPENDIX A. PROGRAM CODE

category([n,c,f,s,n,_,n],[n,n,t]).
category([n,c,m,s,n,_,n],[o,n,t]).
category([n,c,m,s,n,_,n],[u,n,t]).
category([s,p,c,-,s,2],lexical([d,f(i),o,t])).
category([s,p,c,-,s,2],lexical([c(d),f(i),o,t])).
category([s,p,c,-,s,2],lexical([c(t),r,f(i),o,t])).
category([n,c,m,s,n,_,n],[o,t]).
category([n,c,f,s,n,_,n],[p,t]).
category([n,c,m,s,n,_,n],[a,r,t]).
category([n,c,m,s,n,_,n],[f(a),r,t]).
category([n,c,f,s,n,_,n],[i,r,t]).
category([n,c,m,p,_,y],[i,r,t]).
category([n,c,m,s,g,_,y],[i,r,t]).
category([s,p,c,-,s,2],lexical([o,r,t])).
category([s,p,c,-,s,2],lexical([c(o),r,t])).
category([n,c,m,s,n,_,n],[o,r,t]).
category([n,c,m,s,n,_,n],[f(o),r,t]).
category([n,c,m,s,n,_,n],[a,s,t]).
category([v,m,m,p,2,s,_,n],lexical([f(e),i,s,t])).
category([v,m,m,p,2,s,_,n],lexical([c(f(e)),i,s,t])).
category([n,c,f,s,n,_,n],[i,s,t]).
category([n,c,m,p,_,y],[i,s,t]).
category([n,c,m,s,g,_,y],[i,s,t]).
category([v,m,m,p,2,s,_,n],lexical([s,m,f(i),s,t])).
category([v,m,m,p,2,s,_,n],lexical([c(s),m,f(i),s,t])).
category([v,m,m,p,2,s,_,n],lexical([t,o,s,t])).
category([v,m,m,p,2,s,_,n],lexical([c(t),o,s,t])).
category([n,c,m,s,n,_,n],[o,s,t]).
category([v,m,m,p,2,s,_,n],lexical([r,f(o),s,t])).
category([v,m,m,p,2,s,_,n],lexical([c(r),f(o),s,t])).
category([n,c,m,s,n,_,n],[u,s,t]).
category([s,p,c,-,s,2],lexical([f,f(u),t])).
category([s,p,c,-,s,2],lexical([c(f),f(u),t])).
category([n,c,m,s,n,_,n],[f(u),t]).

% category(CategoryInfo,[u]).

% Tá sás anseo inniu: he is here today
category([r,g,-,-,],lexical([i,n,n,i,u])).
category([r,g,-,-,],lexical([c(i),n,n,i,u])).
% ar maidin inniu: this morning
category([a,q,p,-,-,],lexical([i,n,n,i,u])).
category([a,q,p,-,-,],lexical([c(i),n,n,i,u])).
% prepositional pronoun 3rd person plural
category([s,p,c,-,p,3],[u]).

% category(CategoryInfo,[ú]).
category([n,c,m,s,n,_,n],[f(u)]). % verbal noun
category([n,c,m,s,-,n], [f(u)]).  % m4, su=juice
category([m,o,-,n], lexical([a,n,f(u)])).  % 1st
category([m,o,-,n], lexical([a,a,n,f(u)])).
category([m,o,-,n], lexical([d,f(o),f(u)])).  % 2nd
category([m,o,-,n], lexical([c,d,f(o),f(u)]))
category([m,o,-,n], lexical([t,r,f(i),f(u)]))  % 3rd
category([m,o,-,n], lexical([c(t),r,f(i),f(u)]))
category([m,o,-,n], lexical([c,e,a,t,h,r,f(u)]))  % 4th
category([m,o,-,n], lexical([c(c),e,a,t,h,r,f(u)]))
category([m,o,-,n], lexical([c,f(u),i,g,f(i),f(u)]))  % 5th
category([m,o,-,n], lexical([c(o),f(u),i,g,f(i),f(u)]))
category([m,o,-,n], lexical([s,f(e),f(u)]))  % 6th
category([m,o,-,n], lexical([c(s),f(e),f(u)]))
category([m,o,-,n], lexical([s,e,a,c,h,t,f(u)]))  % 7th
category([m,o,-,n], lexical([c(s),e,a,c,h,t,f(u)]))
category([m,o,-,n], lexical([o,c,h,t,f(u)]))  % 8th
category([m,o,-,n], lexical([c(o),c,h,t,f(u)]))  % 9th
category([m,o,-,n], lexical([n,a,o,f(u)]))  % 9th
category([m,o,-,n], lexical([c(n),a,o,f(u)]))  % 9th
category([m,o,-,n], lexical([d,e,i,c,h,i,f(u)]))  % 10th
category([m,o,-,n], lexical([c(d),e,i,c,h,i,f(u)]))  % 10th
category([m,o,-,n], lexical([f,i,c,h,i,f(u)]))  % 20th
category([m,o,-,n], lexical([c(f),i,c,h,i,f(u)]))  % 20th
category([m,o,-,n], lexical([t,r,f(i),o,c,h,a,d,f(u)]))  % 30th
category([m,o,-,n], lexical([c(t),r,f(i),o,c,h,a,d,f(u)]))
category([m,o,-,n], lexical([c(d),a,i,c,h,e,a,d,f(u)]))  % 40th
category([m,o,-,n], lexical([c(c),a,o,g,a,d,f(u)]))  % 50th
category([m,o,-,n], lexical([c(c),a,o,g,a,d,f(u)]))  % 50th
category([m,o,-,n], lexical([s,e,a,c,h,a,d,f(u)]))  % 60th
category([m,o,-,n], lexical([c(s),e,a,c,h,a,d,f(u)]))
category([m,o,-,n], lexical([s,e,a,c,h,t,f(o),d,f(u)]))  % 70th
category([m,o,-,n], lexical([c(s),e,a,c,h,t,f(o),d,f(u)]))
category([m,o,-,n], lexical([o,c,h,t,f(o),d,f(u)]))  % 80th
category([m,o,-,n], lexical([c(o),c,h,t,f(o),d,f(u)]))
category([m,o,-,n], lexical([n,f(o),c,h,a,d,f(u)]))  % 90th
category([m,o,-,n], lexical([c(f),e,a,d,f(u)]))  % 100th
category([m,o,-,n], lexical([c(c),f(e),a,d,f(u)]))  % 100th
category([p,p,2,-,s,n,], lexical([t,f(u)]))
category([p,p,2,-,s,n,], lexical([c(t),f(u)]))
category([p,p,2,-,s,a,], lexical([t,h,f(u)]))
category([p,p,2,-,s,a,], lexical([c(t),h,f(u)]))
category([a], lexical([b,f(u)]))  % interjection
category([a], lexical([c(b),f(u)]))
category([a], lexical([a,r,f(u)]))  % interjection
category([a], lexical([c(a),r,f(u)]))  % adjective as part of adverbial phrase
category([a,q,p,-,-,], lexical([a,r,f(u)]))
category([a,q,p,-,-,], lexical([c(a),r,f(u)]))
APPENDIX A. PROGRAM CODE

category([n,s,−,s,−,−,n],lexical([f,i,f(u)]))
category([n,s,−,s,−,−,n],lexical([c(f),i,f(u)]))
category([c,s,−,−],lexical([f,i,f(u)]))
category([c,s,−,−],lexical([c(f),i,f(u)]))
% predicative adjective with copula
category([a,i,p,−,−,−],lexical([f,i,f(u)]))
category([a,i,p,−,−,−],lexical([c(f),i,f(u)]))
category([a,q,c,−,−,−],lexical([l,f(u)])) % comparative
category([a,q,c,−,−,−],lexical([c(l),f(u)]))
category([a,q,s,−,−,−],lexical([l,f(u)])) % superlative
category([a,q,s,−,−,−],lexical([c(l),f(u)])).
A.4 utilities.pl

/***/
/***/
/**% Author: Brian Ó Raghallaigh */
/**% Date: March 2003 */
/**% */
/**% utilities.pl */
/**% A set of utility functions used by tagger.pl and grammarcheck.pl */
/**% */
/*****************************************************************************/

:- dynamic(not/1).
:- dynamic(flatten/2).
:- dynamic(append/3).

% negation as failure

not(X) :- X,!; fail.
not(_).

% flatten

flatten([],[]).
flatten([H|T],[H|FL]) :-
  atomic(H),
  flatten(T,FL).
flatten([H|T],FlatList) :-
  flatten(H,FH),
  flatten(T,FT),
  append(FH,FT,FlatList).

% append

append([],X,X).
append([H|T1],X,[H|T2]) :-
  append(T1,X,T2).

% compatible

compatible(Category,[Cat1|Cats]) :-
  equal(Category,Cat1).
compatible(Category,[Cat1|Cats]) :-
  compatible(Category,Cats).

% equal

equal(X,Y) :- unifiable(X,Y).
unifiable(X,Y) :-
    copy_term(X,X1),
    copy_term(Y,X1).

% buildcall/3
% builds a system call from Infile of the form:
% 'Program Infile.txt Infile.pl'
buildcall(Program,Infile,SystemCall) :-
    name(Program,PList),
    name(Infile,IList),
    flatten([PList|[32|[[IList|
    name(SystemCall,SCFlat).

% buildcall/4
% builds a system call from CFileOut of the form:
% 'Script Program CFileOut CFileOut.txt'
buildcall(Script,Program,Infile,SystemCall) :-
    name(Script,SList),
    name(Program,PList),
    name(Infile,IList),
    flatten([SList|[32|[PList|[46,112,108]]|
      [32|[IList|[32|[IList|[46,116,120,116]]]]]]],SCFlat),
    name(SystemCall,SCFlat).

% addextension: converts 'File' to 'File.pl'
addextension(Name,FullName) :-
    name(Name,NList),
    append(NList,[46,112,108],FName),
    name(FullName,FName).
A.5 pl2txt.pl

#!/usr/bin/perl

# the program can be run at the unix command line as follows:
# perl pl2txt.pl input.pl output.txt

# declare all variables
use strict;

open(IN, $ARGV[0]); # open the file specified in argument 0
open(OUT, "">".$ARGV[1]); # open the file specified in argument 1
my $sentence = "";

while(<IN>){ # go through each line in the input file

    # if the line follows the format of 'word([...])'
    if(/^word\[(.*)\]/){
        # $1 is the ... bit
        # split on , and return as an array of letters
        # can be a,b,c or a, b, c or cf(a), b,e etc...
        my @letters = split(/\[[ ]*, [ ]*/ $1);

        # go through each letter in the list
        foreach my $letter (@letters){
            if($letter eq "apostrophe"){
                $letter = ";";
            }

            elsif($letter eq "hyphen"){
                $letter = "-";
            }

            # if the "letter" looks like cf(c) or c(c) or f(e) or whatever
            elsif($letter =~ /([c])?([f])?\([([a-z])]\)/){
                $letter = $3;

                # give the letter a fada
                if($2 eq "f"){
                    # letter + 0x84 gives letter with fada
                    if($letter eq "a"){
                        $letter = chr(ord($letter) + 0x80);
                    }

                    elsif($letter eq "u"){
                        $letter = chr(ord($letter) + 0x85);
                    }

                    else{...}
```perl
$letter = chr(ord($letter) + 0x84);
}
}

# make the letter upper case
if($i eq "c"){
    # letter - 0x20 give capital version of letter
    $letter = chr(ord($letter) - 0x20);
}

}sentence .= $letter;
}

}elsif(/LENITION_OMITTED!/){
    my $letter = "LENITION_OMITTED! ";
    $sentence .= $letter;
}

}elsif(/ECLIPSES_OMITTED!/){
    my $letter = "ECLIPSES_OMITTED! ";
    $sentence .= $letter;
}

}elsif(/PREFIXED_H_OMITTED!/){
    my $letter = "PREFIXED_H_OMITTED! ";
    $sentence .= $letter;
}

}elsif(/UNNECESSARY_LENITION!/){
    my $letter = "UNNECESSARY_LENITION! ";
    $sentence .= $letter;
}

}elsif(/UNNECESSARY_ECLIPSES!/){
    my $letter = "UNNECESSARY_ECLIPSES! ";
    $sentence .= $letter;
}

}elsif(/MISSING_CONTRACTION!/){
    my $letter = "MISSING_CONTRACTION! ";
    $sentence .= $letter;
}

# if the line follows the format of 'punct([...])'
elsif(/punct\([\'\(.*?\)\]\])/){
    # $1 is the ... bit
    my $letter = $1;

    if($letter eq "."){
        #$sentence .= ".\n"; # put a new line as well
        $sentence .= " ";
        print OUT "$sentence"; # print it into the output file
        $sentence = " "; # new blank sentence
    }elsif($letter eq "space"){
```
```perl
$sentence .= " ";
}elsif($letter eq "newline"){
    $sentence .= "\n";
}elsif($letter eq "return"){
    $sentence .= "\r";
}elsif($letter eq "quotes"){
    $sentence .= "\\";
}elsif($letter eq "percent"){
    $sentence .= "\%";
}elsif($letter eq "apostrophe"){
    $sentence .= "\"";
}elsif($letter eq "openround"){
    $sentence .= "\";(
}elsif($letter eq "closeround"){
    $sentence .= "\")";
}elsif($letter eq "opencurly"){
    $sentence .= "\{";
}elsif($letter eq "openround"){
    $sentence .= "\}";
}elsif($letter eq "opensquare"){
    $sentence .= "\[";
}elsif($letter eq "closesquare"){
    $sentence .= "\]";
}
else{
    $sentence .= $letter; # deals with all other punctuation
}

# close files
close(IN);
close(OUT);
```
Appendix B

Parole Morphosyntactic Tagset
### B.1 Parole Common Morphosyntactic Tagset

The tables below give a full description of the part-of-speech (morpho-syntactical) mark-up used in the linguistically annotated subset of the PAROLE Irish Corpus. As the name suggests these categories and attributes are common to the 14 participating languages (Belgian & Swiss French, Catalan, Danish, Dutch, English, French, Finnish, German, Greek, Irish, Italian, Portuguese, Spanish and Swedish). All underlined items and categories in the tables marked with * are additional items specific to Irish. The tables and examples here were obtained from the website of Institúid Teangeolaíochta Éireann (‘The Linguistics Institute of Irish’) at [http://www.ite.ie/pos.htm](http://www.ite.ie/pos.htm).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>c = common</td>
<td>f = fem</td>
<td>s = sing</td>
<td>n = nom</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>p = proper</td>
<td>m = mas</td>
<td>p = pl.</td>
<td>g = gen.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>s = substantive</td>
<td></td>
<td></td>
<td>v = voc.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>d = dat.</td>
<td></td>
</tr>
</tbody>
</table>

#### 1.NOUN (-cont.)

<table>
<thead>
<tr>
<th>*7. Contrast</th>
<th>*8. Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = unmked</td>
<td>v = as verb</td>
</tr>
<tr>
<td>y = marked</td>
<td>c = as conj.</td>
</tr>
<tr>
<td>a = as adverb</td>
<td></td>
</tr>
</tbody>
</table>

Example:
<w msd=Npfg~n>Feabhra</w>
### 2. VERB

<table>
<thead>
<tr>
<th>1.POS</th>
<th>2.Type</th>
<th>3.Mood</th>
<th>4.Tense</th>
<th>5.Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>m = main</td>
<td>i = indic.</td>
<td>p = pres.</td>
<td>1 = first</td>
</tr>
<tr>
<td></td>
<td>t = tá</td>
<td>s = subj.</td>
<td>s = past</td>
<td>2 = sec.</td>
</tr>
<tr>
<td></td>
<td>i = is</td>
<td>m = imper.</td>
<td>h = past hab.</td>
<td>3 = third</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c = cond.</td>
<td>f = future</td>
<td>0 = free</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n = infinitive</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>p = participle</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>a = adjectival</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>r = relative</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 2. VERB (-cont.)

<table>
<thead>
<tr>
<th>Number</th>
<th>Gender</th>
<th>7.Dependency</th>
<th>8.Neg</th>
</tr>
</thead>
<tbody>
<tr>
<td>s = sing.</td>
<td>n/a</td>
<td>i = independant</td>
<td>Tá &amp; is</td>
</tr>
<tr>
<td>p = pl.</td>
<td></td>
<td>d = dependant</td>
<td>n = neg.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>a = affirn.</td>
</tr>
</tbody>
</table>

Example:

<w msd=Vmip→i>Fáiltíonn</w>
<w msd=Vtis→ja>bhí</w>
<w msd=Viis→ja>ba</w>

### 3. ADJECTIVE

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>q = qualifier</td>
<td>p = positive</td>
<td>f = fem.</td>
<td>s = sing</td>
<td>n = nom.</td>
<td>i = with verb 'is'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>i = with verb 'is'</td>
<td>c = comparative</td>
<td>m = masc.</td>
<td>p = pl.</td>
<td>g = gen</td>
<td>d = degree</td>
<td>v = voc.</td>
</tr>
</tbody>
</table>

Example:

<w msd=Aqp→>glan</w>
<w msd=Aip→>mór</w>
APPENDIX B. PAROLE MORPHOSYNTACTIC TAGSET

Example:
<w msd=Pp3ms__>sé</w>
<w msd=Pc3ms__>seisean</w>
<w msd=Px--__>féin</w>
<w msd=Po-s__>ceachtar</w>

Example:
<w msd=Dd--__>seó</w>
<w msd=Dp3--__>a</w>
<w msd=Dq--__>cé</w>

Example:
<w msd=Td-s__>an</w>

Example:
<w msd=Rg--__>sós</w>
<w msd=Rdb--__>holc</w>
<w msd=Rr--__>mar</w>
<w msd=Rq--__>conas</w>
<w msd=R-v__>siar</w>

Example:
<w msd=Spsmn0>le</w>
<w msd=Spe-s1>uaimse</w>
<w msd=Spxns0>sa</w>
<w msd=Spcms3>air</w>

Example:
<w msd=Cc__>agus</w>
<w msd=Cs__>go</w>
### APPENDIX B. PAROLE MORPHOSYNTACTIC TAGSET

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. POS</strong></td>
<td><strong>2. Type</strong></td>
<td><strong>3. Person</strong></td>
<td><strong>4. Gender</strong></td>
<td><strong>5. Number</strong></td>
<td><strong>6. Case</strong></td>
</tr>
<tr>
<td>P</td>
<td>p = personal</td>
<td>1 = first</td>
<td>f = fem.</td>
<td>s = sing.</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>c = contrastive</td>
<td>2 = second</td>
<td>m = masc.</td>
<td>p = pl.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x = reflexive</td>
<td>3 = third</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>o = other</td>
<td>0 = null</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>1. POS</strong></td>
<td><strong>2. Type</strong></td>
<td><strong>3. Person</strong></td>
<td><strong>4. Gender</strong></td>
<td><strong>5. Number</strong></td>
<td><strong>6. Case</strong></td>
</tr>
<tr>
<td>D</td>
<td>d = demonstrative</td>
<td>1 = first</td>
<td>f = fem.</td>
<td>s = sing.</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>p = possessive</td>
<td>2 = second</td>
<td>m = masc.</td>
<td>p = pl.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>q = interrogative</td>
<td>3 = third</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>1. POS</strong></td>
<td><strong>2. Type</strong></td>
<td><strong>3. Gender</strong></td>
<td><strong>4. Number</strong></td>
<td><strong>5. Case</strong></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>d = definite</td>
<td>f = fem.</td>
<td>s = sing</td>
<td>n = nom.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m = masc.</td>
<td>p = pl.</td>
<td></td>
<td>G = gen.</td>
<td></td>
</tr>
<tr>
<td><strong>1. POS</strong></td>
<td><strong>2. Type</strong></td>
<td><strong>3. Degree</strong></td>
<td><strong>4. Function</strong></td>
<td><strong>5. Wh-ness</strong></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>g = general</td>
<td>b = base</td>
<td>m = mod.</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>d = degree</td>
<td>c = comparative</td>
<td>s = spe.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p = phrasal</td>
<td>s = superlative</td>
<td>v = verbal</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>q = interrogative</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>r = relative</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>1. POS</strong></td>
<td><strong>2. Type</strong></td>
<td><strong>3. Formation</strong></td>
<td><strong>4. Gender</strong></td>
<td><strong>5. Number</strong></td>
<td><strong>6. Person</strong></td>
</tr>
<tr>
<td>S</td>
<td>p = preposition</td>
<td>s = simple</td>
<td>f = fem.</td>
<td>s = sing.</td>
<td>0 = null</td>
</tr>
<tr>
<td></td>
<td>c = compound</td>
<td>m = masc.</td>
<td>p = pl.</td>
<td>n = null</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e = emphatic compound</td>
<td>n = null</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>x = complex compound</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>r = complex relative compound</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>1. POS</strong></td>
<td><strong>2. Type</strong></td>
<td><strong>3. Type</strong></td>
<td><strong>4. Coord-pos</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>c = coordinate</td>
<td>n/a</td>
<td>n/a</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s = subordinative</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX B. PAROLE MORPHOSYNTACTIC TAGSET

10. NUMERALS

<table>
<thead>
<tr>
<th>1.POS</th>
<th>2.Type</th>
<th>3.Gender</th>
<th>4.Number</th>
<th>5.Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>c = cardinal</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>o = ordinal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p = personal</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:
<w msd=Cc>_trí_/w>
<w msd=Cs>_chéad_/w>
<w msd=Cs>_triúr_/w>

11. INTERJECTION

<table>
<thead>
<tr>
<th>1.POS</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
</tr>
</tbody>
</table>
Example:
<w msd=I>Ora</w>

<table>
<thead>
<tr>
<th>1.POS</th>
<th>2. Particle Type</th>
<th>3. B-Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>1 = infinitive</td>
<td>q = quotative</td>
</tr>
<tr>
<td></td>
<td>n = negative</td>
<td>I = impressionistic</td>
</tr>
<tr>
<td></td>
<td>c = continuative</td>
<td>a = almost</td>
</tr>
<tr>
<td></td>
<td>c(p) = con. passive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>q = interrogative</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a = adverbial</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r = relative</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = defective verb</td>
<td></td>
</tr>
<tr>
<td></td>
<td>v = vocative</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m = numeral</td>
<td></td>
</tr>
<tr>
<td></td>
<td>d = degree</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p = patronym</td>
<td></td>
</tr>
<tr>
<td></td>
<td>o = other</td>
<td></td>
</tr>
</tbody>
</table>

Example:
<w msd=Un>nach</w>
<w msd=Ur>a</w>
<w msd=Uv>a</w>
<w msd=Ubp>arsa</w>
<w msd=Uc>ag</w>
<w msd=Up>Ui</w>

<table>
<thead>
<tr>
<th>1.POS</th>
<th>2. Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>f = foreign</td>
</tr>
<tr>
<td></td>
<td>s = symbol</td>
</tr>
<tr>
<td></td>
<td>t = toponym</td>
</tr>
<tr>
<td></td>
<td>a = acronym</td>
</tr>
<tr>
<td></td>
<td>b = abbreviation</td>
</tr>
<tr>
<td></td>
<td>n = number</td>
</tr>
</tbody>
</table>

Example:
<w msd=Xf>chevalier</w>
<w msd=Xt>Maigh</w>
APPENDIX B. PAROLE MORPHOSYNTACTIC TAGSET

14. PUNCTUATION

<table>
<thead>
<tr>
<th>POS</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>e = sentence final</td>
</tr>
<tr>
<td></td>
<td>i = sen. Internal</td>
</tr>
<tr>
<td></td>
<td>a = quote/par init.</td>
</tr>
<tr>
<td></td>
<td>z = quote/par fin.</td>
</tr>
<tr>
<td></td>
<td>b = hyphen/underscore/dash</td>
</tr>
<tr>
<td></td>
<td>u =</td>
</tr>
<tr>
<td></td>
<td>q = ?</td>
</tr>
<tr>
<td></td>
<td>x = apostrophe</td>
</tr>
</tbody>
</table>

Example:
<w msd=Fi>;</w>
<w msd=Fb>-</w>
<w msd=Fu>!</w>

15. ABBREVIATION

<table>
<thead>
<tr>
<th>POS</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
</tr>
</tbody>
</table>

Example:
<w msd=Y>lch</w>