Algorithmic Techniques for Regular Expression Matching

Philip Bille

Regular Expression Matching

Regular Expressions

- Regular expressions.
  - A character α is a regular expression.
    - If S and T are regular expressions, then so is the union $S \mid T$, the concatenation $S \cdot T$ (ST) and the Kleene star S^*.

- Regular languages.
  - The language of a regular expression is given by
    - $L(\alpha) = \{\alpha\}$
    - $L(S \mid T) = L(S) \cup L(T)$
    - $L(S \cdot T) = L(S) \cdot L(T)$
    - $L(S^*) = \{\varepsilon\} \cup L(S) \cup L(S)^2 \cup L(S)^3 \cdots$

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

Regular Expression Matching

- Given a regular expression R and a string Q, decide if Q $\in$ L(R).
- What are the best known time/space bounds for regular expression matching?

Regular Expression Matching

Time	Space	Reference
$O(nm)$	$O(m)$	[Thompson 1968]
$O\left(\frac{nm}{\log n} + (n+m)\log n\right)$	$O(n^{\varepsilon}m)$	[Myers 1992]
$O\left(\frac{nm}{\log n} + n + m\log n\right)$	$O(n^{\varepsilon} + m)$	[B., Farach-Colton 2005]
$O\left(\frac{nm\log w}{w} + n + m\log n\right)$	$O(m)$	[B. 2006]
$O\left(\frac{nm\log\log n}{\log^{3/2} n} + n + m\right)$	$O(n^{\varepsilon} + m)$	[B., Thorup 2009]
$\Omega((nm)^{1-\varepsilon})$		[Backurs, Indyk 2016, Bringmann et al. 2016, Schepper 2020]

$m = |R|$, $n = |Q|$, unit cost word RAM with word length $w \geq \log n$

Regular Expression Matching: Adaptive Bounds

Time	Space	Reference
$O\left(\frac{nk\log w}{w} + n + m\log k\right)$	$O(m)$	[B., Thorup 2010]
$O\left(\Delta \log\log \frac{nm}{\Delta} + n + m\right)$	$O(m)$	[B., Gørtz 2024]
$\Omega(\Delta^{1-\varepsilon})$		[B., Gørtz 2024]

k = number of strings in R, Δ = total size of state sets in simulation of position automaton

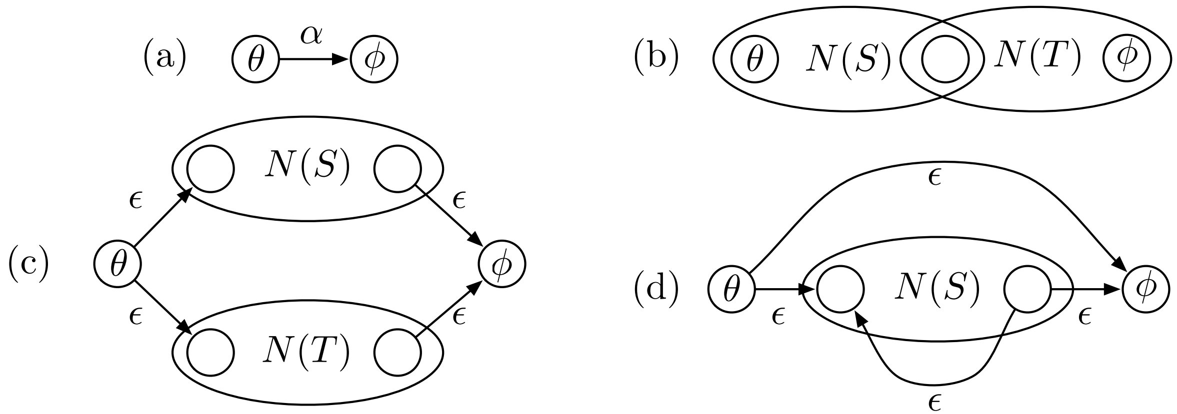
m = |R|, n = |Q|, unit cost word RAM with word length w ≥ log n

Outline

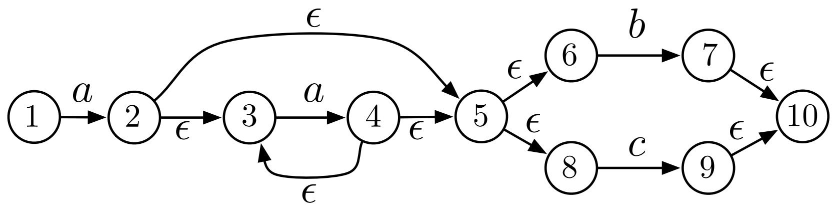
- The problem.
- Results.
- Tour of algorithmic techniques.
  - NFAs and state-set simulation.
  - NFA decomposition and micro TNFAs.
  - Tabulation-based micro TNFA simulation.
  - Word-level parallel micro TNFA simulation.
  - 2D decomposition.
  - Adaptive techniques.
- NFAs vs. DFAs.
- Open problems.

NFAs and State-set Simulation

NFAs and State-Set Simulation

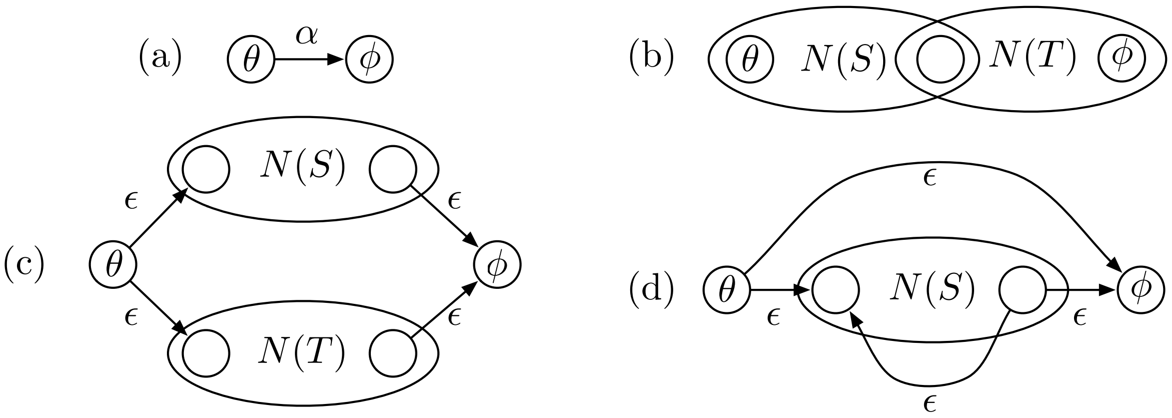


R = a·(a*)·(b|c)

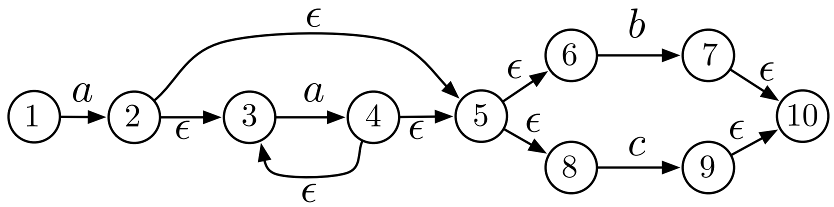


- Non-deterministic finite automaton (NFA).
  - Construct Thompson's NFA (TNFA) N(R) from R [Thompson 1968]
  - N(R) accepts L(R). Any path from start to accept state matches exactly the strings in L(R).
  - O(m) states and transitions.
  - States with an incoming character transition have exactly 1 predecessor.
  - Almost a series-parallel graph:
    - Separator of size 2.
    - Any cycle-free path has at most 1 back edge.

NFAs and State-Set Simulation



R = a·(a*)·(b|c)

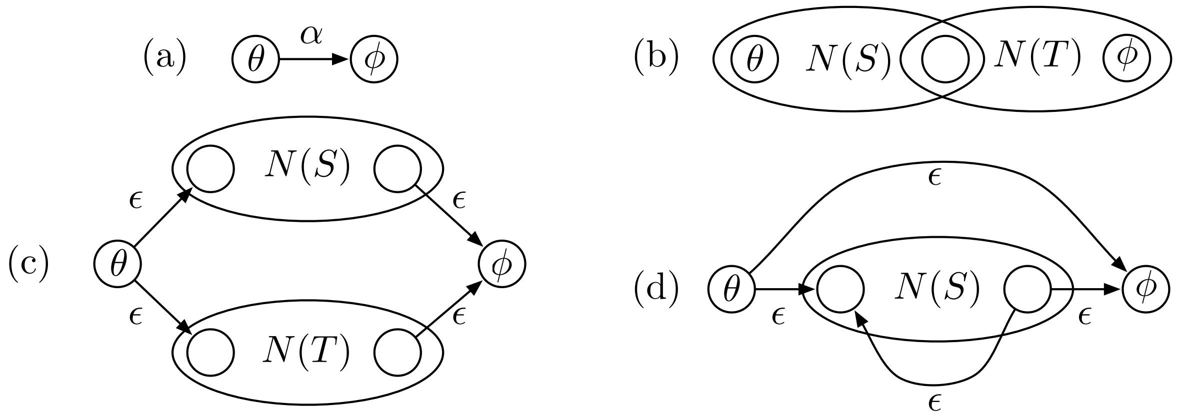


- State-set transition.
  - Given state set S and character α.
  - δ(S, α): set of states reachable from S via paths matching α.

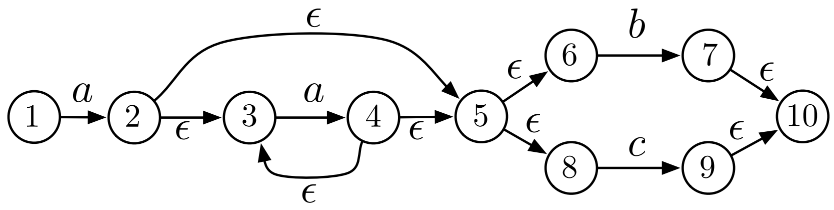
- Split into two operations:
  - Move(S, α): set of states reachable from S via single α-transition.
  - Close(S): set of states reachable from S via paths of ε-transitions.

- O(m) time.

NFAs and State-Set Simulation



R = a·(a*)·(b|c)

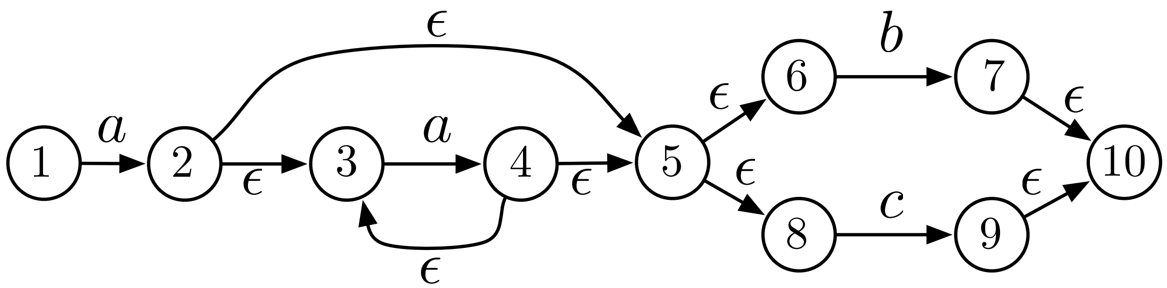


- State-set simulation.
  - Given string Q of length n, compute sequence of state sets S_0, ..., S_n
  - S_0 = Close({θ})
  - S_i = Close(Move(S_{i-1}, α))
- Q ∈ L(R) iff ϕ ∈ S_n.

- ⇒ O(nm) time and O(m) space [Thompson 1968].
- Top ten list of problems in stringology 1985 [Galil 1985].

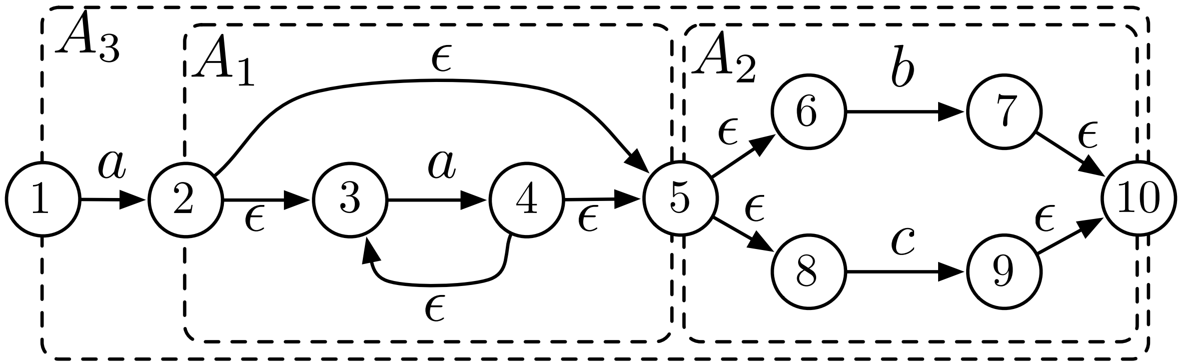
NFA Decomposition

Large and Small TNFAs



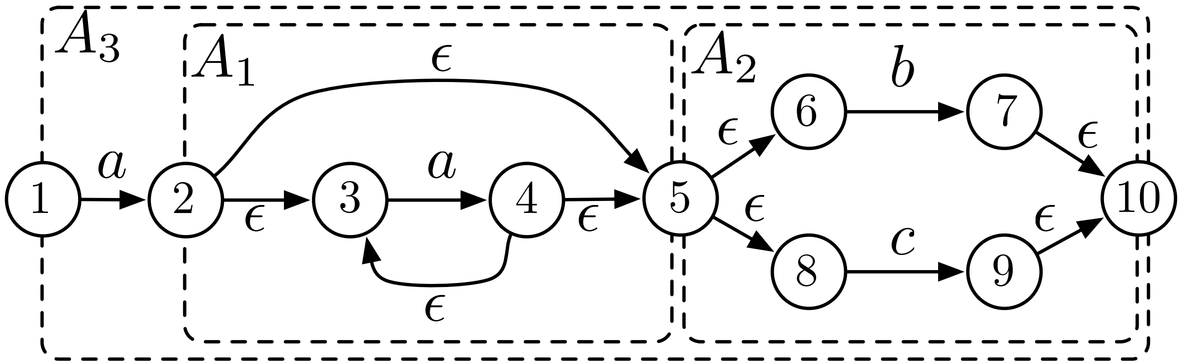
- TNFA decomposition.
  - Suppose we can do state-set transition fast on a micro TNFA of size x « m.
  - Can we use that to get efficient state-set transition for N(R)?
  - Main problem is non-local dependencies from ε-transitions.

Large and Small TNFAs



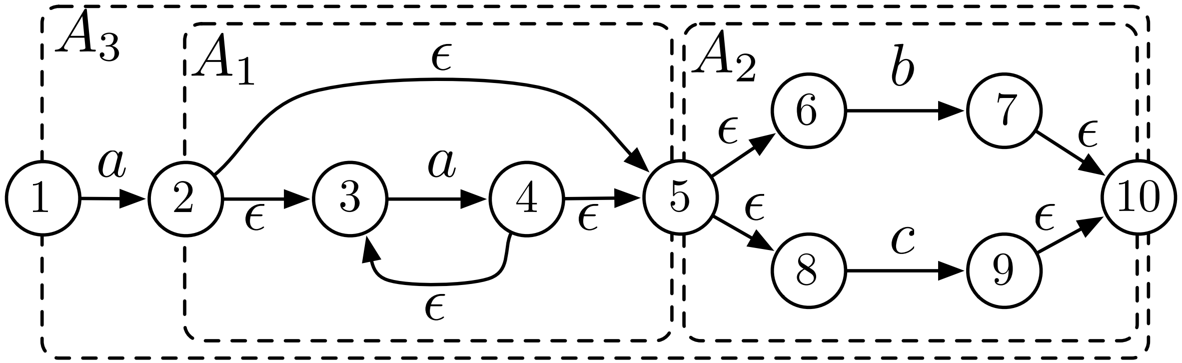
- TNFA decomposition.
  - Decompose N(R) into tree of O(m/x) micro TNFAs with at most x states. Each micro TNFA overlaps with enclosing micro TNFA in 2 states.
  - Implement state-set transition on N(R) by state-set transition on micro TNFAs in topological order twice. Propagate reachable overlapping states.
  - Implement state-set transition on micro TNFA in t(x) time ⇒ state-set transition on N(R) in $O\left(\frac{mt(x)}{x}\right)$ time [Myers1992, B. 2006].

Tabulation for Micro TNFAs



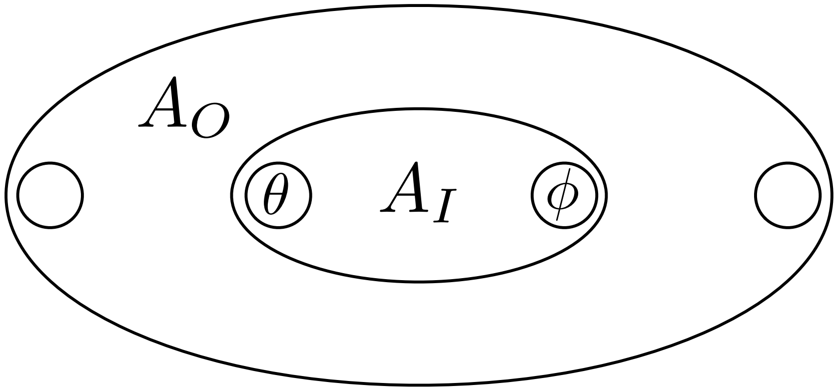
- Tabulation on micro TNFA with x states.
  - Encode state-set as bit string of length x.
  - Move(S, α): Use shift-and technique.
  - Close(S): Universal tabulation for all possible micro TNFAs. Table size $2^{O(x)}$.
  - With x = Θ(log n) ⇒ state-set transition on micro TNFA in constant time and $O(n^\varepsilon)$ space.
  - ⇒ Regular expression matching in $O\left(\frac{nm}{\log n}\right)$ time and $O(n^\varepsilon + m)$ space [B., Farach-Colton 2005].

Word-Level Parallelism for Micro TNFAs



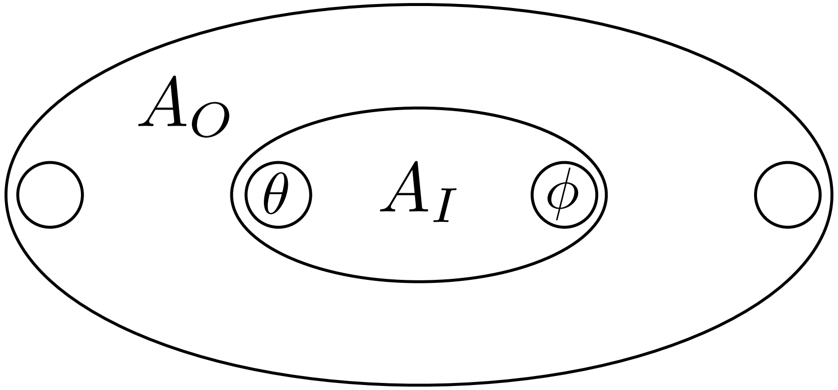
- Word-level parallelism on micro TNFA with Θ(w) states.
  - Can we simulate micro TNFA with bitwise logical and arithmetic operations of the w-bit words instead of tabulation?
  - Main challenge is Close operation. How to deal with long paths of ε-transitions?

Word-Level Parallelism for Micro TNFAs



- Separator decompositions on micro TFNAs
  - There exists two states θ and ϕ whose removal partitions a micro TNFA A into two subgraphs, A_O and A_I, of roughly equal size such that:
  - Any path from A_O to A_I goes through θ.
  - Any path from A_I to A_O goes through ϕ.

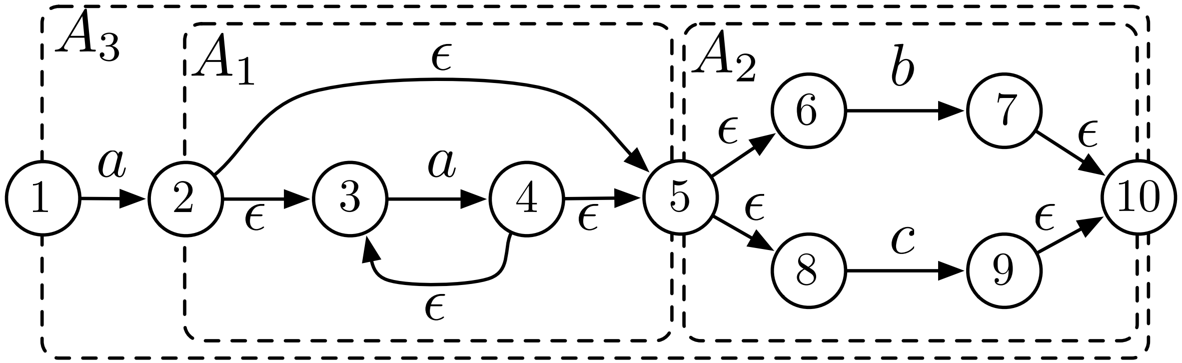
Word-Level Parallelism for Micro TNFAs



- Recursive Close computation
  - Compute which of θ and ϕ are reachable.
  - Update current set of reachable states
  - Recurse on A_O and A_I in *parallel*.
  - O(log w) levels of recursion each using O(1) time ⇒ state-set transition on micro TNFA in O(log w) time.
  - ⇒ Regular expression matching in $O\left(\frac{nm \log w}{w}\right)$ time and O(m) space [B. 2006].

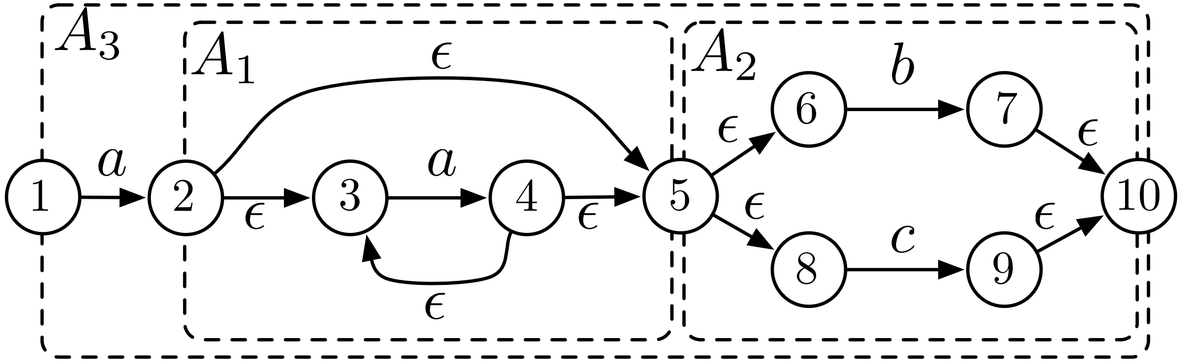
2D Decomposition

Beyond State-Set Simulation



- Limits of state-set transitions algorithms.
  - To explicitly read/write state-sets at each character we need Ω(m/w) time for state-set transition.
  - ⇒ Any algorithm must use Ω(nm/w) time with this approach.
  - Can we process multiple characters quickly?
  - Even larger challenges from non-local ε-transitions.

2D Decomposition Algorithm



- Processing multiple characters at a time.
  - Decompose N(R) into O(m/x) micro TNFAs with at most x = Θ(log n) states (as earlier).
  - Partition Q into segments of length y = Θ($\log^{1/2}$ n).
  - State-set transition on segments in O(m/x) time.
  - ⇒ Regular expression matching in O(nm/xy) = O(nm/$\log^{1,5}$n) time [B., Thorup 2009].

2D Decomposition Algorithm

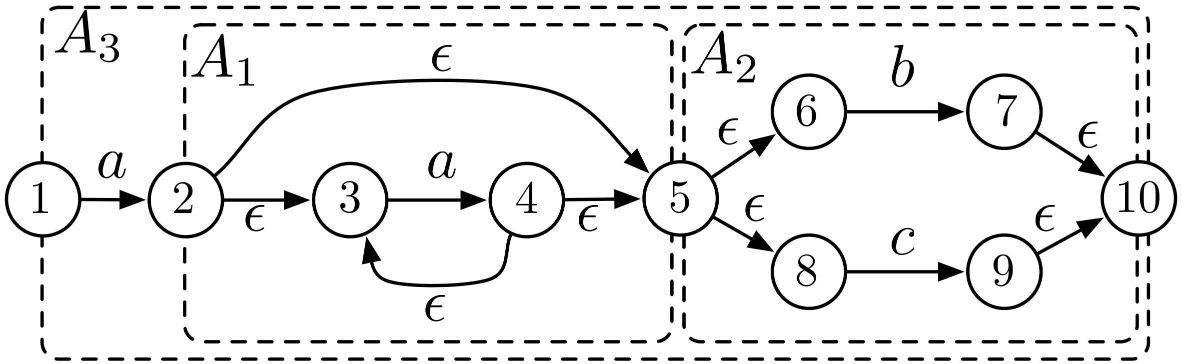
- Goal. Do a state set transition on $y = \Theta(\log^{1/2} n)$ characters in $O(m/x) = O(m/\log n)$ time.

- Algorithm overview. 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.

2D Decomposition Algorithm

q = aab



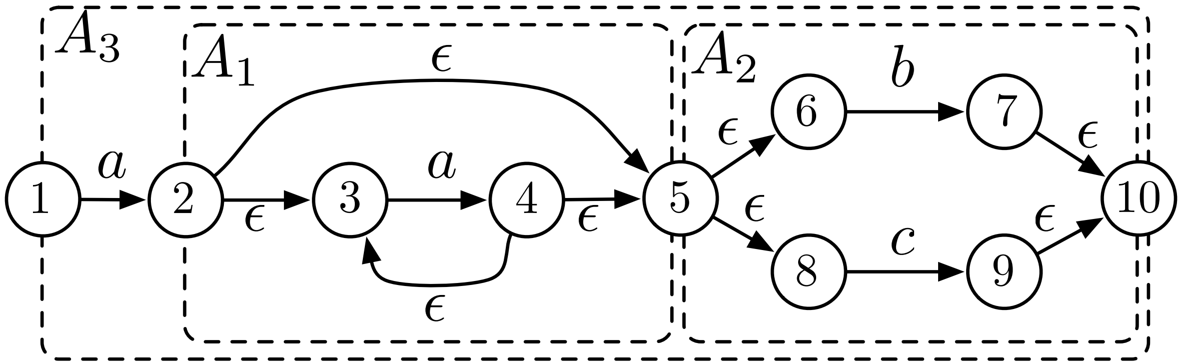
- Step 1. Computing Accepted Substrings.
  - Goal: For micro TNFA A compute the substrings of q that are accepted by $\bar{A}$. We have A_1 : {ε,a,aa}, A_2 : {b}, A_3 : {ab,aab}.

  - Bottom-up traversal using tabulation in constant time per micro TNFA.
  - Encode set of substrings in $O(y^2) = O(\log n)$ bits.
  - Table input: micro TNFA, substrings of children, q.
  - Table size $2^{O(x + y^2 + y)} = 2^{O(x + y^2)} = O(n^\varepsilon)$.

2D Decomposition Algorithm

q = aab

S = {1,3}

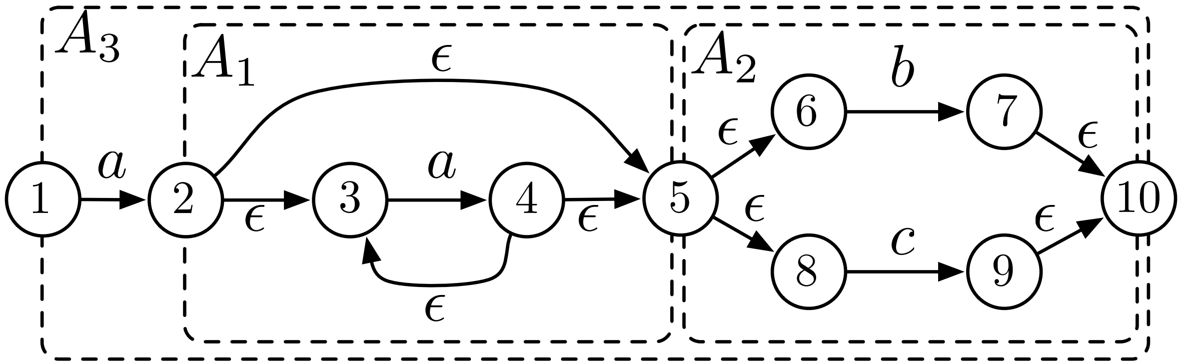


- Step 2. Computing Path Prefixes to Accepting States.
  - Goal: For micro TNFA A compute the prefixes of q matching a path from S to the accepting state in $\bar{A}$. We have A_1 : {a, aa}, A_2 : ∅, A_3 : {aab}.
  - Bottom-up traversal using tabulation in constant time per micro TNFA.
  - Encode prefixes 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 $2^{O(x + y^2)} = O(n^{\varepsilon})$.

2D Decomposition Algorithm

q = aab

S = {1,3}



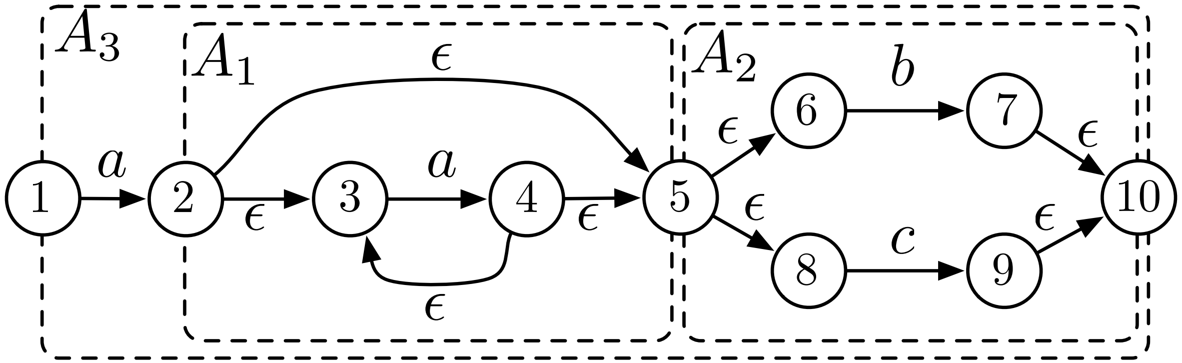
- Step 3. Computing Path Prefixes to Start States.
  - Goal: For micro TNFA A compute the prefixes of q matching a path from S to the start state in N(R). We have A_1 : {a}, A_2 : {a, aa}, A_3 : {ε}.

  - Top-down traversal using tabulation in constant time per micro TNFA.
  - Tabulation: Similar to previous traversal.

2D Decomposition Algorithm

q = aab

S = {1,3}



- Step 4. Updating State-Sets
  - 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.

2D Decomposition 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.
  - $\Rightarrow$ Regular expression matching in $O(nm/xy) = O(nm/\log^{1,5} n)$ time and $O(n^\varepsilon + m)$ space [B., Thorup 2009]

Adaptive Techniques

Regular Expression Matching with Multi-Strings

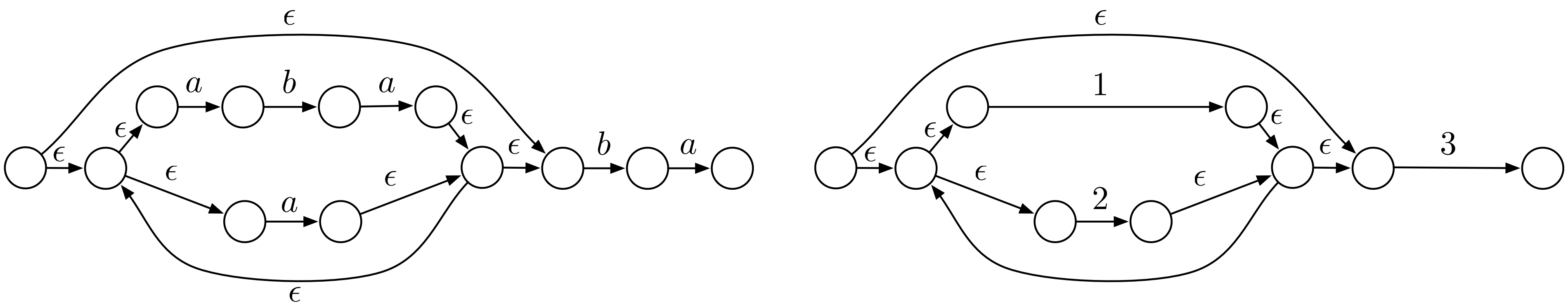
- Many regular expressions consist of $k \ll m$ strings.

- Example. Gnutella download stream detection:
- `(Server:|User-Agent:)( |\t)*(LimeWire|BearShare|Gnucleus|Morpheus|XoloX|gtk-gnutella|Mutella|MyNapster|Qtella|AquaLime|NapShare|Comback|PHEX|SwapNut|FreeWire|Openext|Toadnode)`

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

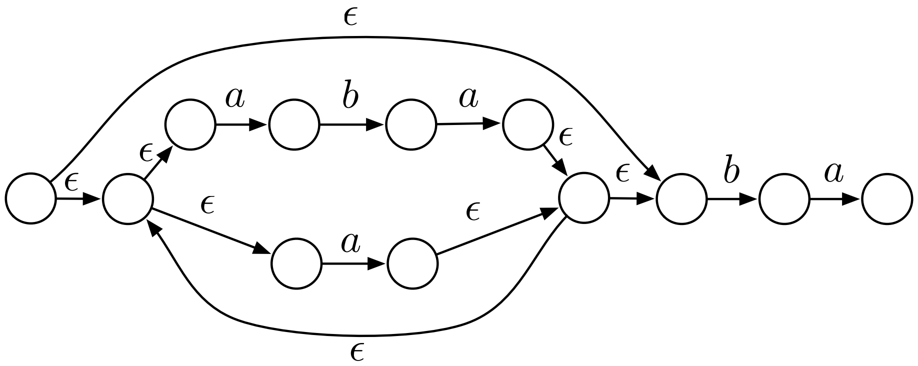
- Can we exploit $k \ll m$ in algorithms for regular expression matching?

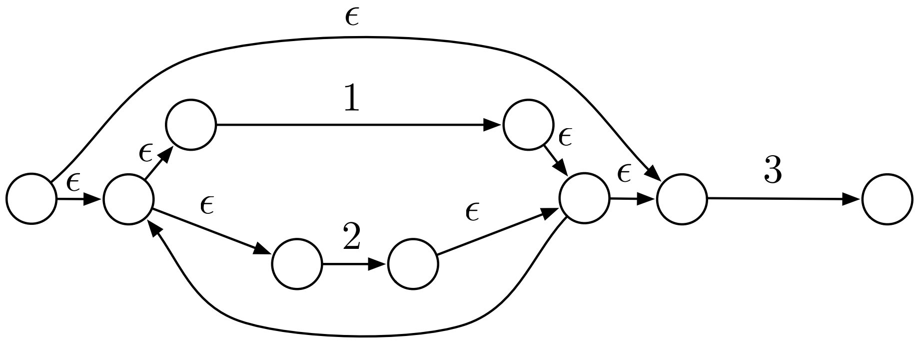
Regular Expression Matching with Multi-Strings



- Algorithm components.
  - Construct *pruned* TNFA: Replace strings L = {L_1, ..., 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).

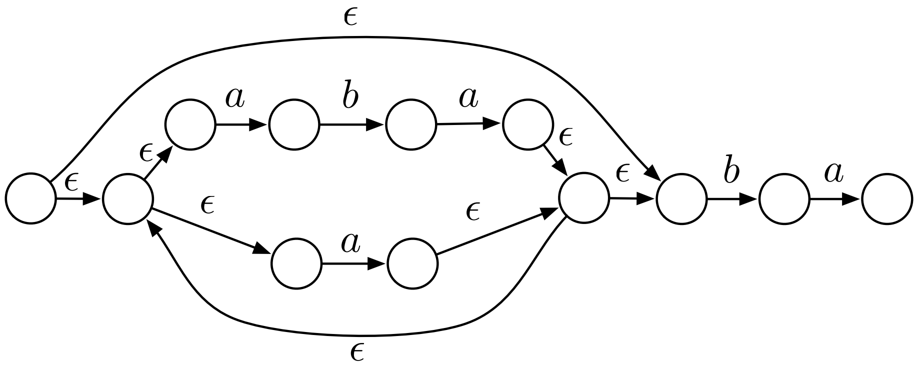
Regular Expression Matching with Multi-Strings

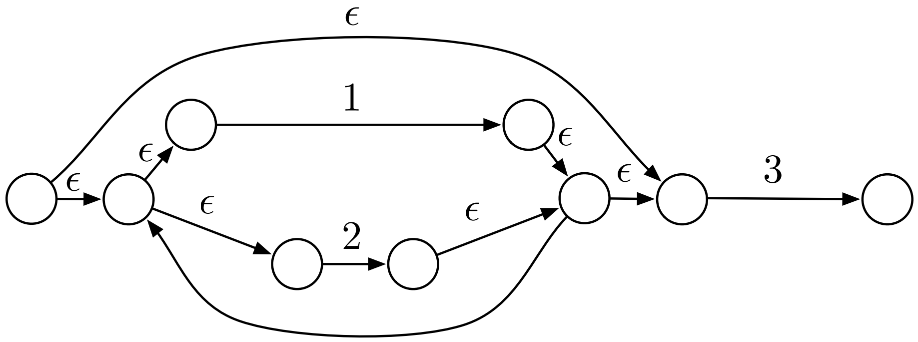




- Algorithm processing.
  - Interleaved traversal of pruned TNFA and multi-string matching on one character from Q at a time:
    - Startpoint of string transition active => Enqueue 1 else 0.
    - Front of queue 1 and match of string => Make endpoint active.
- O(k) states and transition, k queues, multi-string matching is fast ⇒ O(k) time per character
- ⇒ Regular expression matching in O(nk) time and O(m) space [B., Thorup 2010].

Regular Expression Matching with Multi-Strings





- Decomposition with multi-strings.
  - Apply decomposition on pruned TNFA: tree of O(k/w) micro TNFAs with at most w states and w strings.
  - Apply Close operation based on word-level parallelism [B. 2006] ⇒ O(log w) per micro TNFA
  - Reuse multi-string matching algorithm.
  - Advanced word-level parallel techniques to maintain parallel bit-queue, etc.
  - ⇒ regular expression matching in $O\left(\frac{nk\log w}{w}\right)$ time and O(m) space [B., Thorup 2010].

NFAs vs DFAs

NFAs vs DFAs

- We can convert TNFA N(R) to a deterministic finite automaton (DFA) using the subset construction.
  - $\Rightarrow$ DFA D with $O(2^{2m})$ states and $O(2^{2m}\sigma)$ transitions.
  - Only need to keep track of a single state during matching.
  - Does this imply solution to regular expression matching in $O(2^{2m}\sigma + n)$ time?
- No! Only true for $m = O(w)$.
  - Word RAM model options:
    - Space is limited to 2^w.
    - Any data structure D with $2^{\Omega(m)}$ space needs $\Omega(m/w)$ time just to specify address in D.
  - $\Rightarrow$ a state-transition in D uses $\Omega(m/w)$ time $\Rightarrow$ regular expression matching in $\Omega\left(\frac{nm}{w}\right)$ time

    [B. 2015].

Open Problems

Open Problems

- Better than 2D decomposition.
- Other adaptive measures.
- Regular expression extensions and variations.

References

- Kleene. Representation of events in nerve nets and finite automata. Ann. Math. Stud. 1956.
- Thompson. Regular expression search algorithm. CACM 1968.
- Galil. Open problems in stringology. In A. Apostolico and Z. Galil, editors, Combinatorial problems on words, NATO ASI Series 1985.
- Myers. A four-russian algorithm for regular expression pattern matching. JACM 1992.
- Bille and Farach-Colton. Fast and compact regular expression matching. Theoret. Comput. Sci. 2008.
- Bille. New algorithms for regular expression matching. ICALP 2006.
- Bille and Thorup. Faster regular expression matching. ICALP 2009
- Bille and Thorup. Regular Expression Matching with Multi-Strings and Character-Class Intervals, SODA 2010
- Bille. On Regular Expression Matching and Deterministic Finite Automata, Tiny Transactions on Computer Science 3, 2015.
- Backurs and Indyk. Which Regular Expressions Patterns are Hard to Match? FOCS 2016.
- Bringmann, Grønlund, and Larsen. A dichotomy for regular expression membership testing FOCS 2017.
- Schepper. Fine-grained complexity of regular expression pattern matching and membership. ESA 2020.
- Bille and Gørtz. Sparse Regular Expression Matching, SODA 2024