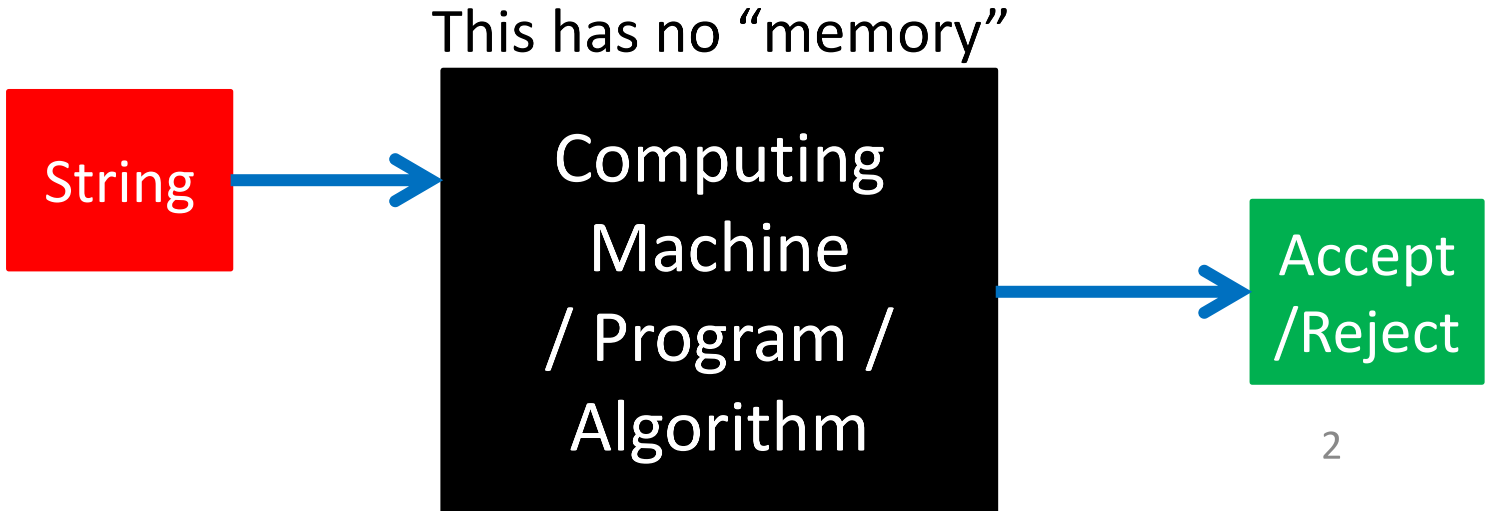
CS3102 Theory of Computation

Recap

- So far:
  - Mathematical foundations (sets, functions, etc.)
  - Finite State automata (NFAs, DFAs)
  - Regular expressions



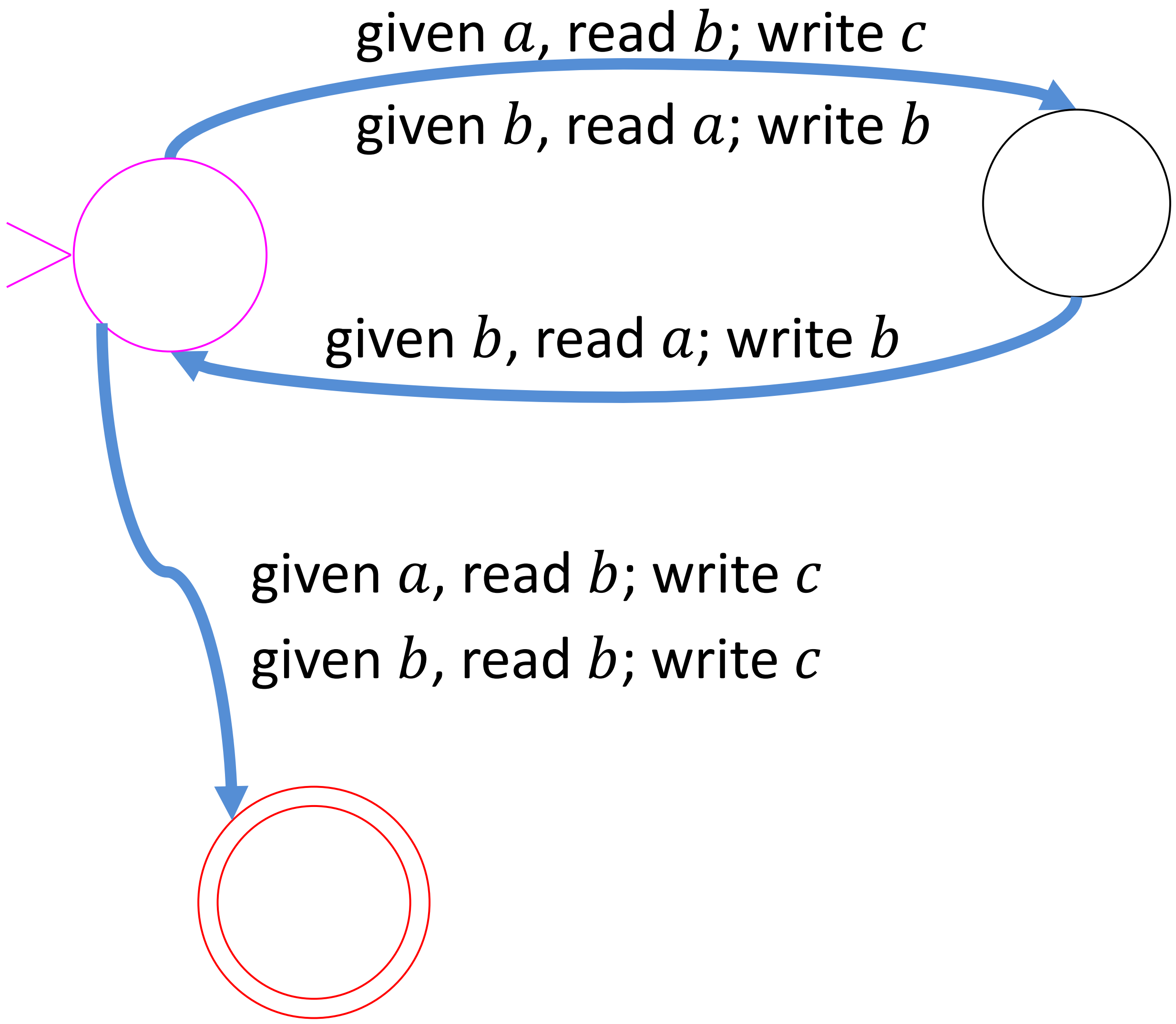
Computing with Memory

- What do we mean by “memory”?
  - There are potentially an infinite number of machine “configurations”
- What do we mean by “configuration”?
  - Snapshot/callstack of the computation
  - In Java: What line are we on in which function, what is in each variable/data structure?
  - In DFAs/NFAs: Which state(s) are we in?
- What gives more computing power?
  - Longer and longer inputs can result in larger configurations

Adding memory to a FSA

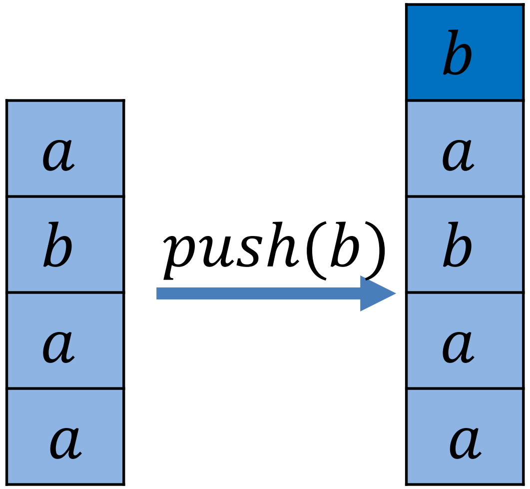
- Idea: Allow FSA to read from/write to memory
- Reading from memory:
  - FSA will transition depending on both the input and some location in memory
- Writing to memory:
  - When transition, FSA can change what’s in the memory

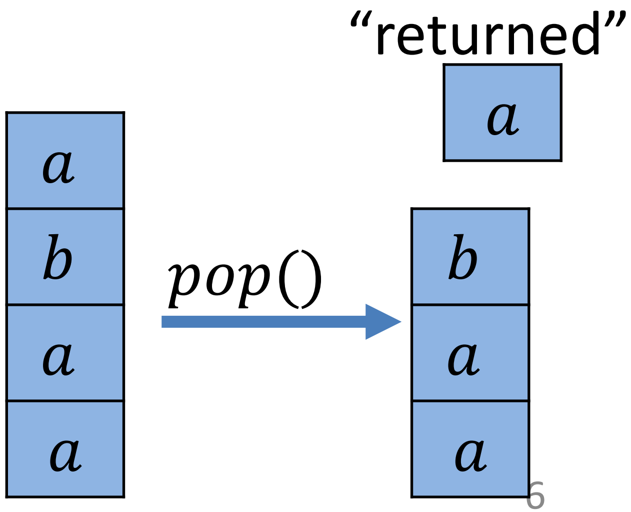
Vague example



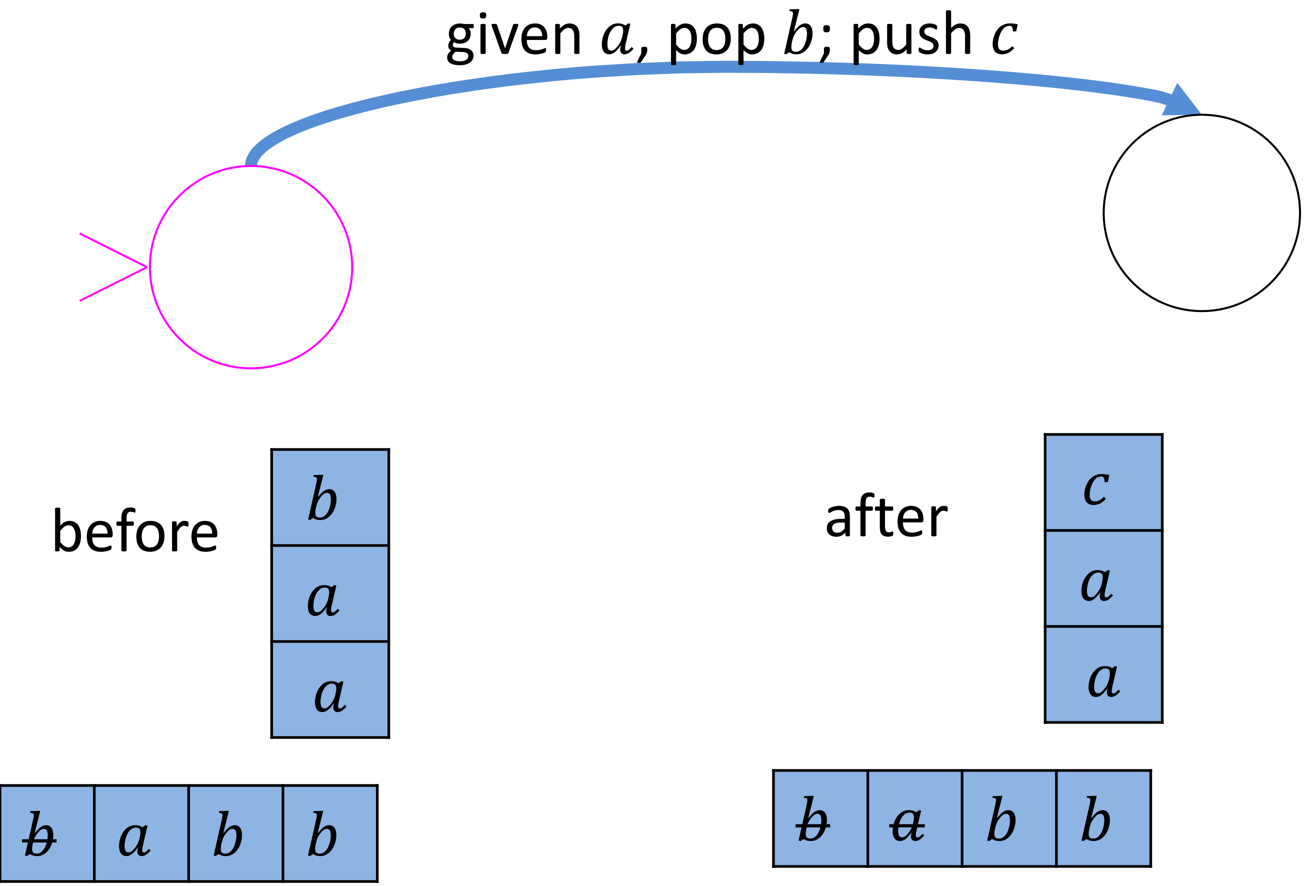
How to use the memory?

- Let's start with something simple: a stack
- Stack operations:
  - $push(x)$: place item x on top of everything else
  - $pop()$: remove the thing on top and return it





Less Vague example



We take this transition when:

1. We're in the start state
2. The next character of the input is a
3. The character on the top of the stack is b

Which results in:

1. Changing states
2. Consuming that input charcter
3. Removing b from the top of the stack
4. Pushing c to the top of the stack

Pushdown Automata

Basic idea: a pushdown automaton is a finite automaton that can optionally write to an unbounded stack.

- Finite set of states: $Q = \{q_0, q_1, q_2, \dots, q_k\}$ q_1
- Input alphabet: Σ
- Stack alphabet: Γ
- Transition function: $\delta: Q \times (\overset{\text{Input}}{\Sigma \cup \{\varepsilon\}}) \times \overset{\text{popped}}{\Gamma} \to \overset{\text{New states and stack}}{2^{Q \times \Gamma^*}}$
- Initial state: $q_0 \in Q$ $\rightarrow q_0$
- Final states: $F \subseteq Q$ q_k

$$q_1 \xrightarrow{a,\, b,\, c} q_2$$

Input, popped, push

Pushdown automaton is $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$

Pushdown Automata

- Stack alphabet can be different from input alphabet
- Typically non-deterministic
  – Can be in multiple states at once
  – Can have multiple “parallel” stacks
  – Non-deterministic “configurations”
- Accept when:
  – The entire input has been read
  – There is at least one “path” in a final state with an empty stack

Non-Vague Example

PDA for $a^n b^n$

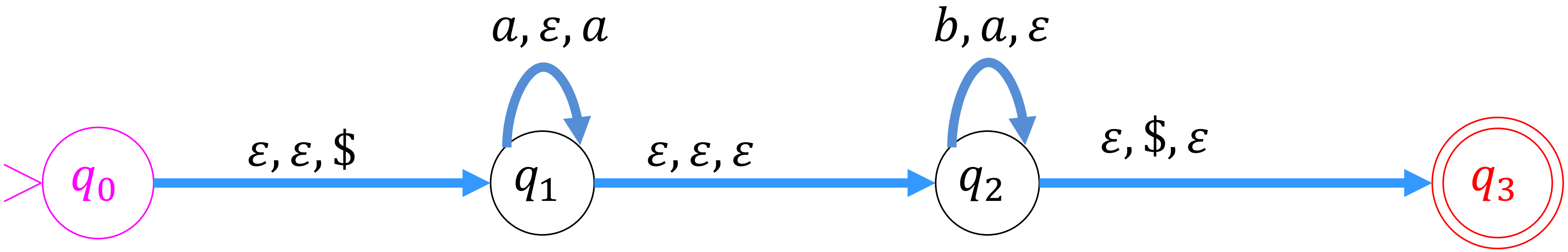
Strategy:

1. Push \$ on the stack to identify the bottom
2. For each a I see, push a onto the stack
3. For each b I see, pop a off of the stack
4. Reject if I see another a
5. Accept if I pop \$ after the last b

PDA for $a^n b^n$

Consume a from input
Don’t pop anything
Push a

Consume b from input
Pop a from the stack
Don’t push anything



Consume no input
Don’t pop anything
Push \$

Consume no input
Don’t pop anything
Don’t push anything

Consume no input
Pop \$ from the stack
Don’t push anything

PDA for “Palindromes”

PDA for wxw^R where $w \in \{a, b\}^*$

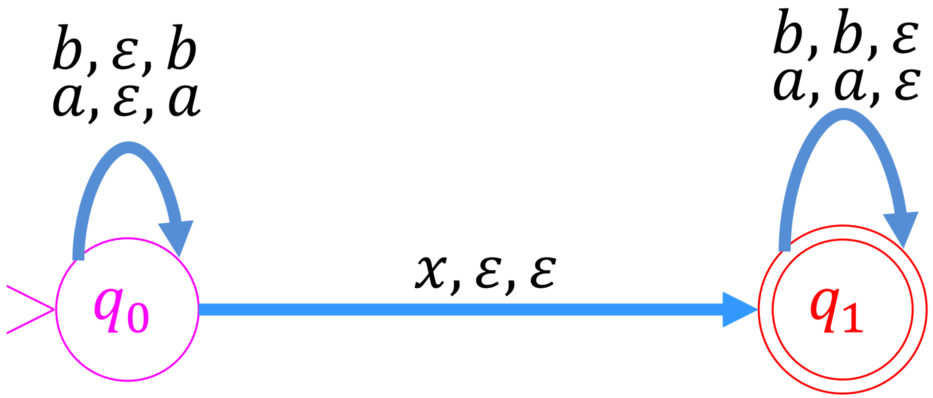
Strategy:

1. For each a I see, push a onto the stack
2. For each b I see, push b onto the stack
3. See the x
4. For each a I see, pop a off of the stack
5. For each b I see, pop b off of the stack

PDA for wxw^R

Consume a/b from input
Don't pop anything
Push a/b respectively

Consume a/b from input
Pop a/b respectively
Don't push anything



Consume x from input
Don't pop anything
Don't push anything

PDA for even-length Palindromes

PDA for ww^R where $w \in \{a, b\}^*$

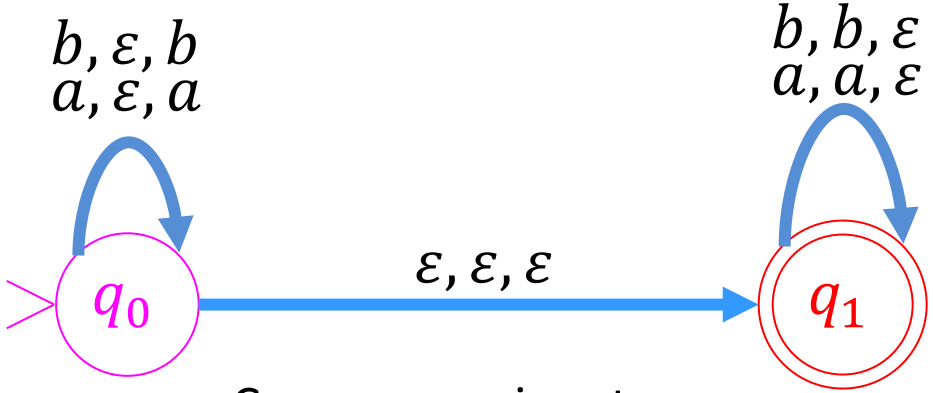
Strategy:

1. For each a I see, push a onto the stack
2. For each b I see, push b onto the stack
3. Guess that this is the middle
4. For each a I see, pop a off of the stack
5. For each b I see, pop b off of the stack

PDA for ww^R

Consume a/b from input
Don’t pop anything
Push a/b respectively

Consume a/b from input
Pop a/b respectively
Don’t push anything



Consume no input
Don’t pop anything
Don’t push anything

(going from forward
half to backwards
half of string)

PDA for Palindromes

PDA for $w = w^R$ where $w \in \{a, b\}^*$

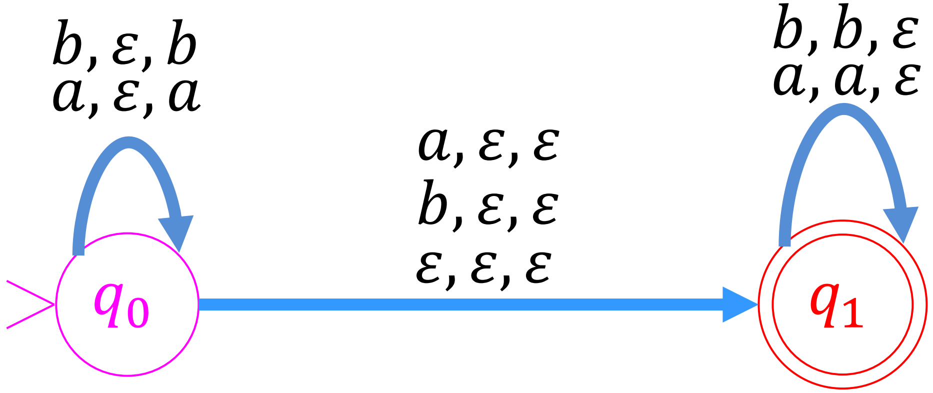
Strategy:

1. For each a I see, push a onto the stack
2. For each b I see, push b onto the stack
3. Guess that this is the middle
4. Guess that the string is even/odd length
5. For each a I see, pop a off of the stack
6. For each b I see, pop b off of the stack

PDA for $w = w^R$

Consume a/b from input
Don't pop anything
Push a/b respectively

Consume a/b from input
Pop a/b respectively
Don't push anything



Consume a/b or nothing from input
Don't pop anything
Don't push anything

(going from forward half to
backwards half of string)