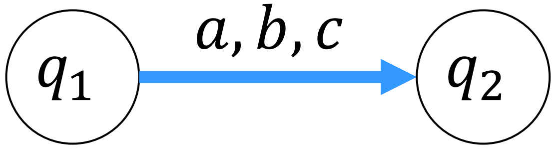
CS3102 Theory of Computation

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: $\delta : Q \times \underbrace{(\Sigma \cup \{\varepsilon\})}_{\text{Input}} \times \underbrace{\Gamma}_{\text{popped}} \to \underbrace{2^{Q \times \Gamma^*}}_{\text{New states and stack}}$
- Initial state: $q_0 \in Q$ q_0
- Final states: $F \subseteq Q$ q_k



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

PDA for "Palindromes"

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

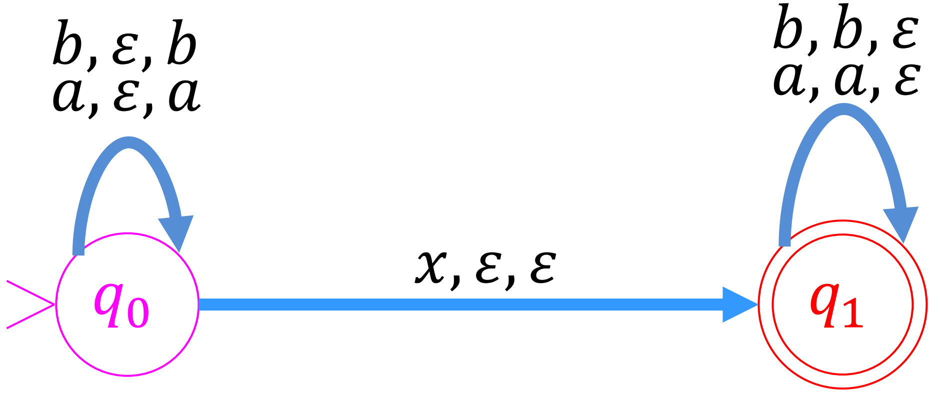
Strategy:

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 $w \in \{a, b\}^*$

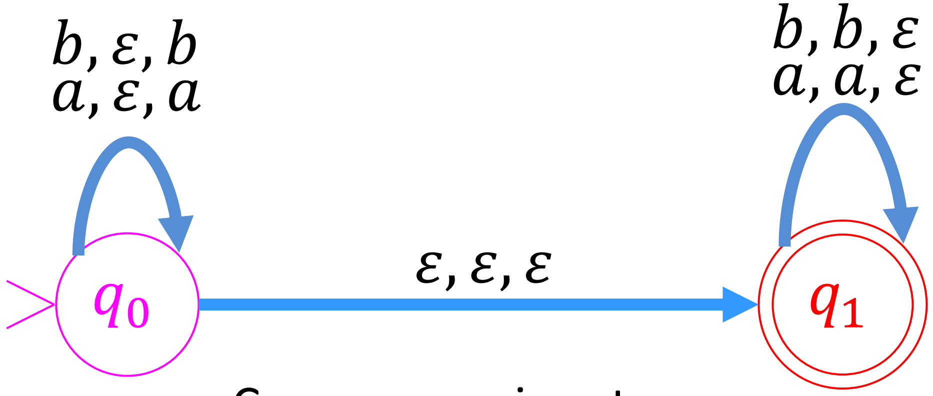
Strategy:

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 $w \in \{a, b\}^*$

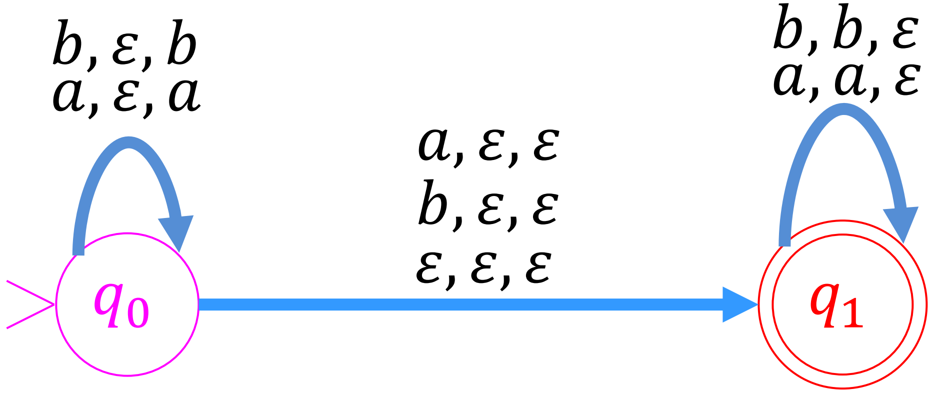
Strategy:

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)

Context Free Languages

- For a PDA M, the language of M (denoted $L(M)$) refers to the set of strings accepted by the machine
  - $L(M) = \{s \in \Sigma^* | M \text{ accepts } s\}$
- The set of all languages decided by some PDA is call the **Context Free Languages**
  - Equivalent to the languages describable by Context Free Grammars
- A particular language decided by some FSA is called a **Context Free Language**
- All regular languages are context free (because if we choose not to use the stack, a PDA is a NFA)
- All context free languages can be decided by a Java program using only linear memory (relative to length of word)

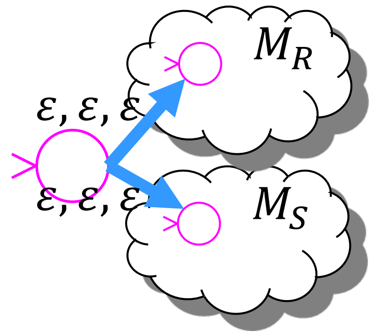
Non-Context-Free Languages

- Pumping Lemma for CFLs exists (but we won't cover it)
- $a^n b^n c^n$
  - Intuition: When deciding $a^n b^n$ we pushed for a's and popped for b's. Once we popped everything for b's, we "forgot" what n was.
- $a^n b^m c^n d^m$
  - Intuition: If I count the number of a's and b's using the stack, the b's are "blocking" the a's count from being checked against c's.
- $\{ww \mid w \in \Sigma^*\}$
  - Intuition: we could push the first half onto the stack, but when popping off it's in reverse order.

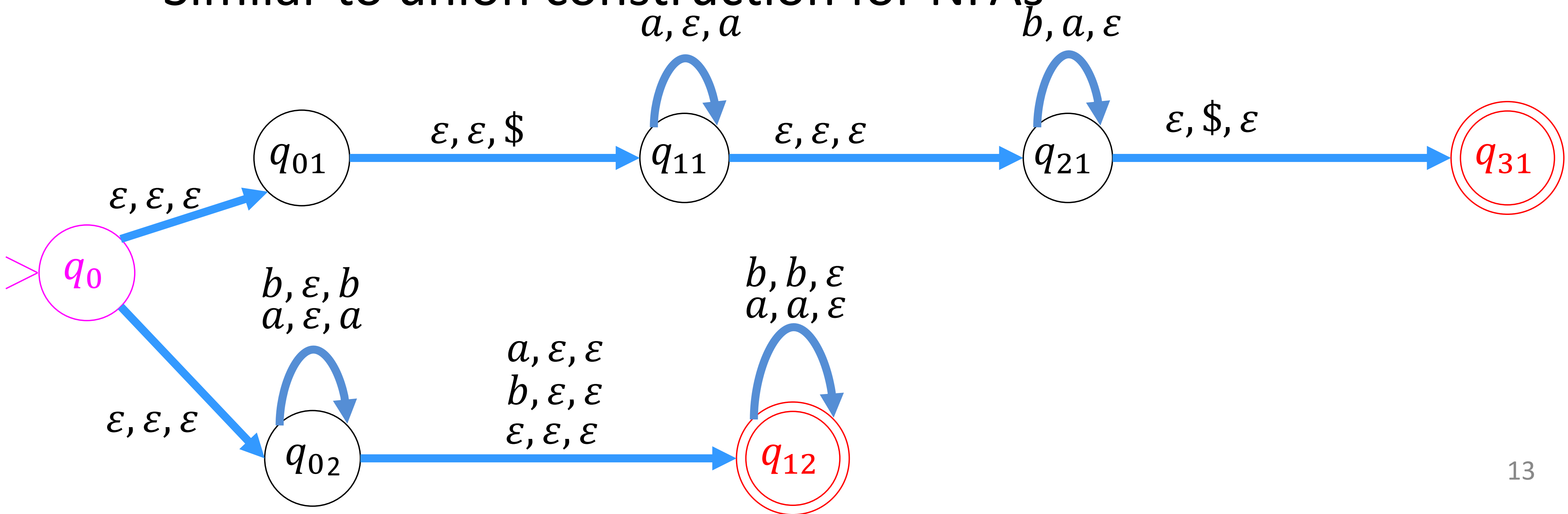
Closure Properties of CFLs

- Context Free Languages are closed under:
  - Union
  - Concatenation
  - Kleene Star
  - Intersection with Regular languages
- Context Free Languages are not closed under:
  - Complementation
  - Intersection with CFLs

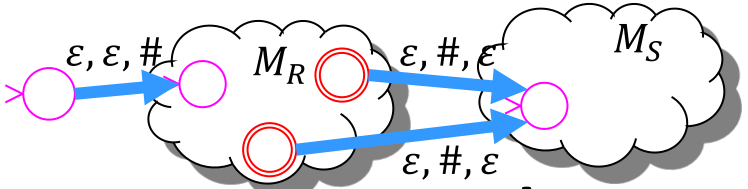
CFLs closed under union



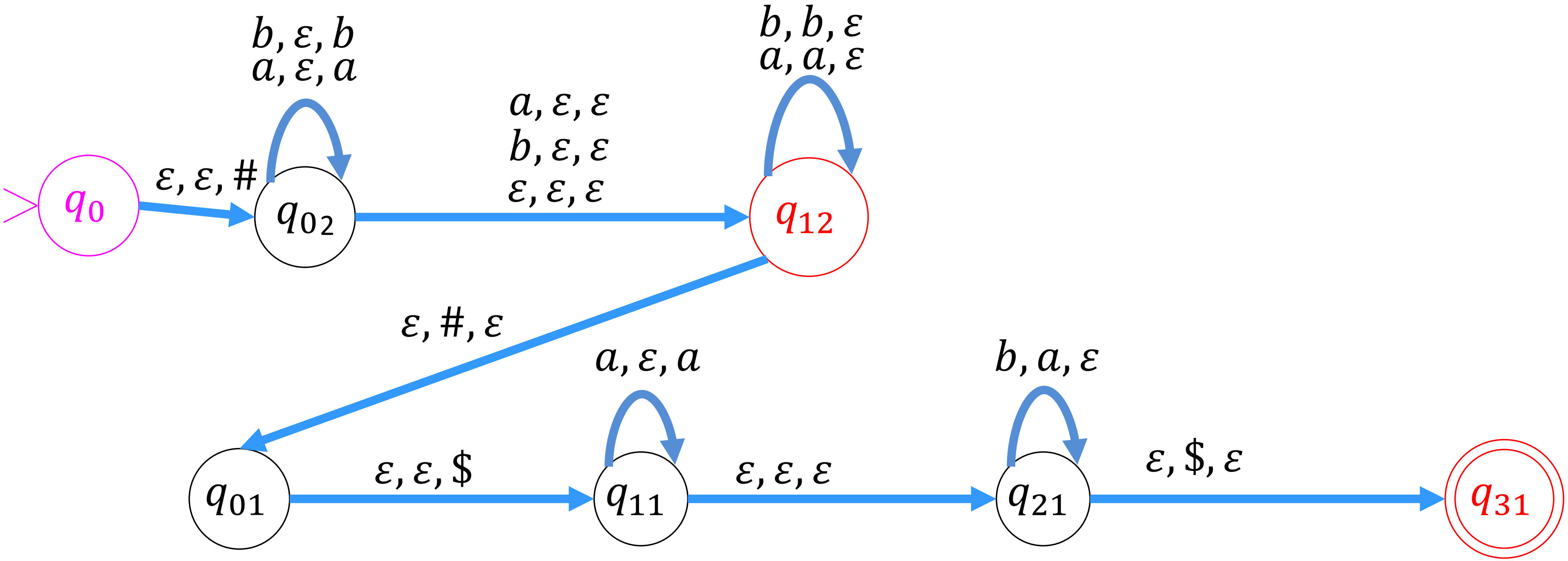
- Add in a new start state, epsilon transition to start states of source machines
  - Similar to union construction for NFAs



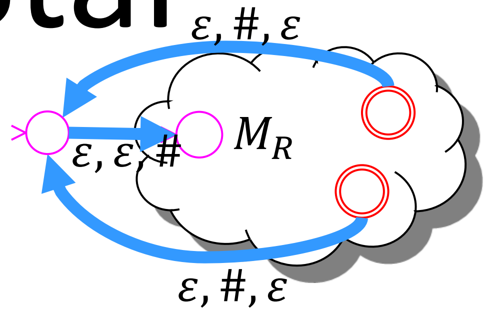
CFLs closed under Concatenation



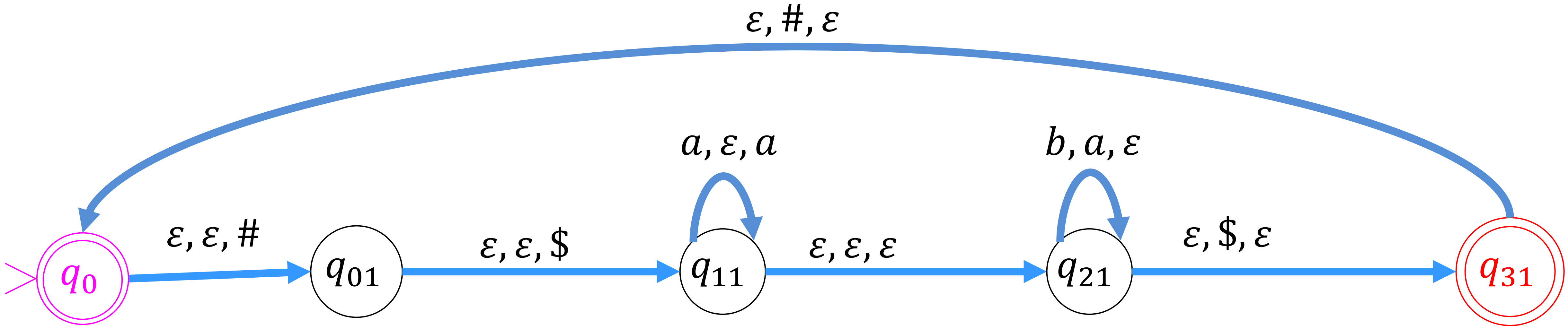
- When in an accept state with an empty stack in machine 1, transition into machine 2



CFLs closed under Kleene Star

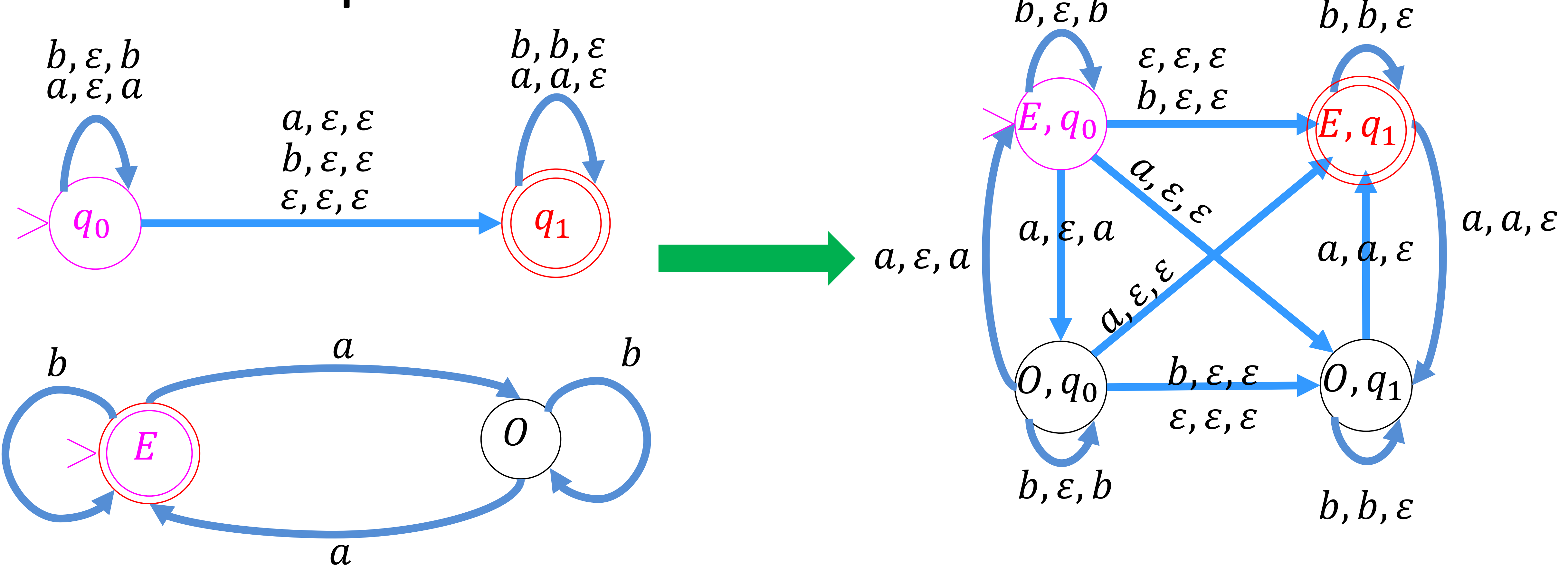


- When in an accept state with an empty stack, return to start state



CFLs closed under Intersection with Regular Languages

- Cross product construction



CFLs not closed under Intersection

- $a^n b^n c^k \cap a^j b^n c^n = a^n b^n c^n$
- Intersecting:
  - Some number of a’s, same number of b’s, any number of c’s (a’s match b’s)
  - Any number of a’s, some number of b’s, some number of c’s (b’s match c’s)
- Results in:
  - Some number of a’s, same number of b’s, same number of c’s (a’s match b’s match c’s)

CFLs not closed under Complement