1. Introduction and Motivations

It does not happen often that the population of a country expands by 5% in a fortnight. Yet this kind of unprecedented demographic explosion was exactly what unfolded in Poland in the wake of Russia's aggression on Ukraine. Over the course of mere couple weeks over 2.5 million Ukrainian refugees crossed the Polish border looking for shelter and safety. It goes without saying that a change of this magnitude resulted in a host of logistical challenges encompassing food supply, schooling, healthcare, to name a few. One aspect that links all these problems is the necessity of initial registration that the refugees are subject to with the Polish authorities.

The issue at hand is not trivial as the majority of Poles do not know Ukrainian. Not only do they not speak or read it, but they also have little knowledge of the Cyrillic alphabet. Consequently, the majority of Poles tasked with receiving and registering Ukrainian refugees have a hard time decoding the names of the very people they are trying to help. Unfortunately, the available solutions are far from ideal. The supply of Ukrainian interpreters is inadequate to meet the demand, and besides, humans are slow and prone to making mistakes. Other analog solutions, such as dictionaries, provide little if any assistance when working with anthroponyms. Common sense may point to Google Translate, but even here the generated Polish transcriptions are frequently incorrect (link). Therefore there seems to be a need for a tool that would facilitate registration procedures by automating the transcription of Ukrainian names into Polish. It is the goal of this project to develop such a solution.

2. Proposed Solution

The proposed program would take a Ukrainian anthroponym, or a list thereof, encoded in the Ukrainian Cyrillic, and transcribe it into Polish. Here it is important to make a distinction between transcription and transliteration. Transliteration refers to a character transduction process where each Ukrainian Cyrillic grapheme is mapped onto exactly one grapheme derived from the Latin alphabet with optional diacritics (e.g. <Л> → <L> or <Ч> → <Č>). The Ukrainian soft-sign (Ь) is here accounted for by replacing it with an apostrophe (such that <Ь> → < ‘ >). All stress markers are left intact. The full list of transliterations from Ukrainian to Polish is available at: (link). The Polish transliteration standard is codified under PN-ISO 9:2000, which in turn is based on the international ISO 9:1995. As such, the standard is used mainly in contexts devoid of Polish graphemes (e.g. personal documents designed for global use) (Starzak, n.d.). Transcription on the other hand is more applicable in Polish-speaking contexts, as it is informed by phonological and orthographic conventions. Not only does it employ Polish graphemes, digraphs, and plenographs when appropriate, but it also takes into account each token's surrounding context. For instance, the Ukrainian <Ю> can be transcribed as either <Ju>, <U>, or <Iu>, depending on whether it occurs word-initially, is preceded by vowels, consonants, an apostrophe, or the letter <Л>. The full list of correspondences in transcription can be accessed at: (link). The list of rules governing each correspondence is available at (link). The rules for transcription used for this project were published by Wydawnictwo Naukowe PWN (Polish Scientific Publishers PWN, henceforth PWN) in the Dictionary of the Polish Language (Słownik Języka Polskiego).

At this point it may also be helpful to examine any similar solutions already in place. At the moment there are only two free online tools that automate Ukrainian-to-Polish grapheme transduction. The first one, *Grafiati* (https://www.grafiati.com/pl/transliteration/) is a plausible solution, but only for transliteration. At the time of testing, no transcription functionality was available. The second tool, *Ushuaia* (https://www.ushuaia.pl/transliterate/), offers both transcription and transliteration available in multiple languages and scripts. While this solution covers the most ground, it requires the user to select the transcription or transliteration standard appropriate for the occasion. Such a requirement assumes that the user is familiar with and aware of the differences between the available options - a risky assumption at best. In

light of the above, a simple and accessible tool may come a long way in making a positive impact on the current refugee situation.

3. Software

The program was developed with `Pynini`, a Python library for compiling a grammar of strings, regular expressions, and context-dependent rewrite rules into weighted final-state transducers. It takes a Ukrainian anthroponym string as input and transcribes it into its Polish equivalent using a set of ordered, context-dependent rewrite rules. Each rule is expressed as a final-state transducer (FST) using `Pynini`'s `cdrewrite()` function. Each `cdrewrite` FST is described with a four-tuple `x = (`$\tau$`, `$\lambda$`, `$\varrho$`, `$\Sigma^*$`)`, where:

- $\tau$ represents the transduction ("$a$" → "$b$")
- $\lambda$ represents the left context
- $\varrho$ represents the right context
- $\Sigma^*$ represents the closure over the alphabet

Optionally, `cdrewrite()` can also take the `direction` and `mode` parameters, the first of which specifies the direction of the rule application (*L* → *R* or *"*`ltr`*"* in default), and the second declares if the rule should be optional (*"*`opt`*"*) or obligatory (*"*`obl`*"* in default). Seeing how all transcription rules consulted for this project apply from left to right, and how they are all obligatory, there is no need to explicitly declare these two parameters. Thus, an example rule FST that transcribes "щ" to "szcz" can be encoded as:

```
pynini.cdrewrite(pynini.cross("щ", "szcz"), "", "", SIGMA_STAR),
```

where, the transduction $\tau$ = `pynini.cross("щ", "szcz")` and the closure over the alphabet Σ* = `SIGMA_STAR`. Since this particular rule is context-independent (i.e. applies everywhere), both the left context $\lambda$ and the right context $\varrho$ are assigned empty strings.

Before diving into the transcription rules, it may be useful to discuss the alphabets, over which `SIGMA_STAR` represents closure. Contained in `g2g_alphabets.py` are Ukrainian and Polish alphabet sets. Each alphabet is an FST representing a union of that alphabet's upper and lower case symbols. Ukrainian vowel and consonant sets were isolated so as to facilitate rewrite rule composition. The final alphabet also contains a `punctuation` set of characters commonly used with anthroponyms (space, hyphen, and comma).

The rules for transcription can be found in `g2g_rules.py`. Each line represents an individual rule composed based on the PWN specification outlined below. The first cluster of rules handles surname suffixes. As they transform strings consisting of a number of graphemes, the rules in this cluster need to apply before any of the component graphemes are transcribed. They are therefore ordered first in the sequence:

- Surname suffixes
    - The suffix "-ський" is transcribed as "-ski"
    - The suffix "-цький" is transcribed as "-cki"
    - The suffix "-ий" is transcribed as "-y"
    - Feminine surname suffixes can be transcribed by the letter.

The rules for iotated vowels "Є", "Ю", and "Я" follow a similar pattern, which allows them to be grouped together. It is important to note here that in the context of anthroponyms, certain rules will be applicable only to lower-case graphemes. In particular, rules that are triggered by a preceding grapheme, will by default only apply to lower-case characters, as only the first character in a word token is assumed to be capitalized.

- The rules for “Є” and “є” transcription
    - “Є” and “є” are transcribed as “Je” and “je” respectively, word-initially and after vowels.
    - “Є” and “є” are transcribed as “E” and “e” after “Л” and “л”.
    - “Є” and “є” are transcribed as “Ie” and “ie” after consonants except for “Л” and “л”.
- The rules for “Ю” and “ю” transcription happen to be similar to those for or “Є” and “є”:
    - “Ю” and “ю” are transcribed as “Ju” and “ju” respectively, word-initially and after vowels
    - “Ю” and “ю” are transcribed as “U” and “u” respectively, after “Л” and “л”.
    - “Ю” and “ю” are transcribed as “Iu” and “iu” respectively, after consonants except for “Л” and “л”.
- The rules for “Я” and “я” transcription follow the pattern of the previous two characters:
    - “Я” and “я” are transcribed as “Ja” and “ja” respectively, word-initially and after vowels.
    - “Я” and “я” are transcribed as “A” and “a” respectively, after “Л” and “л”.
    - “Я” and “я” are transcribed as “Ia” and “ia” respectively, after consonants except for “Л” and “л”.

The above pattern reveals a number of regularities governing the transcription of the iotated consonants. Among them is the importance of “Л” and “л” in triggering these transductions. Seeing how the presence of “Л” and “л” feeds the above rules, it is necessary to order them before the rules applicable to “Л” and “л”. The rules for transcribing the latter are enumerated below:

- “Льо” and “льо” are always transcribed as “Lo” and “lo” respectively.
- “Л” and “л” are transcribed as “L” and “l” respectively before "я", "є", "ю", "і", "ї", and "ь".
- “Л” and “л” are transcribed as “Ł” and “ł” respectively before "а", "е", "и", "о", "у", consonants, and word-finally.

The remainder of the characters is for the most part context-independent. The only exception here are the rules governing soft-sign ("ь") palatalization. When followed by the soft-sign, the consonants “З”, “С”, “Ц”, and “Н” are transcribed in the following way:

- “Зь” and “зь” are transcribed as “Ź” and “ź” respectively.
- “Сь” and “сь” are transcribed as “Ś” and “ś” respectively.
- “Ць” and “ць” are transcribed as “Ć” and “ć” respectively.
- “Нь” and “нь” are transcribed as “Ń” and “ń” respectively.

Lastly, when preceded by the soft-sign, “о” is always transcribed as “io”, such that: <o> → <io> / <ь> _ .

The rules ordered after soft-sign palatalization are summarized (here). They are all context independent and apply to individual Ukrainian characters. The last two rules in the chain resolve all the outstanding soft-signs and apostrophes. Thus created FST grammar `G2G` is then optimized in-place with the `.optimize()` function.

With `g2g_alphabets.py` and `g2g_rules.py` developed, we can move to the `main()` method. Included in `transcribe.py`, are two functions. The first, `ukr2pl()`, takes a single Ukrainian `str` argument. This string is then passed on through `rewrite.one_top_rewrite()`, and transduced into its Polish transcription using the `G2G` grammar imported from `g2g_rules.py`. Thus obtained transcription is stored in the `polonized_string` variable and subsequently returned. The `main()` method first prompts the user for an input string in Ukrainian. The input string is saved under `ukr_name`, which is then passed through the `ukr2pl()` function defined above. The Polish transcription is stored under `pl_name` and subsequently printed as part of a longer print statement. The module has a `main` guard clause to indicate that the script is the entry point for the program.

4. Input Data

The data used for this project consists of a list of Ukrainian anthroponyms (names, surnames, and last names). Middle names, patronymics, and hyphenated anthroponyms are also supported. It is important to make sure that the input anthroponym strings are in Ukrainian cyrillic as the cyrillic alphabets used by other languages will not transcribe correctly. The program assumes standard capitalization, where the first letter of each word token in the input string is uppercase. It will also accept all-lowercase strings but it is not optimized to handle all-uppercase input.

5. Testing Data

The program was tested against data from the Polish Wikipedia. A sample of anthroponyms identifying prominent Ukrainian political figures was selected. The selection was made based on the assumption that due to their importance, political leaders are more likely to have their names and last names correctly transcribed. In an effort to make it more representative, the sample included politicians past and present. Each politician's Wikipedia profile opened with the Polish transcription of their first and last name in the header (link), followed by the original Ukrainian spelling.

Unfortunately, as it can be the case with Wikipedia, the data were rather inconsistent. There observed inconsistencies could be roughly grouped into the following categories:

- Some of the selected individuals were only cataloged under their Polish names, without the cyrilic transcription provided (example).
- For others, a cyrilic transcription was available, but not in Ukrainian (example).
- Others still did list the original Ukrainian anthroponym, but their Polish transcription did not follow the official transcription rules (e.g. Роман Перфецький should have been transcribed as *Roman Perfećki*, and not as *Roman Perfećkyj* (link)).

It may be useful to point out that the above irregularities would surface largely for historical figures or Ukrainian politicians of non-Ukrainian (e.g. Russian) background. Whereas the latter may be self-explanatory, the former could be justified by the fact that historically the rules of Ukrainian transliteration and transcription into Polish had been subject to many changes, and did not begin to coalesce until the late twentieth century (Sojka-Masztalerz, 2015). Consequently, and perhaps unsurprisingly, the automatically transcribed Ukrainian anthroponyms tested against the Wikipedia-sourced Polish transcriptions that did not reflect the current transcription rules would fail the test. In contrast, the Polish transcriptions that followed these rules were identical to their automatically generated counterparts. The test sample of Ukrainian anthroponyms and their Wikipedia-sourced Polish transcriptions is available in `transcribe_test.py`.

6. Evaluation

In order to test how well the program applies PWN's rules to transcribe from Ukrainian to Polish, it was essential that the data gathered reflected the current transcription rules. The anthroponyms used for evaluation were selected at random, from among the names transcribed against the current standard. Thus collected data were tested using the `unittest` module, where thirty separate tests were performed. Each test case checked the output string produced by the `transcribe()` function against the Polish data extracted from Wikipedia. The test would pass if the automatically generated string was identical to the Polish benchmark. Running `transcribe_test.py` demonstrates that the program generated correct transcriptions in all test cases.

7. Future development

The idea for this project emerged out of conversations with my relatives and friends living in Poland. Faced with a historically unprecedented scenario, the Polish have suddenly found themselves navigating a multitude of challenges - registration with the authorities being just one of them. While there is room to make the process more efficient via automation, the tools utilized to that end need to be widely available and easy to use. Unfortunately, in its current shape this program is neither. One way to make it more user-friendly and accessible is to deploy it as an online application working out of a web browser. To that end I was thinking about utilizing Flask - an online framework allowing for development of fully featured web applications in Python. Once developed, the transcriber can be deployed on Heroku, and shared with the world.

The functionality of the program may also be expanded to include transliteration. While the Polish transcription is obviously relevant in the Polish context, transliteration is more applicable internationally. It would be useful if both options were available to the user. Somewhat related, and a bit more difficult to implement, would be including support for all proper nouns (e.g. geographic locations, administrative regions, cities, etc.). Here the challenge may lay in correctly navigating between transliteration, transcription, and translation. For instance, the city of “Львів” transliterates to “Lviv”, transcribes to “Lwiw”, but translates to “Lwów”. Being able to generate the correct translation on top of providing a reliable transcription and transliteration may help the Polish get to know their new neighbors beyond their transcribed names and contribute to making the current situation a bit more manageable.

## Works Cited

Państwowe Wydawnictwo Naukowe. (n.d.). Transliteracja i transkrypcja współczesnego alfabetu ukraińskiego. Słownik Języka Polskiego PWN. Retrieved May 21, 2022, from https://sjp.pwn.pl/zasady/Transliteracja-i-transkrypcja-wspolczesnego-alfabetu-ukrainskiego;629710.html

Sojka-Masztalerz, H. (2015). Transkrypcja czy transliteracja nazwisk ukraińskich w polszczyźnie? Z dziejów ortografii. Studia Ukrainica Posnaniensia, (3), 295-301.

Starzak, Ł. (n.d.). Alfabet Ukraiński. Alfabet ukraiński. Retrieved May 23, 2022, from https://www.kul.pl/files/602/transkrypcja/Alfabet_ukrainski.pdf