# Utok: The Fast Rule-based Tokenizer

Pavel Rychlý and Samuel Špalek

Natural Language Processing Centre
Faculty of Informatics, Masaryk University
Botanická 68a, Brno, Czech Republic
pary@fi.muni.cz

**Abstract.** Tokenization is one of the first processing steps in most natural language processing applications. The papper introduces a new tokenizer `Utok` which follows the Unitok tokenizer in the form of simplicity of configuration for different languages and is much faster in processing speed.

## 1 Introduction

Tokenization is a process of breaking down text data into minimal meaningful elements called tokens. That gives the machine ability to analyze each element in the context of other elements. It is one of the first tasks in the NLP text processing pipeline.

Most basic tokenizers use a simple idea of splitting text based on whitespace. That works well enough for some cases. The rest of the cases are more complicated, language-dependent, and require special treatment.

Examples of tokens we might want to recognize are:

- Words
- Numbers
- Dates
- Punctuation characters such as `( ) . , [ ] { }`
- Abbreviations (Etc., Jan., Mr., Gov.)
- Email
- URL
- SGML tags (HTML, XML, ...)

Tokens can also contain metadata. Most basic metadata would be a token position in the original text. Some tokenizers even categorize tokens into groups (e.g., POS tagging). We will focus on tokenizers that only separate tokens. Additional metadata can be added later upon further processing.

This paper introduces a new tokenizer: `utok`. It tries to solve to main problem of the `unitok` tokenizer: a slow processing of certain types of inputs. The paper also describes differences between `unitok` and `utok` tokenizers.

A. Horák, P. Rychlý, A. Rambousek (eds.): Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN 2022, pp. 149–154, 2022. 

## 2 Tokenizers

### 2.1 Unitok

Unitok [4] is a tokenizer that takes text stream and outputs tokens separated by a newline. It is written in python language and uses the native python library for regular expressions (regexes). That results in not ideal performance. Configuration for a specific language is defined in the python code. Therefore configuration files are not universal and cannot be used out of the box in a different tokenizer.

Unitok supports many languages, such as Czech, Danish, Dutch, English, French, Finnish, German, Greek, Hindi, Italian, Maldivian, Spanish, Swedish, Yoruba. Other languages can be processed with default settings.

**Glue tag** In cases where there are two tokens next to each other, not separated with whitespace, a special glue tag `<g/>` is used.

An example of such case would be a dot at the end of a sentence. The last word in the sentence and the dot at the end is not separated with whitespace. Unitok produces tokens: `last_word` `<g/>` `.`

Using the glue tag makes the tokenization reversible. It is possible to reconstruct original text from unitok output (same for utok).

**Configuration files** The `unitok` requires a configuration file for each language. It contains a list of regular expressions representing different types of tokens and the order of evaluation of that requlat expressions. The form configuration file has a form of a Python source file, any Python constructs could be used.

Configuration of many languages uses advanced techniques like look-ahead to define some tokens. For example the following regular expression is used to describe order numbers in English:

```
ORDINAL = ur"""
(?<![-\w])
    ( \d+th | \d*(1st|2nd|3rd) )
(?![-\w])
"""
```

**Processing method** The main part of the Unitok algorithm processes the input line by line and tries to find selected tokens in the line and splitting the line on that tokens. Then the smaller parts of the line are processed the same way in recurrence. That special tokens are selected in the order defined in the language configuration file.

### 2.2 Utok

Utok is a tokenizer created as a better version of unitok. It offers faster processing and a better, more universal configuration method.

For now, there is support for English and Czech language. Supporting a new language is pretty straightforward and basic configuration file (3.2) is good starting point.

Under the hood, utok is written in C++ and uses re2 regex library [2]. This library uses finite-state automaton and does not support advanced regex options that need to use functionality like backtracking (e.g., backreferences, look-around). That guarantees linear time complexity for every regex search.

Utok also supports ”glue tag” and special ”split” feature (3.1).

## 3 Utok configuration

Configuration is specified in a separate file. Each non-empty line contains a regex expression. All these regexes are parsed and concatenated into one big regex, that is then compiled using the re2 library. The configuration file supports comments using # symbol at the start of a line.

Each supported language has its own configuration file. Section 3.2 describes basic configuration file. It can be easily extended with language-specific tokens like abbreviations.

### 3.1 Split feature

Utok supports a special ”split” feature. It is annotated with `*SPLIT` at the start of a line in a config file.

Utok goes through the text and tries to match one of the regexes from the config file. After a match is found, it tries to match regexes annotated with `*SPLIT` in order to split the token more and connect the parts with glue tag `<g/>`.

In English configuration it is used to separate apostrophe at the end of a word. For input “don’t”, the output tokens are `do` `<g/>` `n't`

This functionality produces significant slow down (see benchmarks 1). Utok performance is good, but it should be considered if the configuration for language needs to use this split feature.

### 3.2 Basic configuration

The example of what could be considered the default configuration for most languages is in Figure 1.

When creating a new language configuration, this configuration can be extended with language specific rules such as clitics, abbreviations, special word characters and emojis. The following example shows such extension for English which uses the SPLIT feature to separate English clitics as individual tokens.

```
*SPLIT (?i)(.+)('s|'re|'ve|'d|'m|'em|'ll|n't)
*SPLIT (?i)(can)(not)
```

```
# SGML tags
<[/?!]?[a-zA-Z][-.:\w]*\s*/?.*>
<!--.*?-->

# XML entity
&(amp|lt|gt|quot|apos);

# URL
(https?|ftps?|file)://\S*
\bwww\.([-a-zA-Z0-9]+\.)+[a-zA-Z]{2,}(/\S*)?
\b[a-zA-Z]([-a-zA-Z0-9]+\.)+(com|org|net|edu|gov|co\.uk)(/\S*)?

# Email
\w[-'.\w]*@([-a-zA-Z0-9]+\.)+[a-zA-Z]{2,}

# Hashtag
[#@][a-zA-Z][a-zA-Z0-9]+

# Numbers
\d+([-+/.,]\d+)*

# U.S.A.
([A-Z]\.)+\B

# Words
([\pL\pM\pN]|[\pL\pM\pN])+(['\-\p{Pc}][\pL\pM\pN]+)*

# Puctuation = any non-word, non-space
[?!]+
''
\.+|\*+|:+|=+
[^\pL\pM\pN\pZ]
```

Fig. 1: Basic configuration of `utok`

Table 1: Running time of Utok and Unitok on inputs of different size.

| **Program** | **English (17 MB)** | **English (91 MB)** | **Lorem ipsum with html (96 KB)** | **Czech (369 MB)** |
|---|---|---|---|---|
| Utok | 2.0s | 16.7s | 56.9ms | 25.8s |
| Utok (no split) | 1.0s | 7.5s | 49.9ms | 25.8s |
| Unitok | 12.3s | 224.3s | 426.3ms | 621.6s |

## 4 Benchmark

We have done basic evaluation of the speed of both Unitok and Utok on English and Czech texts. The results are summarized in Table 1. We can see the

slowdown of Utok with the SPLIT feature in the configuration file. The Czech configuration does not uset the SPLIT feature.

## 5 Differences in output

During the creation of a utok configuration, the unitok was used as a baseline of good tokenization. However, there are cases in which it is harder to decide what behavior is better. In this section, we will explore a few differences found on English texts when comparing Unitok and the first version of Utok configuration.

The most frequent differences in our test data are the following:

1. Double symbol: for double symbols such as "`// $$ %% --`" unitok keeps them together `//`. On the other hand utok separates symbols with the glue tag `/` `<g/>` `/`.
2. When hash symbols is used like hashtag "#hashtag", both tokenizers keep it as one token. Difference is when the hash symbols is between two words "word#hashtag". Unitok separates the hash symbol with the glue tag on both sides `word` `<g/>` `#` `<g/>` `hashtag`. Utok keeps the hash symbol with the second word `word` `<g/>` `#hashtag`.
3. When talking about years in English we might encounter input "1980's". Unitok keeps it as one token. Utok separates it with the glue tag `1980` `<g/>` `'s`
4. Apostrophe in non-usual places, for example foreign name "Tour de l'Aude". Unitok separates apostrophe from both sides with glue tag `Tour` `de` `l` `<g/>` `'` `<g/>` `Aude`. Utok keeps the word together `Tour` `de` `l'Aude`.
5. Two dots at the end of a sentence in input "Title G. A. S..". Unitok keeps the two dots together `Title` `G.` `A.` `S` `<g/>` `..`. Utok has more natural behavior `Title` `G.` `A.` `S.` `<g/>` `.`.

We can see that the differences are small and in many cases we think that Utok tokenization is better.

## 6 Conclusion

`Utok` is a tokenization tool that is built upon the previous work on `unitok`. The main advantage is speed and ease of configuration. Utok is an order of magnitude faster than Unitok. With even the basic configuration, it works well enough for unsupported languages.

In the future work we will adapt configuration files for all languages in Unitok to respective configuration in Utok. We will also experiment with other regular expression engines or implementation in other programming languges. We also plan to compare the tokenization of Utok with some highly used tokenizers for English ([1,3]).

**Acknowledgements** This work has been partly supported by the Ministry of Education of CR within the LINDAT/CLARIAH-CZ project (LM201810).

## References

1. Altinok, D.: Mastering spaCy: An end-to-end practical guide to implementing NLP applications using the Python ecosystem. Packt Publishing Ltd (2021)
2. Google: Google/re2. `https://github.com/google/re2` (2022)
3. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S.J., McClosky, D.: The Stanford CoreNLP natural language processing toolkit. In: Association for Computational Linguistics (ACL) System Demonstrations. pp. 55–60 (2014), `http://www.aclweb.org/anthology/P/P14/P14-5010`
4. Michelfeit, J., Pomikálek, J., Suchomel, V.: Text tokenisation using unitok. In: Horák, A., Rychlý, P. (eds.) RASLAN 2014. pp. 71–75. Tribun EU, Brno, Czech Republic (2014)