UDC 81:004.9
DOI https://doi.org/10.32782/tps2663-4880/2021.16.44

USING THE BASICS OF REGULAR EXPRESSIONS IN TRANSLATION AND TEXT PROCESSING

ВИКОРИСТАННЯ ОСНОВ РЕГУЛЯРНИХ ВИРАЗІВ ПІД ЧАС ПЕРЕКЛАДУ ТА ОБРОБКИ ТЕКСТУ

Ivashkevych L.S.,
orcid.org/0000-0001-7166-5331
Candidate of Philological Sciences, Associate Professor,
Associate Professor at the Department of Theory, Practice and Translation of German
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

The article shows why it is worth introducing the basics of regular expressions in translation curricula. It briefly explains what regular expressions are and presents simple examples of their functionality, illustrating how regular expressions help execute search-and-replace tasks, not accessible for the most commonly used WYSYWIG-text processors.

Regular expressions can be widely used in translation processes, namely while defining parsers and segmentation rules; while searching certain words or strings and replacing them; extracting strings for translation; creating checklists for quality assurance; post-editing machine translation, improving the quality of translation memories. Here we show how regular expressions used in a CAT-tool environment help accomplish such tasks as finding different spellings of the same word, finding word forms, finding the source and target segments with different capitalization, replacing the decimal separator, finding and deleting the odd article in the machine-translation output.

The basics of regular expressions are explained in the article divided into five main groups: literal characters, metacharacters, quantifiers, characters classes, and groups. Each of these groups is an essential part of regular expressions functionality. Even each one of them separately can significantly widen the search-and-replace possibilities and simplify other text-processing tasks. In many cases, even using one single tool from the regular expressions toolbox can measurably save time for text processing.

Besides, regular expressions are also integrated as a tool in many text-processing surroundings like corpus managers or text editors. They are widely used in programming in the form of special libraries. Several examples of such usage are also given at the end of the article.

Key words: regular expressions, translation processes, text processing, programming, corpus managers.

У статті продемонстровано, чому основи регулярних виразів варто включати до програм підготовки перекладачів. Стаття коротко розповідає про те, що таке регулярні вирази та яка їхня функціональність, ілюструючи це простими прикладами, як регулярні вирази дозволяють істотно розширити можливості пошуку, наявні у звичайних текстових редакторах, що працюють за принципом WYSYWIG (what you see is what you get).

Регулярні вирази широко використовуються в перекладацьких процесах, а саме у таких його етапах, як визначення правил парсингу та сегментування, пошук певних слів та текстових фрагментів та їх заміна, екстрагування тексту для перекладу із коду чи тегів, створення списків частотних для певних мов та мовних пар помилок для забезпечення кращої якості перекладу, постредагування машинного перекладу, покращення якості та консистентності перекладацьких пам'ятей. У статті наведені приклади кількох із цих завдань, виконані у середовищі програмного забезпечення для перекладу.

Стаття пропонує ознайомитися з основами регулярних виразів, для зручності розділивши їх на п'ять головних категорій: буквальні символи, метасимволи, квантифікатори, класи символів та групи. Опанування навіть однієї з них може істотно розширити можливості функціоналу пошуку та заміни та полегшити виконання інших завдань, пов'язаних із обробкою текстів, що допоможе відчутно заощадити час.

Крім цього, регулярні вирази інтегровані в багато середовищ для роботи із текстами, як-то корпусні менеджери та текстові редактори. У більшості мов програмування їх функціонал використовується через імпорт спеціальних бібліотек. У статті наведені два приклади використання регулярних виразів у текстових середовищах – у корпусному менеджері та інтерактивному середовищі для програмування repl.it під час програмування мовою Python.

Ключові слова: регулярні вирази, перекладацькі процеси, програмування, корпусні менеджери.

Introduction. According to the Oxford Handbook of Computational Linguistics, regular expressions are expressions that describe a set of strings or a set of ordered pairs of strings [8]. In other words, a regular expression is a pattern describing a certain amount of text. Regular expressions were developed in the 1950s by the American mathematician Stephen Cole Kleene, and their name comes from the mathematical theory on which they are based. Since 1980, two foremost regular expression syntaxes were developed, POSIX and Perl [2; 9].

Regular expressions are widely used in text processing tasks [3]. Their typical applications include, f.e., data scraping, data validation, data wrangling, parsing, production of syntax highlighting systems, etc. [6]. Regular expressions are integrated into many working surroundings like text editors, translation tools, corpus managers and are also used in program-

ming processes. Almost all programming languages provide the functionality of regular expressions either built-in or via libraries.

The **purpose of this article** is to demonstrate, that the functionality of regular expressions can essentially widen the possibilities of text processing, significantly saving time need for different kinds of tasks and should be therefore implemented into the linguistics and translation curricula.

Discussion. Regular expressions allow effectively search and replace certain strings in texts of any type. Here are some simple examples of what one can match in text using the very basics of regular expressions.

In Figure 1 you can see telephone numbers in different formats. Imagine you have to match them all at once in your text and, for example, copy and paste them into your database. The Figure 2, the way is shown how you can do this with the help of a regular expression typed below in red. Although this regular expression can look complicated at first glance, it is very simple and clear after you learn the basics of regular expression syntax.

In Figure 3 you can see how one can match all long words (in this case, words consisting of 12 letters and more) in a text with another simple regular expression.

There are many resources that can help you to learn the basics of regular expressions and to practice them. I want to recommend you an online tutorial RegexOne.com, where you are successively introduced to the principles of regular expressions based on practical tasks, and an online editor RegExr.com, where you can conveniently search with regular expressions in your texts. Though, if you need to work with big texts, you should install some desktop text editor, which will not have restrictions on the text length, f.i. Atom.

1. Regular expressions in translation processes

Regular expressions are used in many different stages of translation processes [7], among others in following:

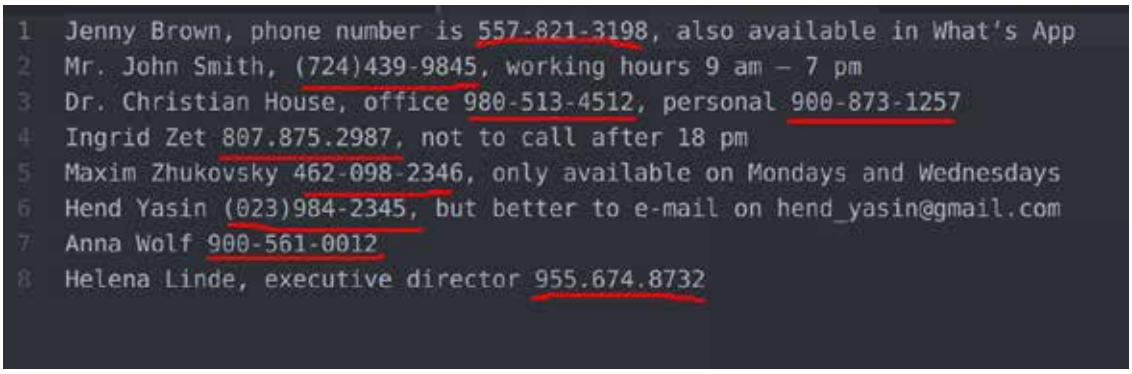

Fig. 1. Task of finding telephone numbers of different formats

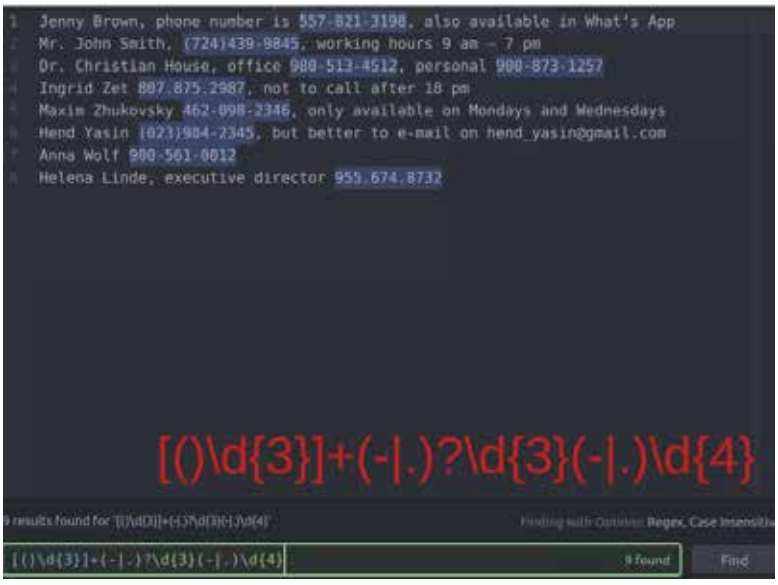

Fig. 2. Finding telephone numbers of different formats with regular expressions

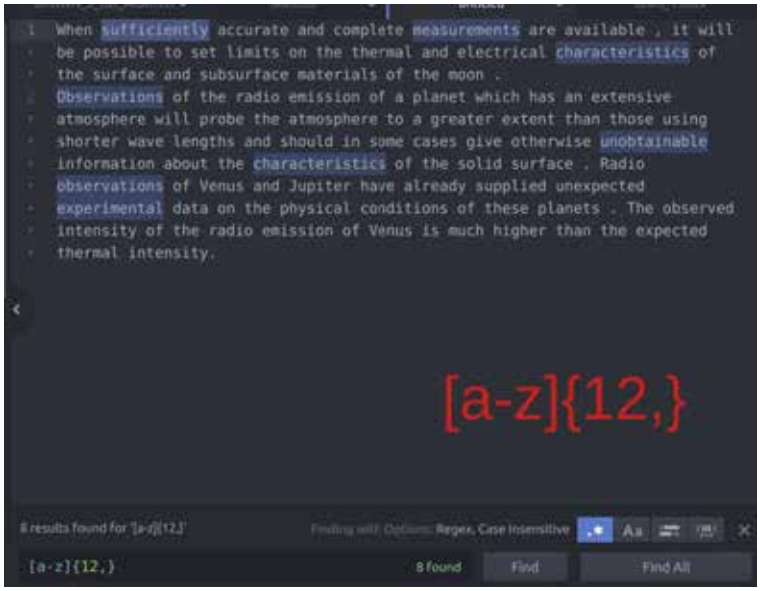

Fig. 3. Finding all words of 12 and more letters

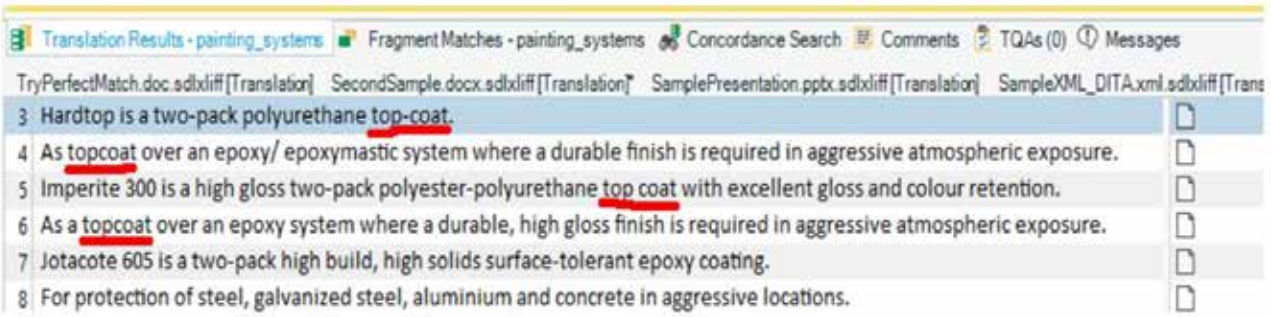

Fig. 4. Task to find different spellings of the word

– in parsers and segmentation rules;
– searching certain words or strings;
– replacing words or characters;
– extracting strings for translation;
– creating checklists for QA;
– QA of translation memories [4];
– post-editing MT [5; 1].

Below I will show some examples of how exactly they can be used.

1.1. Finding different spellings of the same word

Regular expressions can help you effectively process your text when you have to repeat some actions with strings many times.

Thus, if your text contains different writing variations of the same word as shown in Figure 4, you can easily match all of them and replace them with one form to reach consistency in the whole text. For that, you just use a simple regular expression consisting of two signs: "." means any symbol, and "?" means that the previous symbol is optional (Fig. 5). In such a way you can find all three forms, "top-coat", "topcoat" and "top coat".

In Figure 6 you can see how you can with the help of regular expressions replace non-consequent spelling of the word with the consequent one.

1.2. Finding word forms

Not all CAT-Tools support flexions search. If you work with inflected languages, it might be beneficial for you to use simple regular expressions to find all the word forms you need at once. Sign "." means any digit, sign "+" means one or more times. So, their combination helps you to find the stem "модул" with any possible continuation (Fig. 7).

1.3. Finding the source and target segments with different punctuation

Regular expression can also be used to check whether the source and target sentences have the same punctuation. Thus, in Figure 8 you can find all segments in your translation, where the source has a dot in the end and the target does not, and correct it.

1.4. Finding source and target segments with different capitalization

Thanks to the option to refer to the beginning of the line, regular expressions can also be helpful while

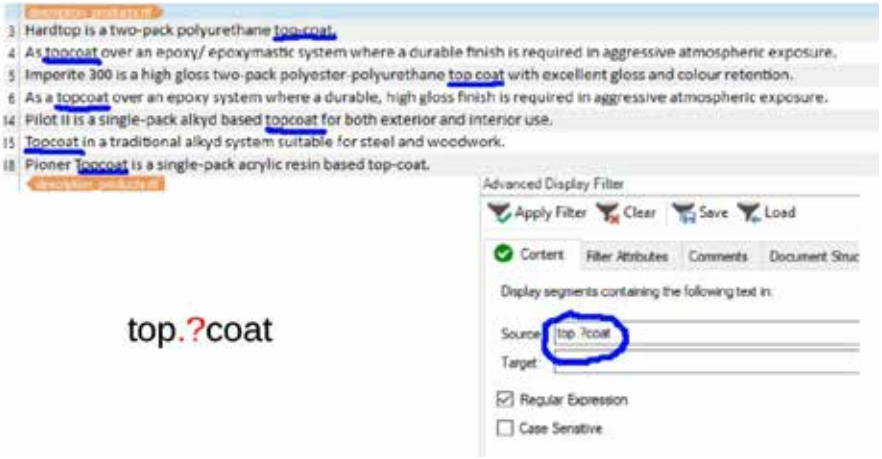

Fig. 5. Finding different spellings of the word with regular expressions

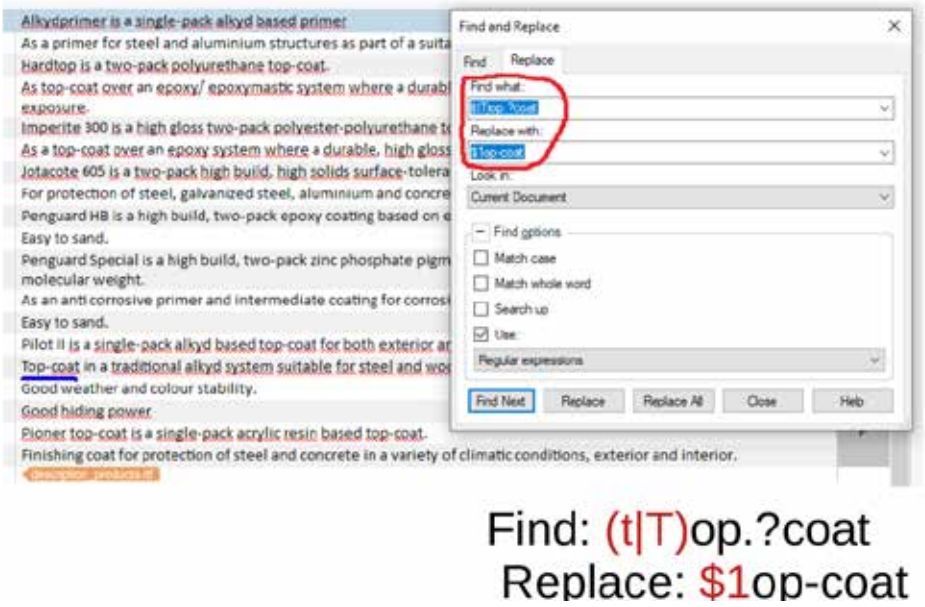

Fig. 6. Considering the different capitalization while searching and replacing

checking whether segments have equal capitalization. In Figure 9, you can see an example where target segments start with lowercase words, and simple regexes used within a CAT-Tool help identify such cases.

1.5. Replace the decimal separator

In Figure 10 you can see the regular expression with which we can easily change the decimal separator in different document segments at once. In the Find line, you have two groups consisting of one or more digits, and these groups are divided with a comma. In the Replace line, you refer to these groups with the sign $ and their number and put a dot between them, replacing the comma as a decimal separator.

1.6. Find and delete the odd article in the machine translation output

Machine translation output often contains reoccurring mistakes. You can use regular expressions to correct particular type of such mistakes. For example: in Figure 11, you can see the odd article "el," which was placed before the different names of the operating systems: Windows, Linux, Mac OS. With regular expressions, you can remove the article in all these cases. Thus, in Figure 12 in the Find line, we are looking for all three names of operational systems alternatively, before which "el" is placed, having built a corresponding group, and replace it with the group only, without article, in the line Replace.

2. The basics of regular expressions

Regular expressions are not difficult to learn, and even their basics can be very useful in translation processes. It is important to understand the principles how regular expressions work and not to learn all the technical signs and their functionality by heart. To better conceive the functionality, it may be helpful to divide the characters, which are used while writing regular expressions, into several categories.

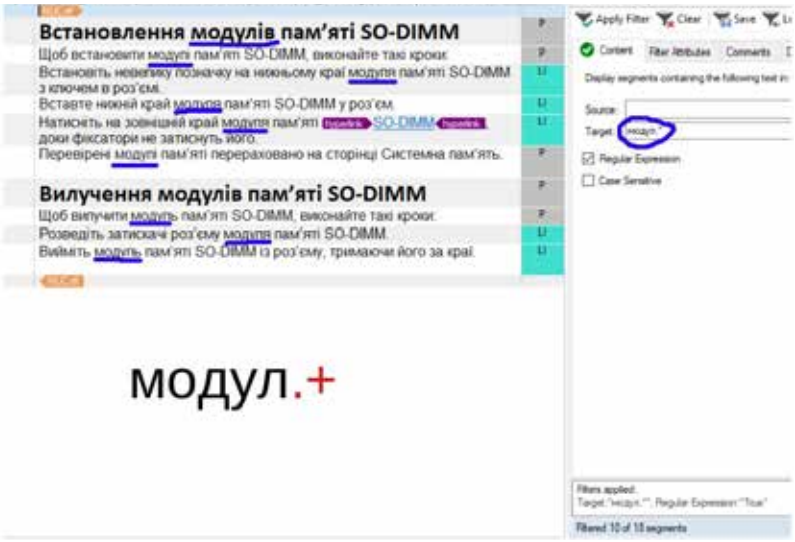

Fig. 7. Finding word forms in inflected languages

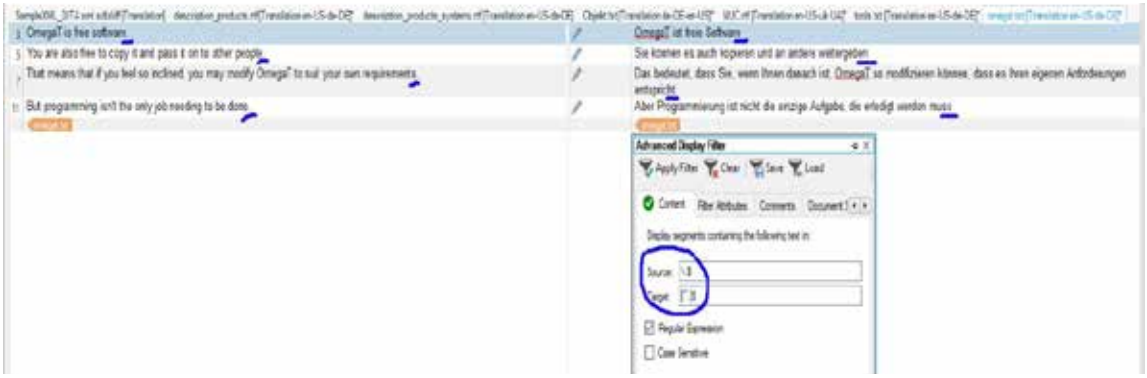

Fig. 8. Checking punctuation in source and target segments

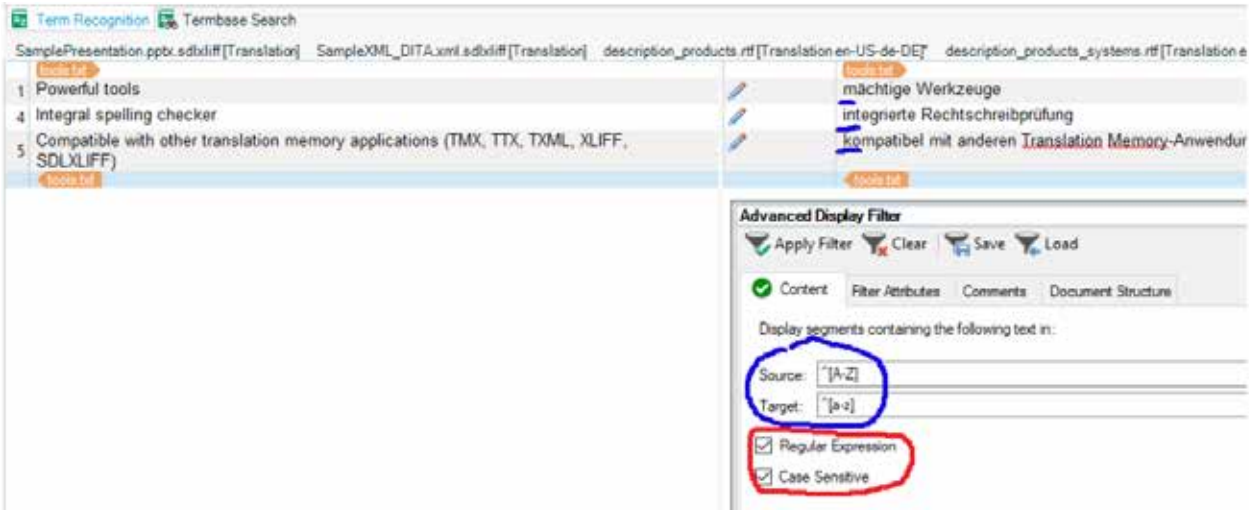

Fig. 9. Checking capitalization in source and target segments

2.1. Literal characters

These are the actual characters contained in the string we need to find, f.i., letters, numbers, punctuation etc. For this category, there is one special aspect: those characters, for which some regex functionality is reserved, must be escaped with a "\" sign, so that to search for a dot, you have to type "\." and not just ".".

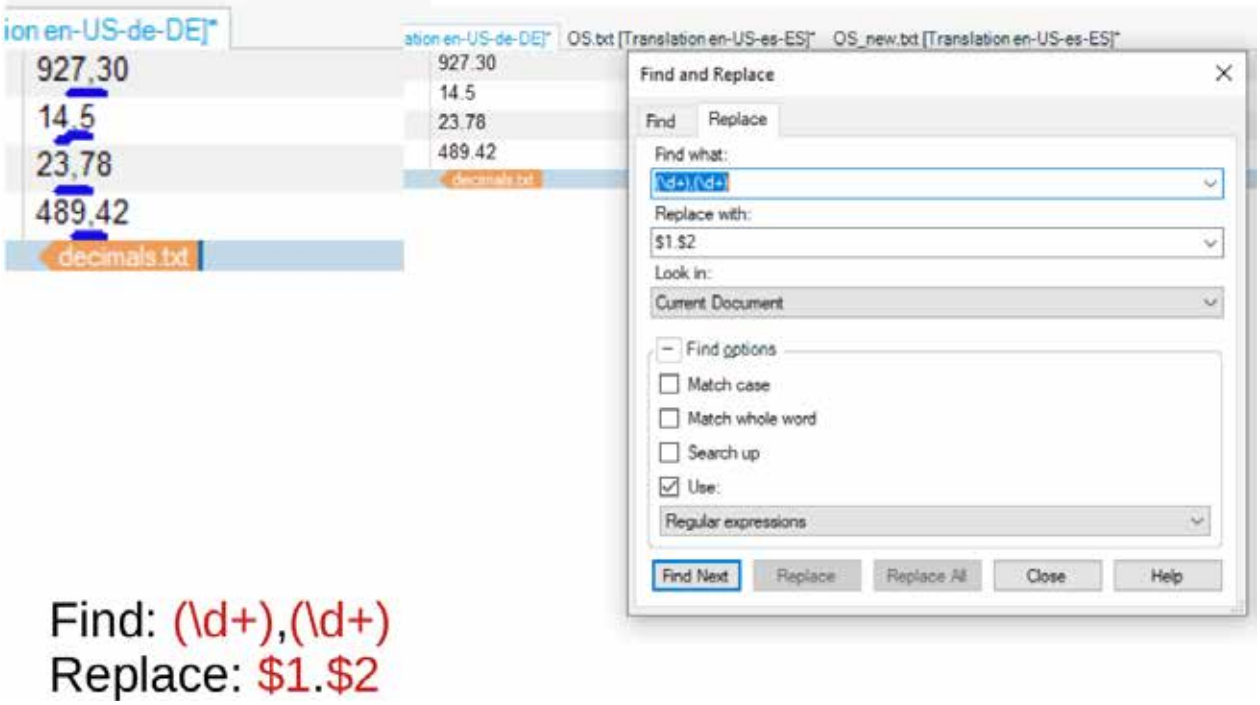

Fig. 10. Replacing the decimal separator

Fig. 11. Odd article in the MT-output

However, if you, for example, want to find all the prepositions “for” in a text, the usual search finds also “for” as a part of the word. You can add spaces around “for”, but then you won’t find places with punctuation after it. To solve this problem, we have a further category of the signs in regular expressions.

2.2. Metacharacters

These characters help us to find some classes of signs in a string. Thus, \d finds all digits, whereas \D., to the contrary, finds all non-digit characters. Similarly, \w finds all alphanumeric symbols, whereas \W finds all non-alphanumeric symbols.

Next, \s finds spaces, \S finds non-spaces, \b finds all word boundaries.

So, to find for only as a separate word even before some punctuation, you have to use the word boundaries: \bfor\b.

Another very widely used metacharacter is “.”, which stands for any character. So, for example, the search \ bth.n\b will find “thin”, “than,” and “then” at once.

2.3. Quantifiers

Quantifiers allow us to specify the quantity of the particular character or group of characters. There are several ways to indicate it:

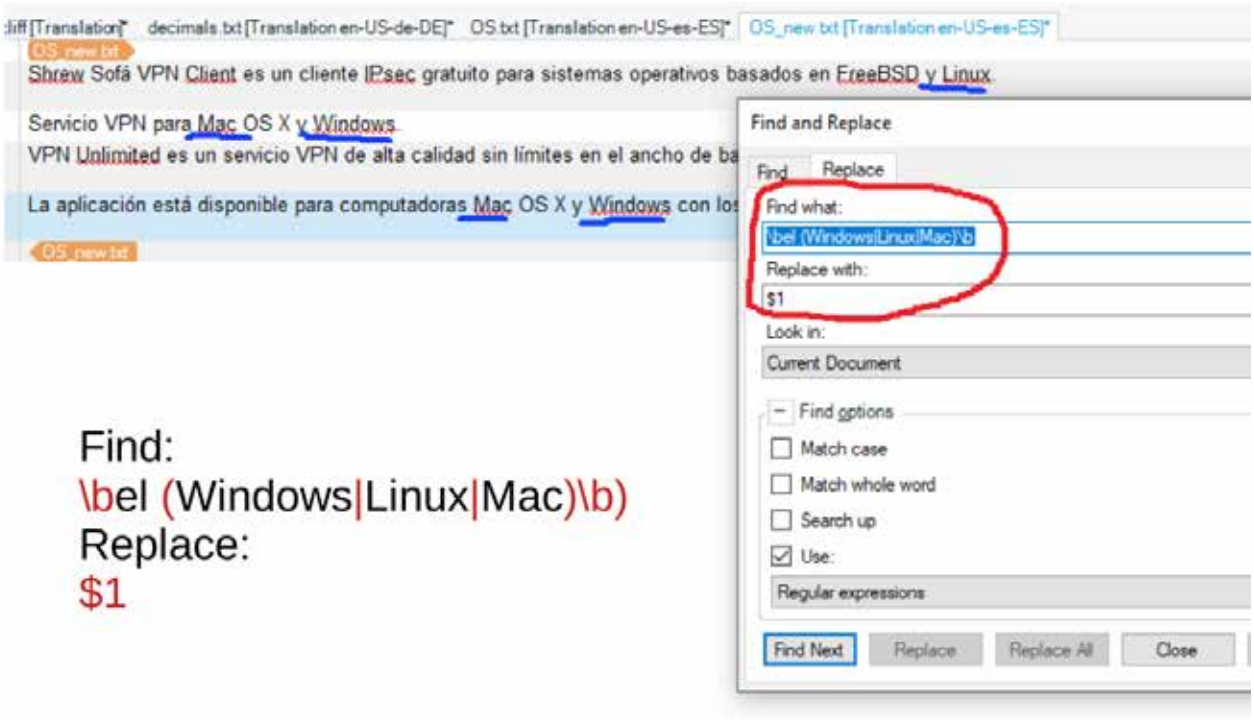

Fig. 12. Removing the odd article while post-editing

```
@misc{diaz2006automatic,
  title={AUTOMATIC ROCKING DEVICE},
  author={Diaz, Navarro David and Gines, Rodriguez Noe},
  year={2006},
  month=jul # "~12",
  note={EP Patent 1,678,025}
}
```

Fig. 13. Task to extract a string for translation

* stands for 0 or more repetitions of the character. Thus, \btaken*\b finds strings like *take, taken, takenn, takennn*.

+ stands for one or more repetitions. For example, \d+ finds numbers of one and more digits like *3, 34, 345, 3456*, etc.

? stands for optionality, i.e., 0 or 1. Thus, search \bstudents?\b finds both forms *student* and *students*.

{} allow specifying the particular number of the characters. Thus, ll{2} means we want to find all double ll in words, while \b[a-z]{6,9}\b means we search for all words with the length of 6 to 9 characters. The form \b[a-z]{9,}, further, finds all words of 9 and more characters.

2.4. Character classes

Square brackets help us to search for some range of characters. In particular, if you just list the characters within square brackets, you then search any of the listed ones. Thus, [abc] find any letter a, b or c only one time, [new] finds one of the listed characters one time, etc. If you use a dash, you can search within some range: [a–z] finds any letter from a to z and [0–9] – any digit from 0 to 9. Circumflex helps to search all except for the listed characters: [^a–k] finds all letters after k in the alphabet.

2.5 Groups

Parentheses help to find precisely this particular string. F.i. \b(new)\b finds exactly the word new and nothing else. Besides this, parentheses allow us to operate with these groups by backreferencing them with the sign $ and the number of the group. Thus, to extract the author in the metadata in Fig. 13, we can use the expression author={([\w\s,]+)} and then replace it with the group backreference $1. Using groups is the perfect way to extract from text the necessary data.

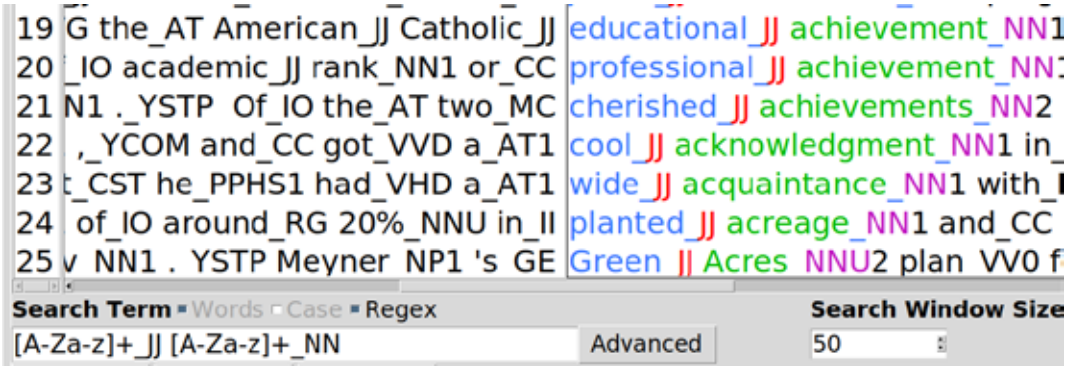

Fig. 14. Regular expressions functionality in corpus manager

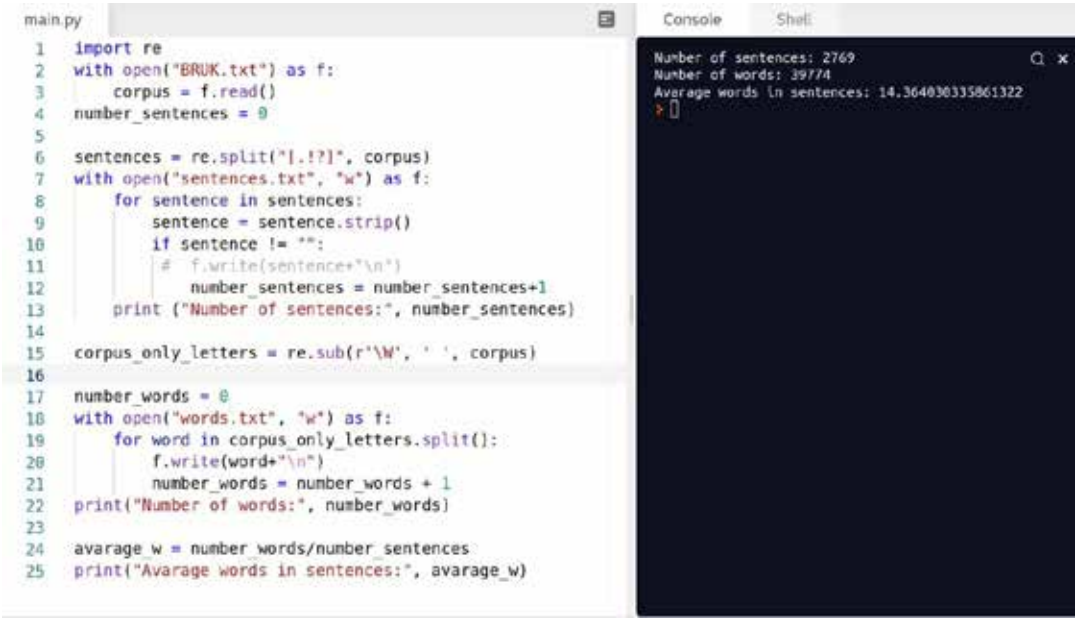

Fig. 15. Regular expressions used in Python code

3. Regular expressions in text processing environments

We have already described above, in section 2, how regular expressions can be used in translation processes. Except for this, they are also built in many text processing environments like corpus managers or text editors. Programming languages, too, have special libraries for regular expressions, which allow using their functionality while writing code.

In Fig. 14 you can see an example of how regular expressions can be used in the AntConc corpus manager for searching all the constructions "adjective+noun" in the annotated corpus:

In Fig. 15 you can see an example, how one can import a regular expression library while writing code in Python and use regular expressions to split the corpus in sentences by such punctuation marks as ".", "!" and "?".

After learning the basics of regular expressions, presented above, you can already use them for your specific purposes while processing text in different ways.

Conclusions. As we can see, the basics of regular expressions are not complicated to learn and they can give us a lot of advantage while processing text in many different surroundings. Thus, we consider it to be reasonable to integrate the course of regular expressions into the translation and linguistic curricula.

REFERENCES:

1. Arenas A.G., Moorkens J. Machine translation and post-editing training as part of a master's programme. *The Journal of Specialised Translation.* 2019. Pp. 217–238.

2. Câmpeanu C., Santean N. On the intersection of regex languages with regular languages. *Theoretical Computer Science.* No. 410 (2009), Pp. 2336–2344. URL: https://core.ac.uk/download/pdf/82684254.pdf.

3. Dorosz K., Szczerbińska A. Enhancing regular expressions for Polish text processing. *Computer Science.* Vol. 10, 2009. Pp. 19–35.

4. Gintrowicz J., Jassem K. Using Regular Expressions in Translation Memories: Systems Science. January 2008. Pp. 87–92.

5. Guzmán R. Automating MT post-editing using regular expressions. *Multilingual.* No. 90. 2007. Volume 18. Issue 6. Pp. 49–52.

6. Mosel U. Advances in the accountability of grammatical analysis and description by using regular expressions. *Language Documentation & Conservation Special Publication*. No. 4 (October 2012). Electronic Grammaticography ed. by Sebastian Nordhoff. Pp. 235–250.

7. Regular Expressions: An Introduction for Translators. *The ATA Chronicle.* American Translators Association. URL: https://www.ata-chronicle.online/highlights/regular-expressions-an-introduction-for-translators/.

8. The Oxford Handbook of Computational Linguistics. OUP Oxford. 2004. 786. P. 754.

9. Wang P., Bai G.R., Stolee K.T. Exploring Regular Expression Evolution. *Computer Science.* 2019. URL: https://wangpeipei90.github.io/papers/saner2019_preprint.pdf.