Universität Konstanz
Finite-State Morphology
WS 20/21

Luca Bosch
luca.bosch@uni.kn
01/851145

Project: Spanish Verb Conjugation

Abstract

We discuss the implementation of a finite-state machine capable of generating and detecting Spanish verb forms using *xfst* and *lexc*. In this report, you can find a brief introduction to the Spanish language and the tools used for modeling parts of it, as well as a detailed description of the necessary linguistic concepts. Furthermore, I explain the implementation process, highlighting the thought processes and difficulties at different stages. Lastly, I summarize my findings and discuss how the project's concepts can be applied in practice.

Contents

1 Introduction

In this first section, you can find a quick overview of the concepts that were used in the creation of this project and an introduction to the Spanish language, as well as an explanation of why this project is relevant. The second section explains all of the linguistic concepts that I tried to model, while the third section records their implementation process. In the fourth section, the results and gained insights are presented, whereas the fifth section can help with verifying them experimentally.

1.1 Preliminaries

As part of the course *Finite-State Morphology*, this final project serves the purpose of transferring the knowledge gained during the course to a practical problem. We use the tools *xfst* and *lexc* created by Karttunen et al. at the PARC[1] along with the Book *Finite State Morphology*[2] as guidance for their usage.

1.1.1 Finite-State Networks

The idea is based on modeling a language as a finite automaton that can process an input and give an output. This can, for instance, be used to accept or reject a series of letters based on whether it is a valid word, by creating an automaton in such a way that it can make transitions between states depending on the input letter.

[1]Xerox Palo Alto Research Center

[2]K. Beesley and L. Karttunen, 2003, CSLI

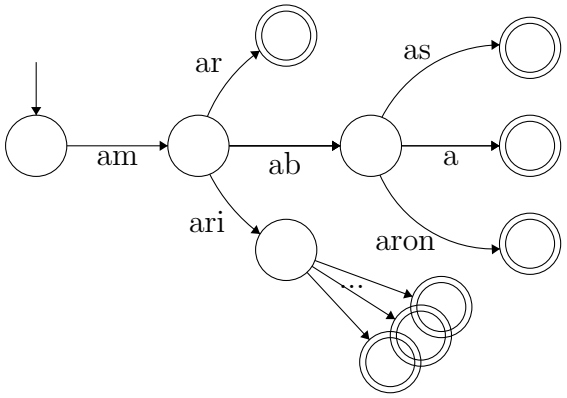

Figure 1: Incomplete automaton for *amar*. Imagine we consider the automaton to accept an input when we are in one of the double-circle states and there are no more letters to be processed.

That serves as a great base for this project since it alleviates us from the tedious work of listing every valid verb form separately and instead we only have to change the first transition. However, we are interested in more detailed linguistic information rather than just telling whether a word exists or not. For that purpose, we can use so-called transducers, which are a modified version of this automaton concept, and allow us to 'track' the path that an input generated.

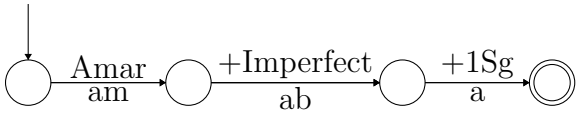

Figure 2: Transducer which accepts 'amaba' and can then output the path information which would be 'Amar + Imperfect + 1Sg'.

1.1.2 Used Tools

An efficient way of creating these automata is by using a *lexc*-File which provides a special syntax and is structured into *Lexicon*s, which textually represent the states and list all the possible transitions, with the initial state being *Root* and the accepting ones denoted with #. However, these basic automata are rather 'stupid' - they cannot detect irregularities, which are commonplace in almost every language - so we additionally need *regular expressions*[3], also known as *regex*. *xfst* offers possibilities of combining *lexc* and *regex* in such a way that we can specify irregularities separately, which again reduces the amount of manual work significantly.

1.2 Language Background

Spanish (*español*), also known as Castilian (*castellano*), is mainly spoken in Spain, Mexico, Central, and South America as well as the southwestern parts of the United States, Equatorial Guinea, and the Philippines.

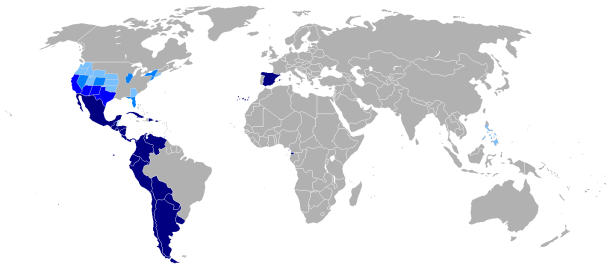

Figure 3: Darker shades of blue indicate more Spanish speakers as a percentage of the population while lighter shades equal less. *Public Domain*

It originated in the Iberian peninsula and is derived from Latin, with some Arabic influences. The dialects spoken in the different regions vary slightly, for instance, *tú* is replaced by *vos* and the verb forms change in Argentina and surrounding countries. Most other differences are only pronunciation-based, though. Spanish is written with the Latin alphabet, including the accentuated characters *á, é, í, ó, ú, ü,* and *ñ*.

1.3 Motivation

Spanish is not only the second-most spoken native language in the world[4] but also the fastest growing one, especially in Europe[5]. As finite-state morphology can be used for applications that help people with learning a new language, this is a good starting point for a project such as this one. The focus was laid on verb conjugation since that is the domain where we can profit most from an approach with the provided tools, as verb tenses are made with suffixes and there exist lots of regular verbs whose rules can be modeled in *lexc*. I will subsequently focus on the variation of the language that is spoken in Spain since everything originated there.

[3] Regular expressions have been invented by Stephen C. Kleene in 1951 and offer a different syntax for textually creating automata

[4] *The Ethnologue 200*, Ethnologue, 2021

[5] *Spanish: A Living Language*, Instituto Cervantes, 2018

2 Linguistics

Like most natural languages, Spanish has some irregular conjugation patterns. Since it is unfeasible to cover all of them, this section explains the ones I analyzed.

2.1 Scope

To make a project like this interesting, we are interested in a mix of regular and irregular verbs. The idea is to build a basis on conjugating regular verbs in *lexc* and then adjusting it in such a way, that irregular verbs can be inserted in the same way, without needing any additional *Lexicon*s. For that purpose, we will have to pick some irregular verbs that show minor stem or accentuation changes.
One of the issues is that the most common and useful verbs like *ser* (be), *haber* (have), and *ir* (go) show so many differences, that creating a *regex* to recognize the correct form would be even more complicated than just listing every single form separately. The latter, however, would defeat the purpose of this project, which was to simplify things. As a compromise, I decided to mainly cover verbs with minor changes.

2.2 Conjugation

Spanish has the following moods and tenses:

- **Indicative**
  - **Present**
  - **Imperfect**
  - **Preterite**
  - **Future**
  - Preterite Imperfect
  - Past Perfect
  - Future Perfect
  - Past Anterior
- **Subjunctive**
  - **Present**
  - **Imperfect**
  - Preterite
  - Future
  - Preterite Imperfect
  - Past Perfect
  - Future Perfect
- **Conditional**
  - **Present**
  - Perfect
- **Imperative**
  - **Affirmative**
  - Negative

The forms I used are bold. The reason for imperative negative not being used is that it is no form in itself but simply *No* and the subjunctive form, while all the other forms are an auxiliary verb and the gerund. There are three types of verbs, namely those ending in *-ar*, *-er*, and *-ir*. These have different suffixes and infixes as can be seen in Table 1 below.

Infixes After Root	-ar	-er	-ir
Indicative Perfect	-ab-	-í-	-í-
Subjunctive Perfect	-ar-	-ier-	-ier-
Conditional Present	-ará-	-erí-	-irí-

Table 1: This table shows what infix is used for verbs in which mood and tense depending on its ending which can be on *-ar*, *-er*, or *-ir*. For example, *beber* ends on *-er* and therefore the indicative perfect stem is *bebí-*.

2.3 Stem Changes

- **g → gu.** 'ge' is pronounced [xe], but 'ga' is pronounced [ga], so 'u' is inserted between the letters, and the pronunciation of 'gue' is [ge].
  Examples: *llegue, agregue, conjuege*

- **c → qu.** 'ce' is pronounced [θe], but 'ca' is pronounced [ka], so it is replaced by 'qu', and the pronunciation of 'que' is [ke].
  Examples: *saque, ataque, pique*

- **e → ie.** This empenthesis happens for the $1^{st}/2^{nd}/3^{rd}$ persons singular and 3^{rd} plural when 'e' is followed by a 'z' or or a consonant and a 'z'. Examples:
  empiezo, comienzo, tropiezo

- **c → zc.** 'co' is pronounced [ko], but 'ce' is pronounced [θe], , so 'z' is inserted before the letters, and the pronunciation of 'zco' is [θko].
  Examples: *parezco, aparezco, crezco*

- **i → y.** This avoids sequences of many vowels, after an e, 'i' is replaced by 'y' when followed by 'ó' or 'eron'.
  Examples: *creyó, poseyó, leyó*

- **o → ue.** This was a shift away from the Latin stem.
  Examples: *vuelve, mueve, duele*

- **u → ue.** This was another shift away from the Latin stem, but here the inifinitive changed as well. Example: *juego*

2.4 Accentuation Changes

- **a → á.** The emphasis in 2^{nd} person indicative imperfect verbs would be on the 'a' of the suffix 'amos'. But it is meant to be on the 'a' of the infix '-ab-' so the pronunciation stays in line with the rest of the persons. Therefore we add an accent to the latter 'a'.
  Examples: *llamábamos, ayudábamos*

- **u → ú.** The emphasis on some verbs changes when it is conjugated. To avoid confusion with some nouns or adjectives, we add an accent to the last letter of the stem. Examples: *actúo, sitúo, continúo*

Note that for the second change, there also exist verbs whose last letter of the stem is 'i', like *enviar*, but this change does not happen to every verb whose stem ends in a certain vowel, so I chose 'u' because there were no conflicts with other verbs. For a complete dictionary, these cases would have to be treated separately.

3 Implementation

This section documents how I built the network with the different files and describes how I solved the challenges that appeared as I progressed.
During that process, I used the default Windows command prompt and Notepad++ with UTF-8 character encoding and CR LF as line breaks.

3.1 Methodology

I started by looking up the conjugation of a regular verb ending in *-ar* and modeling the *lexc*-File around it, by choosing a few of the planned forms and concatenating them with the stem. Then I also added regular *-er* and *-ir* verbs. I decided not to use flag diacritics yet, as the suffixes differ a lot, and therefore it was clearer to create separate *Lexicon*s.
Thereafter, I implemented all planned forms and for that reason separated the *Lexicon*s into one that contains all the moods and another one that contains all the tenses. This had the advantage of allowing the use of flag diacritics for the right infixes, making the file a lot more structured. Next up, I tried to model the $o \rightarrow ue$ change by converting the theoretical rules into a *regex*-expression

```
o -> u e || _ [v|l]+ [e .#.]|
[e n .#.]|[e s .#.]|[a n .#.]
|[o .#.]|[a s .#.]|[a .#.]
```

To make the creation of these rules easier, I temporarily worked without flag diacritics again and replaced the long lists of *regex* conditions by defining them in the *xfst*-script since they are just indicating that this change only happens for the $1^{st}/2^{nd}/3^{rd}$ person singular and 3^{rd} plural.

```
define IndSubjYoTuElEllos
[[e .#.]|[e n .#.]|[e s .#.]|[a n .#.]
|[o .#.]|[a s .#.]|[a .#.]];
```

I proceeded similarly for the rest of the stem and accentuation changes, creating wordlists to check whether everything is correct and in some cases adjusting the order and conditions for some of the entries. Then I tried to reintegrate the flag diacritics into the *lexc*-file again, but to no avail (an explanation of the problems can be found in the following subsection
Lastly, I tried to model the uniquely irregular verb *jugar* which was rather tricky but in the end, worked out. Before submission, I commented and restructured the files so they would be easier to understand.

3.2 Issues

3.2.1 Accents

In the beginning, I was unsure whether to use accents or not for this project, since you could enter them into the files, but they would not be shown correctly on the Windows terminal. It would not have been terrible to work without them since *xfst* has no problem if the same word can be generated by multiple paths. But after finding out that you could reconfigure the Windows terminal to work with any Unicode characters, I was able to have them displayed, but entering them for testing purposes was not possible, since the tilde and the vowel were always separate. I could not find any solution to that on the internet, but to circumvent this problem I created a word list with a single entry that then could be applied.

3.2.2 Flag Diacritics

The more complicated issue which I was not able to resolve, was the combination of flag

diacritics with *regex*-rules. Though it was possible to use them with a single check, like this:

```
Lexicon Root
beber@P.END.er@:beb@P.END.er@ Mode;

Lexicon Example
+Ind+Pres@R.END.ar@:ará@R.END.ar@ IndPresAR;
+Ind+Pres@R.END.er@:erá@R.END.er@ IndPresER;
+Ind+Pres@R.END.ir@:irá@R.END.ir@ IndPresIR;
```

I spent a lot of time to get it to work for multiple checks, creating rather wild *regex*-rules to remove flags and testing some of the *xfst* settings like 'flag-is-epsilon'. At some point, it even worked, but the network was almost 100 MB large and took minutes to build even for a single verb and a small subset of the forms, so that did not seem useful in practice. I also tried to use the unification flag diacritic instead but the problem remained the same.

The idea I initially had, was to structure the forms just like in the categorization in Section 2.2:

```
Lexicon Root
jugar@P.END.ar@:jug@P.END.ar@ Mood;
beber@P.END.er@:beb@P.END.er@ Mood;
recibir@P.END.ir@:recib@P.END.ir@ Mood;

Lexicon Mood
+Indicative:0 TenseInd;
+Subjunctive:0 TenseSubj;

+Conditional@R.END.ar@:ará@R.END.ar@ Person;
+Conditional@R.END.er@:erá@R.END.er@ Person;
+Conditional@R.END.ir@:irá@R.END.ir@ Person;

Lexicon TenseInd
+Present@R.END.ar@:0@R.END.ar@ PersonAR;
+Present@R.END.er@:0@R.END.er@ PersonER;
+Present@R.END.ir@:0@R.END.ir@ PersonIR;

+Imperfect@R.END.ar@:ab@R.END.ar@ Person;
+Imperfect@R.END.er@:í@R.END.er@ Person;
+Imperfect@R.END.ir@:í@R.END.ir@ Person;

+Future@R.END.ar@:ar@R.END.ar@ PersonFuture;
+Future@R.END.er@:er@R.END.er@ PersonFuture;
+Future@R.END.ir@:ir@R.END.ir@ PersonFuture;
```

For some reason, it made a difference whether I incorporated verbs ending in *-er* and *-ir* or not, even though all the 'require'-flags stayed the same. Unfortunately, I could not find a solution to that problem in the book and even when building from a working template, the problem persisted. Thus I decided to simply go with separate *Lexicon*s, which is a more explicit approach with the downside of losing some of the theoretical structure.

3.3 Documentation

The submission consists of six files:

- **exam.xfst** containing all of the *xfst*-commands necessary to make a correct transducer
- **lexicon.txt** which is a *lexc*-file that defines all of the verbs and forms
- **accents.regex** which handles the changes in accentuation
- **stemChange.regex** which handles the changes in the root
- **testUp.wordlist** containing a set of conjugated verbs
- **testDown.wordlist** containing a set of descriptions for a verb form

What happens in the script is that we first create definitions that are used in the *regex*-files, then combine the two *regex*-files with the *lexc*-file so we can test the resulting network with some examples from the word lists.

4 Discussion and Conclusion

What we can infer from this project is that lots of work can be saved and new knowledge about the structure of languages can be gained by modeling them using these methods. They provide a simple yet powerful means of generating and analyzing words, which can be applied to many real-world scenarios like translation or spellchecking.

4.1 Future Work

We could go further and extend these concepts to include the complete list of verbs, which would take much more time. Or even extend it to all words, though here, again, we are increasing manual work by a lot, since no matter how efficient the finite-state method is, it can only derive words from inputs. There would be no way of generating all nouns without entering them first. We could, however, save work for plural forms, or nouns composed of multiple ones (e.g. German - *DiskussionsBeitrag*).
Another interesting experiment would be to do a syntactical analysis of entire sentences. An easy starting point for that could be using strictly regular programming languages whose syntax comes with a rather small set of rules. This could be of use for code block folding or syntax highlighting purposes.

4.2 Evaluation

In the end, it was a bit disappointing that I could not figure out how to solve the issues with flag diacritics, but all in all, the desired goal which was to create a network that can generate and analyze Spanish verbs could still be completed with some detours. I could not find any restrictions that would hinder the proposed approach from being expanded to all Spanish verbs, resulting in much larger files, but with even more practical usage potential.

5 Test Instructions

To test the network, you first have to execute the script and then you can apply up or down the corresponding word lists I provided. You can also manually apply single words, though in that case, I cannot guarantee that accentuated characters are registered correctly in your terminal. Words without accents are no problem though.
If your output includes strange characters like ¶, |, or ¡, on Windows, you can use the following command before starting xfst:

```
chcp 65001
```

As far as I know, this should not be an issue on Mac OS or Linux, although I did not test that.