Complexity Analysis of Regular Expression Matching Based on Backtracking

Yasuhiko Minamide

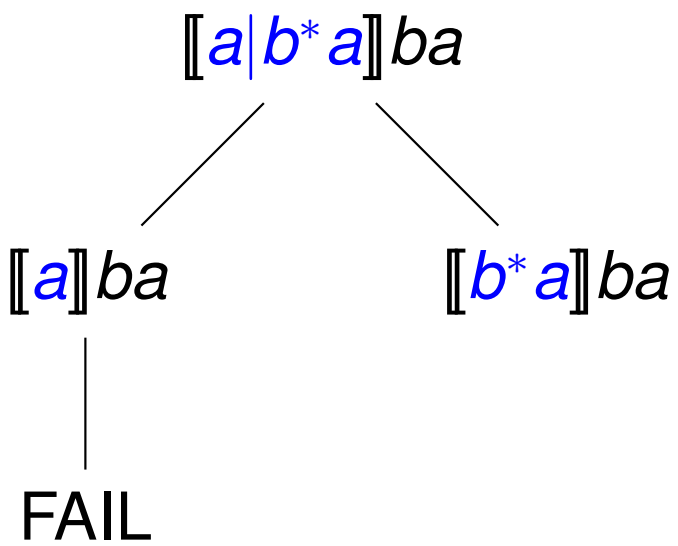
Tokyo Institute of Technology

Shonan Seminar

Matching Based on Backtracking

- Regular expression matching is implemented with backtracking in most programming languages.

Example: matching $a|b^*a$ with ba

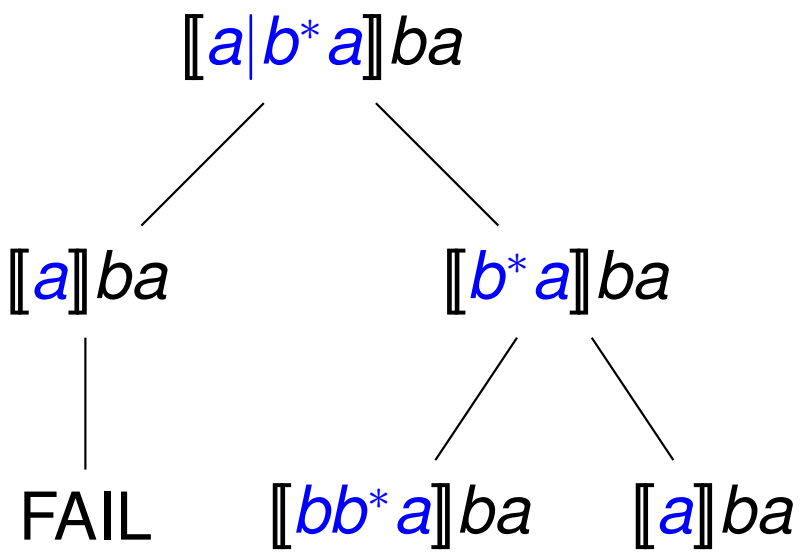


- $r_1|r_2$: r_1 has a higher priority.

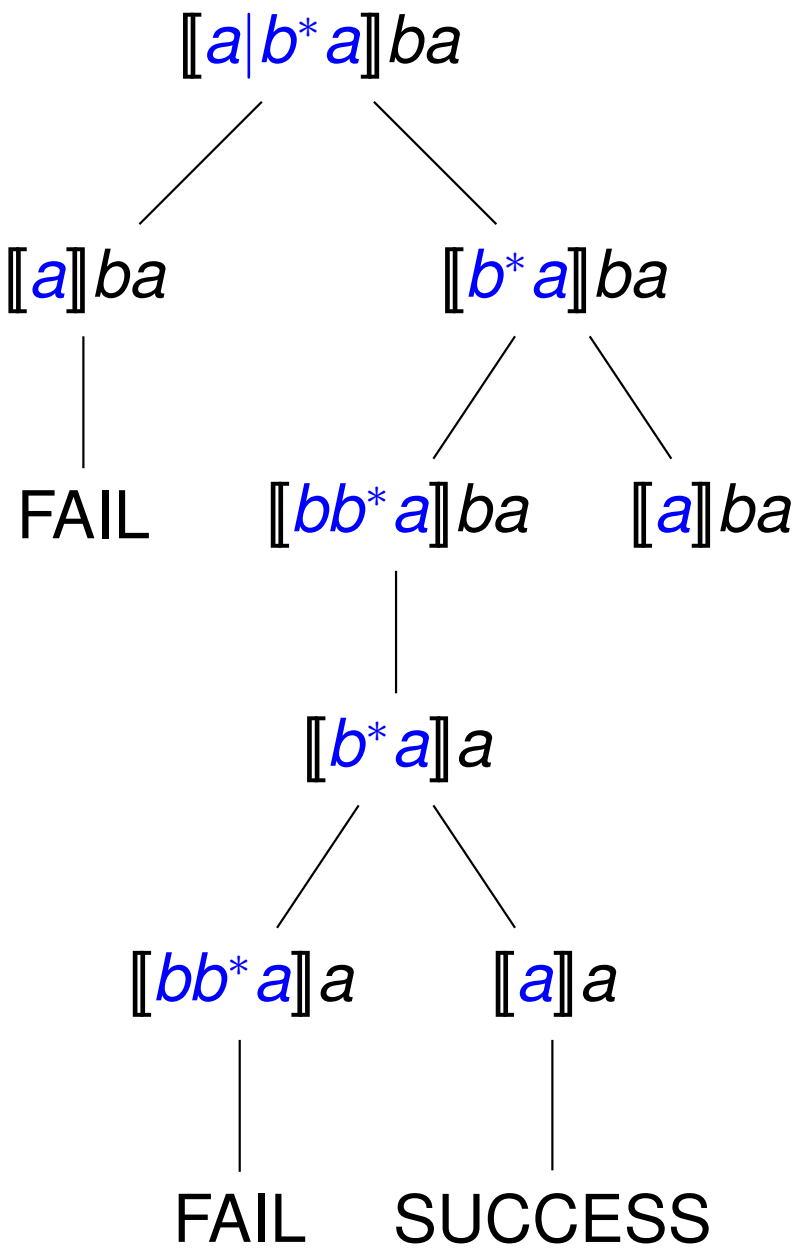
Matching Based on Backtracking

- Regular expression matching is implemented with backtracking in most programming languages.

Example: matching $a|b^*a$ with ba



- r^* is expanded to $rr^*|\epsilon$.



Running time

$\approx$

Size of search tree

Time Complexity

- Exponential in worst case
  - DoS vulnerabilities
  - may affect the result of matching in some implementations
    - limit on the number of matching steps (PCRE)
- Very hard to guess the complexity

Example:

$$\begin{array}{lcl} a^* & : & O(n) \\ a^*a^* & : & O(n^2) \\ (a|a)^*, (a^*)^* & : & O(2^n) \end{array}$$

Can you guess? (. (dot) matches any character)

$$\begin{array}{lcl} .^*.^* & : & O(?) \\ .^*a.^*a & : & O(?) \\ .^*a.^*b & : & O(?) \end{array}$$

Time Complexity

- Exponential in worst case
  - DoS vulnerabilities
  - may affect the result of matching in some implementations
    - limit on the number of matching steps (PCRE)
- Very hard to guess the complexity

Example:

$$
\begin{array}{lcl}
a^* & : & O(n) \\
a^*a^* & : & O(n^2) \\
(a|a)^*, (a^*)^* & : & O(2^n)
\end{array}
$$

Can you guess? (. matches any character)

$$
\begin{array}{lcll}
.^*.^* & : & O(n) & \text{(first branch succeeds)} \\
.^*a.^*a & : & O(n) & \\
.^*a.^*b & : & O(n^2) &
\end{array}
$$

Our Complexity Analysis

- Given regular expression r
- Precisely decide the time complexity of $[\![r]\!]w$:
  - $\Theta(n^i)$: the degree i is decided.
  - exponential

  where n is the length of w.

$$
\begin{aligned}
O(f(n)) &= \{g(n) \mid \exists c.\exists n_0.\forall n \geq n_0.g(n) \leq cf(n)\} \\
\Omega(f(n)) &= \{g(n) \mid \exists c.\exists n_0.\forall n \geq n_0.g(n) \geq cf(n)\} \\
\Theta(f(n)) &= O(f(n)) \cap \Omega(f(n))
\end{aligned}
$$

Our Complexity Analysis

- Given regular expression r
- Precisely decide the time complexity of $[\![r]\!]w$:
  - $\overset{\infty}{\Theta}(n^i)$: the degree i is decided.
  - exponential

$$\begin{aligned}
\overset{\infty}{\Omega}(f(n)) &= \{g(n) \mid \exists c.\forall n_0.\exists n \geq n_0.g(n) \geq cf(n)\} \\
&\qquad g(n) \geq cf(n) \text{ for infinitely many } n \\
\overset{\infty}{\Theta}(f(n)) &= O(f(n)) \cap \overset{\infty}{\Omega}(f(n))
\end{aligned}$$

Our Complexity Analysis

- Given regular expression r
- Precisely decide the time complexity of $[\![r]\!]w$:
  - $\overset{\infty}{\Theta}(n^i)$: the degree i is decided.
  - exponential

Why $\overset{\infty}{\Theta}(n^i)$?

- Consider $(\ .\ .\ )^*|a^*a^*$
  - Even length: $(\ .\ .\ )^*$ succeeds. $\Rightarrow \Theta(n)$
  - Odd length: $(\ .\ .\ )^*$ fails. $\Rightarrow \Theta(n^2)$

  Not in $\Theta(n^2)$, but in $\overset{\infty}{\Theta}(n^2)$.

Experimental Results

- 393 regular expressions obtained from popular Web programs.

order	# expressions
$O(n)$	338
$\overset{\infty}{\Theta}(n^2)$	44
$\overset{\infty}{\Theta}(n^3)$	6
timeout (900s)	5

Experimental Results

Our implementation generates a pattern of strings that shows the behavior of its order.

- pattern for $\Theta(n^i)$ $(i > 1)$

$$\overbrace{u_1 v_1^n u_2 v_2^n \cdots u_{i-1} v_{i-1}^n}^{i-1} w$$

- Pattern generated by our implementation
  - regular expression: `<image.*?>(.*?)</image>` $\in \overset{\infty}{\Theta}(n^3)$
  - pattern: `><imag (e><image)`^{n} `>` $\gt^n$ `>/imag`

Our Complexity Analysis: Internals

- Translate r into a topdown tree transducer T_r with regular lookahead
  - output a tree representing computation by backtracking
  - lookahead is used to encode depth-first search strategy
- Apply an extension of growth complexity analysis for topdown tree transducers [Aho & Ullman, 1971]

Example: Tree Transducer for Matching

$[\![a^*a^*]\!]$: state corresponding to regular expression a^*a^*.

- Transducer that searches all possible matches

$$\begin{aligned} [\![a^*a^*]\!](a(x)) &\to \mathrm{or}([\![a^*a^*]\!](x), [\![a^*]\!](x)) \\ [\![a^*]\!](a(x)) &\to \mathrm{or}([\![a^*a^*]\!](x), [\![\epsilon]\!](x)) \\ &\vdots \end{aligned}$$

- Transducer with regular lookahead simulating matching based on backtracking

$$\begin{aligned} [\![a^*a^*]\!](a(x)) &\overset{x\notin a^*}{\to} \mathrm{or}([\![a^*a^*]\!](x), [\![a^*]\!](x)) \\ [\![a^*a^*]\!](a(x)) &\overset{x\in a^*}{\to} \mathrm{lft}([\![a^*a^*]\!](x)) \end{aligned}$$

  - if $x \in a^*$, $[\![a^*a^*]\!](x)$ succeeds and $[\![a^*]\!](x)$ is not executed.

Growth Complexity Analysis [Aho & Ullman, 1971]

Definition [growth function]
For a topdown (string-to-tree) transducer T

$$\mathcal{S}_T(n) = \max_{w \in \Sigma^n} |T(w)|$$

(for simplicity, we consider string-to-tree transducers)

Theorem
We can precisely decide the growth complexity of transducer T.

- $\mathcal{S}_T(n) \in \Theta(n^i)$: the degree i is decided.
- $\mathcal{S}_T(n)$ is exponential

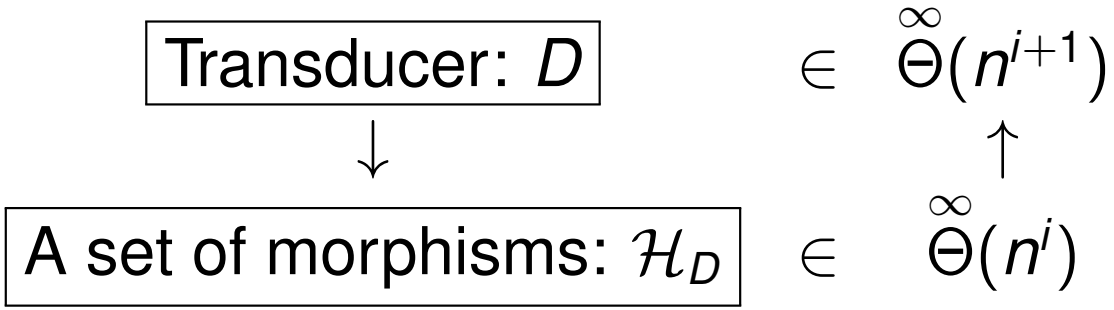
Aho and Ullman, Translations on a context free grammar Information and Control, 19(5), 1971

Our Revision: Growth Complexity Analysis

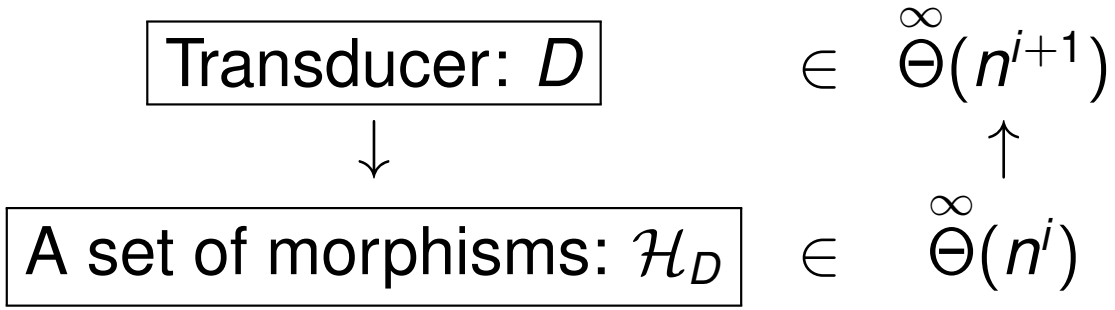
We can precisely decide the growth complexity of $\mathcal{S}_T(n)$.

- Aho & Ullman: topdown tree transducers
  - $\Theta(n^i)$: the degree i is decided.
  - exponential
- Our Revision: topdown tree transducers with regular lookahead
  - $\overset{\infty}{\Theta}(n^i)$: the degree i is decided.
  - exponential

Outline of Growth Complexity Analysis



Outline of Growth Complexity Analysis



- $\mathcal{H}_D = \{h_a \mid a \in \Sigma\}$
- $h_a : Q \to Q^*$
  - Transition rule:

$$q_1(a(x)) \to \mathrm{or}(q_1(x), \mathrm{or}(q_2(x), q_1(x)))$$

$$q_2(a(x)) \to \mathrm{fail}$$

  - Morphism:

$$h_a(q_1) = q_1 q_2 q_1, \qquad h_a(q_2) = \epsilon$$

Proliferation Rate of Morphisms

Let $\mathcal{H}$ be a finite set of moprhisms over Q.

Definition: proliferation rate of $q \in Q$

$$\mathcal{S}_{\mathcal{H},q}(n) = \max_{\alpha \in \mathcal{H}^n} |\alpha(q)|$$

Theorem

The growth complexity can be decided by the order of the proliferation rate of the initial state q_0.

Morphisms: $\mathcal{S}_{\mathcal{H}_D,q_0}$	Transducer: $\mathcal{S}_D$
$\overset{\infty}{\Theta}(0)$	$\overset{\infty}{\Theta}(1)$
$\overset{\infty}{\Theta}(n^i)$	$\overset{\infty}{\Theta}(n^{i+1})$
exponential	exponential

Pumping Lemma for Morphisms

Definition [Pumpable] (Aho & Ullman 1971, Engelfriet & Maneth 2003)

q is pumpable if there exist $\alpha, \beta \in \mathcal{H}^*$ and $q_1, q_2 \in Q$ s.t.

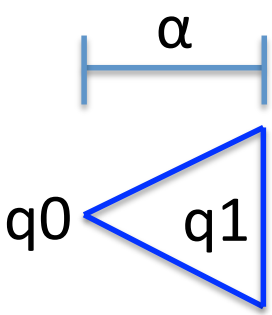
- $q_1 \in \alpha(q)$
- $q_1, q_2 \in \beta(q_1)$
  - q_1, q_2 must appear at different positions in $\beta(q_1)$.
- $q_2 \in \beta(q_2)$

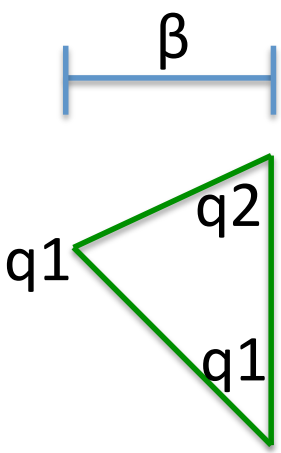
Proposition

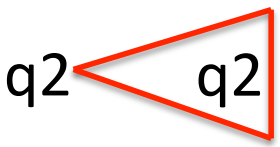
- q is pumpable $\Longrightarrow q \in \overset{\infty}{\Omega}(n)$
- $q \notin O(1) \Longrightarrow q$ is pumpable
  - q is not pumpable $\Longrightarrow q \in O(1)$

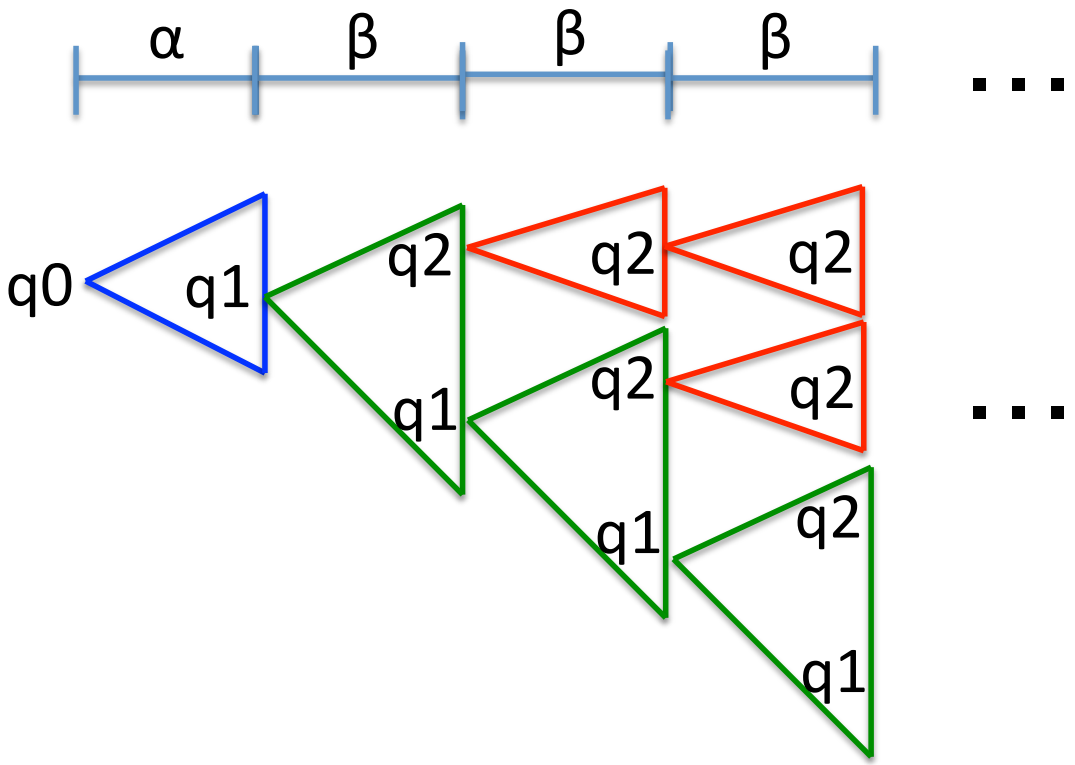
(we write $q \in O(f(n))$ for $\mathcal{S}_{\mathcal{H},q}(n) \in O(f(n))$)

q_0 is pumpable $\Longrightarrow q_0 \in \overset{\infty}{\Omega}(n)$









$$|\beta^n(\alpha(q_0))| > n$$

$$\Rightarrow q_0 \in \overset{\infty}{\Omega}(n)$$

Higher Degree: $\overset{\infty}{\Theta}(n^i)$

Restriction $\mathcal{H}_Q$:

$$\mathcal{H}_Q = \{\iota_Q \circ h \mid h \in \mathcal{H}\}$$

$$\iota_Q(q) = \begin{cases} q & \text{if } q \in Q \\ \epsilon & \text{otherwise} \end{cases}$$

Lemma Let $Q = Q_1 \cup Q_2$ and $q \in Q_2$

- $Q_1 = \{q \mid q \in O(n^{i-1})\}$ and $Q_2 = \{q \mid q \in \overset{\infty}{\Omega}(n^i)\}$

Then,

- If q is not pumpable in $\mathcal{H}_{Q_2}$, then $q \in O(n^i)$.
- If q is pumpable in $\mathcal{H}_{Q_2}$, then $q \in \overset{\infty}{\Omega}(n^{i+1})$.

Transducer with Regular Lookahead

Transducer with Regular Lookahead

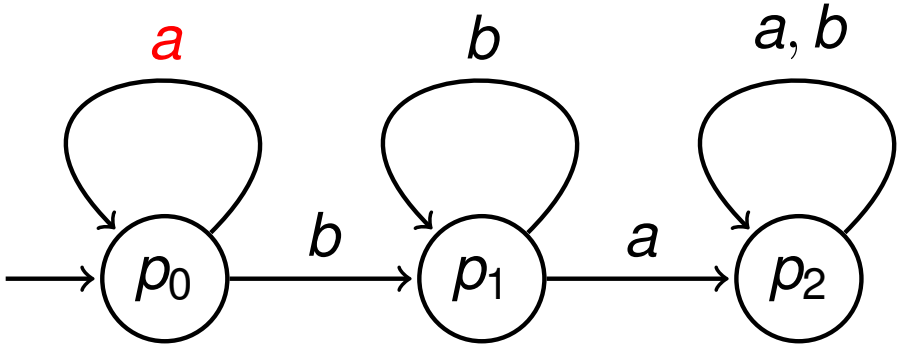
- Lookahead automaton: $A = (P, \Sigma, \delta, p_0)$.
- Transition rule: $q(a(x)) \xrightarrow{p} t$
  - The rule is taken if $\hat{\delta}(p_0, x^{\mathrm{rev}}) = p$.

Morphism:

- $h_{a,p}(q)$ is derived from $q(a(x)) \xrightarrow{p} t$
- $\mathcal{H}_{p_1,p_2} = \{h_{a,p_2} \mid \delta(p_2, a) = p_1\}$
- $\mathcal{H}_{p_1,p_2} \circ \mathcal{H}_{p_2',p_3}$ is only defined when $p_2 = p_2'$.

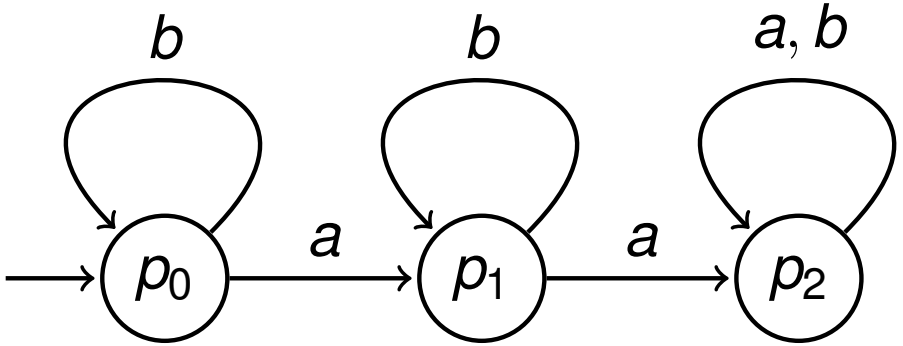
- $.^*a.^*b \in \overset{\infty}{\Theta}(n^2)$

$$[\![.^*a.^*b]\!](a(x)) \xrightarrow{p_0} \mathrm{or}([\![.^*a.^*b]\!](x), [\![.^*b]\!](x))$$
$$[\![.^*b]\!](a(x)) \xrightarrow{p_0} [\![.^*b]\!](x)$$



- $.^*a.^*a \in \overset{\infty}{\Theta}(n)$

$$[\![.^*a.^*a]\!](a(x)) \xrightarrow{p_0} \mathrm{or}([\![.^*a.^*a]\!](x), [\![.^*a]\!](x))$$
$$[\![.^*a]\!](a(x)) \xrightarrow{p_0} \mathrm{or}([\![.^*a]\!](x), [\![\epsilon]\!](x))$$



Reference

- Checking Time Linearity of Regular Expression Matching Based on Backtracking, Satoshi Sugiyama, Yasuhiko Minamide, IPSJ Transactions on Programming, 7(3), 2014.
  `www.is.titech.ac.jp/~minamide/papers.html`
- Static Analysis for Regular Expression Exponential Runtime via Substructural Logics, Asiri Rathnayake, Hayo Thielecke
  `www.cs.bham.ac.uk/~hxt/research/rxxr2/`