Faster Regular Expression Matching

Philip Bille
Mikkel Thorup

Outline

- Definition
- Applications
- History tour of regular expression matching
    - Thompson’s algorithm
    - Myers’ algorithm
    - New algorithm
- Results and extensions

Regular Expressions

- A character α is a regular expression.
- If S and T are regular expressions, then so is
  - The *union* S | T
  - The *concatenation* ST (S·T)
  - The *kleene star* S*

Languages

- The *language* L(R) of a regular expression R is:
- $L(\alpha) = \{\alpha\}$
- $L(S|T) = L(S) \cup L(T)$
- $L(ST) = L(S)L(T)$
- $L(S^*) = \{\varepsilon\} \cup L(S) \cup L(S)^2 \cup L(S)^3 \cup \ldots$

An example

- R = a(a*)(b|c)
- L(R) = {ab, ac, aab, aac, aaab, aaac, ...}

Regular Expression Matching

- Given regular expression R and string Q the regular expression matching problem is to decide if $Q \in L(R)$.
- How fast can we solve regular expression matching for $|R| = m$ and $|Q| = n$?

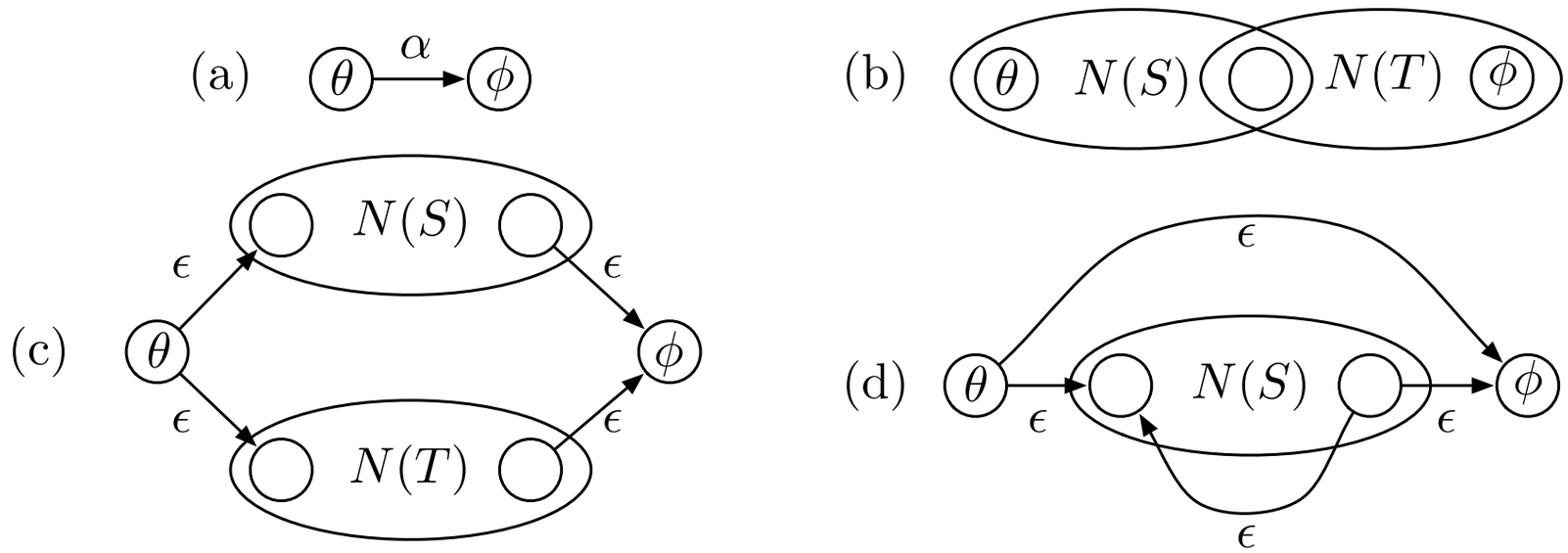
Applications

- Primitive in large scale data processing:
  - Internet Traffic Analysis
  - Protein searching
  - XML queries
- Standard utilities and tools
  - Grep and Sed
  - Perl

Outline

- Definition
- Applications
- **History tour of regular expression matching**
    - Thompson’s algorithm
    - Myers’ algorithm
    - New algorithm
- Results and extensions

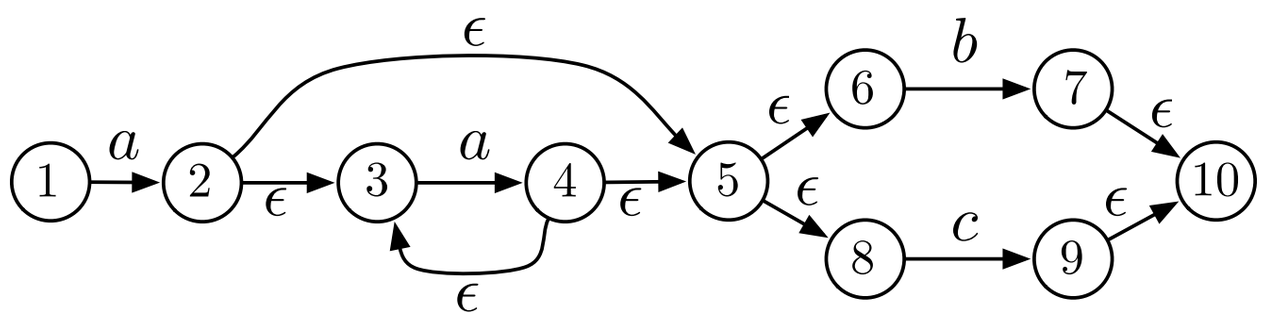
Thompson's Algorithm 1968



- Construct non-deterministic finite automaton (NFA) from R.

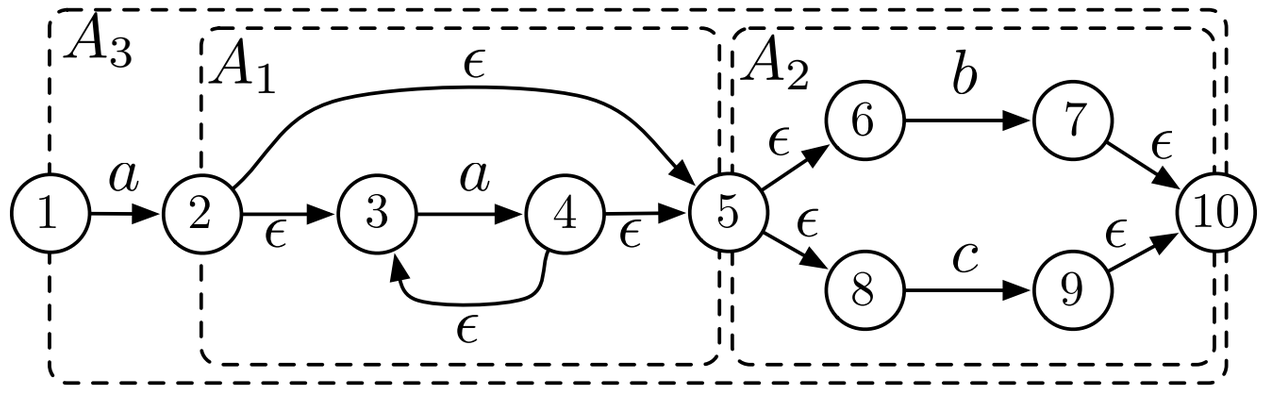
Thompson's Algorithm 1968

R = a·(a*)·(b|c)



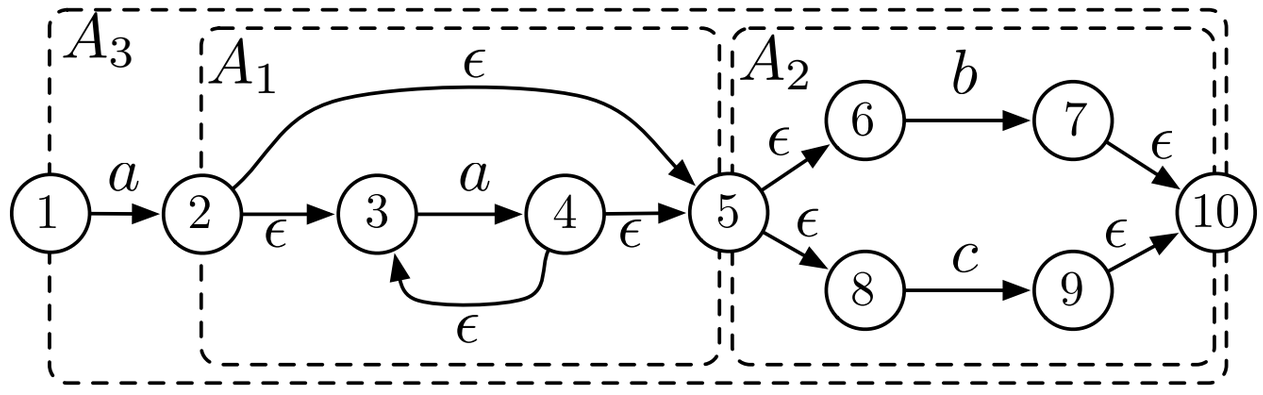
- *Thompson NFA* (TNFA) N(R) has O(|R|) = O(m) states and transitions.
- N(R) *accepts* L(R). Any path from start to accept state corresponds to a string in L(R) and vice versa.
- Traverse TNFA on Q one character at a time.
- O(m) per character => O(|Q|m) = O(nm) time algorithm.
- Top ten list of problems in stringology 1985 [Galil1985].

Myers' Algorithm 1992: 1-D decomposition



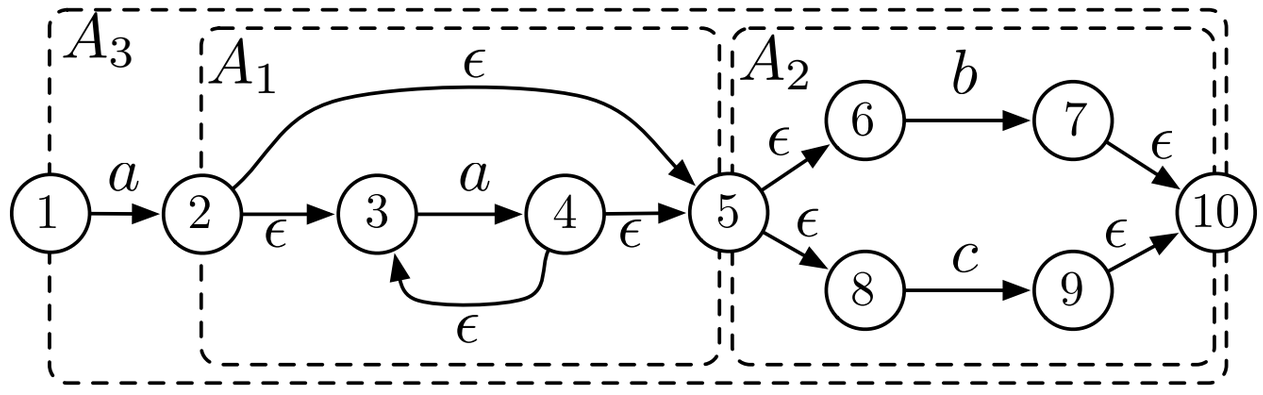
- Decompose N(R) into tree O(m/x) micro TNFAs with at most x = Θ(log n) states.
- State-sets and micro TNFAs encoded in O(x) bits ([BFC2008]).
- Tabulate state-set transitions for micro-TNFAs. Table size: $2^{O(x)} = O(n^{\varepsilon})$.
- => constant time for micro TNFA state-set transition.
- O(m/x) = O(m/log n) state-set transition algorithm for N(R).
- O(nm/log n) algorithm.

How can we improve Myers' algorithm?



- Fast algorithms [Myers1992, BFC2008, Bille2006] all aim to speedup state-set transition for 1 character.
- To read/write a state-set we need Ω(m/log n) time (we assume log n word length).
- To improve we need to handle multiple characters quickly.
- Main challenge is dependencies from ε-transitions.

New Algorithm: 2-D decomposition



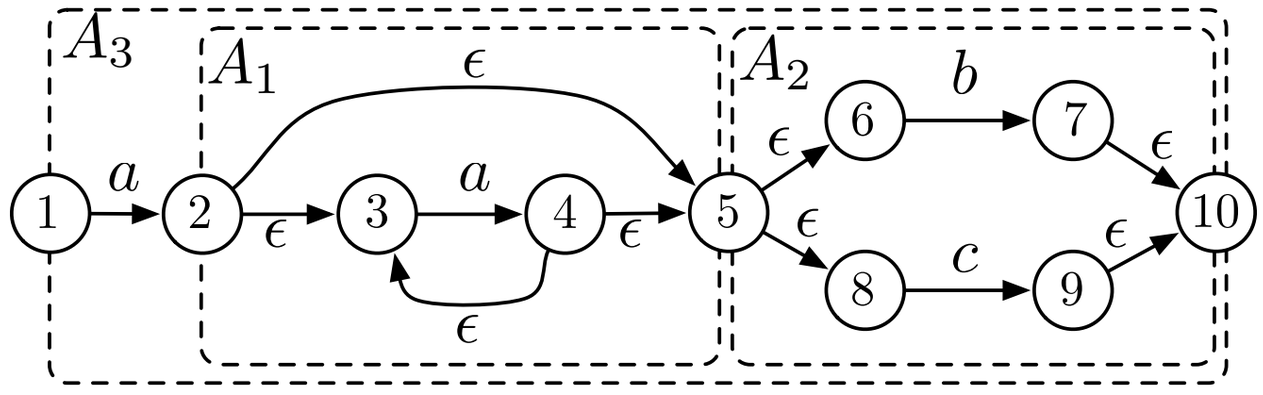
- Decompose N(R) into O(m/x) micro TNFAs with at most x = Θ(log n) states as Myers' algorithm.
- Partition Q into segments of length y = $\Theta(\log^{1/2} n)$.
- State-set transition on segments in O(m/x) time.
- => algorithm using $O(nm/xy) = O(nm/\log^{1,5} n)$ time.

Overview

- Goal: Do a state set transition on $y = \Theta(\log^{1/2} n)$ characters in $O(m/x) = O(m/\log n)$ time.
- Simplifying assumption: constant size alphabet.
- Algorithm: 4 traversals on tree of micro TNFAs.
  - 1-3 iteratively “builds” information.
  - 4 computes the actual state-set transition.
- Tabulation to do each traversal in constant time per micro TNFA
- => O(m/x) time algorithm.

Computing Accepted Substrings

q = aab

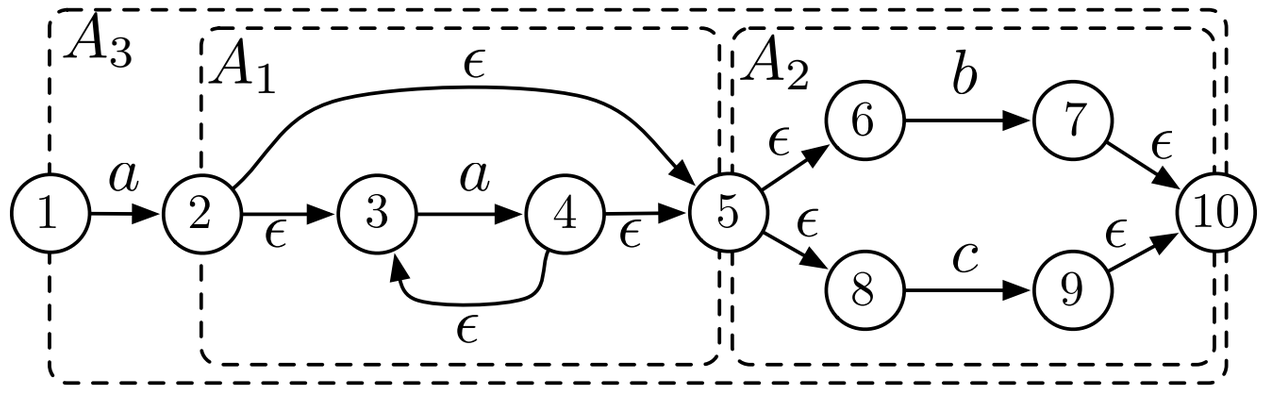


- Goal: For micro TNFA A compute the substrings of q that are accepted by $\bar{A}$. We have $A_1 : \{\varepsilon,a,aa\}$, $A_2 : \{b\}$, $A_3 : \{ab,aab\}$.
- Bottom-up traversal using tabulation in constant time per micro TNFA.
- Encode set of substring in $O(y^2) = O(\log n)$ bits.
- Table input: micro TNFA, substrings of children, q.
- Table size $2^{O(x + y^2 + y)} = 2^{O(x + y^2)} = O(n^{\varepsilon})$.

Computing Path Prefixes to Accepting States

q = aab

S = {1,3}

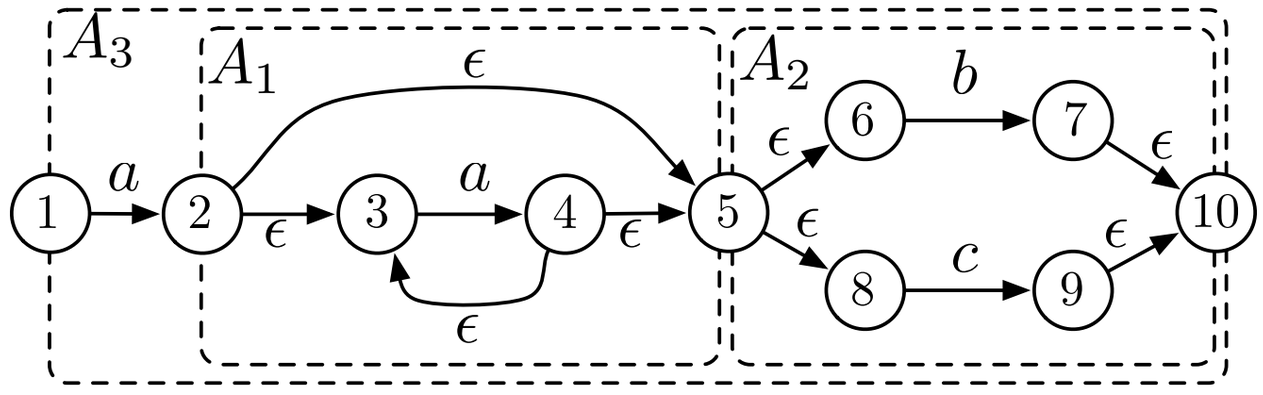


- Goal: For micro TNFA A compute the prefixes of q matching a path from S to the accepting state in $\bar{A}$. We have $A_1 : \{a, aa\}$, $A_2 : \varnothing$, $A_3 : \{aab\}$.
- Bottom-up traversal using tabulation in constant time per micro TNFA.
- Encode prefixes in $O(y) = O(\log^{1/2} n)$ bits.
- Table input: micro TNFA, substrings and path prefixes of children, q, state-set for A.
- Table size $2^{O(x + y^2)} = O(n^{\varepsilon})$.

Computing Path Prefixes to Start States

q = aab

S = {1,3}

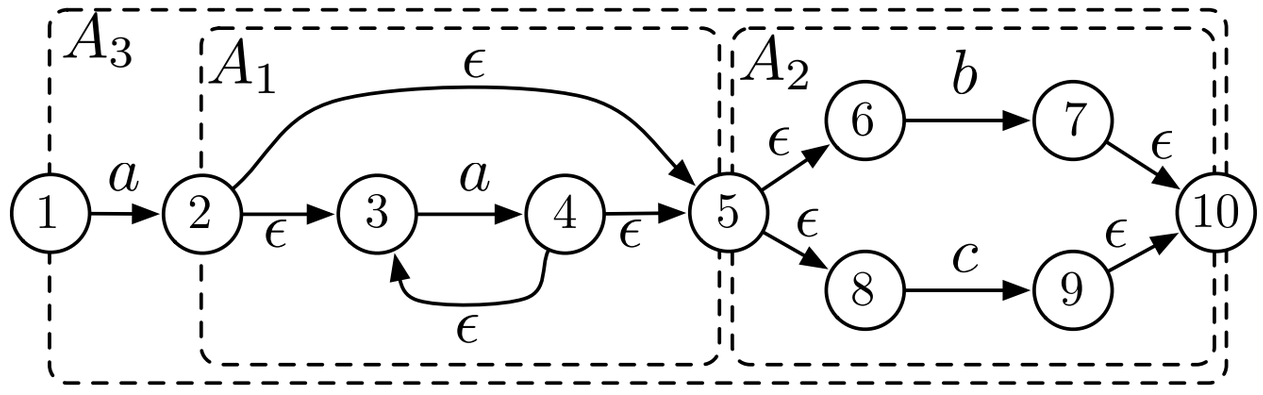


- Goal: For micro TNFA A compute the prefixes of q matching a path from S to the start state in N(R). We have A_1 : {a}, A_2 : {a, aa}, A_3 : {ε}.
- Top-down traversal using tabulation in constant time per micro TNFA.
- Tabulation: Similar to previous traversal.

Updating State-Sets

q = aab

S = {1,3}



- Goal: For micro TNFA A compute the next state-set. We have $A_1 : \varnothing$, $A_2 : \{7,10\}$, $A_3 : \{10\}$.
- Traversal using tabulation in constant time per micro TNFA.
- Tabulation: Similar to previous traversal.

Algorithm Summary

- Tabulation in $2^{O(x + y^2)} = O(n^{\varepsilon})$ time and space.
- 4 traversals each using O(m/x) time to process length y segment of Q.
- => algorithm using $O(nm/xy) = O(nm/\log^{1,5} n)$ time and $O(n^{\varepsilon})$ space.
- Hence, we have $\sqrt{\log n}$'ed (slogn'ed) Myers' result.

Extensions

- Unbounded alphabets cost additional log log n factor in speed.
- Unbounded alphabets for free if $m \leq n^{1-\varepsilon}$.
- I/O bounds: 1-D decomposition gives O(nm/B) I/Os, we get O(nm/ $M^{1/2}$B) I/Os.

- Other features:
  - Independent tabulation.
  - Streaming.