Extending Finite Automata to Efficiently Match Perl-Compatible Regular Expressions

Michela Becchi and Patrick Crowley

ACM CoNEXT 2008
Context

- Network intrusion detection and prevention systems

Matching Engine and RegEx set

Incoming packets

FTP.OPEN.*
www.spyware
Host=
Server.*HTTP

Safe packets

Safe_payload
Safe_payload

Malicious packets

xHost=
ServerABHTTP

- Email monitoring
- Content based routing
- Application level prioritizing and filtering
- ...
Challenges

■ Networking context
  » Line rate operation (several Gbps)
  » Parallel search over data-sets consisting of hundreds or thousands of patterns
     - Bound per-character processing
     - Pre-computed large data structures

» On memory-centric architectures

Memory bandwidth   Memory size
Challenges (cont’d)

- Snort rule-set, November 2007 snapshot
  - 8536 rules
    - 5549 Perl Compatible Regular Expressions
      - 99% with character ranges
      - 16.3% with dot-star terms
      - 44% with counting constraints
      - 6% with back-references
      - Lazy/greedy quantifiers
      - Positive/negative lookahead
      - Atomic groups
      - No expressive power added
      - Speed up text-based engines

• Note:
Deterministic vs. Non-Deterministic FA

RegEx: (1) $a^+bc$ (2) $bcd^+$ (3) $cde$

Text: $a\ b\ c\ d$

**NFA**

**DFA**

MEMORY SIZE: 
# of states and transitions
Better for NFAs

MEMORY BANDWIDTH: 
# of state traversals per input character
Better for DFAs
Counting constraints – NFA

E.g: $a \{n\} bc$

- **Memory size**
  - For large $n$, number of states $N_{NFA}$ linear in $n$

- **Memory bandwidth**
  - Input text: $aaaaaaa...aaabc \Rightarrow n$ states active in parallel
  - For large $n \sim N_{NFA}$ memory accesses/input character
Counting constraints - DFA

E.g: a.$\{n\}$bc

- Memory size
  - For large $n$, number of states $D_{NFA}$ exponential in $n$
  - For large $n$ DFA practically infeasible
    - e.g. $n=40 \rightarrow \sim 1000$ billion states
Counting-NFAs

E.g: a.{n}bc

- Advantage: Limited size *(independent of n)*
- Functional equivalence: is one counter enough?
  - E.g.: a.{3}bc:
    - text: axaybcz ⇒ match is detected
    - text: axaybzc ⇒ match is missed!
  - Multiple (up to n) counter instances necessary

- n active counter instances ⇒ unmodified memory bandwidth requirement!
Counting-NFAs: limiting memory bandwidth

E.g: \texttt{a.\{n\}bc}

- **Observation:**
  - Counter instances updated in parallel
  - Difference between $c_i$ and $c_j$ constant over time

- **Idea:**
  - *Differential representation*: store oldest (and largest) instance $c_i'$ and, for $j>i$, $\Delta c_j = c_j - c_{j-1}$

  \[
  \begin{array}{cccc}
  8 & 5 & 3 & 1 \\
  \end{array}
  \]

  - *Condition evaluation*:
    - $\text{cnt}=n$: $c_i'=n$
    - $\text{cnt}\neq n$: $c_i'\neq n$ OR another instances $c_j$ exists

- **Advantage:**
  - Even if $n$ instances are active, **only 2 must be queried/updated**

\[\begin{array}{|c|c|c|}
\hline
  c' & \Delta c_i \\
  \hline
  8 & 3 & 2 & 2 \\
  9 & 3 & 2 & 2 \\
  7 & - & 2 & 2 \\
  \hline
\end{array}\]
Counting-DFAs

E.g: a.{n}bc

Counting-NFA

- Extended NFA-DFA transformation
  - Counting states
  - Instantiating transitions
  - Conditional transitions

- Possible conditions:
  - cnt=n: \( c_i' = n \) and \( c_i' \) is single instance
  - cnt\( \neq \)n: \( c_i' \neq n \)
  - cnt=\( \perp \): \( c_i' = n \) and another instance \( c_j \) exists

- Consequences:
  - Limited memory bandwidth (1 state + 2 counter instances)
  - Limited size (independent of \( n \))
Combining multiple regex

Patterns = \{RE_1, RE_2, RE_3, ... RE_n\}

1\textsuperscript{st} solution: regex partitioning [Brodie, ISCA’06][Yu, ANCS’06]

\[
\begin{array}{cccccc}
RE_1 & RE_2 & RE_3 & RE_4 & \ldots & RE_{i-1} & RE_i & RE_{i+1} & \ldots & R_{n-1} & R_n \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
DFA_1 & DFA_2 & DFA_3 & DFA_4 & \ldots & DFA_k \\
\end{array}
\]

\(k\) concurrent DFAs \(\Rightarrow\) \(k\) memory accesses/input char

High parallelism and memory bandwidth: ASIC, FPGA
Combining multiple regex (cont’d)

- 2nd solution: hybrid-FA [Becchi, CoNEXT 2007]

  \[ \Sigma [^c_1..c_k] \quad \Sigma [^c_1..c_k] \]

- Memory Size:
  » Limited, independent of # of closures states

- Memory Bandwidth:
  » Average:
    » only head-DFA active
    » one state traversal/character
  » Worst case:
    - All tail-FAs are active
    - Bandwidth = # DFAs state traversal + 2 accesses/counters, per char

Low-Medium parallelism and memory bandwidth: GPP, small CMP
Back-references

- Idea: a given sub-expression must be matched multiple times with the same text

- Examples
  - `(abc|bcd).\1y` matches `abcdabcdy`, does not match `abcdaabcy`
  - `a([a-z]+)a\1y` matches `babacabacy`

- Observations
  - The alternative in the referenced sub-expression may overlap
  - The capture sub-expression may overlap w/ previous/next char
  - The length of the referenced sub-expression may be variable

**GOAL: preserve NFA-like operation:**
- Find all matches/stop at the first
- Process each char once
- Allow parallel RegEx processing
Extended-FA

E.g.: (abc|bcd).\1y

- Recording transitions
- Conditional transitions
- Consuming state

Extensions:

» Recording and conditional transitions, consuming states
» Each state associated with a set \{PM_k\} of partial match strings
Extended-FA operation

E.g.: (abc|bcd).\1y

Text:  a  b  c  e  a  b  c  y
Results

Memory size

Worst case memory bandwidth

Hybrid-FA has memory bandwidth from 10X to 100X lower

<table>
<thead>
<tr>
<th>Data set</th>
<th>NFA # counters</th>
<th>Extended-NFA # backref</th>
<th>Extended-hybrid-FA # backref</th>
</tr>
</thead>
<tbody>
<tr>
<td>snort76</td>
<td>46</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>snort676</td>
<td>37</td>
<td>181</td>
<td>181</td>
</tr>
<tr>
<td>snort702</td>
<td>48</td>
<td>172</td>
<td>172</td>
</tr>
</tbody>
</table>
Conclusion

Extended Finite Automata:
- Dynamic state information: counters, partial matches
- Manipulating states: counting, consuming states
- Producing transitions: counter, partial match instantiation
- Conditional transitions

Goals:
- Limited memory size and bandwidth requirement
- Integration w/ existing proposals
  - Multiple-DFAs
  - Hybrid-FA
  - DFA compression techniques

Future direction:
- Use extensions for structured data parsing (e.g.: XML, application protocols)
Thank you!

- Questions?
Matching architecture
Observations

- Cost of handling partial matches
  - Store \((\text{starting posn, ending posn})\) instead of sub-strings
  - Worst case: \(O(m^2N_{\text{NFA}})\) pointers (\(m\) being length of input text)
  - Control size of \(\{PM_k\}\)
    - Defer NFA-state evaluation if needed (\(\Leftrightarrow\) back-tracking)
    - Trade-off depth-first \(w/\) breath-first traversal

- Comparison \(w/\) counting-FAs
  - Similar mechanisms
    - Dynamic memory usage: counters \(\sim\) partial matches
    - Counting states \(\sim\) consuming states
    - Counter instantiation \(\sim\) recording transitions
    - Conditional transitions
  - DFA-transformation and integration \(w/\) hybrid-FAs
Extended-FA operation

E.g.: $\text{(abc|bcd)}*1y$

Text: $abcdabcdy$
- $a$: 0 – 1($a$)
- $b$: 0 – 3($b$) – 2 ($ab$)
- $c$: 0 – 4($bc$) – 5 ($abc$)
- $d$: 0 – 5 ($bcd$) – 6($abc$)
- $a$: 0 – 1($a$) – 6 ($bcd,bc$)
- $b$: 0 – 3($b$) – 2($ab$) – 6($cd, c$)
- $c$: 0 – 4($bc$) – 5($abc$) – 6($d$) – 7
- $d$: 0 – 5($bcd$) – 6($abc$) – 7
- $y$: 0 – 6($bcd$) – 8