A Roadmap for Universal Syllabic Segmentation

Ondřej Sojka, Petr Sojka, Jakub Máca

Abstract

Space- and time-effective segmentation (word hyphenation) of natural languages remain at the core of every document rendering system, be it TEX, web browser, or mobile operating system. In most languages, segmentation mimicking syllabic pronunciation is a pragmatic preference today.

As language switching is often not marked in rendered texts, the typesetting engine needs *universal syllabic segmentation*. In this article, we show the feasibility of this idea by offering a prototypical solution to two main problems:

A) `Patgen` generation process for several languages at once;

B) no wide character support in tools like `Patgen` or TEX hyphenation, e.g. internal Unicode compliance is missing.

For A), we have applied it to generating universal syllabic patterns from wordlists of nine syllabic, as opposed to etymology-based, languages. For B), we have created a version of `Patgen` that uses the Judy array data structure and compared its effectiveness with the trie implementation.

With the data from nine languages (Czech, Slovak, Georgian, Greek, Polish, Russian, Turkish, Turkmen, and Ukrainian) we showed that

A) developing universal, up-to-date, high-coverage, and highly generalized universal syllabic segmentation patterns is possible, with high impact on virtually all typesetting engines, including web page renderers, and

B) bringing wide character support into the hyphenation part of TEX suite of programs is possible by using the Judy array.

1 Motivation

Justified alignment achieved with quality hyphenation algorithm is both optically pleasing and saves time to read, in addition to saving trees. Only *quality hyphenation* allow interword spaces to be as uniform as possible, close to Gutenberg's ideal of space of fixed width. High coverage, space- and time-effective hyphenation (segmentation) algorithm of all natural languages is badly needed[1] as it remains at the core of every document rendering system, be it TEX, web browser supporting HTML with CSS3, or an operating system providing text rendering for mobile applications.

In most languages, segmentation mimicking syllabic pronunciation is pragmatically preferred today. As language switching is often not marked in rendered texts, and cannot be safely guessed only from words themselves, language-agnostic orthographic syllabification, is needed. We call this task *universal syllabic segmentation*, or in short, syllabification problem.

The syllabification problem has been tackled by several finite state [2] or machine learning techniques recently [11, 1, 22, 14]. Bartlett et al. [1] uses structured support vector machines (SVM) to solve syllabification as tagging problem. Krantz et al. [6] leverage modern neural network techniques with long short-term memory (LSTM) cells, a convolutional component, and a conditional random field (CRF) output layer, and demonstrated cross-linguistic generalizability, syllabifying English, Dutch, Italian, French, Manipuri, and Basque datasets together.

From an orthographic viewpoint (hyphenation), universal language solutions today should reflect the Unicode standard [21]. Internal UCS-2 (two byte) Unicode compliance, a must in today's operating systems and applications, *is missing* in TEX family of programs, e.g. in `Patgen` or TEX itself. The internal processing is thus limited by the internal one-byte representation of language characters and is hardwired into the optimized code of these programs. Therefore processing languages with huge alphabets (Chinese, Japanese, Korean) and sets of languages whose character representations need *wide character* support is close to impossible. Special "hacks" are needed for character and font encodings both on the input side (package `inputenc`) and output side (packages `fontenc` or `fontspec`) are not backed by wide character support internally.

The problem arises both in typesetting texts in these languages that need automated hyphenation for words using wide character representation. Both TEX and `Patgen` have hardwired 8-bit character representation. To develop practically useable universal syllabic hyphenation, one needs to overcome these constraints.

In this paper we **a)** constructively show the feasibility of preparation of universal syllabic patterns, **b)** demonstrate the version of `Patgen` with wide character support, and **c)** discuss further steps to do in TEX program suite to make language hyphenation "Unicode compliant".

The paper is structured as follows. In Section 2 we set the terminology and describe language data we have used in our experiments. Section 3 reminds

[1] `https://bugzilla.mozilla.org/show_bug.cgi?id=672320`

doi.org/10.47397/tb/44-2/main

the reader about the principles of the hyphenation algorithm in TEX and of Patgen-based pattern generation and pattern representation possibilities. Section 4 evaluates the experiments with universal pattern generation. In Section 5 we elaborate on the possible routes towards wide character support in the typesetting engines and Patgen. As usual, we sum up and conclude in the final Section 6.

> "The concept of the syllable is cross-linguistic, though formal definitions are rarely agreed upon, even within a language. In response, data-driven syllabification methods have been developed to learn from syllabified examples. . . . Syllabification can be considered a sequence labeling task where each label delineates the existence or absence of a syllable boundary." [6].

2 Syllabification

Human beings convey the meaning by pronouncing words as sequences of phonemes. Phonology studies the structure of phonemes we are able to pronounce as syllables [10]. Etymologically, a syllable is an Anglo-Norman variation of Old French sillabe, from Latin syllaba, from Greek συλλαβή syllabē "that which is held together; a syllable, several sounds or letters taken together" to make a single sound. [3]

When we delineate boundaries in the orthographic representation of words, we speak about *hyphenation* of words as sequences of *characters*.

2.1 Hyphenation as syllabification

There are subtle differences between syllabification and hyphenation, though. Let us take the Czech word *sestra*. The Czech language authorities [23] allow hyphenations as *se-s-t-ra*, while agreeing that there are only two syllables based on **C**onsonant and **V**owel sequencing: either *se-stra* (CV-CCCV), or *ses-tra* (CVC-CCV), or *sest-ra* (CVCC-CV). Similarly as with hyphenation, defining segments for syllabification is full of exceptions. The Czech sentence *Strč prst skrz krk* or word *scvrnkls* (CCCCCCCC) contain consonants-only syllables.

There are also rare cases where word segmentation should differ in different contexts. It may be necessary within one language (different hyphenation *re-cord* and *rec-ord* depending on part-of-speech), or in different languages. When developing universal syllabic patterns, these theoretically possible segmentations should not be allowed in the input hyphenated wordlist in training. But this should not matter, as e.g. implicit Liang's hyphen.tex patterns do not cover more than 10% of positions [8] and nobody complains about this coverage.

2.2 Data Preparation

To show the feasibility of universal pattern generation, we have collected wordlists of dozen of languages as shown in Table 1. The picked languages that **a)** have a wide diversity in alphabets and syllables and **b)** have existing hyphenation patterns as an approximation for syllable segments. The wordlists were collected from public sources or provided for our research as stratified dictionaries from TenTen corpora [4] by Lexical Computing. In the wordlist sorted by frequency cut at low 5% of word occurrences to eliminate typos appearing in documents. Each tenth word was taken into a wordlist — a stratified sampling technique inspired by Knuth [5] that was already used successfully in pattern generation [20]. Wordlists were hyphenated by legacy patterns, mostly taken from [13].

Alphabet analysis and statistics are shown in Table 2. Total number of characters appearing in all languages exceeds 245, the maximum number of characters that current Patgen could cope with. That is why wide-char representation (of Unicode UCS-2 type) support in Patgen (and then in the hyphenator library in a typesetting engine) would be needed to extend our generation to more languages.

3 Pattern development

The idea of squeezing the hyphenated wordlist into the set of patterns was coined in the dissertation of Frank Liang [8] supervised by Don Knuth. For the automated generation of patterns from a wordlist, he wrote the Patgen program. Patgen was one of the very first approaches that harnessed the power of data with supervised machine learning. Programmed originally to support English and ASCII, it was later extended to be used for 8 bit characters and for wordlists that contain at most 245 characters [9]. It is capable of efficient lossy or lossless *compression* of hyphenated dictionary with several orders of magnitude compression ratio. Generated patterns have minimal length, e.g., the shortest context possible, which results in their *generalization* properties.

Generally, *exact lossless* pattern *minimization is non-polynomial* by reduction to the minimum set cover problem [16]. For Czech, *exact lossless* pattern generation *is feasible* [17], while reaching *100% coverage* and simultaneously *no errors*. Strict pattern minimality (size) is not an issue nowadays.

Table 1: Language resources and patterns used in pattern development experiments. All data have been converted to UTF-8 encoding and contain lowercase alphabetic characters only. Alphabet size (# chars) counts characters appearing in the language wordlist collected. Languages were chosen by the diversity in the size of patterns, syllables

Language	# words	# chars	# patterns	# syllables	pattern source, alphabet
Czech+Slovak (cz+sk)	606,499	47	8,231	2,288,413	[19] correct optimized parameters, Latin
Georgian (ka)	50,644	33	2,110	224,799	[13] tex-hyphen repo, Georgian
Greek (el-monoton)	10,432	48	1,208	37,736	[13] tex-hyphen repo, Greek
Panjabi (pa)	892	52	60	2,579	[13] tex-hyphen repo, Gurmukhi
Polish (pl)	20,490	34	4,053	65,510	[13] tex-hyphen repo, Latin
Russian (ru)	19,698	33	4,808	75,532	[13] tex-hyphen repo
Tamil (ta)	46,526	48	71	209,380	[13] tex-hyphen repo, Tamil
Telugu (te)	28,849	66	72	125,508	[13] tex-hyphen repo, Telugu
Thai (th)	757	64	4,342	1,185	[13] tex-hyphen repo, Thai
Turkish (tr)	24,634	32	597	103,989	[13] tex-hyphen repo, Latin
Turkmen (tk)	9,262	30	2,371	33,080	[13] tex-hyphen repo, Latin
Ukrainian (ua)	17,007	33	1,990	65,099	[13] tex-hyphen repo, Cyrillic

Table 2: Language alphabet overlaps. Cells contain the number of lowercase characters that overlap between languages. In total, 13 languages contain in total 412 different lowercase characters, more than `Patgen` is capable of digesting.

Language	cz+sk	ka	el	pa	pl	ru	ta	te	th	tr	tk	ua
Czech+Slovak (cz+sk)	47	0	0	0	26	0	0	0	0	25	28	0
Georgian (ka)	0	33	0	0	0	0	0	0	0	0	0	0
Greek (el-monoton)	0	0	48	0	0	0	0	0	0	0	0	0
Panjabi (pa)	0	0	0	52	0	0	0	0	0	0	0	0
Polish (pl)	26	0	0	0	34	0	0	0	0	23	22	0
Russian (ru)	0	0	0	0	0	33	0	0	0	0	0	29
Tamil (ta)	0	0	0	0	0	0	48	0	0	0	0	0
Telugu (te)	0	0	0	0	0	0	0	66	0	0	0	0
Thai (th)	0	0	0	0	0	0	0	0	64	0	0	0
Turkish (tr)	25	0	0	0	23	0	0	0	0	32	25	0
Turkmen (tk)	28	0	0	0	22	0	0	0	0	25	30	0
Ukrainian (ua)	0	0	0	0	0	29	0	0	0	0	0	33

The idea and its realization is a programming pearl. Motivated by space and time constraints, instead of the classical solution of dictionary problem in the logarithmic time of dictionary size, the word hyphenation is computed from patterns in constant time, where the constant is given by *word* length. It is capable of efficient lossy or lossless *compression* of hyphenated dictionary with a compression ratio of several orders of magnitude. Generated patterns have minimal length, e.g., the shortest context possible, which results in their *generalization* properties.

Generally, *exact lossless* pattern *minimization is non-polynomial* by reduction to the minimum set cover problem [16]. For Czech, *exact lossless* pattern generation *is feasible* [17], while reaching *100% coverage* and simultaneously *no errors.* Strict pattern minimality (size) is not an issue nowadays.

Space needed for patterns in the *packed trie* data structure is typically in tens of kB, which is several orders of magnitude smaller than the wordlist size. With fine-tuned parameters of pattern generation in the so-called *levels*, one could prepare patterns with zero errors and almost full coverage of hyphenation points from the input dictionary.

The patterns are collected in the repository maintained by the TEX community [13]. It is no surprise that most, if not all leading typesetting engines deploy this "competing pattern engineering technology" [15].

3.1 Patterns

The patterns "compete" with each other whether to split the word at a position, given varying characters in both side contexts, see Figure 1.

We have shown how effective and powerful the technique is, and that its power depends on the *parameters* of pattern generation. [17] The key is the proper setting of `Patgen` parameters for pattern generation. The universality idea of segmentation with `Patgen` has been coined already in [18]. Then we actually demonstrated the techniques for the development of two languages together, Czech and Slovak, and developed a joint wordlist and patterns [19].

We thought of extending the technique to other Slavic and syllabic languages. The bottleneck for adding new languages was `Patgen` and TEX´s constraint of one-byte character support only for storing patterns in tries. We thought of using a modern data structure that would allow wide char trie representation. That was the task for the bachelor thesis Judy: [12].

3.2 Judy array

Judy array, also known as simple Judy, is a data structure that implements a sparse dynamic array, allowing for versatile applications such as a dynamically sized array or an associative array. Judy is internally implemented as a tree structure, where every internal node has 256 ancestor nodes. The most interesting thing about this structure is that it tries to be as memory efficient as possible by effectively using cache and avoiding unnecessary access to main memory. As a result, Judy array is fast and memory efficient.

The feasibility of utilizing the Judy structure for storing hyphenation patterns is demonstrated in the thesis [12]. In Chapter 4, it is shown that Judy has the potential to be faster and more memory-efficient compared to Trie when working with patterns. Furthermore, Chapter 5 explores the potential integration of Judy into `Patgen` and the consequent impact on `Patgen`'s generation process. The results from this chapter indicate that rewriting `Patgen` with Judy is possible but would require an almost complete overhaul of `Patgen` s code and algorithms. This redevelopment would yield a `Patgen` version capable of handling input of any kind, enabling the generation of patterns composed of arbitrary alphabets. However, it is important to note that the generation process would be approximately four times slower than the current implementation. This is due to the hiding of access to the inner nodes of stored tries in Judy. As this access is not needed in TEX for the hyphenation of individual words, using Judy in some variant of TEX successor would make the hyphenation faster.

3.3 Universal pattern generation

To pursue the idea of universal syllabic pattern generation, we have checked whether the legacy patterns hyphenate the same word valid in different languages differently. The result with a short discussion is in Table 3 on the facing page. The expectation that syllable forming principles are universal as phonology theory confirms, and the errors we have found were due to the difference between hyphenation and syllabification caused by inconsistent markup rather than a principled difference in the word morphology, e.g. compound word segmentation in one language and a single word in the other.[2]

[2] Compound word could evolve in perception as a single word even in language itself. As examples may serve evolution of *e-mail* into *email* or *roz-um* into syllabic *ro-zum* in Czech.

```
    h y p h e n a t i o n
p1           1n a
p1               1t i o n
p2            n2a t
p2                 2i o
p2          h e2n
p3  h y3p h
p4          h e n a4
p5          h e n5a t
    h0y3p0h0e2n5a4t2i0o0n
```

hy-phen-ation → 2 6
. . . → . . .
. . . → . . .
key → data

Solution to the dictionary problem:
For key part (the word) to store
the data part (its division)

Figure 1: 8 patterns "compete" how to hyphenate *hyphenation*. Winners are patterns `hy3ph` and `hen5at` generated at the highest covering level (odd numbers) generation. Level hierarchy allows for storing exceptions, exceptions to exceptions, exceptions to exceptions to exceptions,. . . , with character contexts as parameters. [8]

Table 3: Different word hyphenations overlaps. Cells contain the number of same words that are segmented differently between languages. Differences are caused typically by suboptimal coverage patterns used to hyphenate wordlist (*vi-bram* vs. *vib-ram*, *up-gra-de* vs. *upg-ra-de*). We remove the differently hyphenated words when joining wordlists for the final syllabic generation.

Language	cz+sk	ka	el	pa	pl	ru	ta	te	th	tr	tk	ua
Czech+Slovak (cz+sk)	9	0	0	0	388	0	0	0	0	640	69	0
Georgian (ka)	0	0	0	0	0	0	0	0	0	0	0	0
Greek (el-monoton)	0	0	0	0	0	0	0	0	0	0	0	0
Panjabi (pa)	0	0	0	0	0	0	0	0	0	0	0	0
Polish (pl)	388	0	0	0	0	0	0	0	0	187	9	0
Russian (ru)	0	0	0	0	0	0	0	0	0	0	0	125
Tamil (ta)	0	0	0	0	0	0	0	0	0	0	0	0
Telugu (te)	0	0	0	0	0	0	0	0	0	0	0	0
Thai (th)	0	0	0	0	0	0	0	0	0	0	0	0
Turkish (tr)	640	0	0	0	187	0	0	0	0	0	80	0
Turkmen (tk)	69	0	0	0	9	0	0	0	0	80	0	0
Ukrainian (ua)	0	0	0	0	0	125	0	0	0	0	0	0

We removed all colliding words when joining wordlists into the wordlist universal pattern generation — we collected words for nine languages (cz, sk, ka, el, pl, ru, tr, tk a ua).

We generated universal patterns with the same three sets of `Patgen` parameters (custom, correct optimized, and size optimized) as when generating Czechoslovak patterns. The results are at Tables 4 (custom), 5 (correct optimized) and 6 (size optimized). The results are comparable with generation for two languages and confirm the feasibility of universal pattern development.

We did not pursue 100% coverage at all costs because the source data is noisy, and we do not want the patterns to learn all the typos and inconsistencies. Also, the size of the new languages was rather small, compared to the Czechoslovak one.

4 Evaluation

We evaluated the quality of developed patterns by two metrics. *Coverage* of hyphenation points in the training word list tells how the patterns correctly predicted hyphenation points used in training. *Generalization* means how the patterns behave on unseen data, on words not available in the data used during `Patgen` training. The methodology is the same as we used in the development of Czechoslovak patterns [19].

In Table 8, we compare the efficiency of different approaches to hyphenating 2 languages and 9 languages from one pattern set. We see that the performance of universal patterns is comparable in size and quality to double- or single-language ones — there is only a negligible difference. The table shows that generalization qualities, given the small input size wordlists, are very good, and comparable to the

Table 4: Statistics from the generation of universal patterns for cz+sk, ka, el, pl, ru, tr, tk a ua with *custom* parameters and `\lefthyphenmin=2`, `\righthyphenmin=2`. Generation took 33.23 seconds, 11,238 patterns, 77 kB.

Level	Patterns	Good	Bad	Missed	Lengths	Params
1	2,407	2,066,410	280,020	70,588	1 3	1 3 12
2	2,375	2,025,245	8,866	111,753	2 4	1 1 5
3	4,626	2,118,063	19,213	18,935	3 6	1 2 4
4	2,993	2,117,739	5,920	19,259	3 7	1 4 2

Table 5: Statistics from the generation of universal patterns for cz+sk, ka, el, pl, ru, tr, tk a ua with *correct optimized* parameters and `\lefthyphenmin=2`, `\righthyphenmin=2`. Generation took 35.43 seconds, 29,742 patterns, 219 kB.

Level	Patterns	Good	Bad	Missed	Lengths	Params
1	7,188	2,049,375	164,224	87,623	1 3	1 5 1
2	4,108	2,042,249	14,094	94,749	1 3	1 5 1
3	15,010	2,134,692	20,544	2,306	2 6	1 3 1
4	6,920	2,133,458	815	3,540	2 7	1 3 1

Table 6: Statistics from the generation of universal patterns for cz+sk, ka, el, pl, ru, tr, tk a ua with *size optimized* parameters and `\lefthyphenmin=2`, `\righthyphenmin=2`. Generation took 29.75 seconds, 14,321 patterns, 101 kB.

Level	Patterns	Good	Bad	Missed	Lengths	Params
1	1,201	2,092,928	598,321	44,070	1 3	1 2 20
2	2,695	1,736,372	5,274	400,626	2 4	2 1 8
3	4,835	2,102,803	20,094	34,195	3 5	1 4 7
4	6,508	2,099,607	210	37,391	4 7	3 2 1

fine-tuned Czechoslovak ones. Investing in the purification and consistency of input wordlists (as we did for Czech and Slovak) would result in near to perfect syllabic patterns with almost 100% coverage and no errors.

5 Future work

A natural further step is to merge further languages, where the syllabic principle is used for hyphenation. For that, one would need a version of `Patgen` we provisionally call `UniPatgen`. This version would support Unicode not only in I/O but also internally as a wide char (UCS-2) character encoded in the pattern representation in packed trie or in Judy array. This would allow merging more languages *without* increasing the computational complexity of hyphenation, and only sublinear increase of pattern size. We believe that coverage may differ from 100% only by words that should be hyphenated differently in different languages — our estimate is in small, single-digit percents, while currently widely used `hyphen.tex` patterns do not cover 10+%!

Another possible demanded extension in pattern development is the support of a specific hyphenation penalty for compound word borders. This extension, discussed already 30 years ago [20], would generate patterns first for compound words, and after fixing them continue with pattern generation for all other hyphenation points. ε-TeX would then set the hyphenation penalties depending on level ranges in patterns found for the hyphenated word. This extension is orthogonal with support for universal patterns but might need to increase the maximal number of levels allowed in patterns to two digits.

There are several questions for the TeX developers' community:

1. Should the universal syllabic patterns ever be developed?

2. If so, should the needed wide char representations be added in the TeX suite of programs

Table 7: Comparison of the efficiency of different approaches to pattern generation of Czechoslovak and of universal patterns. Note that the size of universal patterns grows sublinearly with the number of languages. The generalization ability of universal patterns is only slightly worse than that of Czechoslovak ones. The experience from the development of Czechoslovak patterns shows that performance could be improved by consistent markup of wordlist data.

Word list	Parameters	Good	Bad	Missed	Size	Patterns
Czechoslovak	custom	99.87%	0.03%	0.13%	32 kB	5,907
Czechoslovak	correctopt	99.99%	0.00%	0.01%	45 kB	8,231
Czechoslovak	sizeopt	99.67%	0.00%	0.33%	40 kB	7,417
Universal	custom	99.10%	0.28%	0.90%	77 kB	11,238
Universal	correctopt	99.83%	0.04%	0.17%	219 kB	29,742
Universal	sizeopt	98.25%	0.01%	1.75%	101 kB	14,321

Table 8: Results of 10-fold cross-validation (learning on 90%, and testing on remaining 10%). Generalization properties (performance on words not seen during training) are compared with Czechoslovak patterns. By adding 7 languages, the generalization abilities of universal patterns are only slightly worse.

Wordlist	Parameters	Good	Bad	Missed
Czechoslovak	custom	99.64%	0.22%	0.14%
Czechoslovak	correctopt	99.81%	0.15%	0.04%
Czechoslovak	sizeopt	99.41%	0.18%	0.40%
Universal	custom	97.99%	1.06%	0.95%
Universal	correctopt	98.10%	1.28%	0.62%
Universal	sizeopt	97.50%	0.94%	1.56%

(TEX-based engines not yet supporting it[3] and `Patgen` or `UniPatgen`) *internally*?

3. If not, should it be handled by external segmenters on TEXs input, based on `Patgen`'s successor as `UniPatgen`?
4. If `UniPatgen` would be developed, should it be added to the distribution, together with Unicode patterns included and supported in repositories like [13]?
5. Should `UniPatgen` (and LuaTEX) add dependence on a Judy library, or should a more conservative solution be sought and implemented? With a conservative solution, which data structure to use for storing patterns? Should the memory be allocated dynamically, to overcome abundant explosion of format size that stores the patterns during iniTEX phase?
6. Should `UniPatgen` (and ε-TEX) additionally and orthogonally support patterns and different hyphenation penalty for compound word borders, currently available in e.g. German wordlist [7]?

We would appreciate qualified opinions on these decisions being sent to authors.

“All we are saying, give patterns a chance.”
Our paraphrase of John Lennon’s protest song refrain

6 Conclusion

Preparation of language-agnostic, e.g. universal syllabic segmentation patterns could be done! We have demonstrated this possibility by generating it based on the wordlists of nine languages with current `Patgen`. They can have superb generalization qualities, high coverage of hyphenation points (more than most legacy patterns), and virtually no errors. Their use can have a high impact on virtually all typesetting engines including web page renderers.

Supporting wide characters in `Patgen` is a showstopper for adding more languages. We have shown that bringing wide character support into the hyphenation part of TEX suite of programs is possible by using Judy array. It will allow to generate and deploy patterns for the whole Unicode character sets. We have discussed the possible roadmap to make

[3] `https://cs.overleaf.com/learn/latex/TeX_primitives_listed_by_TeX_engine`

this a reality in typesetting engines including TEX successors.

Acknowledgments

We are indebted to Don Knuth for questioning the common properties of Czech and Slovak hyphenation during our presentation of [17] at TUG 2019, which has led us in this research direction. We also thank everyone whose shoulders we build our work on, e.g. for wordlists by Lexical Computing, and to all who commented on our work at TUG 2021 [19] and TUG 2023.

References

[1] S. Bartlett, G. Kondrak, C. Cherry. Automatic Syllabification with Structured SVMs for Letter-to-Phoneme Conversion. In *Proceedings of ACL-08: HLT*, pp. 568–576, Columbus, Ohio, June 2008. ACL. `https://www.aclweb.org/anthology/P08-1065`

[2] Y. Haralambous. New hyphenation techniques in Ω_2. *TUGboat* 27(1):98–103, 2006. `http://www.ftp.tug.org/TUGboat/Articles/tb27-1/tb86haralambous-hyph.pdf`

[3] Online etymology dictionary. `"syllable"`. `https://www.etymonline.com/word/syllable`

[4] M. Jakubíček, A. Kilgarriff, et al. The TenTen Corpus Family. In *Proc. of the 7th International Corpus Linguistics Conference (CL)*, pp. 125–127, Lancaster, July 2013.

[5] D.E. Knuth. *3 : 16 Bible texts illuminated.* A-R Editions, Inc., 1991.

[6] J. Krantz, M. Dulin, P.D. Palma. Language-agnostic syllabification with neural sequence labeling. *CoRR* abs/1909.13362, 2019. `http://arxiv.org/abs/1909.13362`

[7] W. Lemberg. A database of German words with hyphenation information, 2023. `https://repo.or.cz/wortliste.git`

[8] F.M. Liang. *Word Hy-phen-a-tion by Com-put-er.* Ph.D. thesis, Department of Computer Science, Stanford University, Aug. 1983. `https://www.tug.org/docs/liang/liang-thesis.pdf`

[9] F.M. Liang, P. Breitenlohner. `PAT`tern `GEN`eration program for the TEX82 hyphenator. Electronic documentation of `PATGEN` program version 2.3 from web2c distribution on CTAN, 1999.

[10] I. Maddieson. Syllable Structure. In *The World Atlas of Language Structures Online*, M.S. Dryer, M. Haspelmath, eds. Max Planck Institute for Evolutionary Anthropology, Leipzig, 2013. `https://wals.info/chapter/12`

[11] Y. Marchand, C.R. Adsett, R.I. Damper. Automatic Syllabification in English: A Comparison of Different Algorithms. *Language and Speech* 52(1):1–27, 2009. `10.1177/0023830908099881`

[12] J. Máca. Judy, May 2023. Bachelor Thesis supervised by Petr Sojka and defended at Masaryk University, Faculty of Informatics. `https://is.muni.cz/th/kru3j`

[13] A. Rosendahl, M. Miklavec. TEX hyphenation patterns, 2023. Accessed 2023-07-05. `http://hyphenation.org/tex`

[14] Y. Shao, C. Hardmeier, J. Nivre. Universal Word Segmentation: Implementation and Interpretation. *Transactions of the Association for Computational Linguistics* 6:421–435, 2018. `10.1162/tacl_a_00033`

[15] P. Sojka. Competing Patterns for Language Engineering. In *Proceedings of the Third International Workshop on Text, Speech and Dialogue—TSD 2000*, P. Sojka, I. Kopeček, K. Pala, eds., LNAI 1902, pp. 157–162, Brno, Czech Republic, Sept. 2000. Springer-Verlag. `10.1007/3-540-45323-7_27`

[16] P. Sojka. *Competing Patterns in Language Engineering and Computer Typesetting.* Ph.D. thesis, Masaryk University, Brno, Jan. 2005. `https://www.researchgate.net/publication/265246931_Competing_Patterns_in_Language_Engineering_and_Computer_Typesetting/`

[17] P. Sojka, O. Sojka. The Unreasonable Effectiveness of Pattern Generation. *TUGboat* 40(2):187–193, 2019. `https://tug.org/TUGboat/tb40-2/tb125sojka-patgen.pdf`

[18] P. Sojka, O. Sojka. Towards Universal Hyphenation Patterns. In *Proceedings of Recent Advances in Slavonic Natural Language Processing—RASLAN 2019*, A. Horák, P. Rychlý, A. Rambousek, eds., pp. 63–68, Karlova Studánka, Czech Republic, 2019. Tribun EU. `https://is.muni.cz/publication/1585259/?lang=en`. `https://nlp.fi.muni.cz/raslan/2019/paper13-sojka.pdf`

[19] P. Sojka, O. Sojka. New Czechoslovak Hyphenation Patterns, Word Lists, and Workflow. *TUGboat* 42(2), 2021. `https://doi.org/10.47397/tb/42-2/tb131sojka-czech`

[20] P. Sojka, P. Ševeček. Hyphenation in TEX — Quo Vadis? *TUGboat* 16(3):280–289, 1995. `https://tug.org/TUGboat/tb16-3/tb48soj1.pdf`

[21] The Unicode Consortium. *The Unicode Standard: Worldwide Character Encoding. Version 15.0.* Unicode, Inc., Mountain View, CA, USA, 2022. `https://www.unicode.org/versions/Unicode15.0.0`

[22] N. Trogkanis, C. Elkan. Conditional Random Fields for Word Hyphenation. In *Proceedings of the 48th Annual Meeting of the ACL*, pp. 366–374, Uppsala, Sweden, July 2010. ACL. `https://www.aclweb.org/anthology/P10-1038`

[23] Internetová jazyková příručka (Internet Language Reference Book), 2023. `https://prirucka.ujc.cas.cz/?id=135`

◇ Ondřej Sojka
Faculty of Informatics, Masaryk Univ.,
Brno, Czech Republic
`454904 (at) mail dot muni dot cz`
ORCID 0000-0003-2048-9977

◇ Petr Sojka
Faculty of Informatics, Masaryk Univ.,
Brno, Czech Republic
`sojka (at) fi dot muni dot cz`
`https://www.fi.muni.cz/usr/sojka/`
ORCID 0000-0002-5768-4007

◇ Jakub Máca
Faculty of Informatics, Masaryk Univ.,
Brno, Czech Republic
`514024 (at) mail dot muni dot cz`
ORCID 0009-0008-1583-3183