Large Scale Keyword Extraction using a Finite State Backend

Miloš Jakubíček[1,2] Pavel Šmerk[1]

[1]Natural Language Processing Centre
Faculty of Informatics, Masaryk University

[2]Lexical Computing
Brighton, United Kingdom and Brno, Czech Republic

3. 12. 2016

FSA3

- original motivation some years ago: create a simple morphological analyser based on Jan Daciuk's algorithms and tools
- the core algorithm builds a minimal DFSA from a sorted set of strings
  - FSA is minimal during the whole incremental construction
  - very compact representation of the set
- smart idea, rather poor author's implementation
  - complicated code, ineffective implementation $\Rightarrow$ reimplementation
- proved to be useful also for general wordlists
  - minimal perfect hashing, i. e. bijection words $\leftrightarrow \{1 .. |words|\}$
  - the algorithm is linear, the original implementation is not
  - not obvious until run on really big data

Example: Data for Morphological Analysis

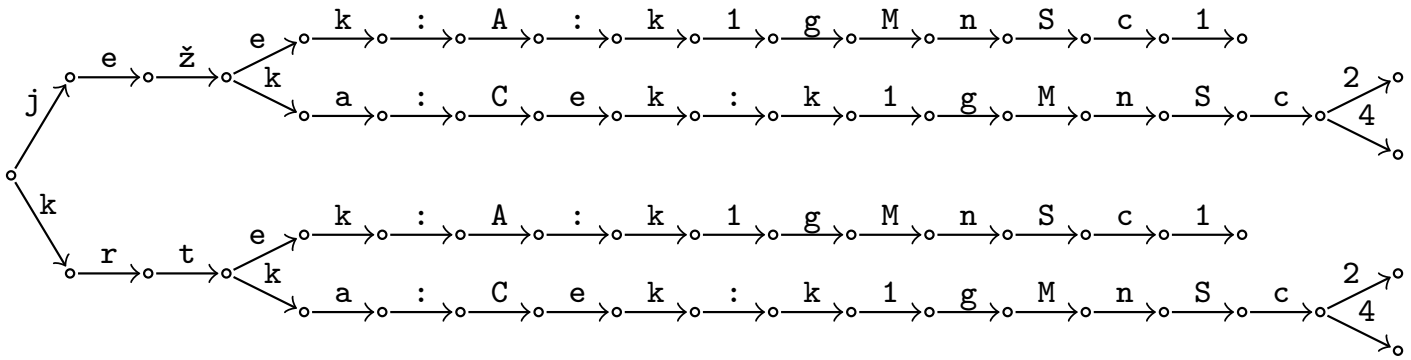
- list of strings `query:answer`
- pairs of words are encoded as the first word and the difference

`ježek:A:k1gMnSc1`	←	`ježek:ježek:k1gMnSc1`
`ježka:Cek:k1gMnSc2`	←	`ježka:ježek:k1gMnSc2`
`ježka:Cek:k1gMnSc4`	←	`ježka:ježek:k1gMnSc4`
`krtek:A:k1gMnSc1`	←	`krtek:krtek:k1gMnSc1`
`krtka:Cek:k1gMnSc2`	←	`krtka:krtek:k1gMnSc2`
`krtka:Cek:k1gMnSc4`	←	`krtka:krtek:k1gMnSc4`

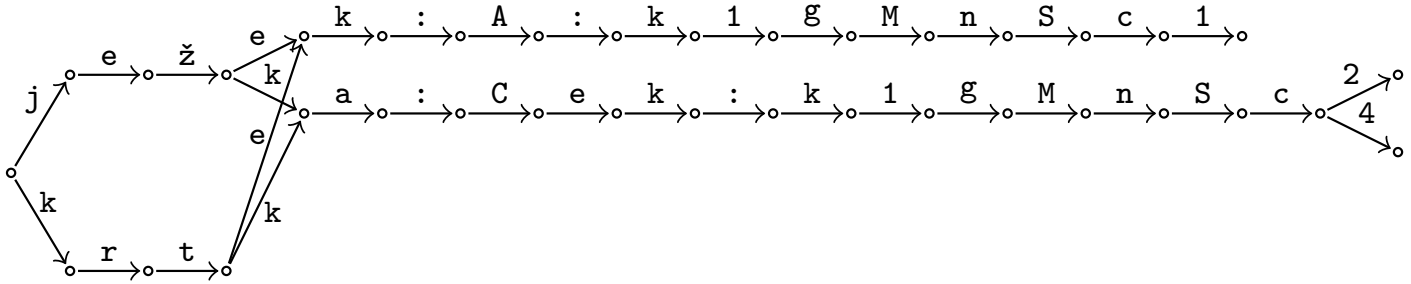
- raw list is too big, but due to its redundancy, the resulting FSA is small

Example: Data for Morphological Analysis

- deterministic FSA



- deterministic FSA after minimization



- "analysis" is only fast and simple pass through this FSA
  - deterministic pass through the FSA according to the "query"
  - and recursive retrieval of all possible "answers"

Minimal Perfect Hashing and General Wordlists

- each word has a unique path in FSA
  - $\Rightarrow$ numbering these paths makes MPH
  - each node is assigned the size of its right language
- general wordlists (corpora etc.) are much less redundant
  - unlike morphology data with full paradigms
  - the compression is much worse, but still roughly comparable to *zip
    - some enTenTen index, 4 GB, 160 M lines
    - gzip -1 — 1 min, 1 GB
    - xz -6 — 1 hour, 0.66 GB
    - xz -0 — 4 min, 0.84 GB
    - mkfsa — 5 min, unoptimized & with numbers 2 GB, 1 GB achievable
      - processes >half a milion words per second
      - recent improvements: linear implementation, smart hashing
      - e. g. the biggest RASLAN13 example is now built 14$\times$ faster
  - unlike *zip, fsa allows "random access" to the compressed data
    - and perfect hashing and RE evaluation and ...

Future Optimizations

- it's a work in progress, many possibilities to optimize the process
- main problem: long chains of nodes with one parent and one child
  - and unique to some weird strings
- we want to reduce or keep reduced:
  - compile space and time, run space and time, code size & generality
- compile space: fsa can be built directly in memory
- compile space and time: avoid hashing nodes in chains
- run space: variable length encoded information, relative adresses, ...
  - chains of nodes as raw strings
- run time: smaller run space
- possible improvements
  - regular expressions
  - mapping words to any numbers