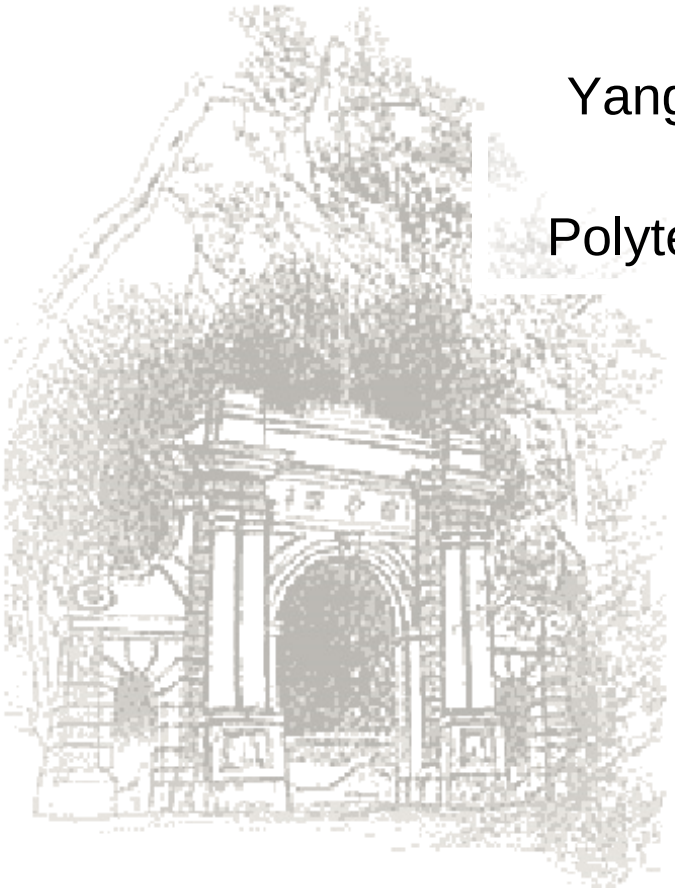
TFA: A Tunable Finite Automaton for Regular Expression Matching



Yang Xu, Junchen Jiang, Rihua Wei, Yang Song and H. Jonathan Chao

Polytechnic Institute of New York University, USA

Carnegie Mellon University, USA

Presented by Zhe Fu

April 9, 2014

NFA and DFA

- NFA
  - limited size
  - expensive per-character processing
- DFA
  - limited per character processing
  - large automaton

A ***trade-off***

- size of the automaton
- worst-case bound on the amount of per-character processing

NFA and DFA

- NFA
  - limited size
  - expensive per-character processing
- DFA
  - limited per character processing
  - large automaton

A ***trade-off***

- size of the automaton
- worst-case bound on the amount of per-character processing

Tunable Finite Automaton (TFA)

Main contributions:

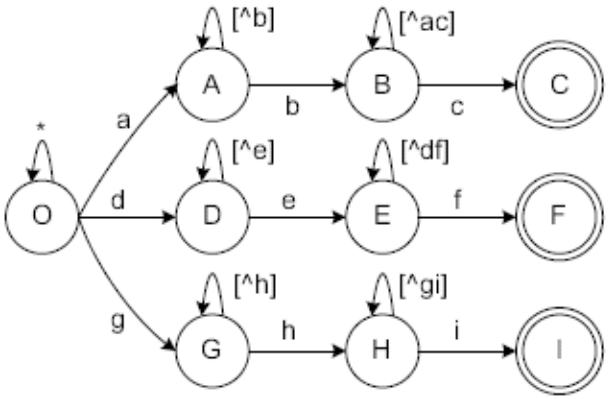
- Introducing TFA, the first finite automaton model with a clear and tunable bound on the number of concurrent active states (more than one) independent of the number and patterns of regular expressions.
- Building a mathematical model to analyze the set split problem (SSP).
- Developing a novel state encoding scheme to facilitate the implementation of a TFA.
- Evaluating the proposed TFA using regular expression sets from Snort and Bro.

Related Work

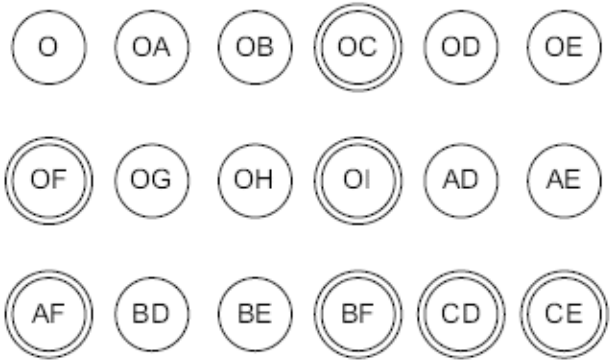
- Transition reduction
  - D^2FA CD^2FA
- State reduction
  - Yu combine regular expressions into multiple DFAs
  - XFA, history-based finite automatons
- Hybrid Finite Automaton
  - Hybrid-FA
  - Lazy DFA

Motivation

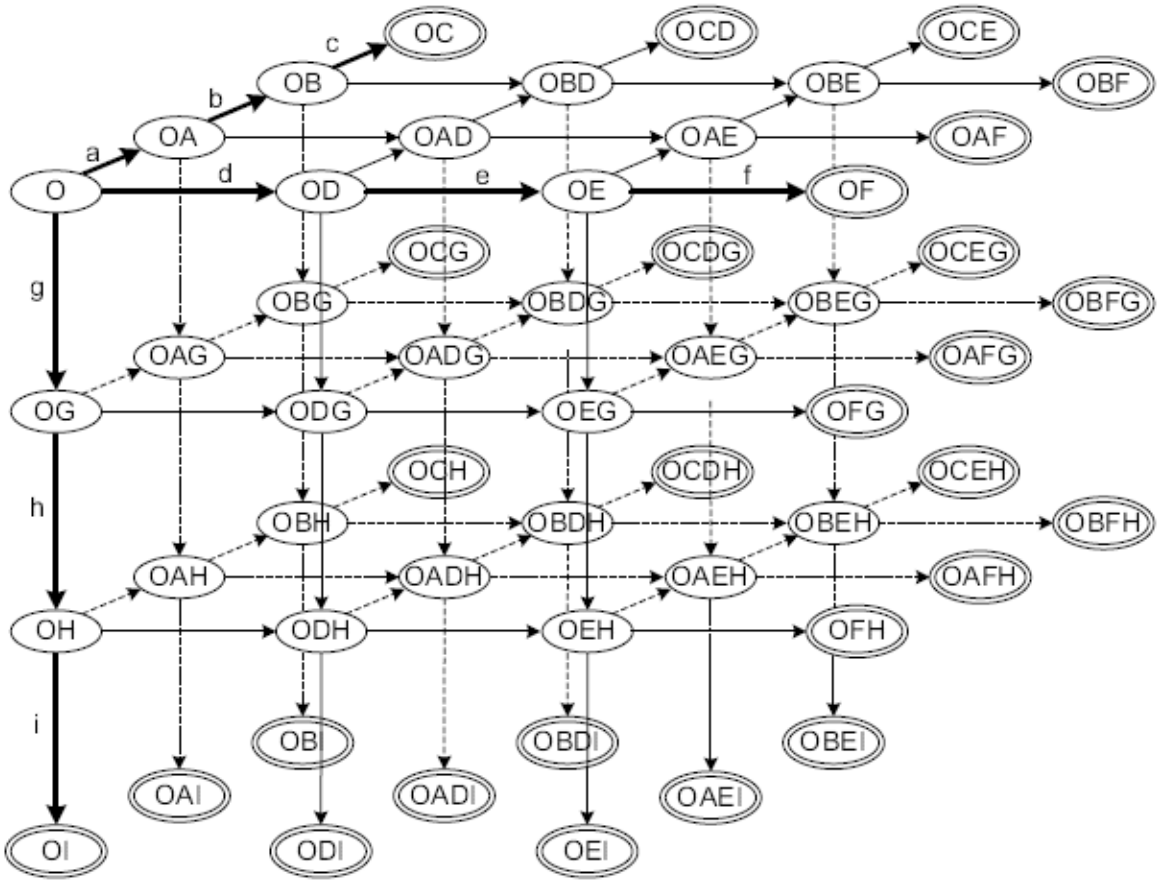
.*a.*b[^a]*c, .*d.*e[^d]*f, .*g.*h[^g]*i



(a) NFA



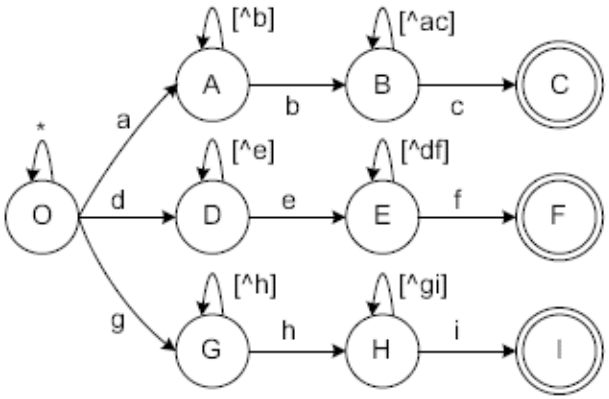
(c) 2-TFA (only states)



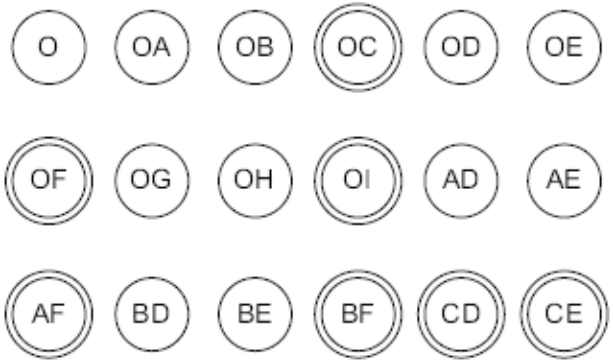
(b) DFA (For simplicity, some less important transitions are omitted)

Motivation

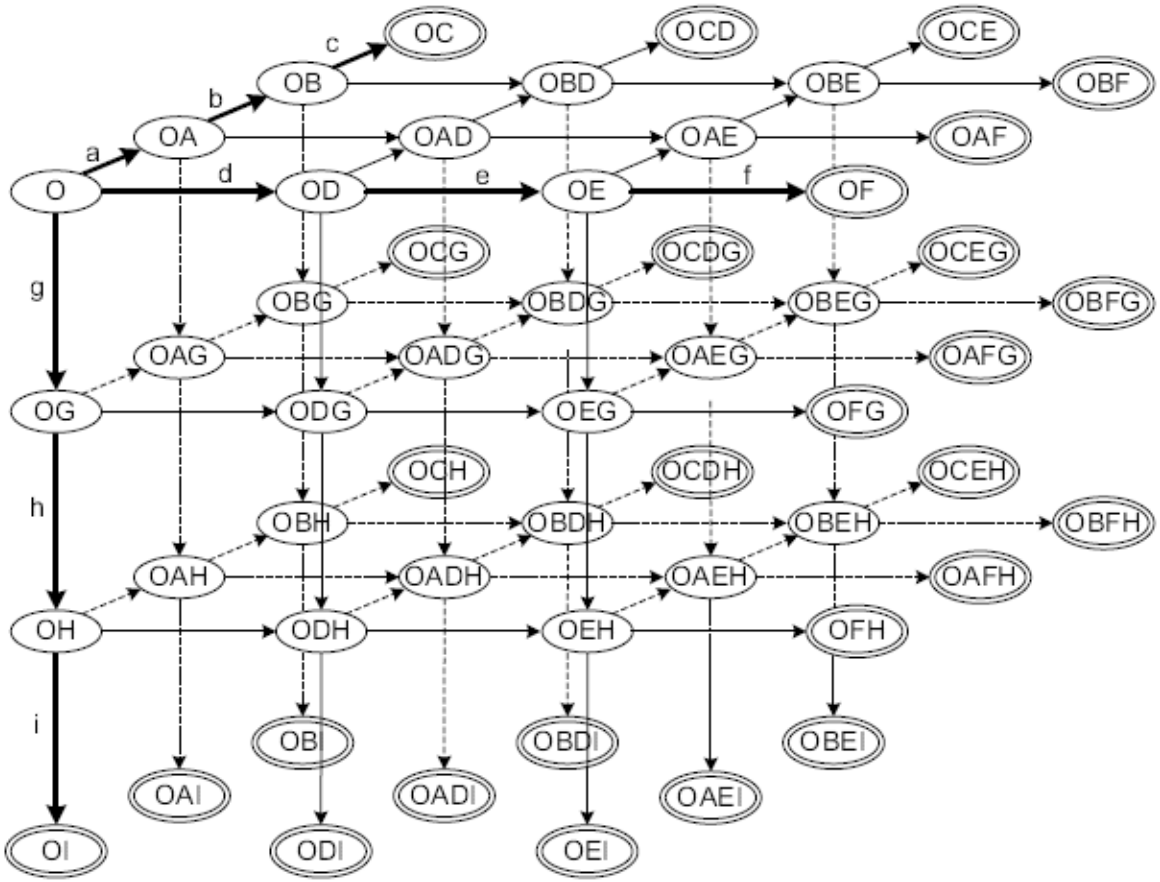
.*a.*b[^a]*c, .*d.*e[^d]*f, .*g.*h[^g]*i adegf



(a) NFA



(c) 2-TFA (only states)



(b) DFA (For simplicity, some less important transitions are omitted)

Motivation

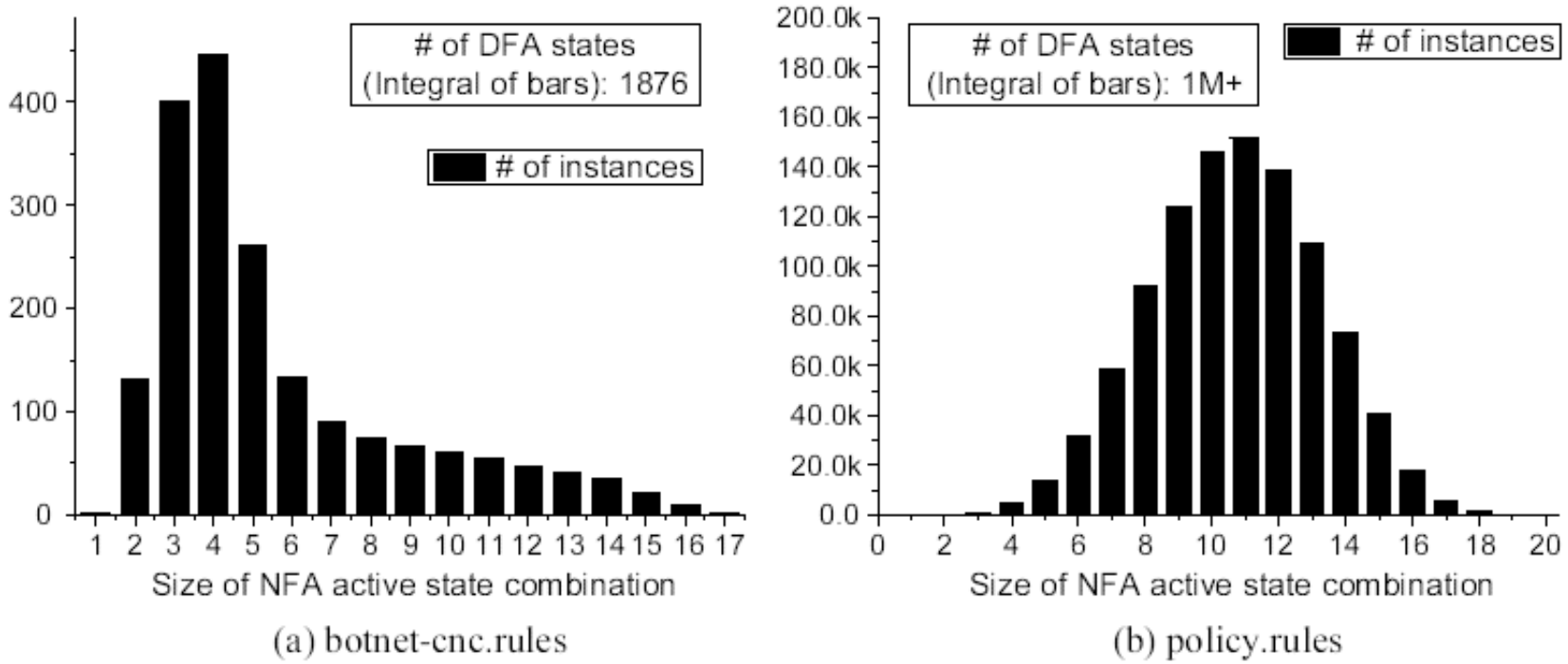


Fig. 2. Distribution of sizes of NFA active state combinations.

Motivation

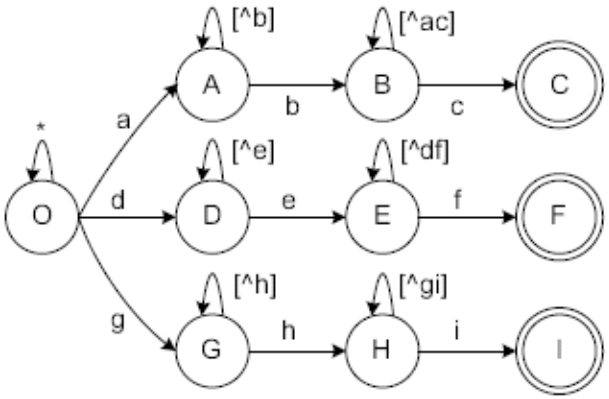
- We have seen the main reason for the DFA having far more states than the corresponding NFA is that the DFA needs one state for each NFA active state combination
- One possible solution is to allow multiple automaton states *(bounded by a given bound factor b)* to represent each combination of NFA active states. We name it Tunable Finite Automaton (TFA).
- N_T : the number of TFA states
  P : the number of all possible statuses that can be represented by at most *b* active states

$$P = \sum_{i=1}^{b} \binom{N_T}{i} = O\left({N_T}^b\right) \qquad (1)$$

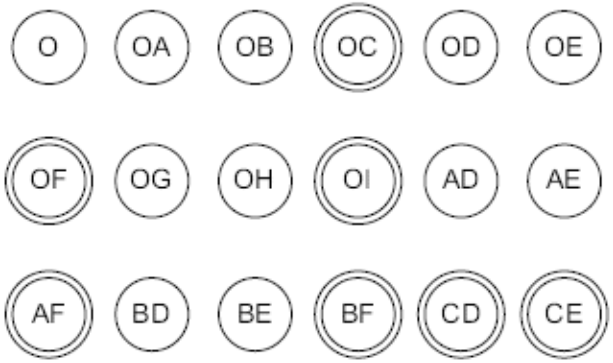
- A TFA with $N_T = O(\log_b(N_D))$ states can represent a DFA with N_D states

TFA States

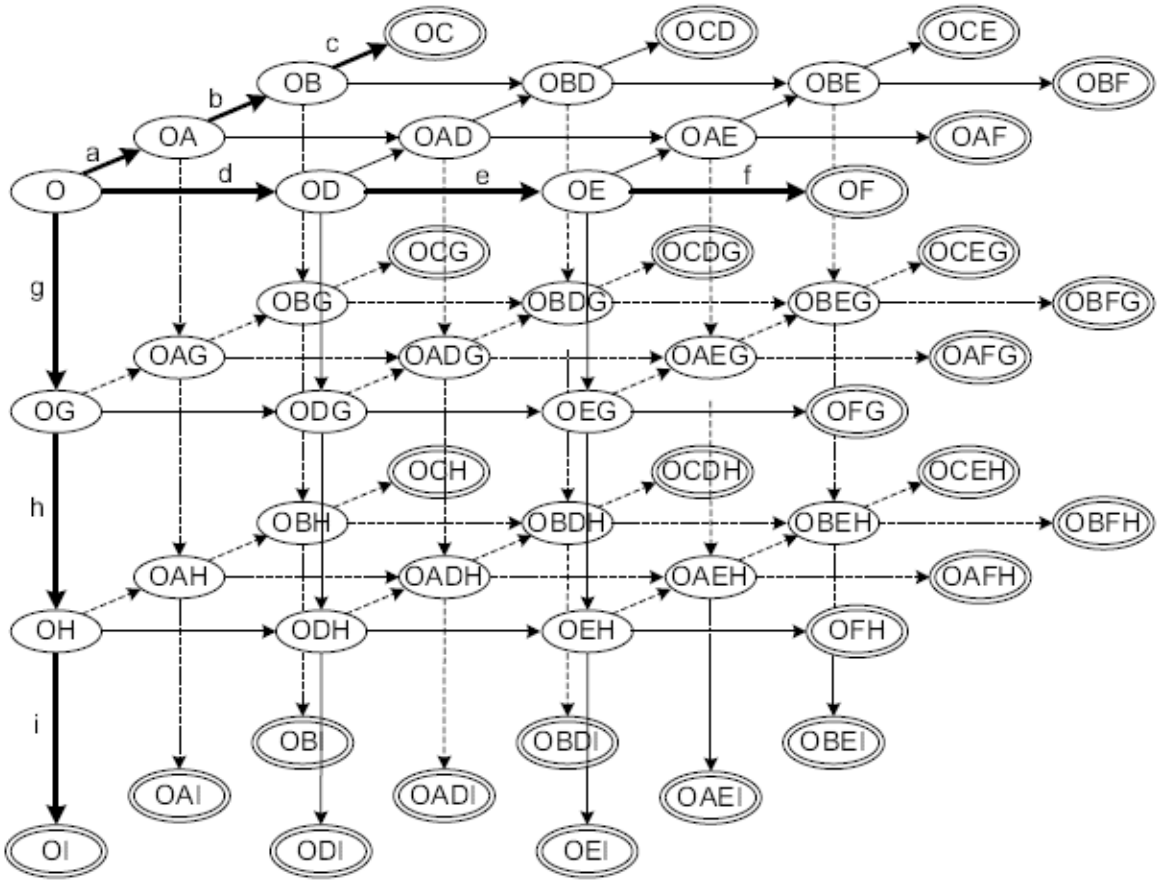
.*a.*b[^a]*c, .*d.*e[^d]*f, .*g.*h[^g]*i



(a) NFA



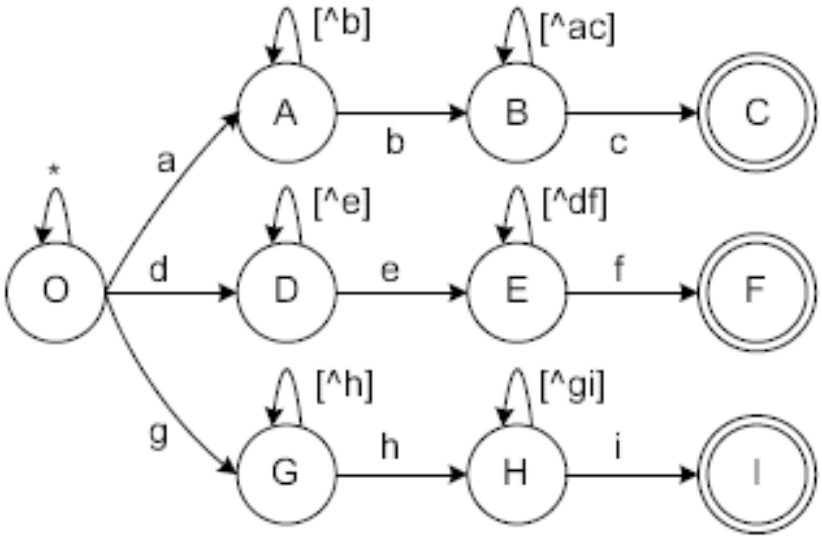
(c) 2-TFA (only states)



(b) DFA (For simplicity, some less important transitions are omitted)

TFA Transitions

.*a.*b[^a]*c, .*d.*e[^d]*f, .*g.*h[^g]*i



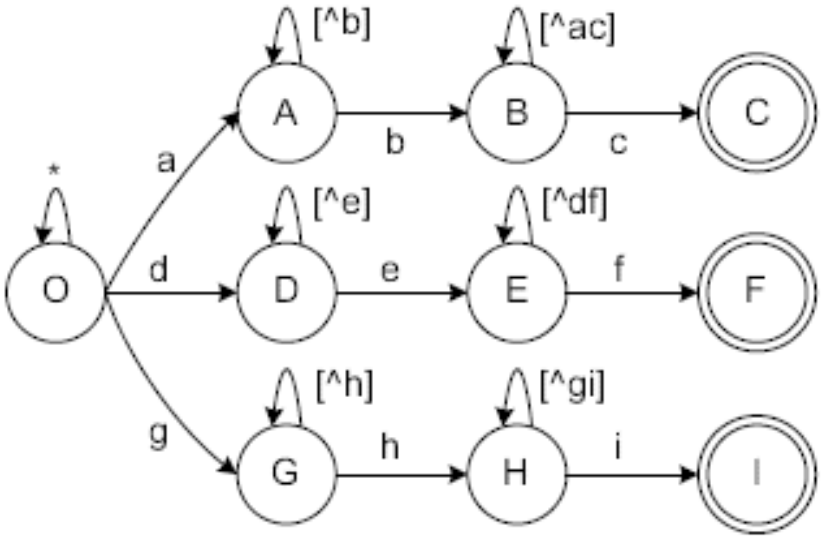
(a) NFA

(c) 2-TFA (only states)

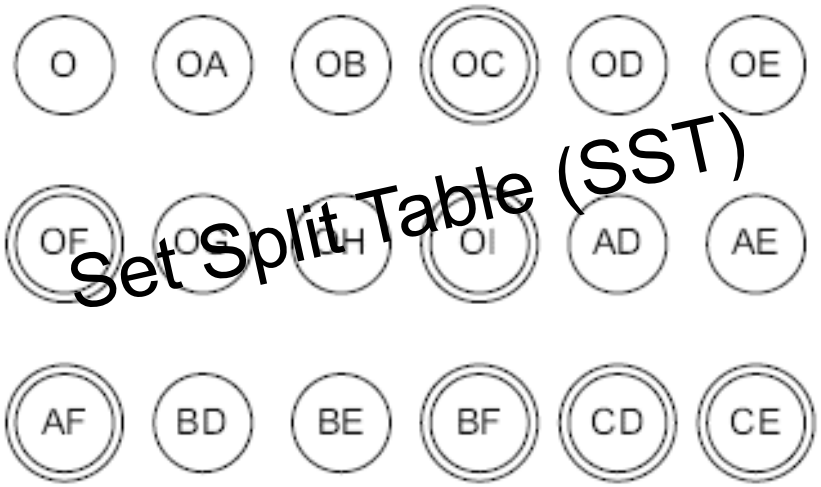
	TFA	**NFA**
T	OD, OG	O, D, G
T+1 input: a	?	O, A, D, G

TFA Transitions

.*a.*b[^a]*c, .*d.*e[^d]*f, .*g.*h[^g]*i



(a) NFA



(c) 2-TFA (only states)

	TFA	**NFA**
T	OD, OG	O, D, G
T+1 input: a	?	O, A, D, G

TUNABLE FINITE AUTOMMATON(TFA)

Definition :

A *b*-TFA is a 6-tuple $< Q_T , \Sigma, \delta_T , I, F_T , SS >$

- A finite set of TFA states Q_T ;
- A finite set of input symbols Σ;
- A transition function $\delta_T : Q_T \times \Sigma \rightarrow P(Q_N)$;
- A set of initial states $I \subseteq Q_T$, $|I| \leq b$;
- A set of accept states $F_T \subseteq Q_T$;
- A set split function $SS : Q_D \rightarrow (Q_T)^b \cup \ldots \cup (Q_T)^1$.

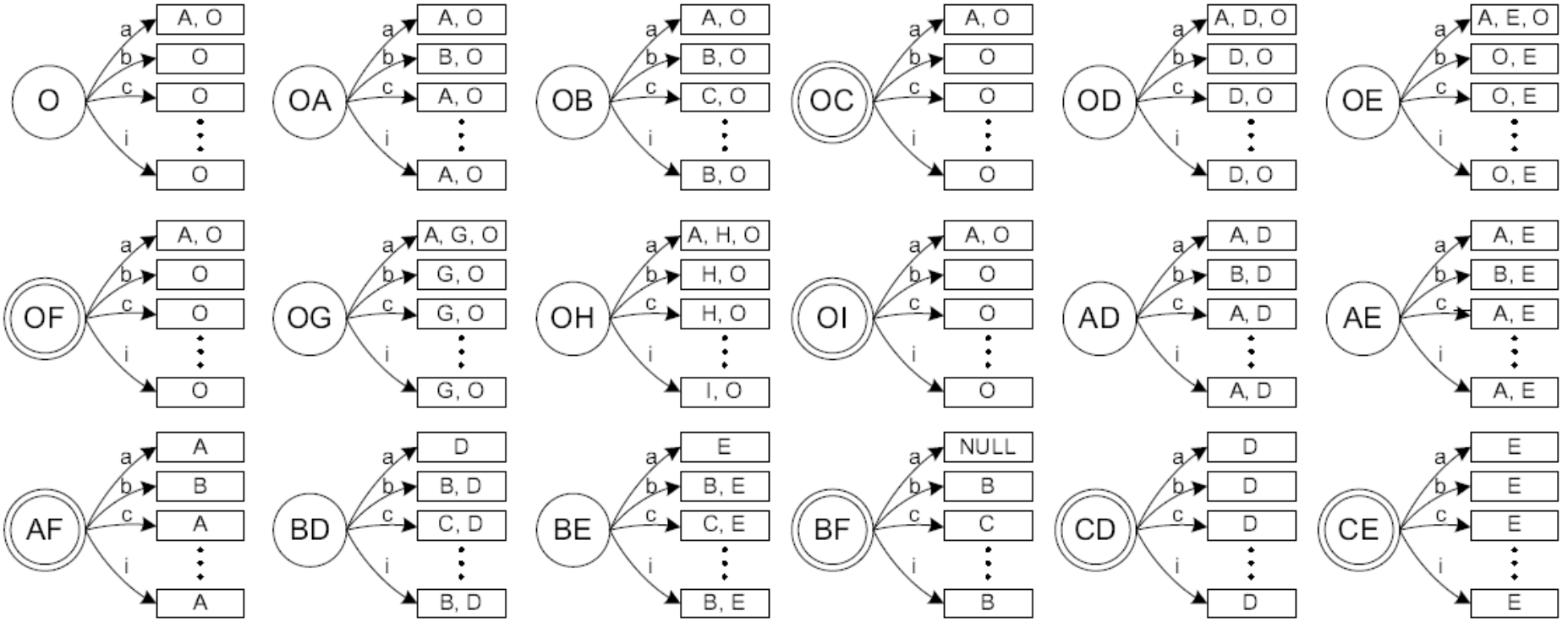
Constructing a TFA

1. **A TFA structure**
2. **Set Split Table (SST) :** Each entry of the SST table corresponds to one combination of NFA active states (i.e., a DFA state) recording how to split the combination into multiple TFA states

{O}	&O	NULL
{A,O}	&OA	NULL
{B,O}	&OB	NULL
…	…	…
{A,D,O}	&O	&AD
{A,E,O}	&OA	&AE
…	…	…
{A,E,G,O}	&OG	&AE
{A,F,G,O}	&OG	&AF
…	…	…

(b) Set Split Table (SST)

Constructing a TFA



(a) A TFA structure with b=2.

Constructing A TFA

Generating a equivalent *b*-TFA from an NFA consists of four steps:

1. Generate the DFA states using the subset construction scheme
2. Split each NFA active state combination into up to *b* subsets. After this step, we obtain the TFA state set Q_T and the set split function *SS;*
3. Decide the transition function δ_T . Outgoing transitions of TFA states point to a data structure called state label, which contains a set of NFA state IDs.
4. Decide the set of initial states (*I*) and the set of accept states (F_T)

Constructing A TFA

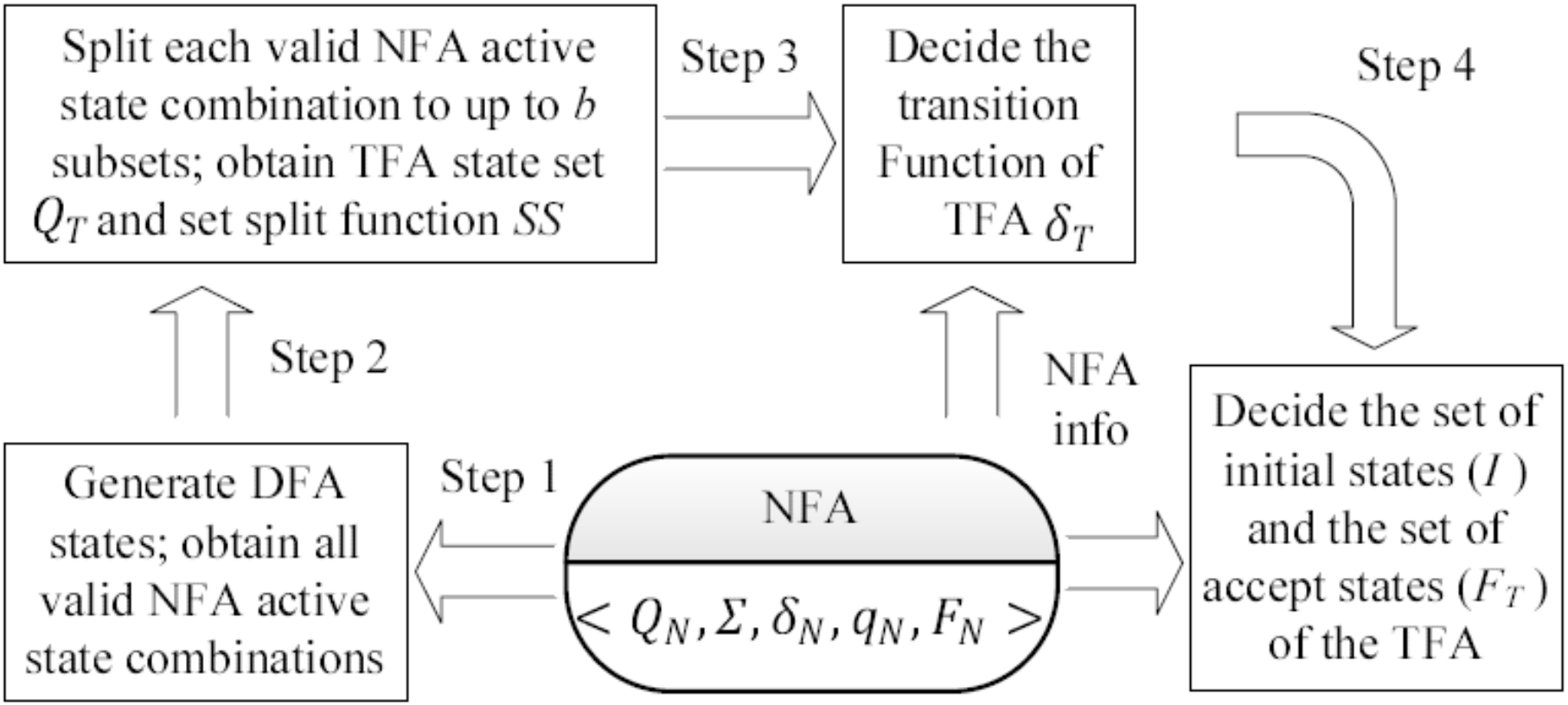


Fig. 4. The flowchart for generating a TFA.

Operating A TFA

Algorithm 1 Operations of b-TFA in each time slot

Input:
  s $(s \leq b)$; ▷ no. of active states in current time slot
  $A[j](j = 1, \ldots, s)$; ▷ current active states
  c; ▷ input character
Output:
  s' $(s' \leq b)$; ▷ no. of active states in next time slot
  $A'[j]\,(j = 1, ..., s')$; ▷ active states in next time slot

$T = NULL$;
for $(j = 1, ..., s)$ **do**
  $T = T \cup$ state label on state $A[j]$ labeled with c;
end for
use T to access SST table, returning s' and $A'[j]\,(j = 1, ..., s')$

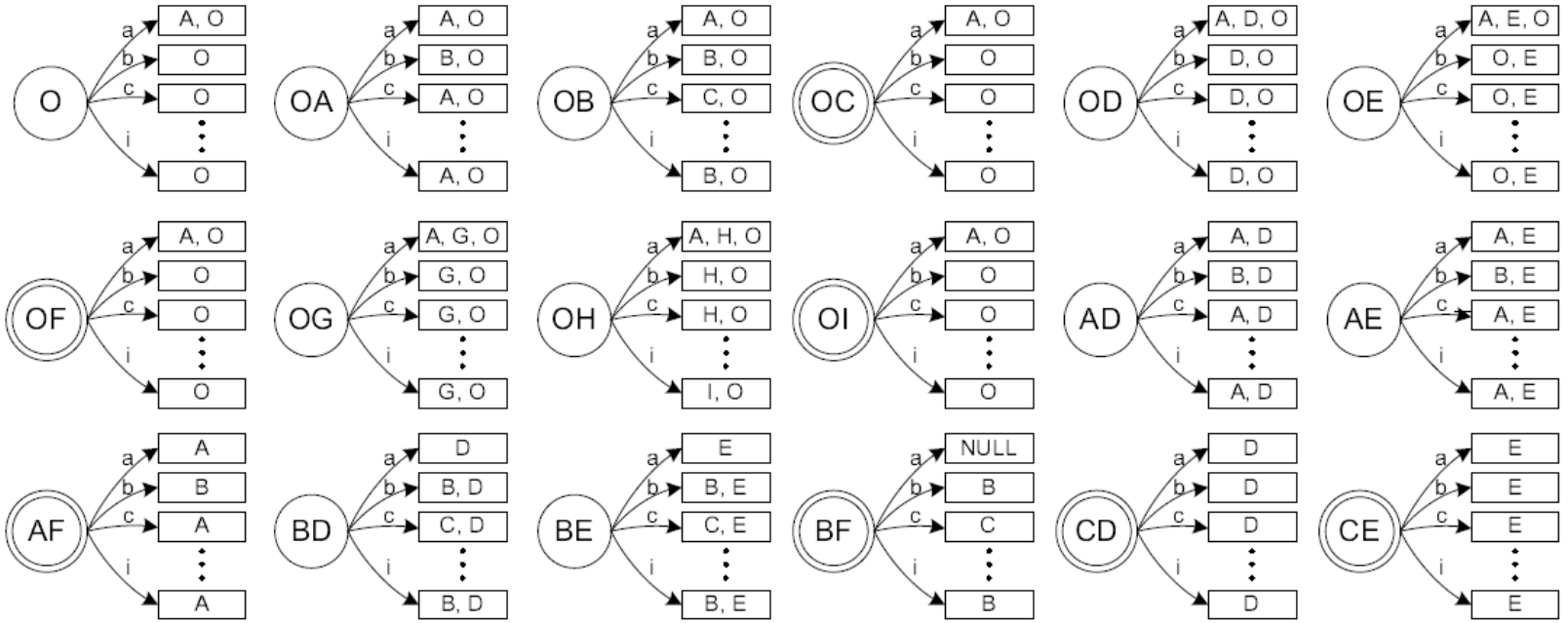
Constructing a TFA

assume the input string is "*adegf*". Initial active state : O

1. a : return label {A,O}, next active states : OA
2. d : return label {A,D,O}, next active states :O , AD
3. e : return label {A,E,O}, next active states :O , AE
4. g : return label {A,E,G,O}, next active states :OG , AE
5. f : return label {A ,F,G,O}, next active states :OG , AF
6. AF is an accept state => match!

{O}	&O	NULL
{A,O}	&OA	NULL
{B,O}	&OB	NULL
…	…	…
{A,D,O}	&O	&AD
{A,E,O}	&OA	&AE
…	…	…
{A,E,G,O}	&OG	&AE
{A,F,G,O}	&OG	&AF
…	…	…

(b) Set Split Table (SST)

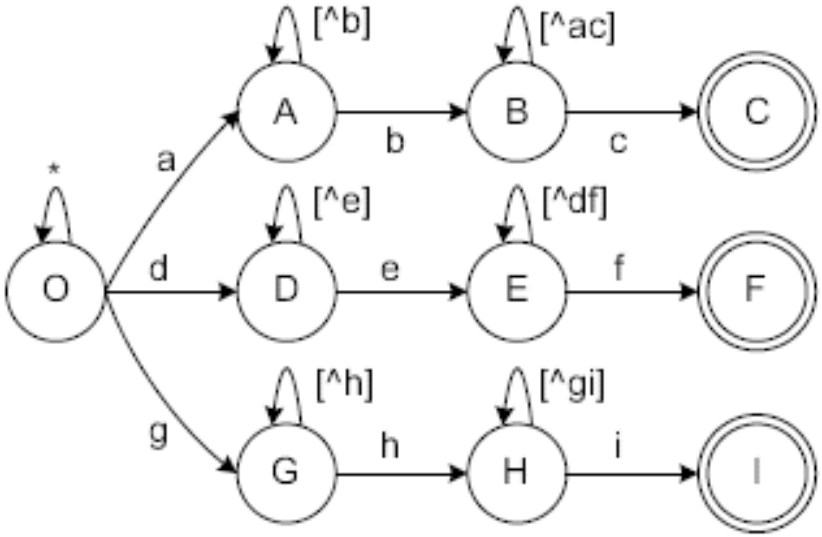


(a) A TFA structure with b=2.

Constructing a TFA

assume the input string is "*adegf*". Initial active state : O

1. a : return label {A,O}, next active states : OA
2. d : return label {A,D,O}, next active states :O , AD
3. e : return label {A,E,O}, next active states :O , AE
4. g : return label {A,E,G,O}, next active states :OG , AE
5. f : return label {A ,F,G,O}, next active states :OG , AF
6. AF is an accept state => match!



(a) NFA

(c) 2-TFA (only states)

Next

Next, we will address two critical issues in the implementation of a TFA:

(1) How to split the NFA active state combinations to obtain a small TFA;

(2) How to store TFA and SST table efficiently to facilitate the TFA operation.

Splitting NFA Active State Combinations

Set Split Problem (SSP) :

- Find a minimal number of subsets from the NFA state set, so that for any valid NFA active state combination, we can always find up to *b* subsets to exactly cover it
- *b*-SSP problem is an NP-hard problem for any *b > 1*

Notations	Descriptions
Q_N	The set of all NFA states
N_D	The number of different combinations of NFA active states (i.e., the number of states in the corresponding DFA, $N_D = \|Q_D\|$)
S_i	The i-th combination of NFA active states $(i = 1, ..., N_D)$
$S_{i,j}$	The j-th subset split from S_i $(j = 1, ..., b)$
Q_T	The union of $S_{i,j}$ $(i = 1, ..., N_D; j = 1, ..., b)$

SET SPLIT PROBLEM

Subject to.

$$\bigcup_j S_{i,j} = S_i; (i = 1, ..., N_D; j = 1, ..., b) \quad (2)$$

$$Q_T = \{S_{i,j} \mid i = 1, ..., N_D; j = 1, ..., b\} - \{\emptyset\} \quad (3)$$

Objective.

$$\text{Minimize: } |Q_T| \quad (4)$$

To simplify the problem, we add another constraint on the model of the *b*-SSP problem :

$$S_{i,j} \bigcap S_{i,k} = \emptyset \quad (\forall j \neq k;\ i = 1, ..., N_D)$$

2-SSP Problem

- Let v_i be the number of states in the i-th NFA active state combination. The number of different ways to split the combination (denoted as Fi) under the 2-SSP problem can be expressed as follows.

$$F_i = 2^{v_i - 1}$$

- Given an NFA active state combination with v states, we consider only two kinds of special splits:
  - (1) No split at all (i.e., one subset is empty);
  - (2) Splits that divide the combination into two subsets whose sizes are 1 and v − 1, respectively.

    e.g.: {A,B,C,D}, {A,B,C,E}, {A,B,C,F}, {A,B,C,G}

2-SSP Problem

- Initially, Q_T is empty. In each iteration, the algorithm starts with the subset vertices in the right partition,
- select the largest-degree subset among the subsets whose sizes and degrees are both larger than 1
  - If no such subset is found, select the largest-degree subset from among all the subsets
- The split-decision vertices connected with the selected subset vertices will also be selected and stored in the SST table.
- Then remove those vertices that will not be used from the tripartite graph.
- Now one iteration is finished. The iteration repeats until all subset vertices are removed or selected.

2-SSP Problem

- **first iteration**: select subset vertex {A,D,O}, and split-decision vertices {A,D,O} and {G}, {A,D,O}.
- **second iteration**: select the subset vertex {O}, and split-decision vertices {O} and {G}, {O}.
- **third iteration**: select the subset vertex {G}.

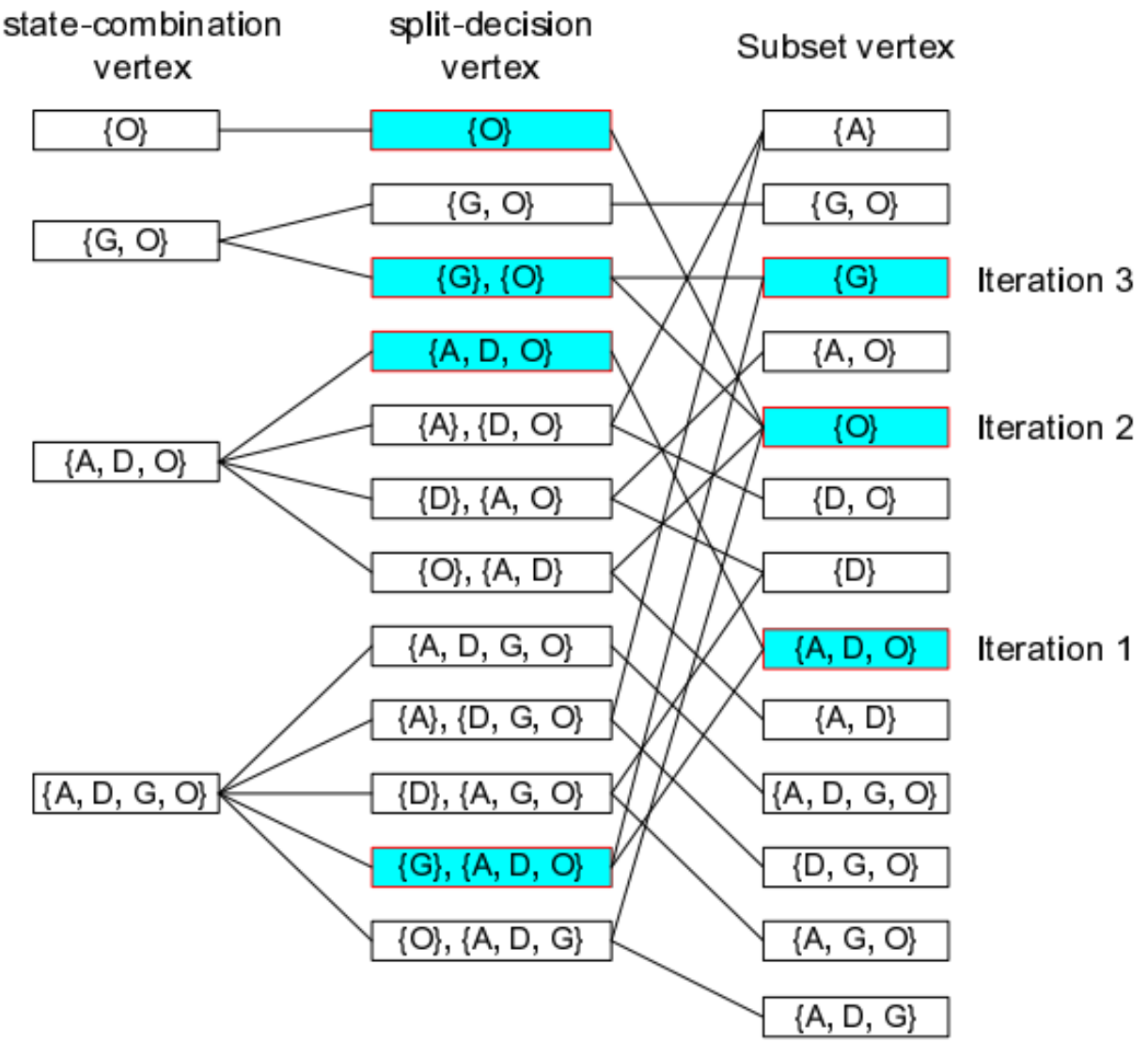


Fig. 6. Tripartite graph used in the heuristic algorithm.

b-SSP Problem

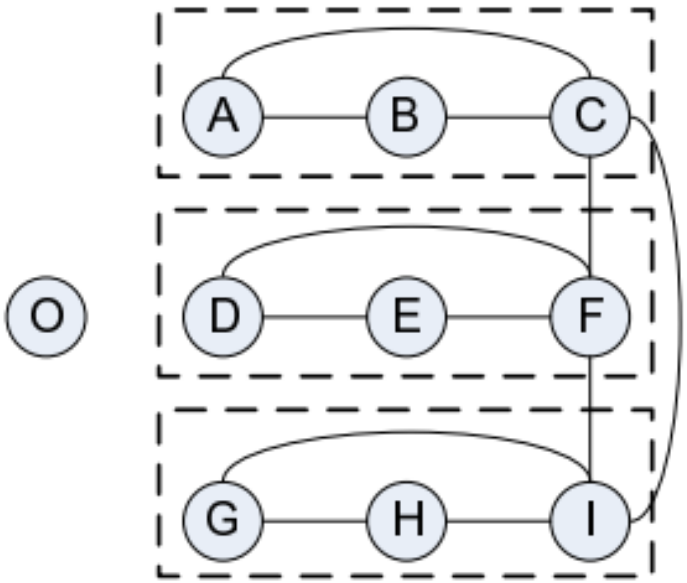
- if b is equal to the power of 2
  - just need to run the algorithm of the 2-SSP problem recursively for $\log_2 b$ times
- If b is an arbitrary integer
  - run the proposed heuristic algorithm for $b - 1$ times

State Encoding

- Original : using an array to store state labels which include different numbers of NFA state IDs
- Problem :
  - High storage cost
  - TFA operation overhead :
    - Redundancy elimination {D, O}, {G, O}
    - Sorting {O, D, G} and {D, G, O}
- Solution : bit vector

State Encoding

Two nodes in the independent graph are connected by an edge, if and only if their associated NFA states are never active together (i.e., never together in one NFA active state combination)



(a) An independent graph corresponding to the NFA in Figure 1 (a).

A:	0	1	0	0	0	0	0
B:	1	0	0	0	0	0	0
C:	1	1	0	0	0	0	0
D:	0	0	0	1	0	0	0
E:	0	0	1	0	0	0	0
F:	0	0	1	1	0	0	0
G:	0	0	0	0	0	1	0
H:	0	0	0	0	1	0	0
I:	0	0	0	0	1	1	0
O:	0	0	0	0	0	0	1

(b) State encoding map based on cliques in (a).

Fig. 7. Independent graph and state encoding map.

Performance Evaluation

TABLE III
BASIC STATISTICS OF REGULAR EXPRESSION RULE SETS

Rule set	# of RegEx	Avg. ASCII length of RegEx	% RegEx having infinite repetitions, e.g., "*", "+"	% RegEx having finite repetitions, e.g., {n,m},"?"	% RegEx having char ranges, e.g., \s,".", [a-z]
Snort 1	17	38.6	88.2%	41.2%	94.1%
Snort 2	12	16.7	16.7%	83.3%	100.0%
Snort 3	1	19.0	100.0%	100.0%	100.0%
Snort 4	34	29.7	32.4%	58.8%	82.4%
Bro	63	14.6	11.1%	12.7%	23.8%

Performance Evaluation

STATE NUMBERS OF AUTOMATONS UNDER DIFFERENT REGULAR EXPRESSION SETS

Rule set	NFA	DFA	Hybrid-FA	DFA Grouping		
				b=2	b=3	b=4
Snort 1	385	369870	7167	63637	10166	1857
Snort 2	191	100697	7955	16851	8396	3816
Snort 3	38	363154	63443	363154	363154	363154
Snort 4	417	158060	11714	11459	6181	2434
Bro	567	407677	22305	148665	6890	2316

Rule set	TFA					
	b=2	% reduction	b=3	% reduction	b=4	% reduction
Snort 1	6563	98.23%	1461	99.60%	871	99.76%
Snort 2	5986	94.06%	805	99.20%	284	99.72%
Snort 3	62133	82.89%	22418	93.83%	14668	95.96%
Snort 4	10846	93.14%	3005	98.10%	1597	98.99%
Bro	16576	95.93%	3685	99.10%	2000	99.51%

Performance Evaluation

STATISTICS OF STATE ENCODINGS IN THE TFA

rule set	# of bits in state encoding	b=2		b=3		b=4	
		# of state lables	% of distinct state lables	# of state lables	% of distinct state lables	# of state lables	% of distinct state lables
Snort 1	40	1680128	0.87%	374016	0.98%	222976	0.88%
Snort 2	28	1532416	0.52%	206080	0.56%	72704	0.68%
Snort 3	38	15906048	0.48%	5739008	0.42%	3755008	0.46%
Snort 4	61	2776576	0.69%	769280	0.60%	408832	0.63%
Bro	55	4243456	0.64%	943360	0.77%	512000	0.90%

Performance Evaluation

MEMORY COSTS OF AUTOMATONS

rule set	DFA (MB)	Hybrid-FA (MB)	DFA Grouping (b=2) (MB)	TFA (b=2)	
				MB	% reduction
Snort 1	379	7.34	65.16	9.68	97.44%
Snort 2	103	8.15	17.26	6.94	93.27%
Snort 3	372	64.97	371.87	66.53	82.11%
Snort 4	162	12.00	11.73	9.59	94.07%
Bro	417	22.84	152.23	20.24	95.15%

Performance Evaluation

NUMBER OF ACTIVE STATES

rule set	NFA		Hybrid-FA			TFA (b=2)
	Max	Worst case	Max	# tail-FA	Worst case	Worst case
Snort 1	10	16	3	12	13+	2
Snort 2	3	8	3	5	6+	2
Snort 3	3	33	3	1	2+	2
Snort 4	8	18	5	15	16+	2
Bro	21	25	10	22	23+	2

Conclusion

- This paper proposed a new finite automaton representation, TFA, for regular expression matching.
- It combines the strengths of both NFAs and DFAs by allowing a small bounded number of active states during the matching processing, so that the worst case performance of a TFA is guaranteed.
- Better solutions for the problem for SSP, compressing, etc., are still available.

Thanks!