

CS103  
FALL 2025



# Lecture 15: **Finite Automata**

**Part 2 of 3**

# Outline for Today

- ***Recap from Last Time***
  - Where are we, again?
- ***Regular Languages***
  - A fundamental class of languages.
- ***NFAs***
  - Automata with Magic Superpowers.
- ***Designing NFAs***
  - Harnessing an awesome power.

Recap from Last Time

Suppose you know the following:

$$x \in \Sigma \quad y \in \Sigma^*$$

Which of the following options is correct?

- (A)  $x$  is a character and  $y$  is a character.
- (B)  $x$  is a character and  $y$  is a string.
- (C)  $x$  is a string and  $y$  is a character.
- (D)  $x$  is a string and  $y$  is a string.
- (E) None of these

Answer at <https://cs103.stanford.edu/pollev>

# Formal Language Theory

- An **alphabet** is a set, usually denoted  $\Sigma$ , consisting of elements called **characters**.
  - $a \in \Sigma$  means “ $a$  is a single character.”
- A **string over  $\Sigma$**  is a finite sequence of zero or more characters taken from  $\Sigma$ .
- The **empty string** has no characters and is denoted  $\varepsilon$ .
- A **language over  $\Sigma$**  is a set of strings over  $\Sigma$ .
- The language  $\Sigma^*$  is the set of all strings over  $\Sigma$ .
  - $w \in \Sigma^*$  means “ $w$  is a string of characters from  $\Sigma$ .”

# The Language of an Automaton

- If  $A$  is an automaton that processes strings over  $\Sigma$ , the *language of  $A$* , denoted  $\mathcal{L}(A)$ , is the set of all strings  $A$  accepts.
- Formally:

$$\mathcal{L}(A) = \{ w \in \Sigma^* \mid A \text{ accepts } w \}$$

# DFA<sub>s</sub>

- A **DFA** is a
  - **D**eterministic
  - **F**inite
  - **A**utomaton
- A DFA is defined relative to some alphabet  $\Sigma$ .
- For each state in the DFA, there must be **exactly one** transition defined for each symbol in  $\Sigma$ .
  - This is the “deterministic” part of DFA.
- There is a unique start state.
- There are zero or more accepting states.

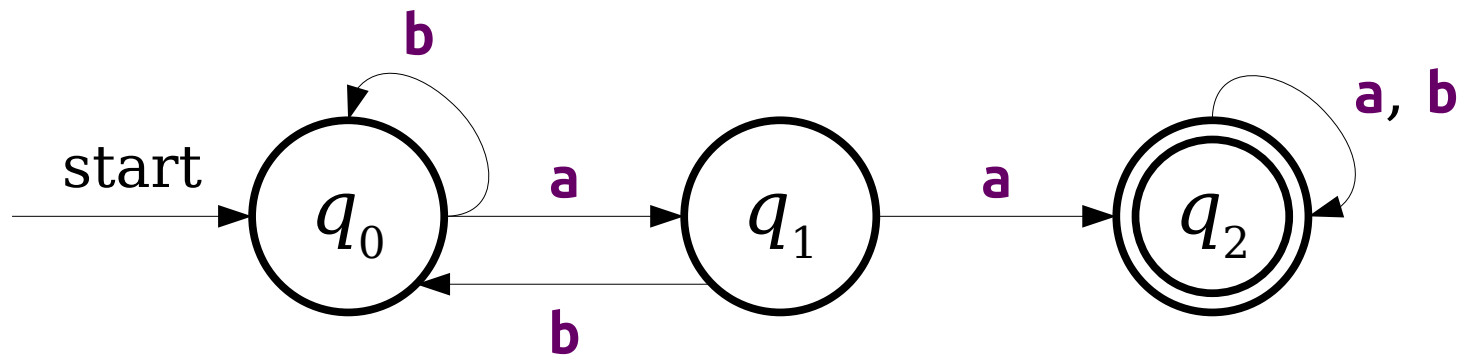
# Recognizing Languages with DFAs

$L = \{ w \in \{\mathbf{a}, \mathbf{b}\}^* \mid w \text{ contains } \mathbf{aa} \text{ as a substring} \}$



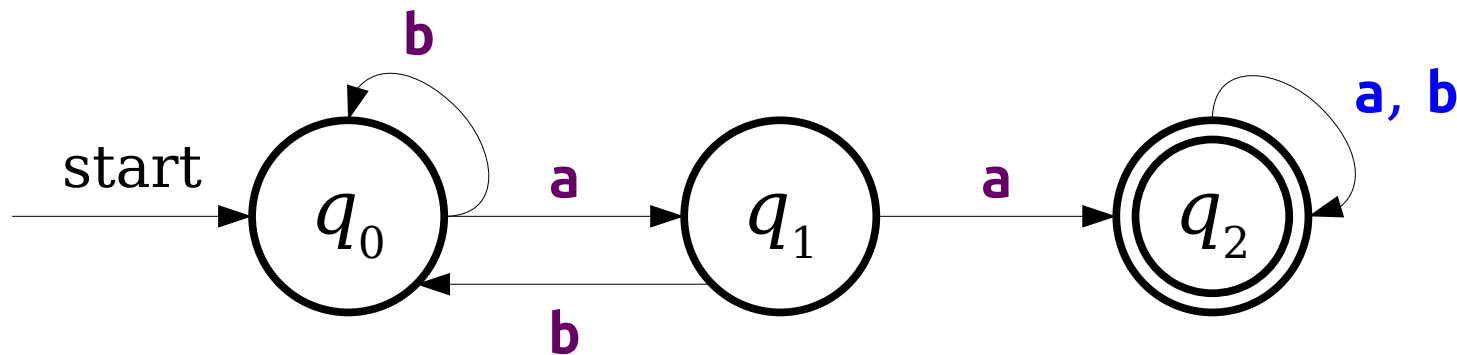
# Recognizing Languages with DFAs

$L = \{ w \in \{\mathbf{a}, \mathbf{b}\}^* \mid w \text{ contains } \mathbf{aa} \text{ as a substring} \}$



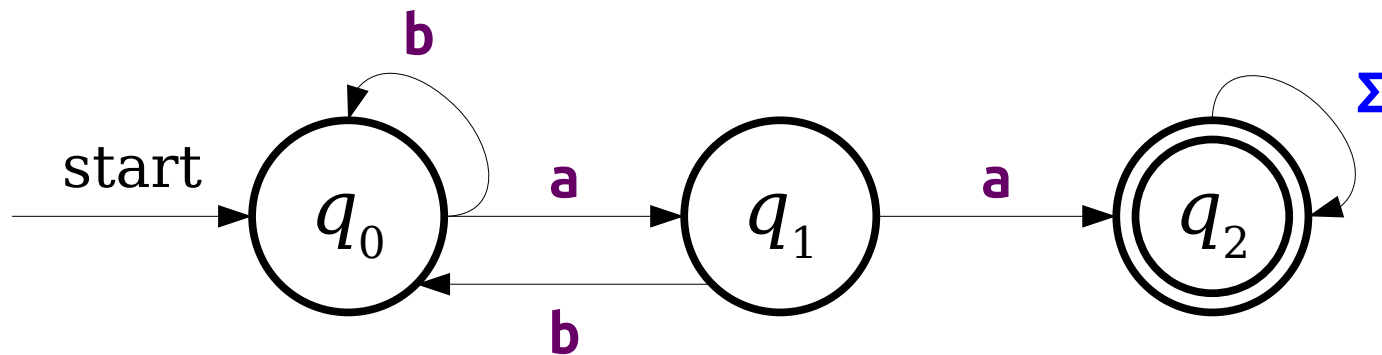
# Recognizing Languages with DFAs

$L = \{ w \in \{\mathbf{a}, \mathbf{b}\}^* \mid w \text{ contains } \mathbf{aa} \text{ as a substring} \}$



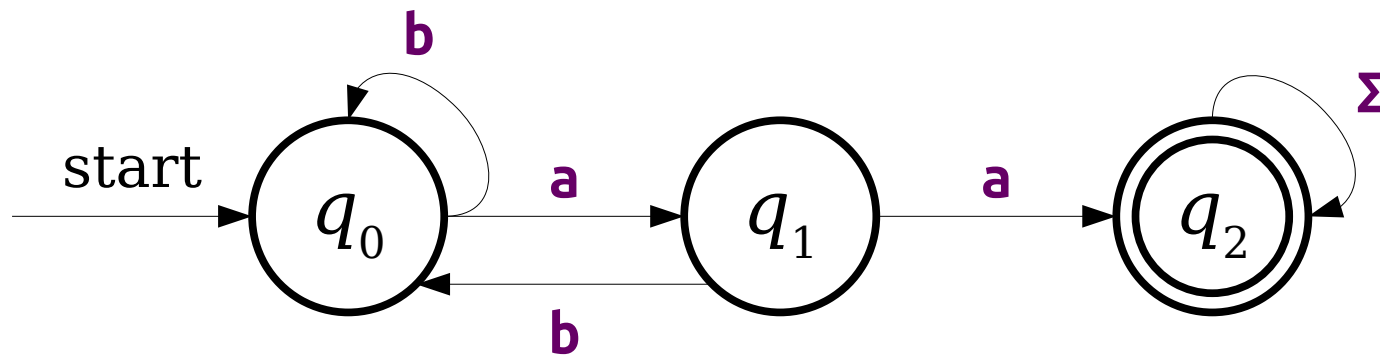
# Recognizing Languages with DFAs

$L = \{ w \in \{\mathbf{a}, \mathbf{b}\}^* \mid w \text{ contains } \mathbf{aa} \text{ as a substring} \}$



# Recognizing Languages with DFAs

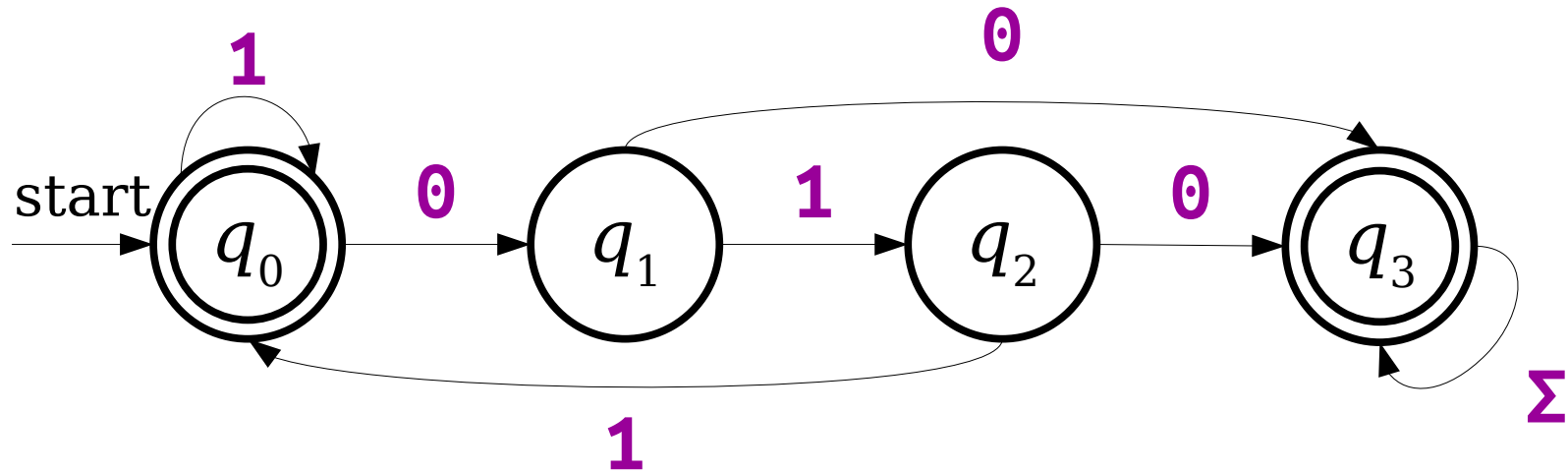
$L = \{ w \in \{\mathbf{a}, \mathbf{b}\}^* \mid w \text{ contains } \mathbf{aa} \text{ as a substring} \}$



New Stuff!

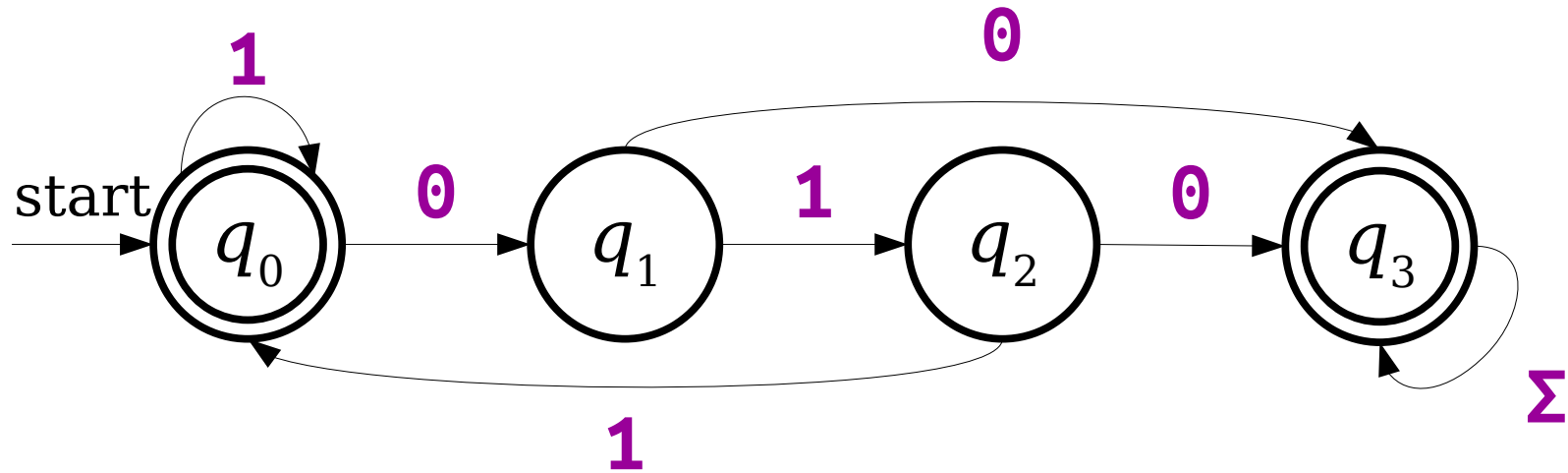
# Tabular DFAs

# Tabular DFAs



	0	1
$q_0$		
$q_1$		
$q_2$		
$q_3$		

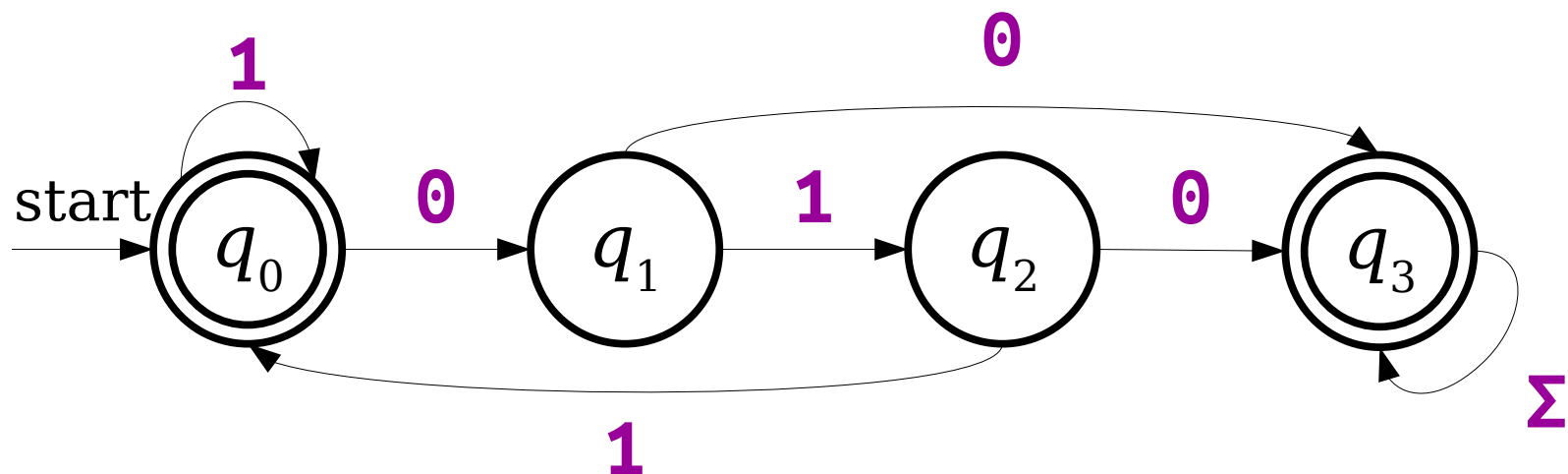
# Tabular DFAs



	0	1
$q_0$	$q_1$	$q_0$
$q_1$	$q_3$	$q_2$
$q_2$	$q_3$	$q_0$
$q_3$	$q_3$	$q_3$

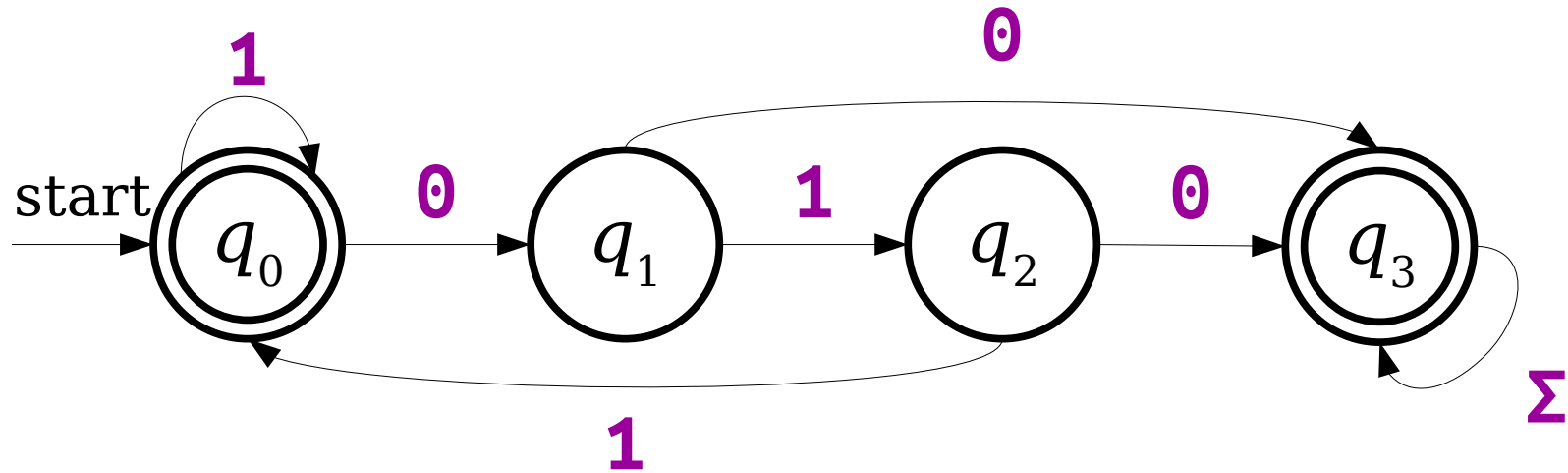


# Tabular DFAs



	0	1
* $q_0$	$q_1$	$q_0$
$q_1$	$q_3$	$q_2$
$q_2$	$q_3$	$q_0$
* $q_3$	$q_3$	$q_3$

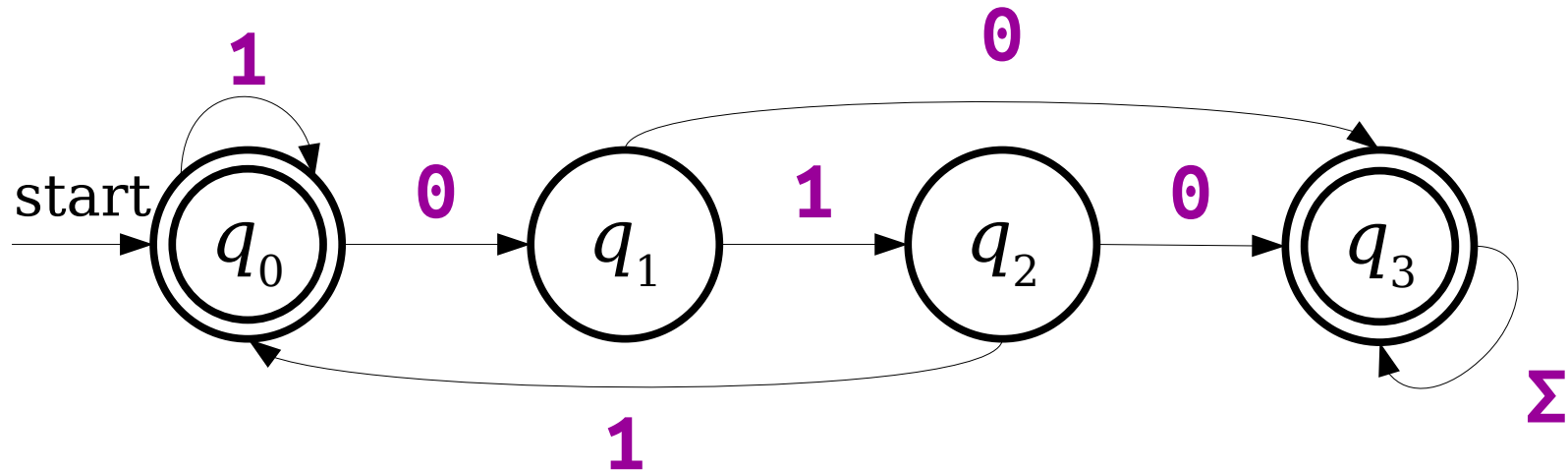
# Tabular DFAs



	0	1
* $q_0$	$q_1$	$q_0$
$q_1$	$q_3$	$q_2$
$q_2$	$q_3$	$q_0$
* $q_3$	$q_3$	$q_3$

These stars indicate accepting states.

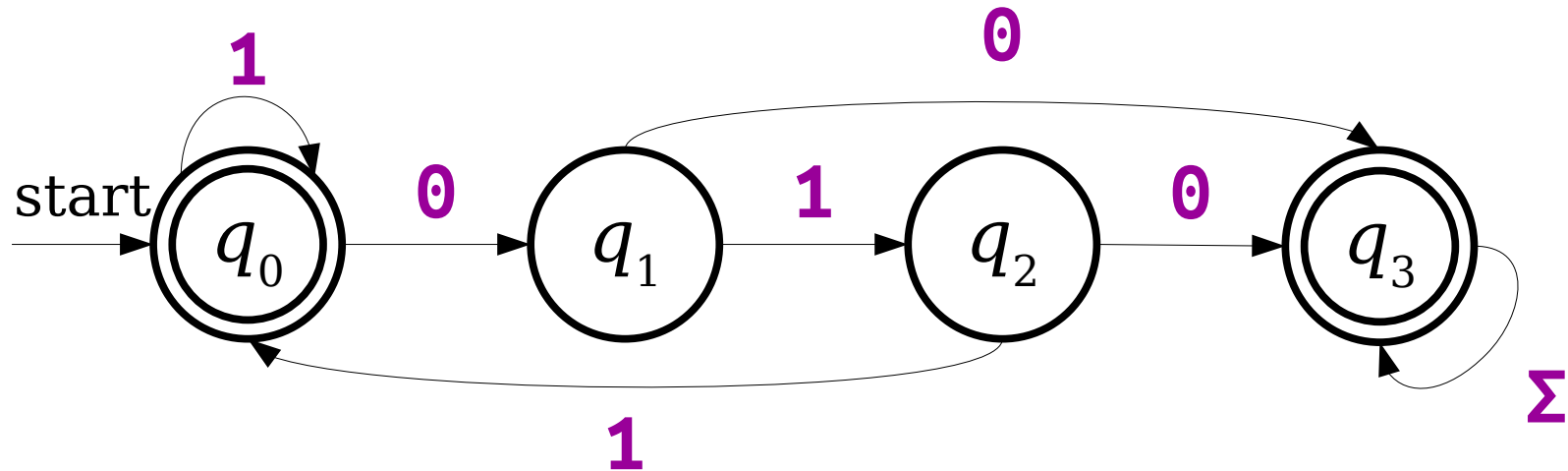
# Tabular DFAs



Since this is the first row, it's the start state.

	0	1
* $q_0$	$q_1$	$q_0$
$q_1$	$q_3$	$q_2$
$q_2$	$q_3$	$q_0$
* $q_3$	$q_3$	$q_3$

# Tabular DFAs



	0	1
* $q_0$	$q_1$	$q_0$
$q_1$	$q_3$	$q_2$
$q_2$	$q_3$	$q_0$
* $q_3$	$q_3$	$q_3$

Why isn't there a column here for  $\Sigma$ ?

Answer at  
<https://cs103.stanford.edu/pollev>

# Simulating a DFA

```
int kTransitionTable[kNumStates][kNumSymbols] = {
    {0, 0, 1, 3, 7, 1, ...},
    ...
};

bool kAcceptTable[kNumStates] = {
    false,
    true,
    true,
    ...
};

bool accepts(string input) {
    int state = 0;
    for (char ch: input) {
        state = kTransitionTable[state][ch];
    }
    return kAcceptTable[state];
}
```

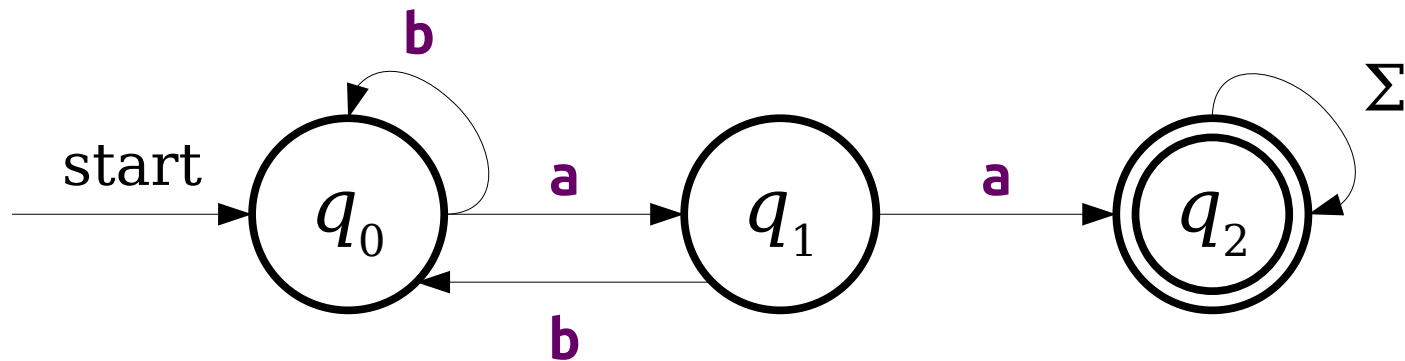
# The Regular Languages

A language  $L$  is a ***regular language*** when there exists a DFA  $D$  such that  $\mathcal{L}(D) = L$ .

If  $L$  is a language and  $\mathcal{L}(D) = L$ , we say that  $D$  ***recognizes*** the language  $L$ .

# Complementing Regular Languages

$$L = \{ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring} \}$$



$$\bar{L} = \{ w \in \{a, b\}^* \mid w \text{ **does not** contain } aa \text{ as a substring} \}$$

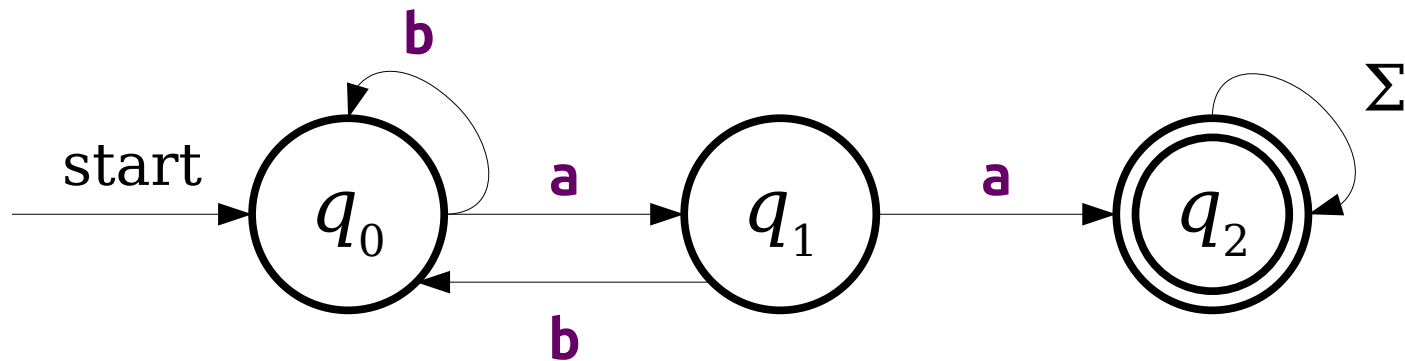
How do we turn the DFA above  
into a DFA for  $\bar{L}$ ?

Answer at  
<https://cs103.stanford.edu/pollev>

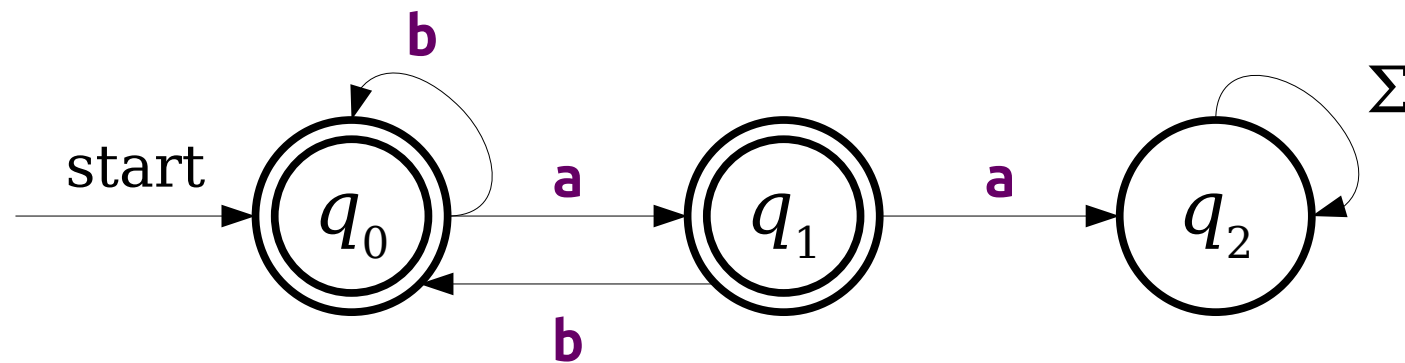


# Complementing Regular Languages

$$L = \{ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring} \}$$

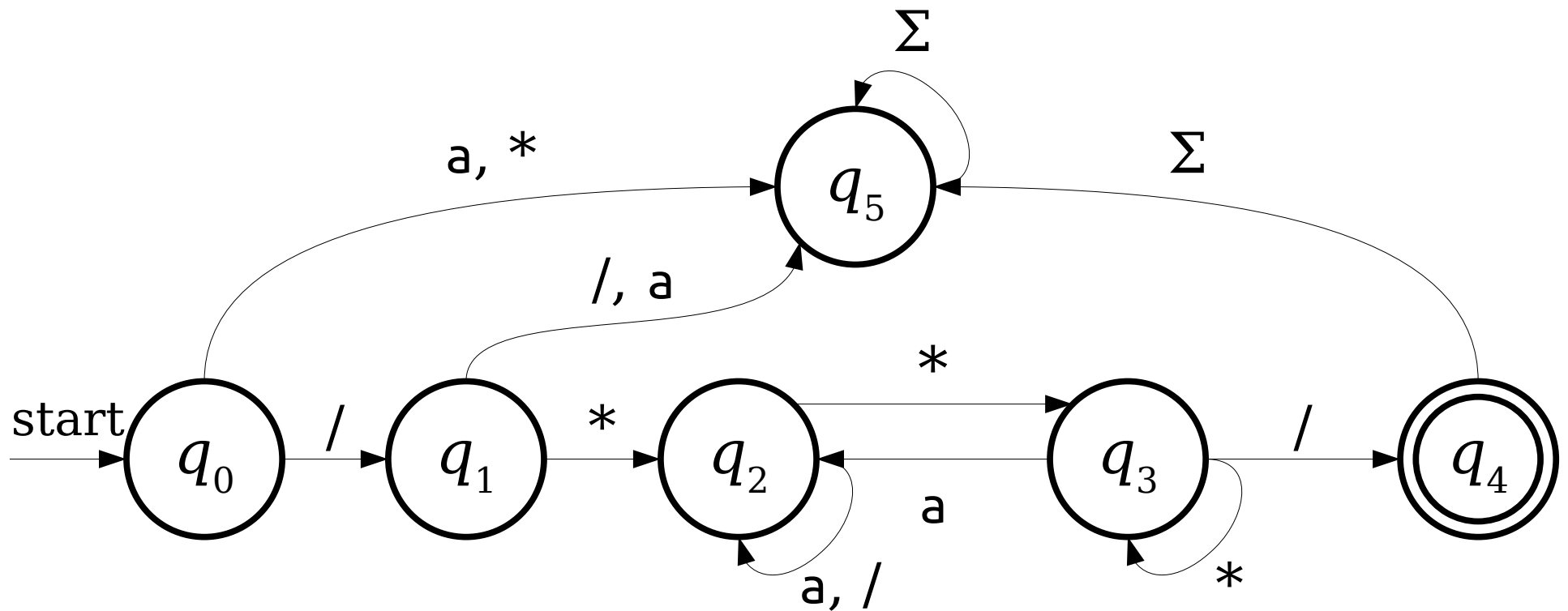


$$\bar{L} = \{ w \in \{a, b\}^* \mid w \text{ **does not** contain } aa \text{ as a substring} \}$$



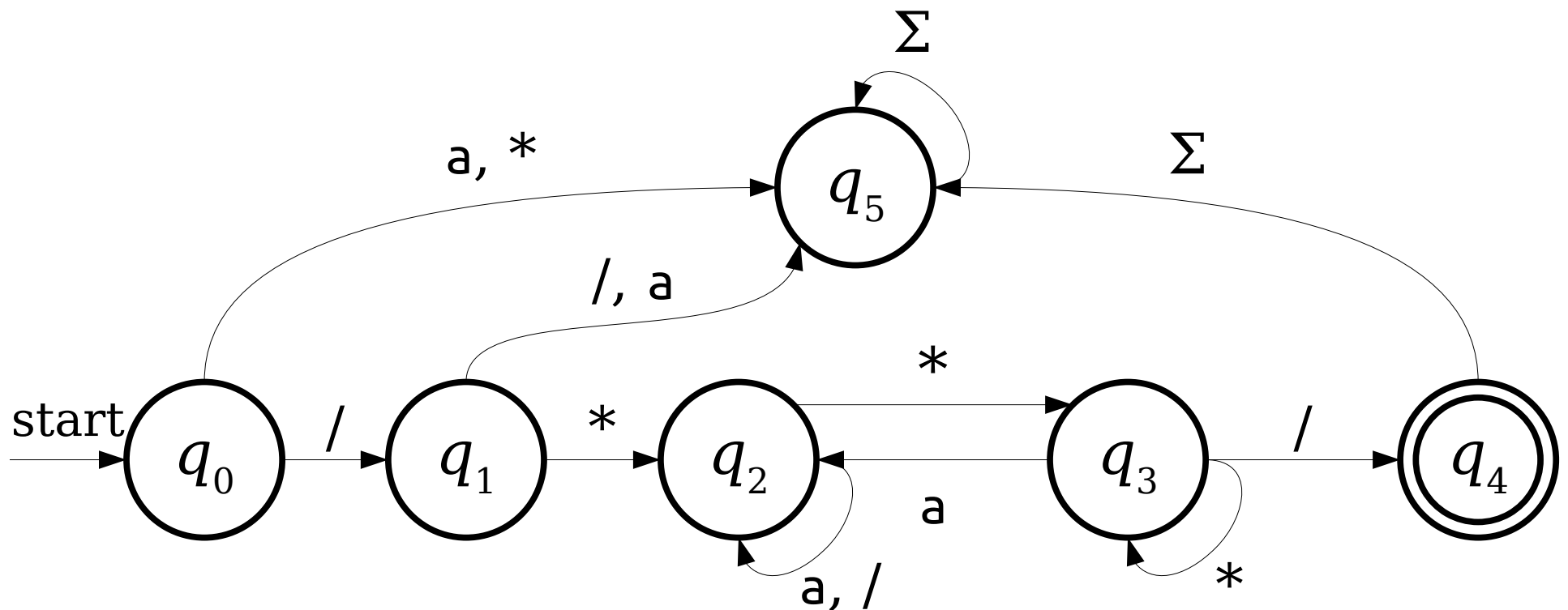
# Complementing Regular Languages

$L = \{ w \in \{\mathbf{a}, \mathbf{*}, \mathbf{/}\}^* \mid w \text{ represents a C-style comment} \}$



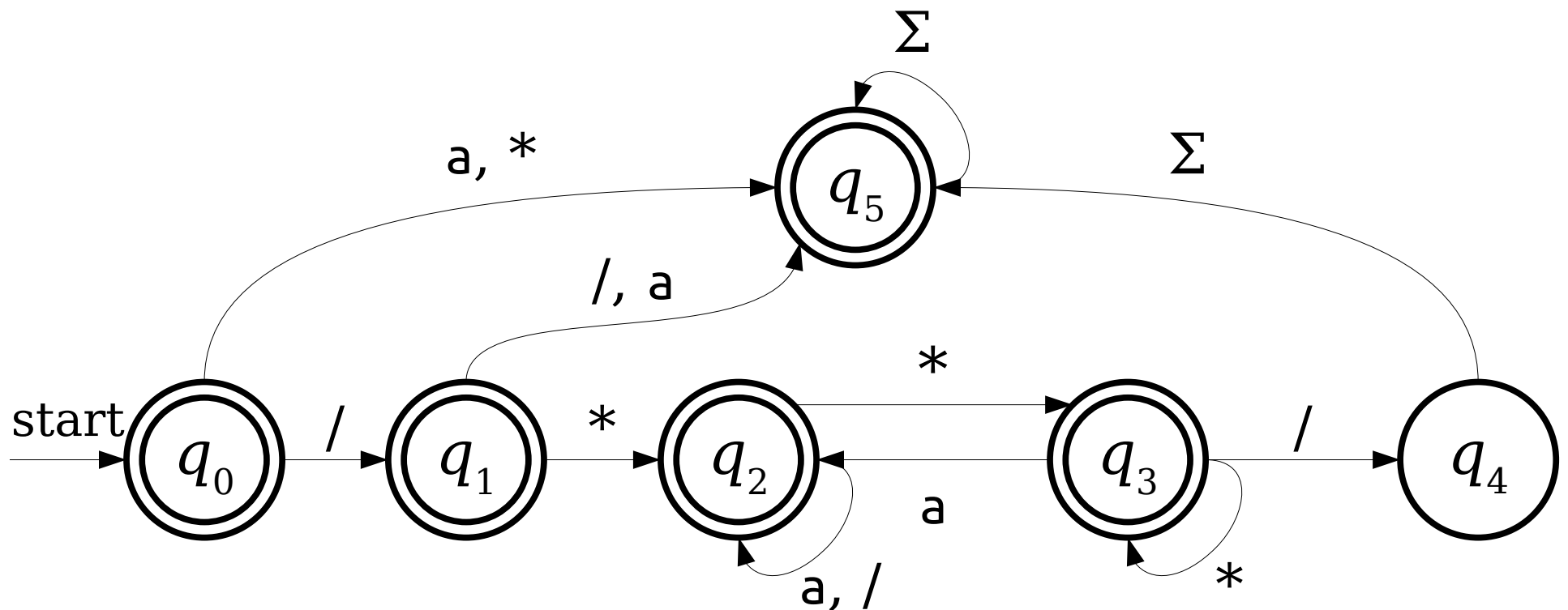
# Complementing Regular Languages

$\bar{L} = \{ w \in \{\mathbf{a}, *, /\}^* \mid w \text{ *doesn't* represent a C-style comment} \}$



# Complementing Regular Languages

$\bar{L} = \{ w \in \{\text{a}, *, /\}^* \mid w \text{ *doesn't* represent a C-style comment} \}$



# The Complement of a Language

- Given a language  $L \subseteq \Sigma^*$ , the **complement** of that language (denoted  $\bar{L}$ ) is the language of all strings in  $\Sigma^*$  that aren't in  $L$ .
- Formally:

$$\bar{L} = \Sigma^* - L$$

# The Complement of a Language

- Given a language  $L \subseteq \Sigma^*$ , the **complement** of that language (denoted  $\bar{L}$ ) is the language of all strings in  $\Sigma^*$  that aren't in  $L$ .
- Formally:

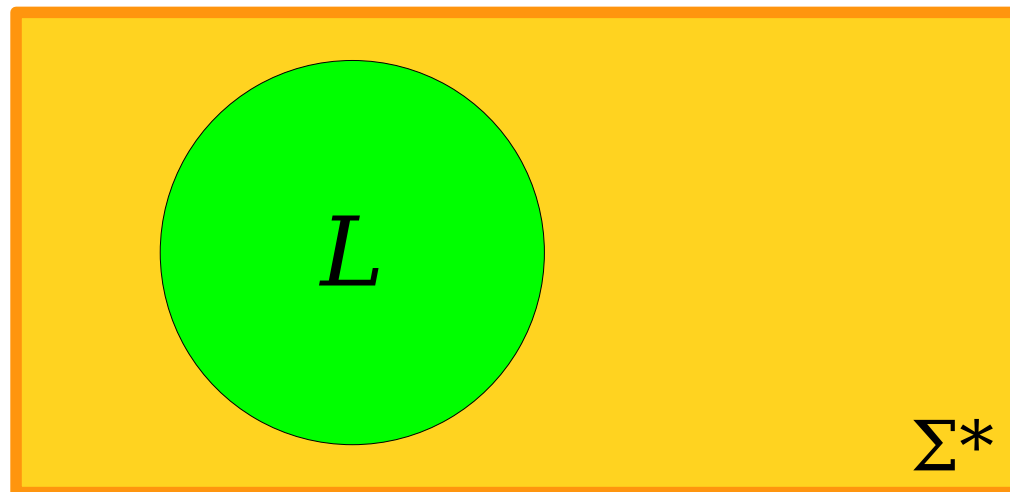
$$\bar{L} = \Sigma^* - L$$



# The Complement of a Language

- Given a language  $L \subseteq \Sigma^*$ , the **complement** of that language (denoted  $\bar{L}$ ) is the language of all strings in  $\Sigma^*$  that aren't in  $L$ .
- Formally:

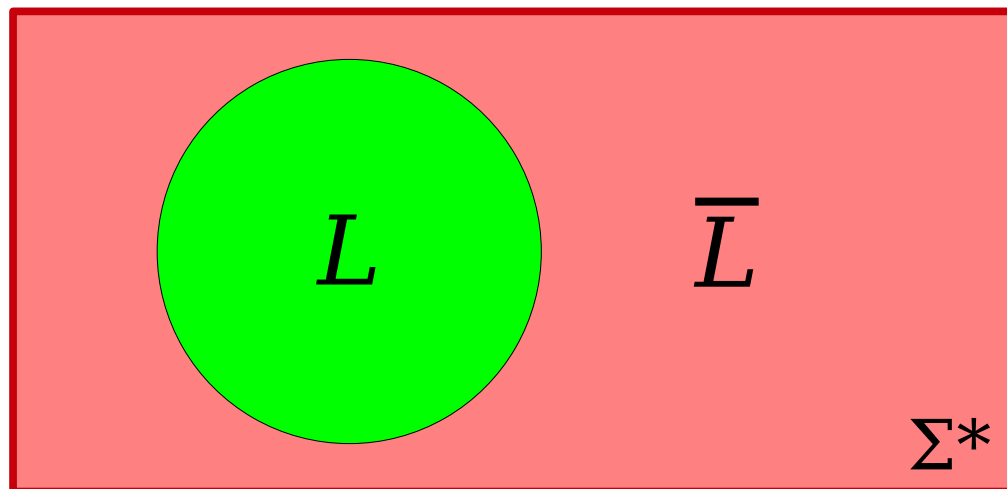
$$\bar{L} = \Sigma^* - L$$



# The Complement of a Language

- Given a language  $L \subseteq \Sigma^*$ , the **complement** of that language (denoted  $\bar{L}$ ) is the language of all strings in  $\Sigma^*$  that aren't in  $L$ .
- Formally:

$$\bar{L} = \Sigma^* - L$$

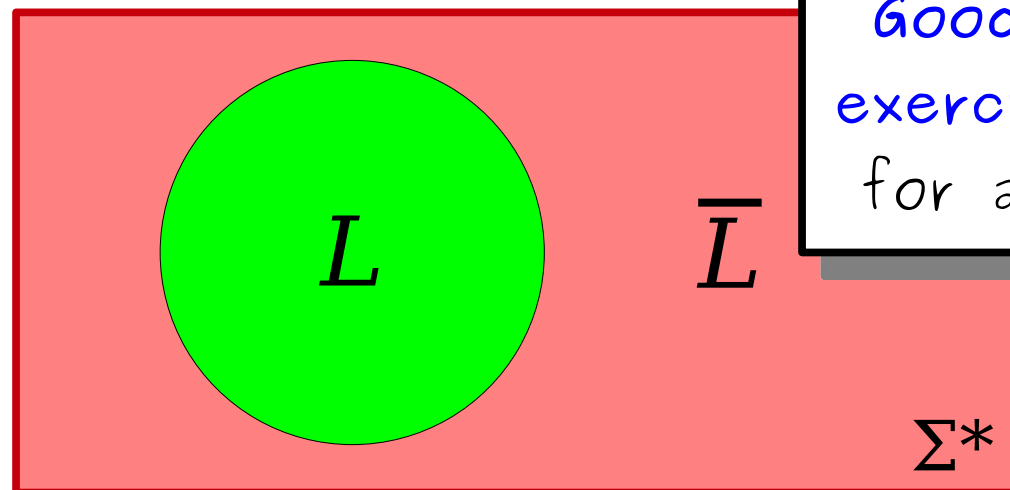




# The Complement of a Language

- Given a language  $L \subseteq \Sigma^*$ , the **complement** of that language (denoted  $\bar{L}$ ) is the language of all strings in  $\Sigma^*$  that aren't in  $L$ .
- Formally:

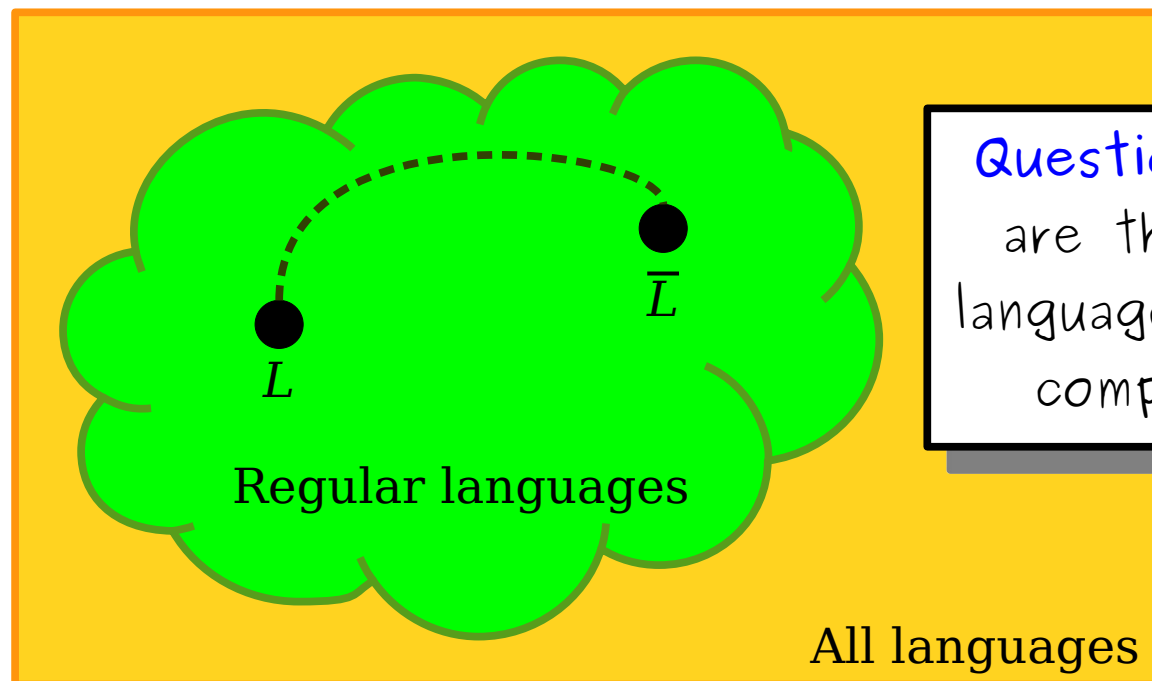
$$\bar{L} = \Sigma^* - L$$



Good proofwriting  
exercise: prove  $\bar{\bar{L}} = L$   
for any language  $L$ .

# Closure Properties

- **Theorem:** If  $L$  is a regular language, then  $\bar{L}$  is also a regular language.
  - (“The regular languages are **closed under complementation**.”)
- **Proof idea:** Show that swapping the accepting and rejecting states of a DFA for  $L$  gives a DFA for  $\bar{L}$ .



Question to ponder:  
are the *nonregular*  
languages closed under  
complementation?

Time-Out for Announcements!

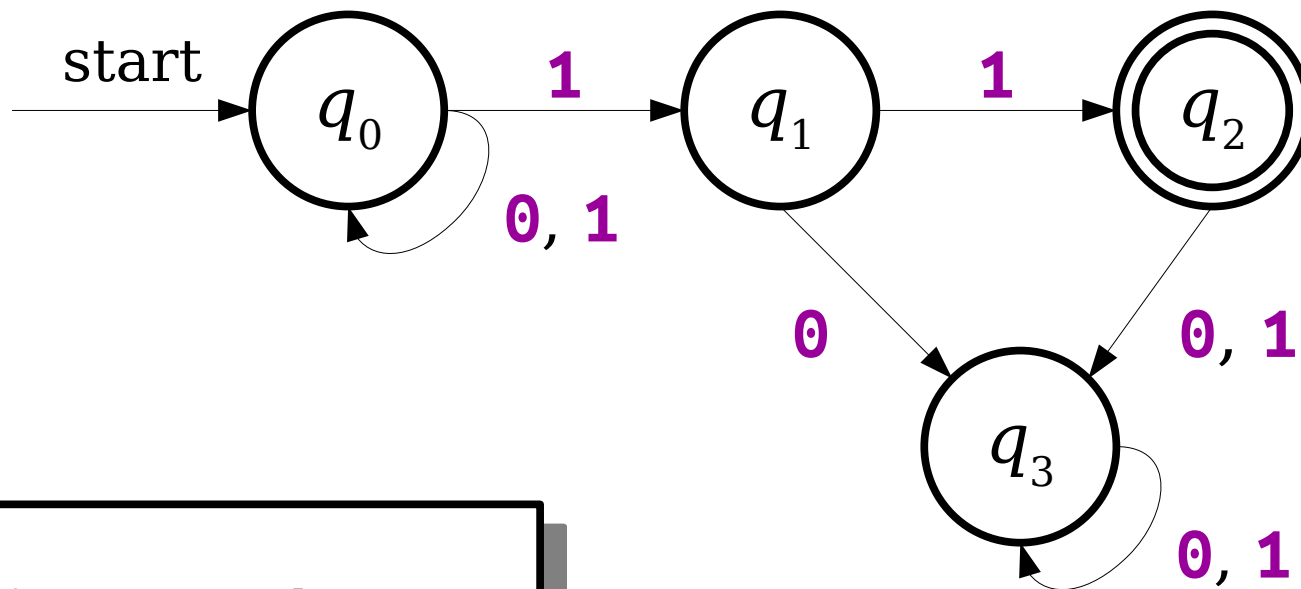
# Midterm Exam Graded

- Want to talk to us about the exam and next steps?
  - Stop by our office hours!
  - Chat with us after class!
  - Schedule a 1-on-1 with Kaia!
- Regrade requests for the exam are open and close Monday at 6:00PM.
  - 👉 ***Read the instructions before submitting regrade requests.*** 👉
  - Regrade requests are intended as a way to correct grading errors. We're happy to correct mistakes we've made.
  - If you're looking to talk to someone about the exam, or discuss plans for the tail end of the quarter, or just get a pep talk, that's best handled outside the regrade request system.
  - To make sure regrade requests are used as intended, we're strict about their structure / formatting.

Back to CS103!

# Beyond DFAs

# The Motivation



**Question:** How do we interpret an automaton like this one?

# NFAs

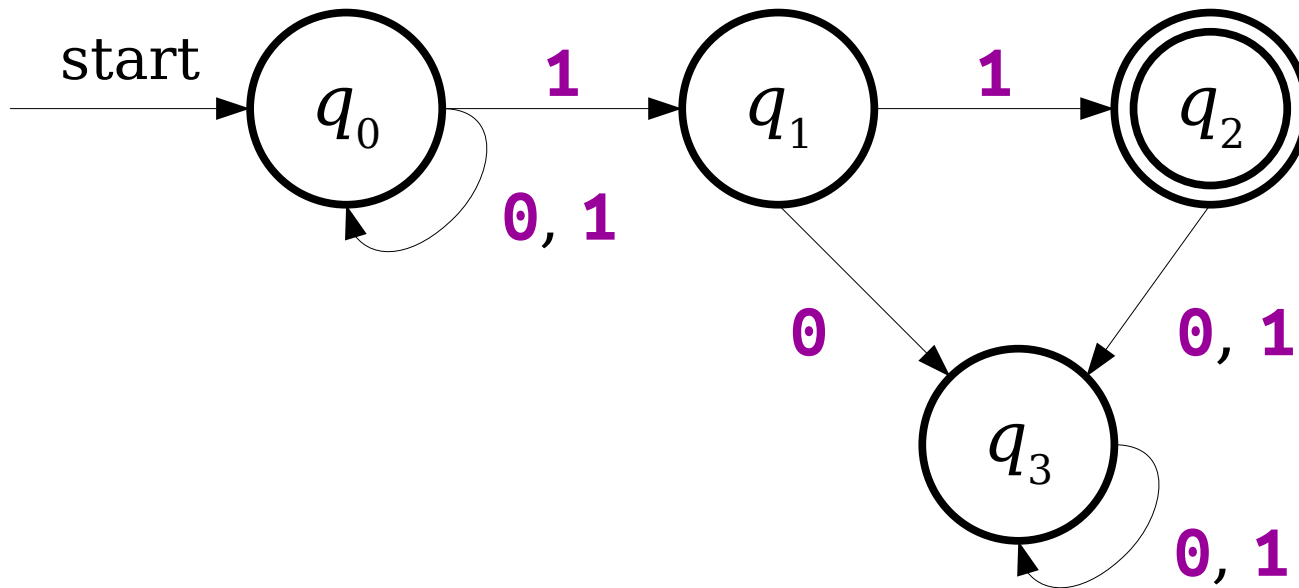
- An *NFA* is a
  - *N*ondeterministic
  - *F*inite
  - *A*utomaton
- NFAs are structurally similar to a DFA, but represents a fundamental shift in how we'll think about com



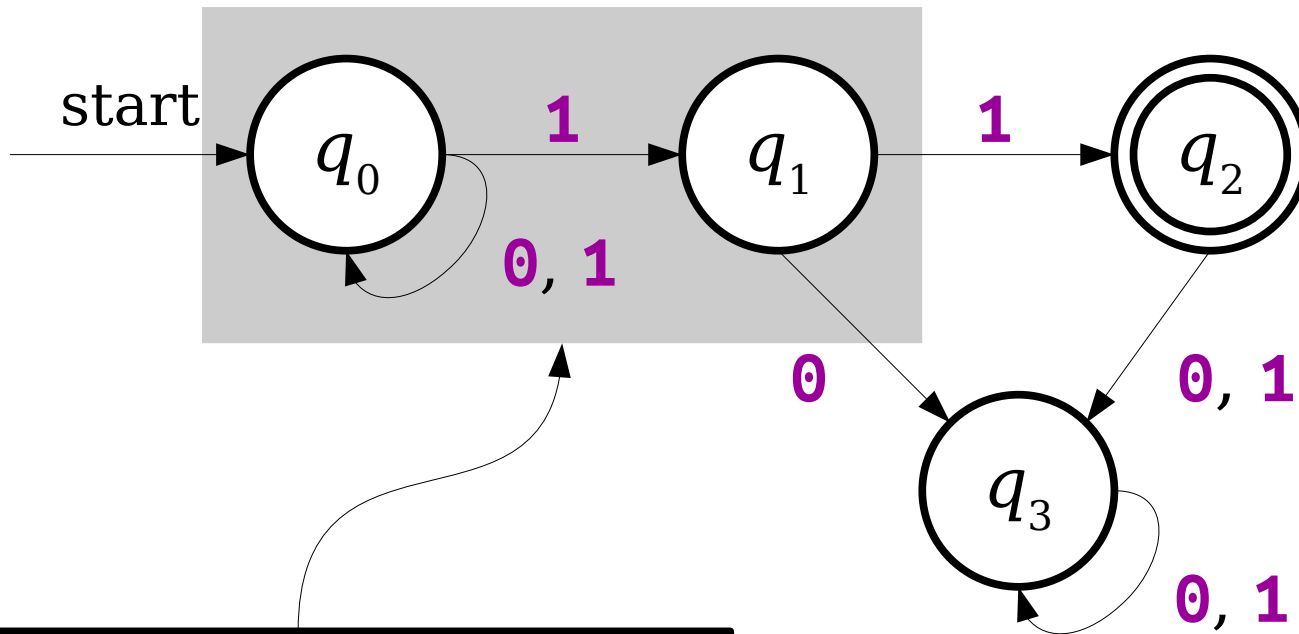
# (Non)determinism

- A model of computation is **deterministic** if at every point in the computation, there is exactly one choice that can make.
  - The machine accepts if that series of choices leads to an accepting state.
- A model of computation is **nondeterministic** if the computing machine has a finite number of choices available to make at each point, possibly including zero.
- The machine accepts if **any** series of choices leads to an accepting state.
  - (This sort of nondeterminism is technically called **existential nondeterminism**, the most philosophical-sounding term we'll introduce all quarter.)
- This idea was introduced by Michael Rabin and Dana Scott as an internship project (!) at IBM in 1957. It won them the Turing Award (the “Nobel Prize of Computer Science”) in 1976.

# A Simple NFA

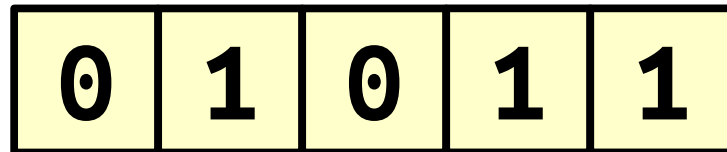
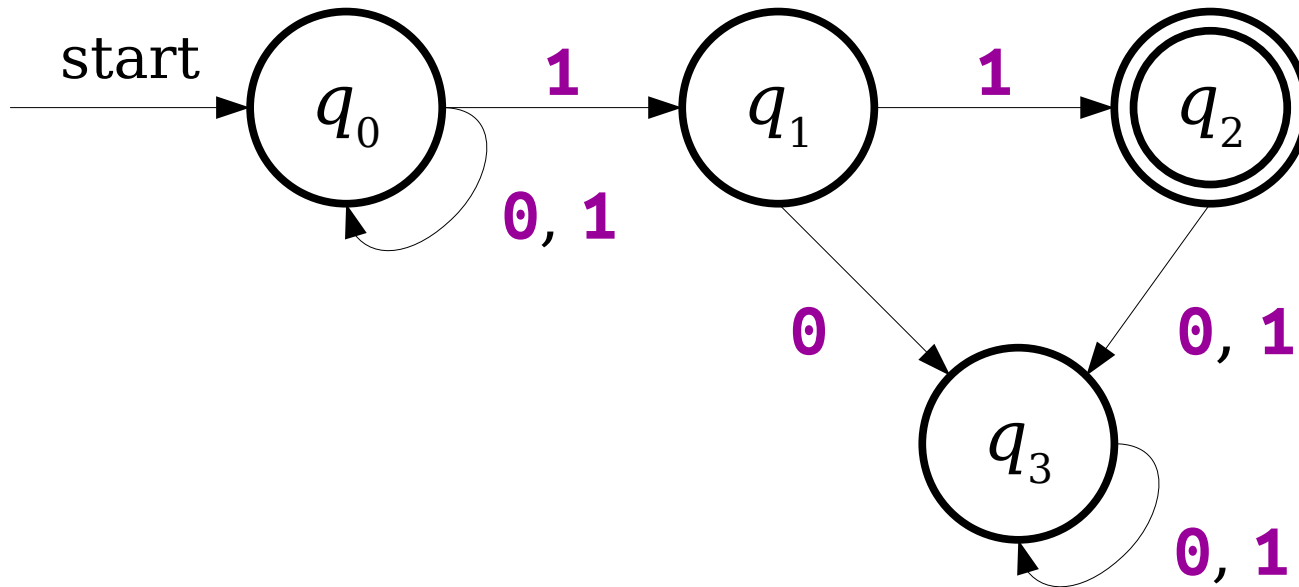


# A Simple NFA

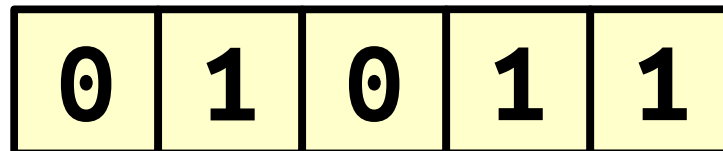
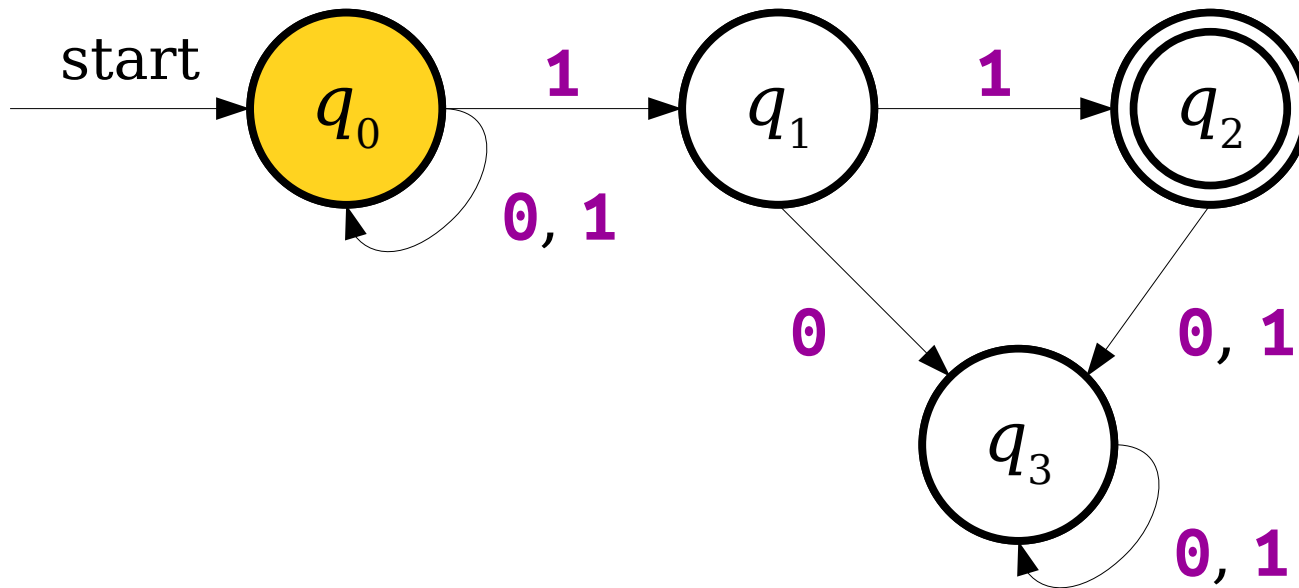


$q_0$  has two transitions defined on 1!

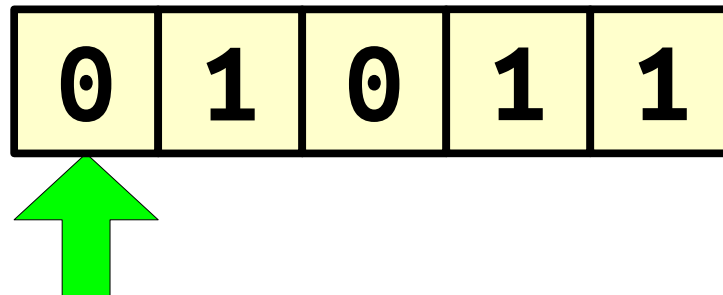
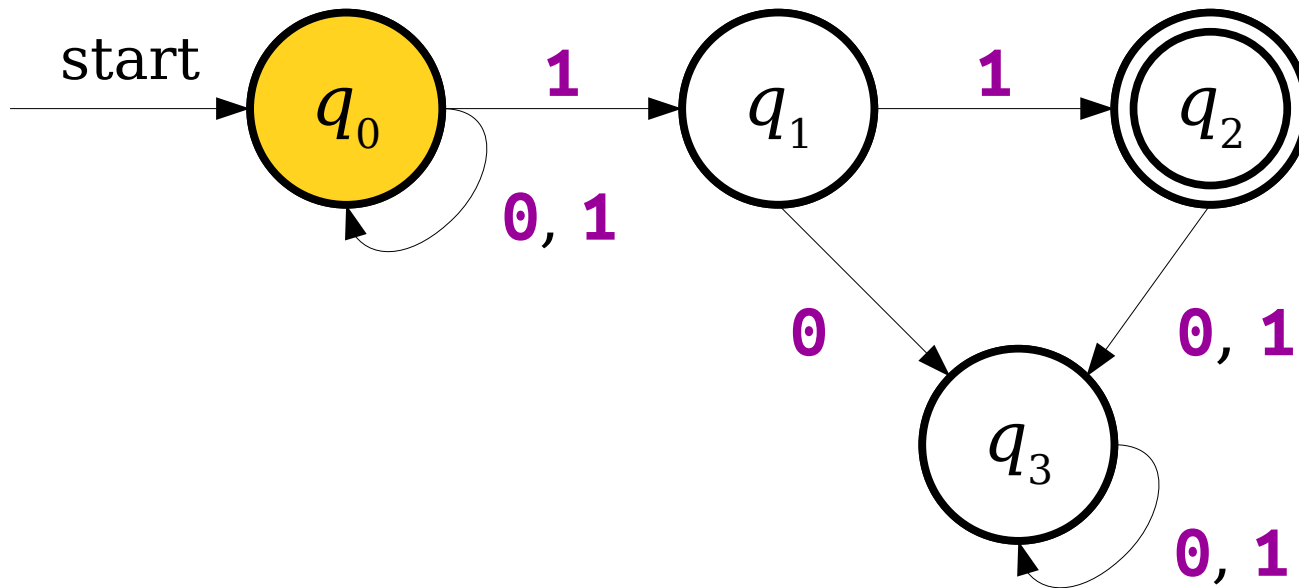
# A Simple NFA



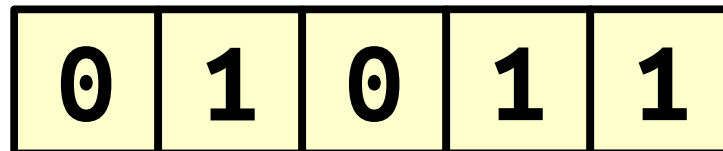
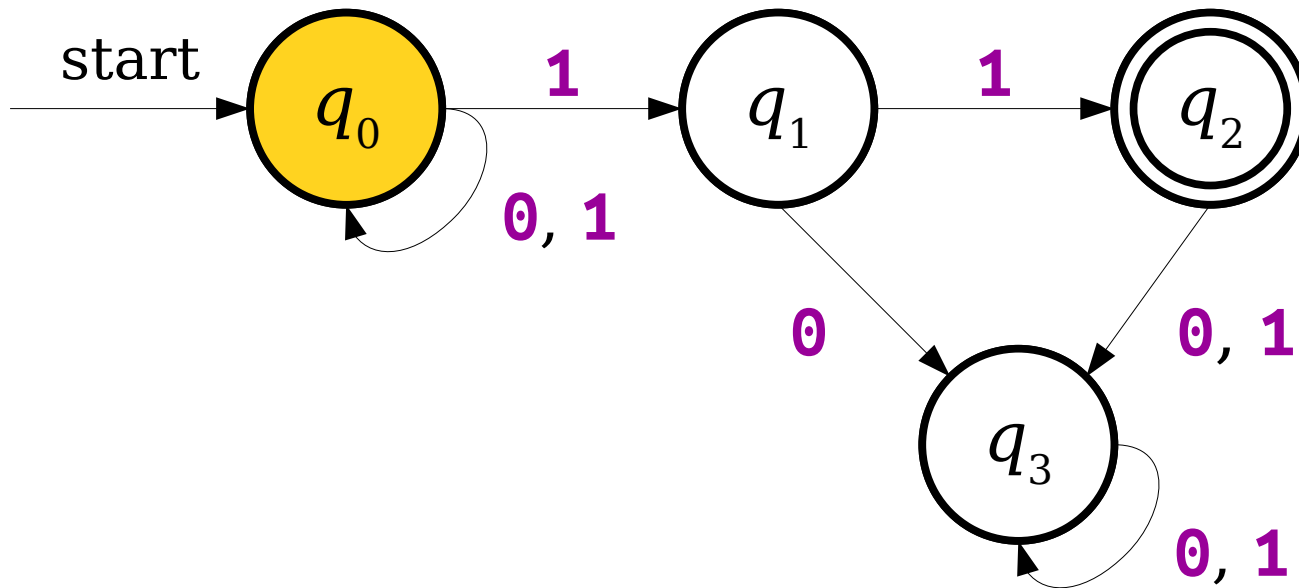
# A Simple NFA



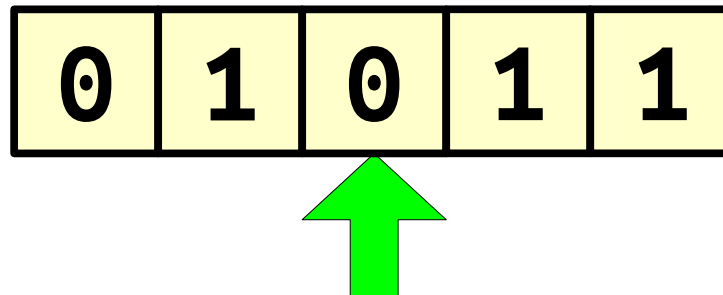
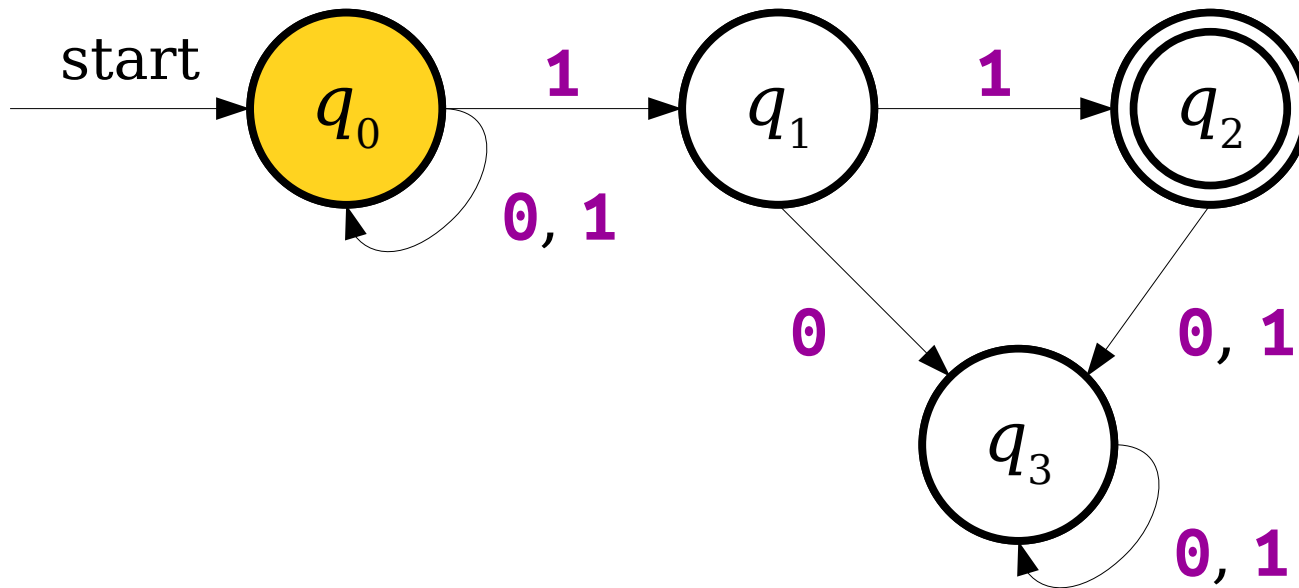
# A Simple NFA



# A Simple NFA

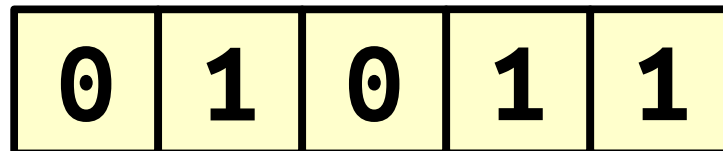
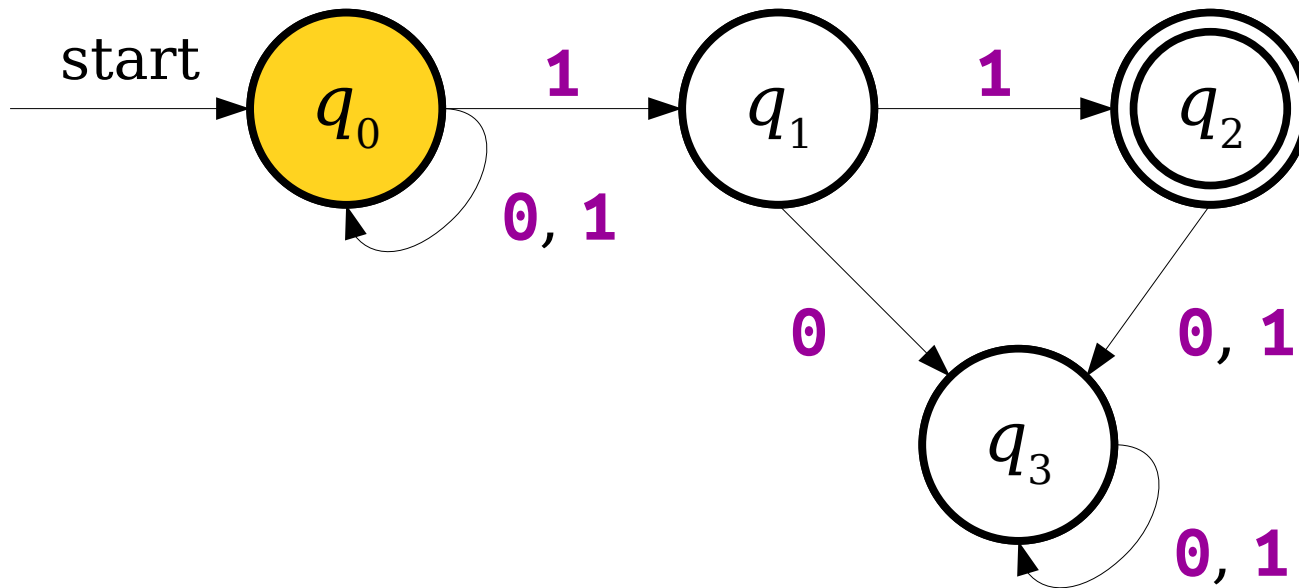


# A Simple NFA

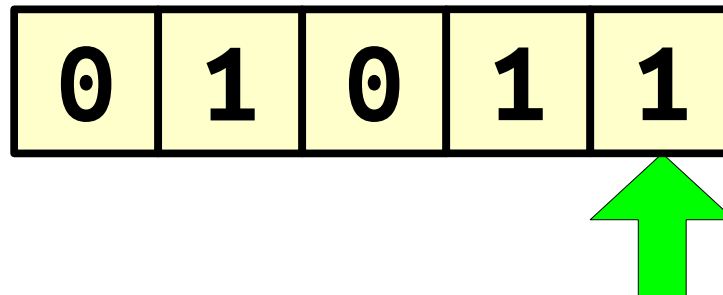
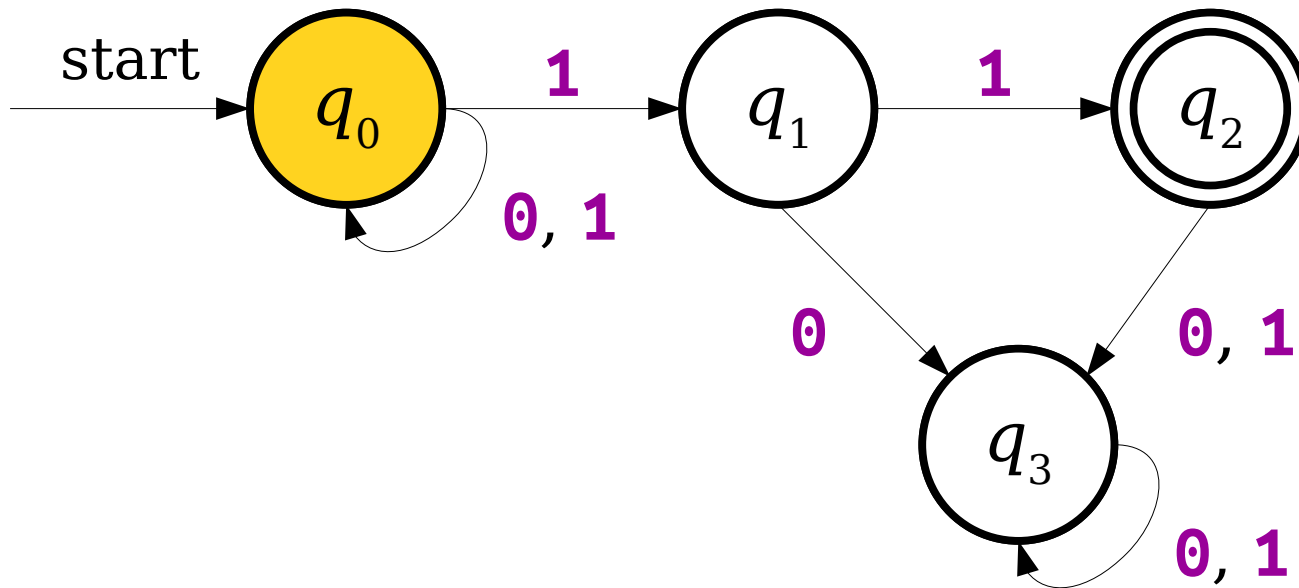




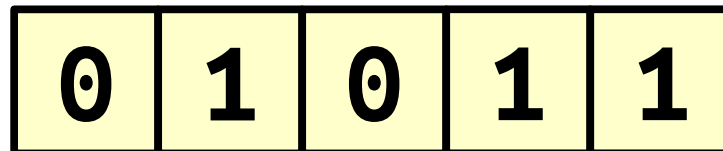
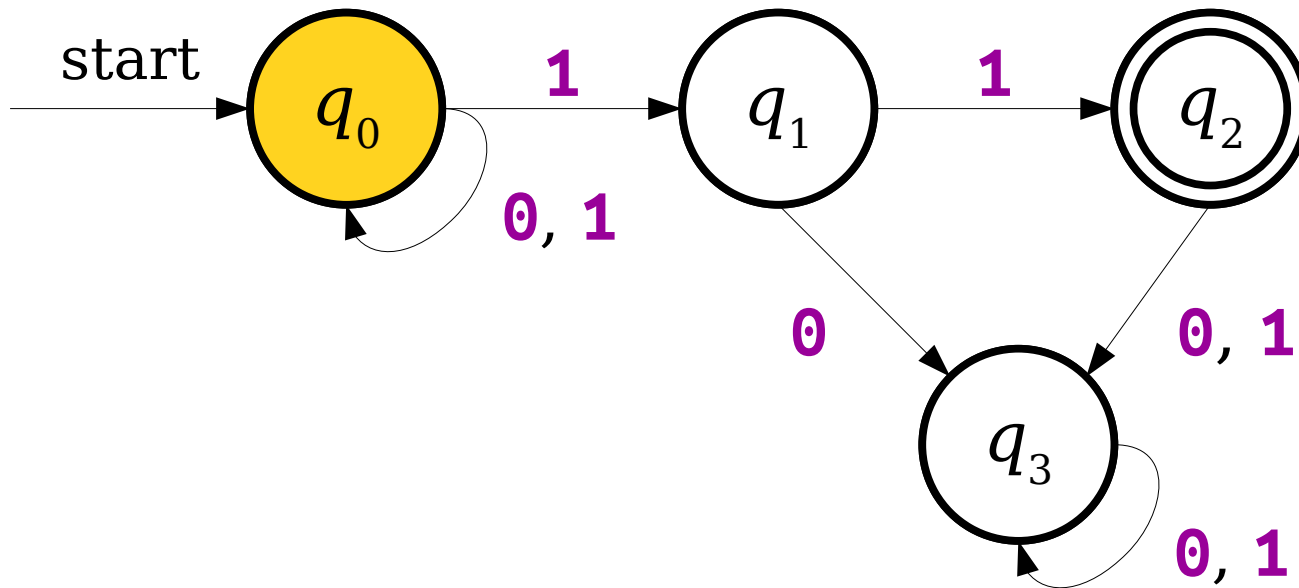
# A Simple NFA



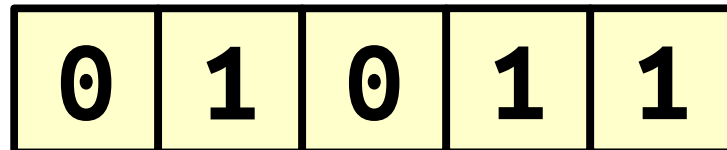
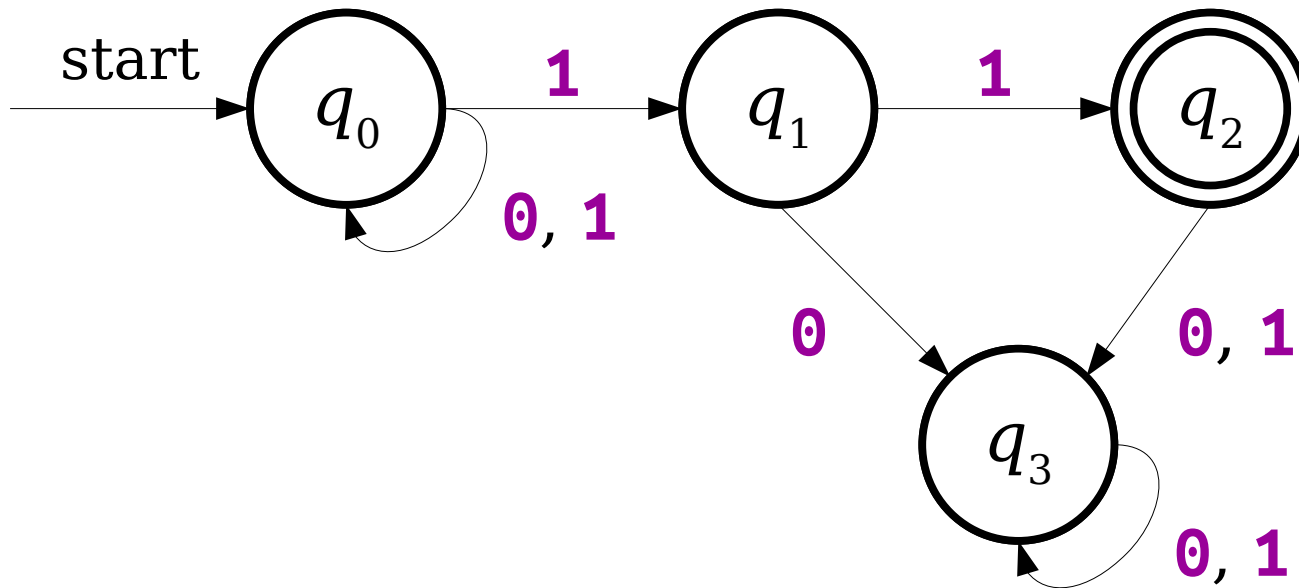
# A Simple NFA



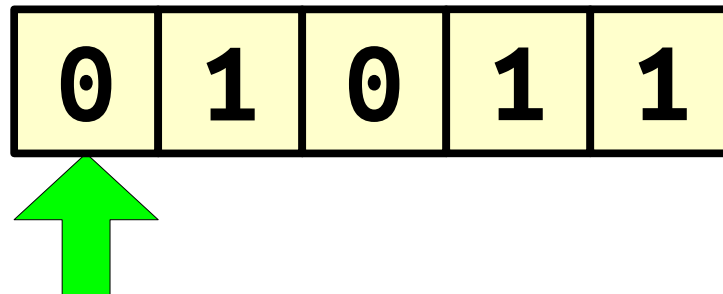
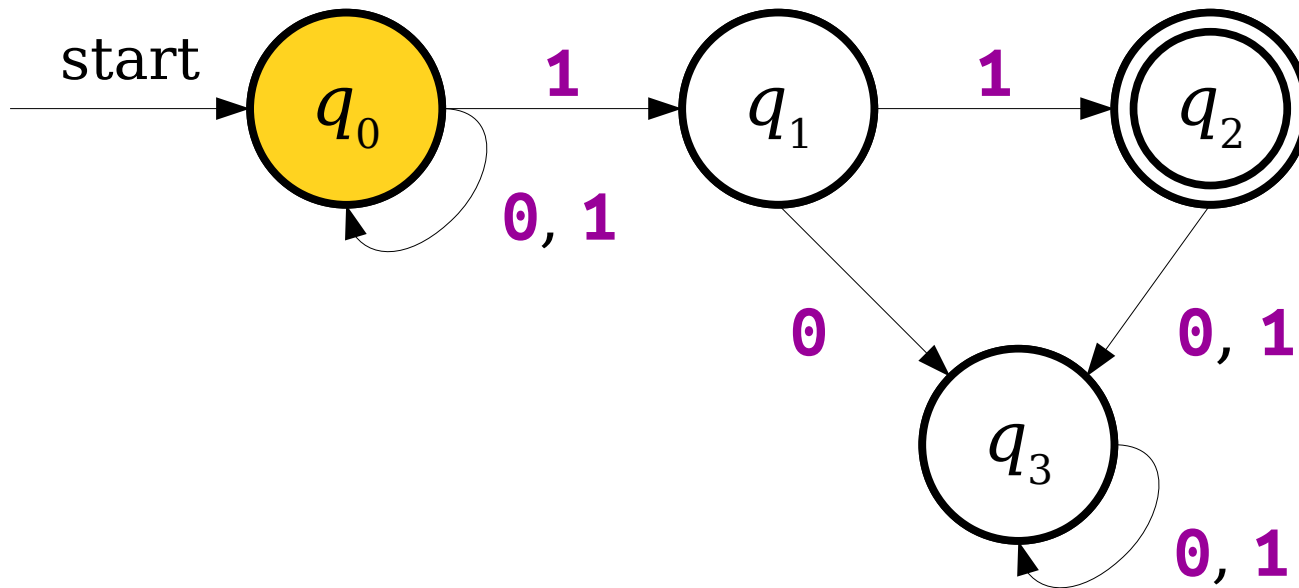
# A Simple NFA



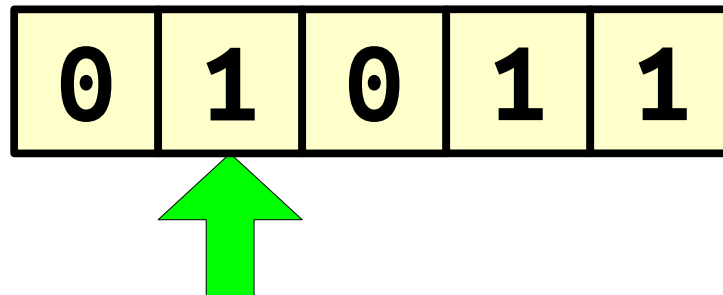
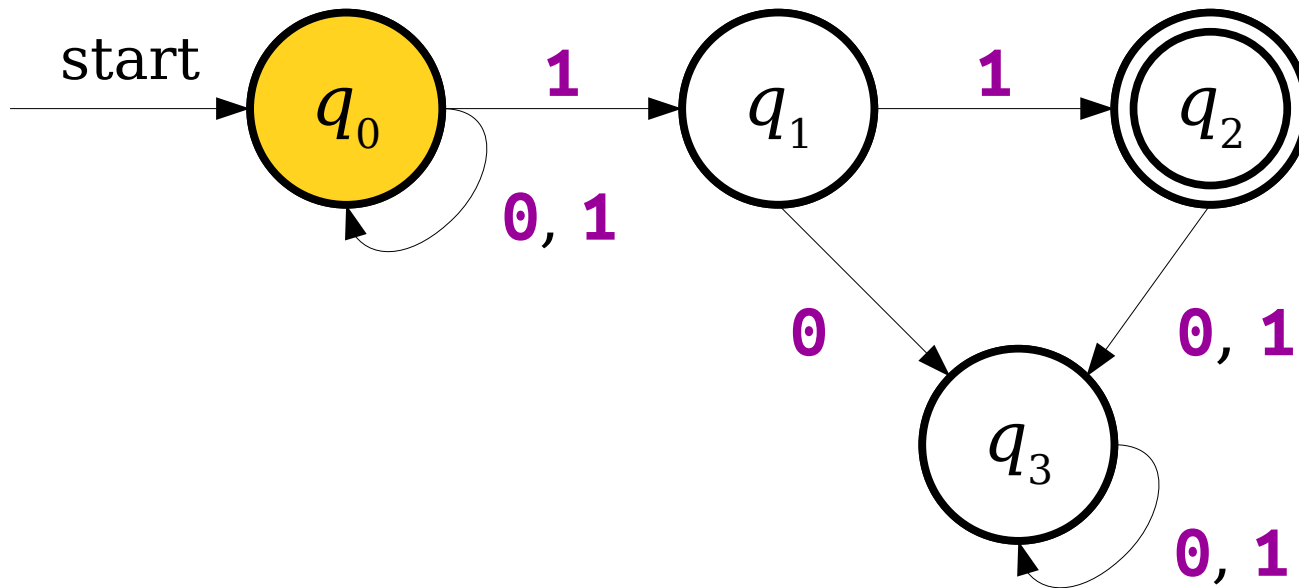
# A Simple NFA



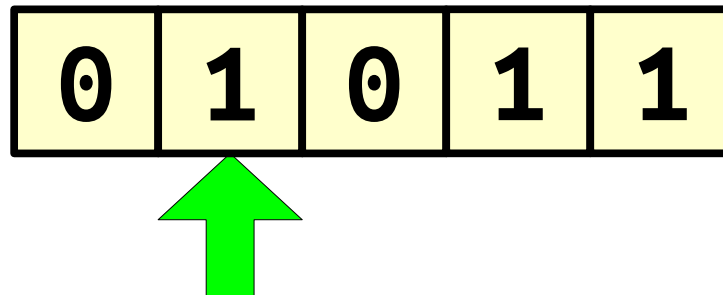
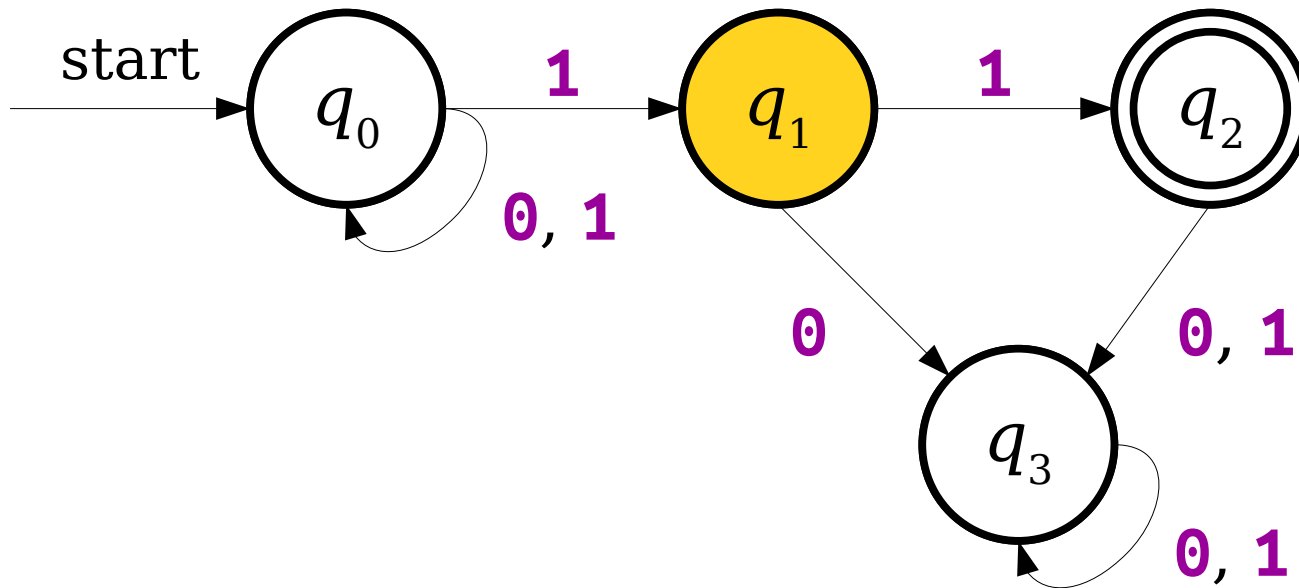
# A Simple NFA



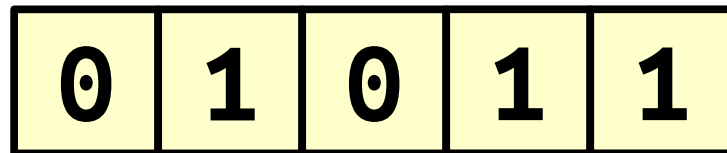
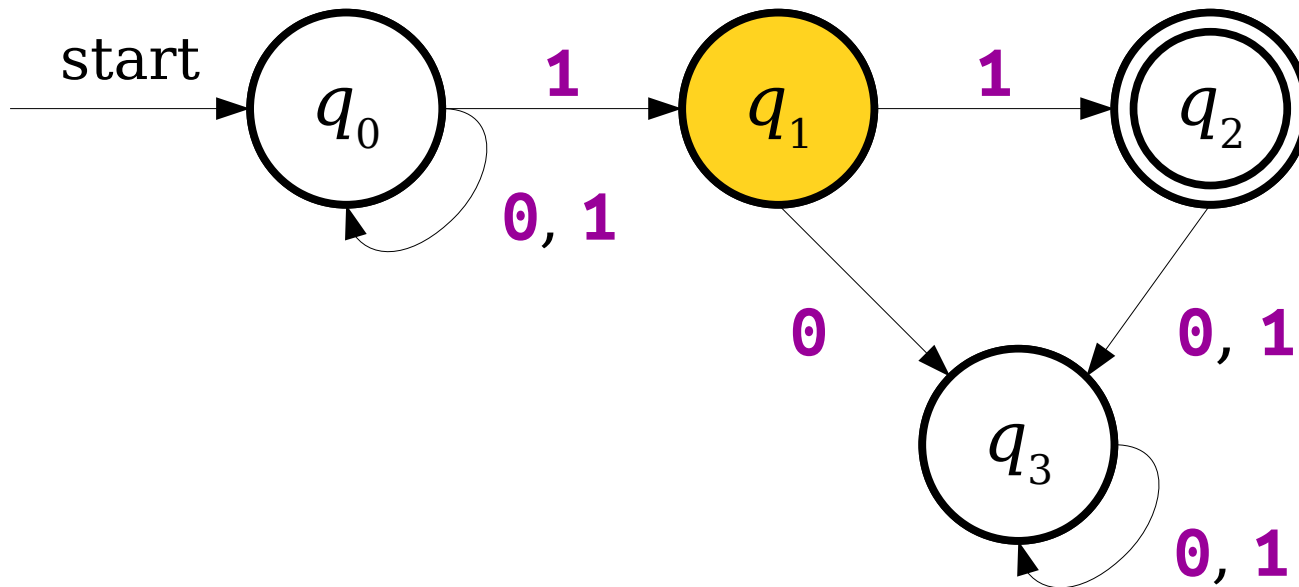
# A Simple NFA



# A Simple NFA

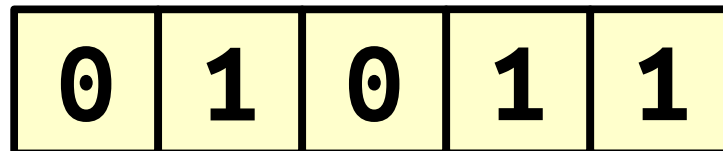
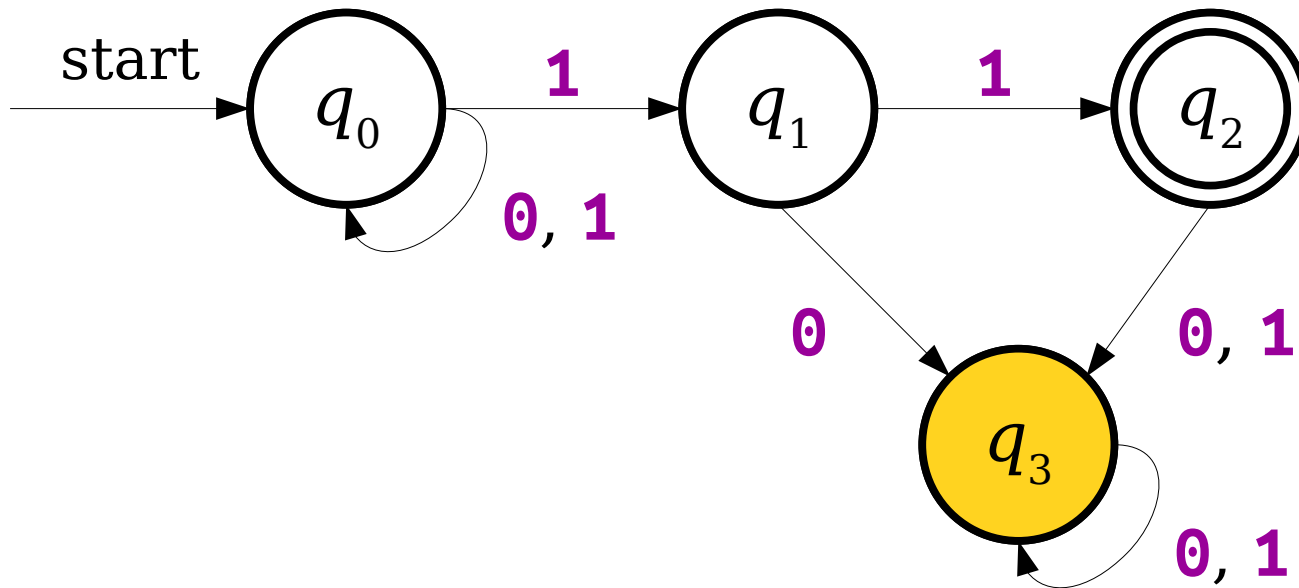


# A Simple NFA

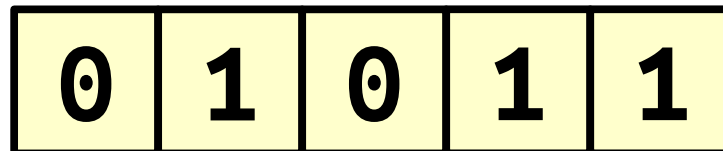
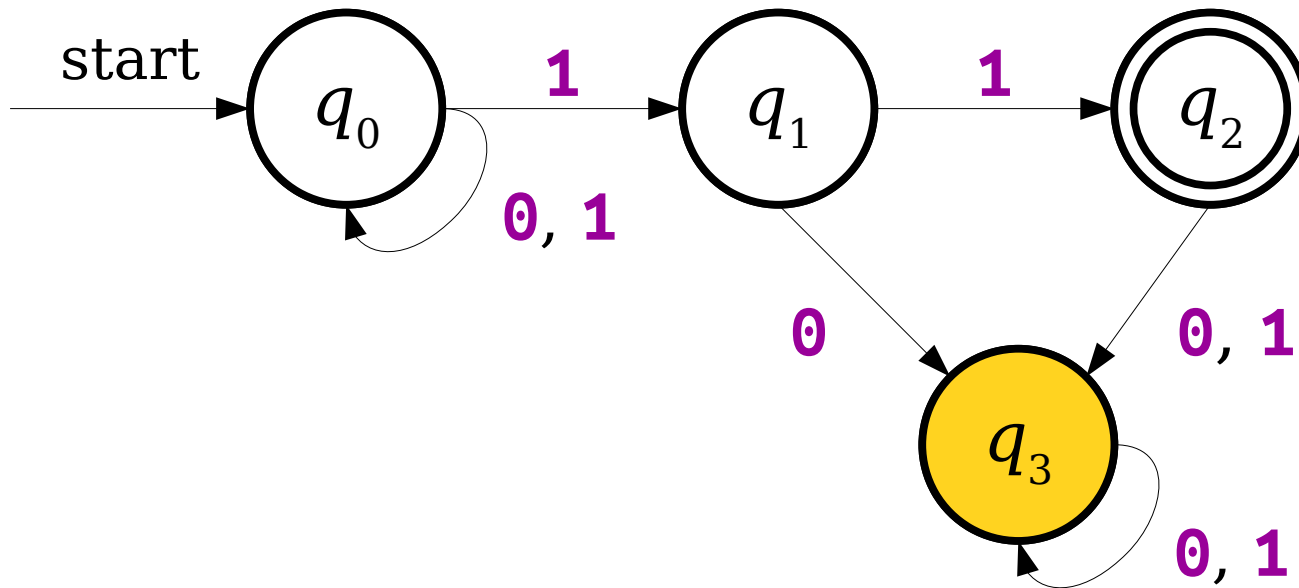




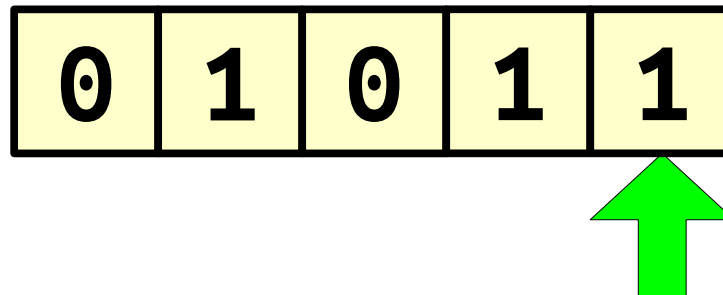
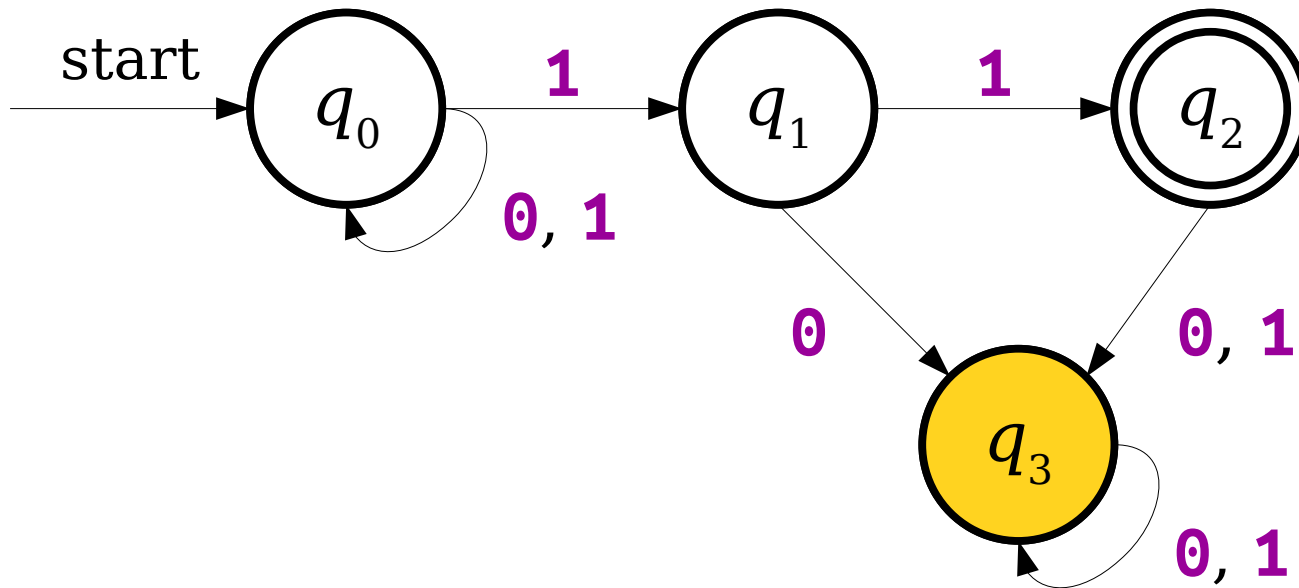
# A Simple NFA



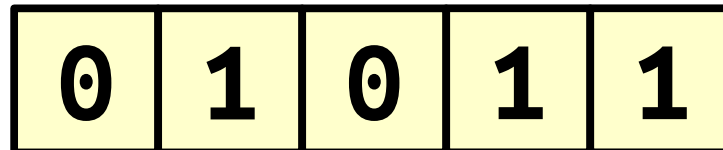
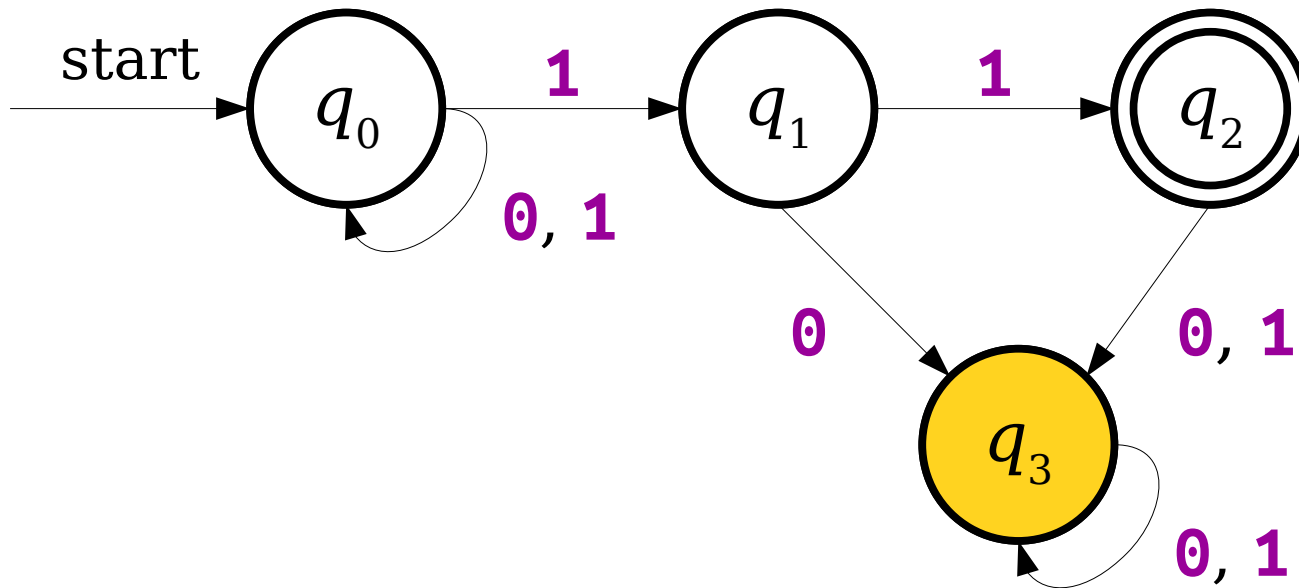
# A Simple NFA



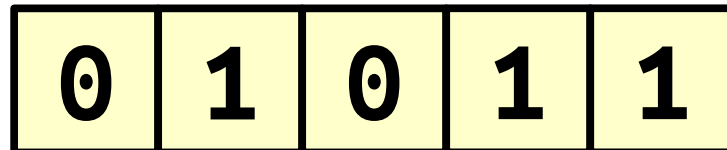
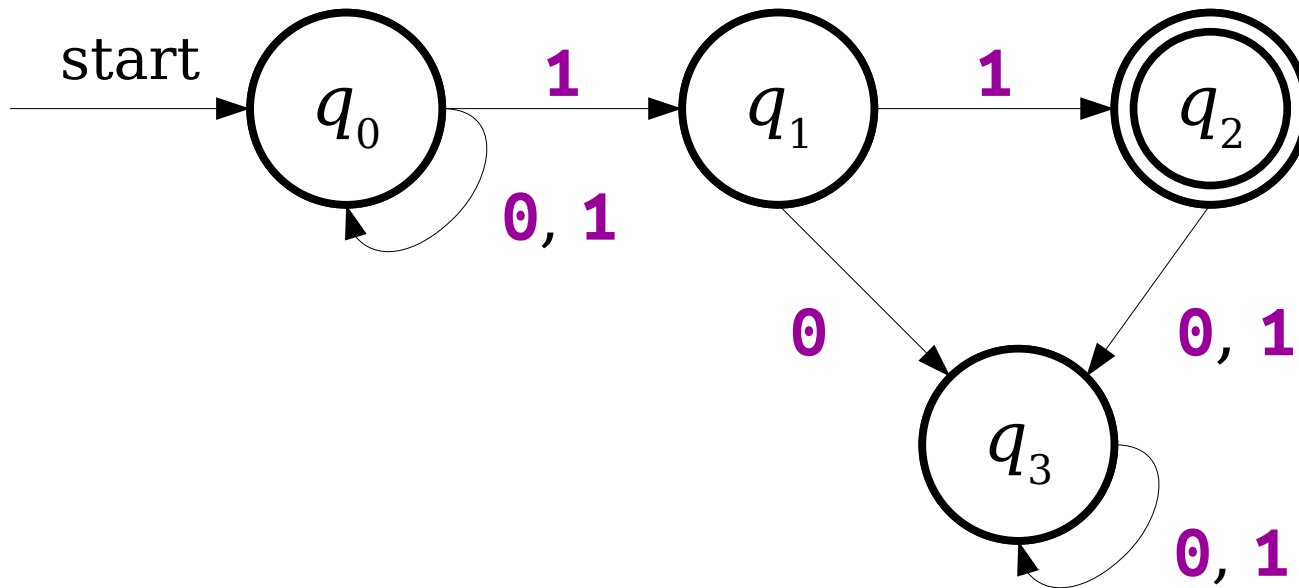
# A Simple NFA



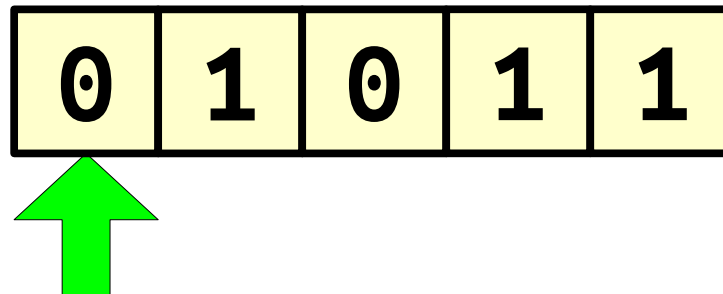
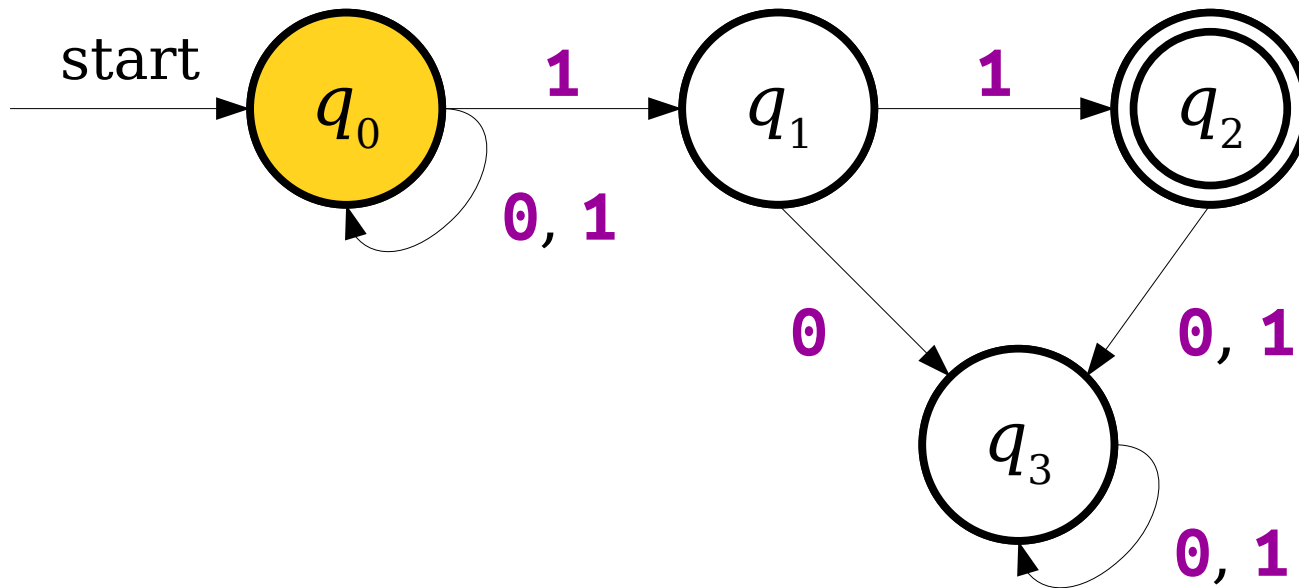
# A Simple NFA



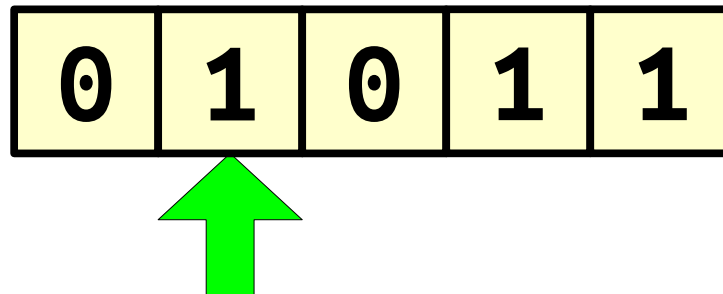
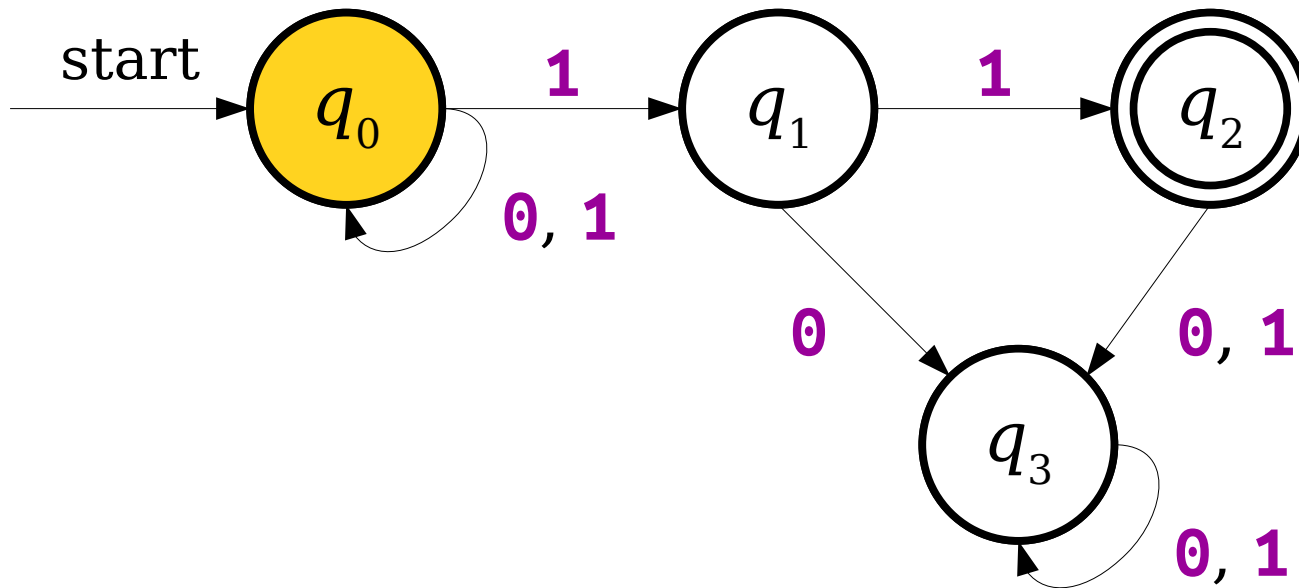
# A Simple NFA



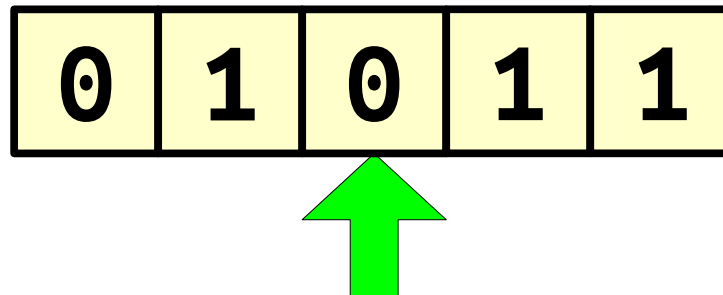
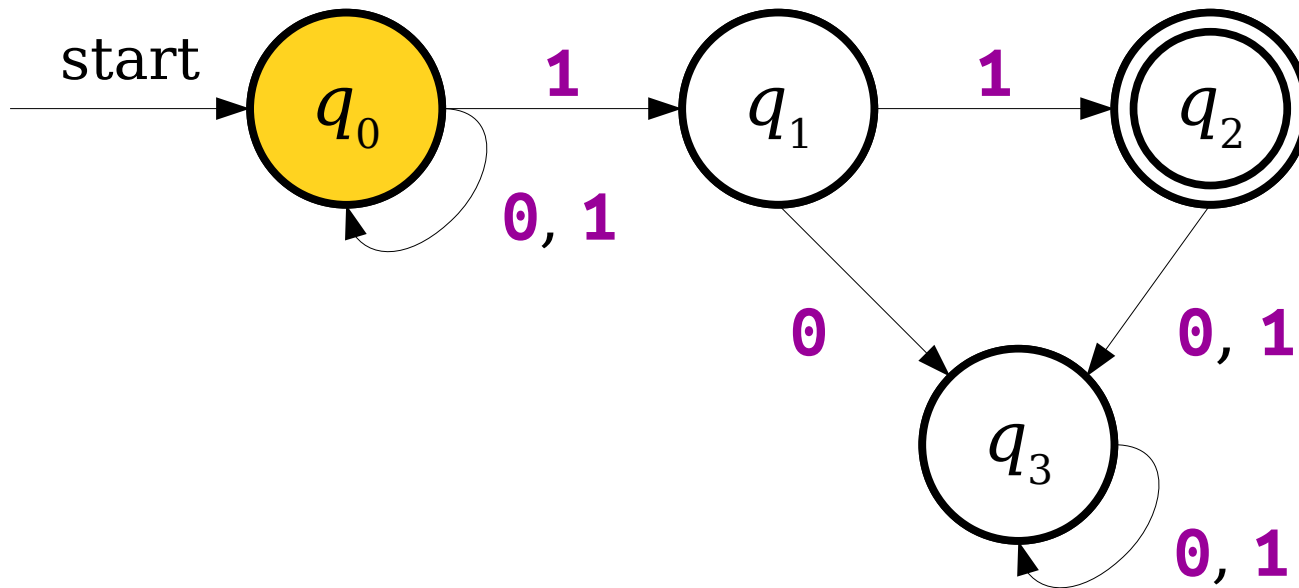
# A Simple NFA



# A Simple NFA

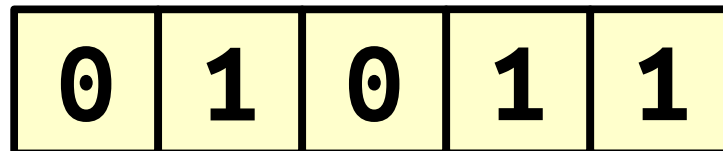
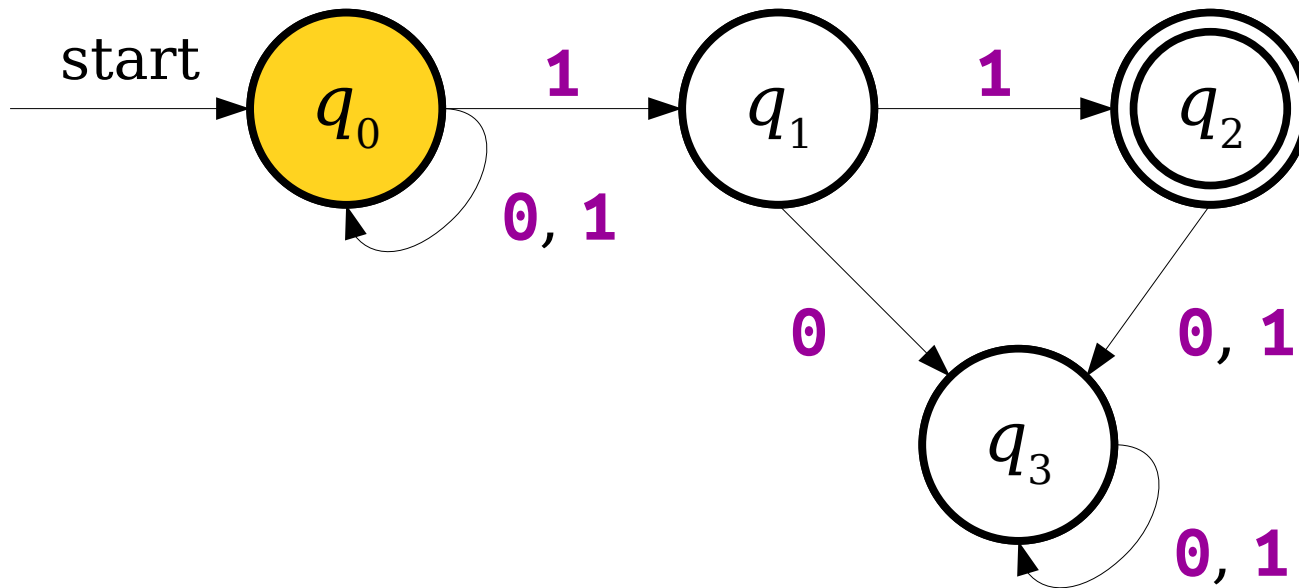


# A Simple NFA

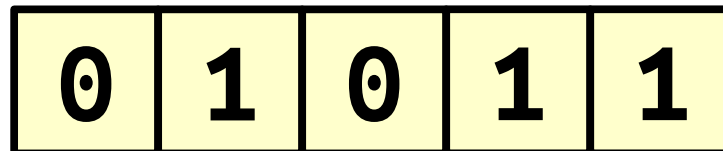
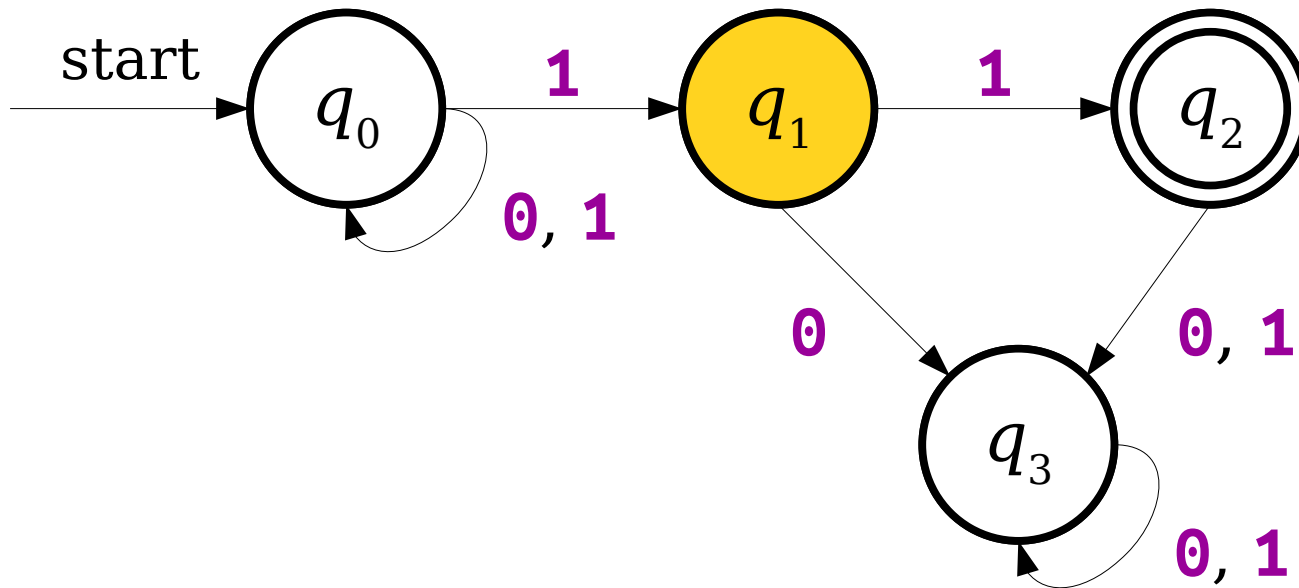




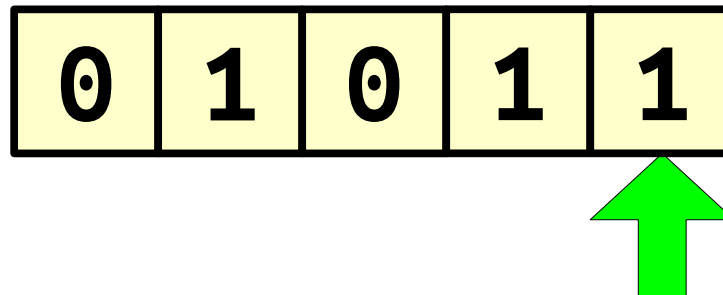
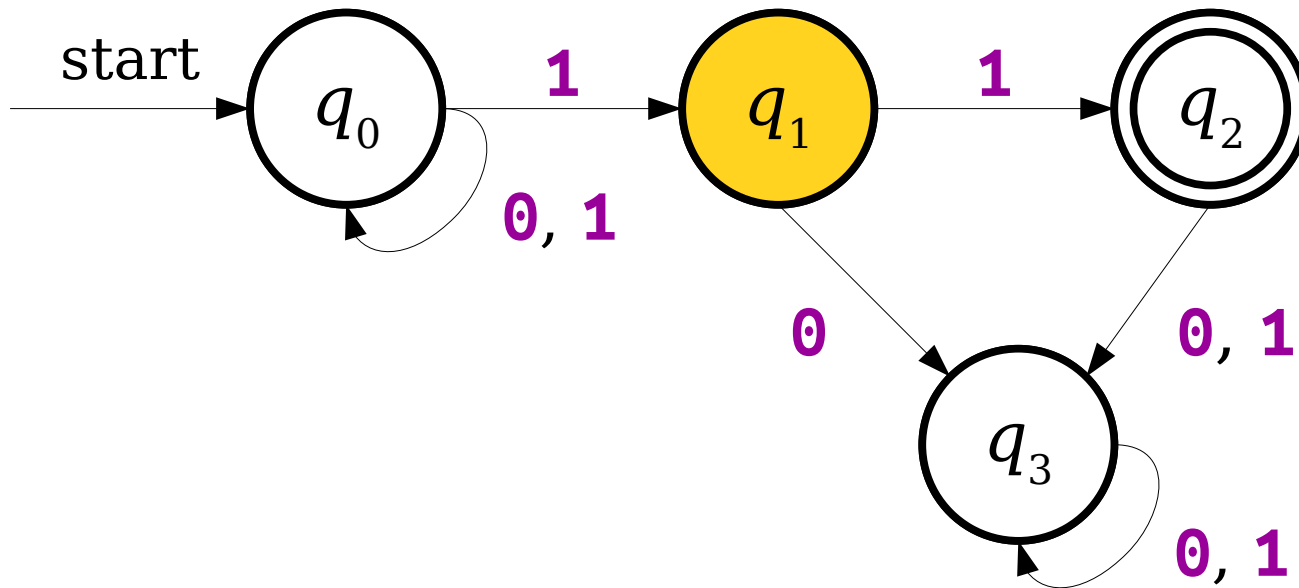
# A Simple NFA



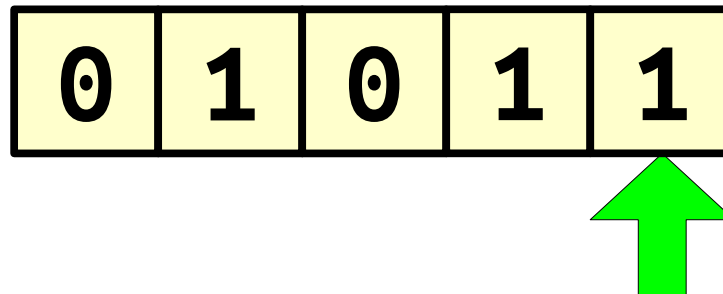
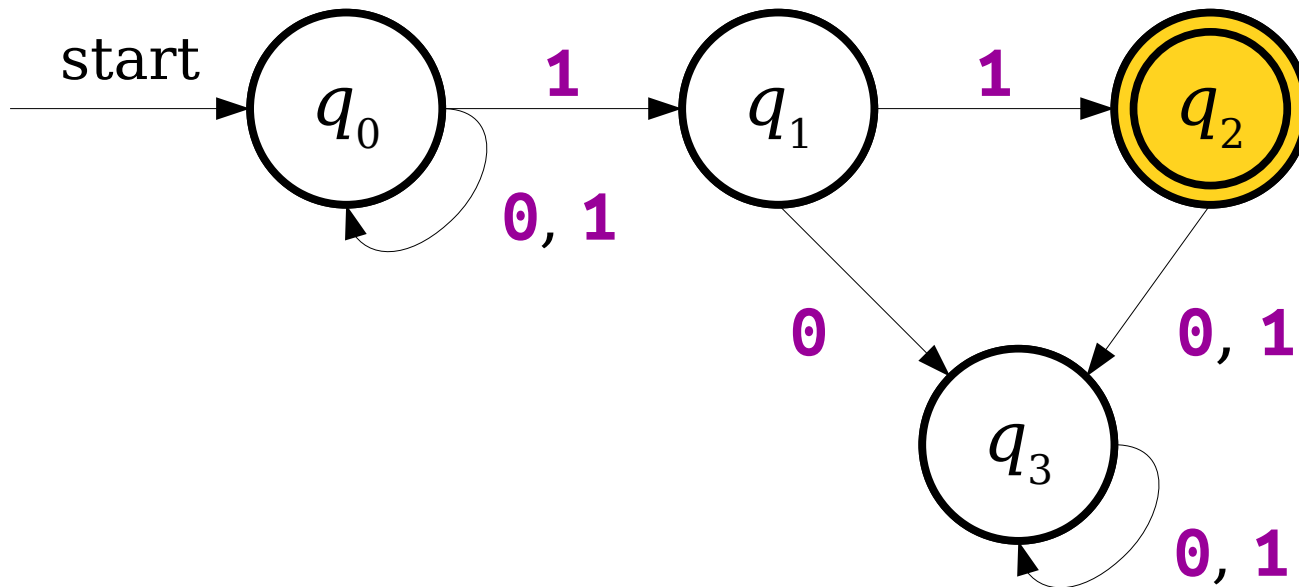
# A Simple NFA



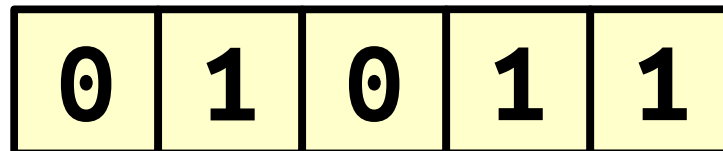
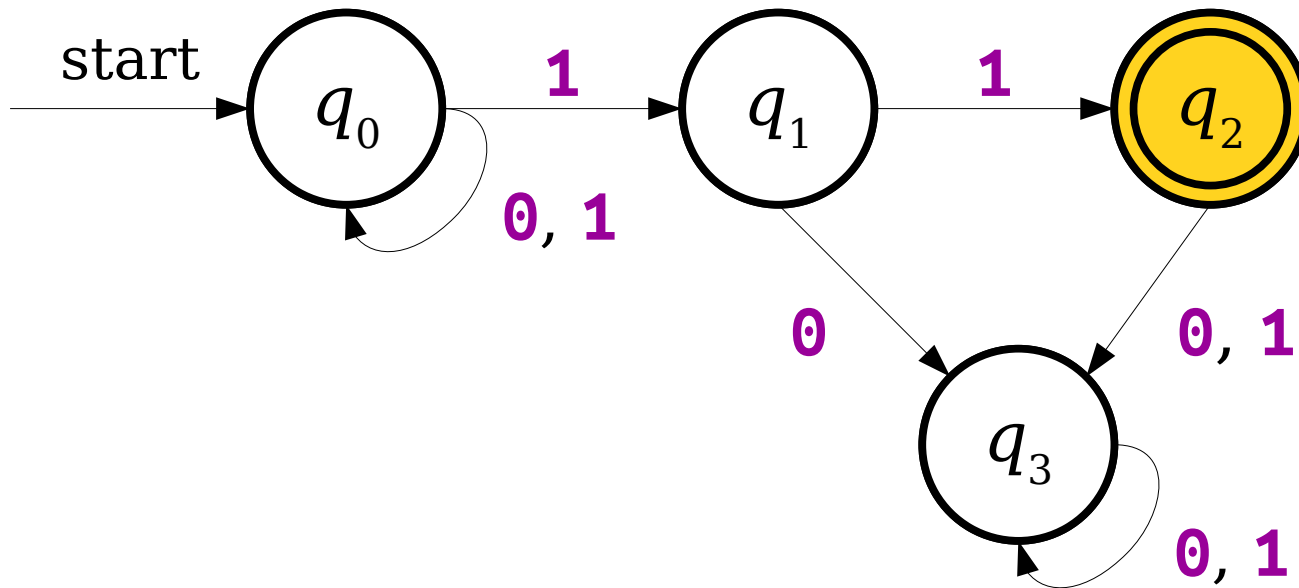
# A Simple NFA



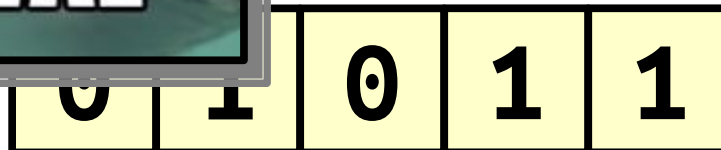
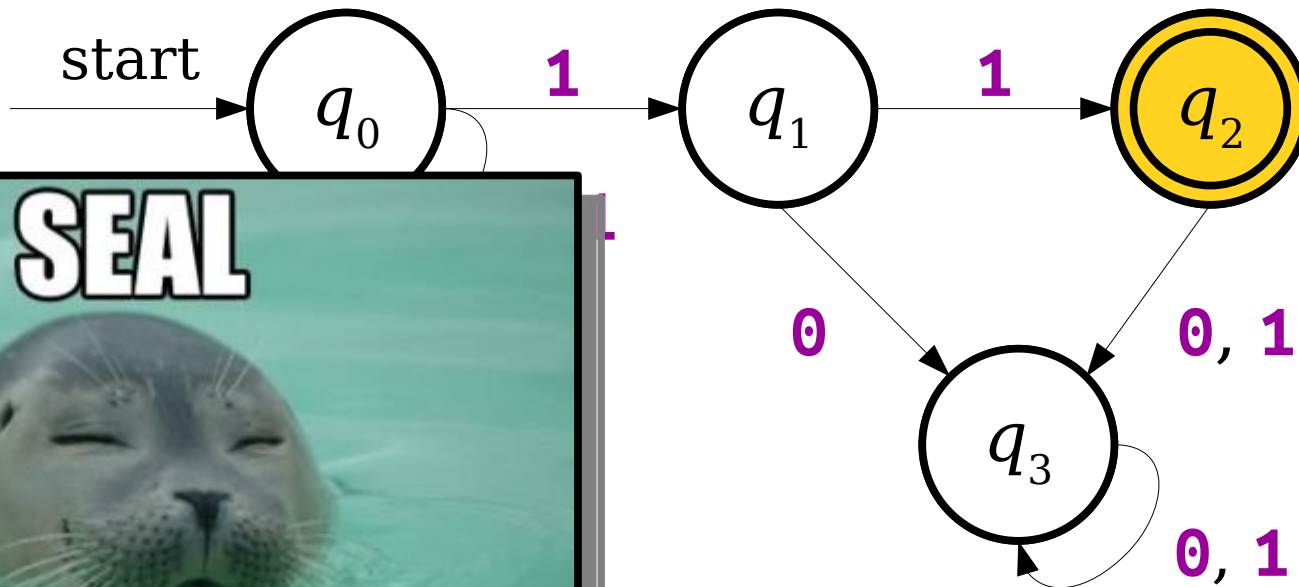
# A Simple NFA



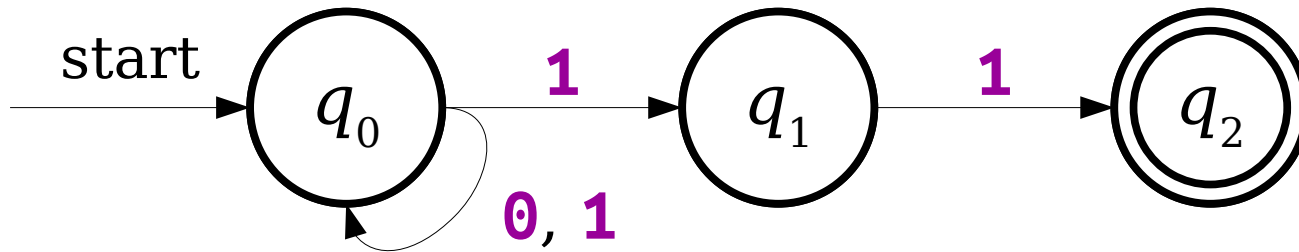
# A Simple NFA



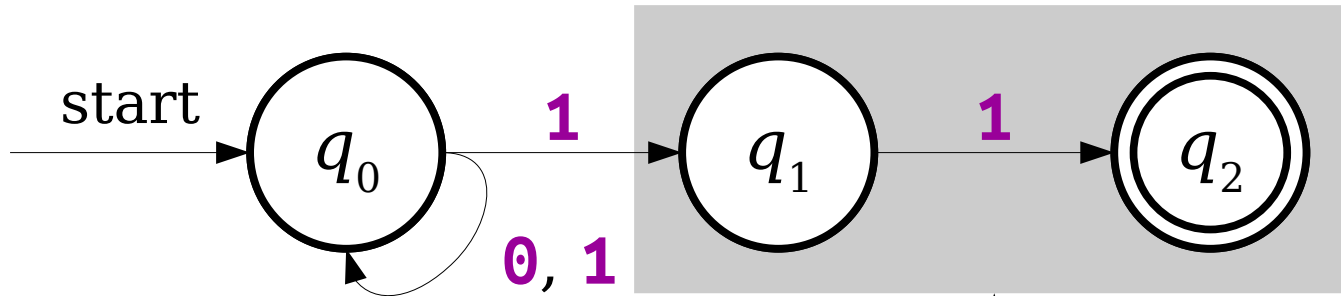
# A Simple NFA



# A More Complex NFA



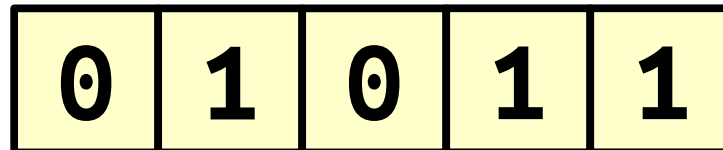
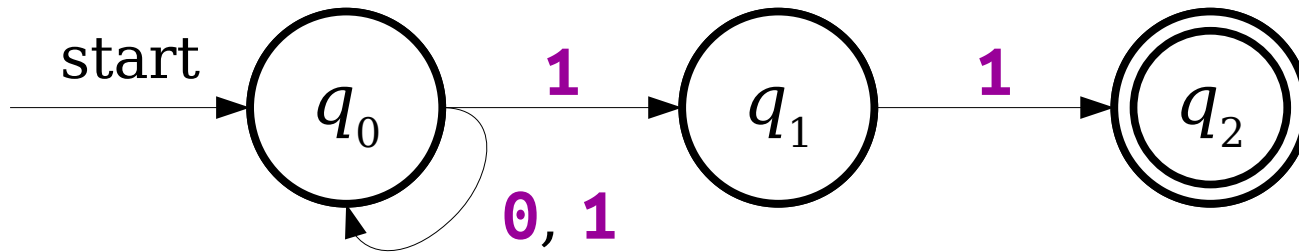
# A More Complex NFA



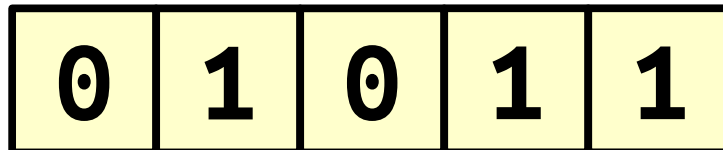
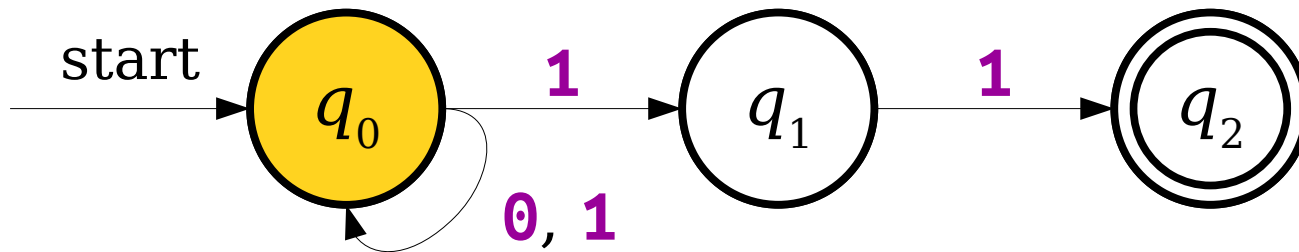
If a NFA needs to make a transition when no transition exists, the automaton **dies** and that particular path does not accept.



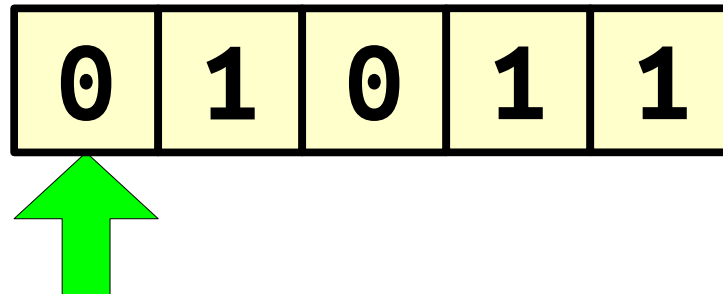
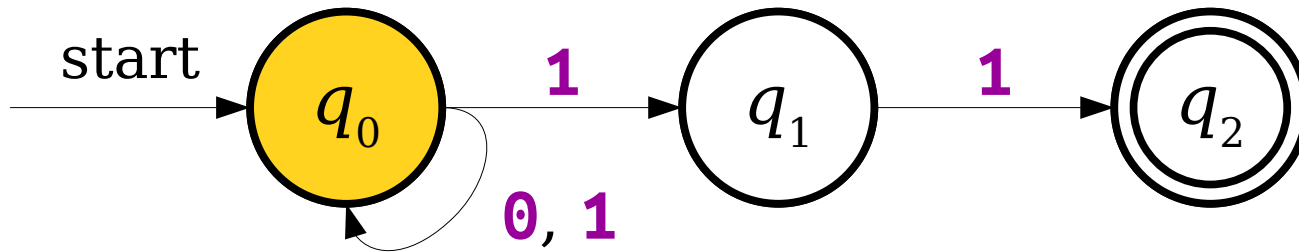
# A More Complex NFA



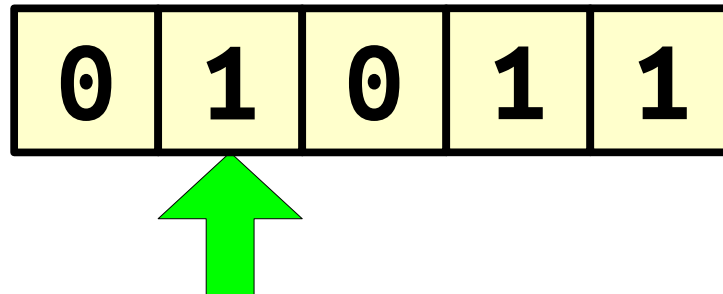
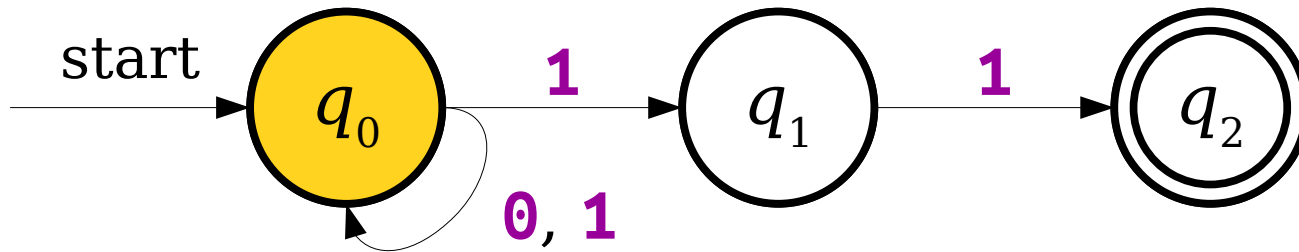
# A More Complex NFA



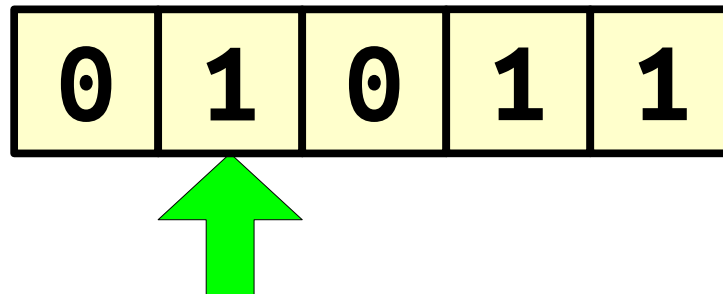
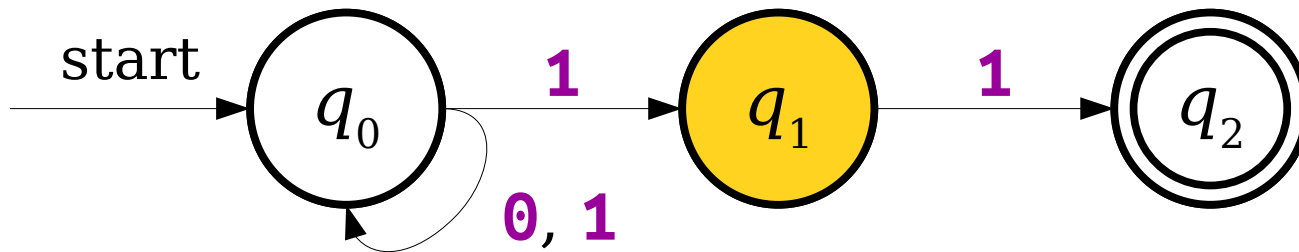
# A More Complex NFA



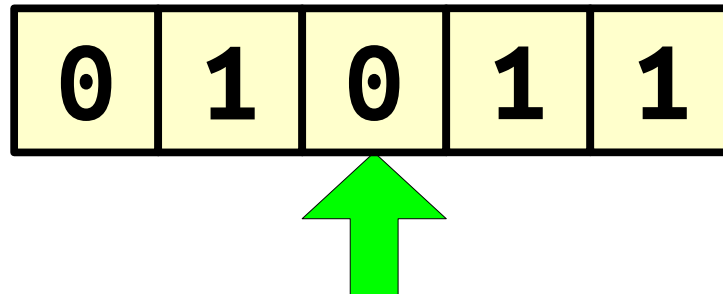
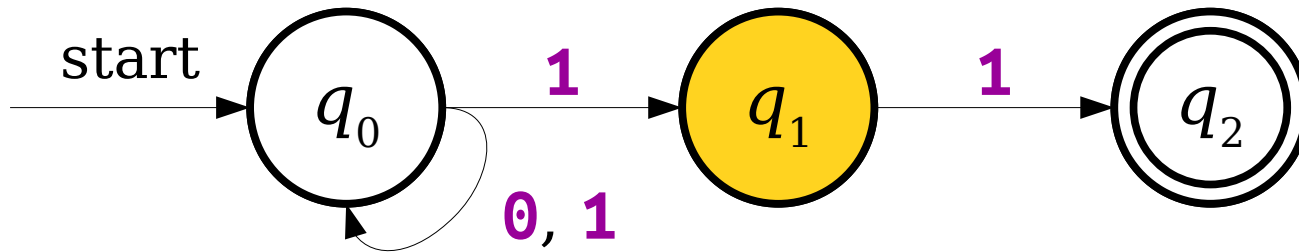
# A More Complex NFA



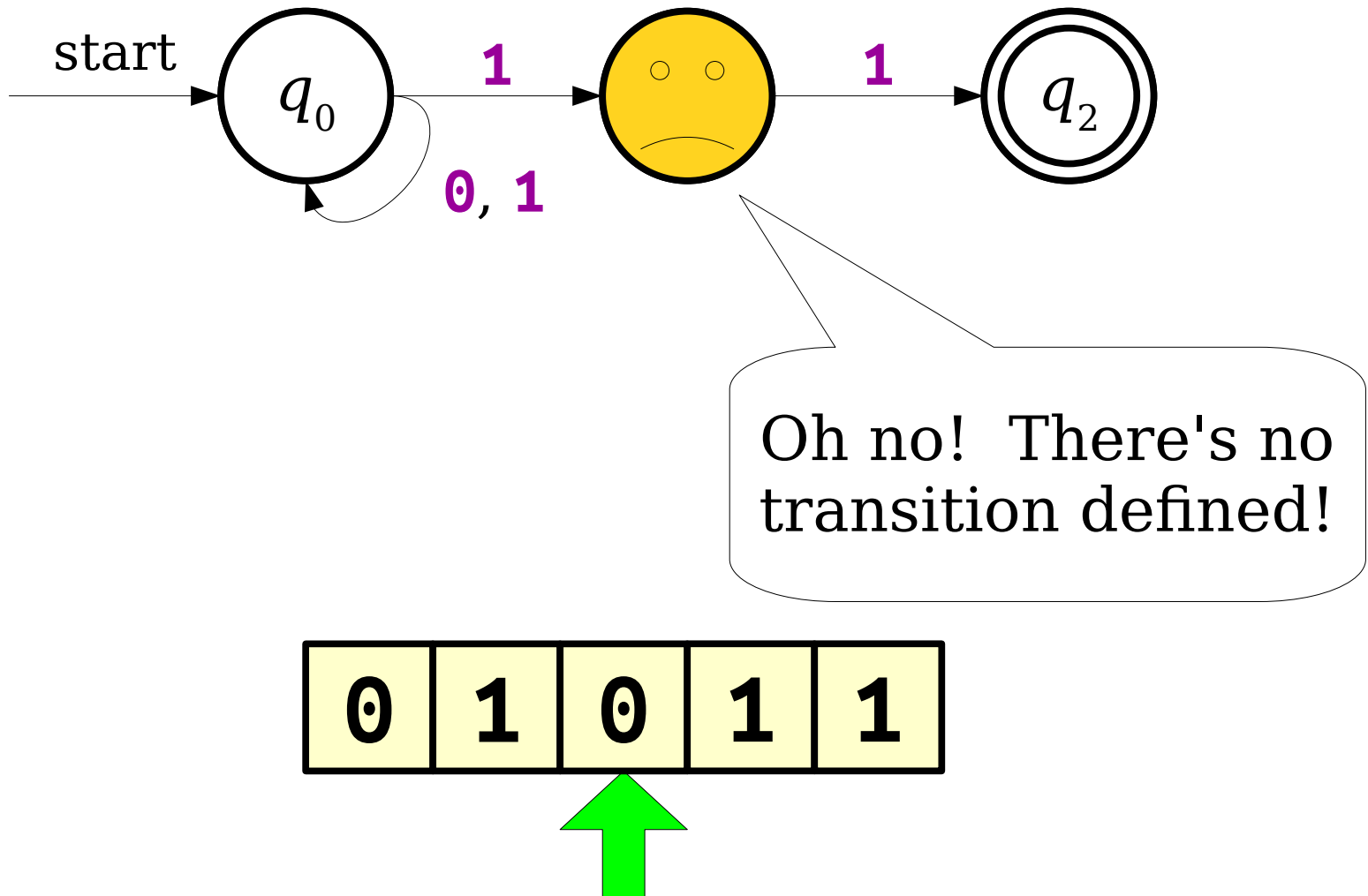
# A More Complex NFA



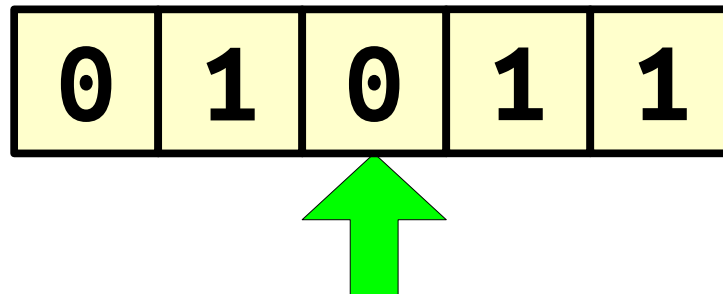
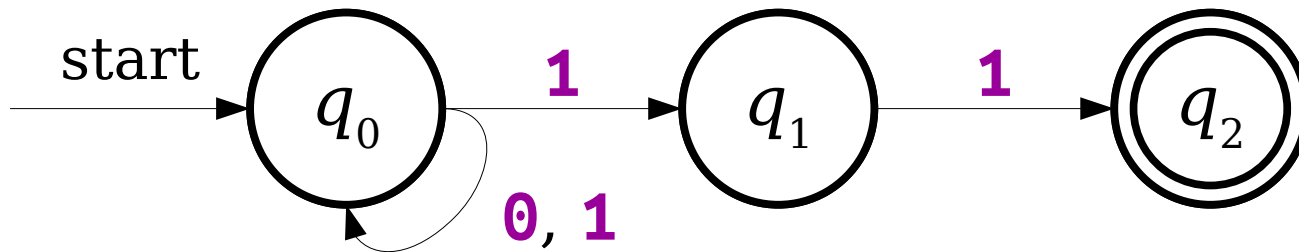
# A More Complex NFA



# A More Complex NFA

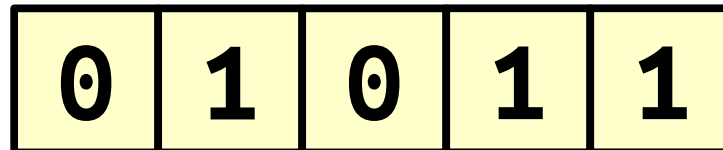
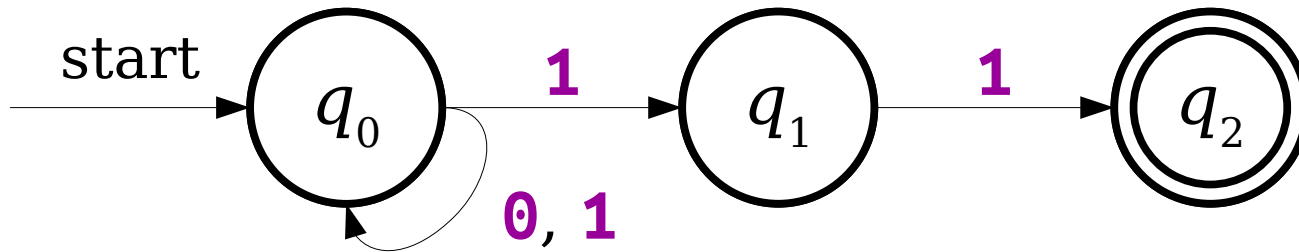


# A More Complex NFA

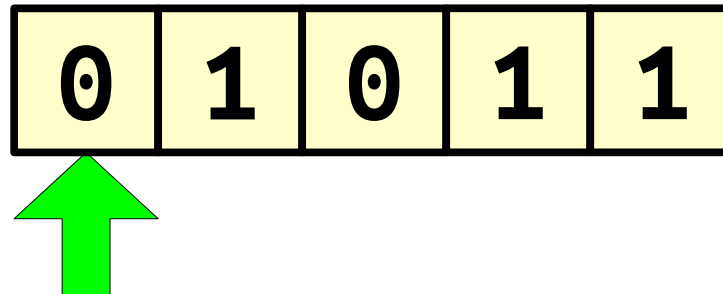
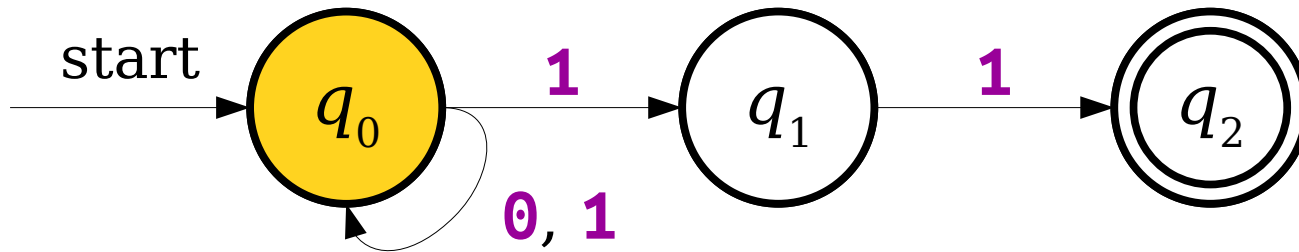




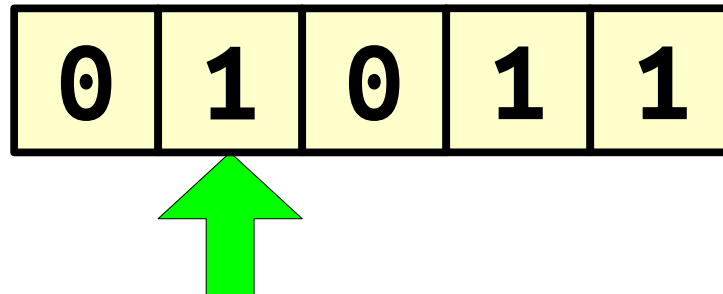
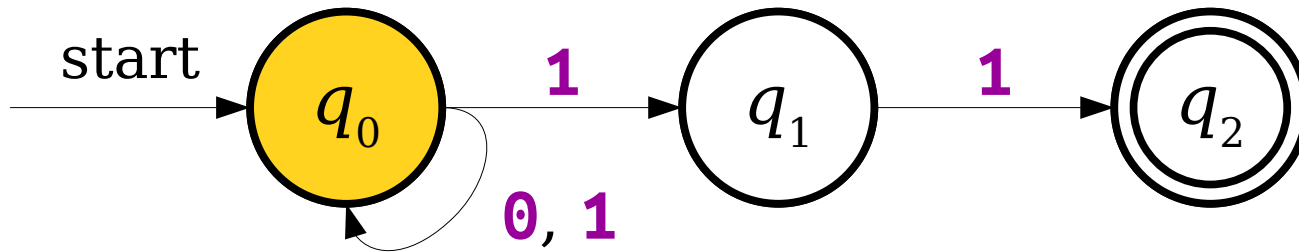
# A More Complex NFA



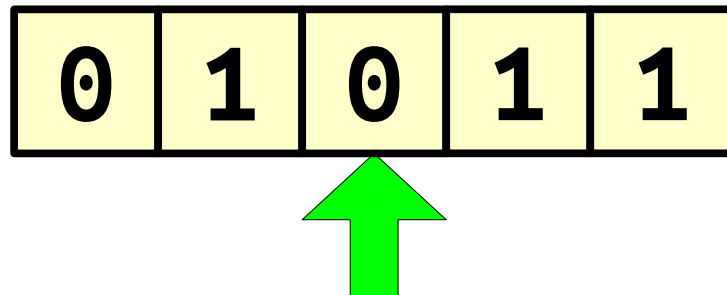
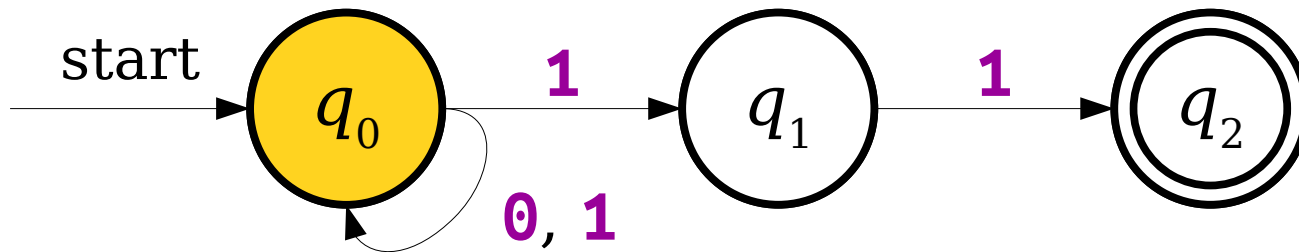
# A More Complex NFA



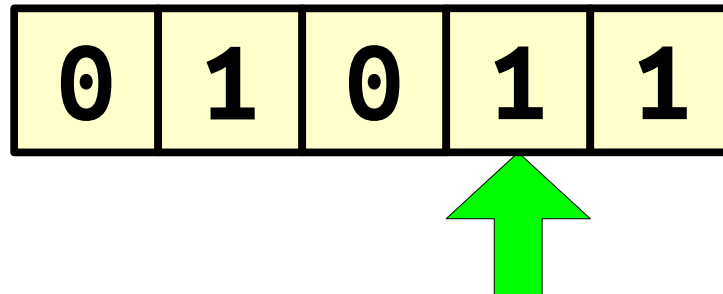
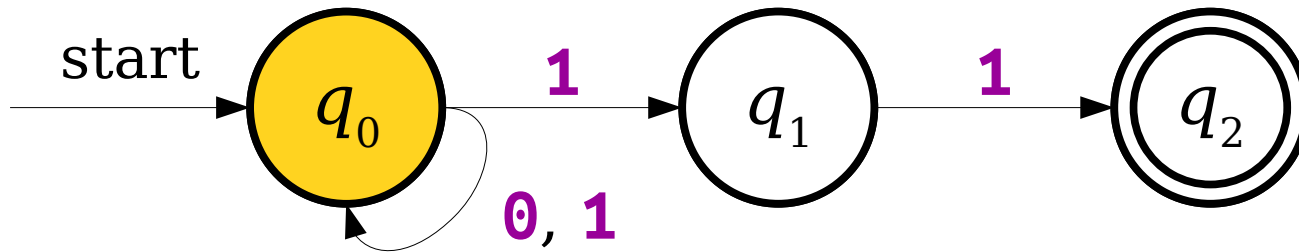
# A More Complex NFA



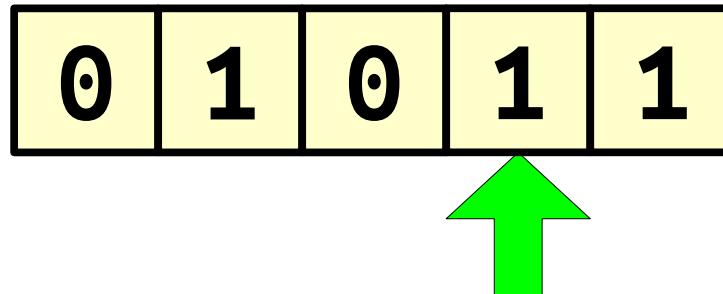
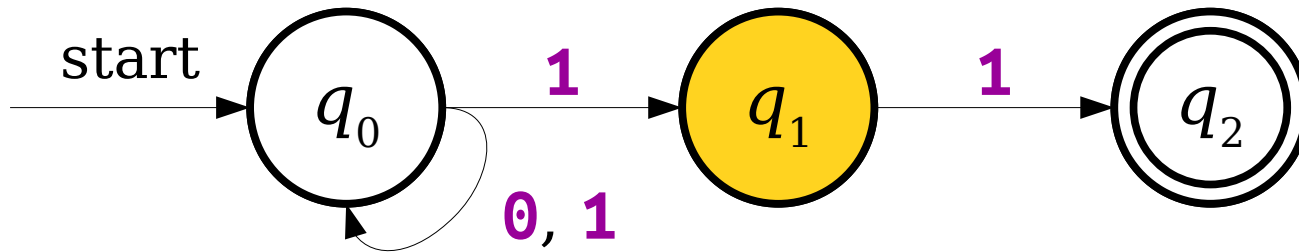
# A More Complex NFA



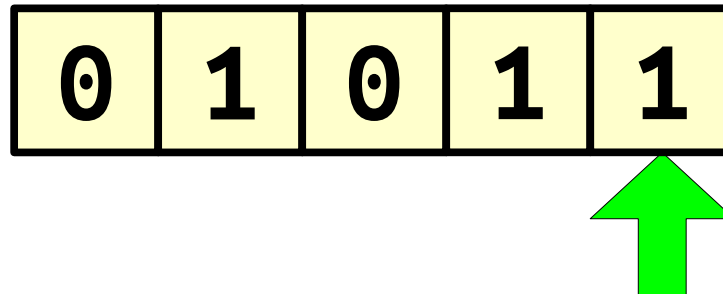
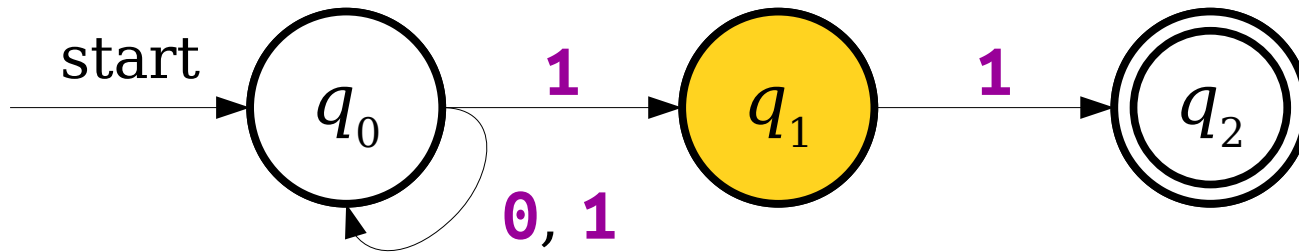
# A More Complex NFA



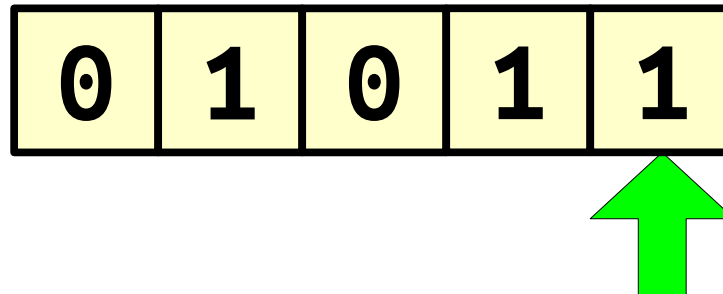
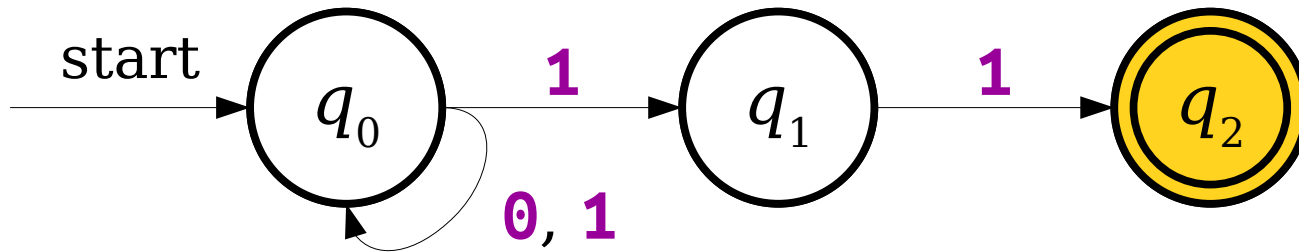
# A More Complex NFA



# A More Complex NFA

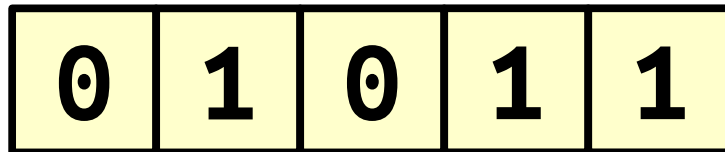
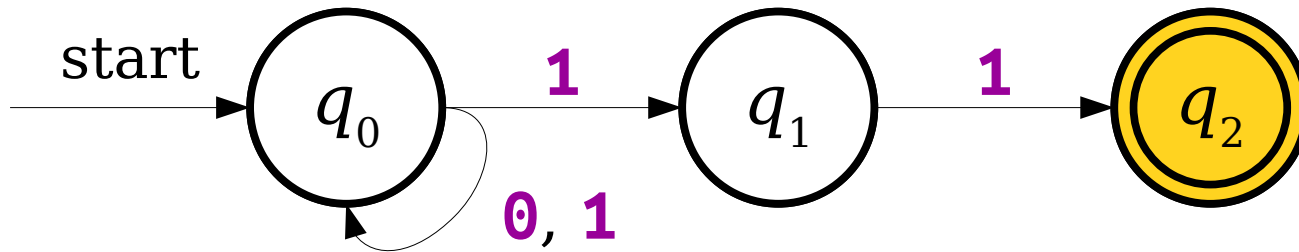


# A More Complex NFA

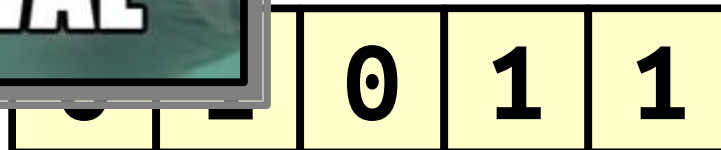
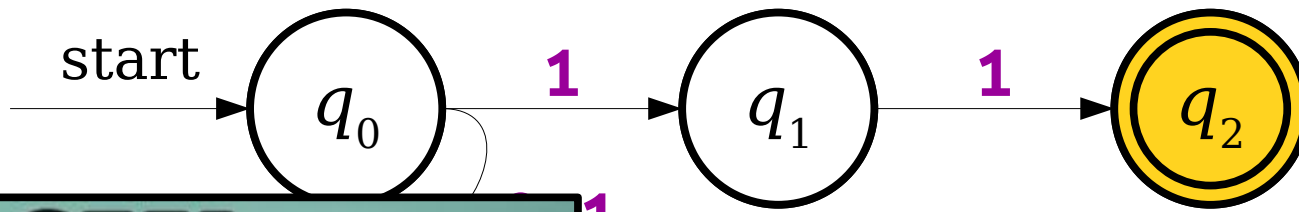




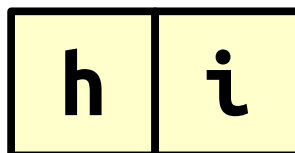
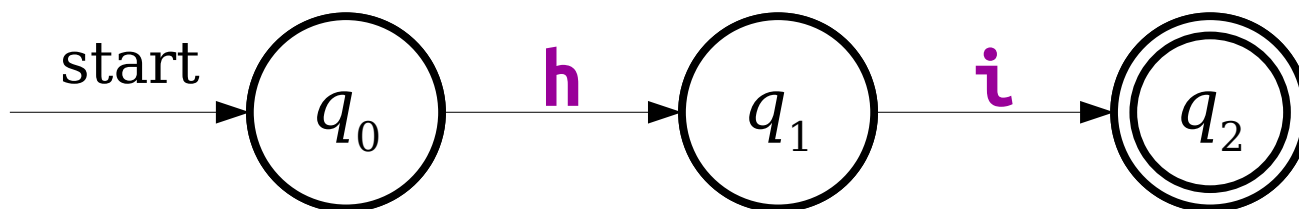
# A More Complex NFA



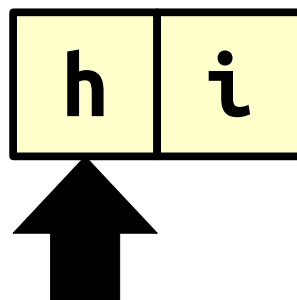
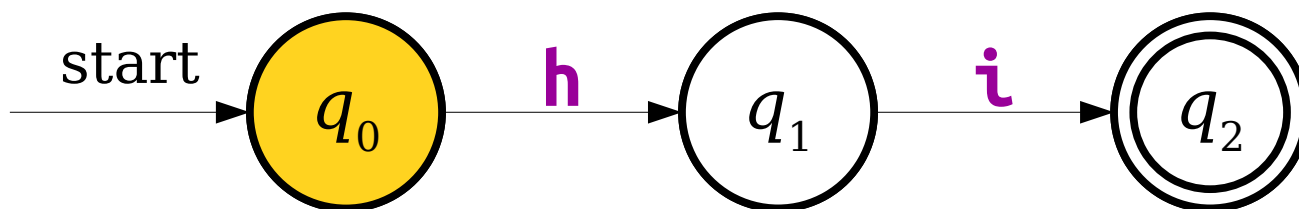
# A More Complex NFA



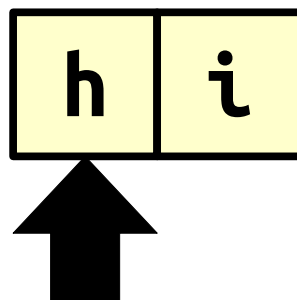
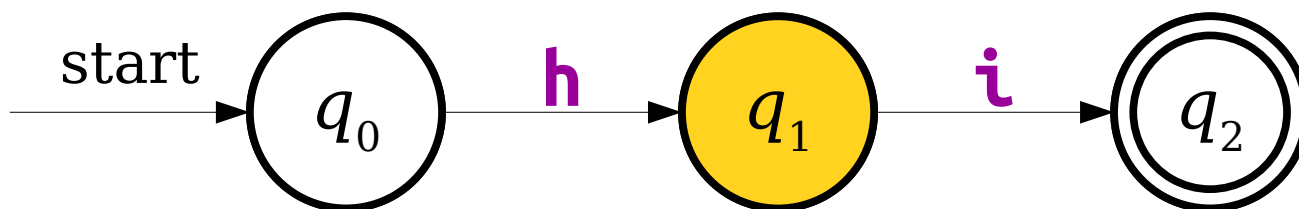
# Hello, NFA!



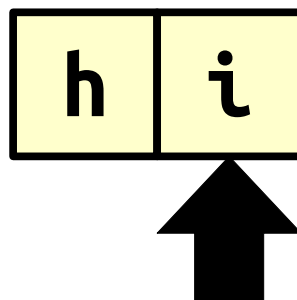
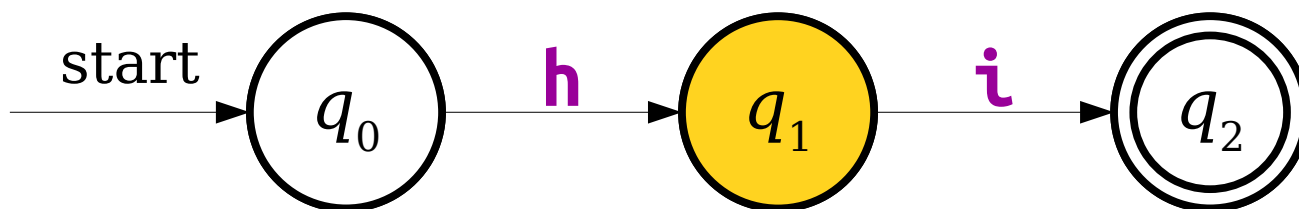
# Hello, NFA!



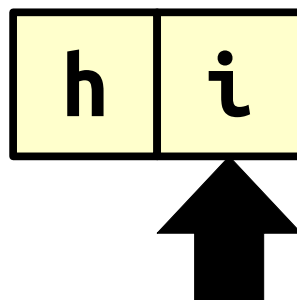
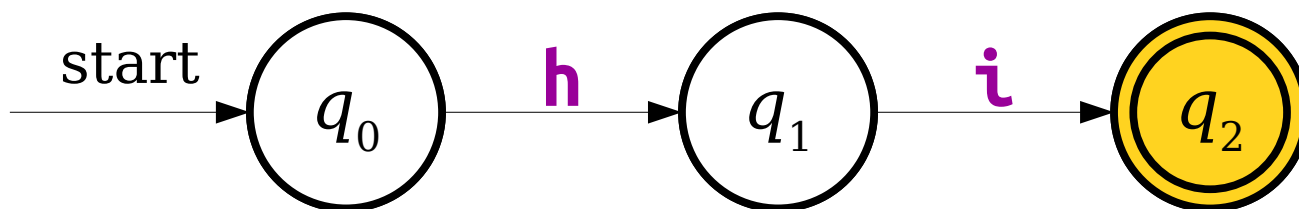
# Hello, NFA!



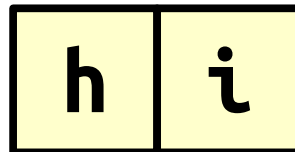
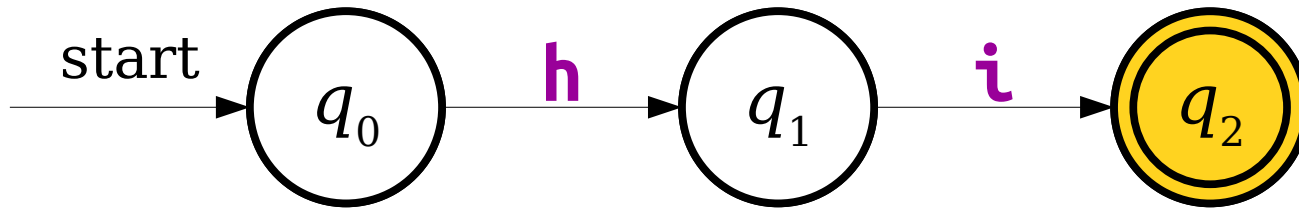
# Hello, NFA!



# Hello, NFA!

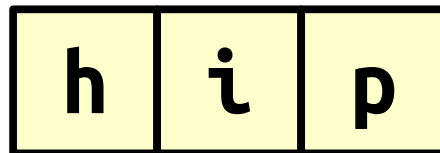
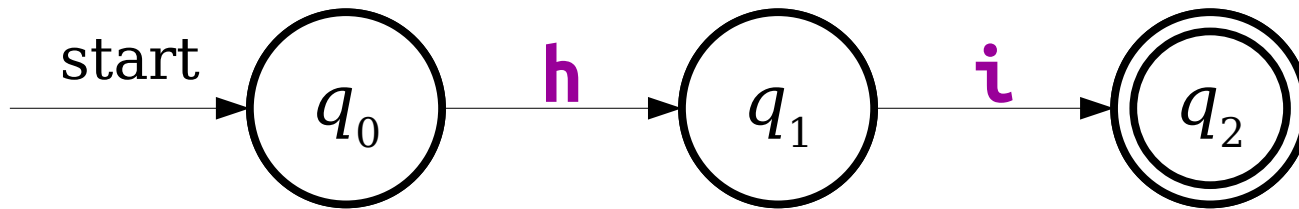


# Hello, NFA!

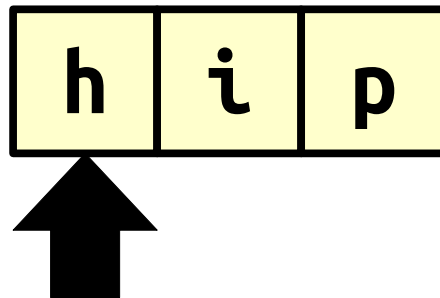
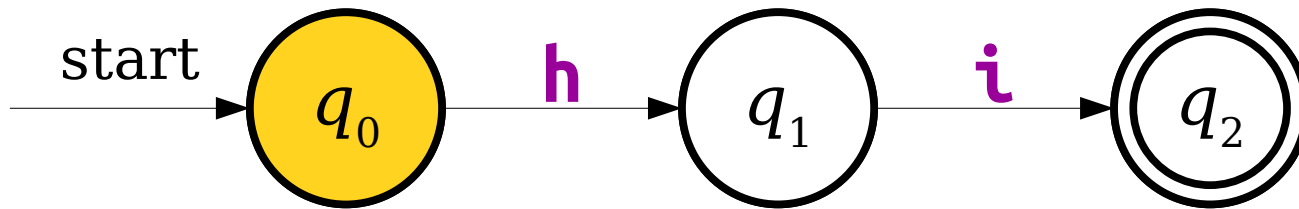




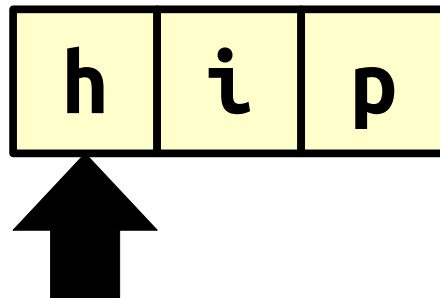
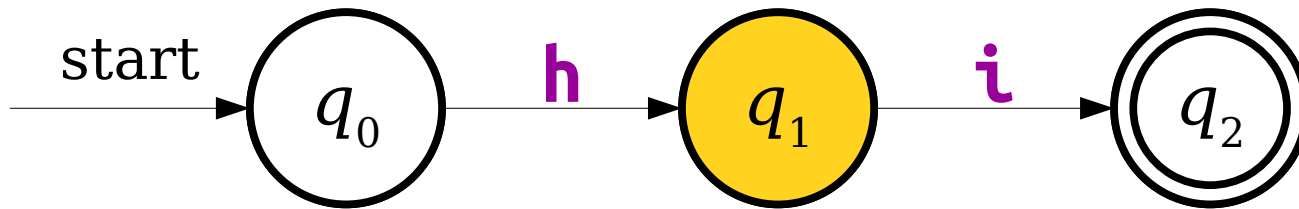
# Tragedy in Paradise



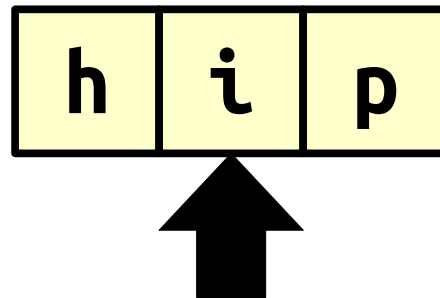
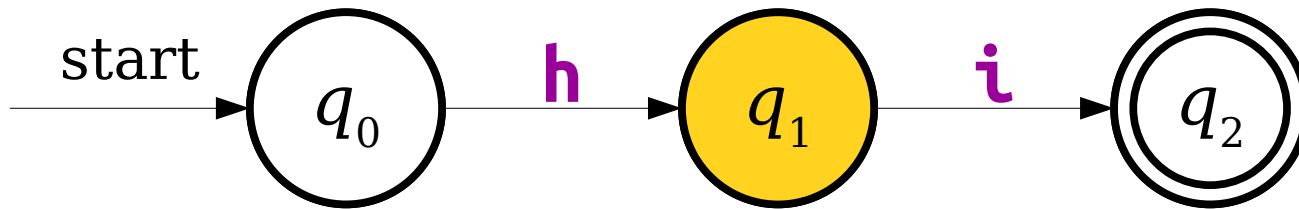
# Tragedy in Paradise



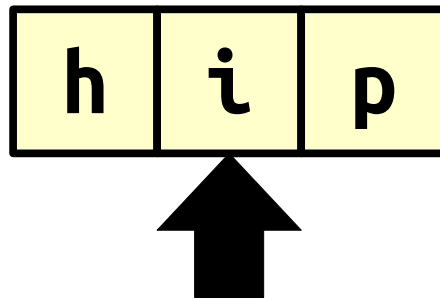
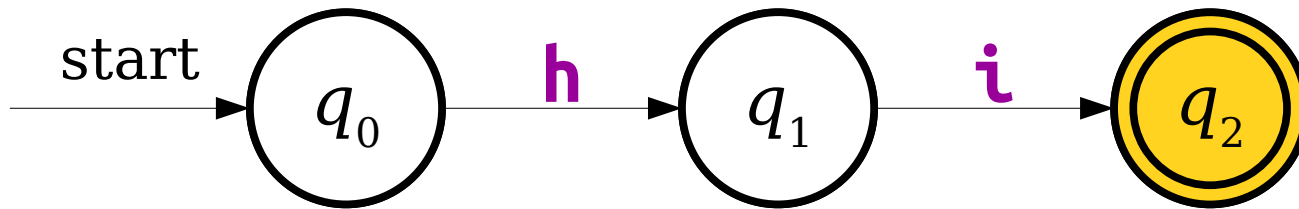
# Tragedy in Paradise



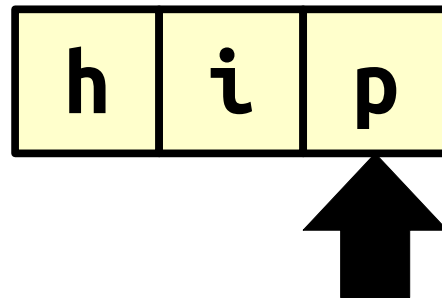
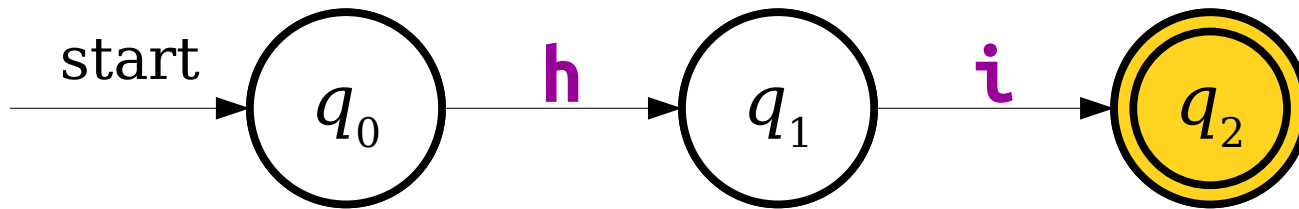
# Tragedy in Paradise



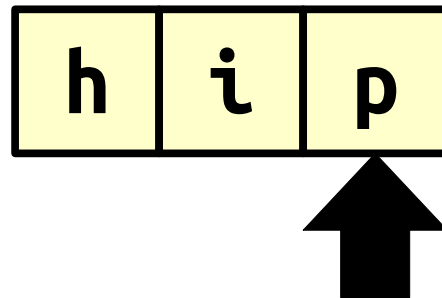
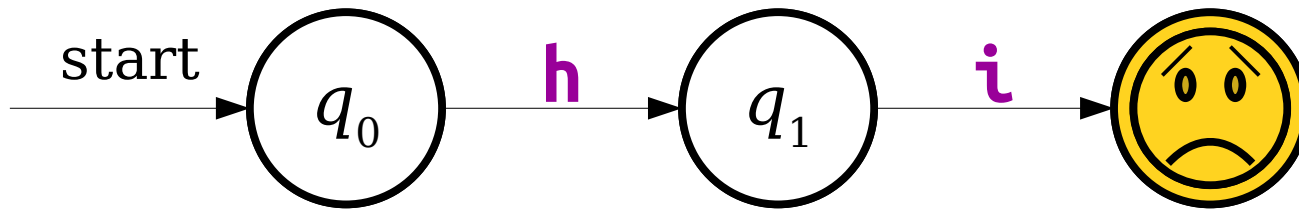
# Tragedy in Paradise



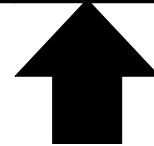
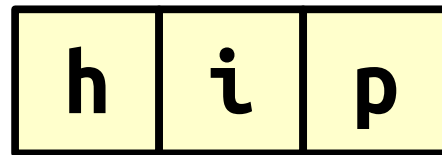
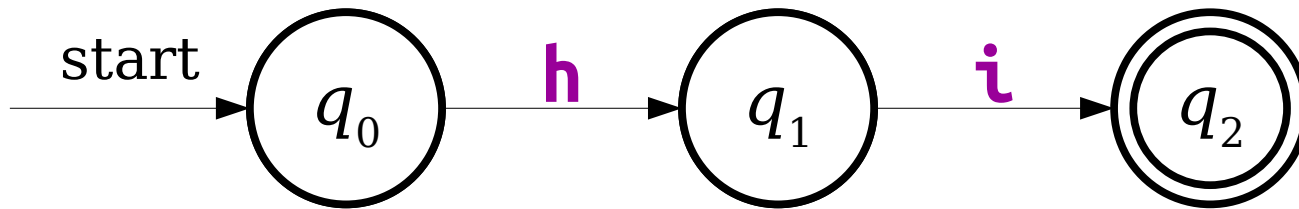
# Tragedy in Paradise



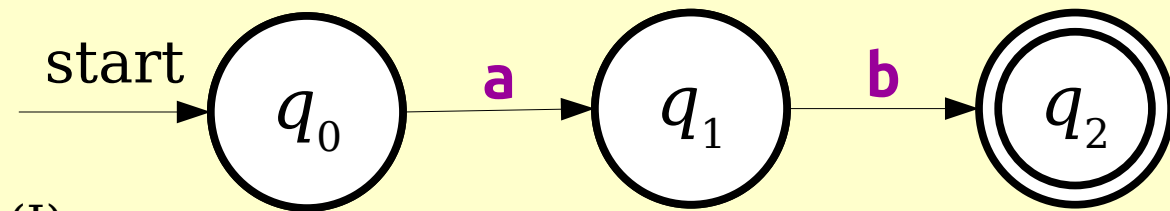
# Tragedy in Paradise



# Tragedy in Paradise





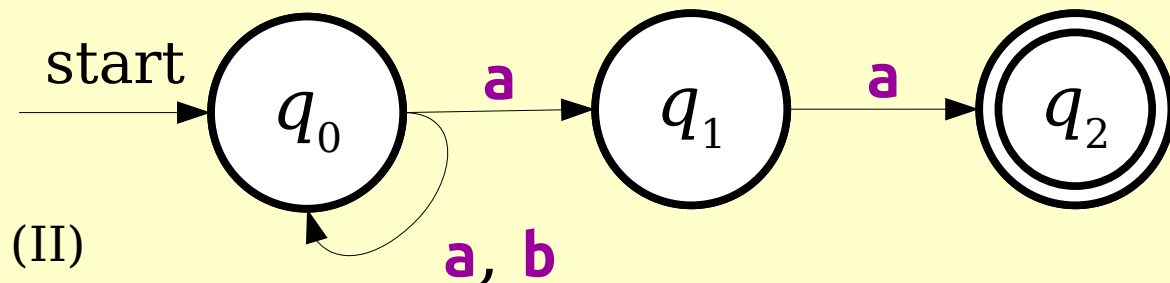


$\{ab\}$

(I)

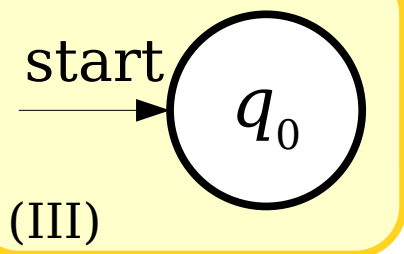
**Question to ponder:**

Why is the answer  
 $\{ w \in \Sigma^* \mid w \text{ ends in } \mathbf{aaa} \}$   
 not correct?



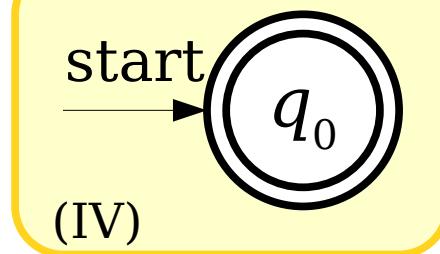
$\{ w \in \Sigma^* \mid w \text{ ends in } \mathbf{aa} \}$

(II)



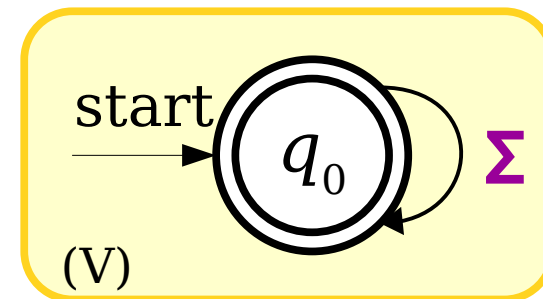
(III)

$\emptyset$



(IV)

$\{\epsilon\}$



(V)

$\Sigma^*$

The **language of an NFA** is

$$\mathcal{L}(N) = \{ w \in \Sigma^* \mid N \text{ accepts } w \}.$$

What is the language of each NFA? (Assume  $\Sigma = \{\mathbf{a}, \mathbf{b}\}$ .)

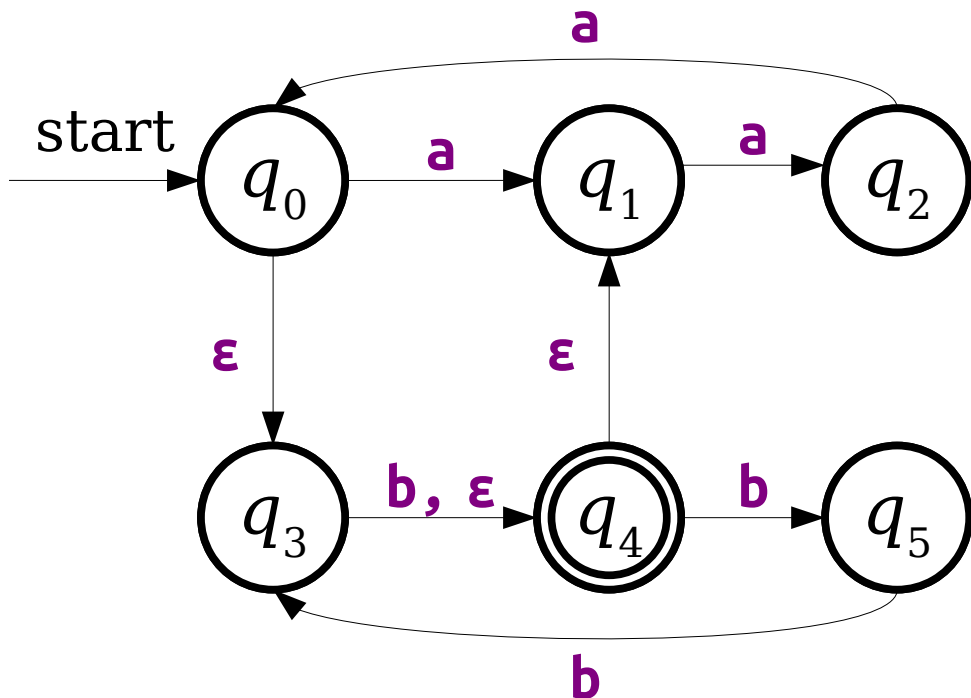
Answer at <https://cs103.stanford.edu/pollev>

# $\epsilon$ -Transitions

- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.

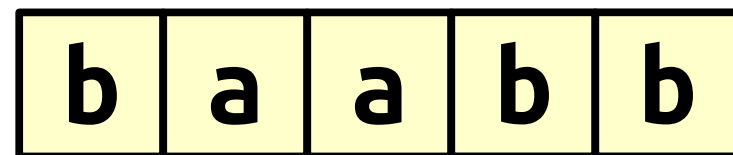
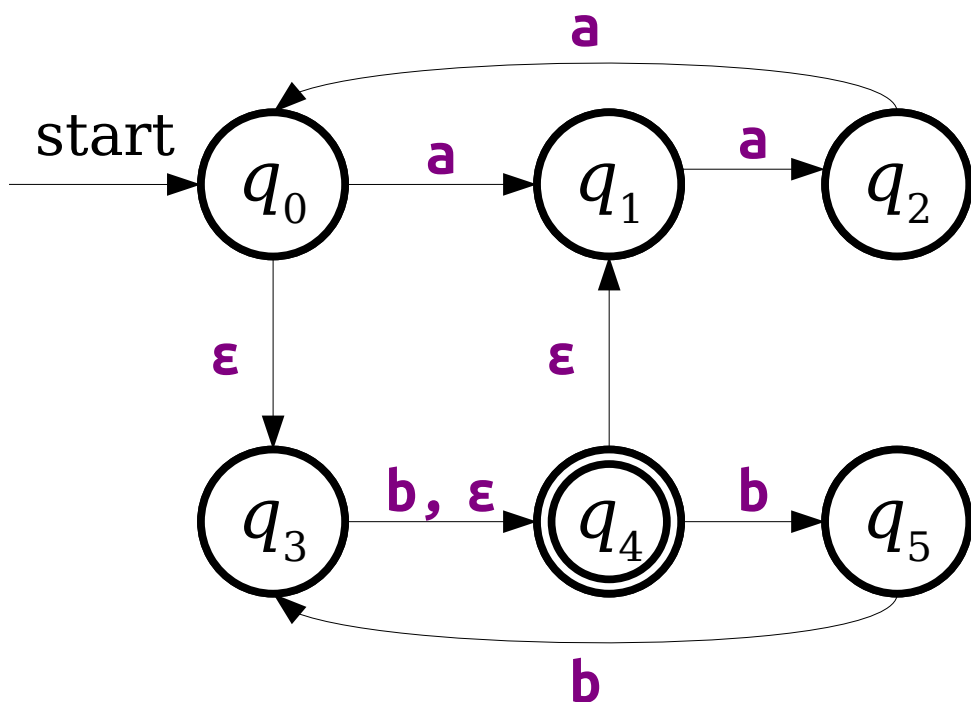
# $\epsilon$ -Transitions

- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.



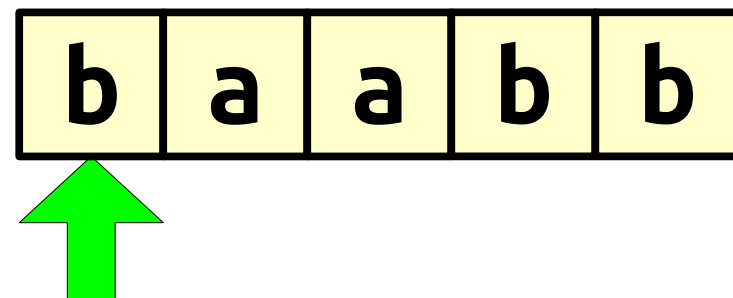
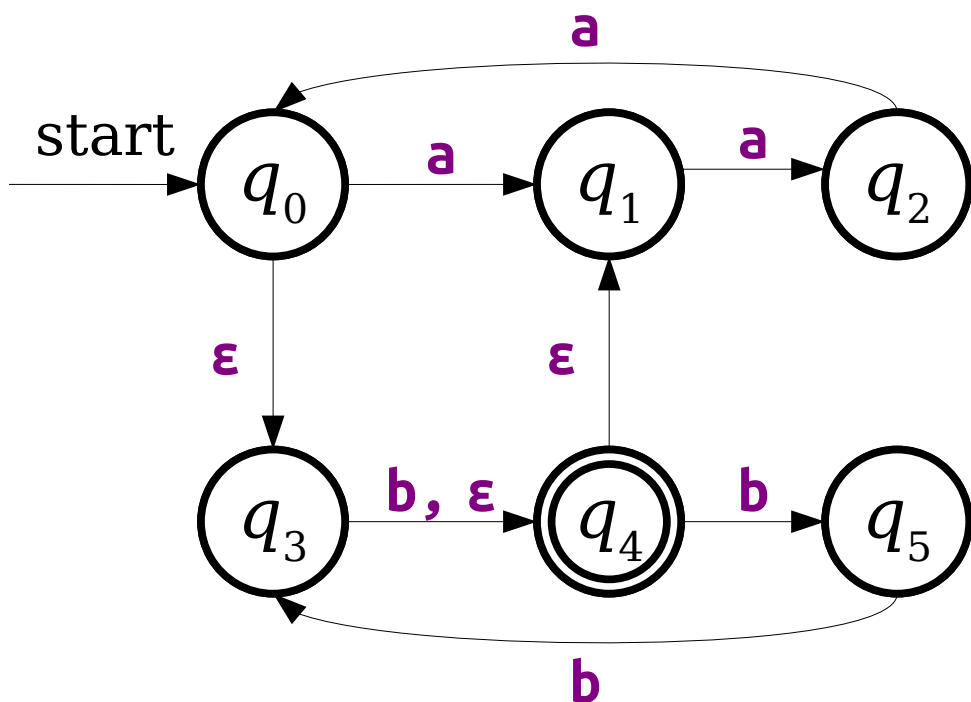
# $\epsilon$ -Transitions

- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.



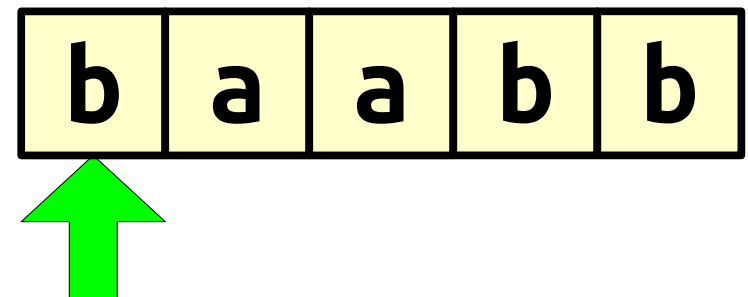
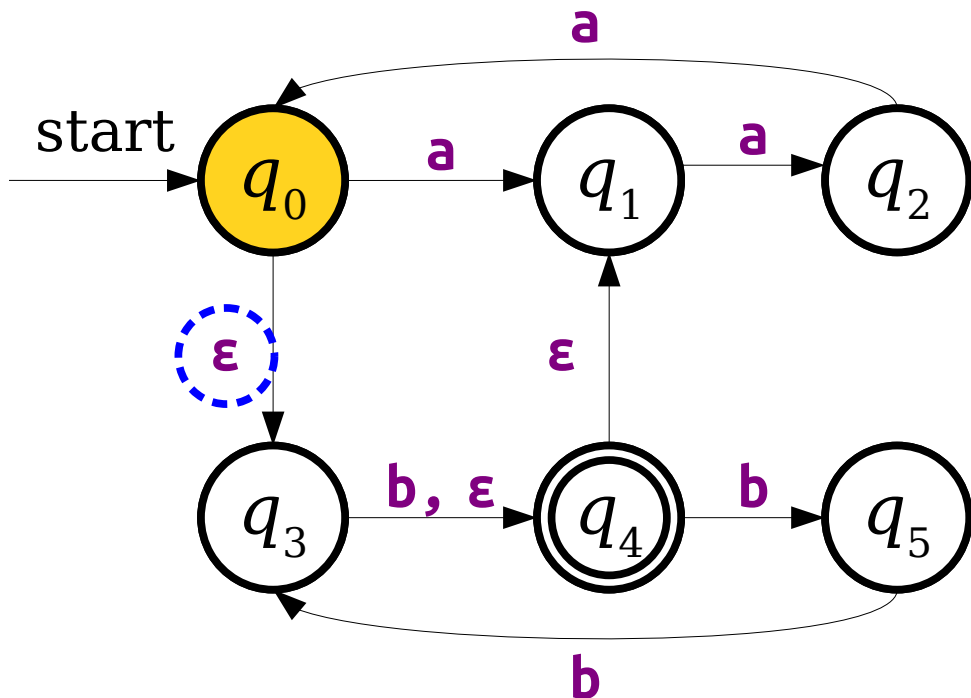
# $\epsilon$ -Transitions

- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.



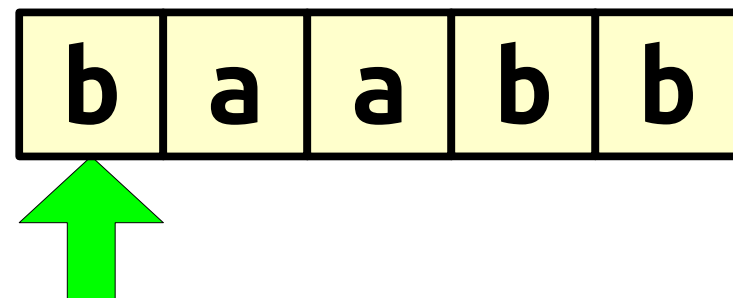
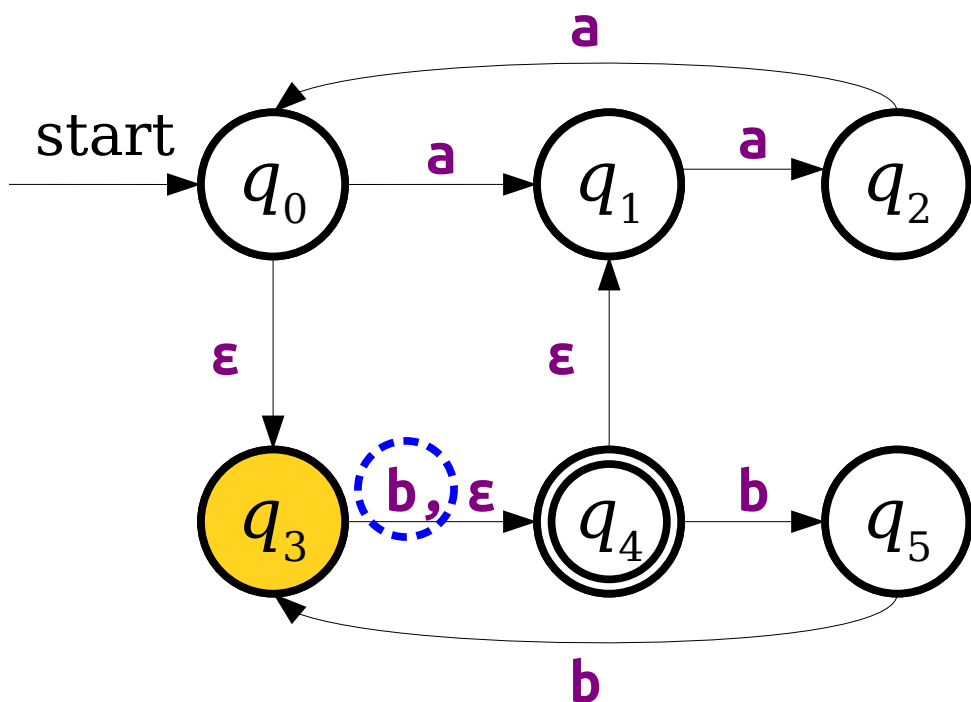
# $\epsilon$ -Transitions

- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.



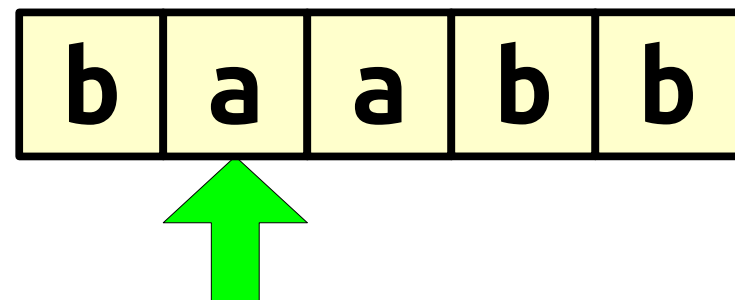
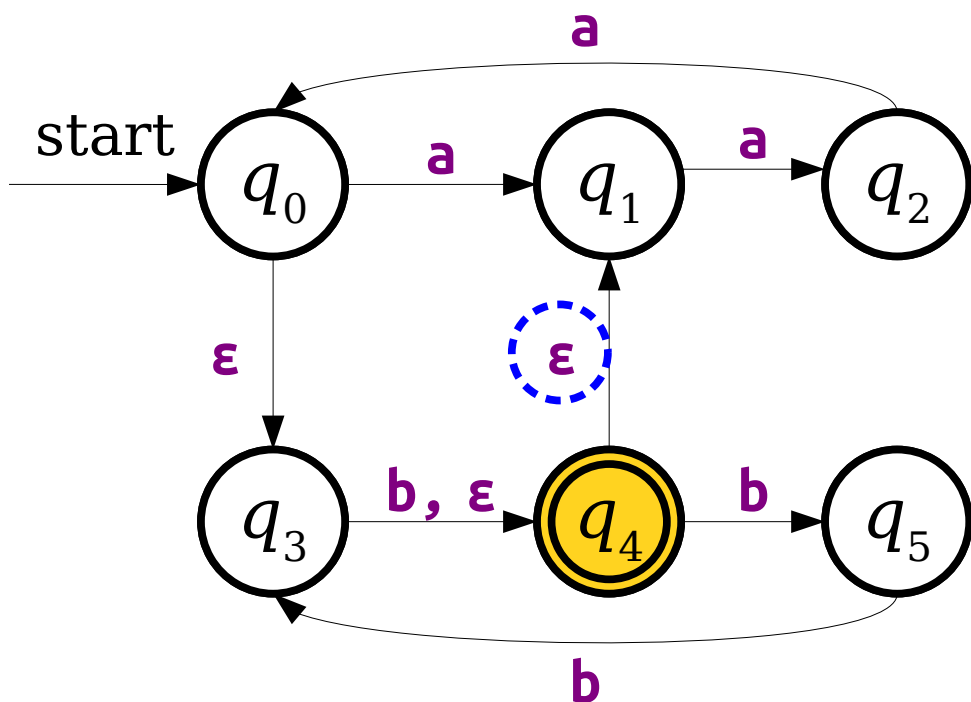
# $\epsilon$ -Transitions

- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.



# $\epsilon$ -Transitions

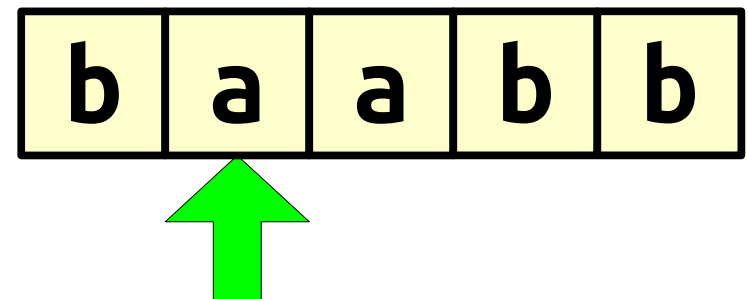
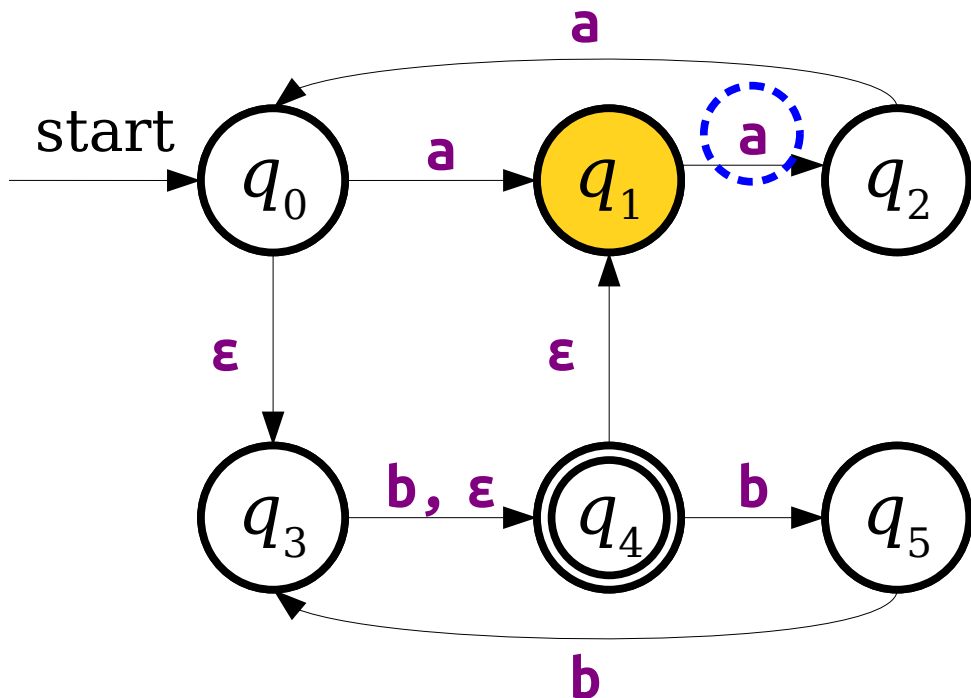
- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.





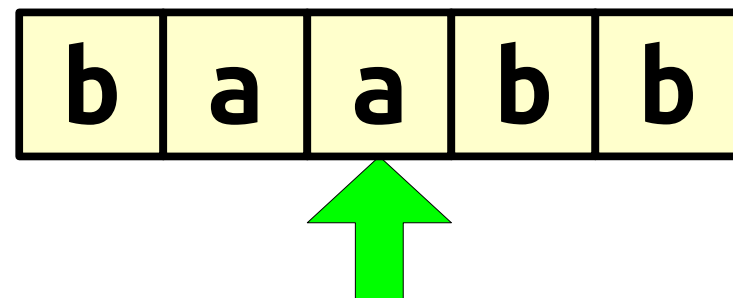
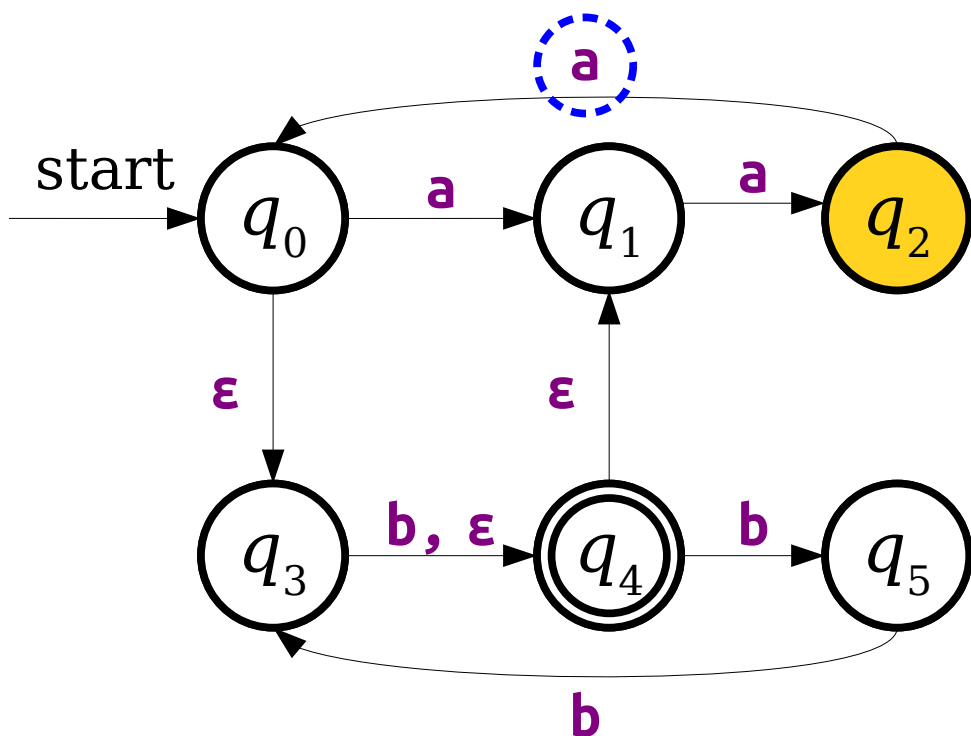
# $\epsilon$ -Transitions

- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.



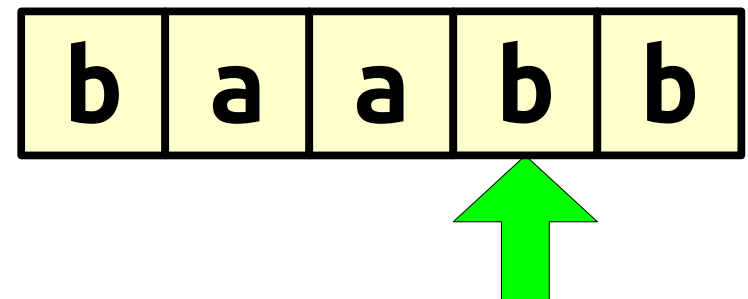
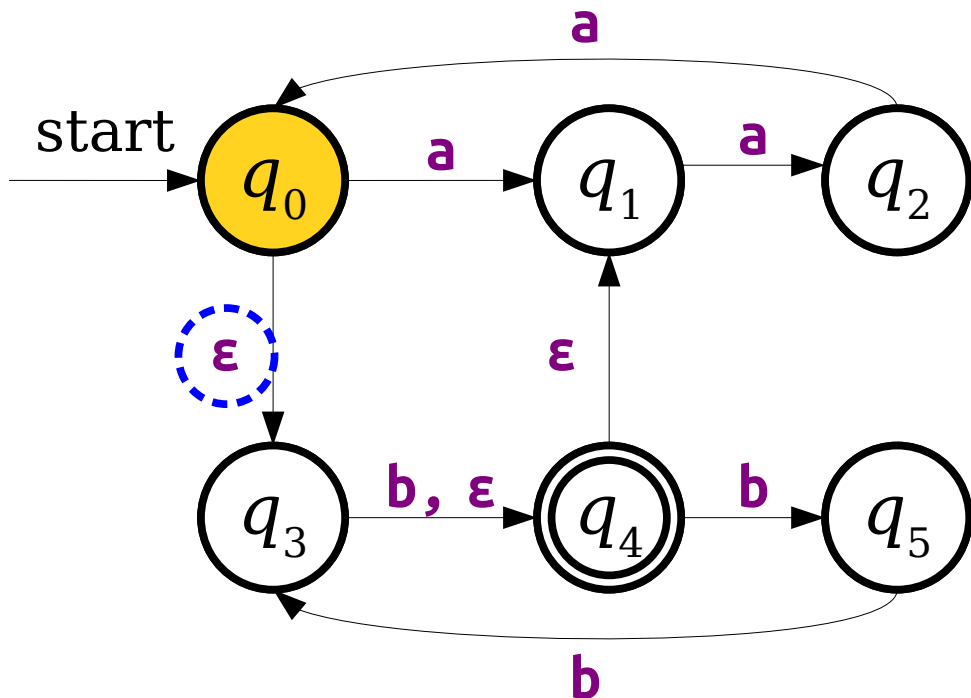
# $\epsilon$ -Transitions

- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.



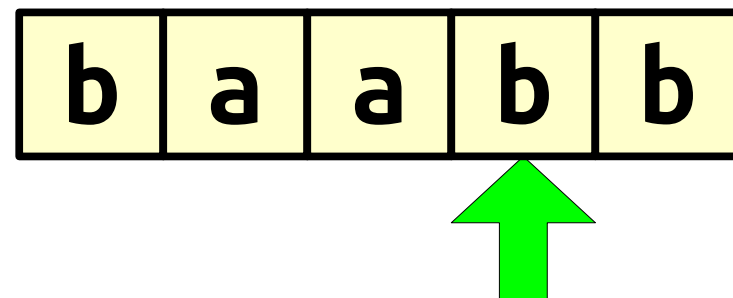
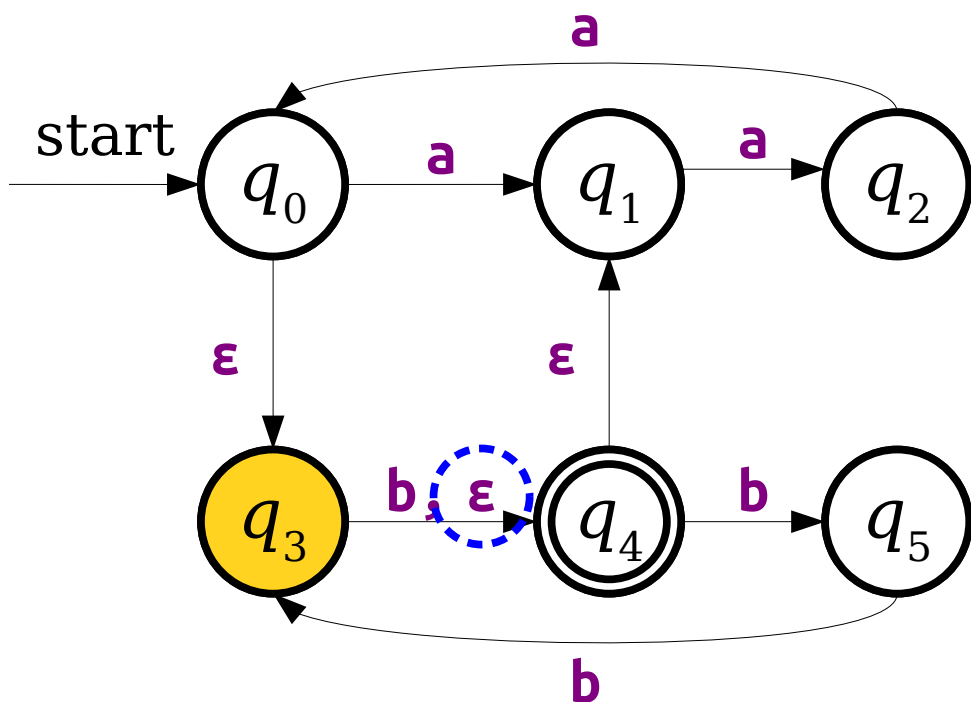
# $\epsilon$ -Transitions

- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.



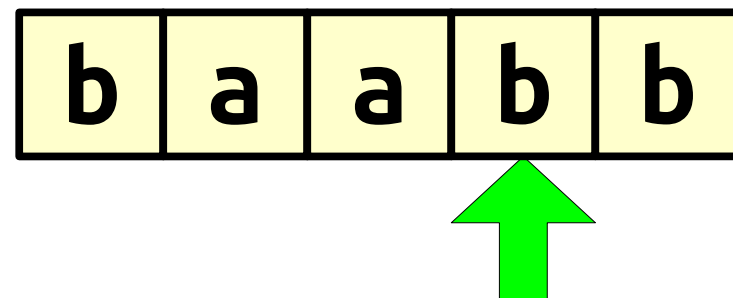
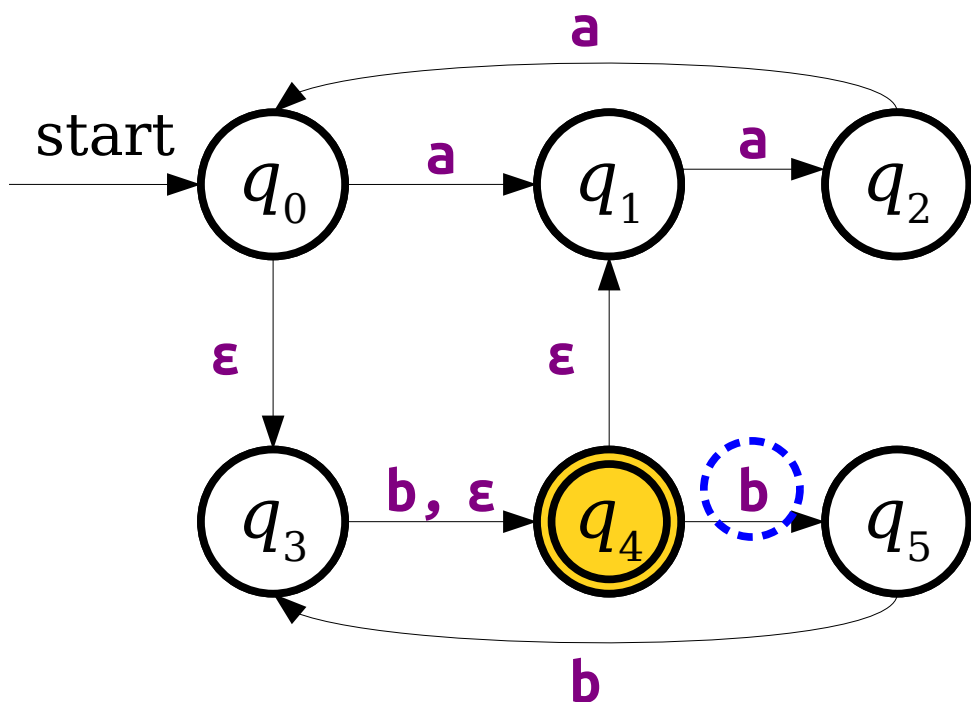
# $\epsilon$ -Transitions

- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.



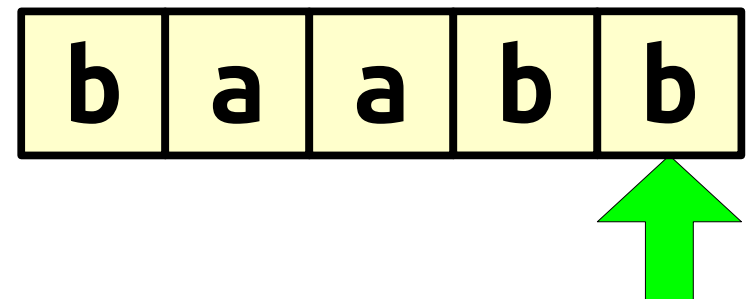
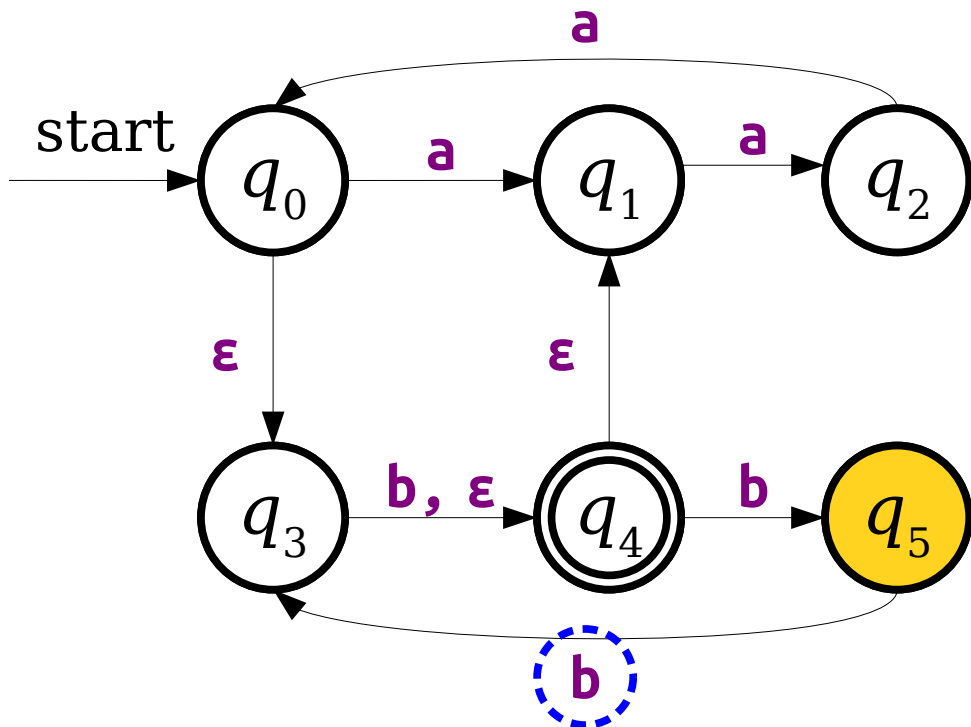
# $\epsilon$ -Transitions

- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.



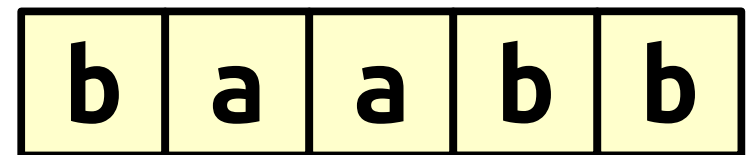
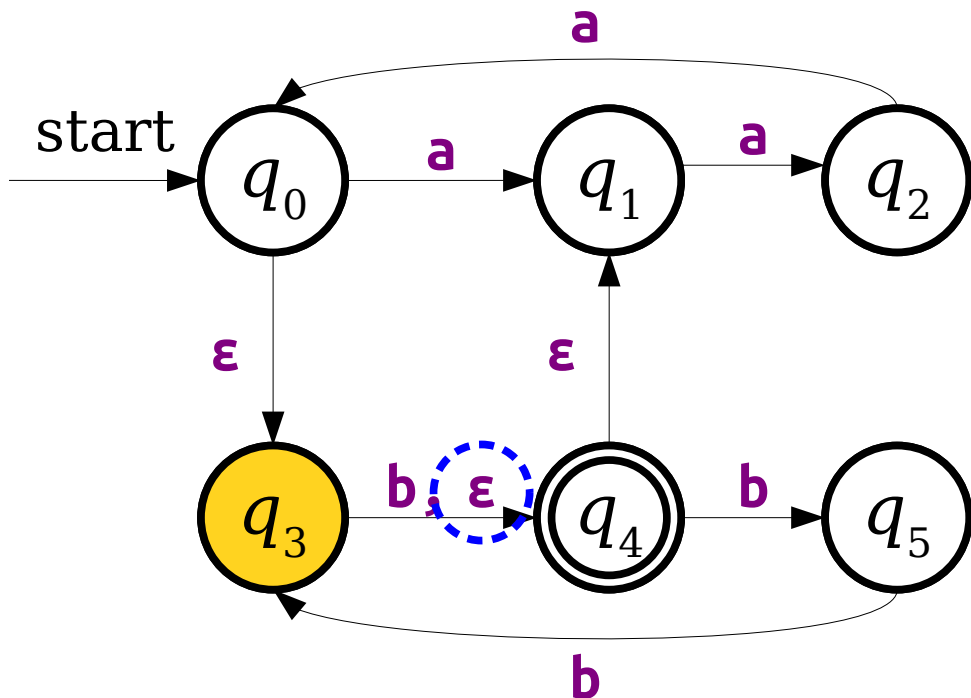
# $\epsilon$ -Transitions

- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.



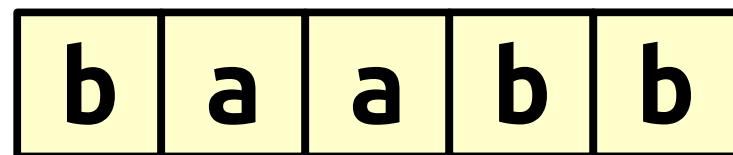
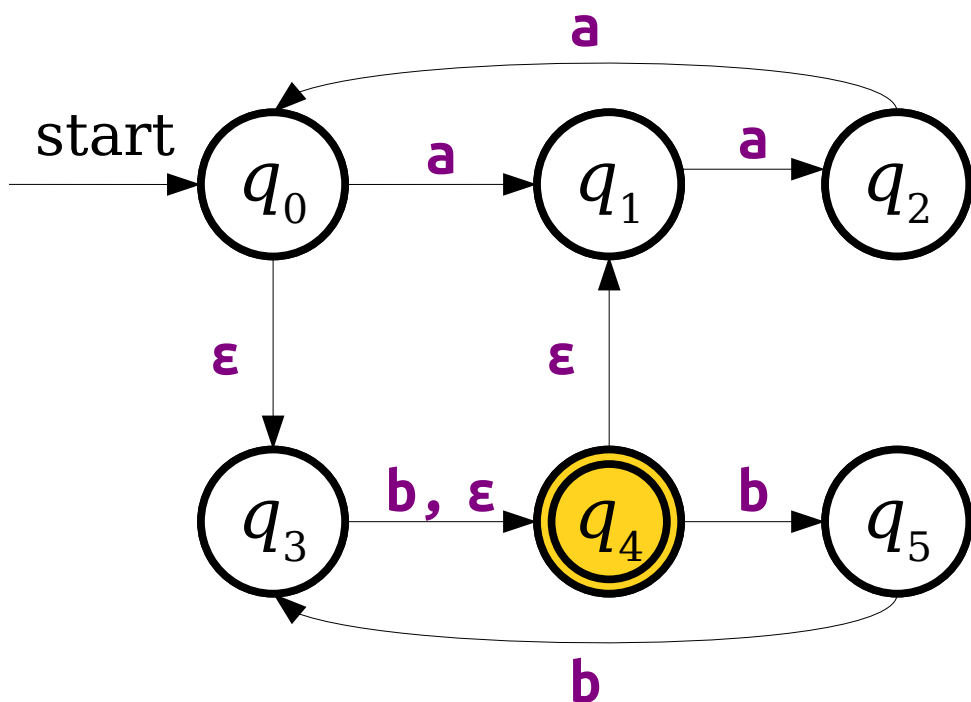
# $\epsilon$ -Transitions

- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.



# $\epsilon$ -Transitions

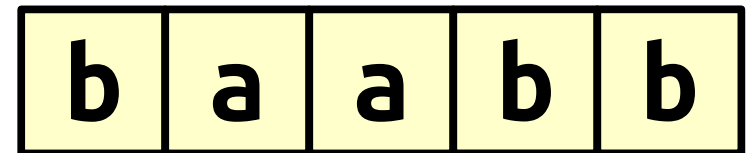
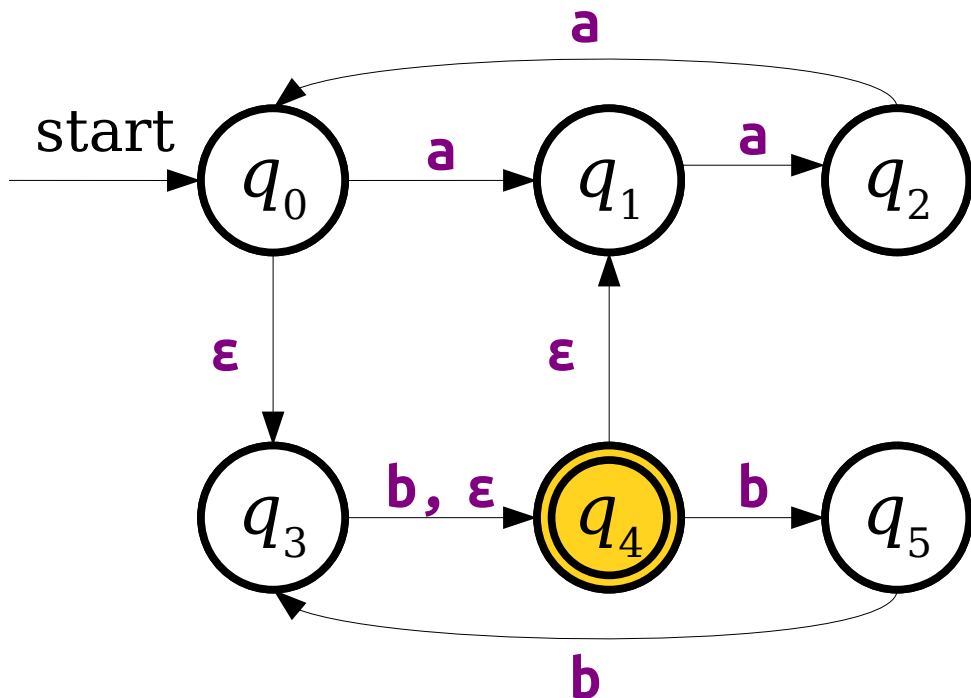
- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.





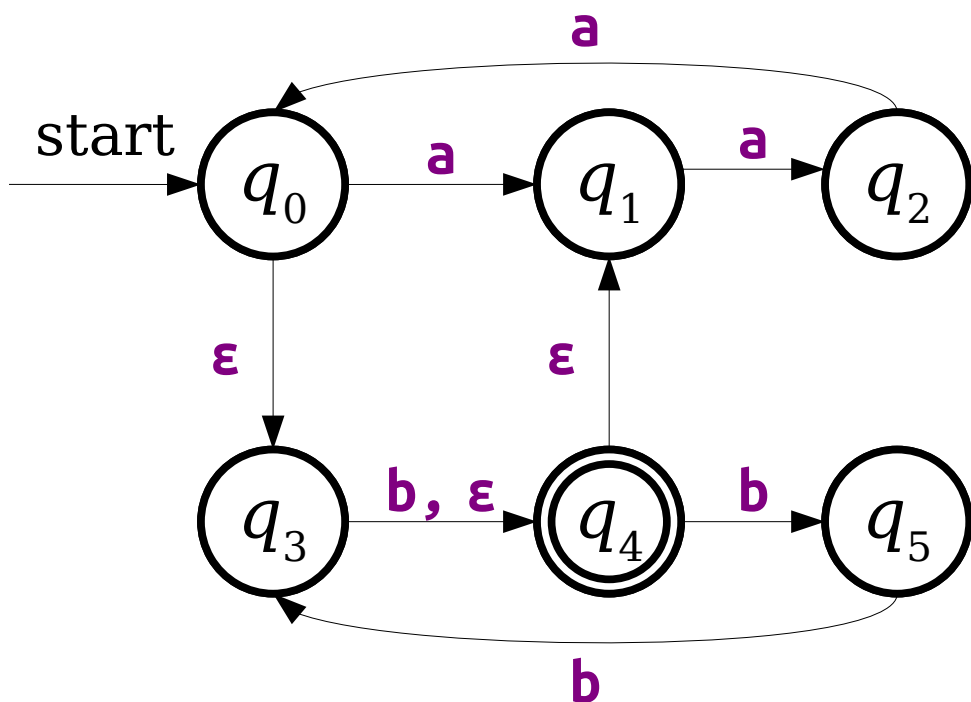
# $\epsilon$ -Transitions

- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.



# $\epsilon$ -Transitions

- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.



Not at all fun or rewarding exercise: what is the language of this NFA?

(I actually don't know the answer. I made up this NFA just to show off  $\epsilon$ -transitions.)

# $\epsilon$ -Transitions

- NFAs have a special type of transition called the  **$\epsilon$ -transition**.
- An NFA may follow any number of  $\epsilon$ -transitions at any time without consuming any input.
- NFAs are not *required* to follow  $\epsilon$ -transitions. It's simply another option at the machine's disposal.

# NFAs

- An NFA is defined relative to some alphabet  $\Sigma$ .
- For each state in the NFA, there may be *any number of* transitions defined for each symbol in  $\Sigma$ , plus any number of  $\epsilon$ -transitions.
  - This is the “nondeterministic” part of NFA.
- There is a unique start state.
- There are zero or more accepting states.

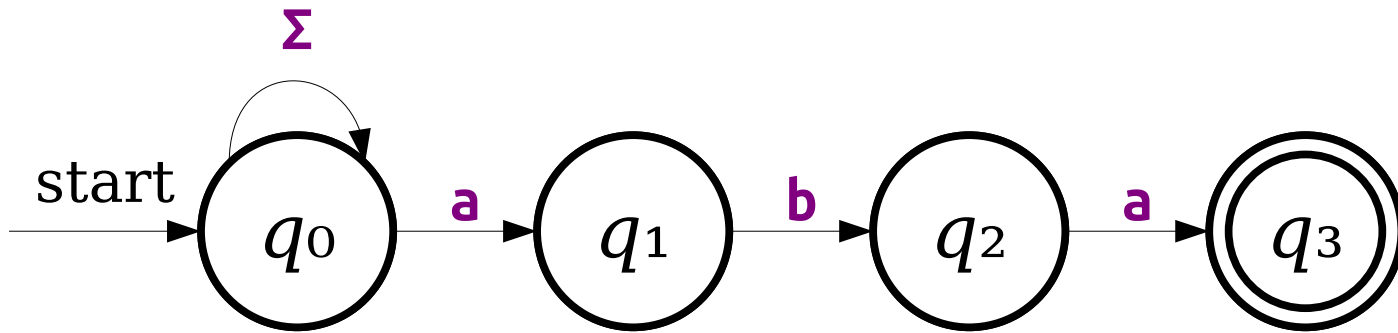
# DFA's

- A DFA is defined relative to some alphabet  $\Sigma$ .
- For each state in the DFA, there must be *exactly one* transition defined for each symbol in  $\Sigma$ . Additionally,  $\epsilon$ -transitions are not allowed.
  - This is the “deterministic” part of DFA.
- There is a unique start state.
- There are zero or more accepting states.

Thinking about Nondeterminism

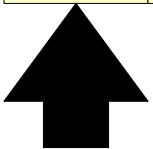
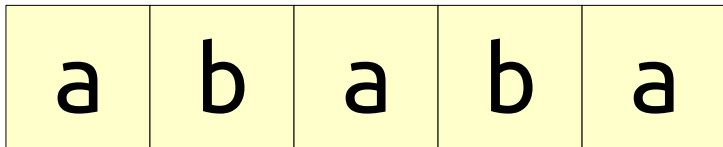
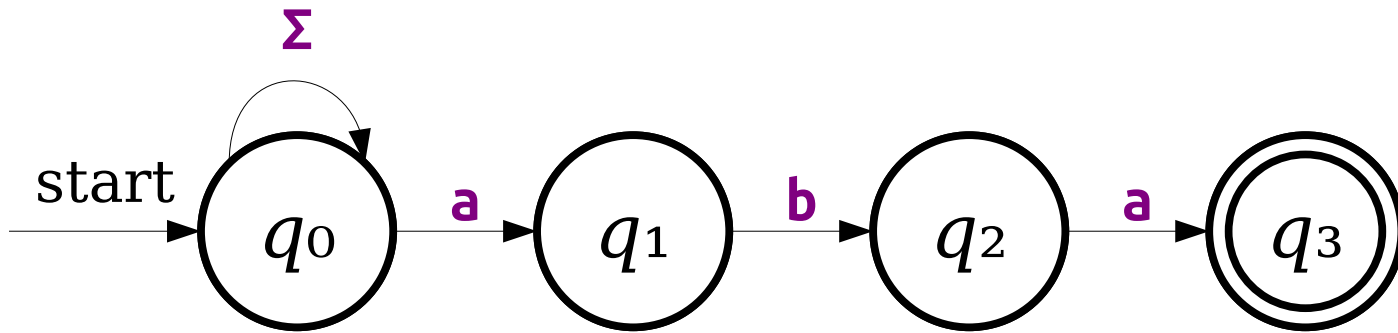
***Intuition 1:*** Perfect Positive Guessing

# Perfect Positive Guessing

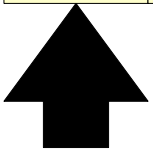
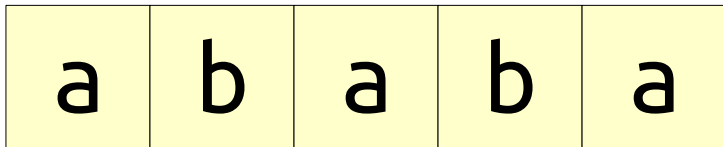
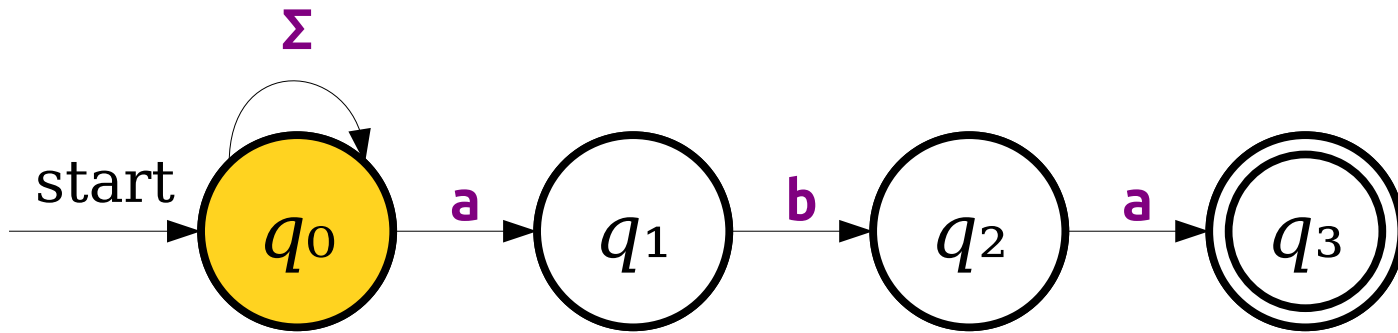




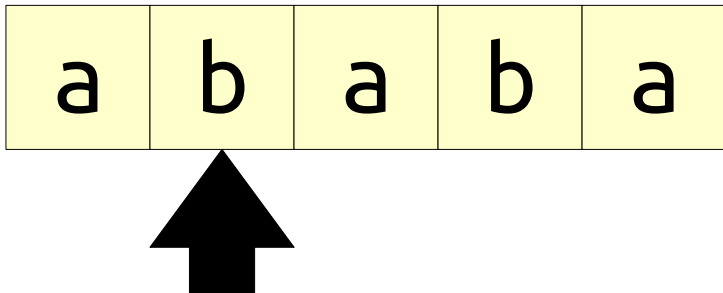
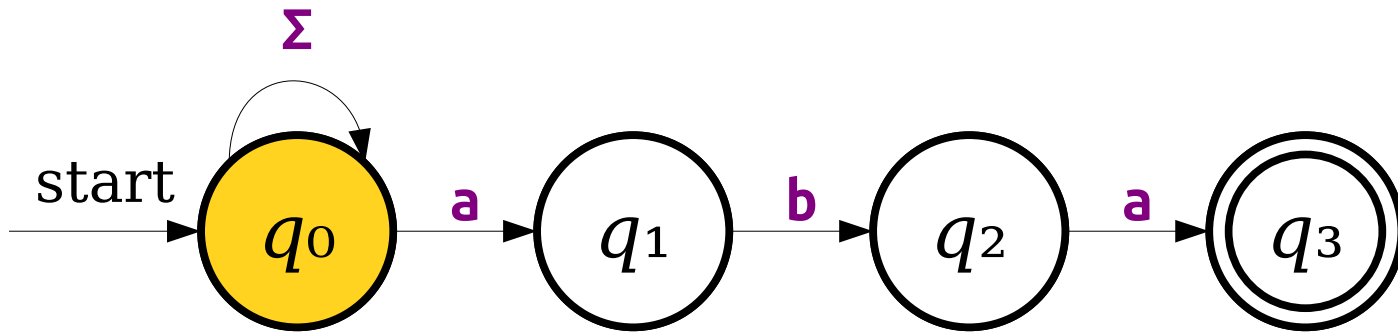
# Perfect Positive Guessing



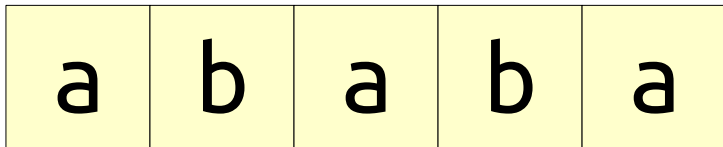
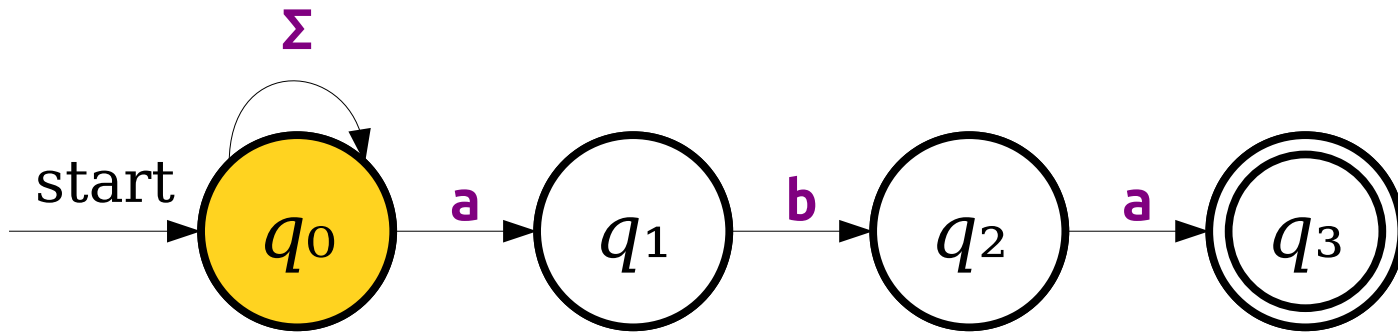
# Perfect Positive Guessing



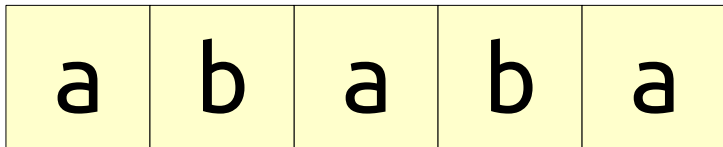
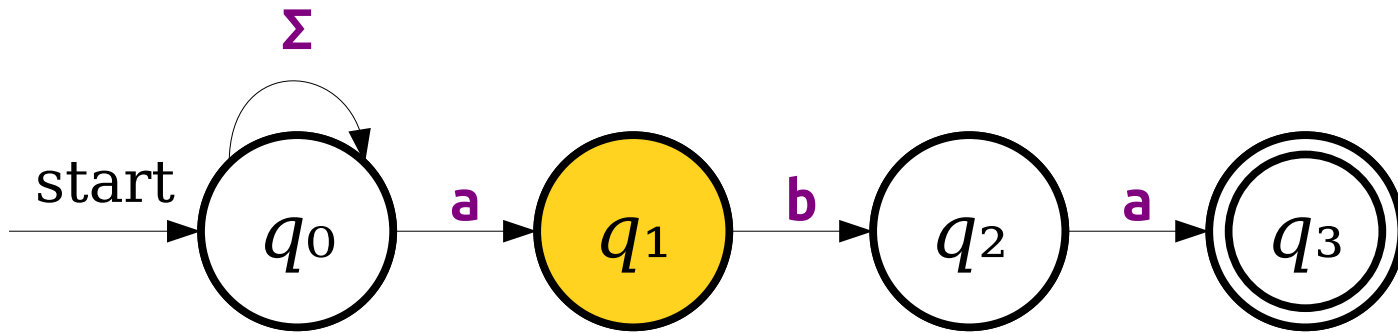
# Perfect Positive Guessing



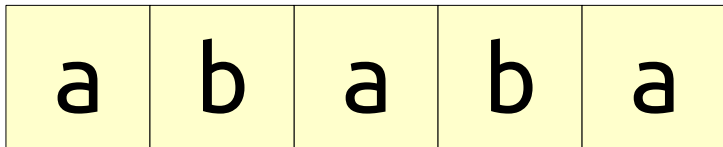
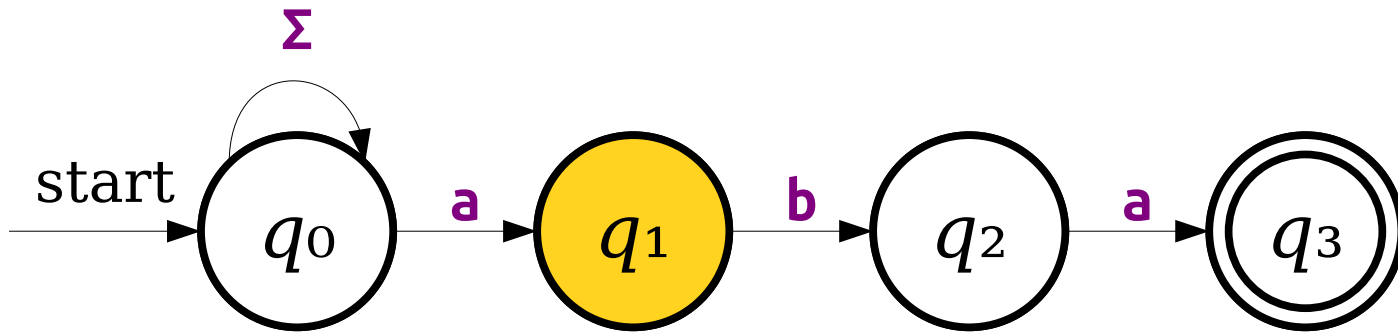
# Perfect Positive Guessing



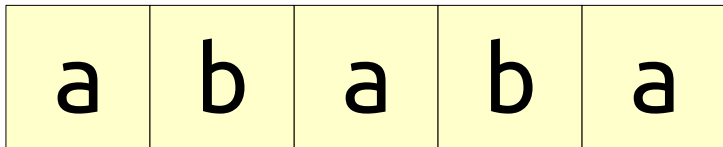
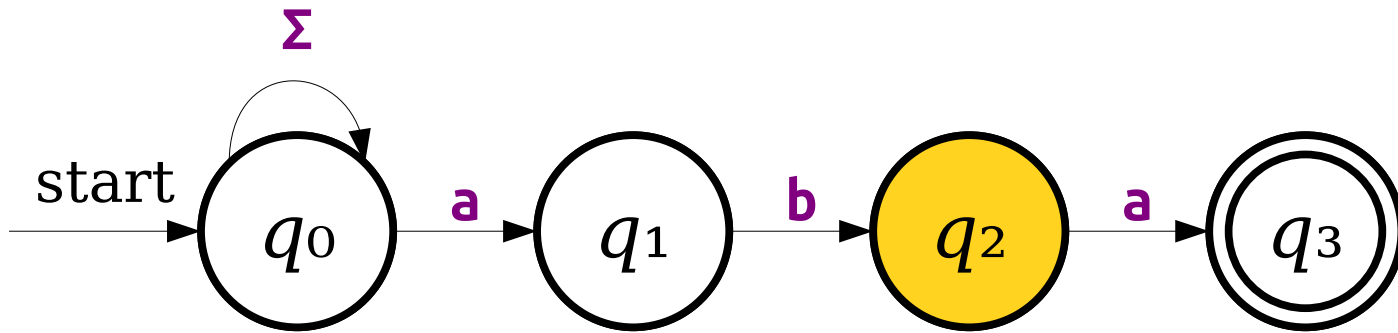
# Perfect Positive Guessing



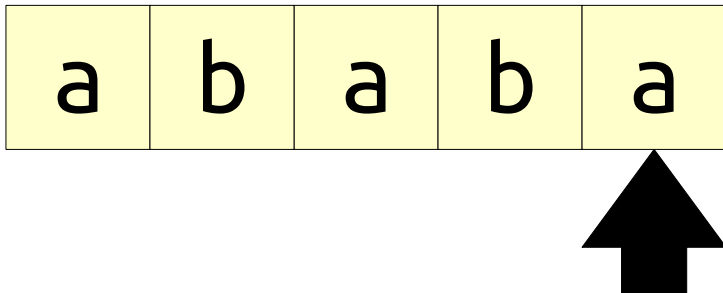
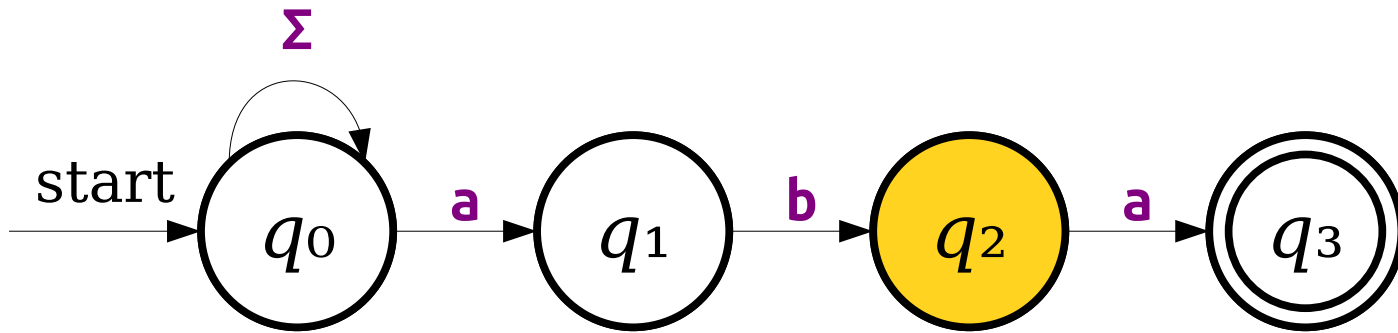
# Perfect Positive Guessing



# Perfect Positive Guessing

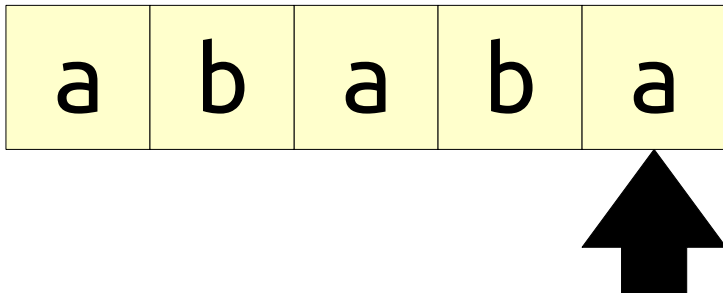
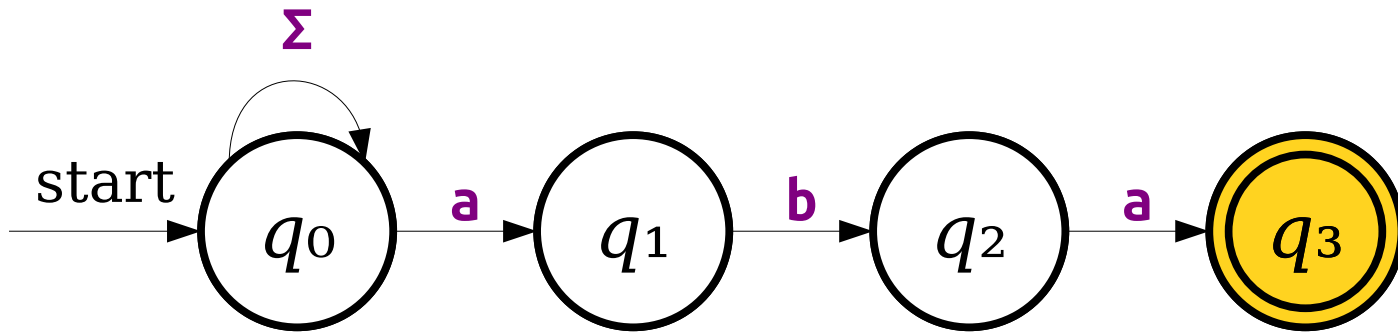


# Perfect Positive Guessing

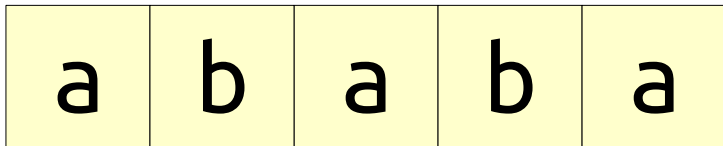
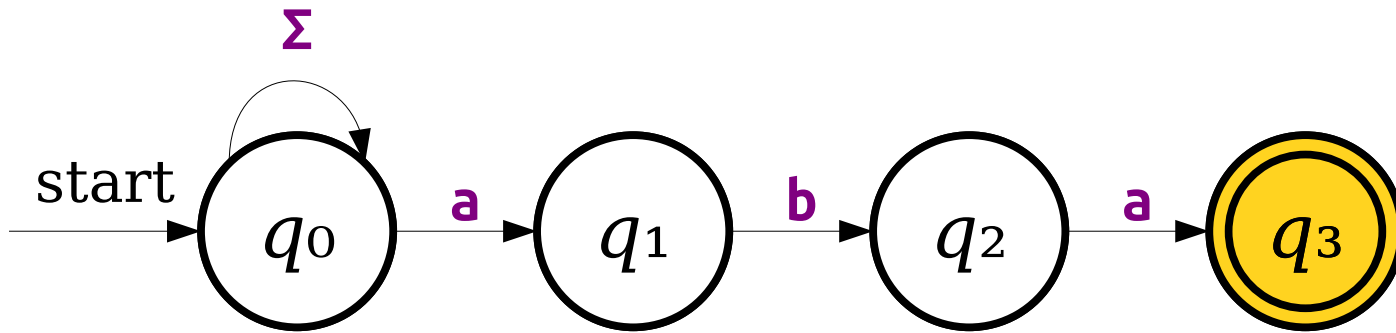




# Perfect Positive Guessing



# Perfect Positive Guessing

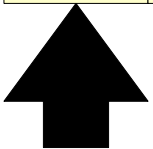
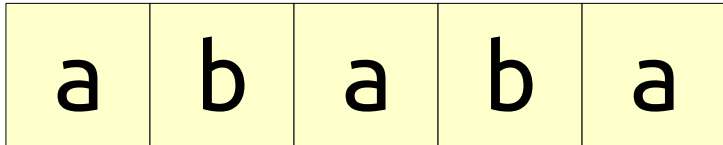
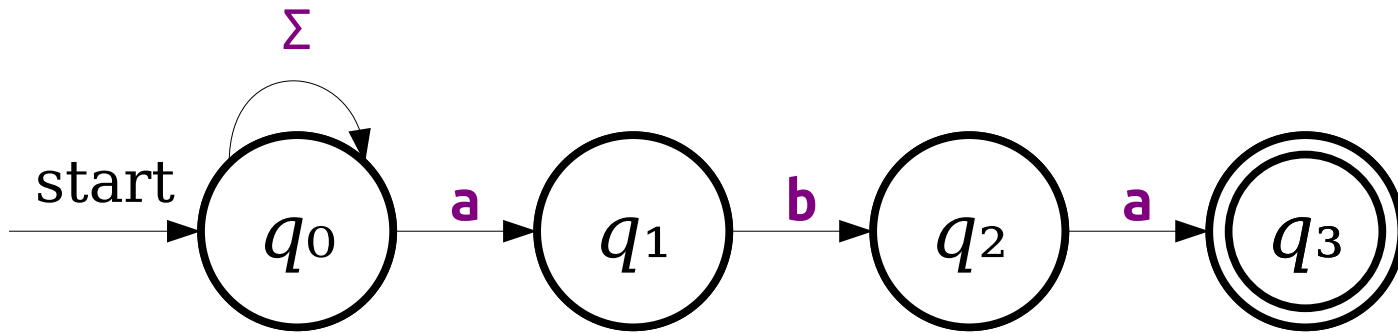


# Perfect Positive Guessing

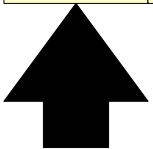
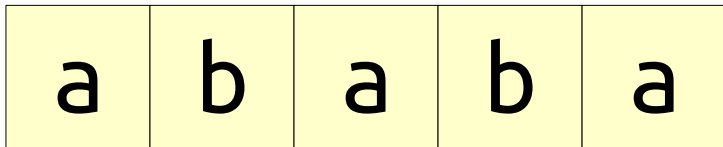
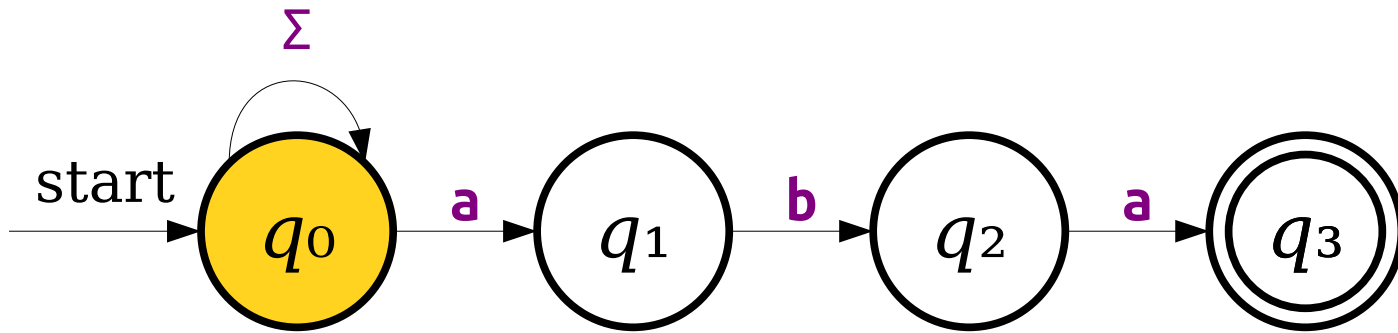
- We can view nondeterministic machines as having ***Magic Superpowers*** that enable them to guess choices that lead to an accepting state.
  - If there is at least one choice that leads to an accepting state, the machine will guess it.
  - If there are no choices, the machine guesses any one of the wrong guesses.
- There is no known way to physically model this intuition of nondeterminism – this is quite a departure from reality!
- (And no, this is not the same as a quantum computer. Come talk to me after class to learn why!)

***Intuition 2:*** Massive Parallelism

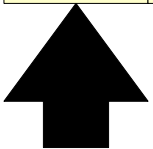
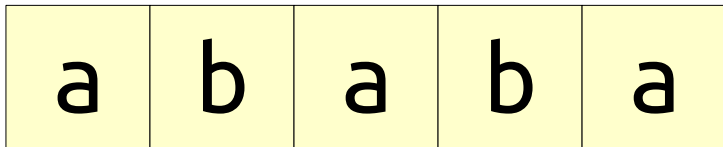
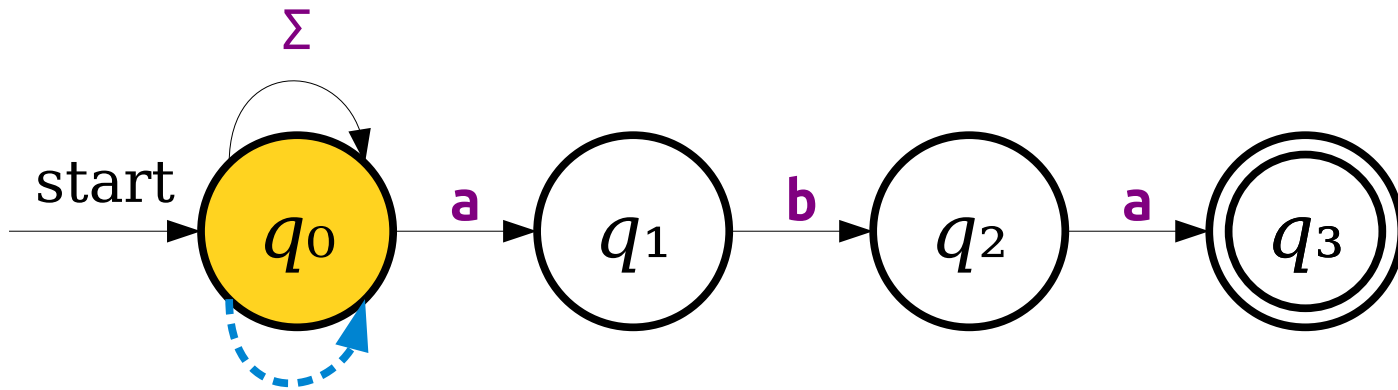
# Massive Parallelism



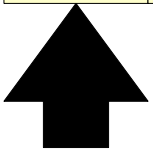
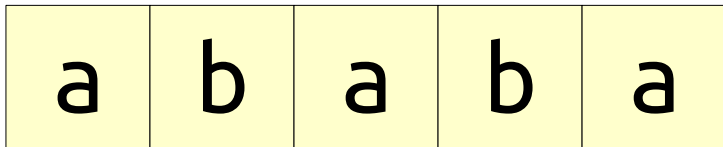
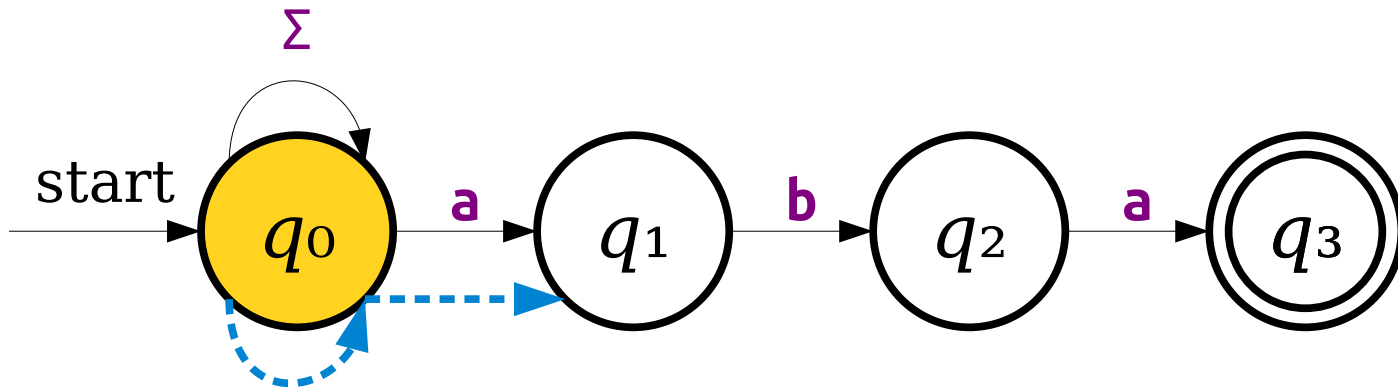
# Massive Parallelism



# Massive Parallelism

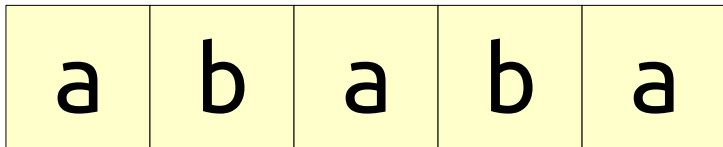
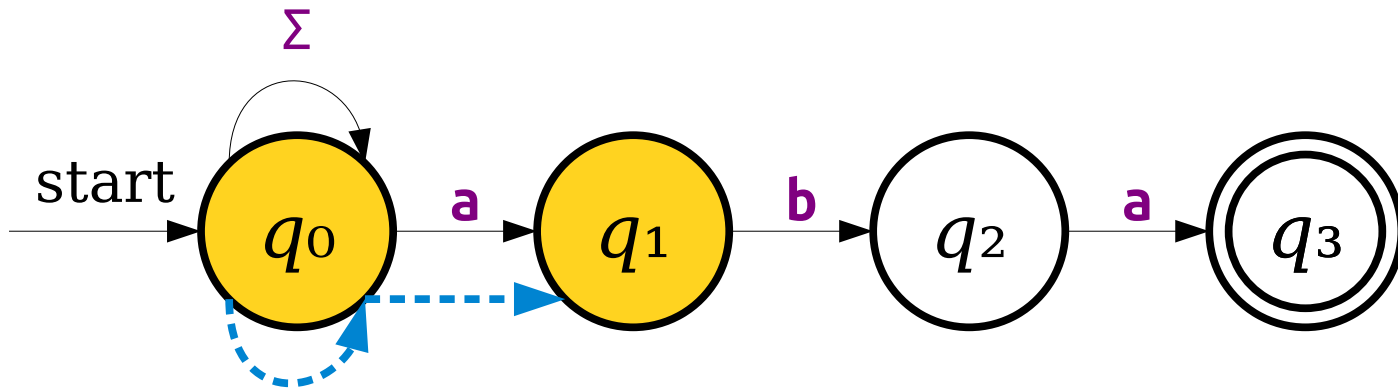


# Massive Parallelism

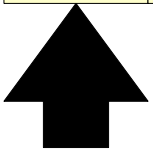
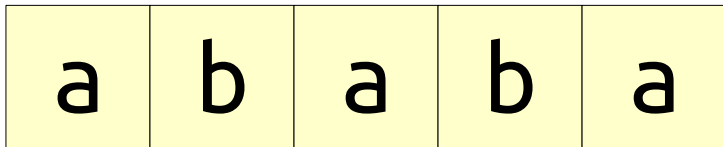
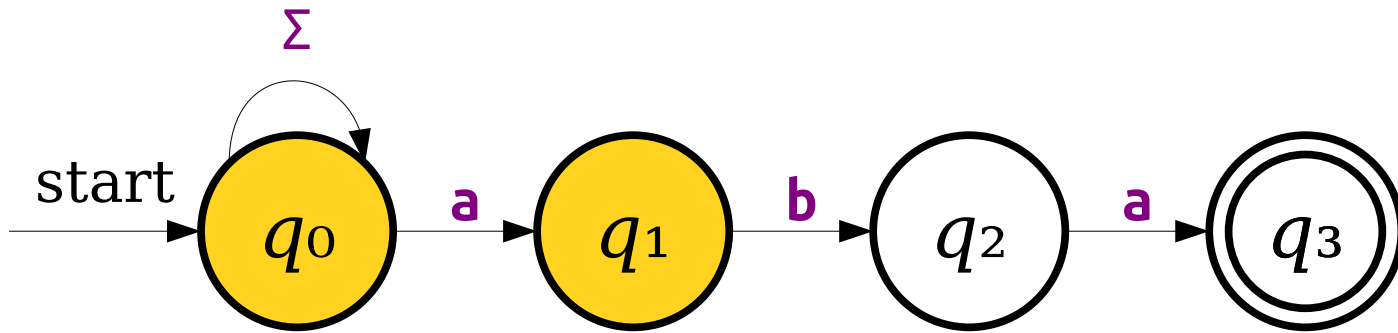




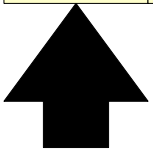
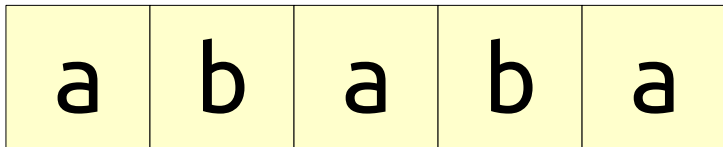
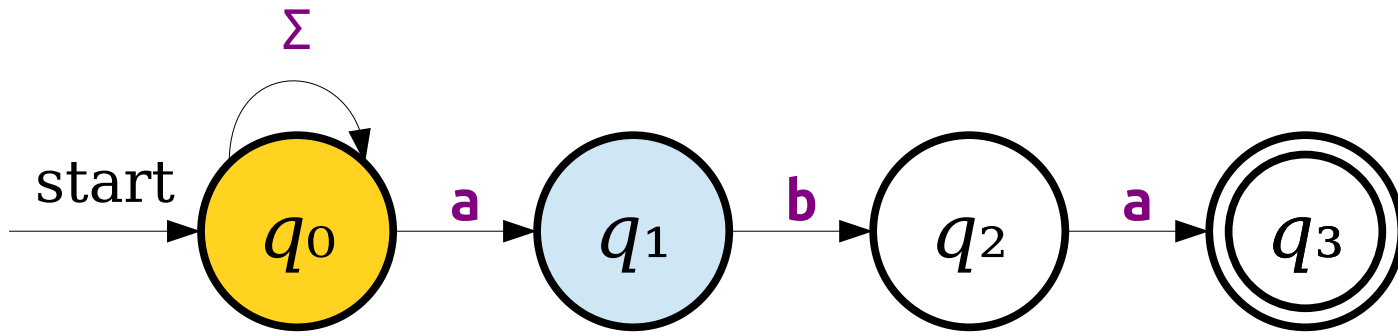
# Massive Parallelism



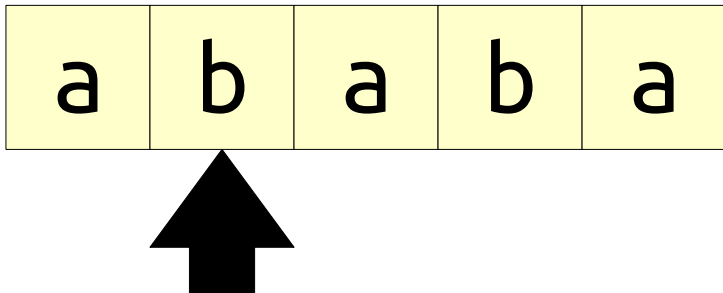
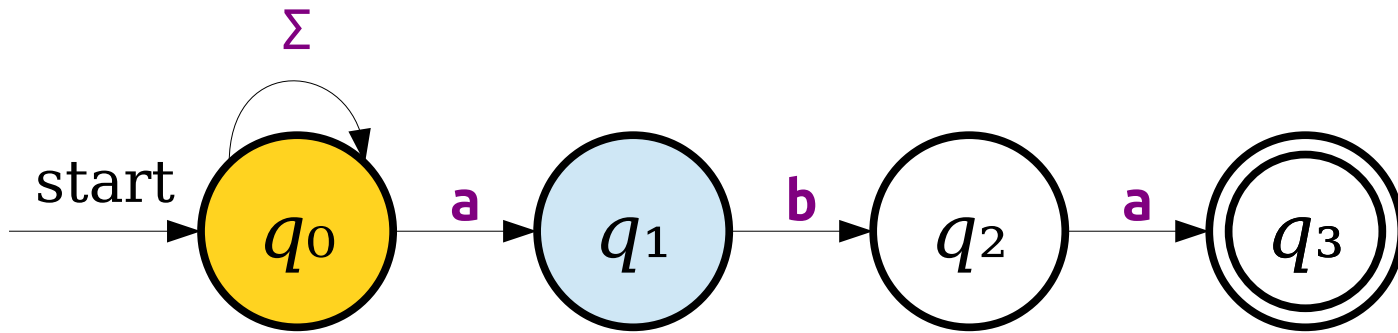
# Massive Parallelism



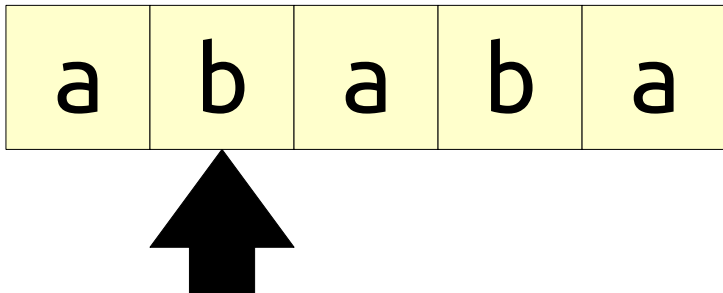
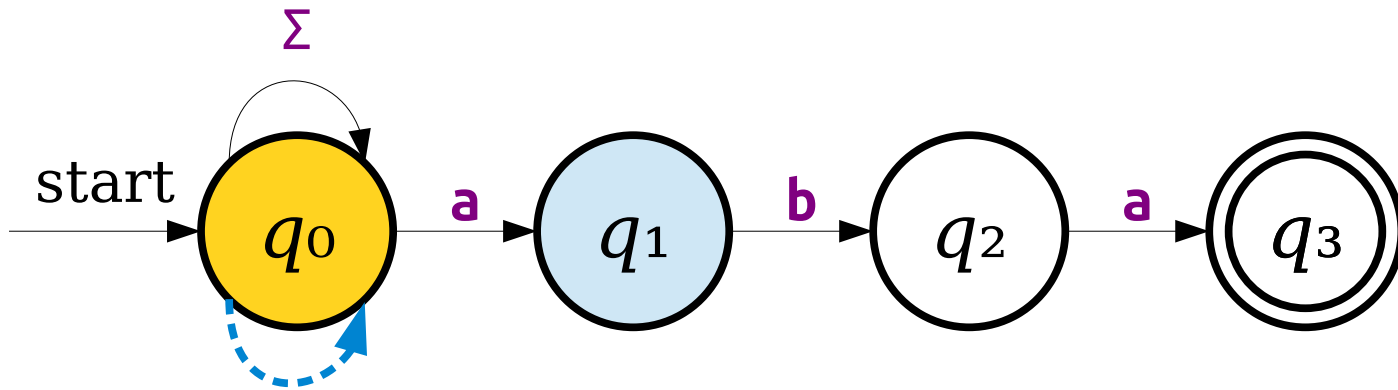
# Massive Parallelism



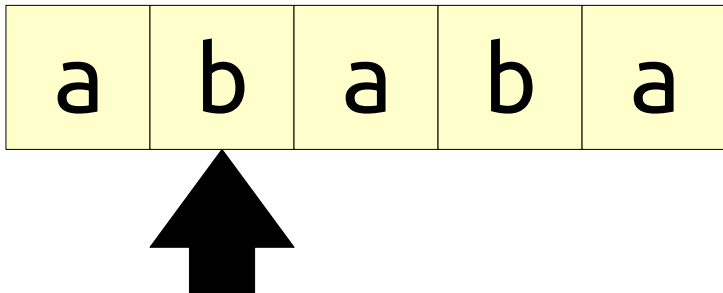
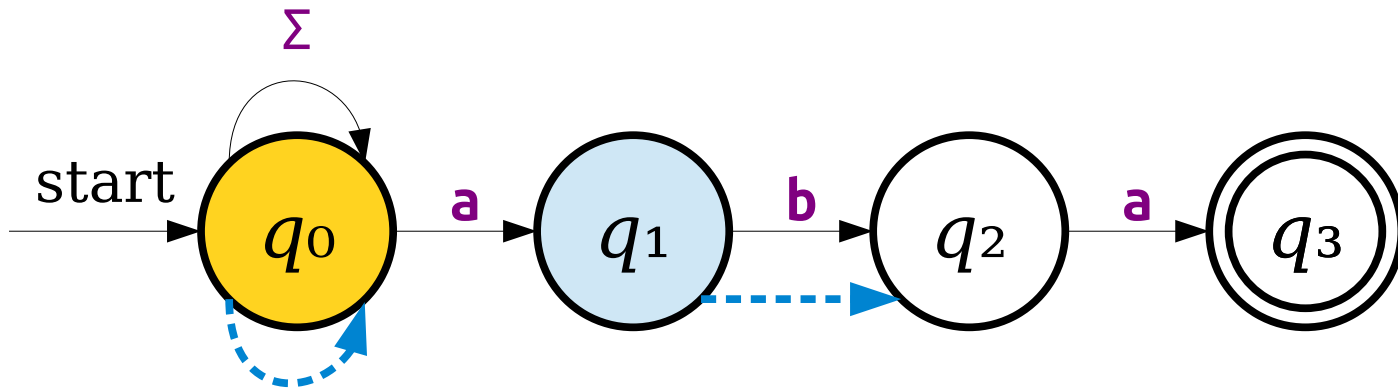
# Massive Parallelism



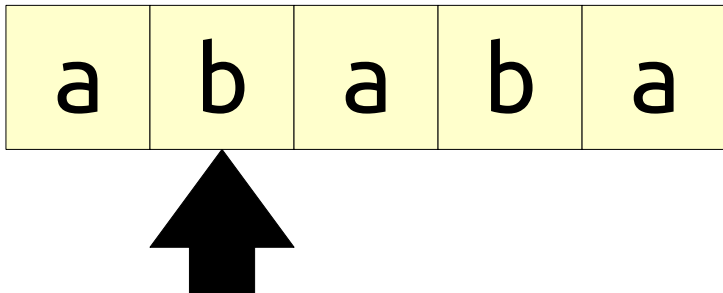
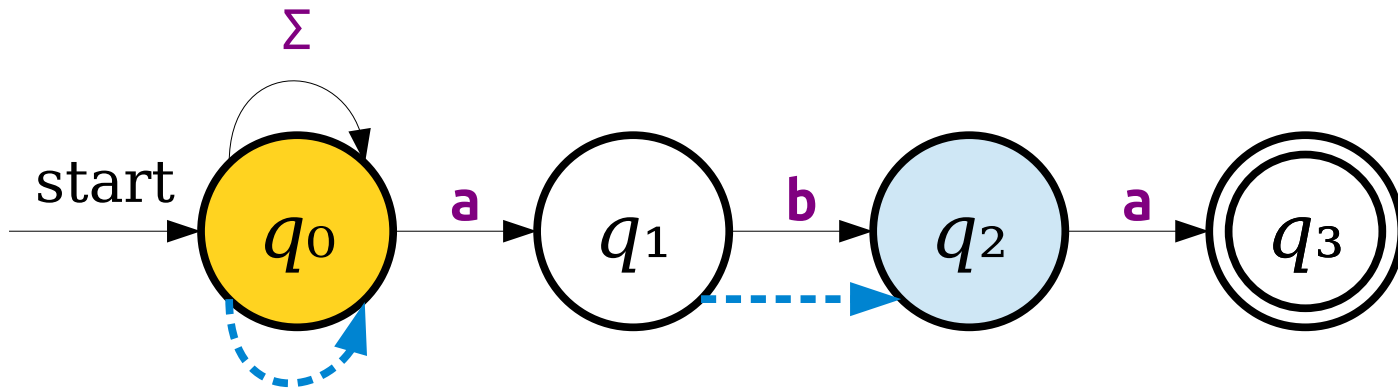
# Massive Parallelism



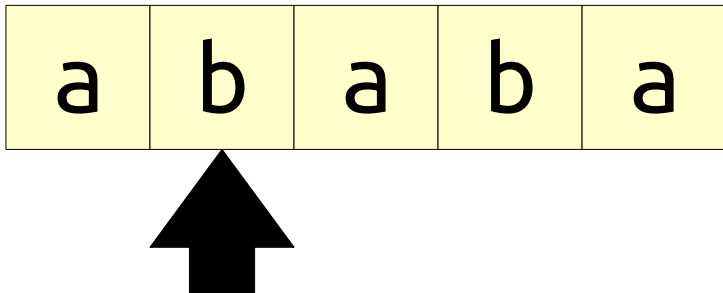
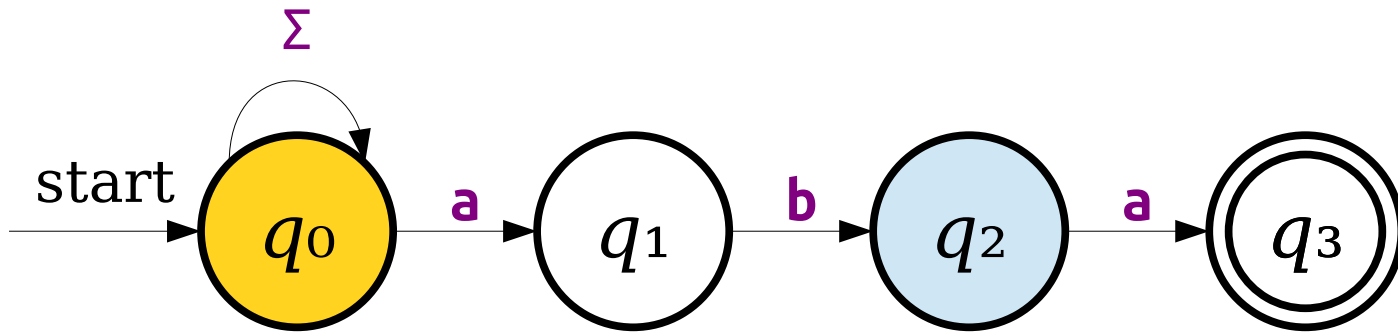
# Massive Parallelism



# Massive Parallelism

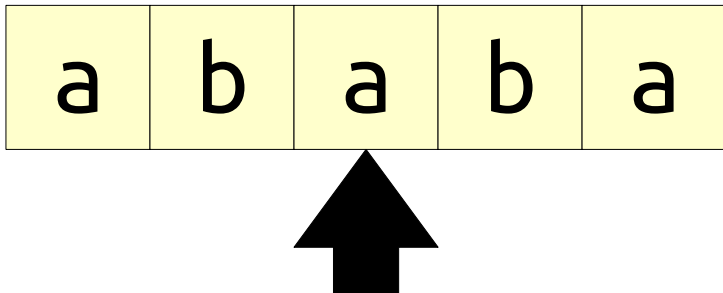
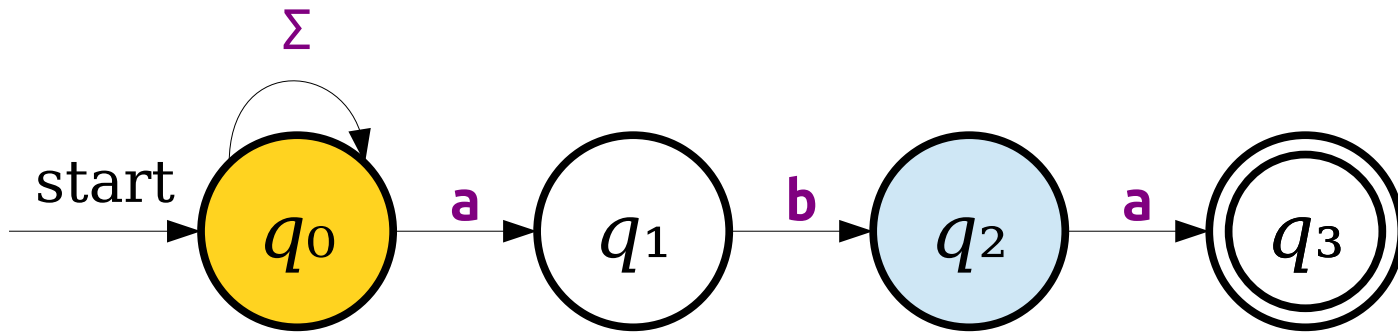


# Massive Parallelism

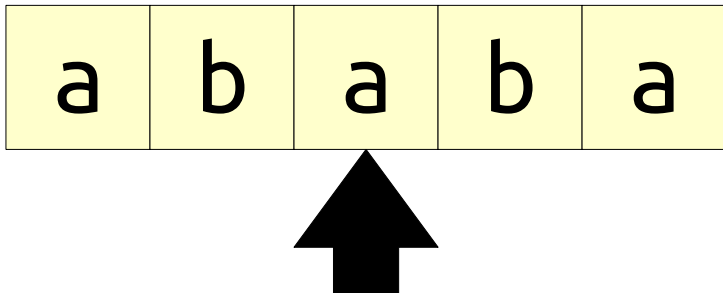
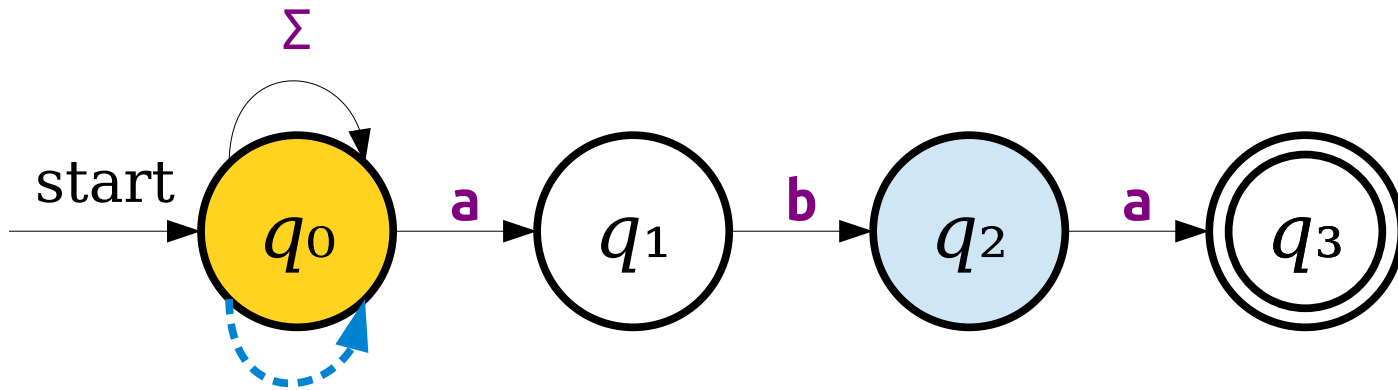




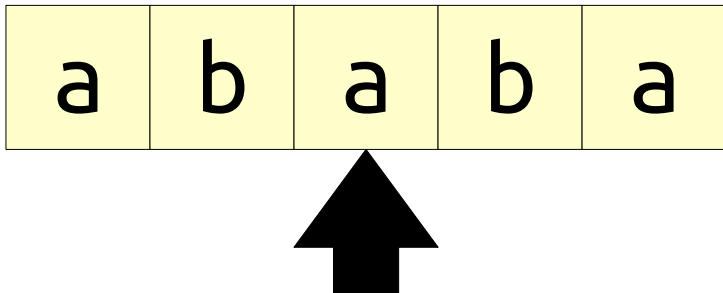
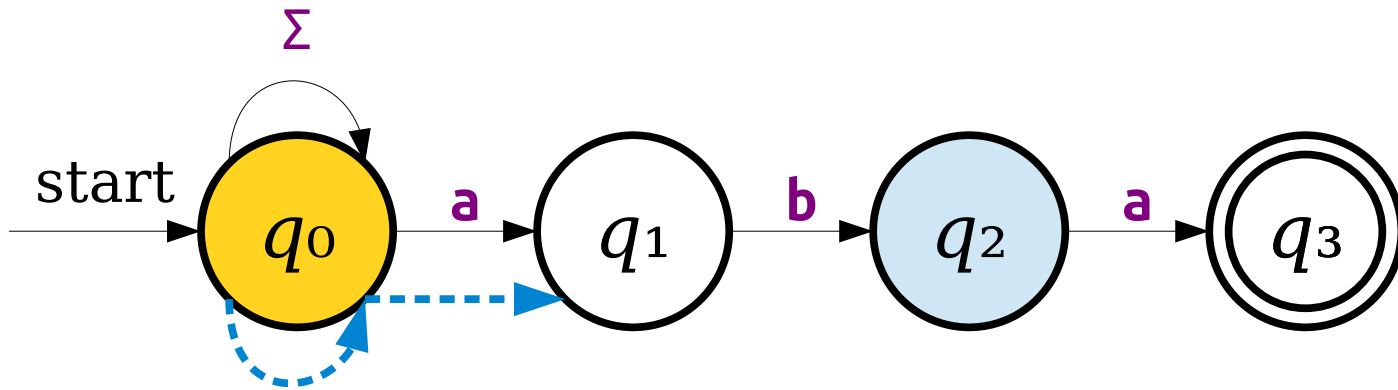
# Massive Parallelism



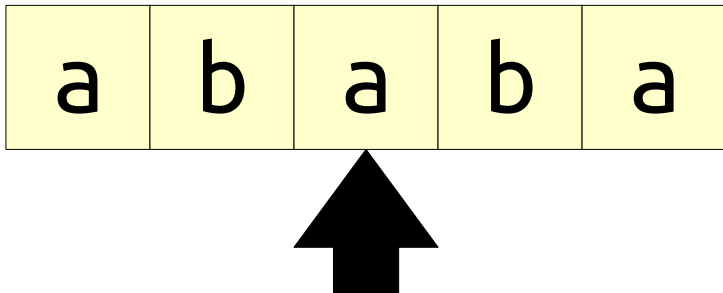
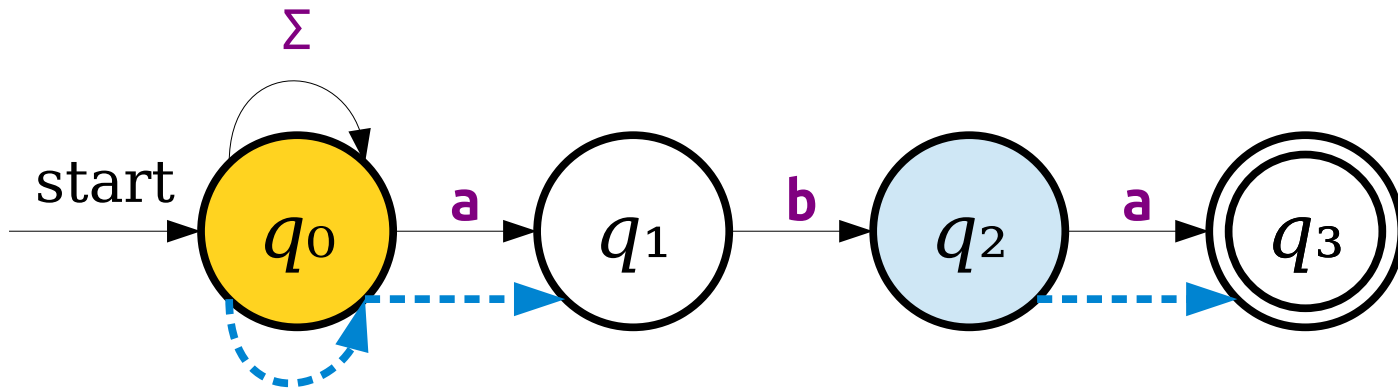
# Massive Parallelism



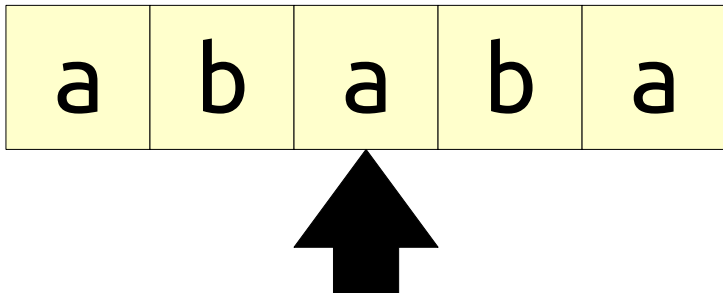
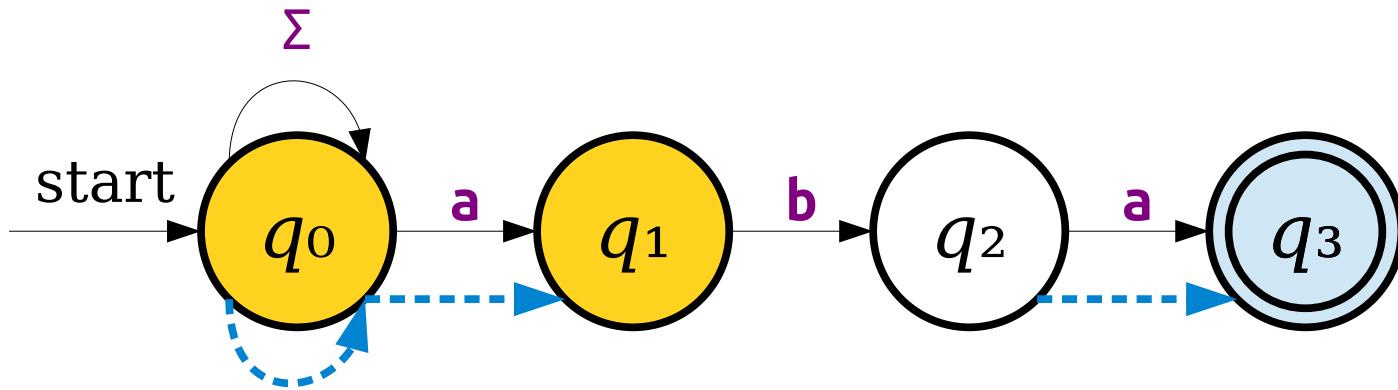
# Massive Parallelism



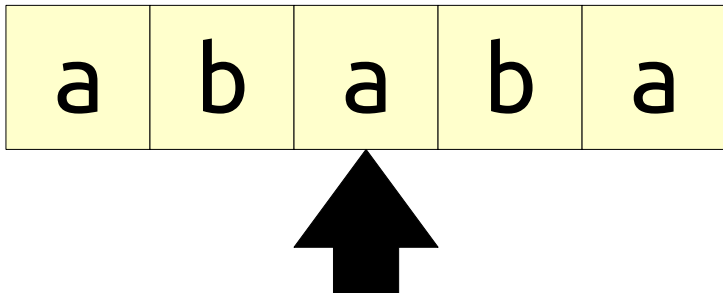
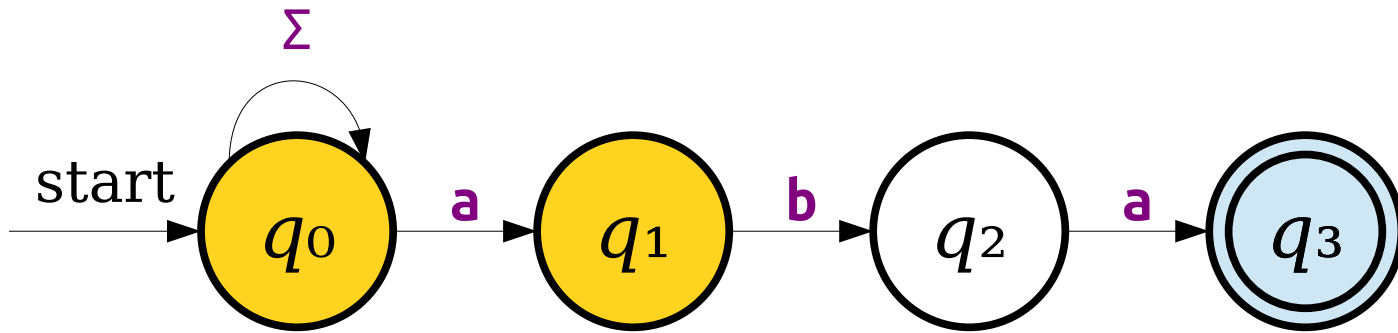
# Massive Parallelism



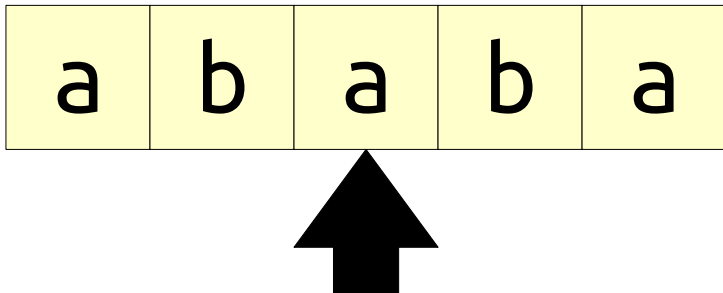
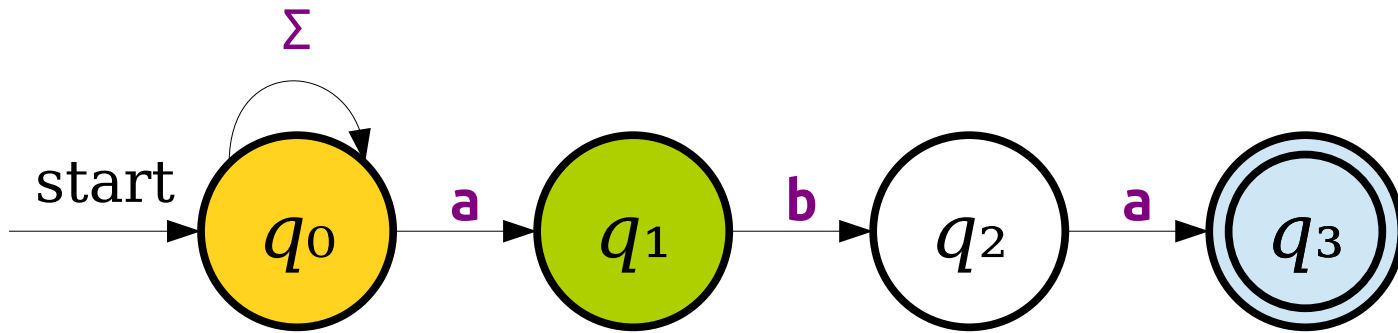
# Massive Parallelism



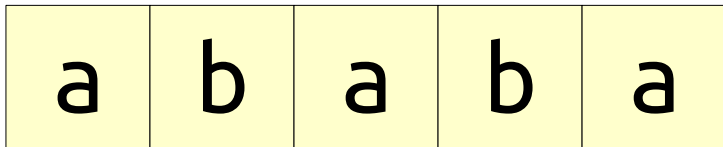
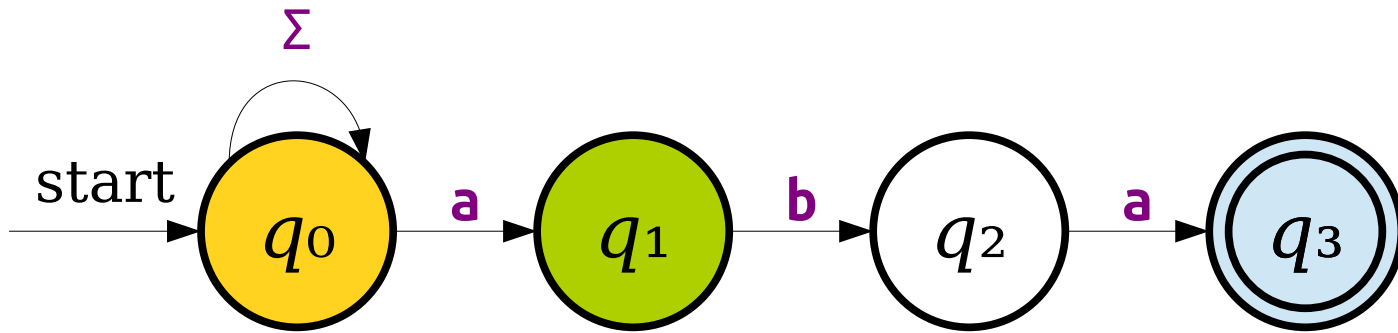
# Massive Parallelism



# Massive Parallelism

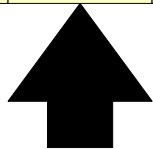
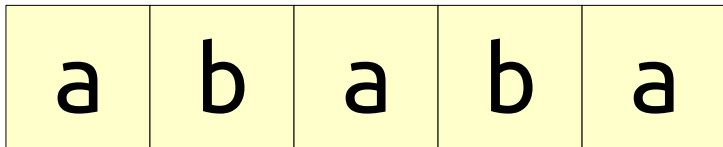
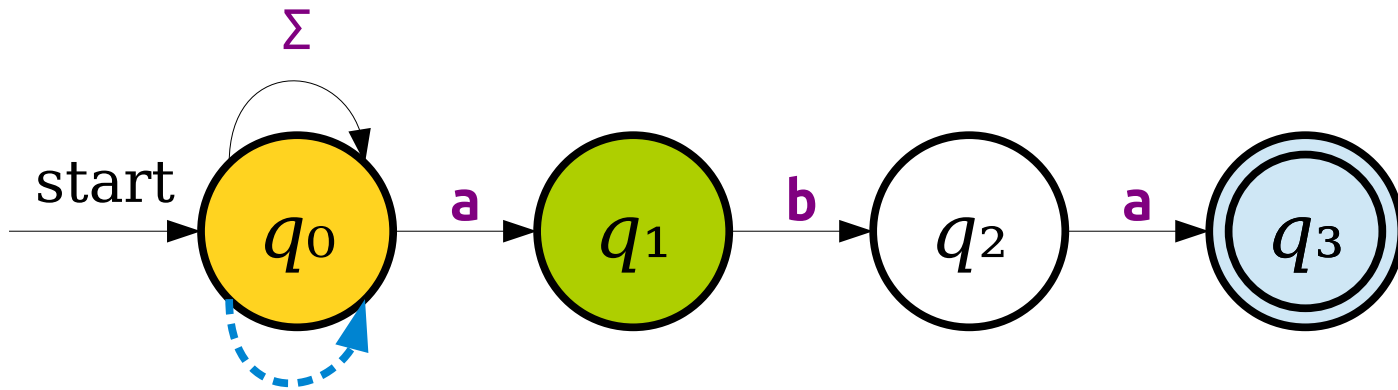


# Massive Parallelism

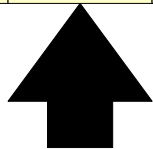
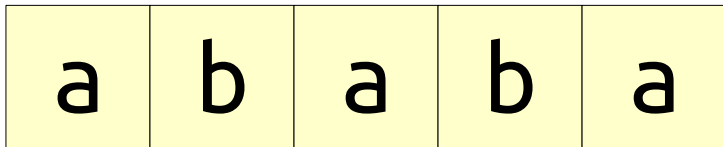
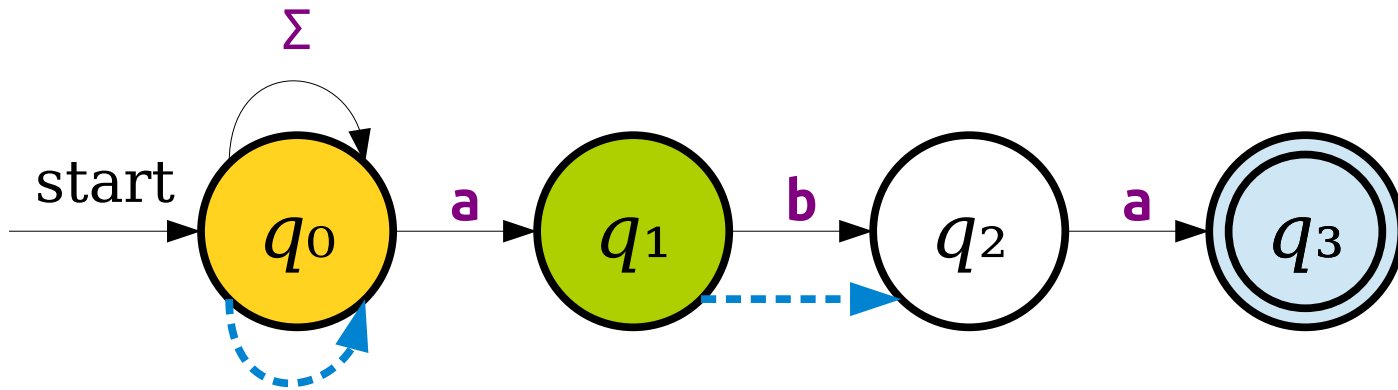




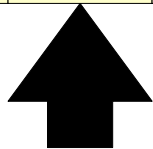
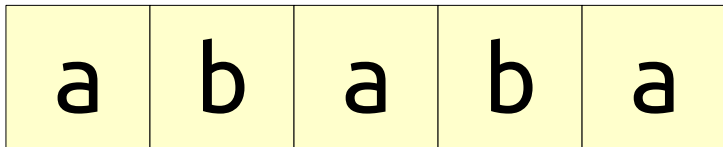
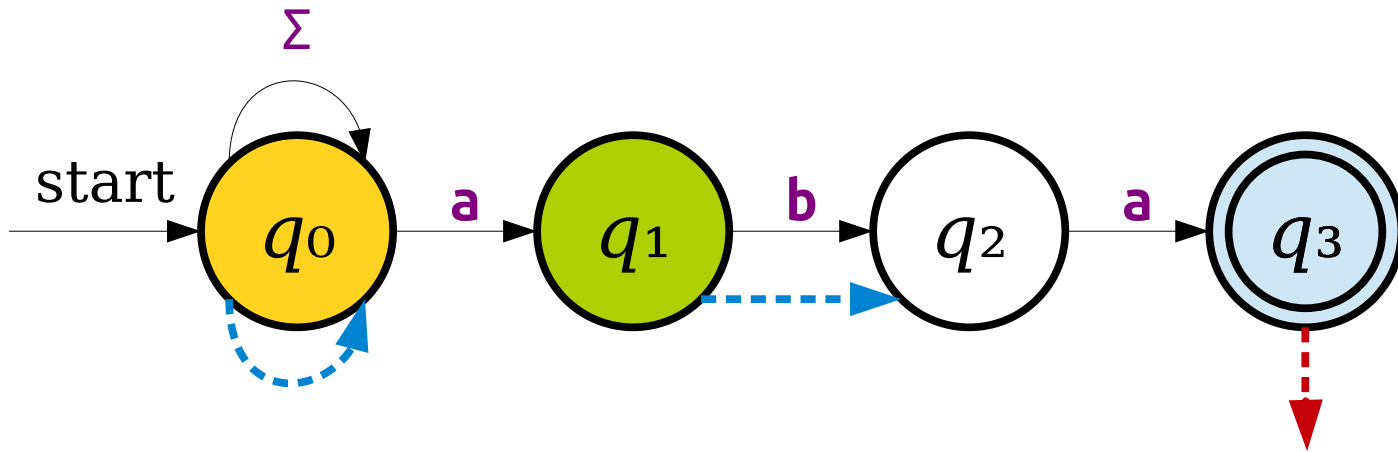
# Massive Parallelism



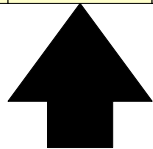
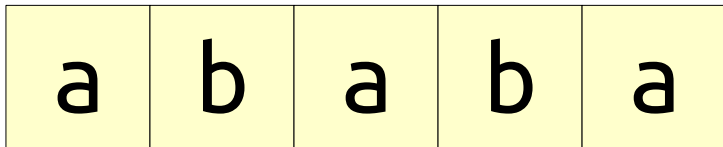
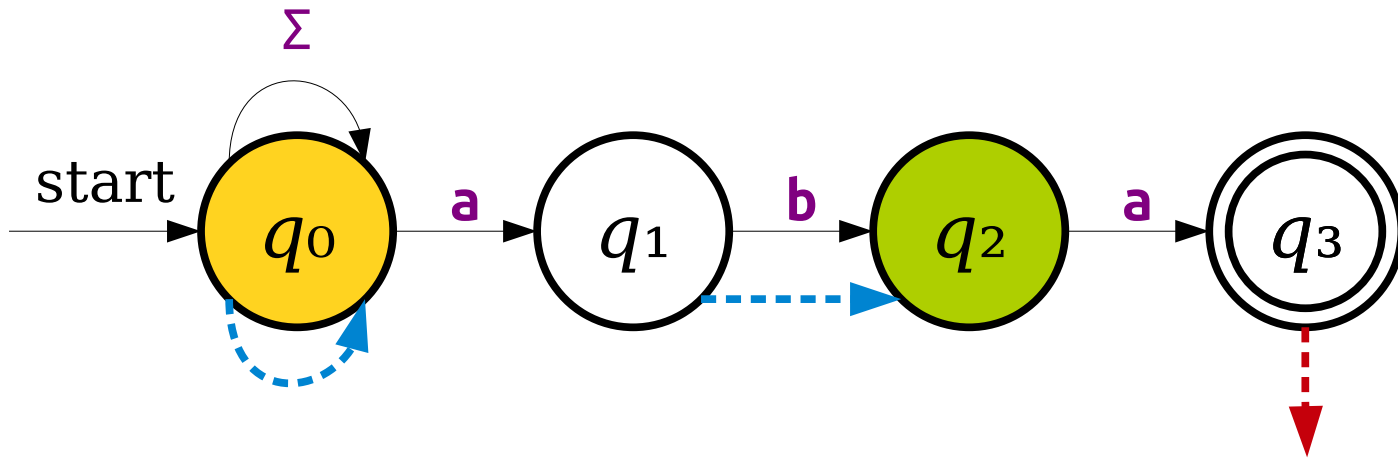
# Massive Parallelism



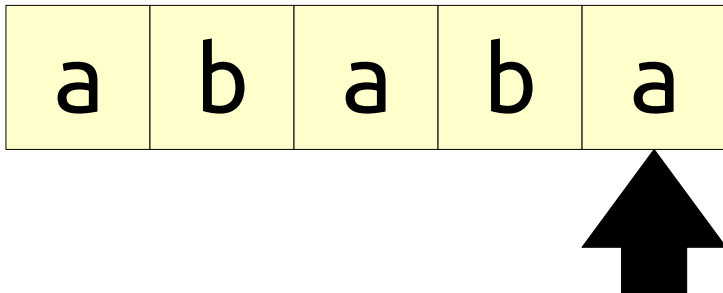
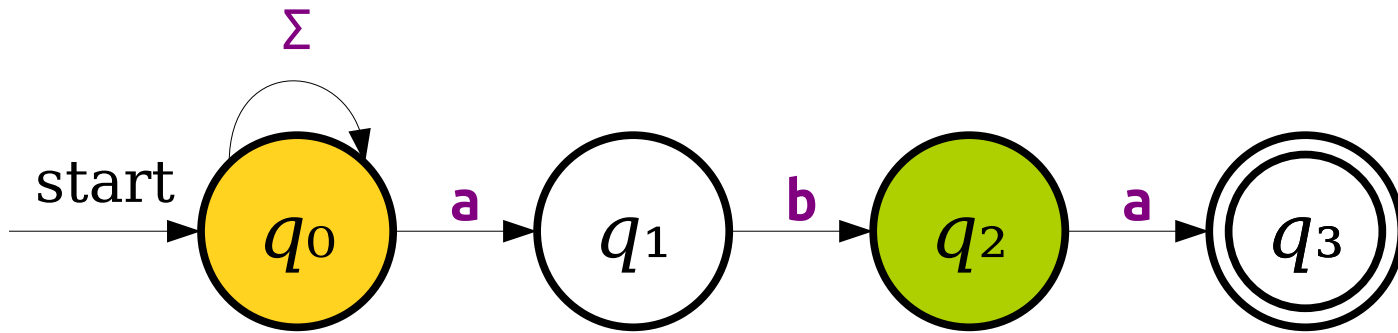
# Massive Parallelism



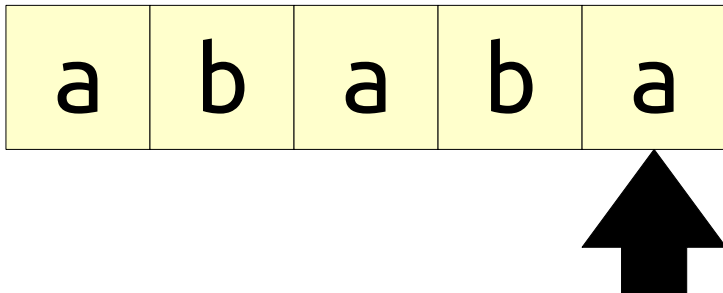
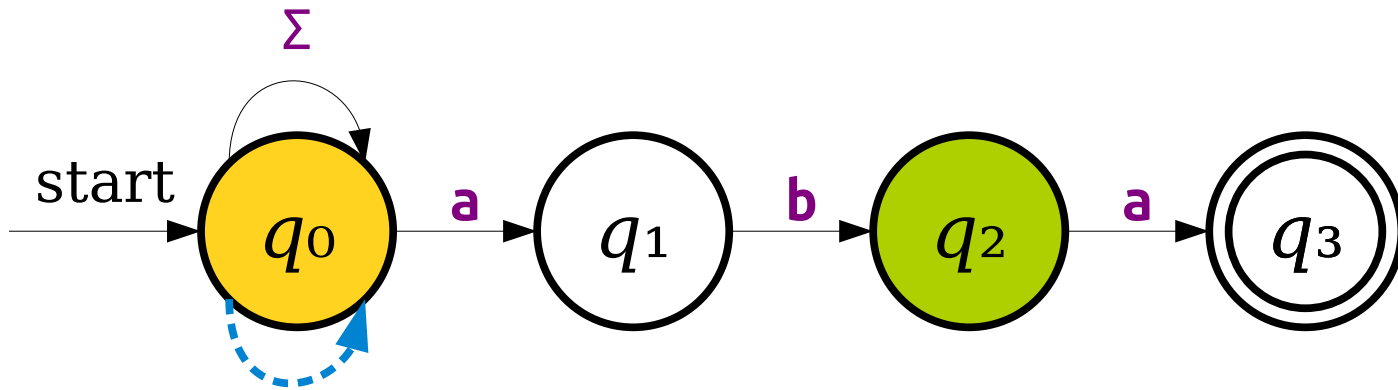
# Massive Parallelism



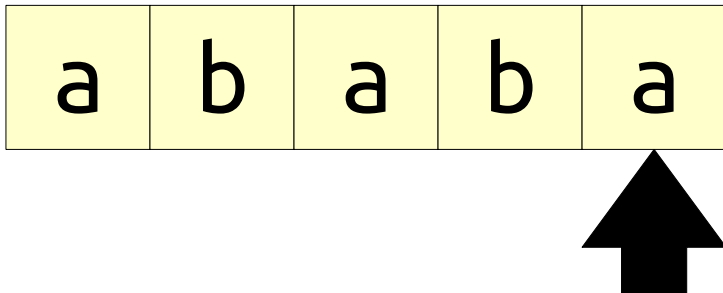
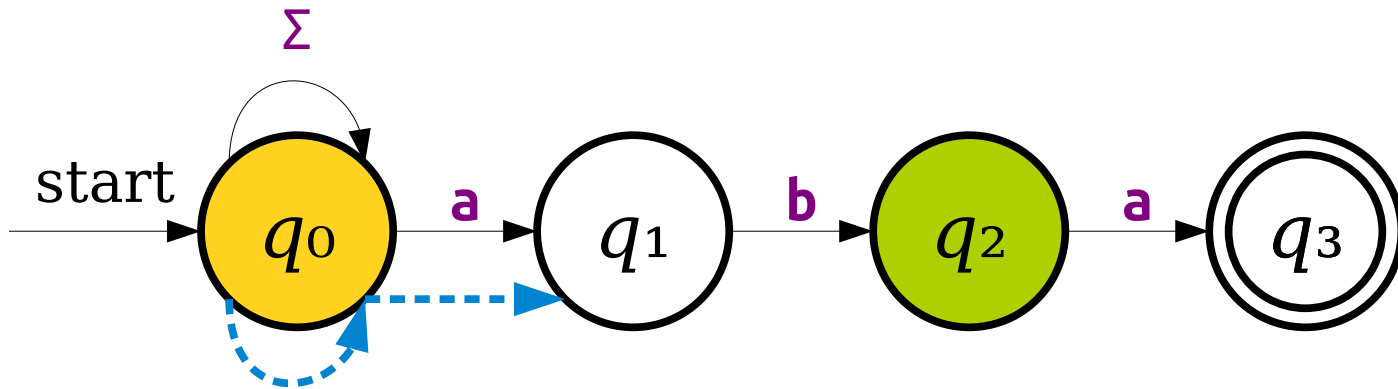
# Massive Parallelism



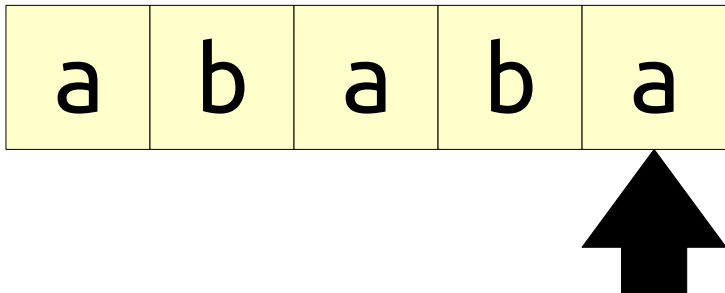
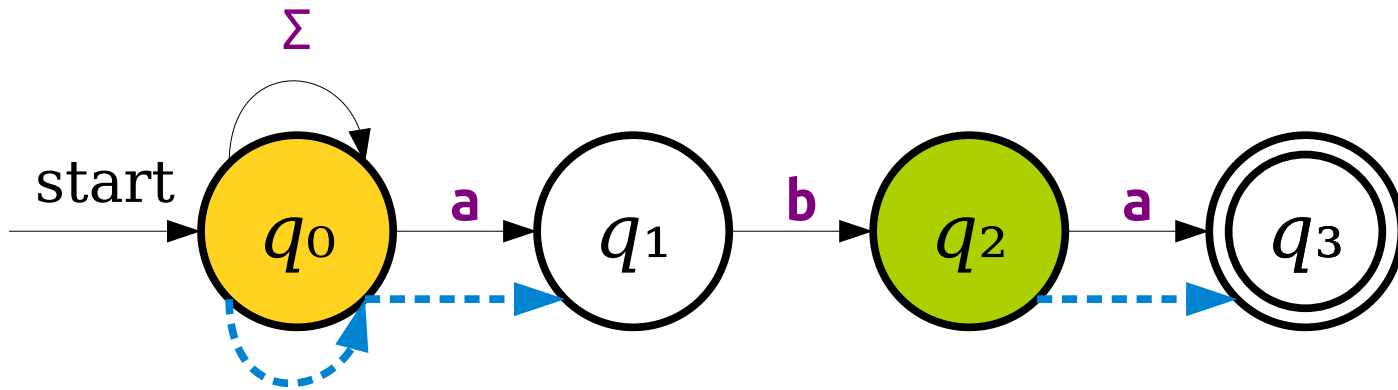
# Massive Parallelism



# Massive Parallelism

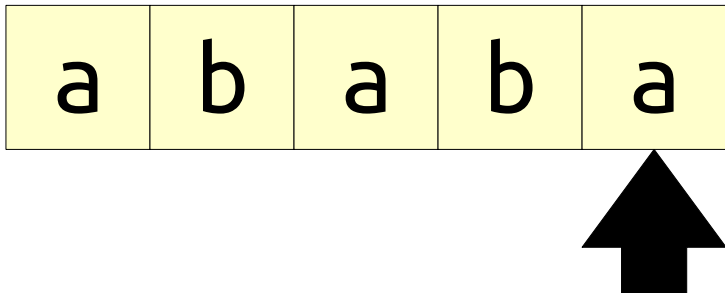
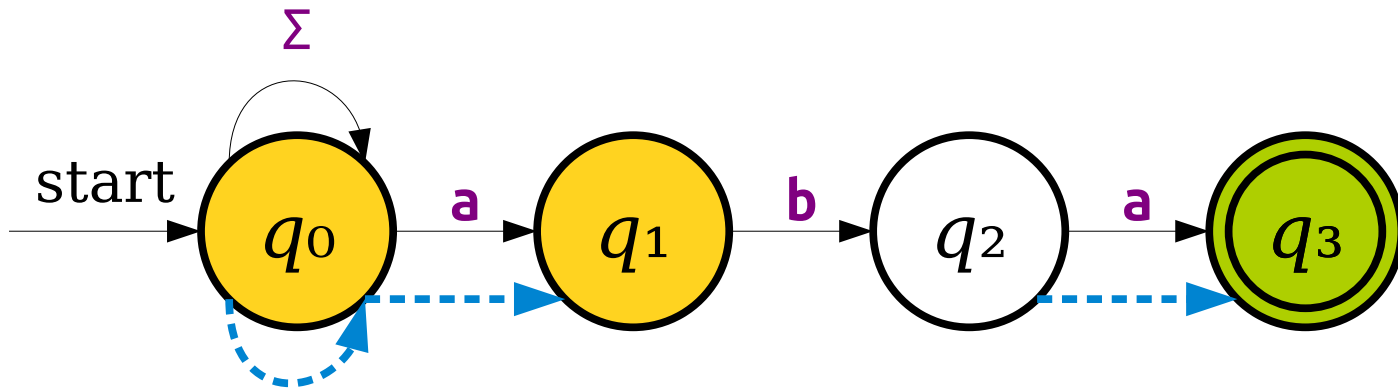


# Massive Parallelism

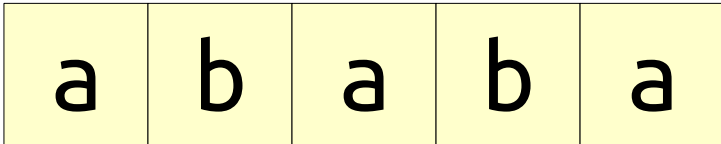
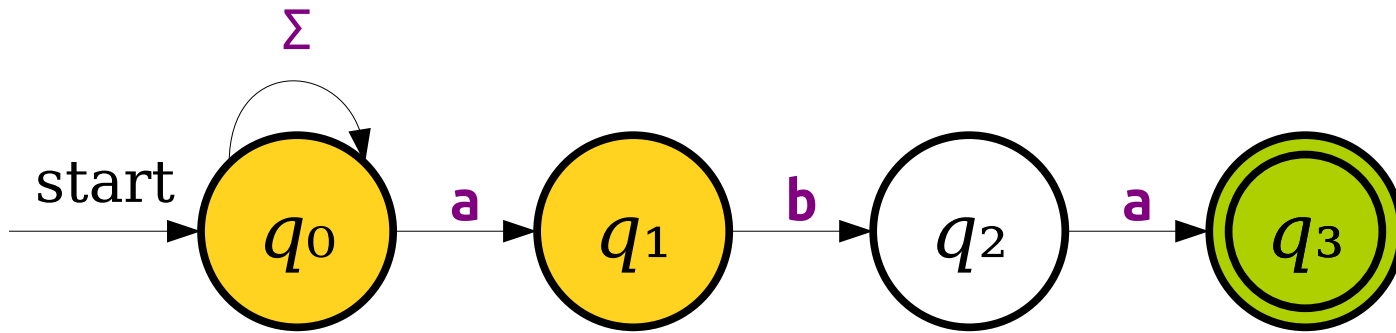




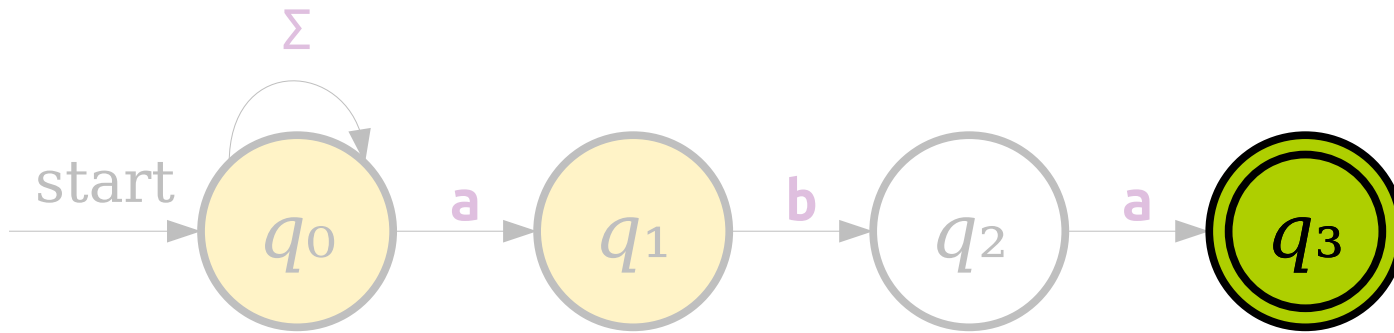
# Massive Parallelism



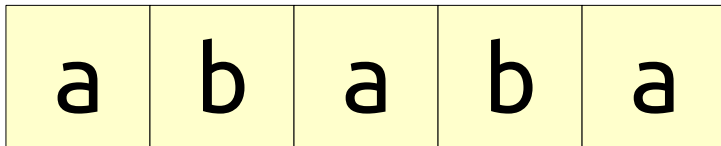
# Massive Parallelism



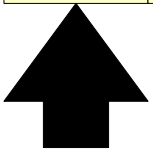
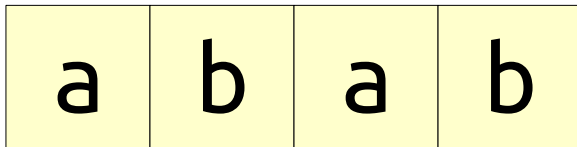
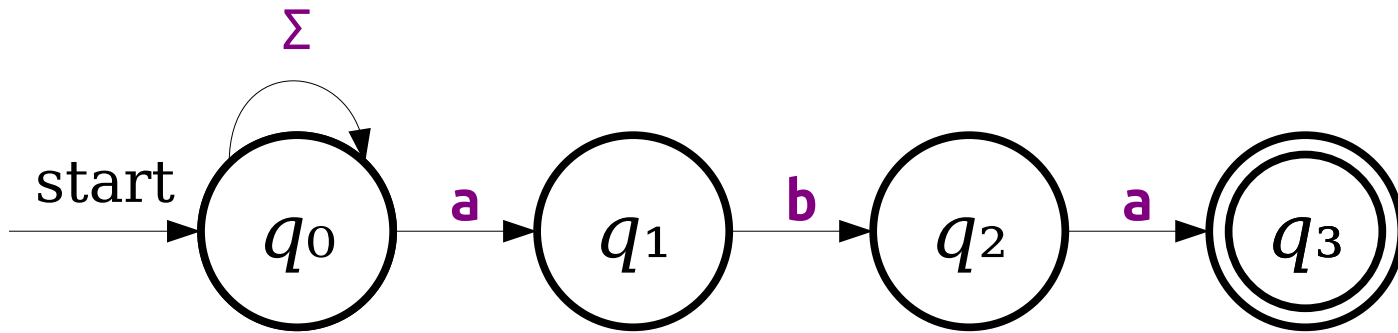
# Massive Parallelism



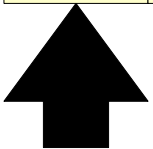
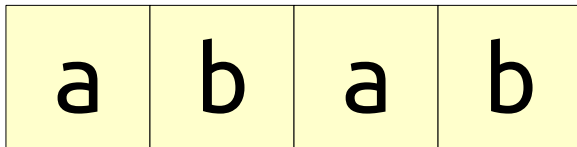
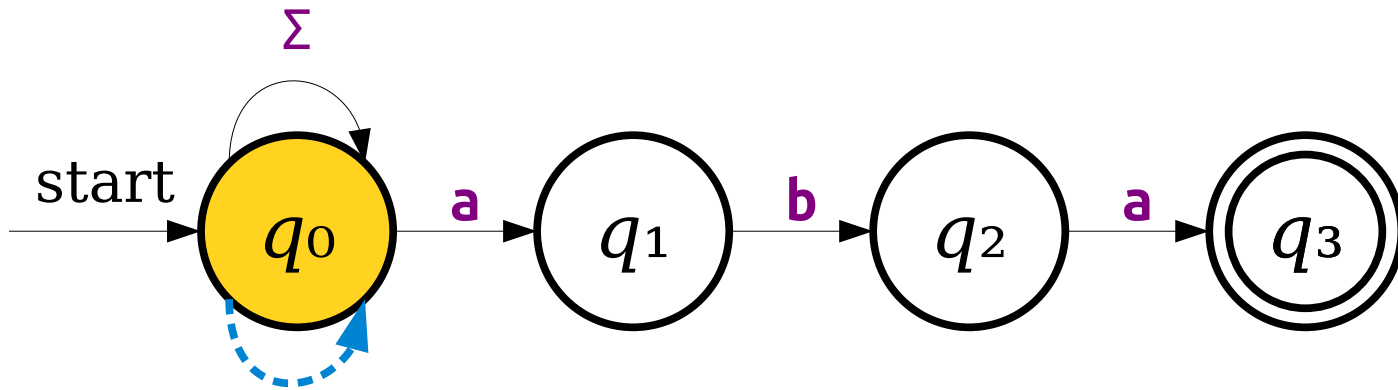
We're in at least one accepting state, so there's some path that gets us to an accepting state.



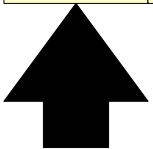
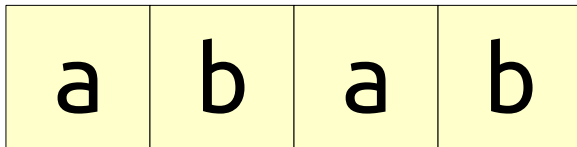
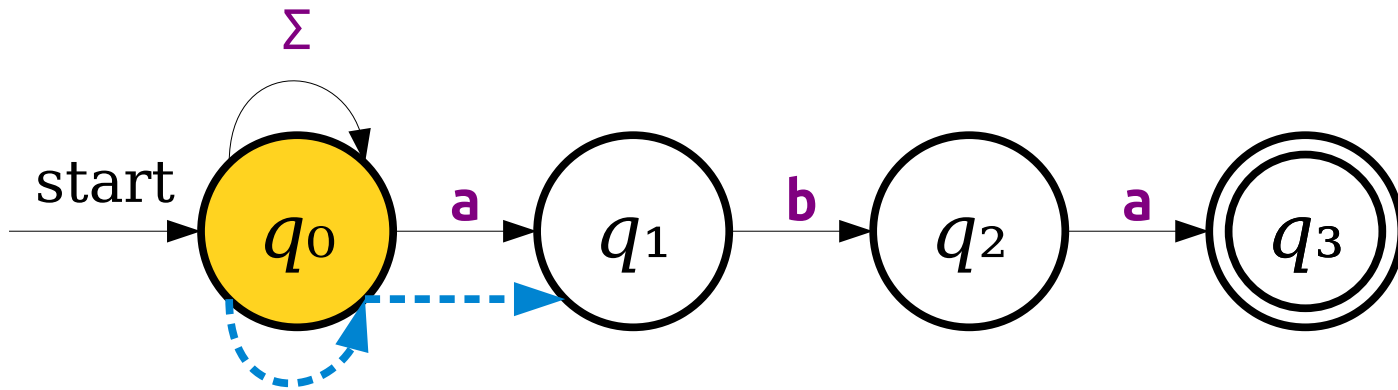
# Massive Parallelism



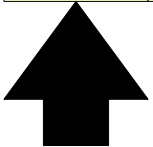
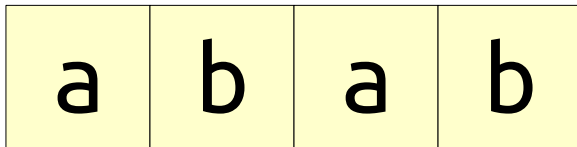
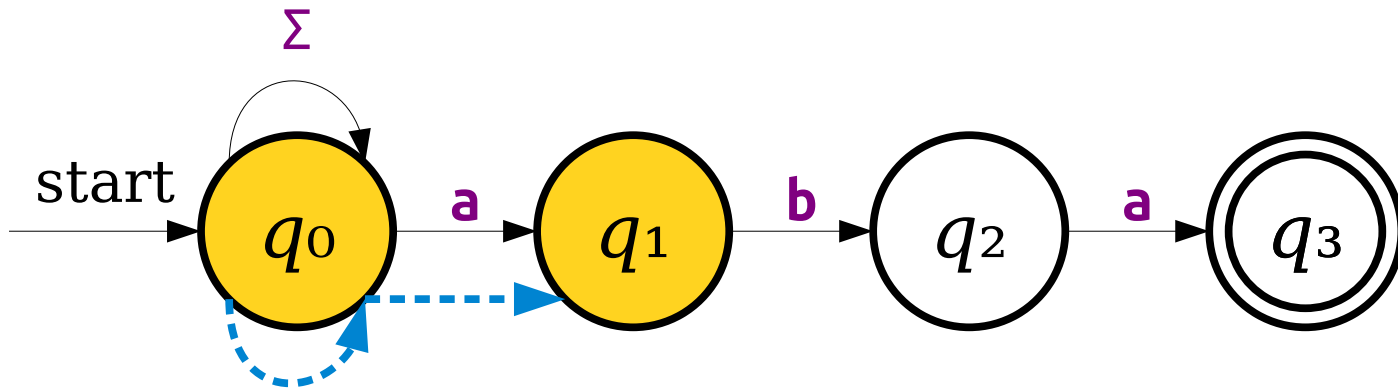
# Massive Parallelism



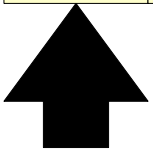
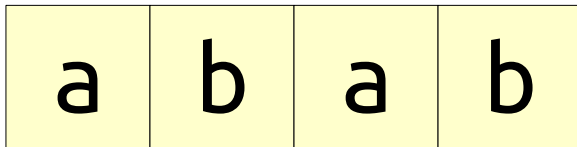
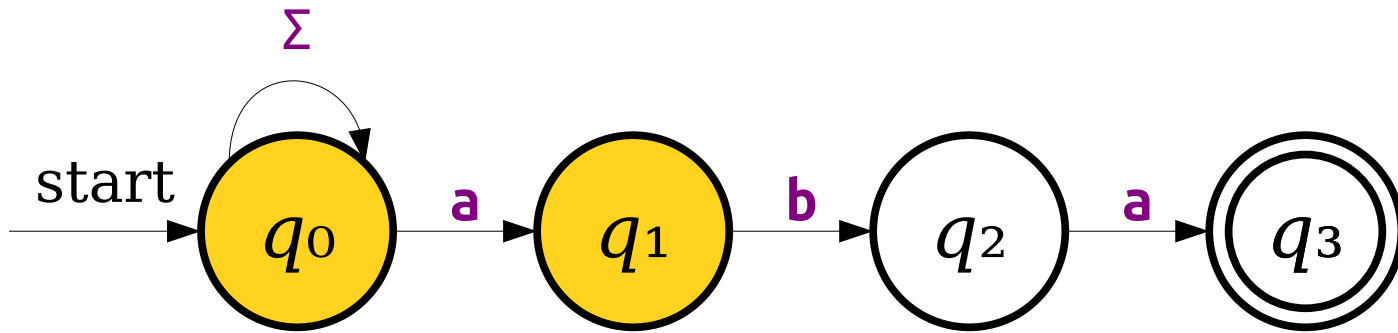
# Massive Parallelism



# Massive Parallelism

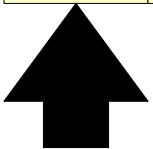
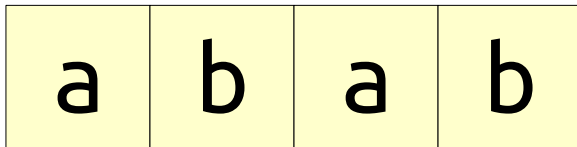
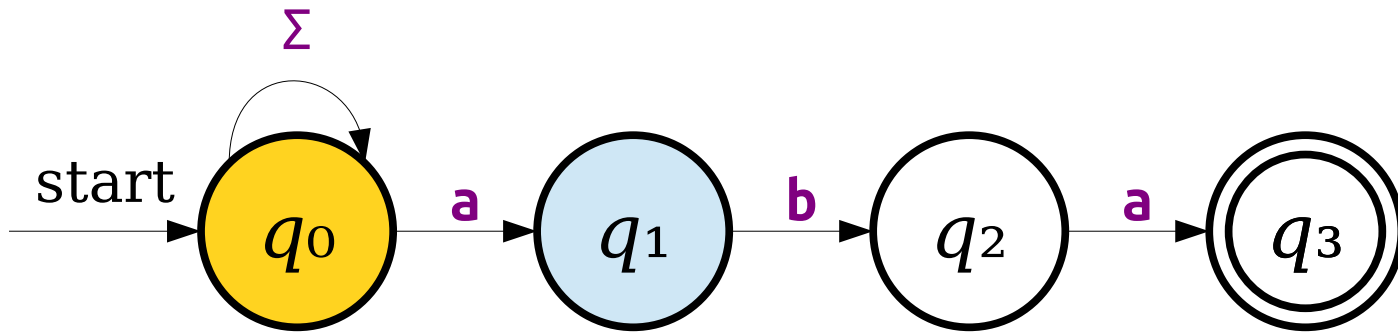


# Massive Parallelism

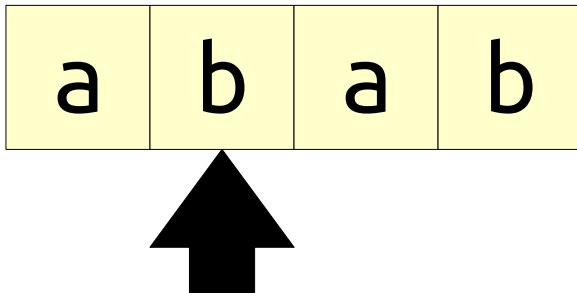
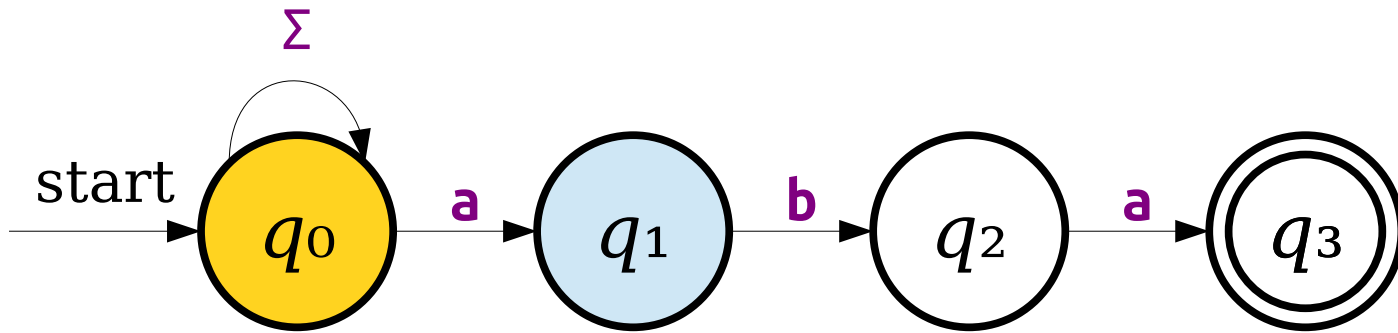




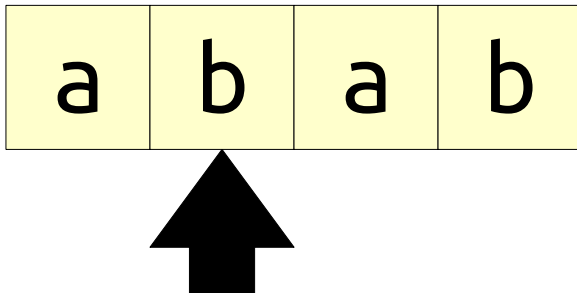
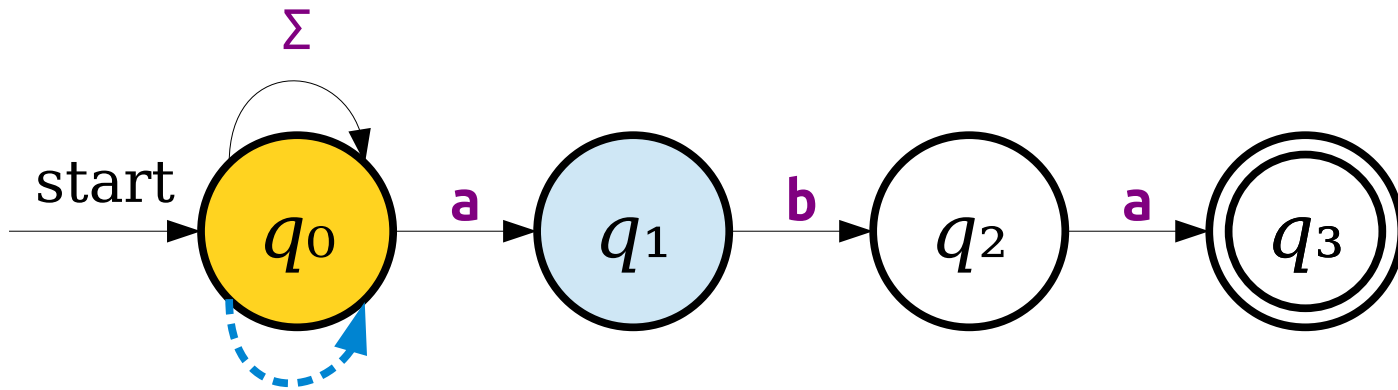
# Massive Parallelism



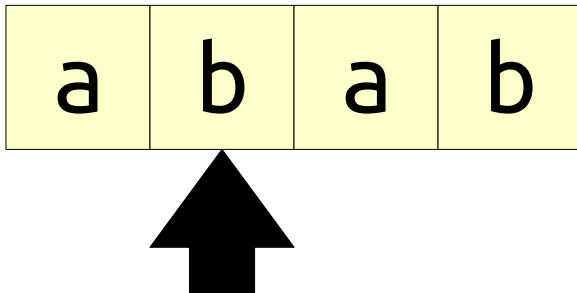
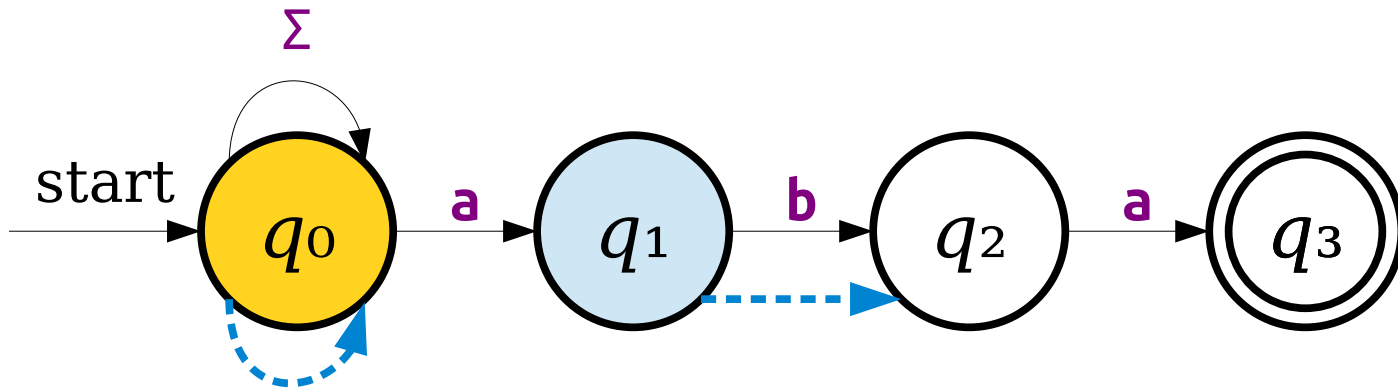
# Massive Parallelism



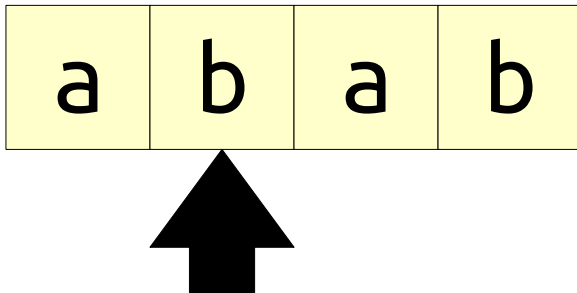
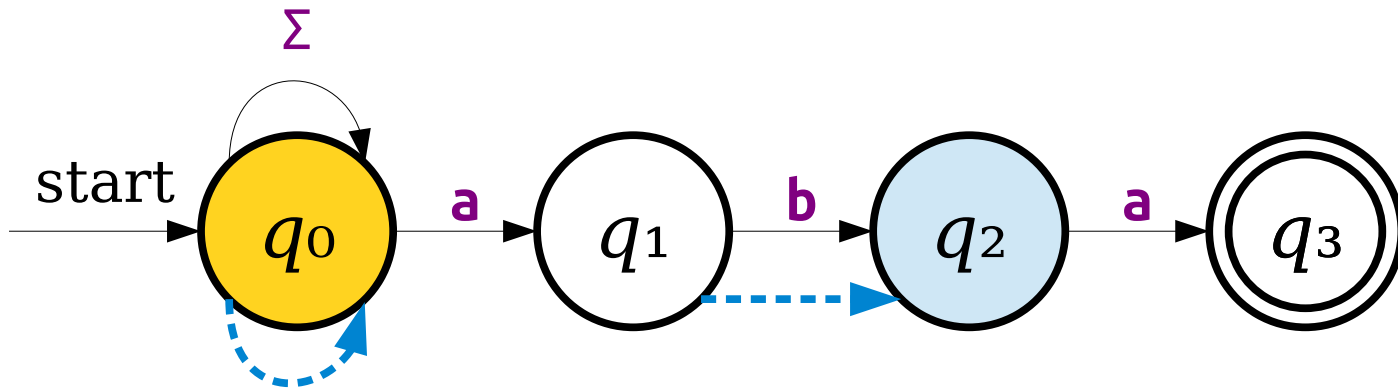
# Massive Parallelism



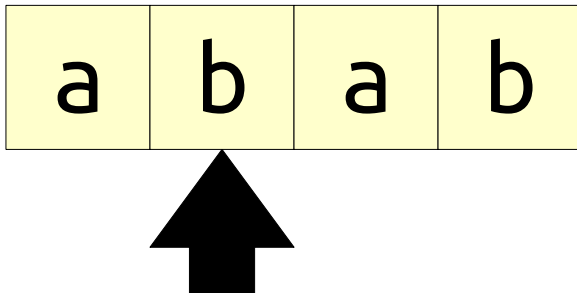
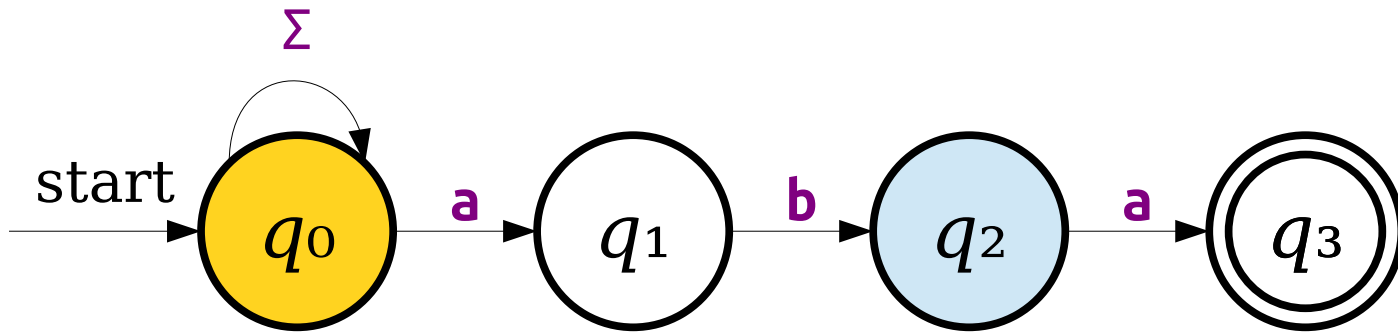
# Massive Parallelism



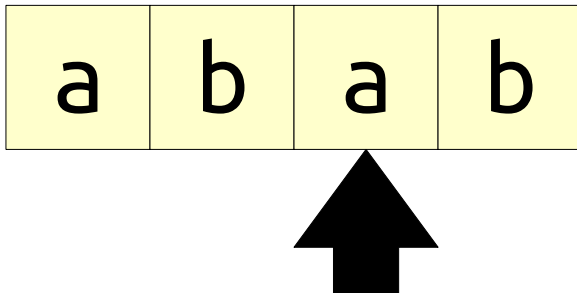
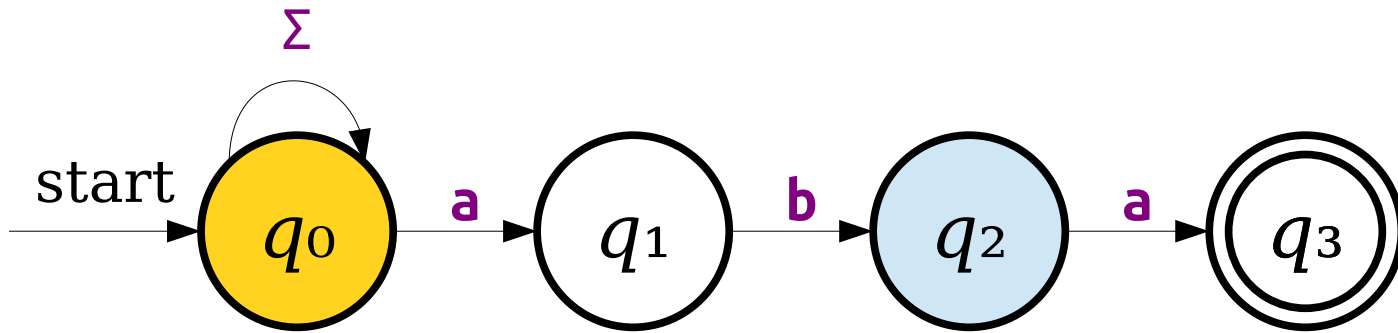
# Massive Parallelism



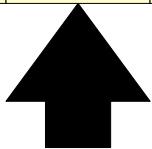
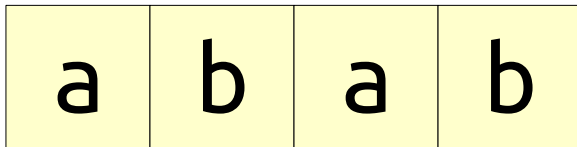
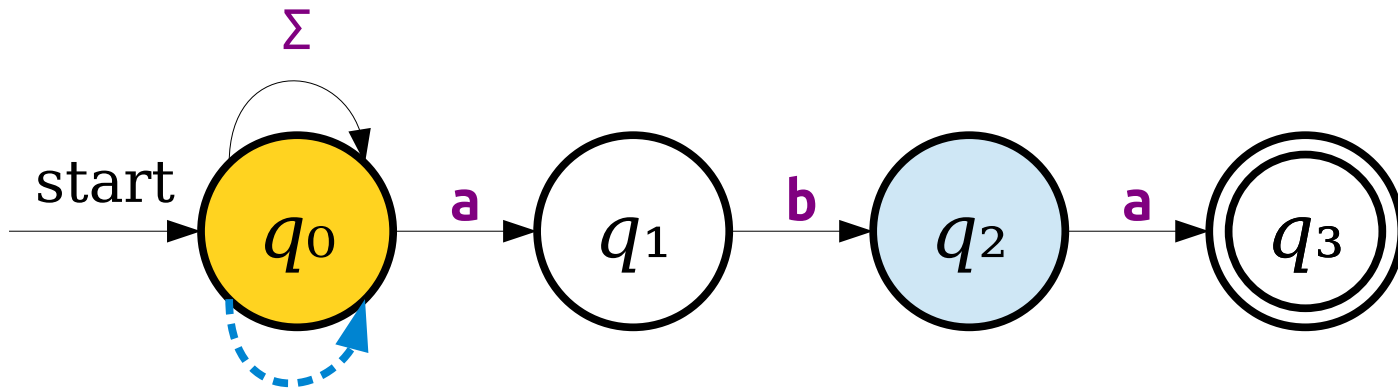
# Massive Parallelism



# Massive Parallelism

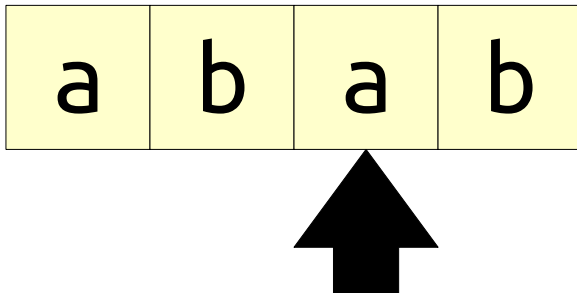
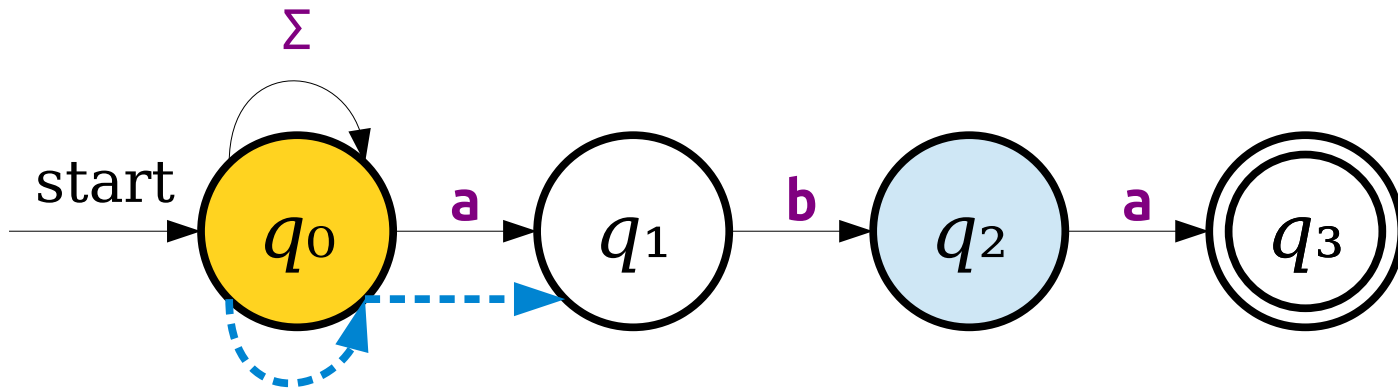


# Massive Parallelism

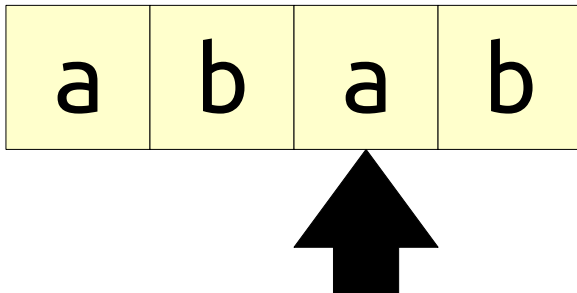
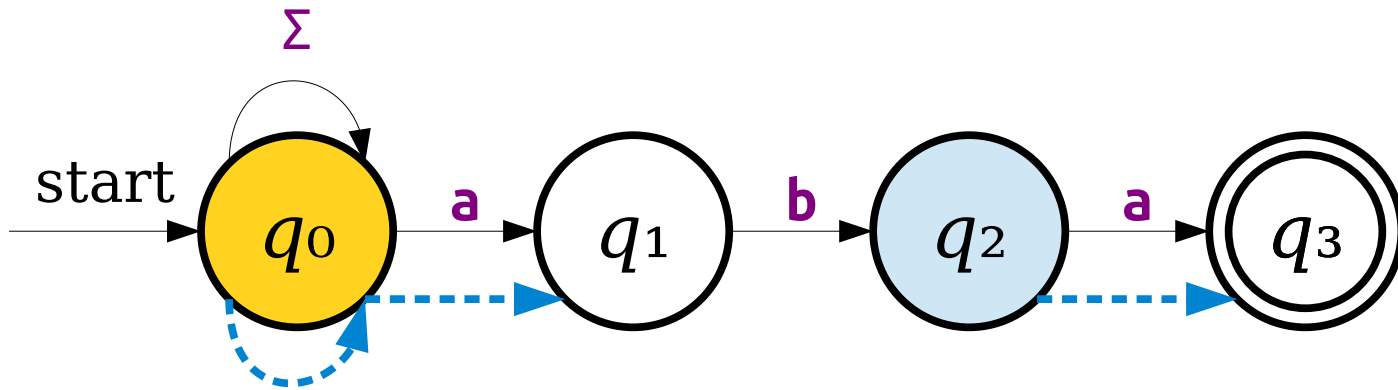




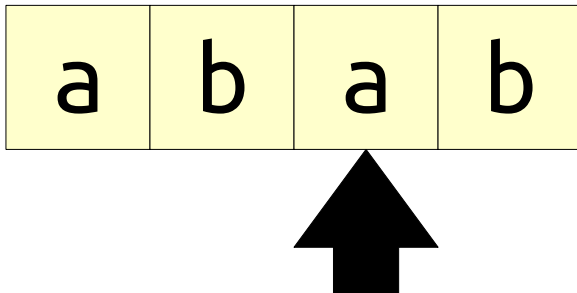
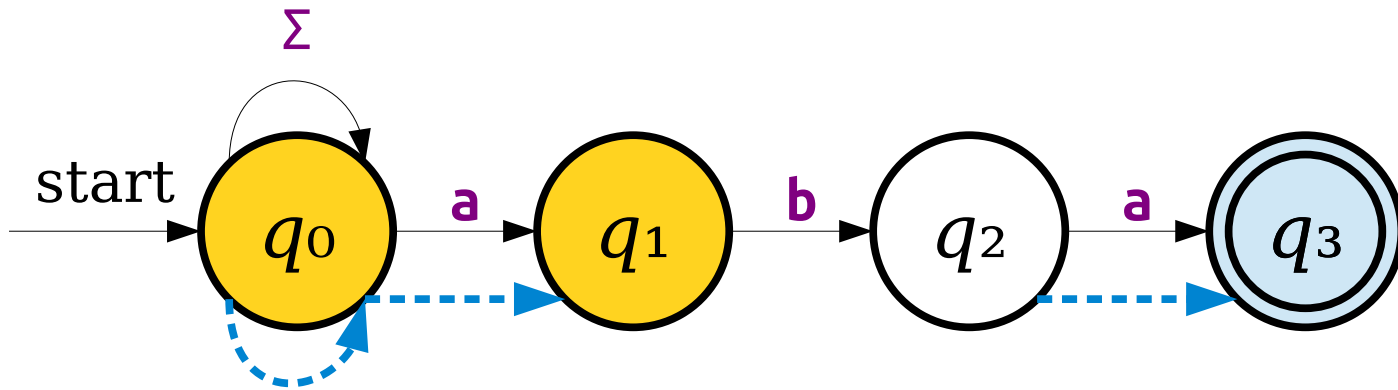
# Massive Parallelism



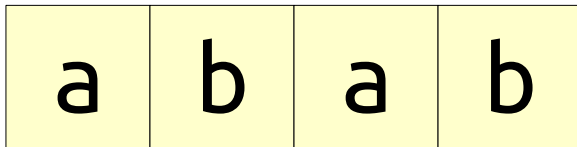
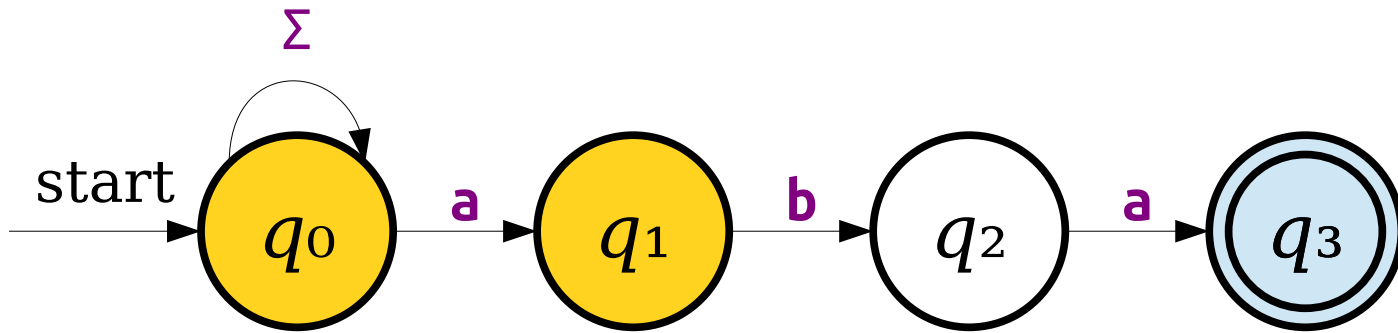
# Massive Parallelism



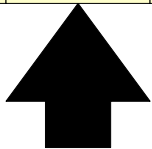
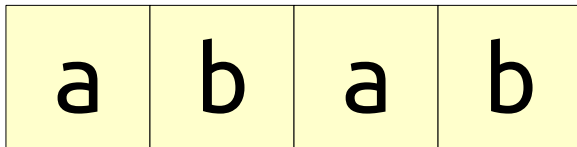
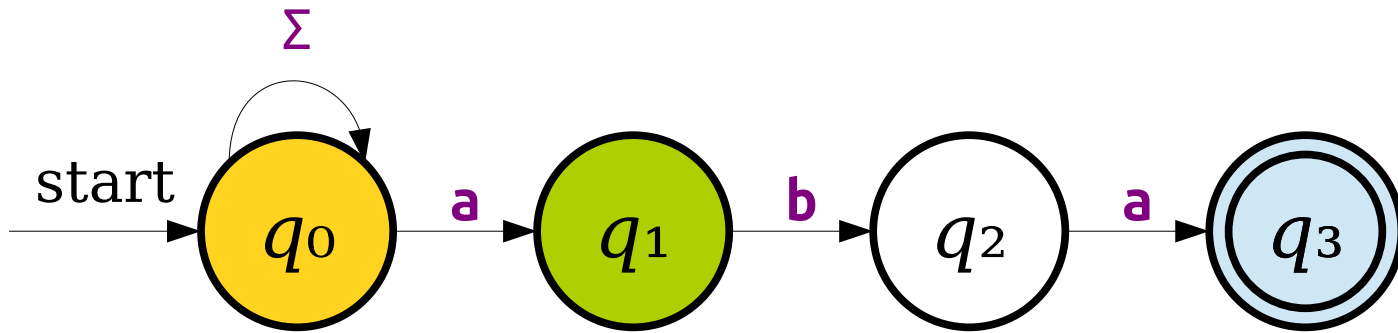
# Massive Parallelism



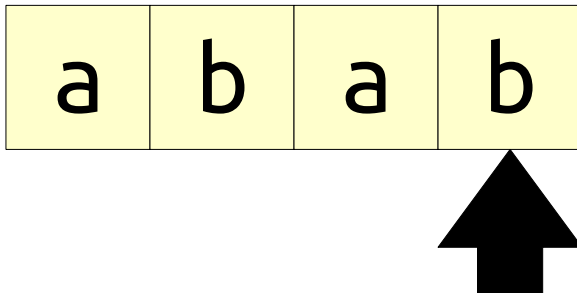
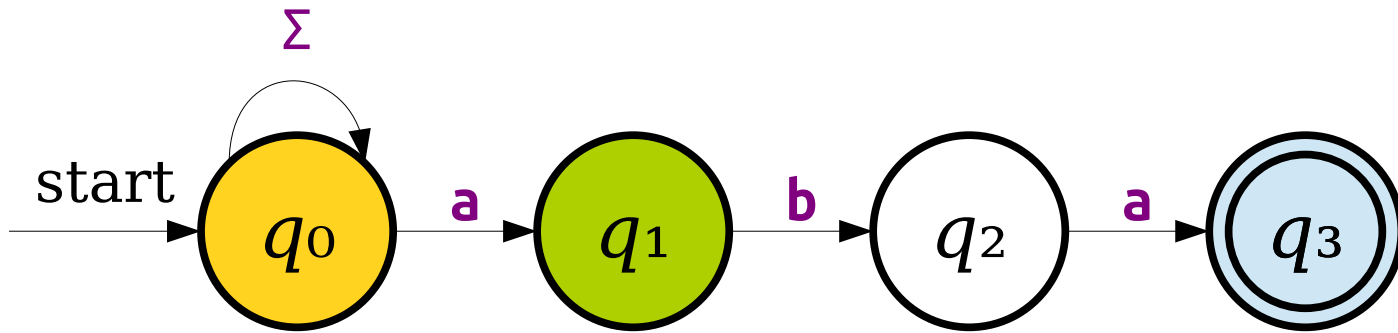
# Massive Parallelism



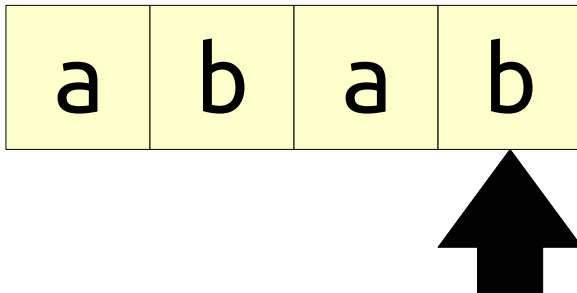
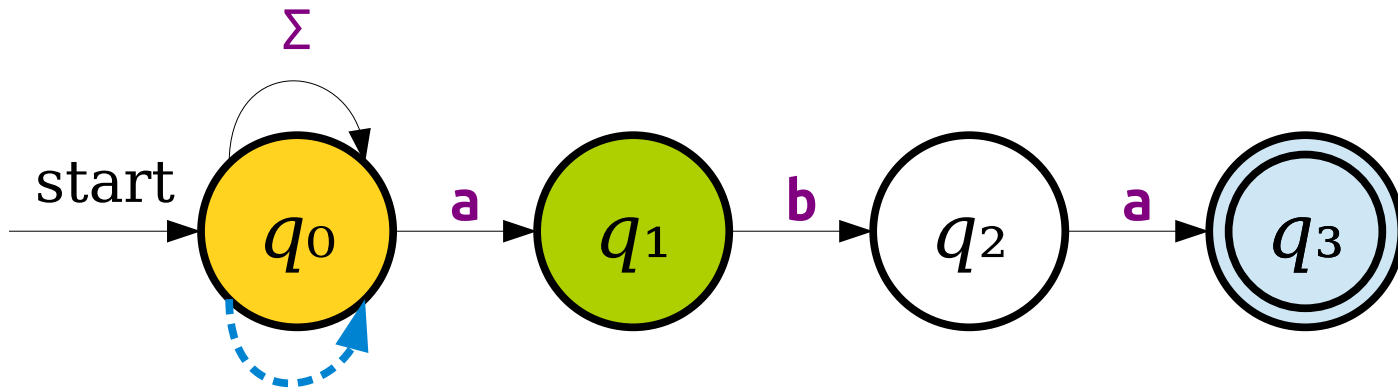
# Massive Parallelism



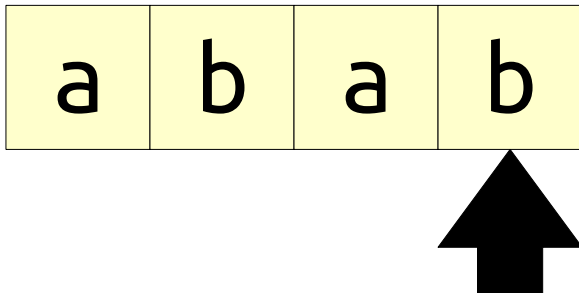
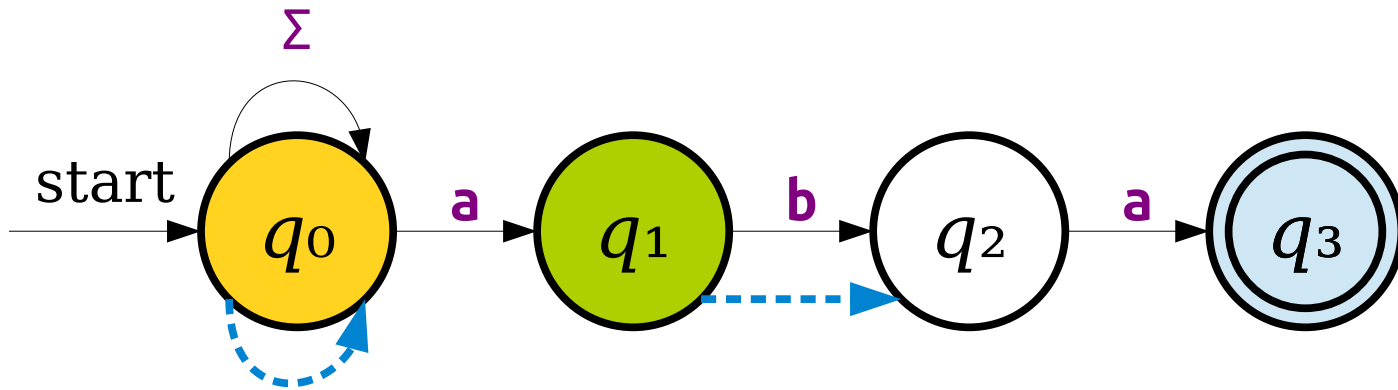
# Massive Parallelism



# Massive Parallelism

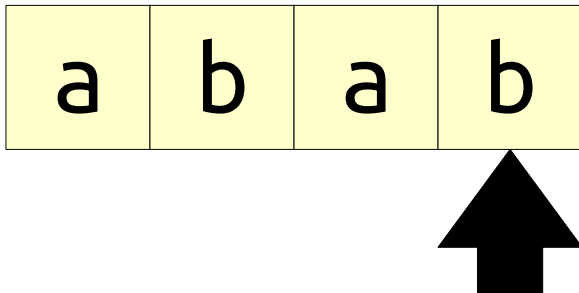
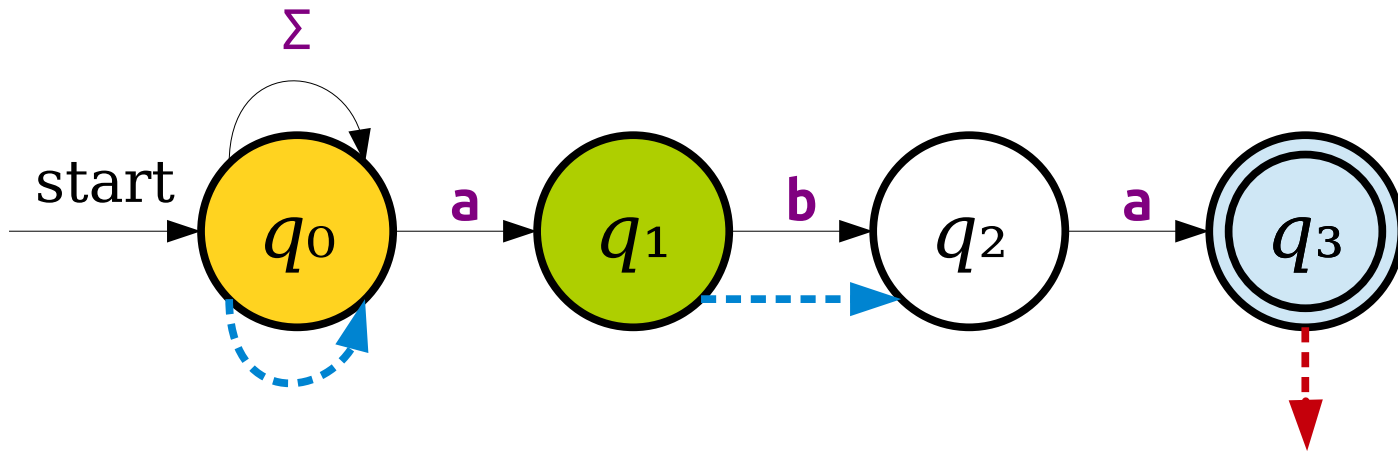


# Massive Parallelism

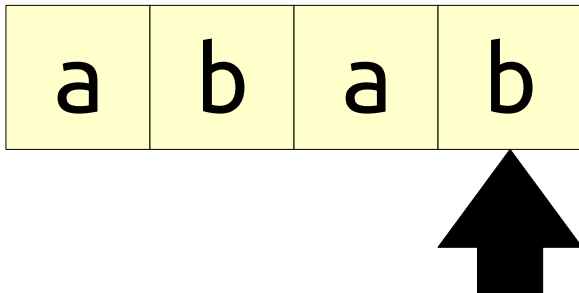
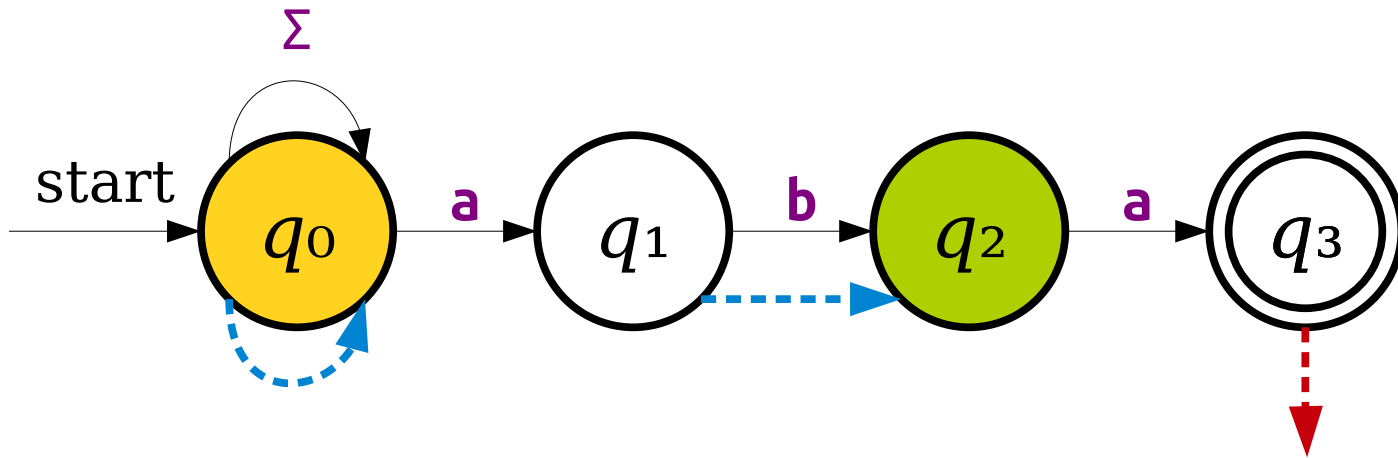




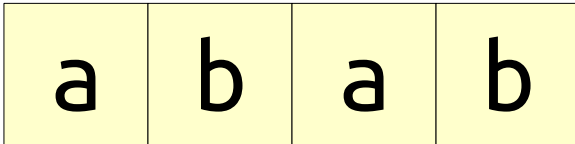
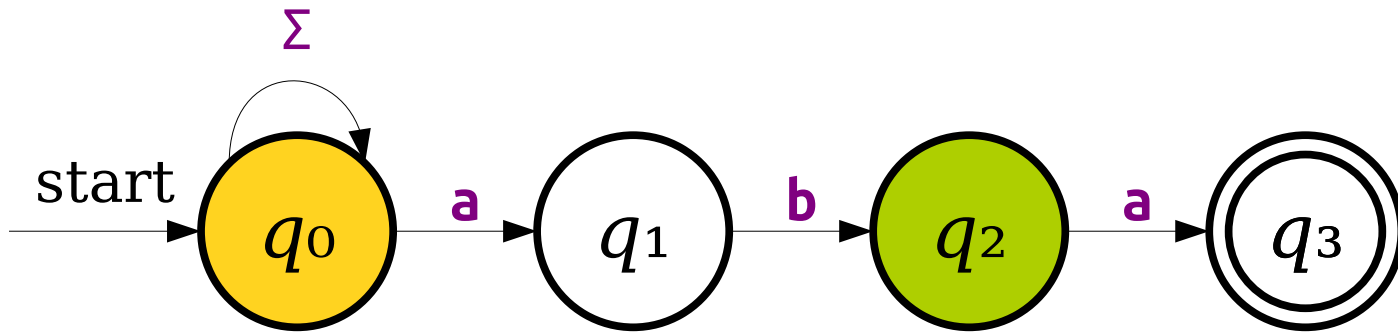
# Massive Parallelism



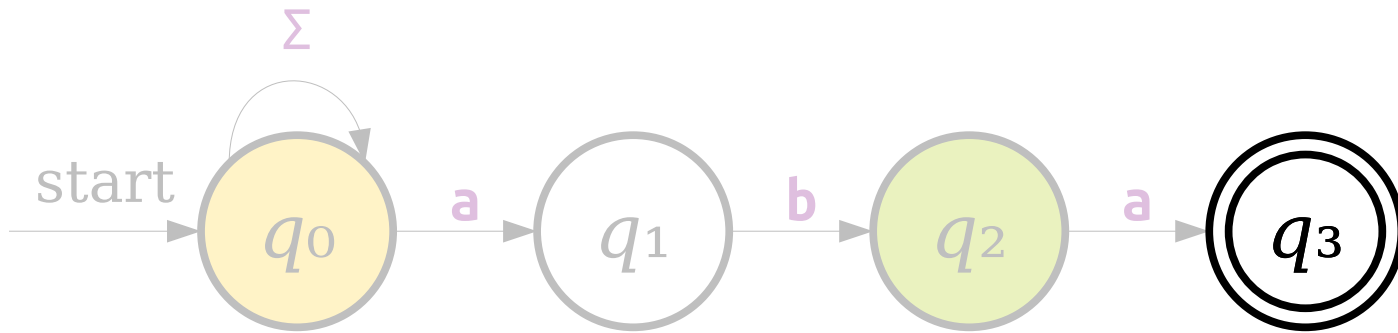
# Massive Parallelism



# Massive Parallelism



# Massive Parallelism



We're not in any accepting state, so no possible path accepts.

a	b	a	b
---	---	---	---



# Massive Parallelism

- **Key Idea:** Imagine the NFA can be in many states at once. The NFA tries all possible transitions in parallel with one another.
- Here's a rigorous explanation about how this works; read this on your own time.
  - Start off with the start state active, plus all states that can be reached by zero or more  $\epsilon$ -transitions.
  - When you read a symbol **a** in a set of states  $S$ :
    - Form the set  $S'$  of states that can be reached by following a single **a** transition from some state in  $S$ .
    - Your new set of states is the set of states in  $S'$ , plus the states reachable from  $S'$  by following zero or more  $\epsilon$ -transitions.

# Designing NFAs

# Designing NFAs

- ***Embrace the nondeterminism!***
- Good model: ***Guess-and-check***:
  - Is there some information that you'd really like to have? Have the machine *nondeterministically guess* that information.
  - Then, have the machine *deterministically check* that the choice was correct.
- The *guess* phase corresponds to trying lots of different options.
- The *check* phase corresponds to filtering out bad guesses or wrong options.

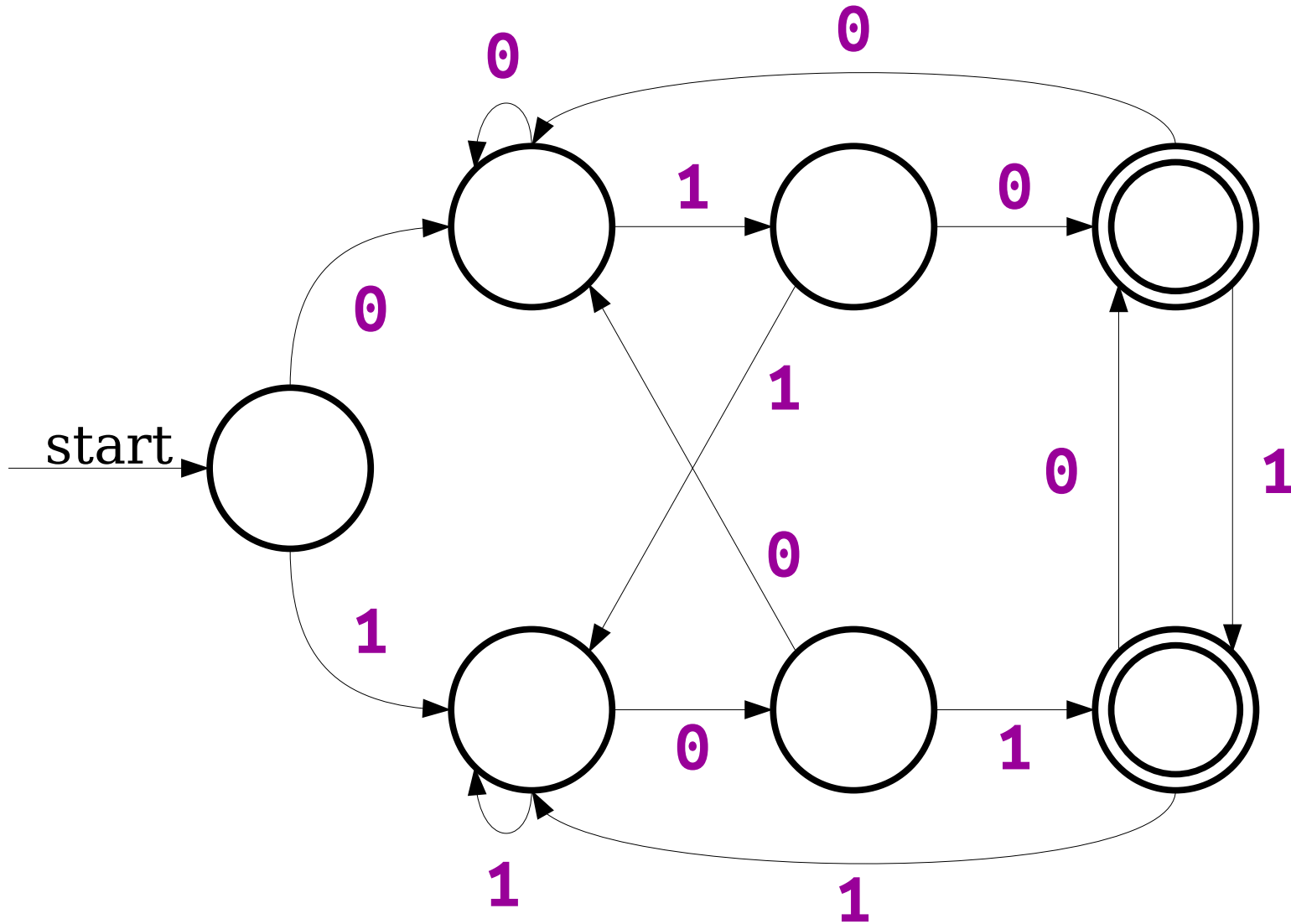
# Guess-and-Check

$$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$$



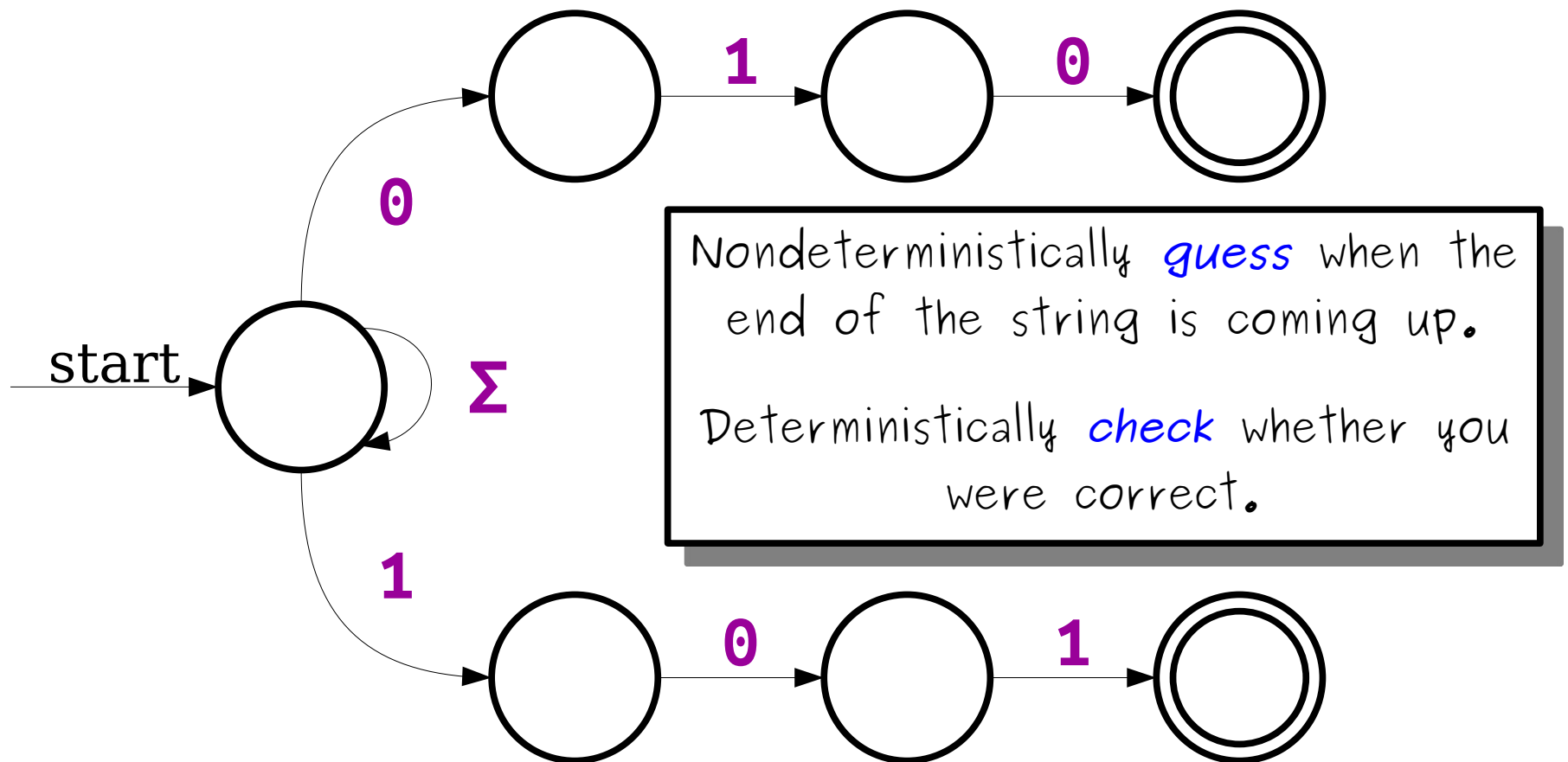
# Guess-and-Check

$$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$$



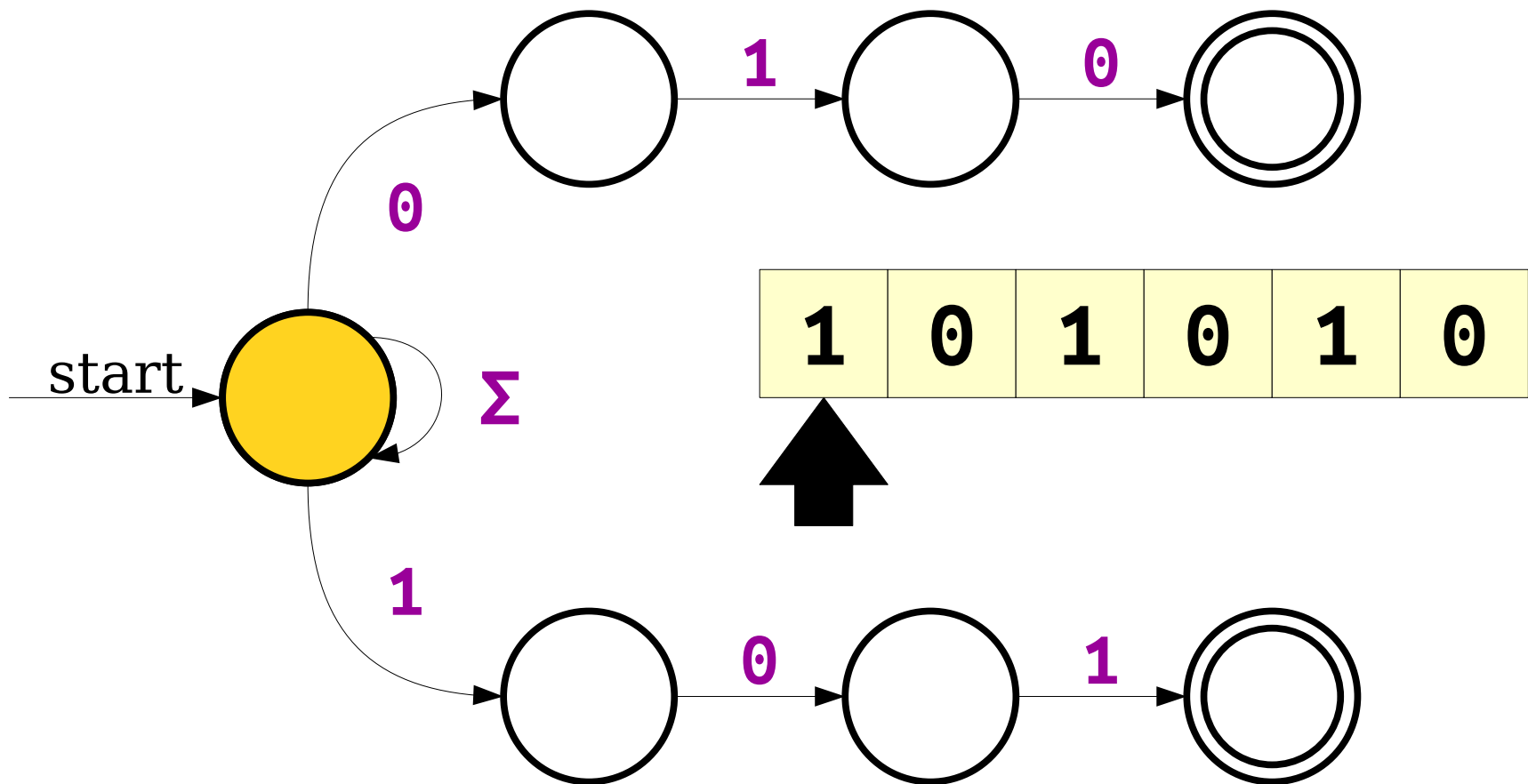
# Guess-and-Check

$$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$$



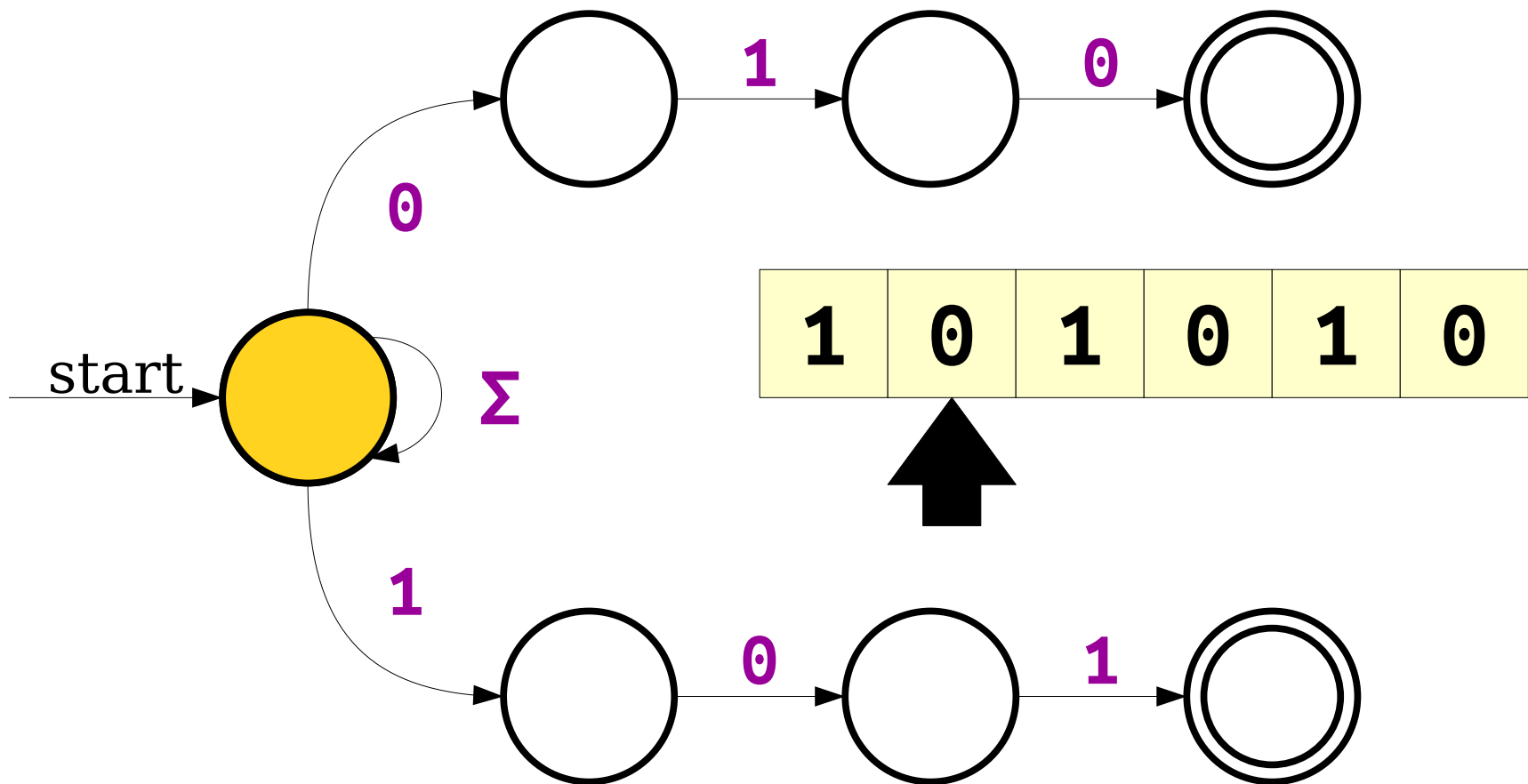
# Guess-and-Check

$$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$$



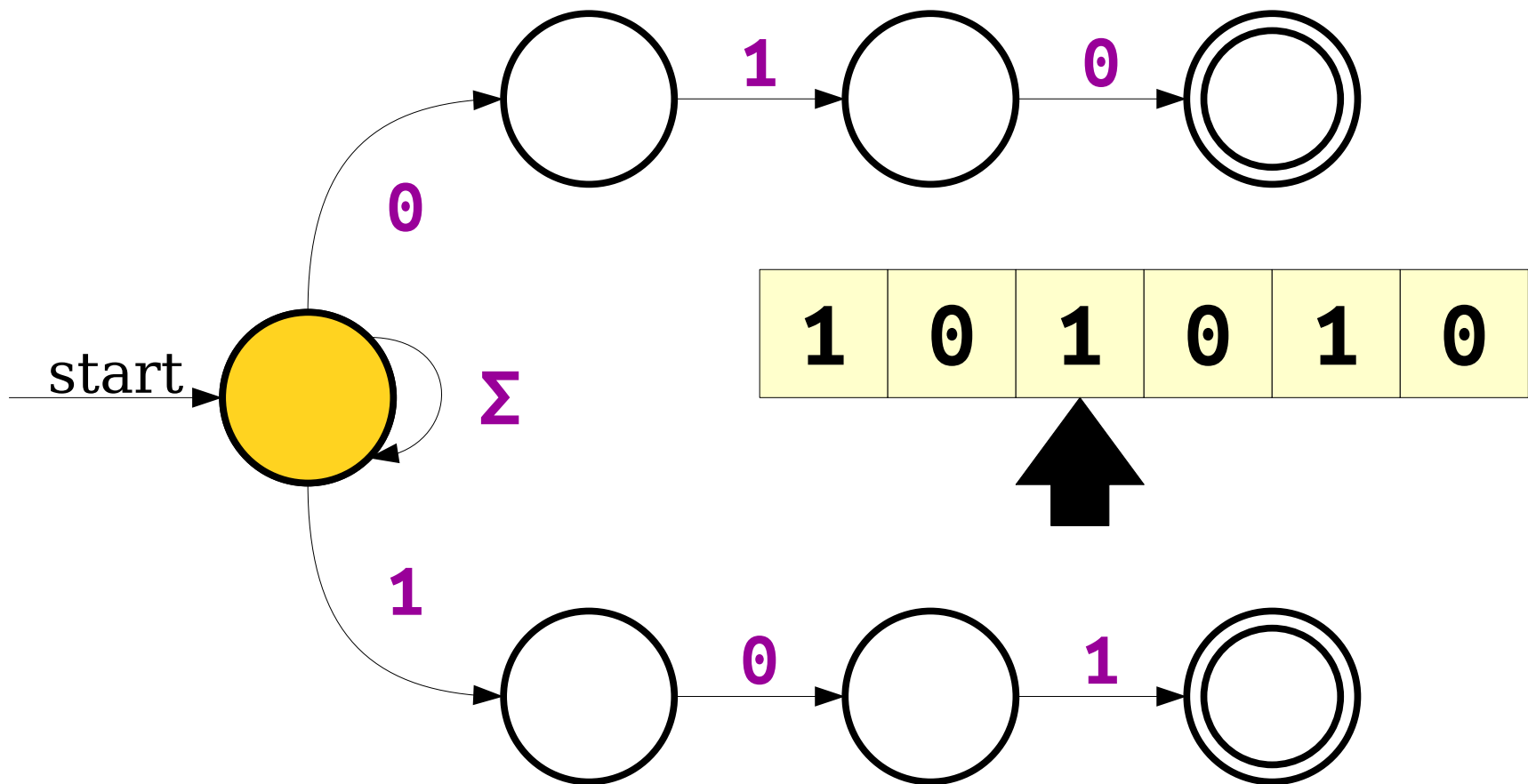
# Guess-and-Check

$$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$$



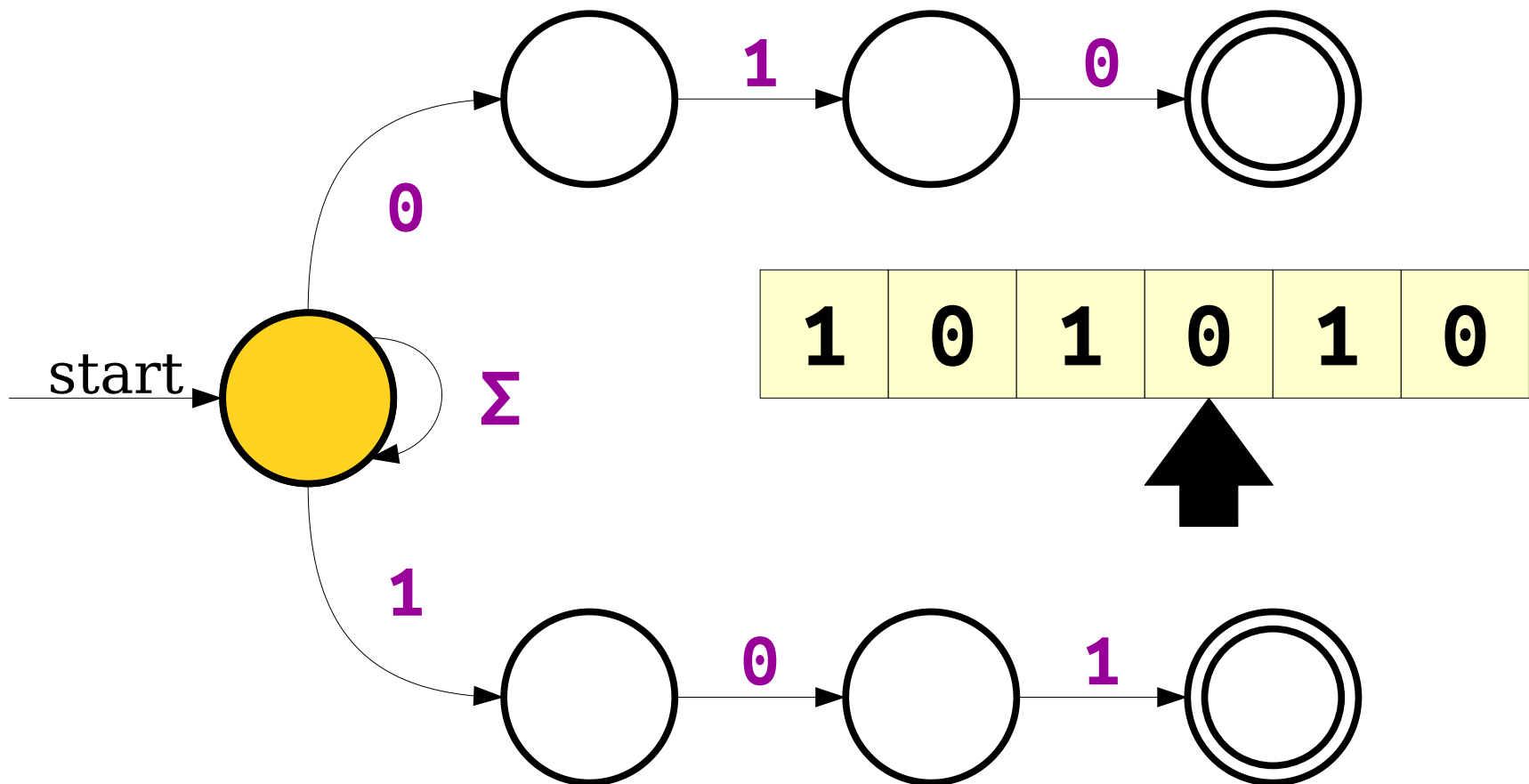
# Guess-and-Check

$$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$$



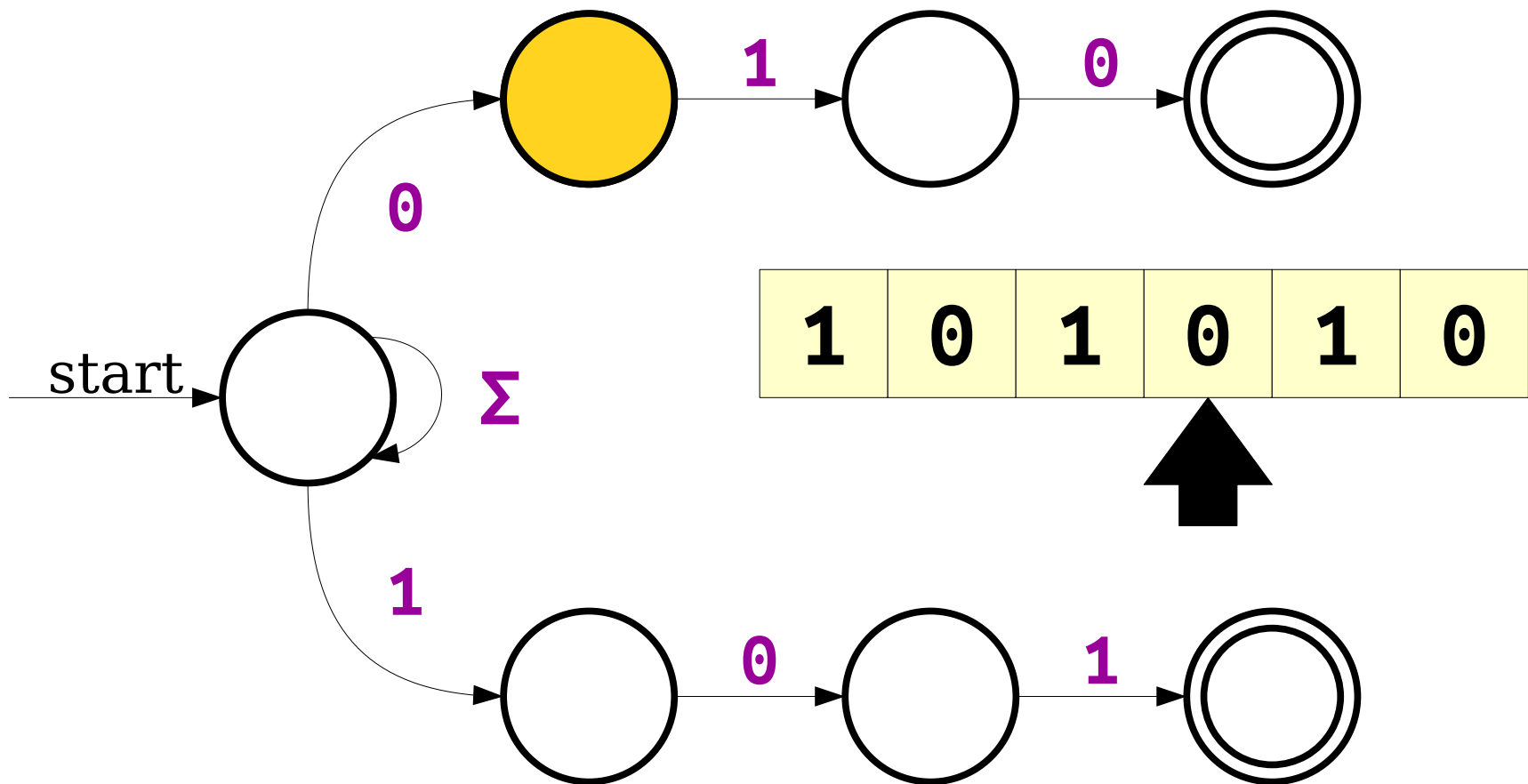
# Guess-and-Check

$$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$$



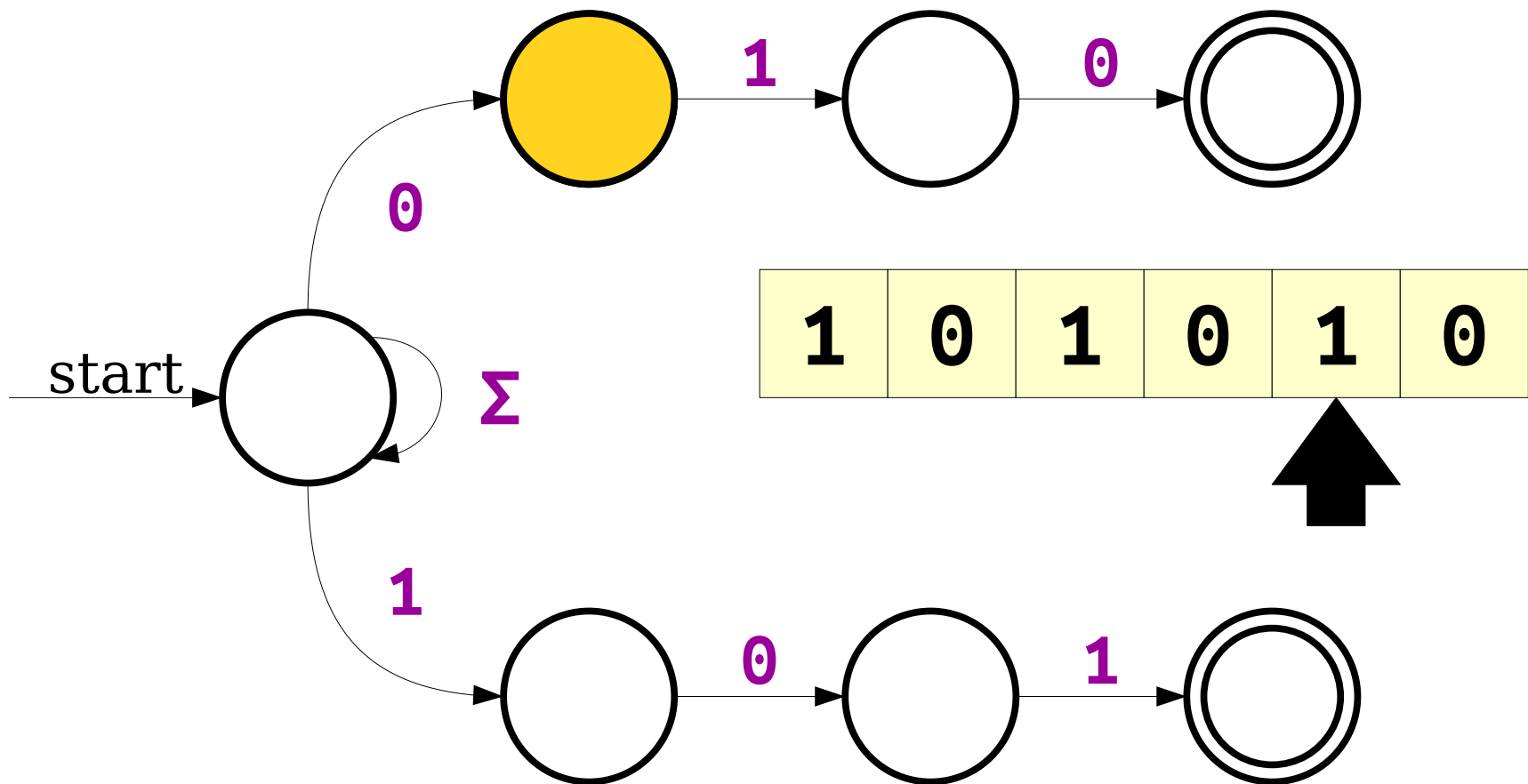
# Guess-and-Check

$$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$$



# Guess-and-Check

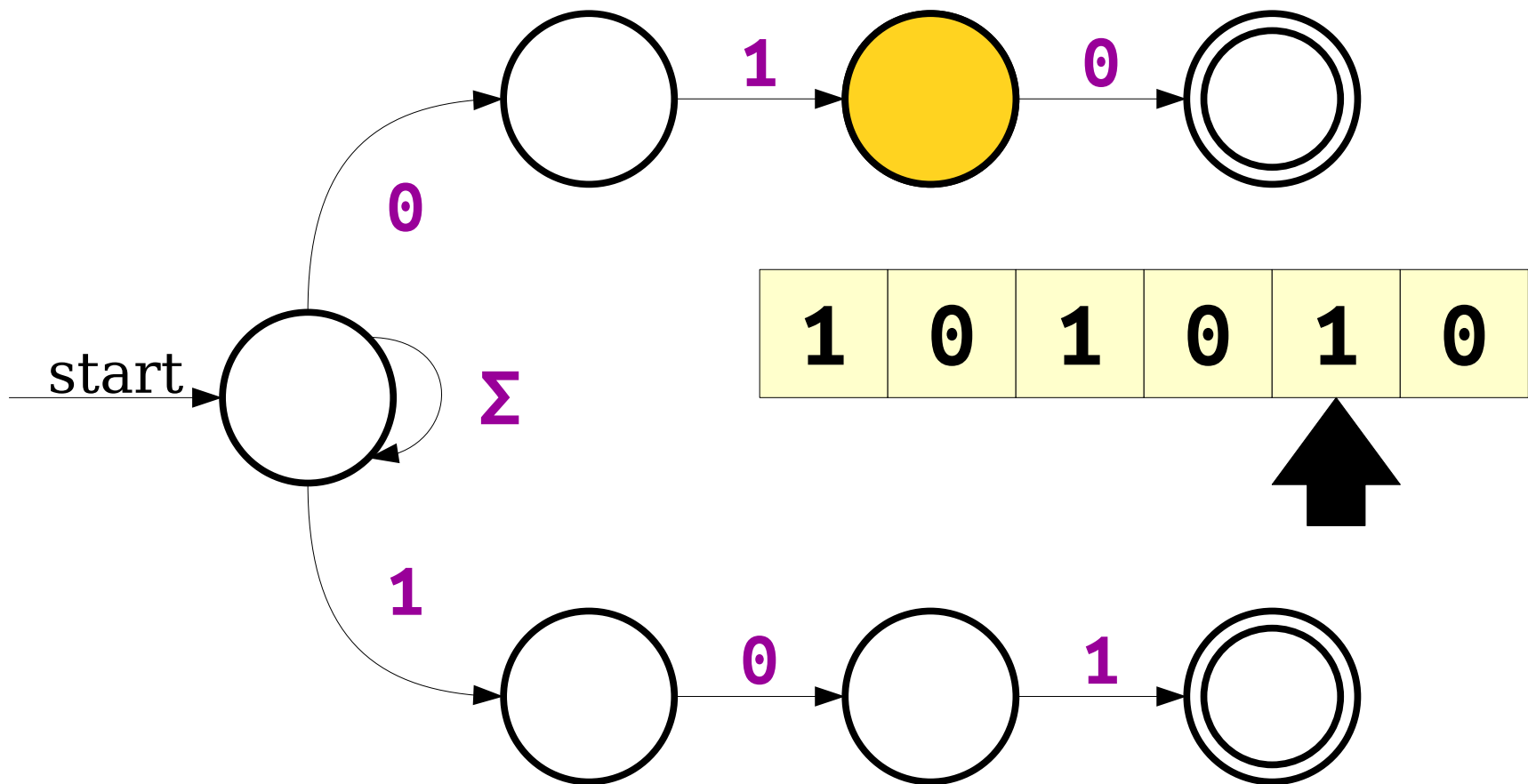
$$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$$





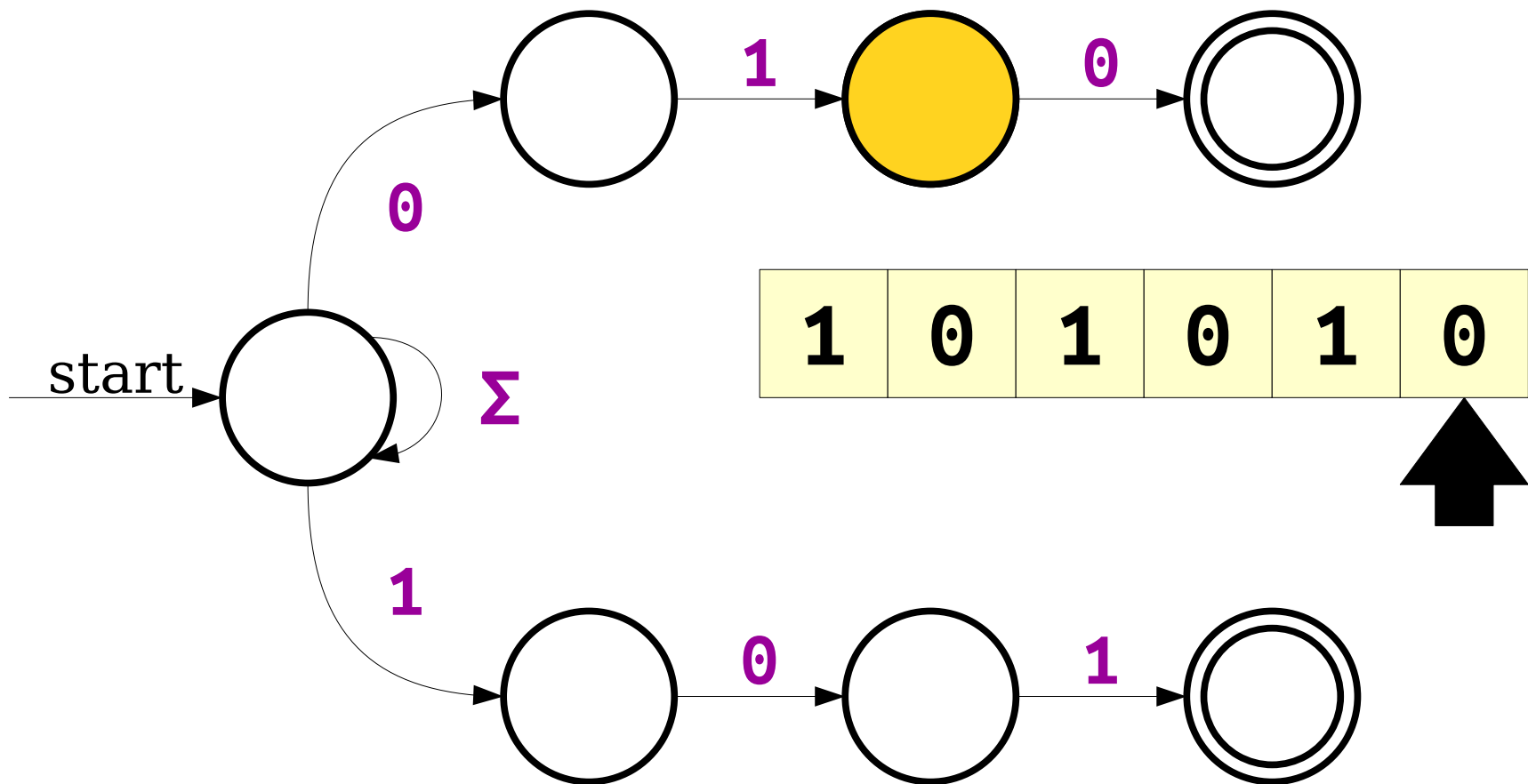
# Guess-and-Check

$$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$$



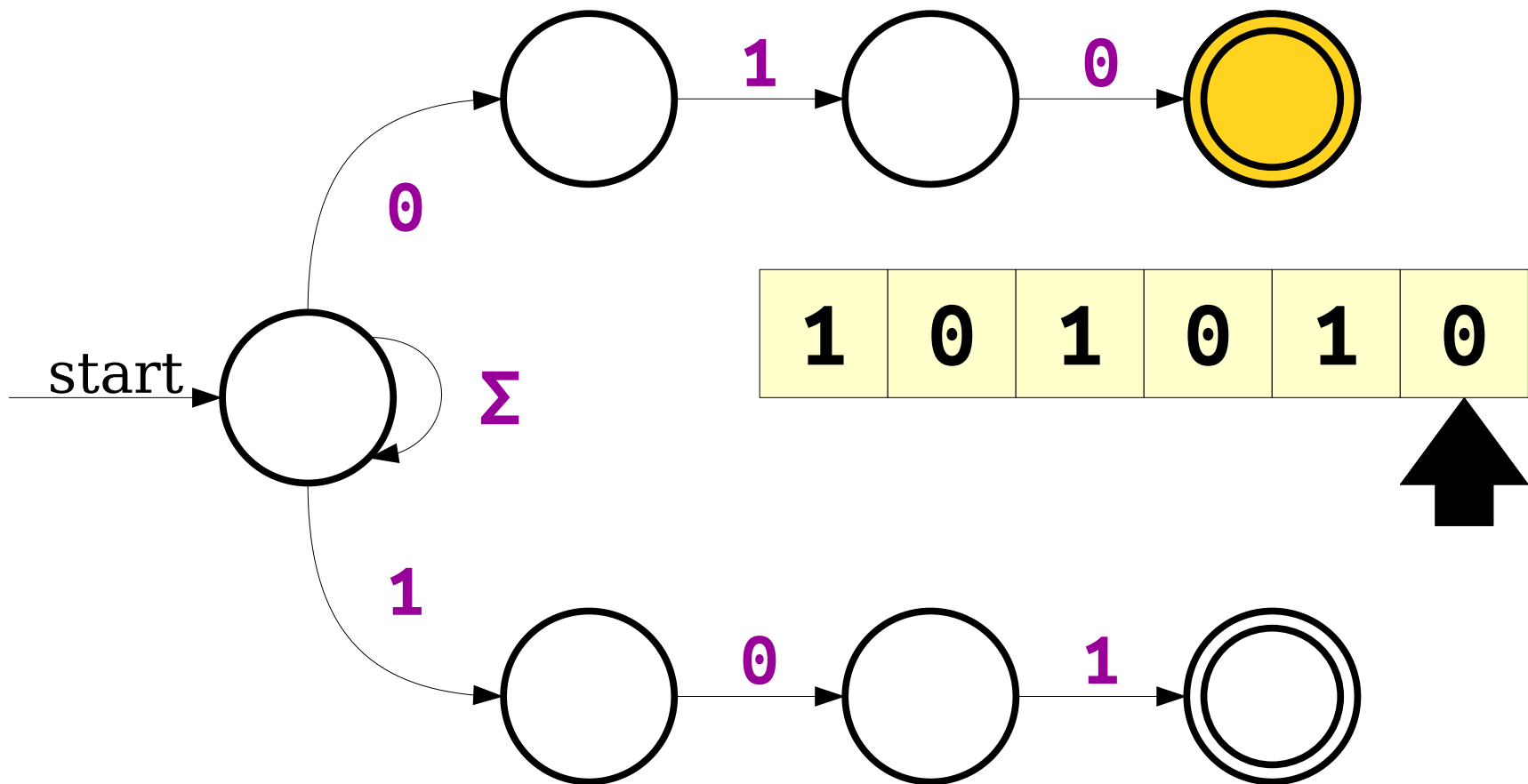
# Guess-and-Check

$$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$$



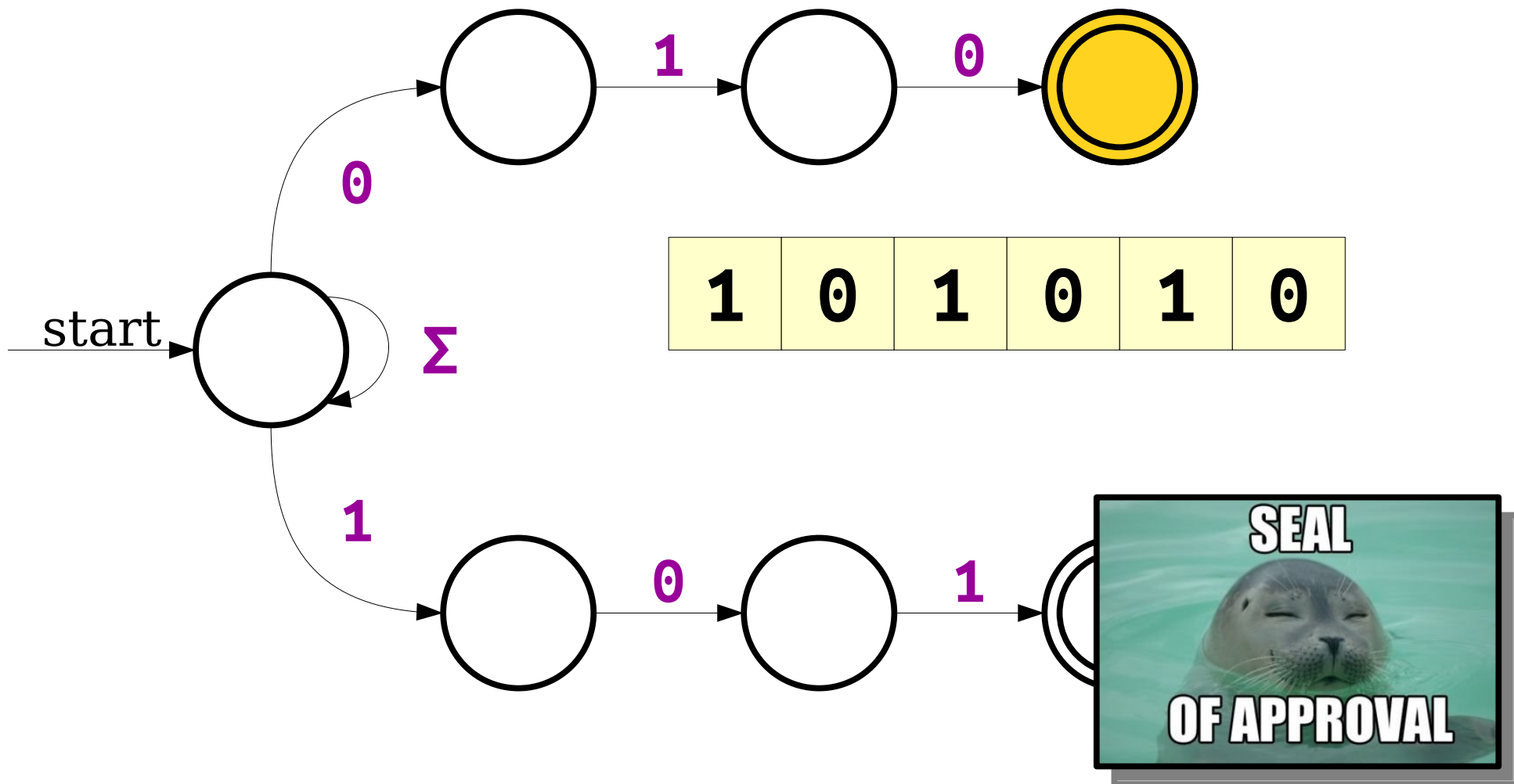
# Guess-and-Check

$$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$$



# Guess-and-Check

$$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$$

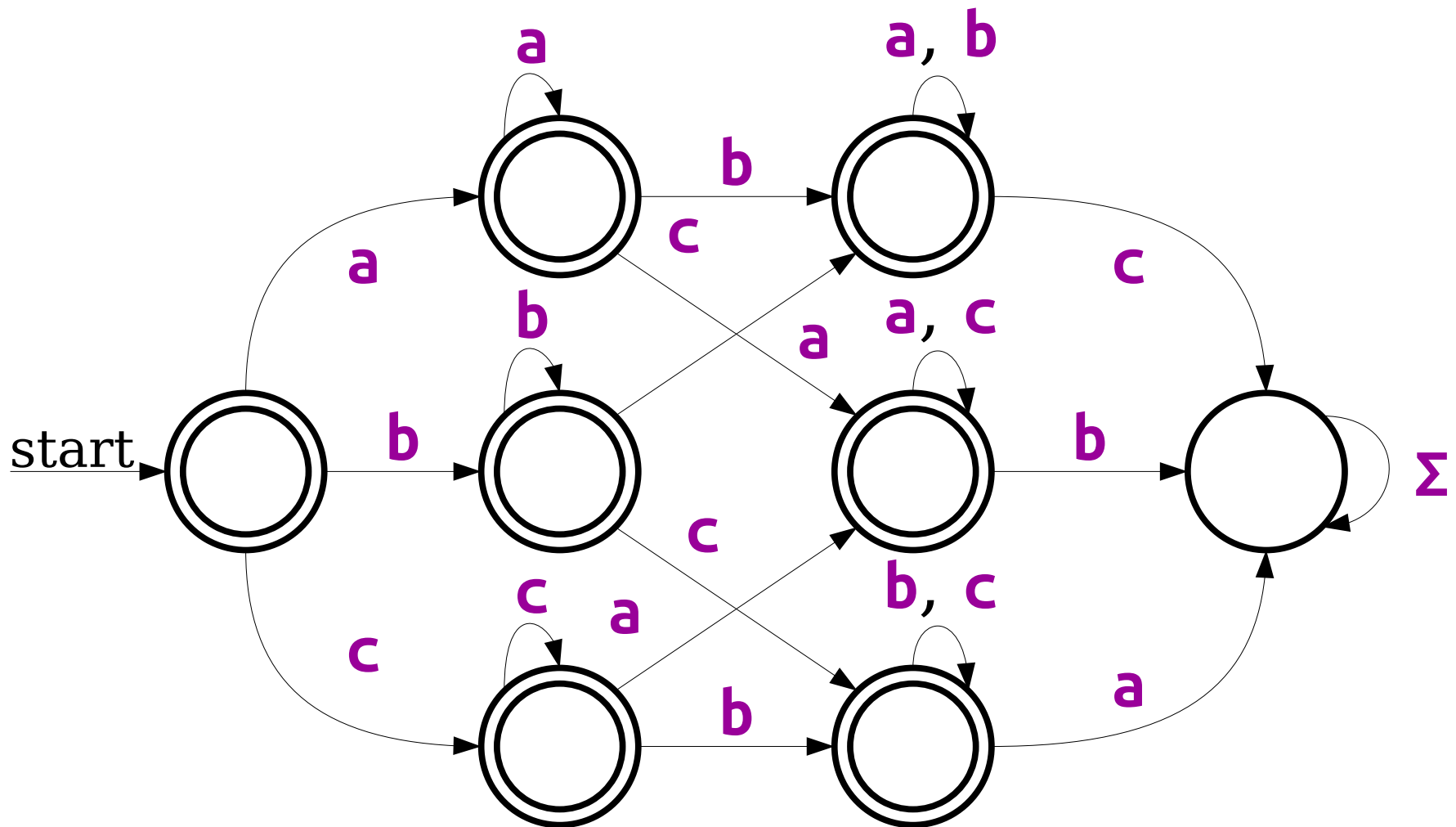


# Guess-and-Check

$L = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid \text{at least one of } \mathbf{a}, \mathbf{b}, \text{ or } \mathbf{c} \text{ is not in } w \}$

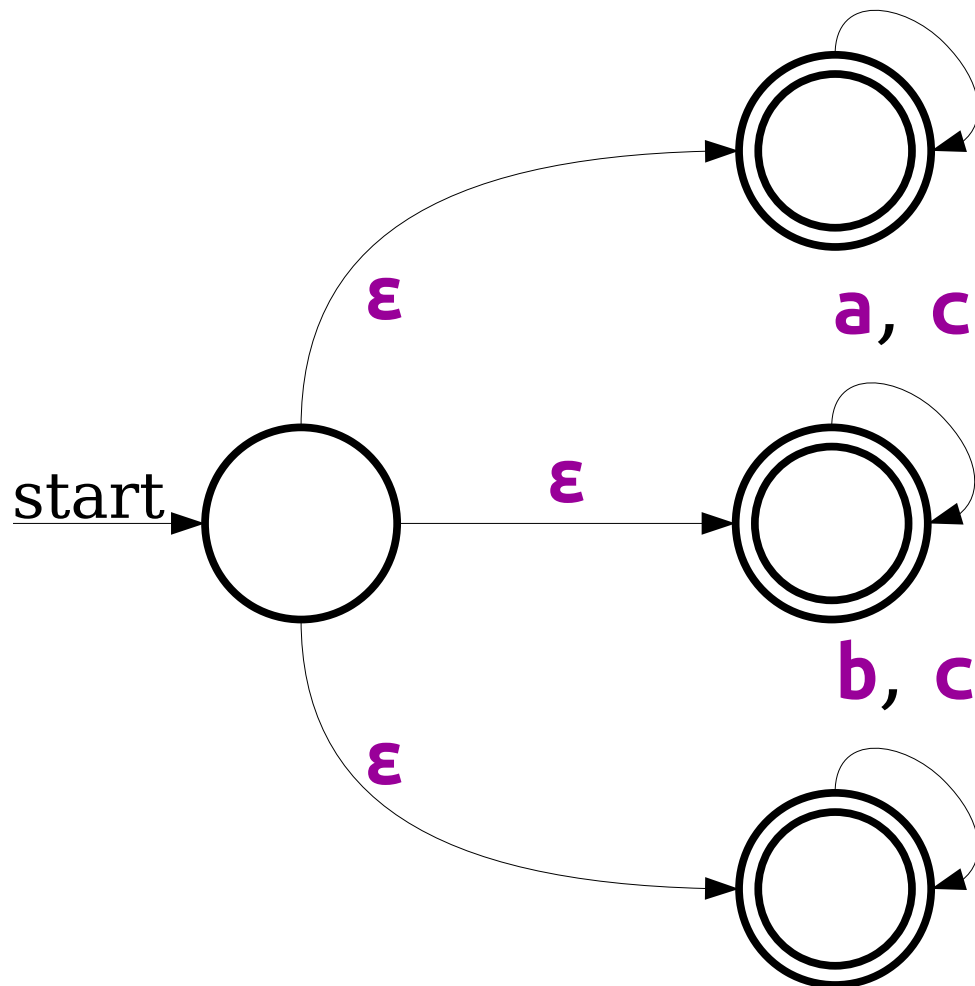
# Guess-and-Check

$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$



# Guess-and-Check

$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$

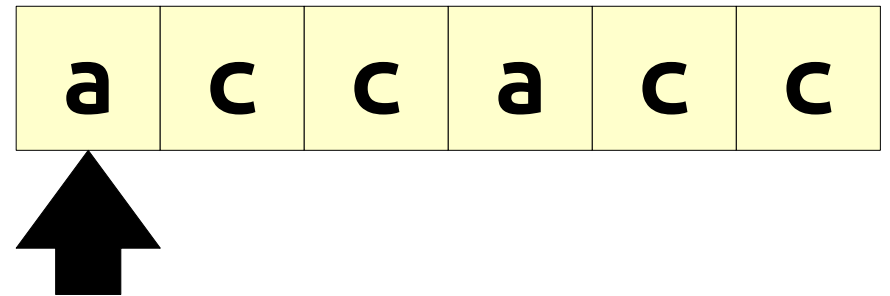
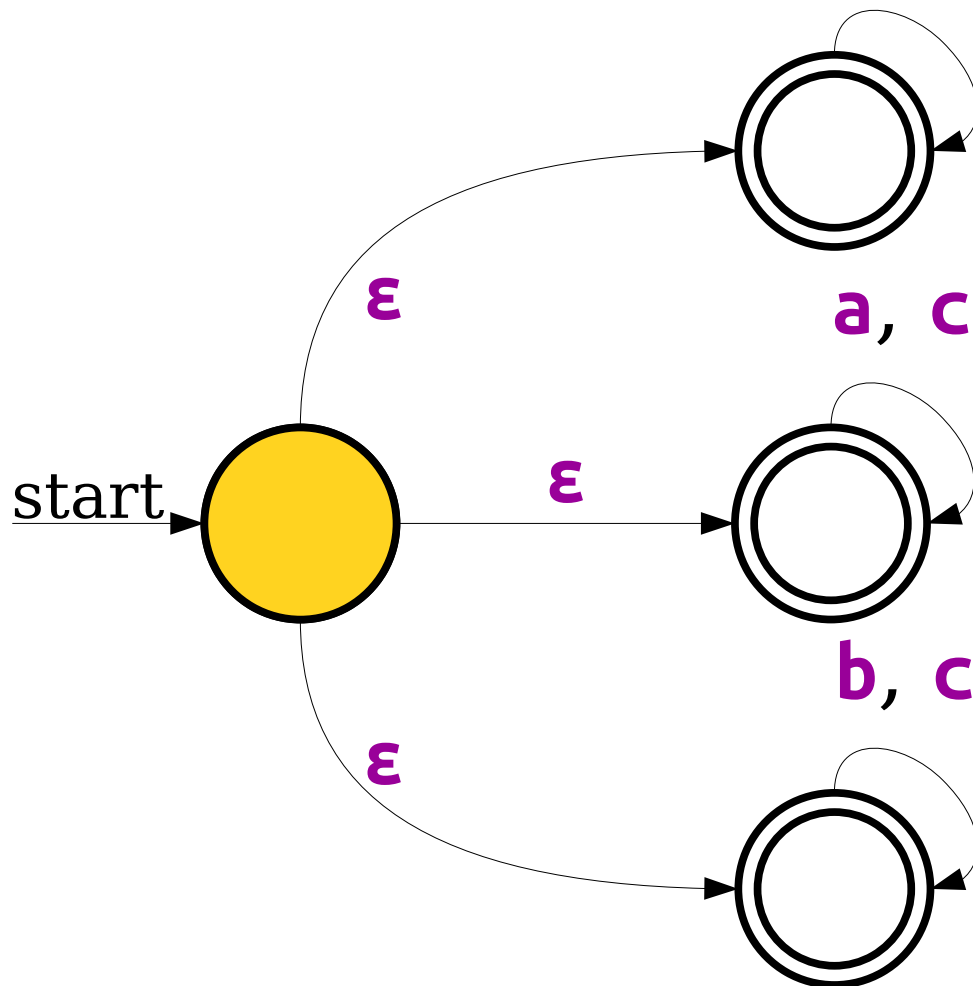


Nondeterministically *guess* which character is missing.

Deterministically *check* whether that character is indeed missing.

# Guess-and-Check

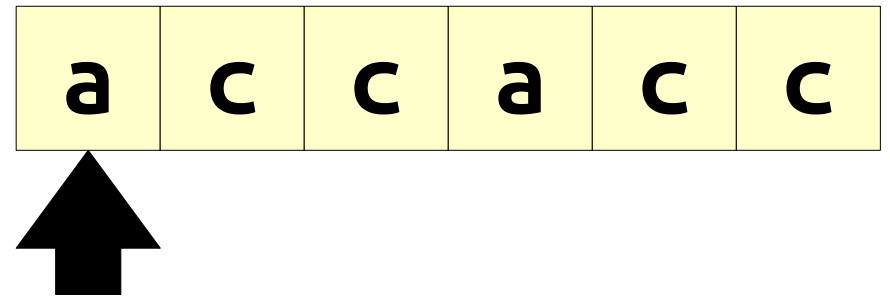
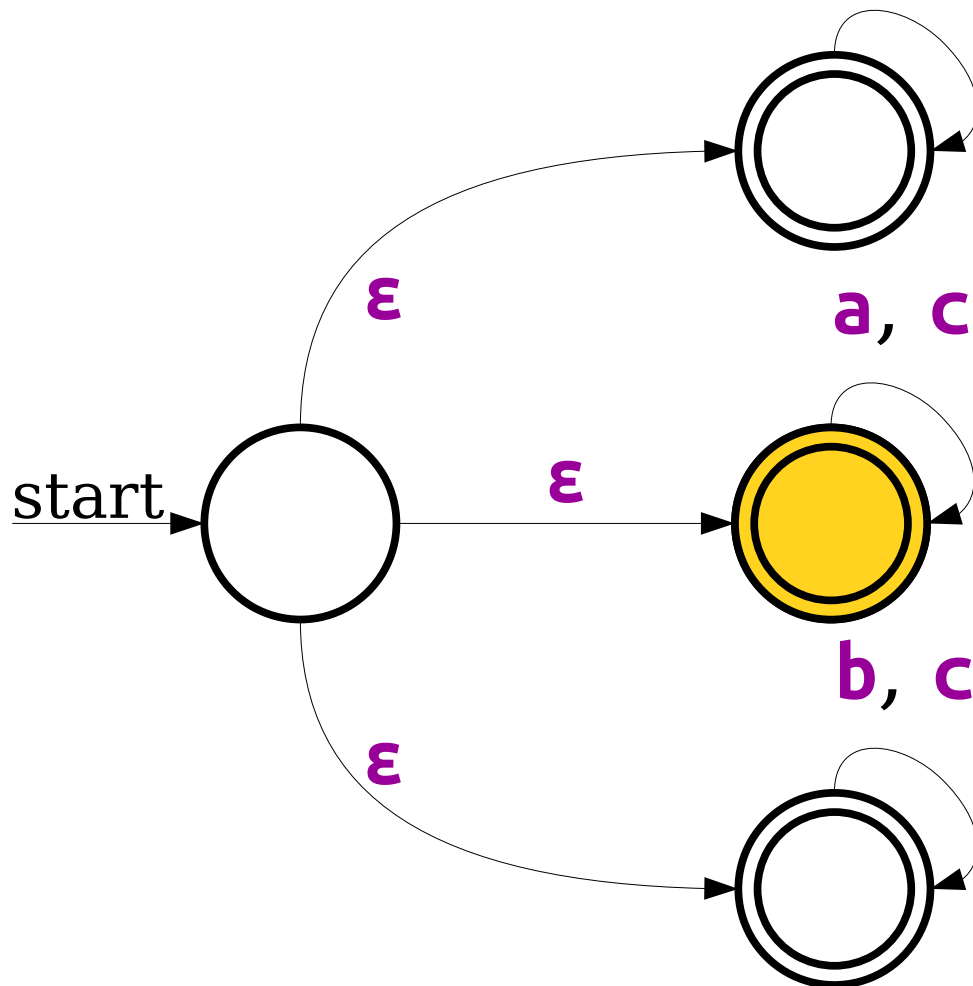
$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$





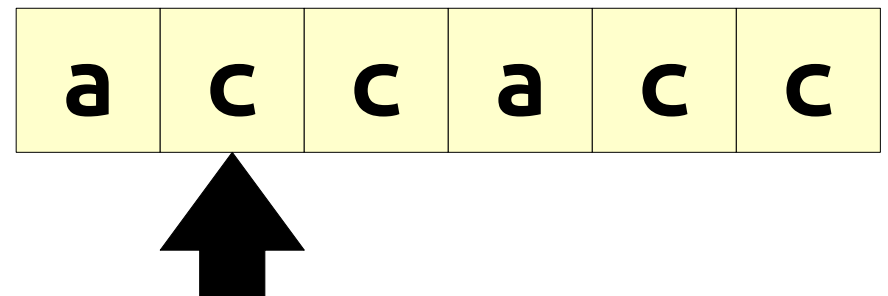
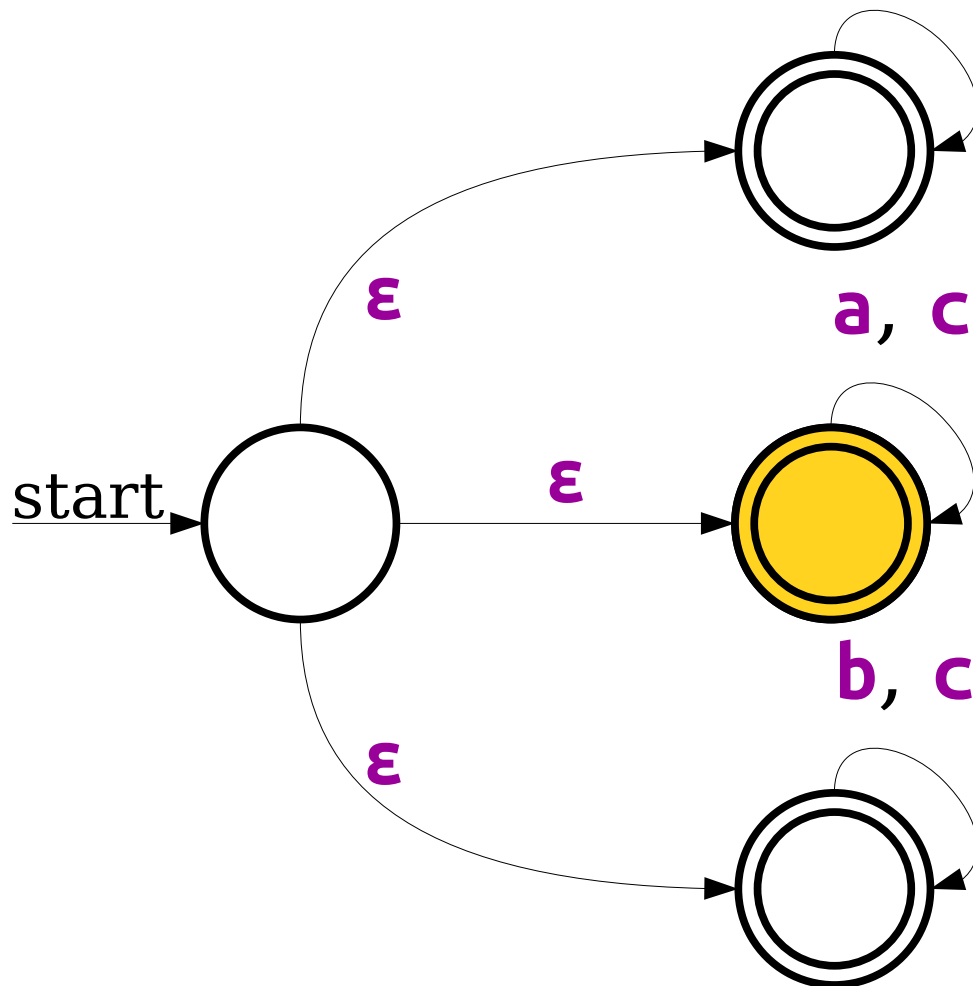
# Guess-and-Check

$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$



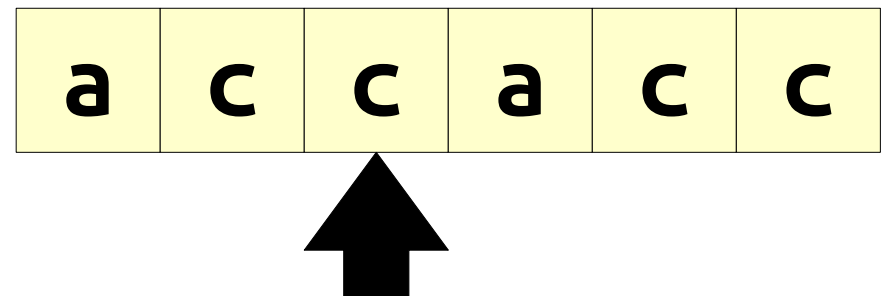
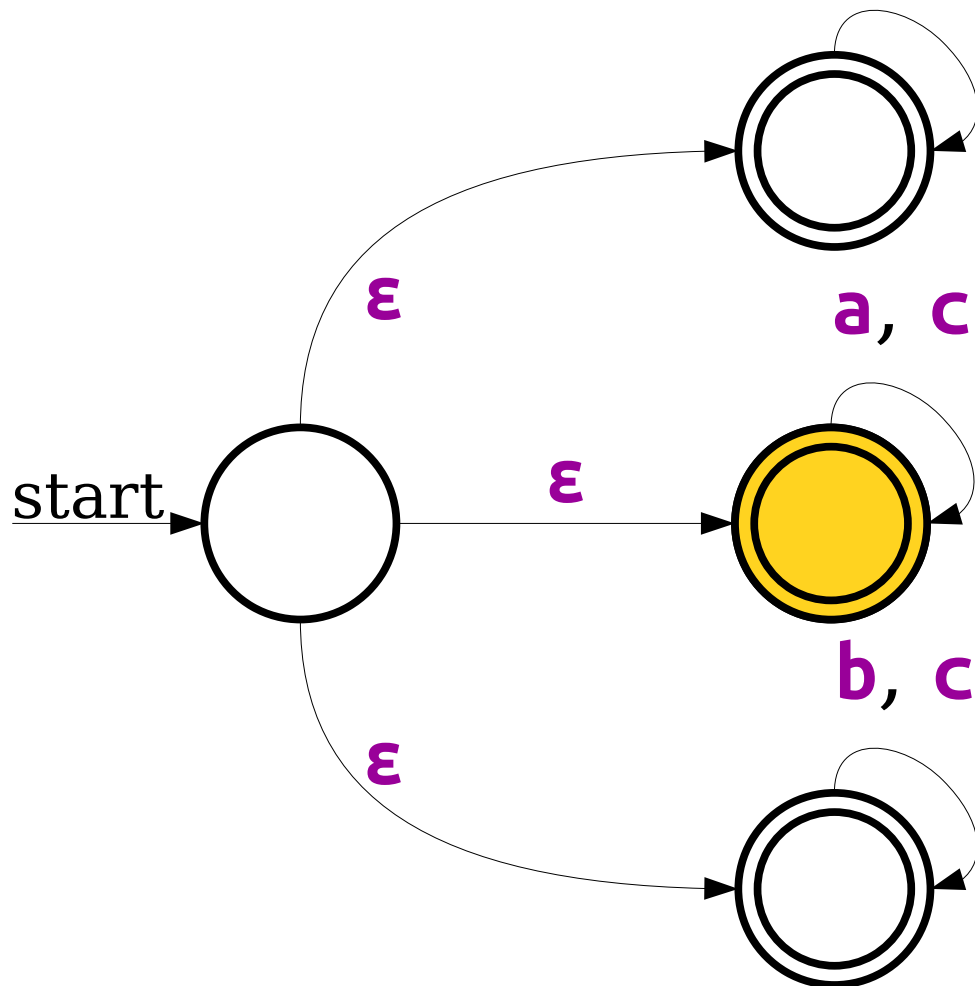
# Guess-and-Check

$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$



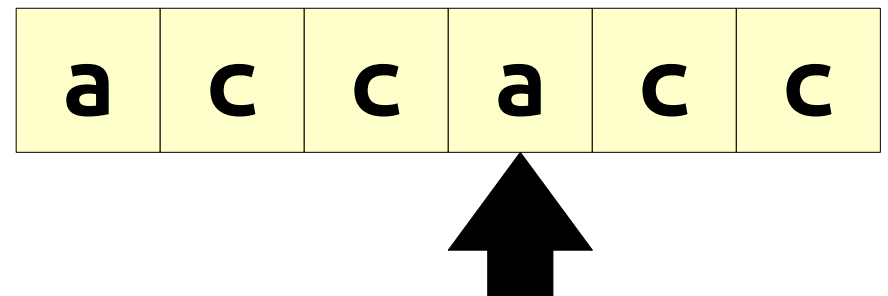
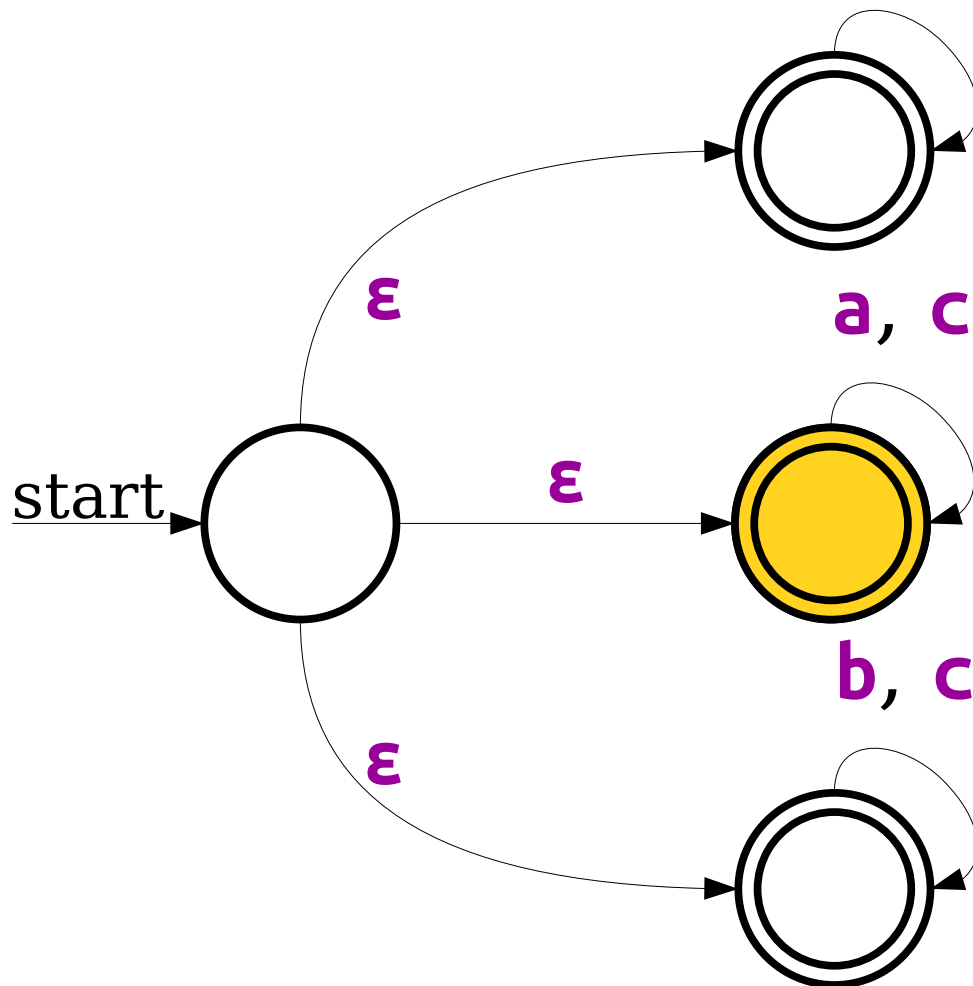
# Guess-and-Check

$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$



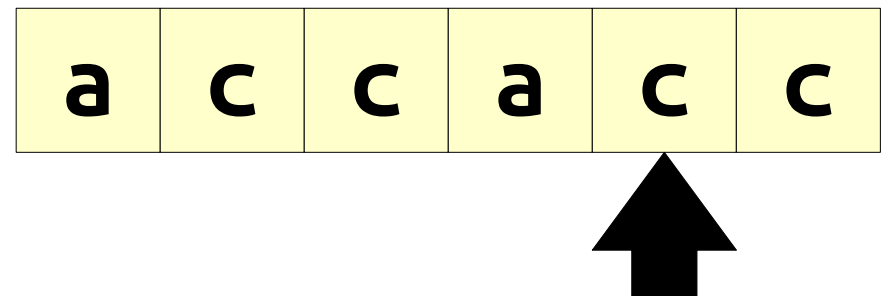
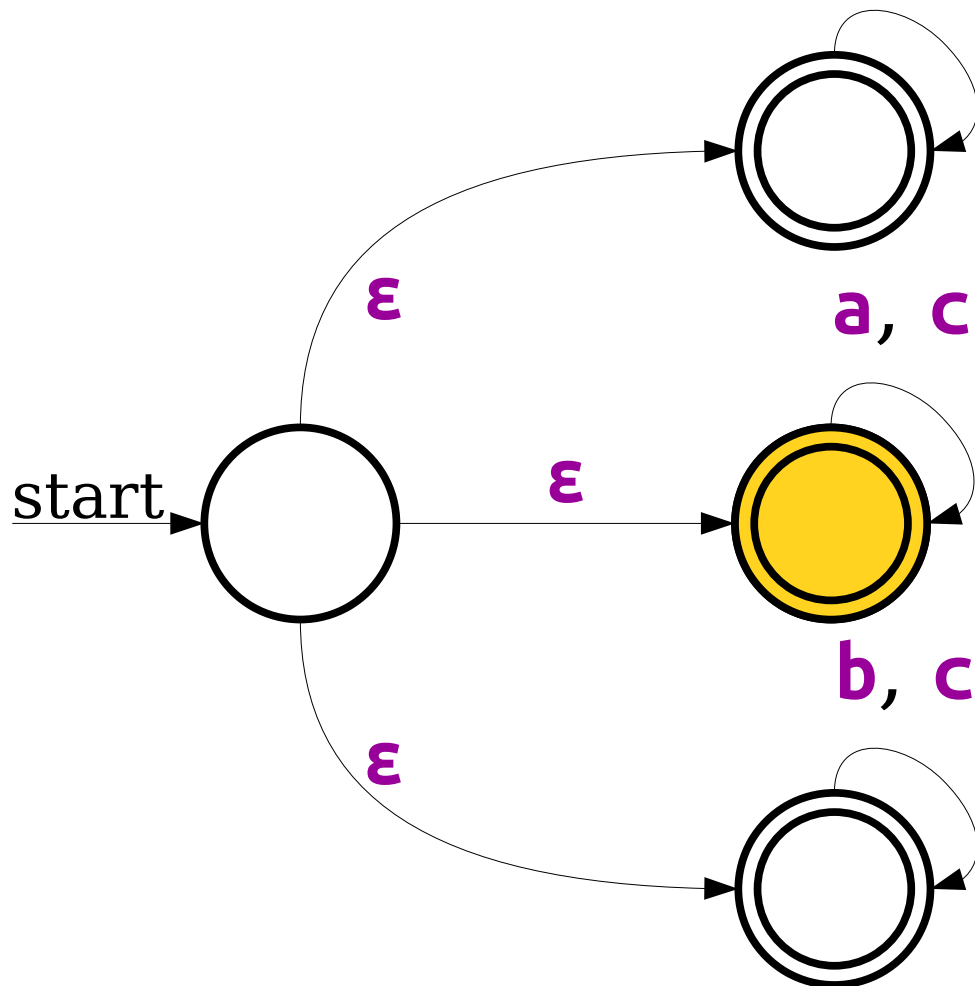
# Guess-and-Check

$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$



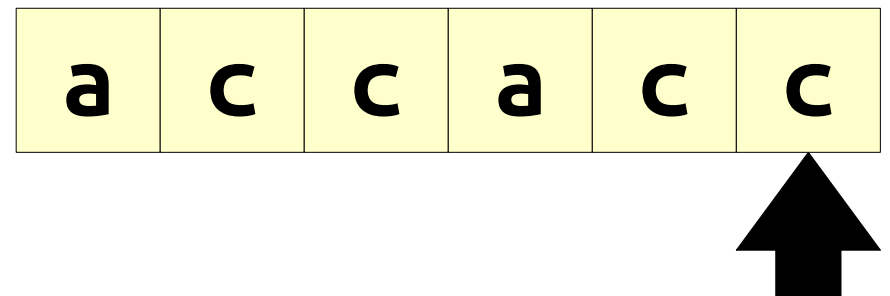
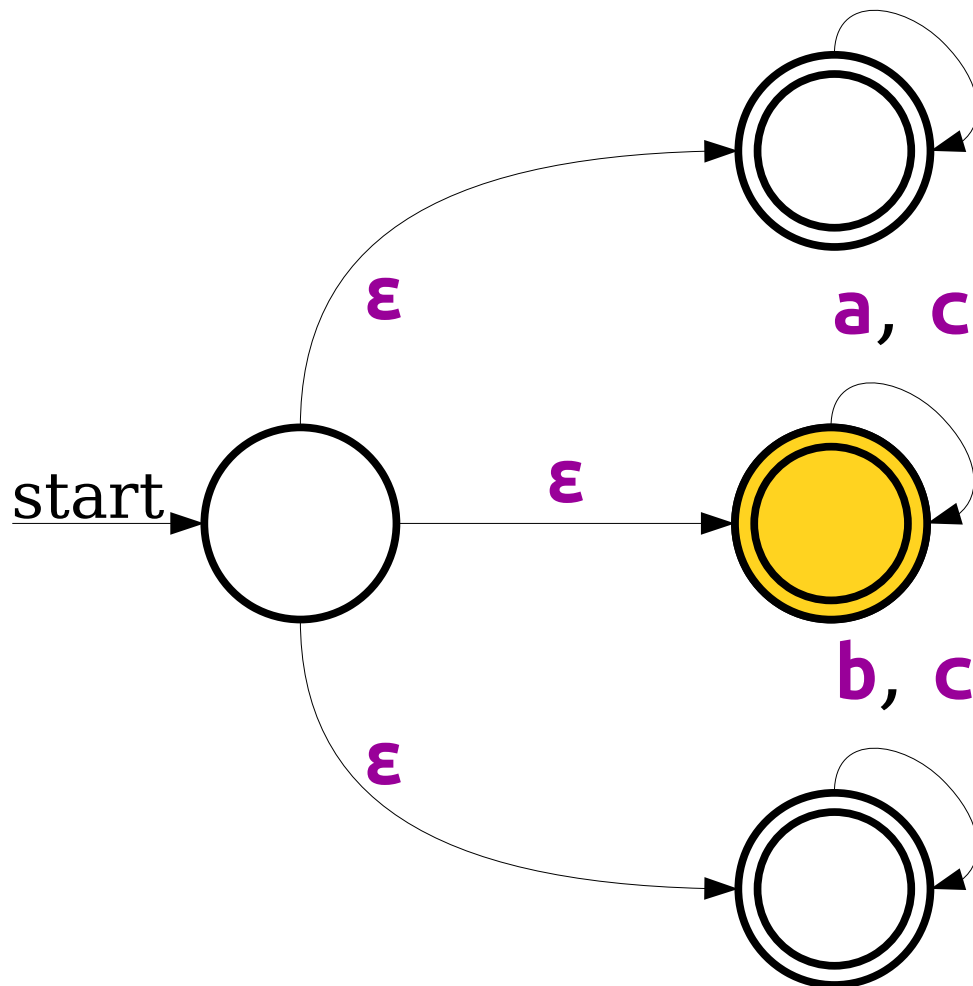
# Guess-and-Check

$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$



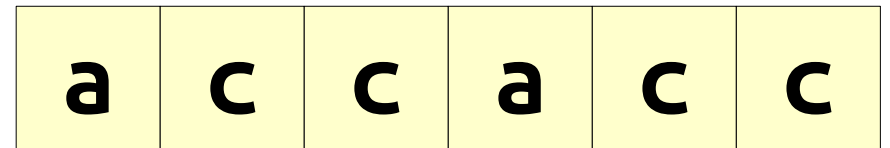
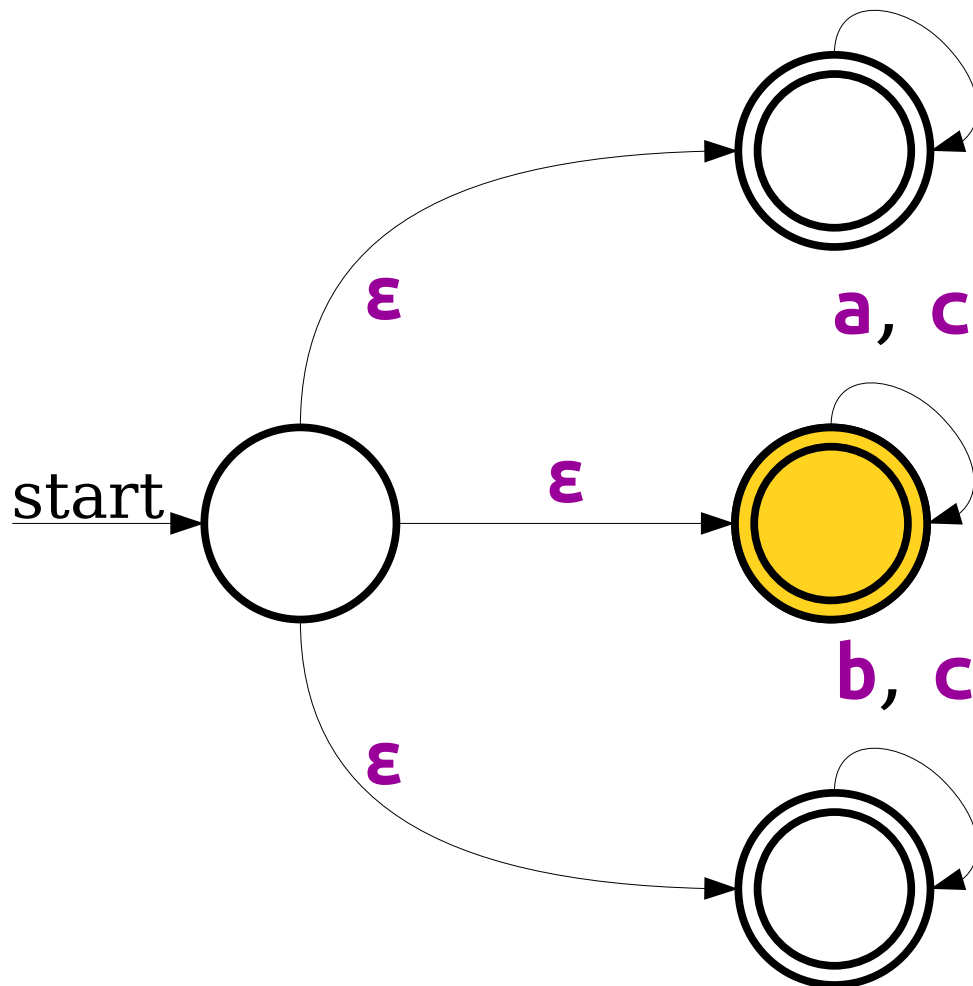
# Guess-and-Check

$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$



# Guess-and-Check

$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$



Just how powerful are NFAs?



# Next Time

- ***The Subset Construction***
  - So beautiful. So elegant. So cool!
- ***More Closure Properties***
  - Transforming languages by transforming machines.
- ***The Kleene Closure***
  - What's the deal with the notation  $\Sigma^*$ ?