Finite-State Technology

Torbjörn Lager,
Uppsala University

Part 3

In this talk:
- More about replacement
- Filtering
- Mark-up
- Reverse & invert
- Various equivalences in the regular expression calculus
- Don't be a regex hacker!
- Normalization and tokenization

More about replacement

- Examples
  - a b . x . x
  - a b | b c -> x
  - a b -> x . o . b c -> x
  - [a -> x] | [b -> y]
  - [] -> x
  - a* -> x
  - [...] -> x
  - [] .@ -> x
  - [...] -> START%^ || .#. _

Filtering

- Examples
  - a+ -> a
  - a+ @ -> a
  - f a n -> []
  - f a n -> [] || " " _ " 
  - ?]:[[]]* f e s t ?]:[[]]*

Mark-up

- Examples
  - s k i t -> "<CENSUR>" ... "</CENSUR>"
  - define UGLY [s k i t | f a n | n e d r a n s];
  - UGLY -> "<CENSUR>" ... "</CENSUR>"

Reverse & Invert

- Reverse:.r
- Invert:.i

Note:
- Inversion of a regular language has no effect

Example:
- xfs[0]: define AB a b;
- xfs[0]: define BA A B r;
- xfs[0]: print word BA BA;
- xfs[0]: print word AB BA
- xfs[0]: define AB [a . x . b];
- xfs[0]: print word BA BA;
- xfs[0]: define BA AB i;
- xfs[0]: print word BA BA
Various equivalences

- $A = A[]$
- $A+ = A A^*$
- $A+ = A^* - []$
- $A - B = A \& \sim B$
- $\sim A = ?^* - A$
- $\sim [A | B] = \sim A \& \sim B$
- $\sim [A \& B] = \sim A | \sim B$

Various equivalences

- $A - A = \sim[?^*]$
- $A [\sim[?^*]] = A$
- $A [\sim[?^*]] = \sim[?^*]$
- $A \& ?^* = A$
- $A | ?^* = ?^*$

Various equivalences

- $A \Rightarrow B - C = \sim[[\sim[?^*] B] [A ?^*] | ?^* [A \sim[C ?^*]]]]$
- $A \Rightarrow B = \sim[[A - []]] [A \cdot B] ?^* \sim[A - []]]$
- $A \Leftarrow B = (A \Rightarrow B).i$

Various equivalences

- Contextual replacement

Don't be a regex hacker!

- Think of languages and relations!
- Don't think procedurally, think declaratively, not in terms of rewriting, but in terms of relations.
- But don't be naive either! Think quantitatively too, in terms of number of states, arcs, etc.

An (interesting?) analogy

- Regex programming
  - Regular expressions
  - Finite-state transducers

- Logic programming
  - Horn-clause logic formulae
  - Herbrand models
  - Turing machines
Text normalization

- Cleaning up
  - "-" → ""
  - "#" → [] [] _
  - "#" "\n" → []

- Note that apply down/up may not handle newlines the way you expect!
  - "#" → [] [] _ #.

Text tokenization

- Two kinds of tokenization
  - sentence splitting
  - word tokenization

- Problems of tokenization
  - multi-word tokens (e.g. "in spite of")
  - merged words (e.g. "dunno" = "do not know")
  - compounds (e.g. "rainbow", mark-up, ...)
  - token ambiguity

Running XFST with scripts

- In a file myscript.script:
  
  read regex a → x ;
  apply down < myinput.txt

- From the UNIX prompter:
  
  unix% xfst -q -f myscript.script > myoutput.txt