# $gr2\epsilon\lambda$: A Greeklish-to-Greek converter

**Spyros Blanas**

## Introduction

Greeklish is a transliteration of the Greek language written using Roman characters. This phenomenon started in the 1980's, when the Greek language was unfortunately covered by multiple ASCII extensions (codepages) which were incompatible. This lead to communication problems, with users being forced to guess the correct encoding of every message, document and webpage. Making matters worse, public discussions were frequently in different encodings, requiring the user to switch encodings to read each reply in a threaded conversation.

In the present day, despite the success of Unicode standardization, Greeklish still is the de facto standard for electronic communication. It is used almost exclusively for personal e-mails, instant messages and sometimes even for business correspondance or marketing e-mails! The Academy of Athens swiftly criticized the romanization of the Greek language, considering it "a full-fledged attack against the classical greek thinking" (aca ). Although this is an exaggeration, a study shows that most users under 35 consider Greeklish a necessary evil when using the Internet and every participant in the study has used Greeklish at least once (Androutsopoulos 1999).

Apart from being aesthetically unpleasent, Greeklish creates three problems:

- Ambiguity, as the Greek accent symbol ('tonos') cannot be represented and is sometimes necessary to discern between words.
- Comprehension, as some Greek characters do not have Roman equivalents and therefore Greeklish words require creative thinking to decipher. In fact, Greeklish has been shown to require at least 40% more time to read and disambiguate (**?**).
- Entity matching problems, as a Greek name and the same name in Greeklish will be recognized as different by search engines.

In this class project, I created an open-source automatic translator that converts Greeklish sentences into real Greek. I have already programmed a plugin for `libpurple`, a popular instant messaging library used by the popular Linux IM application `pidgin`, among others (lib ).



## Related work

Previous research has indicated that three Greeklish writings are prevalent (Macrakis 1996):

- The phonetic writing, where letters are mapped to the closest letter sound in English.
- The glyph-based writing, where letters are mapped to Roman characters that visually resemble the Greek writing.
- The keyboard-based writing, where the standard Greek keyboard layout[1] mapping is followed.

Unfortunately, a given person uses a hybrid of all three writings.

The Institute of Language and Speech Processing (ILSP) has created an automatic transliteration system (Chalamandaris et al. 2006) which has resulted in a spin-off commercial product, sold by Innoetics. The original version of the system first constructs the phonetic representation of the input, checks if this matches Greek and then converts the phonetic representation to the most probable Greek word. Unfortunately, the paper is thin on details about the algorithm.

Galatas developed DeGreeklish (Galatas 2008), a C++ library that builds a finite state automaton for the transliteration. Unfortunately, the library is combined with the `aspell` spell checker, which often makes spelling corrections which result in the conversion of valid English sentences to meaningless Greek.

## Algorithms

The first algorithm converts sentences directly from Greeklish to Greek. It starts with a lookup table, where each mapping between Greeklish-to-Greek characters is indicated. Then, for each Greeklish character[2] the algorithm follows all paths to valid mappings, greedily following all possible

[1] Luckily, only one standard existed for keyboard layouts.

[2] Reality is a little more complicated, as we have to store mappings from 2-to-1 and reconsider them only after the next iteration. For example, suppose the word is "ks" and "ks" may be translated as $\xi$, "s" may become $\sigma$ and "k" may become $\kappa$. We need to take care that we don't produce $\xi\sigma$.

paths. This exponentially increases the number of words being considered and produces many non-existent words. I use the unigram Greek language model to pick the word that is most likely to occur.

The second algorithm relies on Bayes rule. If $g$ is Greeklish and $\epsilon$ is Greek, then I am looking for $\epsilon$ such that:

$$\begin{aligned} & argmax_\epsilon p(\epsilon|g) = \\ & = argmax_\epsilon p(g|\epsilon)p(\epsilon) = \\ & = argmax_\epsilon \prod_i p(g_i|\epsilon_i)p(\epsilon) = \\ & = argmax_\epsilon \sum_i logp(g_i|\epsilon_i) + logp(\epsilon) \end{aligned}$$

$p(g_i|\epsilon_i)$ is considered to be have uniform probability among the different ways of converting a Greek word to Greelish. We will re-examine this assumption during our experiments.

## Dataset

Two issues during this project were creating a language model and a bilingual dataset.

In order to create a unigram Greek language model, I crawled the web sites of two popular Greek newspapers, "Ta Nea" (TaN ) and "To Vima" (ToV ). I retrieved 2GB of documents, extracted the article text and removed punctuation and HTML tags. This resulted in a corpus of nearly 5 million words, with 157,316 unique words. 73% of the words appear 5 times or less in the corpus.

A bigger problem is creating a bilingual corpus. Each document in the bilingual corpus consists of a Greeklish sentence, the equivalent Greek sentence and a mapping for each character. For this purpose I crawled 300 popular Greek videos on YouTube, extracting the comments. I manually removed comments that contained invalid characters and ASCII art, leaving a corpus of 350,000 words, mostly in Greeklish. Then, I created a web application for users to input data (add ), where a Greeklish sentence would be shown and the user would have to transliterate it to Greek and give the character-to-character mapping. Because of the error-prone nature of this procedure, I chose to retain only the sentences for which at least two users gave the same mapping and transliteration. Unfortunately, this brings the size of the dataset down to 911 words, a tiny number for meaningful evaluation.

On the positive side, the small size of the dataset allowed for manual inspection. 20% of the words are not Greeklish. Cases of ambiguity arise in our dataset for roughly 3% of all words. One example is the placement of the accent: The conjunction $\acute{\eta}$ ("or") and the pronoun $\eta$ ("the" for feminine gender nouns). Both are very popular words, but are used differently. An $n$-gram language model might help in distinguishing among the two – for example, conjunctions rarely appear at the beginning of the sentence). Unfortunately, this is not always true, as in the case of the word $\alpha\lambda\lambda\acute{\alpha}$ ("but") and $\acute{\alpha}\lambda\lambda\alpha$ ("other"). The only difference is the placement of the accent, but an $n$-gram language model will have a hard time telling those two words apart. Finally, roughly 10% of our words in the dataset contain gross spelling mistakes, a common phenomenon given the hasty nature of internet communication.

| Algorithm | Time | Speed | Accuracy |
|---|---|---|---|
| 1 | 34sec | 26.79 words/sec | 71.9% |
| 2 | 327sec | 2.78 words/sec | 75.0% |
| 1+ | 34sec | 26.79 words/sec | 80.1% |
| DeGreeklish | N/A | N/A | 73.8% |
| Innoetics | 6sec | 151.83 words/sec | 82.7% |

Table 1: Accuracy and speed of algorithms.

## Results

The results are shown in Table 1.

Algorithm 2 has improved accuracy because it tends to leave unknown words in their original format. When I tweaked the threshold of Algorithm 1 to reject improbable Greek words, I managed to improve the accuracy to 80.1% (see Algorithm 1+, above). Algorithm 2 has a great problem in 2-to-1 mappings from Greek to Greeklish, as it cannot detect that the 2-character combination should be treated as a single character, for transliteration purposes. By tuning the parameters and the 2-to-1 mappings specifically for the dataset, Algorithm 2 was accurate 83.4% of the times.

All in all, this dataset proved to be challenging even for commercial systems because of the significant number of spelling mistakes and non-Greek words. Algorithm 1+, despite its simplicity, works really well!

## Future work

In the immediate future I will develop two plugins for the popular Mozilla Foundation web browser, Firefox, and their e-mail application, Thunderbird. I hope to reuse my work from the `libpurple` plugin.

The public project page for this work is at `http://sourceforge.net/projects/gr2el`, were users can already download and build the plugin for the `pidgin` IM application.

## Acknowledgments

This research has been supported by a grant from EEL/LAK, a non-profit organization for free / open source software.

## References

Academy of Athens. Declaration against efforts for replacement of the greek alphabet by the roman.

Bilignual data insertion application. http://www.cs.wisc.edu/ sblanas/code/type.py.

Androutsopoulos, J. 1999. Lating-greek orthography in electronic mails: Use and stances. Technical report, Aristotle University of Thessaloniki.

Chalamandaris, A.; Protopapas, A.; Tsiakoulis, P.; and Raptis, S. 2006. All greek to me! an automatic greeklish to greek transliteration system. In *5th International Conference on Language Resources and Evaluation (LREC)*.

Galatas, G. 2008. Greeklish to greek converter based on lexicon structures. Master's thesis, University of Patras.

Who uses libpurple? http://developer.pidgin.im/wiki/WhatIsLibpurple.

Macrakis, M. S., ed. 1996. *Greek letters : from tablets to pixels*. Oak Knoll Press.

Ta nea. http://www.tanea.gr.

To vima. http://www.tovima.gr.