# A Revisited Small Tutorial on Patgen, 28 Years After

Yannis Haralambous
IMT Atlantique, CS 83818, 29238 Brest Cedex 3, France
yannis.haralambous@imt-atlantique.fr

March 29, 2021

*This the revisited version of a document written in 1993, long before Unicode, the PDF format, the Web, the n-th Doctors (for $n \geq 9$), XeTeX, smartphones, emojis, fake news, COVID, and whatever makes the world we currently live in. As a tutorial, it was utterly* bad*: it was unclear, missed important information and focalized on (semi-private) jokes. I realized this fact in March 2021, when I tried to use Patgen again after decades of oblivion. Having re-discovered the use of Patgen, which is still as scarcely documented as it was in 1993, I decided to rewrite this tutorial hoping to succeed where I once failed. I ask in advance for your indulgence. Do not hesitate to contact me if parts of it are (still) obscure!*

This tutorial is based on PATGEN v2.4 from the TeXlive distribution 2020.

## 1 Introduction

Patgen is a program allowing to build hyphenation patterns (cf. [2, Appendix H] for more details on these) out of (relatively large) corpora of already-hyphenated words. It has been developed in the early eighties by Frank Liang, a PhD student of Donald Knuth. Patgen's algorithm is thoroughly described in Liang's thesis [4]. In the meantime the problem of "syllabification" has been tackled by many machine learning approaches, such as SVMs [1] bidirectional LSTM networks [3], etc. In some languages hyphenation is strictly syllabic and hence syllabification is equivalent to hyphenation, in other languages this is not the case, but the same tools can be applied since the two problems are quite similar. Also there is abundant bibliography on hyphenation in the *TUGboat*.

## 2 The Various Files Involved

Patgen reads information from three files (the dictionary file, the pre-existing patterns file and the translate file) and writes information into two files: the output file, where patterns are written and an additional output file called `pattmp.*` containing the result of the application of patterns to the dictionary file.

### 2.1 The Translate File

This file comes last on the command line syntax, but because of its importance we will cover it first. The original Patgen (1983) was hardwired to use only the 26 letters of the Latin alphabet. Its November 1991 revision (version 2) by Peter Breitenlohner extended it so that up to 245 different "abstract characters" can be used. By "abstract characters" I mean members of the alphabet of a formal language that can be represented by multiple, possibly multi-character, representations.

Each member of this formal alphabet must have a primary representation. Optionally it may have alternative representations which will be detected in the dictionary file and mapped to the primary one. In the test file, only primary representations are used.

The fact of using multi-character (and hence multi-byte) representations was a brillant idea by Peter Breitenlohner since it makes Patgen encoding-agnostic: you can use ASCII or ISO Latin-16 or Unicode it will be the same for Patgen.

Let us move on to the syntax of the translate file, which is quite peculiar.
The first line of the file has fixed position-dependent semantics:

- positions 1 and 2 are used to indicate the `\lefthyphenmin` value,
- positions 3 and 4 are used for the `\righthyphenmin` value. (As there seems to be no language using left or right hyphen min values greater than 9, you will have to systematically leave a space in front of the one-digit values (spaces at positions 1 and 3).)
- positions 5, 6 and 7 of the first line are used to give alternative values for characters `.`, `-` and `*`: the first one is used for word limits, the second is used to indicate breaking points in the patterns file and in the dictionary, the third is used to indicate correct hyphenations in the test file;
- starting with line 2, you add one line per symbol of the formal alphabet you will use in your dictionary and patterns. The first character position of each line is used for the *delimiter* of that line. This delimiter will surround the letter of your formal alphabet as well as all alternative representations of it. Writing the delimiter twice ends the parsing process for that line, so that one can add comments after the double delimiter.

For example, in the line

```
XeXEXX
```

we use `X` as delimiter, our primary representative of this formal letter is `e` (the primary representative is used in the patterns that will be created), and a secondary representative of it is `E`. Secondary representatives can be multi-character sequences.

Naturally we will use the space as delimiter, since it makes the translate file more readable:

```
 e E
```

But do not omit the space at position 1, otherwise your file will not be correctly parsed.

One last detail: since lines $\geq 2$ use a delimiter mechanisme, you can use multi-character and multibyte sequences. Not so for line 1 which is byte-oriented: better choose ASCII characters for the potential replacements of `.`, `-` and `*`.

## 2.2 The Pre-Existing Patterns File

If you are not creating patterns ab ovo but rather wish to update a pre-existing set of patterns, then you can place these patterns (in standard pattern notation) in the pre-existing patterns file (second in the list of patgen command line arguments).

If you don't have pre-existing patterns, you should create ("`touch`" on Unix systems) an empty pre-existing patterns file nevertheless.

## 2.3 The Dictionary File

The dictionary file contains words using the primary or secondary representatives of formal letters, in which breaking points are indicated by hyphens `-`.

The dictionary file is also used to test the patterns at the end of the pattern creation process (see § 2.5).

## 2.4 The Output File

This one is written by Patgen, it contains patterns in the standard TEX notation.

### 2.5 The Test Result Files

Test Result files carry names `pattmp.*`, where `*` is a number. They contain the test results. Patgen will continue creating such files, giving them increasing numbers so that you can use to them to evaluate your progress.

Their syntax is as follows: the period `.` (or its alternative given in position 5 of the first line of the translate file) represents wrong breaking points introduced by the patterns, the hyphen `-` (or its alternative given in position 6 of the first line of the translate file) represents missed breaking points, and the asterisk `*` (or its alternative given in position 7 of the first line of the translate file) represents correct breaking points detected by the patterns. A perfect result is one with only asterisks.

## 3 Command Line Syntax

Patgen expects four command line arguments:

```
patgen greek.dic greek.pat greek.out greek.tra
```

namely the names (or paths) of the dictionary file, the pre-existing patterns file, the output file and the translate file. Failure of respecting this order will result in Patgen's nervous breakdown, hardly something you would like to inflict to such an old and faithful friend.

## 4 Strategy

As Frank Liang explains in his thesis [4, p. 37], at each level patterns are tested over all words, they are kept only if they satisfy the following formula:

$$\alpha * \#\text{good matches} - \beta * \#\text{bad matches} \geq \eta,$$

where $\alpha$ is the "good weight," $\beta$ the "bad weight" and $\eta$ the threshold.

Once Patgen launched and the parsing of files completed, it will ask you

```
This is PATGEN, Version 2.4 (TeX Live 2020)
left_hyphen_min = 1, right_hyphen_min = 1, 33 letters
0 patterns read in
pattern trie has 256 nodes, trie_max = 256, 0 outputs
hyph_start, hyph_finish:
```

You have to give two numbers between 1 and 9 (separated by a space), representing the pattern levels you want to create (odd pattern levels are hyphenating levels, even pattern levels are inhibiting levels, higher numbers prevail over lower numbers).

It will then ask:

```
pat_start, pat_finish:
```

These are the minimum and maximum lengths of patterns you are interesting in, between 1 and 15. With modern computers it is by no means extravagant to ask for the whole range, both your CPU and your RAM can handle it easily. Again you have to supply two numbers (separated by a space), the min and the max.

Then Patgen will ask you:

```
good weight, bad weight, threshold:
```

These are the three values in our formula above ($\alpha$, $\beta$, $\eta$). In his thesis, Liang describes the strategy used for the TEX82 version of US English patterns:

| level | $\alpha$ | $\beta$ | $\eta$ | max length | nb of patterns | accuracy |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 20 | 4 | 458 | 76.6% |
| 2 | 2 | 1 | 8 | 4 | 509 | 68.2% |
| 3 | 1 | 4 | 7 | 5 | 985 | 83.2% |
| 4 | 3 | 2 | 1 | 6 | 1647 | 82.0% |
| 5 | 1 | $\infty$ | 4 | 8 | 1320 | 89.3% |

I have used Patgen by consistently giving the same values `1 1 1` for $\alpha, \beta, \eta$, and also got good results. Testing the parameters may need several attempts, but today these attempts are measured in seconds and not in hours anymore.

Once you have given the parameter values, Patgen will starting calculating all pattern lengths for level 1, and will report:

```
593480 good, 210386 bad, 0 missed
100.00 %, 35.45 %, 0.00 %
97 patterns, 712 nodes in count trie, triec_max = 734
0 good and 97 bad patterns added
finding 0 good and 0 bad hyphens
pattern trie has 47469 nodes, trie_max = 82932, 74 outputs
42121 nodes and 53 outputs deleted
total of 2510 patterns at hyph_level 1

pat_start, pat_finish:
```

It's up to you to give values of pattern length for the next run. Once these given, you will be asked again

```
good weight, bad weight, threshold:
```

and so on, for each level.

When there are no more levels left, you get the following message:

```
593480 good, 0 bad, 0 missed
100.00 %, 0.00 %, 0.00 %
0 patterns, 256 nodes in count trie, triec_max = 256
0 good and 34 bad patterns added
finding 0 good and 0 bad hyphens
pattern trie has 11603 nodes, trie_max = 146189, 496 outputs
0 nodes and 10 outputs deleted
total of 0 patterns at hyph_level 9
hyphenate word list?
```

where you reply 'y' to get the following answer:

```
writing pattmp.9

593480 good, 0 bad, 0 missed
100.00 %, 0.00 %, 0.00 %
```

and the program stops. Now the output file contains the patterns and file `pattmp.9` (the name is given in the message above, and may vary) contains the result of applying the patterns to the dictionary file.

# 5 An example

*This toy example originates from the 1993 version of the tutorial, I found it so pittoresque that I couldn't help but leave it. Nevertheless I ran Patgen anew to get actualized console output (in the*

*previous document I used Wilfried Ricken's memorable TEX implementation for MacOS 9, from 1993).*

To illustrate this I have taken some Greek words (people who know *Χάρρυ Κλύννν* and his album *Πατάτες* will recognize some of these words…) and included an *α* with accent, and a variant representation of *π*, in form of a macro `\varpi␣`. Also I followed the Greek alphabetical order in the translate file.

Here is my list of words:

```
A-NU-PO-TA-QTOS
A-KA-TA-M'A-QH-TOS
A-EI-MNH-STOS
MA-NA
TOUR-KO-GU-FTIS-SA
NU-QO-KO-PTHS
LI-ME-NO-FU-LA-KAS
MPRE-LOK
A-GA-\varpi A-EI
PEI-RAI-'AS
```

(file `greek.dic`); and here is my translate file (`greek.tra`):

```
 1 1
 a A
 'a 'A
 b B
 g G
 d D
 e E
 z Z
 h H
 j J
 i I
 k K
 l L
 m M
 n N
 o O
#p#P#\varpi ##
 r R
 s S
 t T
 u U
 f F
 q Q
 y Y
 w W
```

(in case you are wondering, Greek indeed uses values 1 and 1 for minimal right and left hyphenations!). As you see, I have chosen `#` as delimiter in the case of *π*, because `\varpi` is supposed to be followed by a blank space. And then I wrote two delimiters (`##`) to let Patgen know that I finished giving secondary representations of that formal letter.

Here is what I got on my console:

```
patgen greek.dic greek.pat greek.out greek.tra

This is PATGEN, Version 2.4 (TeX Live 2020)
```

```
left_hyphen_min = 1, right_hyphen_min = 1, 24 letters
0 patterns read in
pattern trie has 266 nodes, trie_max = 290, 0 outputs
hyph_start, hyph_finish: 1 2
pat_start, pat_finish: 2 4
good weight, bad weight, threshold: 1 1 1
processing dictionary with pat_len = 2, pat_dot = 1

0 good, 0 bad, 31 missed
0.00 %, 0.00 %, 100.00 %
69 patterns, 325 nodes in count trie, triec_max = 440
25 good and 42 bad patterns added (more to come)
finding 29 good and 0 bad hyphens, efficiency = 1.16
pattern trie has 333 nodes, trie_max = 349, 2 outputs
processing dictionary with pat_len = 2, pat_dot = 0

29 good, 0 bad, 2 missed
93.55 %, 0.00 %, 6.45 %
45 patterns, 301 nodes in count trie, triec_max = 378
2 good and 43 bad patterns added
finding 2 good and 0 bad hyphens, efficiency = 1.00
pattern trie has 339 nodes, trie_max = 386, 6 outputs
processing dictionary with pat_len = 2, pat_dot = 2

31 good, 0 bad, 0 missed
100.00 %, 0.00 %, 0.00 %
47 patterns, 303 nodes in count trie, triec_max = 369
0 good and 47 bad patterns added
finding 0 good and 0 bad hyphens
pattern trie has 344 nodes, trie_max = 386, 13 outputs
51 nodes and 11 outputs deleted
total of 27 patterns at hyph_level 1

pat_start, pat_finish: 2 4
good weight, bad weight, threshold: 1 1 1
processing dictionary with pat_len = 2, pat_dot = 1

31 good, 0 bad, 0 missed
100.00 %, 0.00 %, 0.00 %
27 patterns, 283 nodes in count trie, triec_max = 315
0 good and 27 bad patterns added
finding 0 good and 0 bad hyphens
pattern trie has 295 nodes, trie_max = 386, 4 outputs
processing dictionary with pat_len = 2, pat_dot = 0

31 good, 0 bad, 0 missed
100.00 %, 0.00 %, 0.00 %
27 patterns, 283 nodes in count trie, triec_max = 303
0 good and 27 bad patterns added
finding 0 good and 0 bad hyphens
pattern trie has 320 nodes, trie_max = 386, 6 outputs
processing dictionary with pat_len = 2, pat_dot = 2

31 good, 0 bad, 0 missed
```

```
100.00 %, 0.00 %, 0.00 %
24 patterns, 280 nodes in count trie, triec_max = 286
0 good and 24 bad patterns added
finding 0 good and 0 bad hyphens
pattern trie has 328 nodes, trie_max = 386, 9 outputs
35 nodes and 7 outputs deleted
total of 0 patterns at hyph_level 2
hyphenate word list? y
writing pattmp.2

31 good, 0 bad, 0 missed
100.00 %, 0.00 %, 0.00 %
```

and here are the results: first of all the output file `greek.out`:

```
a1g
a1e
a1k
a1m
a1n
a1p
a1t
a1q
'a1q
e1l
e1n
h1t
i1'a
i1m
i1r
1ko
o1g
o1p
o1t
o1f
r1k
s1s
1st
u1l
u1p
u1f
u1q
```

As you see, `o1t` is sorted before `o1f`, because of the order used for the primary representatives in the translate file. Also you see that Patgen has read the word `A-GA-\varpi␣A-EI` exactly as if it were written `a-ga-pa-ei` and produced the pattern `a1p` (if you look in the dictionary word list there is no other reason for this pattern to exist).

And here is our result (file `pattmp.2`):

```
a*nu*po*ta*qtos
a*ka*ta*m'a*qh*tos
a*ei*mnh*stos
ma*na
tour*ko*gu*ftis*sa
nu*qo*ko*pths
```

```
li*me*no*fu*la*kas
mpre*lok
a*ga*pa*ei
pei*rai*'as
```

As you see there is no `\varpi␣` anymore: Patgen has replaced it by `p`, and so `A-GA-\varpi␣A-EI` has become `a*ga*pa*ei`.

# 6 Go Forth, etc., etc.

Imitating the Grand Wizard, I invite you to Go Forth and make masterpieces of hyphenation patterns…

And once you created them, you can contact the maintainers of packages such as Polyglossia or Babel to have them included in them—or you can directly send your patterns to CTAN so that everybody in the TEX community can use them.

# References


[1] Susan Bartlett, Grzegorz Kondrak, and Colin Cherry. Automatic syllabification with structured svms for letter-to-phoneme conversion. In *ACL 2008, Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, June 15-20, 2008, Columbus, Ohio, USA*, pages 568–576. Association for Computational Linguistics, 2008.

[2] Donald E. Knuth. *The TEXbook*. Addison-Wesley, 1984.

[3] Jacob Krantz, Maxwell Dulin, and Paul De Palma. Language-agnostic syllabification with neural sequence labeling. arXiv:1909.13362, 2019.

[4] Franklin Mark Liang. *Word Hy-phen-a-tion by Comp-put-er*. PhD thesis, Stanford University, 1983. `https://www.tug.org/docs/liang/liang-thesis.pdf`.