Kleenex: From nondeterministic finite state transducers to streaming string transducers

Fritz Henglein

DIKU, University of Copenhagen

2015-05-28

WG 2.8 meeting, Kefalonia

Joint work with Bjørn Bugge Grathwohl, Ulrik Terp Rasmussen, Kristoffer Aalund Søholm and Sebastian Paaske Tørholm (DIKU)

Streaming regular expression processing

Input:

- Regular expression (maybe annotated)
- Stream of characters

Output:

- Parse tree
- Parse tree, but with parts left out (includes subgroup matching)
- Parse tree, but with parts substituted

Examples:

- Web-UI data (issuu.com, JSON, 10 TB/month)
- DNA (UCPH Department of Biology, text, 1 PB stored)
- High-frequency trading (X, Y, continuous)

Think Perl regex processing.

Challenges

- Grammatical ambiguity: Which parse tree to return?
- How to represent parse trees compactly?
- Time: Straightforward backtracking algorithm, but impractical: $\Theta(m\,2^n)$ time, where $m = |E|$, $n = |s|$.
- Space: How to minimize RAM consumption? How to stream?

Regular Expressions as Types

- Regular Expressions (RE):

$$E ::= 0 \mid 1 \mid a \mid E_1 E_2 \mid E_1|E_2 \mid E_1^* \qquad (a \in \Sigma)$$

- Type interpretation $\mathcal{T}[\![E]\!]$:

$$\begin{array}{rcl cl}
\mathcal{T}[\![0]\!] & = & 0 & = & \emptyset \\
\mathcal{T}[\![1]\!] & = & 1 & = & \{()\} \\
\mathcal{T}[\![a]\!] & = & \{a\} & = & \{a\} \\
\mathcal{T}[\![E_1 E_2]\!] & = & E_1 \times E_2 & = & \{(V_1, V_2) \mid V_1 \in \mathcal{T}[\![E_1]\!], V_2 \in \mathcal{T}[\![E_2]\!]\} \\
\mathcal{T}[\![E_1|E_2]\!] & = & E_1 + E_2 & = & \{\mathsf{inl}\ V_1 \mid V_1 \in \mathcal{T}[\![E_1]\!]\} \cup \{\mathsf{inr}\ V_2 \mid V_2 \in \mathcal{T}[\![E_2]\!]\} \\
\mathcal{T}[\![E^*]\!] & = & E\ \mathrm{list} & = & \{[V_1, \ldots, V_n] \mid n \geqslant 0 \wedge \forall 1 \leqslant i \leqslant n.\, V_i \in \mathcal{T}[\![E]\!]\}
\end{array}$$

- Not the language interpretation $\mathcal{L}[\![E]\!]$!
- "Value" = Element of type = parse tree = proof of inhabitation
- Frisch, Cardelli (2004). Henglein, Nielsen (2011)

Bit-Coding: Serialized parse trees

- Prefix code for parse trees.
- Encoding $\ulcorner \cdot \urcorner : \mathcal{V} \rightarrow \{1, 0\}^*$,

$$
\begin{array}{rcl}
\ulcorner () \urcorner & = & \epsilon \\
\ulcorner a \urcorner & = & \epsilon \\
\ulcorner (V_1, V_2) \urcorner & = & \ulcorner V_1 \urcorner \ulcorner V_2 \urcorner \\
\ulcorner \mathsf{inl}\ (V_1) \urcorner & = & 0 \ulcorner V_1 \urcorner \\
\ulcorner \mathsf{inr}\ (V_2) \urcorner & = & 1 \ulcorner V_2 \urcorner \\
\ulcorner [V_1, \ldots, V_n] \urcorner & = & 0 \ulcorner V_1 \urcorner \cdots 0 \ulcorner V_n \urcorner\ 1
\end{array}
$$

- Type-indexed decoding $\llcorner \cdot \lrcorner_E : \{1, 0\}^* \rightharpoonup \mathcal{T}[\![E]\!]$: Interpret RE as nondeterministic algorithm to construct parse tree, with bit-code as oracle.
- C.f. Vytinionitis, Kennedy, *Every bit counts* (2010).

Example

RE $= ((a|b)(c|d))^*$. Input string $= acbd$.

1. Acceptance testing: Yes!
2. Pattern matching: $(0,4), (2,4), (2,3), (3,4)$
3. Parsing: $[(\text{inl } a, \text{inl } c), (\text{inr } b, \text{inr } d)]$
    - Bit-code: 0 00 0 11 1.

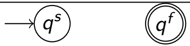
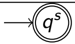
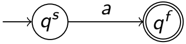
Bit-coding: Examples

- Bit codes for the string `abcbcba`

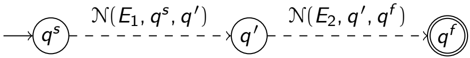
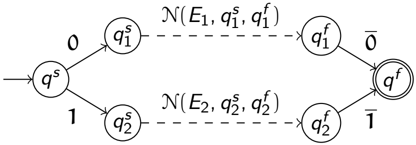
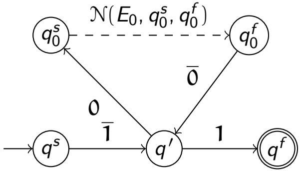
Regular expression	Representation	Size
Latin1	abcbcba00000000	64
Σ^*	0a0b0c0b0c0b0a1	64
$((a + b) + (c + d))^*$	0000010100010100010001	22
$a \times b \times c \times b \times c \times b \times a$		0

Augmented Thompson NFAs

- Thompson NFA with output labels on split- and join-nodes.
- Construction:

E	$\mathcal{N}(E, q^s, q^f)$
0	
1	 (implies $q^s = q^f$)
a	

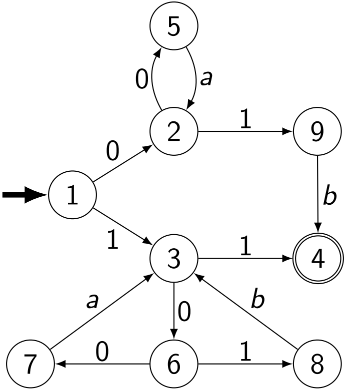
Augmented Thompson NFAs

E	$\mathcal{N}(E, q^s, q^f)$
$E_1 E_2$	
$E_1 \| E_2$	
E_0^*	

Simplification: $\overline{0}$- and $\overline{1}$-labeled edges contracted.

Augmented Thompson NFA: Example

Augmented Thompson NFA for $a^* b | (a|b)^*$



Representation Theorem

Theorem

One-to-one correspondence between

- *parse trees for E,*
- *paths in augmented Thompson automaton for E,*
- *bit-coded parse trees = bit subsequences of automaton paths.*

Lexicographically least bit-code = greedy parse.

- Important to use Thompson-style ϵ-NFAs. Does not hold for DFAs, ϵ-free NFAs.
- Grathwohl, Henglein, Rasmussen (2013). Already observed by Brüggemann-Klein (1993).

Optimal streaming

- Assume partial $f : \Sigma^* \hookrightarrow \Delta^*$.
  - Example: Bit-coded greedy parse of input sequence
- *Optimally streaming version* of f:

$$f^{\#}(s) = \bigsqcap \{ f(ss') \mid ss' \in \mathrm{dom} f \}$$

  where $\bigsqcap$ = longest common prefix.
- Outputs bits as soon as those are semantically determined by the prefix seen so far.

Regular matching algorithms

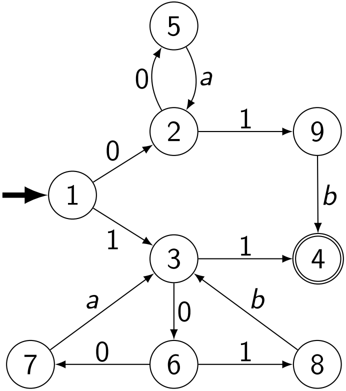
Problem	Time	Space	Aux	Answer
NFA simulation	$O(mn)$	$O(m)$	0	0/1
Perl	$O(m2^n)$	$O(m)$	0	k groups
RE2[1]	$O(mn)$	$O(m+n)$	0	k groups
Parse (3-p)[2]	$O(mn)$	$O(m)$	$O(n)$	greedy parse
Parse (2-p)[3]	$O(mn)$	$O(m)$	$O(n)$	greedy parse
Parse (str.)[4]	$O(mn+2^{m \log m}))$	$O(m)$	$O(n)$	greedy parse

(n size of input, m size of RE)

[1] Cox (2007)
[2] Frisch, Cardelli (2004)
[3] Grathwohl, Henglein, Nielsen, Rasmussen (2013)
[4] Optimally streaming. Grathwohl, Henglein, Rasmussen (2014)

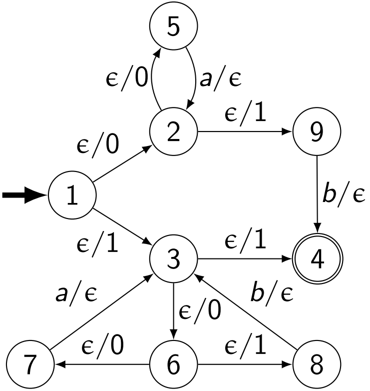
Augmented Thompson NFA: Example

Augmented Thompson NFA for $a^* b|(a|b)^*$



Augmented Thompson NFA as NFST

Augmented Thompson NFA for $a^*b|(a|b)^*$



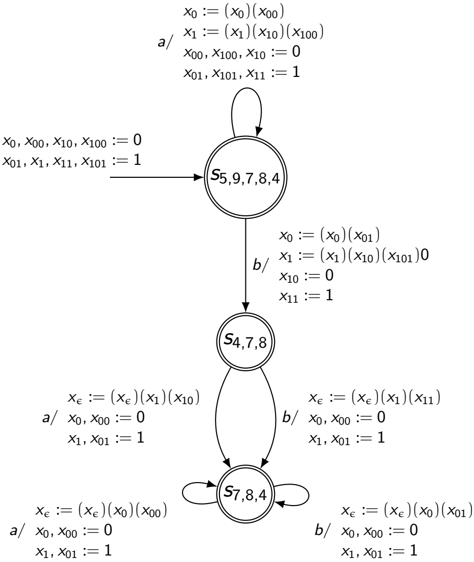
Generalizations

- Techniques work for *arbitrary* NFSTs:
  - arbitrary outputs (and output actions), not just ϵ and individual bits;
  - intuitively fusion of parsing with subsequent catamorphism.
- NFSTs (with ϵ-transitions) are more compact than RE.
  - DFA as RE: $\Omega(m^2)$ blow-up.
  - NFA as ϵ-free NFA (matrix representation): $\Omega(m \log m)$ blow-up; standard construction (Glushkov): $\Theta(m^2)$ blow-up.
  - NFSTs correspond to left-linear grammars with output actions.
  - Kleenex: Surface language for linear grammars with output actions.

Determinization: Streaming string transformers

- Streaming string transducer:
  - deterministic finite automata,
  - each state equipped with fixed number of registers containing strings
  - registers updated on transititon by affine function;
  - Alur, D'Antoni, Raghothaman (2015).
- Determinization:
  - Finite number of possible path trees during NFST-simulation
  - Edges in a path tree $\cong$ registers

Determinization: Example



Implementation

- Compilation of Kleenex to streaming string transformer in Haskell;
- generates C code (goto-form), linked with string concatenation library.
- Optimizations: Lookahead processing, symbolic transitions, register constant progagation.

Performance evaluation

- Comparison RE2, RE2J, Oniglib, Ragel, awk, sed, grep, Perl, Python, specialized tools.
- Standard desktop
- Single-core Kleenex:
  - High throughput even for complex specifications
  - Typically around 1 Gb/s, for simple specifications more (6 Gb/s)

Performance test: Issuu simple

```
({("[a-z_]*":(-?[0-9]*|"(([^"]|\\")*)"),?)*}\n?)*
```

Performance test: Issuu

```
({("(((((ts|visitor_username)|(visitor_uuid|
visitor_source))|((visitor_useragent|visitor_referrer)
|(visitor_country|visitor_device)))
|(((visitor_ip|env_type)|(env_doc_id|env_adid))
|((env_ranking|env_build)|(env_name|env_component))))
|((((event_type|event_service)|(event_readtime
|event_index))|((subject_type|subject_doc_id)
|(subject_page|subject_infoboxid)))|(((subject_url
|subject_link_position)|(cause_type|cause_position))
|((cause_adid|cause_embedid)|(cause_token|cause)))))"
:(-?[0-9]*|"(((((internal|external)|([A-Z][A-Z]|(browser
|android)))|(([0-9a-f]{16}|reader)|(stream|(website
|impression))))|(((click|read)|(download|(share
|pageread)))|((pagereadtime|(continuation_load|doc))
|(infobox|(link|page)))))|((((ad|related)|(archive
|(embed|email)))|((facebook|(twitter|google))|(tumblr
|(linkedin|[0-9]{12}-[a-z0-9]{32}))))|(((Mozilla/
|Windows NT)|(WOW64|(Linux|Android)))|((Mobile
|(AppleWebKit/|(KHTML, like Gecko)))|(Chrome/|(Safari/
|([^"]|\\")*)))))))"),?)*}\n?)*
```

Towards 5 Gbps/core

- Multistriding with tabling (8 bytes at a time)
- Transducer optimizations (shrinking)
- Hardware- and systems-specific optimizations

Future work

- Parallel RE processing
  - Mytkowicz et al. (ASPLOS 2014, PPoPP 2014, POPL 2015)
- Optimally streaming substitution and aggregation
- Probabilistic matching
- ...
- Characterization of 1NFSTs
- Visibly PDAs/nested word automata
- ...
- Applications (bioinformatics, finance, weblogs, ...)

Summary

- Regular expressions as types
  - Grammars as types
- Bitcoding
- Augmented Thompson NFAs
- Characterization: (lex. least) path = (greedy) parse tree
- Optimal streaming
- (Augmented Thompson NFA simulation)
- Determinization: Streaming string transformers
- ...to get raw speed.

More information: `www.diku.dk/kmc`.