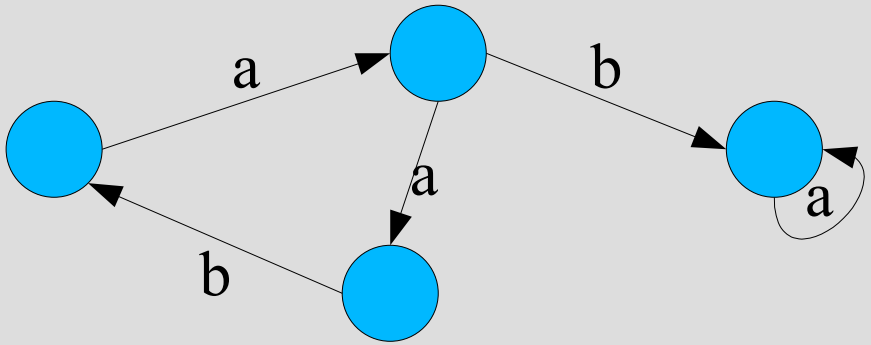
State machines and strings

Bruce Merry

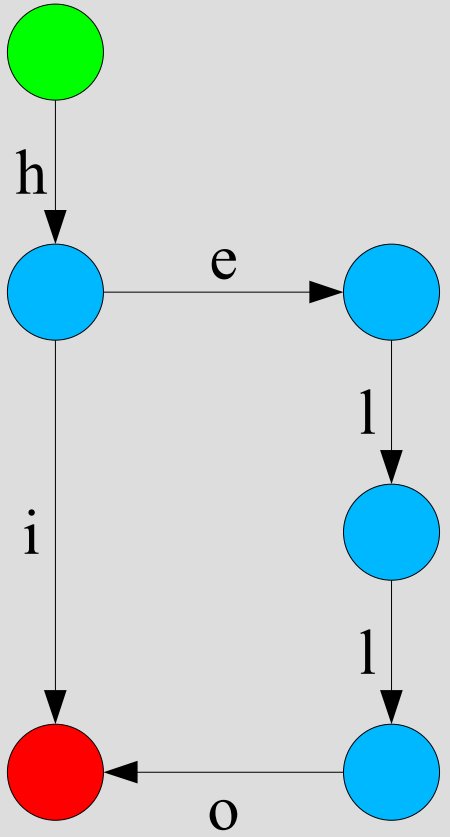
Finite state machine

- Conceptual machine that processes a string of symbols
- Has only one piece of memory: the **state**
- Based on the current state and the next symbol, we transition to a new state



Special states

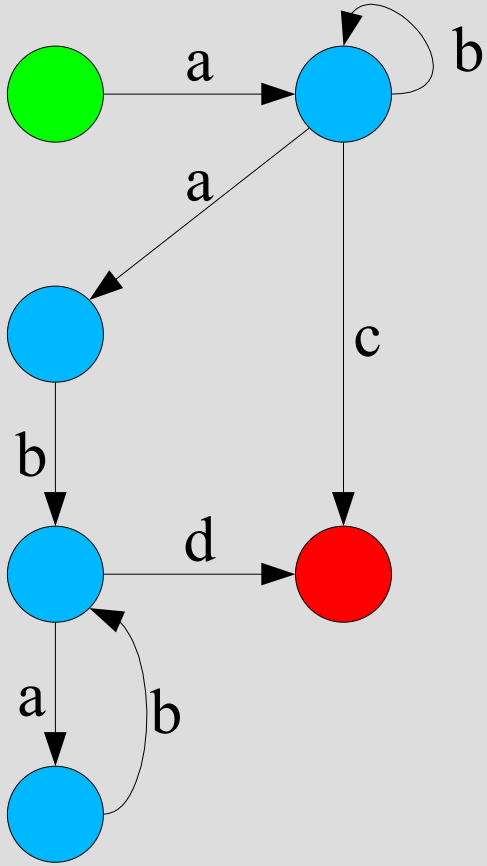
- start state
- accept state
- matches `hi|hello`



Regular expressions

- Easy for some expressions:

  `a(b*(ab)+d|c)`

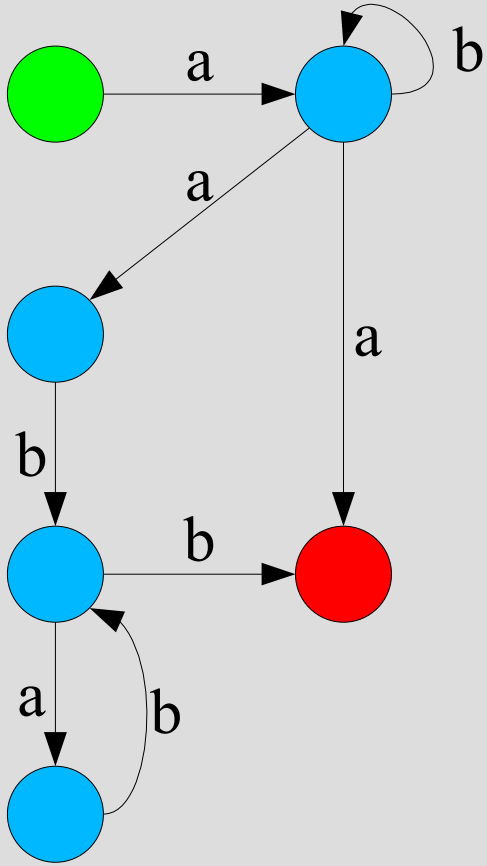


Non-determinism

- What about

  `a(b*(ab)+b|a)`

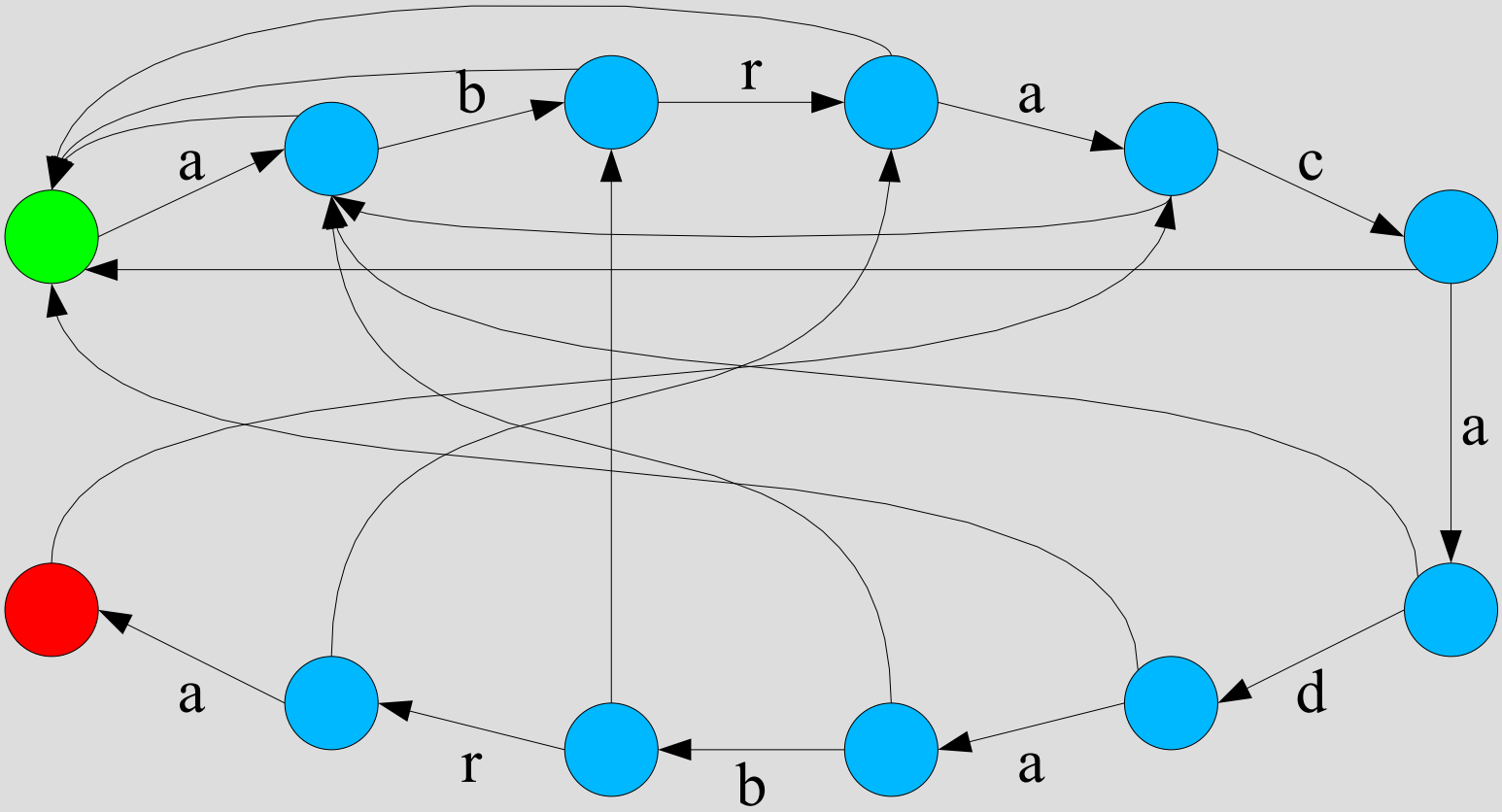
- A NFSM “guesses”
- To simulate, track all possibilities
- Backreferences not possible with FSM



Knuth-Morris-Pratt

- String searching
- Like Boyer-Moore, is O(N+M) in worst case
- Unlike Boyer-Moore, is O(N+M) on average
- Processes the haystack one letter at a time
- Keeps track of how much of the needle is matched at the current point
- State machine used to update the “how much”

KMP state machine



KMP search

- matched = 0
- for each haystack letter X
  - while matched != 0 and X does not match
    - matched = failure[matched]
  - if X matches, matched++

Building the failure function

- Bootstrap by running KMP on itself:
- for each i:
  - failure[i] = failure[i – 1]
  - while needle[i] != needle[failure[i]]
    - failure[i] = failure[failure[i]]
  - if needle[i] == needle[failure[i]]
    - failure[i]++

Multi-string search

- KMP uses a linear state machine with failure transitions
- To search multiple strings, structure the FSM as a trie and use failure transitions
- Not trivial to bootstrap: it needs to be done breadth-first.

Multi-string search