New hyphenation techniques in Ω_2

YANNIS HARALAMBOUS
Département Informatique, ENST Bretagne, CS 83 818, 29 283 Brest Cedex 3, France
`yannis dot haralambous (at) enst dash bretagne dot fr`

Abstract

By replacing the internal hyphenation engine of TEX by an external Omega$_2$ module, we are able to solve all shortcomings related to hyphenation and to add new features: segmentation of compound words, excentricity, preferential hyphenation.

Introduction

Ever since a computer hyphenated the word "God" and ruined a night's sleep of an RCA employee, there has been quite a lot of literature on hyphenation:[1] it is a complex linguistic operation, permanently in use (at least for languages that are hyphenated), and requiring high efficiency. Nevertheless there are some flaws in TEX's approach to hyphenation, as well as some areas where extra features could be helpful.

The flaws are mainly (a) the fact that hyphenation patterns are stored in the format, so that one needs to know in advance which languages will be used in the document and create the appropriate format file; (b) in some contexts, words are not hyphenatable because they are not preceded by glue (for example, a word preceded by a penalty, or the first word of a paragraph); (c) font or color changes prohibit hyphenation, so that a word like "différance" cannot be hyphenated; and (d) special hyphenations such as German *backen* becoming *bak-ken* are possible (through the *discretionary* primitive) but cannot be automatic.

New features have been suggested on many occasions: for example, it would be very useful for some languages to prioritize hyphenation between word components rather than between syllables in the same component. In German, the priority list is even threefold: first comes hyphenation between components, then hyphenation inside the last component, and last *and* least: hyphenation inside the other components. Another useful extra feature is weighted hyphenation: for example, in French, words starting with "con" should not be hyphenated at that location, but this restriction should not be absolute: if one cannot do otherwise, it should be allowed to hyphenate nevertheless (a typical example is the word "conscience" which can be hyphenated only after "con"). So one should be able to specify a penalty value for each hyphen. Another useful feature would be interaction with the user in case of ambiguity requiring morphological, syntactical or even semantic input: a typical case is already stated in *The TEXbook*: "rec-ord" (the noun) vs. "re-cord" (the verb). Whenever the algorithm detects such an ambiguity, the user should be warned and, why not, asked to disambiguate.

In languages like Greek there is no requirement for hyphenating between word components.[2] Nevertheless, although it is allowed, it looks silly to hyphenate a word such as Π*απαχατζηχαραλαμπό- πουλος* after the two first letters. In such cases it would be useful to prioritize hyphenation towards the middle of the word rather than near its borders.

In this paper we present the new hyphenation module of Omega$_2$, which solved the problems mentioned and provides infrastructure for the extra features described above.

[1]See, in particular, [1, 6, 7, 8, 10, 11, 12, 13, 14].

[2]The reader may wonder why there is such a requirement for German and not for Greek, which uses as least as many compound words as German. Here is a possible explanation: in German, compound words are separated by a glottal stop. For example, *Satzende* will be pronounced "zats[break]ende" to distinguish the components *Satz* (= sentence) and *Ende* (= end). The visually similar *Sitzende* will be pronounced continuously "zitsende" (= sitting). In Greek, however, there is no glottal stop: *συνάδϵλφος* (= colleague) is pronounced continuously "sinathelfos" although it is composed by *συν* (= plus, together) and *αδϵλφος* (= brother). This feature of the modern Greek language has influenced hyphenation practice. In fact, there are two ways to hyphenate this word in Greek: in modern *dêmotikê* [5] it will be hyphenated phonetically *συ-νά-δϵλ-φος* (which contradicts component segmentation), and in *kathareuousa* it will be hyphenated etymologically *συν-άδϵλ-φος*. The question of whether *kathareuousa* hyphenation patterns should give priority to word components is open.

Description

A *module* for Omega$_2$ is a binary reading and writing horizontal node lists serialized in XML. It is hooked into Omega$_2$ at two possible locations: one is inside the *end_graph* procedure (§1096 of [9], just before the call of *line_break*: that's when a complete paragraph is sent to the paragraph builder). The second location (which has not been implemented yet) is inside the paragraph builder, for a given vertice of the graph of break nodes.

To say it simply: a module extracts horizontal node lists from Omega's stomach, modifies them, and puts them back so that typesetting can go on.

But there is something more in Omega$_2$: instead of character nodes, we use *texteme* nodes [3], so that we clearly separate glyphs from characters and that we can add arbitrary name/value pairs to each node.

In our case, the hyphenation module will study the paragraph, apply the hyphenation algorithm to textemes and add a "potential hyphenation point" property to some of them. The value of this property is not simply a boolean but rather a penalty, so that the paragraph builder will automatically prioritize some hyphenation points (for example, those between word components).

Problems solved

Let us see how our approach solves the five problems described in the first section.

Problem a: Preloading of patterns

Omega$_2$ does not preload any patterns in the format file. Patterns contained in external files are dynamically loaded by the module (and subsequently kept in cache), whenever they are needed.

Problem b: Unhyphenatable words

This problem has always been a nightmare for TEX users: now and then, for reasons which seem obscure to the average user, a word will not be hyphenated, resulting in a horrible overfull box. Most of the time the reason is the fact that the word is preceded by something which is not glue. Indeed, according to §891 of [9], *a "potentially hyphenatable part" of a paragraph is a sequence of nodes* $p_0p_1\ldots p_m$ *where* p_0 *is a glue node,* $p_1\ldots p_{m-1}$ *are character, or ligature or whatsit or implicit kern nodes, and* p_m *is a glue or penalty or insertion or adjust or mark or whatsit or explicit kern node* (see also [4]).

Since now hyphenation is external to Omega, we can define our own rules. One can apply the original TEX rules so that we obtain the same results. Or one can define new rules and obtain potentially better results.

Problem c: Font or color change

This problem comes from the fact that, as often in TEX, the rules for hyphenatable words we described above are not complete. Other restrictions arrive as we read §891: *a whatsit node found after* p_1 *will be the terminating node* p_m and *all character nodes that do not have the same font as the first character node, will be treated as nonletters*. That means that a *special* primitive or a font change inside a word prohibits hyphenation after them.

In fact, the use of textemes allows us to inject into text properties which will not disable hyphenation. A less typical example is vertical offset: up to now, the TEX logo was not a word, but a graphical construction using glyphs. Using texteme properties to specify the lowering of letter 'E' it will be at last possible to hyphenate the title of the well-known journal *Die TEXnische Komödie*!

As said in the previous section, since we define our own rules, this restriction can very well be abandoned.

Problem d: Special hyphenations

There is no miracle for that: the various special cases have been hard-coded in the module (one could imagine a syntax for including them in the patterns, but the author considers that their extreme rarity does not justify a syntax enhancement).

New features

Penalties

As said in [9] §145, a discretionary node produces either a *hyphen_penalty* or an *ex_hyphen_penalty* depending on its pre-break text. This penalty can be changed by the user, but on a global level only, and certainly not separately for each hyphen point. In Omega$_2$, there are 65,536 hyphenation penalty registers. Patterns can contain a hyphenation register number (the default register being 0). The hyphenation engine will transmit the highest register number value to the paragraph builder through a texteme property.

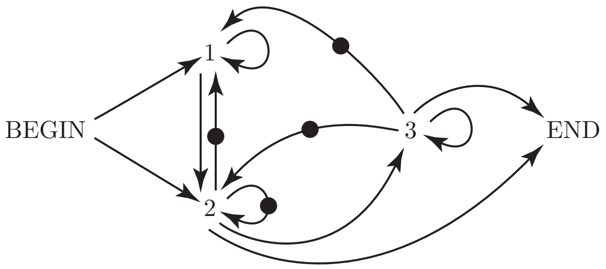


Figure 1: Finite-state engine of *SiSiSi*

The paragraph builder will use the penalty value of that register.

It is up to the user to take advantage of these penalties to prioritize hyphenation between word components, or simply to make the paragraph builder prefer a given hyphenation point rather than some other.

The obvious question which remains is: how do we calculate the various hyphenation penalties in the case of, for example, word component boundaries.

Excentricity

To prioritize hyphenation points that occur near the middle of words, we have introduced a number called *excentricity factor* (a deliberate neologism). This number is the slope of a linear function giving the hyphenation register number according to the distance of an hyphenation point from the center of the word: if ϕ is the factor, i the position of the letter in the word and c the center of the word, then the hyphenation penalty register will be 0 for letter c, $\mathrm{int}((c \cdot i) \cdot s)$ when $c > i$ and $\mathrm{int}((i \cdot c + 1) \cdot s)$ otherwise. So, for example, the word

$$\Pi\alpha|\pi\alpha|\chi\alpha|\tau\zeta\eta|\chi\alpha|\rho\underline{\alpha}|\lambda\alpha\mu|\pi\acute{o}|\pi o\upsilon|\lambda o\varsigma$$

with an excentricity factor of, for example, $\phi = 0.33$, will be hyphenated with penalties contained in the following registers:

$$\Pi\alpha|_3\pi\alpha|_3\chi\alpha|_2\tau\zeta\eta|_1\chi\alpha|_0\rho\underline{\alpha}|_0\lambda\alpha\mu|_1\pi\acute{o}|_1\pi o\upsilon|_2\lambda o\varsigma$$

It is up to the user to choose the penalty values for each of registers 0, 1, 2, 3. With a higher slope, we get more registers and are able to control hyphenation penalties more finely.

A Finite-State Engine

There are two ways to calculate word component boundaries. Either by using again patterns (as suggested by Antoš in [1]), or by using a finite-state engine, as used by spelling checkers such as *huspell*.

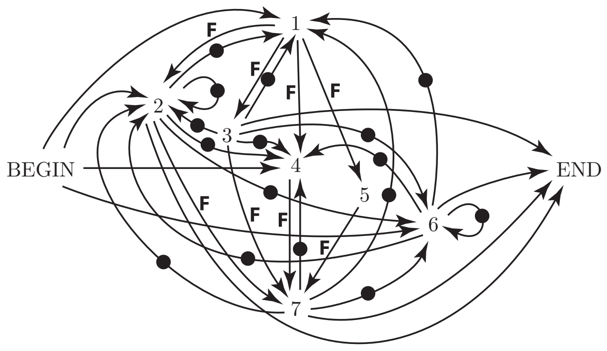


Figure 2: Finite-state engine of *huspell*

Let us develop the second case. A first approach to word component detection through a finite state engine was *SiSiSi* (= *Sichere sinnentsprechende Silbentrennung für die deutsche Sprache* = "Reliable and Sense-Conveying Hyphenation for the German language"), a TEX extension developed in the early nineties by Barth and Steiner [2] based on their work on hyphenation from the eighties. In *SiSiSi* a word is segmented in three parts: a series of prefixes, a single stem and a series of suffixes. This can be described by the finite-state engine in fig. 1: states 1, 2 and 3 are resp. the one of prefix, stem and suffix. We have twelve transitions BEGIN→1 BEGIN→2 1→1 1→2 2→*1 2→*2 2→3 2→END 3→*1 3→*2 3→3 3→END, where the asterisk (or • on the figure) means that going through this transition we enter a new word component.

This approach has been proven to have relatively low efficiency (the *SiSiSi* system was strongly relying on interaction with the user to find and store exceptions to rules).

Another word segmentation approach is the one of spelling checkers. Ispell-based checkers (like *huspell*, which seems to be the most advanced variant) use a "file of affixes" containing lines of the like:

```
SFX A   lig   elig   [^aeiouhlräüö]lig
```

which means: there is a set of rules called A containing this rule; this rule says: when you see a word ending by some string matching the regular expression `/[^aeiouhlräüö]lig/` then strip `lig` and add `elig`. These rules generate new word forms from the existing ones, whenever the latter satisfy the requirements. Similar rules exist for prefixes (starting with PFX).

We have modeled this approach as a finite-state machine with 7 states and 32 transitions. The 7 states are: (1) prefix, (2) stems which are not modified by an

SFX or PFX rule, (3) stems which have been modified by a PFX rule (after modification), (4) stems which have been modified by a SFX rule (after modification), (5) stems which have been modified by both a PFX and a SFX rule (after modification), (6) stem which cannot be combined with either a prefix or a suffix, (7) suffix. It should be noted that only 1, 2, 4 and 6 can be at word begin, and only 2, 3, 6 and 7 can be at word end. The reader can find a graphical representation of this engine in fig. 2.

Here are the transitions: BEGIN→1 BEGIN→2 BEGIN→4 BEGIN→6 1→2F 1→3F 1→4F 1→5F 2→*1 2→*2 2→*4 2→*6 2→7F 2→END 3→*1 3→*2 3→*4 3→*6 3→7F 3→END 4→7F 5→7F 6→*1 6→*2 6→*4 6→*6 6→END 7→*1 7→*2 7→*4 7→*6 7→END. Those marked by an 'F' are conditional transitions: they only apply whenever left and right string belong to the same "family." Families are necessary because of the regular expressions in SFX and PFX rules.

We will see in the next section how this information is included in the patterns file.

Patterns file

TeX users are used to patterns files containing a *patterns* primitive and, in many cases, a *hyphenation* primitive for exceptions. These files sometimes also contain *lccode* commands because only characters with nonzero *lccode* are recognized by TeX's hyphenation algorithm. This part of the code works also as a mapper to equivalence classes, so that patterns can be written using those classes rather than explicit characters. For example, in the case of Greek one can map all combinations of letter *alpha* and diacritics to a single equivalence class and use that class in the patterns. That way, one has fewer patterns and the result is the same (provided, of course, that hyphenation rules are independent of diacritics).

In our new hyphenation patterns files we keep the same pseudo-commands `\patterns` and `\hyphenation`—"pseudo" because these files are not read by TeX anymore, but by a tool of our own, called *inittrie*, written in C. These files, for which we recommend the file extension `.pat`, are compiled into compressed binary form (file extension `.hyp`) by *inittrie*. This binary form contains a certain number of tries, exactly as formerly stored in TeX format files.

Here is a detailed description of the ingredients of `.pat` files.

(left | right)hyphenmin
The primitives *lefthyphenmin* and *righthyphenmin* are used in TeX to specify the minimal number of letters to leave on a line, or to allow on the next line. In our case, these values are included in the patterns file, so that there is no need to worry about them in the TeX file, or Babel language file, etc.

equivalents
The argument of this command consists of a big number of character pairs, separated by blanks. These character pairs play the same role as *lccode* instructions in TeX. In each pair, the first character is a character considered to be a letter by the hyphenation algorithm, and the second one is its equivalence class. These characters are, of course, all written in Unicode UTF-8. In fig. 3 the reader can see this command displayed under Mac OS X.

patterns
Patterns are described in the same way as in TeX hyphenation files, with an extra convention: numbers between brackets specify the number of the hyphenation penalty register requested. So, for example, the hypothetical pattern

```
.con8s
```

for French language can be replaced by

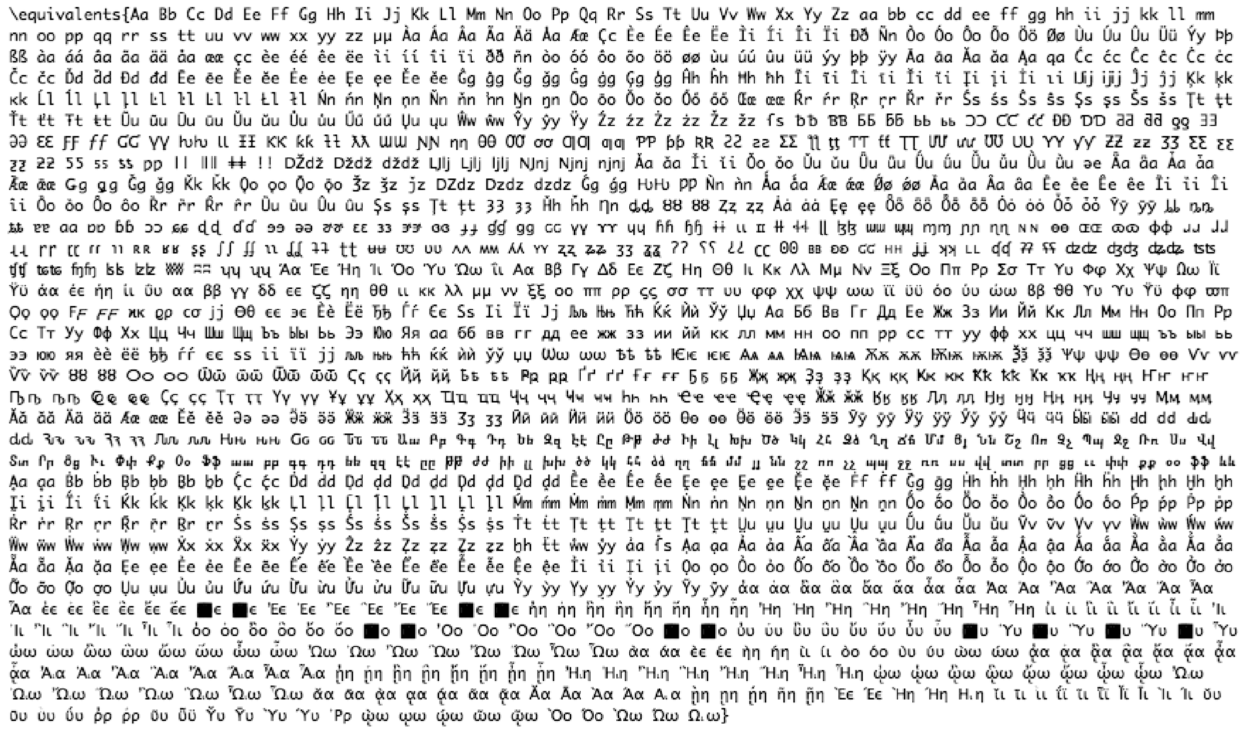
```
.con8[17]s
```

so that the value of hyphenation penalty register 17 will be used.

segmenthyphenpenalty
Using this pseudo-command one can specify the hyphenation penalty register to be used between word components. Default value is 1.

segmenthyphencharacter
Through this pseudo-command one can specify the hyphenation character to be used between word components. Default value is the hyphen. In cases like Thai, where we really segment a sentence into words (and words into syllables), the segment hyphenation character will be empty.

segmentpatterns
The syntax of the argument of this pseudo-command is the same as for *patterns*. These patterns will be used to obtain word components rather than syllables. The other two arguments specify the number of hyphen-



```
\equivalents{Aa Bb Cc Dd Ee Ff Gg Hh Ii Jj Kk Ll Mm Nn Oo Pp Qq Rr Ss Tt Uu Vv Ww Xx Yy Zz aa bb cc dd ee ff gg hh ii jj kk ll mm
nn oo pp qq rr ss tt uu vv ww xx yy zz µμ Àa Áa Âa Ãa Ää Åa Ææ Çc Èe Ée Êe Ëe Ìi Íi Îi Ïi Ðð Ñn Òo Óo Ôo Õo Öö Øø Ùu Úu Ûu Üü Ýy Þþ
ẞß àa áá âa ãa ää åa ææ çc èe éé êe ëe ìi íí îi ïi ðð ñn òo óó ôo õo öö øø ùu úú ûu üü ýy þþ ÿy Āa āa Ăa ăa Ąa ąa Ćc ćc Ĉc ĉc Ċc ċc
Čc čc Ďd ďd Đd đd Ēe ēe Ĕe ĕe Ėe ėe Ęe ęe Ěe ěe Ĝg ĝg Ğg ğg Ġg ġg Ģg ģg Ĥh ĥh Ħh ħh Ĩi ĩi Īi īi Ĭi ĭi Įi įi İi ıi Ĳij ĳij Ĵj ĵj Ķk ķk
ĸk Ĺl ĺl Ļl ļl Ľl ľl Ŀl ŀl Łl łl Ńn ńn Ņn ņn Ňn ňn ŉn Ŋn ŋn Ōo ōo Ŏo ŏo Őő őő Œœ œœ Ŕr ŕr Ŗr ŗr Řr řr Śs śs Ŝs ŝs Şs şs Šs šs Ţt ţt
Ťt ťt Ŧt ŧt Ũu ũu Ūu ūu Ŭu ŭu Ůu ůu Űű űű Ųu ųu Ŵw ŵw Ŷy ŷy Ÿy Źz źz Żz żz Žz žz ſs ƀƀ Ɓɓ Ƃƃ ƃƃ Ƅƅ ƅƅ Ɔɔ Ƈƈ ƈƈ Ɖɖ Ɗɗ Ƌƌ ƌƌ ƍƍ Ǝǝ
Əə Ɛɛ Ƒƒ ƒƒ Ɠɠ Ɣɣ ƕƕ Ɩɩ Ɨɨ Ƙƙ ƙƙ ƚƚ ƛƛ Ɯɯ Ɲɲ ƞƞ Ɵɵ Ơơ ơơ Ƣƣ ƣƣ Ƥƥ ƥƥ Ʀʀ Ƨƨ ƨƨ Ʃʃ ƪƪ ƫƫ Ƭƭ ƭƭ Ʈʈ Ưư ưư Ʊʊ Ʋʋ Ƴƴ ƴƴ Ƶƶ ƶƶ Ʒʒ Ƹƹ ƹƹ
ƺƺ ƻƻ Ƽƽ ƽƽ ƾƾ ƿƿ ǀǀ ǁǁ ǂǂ ǃǃ DŽdž Dždž dždž LJlj Ljlj ljlj NJnj Njnj njnj Ǎa ǎa Ǐi ǐi Ǒo ǒo Ǔu ǔu Ǘu ǘu Ǚu ǚu Ǜu ǜu ǝe Ǟa ǟa Ǡa ǡa
Ǣæ ǣæ Ǥg ǥg Ǧg ǧg Ǩk ǩk Ǫo ǫo Ǭo ǭo Ǯz ǯz ǰj DZdz Dzdz dzdz Ǵg ǵg Ƕƕ Ƿƿ Ǹn ǹn Ǻa ǻa Ǽæ ǽæ Ǿø ǿø Ȁa ȁa Ȃa ȃa Ȅe ȅe Ȇe ȇe Ȉi ȉi Ȋi
ȋi Ȍo ȍo Ȏo ȏo Ȑr ȑr Ȓr ȓr Ȕu ȕu Ȗu ȗu Șs șs Țt țt Ȝȝ ȝȝ Ȟh ȟh Ƞn ȡȡ Ȣȣ ȣȣ Ȥz ȥz Ȧa ȧa Ȩę ȩę Ȫö ȫö Ȭõ ȭõ Ȯo ȯo Ȱȯ ȱȯ Ȳȳ ȳȳ ȴȴ ȵȵ
ȶȶ ɐɐ ɑɑ ɒɒ ɓɓ ɔɔ ɕɕ ɖɖ ɗɗ ɘɘ əə ɚɚ ɛɛ ɜɜ ɝɝ ɞɞ ɟɟ ɠɠ ɡɡ ɢɢ ɣɣ ɤɤ ɥɥ ɦɦ ɧɧ ɨɨ ɩɩ ɪɪ ɫɫ ɬɬ ɭɭ ɮɮ ɯɯ ɰɰ ɱɱ ɲɲ ɳɳ ɴɴ ɵɵ ɶɶ ɷɷ ɸɸ ɹɹ ɺɺ
ɻɻ ɼɼ ɽɽ ɾɾ ɿɿ ʀʀ ʁʁ ʂʂ ʃʃ ʄʄ ʅʅ ʆʆ ʇʇ ʈʈ ʉʉ ʊʊ ʋʋ ʌʌ ʍʍ ʎʎ ʏʏ ʐʐ ʑʑ ʒʒ ʓʓ ʔʔ ʕʕ ʖʖ ʗʗ ʘʘ ʙʙ ʚʚ ʛʛ ʜʜ ʝʝ ʞʞ ʟʟ ʠʠ ʡʡ ʢʢ ʣʣ ʤʤ ʥʥ ʦʦ
ʧʧ ʨʨ ʩʩ ʪʪ ʫʫ ʬʬ ʭʭ ʮʮ ʯʯ Άα Έε Ήη Ίι Όο Ύυ Ώω ΐι Αα Ββ Γγ Δδ Εε Ζζ Ηη Θθ Ιι Κκ Λλ Μμ Νν Ξξ Οο Ππ Ρρ Σσ Ττ Υυ Φφ Χχ Ψψ Ωω Ϊϊ
Ϋϋ άα έε ήη ίι ΰυ αα ββ γγ δδ εε ζζ ηη θθ ιι κκ λλ μμ νν ξξ οο ππ ρρ ςς σσ ττ υυ φφ χχ ψψ ωω ϊϊ ϋϋ όο ύυ ώω ϐβ ϑθ ϒυ ϓυ ϔϋ ϕφ ϖπ
Ϙϙ ϙϙ Ϝϝ ϝϝ ϰκ ϱρ ϲσ ϳj ϴθ ϵε ϶϶ Ѐѐ Ёё Ђђ Ѓѓ Єє Ѕs Іi Її Јj Љљ Њњ Ћћ Ќќ Ѝѝ Ўў Џџ Аа Бб Вв Гг Дд Ее Жж Зз Ии Йй Кк Лл Мм Нн Оо Пп Рр
Сс Тт Уу Фф Хх Цц Чч Шш Щщ Ъъ Ыы Ьь Ээ Юю Яя аа бб вв гг дд ее жж зз ии йй кк лл мм нн оо пп рр сс тт уу фф хх цц чч шш щщ ъъ ыы ьь
ээ юю яя ѐѐ ёё ђђ ѓѓ єє ѕѕ іі її јј љљ њњ ћћ ќќ ѝѝ ўў џџ Ѡѡ ѡѡ Ѣѣ ѣѣ Ѥѥ ѥѥ Ѧѧ ѧѧ Ѩѩ ѩѩ Ѫѫ ѫѫ Ѭѭ ѭѭ Ѯѯ ѯѯ Ѱѱ ѱѱ Ѳѳ ѳѳ Ѵѵ ѵѵ
Ѷѷ ѷѷ Ѹѹ ѹѹ Ѻѻ ѻѻ Ѽѽ ѽѽ Ѿѿ ѿѿ Ҁҁ ҁҁ Ҋҋ ҋҋ Ҍҍ ҍҍ Ҏҏ ҏҏ Ґґ ґґ Ғғ ғғ Ҕҕ ҕҕ Җҗ җҗ Ҙҙ ҙҙ Ққ ққ Ҝҝ ҝҝ Ҟҟ ҟҟ Ҡҡ ҡҡ Ңң ңң Ҥҥ ҥҥ
Ҧҧ ҧҧ Ҩҩ ҩҩ Ҫҫ ҫҫ Ҭҭ ҭҭ Үү үү Ұұ ұұ Ҳҳ ҳҳ Ҵҵ ҵҵ Ҷҷ ҷҷ Ҹҹ ҹҹ Һһ һһ Ҽҽ ҽҽ Ҿҿ ҿҿ Ӂӂ ӂӂ Ӄӄ ӄӄ Ӆӆ ӆӆ Ӈӈ ӈӈ Ӊӊ ӊӊ Ӌӌ ӌӌ Ӎӎ ӎӎ
Ӑӑ ӑӑ Ӓӓ ӓӓ Ӕӕ ӕӕ Ӗӗ ӗӗ Әә әә Ӛӛ ӛӛ Ӝӝ ӝӝ Ӟӟ ӟӟ Ӡӡ ӡӡ Ӣӣ ӣӣ Ӥӥ ӥӥ Ӧӧ ӧӧ Өө өө Ӫӫ ӫӫ Ӭӭ ӭӭ Ӯӯ ӯӯ Ӱӱ ӱӱ Ӳӳ ӳӳ Ӵӵ ӵӵ Ӹӹ ӹӹ Ԁԁ ԁԁ Ԃԃ
ԃԃ Ԅԅ ԅԅ Ԇԇ ԇԇ Ԉԉ ԉԉ Ԋԋ ԋԋ Ԍԍ ԍԍ Ԏԏ ԏԏ Աա Բբ Գգ Դդ Եե Զզ Էէ Ըը Թթ Ժժ Իի Լլ Խխ Ծծ Կկ Հհ Ձձ Ղղ Ճճ Մմ Յյ Նն Շշ Ոո Չչ Պպ Ջջ Ռռ Սս Վվ
Տտ Րր Ցց Ււ Փփ Քք Օօ Ֆֆ աա բբ գգ դդ եե զզ էէ ըը թթ ժժ իի լլ խխ ծծ կկ հհ ձձ ղղ ճճ մմ յյ նն շշ ոո չչ պպ ջջ ռռ սս վվ տտ րր ցց ււ փփ քք օօ ֆֆ ևև
Ḁḁ ḁḁ Ḃḃ ḃḃ Ḅḅ ḅḅ Ḇḇ ḇḇ Ḉḉ ḉḉ Ḋḋ ḋḋ Ḍḍ ḍḍ Ḏḏ ḏḏ Ḑḑ ḑḑ Ḓḓ ḓḓ Ḕḕ ḕḕ Ḗḗ ḗḗ Ḙḙ ḙḙ Ḛḛ ḛḛ Ḝḝ ḝḝ Ḟḟ ḟḟ Ḡḡ ḡḡ Ḣḣ ḣḣ Ḥḥ ḥḥ Ḧḧ ḧḧ Ḩḩ ḩḩ Ḫḫ ḫḫ
Ḭḭ ḭḭ Ḯḯ ḯḯ Ḱḱ ḱḱ Ḳḳ ḳḳ Ḵḵ ḵḵ Ḷḷ ḷḷ Ḹḹ ḹḹ Ḻḻ ḻḻ Ḽḽ ḽḽ Ḿḿ ḿḿ Ṁṁ ṁṁ Ṃṃ ṃṃ Ṅṅ ṅṅ Ṇṇ ṇṇ Ṉṉ ṉṉ Ṋṋ ṋṋ Ṍṍ ṍṍ Ṏṏ ṏṏ Ṑṑ ṑṑ Ṓṓ ṓṓ Ṕṕ ṕṕ Ṗṗ ṗṗ
Ṙṙ ṙṙ Ṛṛ ṛṛ Ṝṝ ṝṝ Ṟṟ ṟṟ Ṡṡ ṡṡ Ṣṣ ṣṣ Ṥṥ ṥṥ Ṧṧ ṧṧ Ṩṩ ṩṩ Ṫṫ ṫṫ Ṭṭ ṭṭ Ṯṯ ṯṯ Ṱṱ ṱṱ Ṳṳ ṳṳ Ṵṵ ṵṵ Ṷṷ ṷṷ Ṹṹ ṹṹ Ṻṻ ṻṻ Ṽṽ ṽṽ Ṿṿ ṿṿ Ẁẁ ẁẁ Ẃẃ ẃẃ
Ẅẅ ẅẅ Ẇẇ ẇẇ Ẉẉ ẉẉ Ẋẋ ẋẋ Ẍẍ ẍẍ Ẏẏ ẏẏ Ẑẑ ẑẑ Ẓẓ ẓẓ Ẕẕ ẕẕ ẖh ẗt ẘw ẙy ẚa ẛs Ạạ ạạ Ảả ảả Ấấ ấấ Ầầ ầầ Ẩẩ ẩẩ Ẫẫ ẫẫ Ậậ ậậ Ắắ ắắ Ằằ ằằ Ẳẳ ẳẳ
Ẵẵ ẵẵ Ặặ ặặ Ẹẹ ẹẹ Ẻẻ ẻẻ Ẽẽ ẽẽ Ếế ếế Ềề ềề Ểể ểể Ễễ ễễ Ệệ ệệ Ỉỉ ỉỉ Ịị ịị Ọọ ọọ Ỏỏ ỏỏ Ốố ốố Ồồ ồồ Ổổ ổổ Ỗỗ ỗỗ Ộộ ộộ Ớớ ớớ Ờờ ờờ Ởở ởở
Ỡỡ ỡỡ Ợợ ợợ Ụụ ụụ Ủủ ủủ Ứứ ứứ Ừừ ừừ Ửử ửử Ữữ ữữ Ựự ựự Ỳỳ ỳỳ Ỵỵ ỵỵ Ỷỷ ỷỷ Ỹỹ ỹỹ ἀα ἁα ἂα ἃα ἄα ἅα ἆα ἇα Ἀα Ἁα Ἂα Ἃα Ἄα Ἅα Ἆα
Ἇα ἐε ἑε ἒε ἓε ἔε ἕε ■ε ■ε Ἐε Ἑε Ἒε Ἓε Ἔε Ἕε ■ε ■ε ἠη ἡη ἢη ἣη ἤη ἥη ἦη ἧη Ἠη Ἡη Ἢη Ἣη Ἤη Ἥη Ἦη Ἧη ἰι ἱι ἲι ἳι ἴι ἵι ἶι ἷι Ἰι
Ἱι Ἲι Ἳι Ἴι Ἵι Ἶι Ἷι ὀο ὁο ὂο ὃο ὄο ὅο ■ο ■ο Ὀο Ὁο Ὂο Ὃο Ὄο Ὅο ■ο ■ο ὐυ ὑυ ὒυ ὓυ ὔυ ὕυ ὖυ ὗυ ■υ Ὑυ ■υ Ὓυ ■υ Ὕυ ■υ Ὗυ
ὠω ὡω ὢω ὣω ὤω ὥω ὦω ὧω Ὠω Ὡω Ὢω Ὣω Ὤω Ὥω Ὦω Ὧω ὰα άα ὲε έε ὴη ήη ὶι ίι ὸο όο ὺυ ύυ ὼω ώω ᾀα ᾁα ᾂα ᾃα ᾄα ᾅα ᾆα
ᾇα ᾈα ᾉα ᾊα ᾋα ᾌα ᾍα ᾎα ᾏα ᾐη ᾑη ᾒη ᾓη ᾔη ᾕη ᾖη ᾗη ᾘη ᾙη ᾚη ᾛη ᾜη ᾝη ᾞη ᾟη ᾠω ᾡω ᾢω ᾣω ᾤω ᾥω ᾦω ᾧω ᾨω
ᾩω ᾪω ᾫω ᾬω ᾭω ᾮω ᾯω ᾰα ᾱα ᾲα ᾳα ᾴα ᾶα ᾷα Ᾰα Ᾱα Ὰα Άα ᾼα ῂη ῃη ῄη ῆη ῇη Ὲε Έε Ὴη Ήη ῌη ῐι ῑι ῒι ΐι ῖι ῗι Ῐι Ῑι Ὶι Ίι ῠυ
ῡυ ῢυ ΰυ ῤρ ῥρ ῦυ ῧυ Ῠυ Ῡυ Ὺυ Ύυ Ῥρ ῲω ῳω ῴω ῶω ῷω Ὸο Όο Ὼω Ώω ῼω}
```

Figure 3: Table of equivalents

ation penalty register to be used, and the requested secondary hyphenation character.

transitions

To use a finite-state engine, we need the commands `\transitions`, `\references` and `\segments` instead of the command `\segmentpatterns`. In the argument of the `\transitions` command we will write transitions in the syntax given above: strings `BEGIN` and `END` for beginnings and ends of words, numbers for all other transitions, the string `->` between origin and destination of a transition. Example: `BEGIN->1 BEGIN->2 1->1 1->2 2->*1 2->*2 2->3 2->END 3->*1 3->*2 3->3 3->END` for the *SiSiSi* model. The asterisk means that the given transition produces a new segment. A destination followed by an `F` means that there is a filter: the transition occurs only if origin and destination belong to the same family.

references

This command contains “references”. A reference is a set of families. The idea is the following: a segment very often belongs to several families (meaning that it can be combined with many other segments, in order to form components and words). Instead of writing all the families to which each segment belongs, we will in fact use a single number. This number will be an index to the set of families to which it belongs, its reference.

The syntax is shown by the following example:

```
2368=/843/844/845/921/943
```

meaning that segments followed by number 2368 belong to families 843, 844, 845, 921 and 943. When checking whether a transition is allowed, our algorithm will not check if the references are the same, but rather if they have a non-empty common set of families. References are separated by blanks.

segments

Segments are classified by the state to which they belong. The argument of `\segments` looks like:

```
\segments{
1: %prefixes
a2083 abba2084 agyon2084 alá2084 b2085
be2084 bele2084 benn2084 benn-2084
billió2086 c2087 d2088 e2089 egy2086
egybe2084 el2084 ...
2: %original stems
aba3 abafala3 abafalva3 abaffy5 abafi6
abafája3 abajgat8 abakteriális9 ...
3: %pre-altered stems
...
4: %post-altered stems
...
ab2 abafal2 abafalv4 abafáj2 abajga7
abalehot2 abar2 abaújharaszt14
abaújszakal18 abaújszin2 abaújtorn2
abd2 abell25 abelov2 ...
5: %prepost-altered stems
...
6: %stems without affixes
...
7: %suffixes
abbak2137 abbakat2138 abbakba2138
abbakban2138 abbakból2138 abbakhoz2138
abbakig2138 abbakkal2138 abbakká2138
abbaknak2138 ...
}
```

A number followed by a colon denotes a state. Segments are separated by blanks. They are followed by reference numbers.

If the finite-state engine does not require family filtering, then the *references* command will be empty and segments will not be followed by any number.

lastsegmentpriority
Whenever this option is included in the patterns file, the hyphenation points in the last segment have their hyphenation penalty registers increased by a given amount.

excentricity
Gives the excentricity factor.

References

[1] Antoš D., "PATLIB, Pattern Manipulation Library", Master Thesis, Masaryk University Brno, 2001. `http://www.fi.muni.cz/~xantos/patlib/thesis/thesis-p.pdf`

[2] Barth W., Steiner H., "Deutsche Silbentrennung für TEX 3.1", *Die TEXnische Komödie*, 1, 1992, pp. 33-35. `http://www.dante.de/dante/DTK/dtk92_1/dtk92_1_barth_steiner_deutsche.html` and `http://www.ads.tuwien.ac.at/research/SiSiSi.html`

[3] Haralambous Y., Bella G., "Injecting Information into Atomic Units of Text", in Proceedings of the ACM Symposium on Document Engineering, Bristol, 2005. `http://omega.enstb.org/yannis/pdf/fp10174-haralambous.pdf`

[4] Haralambous Y., "Voyage au centre de TEX : composition, paragraphage, césure", *Cahiers GUTenberg* 44-45, 2004, pp. 3-53. `http://omega.enstb.org/yannis/pdf/voyage.pdf`

[5] Haralambous Y., "From Unicode to Typography, a Case Study: the Greek Script", Proceedings of International Unicode Conference XIV, Boston, 1999, pp. b.10.1–b.10.36. `http://omega.enstb.org/yannis/pdf/boston99.pdf`

[6] Haralambous Y., "A Small Tutorial on the Multilingual Features of PATGEN2", in electronic form, available from CTAN as `info/patgen2.tutorial`, 1994.

[7] Haralambous Y., "Using PATGEN to Create Welsh Patterns", submitted to *TUGboat*, 1993.

[8] Haralambous Y., "Hyphenation Patterns for Ancient Greek and Latin", *TUGboat* 13 (4), 1992, pp. 457-469. `http://omega.enstb.org/yannis/pdf/ancgreek92.pdf`

[9] Knuth D.E., *Computers & Typesetting, Vol. B: TEX, The Program*, Addison-Wesley, 1986.

[10] Raichle B., "Hyphenation patterns for words containing explicit hyphens", `CTAN/language/hyphenation/hypht1.tex`, 1997.

[11] Scannell K, "Hyphenation Patterns for Minority Languages", *TUGboat* 24 (2), 2003, pp. 236–239. `http://www.tug.org/TUGboat/Articles/tb24-2/tb77scannell.pdf`

[12] Sojka P., "Hyphenation on Demand", *TUGboat* 20 (3), 1999. `http://www.tug.org/TUGboat/Articles/tb20-1/tb62sched.pdf`

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

[14] Sojka P., "Notes on Compound Word Hyphenation in TEX", *TUGboat* 16 (3), pp. 290–297, 1995. `http://www.tug.org/TUGboat/Articles/tb16-3/tb48soj2.pdf`