Pattern Generation Revisited

David Antoš, Petr Sojka

Faculty of Informatics
Masaryk University Brno

Czech Republic

What is this lecture about

- How TEX hyphenates words
- Generation of patterns
- Using PATGEN
- Why PATGEN doesn't suit us today
- PATLIB and OPATGEN, system architecture
- Pattern recognition in typesetting
- Future applications

How TEX hyphenates words

- Before 2nd pass of paragraph breaking, all words
- TEX finds all *patterns* of the word, all hyphens
- Pattern is a subword, with hyphenation information between symbols
  - `n2at hen5at`
- Hyphenation information—numbers
  - *odd*—allow breaking—*covering*
  - *even*—disallow breaking—*inhibiting*
- Patterns *compete*—higher value wins

Example of application of patterns

```
 h y p h e n a t i o n
          1n a
              1t i o
           n2a t
                2i o
       h e2n
.h y3p h
       h e n a4
       h e n5a t

.h0y3p0h0e2n5a4t2i0o0n.
 h y-p h e n-a t i o n
```

- Patterns hold context of hyphenation point occurrence
- Pattern recognition—a kind of intelligence (magic?)

Pattern generating process

- We want *minimal* set of patterns
- Generating completely covering minimal patterns is NP-complete
- Good iterative methods for "near optimal" solution
- One way to do it:
  - dic-tio-nary with marked hy-phen points
  - we repeat going through the dictionary making *levels*
    - ⋆ *odd*—covering
    - ⋆ *even*—inhibiting

- *candidate choosing rule*—we take subwords (k-character subword)
  - ⋆ and count the number of their good and bad work
  - ⋆ `hen(35,4)a`
- not all the candidates are good
  - ⋆ *pattern choosing rule*—linear function over the number of good and bad work of the pattern compared to a threshold: $\text{good\_count} * \text{good\_wt} - \text{bad\_count} * \text{bad\_wt} \geq \text{threshold}$
- we put good candidates into pattern set with current level number
  - ⋆ `hen5a`
- they still make errors, next level will correct errors of patterns selected so far
- exception list at the end of the whole process, to correct remaining errors

How good the generation process is

- Input data $\approx$ MB
  - patterns $\approx$ tens of KB
  - covering > 98 %
  - with error < 0.1 %
  - usual number of levels ≤ 5
- Results depend on parameter setting
- Nobody knows how to set the parameters

Generating with PATGEN

- PATGEN
  - Frank Liang, 1982, 7-bit ASCII
  - later modified for accents, 8-bit ASCII,... (Peter Breitenlohner, Yannis Haralambous, Karl Berry,...)
- Patterns for tens of languages exist
  - from a dictionary
  - first levels by hand

Why PATGEN **does not suit us today**

- Monolithic structured code, difficult to maintain
- Only ASCII, approx. 240 symbols
- Ω handles UNICODE
- Static data structures
- Difficult to use for purposes other than hyphenation

PATLIB **and** OPATGEN

- PATtern manipulating LIBrary—from scratch
- O (Ω?) PATtern GENerator—handles UTF-8 UNICODE encoding
- Generalisation of PATGEN
- Implementation: C++ in CWEB
- http://www.fi.muni.cz/~xantos/patlib
- GPL

PATLIB **architecture**

- PATLIB
  - Finite language store (pattern manipulator) implemented using packed trie (see the Proceedings)
    - ⋆ insert pattern
    - ⋆ delete pattern
    - ⋆ . . . low-level operations
  - Generator
    - ⋆ handles creating patterns using the same strategy as PATGEN
    - ⋆ easy to change
- OPATGEN is in fact "only" the word list input/output interface

Pros and cons

- What do we get
  - implementation using logical operations
  - any strategy of generating is easier to implement
  - generality and flexibility
  - user manual :-)
- What do we pay for it
  - performance loss

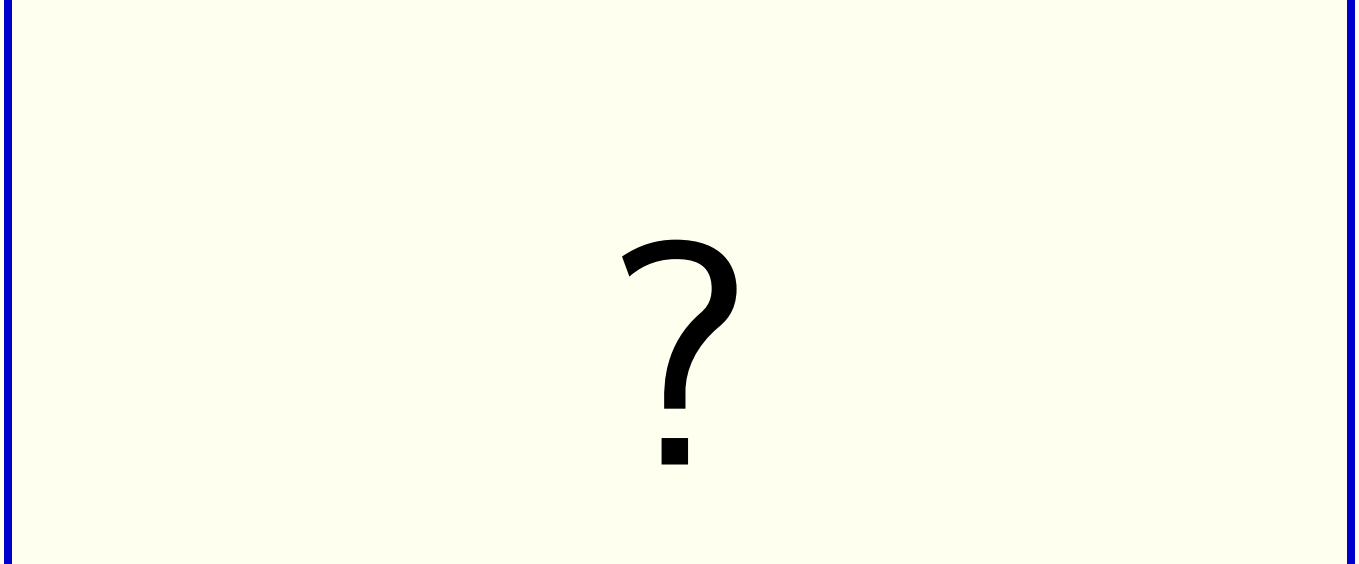
Generalisation, abstraction

- Hyphenating point → *point of interest*
- Patterns *capture* information related to points of interest
- Patterns *compress* information on points of interest
- General, not tight to hyphenation
- Using of patterns is effective, information retrieval is *linear* wrt. the "word length"
- May be implemented as Pattern Translation Process (PTP)
- Some can be done as ΩTP in Omega

Examples of usage

- Hyphenation of compound words
  - we want to prefer breaking on word boundaries
  - needed for German (Wort-silben-trennung)
- Context dependent ligatures
  - ligatures over compound word boundaries are wrong
  - English: shelfful vs. shelfful
  - Czech: šéflékař vs. šéflékař (doctor in chief)
- Fraktur long s versus short s
  - morphology dependent

- End of sentence
  - different space width
  - \@ and \␣ in LaTeX (sorry, the slides are in ConTeXt)
- Thai segmentation
  - word/sentence boundaries are not present in the input Thai transcription, needed for line-breaking
- Arabic letter hamza
  - five hamza variants, depending on context
- Adding Greek accents
- Adding Czech accents to 7-bit ASCII texts
  - e-mail,...



Performance loss

- 10–15 times to PATGEN
- Caused by
  - breaking the program into logical layers
  - using dynamic memory
  - alphabet symbol is an object, not mere number
  - not (yet, if ever) optimised