# Concurrent FA Algorithms

Tinus Strauss  Derrick G. Kourie  Bruce W. Watson

FASTAR
University of Pretoria
South Africa

Espresso / Workshop 2007

# 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

| | |
|---|---|
| $a \rightarrow P$ | event $a$ then process $Q$ |
| $a \rightarrow P \vert b \rightarrow Q$ | $a$ then $P$ choice $b$ then $Q$ |
| $x : A \rightarrow P(x)$ | 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 $x$ |
| $P \not< C \not> Q$ | if $C$ then process $P$ else process $Q$ |
| $P; Q$ | process $P$ followed by process $Q$ |
| $P \Box Q$ | process $P$ choice process $Q$ |

Universiteit van Pretoria
University of Pretoria

# The BRZ process

$$BRZ(D, T) = OUTER(D, T) \parallel DERIVE \parallel UPDATE$$

- OUTER corresponds with outer loop.
- DERIVE caters for the computation of derivatives.
- UPDATE caters for the determination of which regular expressions should be used to update $T$ and also for updating $\delta$.

# The OUTER process

- Some $q \in T$ is selected to build its outgoing transitions.
- Updating of set $D$, $T$, and $F$.

$$OUTER(D, T) =$$

$$(q : T \rightarrow (EXTRACT(q) \parallel FINAL(q)); OUTER(D \cup \{q\}, T \setminus \{q\}))$$

$$\Box$$

$$(insert?q \rightarrow OUTER(D, T \cup \{q\}))$$

# The EXTRACT and FINAL processes

- Broadcasts the selected $q$ to the DERIVE processes.

$$EXTRACT(q) = \|_{i:\Sigma}\ EXTRACT_i(q) \quad \text{where}$$

$$EXTRACT_i(q) = (dIn_i!q \rightarrow SKIP)$$

- Updates $F$, if needed.

$$FINAL(q) = F := F \cup \{q\} \not< \varepsilon \in \mathcal{L}(q) \not> SKIP$$

# The DERIVE process

- Finds the derivatives of a regular expression in parallel.

$$DERIVE = \|_{i:\Sigma} DERIVE_i \quad \text{where}$$

$$DERIVE_i = dIn_i?q \rightarrow dOut_i!(q, \frac{d}{di}q) \rightarrow DERIVE_i$$

# The UPDATE process

- Receives derivatives and updates $\delta$.
- Feeds back the derivative to OUTER.

$$UPDATE = \|_{i:\Sigma}\ UPDATE_i$$

$$UPDATE_i = (dOut_i?(q,d) \rightarrow (UPT_i(d) \parallel UPD_i(q,d)) ); UPDATE_i$$

$$UPT_i(d) = insert!d \rightarrow SKIP \nleftarrow d \notin (D \cup T) \nrightarrow SKIP$$

$$UPD_i(q,d) = \delta(q,i) := d$$

Universiteit van Pretoria
University of Pretoria

# Minimisation

$S, G, H := \emptyset, ((Q \setminus F) \times F) \cup (F \times (Q \setminus F)), \{ (q, q) \mid q \in Q \};$
$\{$ invariant: $G \subseteq \neg Equiv \wedge H \subseteq Equiv$ $\}$
**do** $(G \cup H) \neq Q \times Q \rightarrow$
  **let** $p, q : (p, q) \in ((Q \times Q) \setminus (G \cup H))$;
  **if** $equiv(p, q, (|Q| - 2)\,\mathbf{max}\,0) \rightarrow$
    $H := H \cup \{(p, q), (q, p)\}$;
    $H := H^+$
  $[\!]$ $\neg equiv(p, q, (|Q| - 2)\,\mathbf{max}\,0) \rightarrow$
    $G := G \cup \{(p, q), (q, p)\}$
  **fi**
**od**; $\{$ $H = Equiv$ $\}$
merge states according to $H$
$\{$ $(Q, \Sigma, \delta, q_0, F)$ is minimal $\}$

# Minimisation

**func** $equiv(p, q, k) \rightarrow$
  **if** $k = 0 \rightarrow eq := (p \in F \equiv q \in F)$
  $[\!]$ $k \neq 0 \wedge \{p, q\} \in S \rightarrow eq := true$
  $[\!]$ $k \neq 0 \wedge \{p, q\} \notin S \rightarrow$
    $eq := (p \in F \equiv q \in F) \wedge (\Sigma_p = \Sigma_q)$;
    $S := S \cup \{\{p, q\}\}$;
    **for** $a : a \in \Sigma_p \cap \Sigma_q \rightarrow$
      $eq := eq \wedge equiv(\delta(p, a), \delta(q, a), k - 1)$
    **rof**;
    $S := S \setminus \{\{p, q\}\}$
  **fi**;
  **return** $eq$
**cnuf**

Universiteit van Pretoria
University of Pretoria

# Specification

$$
\begin{aligned}
&Equiv_{pq}(S,k) = \\
&\qquad \text{if } (p = q) \text{ then } main_{pq}!true \rightarrow SKIP \\
&\qquad \text{else if } (k = 0) \text{ then } main_{pq}!(p \in F \equiv q \in F) \rightarrow SKIP \\
&\qquad \text{else } (EqSet := \emptyset \\
&\qquad ;\ FanOut_{pq}(S,k) \\
&\qquad ;\ (eq := \wedge_{e \in EqSet}\ e) \\
&\qquad ;\ (main_{pq}!eq \rightarrow SKIP))
\end{aligned}
$$

Universiteit van Pretoria
University of Pretoria

# Specification

$$
\begin{aligned}
FanOut_{pq}(S,k) \quad = \quad & |||_{a \in \Sigma_{pq}} \\
& (\text{if } (\{\delta(p,a),\delta(q,a)\} \notin S) \text{ then} \\
& (\ Equiv_{\delta(p,a),\delta(q,a)}(S \cup \{(p,q)\}, k-1)\ \triangle \\
& (subs_{\delta(p,a),\delta(q,a)}?eq_a \rightarrow (EqSet := EqSet \cup \{eq_a \\
& \text{else } (EqSet := EqSet \cup \{true\})\ )
\end{aligned}
$$

Universiteit van Pretoria
University of Pretoria

# Conclusion

- Presented an abstract concurrent specification.
- Attempted to allow for as much concurrency as possible.

- Next steps
  - Implement schemes.
  - Other FA algorithms.