Optimization of Regular Expression Evaluation within the Manatee Corpus Management System

Miloš Jakubíček, Pavel Rychlý

Lexical Computing Ltd.
Brighton, United Kingdom
{milos.jakubicek,pavel.rychly}
@sketchengine.co.uk

NLP Centre, Masaryk University,
Brno, Czech Republic
{jak,pary}
@fi.muni.cz

RASLAN 2014, 5. 12. 2014

Outline

Motivation

- evaluating user queries with regular expression requires matching the regular expression against whole lexicon for the related corpus attribute
- $\Rightarrow$ it is slow for large corpora
- $\Rightarrow$ it blocks retrieving partial results

Optimizations

So far:

- **any string optimization:** `^(\.\*)+$`
- **prefix optimization:** `re.*` or `mis.*ing`

New:

- **n-gram prefetching**

N-gram prefetching

Basic idea:

- linguistically motivated regular expressions almost always (~99%) contain some fixed-string parts
- $\Rightarrow$ we can pre-index the fixed-strings and use them to prune the lexicon space on matching

N-gram pre-indexing I

- fixed-strings = character uni-, bi- and trigrams
- preindexed as a "dynamic" attribute

```
classical  ->  c|l|a|s|s|i|c|a|l
               cl|la|as|ss|si|ic|ca|al
               cla|las|ass|ssi|sic|ica|cal
```

Figure: Uni-, bi- and trigram string generation.

N-gram pre-indexing II

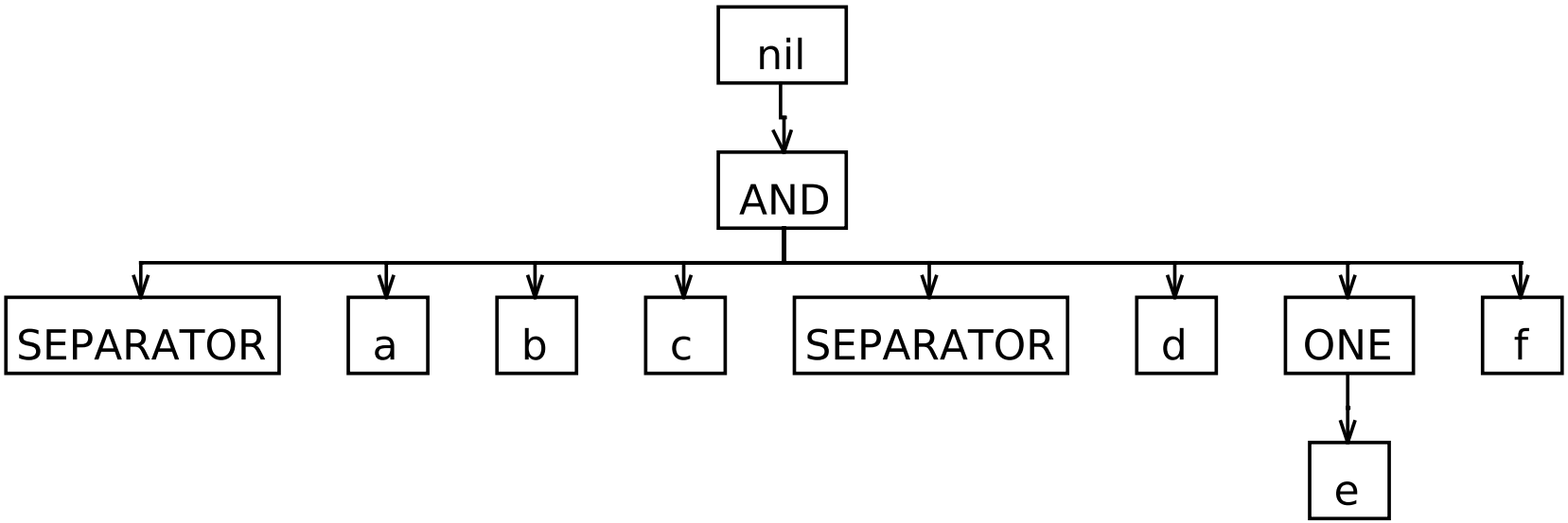
Comparison of original and n-gram lexicon sizes

corpus	language	size $\times 10^9$	attribute	lexicon size	n-gram lexicon size
czTenTen12	Czech	5.126	word	18,978,703	522,745
			lemma	14,151,454	506,580
			tag	12,061	1,702
enTenTen12	English	12.968	word	27,894,538	1,880,911
			lemma	26,426,200	1,880,808
			tag	60	260
enClueWeb09		82.581	word	115,820,931	2,350,697
			lemma	110,606,268	2,296,072
			tag	60	260
jpTenTen11	Japanese	10.322	word	13,844,200	6,353,186
			lemma	13,303,479	3,766,160
			tag	53	297

N-gram matching I

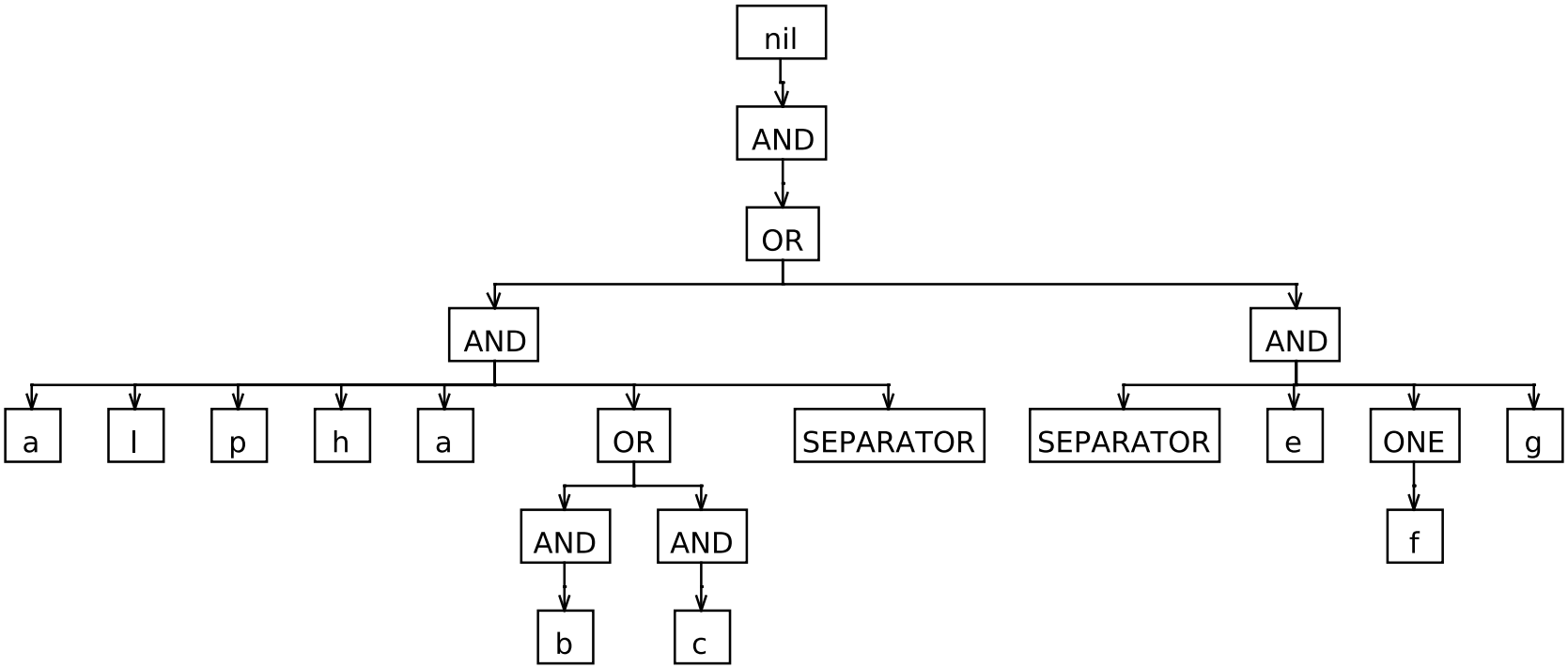
- analyse regular expression using (parsing using an ANTLR3 grammar)
- find obligatory character n-grams
- evaluate regular expression only against lexicon strings containing all obligatory character n-grams

N-gram matching II



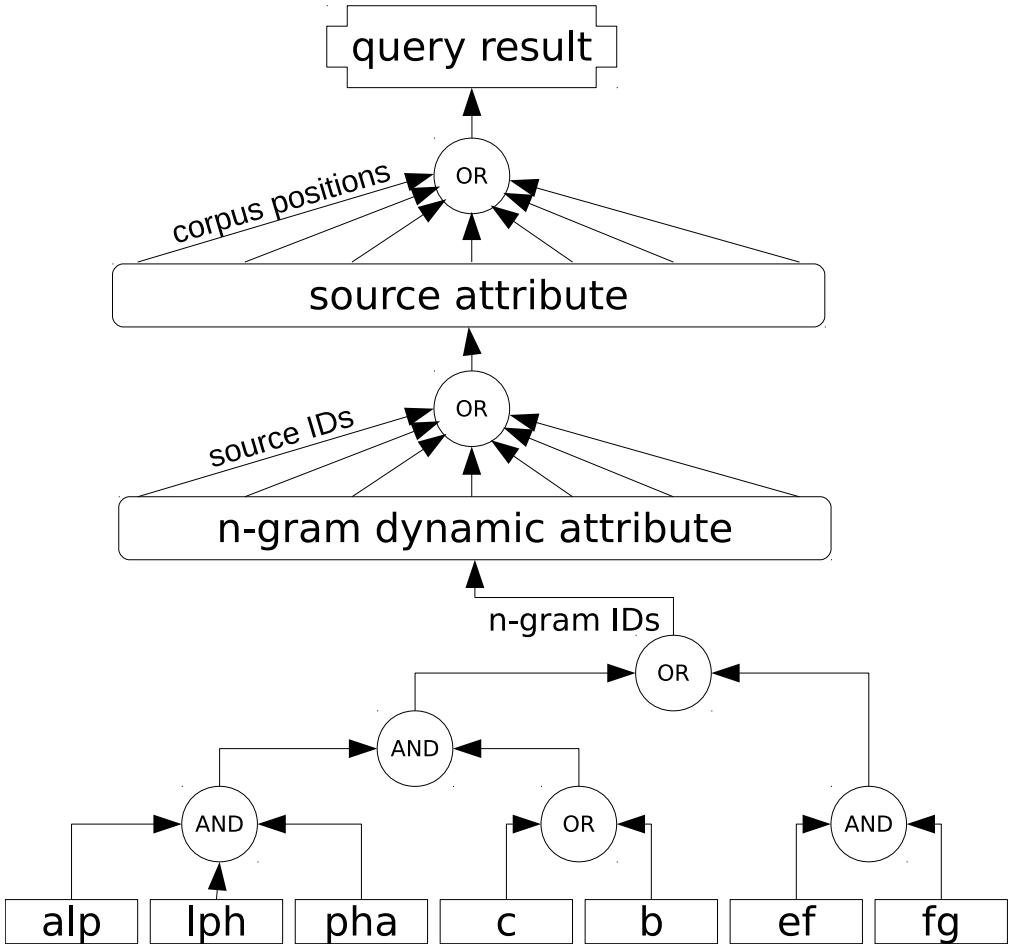
Sample abstract syntax tree for the input query `.abc.*de+f`

N-gram matching III



Sample abstract syntax tree for the input query `(alpha(b|c).*|d*e+f)`

N-gram matching IV



Overview of the evaluation workflow using the n-gram prefetching optimization for input query `(alpha(b|c).*|d*e+f)`

Evaluation

corpus	query	#RE w/o	#RE w/	time w/o	time w/	S
czTenTen12	[word=".*ější"]	18,978,703	41,426	10.030	0.341	29.4
	[lemma=".*strč.*"]	14,151,454	888	6.601	0.066	100.0
	[tag="k1.*c4.*"]	1,357	251	0.058	0.049	1.2
	[word="[sz]p.*"]	18,978,703	115,347	10.023	0.698	14.4
enTenTen12	[word=".*ing"]	27,894,538	913,004	21.931	2.768	7.9
	[lemma=".*ten.*"]	26,426,200	195,758	22.163	1.218	18.2
	[word="pre.*ed"]	80,054	7,553	0.294	0.178	1.7
	[word="pr[oe].*"]	251,924	198,297	1.329	0.920	1.4
	[word=".*[dt]"]	27,894,538	3,466,379	41.100	8.538	4.8
	[tag="N.*"]	60	5	0.056	0.048	1.2
jpTenTen11	[word=".*ち.*"]	13,844,200	30,160	8.182	0.364	22.4
	[lemma=".*ア.*ス"]	13,303,479	69,228	8.388	0.450	18.6
	[word="ンテ.*"]	17,078	17,077	0.199	0.178	1.12

Technical notes

- to build the n-gram index, use:

  `mkregexattr <CORPUS> <ATTRIBUTE>`

- part of Manatee version 2.111, automatically called by `encodevert` for attributes with lexicon exceeding 10,000 items

Conclusions

- n-gram prefetching represents a suitable optimization approach
- speedup usually by one order of magnitude
- method is independent of particular regular expression matching library (PCRE, ICU, re2, . . .)
- further work might be fine-tuning these libraries