02/10/2025

Quiz 2

Hopcroft Alg

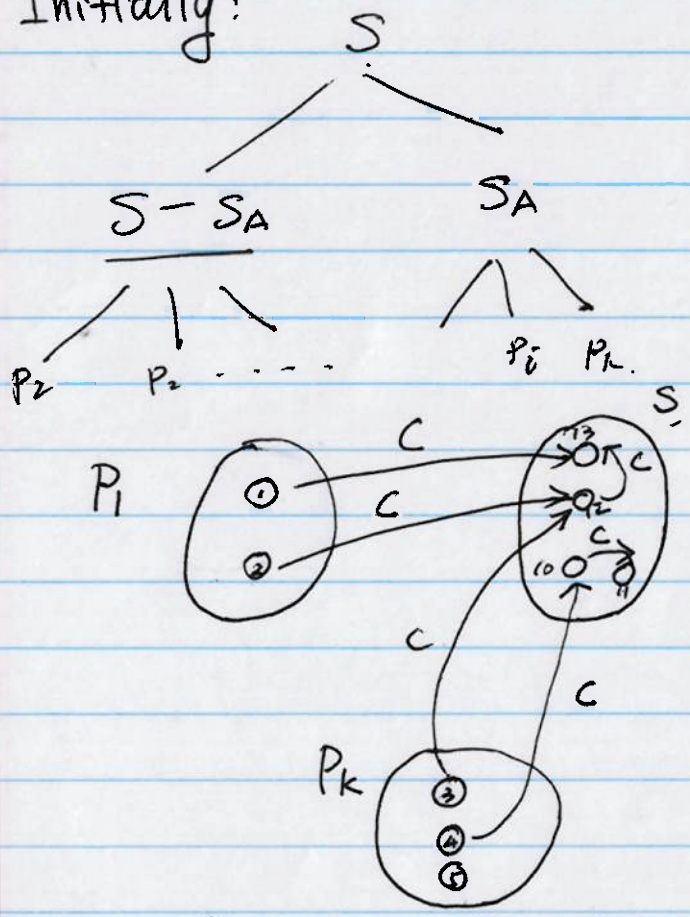
RE ⟶ NFA ⟶ DFA ⟶ minimal DFA

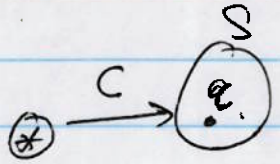
Given. D, a DFA. reduce # of states. from D

by iteratively partition the states into equivalent (not distinguish) classes.

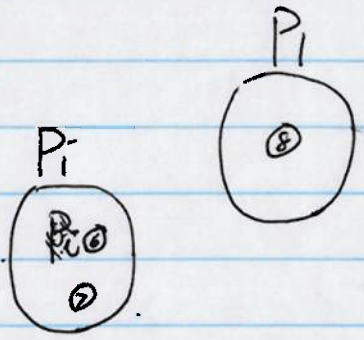
$$D = \{ S, \Sigma, \delta, s_0, S_A \}.$$

Initially:





$\delta(*, c) = q, \quad q \in S$



in terms of S, C.

P_1 : ①, ② are equivalent

both have transitions label with c to a state in S.

P_i: (6) (7)

(6). (7). are equivalent.
neither has transitions to
S. with label c.

P_K: (3) (4) (5)

(3). (4). have . - - - -

(5) doesn't have . . .

therefore. P_K should be partitioned
into. P_{K1} (3) (4) and P_{K2} (5)

S. (12) —c→ (13), (10) —c→ (11)

(12). (10). are equivalent
(have :/trans — —

(11). (13) are equivalent
(no trans — ;)

therefore S. should be partitioned
into S_1 (10), (12) and S_2 (11), (13)

partition ← { S_A , $S - S_A$ }

worklist ← { S_A , $S - S_A$ }.

while worklist ≠ ϕ

* remove an element S from worklist

for each $c \in \Sigma$

image ← { $x \mid \delta(x, c) \in S$ }

for each partition q

$q_1 = q \cap$ image

$q_2 = q - q_1$

if $q_1 \neq \phi$ and $q_2 \neq \phi$

remove q from partition.

add q_1, q_2 into partition.

if q is in worklist

remove q from worklist

add q_1, q_2 to worklist

else if. $|q_1| \leq |q_2|$

then add q_1 to the worklist

else add q_2 to the worklist

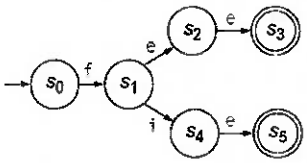
if $S = q$

goto *

02/12/2025

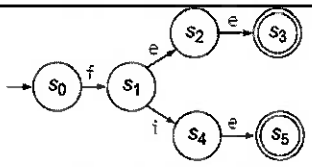
~~2/13/2025~~

36

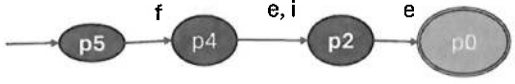


Step	Partition on Entry	Worklist	s	c	Image	q	q_1	q_2	Action
—	p_0: {s_3, s_5}, p_1: {s_0, s_1, s_2, s_4}	p_0, p_1	—	—	—	—	—	—	—
1	p_0: {s_3, s_5}, p_1: {s_0, s_1, s_2, s_4}	p_1	p_0	e	s_2, s_4	p_1 P0 P2 P3	s_2, s_4 {} p2 {}	s_0, s_1 p0 {} p3	*split* $p_1 \rightarrow p_2, p_3$ none none none
	p_0: {s_3, s_5}, p_2: {s_2, s_4}, p_3: {s_0, s_1}	p_2, p_3	p_0	f	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	*none*
		p_2, p_3	p_0	i	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	*none*
2	p_0: {s_3, s_5}, p_2: {s_2, s_4}, p_3: {s_0, s_1}	p_3	p_2	e	s_1	p_3 P0 ... P2 ...	s_1	s_0	*split* $p_3 \rightarrow p_4, p_5$
	p_0: {s_3, s_5}, p_2: {s_2, s_4}, p_4: {s_1}, p_5: {s_0}	p_4, p_5	p_2	f	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	*none*
		p_4, p_5	p_2	i	s_1	$\emptyset$	$\emptyset$	$\emptyset$	*none*

1



(a) DFA for "*fee* | *fie*"



2	p_0: {s_3, s_5}, p_2: {s_2, s_4}, p_3: {s_0, s_1}	p_3	p_2	e	s_1	p_3	s_1	s_0	*split* $p_3 \rightarrow p_4, p_5$
	p_0: {s_3, s_5}, p_2: {s_2, s_4}, p_4: {s_1}, p_5: {s_0}	p_4, p_5	p_2	f	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	*none*
		p_4, p_5	p_2	i	s_1	$\emptyset$	$\emptyset$	$\emptyset$	*none*
3	p_0: {s_3, s_5}, p_2: {s_2, s_4}, p_4: {s_1}, p_5: {s_0}	p_5	p_4	e	$\emptyset$ p_2	$\emptyset$	$\emptyset$	$\emptyset$	*none*
		p_5	p_4	f	s_0	p_5	s_0	$\emptyset$	*none*
		p_5	p_4	i	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	*none*
4	p_0: {s_3, s_5}, p_2: {s_2, s_4}, p_4: {s_1}, p_5: {s_0}	$\emptyset$	p_5	e	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	*none*
		$\emptyset$	p_5	f	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	*none*
		$\emptyset$	p_5	i	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	*none*

(a) Iterations of Hopcroft's Algorithm on the Original DFA for "fee | fie"

2

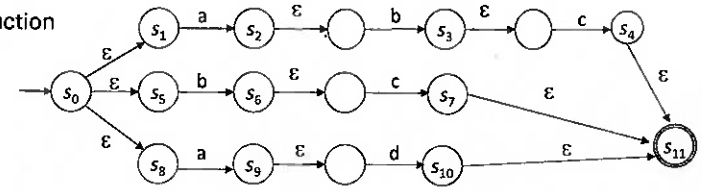
Brzozowski's Algorithm for DFA Minimization

Input:DFA

NFA from Thompson's construction

any NFA

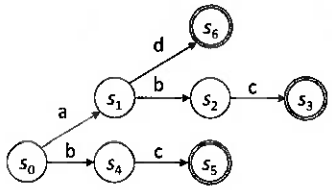
NFA for abc | bc | ad:



NFA -> DFA: subset construction

Subset construction eliminates ε-transitions and merges the paths for a.

It leaves duplicate tails, such as bc, intact



The Intuition

- The subset construction merges prefixes in the NFA.
- Can apply subset construction again to remove duplicate tails.

3

Brzozowski's Algorithm: NFA → Minimal DFA

Idea: use the subset construction twice

- For an NFA *N*
  - Let *reverse(N)* be the NFA constructed by making initial state final, adding a new start state with an ε-transition to each final state (change original final state to non-final), and reversing the other edges
  - Let *subset(N)* be the DFA produced by the subset construction on *N*
  - Let *reachable(N)* be *N* after removing any states that are not reachable from the initial state (clean up)
- Then,

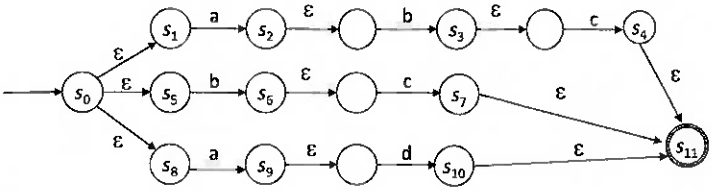
reachable(*subset*(*reverse*(*reachable*(*subset*(*reverse*(*N*))))))

is a minimal **DFA** that implements *N*

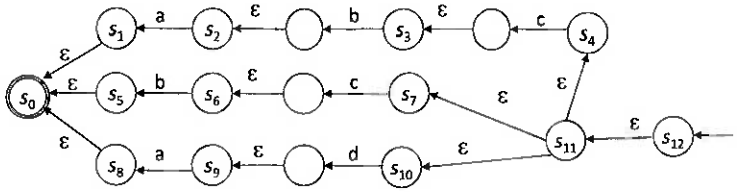
4

Brzozowski's Algorithm

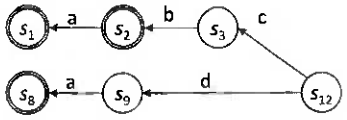
Step 1: The subset construction on *reverse(NFA)* merges suffixes in original NFA



Reversed NFA



subset(reverse(NFA))

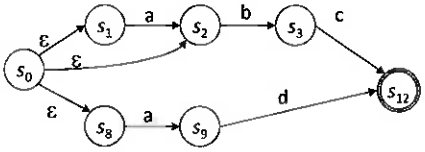


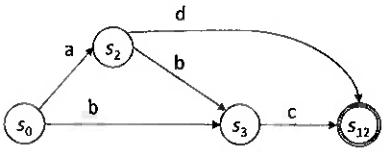
Note each state represents a subset of states, for example S_{12} is actually $\{S_{12}, S_{11}, S_4, S_7, S_{10}\}$

5

Brzozowski's Algorithm

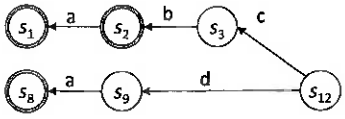
Step 2: Reverse it again & use subset to merge prefixes ...





Minimal DFA abc | bc | ad

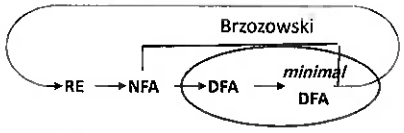
Note each state represents a subset of states, for example S_0 is actually $\{S_0, S_1, S_2, S_8\}$



Reverse it, again

And subset it, again

The Cycle of Constructions

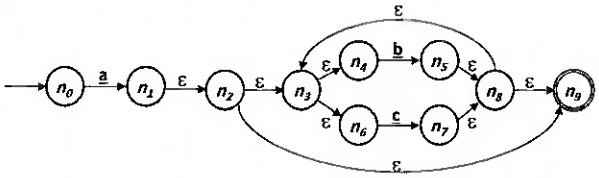


6

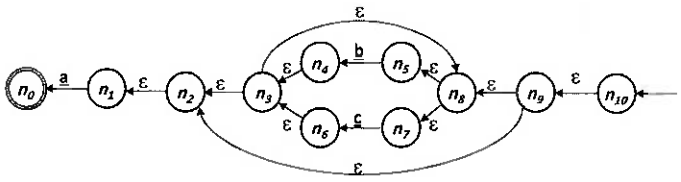
Brzozowski's Algorithm

Step 1: The subset construction on *reverse(NFA)* merges suffixes in original NFA

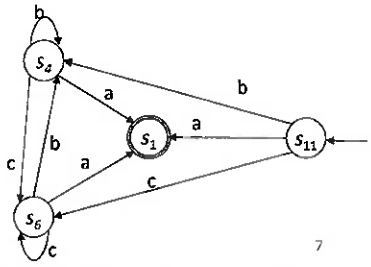
NFA for a (b | c)* :



Reversed NFA



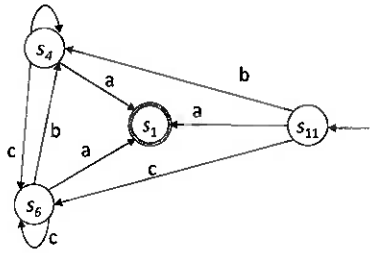
subset(reverse(NFA))



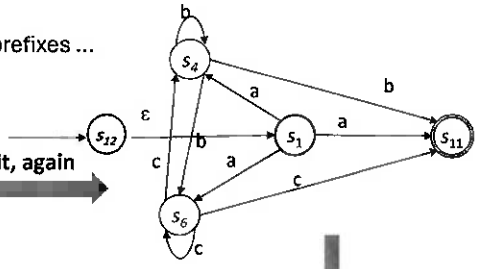
7

Brzozowski's Algorithm

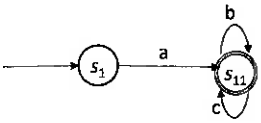
Step 2: Reverse it again & use subset to merge prefixes ...



Reverse it, again



And subset it, again



Minimal DFA a (b | c)* :

8

Limits of Regular Languages

Advantages of Regular Expressions

- Simple & powerful notation for specifying patterns
- Automatic construction of fast recognizers
  - **O**(1) cost per input character
- Many kinds of syntax can be specified with RES

Disadvantages of Regular Expressions

- Many interesting constructs are not regular
  - Balanced parentheses, nested **if-then** and **if-then-else** constructs
- The **DFA** recognizer has no real notion of grammatical structure
  - Gives no help with meaning