A process-oriented implementation of Brzozowski's DFA construction algorithm

Tinus Strauss[1] Derrick G. Kourie[2] Bruce W. Watson[2]
Loek Cleophas[1]

[1]FASTAR Research group
University of Pretoria
South Africa

[2]FASTAR Research group
Stellenbosch University
South Africa

The Prague Stringology Conference 2014

Sequential algorithm

$\delta, S, F := \varnothing, \{E\}, \varnothing;$
$D, T := \varnothing, S;$
do $(T \neq \varnothing) \rightarrow$
  let q be some state such that $q \in T$;
  $D, T := D \cup \{q\}, T \setminus \{q\};$
  { *build out-transitions from q on all alphabet symbols* }
  for $(i : \Sigma) \rightarrow$
    { *find derivative of q with respect to i* }
    $d := i^{-1}q;$
    if $d \notin (D \cup T) \rightarrow T := T \cup \{d\}$
    $[\!]$ $d \in (D \cup T) \rightarrow$ **skip**
    fi;
    { *make a transition from q to d on i* }
    $\delta(q, i) := d$
  rof;
  if $\varepsilon \in \mathcal{L}(q) \rightarrow F := F \cup \{q\}$
  $[\!]$ $\varepsilon \notin \mathcal{L}(q) \rightarrow$ **skip**
  fi
od; **return** $(D, \Sigma, \delta, S, F)$

Selected CSP notation

$a \to P$	event *a* then process *Q*
$a \to P \mid b \to Q$	*a* then *P* choice *b* then *Q*
$x : A \to P(x)$	choice of *x* from set *A* then $P(x)$
$P \parallel Q$	*P* in parallel with *Q*
	Synchronize on common events in alphabets
$b!e$	on channel *b* output event *e*
$b?x$	from channel *b* input to variable *x*
$P \not\leftarrow C \not\rightarrow Q$	if *C* then process *P* else process *Q*
$P; Q$	process *P* followed by process *Q*
$P \, \Box \, Q$	process *P* choice process *Q*

The BRZ process

$BRZ(T, D, F, \delta)$

$$OUTER(T, D, F) \parallel FANOUT \parallel DERIV \parallel UPDATE(\delta)$$

- OUTER corresponds with outer loop.
- DERIV caters for the computation of derivatives.
- UPDATE caters for updating δ.
- FANOUT distributes a regular expression to DERIV subprocesses.

The OUTER process

$OUTER(T, D, F)$

$q : T \rightarrow outNode!q \rightarrow$

$\quad OUTER(T \setminus q, D \cup q, F \cup q) \nless \varepsilon \in \mathcal{L}(q) \ngtr OUTER(T \setminus q, D \cup q, F)$

$\square$

$inNode?d \rightarrow$

$\quad OUTER(T \cup d, D, F) \nless d \notin T \cup D \ngtr OUTER(T, D, F)$

$\square$

$SKIP$

- Some $q \in T$ is selected to build its outgoing transitions.
- A (potentially) new node is received.
- Updating of sets D, T, and F.

The DERIVE process

- Finds the derivatives of a regular expression in parallel.

$DERIV$

$$\|_{i:\Sigma}\ DERIV_i$$

- Each $DERIV_i$ process reads a regular expression and communicates its derivative.

$DERIV_i$

$$dOut_i?re \rightarrow computeDeriv.re \rightarrow derivChan!\langle re, i, i^{-1}re\rangle \rightarrow DERIV_i$$

FANOUT

- Distributes a regular expression to the different $DERIV_i$ processes.

FANOUT

$$(outNode?re \rightarrow \|_{i:\Sigma} (dOut_i!re \rightarrow SKIP)); FANOUT$$

The UPDATE process

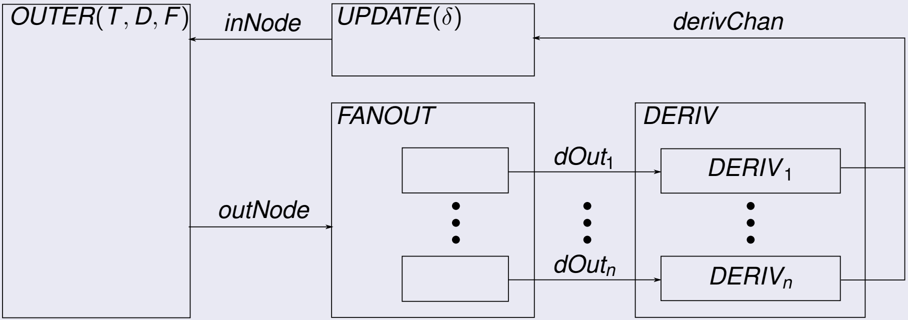
- Receives derivatives and updates δ.
- Communicates the derivative back to OUTER.

$\mathit{UPDATE}(\delta)$

$$\mathit{derivChan}?\langle re, i, d\rangle \rightarrow \mathit{inNode}!d \rightarrow \mathit{UPDATE}(\delta \cup \langle re, i, d\rangle)$$

Graphical representation

$BRZ(T, D, F, \delta)$



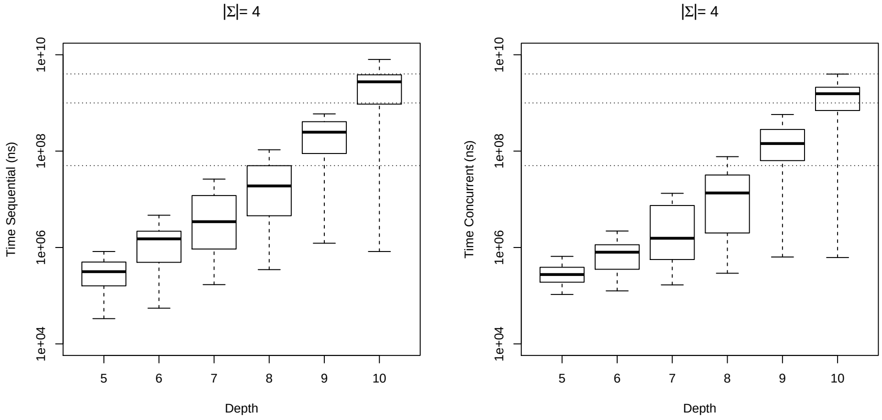
Implementation

- Used Go programming language.
- `golang.org`
- Go's concurrency model resembles CSP.
- Processes implemented as go-routines.
- Language supports channels.
- Synchronisation via channels.

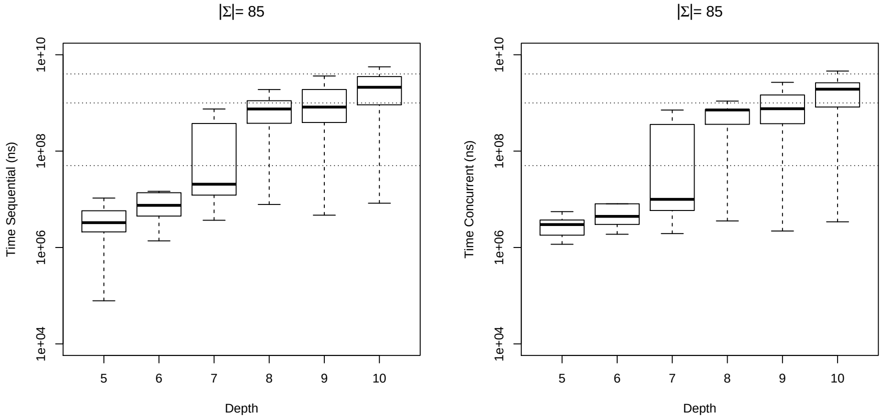
Experiments

- Random regular expressions of various sizes (depths).
- Two alphabet sizes: 4 and 85 symbols.
- Go version 1.2.2
- Machine
  - 2x dual-core Xeon 2.66 GHz
  - 5 GB RAM
  - Mac OS X 10.7.5

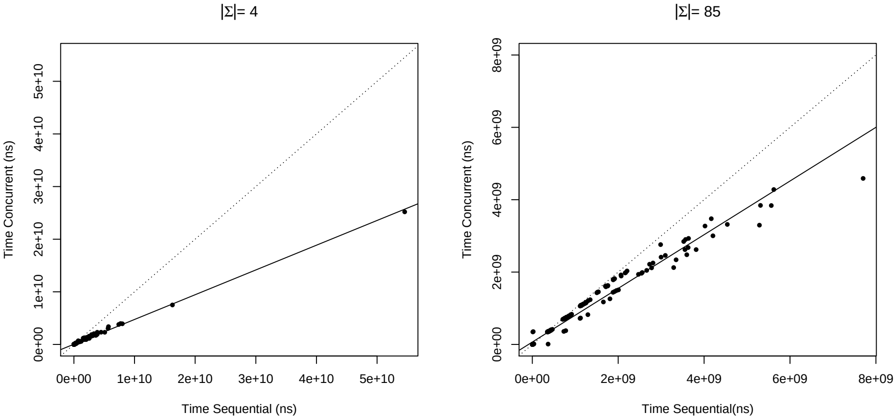
Construction times for $|\Sigma| = 4$



Construction times for $|\Sigma| = 85$



Scatterplots of paired construction times



$$T_c = 39.6\,\text{ms} + 0.47 \cdot T_s \qquad \text{for } |\Sigma| = 4$$

$$T_c = 65.7\,\text{ms} + 0.74 \cdot T_s \qquad \text{for } |\Sigma| = 85$$

Speedup and efficiency

	Speedup		Efficiency	
Depth	$\|\Sigma\| = 4$	$\|\Sigma\| = 85$	$\|\Sigma\| = 4$	$\|\Sigma\| = 85$
All	1.72	1.09	0.43	0.27
5	1.15	1.21	0.29	0.30
6	1.84	1.45	0.46	0.36
7	1.82	1.43	0.46	0.36
8	1.80	1.06	0.45	0.27
9	1.71	1.09	0.43	0.27
10	1.83	1.21	0.46	0.30

$$\text{Speedup} = \frac{T_s}{T_c} \qquad \text{Efficiency} = \frac{\text{Speedup}}{\#[\text{processors}]}$$

Conclusion

- Presented a process-oriented decomposition of the construction algorithm.
- Presented the results of an experiment.
- Obtained speedup
- Efficiency low.

- Next steps
  - Try to improve efficiency.
  - Other FA algorithms such as minimisation.