Computer Mathematics

Week 13

Sequencing and control

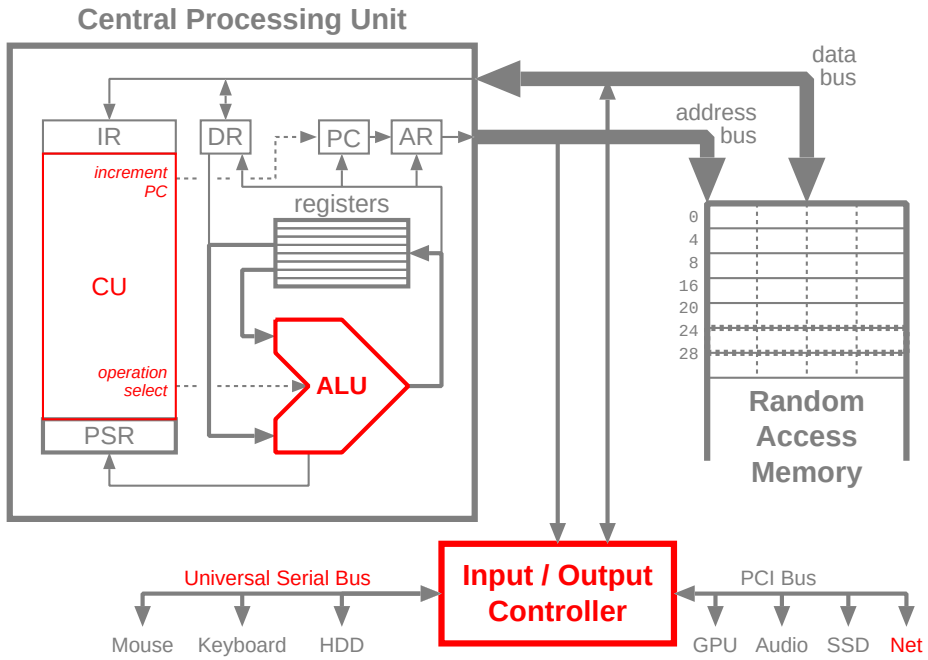
Deterministic and non-deterministic machines

KUAS 京都先端科学大学
KYOTO UNIVERSITY of ADVANCED SCIENCE
Department of Mechanical and Electrical System Engineering

last week

mathematical notation for FSMs

- regular expressions



construction of FSMs

- from arbitrary regular expressions

this week

non-deterministic machines

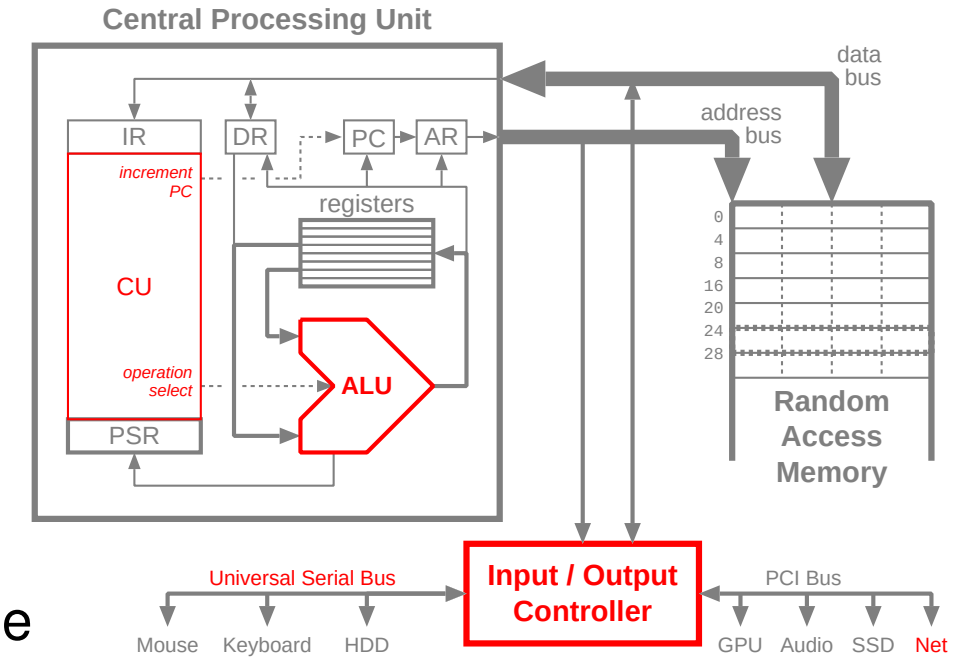
- how to simulate them

eliminating non-determinism

- make an equivalent deterministic machine
- using a clue from the simulation
  - and some real mathematics

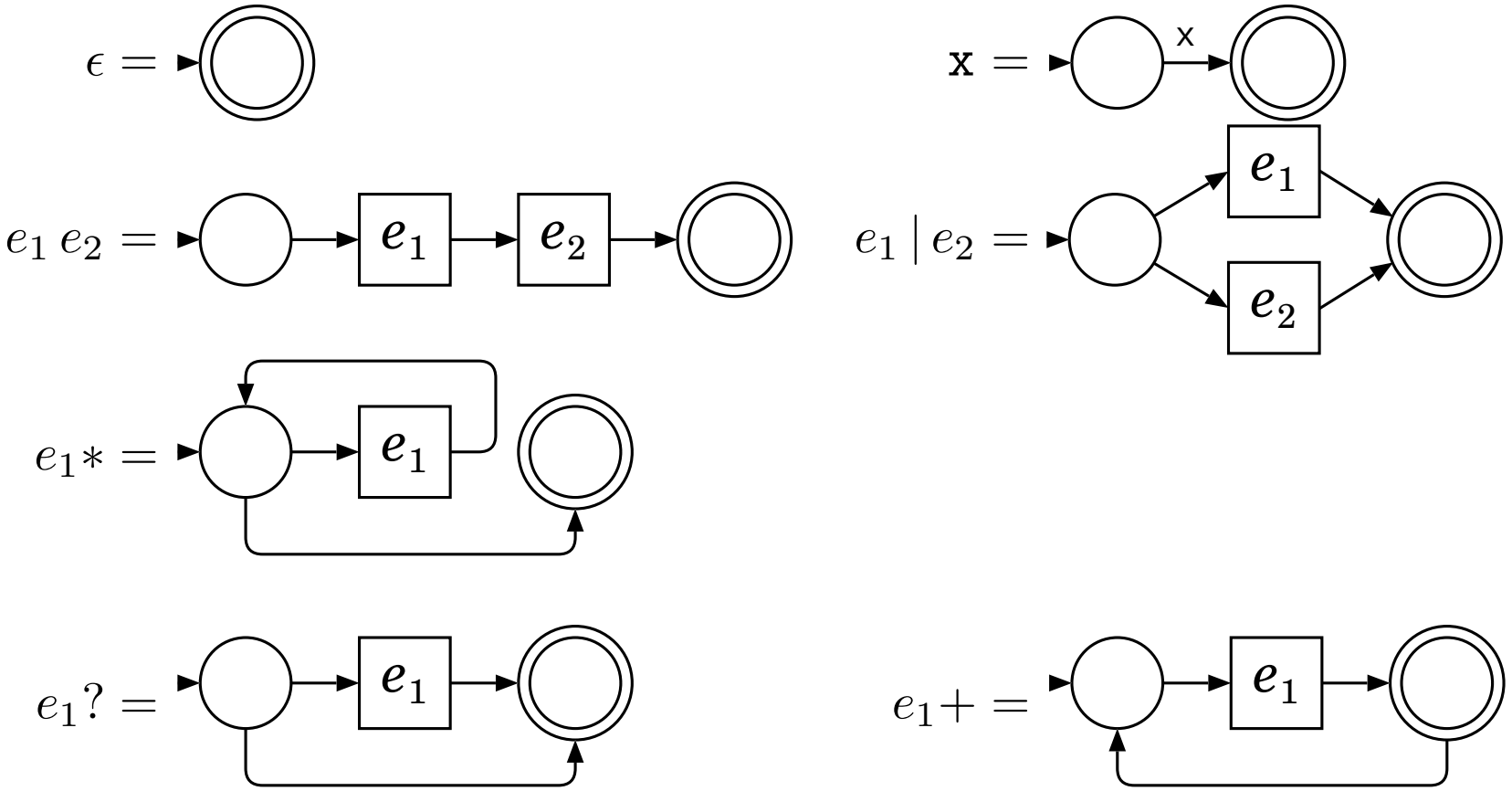
deterministic machines

- advantages and disadvantages
- relative performance



non-deterministic machines

automatic construction

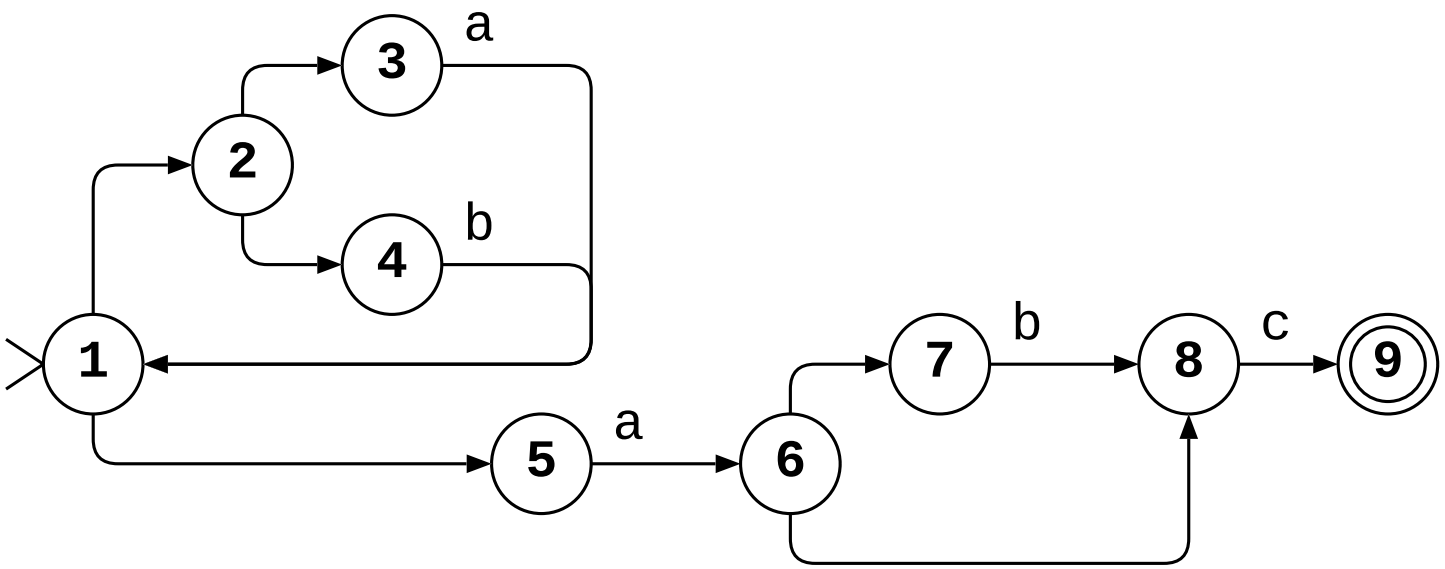


leads to machines with many ϵ-transitions

- when two (or more) transitions can be followed, we have to make a choice
- *non-determinism* means “not knowing what to do next”
- *non-deterministic finite (state) automaton* (NFA)

non-deterministic machines

(a|b)*ab?c



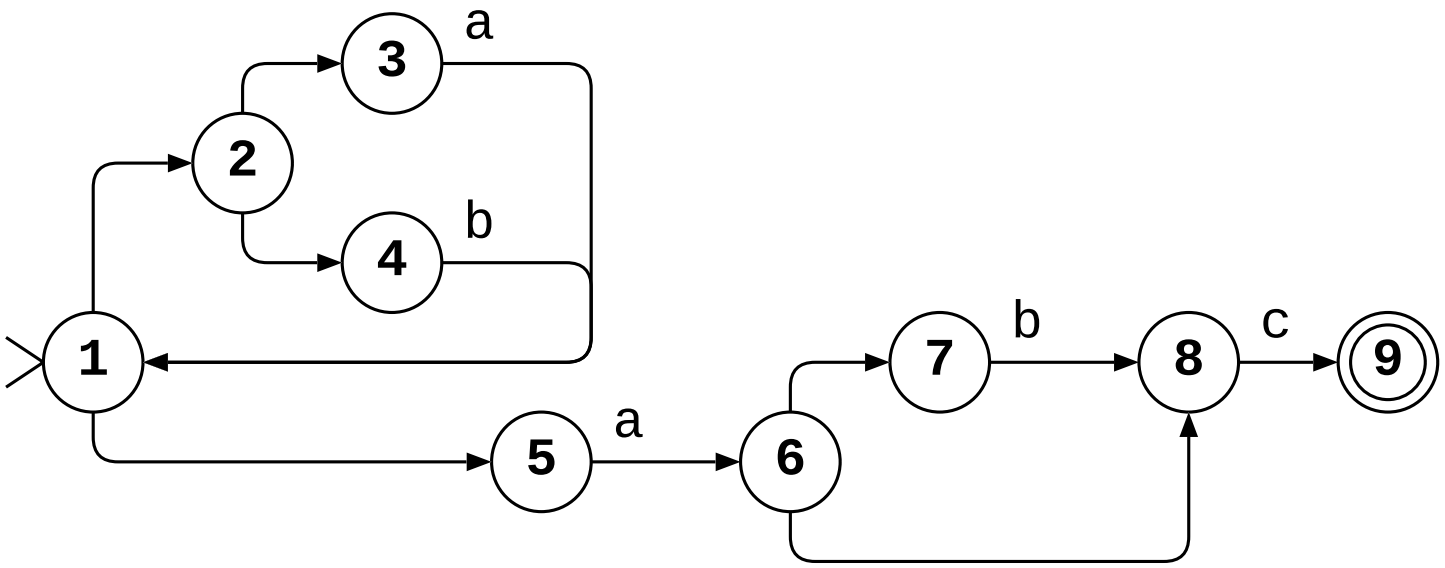
one way to ‘run’ this machine: *back-tracking*

- make arbitrary decisions when two ϵ-transitions can be followed
- use the input string to check when a labelled transition can be made
- when you get stuck, back up to the last decision and try a different one

each symbol scanned many times, many false paths explored: very inefficient

non-deterministic machines

(a|b)*ab?c



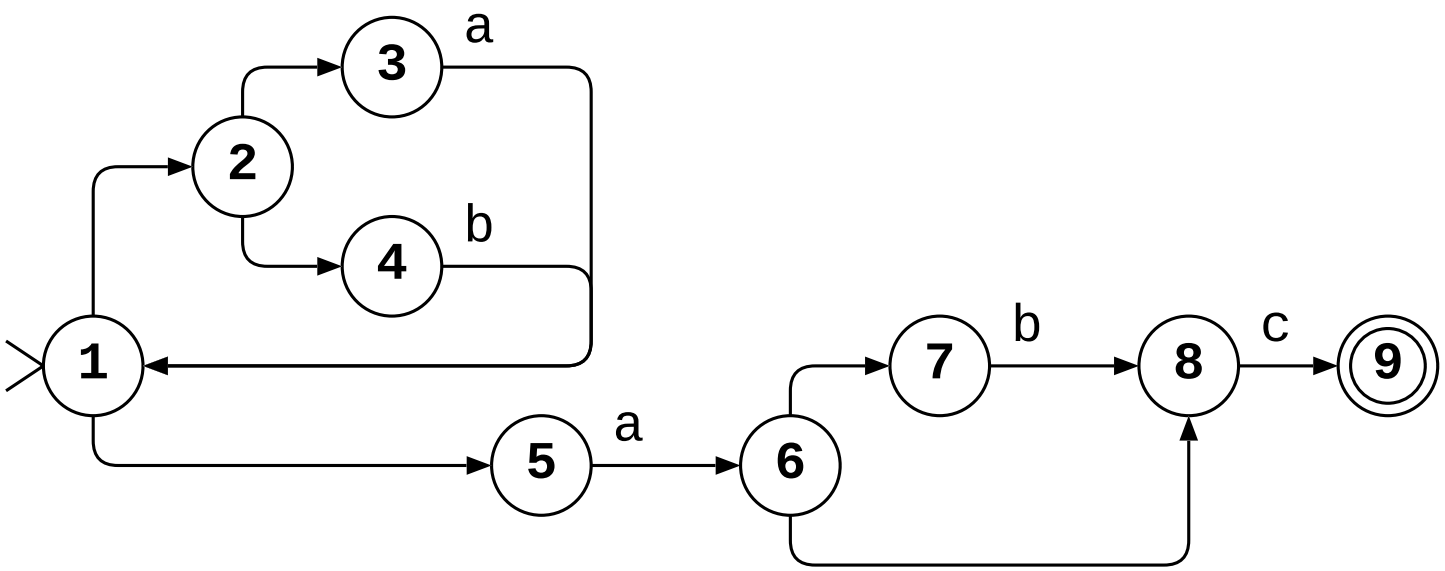
a much better way to ‘run’ this machine: follow all paths in parallel

- keep a set of possible states for each input sentence position
- follow all possible ϵ-transitions (transitively) at the current position
- use the current input symbol to decide which labelled transitions can be followed
- advance and repeat, trying to reach the ‘accepting’ state at the end of the input

each symbol scanned once, only viable paths explored: much more efficient

non-deterministic machines

(a|b)*ab?c



let's try to match "abac"

(beginning with the start state) follow all possible ϵ-transitions (transitively) at the current input position

- leads to a *set* of states with labelled transitions (which cannot be crossed... yet)

use the current input symbol to decide which of these labelled transitions can be followed

- generates a *state set* at the *next* input position
- make the next input position current one, and repeat from the first step

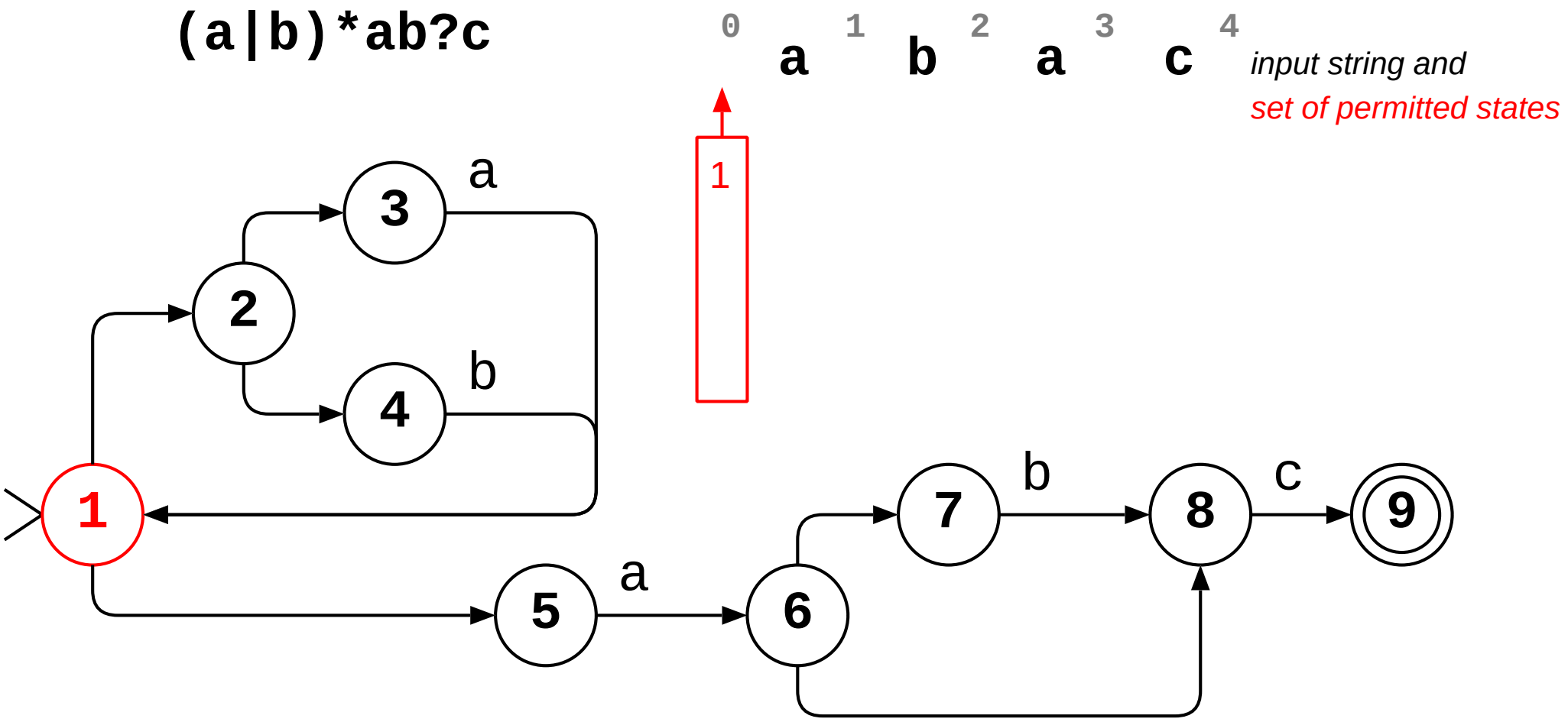
repeat until you reach the end of the input

- if any 'accepting' state is in the final state set, the machine recognised the input

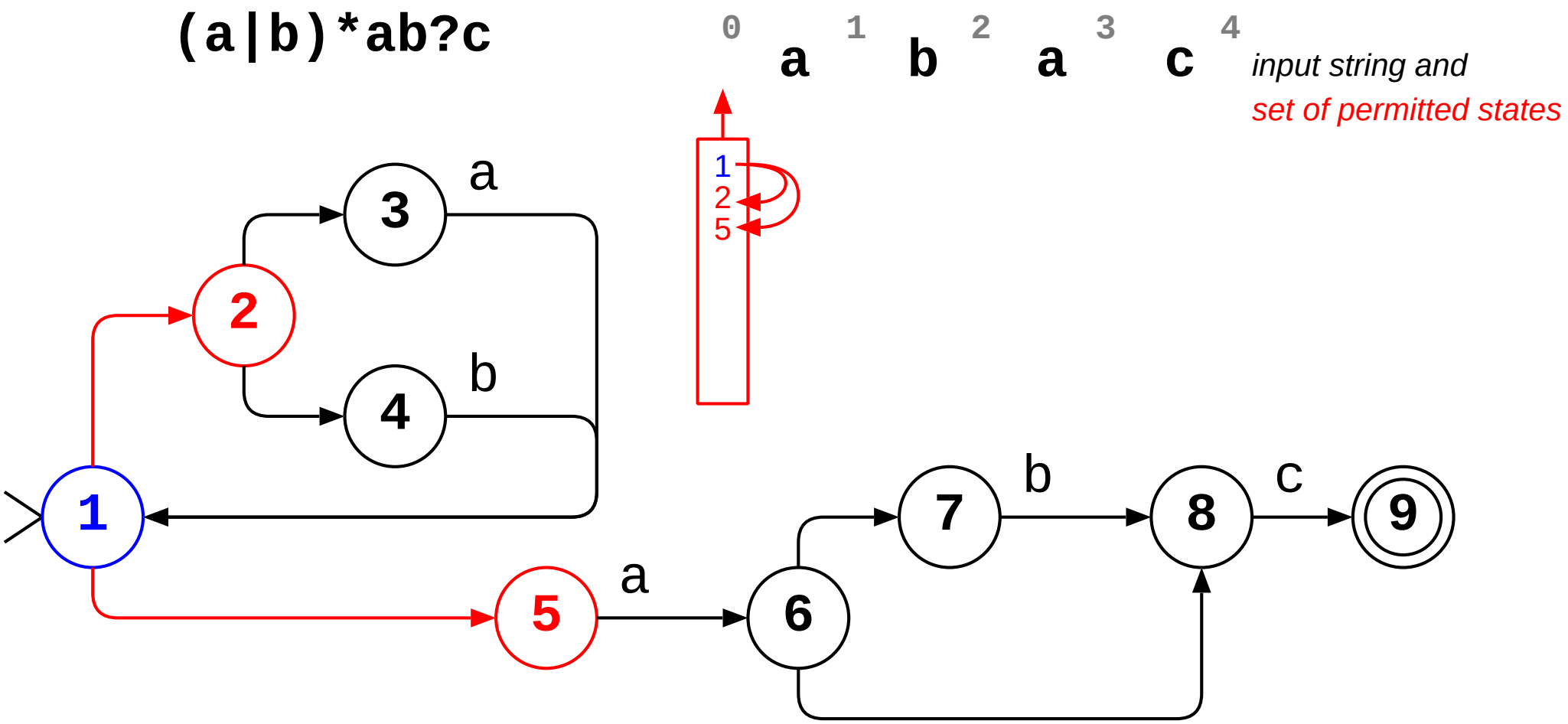
efficient recognition with NFAs

Let's try to match: "abac"

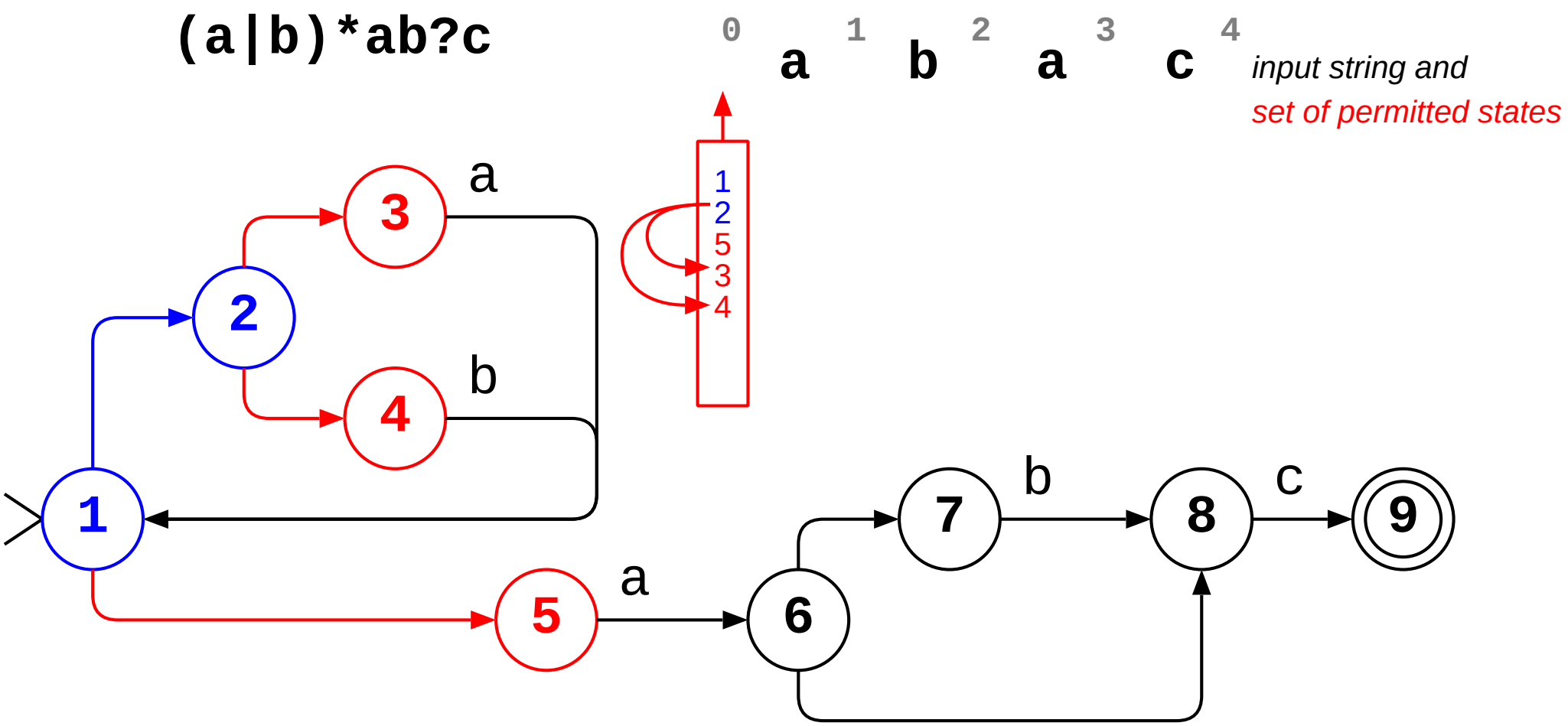
begin by adding just the start state to the first set (input position 0)



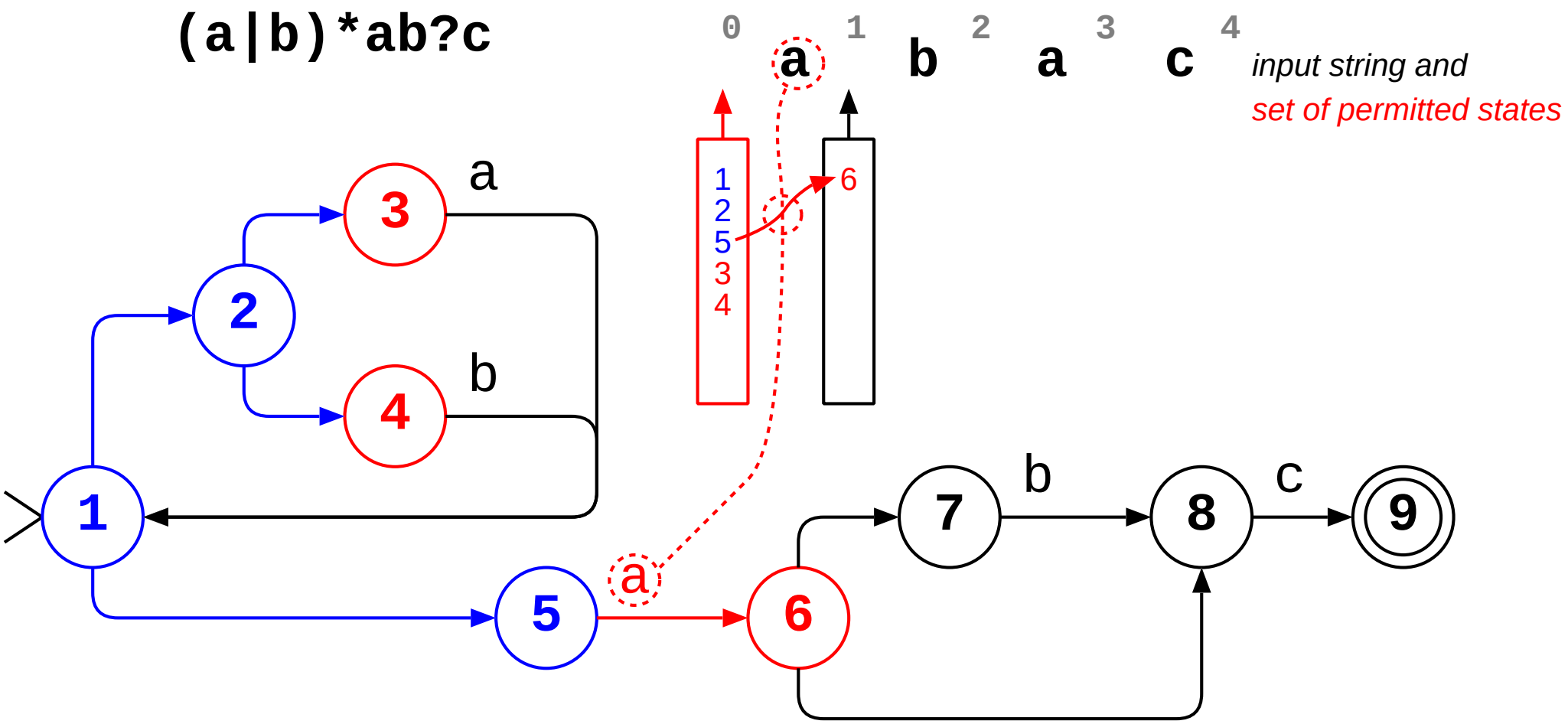
efficient recognition with NFAs



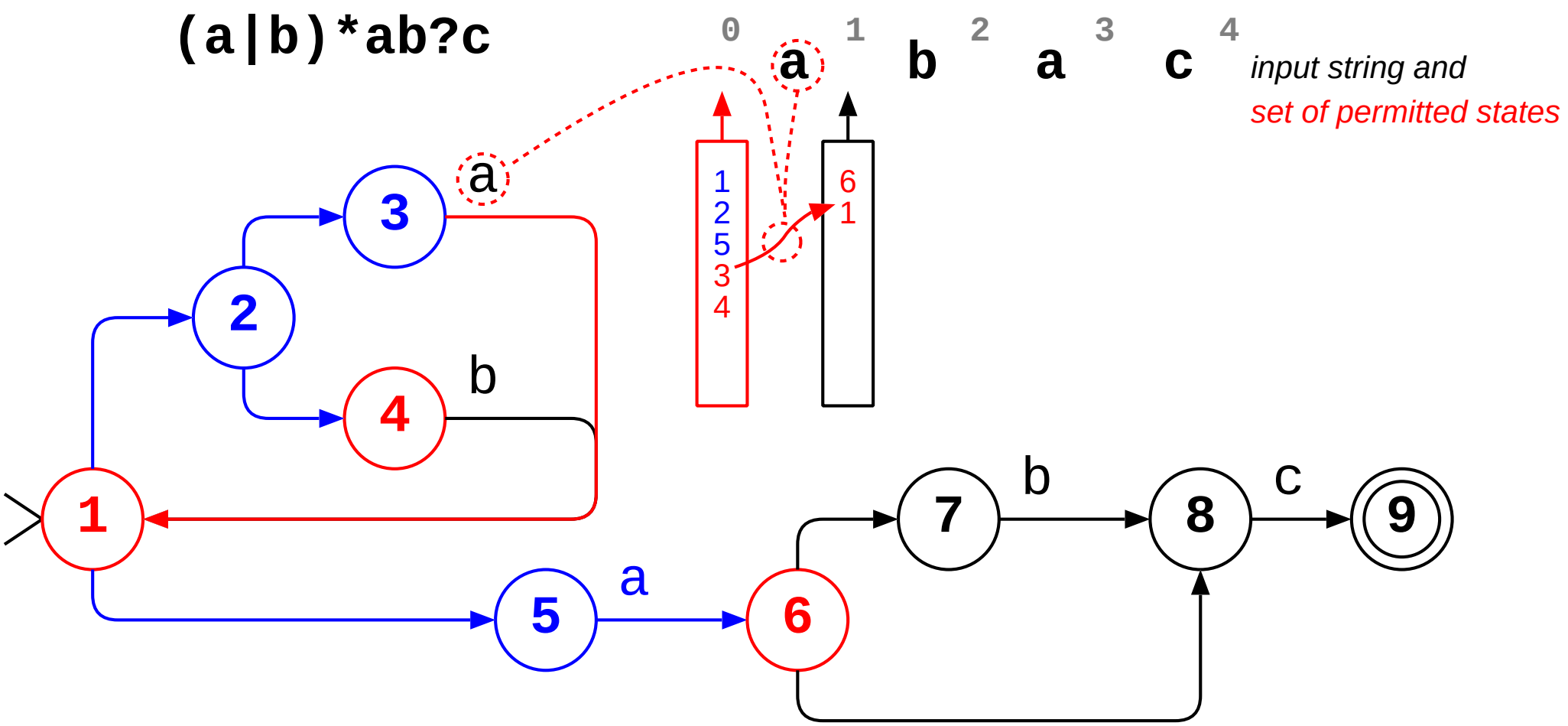
efficient recognition with NFAs



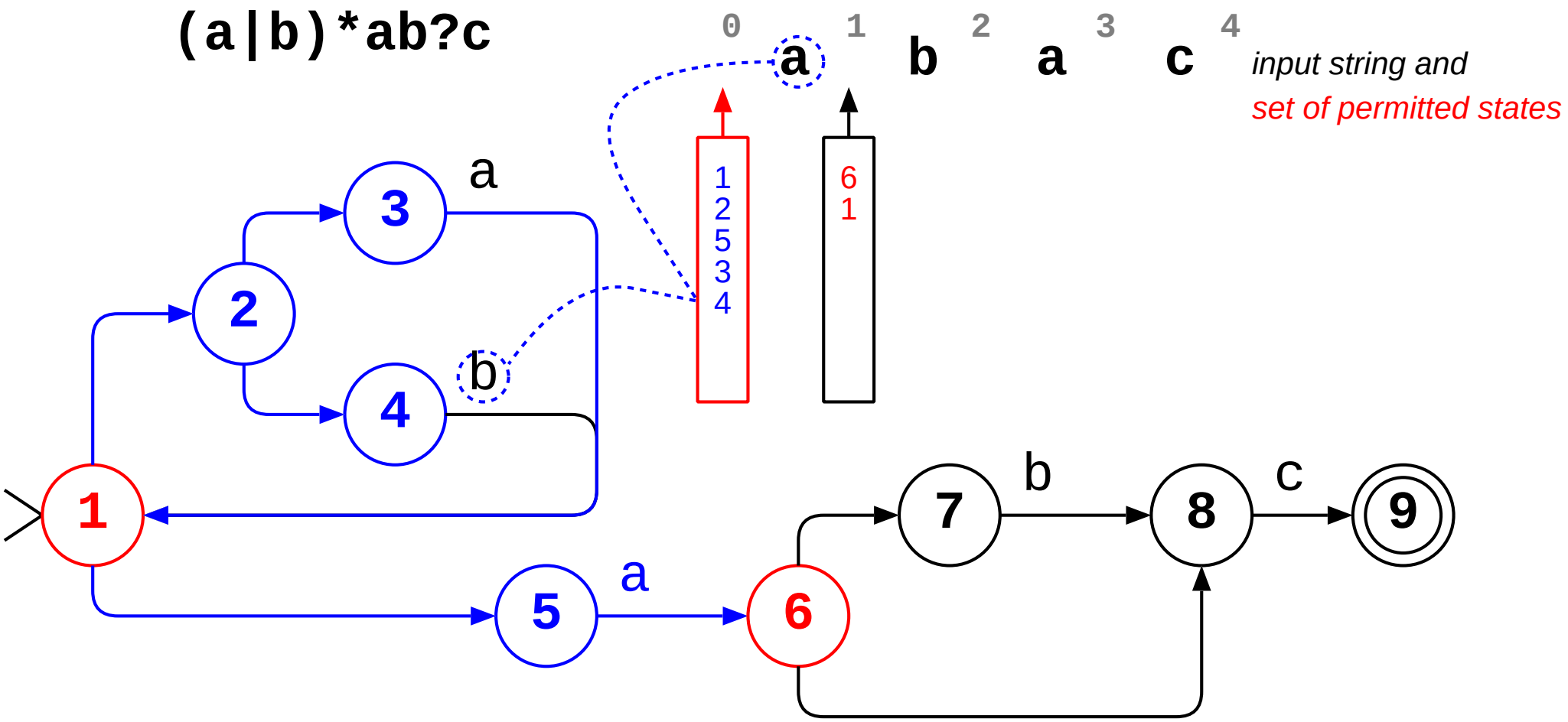
efficient recognition with NFAs



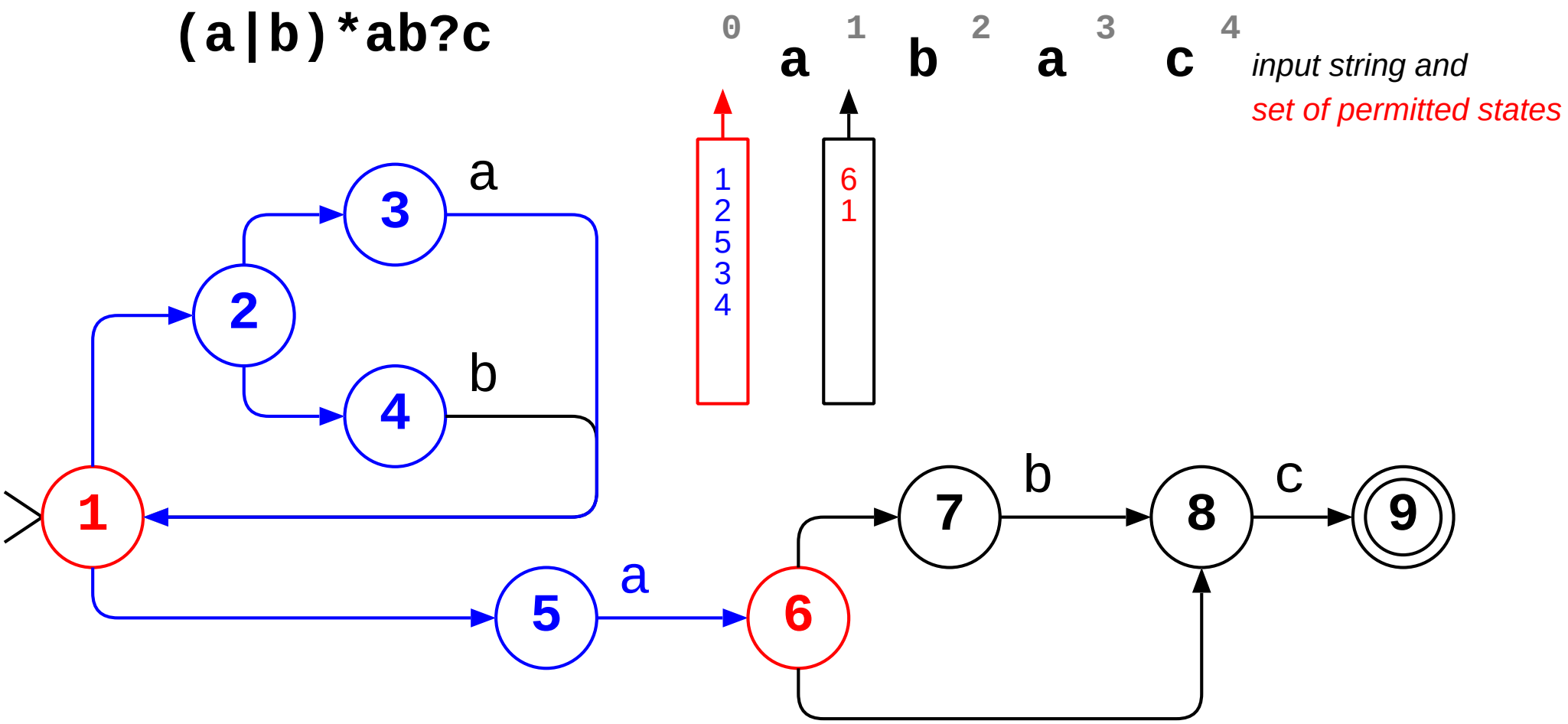
efficient recognition with NFAs



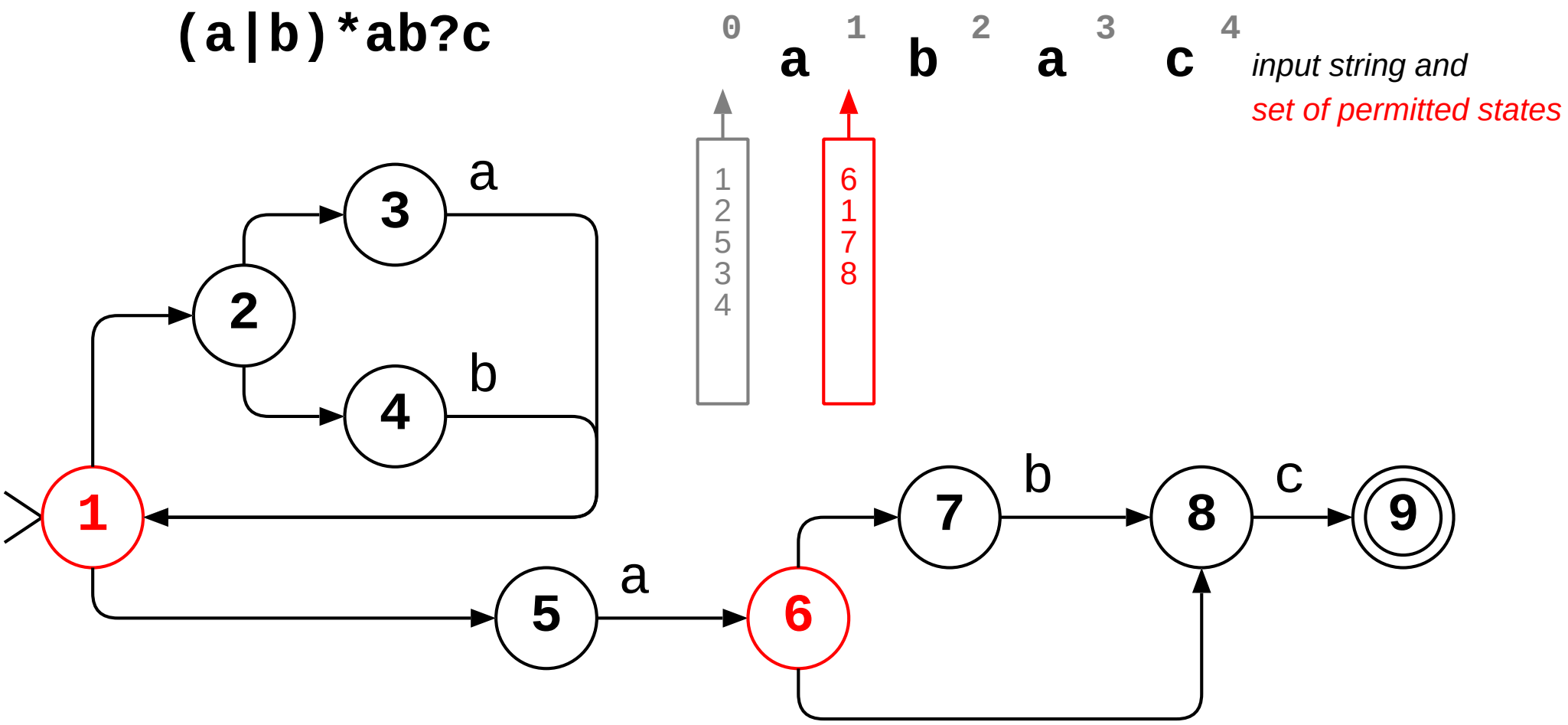
efficient recognition with NFAs



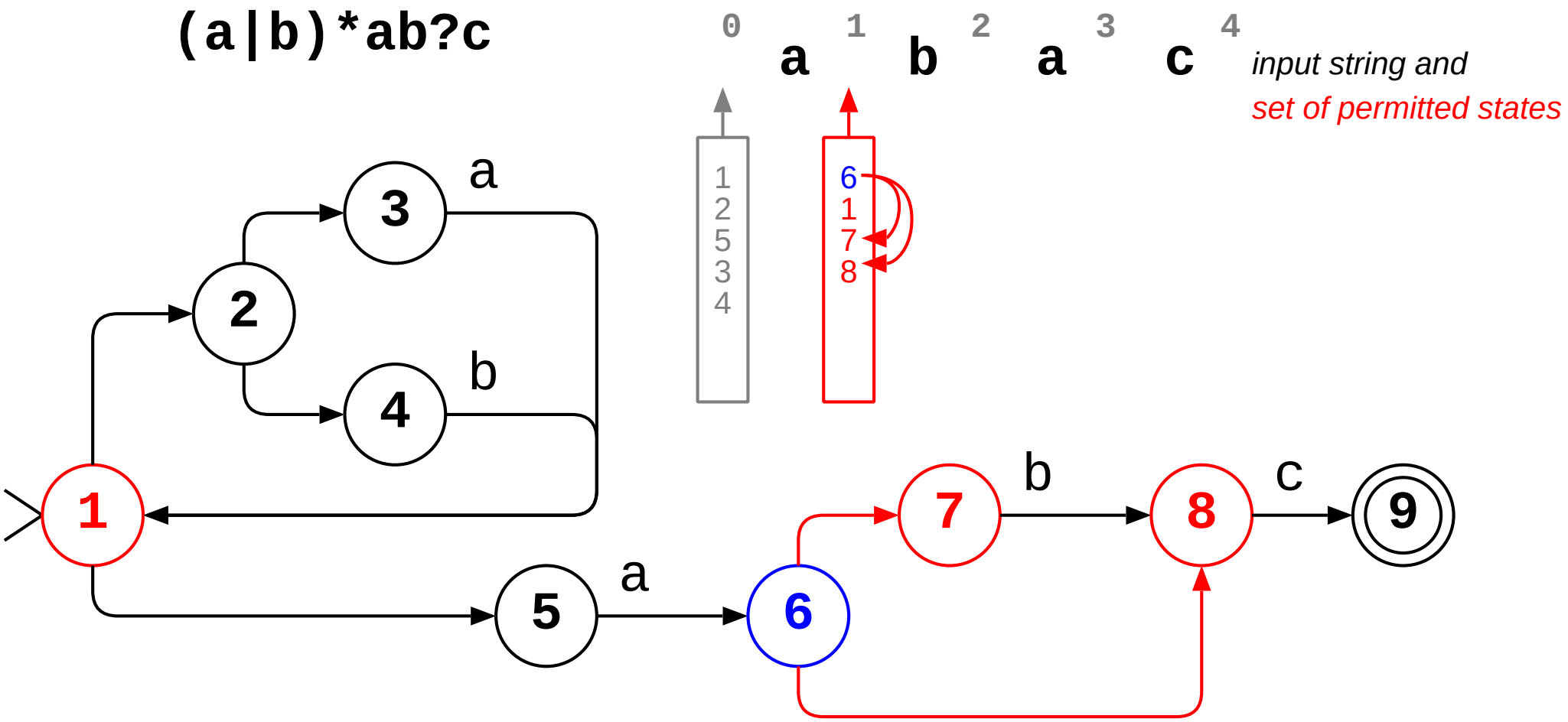
efficient recognition with NFAs



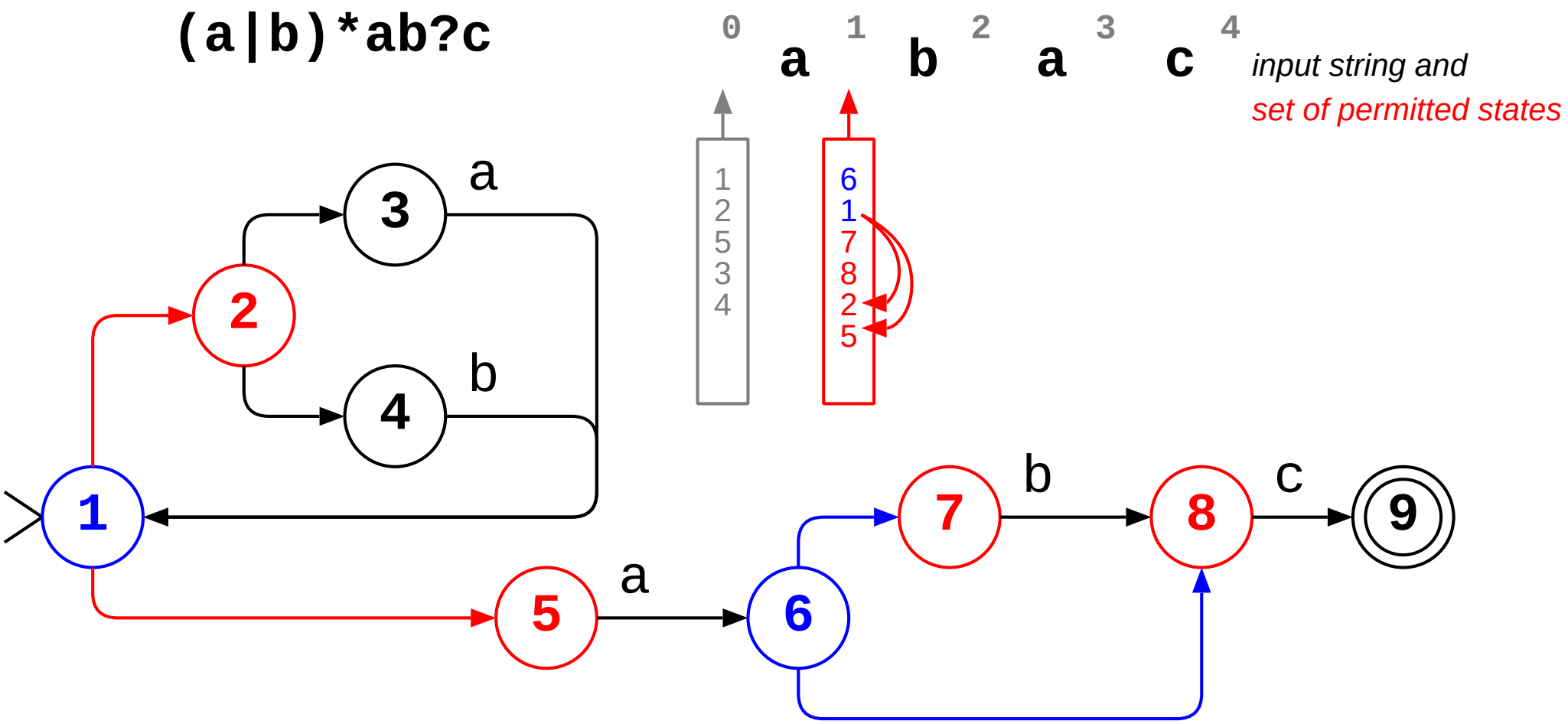
efficient recognition with NFAs



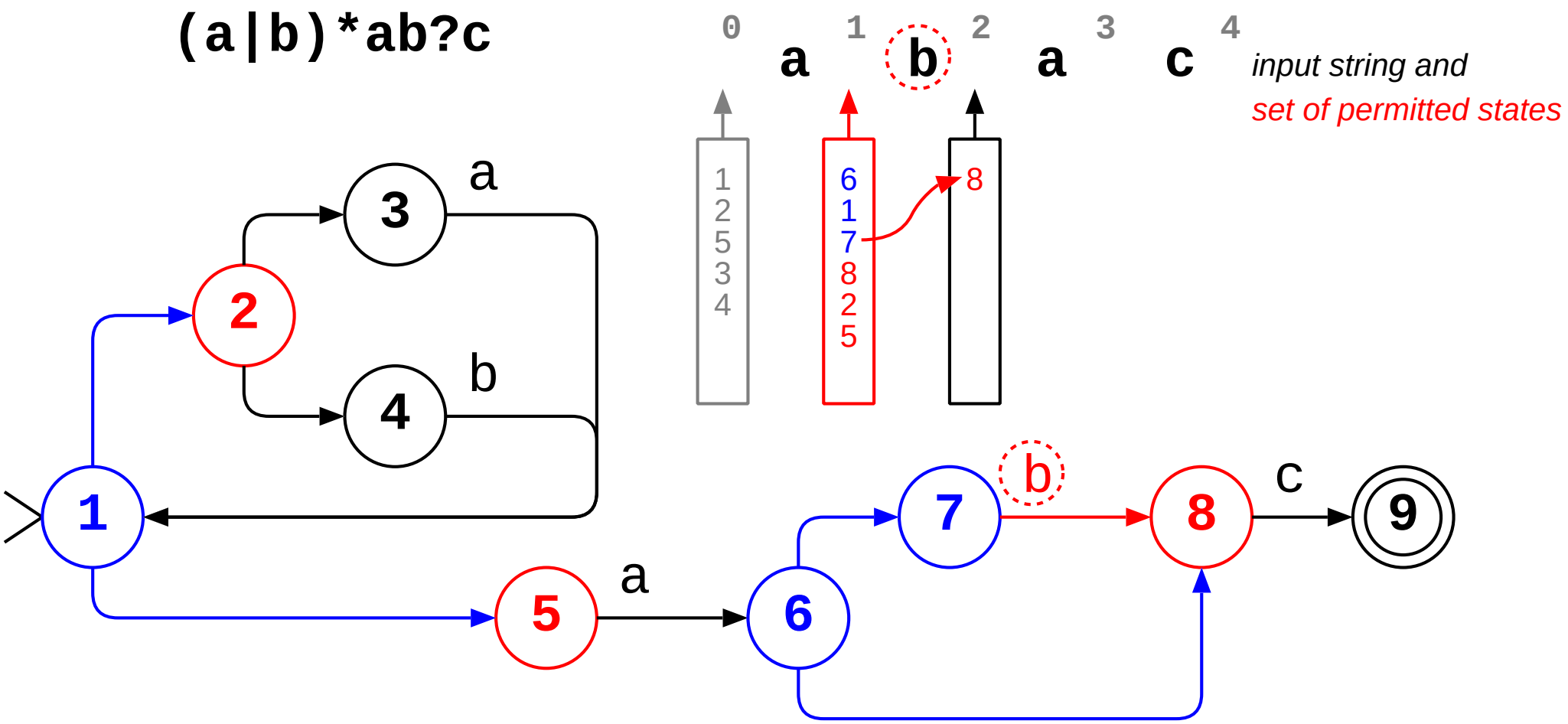
efficient recognition with NFAs



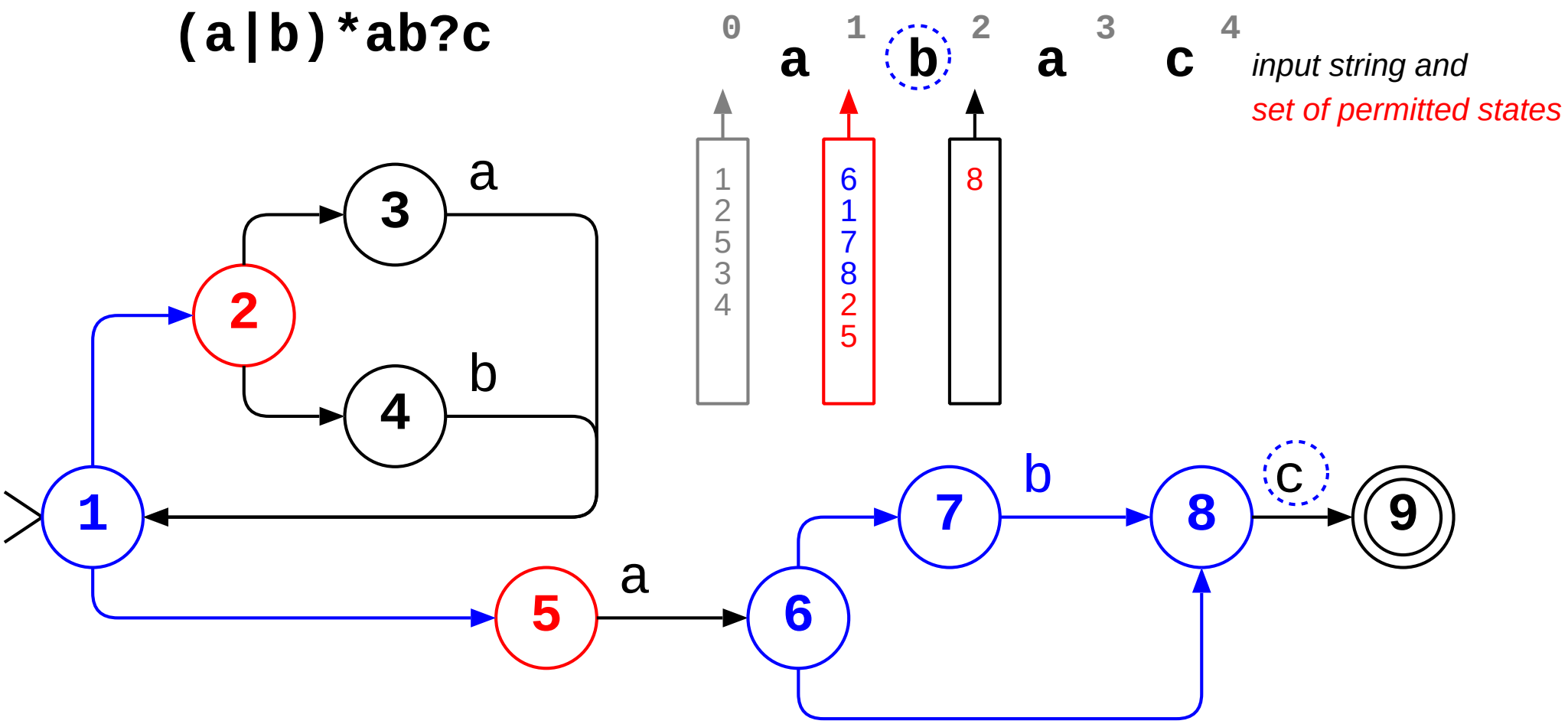
efficient recognition with NFAs



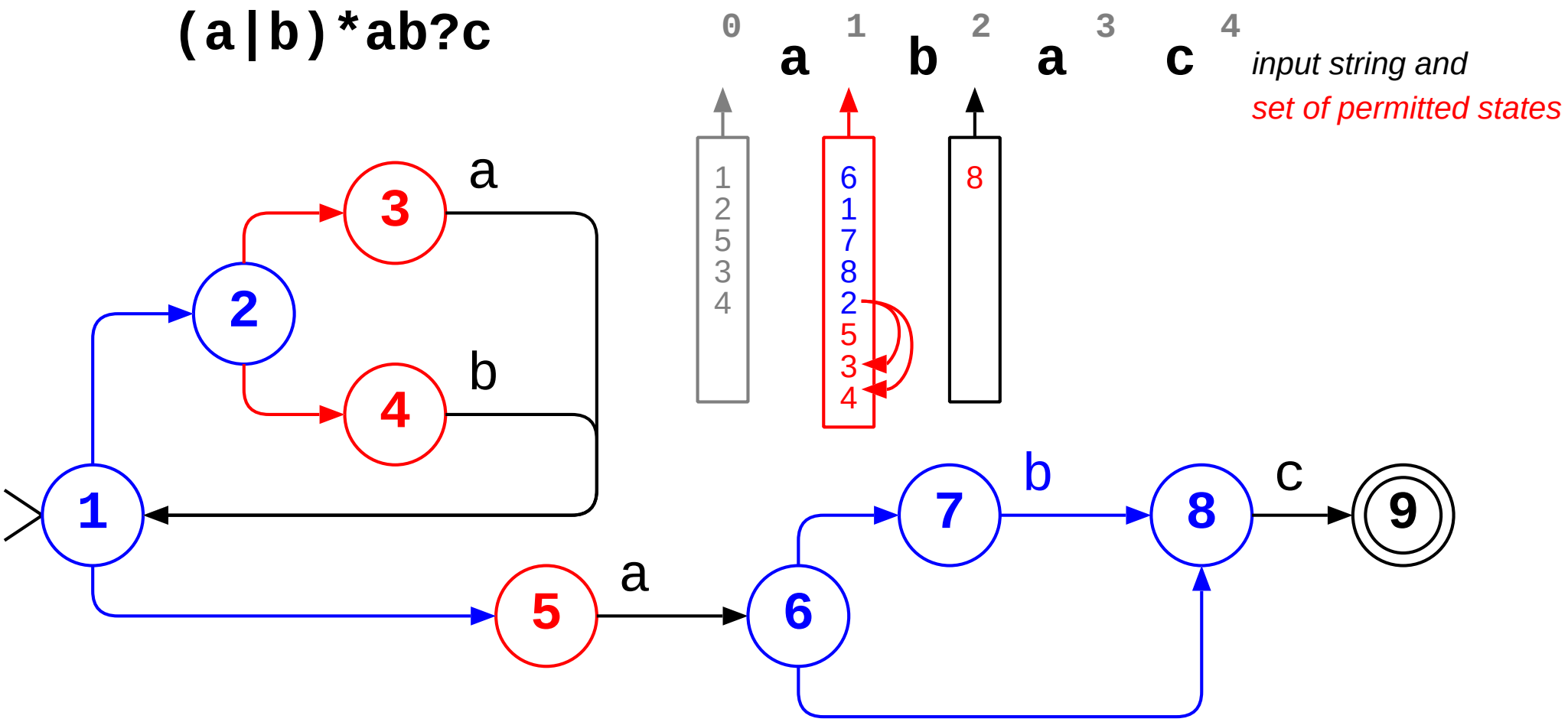
efficient recognition with NFAs



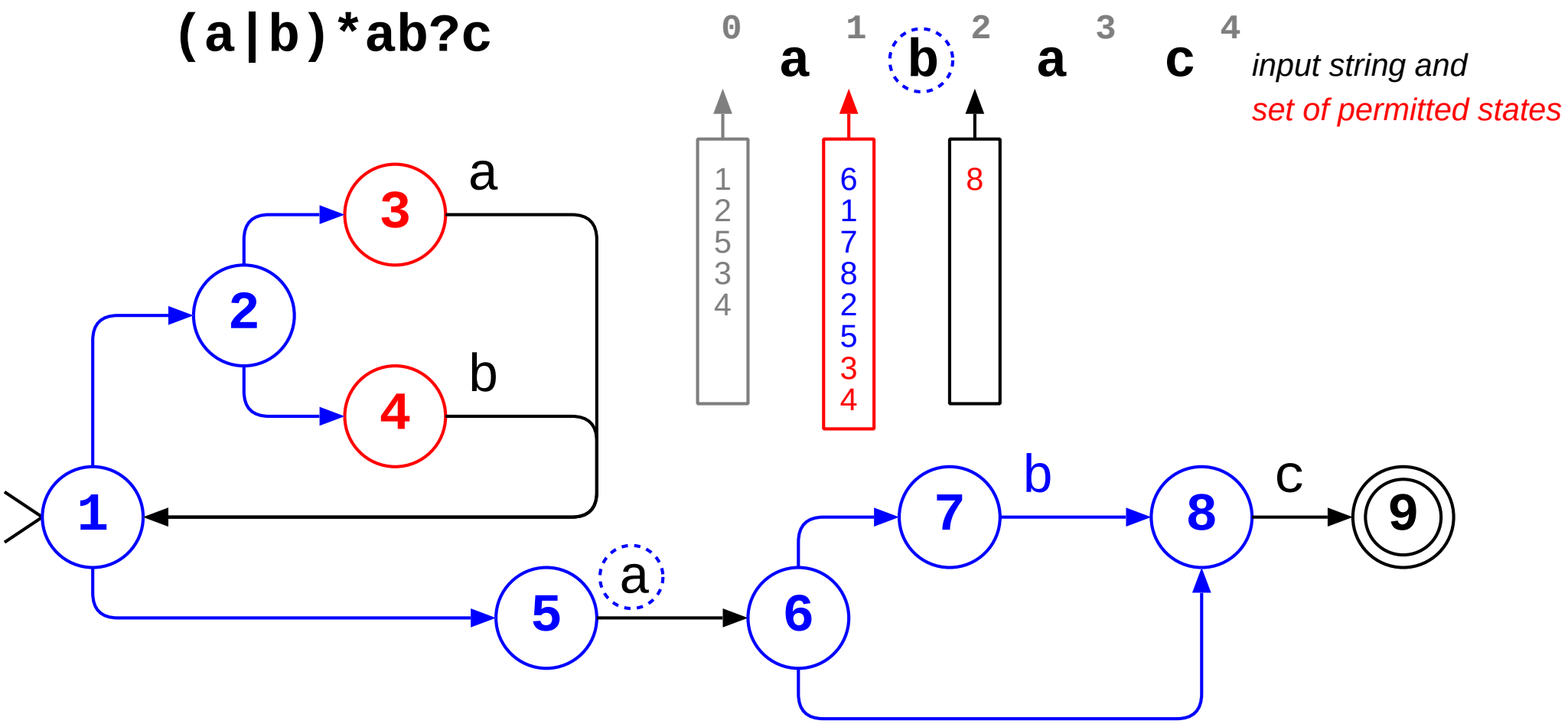
efficient recognition with NFAs



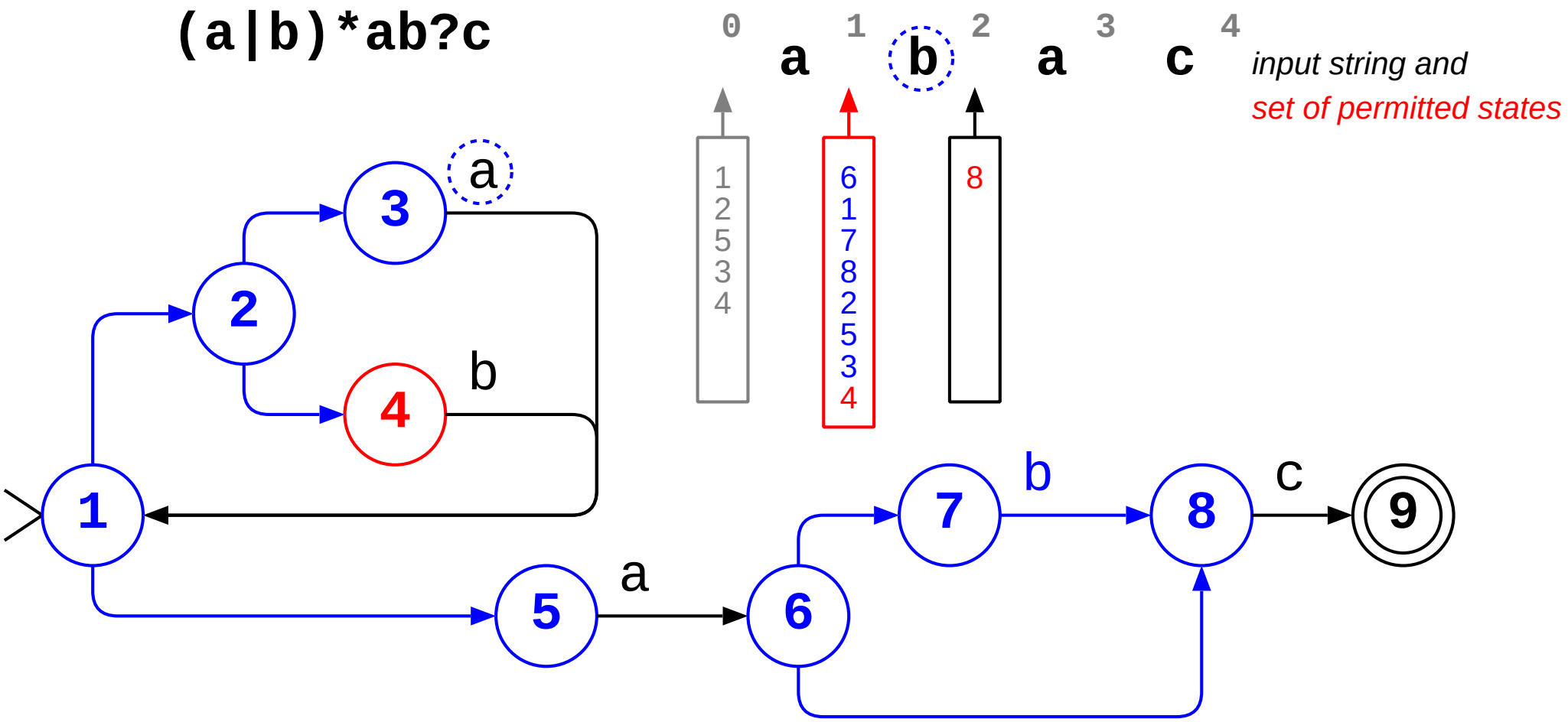
efficient recognition with NFAs



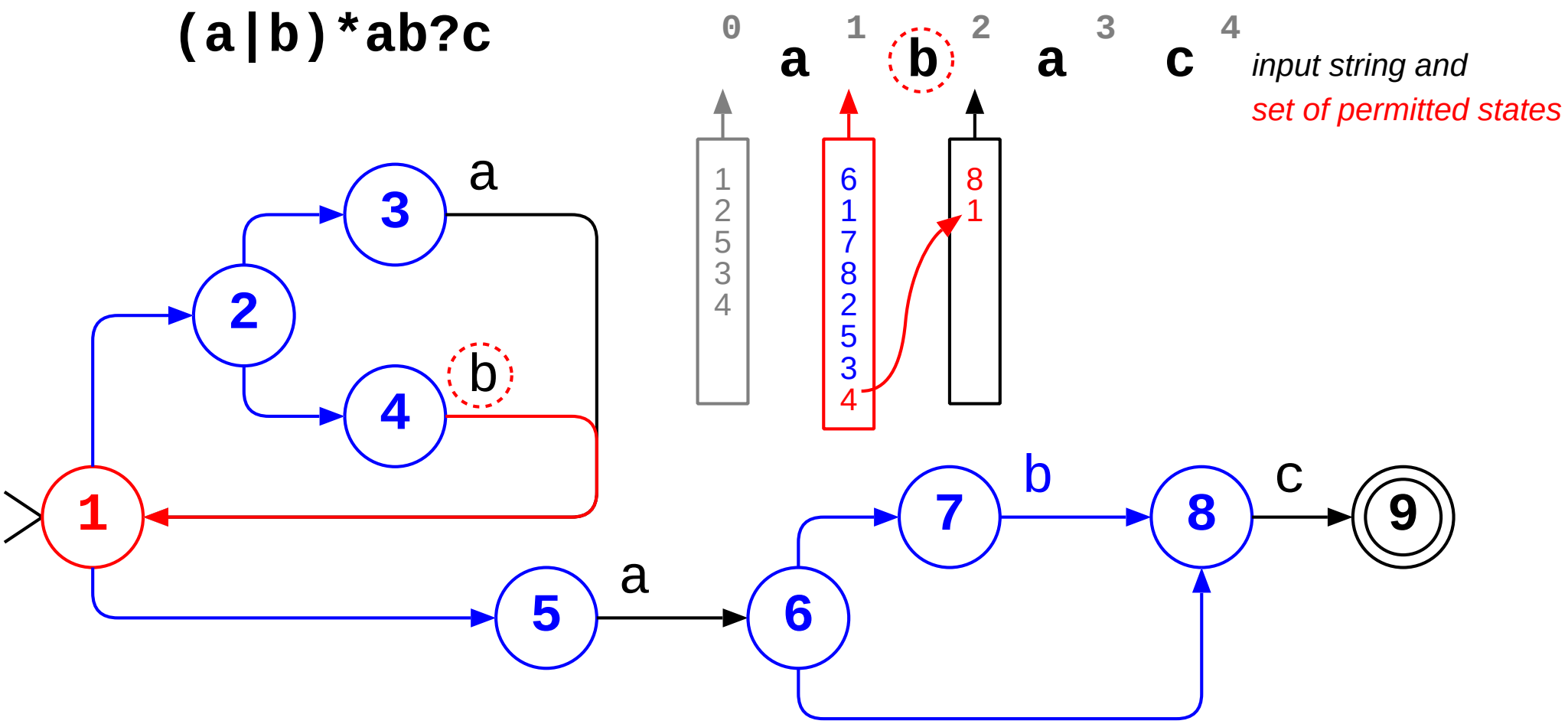
efficient recognition with NFAs



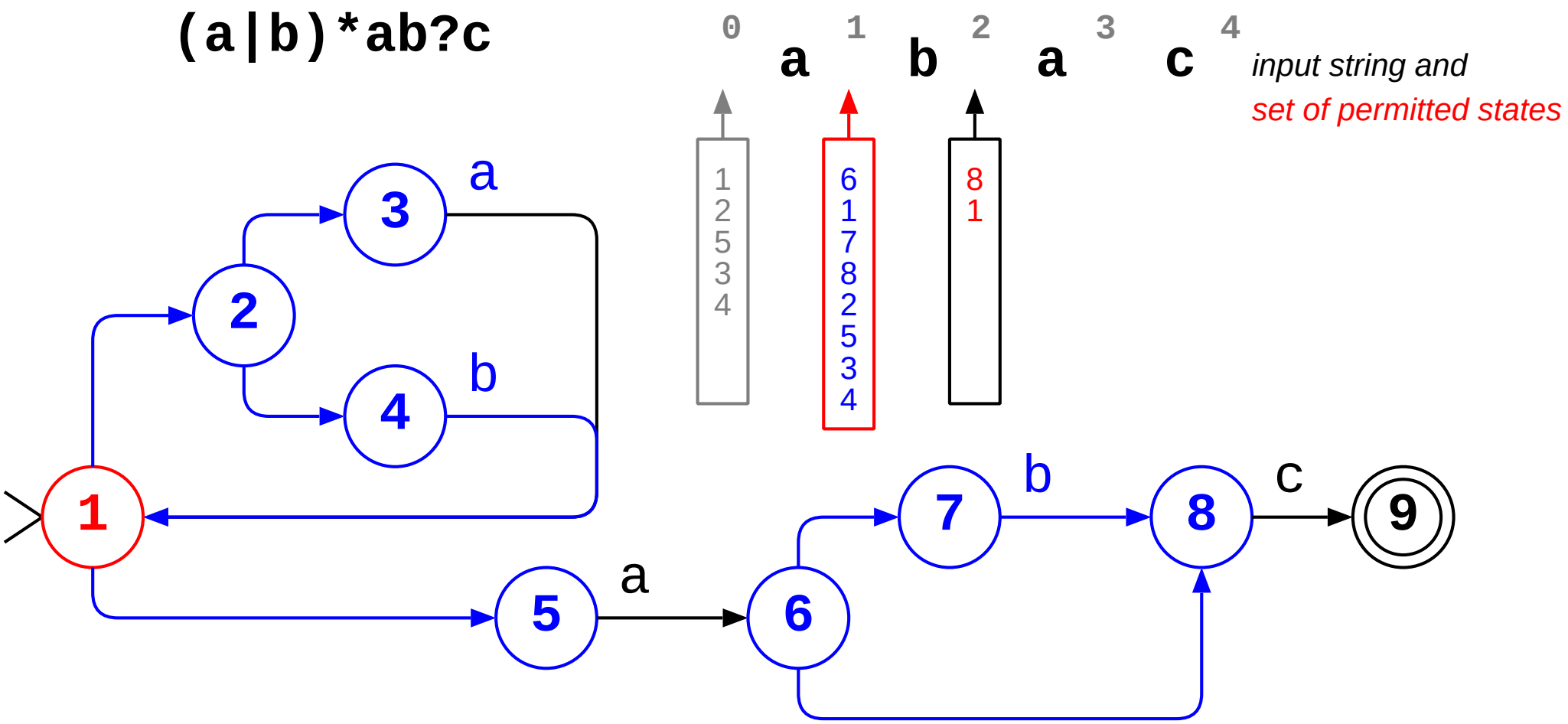
efficient recognition with NFAs



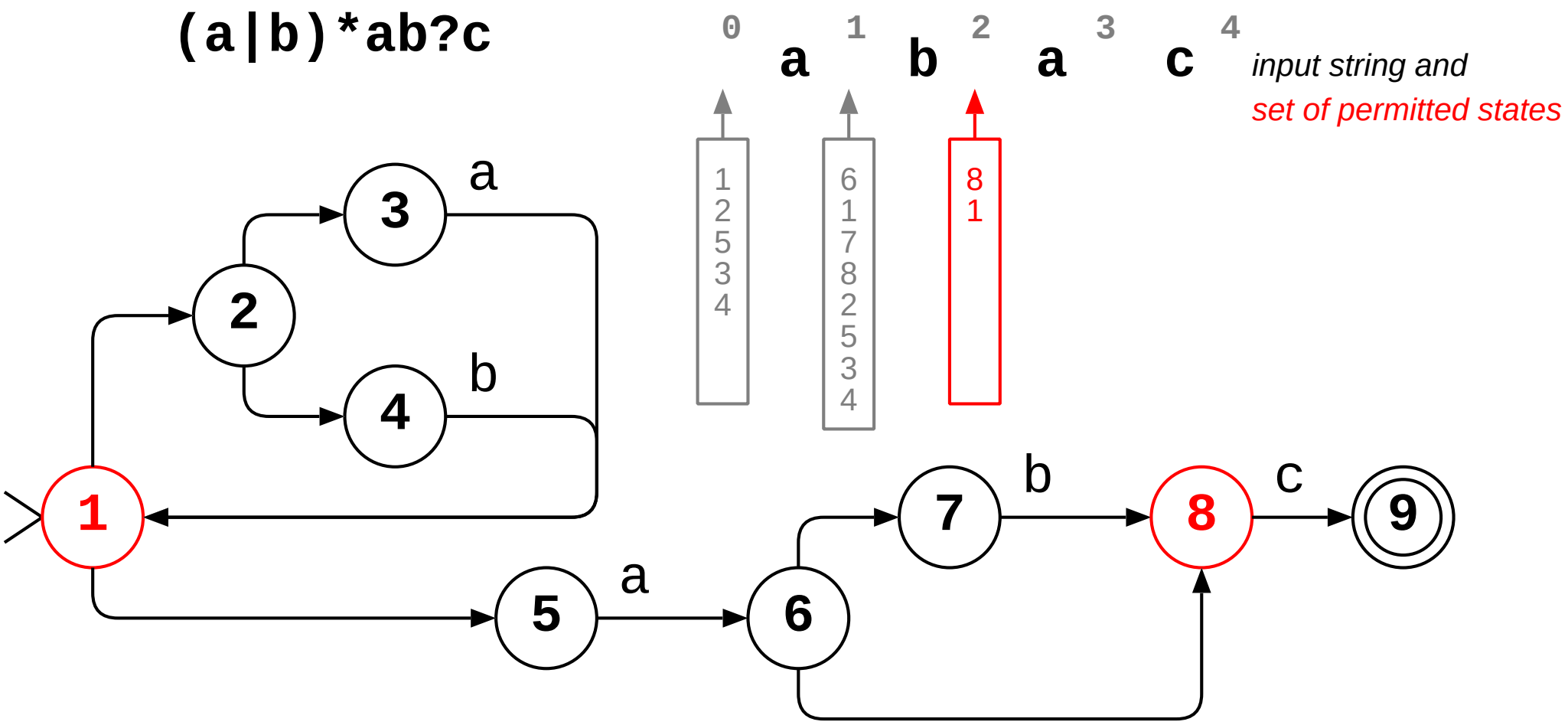
efficient recognition with NFAs



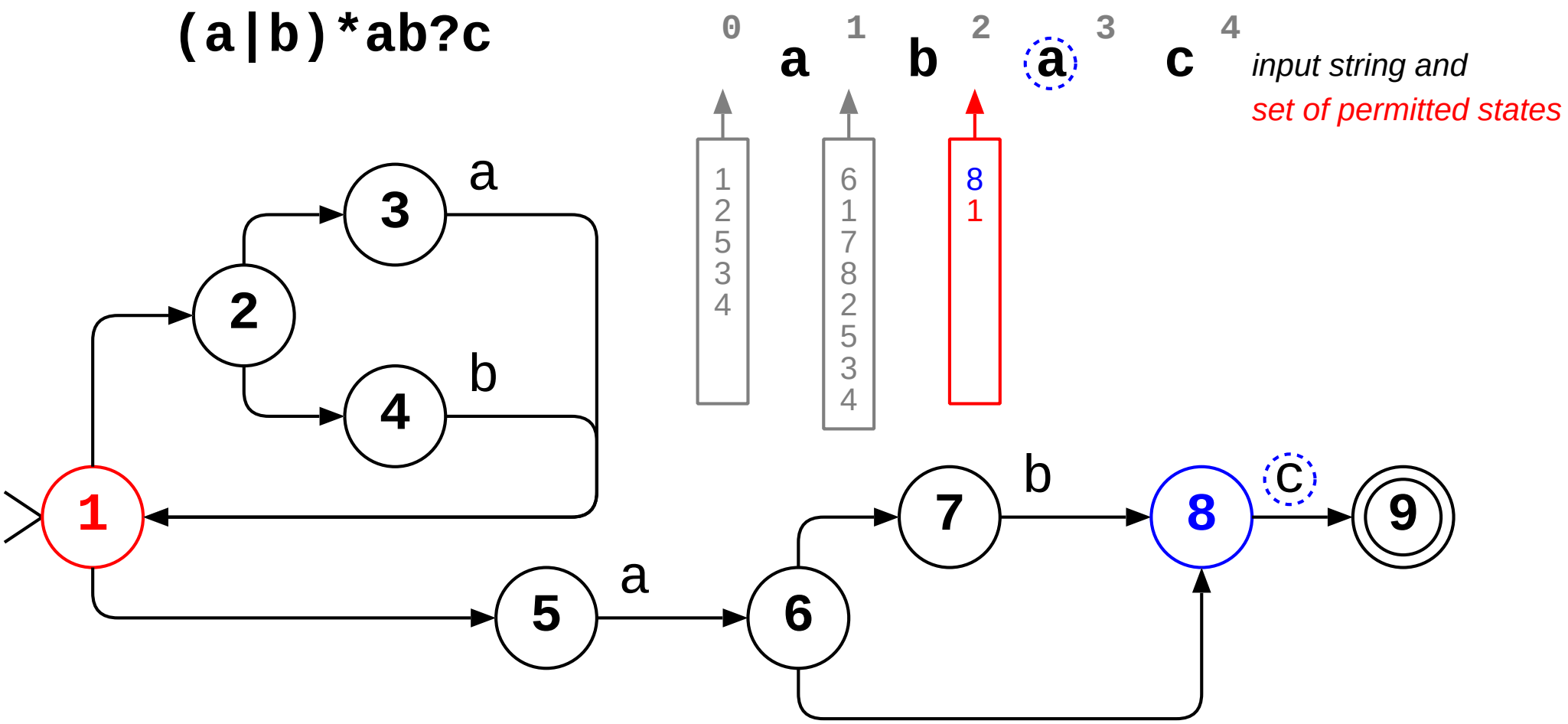
efficient recognition with NFAs



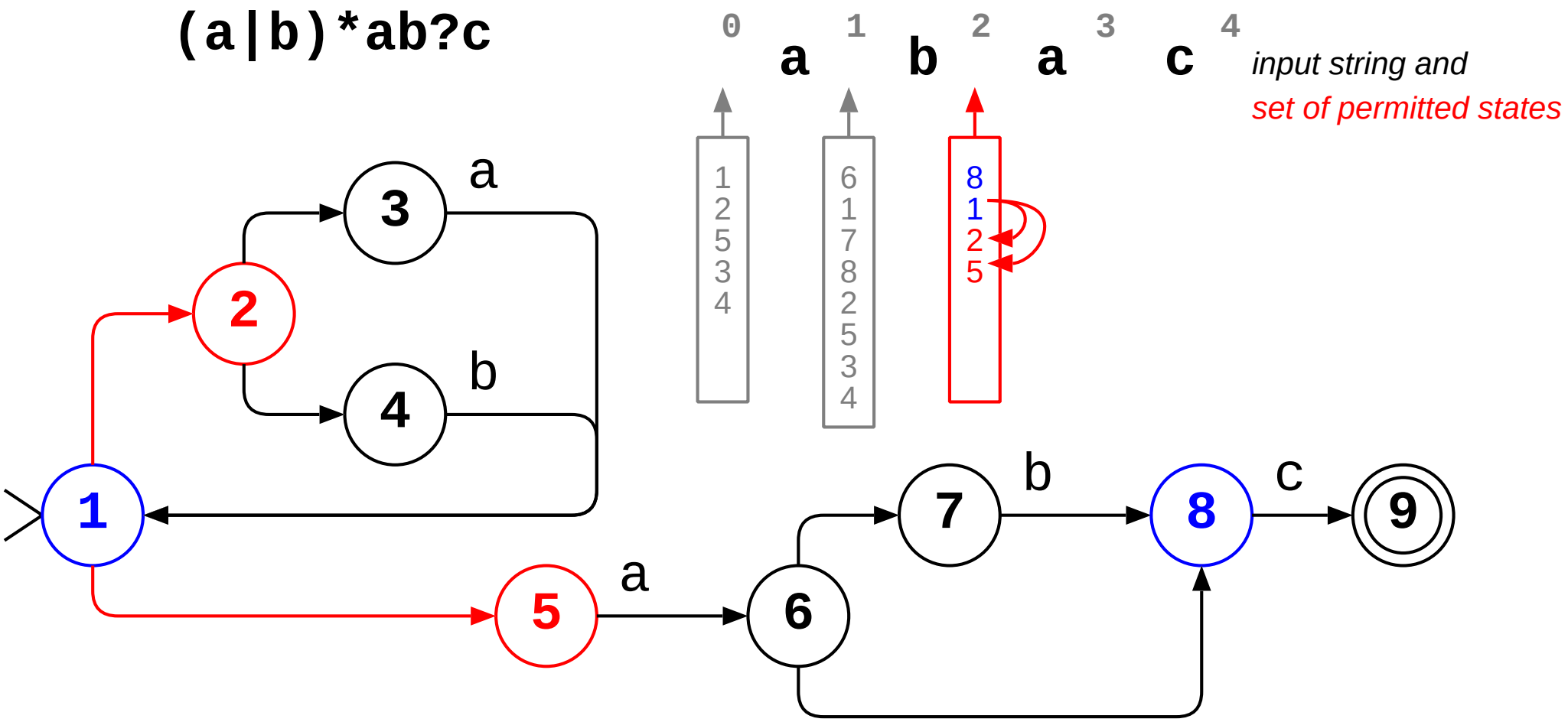
efficient recognition with NFAs



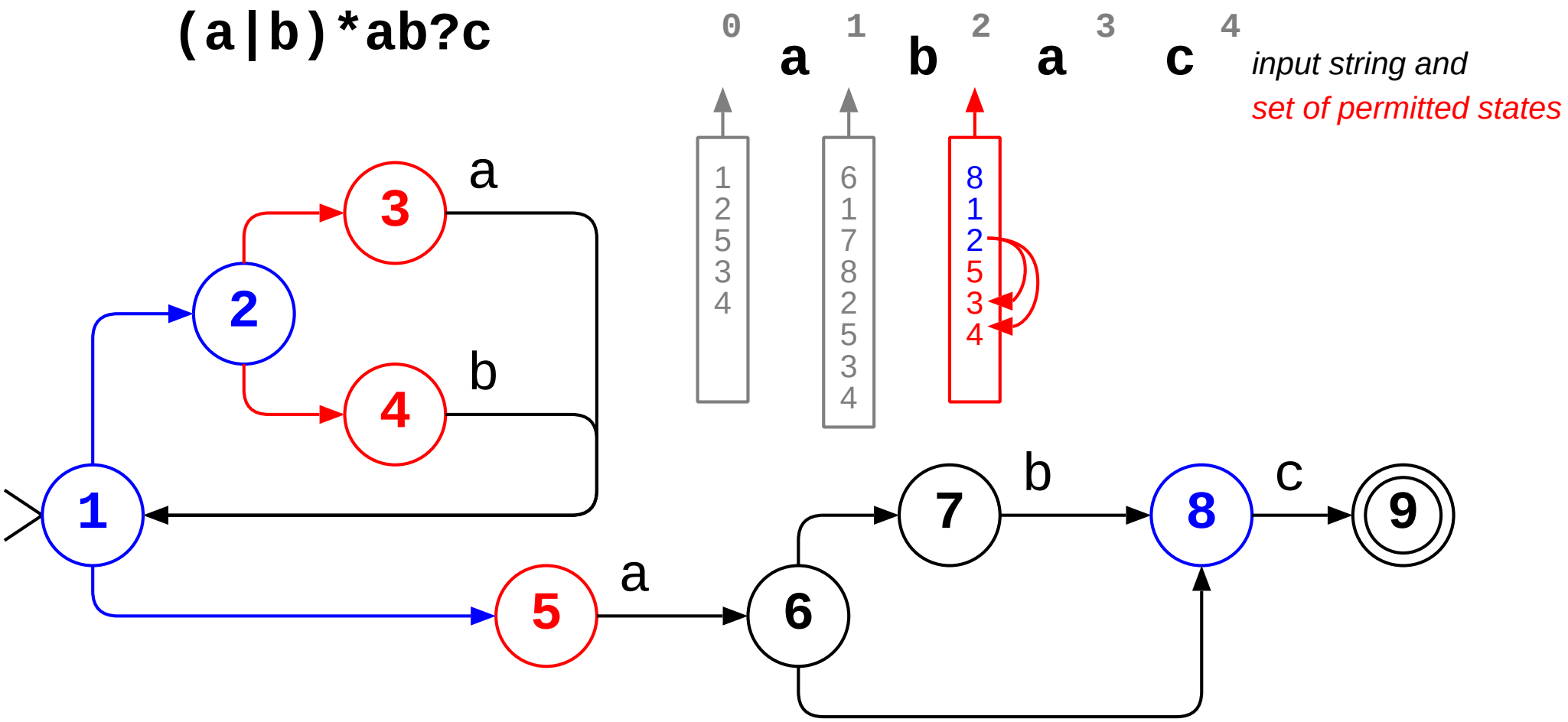
efficient recognition with NFAs



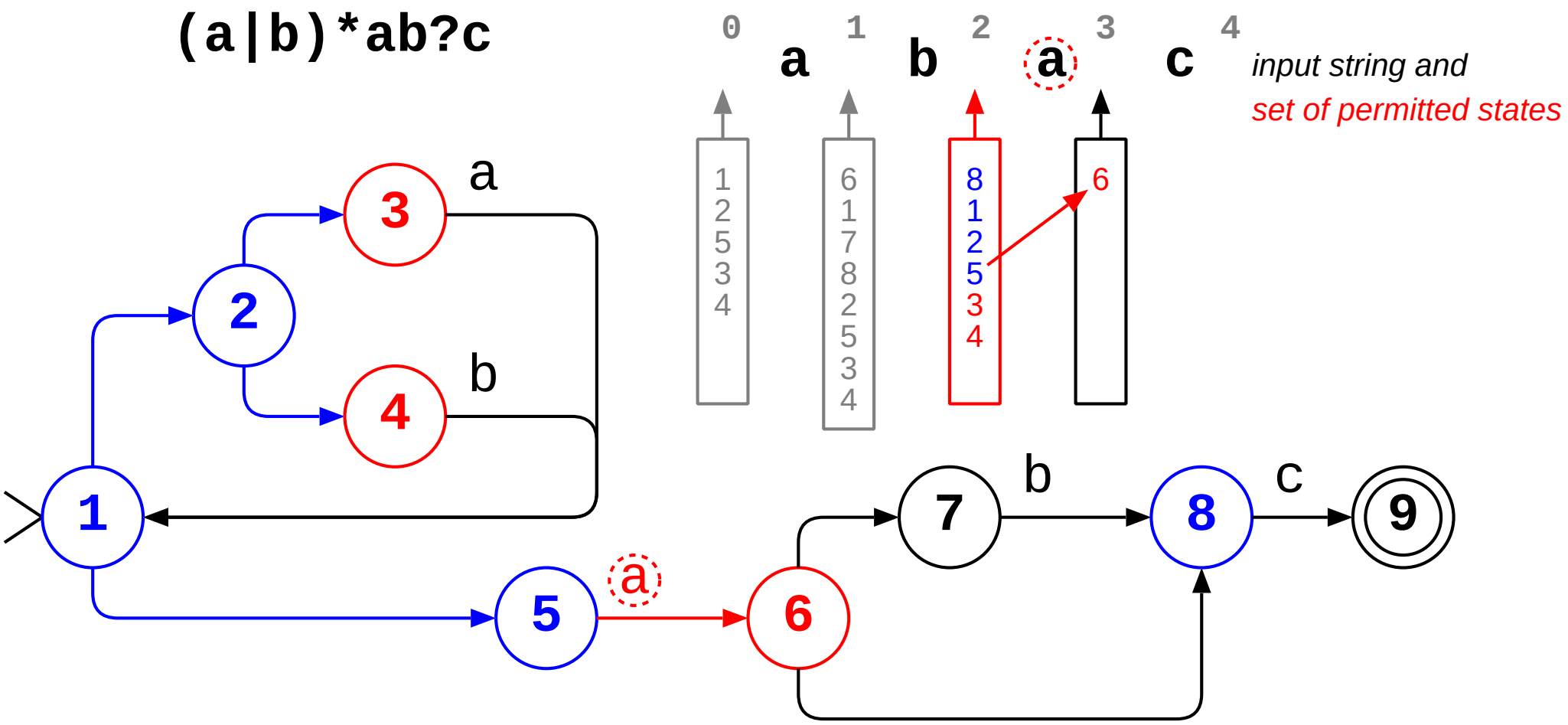
efficient recognition with NFAs



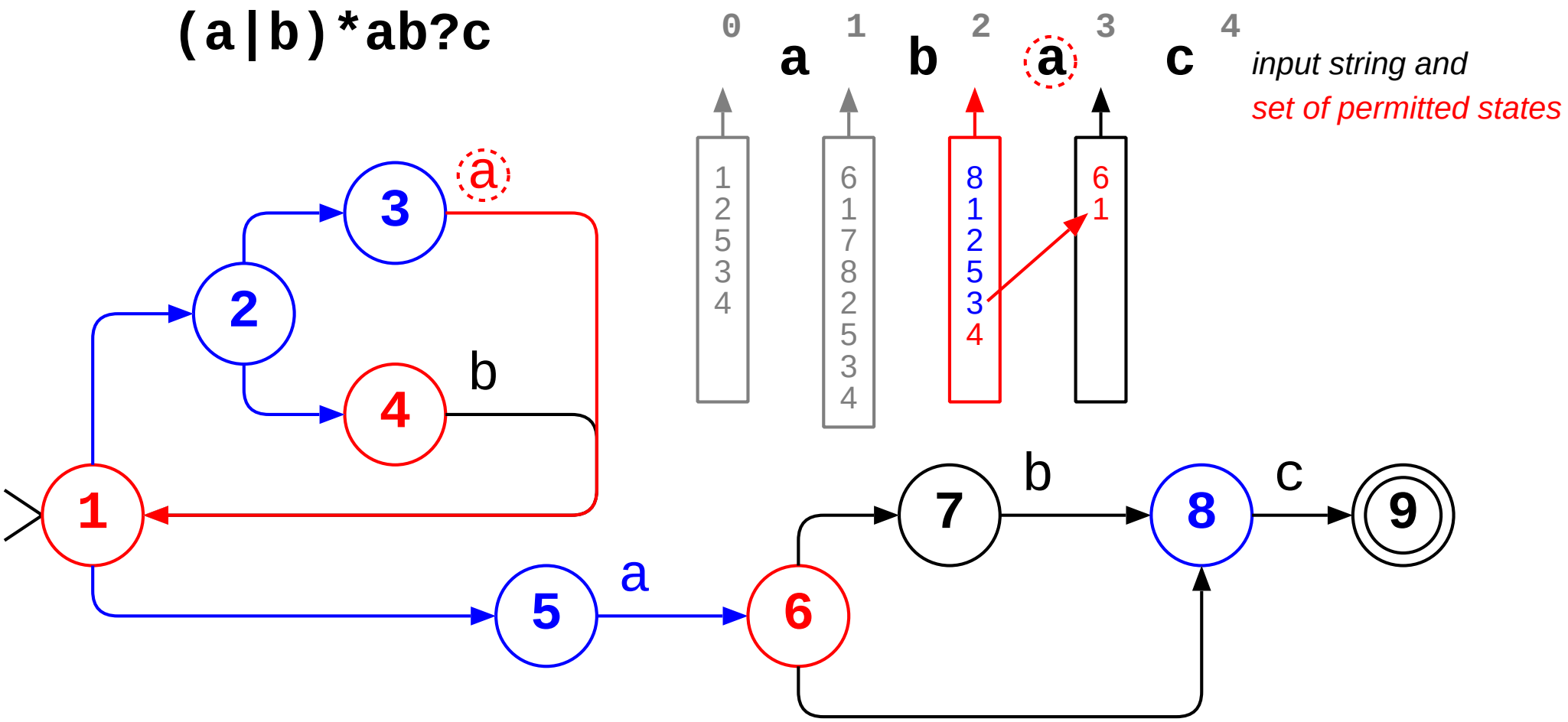
efficient recognition with NFAs



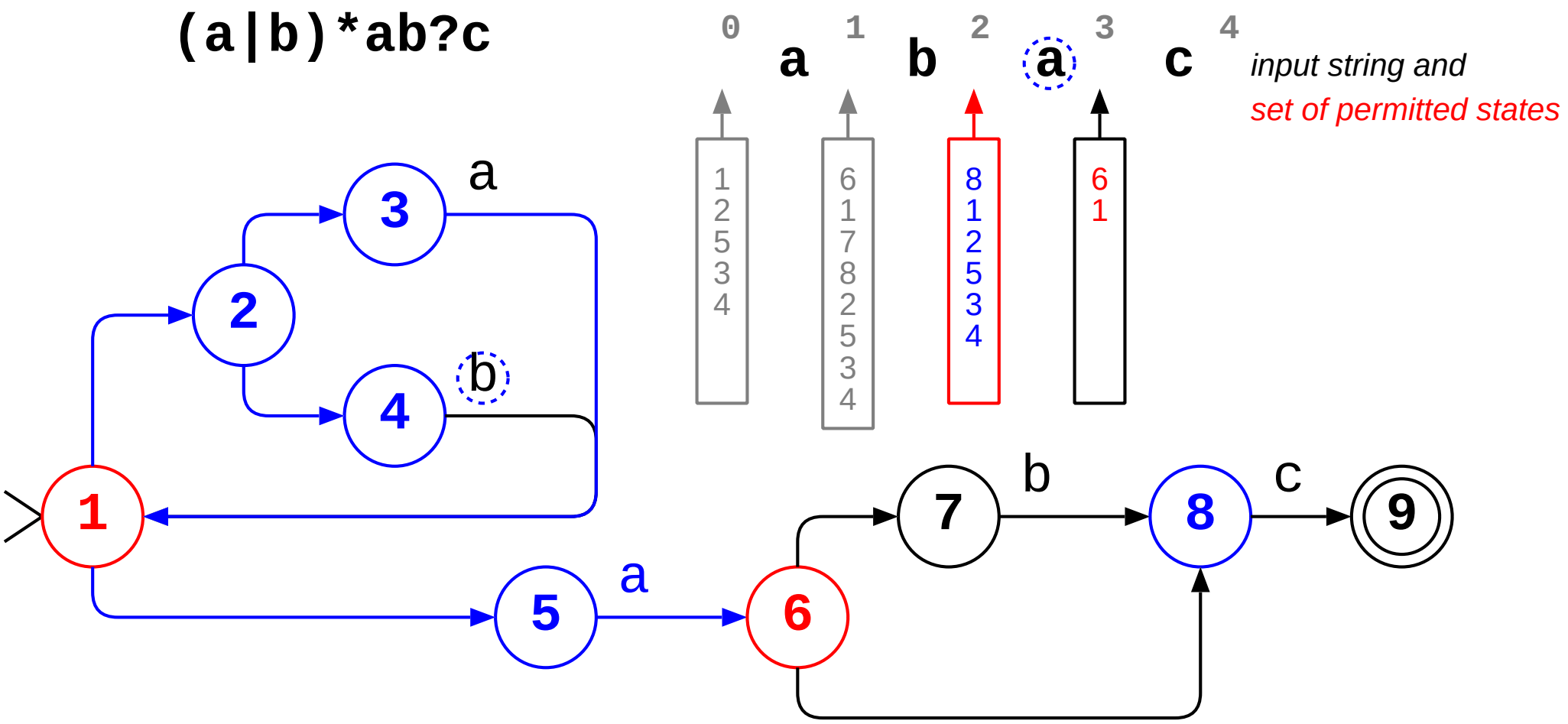
efficient recognition with NFAs



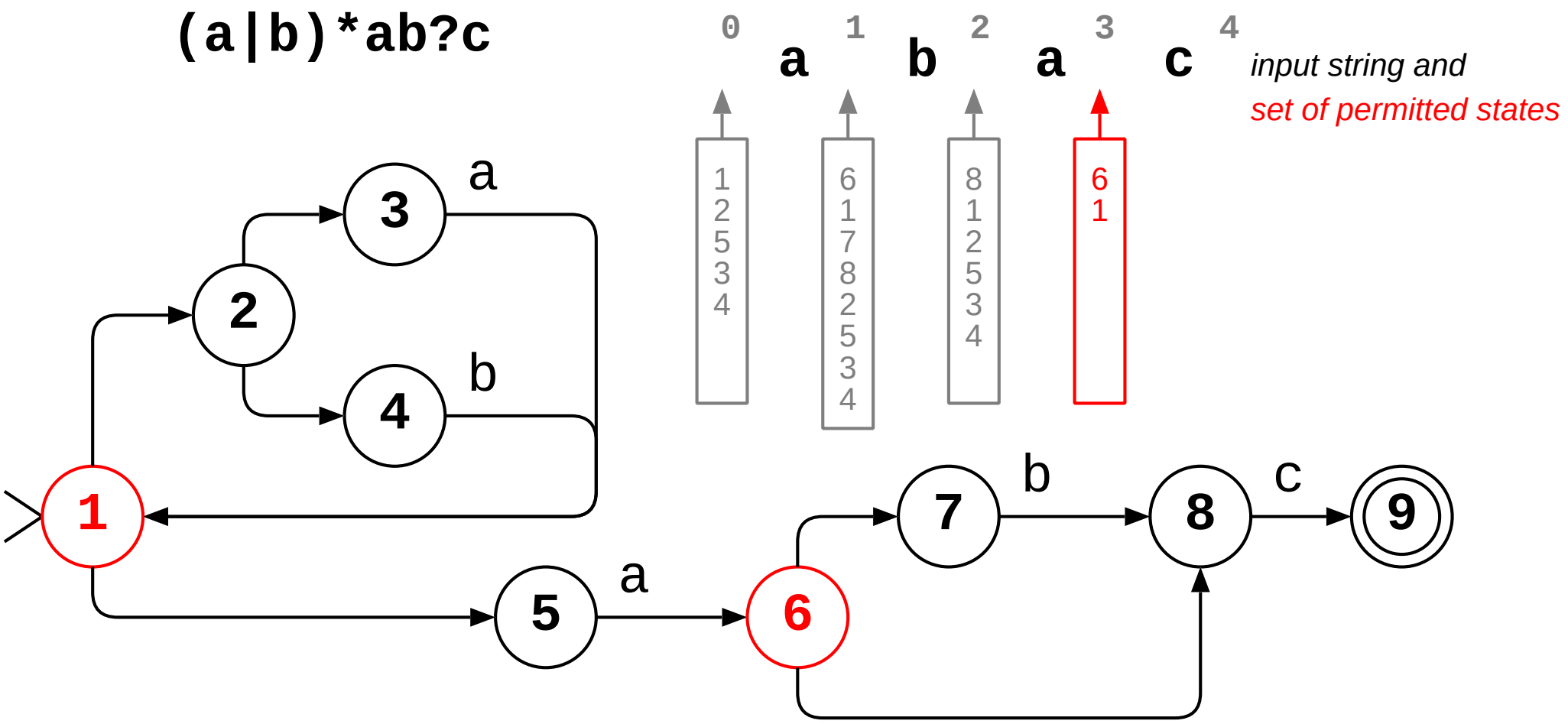
efficient recognition with NFAs



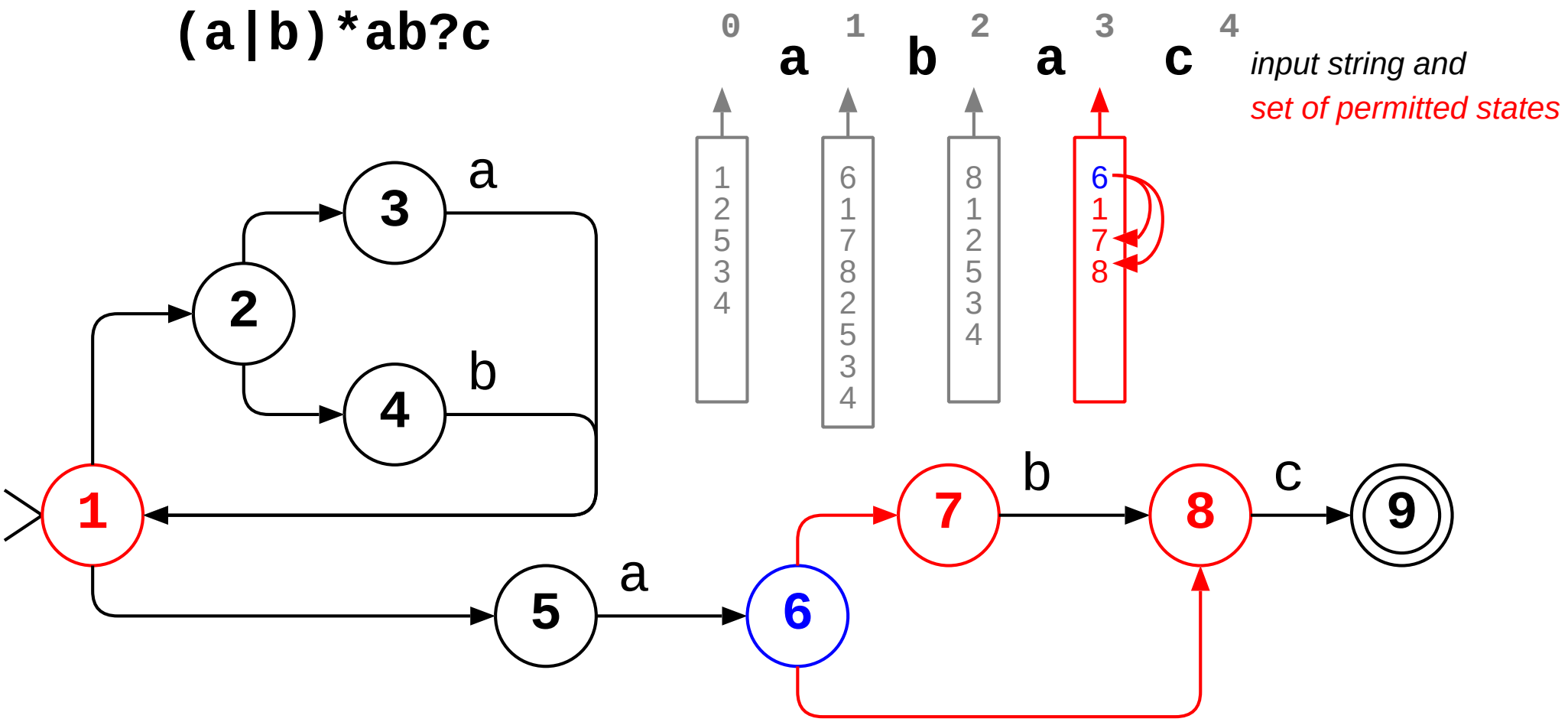
efficient recognition with NFAs



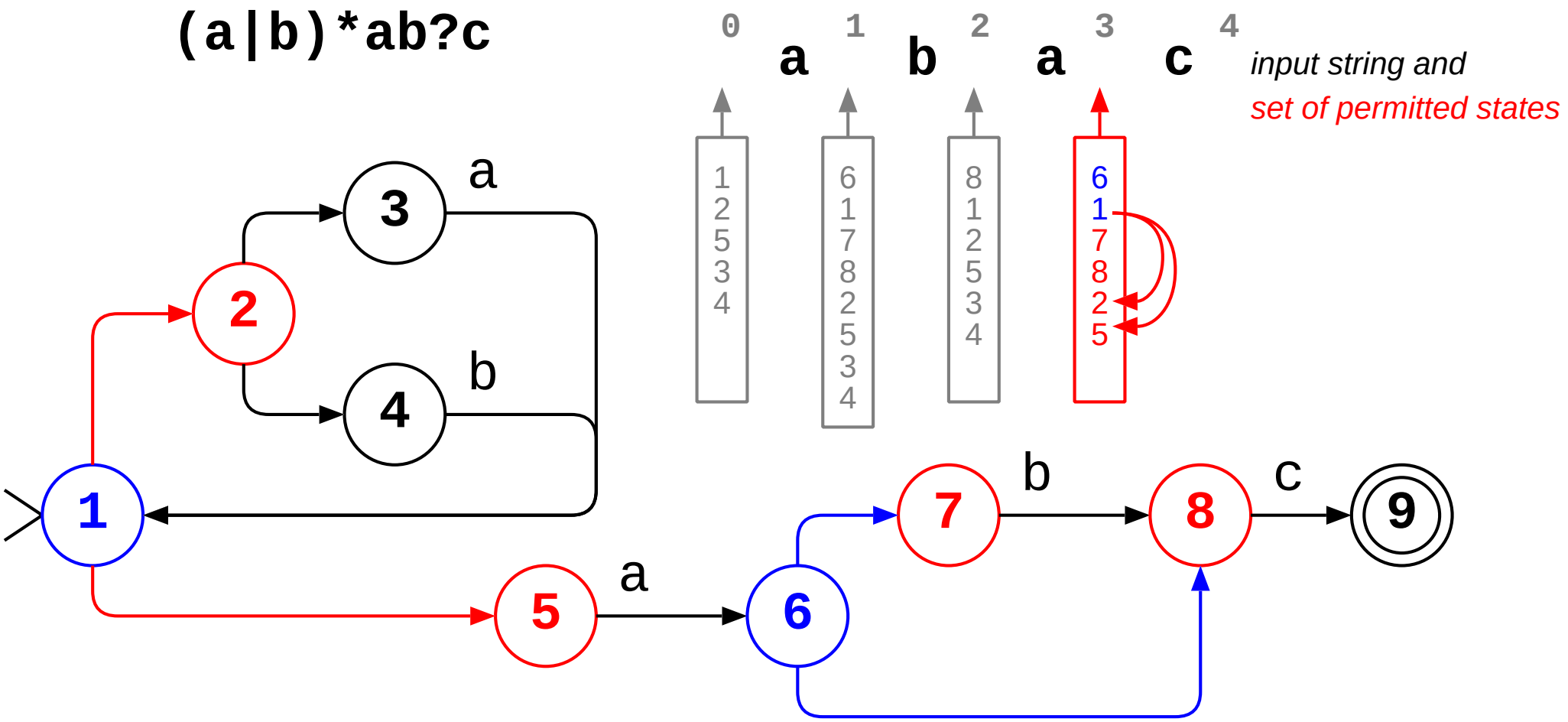
efficient recognition with NFAs



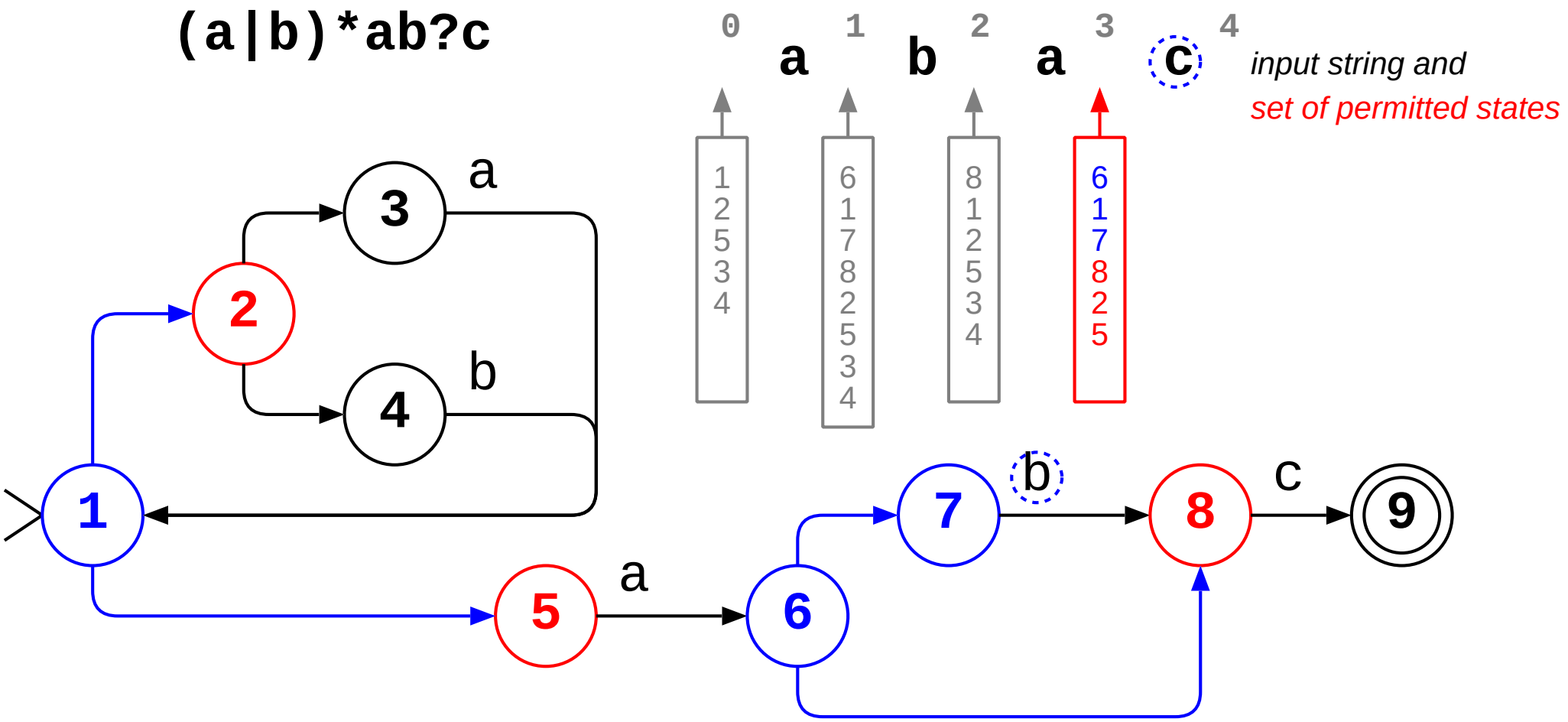
efficient recognition with NFAs



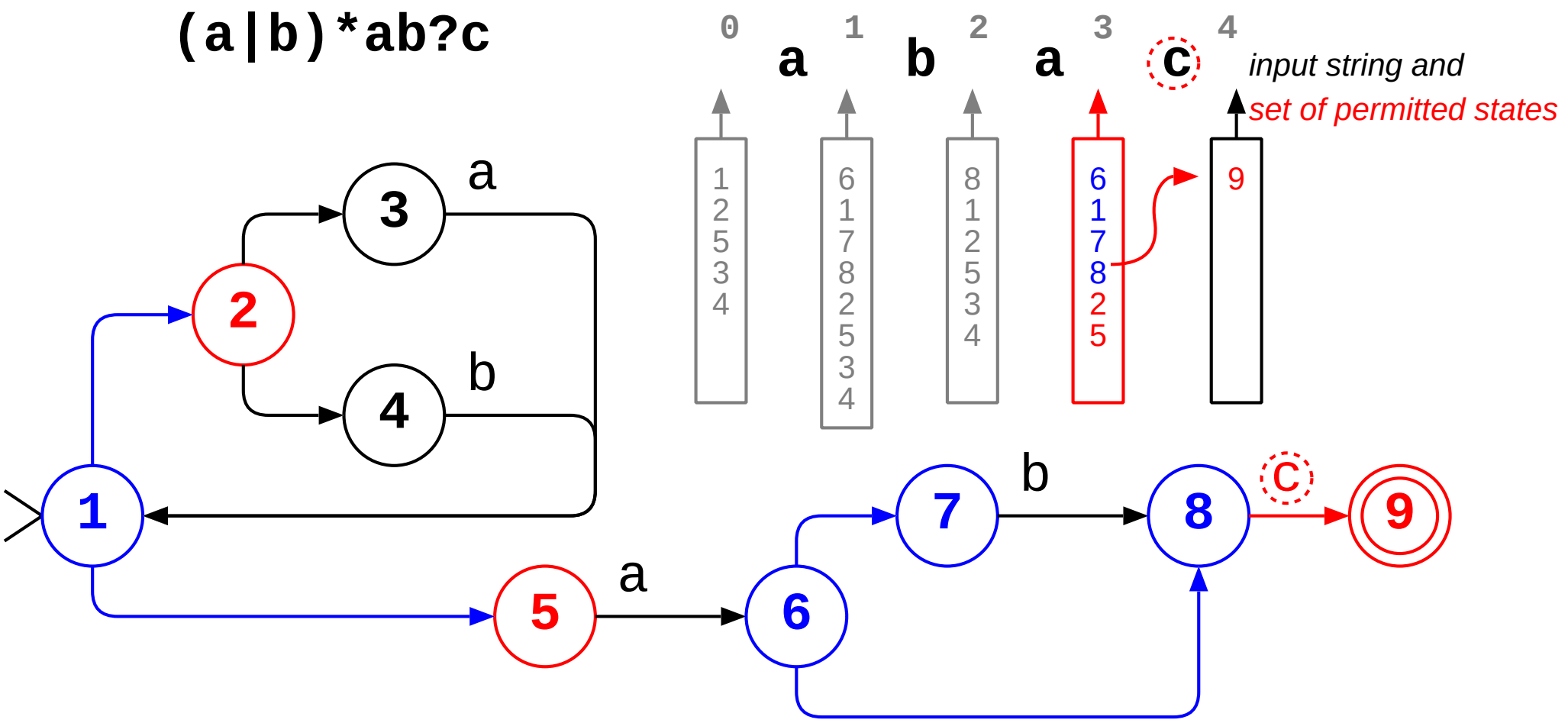
efficient recognition with NFAs



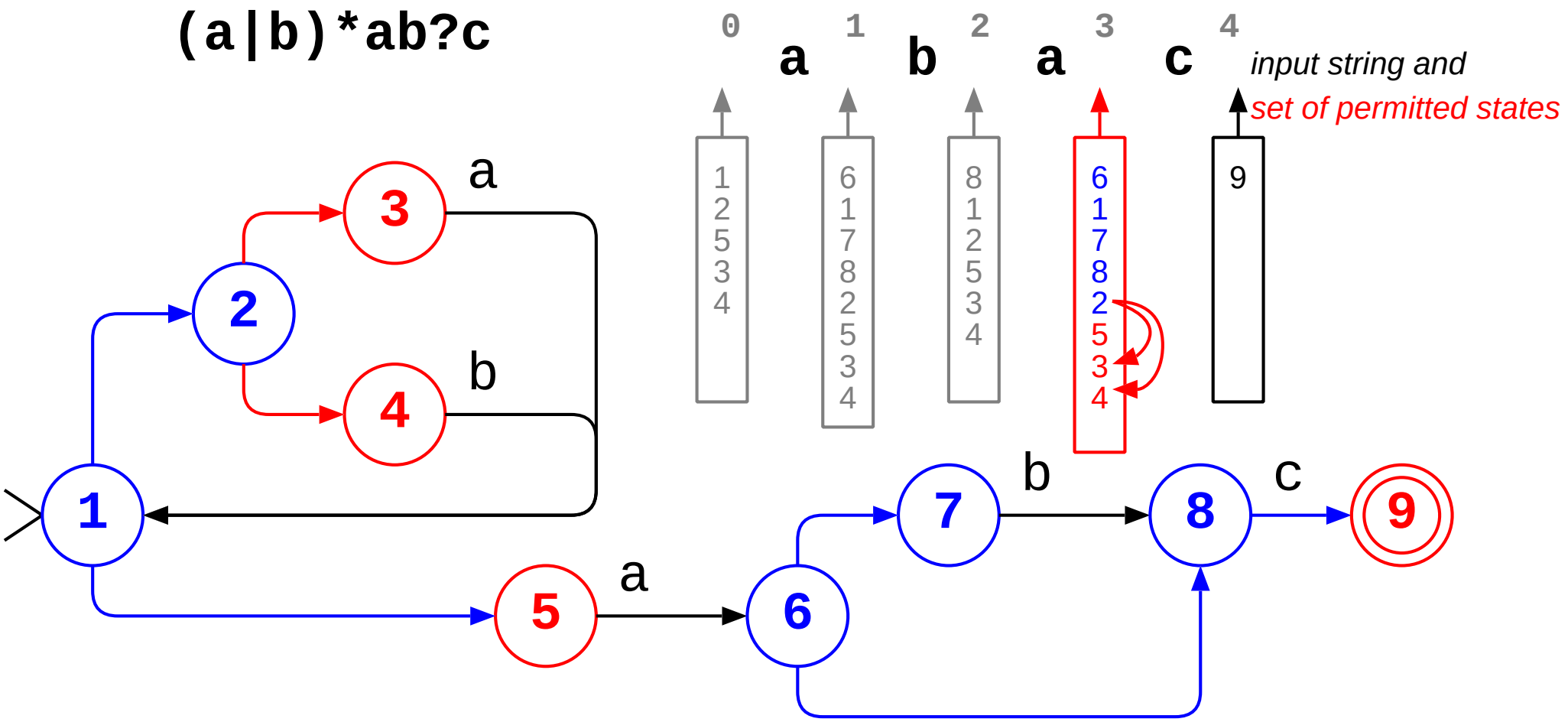
efficient recognition with NFAs



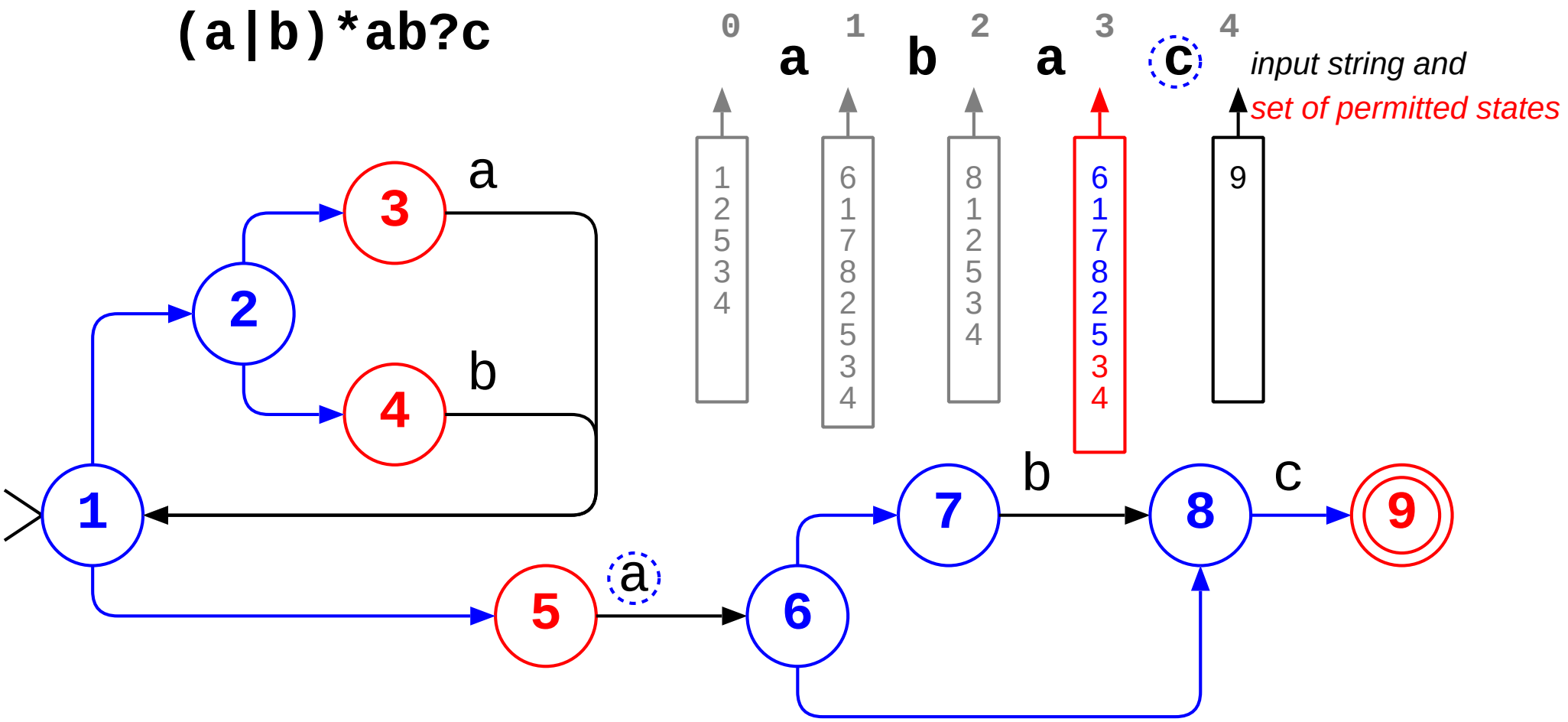
efficient recognition with NFAs



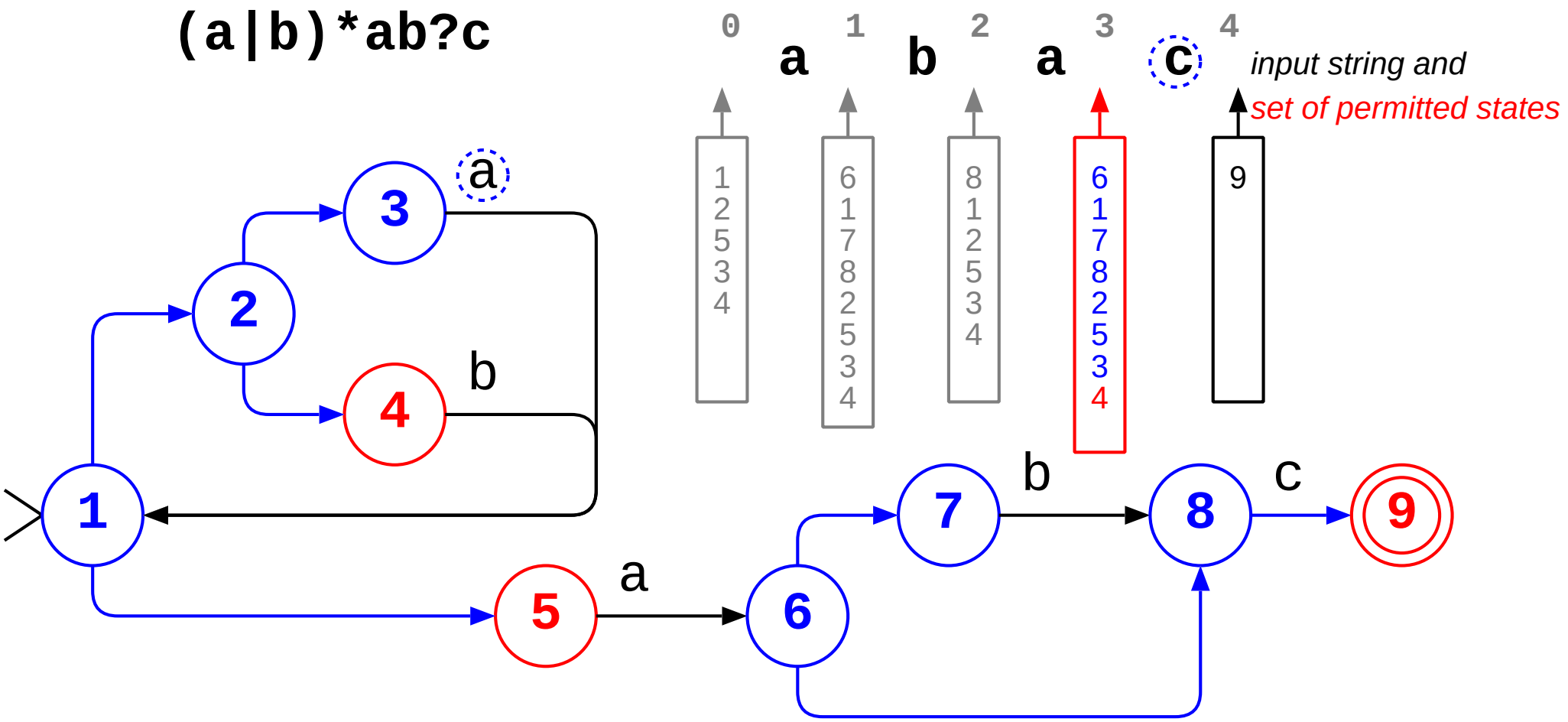
efficient recognition with NFAs



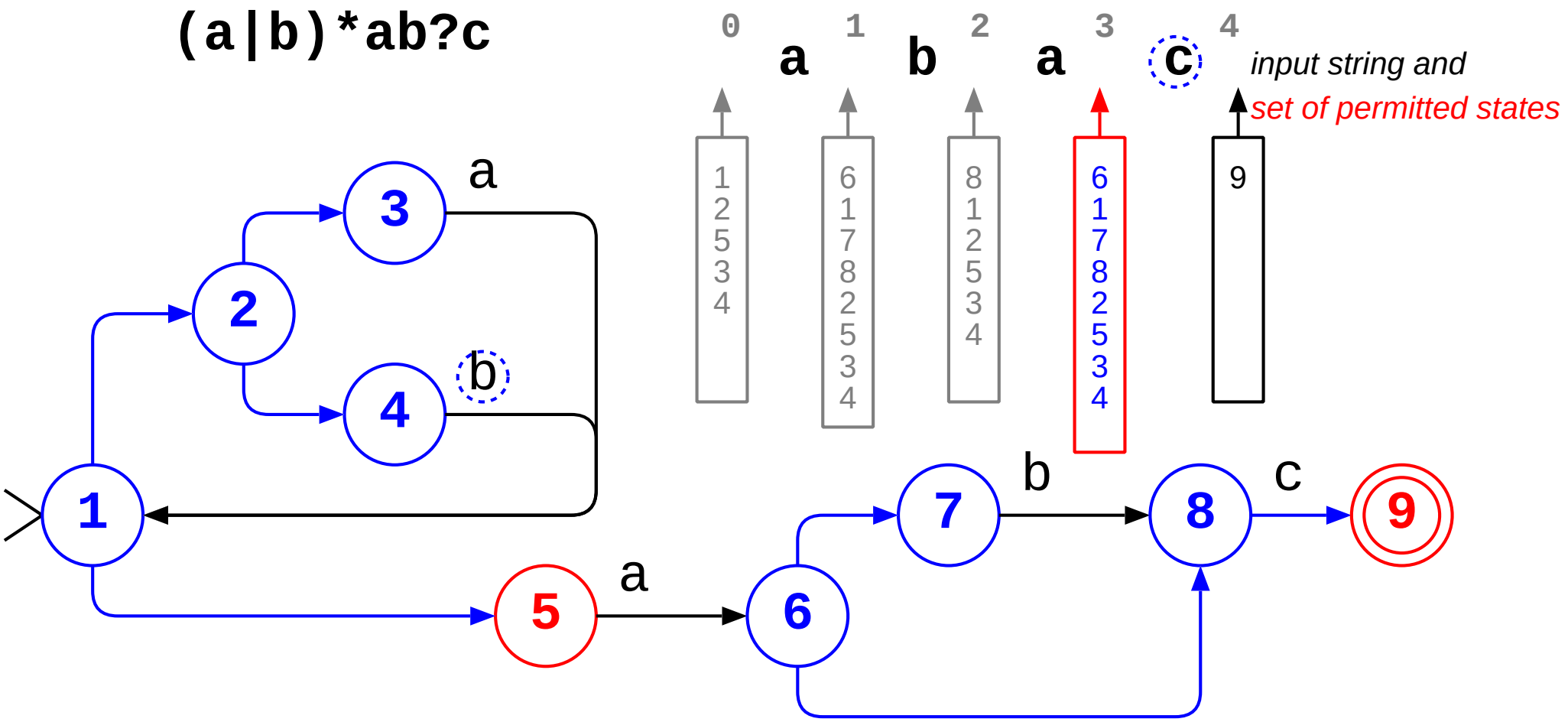
efficient recognition with NFAs



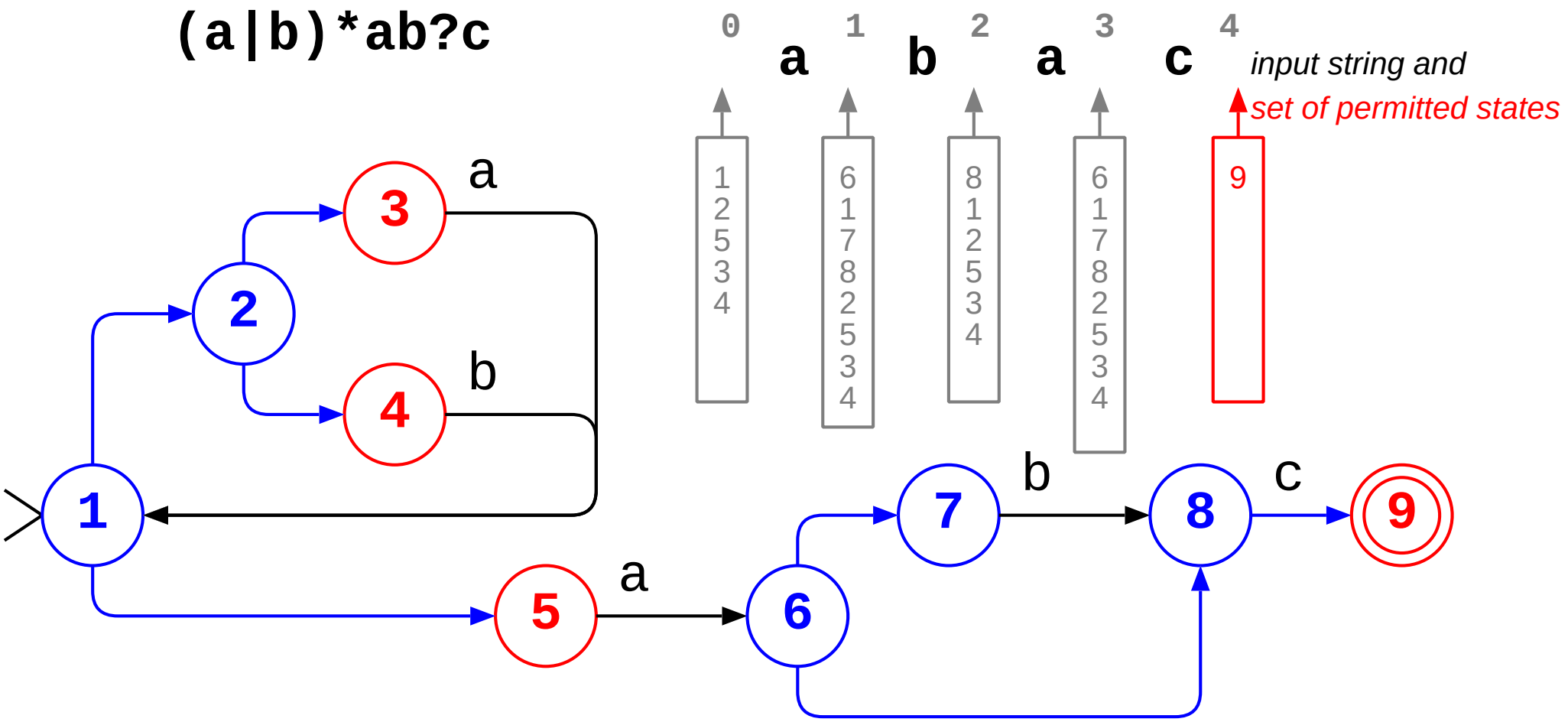
efficient recognition with NFAs



efficient recognition with NFAs

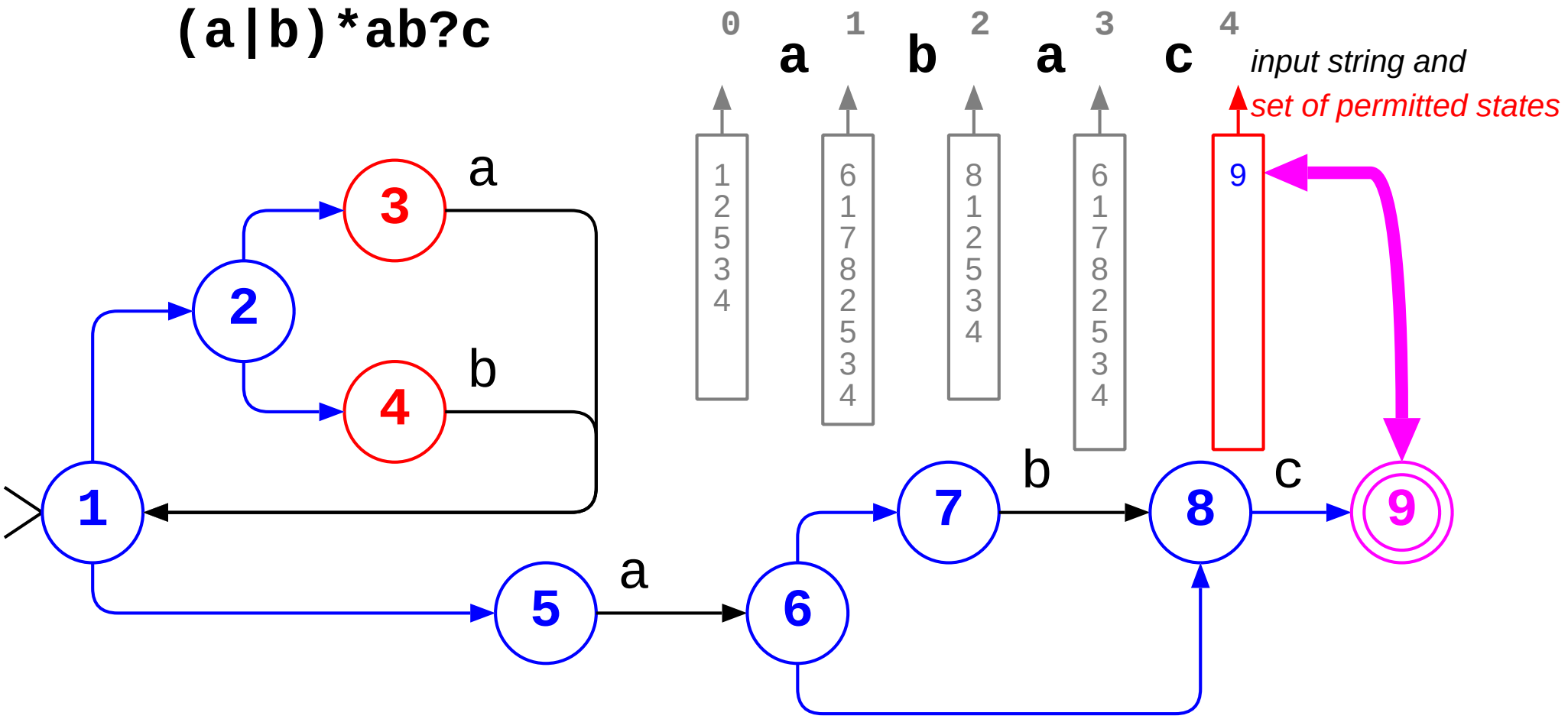


efficient recognition with NFAs

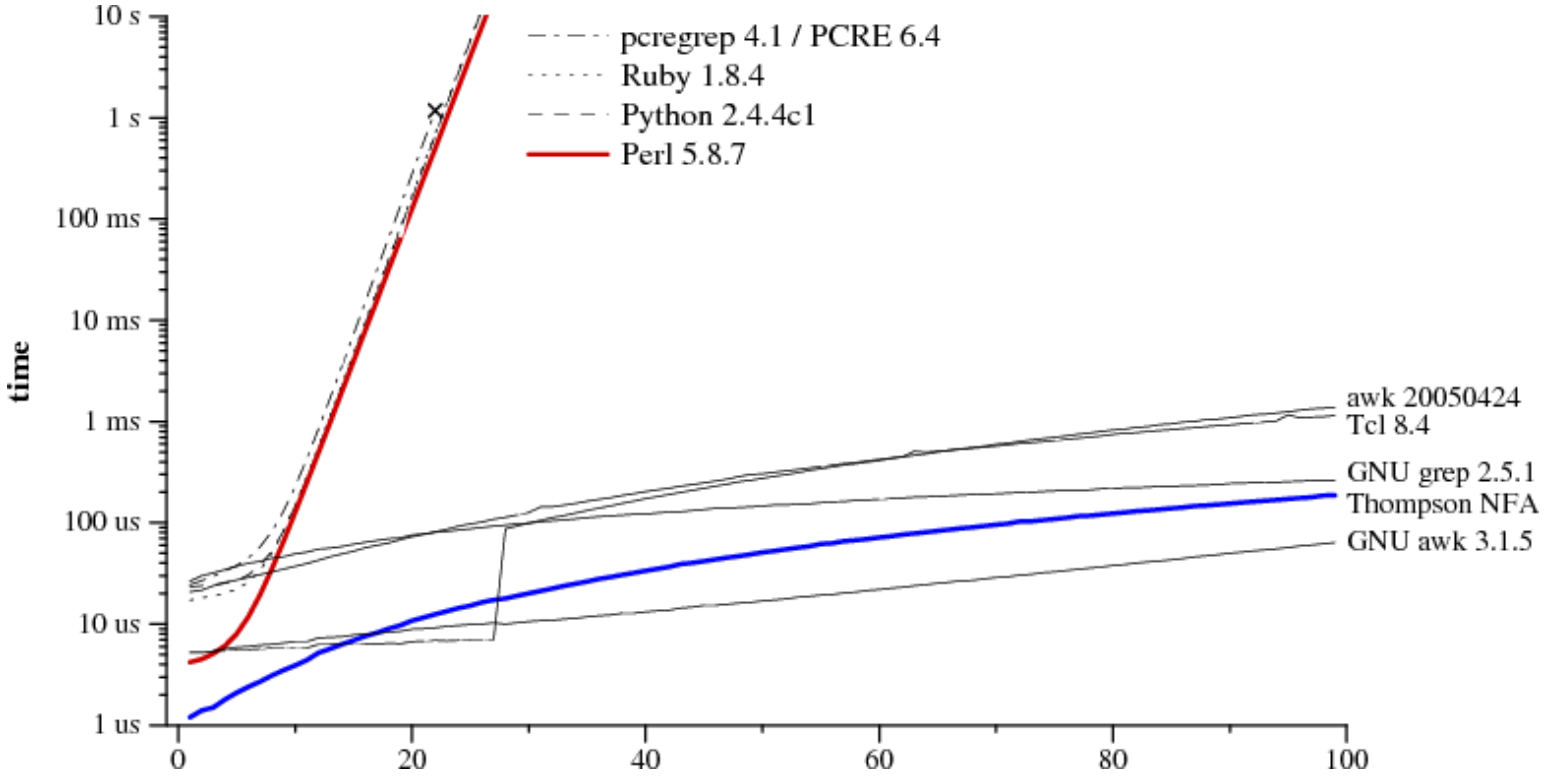


efficient recognition with NFAs

Success!



efficiency of NFA recognisers



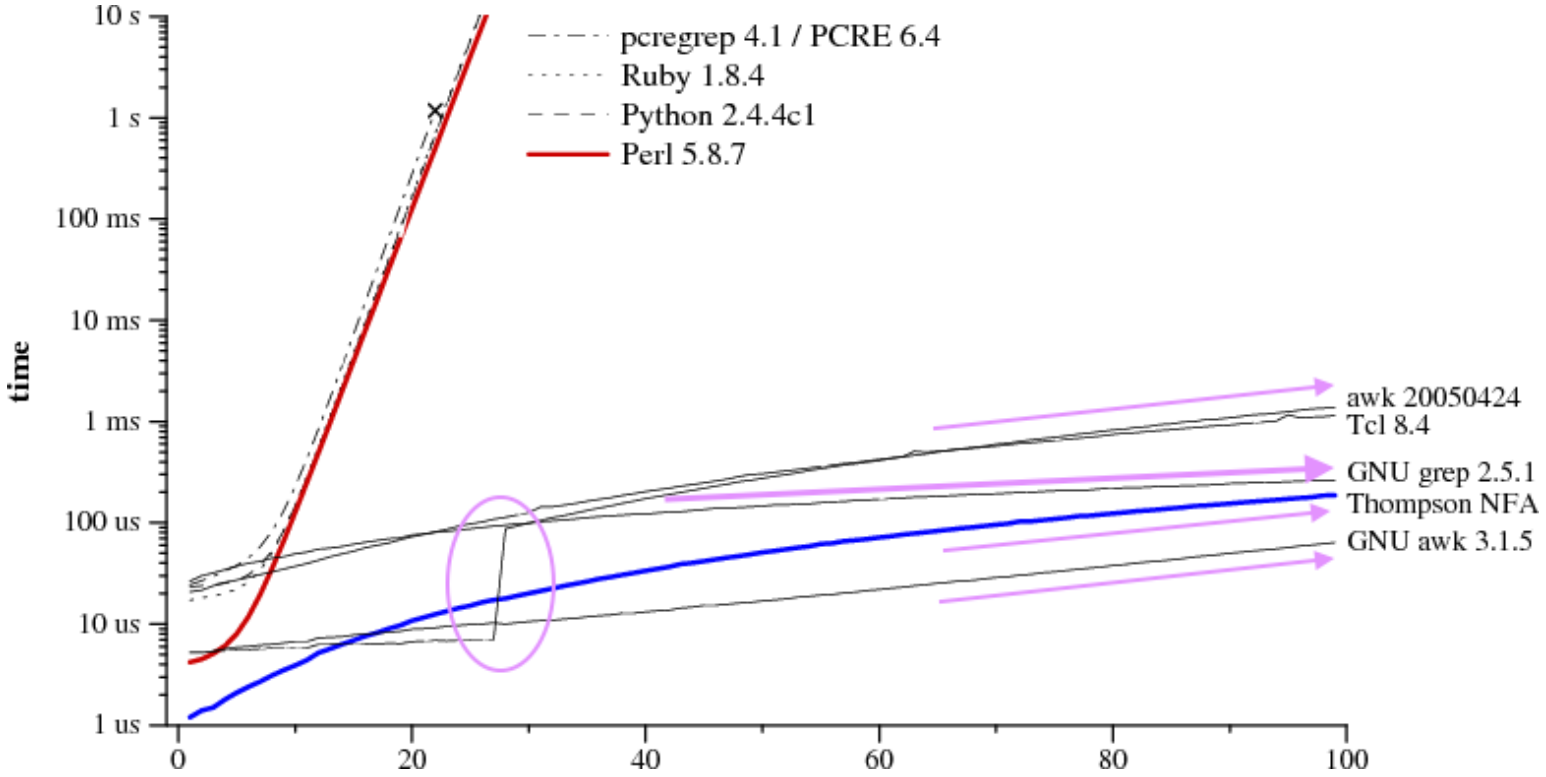
recursive back-tracking (in red)

- *exponential behaviour* when given difficult NFAs and input

parallel (in blue)

- close to *linear behaviour* even for difficult NFAs and input
- slope is quite high, though (cost of state set management)

efficiency of NFA recognisers



can do even better: convert NFA to *deterministic finite (state) automaton* (DFA)

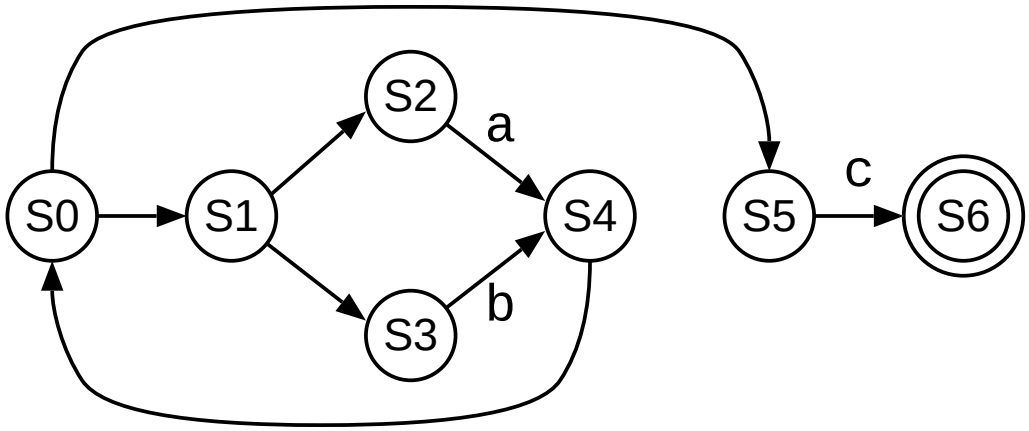
- linear behaviour (proportional to input size)
  + fixed overhead (proportional to expression size)
- *deterministic* means “always know exactly what to do next”

NFA properties

do not require input symbols for state transitions (ϵ-transition)

can transition to any number of new states from a given start state and input symbol

`(a|b)*c`



S	a	b	c	ϵ
0	-	-	-	{1 5}
1	-	-	-	{2 3}
2	{4}	-	-	-
3	-	{4}	-	-
4	-	-	-	{1}
5	-	-	{6}	-
6			*accept*	

can be slow

- recursive back-tracking, *or*
- state set management

DFA properties

contain only labelled transitions (there are no ϵ-transitions)

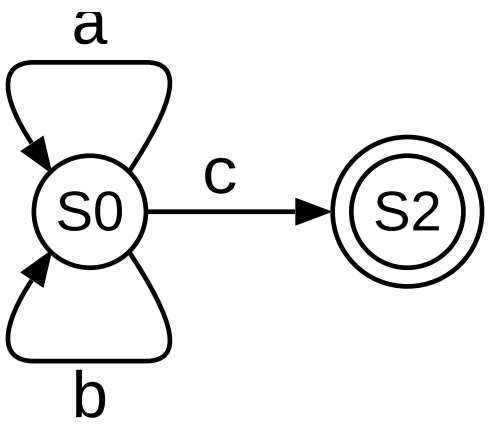
- a matching input symbol is required for all transitions

all transitions are *unambiguous*:

- current state $\times$ input symbol $\rightarrow$ next state (effectively instantaneous!)

transition table entries are *single* states (not sets)

`(a|b)*c`



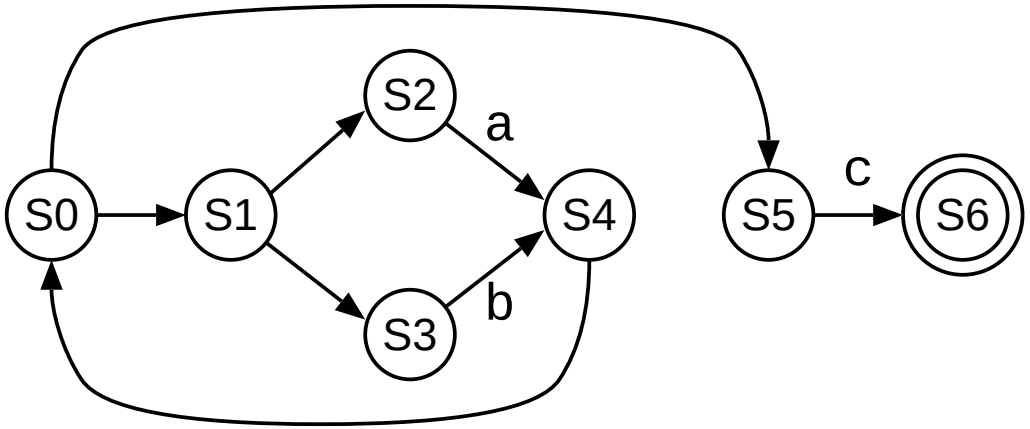
S	a	b	c
0	0	0	1
1	*accept*		

DFA is *much* more efficient than NFA; *linear* behaviour:

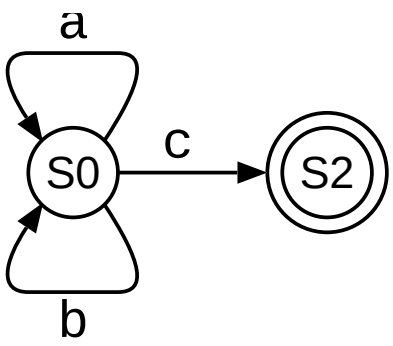
- for input size N, at most N transitions to accept/reject input

NFA to DFA

our next goal is to find an algorithm that will convert any NFA, e.g:



into an equivalent DFA, e.g:

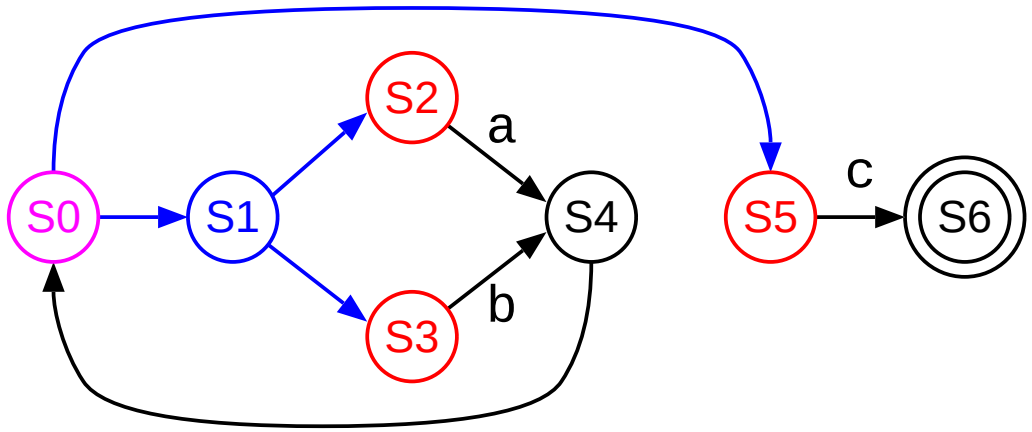


(for this we need to know about ϵ-*closures*)

ϵ-closures

the ϵ-closure of state S in a FSA M, $\epsilon\text{-closure}(S) =$

- the set containing S and
- all states reachable from S by following ϵ-transitions



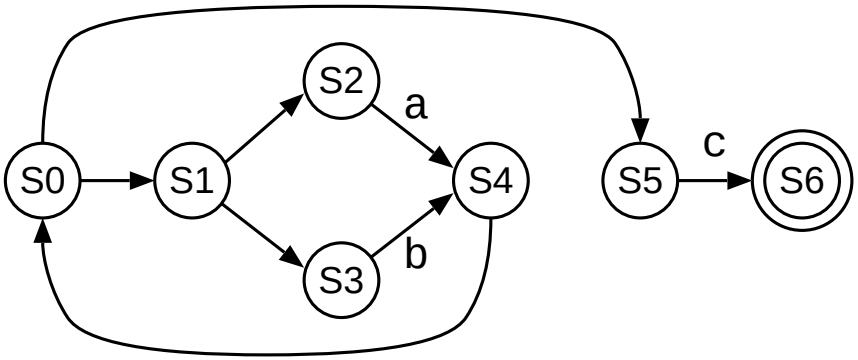
$$\epsilon\text{-closure}(S_0) = \{S_0, S_1, S_2, S_3, S_5\}$$

simplify by deleting states that have no labelled outgoing transitions

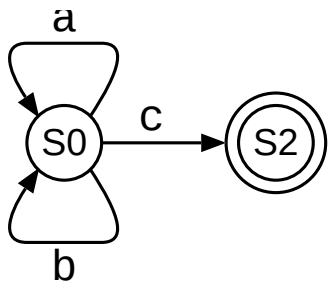
$$\epsilon\text{-closure}(S_0) = \{S_2, S_3, S_5\}$$

NFA to DFA: the goal

remember our goal: transform the NFA



into a DFA



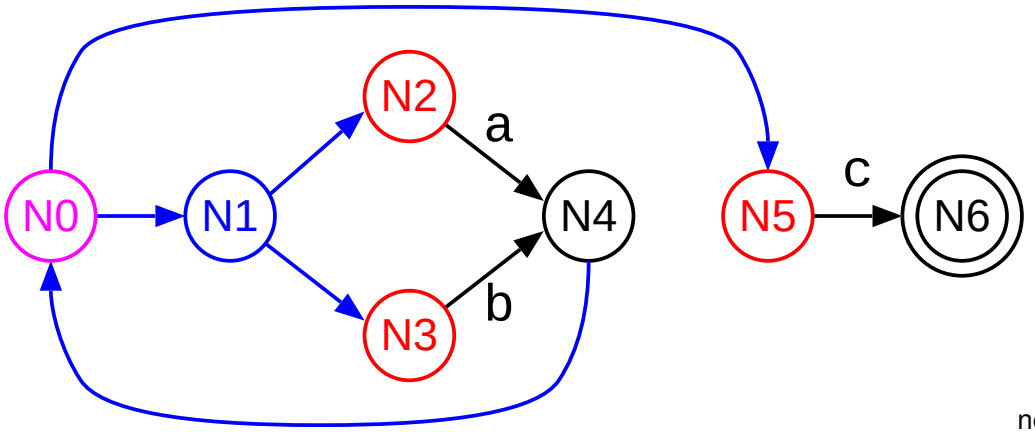
two steps; beginning from the NFA start state N_0 (i.e., with $i = 0$):

- create a new DFA state $D_i = \epsilon\text{-closure}(N_i)$
- for each symbol s in the alphabet
  - create a labelled transition $D_i \xrightarrow{s} D_j$
  - where $D_j = \bigcup \epsilon\text{-closure}(N_j)$ for all $N_i \xrightarrow{s} N_j, N_i \in D_j$

repeat with the target states of $D_i \rightarrow$ until no new D_j are possible

any D_i which contains an accepting N state is an accepting D state

NFA to DFA



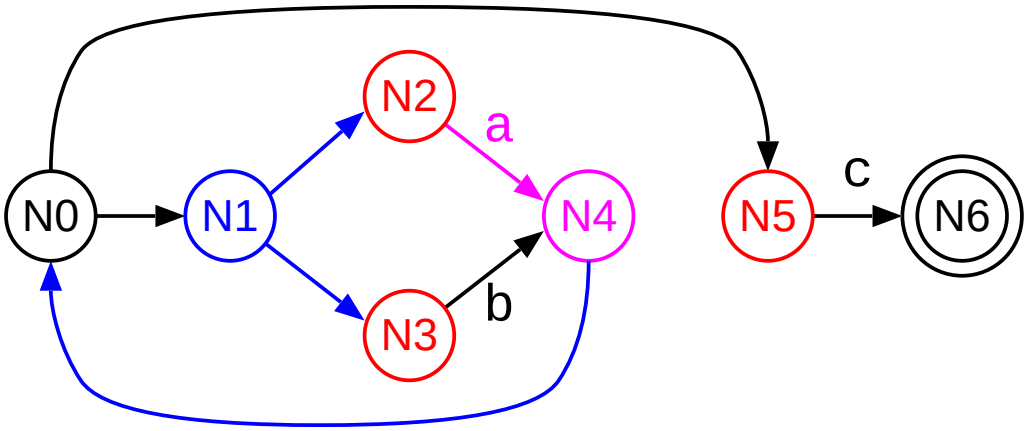
new $D_0 = \{N_2, N_3, N_5\}$

Step 1: construct $D_0 = \epsilon\text{-closure}(N_0)$

$$D_0 = \{N_2, N_3, N_5\}$$

Step 2: take any new state (e.g., D_0) and find its transitions

NFA to DFA



$D_0 = \{N_2, N_3, N_5\}$

Step 2a: find the ϵ-closure of target states for $D_0 \xrightarrow{a}$

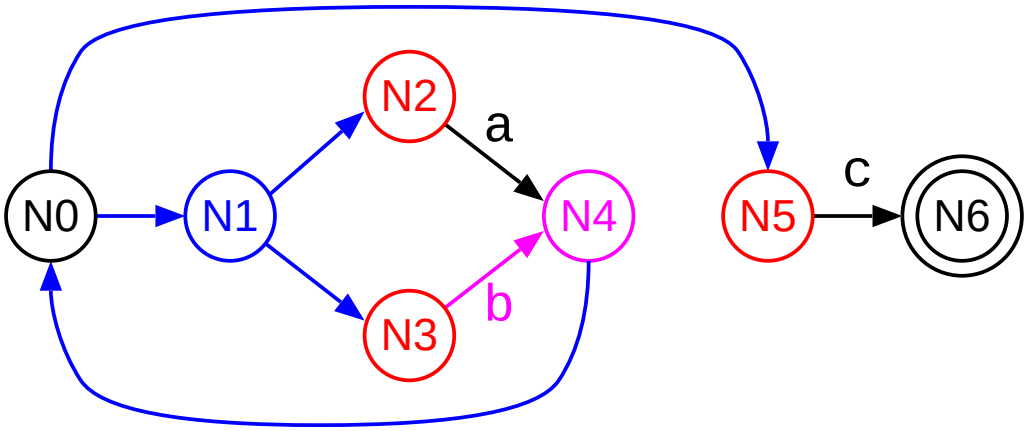
- $N_2 \xrightarrow{a} N_4$
- $\epsilon\text{-closure}(N_4) = \{N_2, N_3, N_5\} = D_0$

$\Rightarrow\ D_0 \times \mathrm{a} = D_0$

Transition Table:

	a	b	c
D_0	D_0		

NFA to DFA



$D_0 = \{N_2, N_3, N_5\}$

Step 2b: find the ϵ-closure of target states for $D_0 \xrightarrow{b}$

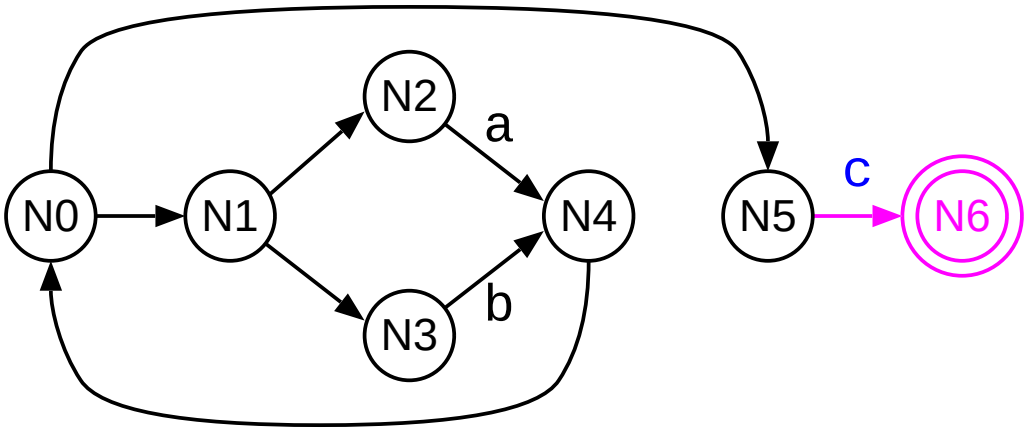
- $N_3 \xrightarrow{b} N_4$
- ϵ-closure$(N_4) = \{N_2, N_3, N_5\} = D_0$

$\Rightarrow \; D_0 \times \mathrm{b} = D_0$

Transition Table:

	a	b	c
D_0	D_0	D_0	

NFA to DFA



$D_0 = \{N_2, N_3, N_5\}$
new $D_1 = \{N_6\}$

Step 2c: find the ϵ-closure of target states for $D_0 \xrightarrow{c}$

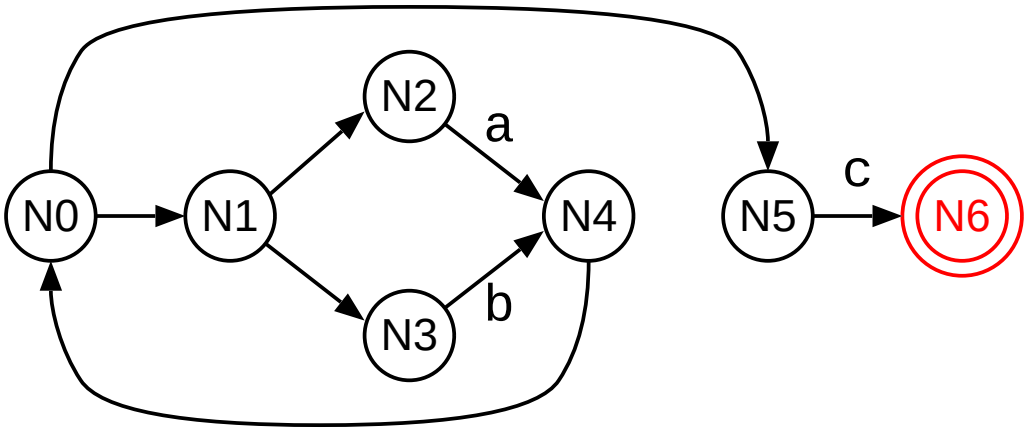
- $N_5 \xrightarrow{c} N_6$
- ϵ-closure$(N_6) = \{N_6\} = D_1$ (N_6 accepts $\Rightarrow D_1$ accepts too)

$\Rightarrow$ $D_0 \times \mathrm{c} = D_1$

Transition Table:

	a	b	c
D_0	D_0	D_0	D_1
D_1		*accept*	

NFA to DFA



$D_0 = \{N_2, N_3, N_5\}$
$D_1 = \{N_6\}$

Step 3: take any new state (e.g., D_1) and find its transitions

Step 3a: No $N_i \in D_1$ has a labelled transition, so nothing to do

Step 4: there are no more new states D_i, so we are finished

Transition Table:

	a	b	c
D_0	D_0	D_0	D_1
D_1		*accept*	

NFA to DFA

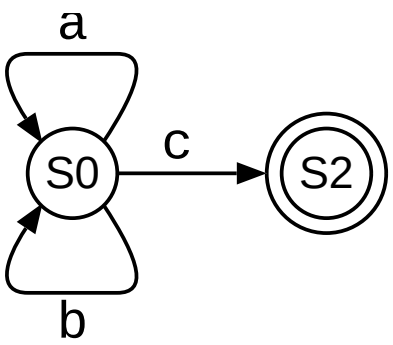
the DFA can be drawn from the transition table

states(D) = $D_0 = \{N_2, N_3, N_5\}$
$D_1 = \{N_6\}$

transitions(D) =

	a	b	c
D_0	D_0	D_0	D_1
D_1		*accept*	

DFA:



this DFA recognises sentences of `(a|b)*c` in *linear* time

- time proportional to the length of the input string

NFA to DFA: the 'subset construction'

let

- a_i be the input sentence alphabet characters
- N_i be the states of an NFA
- D_i be the states of the corresponding DFA
  - each D_i is a set of N_i states
- $\epsilon\text{-closure}(N_i)$ be

  the set of all states reachable (directly or indirectly) from N_i by following ϵ-transitions
- $\epsilon\text{-closure}(D_i)$ be the union of the ϵ-closures of all $N_i \in D_i$

$$\epsilon\text{-closure}(D_i) = \bigcup_{N_j \in D_i} \epsilon\text{-closure}(N_j)$$

we represent a DFA D as

- states(D), the set of states D_i in D
- transitions(D), a set of transitions $D_i \times a_n \to D_j$ in D

NFA to DFA: the ‘subset construction’

let U be an empty set, and N_0 the start state of the NFA

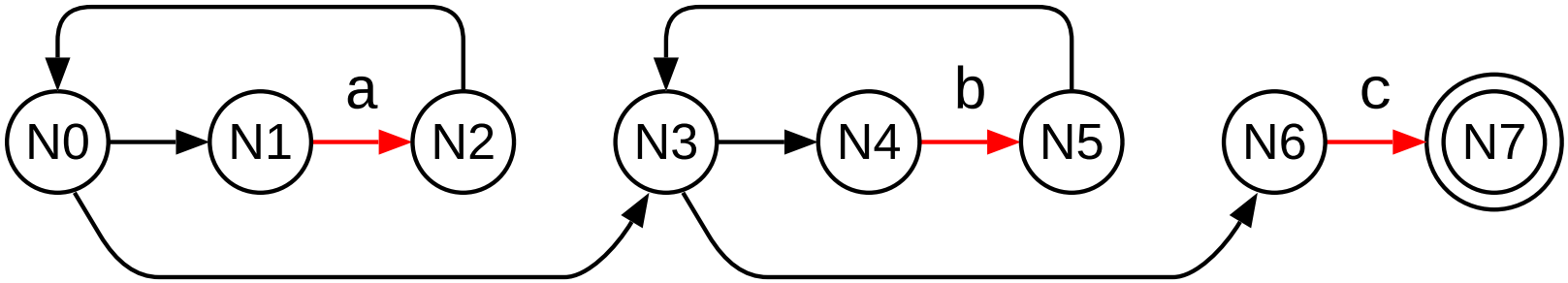
to construct the corresponding DFA from a NFA:

let D_0 (the start state of the DFA) be $\epsilon\text{-closure}(N_0)$
add D_0 to U (the ‘unexplored’ DFA states)
while U is not empty :
    remove an element D_i from U
    for every symbol a_n of the input alphabet :
        let D_m = the set of states $\{N_j : N_i \in D_i \wedge N_i \xrightarrow{a_n} N_j\}$
        let D_n = $\epsilon\text{-closure}(D_m)$
        add $D_i \times a_n \rightarrow D_n$ to transitions(D)
        if any N_i in D_n is an accepting state, then D_n is accepting
        if D_n is not already in states(D) :
            add D_n to states(D)
            add D_n to U

states(D) and transitions(D) now contain the DFA

example NFA to DFA

a∗b∗c



$D_0 = \{N_0\ N_1\ N_3\ N_4\ N_6\}$

$D_0 \times a \rightarrow \{N_0\ N_1\ N_3\ N_4\ N_6\} = D_0$ (from N_1)

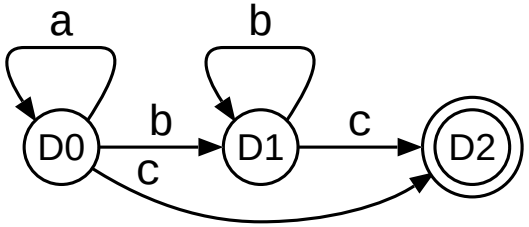
$D_0 \times b \rightarrow \{N_3\ N_5\ N_6\} = D_1$ (from N_4)

$D_0 \times c \rightarrow \{N_7\} = D_2$ (from N_6)

$D_1 \times b \rightarrow \{N_3\ N_5\ N_6\} = D_1$ (from N_4)

$D_1 \times c \rightarrow \{N_7\} = D_2$ (from N_6)

D	a	b	c
0	0	1	2
1	-	1	2
2	*accept*		



next week

the mathematics of languages

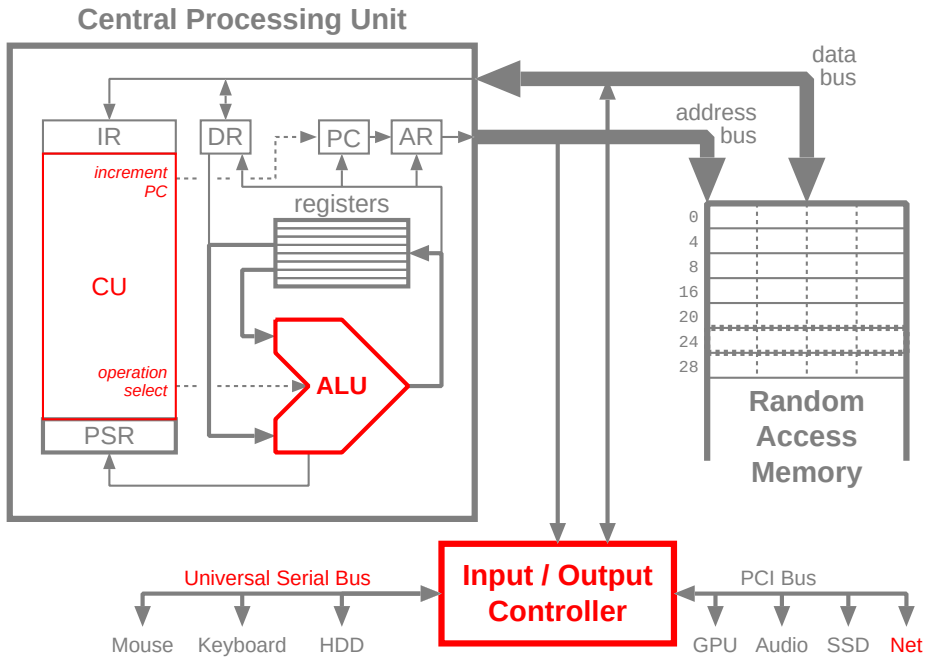
- grammars

regular grammars

- relationship to FSMs
- lack of memory
- limitations

FSMs with memory

- push-down automata



homework

reinforce your understanding

- write a NFA simulator in Python
  - invent a representation for NFA states
  - and another for state sets
  - code the algorithm for generating new states
- write a NFA to DFA converter in Python
  - write a function that compares two state sets
  - and another that generates ϵ-closures
  - code the subset construction algorithm
- write a DFA engine in Python (easy!)
  - current state $+$ current input $=$ new current state
  - repeat until you reach the final state

ask about anything you do not understand

- from any of the classes so far this semester (or the lecture notes)
- it will be too late for you to try to catch up later!
- I am always happy to explain things differently and practice examples with you

glossary

ambiguous — not having an obvious meaning or solution. In a FSM, a state that has more than one outgoing transition that can be taken, either because of ϵ-transitions or because the same label appears on more than one transition.

back-tracking — a method of running a NFA in which easy possible path is attempted until the correct one is found. When an input symbol does not match any transitions from the current state, the machine backs up to an earlier decision (now known to have been wrong) and tries again making a different decision.

deterministic — a system that is free of ambiguity, in which every decision can be made (or predicted) with absolute certainty.

deterministic finite automaton (DFA) — a finite automaton in which there is no ambiguity. There are no ϵ-transitions, and never more than one transition with a given label out of a state.

ϵ-closure — the transitive closure of ϵ-transitions from a given state in a FSM.

exponential — (algorithm) behaviour in which a fixed addition to the problem complexity leads to a fixed multiplication in the amount of time required to solve the problem. For an exponential problem of size N, the time t taken to solve the problem is $t = k^N$ where k is a constant.

linear behaviour — (algorithm) behaviour in which a fixed addition to the problem complexity leads to a fixed addition in the the amount of time required to solve the problem. For a linear problem of size N, the time t taken to solve the problem is $t = kN$ where k is a constant.

non-deterministic — a system that contains ambiguity, in which decisions cannot be made (or predicted) in advance and may eventually turn out to have been incorrect.

non-deterministic finite automaton (NFA) — a finite automaton in which there is ambiguity. There might be multiple ϵ-transitions out of states, and a given state might contain multiple outgoing transitions with the same label.

state set — a set of FSM states, often containing all of the states that can be reached at a given position in the sequence of input symbols.

transitive closure — the set of related items obtained when a transitive operation is applied repeatedly from some starting element until no further elements can be added. For example, the transitive closure of the $<$ relation starting from the element 42 in the counting numbers is the set of all counting numbers greater than 42. The transitive closure of ϵ-transitions in a FSM, starting from some given state S, is the set of all states that are directly or indirectly reachable from S by following one or more ϵ-transitions.

unambiguous — having a single, obvious meaning or solution. In an unambiguous FSM, the single correct transition to follow from any state can be determined trivially by looking at the input symbol and following the single corresponding transition (if any) to the next state.