# Fast Regular Expression Matching Based On Dual Glushkov NFA

Ryutaro Kurai, Norihito Yasuda, Hiroki Arimura, Shinobu Nagayama and Shin-Ichi Minato

PSC 2014
Prague, Czech Republic, September 1–3, 2014

# Outline

- Research Definition
- Regular Expression Matching with NFA
  - Previous works
  - Look-Ahead(LA) Matching
  - Glushkov NFA, and Dual Glushkov NFA
- LA Matching by Using Dual Glushkov NFA
  - Memory Space Analysis
  - Experimental Results

# Outline

- Research Definition
- Regular Expression Matching with NFA
  - Previous works
  - Look-Ahead(LA) Matching
  - Glushkov NFA, and Dual Glushkov NFA
- LA Matching by Using Dual Glushkov NFA
  - Memory Space Analysis
  - Experimental Results

# Research Definition and Purpose

- Regular Expression(RE) Matching Problem
  - Searching substring *s* that matches to the pattern *p* of regular expression *R* from given text *T*.
  - Example
    - R = ‘aa(a|b)’
    - T = ‘**aab**ababb**aaa**’
- Purpose of the Research
  - Develop high-speed and space efficient matching method for RE Matching to process huge amount of data.
  - Investigate efficient algorithm for RE Matching.

# Targeted Application of Our Research

```
^Remote-Party-ID\x3A\s+[^\r\n]+\x40[^\r\n]*?[\x80-\xFF]
^Authorization\x3A[^\r\n]+?response=[\x00-\x09\x0B\x0C\x0E-
\x7F]*[\x80-\xFF]
^Date\x3A[^\r\n]*[\x2D\x2B]
^Content-Type[^\r\n]+[\x01-\x08\x0B\x0C\x0E-\x1F\x80-\xFF]
^Content-Type\x3A\s*[^\r\n%]*%
^Contact\x3a\s+\x22[^\x22]*\x3c
…
```

- Network intrusion detection systems.
- Using huge size of RE patterns.

- Above patterns come from ‘Snort’.
- Each line has about 100 symbols.
- Each patterns will be joined by conjunction symbol ‘|’.
- Number of lines is about 6,000

# Major Approaches To The RE Matching Problem

| Method | Pros | Cons |
|---|---|---|
| Backtracking | Can use back reference<br>Good space efficiency | Slows down for some bad pattern (when it backtracks many times) |
| Deterministic Finite Automata (DFA) | Fast matching speed | Use exponential memory space $O(2^m)$, m is the size of RE pattern |
| Nondeterministic Finite Automata (NFA) | Good space efficiency<br>Matches fast enough | Slows in case active states grow |

# Major Approaches To The RE Matching Problem

| Method | Pros | Cons |
|---|---|---|
| Backtracking | Can use back reference<br>Good space efficiency | Slows down for some bad pattern (when it backtracks many times) |
| Deterministic Finite Automata (DFA) | Fast matching speed | Use exponential memory space $O(2^m)$, m is the size of RE pattern |
| Nondeterministic Finite Automata (NFA) | Good space efficiency<br>Matches fast enough | Slows in case active states grow |

**We have to Control growth of active states!**

# Outline

- Research Definition
- Regular Expression Matching with NFA
  - Previous works
  - Look-Ahead(LA) Matching
  - Glushkov NFA, and Dual Glushkov NFA
- LA Matching by Using Dual Glushkov NFA
  - Memory Space Analysis
  - Experimental Results

# Nondeterministic Finite Automata

- NFA is defined as following 5-tuple.
  - a finite set of states *E*
  - a finite set of input symbols *Σ*
  - a transition function σ
  - a finite set of initial states *I*
  - a finite set of final states *F*

# Previous Research on Improving RE Matching Using NFA

- Bit parallel techniques
  - S.Wu and U. Manber, 92
  - G.Navarro and M.Raffinot, 99
  - Efficient for the patterns that is smaller than word size of processors.
- Multi-stride/multi-character NFA
  - B. Brodie et al. 06
  - Transitions are labeled by multi symbols.
- Look-ahead Matching
  - Well known remedy.

# Outline

- Research Definition
- Regular Expression Matching with NFA
  - Previous works
  - Look-Ahead(LA) Matching
  - Glushkov NFA, and Dual Glushkov NFA
- LA Matching by Using Dual Glushkov NFA
  - Memory Space Analysis
  - Experimental Results

# Regular expression example

- pattern ‘a’
  - matches to text “a”
- pattern ‘ab’ (concatenation)
  - matches to text “ab”
- pattern ‘(a|b)’ (disjunction)
  - matches to text “a” or “b”
- pattern ‘a*’ (cline closure)
  - matches to text “” , “a”, “aa”, or “aa…a”
- combination of above parts
  - pattern ‘(ab|ac)*’ matches to “ab”, “ac” or “abac…”

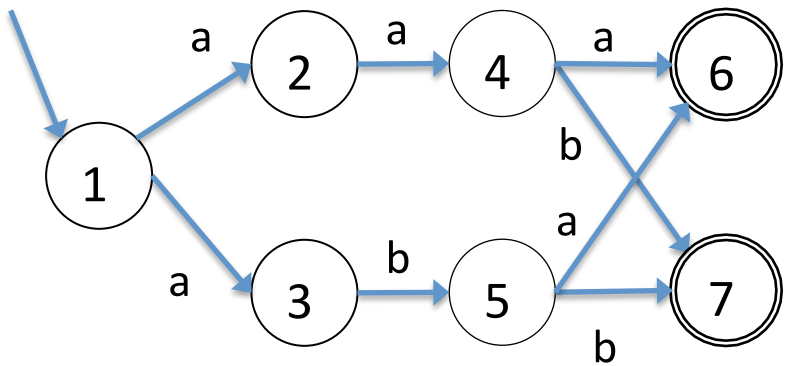
# Look-Ahead Matching Method for Regular Expression

- NFA treats multiple states as active state.
- Each active state needs simulation of transition. It costs almost all of calculation time.
- Therefore, we want to decrease active states.

- Main Idea
  - Make state active if and only if the state have transition that use next input symbol.
  - If we use such transition, we only use states that can transit in connected next state. This property can decrease active states.

# Normal NFA Matching Example

R = (aa|ab)(a|b)

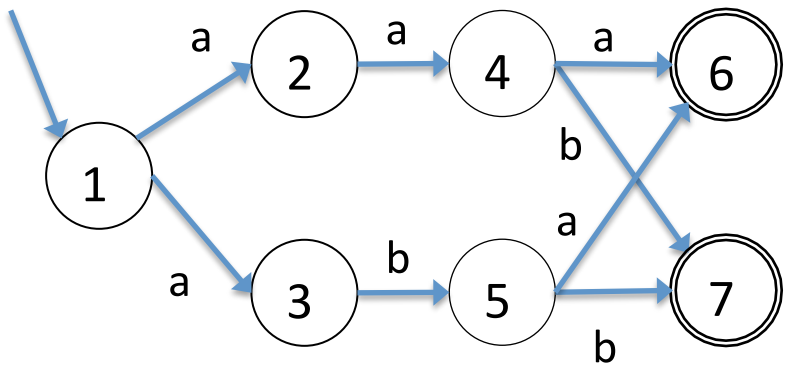



| $t_1$ | source | destination |
|---|---|---|
| a | 1 | 2 |
| a | 1 | 3 |
| a | 2 | 4 |
| b | 3 | 5 |
| a | 4 | 6 |
| b | 4 | 7 |
| a | 5 | 6 |
| b | 6 | 7 |

- Input text: “aab…”
  - We activate state No. 2. and No. 3
  - Then we activate states No. 4 but No. 5.
- Input text: “adb..”
  - we activate state No. 2. and No.3
  - But, for next symbol, we can not activate any state.

# Look-Ahead Matching Example

R = (aa|ab)(a|b)

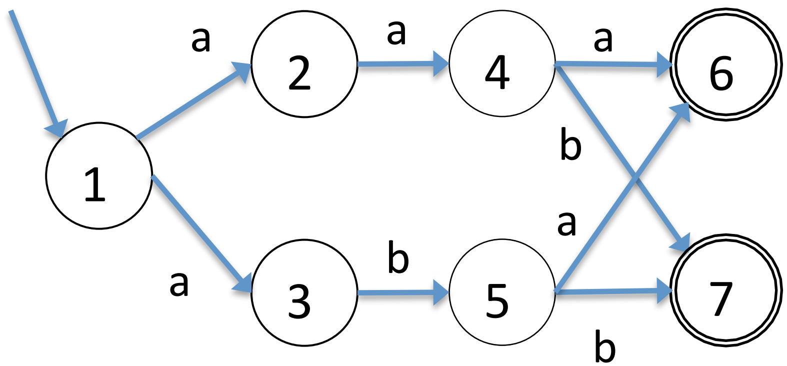



- Input text: "aab…"
  - We activate state only No. 2. # of active state is decreased.
  - Then we activate states No. 4
- Input text: "adb.."
  - Since 'd' is not present in $t_2$ column, no state activated.

| $t_1$ | $t_2$ | source | destination |
|---|---|---|---|
| a | a | 1 | 2 |
| a | b | 1 | 3 |
| a | a | 2 | 4 |
| a | b | 2 | 4 |
| b | a | 3 | 5 |
| b | b | 3 | 5 |
| a | * | 4 | 6 |
| b | * | 4 | 7 |
| a | * | 5 | 6 |
| b | * | 6 | 7 |

# Look-Ahead Matching Example

$R = (aa|ab)(a|b)$




- LA matching can decrease number of active states.
- LA matching can give up matching in early timing.
- But, it use extra memory space for extended transition table.

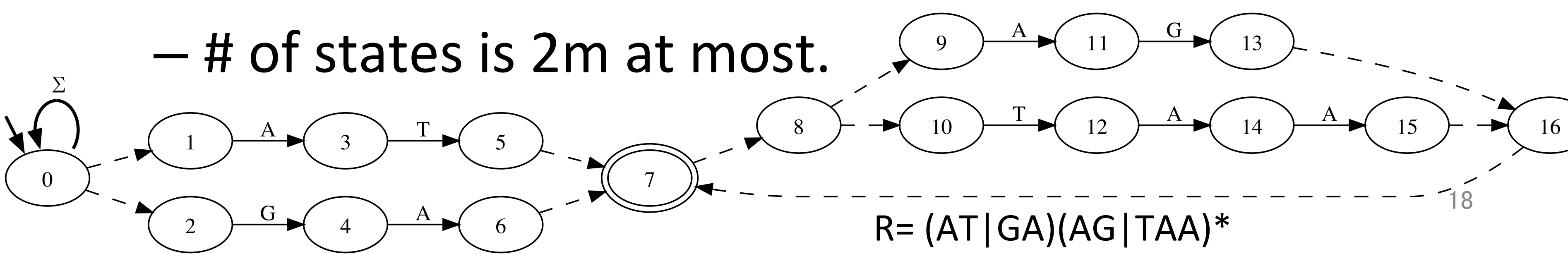
| $t_1$ | $t_2$ | source | destination |
|---|---|---|---|
| a | a | 1 | 2 |
| a | b | 1 | 3 |
| a | a | 2 | 4 |
| a | b | 2 | 4 |
| b | a | 3 | 5 |
| b | b | 3 | 5 |
| a | * | 4 | 6 |
| b | * | 4 | 7 |
| a | * | 5 | 6 |
| b | * | 6 | 7 |

# Outline

- Research Definition
- Regular Expression Matching with NFA
  - Previous works
  - Look-Ahead(LA) Matching
  - Glushkov NFA, and Dual Glushkov NFA
- LA Matching by Using Dual Glushkov NFA
  - Space Analysis
  - Experimental Results

# Thompson NFA (T-NFA)

- NFA that transformed from regular expression by Thompson's method.
- Thompson's method convert regular expression syntax tree into partial NFA inductively.
- It has following property
  – It has epsilon transition
  – # of transitions is 4m at most .
  – # of states is 2m at most.

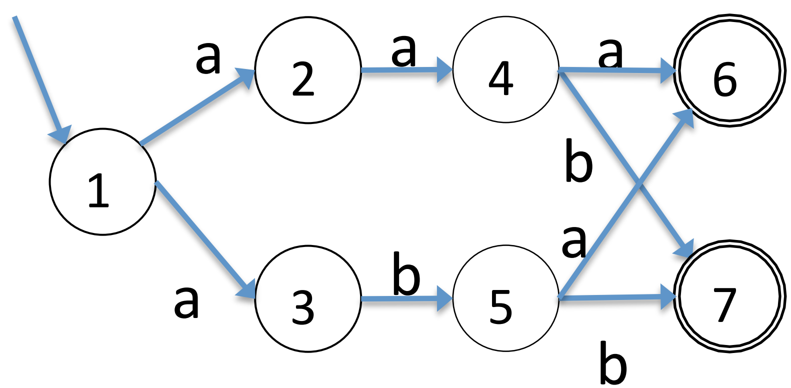



R= (AT|GA)(AG|TAA)*

# Glushkov NFA (G-NFA)

- NFA that transformed from regular expression.
- Glushkov NFA has following properties.
  - It has no ε-transitions.
  - Incoming transitions are labeled by the same symbol.
  - It has only one initial state.
  - It has one or more final states.

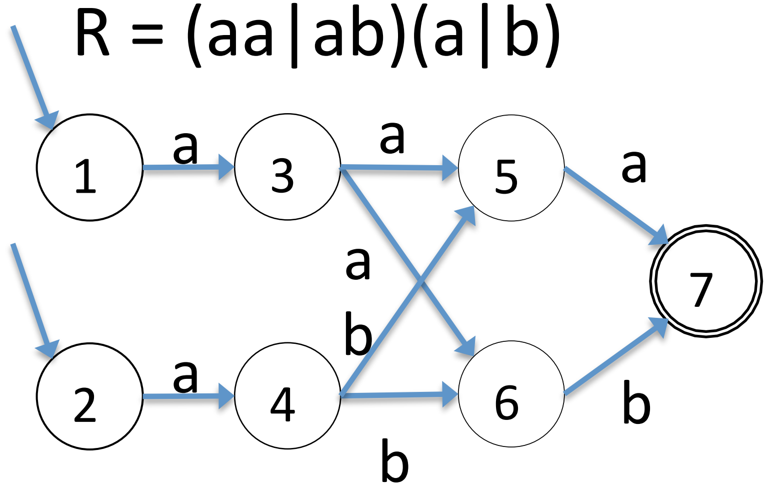
R = (aa|ab)(a|b)



# Dual Glushkov NFA

- NFA with dual property of Glushkov NFA.
  - no ε-transitions.
  - Outgoing transitions are labeled by the same symbol.
  - It has only one final state.
  - It has one or more initial states.

R = (aa|ab)(a|b)



# Outline

- Research Definition
- Regular Expression Matching with NFA
  - Previous works
  - Look-Ahead(LA) Matching
  - Glushkov NFA, and Dual Glushkov NFA
- LA Matching by Using Dual Glushkov NFA
  - Memory Space Analysis
  - Experimental Results

# Proposed Method

- Look-Ahead Matching by Using Dual Glushkov NFA

- To solve the problem of consumption of extra memory space by the Look-Ahead matching method, we propose the approach of Dual G-NFA to reduce memory space consumption.

- If we use Look-Ahead Matching by dual G-NFA, we can simulate NFAs without increasing the size of transition tables.

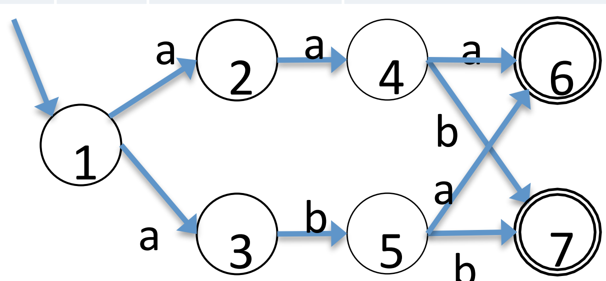
# Memory Space Comparison

## Glushkov NFA

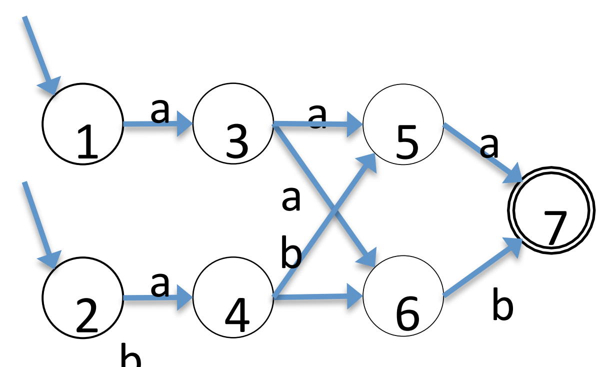
| $t_1$ | $t_2$ | source | destination |
|---|---|---|---|
| a | a | 1 | 2 |
| a | b | 1 | 3 |
| a | a | 2 | 4 |
| a | b | 2 | 4 |
| b | a | 3 | 5 |
| b | b | 3 | 5 |
| a | * | 4 | 6 |
| b | * | 4 | 7 |
| a | * | 5 | 6 |
| b | * | 6 | 7 |

## Dual Glushkov NFA

| $t_1$ | $t_2$ | source | destination |
|---|---|---|---|
| a | a | 1 | 3 |
| a | b | 2 | 4 |
| a | a | 3 | 5 |
| a | b | 3 | 6 |
| b | a | 4 | 5 |
| b | b | 4 | 6 |
| a | * | 5 | 7 |
| b | * | 6 | 7 |




$$R = (aa|ab)(a|b)$$



# Memory Space Comparison

Glushkov NFA

| $t_1$ | $t_2$ | source | destination |
|---|---|---|---|
| a | a | 1 | 2 |
| a | b | 1 | 3 |
| a | a | 2 | 4 |
| a | b | 2 | 4 |
| b | a | 3 | 5 |
| b | b | 3 | 5 |

Dual Glushkov NFA

| $t_1$ | $t_2$ | source | destination |
|---|---|---|---|
| a | a | 1 | 3 |
| a | b | 2 | 4 |
| a | a | 3 | 5 |
| a | b | 3 | 6 |
| b | a | 4 | 5 |
| b | b | 4 | 6 |

- The size of transition table of Glushkov NFA is $O(|E||\Sigma|)$
- The size of Dual Glushkov NFA is $O(|E|)$
- Our method use memory space by $|\Sigma|$ times smaller.

# Look-Ahead Matching by Using Dual Glushkov NFA

- The size of transition table of Glushkov NFA is $O(|E||\Sigma|)$
- The size of Dual Glushkov NFA is $O(|E|)$
- Our method use memory space by $|\Sigma|$ times smaller.

# Outline

- Research Definition
- Regular Expression Matching with NFA
  - Previous works
  - Look-Ahead(LA) Matching
  - Glushkov NFA, and Dual Glushkov NFA
- LA Matching by Using Dual Glushkov NFA
  - Memory Space Analysis
  - Experimental Results

# Experimental Settings

- Method: match 3 types of regular expression patterns with one text data.
- The text data used in the experiments is "English.100MB" from Pizza & Chili corpus.

- We compared proposed method (Dual G-NFA with LA) with Dual G-NFA, G-NFA, G-NFA with LA, Thompson NFA and NR-grep[Navarro 01].

- Each experiment is iterated 10 times, and the average elapsed time is recorded.
- The elapsed time and average size of active states are compared.

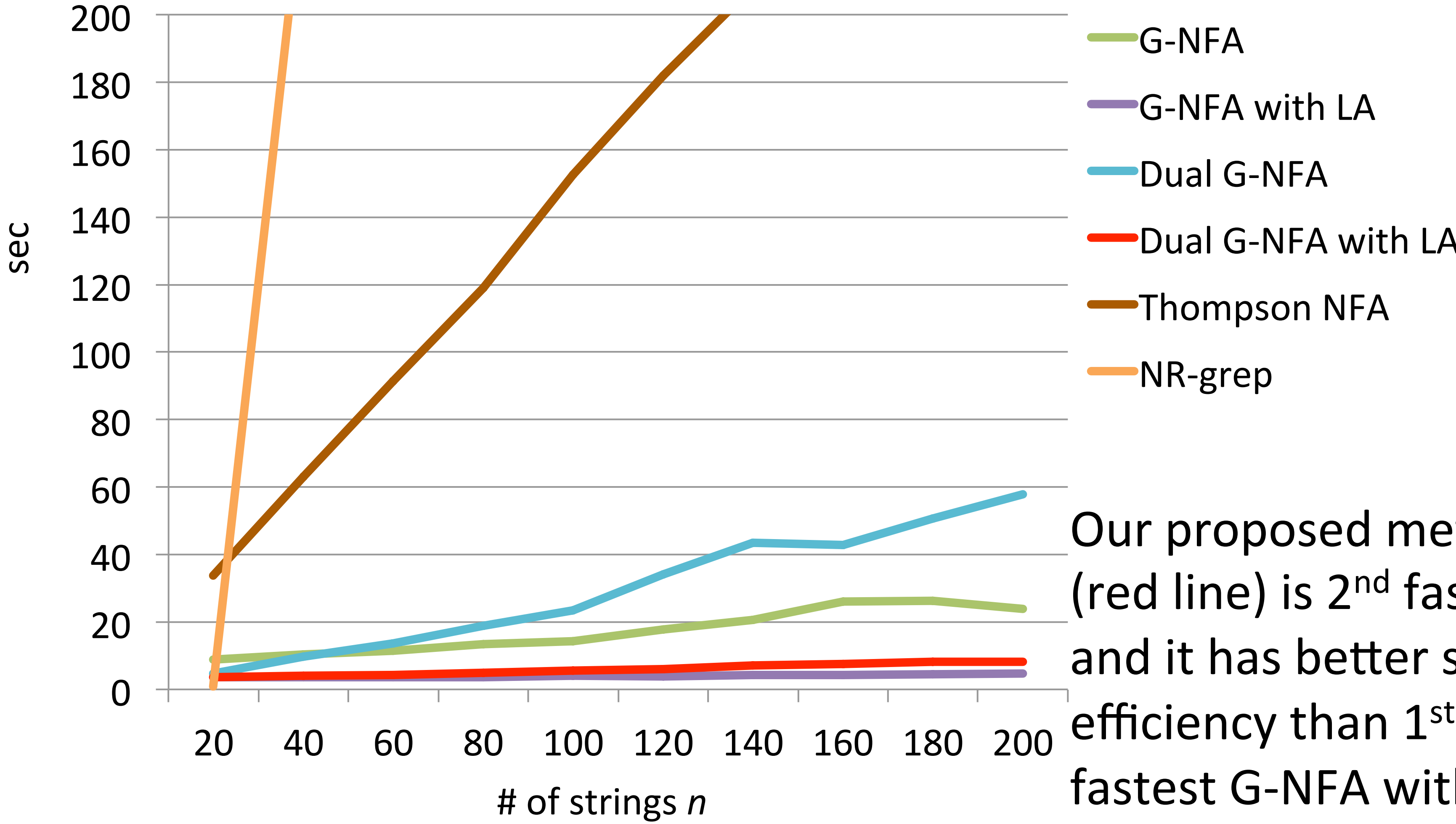
# Fixed pattern

- Pattern
  - n fixed strings are joined by disjunction symbol.
  - n fixed strings are randomly collected from /usr/share/dict/words on Mac OS X 10.9

- Example

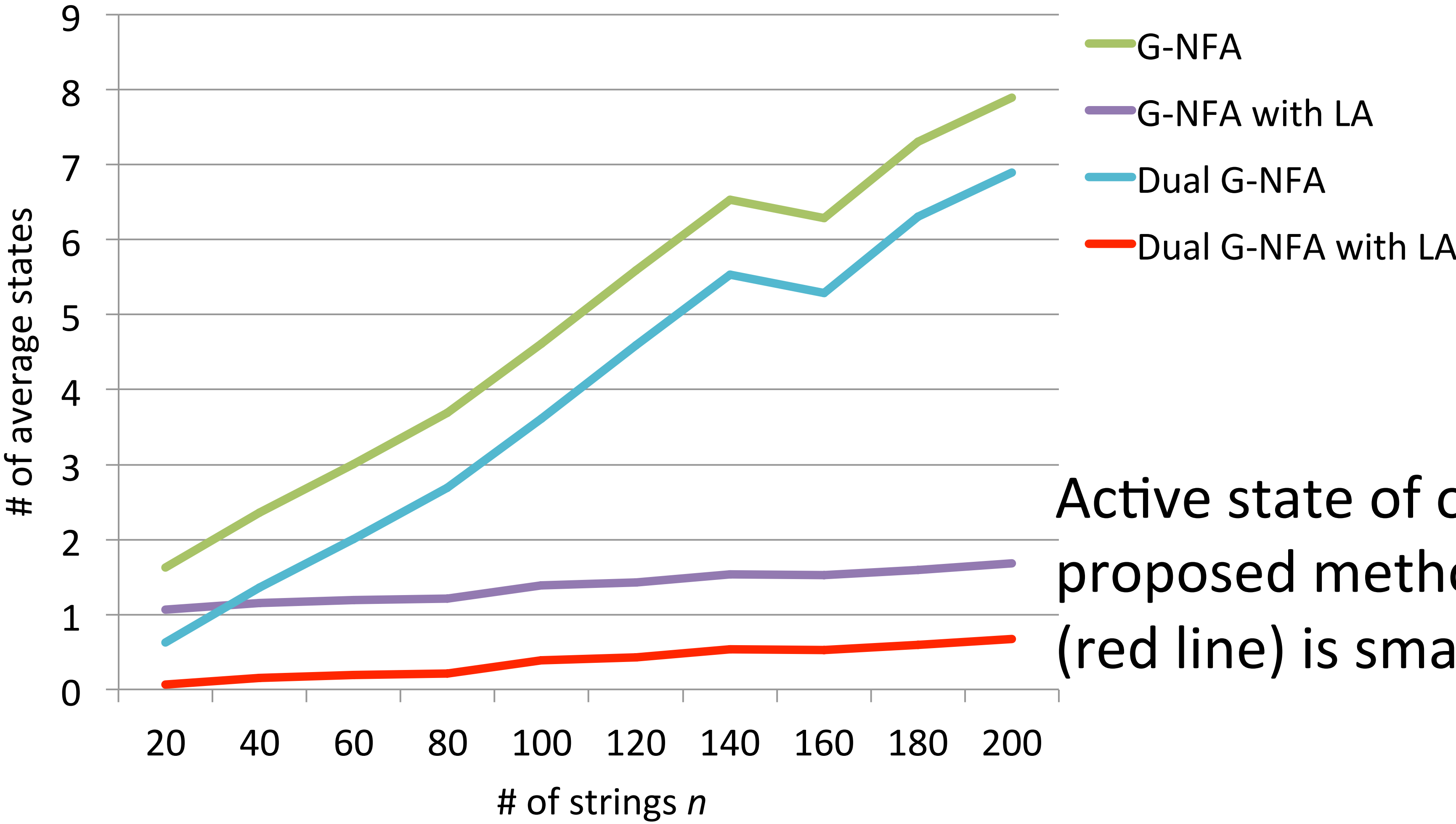
  R = alpha|blabo|chary|delta|…|Juliet

# Results: Elapsed Time(sec)




Our proposed method (red line) is 2nd fastest, and it has better space efficiency than 1st fastest G-NFA with LA

# Results: # of Average Active States

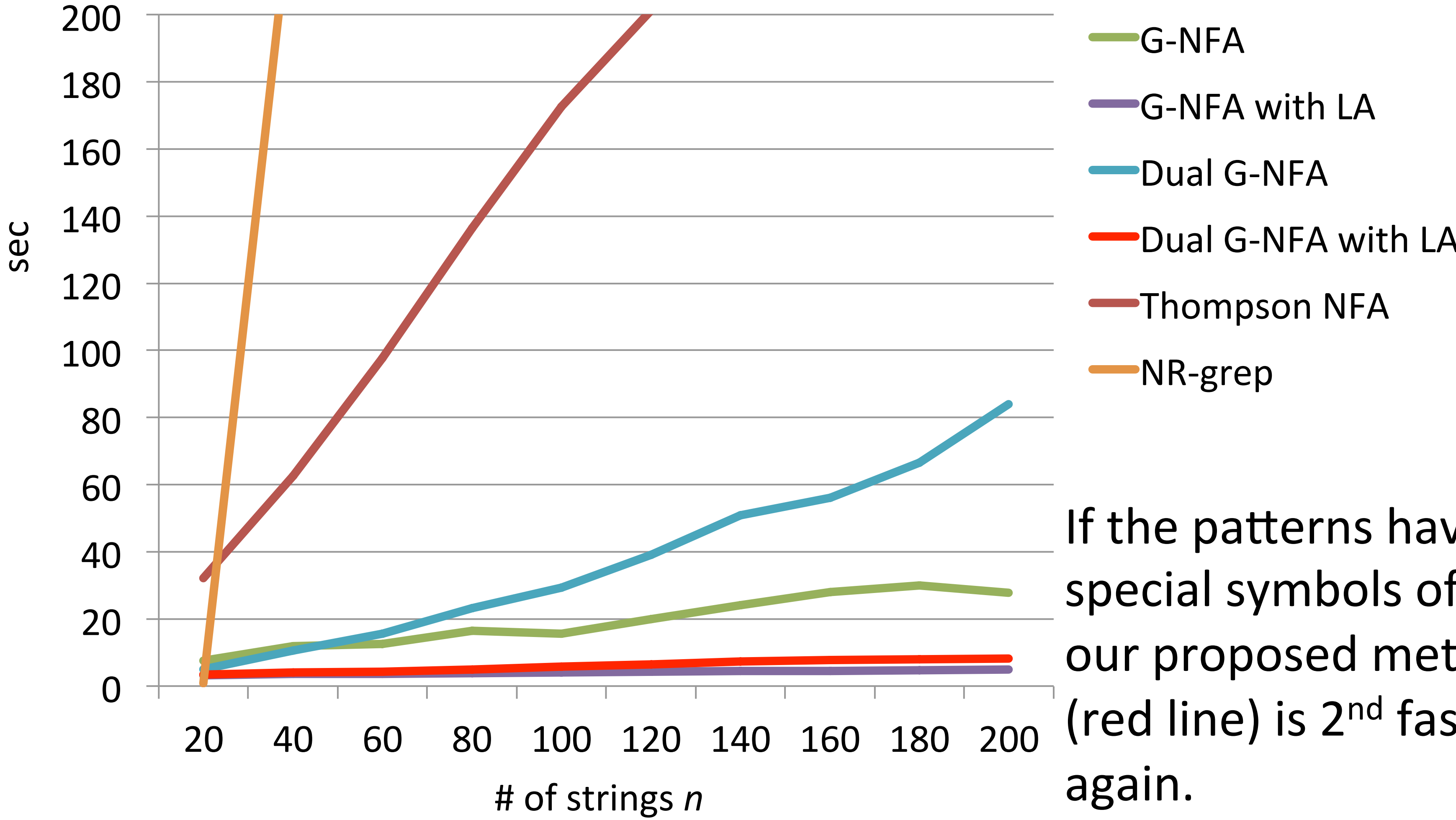



Active state of our proposed method (red line) is smallest.

# Flexible pattern

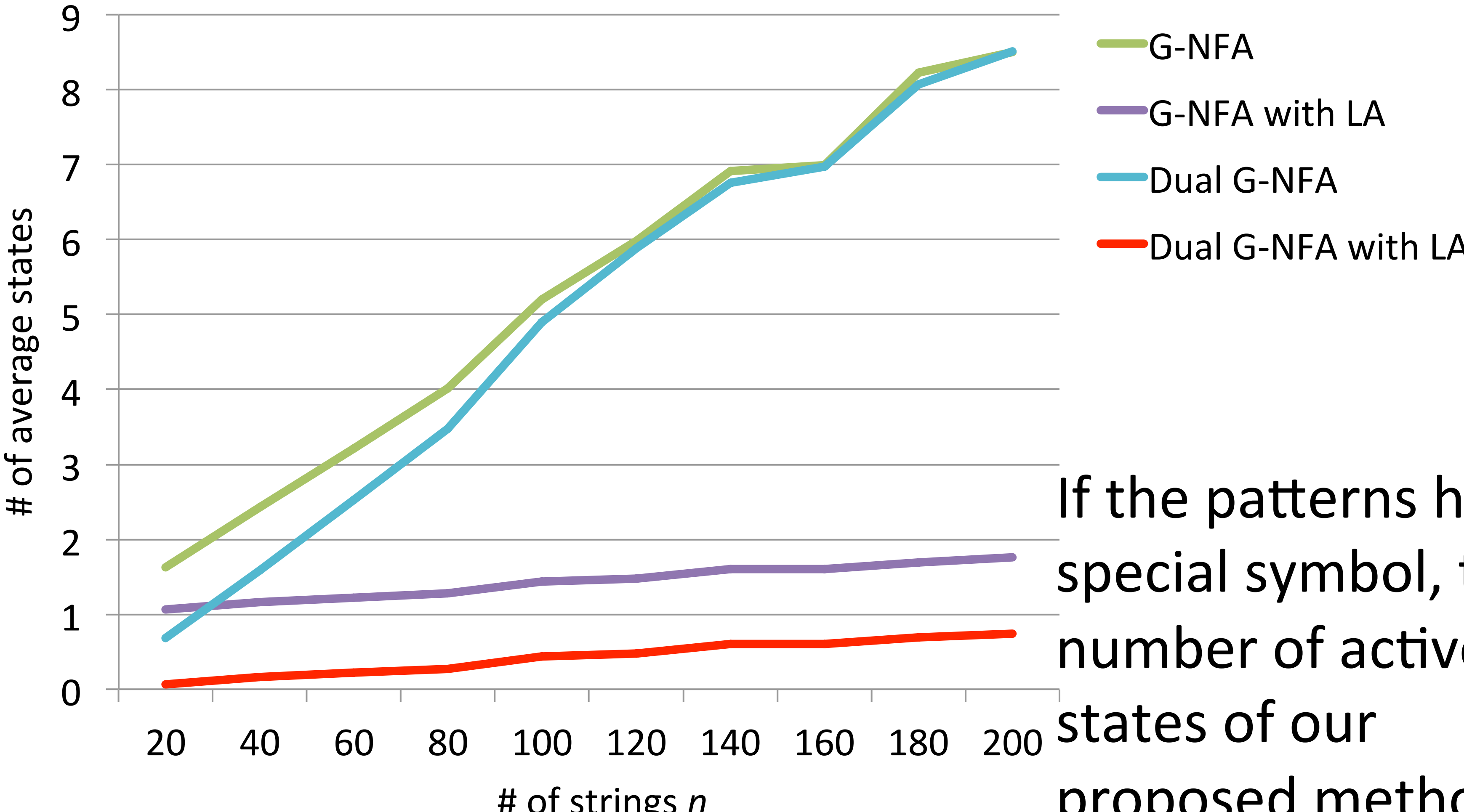
- Pattern
  - n fixed strings used same as previous pattern.
  - a special symbol is inserted to each of the n fixed strings.
  - Inserted strings are joined by disjunction symbol.
- Example
  - R = al*pha|bla+bo|ch+ary|d*elta|…|Juliet

# Results: Elapsed Time(sec)




If the patterns have special symbols of RE, our proposed method (red line) is 2nd fastest again.

# Results: # of Average Active States




If the patterns have special symbol, the number of active states of our proposed method (red line) is smallest.

# Reasonable patterns

- Some reasonable regular expression patterns are made as follows.
- “Suffix”
  – [a-zA-Z]+(able|ible|al|…|ise) (total 35 suffixes)
- “Prefix”
  – (in|il|im|infra|…|under) [a-zA-Z]+ (total 32 prefixes)
- “Names”
  – (Jackson|Aiden|...|Jack) (Smith|Johnson|...|Wilson)
- “User”
  – [a-zA-Z]+@[a-zA-Z]+
- “Title”
  – ([A-Z]+ ␣ )+

# Results

Elapsed time (sec)

| pattern | T-NFA | NR-grep | G-NFA | G-NFA with LA | Dual G-NFA | **Dual G-NFA with LA** |
|---|---|---|---|---|---|---|
| suffix | 113.48 | 20.24 | 9.74 | 7.51 | 106.64 | 3.35 |
| prefix | 14.33 | 5.295 | 2.74 | 3.97 | 78.39 | 3.82 |
| names | 12.95 | 0.216 | 2.97 | 2.74 | 3.21 | 2.76 |
| user | 78.14 | 0.08 | 12.11 | 7.41 | 185.22 | 3.36 |
| title | 38.88 | 0.186 | 2.93 | 2.38 | 2.68 | 2.21 |

Colors

- 1st
- 2nd

Average number of active states

| pattern | G-NFA | G-NFA with LA | Dual G-NFA | **Dual G-NFA with LA** |
|---|---|---|---|---|
| suffix | 2.33 | 1.65 | 50.44 | 1.15 |
| prefix | 1.50 | 1.15 | 8.11 | 0.33 |
| names | 1.01 | 1.00 | 0.01 | 0.001 |
| user | 1.77 | 1.59 | 40.67 | 0.59 |
| title | 1.03 | 1.01 | 0.75 | 0.01 |

Our Method:

- Fastest for 'suffix';
- 2nd fastest for other patterns;
- Decreases active states in all patterns.

# Conclusion

- We developed efficient regular expression matching method that combines Dual Glushkov NFA and Look-Ahead Matching.
- Our proposed method is very fast especially in "large-scale" & "sparsely active" patterns.
- We plan to use our method for extended regular expressions such as ‘character class’, ‘wild card’, and so on.
- We also plan to use our method to network intrusion detection patterns.

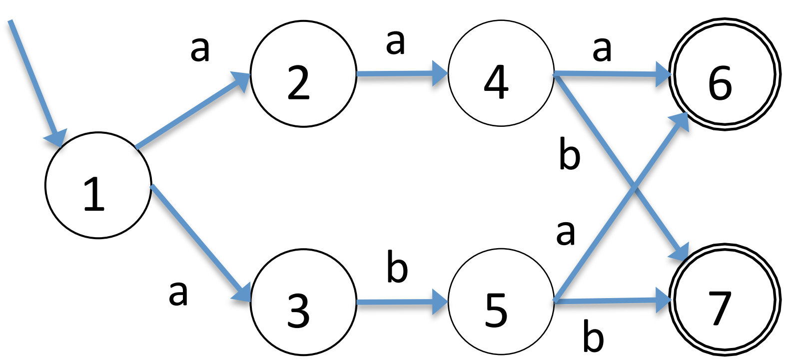
Thank you!

Thank you!

# What is Duality?

- Glushkov NFA
  - Incoming transitions are labeled by the same symbol.
  - It has only one initial state.
  - It has one or more final states.

R = (aa|ab)(a|b)




- Dual Glushkov NFA
  - **Outgoing transitions are labeled by the same symbol.**
  - It has only one final state.
  - It has one or more initial states.

R = (aa|ab)(a|b)