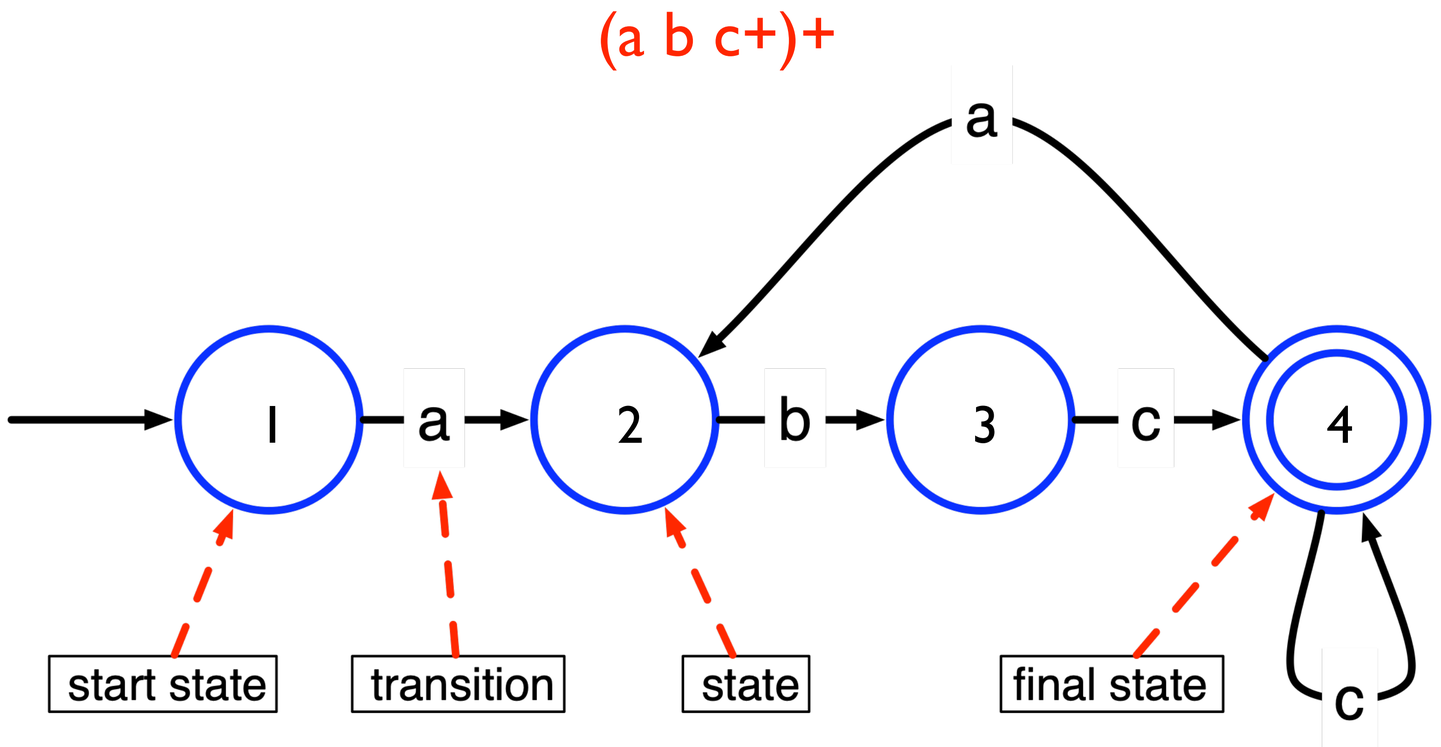
Finite Automata

so you have a regex. now what?

- A regular expression *defines* a regular language
  - Tells us what strings (words) are in the language
- But how do we *recognize* a regular language?
  - Remember that a scanner needs to find the words in a language
  - The problem it is solving is: "does this sequence of characters *match* this regular expression?"
- Key enabling feature: a systematic procedure for converting a regular expression into code that recognizes when a string matches a regular expression

finite automata

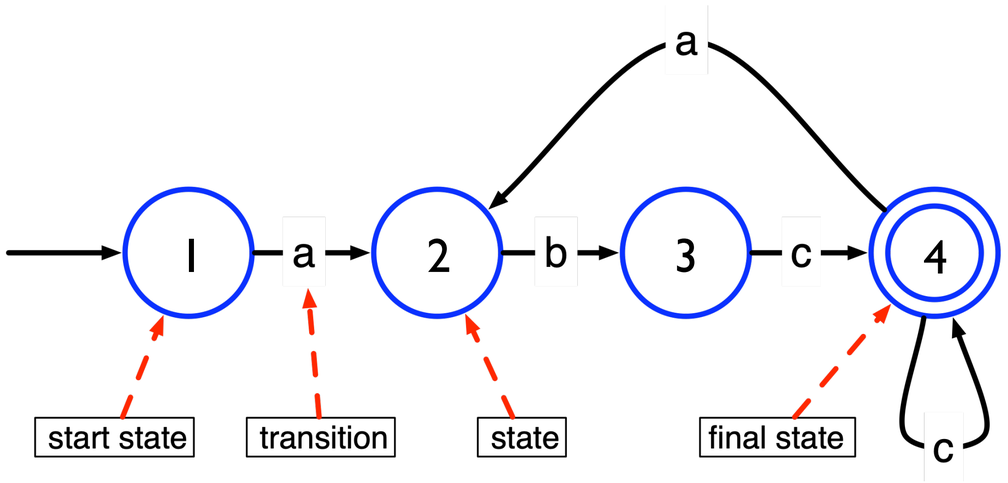
- Finite state machine that will only *accept* a string if it is in the set defined by a regular expression



transition tables

- Table encoding states and transitions of FA
  - 1 row per state, 1 column per possible character
  - Each entry: if automaton in a particular state sees a character, what is the next state?

State	Character		
	a	b	c
1	2		
2		3	
3			4
4	2		4



Non-deterministic Finite Automata

deterministic vs non-deterministic

- An automaton is **deterministic** if at each step there is only one possible transition to take on a given character
  - No choices to be made
- But an automaton can be **non-deterministic**: at a particular state, there may be more than one transition to take on a given character
- Can also have λ-transitions—transitions that consume no characters
  - Think of these as *optional* transitions: from a state, the machine can take a λ transition "for free"

“running” a non-deterministic automaton

- Intuition: take every possible path through an NFA
  - Think: parallel execution of NFA
  - Maintain a “pointer” that tracks the current state
  - Every time there is a choice, “split” the pointer, and have one pointer follow each choice
  - Track each pointer simultaneously
    - If a pointer gets stuck, stop tracking it
    - If any pointer reaches an accept state at the end of input, accept

example

- How does the automaton below handle the string aba