Chain-based DFA Deflation

—— **Novel Approaches for Fast and Scalable Regular Expression Matching**

Presenter: Kyle Wang

2011-9-28

Outline

- Background
  - Significance
    - why research regular expression (regex) matching?
  - Challenges
    - can we solve the problems of regex matching?
- Motivation
  - Observation
    - the feature of DFA (Deterministic Finite Automaton)
- Algorithms
  - Inside chain
    - D^2FA, A-DFA
    - FEACAN, Cluster-DFA, RCDFA
  - Inside chain & inter chain
    - Chain-based DFA deflation
- Conclusion

Background: Why Regular Expressions Acceleration?

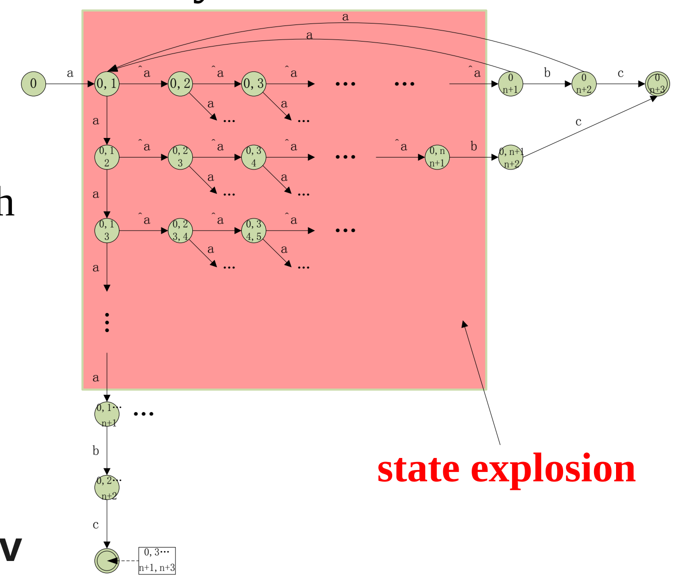
- Regular expressions are now widely used
  - firewalls, filtering, authentication and monitoring
  - network intrusion detection/prevention systems
  - layer 7 switches, traffic billing, load balancing
  - content-based traffic management and routing
- Regular expression matching is expensive
  - space: large amount of memory
  - bandwidth: requires 1+ state traversal per input character
- Regular expression matching is performance bottleneck
  - in enterprise switches from Cisco, etc
  - Cisco security appliances
    - Use DFA, 1+ GB memory, still sub-gigabit throughput

Need to accelerate regular expression matching!

Background: Any way to do Regular Expression Matching Better?

- Construction time versus execution time
  - traditionally, (construction + execution) time is the metric, e.g. grep
  - in networking context, execution time is critical
  - also, there may be thousands of regular expression patterns
- DFAs are fast
  - but can have exponentially large number of states
  - algorithms exist to minimize number of states
  - still 1) low performance and 2) gigabytes of memory
- How to achieve high performance?
  - use ASIC/FPGA/TCAM
    - on-chip memory unit provides ample bandwidth
    - pipelining and parallelism
    - but volume is precious

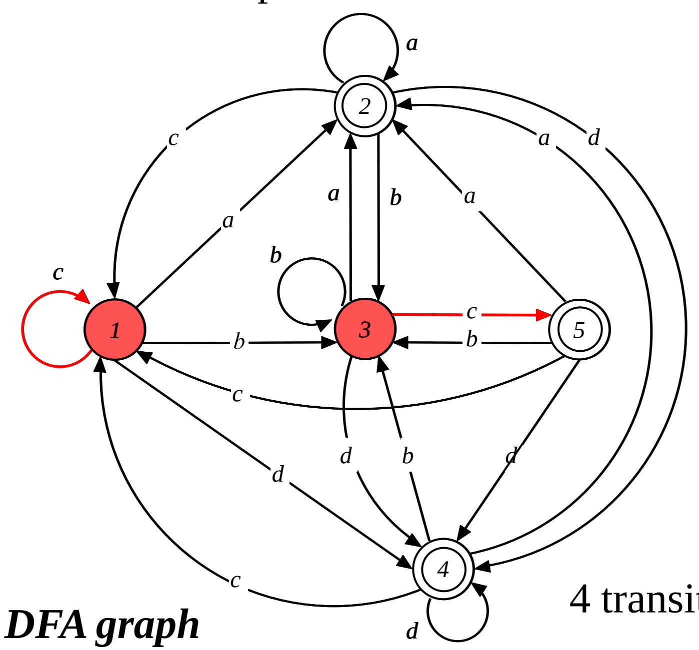
Limited memory, need space efficient representation!



state explosion

Background: Motivation

- How to represent DFA more compactly?
  - cannot reduce the number of states
  - how about reducing the number of transitions?
    - every state is consist of 256 next-hop transitions
    - if use a byte to represent a state ID, then need 256 bytes per state
    - state similarity: almost every state can share the same next-state transitions with multiple other states for most input characters (real world datasets)



DFA graph

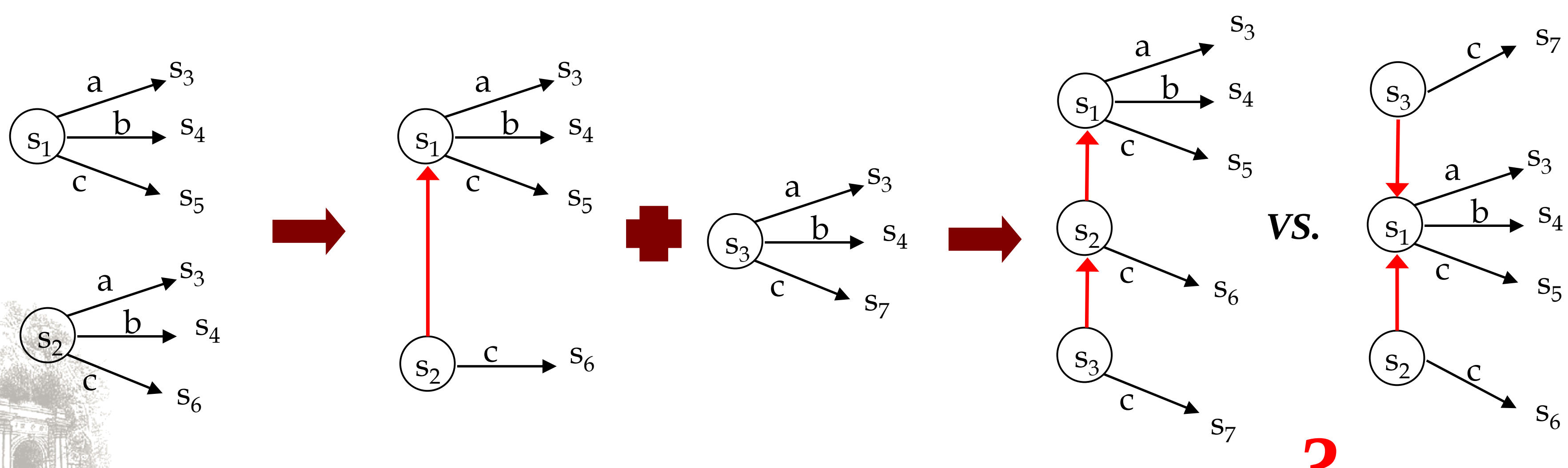
4 transitions per state

RegEx: *a*+, *b*+*c*, *c*d*+**

Look at state pairs:
there are many **common transitions**.
How to remove them?

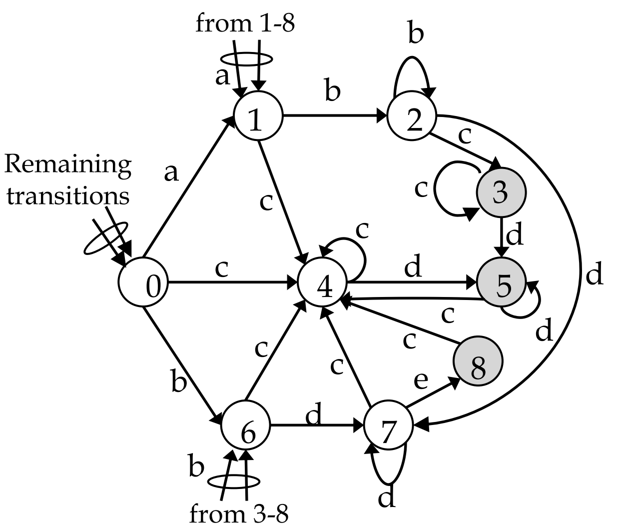
Algorithm: D²FA

- Delay input DFA
  - introduce *default transition*
    - remove the common transitions between states
    - but introduce one more memory access if default transition is taken
    - aim to remove as many transitions as possible, and make as less default transitions as possible per input character

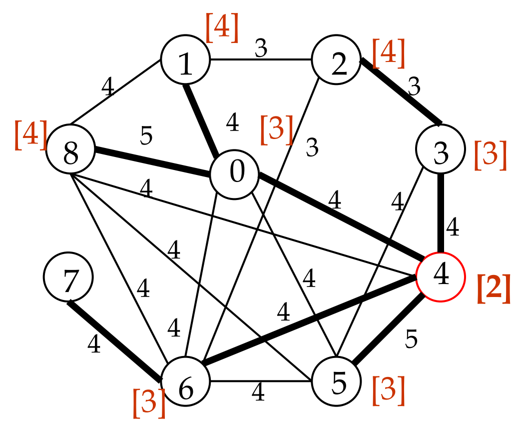


Algorithm: D²FA Construction

DFA

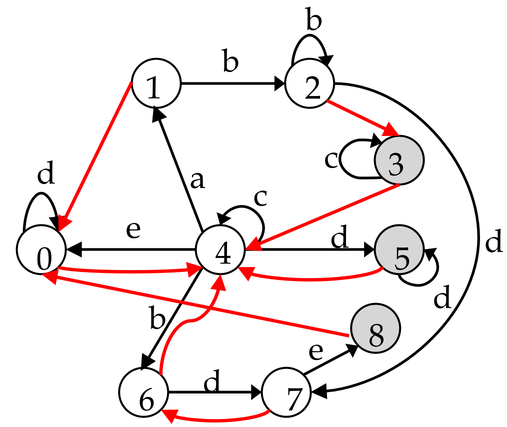


Space reduction graph

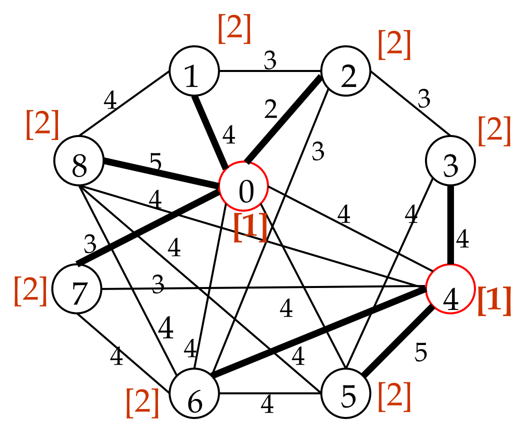


Diameter bound=**4** ⇨ removed transitions=33

D²FA



RegEx: *ab⁺c⁺*, *cd⁺* and *bd⁺e*

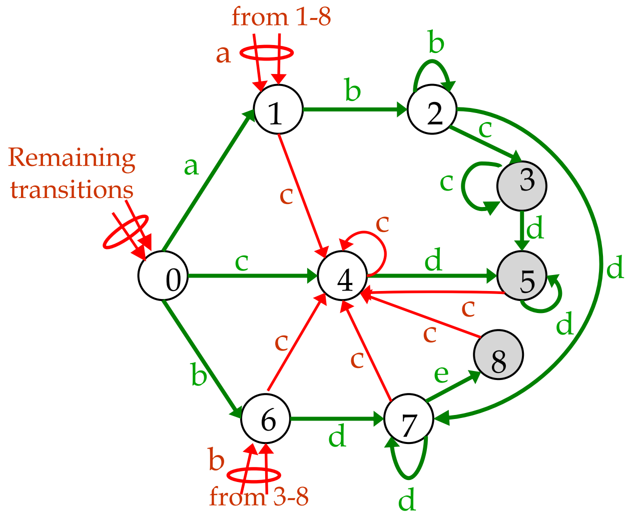


Diameter bound=**2** ⇨ removed transitions=28

Traversal time=O((D/2+1)N)
Time complexity=O(n^2logn)
Space complexity=O(n^2)

Algorithm: A-DFA

- Advanced D^{2}FA
  - introduce *backwards default transition*
    - most transitions are backwards transitions
    - orient default transitions only backwards (towards decreasing depth)

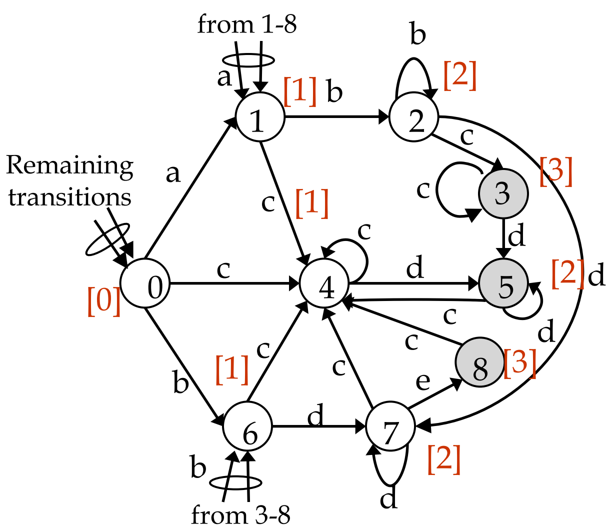


RegEx: *ab*$^+$*c*$^+$, *cd*$^+$ and *bd*$^+$*e*

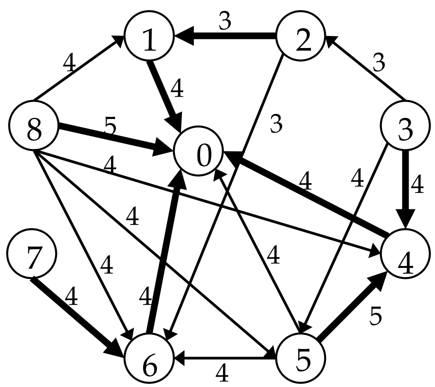
- Forward transitions:
  - Matches
  - State specific
- Backward transitions:
  - Mismatch
  - Shared by multiple states

Algorithm: A-DFA Construction

DFA

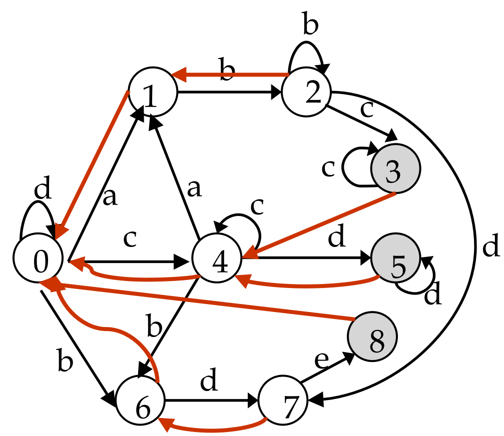


Oriented space reduction graph



NO diameter bound
Time complexity=O(2N)
removed transitions=33

A-DFA



RegEx: ***ab⁺c⁺*, *cd⁺* and *bd⁺e***

Traversal time=O(2N)
Time complexity=O(n^2)
Space complexity=O(n)

Algorithm: D^2FA & A-DFA

- Pros
  - good compression ratio
    - over 95%
  - good worst-case speed
    - O(2N) complexity for A-DFA
- Cons
  - inefficient hardware implementation
    - non-deterministic transition calculation
      - per input character requires comparing all the labeled transitions in current state, before taking the default transition
    - non-deterministic and unbound memory access times per input character
      - transition comparisons are recursively done among all states in the default path, until a non-default transition is found

How to keep pros of D^2FA and overcome the cons!

DFA graph vs. DFA matrix

- DFA matrix Δ_{N*M}
  - regular expressiones
    1. ***def[^ef]*add***
    2. ***def[^df]*bee***
    3. ***def[^de]*cff***
  - alphabet
    - $\Sigma = \{\boldsymbol{a,b,c,d,e,f,g,h}\}$
    - $N = 20$, $M = |\Sigma| = 8$
  - state
  - character
  - (next-state) transition

Character **dimension**

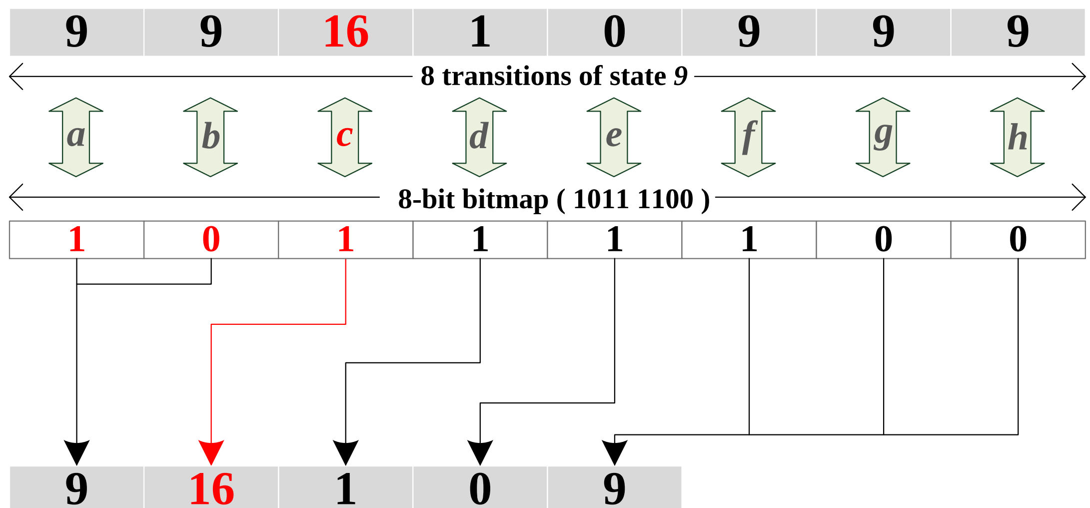
State **dimension**

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
0	0	0	0	1	0	0	0	0
1	0	0	0	1	2	0	0	0
2	0	0	0	1	0	3	0	0
3	4	5	6	7	8	9	3	3
4	4	5	6	10	8	9	3	3
5	4	5	6	7	11	9	3	3
6	4	5	6	7	8	12	3	3
7	13	14	14	7	2	0	14	14
8	8	15	8	1	8	0	8	8
9	9	9	16	1	0	9	9	9
10	13	14	14	17[1]	2	0	14	14
11	8	15	8	1	18[2]	0	8	8
12	9	9	16	1	0	19[3]	9	9
13	13	14	14	10	0	0	14	14
14	13	14	14	7	0	0	14	14
15	8	15	8	1	11	0	8	8
16	9	9	16	1	0	12	9	9
17	13	14	14	7	2	0	14	14
18	8	15	8	1	8	0	8	8
19	9	9	16	1	0	9	9	9

Algorithm cont.

- Bitmap technique
  - Guaranteed deterministic and acceptable memory access

state \ character	a	b	c	d	e	f	g	h
0	0	0	0	1	0	0	0	0
1	0	0	0	1	2	0	0	0
2	0	0	0	1	0	3	0	0
3	4	5	6	7	8	9	3	3
4	4	5	6	10	8	9	3	3
5	4	5	6	7	11	9	3	3
6	4	5	6	7	8	12	3	3
7	13	14	14	7	2	0	14	14
8	8	15	8	1	8	0	8	8
9	9	9	16	1	0	9	9	9
10	13	14	14	17 [1]	2	0	14	14
11	8	15	8	1	18 [2]	0	8	8
12	9	9	16	1	0	19 [3]	9	9
13	13	14	14	10	0	0	14	14
14	13	14	14	7	0	0	14	14
15	8	15	8	1	11	0	8	8
16	9	9	16	1	0	12	9	9
17	13	14	14	7	2	0	14	14
18	8	15	8	1	8	0	8	8
19	9	9	16	1	0	9	9	9



Algorithm: RCDFA

- Compression along the *state* dimension
  - Observation
    - Large redundancy exists among similar states whose distribution is scattered
      - Derived from regular expression characteristic
      - Verified by real-life rule sets
    - Bitmap technique is only effective for consecutively identical transitions
- Proposal
  - Make identical transitions consecutive along the *state* dimension (DFA Reorganization)
  - Leverage bitmap technique (DFA Compression)

Algorithm: RCDFA

- Reorganize DFA
  - DFA matrix transform
  - Cluster similar states adjacent

$$\Delta_{N*M}\xrightarrow[Elementary\ \ Transform]{\begin{bmatrix}E&&&&\\&0&\cdots&1&\\&\vdots&E&\vdots&\\&1&\cdots&0&\\&&&&E\end{bmatrix}_{M*M}\begin{array}{l}\\(\mu)\\\\(\nu)\\\\\end{array}\cdots\begin{bmatrix}E&&&&\\&1&\cdots&\beta&\\&&E&\vdots&\\&&&1&\\&&&&E\end{bmatrix}_{M*M}\begin{array}{l}\\(\mu)\\\\(\nu)\\\\\end{array}}\Delta'_{N*M}$$

0	0	0	1	0	0	0	0
0	0	0	1	2	0	0	0
0	0	0	1	0	3	0	0
4	5	6	7	8	9	3	3
4	5	6	10	8	9	3	3
4	5	6	7	11	9	3	3
4	5	6	7	8	12	3	3
13	14	14	7	2	0	14	14
8	15	8	1	8	0	8	8
9	9	16	1	0	9	9	9
13	14	14	17 [1]	2	0	14	14
8	15	8	1	18 [2]	0	8	8
9	9	16	1	0	19 [3]	9	9
13	14	14	10	0	0	14	14
13	14	14	7	0	0	14	14
8	15	8	1	11	0	8	8
9	9	16	1	0	12	9	9
13	14	14	7	2	0	14	14
8	15	8	1	8	0	8	8
9	9	16	1	0	9	9	9

1	1	1	2	1	13	1	1
1	1	1	2	1	1	1	1
1	1	1	2	0	1	1	1
3	5	3	2	3	1	3	3
3	5	3	2	3	1	3	3
3	5	3	2	6	1	3	3
3	5	3	2	4 [2]	1	3	3
7	7	10	2	1	7	7	7
7	7	10	2	1	7	7	7
7	7	10	2	1	8 [3]	7	7
7	7	10	2	1	9	7	7
14	12	11	17	3	9	13	13
14	12	11	17	6	7	13	13
14	12	11	17	3	7	13	13
14	12	11	19	3	7	13	13
15	16	16	19	1	1	16	16
15	16	16	17	1	1	16	16
15	16	16	17	0	1	16	16
15	16	16	17	0	1	16	16
15	16	16	18 [1]	0	1	16	16

Algorithm: RCDFA

- Compress DFA
  - *State Bitmap* for primary redundancy among states
  - *Character Mapping* for residual redundancy inside states

←MAPPING along the *character* dimension→

20-bit BITMAP along the *state* dimension

1	1	1	2	1	13	1	1
1	1	1	2	1	1	1	1
1	1	1	2	0	1	1	1
3	5	3	2	3	1	3	3
3	5	3	2	3	1	3	3
3	5	3	2	6	1	3	3
3	5	3	2	4 [2]	1	3	3
7	7	10	2	1	7	7	7
7	7	10	2	1	7	7	7
7	7	10	2	1	8 [3]	7	7
7	7	10	2	1	9	7	7
14	12	11	17	3	9	13	13
14	12	11	17	6	7	13	13
14	12	11	17	3	7	13	13
14	12	11	19	3	7	13	13
15	16	16	19	1	1	16	16
15	16	16	17	1	1	16	16
15	16	16	17	0	1	16	16
15	16	16	17	0	1	16	16
15	16	16	18 [1]	0	1	16	16

=

bmp 1	bmp 2	bmp 3	bmp 4
1	1	1	1
0	0	0	1
0	0	1	0
1	0	1	0
0	0	0	0
0	0	1	0
0	0	1	0
1	0	1	1
0	0	0	0
0	0	0	1
0	0	0	1
1	1	1	0
0	0	1	1
0	0	1	0
0	1	0	0
1	0	1	1
0	1	0	0
0	0	1	0
0	0	0	0
0	1	0	0

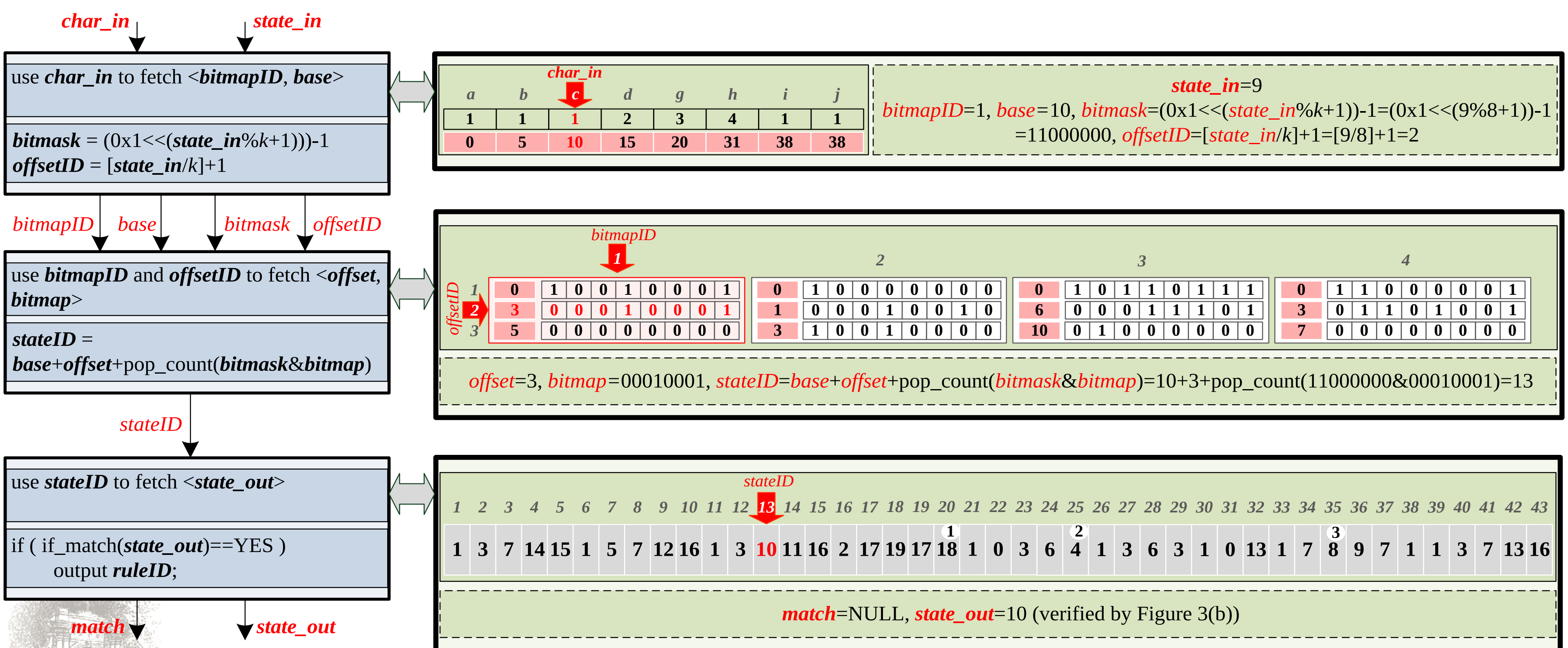
+

bmp 1	bmp 1	bmp 1	bmp 2	bmp 3	bmp 4	bmp 1
1	1	1	2	1	13	1
3	5	3	17	0	1	3
7	7	10	19	3	7	7
14	12	11	17	6	8 [3]	13
15	16	16	18 [1]	4 [2]	9	16
				1	7	
				3	1	
				6		
				3		
				1		
				0		

Algorithm: RCDFA

- Pipeline mapping architecture
  - 3-stage memory access and table lookup
    - Index mapping table, Bitmap table, Transition table

Traversal time=O(N)
Time complexity=O(nlogn)
Space complexity=O(n)



Algorithm: RCDFA

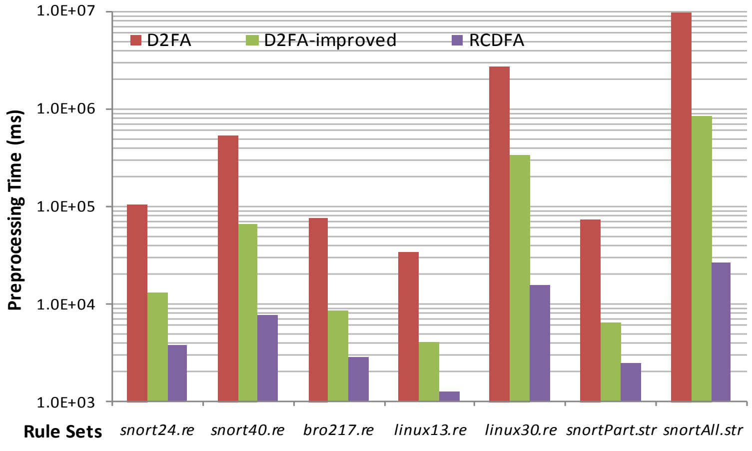
COMPRESSION RATE (% TRANSITION REDUCTION)

Rule Sets	# OF TRANSITIONS	Bitmap	D^2FA	D^2FA-improved	RCDFA
Snort24.re	2133760	88.87	91.04	98.73	99.01
Snort40.re	4868864	92.10	82.68	98.13	98.89
Bro217.re	1672448	70.91	76.09	98.94	98.57
Linux13.re	1246976	96.17	57.12	27.86	96.56
Linux30.re	11148032	90.84	80.94	34.88	96.79
SnortPart.str	1449472	31.88	83.88	99.21	98.69
SnortAll.str	14407680	17.86	84.11	99.22	98.90

NUMBER OF STATE ACCESSED PER INPUT CHARACTER

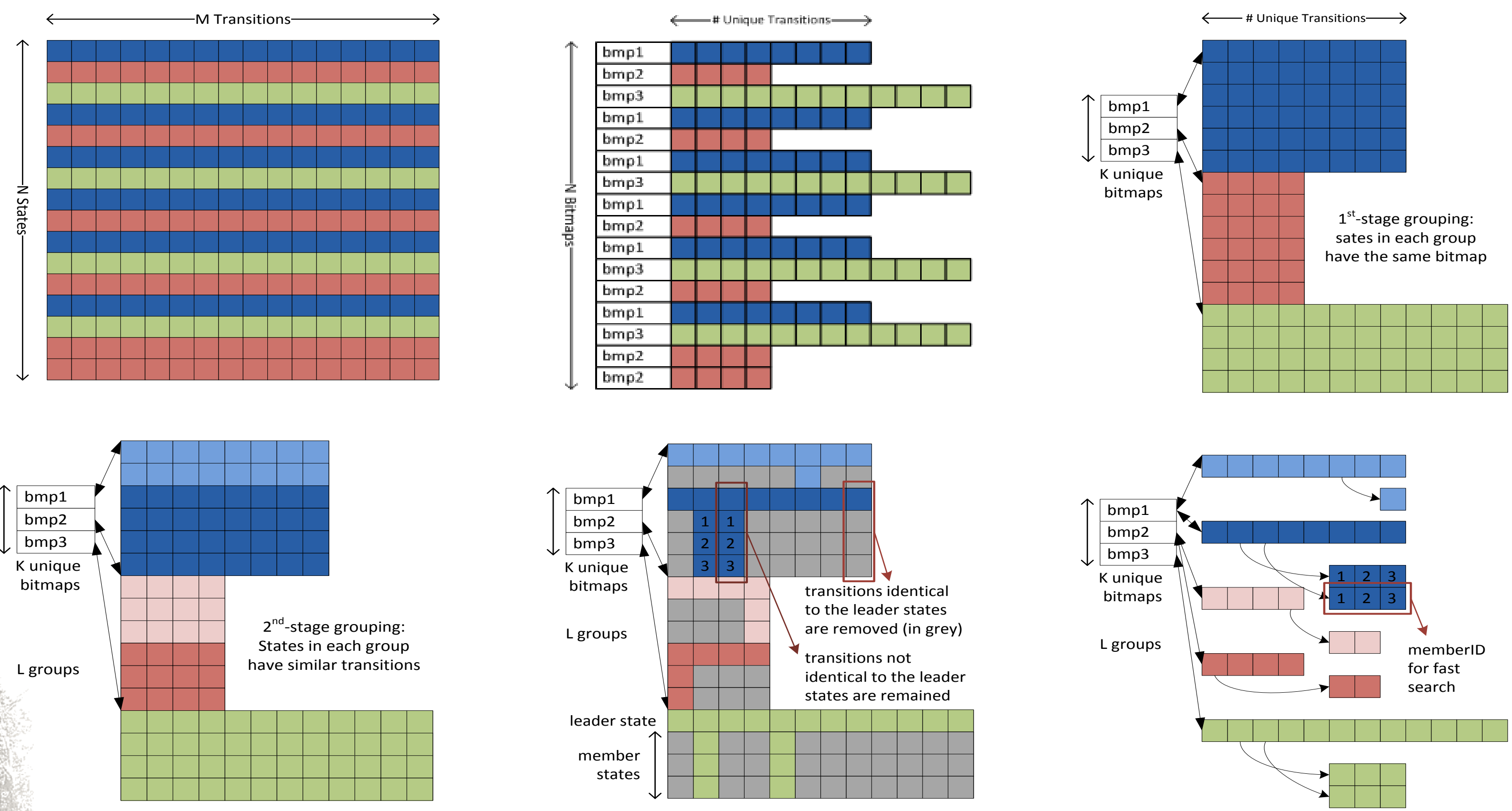
Rule Sets	# OF STATES	D^2FA		D^2FA-improved		RCDFA
		Avg.	Worst	Avg.	Worst	
Snort24.re	8335	1.09	2	1.98	9	1
Snort40.re	19019	1.94	2	1.98	7	1
Bro217.re	6533	1.18	2	1.43	9	1
Linux13.re	4871	1.03	2	1.73	6	1
Linux30.re	43547	1.04	2	1.99	13	1
SnortPart.str	5662	1.39	2	1.74	7	1
SnortAll.str	56280	1.44	2	1.78	10	1

O(Nlog$N$) preprocessing time complexity



Algorithm: FEACAN

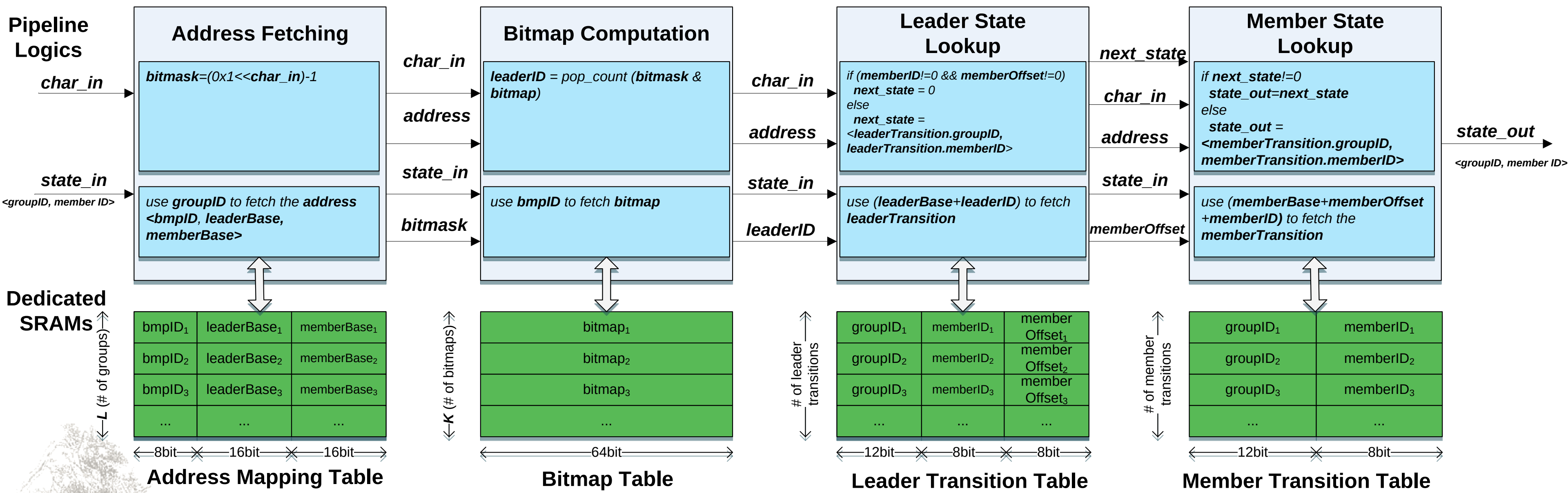
- 2-D compression
  - Intra-state compression
    - compress the number of transitions inside each state via bitmap technique
  - Inter-state compression
    - group similar states sharing identical bitmaps and encode redundant transitions in each group



Algorithm: FEACAN

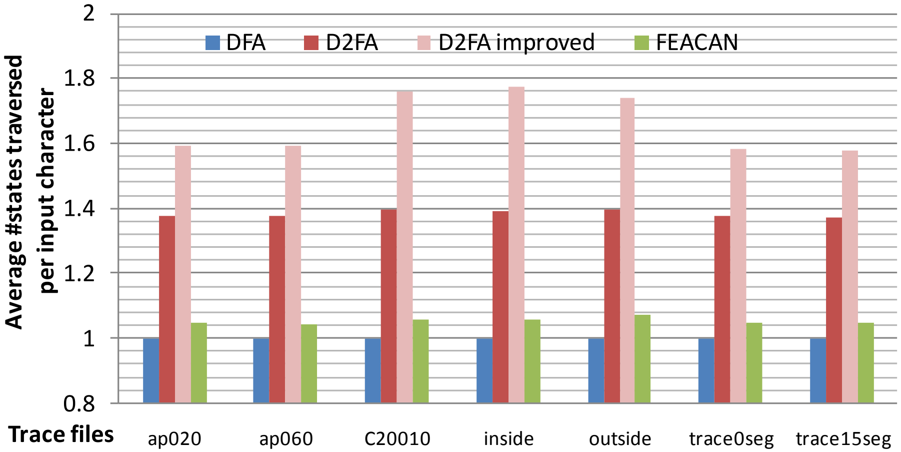
- Pipeline mapping architecture
  - 4-stage memory access and table lookup
    - Address mapping table, Bitmap table, Leader Transition table, Member transition table

Traversal time=O(N)
Time complexity=$O(n^{2}/L)$
Space complexity=O(n)



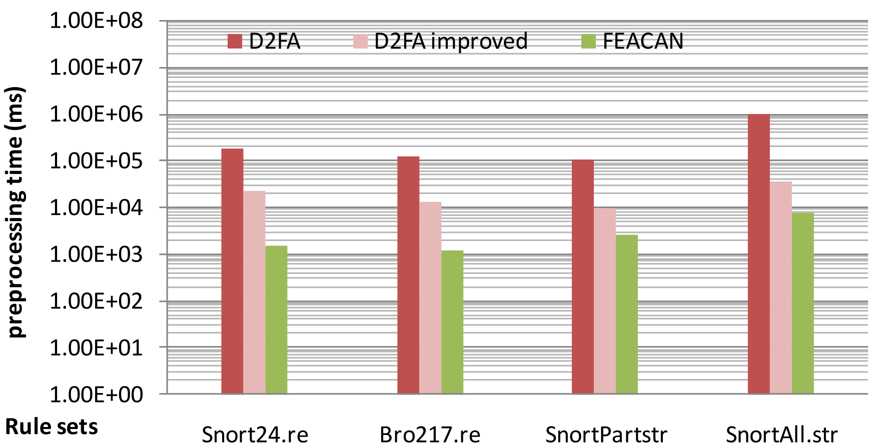
Algorithm: FEACAN

Average 1.03 states access per byte

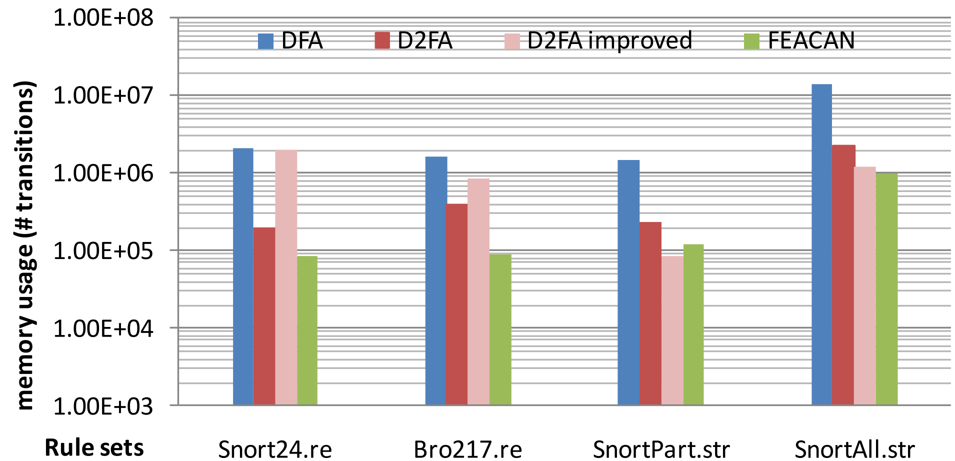


O(N^2 /L) preprocessing time complexity

(L is # of groups)

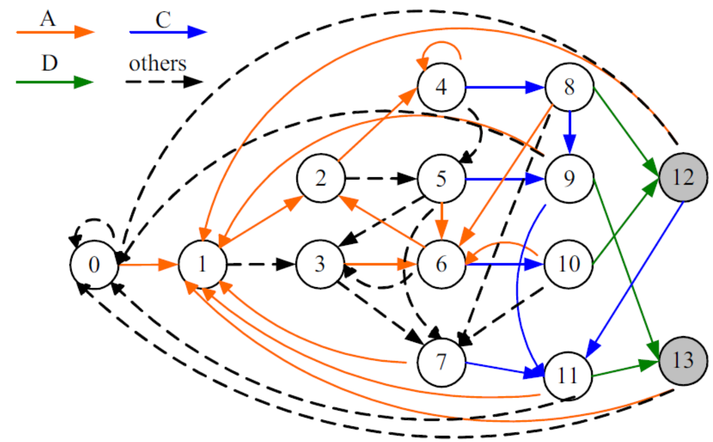


Over 90% trans compression ratio



Algorithm: Cluster-DFA

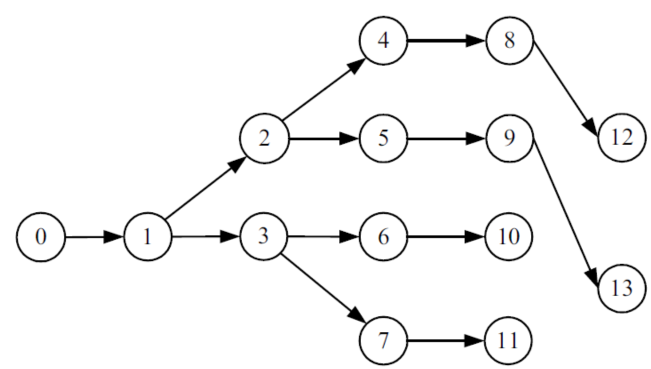
- Cluster-based observation
  - cluster
    - a states set which is composed of all son states of a certain state in the trie-tree



RegEx: ***A.{2}CD***

TRANSITION CHARACTERISTIC INSIDE STATES

pattern set	average distinct "next-states"	average distinct clusters	% transitions in Top-2 clusters
snort-24	14.49	5.06	97.89
snort-31	12.24	4.75	97.92
snort-34	13.11	4.69	98.21
bro-217	54.29	4.23	97.95
type-1	47.79	1.93	99.96



Trie-tree

Algorithm: Cluster-DFA

- Splitting Compression
  - split DFA matrix into T1, T2, T3 matrix
    - According to the TOP-2 clusters

DFA matrix

State	A	C	D	^
0	1	0	0	0
1	2	3	3	3
2	4	5	5	5
3	6	7	7	7
4	4	8	5	5
5	6	9	7	7
6	2	10	3	3
7	1	11	0	0
8	6	9	12	7
9	1	11	13	0
10	6	7	12	7
11	1	0	13	0
12	1	11	0	0
13	1	0	0	0

T1 matrix

State	A	C	D	^
0	X	0	0	0
1	2	3	3	3
2	4	5	5	5
3	6	7	7	7
4	4	X	5	5
5	6	X	7	7
6	2	X	3	3
7	X	X	0	0
8	6	X	X	7
9	X	X	X	0
10	6	7	X	7
11	X	0	X	0
12	X	X	0	0
13	X	0	0	0

T2 matrix

State	A	C	D	^
0	1	X	X	X
1	X	X	X	X
2	X	X	X	X
3	X	X	X	X
4	X	8	X	X
5	X	9	X	X
6	X	10	X	X
7	1	X	X	X
8	X	9	X	X
9	1	X	X	X
10	X	X	12	X
11	1	X	X	X
12	1	X	X	X
13	1	X	X	X

T3 matrix

State	A	C	D	^
0	X	X	X	X
1	X	X	X	X
2	X	X	X	X
3	X	X	X	X
4	X	X	X	X
5	X	X	X	X
6	X	X	X	X
7	X	11	X	X
8	X	X	12	X
9	X	11	13	X
10	X	X	X	X
11	X	X	13	X
12	X	11	X	X
13	X	X	X	X

Algorithm: Cluster-DFA

- Cluster-based Splitting Compression
  - compress each matrix
    - bitmap for T1, T2
    - sparse matrix compression algorithm for T3

state	A	C	D	^
0	X	0	0	0
1	2	3	3	3
2	4	5	5	5
3	6	7	7	7
4	4	X	5	5
5	6	X	7	7
6	2	X	3	3
7	X	X	0	0
8	6	X	X	7
9	X	X	X	0
10	6	7	X	7
11	X	0	X	0
12	X	X	0	0
13	X	0	0	0

⇨

state	A	C	D	^	base
0	X	0	0	0	0
1	0	1	1	1	2
2	0	1	1	1	4
3	0	1	1	1	6
4	0	X	1	1	4
5	0	X	1	1	6
6	0	X	1	1	2
7	X	X	0	0	0
8	0	X	X	1	6
9	X	X	X	0	0
10	0	1	X	1	6
11	X	0	X	0	0
12	X	X	0	0	0
13	X	0	0	0	0

⇨

state	A	C	D	^	base	equal
0	X	0	0	0	0	0
1	0	1	1	1	2	1
					4	1
					6	1
					4	1
					6	1
					2	1
					0	0
					6	1
					0	0
					6	1
					0	0
					0	0
					0	0

Algorithm 2 Pseudo-code for getting next state of CSCA for current state *cur* and the input character *c*

if $bitmap[cur][c] == 1$ **then**
  return $R1[equal1[cur]][c] + base1[cur]$
else if $(temp = R3[cur][c]) \neq' X'$ **then**
  return temp
else
  return $R2[equal2[cur]][c] + base2[cur]$
end if

Algorithm: Cluster-DFA

- Time:
  - at least 4 memory accesses per input character
    - *bitmap, equal1, R1, base1*
- Space:

COMPARISON IN TERMS OF SPATIAL COMPRESSION RATIO

Group name	original DFA		our compression algorithm (CSCA)				SCR of	SCR of
	n	SCR	$n1$	$n2$	r	SCR	δ FA	Default_Row
l7filter-1	3172	1.0	52	22	0.024621	**0.070756**	0.634964	0.232905
l7filter-3	30135	1.0	3	21	0.011429	**0.051420**	0.960985	0.356860
l7filter-4	22608	1.0	7	44	0.054823	**0.095459**	0.097177	0.381078
snort-24	13882	1.0	13	34	0.021074	0.062055	**0.037515**	0.108468
snort-31	19522	1.0	7	34	0.020840	0.061391	**0.053581**	0.061309
snort-34	13834	1.0	6	17	0.017938	0.058231	**0.032259**	0.060473
bro-217	6533	1.0	1	14	0.020456	**0.060557**	0.061814	0.224820
type-1	249	1.0	1	2	0.000016	**0.042212**	0.111281	0.186697
type-2	78337	1.0	512	2	0.000102	0.042026	0.099659	**0.030254**
type-3	8338	1.0	43	5	0.002395	0.043842	0.948123	**0.018575**
type-4	5290	1.0	236	4	0.012469	0.064080	0.990808	**0.046357**
type-5	7828	1.0	1	22	0.002451	0.041732	0.947048	**0.019762**
type-6	14496	1.0	9	22	0.002300	**0.042061**	0.973929	0.173284

Chain

0	0	0	1	0	0	0	0
0	0	0	1	2	0	0	0
0	0	0	1	0	3	0	0
4	5	6	7	8	9	3	3
4	5	6	10	8	9	3	3
4	5	6	7	11	9	3	3
4	5	6	7	8	12	3	3
13	14	14	7	2	0	14	14
8	15	8	1	8	0	8	8
9	9	16	1	0	9	9	9
13	14	14	17 [1]	2	0	14	14
8	15	8	1	18 [2]	0	8	8
9	9	16	1	0	19 [3]	9	9
13	14	14	10	0	0	14	14
13	14	14	7	0	0	14	14
8	15	8	1	11	0	8	8
9	9	16	1	0	12	9	9
13	14	14	7	2	0	14	14
8	15	8	1	8	0	8	8
9	9	16	1	0	9	9	9

DFA

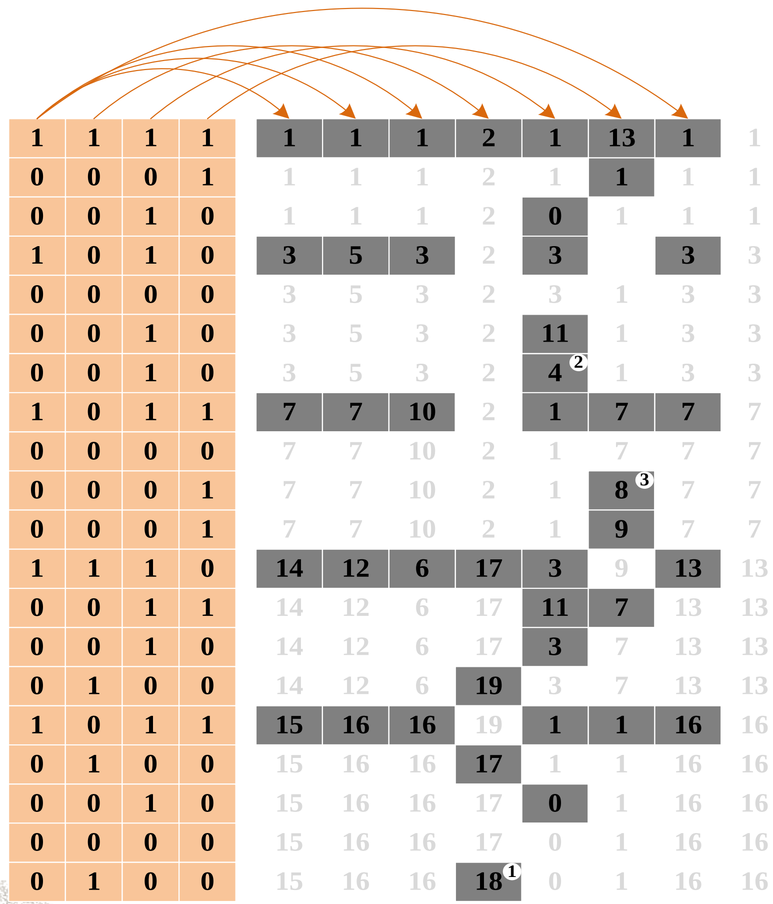
0	a, 0	b, 0	c, 0	d, 1	e, 0	f, 0	g, 0	h, 0
0	e, 2							
0	f, 3							
3	a, 4	b, 5	c, 6	d, 7	e, 8	f, 9	g, 3	h, 3
3	d,10							
3	e,11							
3	f,12							
17								
8	a,8	b,15	c, 8	d, 1	e, 8	f, 0	g, 8	h, 8
9	a,9	b, 9	c,16	d, 1	e, 0	f, 9	g, 9	h, 9
17	d,17 [1]							
8	e,18 [2]							
9	f,19 [3]							
14	d,10							
14	a,13	b,14	c,14	d, 7	e, 0	f, 0	g,14	h,14
8	e,11							
9	f,12							
17	a,13	b,14	c,14	d, 7	e, 2	f, 0	g,14	h,14
8								
9								

D^2FA

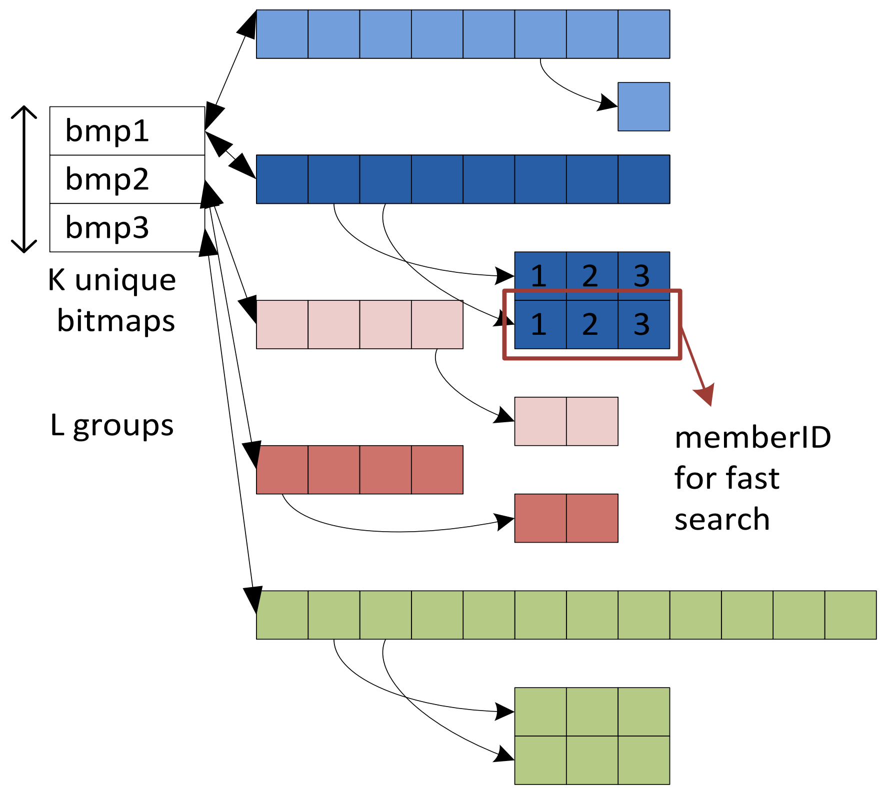
0	a, 0	b, 0	c, 0	d, 1	e, 0	f, 0	g, 0	h, 0
0	e, 2							
0	f, 3							
0	a, 4	b, 5	c, 6	d, 7	e, 8	f, 9	g, 3	h, 3
3	d,10							
3	e,11							
3	f,12							
0	a,13	b,14	c,14	d, 7	e, 2	g,14	h,14	
0	a,8	b,15	c, 8	e, 8	g, 8	h, 8		
0	a,9	b, 9	c,16	f, 9	g, 9	h, 9		
7	d,17 [1]							
8	e,18 [2]							
9	f,19 [3]							
7	d,10							
7	e, 0							
8	e,11							
9	f,12							
7								
8								
9								

A-DFA

Chain



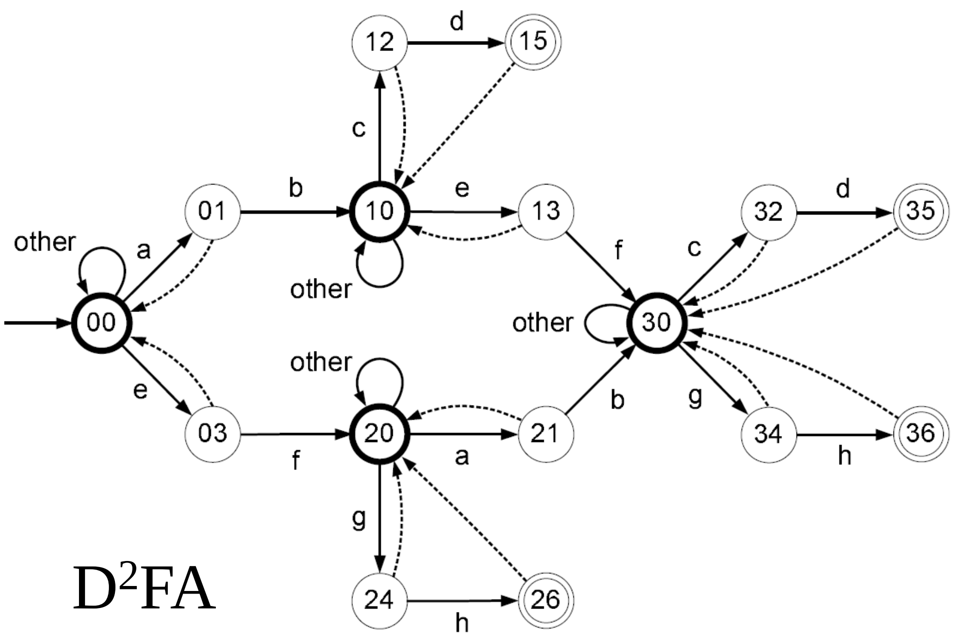
RCDFA



FEACAN

Algorithm: TCAM-based

- TID (Template ID)
  - chain
- PID (Private ID)
  - state in each chain



D^{2}FA

TCAM Source State TID	TCAM Source State PID	TCAM Input	SRAM Destination State TID	SRAM Destination State PID
1000	001	b	0100	000
1000	011	f	0010	000
1000	***	a	1111	001
1000	***	e	1111	011
1000	***	*	1111	000
0100	010	d	1111	101
0100	011	f	0001	000
0100	***	c	1111	010
0100	***	e	1111	011
0100	***	*	1111	000
0010	001	b	0001	000
0010	100	h	1111	110
0010	***	a	1111	001
0010	***	g	1111	100
0010	***	*	1111	000
0001	010	d	1111	101
0001	100	h	1111	110
0001	***	c	1111	010
0001	***	g	1111	100
0001	***	*	1111	000

Algorithm: TCAM-based

- TCAM-based DFA deflation

TCAM			SRAM	
Source State		Input	Destination State	
TID	PID		TID	PID
*0*0	***	a	1111	001
1000	001	b	0100	000
0010	001	b	0001	000
0*0*	***	c	1111	010
0*0*	010	d	1111	101
**00	***	e	1111	011
1000	011	f	0010	000
0100	011	f	0001	000
00**	***	g	1111	100
00**	100	h	1111	110
****	***	*	1111	000

Algorithm: TCAM-based

- Evaluation

Why so many bits?

NUMBER OF TCAM ENTRIES USED.

	Chen's method	Meiners's method	Number of DFA states	Our method	Number of NFA states
Bro-217	21,872	9,118	6,533	4,659	2,131
Snort-34	34,508	16,293	13,825	2,095	887
Snort-24	38,418	16,144	13,886	5,139	575
Snort-31	72,662	41,487	20,068	9,168	912

NUMBER OF BITS USED PER ENTRY.

	Chen's method		Meiners's method		Our method	
	TCAM	SRAM	TCAM	SRAM	TCAM	SRAM
Bro-217	26	18	31	23	23	15
Snort-34	25	17	28	20	70	62
Snort-24	26	18	26	18	52	44
Snort-31	26	18	34	26	57	49

TOTAL NUMBER OF BITS USED (MEGABIT).

	Chen's method		Meiners's method		Our method	
	TCAM	SRAM	TCAM	SRAM	TCAM	SRAM
Bro-217	0.54	0.38	0.27	0.20	0.10	0.07
Snort-34	0.82	0.56	0.44	0.31	0.14	0.12
Snort-24	0.95	0.66	0.40	0.28	0.26	0.22
Snort-31	1.80	1.25	1.35	1.03	0.50	0.43

Conclusion

- What make the combination of coded transitions efficient?
  - *inside chain* & *outside chain*, previous methods only take *inside chain* into account
  - the same input character and similar coding of TID, meanwhile the return state is the template state
  - as long as the coding is good enough, the result of RCDFA will be better, maybe O(|Σ|)

1	1	1	1	1	1	1	2	1	13	1	1
0	0	0	1	1	1	1	2	1	1	1	1
0	0	1	0	1	1	1	2	0	1	1	1
1	0	1	0	3	5	3	2	3		3	3
0	0	0	0	3	5	3	2	3	1	3	3
0	0	1	0	3	5	3	2	11	1	3	3
0	0	1	0	3	5	3	2	4 [2]	1	3	3
1	0	1	1	7	7	10	2	1	7	7	7
0	0	0	0	7	7	10	2	1	7	7	7
0	0	0	1	7	7	10	2	1	8 [3]	7	7
0	0	0	1	7	7	10	2	1	9	7	7
1	1	1	0	14	12	6	17	3	9	13	13
0	0	1	1	14	12	6	17	11	7	13	13
0	0	1	0	14	12	6	17	3	7	13	13
0	1	0	0	14	12	6	19	3	7	13	13
1	0	1	1	15	16	16	19	1	1	16	16
0	1	0	0	15	16	16	17	1	1	16	16
0	0	1	0	15	16	16	17	0	1	16	16
0	0	0	0	15	16	16	17	0	1	16	16
0	1	0	0	15	16	16	18 [1]	0	1	16	16