6.863 - Natural Language Processing

Prof. Berwick

*

Final Project:
Learning Spanish Recognizer

Kenneth Donahue and Julián Hernández Muñoz

*  

18 May 2010
Contents

1 Introduction ........................................ 2
   1.1 Background ...................................... 2
   1.2 Our Project ..................................... 2

2 Rules for Recognizing and Generating Words in Spanish .......... 3
   2.1 Lexical Structure ................................ 3
       2.1.1 Verb conjugations and verb endings .......... 3
       2.1.2 Lexicon automaton ............................. 3
   2.2 Orthographic Rules ............................... 6
       2.2.1 Orthographic changes in regular verbs ...... 6
       2.2.2 Orthographic changes in irregular verbs ..... 7
       2.2.3 Orthographic change automata ............... 7
   2.3 Recognition and Generation Testing .................... 9

3 Software ........................................... 9
   3.1 Spanish Recognizer ............................... 9
   3.2 Keyboard ......................................... 10
   3.3 State (FST) ....................................... 11
   3.4 Learning ......................................... 12

4 Conclusions .......................................... 13

5 Appendix A: Lexicon ................................ 15

6 Appendix B: Rules .................................. 25

7 Appendix C: Excerpts from Keyboard ............................ 33

8 Appendix D: FST ..................................... 34

9 Appendix E: Word Tree ................................ 39

10 Appendix F: GUI ..................................... 40
1 Introduction

1.1 Background

Lab 2 consisted of creating a recognizer/generator for Spanish words which could either take a word and parse it into its components based on lexical and orthographic rules, or instead start with the components and generate the correct surface form of the word. This was done by first systematically creating a lexical hierarchy that allowed for a flow from a word prefix to roots to suffixes, and secondly by creating a set of orthographic rules as finite state automata designed to transform a word from its lexical representation to its surface form and vice versa. Lab 2 only required a few rules to be correctly handled.

![Lexical Hierarchy](image1)

(a) Lexical Hierarchy

![Example Orthographic Rule](image2)

(b) Example Orthographic Rule (Finite State Automata)

Figure 1: Kimmo Lexical Hierarchy and Orthographic Rule Example

1.2 Our Project

For this project, we decided to extend this system such that it correctly handled more orthographic rules as well as more lexical suffixes. The specifics of these changes are discussed in the Lexical Structure and Orthographic Rules sections. As we added rules we had to make sure we were correctly handling all cases while correctly rejecting failures. Therefore, we also created a test suite system that allowed us to quickly assess changes we made to the finite state machines. This test system is discussed in the Recognition and Generation Testing section.

We additionally wanted to make the project more than just an extension of Lab 2. We decided to add a layer of software on top of the recognizer in order to handle more user input. If a word was not recognized as a word in Spanish, we wanted to offer a list of alternative words ranked by their “wrongness” and closeness to the original input. We also thought that it would be interesting if the program could learn which errors were frequently made by the user and adapt to him or her. If the user habitually made a certain typographic error or neglected a rule (e.g. c to z mutation), that error would become less wrong under supervised learning. This
part of the system is discussed in greater detail in the Software section.

2 Rules for Recognizing and Generating Words in Spanish

Our project focused in the recognition of Spanish verbs because they provide a framework with a large number of lexical suffixes and orthographic mutations that can be described using Finite State Automata. Lab 2 handled pluralization in nouns and that carried over our project. However, this discussion will only address verbs and the phenomena observed when conjugating them.

The system is composed of two parts. The first one is a lexicon automaton that describes the lexical structure of words. The second one is a set of orthographic change automata that describe possible orthographic mutations that occur when using words in their various forms. Thus, in the case of verbs, our lexicon automaton identifies the stem of the verb and the verbal suffix that specifies its tense and person. The orthographic change automata identifies any change made to the stem or tense suffix (or both) that can occur when conjugating the verb.

These two components let our system recognize the surface form of a verb and determine its lexical representation. It also lets the system use a lexical representation to generate the surface form of the verb. For example, given the input crucemos the system can recognize the verb cruzar (to cross) in its first person plural present subjunctive form. It also recognizes that the stress of the word is on the e. If instead we ask the system to generate the surface form of cruz+'emos, it will output the verb crucemos (the character ' indicates the location of the stress).

The following sections give an overview of how this system is implemented and describe the phenomena that it can handle. Baldomero Sanín Cano’s *An Elementary Spanish Grammar* served as reference for our work.

2.1 Lexical Structure

2.1.1 Verb conjugations and verb endings

Verbs in Spanish have three conjugations. They are often referred to as the first, second, and third conjugations, or the -ar, -er, and -ir conjugations. The different tenses and persons are formed by adding to the stem verbal suffixes which are common to all regular verbs belonging to the same conjugation [1].

Table 1 shows the verb endings handled by our system. It consists of 11 different tenses. Each column under a heading corresponds to each of the three conjugations. The rows (except in the infinitive, gerund, and participle tenses) correspond to first person singular, second person singular, third person singular, first person plural, second person plural, and third person plural. Different endings for the same tense and person are separated by a forward slash.

2.1.2 Lexicon automaton

The lexicon automaton begins with the recognition of the stem of the word. In our system, this can be either a noun stem or a verb stem. If the machine recognizes a noun stem it will look for a possible plural suffix or no suffix at all. If instead it recognizes a verb stem, the machine will transition to a state where it waits for verb tense suffixes. Figure 2 shows a simplified
A table of verb endings for the three conjugations in Spanish. The character ' marks a stressed vowel.

<table>
<thead>
<tr>
<th>Infinitive</th>
<th>Gerund</th>
<th>Participle</th>
</tr>
</thead>
<tbody>
<tr>
<td>'ar</td>
<td>'er</td>
<td>'ir</td>
</tr>
<tr>
<td>-o</td>
<td>-o</td>
<td>-e</td>
</tr>
<tr>
<td>-as</td>
<td>-es</td>
<td>-es</td>
</tr>
<tr>
<td>-a</td>
<td>-e</td>
<td>-e</td>
</tr>
<tr>
<td>-amos</td>
<td>-emos</td>
<td>-emos</td>
</tr>
<tr>
<td>-'ais / -an</td>
<td>-'ais / -en</td>
<td>-'ais / -en</td>
</tr>
<tr>
<td>-en</td>
<td>-en</td>
<td>-an</td>
</tr>
</tbody>
</table>

### Present Indicative

<table>
<thead>
<tr>
<th>Infinitive</th>
<th>Gerund</th>
<th>Participle</th>
</tr>
</thead>
<tbody>
<tr>
<td>-o</td>
<td>-o</td>
<td>-a</td>
</tr>
<tr>
<td>-as</td>
<td>-es</td>
<td>-as</td>
</tr>
<tr>
<td>-a</td>
<td>-e</td>
<td>-a</td>
</tr>
<tr>
<td>-amos</td>
<td>-emos</td>
<td>-amos</td>
</tr>
<tr>
<td>-'ais / -an</td>
<td>-'ais / -en</td>
<td>-'ais / -an</td>
</tr>
<tr>
<td>-en</td>
<td>-en</td>
<td>-an</td>
</tr>
</tbody>
</table>

### Present Subjunctive

<table>
<thead>
<tr>
<th>Infinitive</th>
<th>Gerund</th>
<th>Participle</th>
</tr>
</thead>
<tbody>
<tr>
<td>-o</td>
<td>-o</td>
<td>-a</td>
</tr>
<tr>
<td>-as</td>
<td>-es</td>
<td>-as</td>
</tr>
<tr>
<td>-a</td>
<td>-e</td>
<td>-a</td>
</tr>
<tr>
<td>-amos</td>
<td>-emos</td>
<td>-amos</td>
</tr>
<tr>
<td>-'ais / -an</td>
<td>-'ais / -en</td>
<td>-'ais / -en</td>
</tr>
<tr>
<td>-en</td>
<td>-en</td>
<td>-an</td>
</tr>
</tbody>
</table>

### Imperfect Indicative

<table>
<thead>
<tr>
<th>Infinitive</th>
<th>Gerund</th>
<th>Participle</th>
</tr>
</thead>
<tbody>
<tr>
<td>-'aba</td>
<td>-'ia</td>
<td>-'ia</td>
</tr>
<tr>
<td>-'abas</td>
<td>-'ias</td>
<td>-'ias</td>
</tr>
<tr>
<td>-'aba</td>
<td>-'ia</td>
<td>-'ia</td>
</tr>
<tr>
<td>-'abamos</td>
<td>-'iamos</td>
<td>-'iamos</td>
</tr>
<tr>
<td>-'abais / -'aban</td>
<td>-'ias / -'ian</td>
<td>-'ias / -'ian</td>
</tr>
<tr>
<td>-'aban</td>
<td>-'ian</td>
<td>-'ian</td>
</tr>
</tbody>
</table>

### Imperfect Subjunctive

<table>
<thead>
<tr>
<th>Infinitive</th>
<th>Gerund</th>
<th>Participle</th>
</tr>
</thead>
<tbody>
<tr>
<td>-'ara / -'ase</td>
<td>-i'era / -i'esee</td>
<td>-i'era / -i'esee</td>
</tr>
<tr>
<td>-'aras / -'ases</td>
<td>-i'eras / -i'esees</td>
<td>-i'eras / -i'esees</td>
</tr>
<tr>
<td>-'ara / -'ase</td>
<td>-i'era / -i'esee</td>
<td>-i'era / -i'esee</td>
</tr>
<tr>
<td>-'áramos / -'ásemos</td>
<td>-i'áramos / -i'ásemos</td>
<td>-i'áramos / -i'ásemos</td>
</tr>
<tr>
<td>-'aráis / -'ásepis / -'arán / -'asen</td>
<td>-i'aráis / -i'ásepis / -i'arán / -i'asen</td>
<td>-i'aráis / -i'ásepis / -i'arán / -i'asen</td>
</tr>
</tbody>
</table>

### Future Indicative

<table>
<thead>
<tr>
<th>Infinitive</th>
<th>Gerund</th>
<th>Participle</th>
</tr>
</thead>
<tbody>
<tr>
<td>-'are</td>
<td>-i're</td>
<td>-i're</td>
</tr>
<tr>
<td>-'aras</td>
<td>-i'aras</td>
<td>-i'ares</td>
</tr>
<tr>
<td>-'ará</td>
<td>-i'ará</td>
<td>-i'are</td>
</tr>
<tr>
<td>-'arémos</td>
<td>-i'arémos</td>
<td>-i'arémos</td>
</tr>
<tr>
<td>-'aréis / -'aréan</td>
<td>-i'aréis / -i'aréan</td>
<td>-i'aréis / -i'aréan</td>
</tr>
<tr>
<td>-'arán</td>
<td>-i'arán</td>
<td>-i'aren</td>
</tr>
</tbody>
</table>

### Future Subjunctive

<table>
<thead>
<tr>
<th>Infinitive</th>
<th>Gerund</th>
<th>Participle</th>
</tr>
</thead>
<tbody>
<tr>
<td>-'are</td>
<td>-i're</td>
<td>-i're</td>
</tr>
<tr>
<td>-'aras</td>
<td>-i'aras</td>
<td>-i'ares</td>
</tr>
<tr>
<td>-'ará</td>
<td>-i'ará</td>
<td>-i'are</td>
</tr>
<tr>
<td>-'arémos</td>
<td>-i'arémos</td>
<td>-i'arémos</td>
</tr>
<tr>
<td>-'aréis / -'aréan</td>
<td>-i'aréis / -i'aréan</td>
<td>-i'aréis / -i'aréan</td>
</tr>
<tr>
<td>-'arán</td>
<td>-i'arán</td>
<td>-i'aren</td>
</tr>
</tbody>
</table>

### Preterite

<table>
<thead>
<tr>
<th>Infinitive</th>
<th>Gerund</th>
<th>Participle</th>
</tr>
</thead>
<tbody>
<tr>
<td>-'aste</td>
<td>-'iste</td>
<td>-'iste</td>
</tr>
<tr>
<td>-'aste</td>
<td>-'iste</td>
<td>-'iste</td>
</tr>
<tr>
<td>-'ó</td>
<td>-ió</td>
<td>-ió</td>
</tr>
<tr>
<td>-'amos</td>
<td>-'imos</td>
<td>-'imos</td>
</tr>
<tr>
<td>-'asteis / -'aron</td>
<td>-'isteis / -'ieron</td>
<td>-'isteis / -'ieron</td>
</tr>
<tr>
<td>-'aron</td>
<td>-'ieron</td>
<td>-'ieron</td>
</tr>
</tbody>
</table>

### Conditional

<table>
<thead>
<tr>
<th>Infinitive</th>
<th>Gerund</th>
<th>Participle</th>
</tr>
</thead>
<tbody>
<tr>
<td>-'aste</td>
<td>-'iste</td>
<td>-'iste</td>
</tr>
<tr>
<td>-'aste</td>
<td>-'iste</td>
<td>-'iste</td>
</tr>
<tr>
<td>-'ó</td>
<td>-ió</td>
<td>-ió</td>
</tr>
<tr>
<td>-'amos</td>
<td>-'imos</td>
<td>-'imos</td>
</tr>
<tr>
<td>-'asteis / -'aron</td>
<td>-'isteis / -'ieron</td>
<td>-'isteis / -'ieron</td>
</tr>
<tr>
<td>-'aron</td>
<td>-'ieron</td>
<td>-'ieron</td>
</tr>
</tbody>
</table>
The transitions and states of the actual automaton are more complicated because there is a different “Verb Stem” state for each of the three conjugations. Thus, verbs can have either a -ar verb stem, a -er verb stem, or a -ir verb stem. Each of these states branches to suffix states specific to each conjugation.

To illustrate an example, the verb **hablar** has stem **habl-** and ending **-ar**, which makes it the infinitive form of a verb of the first conjugation (**-ar** conjugation). Thus, the lexical representation of **hablar** is **habl+ar**. The verb **hablo** has lexical representation **habl+o** and is the first person singular present indicative form of the verb **hablar**.

Table 1 shows that we used the character ' to mark the stressed vowel. Ending **-ar** has a stress in the **a** and ending **-o** has no stress. Furthermore, stem **habl-** can have a stress on the **a** or no stress at all, depending on the appearance of the stress in the ending. Thus, we extended the lexical representation of a word to include its stress mark. The actual lexical representations...
for hablar and hablo are:

\[
\text{hablar} \rightarrow \text{habl}+\acute{\text{a}}r \\
\text{hablo} \rightarrow \text{h}^{\acute{\text{a}}} \text{bl}+\text{o}
\]

To represent this, our lexicon automaton specifies two versions of each verb stem: one with a stress and another with out it. We also defined a finite state machine that makes sure every word has exactly one stress.

2.2 Orthographic Rules

2.2.1 Orthographic changes in regular verbs

In regular verbs, orthographic changes occur in order to preserve the sound of the final consonant of the stem. According to Sanín Cano, these changes are not considered to imply any irregularity in the formation of the tenses. He enumerates the following ten orthographic changes in regular verbs, all of which are handled by our system.

1. The final z of the stem in verbs of the first conjugation changes to c before endings beginning with e.
   - cruzar [to cross]: cruza [PresInd+1P+Sg], cruzamos [PresSubj+1P+Pl] \textit{not} *cruzamos
2. The final c of the stem in verbs of the second and third conjugation changes to z before endings beginning with a or o.
   - vencer [to defeat]: vence [PresInd+3P+Sg], venzo [PresInd+1P+Sg] \textit{not} *venco
3. The final c of the stem in the verbs of the first conjugation changes to qu before endings beginning with e.
   - sacar [to take out]: saco [PresInd+1P+Sg], saquemos [PresSubj+1P+Pl] \textit{not} *sacemos
4. The final qu of the stems in the verbs of the third conjugation becomes c before endings beginning with a or o.
   - delinquir [to commit a criminal offense]: delinque [PresInd+3P+Sg], delinco [PresInd+1P+Sg] \textit{not} *delinquo
5. The final g of the stem in verbs of the third conjugation becomes gu before endings beginning with e.
   - pagar [to pay]: pago [PresInd+1P+Sg], pague [PresSubj+1P+Sg] \textit{not} *page
6. The final gu of the stem in verbs of the third conjugation changes to g before endings beginning with a or o.
   - distinguir [to distinguish]: distingue [PresInd+3P+Sg], distingo [PresInd+1P+Sg] \textit{not} *distinguo
7. The final g of the stem in verbs of the second and third conjugation becomes j before endings beginning with a or o.
   - coger [to take]: coge [PresInd+3P+Sg], cojo [PresInd+1P+Sg] \textit{not} *cogo
8. The final **gu** of the stem in verbs of the first conjugation requires the diaeresis on the **u** before endings beginning with **e**.

**averiguar** [to find out]: averiguo [PresInd+1P+Sg], averigüe [PresSubj+1P+Sg] *not averigue*

9. Verbs whose stem ends in **ll** or **ñ** lose the unstressed initial **i** of the suffix if another vowel-sound follows.

**bullir** [to boil]: bullía [ImpInd+1P+Sg], bulló [Pret+3P+Sg] *not bullió

**gañir** [to yelp]: gañía [ImpInd+1P+Sg], gañó [Pret+3P+Sg] *not gañió

10. Verbs of the second and third conjugation whose stem ends in a vowel change to **y** the unstressed initial **i** of the suffix if another vowel-sound follows.

**leer** [to read]: leía [ImpInd+1P+Sg], leyó [Pret+3P+Sg] *not leió

### 2.2.2 Orthographic changes in irregular verbs

Irregular verbs present three different kinds of irregularities in the form of orthographical changes: new sounds can be introduced between the stem and the suffix, the stem can be altered, or the suffix can be altered. These changes are numerous and are listed by Sanín Cano in his book. Here we enumerate the ones handled by our system.

1. Some verbs ending in **acer**, **ecer**, **ocer**, and **ucir** take a **c** between the stem and the suffix after changing the **c** to **z** (Rule 2 of Section 2.2.1).

**conocer** [to know]: conoce [PresInd+3P+Sg], conozco [PresInd+1P+Sg] *not conozo*

2. Some verbs of the first and second conjugation having **e**, **o** or **u** in the last syllable of the stem change the **e** to **ie** when stressed, the **o** to **ue** when stressed, and the **u** to **ue** when stressed.

**cerrar** [to close]: cerramos [PresInd+1P+Pl], cierro [PresInd+1P+Sg] *not cerro

**mover** [to move]: movemos [PresInd+1P+Pl], muevo [PresInd+1P+Sg] *not movo

**jugar** [to play]: jugamos [PresInd+1P+Pl], juego [PresInd+1P+Sg] *not jugo

3. Some verbs of the third conjugation having **e** in the last syllable of the stem change it to **i** when stressed, or when the initial **i** of the suffix is unstressed and is followed by a vowel-sound, or when the initial letter of the suffix is a stressed vowel that is not an i.

**pedir** [to ask for]: pedimos [PresInd+1P+Pl], pido [PresInd+1P+Sg] *not pedo, pidiera [ImpSubj+1P+S] not pediera, pidamos [PresSubj+1P-Pl] not pedamos

### 2.2.3 Orthographic change automata

Table 2 summarizes the 16 orthographic changes handled by our system. The first nine changes occur at the end of the stem. The next three occur at the suffix but depend on the context at the end of the stem. The last four changes deal with vowels occurring at the last syllable of the stem but might depend on the context at the suffix.

Following these observations, our finite state machines were built keeping in mind the identification of the correct context where an orthographic changes can occur. A **z** that is not at the end of the stem should not be considered for a possible **z → c** change. Thus, our **z → c**
change | context | example
--- | --- | ---
z → c | before e | cruzar
c → z | before a and o | vencer
c → qu | before e | sacar
qu → c | before a and o | delinquir
g → gu | before e | pagar
gu → g | before a and o | distinguir
g → j | before a and o | coger
gu → gü | before e | averiguar
c → zc | before a and o | conocer
lli → ll | unstressed i before vowel | bullir
ńi → ň | unstressed i before vowel | gañir
i → y | unstressed i between two vowels | leer
e → ie | stressed e | cerrar
o → ue | stressed o | mover
u → ue | stressed u | jugar
e → i | stressed e | pedir

Table 2: Summary of the orthographic changes handled by our system.

In addition, we added special lexical characters to identify some of the possible orthographic changes. Table 3 lists these characters. The character Q indicates c hardening when it appears before a front vowel. The character J indicates g softening when it appears before a back vowel. The character V indicates u pronunciation when it appears before a front vowel. The character C indicates a c insertion when it appears before a back vowel. The characters E, O, and U indicate vowel diphthongization when stressed. The character E’ indicates e raising when stressed.

<table>
<thead>
<tr>
<th>lexical character</th>
<th>default surface pair</th>
<th>possible surface pair</th>
<th>rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>c</td>
<td>q</td>
<td>c → qu</td>
</tr>
<tr>
<td>J</td>
<td>g</td>
<td>j</td>
<td>g → j</td>
</tr>
<tr>
<td>V</td>
<td>u</td>
<td>ü</td>
<td>gu → gü</td>
</tr>
<tr>
<td>C</td>
<td>c</td>
<td>z</td>
<td>c → zc</td>
</tr>
<tr>
<td>E</td>
<td>e</td>
<td>i</td>
<td>e → ie</td>
</tr>
<tr>
<td>O</td>
<td>o</td>
<td>u</td>
<td>o → ue</td>
</tr>
<tr>
<td>U</td>
<td>u</td>
<td></td>
<td>u → ue</td>
</tr>
<tr>
<td>E’</td>
<td>e</td>
<td>i</td>
<td>e → i</td>
</tr>
</tbody>
</table>

Table 3: Special lexical characters.

A set of 11 machines handles the 16 orthographical changes. The similarity between some of the rules allowed to condense more than one rule into a single machine. In particular, c → qu and g → gu are handled by the same machine, qu → c and gu → g are handled by the same machine, lli → ll and ňi → ň are handled by the same machine, and e → ie, o → ue, and u → ue are handled by the same machine.
2.3 Recognition and Generation Testing

To test our system, we implemented a program, test.py, that takes a file with surface verbs and checks if the output of the kimmo.recognize() method matches with the corresponding lexical forms in another file. Then it takes the lexical forms and checks that the output of the kimmo.generate() method matches with the corresponding surface verbs.

Surface verb files have .surf extension and lexical verb files have .lex extension. The filename describes the test it performs and should be identical in both the surf file and its lex file. Thus, for every surf file there should be a corresponding lex file. Surface files contain all tense forms for a single verb. The lexical file contains all the corresponding lexical forms.

For example, the g-j.surf and g-j.lex files are created to test the g → j rule. File g-j.surf contains all conjugations of coger (to take) and file g-j.lex contains all the corresponding lexical forms. Conjugations and lexical forms should appear in the same order, each on a separate line. When the program runs, it goes line by line comparing the outputs of kimmo.recognize() and kimmo.generate() with what appears on the files.

We created a set of 22 tests (a total of 44 files) that makes sure our system works as expected. Three tests correspond to verbs with no orthographic change, and 16 tests to verbs that show each of the orthographic changes our system handles. The additional 3 tests correspond to verbs that show combinations of more than one orthographic change. To create these files we used as reference the online conjugator of the Real Academia Española [2].

3 Software

This section discusses the different software modules we created and how they fit into the entire system.

3.1 Spanish Recognizer

Our Spanish recognizer GUI, spanish_recognizer.py, takes a word in Spanish as the user input and attempts to call the kimmo.recognize() method to recognize the word. If recognition succeeds, the proper parse is shown in the output ListBox. If recognition fails, the user input is processed in two ways to create a list of possible alternative words, ranked by the “cost” of moving from the recognizable word to the user input. The two ways in which we processed the user input were to look for:

1. Typos (by doing single insertion, deletion, and mutation errors and attempting to recognize each possibility)
2. Rule Errors (by performing each orthographic rule forwards and backwards to create a tree of possible words)

The alternative words are put into the output ListBox. The user is then given the option to select a word from that list. If the user clicks the “Update Weights” button, the costs that were associated with that selection are all decreased. If there were elements listed above that word (i.e. words with lower scores, meaning that they were perceived to be less bad; see Figure 3), the costs of those words are increased. This should allow the selected word to appear higher on the list the next time the user makes the same mistake.
Figure 3: Words with lower cost score above the selected option need to have their costs increased once the Update button is clicked.

To determine the cost of a typo, we created the Keyboard class and the State class (which is basically a Finite State Transducer), both of which are discussed in more detail in the subsequent subsections. The process consists in taking the input word and applying to it a single insertion, deletion, or mutation to generate many other possible words. This operation is currently done by performing multiple Python string splits and could be optimized to reduce the computing time. The possible typographic error words are then transduced through the State FST along with the input word, and the cost of each of the options is calculated. This is discussed in more detail in the State (FST) section.

To determine the cost of a rule error, we defined default error costs for all rule errors (we discuss later how these error costs could also be read from the FST file). We created a tree that allowed up to three rule errors by applying each rule forwards and backwards and assigning a cost to each word. The assigned cost is the sum of the rule error costs. It is important to note that these rule errors are NOT context specific, making them different from the typographic errors. Therefore, we decided that including them in the same FST did not make sense (see the Learning section). Each node in this word-tree had a list of rules that were applied to the root word to get to that node. Outside of the tree, we kept a list of the words already in the tree to avoid redundantly adding words. We also kept a map from word to node so that we could later easily retrieve any recognized word’s node and its associated rule-path.

A good extension of this would be to search for words that combine both error types, but that would require tree pruning because it has the potential to explode. Currently, for the typos, for a word of length \( n \), there are \( 26n \) mutations, \( 26(n + 1) \) insertions, and \( n \) deletions in the first level of the tree. The next level has

\[
n(26(n - 1) + 26n + n - 1) + 26n(26n + 26(n + 1) + n) + 26(n + 1)(26(n + 1) + 26(n + 2) + n + 1)
\]

options, including duplicates. For an initial word of length 5, there are 291 options at the first level and 85,924 options at the next level.

### 3.2 Keyboard

The Keyboard class maps every letter character on a keyboard to 2-D coordinates. The default keyboard configuration is QWERTY, with the origin at the “~” character. To handle other
configurations, like DVORAK, there is an option to load in a keyboard from a file where each line is of the form:

```
MAPPING <character> <x-coordinate> <y-coordinate>
```

The keyboard class has a character-distance method that computes the euclidian distance between two characters based on their 2-D coordinates. The default distance for “out-of-vocabulary” characters was 4.0. This was used to handle any extra characters that may have been neglected in the keyboard file. To handle any insertion or deletions that may occur between the two input words we used a null character “\0”. A dictionary was used to map all possible mutations to their corresponding weights. The class was initially built to compute the distance between two words by summing the individual character distances and calculating the minimum in the case that the words were of different lengths, implying possible insertions or deletions.

We also wanted the system to learn which insertions/deletions were permissible, so there were functions to increase/decrease the weights of specific mutations. We realized that this system could not learn about the context of certain errors, meaning that it would not truly adjust to the user. We decided to instead create an State (FST) class that would allow all possible transitions and adapt to context-specific errors.

Because the Keyboard class could not learn context-dependent errors, we decided to use it solely for its character distance function so that we could feed in the initial weights to the State (FST) class.

A simple, but good extension of this module would be to handle everything in unicode instead of in ascii so that it would work for different alphabets’ characters. The change would only require changing a few lines, and it would make the module much more powerful.

### 3.3 State (FST)

The State class has a START state, which transitions to every possible letter combination state. Each letter combination state can transition to every other possible letter combination state and also transition to the END state. There is a weight associated with every transition. Because the transitions are context-specific, when the weight of a transition is updated the new cost applies only in that context. This differs from Keyboard, which would update that transition cost for every context.

The State FST can be read in from a file of the form

```
TRANSITION START <S2 a> <S2 b> <cost of trans from START to state S2>
TRANSITION <S1 a> <S1 b> <S2 a> <S2 b> <cost of trans from state S1 to state S2>
TRANSITION <S1 a> <S1 b> END <cost of trans from state S1 to END>
```

Any line that does not begin with the TRANSITION tag is ignored, so other information, like the cost of rule errors can also be put into the file using a different tagging system.

The State class’s main methods include the transduce and edit_costs methods. The transduce method takes two input strings and computes the cost of that path through the FST. In the case that the input strings are of different length, it inserts EPSILON transitions and computes the minimum cost. The output of this method is a tuple of word a (with inserted EPSILONs if necessary), word b (with inserted EPSILONs if necessary), and the cost of the path.
through the FST using those words. This method is useful in determining how bad a typo is from its corresponding correct word.

The edit_costs method is used to either reduce or increase the cost of transitions along the FST path that was determined by the two input words. It is used for learning so that if a typo is selected by the user to be correct, the FST can adapt and learn which typo errors are more permissible than others. Using this method, it is possible to slowly mutate the machine to reflect the user’s habits. If the user has a habit of accidentally hitting the s key after hitting the d, the machine will recognize this error in context and decrease its weight so that this error slowly becomes less bad. When we have many typo options in the GUI, we are the better able to rank them in a very user-specific way.

3.4 Learning

The FST can be written to and read from a file. This allows us to store the user’s settings and load them in on command. When a user selects a word as a correction for the word that was initially not recognizable, the system grabs that selection and see if it is a RULE_ERROR or a TYPO_ERROR. If it is a typo error, it uses the FST’s reduce_weights method. This method is similar to the transduce method, but instead of transducing and summing the weights, it walks through the FST and updates the weights of all the transitions by multiplying each weight by DEFAULT_WEIGHT_REDUCER. If the selection was a rule error, it instead grabs the corresponding leaf in the rule error tree and retrieves its rule-path through the tree. Then, for each of the rules in the rule-path, it multiplies the weight of the rule by DEFAULT_WEIGHT_REDUCER.

Once that is all done, the system grabs each element in the ListBox above the user’s selection and repeat the process, except instead of decreasing the weights, it increases the weights. (See Figure 3).

In this way we are able to have an alternative word that may have initially had a bad weight show up with a much less bad weight on the next iteration. An example of this is shown in Figure 4.

We should reiterate the fact that the typo errors are in the FST because they are (usually) very context specific, whereas the rule errors are more holistic and less context specific. This
may have not been the best assumption, and a possible future extension could be to look into how to add the rules into the FST such that context-specific rule violations are handled properly and the program better adapts to the user.

Another possible extension would be to automatically determine the proper weight updating factors (DEFAULT_WEIGHT_REDUCER and DEFAULT_WEIGHT_INCREASER) instead of having it be arbitrarily set. By having this weight be more dynamic, the learning could be optimized to occur at a much faster rate.

4 Conclusions

We set out to create a module that had the ability to recognize and generate many words in Spanish, covering many tenses, endings, insertions, deletions, and mutations. In order to do this, we added extensively to Lab 2’s lexical structure by including all verb tense suffixes for -ar, -er, and -ir conjugations. We also added extensively to the orthographic finite state automata to handle a total of 16 orthographic phenomena that occur when conjugating verbs. In order to test that these rules did not interfere with one another, we created a test suite composed of 22 tests with 22 different verbs. These tests make sure the recognition and generation of verbs works as expected.

In order to handle words that were not initially recognized by the system, we created an interface that would generate a list of alternative words based on the user input. Words were generated by considering possible mutations, insertions, or deletions, and by applying each orthographic rule forwards and backwards. After selecting from the generated list the words that could be recognized by the system, we ranked them based on distance from the input word and weight of the errors, and then we displayed the results in the spanish_recognizer’s interface. We then could take the user selection of the word he or she meant to type in and adapt the error weights in order to make the system learn and adapt to the user.

There are many possible extensions to this project which have been enumerated in the relevant sections. We feel that the most important future work could be done in

1. extending the orthographic rules to cover more orthographic changes that occur in irregular verbs,

2. merging the typo and rule error systems to allow for words with both types of errors, and

3. having a more dynamic weight update system so the system adapts to the user faster.
References


5 Appendix A: Lexicon

; Definition of a Spanish lexicon automaton.
; Kenny Donahue
; Julian Hernandez

;------------------------
; Alternation states
;------------------------

Begin: Root
Root: NOUN_ROOT VERB_ROOT_AR VERB_ROOT_ER VERB_ROOT_IR

; Transitions after noun roots
AfterNoun: PluralSuffix End

; Plural suffix transition
PluralSuffix: PLURAL

; Transitions after plural suffix
AfterPlural: End

; Transitions after verb roots
AfterVerbAr: VerbTenseSuffixAr
AfterVerbEr: VerbTenseSuffixEr
AfterVerbIr: VerbTenseSuffixIr

; Verb tense transitions
VerbTenseSuffixAr: INFINITIVE_AR PRESENT_INDICATIVE_AR PRESENT_SUBJUNCTIVE_AR
IMPERFECT_INDICATIVE_AR IMPERFECT_SUBJUNCTIVE_AR FUTURE_INDICATIVE_AR
FUTURE_SUBJUNCTIVE_AR PRETERITE_AR CONDITIONAL_AR GERUND_AR PARTICIPLE_AR

VerbTenseSuffixEr: INFINITIVE_ER PRESENT_INDICATIVE_ER PRESENT_SUBJUNCTIVE_ER
IMPERFECT_INDICATIVE_ER IMPERFECT_SUBJUNCTIVE_ER FUTURE_INDICATIVE_ER
FUTURE_SUBJUNCTIVE_ER PRETERITE_ER CONDITIONAL_ER GERUND_ER PARTICIPLE_ER

VerbTenseSuffixIr: INFINITIVE_IR PRESENT_INDICATIVE_IR PRESENT_SUBJUNCTIVE_IR
IMPERFECT_INDICATIVE_IR IMPERFECT_SUBJUNCTIVE_IR FUTURE_INDICATIVE_IR
FUTURE_SUBJUNCTIVE_IR PRETERITE_IR CONDITIONAL_IR GERUND_IR PARTICIPLE_IR

; Transitions after verb tenses
AfterVerbTenseSuffix: End

;------------------------
; Noun lexical states
;------------------------

NOUN_ROOT: l’a’piz AfterNoun Noun(la’piz)
ciud'ad  AfterNoun  Noun(ciudad)
b'ota  AfterNoun  Noun(bota)
capat'az  AfterNoun  Noun(capataz)
rel'oj  AfterNoun  Noun(reloj)

PLURAL:
+s  AfterPlural  +Pl

---------------------

; Verb lexical states ;
---------------------

VERB_ROOT_AR:

h'abl  AfterVerbAr  Verb(hablar)
habl  AfterVerbAr  Verb(hablar)
l1'eg  AfterVerbAr  Verb(llegar)
lleg  AfterVerbAr  Verb(llegar)
p'ag  AfterVerbAr  Verb(pagar)
pag  AfterVerbAr  Verb(pagar)
cr'uz  AfterVerbAr  Verb(cruzar)
cruz  AfterVerbAr  Verb(cruzar)
germin  AfterVerbAr  Verb(germinar)
germin  AfterVerbAr  Verb(germinar)
at'ac  AfterVerbAr  Verb(atacar)
atac  AfterVerbAr  Verb(atacar)
alc'anz  AfterVerbAr  Verb(alcanzar)
alcanz  AfterVerbAr  Verb(alcanzar)
s'aQ  AfterVerbAr  Verb(sacar)
saQ  AfterVerbAr  Verb(sacar)
c'Err  AfterVerbAr  Verb(cerrar)
cErr  AfterVerbAr  Verb(cerrar)
j'Ug  AfterVerbAr  Verb(jugar)
jUg  AfterVerbAr  Verb(jugar)
aver'igV  AfterVerbAr  Verb(averiguar)
averigV  AfterVerbAr  Verb(averiguar)

VERB_ROOT_ER:

c'om  AfterVerbEr  Verb(comer)
com  AfterVerbEr  Verb(comer)
c'oJ  AfterVerbEr  Verb(coger)
coJ  AfterVerbEr  Verb(coger)
prot'eJ  AfterVerbEr  Verb(proteger)
proteJ  AfterVerbEr  Verb(proteger)
t'ej  AfterVerbEr  Verb(tejer)
tej  AfterVerbEr  Verb(tejer)
con'oC  AfterVerbEr  Verb(conocer)
conoC  AfterVerbEr  Verb(conocer)
par'eC  AfterVerbEr  Verb(parecer)
pareC  AfterVerbEr  Verb(parecer)

16
<table>
<thead>
<tr>
<th>Verb</th>
<th>Prefix</th>
<th>Base</th>
<th>Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>compl'aC</td>
<td>AfterVerbEr</td>
<td>Verb</td>
<td>complacer</td>
</tr>
<tr>
<td>complaC</td>
<td>AfterVerbEr</td>
<td>Verb</td>
<td>complacer</td>
</tr>
<tr>
<td>c'Oc</td>
<td>AfterVerbEr</td>
<td>Verb</td>
<td>cocer</td>
</tr>
<tr>
<td>c0c</td>
<td>AfterVerbEr</td>
<td>Verb</td>
<td>cocer</td>
</tr>
<tr>
<td>v'enc</td>
<td>AfterVerbEr</td>
<td>Verb</td>
<td>vencer</td>
</tr>
<tr>
<td>venc</td>
<td>AfterVerbEr</td>
<td>Verb</td>
<td>vencer</td>
</tr>
<tr>
<td>ej'erc</td>
<td>AfterVerbEr</td>
<td>Verb</td>
<td>ejercer</td>
</tr>
<tr>
<td>ejerc</td>
<td>AfterVerbEr</td>
<td>Verb</td>
<td>ejercer</td>
</tr>
<tr>
<td>conv'enC</td>
<td>AfterVerbEr</td>
<td>Verb</td>
<td>convencer</td>
</tr>
<tr>
<td>convenC</td>
<td>AfterVerbEr</td>
<td>Verb</td>
<td>convencer</td>
</tr>
<tr>
<td>l'e</td>
<td>AfterVerbEr</td>
<td>Verb</td>
<td>leer</td>
</tr>
<tr>
<td>le</td>
<td>AfterVerbEr</td>
<td>Verb</td>
<td>leer</td>
</tr>
<tr>
<td>m'Ov</td>
<td>AfterVerbEr</td>
<td>Verb</td>
<td>mover</td>
</tr>
<tr>
<td>mOv</td>
<td>AfterVerbEr</td>
<td>Verb</td>
<td>mover</td>
</tr>
</tbody>
</table>

**VERB_ROOT_IR:**

<table>
<thead>
<tr>
<th>Verb</th>
<th>Prefix</th>
<th>Base</th>
<th>Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>v'iv</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>vivir</td>
</tr>
<tr>
<td>viv</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>vivir</td>
</tr>
<tr>
<td>introduC</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>introducir</td>
</tr>
<tr>
<td>introduC</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>introducir</td>
</tr>
<tr>
<td>afl'iJ</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>afligir</td>
</tr>
<tr>
<td>afliiJ</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>afligir</td>
</tr>
<tr>
<td>f'inJ</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>fingir</td>
</tr>
<tr>
<td>finJ</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>fingir</td>
</tr>
<tr>
<td>d'Orm</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>dormir</td>
</tr>
<tr>
<td>dOrm</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>dormir</td>
</tr>
<tr>
<td>del'inqu</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>delinquir</td>
</tr>
<tr>
<td>delinqu</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>delinquir</td>
</tr>
<tr>
<td>dist'ingu</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>distinguir</td>
</tr>
<tr>
<td>distingu</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>distinguir</td>
</tr>
<tr>
<td>b'ull</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>bullir</td>
</tr>
<tr>
<td>bull</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>bullir</td>
</tr>
<tr>
<td>g'an~</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>gan~ir</td>
</tr>
<tr>
<td>gan~</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>gan~ir</td>
</tr>
<tr>
<td>p'E'd</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>pedir</td>
</tr>
<tr>
<td>pE'd</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>pedir</td>
</tr>
<tr>
<td>s'E'gu</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>seguir</td>
</tr>
<tr>
<td>sE'gu</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>seguir</td>
</tr>
<tr>
<td>c'E'n~</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>cen~ir</td>
</tr>
<tr>
<td>cE'n~</td>
<td>AfterVerbIr</td>
<td>Verb</td>
<td>cen~ir</td>
</tr>
</tbody>
</table>

**INFinitive_AR:**

+’ar | AfterVerbTenseSuffix | +Inf

**INFinitive_ER:**
+'er AfterVerbTenseSuffix +Inf

INFINITIVE_IR:
+'ir AfterVerbTenseSuffix +Inf

;---------------------;
; Gerund             ;
;---------------------;

GERUND_AR:
+'ando AfterVerbTenseSuffix +Ger

GERUND_ER:
+i'endo AfterVerbTenseSuffix +Ger

GERUND_IR:
+i'endo AfterVerbTenseSuffix +Ger

;---------------------;
; Participle         ;
;---------------------;

PARTICIPLE_AR:
+'ado AfterVerbTenseSuffix +Part

PARTICIPLE_ER:
+'ido AfterVerbTenseSuffix +Part

PARTICIPLE_IR:
+'ido AfterVerbTenseSuffix +Part

;---------------------;
; Present Indicative ;
;---------------------;

PRESENT_INDICATIVE_AR:
+o  AfterVerbTenseSuffix  +PresInd+1P+Sg
+as AfterVerbTenseSuffix  +PresInd+2P+Sg
+a  AfterVerbTenseSuffix  +PresInd+3P+Sg
+'amos AfterVerbTenseSuffix  +PresInd+1P+Pl
+an AfterVerbTenseSuffix  +PresInd+2P+Pl
+'a'is AfterVerbTenseSuffix  +PresInd+2P+Pl
+an AfterVerbTenseSuffix  +PresInd+3P+Pl

PRESENT_INDICATIVE_ER:
+o  AfterVerbTenseSuffix  +PresInd+1P+Sg
+es AfterVerbTenseSuffix  +PresInd+2P+Sg
+e  AfterVerbTenseSuffix  +PresInd+3P+Sg
+'emos AfterVerbTenseSuffix  +PresInd+1P+Pl
PRESENT_INDICATIVE_IR:
+en AfterVerbTenseSuffix +PresInd+2P+Pl
+'e’is AfterVerbTenseSuffix +PresInd+2P+Pl
+en AfterVerbTenseSuffix +PresInd+3P+Pl

PRESENT_SUBJUNCTIVE_AR:
+e AfterVerbTenseSuffix +PresSubj+1P+Sg
+es AfterVerbTenseSuffix +PresSubj+2P+Sg
+e AfterVerbTenseSuffix +PresSubj+3P+Sg
+'emos AfterVerbTenseSuffix +PresSubj+1P+Pl
+en AfterVerbTenseSuffix +PresSubj+2P+Pl
+'i’s AfterVerbTenseSuffix +PresSubj+2P+Pl
+en AfterVerbTenseSuffix +PresSubj+3P+Pl

PRESENT_SUBJUNCTIVE_ER:
+a AfterVerbTenseSuffix +PresSubj+1P+Sg
+as AfterVerbTenseSuffix +PresSubj+2P+Sg
+a AfterVerbTenseSuffix +PresSubj+3P+Sg
+'amos AfterVerbTenseSuffix +PresSubj+1P+Pl
+an AfterVerbTenseSuffix +PresSubj+2P+Pl
+'a’is AfterVerbTenseSuffix +PresSubj+2P+Pl
+an AfterVerbTenseSuffix +PresSubj+3P+Pl

PRESENT_SUBJUNCTIVE_IR:
+a AfterVerbTenseSuffix +PresSubj+1P+Sg
+as AfterVerbTenseSuffix +PresSubj+2P+Sg
+a AfterVerbTenseSuffix +PresSubj+3P+Sg
+'amos AfterVerbTenseSuffix +PresSubj+1P+Pl
+an AfterVerbTenseSuffix +PresSubj+2P+Pl
+'a’is AfterVerbTenseSuffix +PresSubj+2P+Pl
+an AfterVerbTenseSuffix +PresSubj+3P+Pl

IMPERFECT_INDICATIVE_AR:
Prof. Berwick  
Natural Language Processing  
kennyd, jchernan  
6.863  
Final Project - Learning Spanish Recognizer  
18 May 2010  

+'aba AfterVerbTenseSuffix +ImpInd+1P+Sg  
+'abas AfterVerbTenseSuffix +ImpInd+2P+Sg  
+'aba AfterVerbTenseSuffix +ImpInd+3P+Sg  
+'a‘bamos AfterVerbTenseSuffix +ImpInd+1P+Pl  
+'aban AfterVerbTenseSuffix +ImpInd+2P+Pl  
+'abais AfterVerbTenseSuffix +ImpInd+2P+Pl  
+'aban AfterVerbTenseSuffix +ImpInd+3P+Pl  

IMPERFECT_INDICATIVE_ER:  
+'i'a AfterVerbTenseSuffix +ImpInd+1P+Sg  
+'i’as AfterVerbTenseSuffix +ImpInd+2P+Sg  
+'i’a AfterVerbTenseSuffix +ImpInd+3P+Sg  
+'i’amos AfterVerbTenseSuffix +ImpInd+1P+Pl  
+'i’an AfterVerbTenseSuffix +ImpInd+2P+Pl  
+'i’ais AfterVerbTenseSuffix +ImpInd+2P+Pl  
+'i’an AfterVerbTenseSuffix +ImpInd+3P+Pl  

IMPERFECT_INDICATIVE_IR:  
+'i’a AfterVerbTenseSuffix +ImpInd+1P+Sg  
+'i’as AfterVerbTenseSuffix +ImpInd+2P+Sg  
+'i’a AfterVerbTenseSuffix +ImpInd+3P+Sg  
+'i’amos AfterVerbTenseSuffix +ImpInd+1P+Pl  
+'i’an AfterVerbTenseSuffix +ImpInd+2P+Pl  
+'i’ais AfterVerbTenseSuffix +ImpInd+2P+Pl  
+'i’an AfterVerbTenseSuffix +ImpInd+3P+Pl  

;--------------------------;  
; Imperfect Subjunctive ;  
;--------------------------;  

IMPERFECT_SUBJUNCTIVE_AR:  
+'ara AfterVerbTenseSuffix +ImpSubj+1P+Sg  
+'aras AfterVerbTenseSuffix +ImpSubj+2P+Sg  
+'ara AfterVerbTenseSuffix +ImpSubj+3P+Sg  
+'a’ramos AfterVerbTenseSuffix +ImpSubj+1P+Pl  
+'aran AfterVerbTenseSuffix +ImpSubj+2P+Pl  
+'arais AfterVerbTenseSuffix +ImpSubj+2P+Pl  
+'aran AfterVerbTenseSuffix +ImpSubj+3P+Pl  
+'ase AfterVerbTenseSuffix +ImpSubj+1P+Sg  
+'ases AfterVerbTenseSuffix +ImpSubj+2P+Sg  
+'ase AfterVerbTenseSuffix +ImpSubj+3P+Sg  
+'a’semos AfterVerbTenseSuffix +ImpSubj+1P+Pl  
+'asen AfterVerbTenseSuffix +ImpSubj+2P+Pl  
+'aseis AfterVerbTenseSuffix +ImpSubj+2P+Pl  
+'asen AfterVerbTenseSuffix +ImpSubj+3P+Pl  

IMPERFECT_SUBJUNCTIVE_ER:  
+i‘era AfterVerbTenseSuffix +ImpSubj+1P+Sg  
+i’eras AfterVerbTenseSuffix +ImpSubj+2P+Sg  

20
+i'era AfterVerbTenseSuffix +ImpSubj+3P+Sg
+i'e'ramos AfterVerbTenseSuffix +ImpSubj+1P+Pl
+i'erasen AfterVerbTenseSuffix +ImpSubj+2P+Pl
+i'erais AfterVerbTenseSuffix +ImpSubj+2P+Pl
+i'e'ese AfterVerbTenseSuffix +ImpSubj+3P+Pl
+i'esen AfterVerbTenseSuffix +ImpSubj+3P+Sg
+i'e'semos AfterVerbTenseSuffix +ImpSubj+1P+Pl
+i'esen AfterVerbTenseSuffix +ImpSubj+2P+Pl
+i'eseis AfterVerbTenseSuffix +ImpSubj+2P+Pl
+i'esen AfterVerbTenseSuffix +ImpSubj+3P+Pl

IMPERFECT_SUBJUNCTIVE_IR:
+i'era AfterVerbTenseSuffix +ImpSubj+1P+Sg
+i'eras AfterVerbTenseSuffix +ImpSubj+2P+Sg
+i'era AfterVerbTenseSuffix +ImpSubj+1P+Sg
+i'e'ramos AfterVerbTenseSuffix +ImpSubj+1P+Pl
+i'esen AfterVerbTenseSuffix +ImpSubj+1P+Sg
+i'esen AfterVerbTenseSuffix +ImpSubj+1P+Sg
+i'e'semos AfterVerbTenseSuffix +ImpSubj+1P+Pl
+i'esen AfterVerbTenseSuffix +ImpSubj+2P+Pl
+i'esen AfterVerbTenseSuffix +ImpSubj+2P+Pl
+i'esen AfterVerbTenseSuffix +ImpSubj+3P+Pl

;---------------------;
; Future             ;
;---------------------;

FUTURE_INDICATIVE_AR:
+ar'e' AfterVerbTenseSuffix +Fut+1P+Sg
+ar'a's AfterVerbTenseSuffix +Fut+2P+Sg
+ar'a' AfterVerbTenseSuffix +Fut+3P+Sg
+ar'emos AfterVerbTenseSuffix +Fut+1P+Pl
+ar'a'n AfterVerbTenseSuffix +Fut+2P+Pl
+ar'e'is AfterVerbTenseSuffix +Fut+2P+Pl
+ar'a'n AfterVerbTenseSuffix +Fut+3P+Pl

FUTURE_INDICATIVE_ER:
+er'e' AfterVerbTenseSuffix +Fut+1P+Sg
+er'a's AfterVerbTenseSuffix +Fut+2P+Sg
+er'a' AfterVerbTenseSuffix +Fut+3P+Sg
+er'emos AfterVerbTenseSuffix +Fut+1P+Pl
+er'a'n AfterVerbTenseSuffix +Fut+2P+Pl
+er'e'is AfterVerbTenseSuffix +Fut+2P+Pl

21
AfterVerbTenseSuffix +Fut+3P+Pl

FUTURE_INDICATIVE_IR:
+ir’e’ AfterVerbTenseSuffix +Fut+1P+Sg
+ir’a’s AfterVerbTenseSuffix +Fut+2P+Sg
+ir’a’ AfterVerbTenseSuffix +Fut+3P+Sg
+ir’emos AfterVerbTenseSuffix +Fut+1P+Pl
+ir’e’is AfterVerbTenseSuffix +Fut+2P+Pl
+ir’a’n AfterVerbTenseSuffix +Fut+3P+Pl

FUTURE_SUBJUNCTIVE_AR:
+i’are AfterVerbTenseSuffix +FutSubj+1P+Sg
+i’ares AfterVerbTenseSuffix +FutSubj+2P+Sg
+i’are AfterVerbTenseSuffix +FutSubj+3P+Sg
+i’a’remos AfterVerbTenseSuffix +FutSubj+1P+Pl
+i’aren AfterVerbTenseSuffix +FutSubj+2P+Pl
+i’areis AfterVerbTenseSuffix +FutSubj+2P+Pl
+i’aren AfterVerbTenseSuffix +FutSubj+3P+Pl

FUTURE_SUBJUNCTIVE_ER:
+i’e’re AfterVerbTenseSuffix +FutSubj+1P+Sg
+i’e’rees AfterVerbTenseSuffix +FutSubj+2P+Sg
+i’e’re AfterVerbTenseSuffix +FutSubj+3P+Sg
+i’e’remos AfterVerbTenseSuffix +FutSubj+1P+Pl
+i’e’eren AfterVerbTenseSuffix +FutSubj+2P+Pl
+i’e’reis AfterVerbTenseSuffix +FutSubj+2P+Pl
+i’e’eren AfterVerbTenseSuffix +FutSubj+3P+Pl

PRETERITE_AR:
+e’ AfterVerbTenseSuffix +Pret+1P+Sg
+aste AfterVerbTenseSuffix +Pret+2P+Sg
+'o' AfterVerbTenseSuffix Pret+3P+Sg
+'amos AfterVerbTenseSuffix Pret+1P+P1
+'aron AfterVerbTenseSuffix Pret+2P+P1
+'asteis AfterVerbTenseSuffix Pret+2P+P1
+'aron AfterVerbTenseSuffix Pret+3P+P1

PRETERITE_ER:
+'i' AfterVerbTenseSuffix Pret+1P+Sg
+'iste AfterVerbTenseSuffix Pret+2P+Sg
+i'o' AfterVerbTenseSuffix Pret+3P+Sg
+'imos AfterVerbTenseSuffix Pret+1P+P1
+i'erone AfterVerbTenseSuffix Pret+2P+P1
+i'istei AfterVerbTenseSuffix Pret+2P+P1
+i'erone AfterVerbTenseSuffix Pret+3P+P1

PRETERITE_IR:
+'i' AfterVerbTenseSuffix Pret+1P+Sg
+'iste AfterVerbTenseSuffix Pret+2P+Sg
+i'o' AfterVerbTenseSuffix Pret+3P+Sg
+'imos AfterVerbTenseSuffix Pret+1P+P1
+i'erone AfterVerbTenseSuffix Pret+2P+P1
+i'istei AfterVerbTenseSuffix Pret+2P+P1
+i'erone AfterVerbTenseSuffix Pret+3P+P1

CONDITIONAL_AR:
+ar'i'a AfterVerbTenseSuffix Cond+1P+Sg
+ar'i'as AfterVerbTenseSuffix Cond+2P+Sg
+ar'i'a AfterVerbTenseSuffix Cond+3P+Sg
+ar'i'amos AfterVerbTenseSuffix Cond+1P+P1
+ar'i'an AfterVerbTenseSuffix Cond+2P+P1
+ar'i'ais AfterVerbTenseSuffix Cond+2P+P1
+ar'i'an AfterVerbTenseSuffix Cond+3P+P1

CONDITIONAL_ER:
+er'i'a AfterVerbTenseSuffix Cond+1P+Sg
+er'i'as AfterVerbTenseSuffix Cond+2P+Sg
+er'i'a AfterVerbTenseSuffix Cond+3P+Sg
+er'i'amos AfterVerbTenseSuffix Cond+1P+P1
+er'i'an AfterVerbTenseSuffix Cond+2P+P1
+er'i'ais AfterVerbTenseSuffix Cond+2P+P1
+er'i'an AfterVerbTenseSuffix Cond+3P+P1

CONDITIONAL_IR:
+ir'i'a AfterVerbTenseSuffix Cond+1P+Sg
+ir'i'as AfterVerbTenseSuffix Cond+2P+Sg
+ir’i’a AfterVerbTenseSuffix  +Cond+3P+Sg
+ir’i’amos AfterVerbTenseSuffix  +Cond+1P+Pl
+ir’i’an AfterVerbTenseSuffix  +Cond+2P+Pl
+ir’i’ais AfterVerbTenseSuffix  +Cond+2P+Pl
+ir’i’an AfterVerbTenseSuffix  +Cond+3P+Pl

; This part is required to deal gracefully with the boundary symbol.
End:
'##' End None
6 Appendix B: Rules

# Definition of Spanish spelling change automata
# Kenny Donahue
# Julian Hernandez

boundary: ' '#
lexicon: spanish.lex
defaults: "a e i o u a' e' i' o' u' u; b c d f g h j k l m n n~ p q r s t v w x y z U:u E:e E':e 0:o J:g C:c Q:q V:v u +:0 # ':0 -"
subsets:
  "CONS": "b c d f g h j k l m n n~ p q r s t v w x y z J C Q"
  "STEMS": "O U E"
  "VOC": "a a' e e' i i' o o' u u' STEMS"
  "FRONT": "e i e' i'"
  "BACK": "a a' o o' u u'"
  "LOW": "e o o a a' e' o'"
  "HIGH": "i i' u u'"
  "@": "a e i o u a' e' i' o' u' u; b c d f g h j k l m n n~ p q r s t v w x y z J C Q O U E E' V + ' # - 0"
  "ALL": "@ 0"
rules:

force one stress: |
FSA
  ' # @
  0 # @
  1: 2 0 1
  2. 0 1 2

z->z | z->c | z to c mutation :
start:
  'z': found_z
  'z:c': found_c
  '@': start
found_z:
  '+'0': z_state
  '@': start
found_c:
  '+'0': c_state
  '@': reject
z_state:
  ':0': z_state
  '@:FRONT': reject
  '@': start
c_state:
  ':0': c_state
  '@:FRONT': start
'@': reject

c->c | c->z | c to z mutation:
    start:
        'c:c': found_c
        'c:z': found_z
        'C:c': found_c
        'C:z': found_zc
        '@': start
    found_c:
        '+:0': c_state
        '@': start
    found_z:
        '+:0': z_state
        '@': reject
    found_zc:
        '0:c': found_zc
        '+:0': z_state
        '@': start
    c_state:
        '':0': c_state
        '@:BACK': reject
        '@': start
    z_state:
        '':0': z_state
        '@:BACK': start
        '@': reject

c -> c | c -> qu | g->g | g->gu | u insertion :
    start:
        '0:u': reject
        'Q:c': found_c
        'Q:q': found_q
        'g': found_g
        '@': start
    found_c:
        '+:0': c_state
        '0:u': reject
        '@': start
    found_q:
        '0:u': u_inserted
        '@': reject
    found_g:
        '0:u': u_inserted
        '+:0': g_state
        '@': start
    c_state:
Prof. Berwick
Natural Language Processing
6.863 Final Project - Learning Spanish Recognizer
kennyd, jchernan
18 May 2010

'':0': c_state
'@:FRONT': reject
'0:u': reject
'@': start

g_state:
'':0': g_state
'FRONT': reject
'0:u': reject
'@': start

u_inserted:
'+:0': u_state
'0:u': reject
'@': reject

u_state:
'':0': u_state
'FRONT': start
'0:u': reject
'@': reject

qu->qu | qu->c | gu->gu | gu->g :

start:
'q:q': found_q
'q:c': found_c
'g': found_g
'u:0': reject
'@': start

found_q:
'u': found_qu_gu
'u:0': reject
'@': start

found_qu_gu:
'+:0': qu_state
'u:0': reject
'@': start

found_c:
'u:0': u_deleted
'@': reject

found_g:
'u': found_qu_gu
'u:0': u_deleted
'FRONT': reject
'@': start

qu_state:
'':0': qu_state
'BACK': reject
'u:0': reject
'@': start

u_deleted:

27
'+;0': u_state
'u;0': reject
'@': start
u_state:
'+';0': u_state
'BACK': start
'u;0': reject
'@': reject

\(gu \rightarrow gu \mid gu \rightarrow gu;\) :

start:
'g': found_g
'@': start
found_g:
'V:u': found_u
'V:u;': found_u;
'@': start
found_u:
'+';0': u_state
'@': start
found_u;:
'+';0': u;_state
'@': start
u;_state:
'+';0': u;_state
'FRONT': reject
'@': start
u;_state:
'+';0': u;_state
'FRONT': start
'@': reject

\(g \rightarrow g \mid g \rightarrow j\) :

start:
'J:g': found_g
'J:j': found_j
'@': start
found_g:
'+;0': g_state
'@': start
found_j:
'+;0': j_state
'@': start
g_state:
'+';0': g_state
'FRONT': start
'@': reject
j_state:
  `'':0': j_state
  'FRONT': reject
  '@': start

z->zc | c insertion :
  start:
  'C': found_z
  '0': reject
  '@': start
  found_z:
  '0': start
  '@': reject

lli->ll | lli->lli | n~i->n~ | n~i->n~i :
  start:
  'l': found_l
  'n~': found_n~_ll
  'i:0': reject
  '@': start
  found_l:
  'l': found_n~_ll
  'i:0': reject
  '@': start
  found_n~_ll:
  '+': found_+
  'i:0': reject
  '@': start
  found_+:
  'i:i': i_state
  'i:0': i_deleted
  '@': start
  i_state:
  `'':0': i_state
  'VOC': reject
  'i:0': reject
  '@': start
  i_deleted:
  `'':0': i_deleted
  'VOC': start
  'i:0': reject
  '@': reject

i->y | i->i :
  start:
  'q': found_q_g
'g': found_q_g
'VOC': found_vowel
'i:y': reject
"i:i'": reject
'@': start

found_q_g:
'@': start
'VOC': found_vowel
'i:y': reject
"i:i'": reject
'@': start

found_vowel:
'+:0': found_+
'q': found_q_g
'g': found_q_g
'VOC': found_vowel
'i:y': reject
"i:i'": reject
'@': start

found_+:
'@': start
'VOC': reject
'@': start

i_state:
'@': start

y_state:
'@': start

E->ie | E->e | U->ue | U->u | O->ue | O->o | stem changing:
start:
'E:e': no_stem_change
'O:o': no_stem_change
'E:i': stem_change
'O:u': stem_change
'U:u': u_stem_change
'O:e': reject
'@': start
no_stem_change:
  '':0': one_stress
  '0:e': reject
  '#': reject
  'STEMS:@': reject
  '@': no_stem_change
one_stress:
  '':0': reject
  '#': start
  '0:e': reject
  '@': one_stress
stem_change:
  '0:e': one_stress
  '@': reject
u_stem_change:
  '0:e': one_stress
  '#': reject
  '':0': one_stress
  'STEMS:@': reject
  '@': no_stem_change

e->i | e->e :
start:
  '':0': found_stress
  "E":e": found_e
  "E":i": found_i
  '@': start
found_stress:
  "E":e": reject
  "E":i": found_i
  '@': start
found_e:
  '+:0': found_e_+
  '@': found_e
found_i:
  '+:0': found_i_+
  '@': found_i
found_e_+:
  'i:@': e_state
  '':0': found_e_+_stress
  '@': start
found_i_+:
  'i:@': i_state
  '':0': found_i_+_stress
  '@': start
i_state:
  '':0': i_state
  'VOC': start
'@': reject
e_state:
  '':0': e_state
  'VOC': reject
  '@': start
found_i_+stress:
  'i:@': reject
  "i':@": reject
  '@': start
found_e_+stress:
  'i:@': start
  "i':@": start
  '@': reject

s->es | pluralization :
  start:
    'CONS': cons
    'VOC': vowel
    '@': start
  cons:
    '+:e': post_conse_suffix
    '+:0': post_cons0_suffix
    'VOC': vowel
    '@': cons
  vowel:
    '+:e': reject
    '+:0': post_vowel0_suffix
    'CONS': cons
    '@': vowel
  post_conse_suffix:
    's': start
    '@': reject
  post_cons0_suffix:
    '0:@': reject
    's': reject
    '@': start
  post_vowel0_suffix:
    '0:@': reject
    '@': start
7 Appendix C: Excerpts from Keyboard

import math
from enum import Enum

DEFAULT_DIST = 4.0

QWERTY = {'a': (2, 1), 'b': (3, 5), 'c': (3, 3), 'd': (2, 3), 'e': (1, 3), 'f': (2, 4),
           'g': (2, 5), 'h': (2, 6), 'i': (1, 8), 'j': (2, 7), 'k': (2, 8), 'l': (2, 9),
           'm': (3, 7), 'n': (3, 6), 'o': (1, 9), 'p': (1, 10), 'q': (1, 1), 'r': (1, 4),
           's': (2, 2), 't': (1, 5), 'u': (1, 7), 'v': (3, 4), 'w': (1, 2), 'x': (3, 2),
           'y': (1, 6), 'z': (3, 1), '\': (0, 0), '1': (0, 1), '2': (0, 2), '3': (0, 3),
           '4': (0, 4), '5': (0, 5), '6': (0, 6), '7': (0, 7), '8': (0, 8), '9': (0, 9),
           '0': (0, 10), '-': (0, 11), '=': (0, 12), '[': (1, 11), ']': (1, 12), '\': (2, 10),
           ';': (3, 8), '\\': (1, 13), '/': (2, 11), '/': (3, 9), 'A': (2, 1),
           'B': (3, 5), 'C': (3, 3), 'D': (2, 3), 'E': (1, 3), 'F': (2, 4), 'G': (2, 5),
           'H': (2, 6), 'I': (1, 8), 'J': (2, 7), 'K': (2, 8), 'L': (2, 9), 'M': (3, 7),
           'N': (3, 6), 'O': (1, 9), 'P': (1, 10), 'Q': (1, 1), 'R': (1, 4), 'S': (2, 2),
           'T': (1, 5), 'U': (1, 7), 'V': (3, 4), 'W': (1, 2), 'X': (3, 2), 'Y': (1, 6),
           'Z': (3, 1), '~': (0, 0), '!': (0, 1), '@': (0, 2), '#': (0, 3), '$': (0, 4),
           '%': (0, 5), '^': (0, 6), '&': (0, 7), '*': (0, 8), '(': (0, 9), ')': (0, 10),
           '<': (3, 8), '>': (3, 1), '1': (1, 11), '2': (1, 12), '3': (1, 13), '4': (2, 10),
           '5': (3, 9), '6': (3, 10)}

def point_distance(p0, p1):
    DBG("point_distance: (%d,%d) and (%d,%d)"%(p0[0], p0[1], p1[0], p1[1]))
    return (p0[0]-p1[0])**2+(p0[1]-p1[1])**2

class Keyboard(object):
    def __init__(self, type=KeyboardTypes.QUERTY, num_allowable_insertions=2):
        self._num_insertions = num_allowable_insertions
        self._mapping = {}
        self._weight_adjustments = {}
        self._type = type
        if(type == KeyboardTypes.QUERTY):
            self._mapping = QWERTY.copy()
        else:
            print "ERROR: " + str(type) + " has not been implemented yet.  "+\"All error estimates based on default dist."

    def character_distance(self, ch0, ch1):
        DBG("character_distance: '%s' and '%s' using %s keyboard"\
             "%(ch0,ch1,str(self.type()))")
        keymap = self._mapping()
        if not ch0 in keymap and not ch1 in keymap:
            return DEFAULT_DIST*2
        if not ch0 in keymap:
            return DEFAULT_DIST
        if not ch1 in keymap:
            return DEFAULT_DIST
        return math.sqrt(point_distance(keymap[ch0], keymap[ch1]))
8 Appendix D: FST

from enum import Enum
from callable import Callable
import sys

DEFAULT_LIKE_TRANSITION_COST = 1.0
DEFAULT_MUTATION_COST = 4.0
DEFAULT_INSERTION_COST = 4.0
EPSILON = '\0'
FILE_EXTENSION = '.fstfile'

DEFAULT_COST_REDUCE = .75
DEFAULT_COST_INCREASE = 1.5

StateFileTags = Enum('TRANSITION', 'START', 'EPSILON', 'END')

def special_min(list_of_tuples):
    ans = None
    for elt in list_of_tuples:
        if ans is None or elt[0] < ans[0]:
            ans = elt
    return ans

class State(object):
    def read_from_file(filename):
        # dictionary from state key to state
        found_states = {}
        FILE = open(filename, 'r')
        for line in FILE:
            parts = line.split()
            parts = [part.strip() for part in parts]
            if len(parts) < 5:
                continue
            if parts[0] == str(StateFileTags.TRANSITION):
                continue
            parts.pop(0)
            start_key = None
            next_key = None
            cost = 0.0
            if parts[0] == str(StateFileTags.START):
                assert len(parts) == 4
                start_key = (parts[0])
                if not start_key in found_states:
                    found_states[start_key] = State(start_key)
            if parts[1] == str(StateFileTags.EPSILON):
                parts[1] = EPSILON
            if parts[2] == str(StateFileTags.EPSILON):
                parts[2] = EPSILON
            if parts[0] == str(StateFileTags.START):
                assert len(parts) == 4
                start_key = (parts[0])
                if not start_key in found_states:
                    found_states[start_key] = State(start_key)
next_key = (parts[1],parts[2])
if not next_key in found_states:
    found_states[next_key] = State(next_key)
    cost = float(parts[3])
elif(parts[2]==str(StateFileTags.END)):
    assert len(parts)==4
    if(parts[0]==str(StateFileTags.EPSILON)):
        parts[0]=EPSILON
    if(parts[1]==str(StateFileTags.EPSILON)):
        parts[1]=EPSILON
    start_key = (parts[0],parts[1])
    if not start_key in found_states:
        found_states[start_key]=State(start_key)
next_key = (parts[2])
if not next_key in found_states:
    found_states[next_key] = State(next_key)
    cost = float(parts[3])
else:
    assert len(parts)==5
    if(parts[0]==str(StateFileTags.EPSILON)):
        parts[0]=EPSILON
    if(parts[1]==str(StateFileTags.EPSILON)):
        parts[1]=EPSILON
    if(parts[2]==str(StateFileTags.EPSILON)):
        parts[2]=EPSILON
    if(parts[3]==str(StateFileTags.EPSILON)):
        parts[3]=EPSILON
    start_key = (parts[0],parts[1])
    if not start_key in found_states:
        found_states[start_key]=State(start_key)
    next_key = (parts[2],parts[3])
    if not next_key in found_states:
        found_states[next_key] = State(next_key)
    cost = float(parts[4])
start_state = found_states[start_key]
next_state = found_states[next_key]
start_state.add_new_next_state(next_state,cost)

FILE.close()
assert (str(StateFileTags.START)) in found_states
return found_states[(str(StateFileTags.START))]

read_from_file = Callable(read_from_file)
def write_to_file(self,filename):
    if(not filename.endswith(FILE_EXTENSION)):
        filename+=FILE_EXTENSION
    FILE=open(filename,'w')
    start_key = self.key()
    start_string=""
```python
# Nothing can start at END, can only start at START or other state
if not isinstance(start_key, tuple):
    start_string += str(StateFileTags.START)
else:
    if start_key[0] == EPSILON:
        start_string += str(StateFileTags.EPSILON)
    else:
        start_string += start_key[0]
    if start_key[1] == EPSILON:
        start_string += str(StateFileTags.EPSILON)
    else:
        start_string += start_key[1]

for next_key in self.next_states():
    next = self.next_states()[next_key]
    next_string = ""
    # Nothing can end at START, can only end at END or other state
    if not isinstance(next_key, tuple):
        next_string += str(StateFileTags.END)
    else:
        if next_key[0] == EPSILON:
            next_string += str(StateFileTags.EPSILON)
        else:
            next_string += next_key[0]
        if next_key[1] == EPSILON:
            next_string += str(StateFileTags.EPSILON)
        else:
            next_string += next_key[1]

    # This handles all the transitions from START to blah
    FILE.write(start_string + ' ' + next_string + ' ' +
               str(self.transition_weight(next_key)) + "\n")

    for next_next_key in next.next_states():
        next_next_string = ""
        # nothing can end at Start, can only end at END or other state
        if not isinstance(next_next_key, tuple):
            next_next_string += str(StateFileTags.END)
        else:
            if next_next_key[0] == EPSILON:
                next_next_string += str(StateFileTags.EPSILON)
            else:
                next_next_string += next_next_key[0]
            if next_next_key[1] == EPSILON:
                next_next_string += str(StateFileTags.EPSILON)
            else:
                next_next_string += next_next_key[1]

        # This handles the transitions from every state after START
        # to every state it can transition to
        FILE.write(next_string + ' ' + next_next_string + ' ' +
                    str(next.transition_weight(next_next_key)) + "\n")

# I think I've got everything covered everything
```
FILE.close()

def __init__(self,key):
    self._key=key
    self._next_states=
    self._transition_weights = {}

def key(self):
    return self._key

def next_states(self):
    return self._next_states

def transition_weights(self):
    return self._transition_weights

def transition_weight(self, next_state_key):
    assert next_state_key in self._transition_weights
    return self._transition_weights[next_state_key]

def transition_to_next_by_key(self,next_state_key):
    assert next_state_key in self._next_states
    assert next_state_key in self._transition_weights
    return (self._next_states[next_state_key],
    self._transition_weights[next_state_key])

def add_new_next_state(self,state,weight):
    assert state.key() not in self._next_states
    self._next_states[state.key()]=state
    assert state.key() not in self._transition_weights
    self._transition_weights[state.key()]=weight

def transduce(self,real_string,typo_string):
    if(len(real_string)==len(typo_string)):
        state_keys = [(real_string[i],typo_string[i]) \n        for i in range(len(real_string))]
        state_keys.append(str(StateFileTags.END))
        total_cost=0.0
        current_state = self
        for state_key in state_keys:
        current_state,cost =current_state.transition_to_next_by_key(state_key)
        total_cost+=cost
        return (total_cost/len(state_keys),real_string,typo_string)
    elif(len(real_string)>len(typo_string)):
        possible = []
        for i in range(len(typo_string)+1):
            possible.append(self.transduce(real_string,typo_string[:i]+\
                                      EPSILON+typo_string[i:]))
        return special_min(possible)
    else:
        possible = []
        for i in range(len(real_string)+1):
possible.append(self.transduce(real_string[:i]+EPSILON+
    real_string[i:], typo_string))

return special_min(possible)

def reduce_costs(self, real_string, typo_string):
    return self.edit_costs(real_string, typo_string, DEFAULT_COST_REDUCER)

def increase_costs(self, real_string, typo_string):
    return self.edit_costs(real_string, typo_string, DEFAULT_COST_INCREASE)

def edit_costs(self, real_string, typo_string, multiplier):
    assert len(real_string) == len(typo_string)
    state_keys = [(real_string[i], typo_string[i]) for i in range(len(real_string))]
    state_keys.append(str(StateFileTags.END))
    current_state = self
    for state_key in state_keys:
        current_state._transition_weights[state_key] *= multiplier
        current_state, cost = current_state.transition_to_next_by_key(state_key)
    return

def __str__(self):
    return str(self._key)

def is_good(self):
    if key == str(StateFileTags.START):
        return True
    return self._key[0] == self._key[1]

def is_mutation(self):
    if key == str(StateFileTags.START):
        return False
    return (not self._key[0] == self._key[1]) and 
    (not (self._key[0] == EPSILON or self._key[1] == EPSILON))

def is_insertion(self):
    if key == str(StateFileTags.START):
        return False
    return self._key[0] == EPSILON or self._key[1] == EPSILON
9 Appendix E: Word Tree

from callable import Callable

class Tree(object):
    
    cost_dict={}

    def set_cost_dict(new_dict):
        Tree.cost_dict = new_dict
        pass
    set_cost_dict = Callable(set_cost_dict)

    def __init__(self, word, rule, parent=None):
        self._word=word[:]
        self._num_children=0
        self._cost=0
        self._depth=0
        self._rule_to_child_dict={}
        self._child_to_rule_dict={}
        self._rulelist = []
        if not parent==None:
            self._cost=parent.cost()+Tree.cost_dict[rule]
            parent.add_child(self,rule)
            self._depth = parent.depth()+1
            self._rulelist = parent._rulelist[:]
            self._rulelist.append(rule)
            self._parent=parent

    def path(self):
        return self._rulelist
    def add_child(self,child_tree,rule):
        self._rule_to_child_dict[rule]=child_tree
        self._child_to_rule_dict[child_tree]=rule
        self._num_children+=1

    def parent(self):
        return self._parent
    def word(self):
        return self._word
    def num_children(self):
        return self._num_children
    def depth(self):
        return self._depth
    def cost(self):
        return self._cost
    def height(self):
        return self._height
## Appendix F: GUI

```python
#!/usr/bin/python
from kimmo import *
from Tkinter import *
from tkFileDialog import askopenfilename
from tkFileDialog import asksaveasfilename
from tkMessageBox import showerror
from wordtree import Tree
from recobject import RecognizeStatus, KimmoRec
from Queue import PriorityQueue
import re
import sys
import subprocess
import shlex
from keyboard_distance import Keyboard
import fst_state

typo_transducer = fst_state.State.read_from_file('fst_clean.fstfile')
filename = 'spanish.yaml'
kim = load(filename)
default_pairs = kim._pair_alphabet
keyboard = Keyboard()
rule_mappings = None
rule_costs = None

DBG=True

def DBG(item):
    if(not DEBUG):
        return
    print str(item)

def save_rule_costs_to_file(filename):
    temp_filename='~spanish_rec.save.temp'
    FILE.open(temp_filename,'w')
    for key in rule_costs:
        FILE.write("RULE_COST "+key[0]+' '+key[1]+' '+rule_costs[key]+'
"
)    FILE.close()
    cat_call = "cat "+temp_filename+' >> filename'
    subprocess.Popen(shlex.split(cat_call)).wait()
    rm_call = "rm -f "+temp_filename
    subprocess.Popen(shlex.split(rm_call)).wait()

def read_rule_costs_from_file(filename):
    FILE.open(filename,'r')
    for line in FILE:
        if line.startswith("RULE_COST"):
            continue
        print line
```

40
elts = line.split()
elts = [elt.strip() for elt in elts]
rule_costs[(elts[1],elts[2])] = elts[3]

FILE.close()

def queue_elt_to_string(elt_tuple):
    (priority, elt) = elt_tuple
    out_string = ""
    if(elt.status()==RecognizeStatus.MATCH):
        out_string += "Recognized as: [Score=%f] "%(float(priority))
        out_string += str(elt.real_word()) + ': ' + str(elt.parse())
    elif(elt.status()==RecognizeStatus.TYPO):
        out_string += "Typo Error(?): [Score=%f] "%(float(priority))
        out_string += str(elt.real_word()) + ': ' + str(elt.parse())
    elif(elt.status()==RecognizeStatus.RULE_ERROR):
        out_string += "Rule Error(?): [Score=%f] "%(float(priority))
        out_string += str(elt.real_word()) + ': ' + str(elt.parse())
    return out_string

#DESTRUCTIVE!!!!

def queue_to_stringlist_and_eltlist(queue):
    ans1 = []
    ans2 = []
    new_queue = PriorityQueue()
    while queue.qsize() > 0:
        elt_tuple = queue.get()
        ans1.append(queue_elt_to_string(elt_tuple))
        ans2.append(elt_tuple)
        new_queue.put(elt_tuple)
    queue = new_queue
    return (ans1, ans2)

def rec_word(word, word2node):
    DBG('rec_word() on %s\n' % (word))
    recognize_list = kim.recognize(word)
    queue = PriorityQueue()
    if(len(recognize_list)<1):
        typo_loop(word, queue)
        rule_loop(word, queue, word2node)
        return queue
    else:
        queue.put(((0, KimmoRec(word, word, recognize_list, RecognizeStatus.MATCH))))
    return queue

def typo_loop(initial_word, queue):
    DBG('typo_loop() with %s\n' % (initial_word))
    all_results = []
    checked_words = set()
for sym in default_pairs:
    if (sym==None):
        continue
    sym = sym.output()
    if(sym=='0' or sym.isupper()):
        continue
    # this subloop does single mutations for typographic errors
    for i in range(1,len(initial_word)+1):
        test_word = initial_word[:max(i-1,0)]+sym+initial_word[i:]
        if(not test_word in checked_words):
            recognized = kim.recognize(test_word)
            checked_words.add(test_word)
            #DBG('Testword: %s recognized:
                 %s\n                 %s
                 (str(test_word),str(recognized))
            if(len(recognized)>0):
                #DBG('found mutation typo %s from %s\n                 %s\n                 (test_word,initial_word)
                transduced_tuple = typo_transducer.transduce(test_word,initial_word)
                queue.put((transduced_tuple[0],
                           KimmoRec(transduced_tuple[1],transduced_tuple[2],
                                    recognized,RecognizeStatus.TYPO)))
    # this subloop does single additions, assuming a typographic error
    for i in range(len(initial_word)+1):
        test_word = initial_word[:i]+sym+initial_word[i:]
        if(not test_word in checked_words):
            recognized = kim.recognize(test_word)
            checked_words.add(test_word)
            #DBG('Testword: %s recognized:
                 %s\n                 %s
                 (str(test_word),str(recognized))
            if(len(recognized)>0):
                #DBG('found insertion typo %s from %s\n                 %s\n                 (test_word,initial_word)
                transduced_tuple = typo_transducer.transduce(test_word,initial_word)
                queue.put((transduced_tuple[0],
                           KimmoRec(transduced_tuple[1],transduced_tuple[2],
                                    recognized,RecognizeStatus.TYPO)))
    # this subloop does single deletions, assuming a typographic error
    for i in range(1,len(initial_word)):
        test_word = initial_word[:max(i-1,0)]+initial_word[i:]
        if(not test_word in checked_words):
            recognized = kim.recognize(test_word)
            checked_words.add(test_word)
            #DBG('Testword: %s recognized:
                 %s\n                 %s
                 (str(test_word),str(recognized))
            if(len(recognized)>0):
                #DBG('found deletion typo %s from %s\n                 %s\n                 (test_word,initial_word)
                transduced_tuple = typo_transducer.transduce(test_word,initial_word)
                queue.put((transduced_tuple[0],
                           KimmoRec(transduced_tuple[1],transduced_tuple[2],
                                    recognized,RecognizeStatus.TYPO)))
return queue

def rule_mappings_init():
    all_possible_changes=[]
names = [rule.name() for rule in kim.rules()]
for name in names:
    mutations = name.split('|')
    for mutation in mutations:
        mutation = mutation.strip()
        if mutation.__contains__('->'):
            m1 = mutation.split('->')
            all_possible_changes.append([m1[0].strip(),m1[1].strip()])
        else:
            if(mutation.__contains__('+')):
                all_possible_changes.append(['',mutation.split('+')[0]])
            elif(mutation.__contains__('-'')):
                all_possible_changes.append([mutation.split('-')[0],''])
for change in all_possible_changes:
    for pair in default_pairs:
        if (not change[0]=='' ) and pair.input()==change[0]:
            change[0]=pair.output()
            break
for i in range(len(all_possible_changes)):
    all_possible_changes[i]=tuple(all_possible_changes[i])
cost_dict={}
for change in all_possible_changes:
    cost_dict[change]=2.0#forward
    cost_dict[(change[1],change[0])]=2.0#backward
return (all_possible_changes,cost_dict)

def make_tree_of_alternatives(tree,depth, word_set,word2node):
    if(depth==0):
        return
    word = tree.word()
    for rule in rule_mappings:
        forward = re.compile(rule[0])
        backward = re.compile(rule[1])
        for match in forward.finditer(word):
            match = match.span()
            new_word = word[:match[0]]+rule[1]+word[match[1]:]
            if not new_word in word_set:
                word_set.add(new_word)
                child = Tree(new_word,(rule[1],rule[0]),parent=tree)
                word2node[new_word]=child
        for match in backward.finditer(word):
            match = match.span()
            new_word = word[:match[0]]+rule[0]+word[match[1]:]
            if not new_word in word_set:
                word_set.add(new_word)
                child = Tree(new_word,rule,parent=tree)
                word2node[new_word]=child
    for child in tree._child_to_rule_dict:
        make_tree_of_alternatives(child,depth-1,word_set,word2node)
def rule_loop(initial_word, queue, word2node):

    # example: coco->cueco->queco->quezc->quezzc->quezzzcue->quezzzzque.
    # possibly restrict ourselves to 2 or 3 rule depth space instead of trying to span every
    # so we don't have to worry about infinite loops
    Tree.set_cost_dict(rule_costs)
    root = Tree(initial_word, None, None)
    used_words = set()
    used_words.add(initial_word)
    word2node.clear()
    word2node[initial_word] = root

    # 3 means we allow up to 3 mutations from initial_word
    make_tree_of_alternatives(root, 3, used_words, word2node)
    for word in used_words:
        recognized = kim.recognize(word)
        if(len(recognized)>0):
            print word, word2node[word].cost()
            cost = word2node[word].cost()
            queue.put((cost, KimmoRec(word, initial_word, recognized, RecognizeStatus.RULE_ERROR)))
            DBG("ROOT:" + str(root))
    return queue

class Application(Frame):

    def set_output(self, elts):
        self.OUTPUT.delete(0, END)
        self.output_dict = {}
        if len(elts) > 0:
            for elt in elts:
                self.output_dict[str(elt)] = elt
                self.OUTPUT.insert(END, str(elt))
        else:
            self.OUTPUT.insert(0, "")

    def load_fst(self):
        filename = askopenfilename(title='Open FST', \
                                   filetypes=[("FST files","*.fstfile")])
        if isinstance(filename, str):
            DBG('Opening file %s' % (filename))
            typo_transducer = fst_state.State.read_from_file(filename)
            rule_costs = read_rule_costs_from_file(filename)
        else:
            DBG('File Load Error: Canceled or invalid file')
        return

    def save_fst(self):
        filename = asksaveasfilename(title='Save FST', \
                                     filetypes=[("FST files","*.fstfile")])
filetypes=([('Keyboard File','*.fstfile')])

if(isinstance(filename,str)):
    DBG('Saving file as %s
%(filename)
keyboard.write_to_file(filename)
save_rule_costs_to_file(filename)
else:
    DBG('File Save Error: Canceled or invalid file')
    return

def createWidgets(self):
    # create a menubar
    menubar = Menu(self)
    filemenu = Menu(menubar,tearoff=0)
    filemenu.add_command(label='Load', command=lambda : self.load_fst())
    filemenu.add_command(label='Save', command=lambda : self.save_fst())
    filemenu.add_command(label='Quit', command=self.master.destroy)
    menubar.add_cascade(label='File',menu=filemenu)
    self.master.config(menu=menubar)

    # stupid label telling people what to put in the entry box
    self.LABEL = Label(self)
    self.LABEL['text']='Spanish Word to Recognize:'
    self.LABEL['fg']='black'
    self.LABEL.pack(side=TOP)

    # Entry box for user input
    self.TEXT = Entry(self)
    self.TEXT['width']=30
    self.TEXT['fg']='black'
    self.TEXT.pack(side=TOP)

    # Recognize button
    self.RECOGNIZE = Button(self)
    self.RECOGNIZE['text']='Recognize'
    self.RECOGNIZE['bg']='green'
    self.RECOGNIZE['command']=lambda :self.recognize()
    self.RECOGNIZE.pack(side=TOP)

    # Frame for output area
    # contains:
    #   Output text
    #   Scroll for output text
    output_frame = Frame(self)
    scrollx = Scrollbar(output_frame,orient=HORIZONTAL)
    scrolly = Scrollbar(output_frame)
    scrolly.pack(side=RIGHT,fill=Y)
    self.OUTPUT = Listbox(output_frame,height=20,width=50, background='white',
        yscrollcommand=scrollx.set, xscrollcommand=scrolly.set,
        selectmode=BROWSE)
self.UPDATE_BUTTON = Button(output_frame)
self.UPDATE_BUTTON['text'] = 'Update Weights'
self.UPDATE_BUTTON['command'] = lambda: self.update_weights()
self.UPDATE_BUTTON.pack(side=BOTTOM)
scrollx.pack(side=BOTTOM, fill=X)
self.OUTPUT.pack(side=TOP)
self.OUTPUT.insert(0, '')
output_frame.pack(side=BOTTOM)
scrolly.config(command=self.OUTPUT.yview)
scrollx.config(command=self.OUTPUT.xview)

def __init__(self, master=None):
    Frame.__init__(self, master, height=300, width=400)
    if (master):
        master.title('Spanish Word Recognizer')
    self.pack_propagate(0)
    self.word = ''
    self.pack()
    self.createWidgets()
    self.queue_output_list = []
    self.word_to_node = {}

def decrease_weight(self, queue_elt):
    (priority, elt) = queue_elt
    if elt.status() == RecognizeStatus.TYP0:
        typo_transducer.reduce_costs(elt.real_word(), elt.error_word())
    elif elt.status() == RecognizeStatus.RULE_ERROR:
        end_node = self.word_to_node[elt.real_word()]
        path = end_node.path()
        for pelem in path:
            rule_costs[pelem] *= fst_state.DEFAULT_COST_REDUCER
    else:
        # must be a RecognizeStatus.MATCH, so do nothing
        pass

def increase_weight(self, queue_elt):
    (priority, elt) = queue_elt
    if elt.status() == RecognizeStatus.TYP0:
        typo_transducer.increase_costs(elt.real_word(), elt.error_word())
        pass
    elif elt.status() == RecognizeStatus.RULE_ERROR:
        end_node = self.word_to_node[elt.real_word()]
        path = end_node.path()
        for pelem in path:
            rule_costs[pelem] *= fst_state.DEFAULT_COST_INCREASER
    else:
        # must be a RecognizeStatus.MATCH, so do nothing
pass
pass

def update_weights(self):
    print "Word to Node Dict: ",self.word_to_node
    sel = self.OUTPUT.curselection()
    print sel
    if len(sel)==0:
        print "STOP THAT! I'm BRITTLE!"
        return
    sel = int(sel[0])
    print sel
    if len(self.queue_output_list)>sel:
        elt = self.queue_output_list[sel]
        print queue_elt_to_string(elt)
        self.decrease_weight(elt)
        for i in range(sel):
            self.increase_weight(self.queue_output_list[i])
        self.queue_output_list = []
        self.set_output([])
        self.word_to_node={}
        return
    print "STOP THAT! There's nothing in that slot of the ListBox."

def recognize(self):
    self.word = self.TEXT.get()
    self.set_output(['Thinking...'])
    self.update()
    DBG('RECOGNIZE BUTTON EVENT: %s
'%(self.word))
    queue = rec_word(self.word,self.word_to_node)
    DBG("QueueSize: %d"%(queue.qsize()))
    if(queue.qsize()==0):
        self.set_output([])
        showerror("Recognition Error","No matches found for %s."%(self.word))
        return
    (string_output_list, self.queue_output_list) = queue_to_stringlist_and_eltlist(queue)
    self.set_output(string_output_list)

if __name__=='__main__':
    (rule_mappings,rule_costs) = rule_mappings_init()
    root = Tk()
    app = Application(master=root)
    app.mainloop()