Regular Expression Matching with Multi-Strings and Intervals

Philip Bille
Mikkel Thorup
Outline

- Definition
- Applications
- Previous work
- Two new problems: Multi-strings and character class intervals
- Algorithms
  - Thompson’s algorithm with multi-strings.
  - Decomposition-based algorithms with multi-strings.
  - Character class intervals extensions.
Regular Expressions

• A character $\alpha$ is a regular expression.

• If $S$ and $T$ are regular expressions, then so is

  • The union $S \mid T$

  • The concatenation $ST$ $(S \cdot T)$

  • The kleene star $S^*$
Languages

• The *language* $L(R)$ of a regular expression $R$ is:

$\cdot L(\alpha) = \{\alpha\}$

$\cdot L(S|T) = L(S) \cup L(T)$

$\cdot L(ST) = L(S)L(T)$

$\cdot L(S^*) = \{\epsilon\} \cup L(S) \cup L(S)^2 \cup L(S)^3 \cup \ldots$
Example

• $R = a(a^*)(b|c)$

• $L(R) = \{ab, ac, aab, aac, aaab, aaac, \ldots\}$
Regular Expression Matching

- Given regular expression R and string Q the regular expression matching problem is to decide if $Q \in L(R)$. 
Applications

• Primitive in large scale data processing:
  • Internet Traffic Analysis
  • Protein searching
  • XML queries

• Standard utilities and tools
  • Grep and Sed
  • Perl
Previous Work (Worst-Case Efficient Algorithms)

- Let $|R| = m$ and $|Q| = n$.
- Standard textbook algorithm [Thompson 1968] simulates a non-deterministic automaton (NFA) in $O(nm)$ time.
  - Decompose NFA into tree of small NFAs and combine with tabulation and/or word-level parallelism to speedup Thompson’s algorithm.
  - We will need $O(n (m \log w/ w + \log m))$ time algorithm [B 2006] for our results. Fastest known algorithm for large $w$. 
Problem 1: Multi-Strings

- Many regular expressions consist $k \ll m$ strings.

- Example: Gnutella download stream detection:

  \[(Server:\ |User-Agent:\)(\ |\t)*\text{\{LimeWire|BearShare|Gnucleus|Morpheus|XoloX|gtk-gnutella|Mutella|MyNapster|Qtella|AquaLime|NapShare|Comback|PHEX|SwapNut|FreeWire|Openext|Toadnode\}\]

- $k = 21$ vs. $m = 174$.

- Can we exploit $k \ll m$ in algorithms for regular expression matching?
Problem 2: Character Class Intervals

- For a subset of characters $C$ a *character class interval* $C\{x, y\}$ represents a string of character from $C$ of length at least $x$ and at most $y$.

- Example: $[afg]\{13, 42\}$

- Special case of *gaps* ($\Sigma\{x, y\}$) is important in protein searching.

- We can always convert a character class interval operator to standard operators but this increases the length of regular expression by $y$.

- Can we efficiently implement character class interval operators in regular expression matching?
Outline

• Definition
• Applications
• Previous work
• Two new problems: Multi-strings and character class intervals

• Algorithms
  • Thompson’s algorithm with multi-strings.
  • Decomposition-based algorithms with multi-strings.
  • Character class intervals extensions.
Thompson’s Algorithm

- Recursively construct non-deterministic finite automaton (NFA) from R.
Thompson’s Algorithm

$R = (aba|a)^* \cdot ba$

- **Thompson NFA (TNFA)** $N(R)$ has $O(|R|) = O(m)$ states and transitions.

- $N(R)$ *accepts* $L(R)$. Any path from start to accept state corresponds to a string in $L(R)$ and vice versa.

- Traverse TNFA on $Q$ one character at a time.

- $O(m)$ per character $\Rightarrow O(|Q|m) = O(nm)$ time algorithm.

- Can we get $O(nk)$?
Thompson’s Algorithm with Multi-Strings

- Construct pruned TNFA: Replace strings $L = \{L_1, \ldots, L_k\}$ with single transitions => number of states and transitions is $O(k)$.

- Maintain FIFO bit queue for $L_i$ of length $|L_i|$.

- Preprocess $L$ for fast multi-string matching (Aho-Corasick automaton).
Thompson’s Algorithm with Multi-Strings

- Interleaved traversal of TNFA and multi-string matching on one character from Q at a time:
  - Startpoint of string transition active => Enqueue 1 else 0.
  - Front of queue 1 and match of string => Make endpoint active.

- O(k) states and transition, k queues, multi-string matching is fast => O(k) time per character => Total time O(nk + m log k) and space O(m).
Decomposition Algorithms

- We use NFA-decomposition algorithm based on word-level parallelism [B 2006]:
  - Simplifying assumption: \( m \geq w \).
  - Decompose TNFA into tree of \( O(m/w) \) micro TNFAs, each with at most \( w \) states.
  - Encode each micro TNFA state-set in \( O(w) \) bits.
  - Micro TNFA traversal on a single character in \( O(\log w) \) time using word-level parallelism.
  - \( \Rightarrow \) \( O(m/w \cdot \log w) \) on a single character for entire TNFA
  - \( \Rightarrow O(nm \log w/w) \) algorithm for regular expression matching.
  - Fastest known for large \( w \).
Decomposition Algorithms with Multi-Strings

- Goal: Replace m with k. Process a character in $O(k \log \frac{w}{w})$ time.

- Apply decomposition on pruned TNFA: Tree of $O(k/w)$ micro TNFAs with at most w states and w strings.

- Reuse $\varepsilon$-transition traversal $\Rightarrow O(\log w)$ per micro TNFA

- Reuse multi-string matching algorithm.

- **The missing piece:** How can we maintain w bit queues in $O(\log w)$ time per operation?
Case 1: Short Bit Queues (length \( \leq 2w \))

- First, suppose all queues have the same length!
- Represent queues “vertically”.
- In each step insert input bits in back of queue and output the front of the queue.
- Implicitly move all bits forward by updating the pointer to the start of the queues.
- \( \Rightarrow O(1) \) time per step.
Case 1: Different lengths?
Case 1: Short Bit Queues (length $\leq 2w$)

- With bit mask and standard bitwise operation we can implement each jump point in $O(1)$ time.
- $\Rightarrow O(\log w)$ time per step.
Case 2: Long Bit Queues (length > 2w)

- Horizontal representation with vertical front and back buffers of length $w$.
- Enqueue and dequeue from buffers in $O(1)$ time.
- Every $w$ steps (full buffers):
  - Transpose the back buffer and insert into horizontal representation.
  - Transpose the front $w$ entries of the horizontal representation and insert into the front buffer.
- Transpose takes time $O(w \log w)$ [T 1997] => Amortized $O(\log w)$ time per step.
Algorithm Summary

• $O(\log w)$ per character per micro TNFA => $O(k \log w /w)$ per character.
• => total time $O(n (k \log w/w + \log k) + m \log k)$ and space $O(m)$. 
Character Class Intervals

• New technique to maintain w counters in parallel with reset and decrement operations.

• Combine with bit queues to support character class intervals.