Prefix-Free Regular-Expression Matching

Yo-Sub Han*, Yajun Wang and Derick Wood

Department of Computer Science
HKUST

Pattern Matching

Given a pattern P and a text T, find all substrings of T that are in P.

- $|P| = 1$: string pattern matching [BM, KMP]
- $|P| = k$: keyword pattern matching [AC]
- P is a regular expression: regular-expression pattern matching !!!

Overview

- Basic Notions
- Related Work
- Regular-Expression Matching
  - Infix-Free Regular-Expression Matching
  - Prefix-Free Regular-Expression Matching
- Determine whether or not $L(E)$ is prefix-free
- Conclusions

Basic Notions

An automaton A is specified by a tuple $(Q, \Sigma, \delta, s, F)$;

Q a finite set of states

Σ a finite alphabet

$\delta \subseteq Q \times \Sigma \times Q$

$s \in Q$ a start state

$F \subseteq Q$ a set of final states

- $\lambda =$ the null-string symbol
- $|A| = |Q| + |\delta|$
- $|E| =$ the number of character appearances in a given regular expression E

Basic Notions

Given a transition (p, a, q) in δ

- p has an out-transition
- q has an in-transition
- p is a source state of q
- q is a target state of p
- A to be non-returning if the start state of A does not have any in-transitions
- A to be non-exiting if a final state of A does not have any out-transitions

Basic Notions

Given two strings x and y over Σ, we say

- x is a prefix of y if there exists $z \in \Sigma^*$ such that $xz = y$.
- x is an infix of y if there exists $u, v \in \Sigma^*$ such that $uxv = y$; we often call x a substring of y.

Basic Notions

We define a language L to be

- prefix-free if no string in L is a prefix of any other strings in L.
- infix-free if no string in L is an infix of any other strings in L.

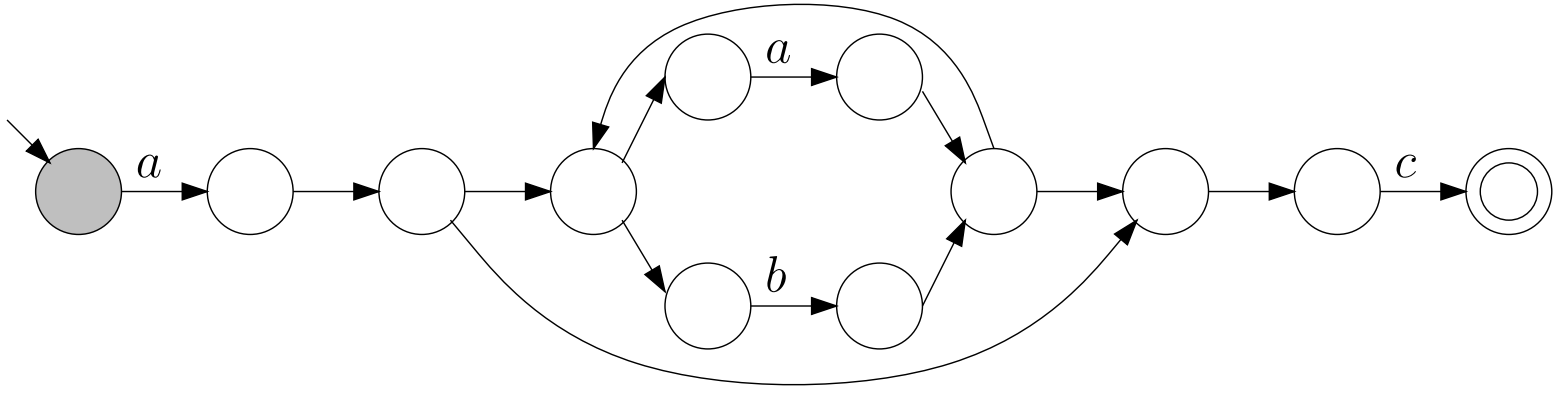
Related Work

Given a regular expression E and a text T,

- The membership problem: We can determine whether or not $T \in L(E)$ in $O(mn)$ time [Thompson]
- The decision problem: We can determine whether or not there is a substring of T that is in $L(E)$) in $O(mn)$ time [Aho] or in $O(m \log n)$ time [Myers]
- The recognition problem: We can report all end positions of matching substrings of T in $O(mn)$ time [Aho] or in $O(m \log n)$ time [Myers]
- The identification problem: We can report all (start, end) positions of matching substrings of T in $O(mn \log n)$ time [Myers et al.]

The Membership Problem

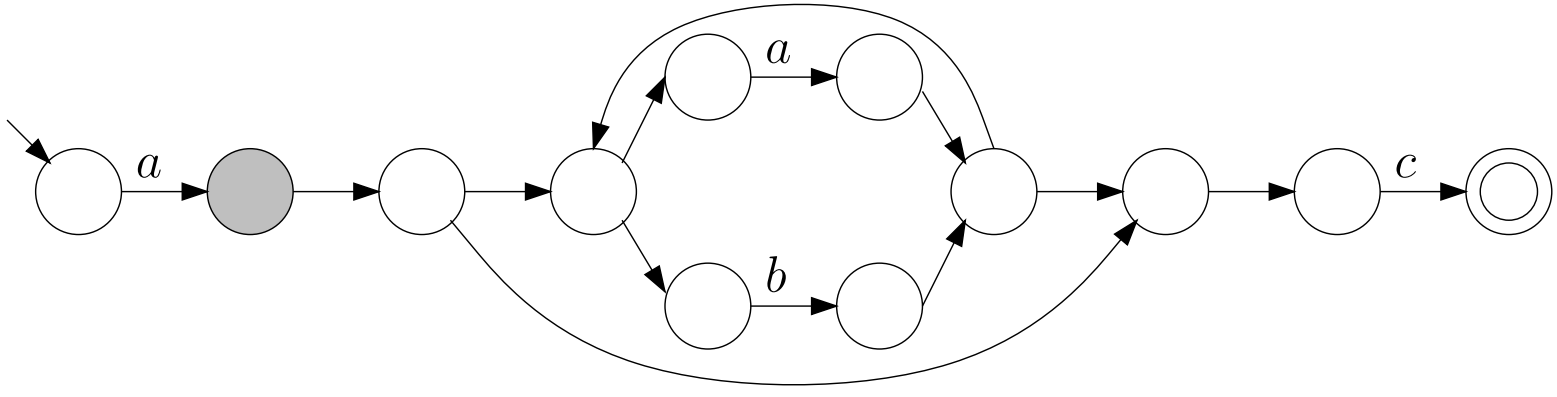
$E = a(a+b)^*c$ and $T = aabbac$



$a\ a\ b\ b\ a\ c$

The Membership Problem

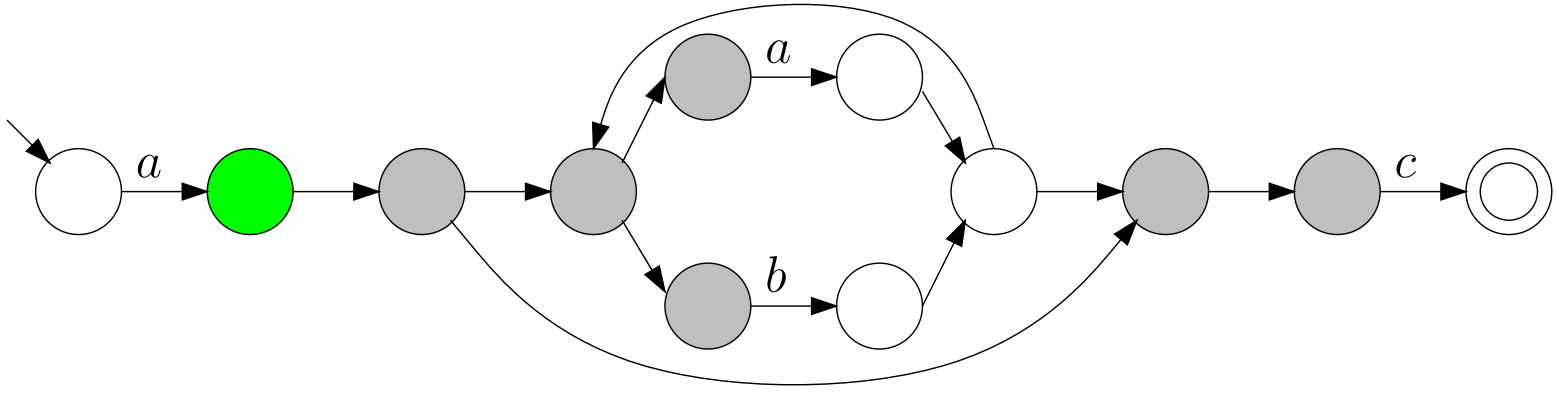
$E = a(a+b)^*c$ and $T = aabbac$



$\boxed{a}\ a\ b\ b\ a\ c$

The Membership Problem

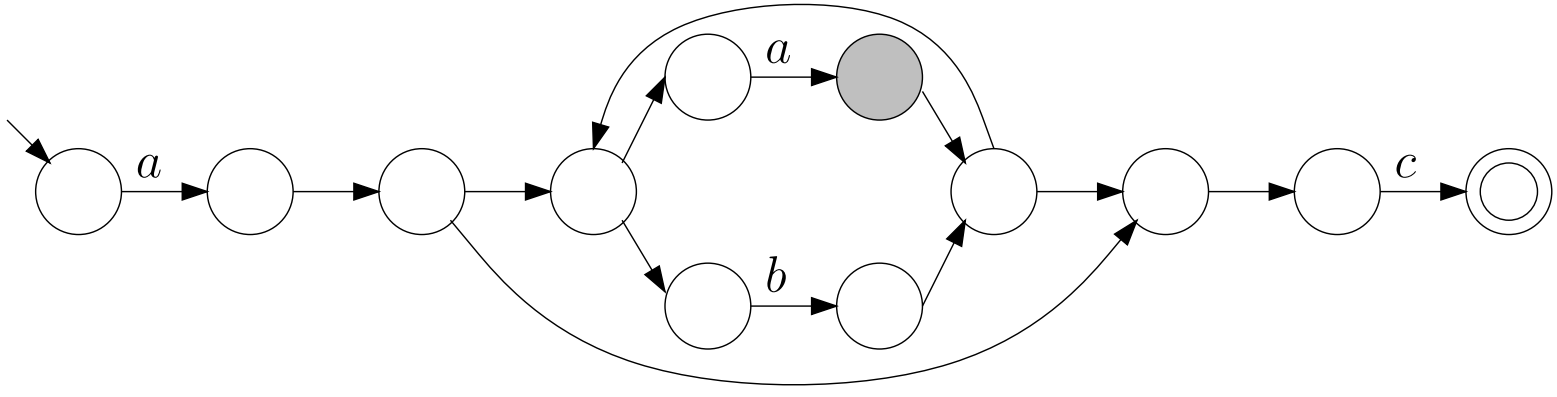
$E = a(a+b)^*c$ and $T = aabbac$



$\boxed{a}\ a\ b\ b\ a\ c$

The Membership Problem

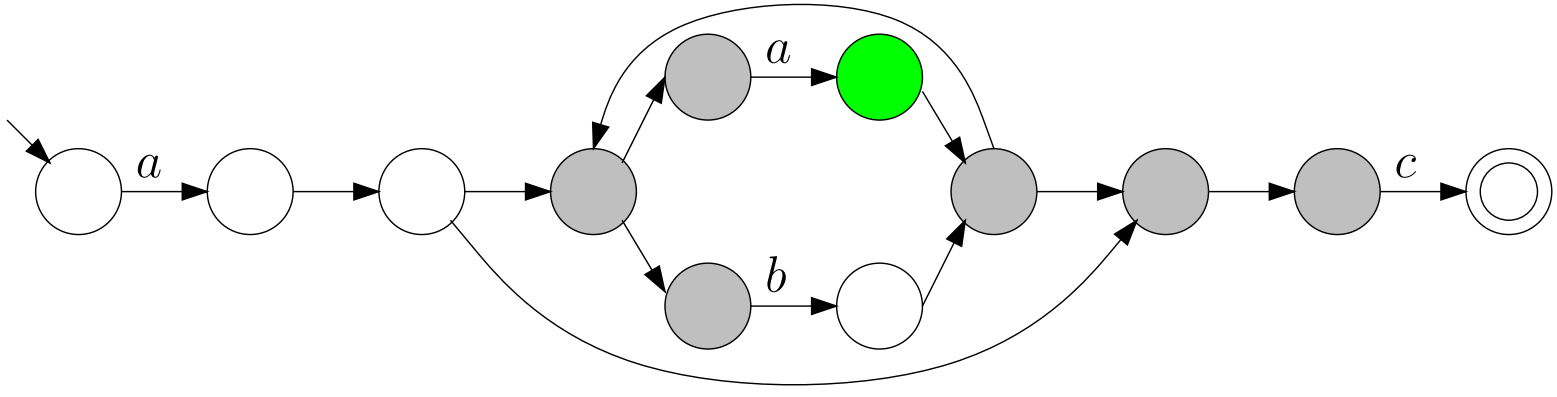
$E = a(a+b)^*c$ and $T = aabbac$



$a\ a\ b\ b\ a\ c$

The Membership Problem

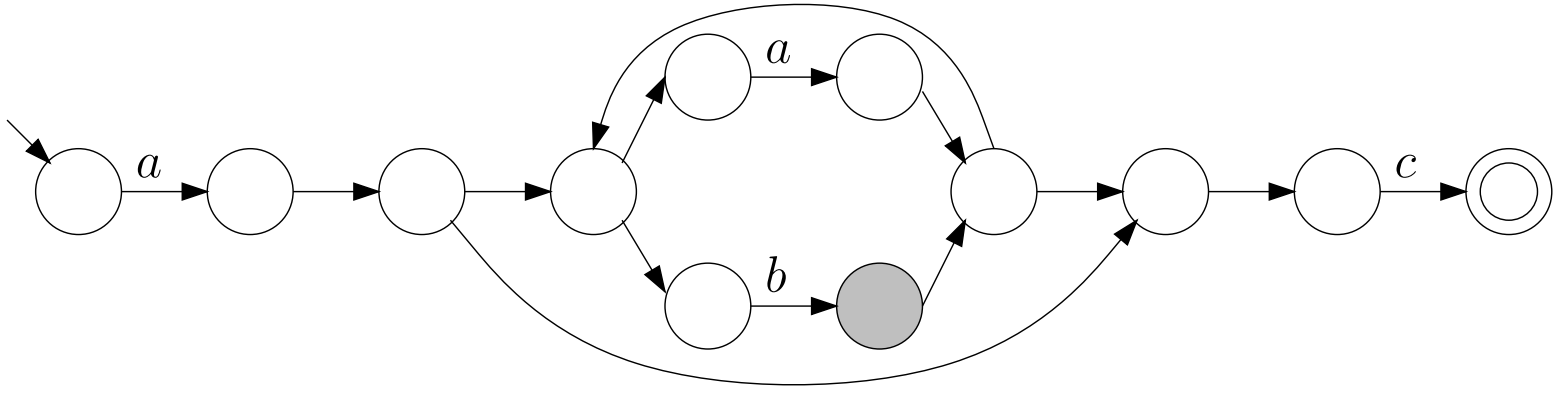
$E = a(a+b)^*c$ and $T = aabbac$



$a\ a\ b\ b\ a\ c$

The Membership Problem

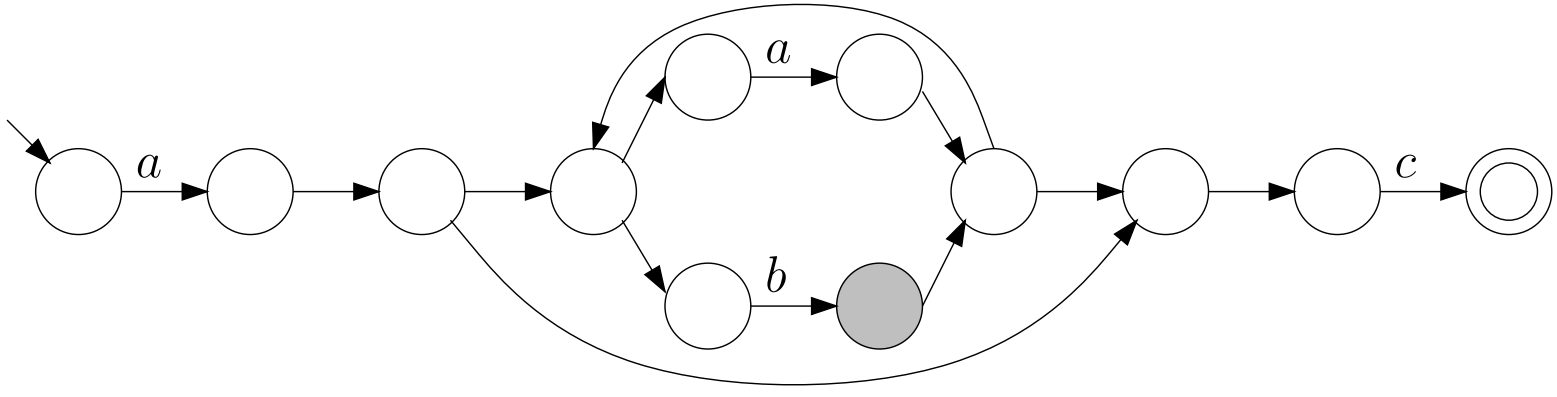
$E = a(a+b)^*c$ and $T = aabbac$



$a\ a\ b\ b\ a\ c$

The Membership Problem

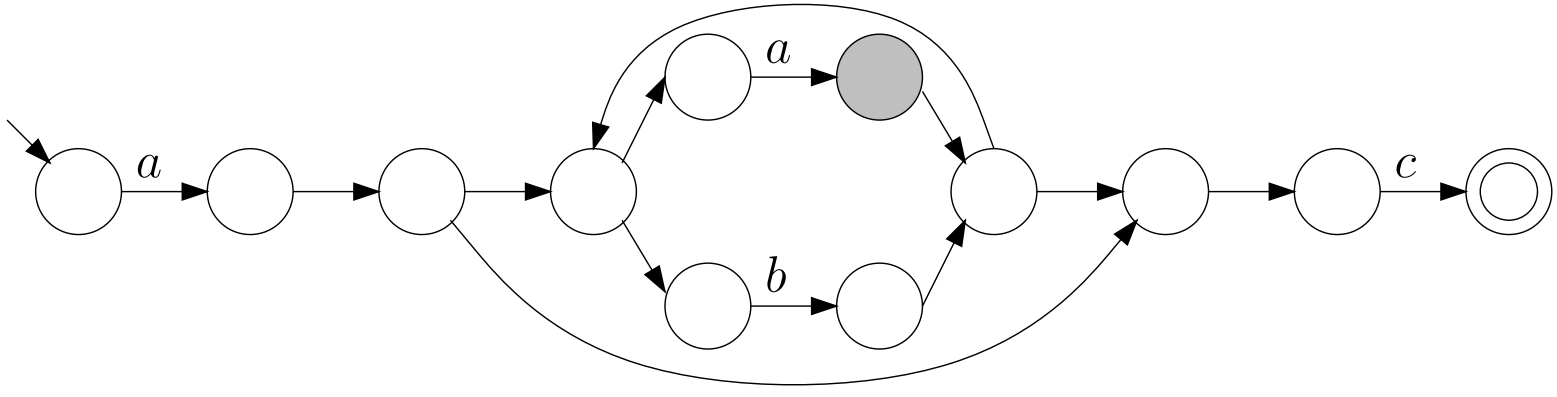
$E = a(a+b)^*c$ and $T = aabbac$



$\boxed{a\ a\ b\ b}\ a\ c$

The Membership Problem

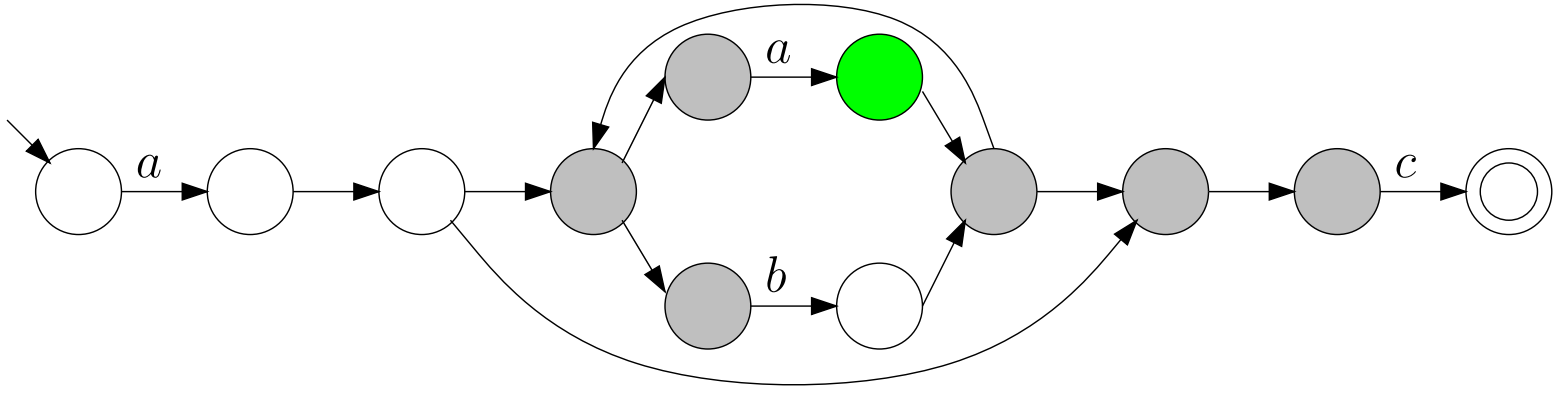
$E = a(a+b)^*c$ and $T = aabbac$



$a\ a\ b\ b\ a\ c$

The Membership Problem

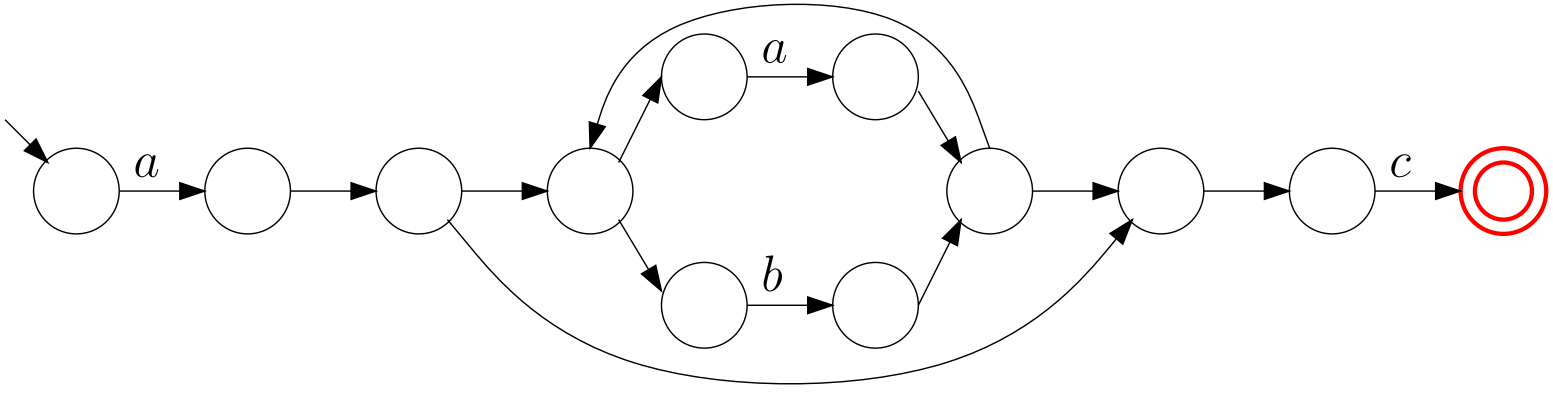
$E = a(a+b)^*c$ and $T = aabbac$



$a\ a\ b\ b\ a\ c$

The Membership Problem

$E = a(a+b)^*c$ and $T = aabbac$

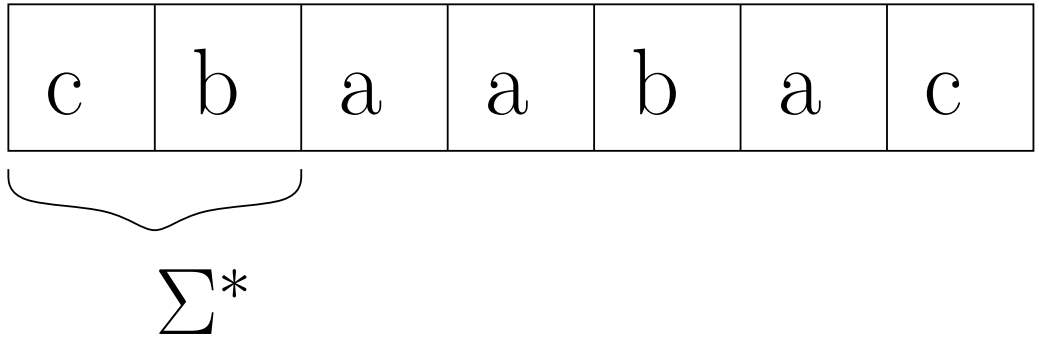


$a\ a\ b\ b\ a\ c$

The Recognition Problem

Given E over Σ, we prepend Σ^* to E; thus, allowing matching to begin at any position in T.

$$\Sigma^* \cdot a(a+b)^*a$$



The Recognition Problem

Given E over Σ, we prepend Σ^* to E; thus, allowing matching to begin at any position in T.

ExpressionMatching (A, T)

$Q = null(\{s\})$

if $f \in Q$ **then output** λ

for j=1 **to** n

  $Q = null(goto(Q, w_j))$

  if $f \in Q$ **then output** j

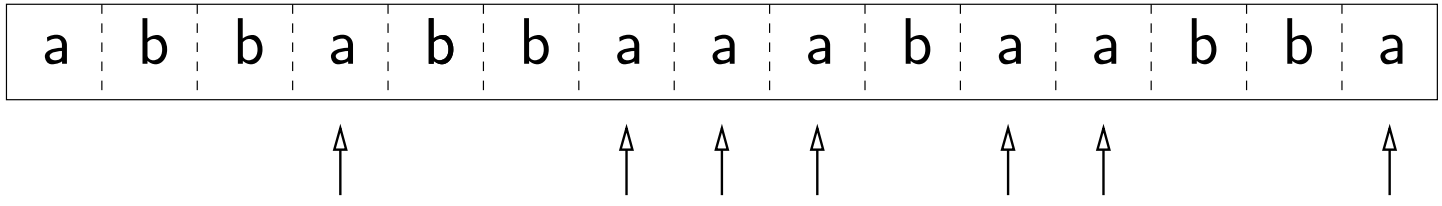
- $null(Q)$ computes all states in A that can be reached from a state in the set Q of states by null transitions
- $goto(Q, w_j)$ gives all states that can be reached from a state in Q by a transition with w_j, the current input character

The Recognition Problem

Given E over Σ, we prepend Σ^* to E; thus, allowing matching to begin at any position in T.

$$E = a(a+b)^*a$$

T



Given a regular expression E and a text T, we can find all end positions of matching substrings of T in $O(mn)$ worst-case time using $O(m)$ space [Crochemore and Hancart].

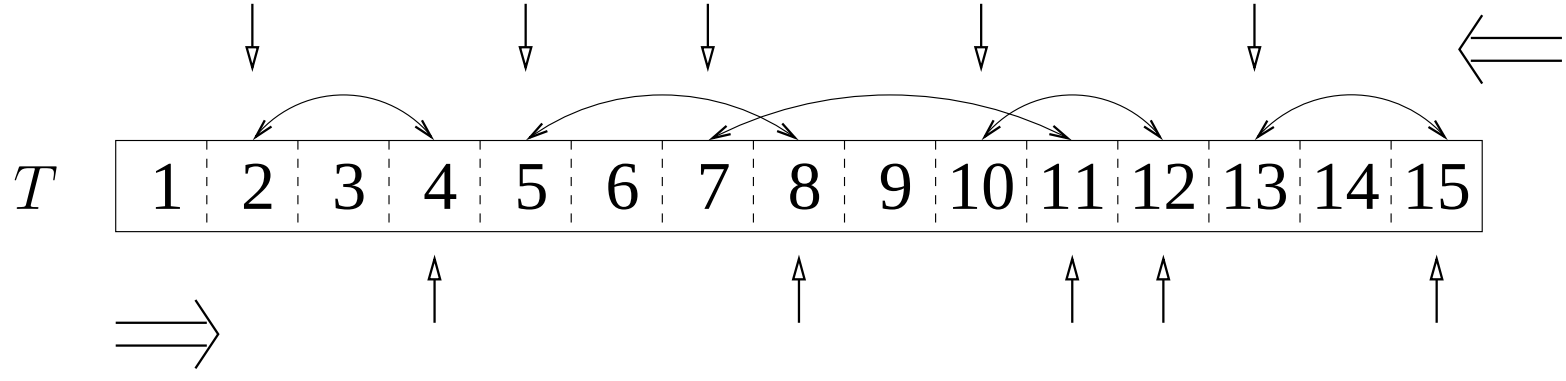
The Identification Problem

Given a regular expression E and a text T, we can identify all matching substrings of T that belong to $L(E)$ in $O(mn^2)$ worst-case time using $O(m)$ space.

Note that the algorithm of Myers et al. takes $O(mn \log n)$ time using $O(m \log n)$ space.

Infix-Free Regular-Expression Matching

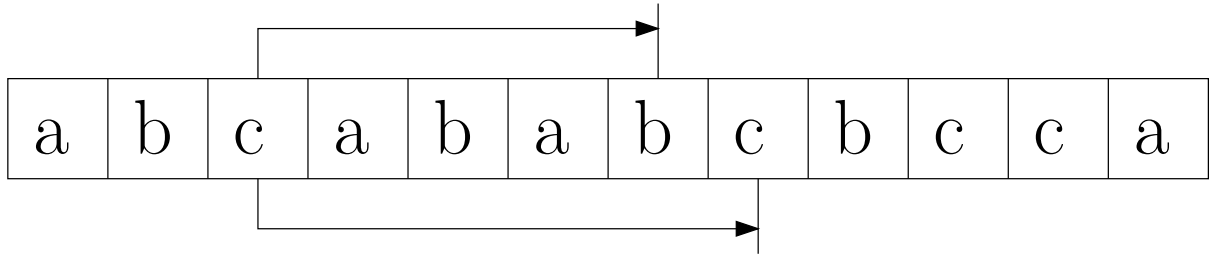
- $\mathcal{L}_{IN} \subsetneq \mathcal{L}_{PRE} \subsetneq \mathcal{L}_{REG}$



Given an infix-free regular expression E and a text T, we can identify all matching substrings of T that belong to $L(E)$ in $O(mn)$ worst-case time using $O(m)$ space.

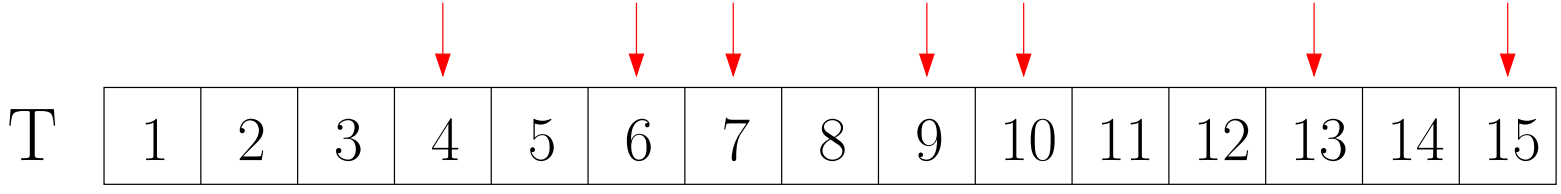
Prefix-Free Regular-Expression Matching

- $\mathcal{L}_{IN} \subsetneq \mathcal{L}_{PRE} \subsetneq \mathcal{L}_{REG}$
- If E is infix-free, we have an $O(mn)$ running time algorithm
- If E is a (normal) regular expression, we have an $O(mn^2)$ running time algorithm
- If E is prefix-free, there are at most n matching substrings of T that belong to $L(E)$, where n is the size of T



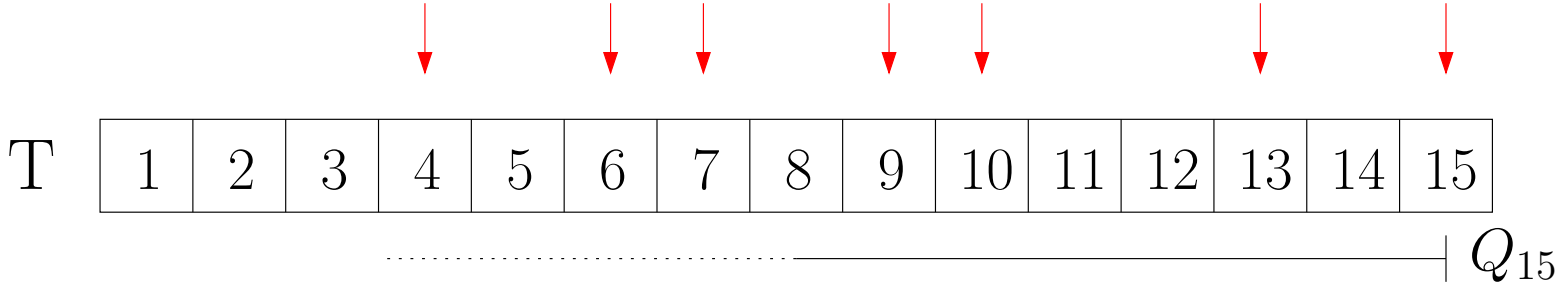
Prefix-Free Regular-Expression Matching

Given a prefix-free regular expression E and a text T, we find all end positions of matching substrings of T in $O(mn)$ time.



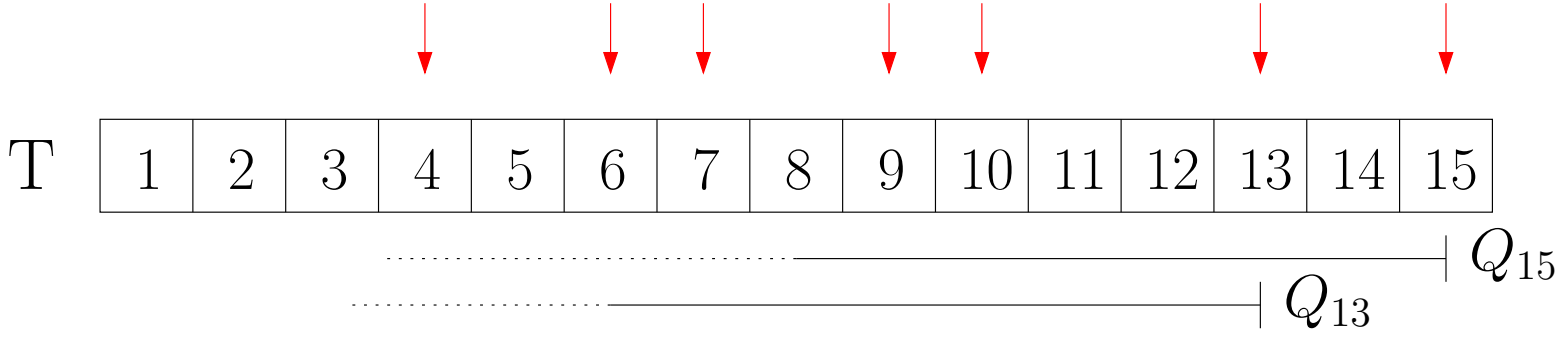
- Let $P = \{p_1, p_2, \ldots, p_k\}$ be the set of end positions of matching substrings for $k \neq n$
- Construct the Thompson automaton $A' = (Q, \Sigma, \delta', s', f')$ for E^R
- Scan $T^R = w_n \cdots w_1$ starting from the last position p_k in P to find the corresponding start position

Prefix-Free Regular-Expression Matching

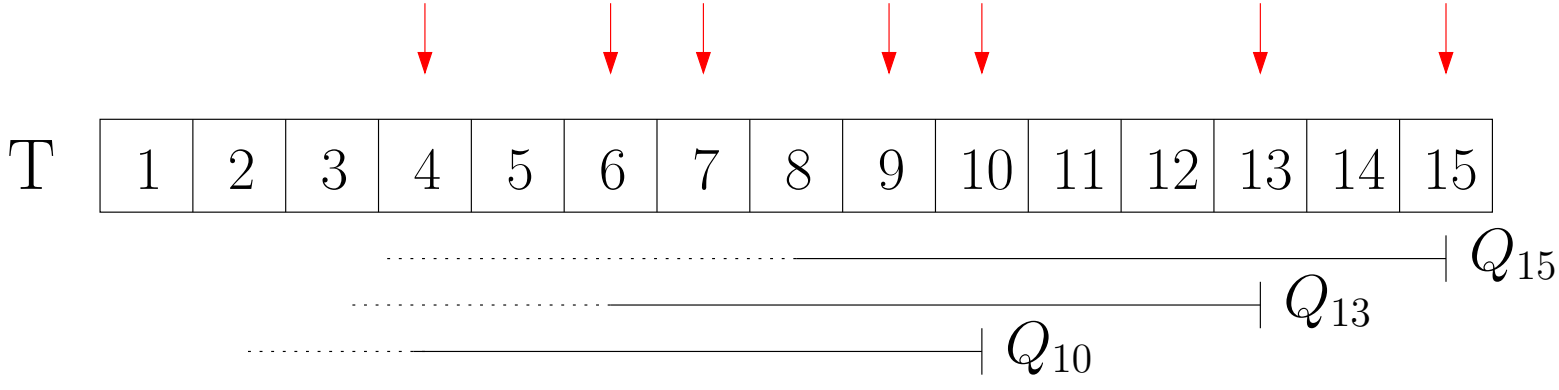


For a current input position i in T^R, Q_{15} is a set of states such that there is a path from s' to each state in Q_{15} that spells out $w_{15}w_{14}\cdots w_i$.
We keep reading T^R until we meet f'.

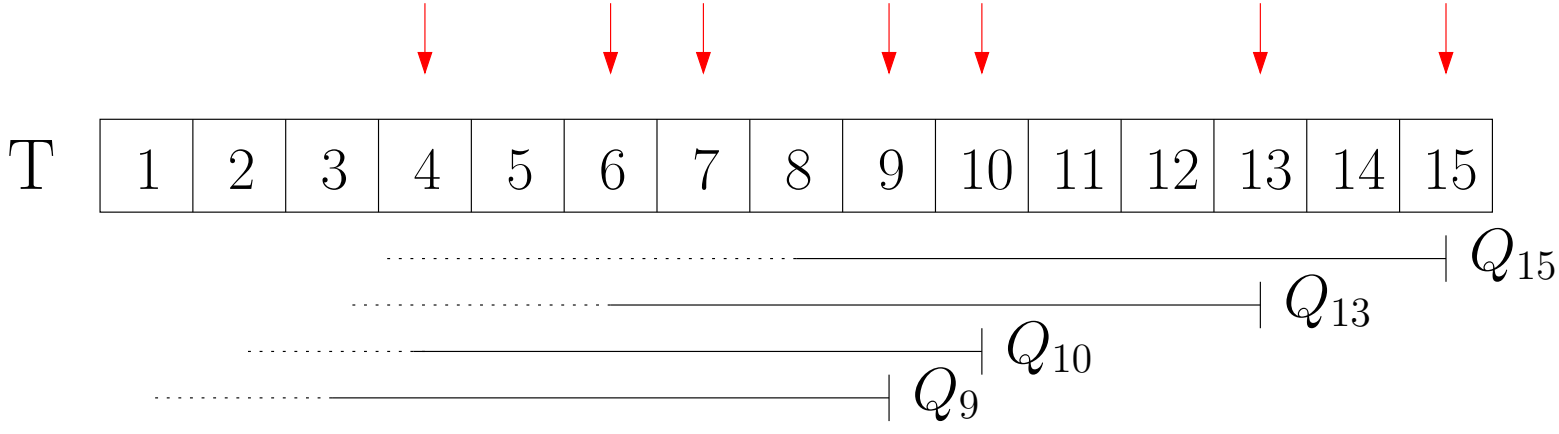
Prefix-Free Regular-Expression Matching



Prefix-Free Regular-Expression Matching



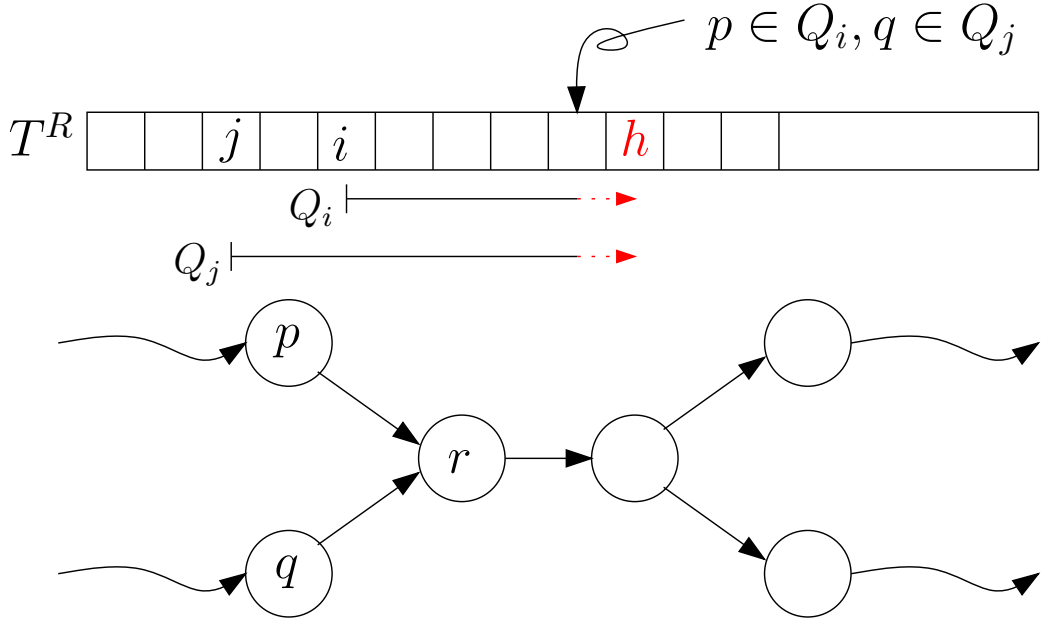
Prefix-Free Regular-Expression Matching



In the worst-case, there are k such sets of states and we need $O(km)$ time for each character of T to update these k sets. Thus, the total running time is $O(mn^2)$ in the worst-case since k is at most n.

Prefix-Free Regular-Expression Matching

If a state r in A' is reached from two different states p and q, where $p \in \mathcal{Q}_i$ and $q \in \mathcal{Q}_j$, when reading a character w_h in EM, where $h \leq i < j$, then both paths from p and q via r cannot reach f' by reading any prefix of the remaining input in EM.



Prefix-Free Regular-Expression Matching

If a state r in A' is reached from two different states p and q, where $p \in \mathcal{Q}_i$ and $q \in \mathcal{Q}_j$, when reading a character w_h in EM, where $h \leq i < j$, then both paths from p and q via r cannot reach f' by reading any prefix of the remaining input in EM.

- Each state in A' appears in at most one reachable set
- Any two sets of reachable states are disjoint
- We need at most $O(m)$ time to update all sets of reachable states simultaneously at each step

Given a prefix-free regular expression E and a text T, we can identify all matching substrings of T that belong to $L(E)$ in $O(mn)$ worst-case time using $O(m)$ space.

Prefix-Freeness

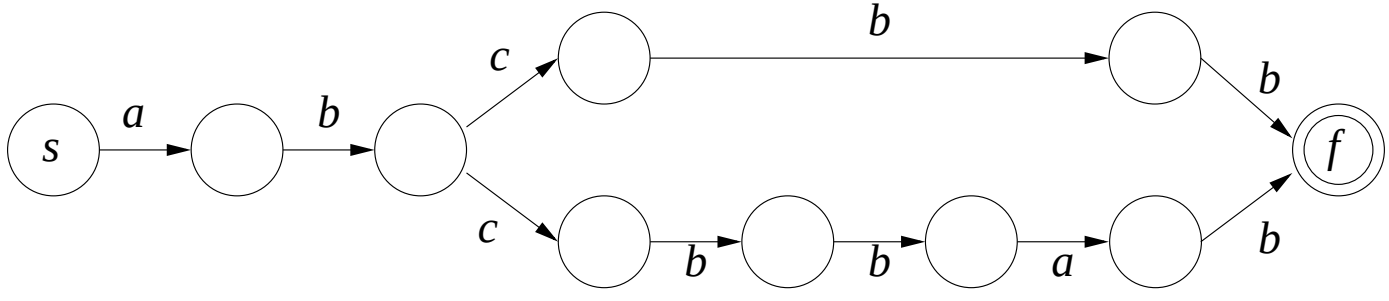
- An FA A is prefix-free if $L(A)$ is prefix-free
- A DFA A is prefix-free if it is non-exiting
- What about the NFA case?

Prefix-Freeness

- An FA A is prefix-free if $L(A)$ is prefix-free
- A DFA A is prefix-free if it is non-exiting
- What about the NFA case?
  - If an NFA A is prefix-free, then A must be non-exiting
  - However, the reverse does not hold

Prefix-Freeness

- An FA A is prefix-free if $L(A)$ is prefix-free
- A DFA A is prefix-free if it is non-exiting
- What about the NFA case?
  - If an NFA A is prefix-free, then A must be non-exiting
  - However, the reverse does not hold

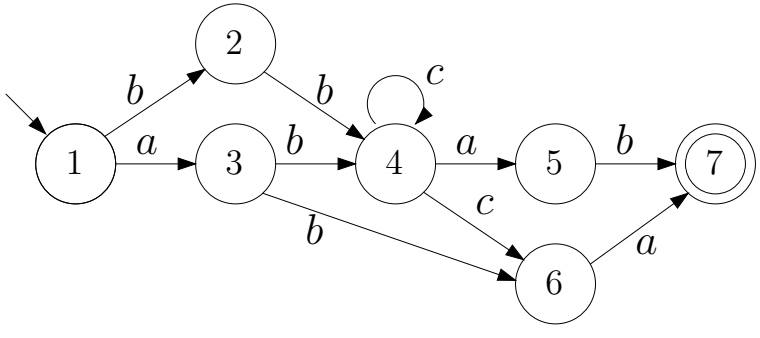


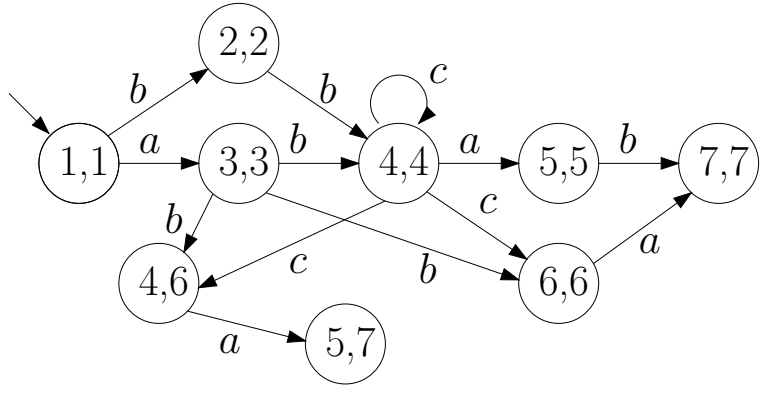
State-Pair Graph

Given a finite-state automaton $A = (Q, \Sigma, \delta, s, f)$, we define the state-pair graph $G_A = (V, E)$, where V is a set of nodes and E is a set of edges, as follows:

$$V = \{(i, j) \mid q_i \text{ and } q_j \in Q\} \text{ and}$$

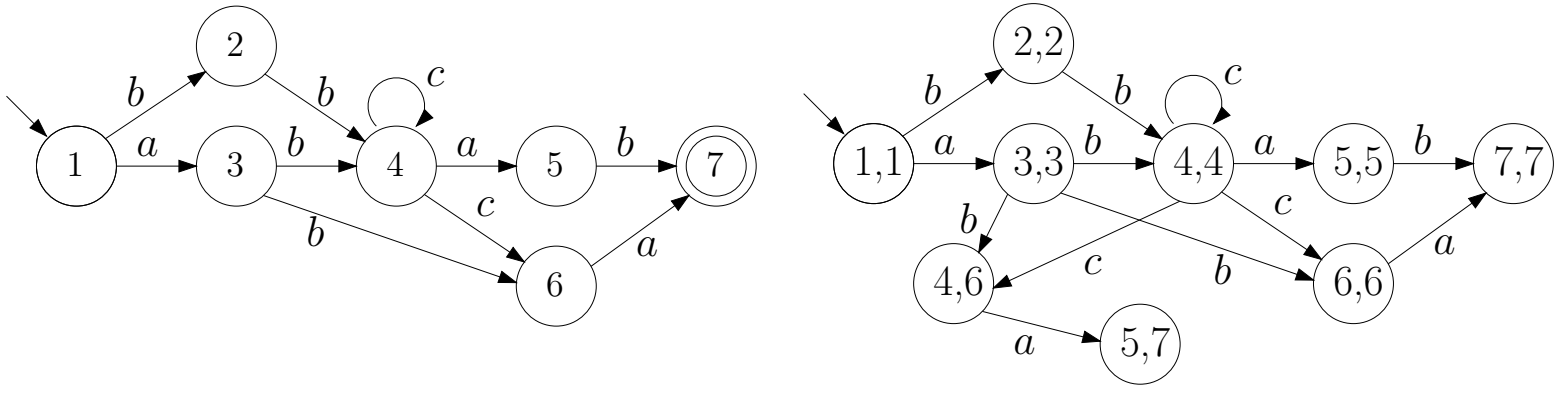
$$E = \{((i, j), a, (x, y)) \mid (q_i, a, q_x) \text{ and } (q_j, a, q_y) \in \delta \text{ and } a \in \Sigma\}.$$





State-Pair Graph & Prefix-Freeness

- Given a finite-state automaton A, $L(A)$ is prefix-free if and only if there is no path from $(1,1)$ to (m,j), for any $j \neq m$, in G_A



State-Pair Graph & Prefix-Freeness

- Given a finite-state automaton A, $L(A)$ is prefix-free if and only if there is no path from $(1,1)$ to (m,j), for any $j \neq m$, in G_A
- Given a finite-state automaton $A = (Q, \Sigma, \delta, s, f)$, we can determine whether or not $L(A)$ is prefix-free in $O(|Q|^2 + |\delta|^2)$ worst-case time
  - Let $G_A = (V, E)$ be the state-pair graph of A
  - $V = |Q|^2$
  - Let δ_i denote the set of out-transitions from state q_i in A
  - $|\delta| = \sum_{i=1}^{m} |\delta_i|$, where $m = |Q|$
  - a node (i,j) in G_A can have at most $|\delta_i| \times |\delta_j|$ out-transitions
  - $|E| = \sum_{i,j=1}^{m} |\delta_i| \times |\delta_j| \leq |\delta|^2$

State-Pair Graph & Prefix-Freeness

- Given a finite-state automaton A, $L(A)$ is prefix-free if and only if there is no path from $(1,1)$ to (m,j), for any $j \neq m$, in G_A
- Given a finite-state automaton $A = (Q, \Sigma, \delta, s, f)$, we can determine whether or not $L(A)$ is prefix-free in $O(|Q|^2 + |\delta|^2)$ worst-case time
- Given a regular expression E, we can determine whether or not $L(E)$ is prefix-free in $O(|E|^2)$ worst-case time
  - Construct the Thompson automaton for E
  - $|Q| = |\delta| = O(|E|)$

Conclusions

- Solve the prefix-free regular-expression matching problem in $O(mn)$ time using $O(m)$ space based on the Thompson automata
- Determine whether or not $L(A)$ is prefix-free for a given NFA A in polynomial time based on state-pair graphs