Finite Automata

Part Two

Recap from Last Time

Formal Language Theory

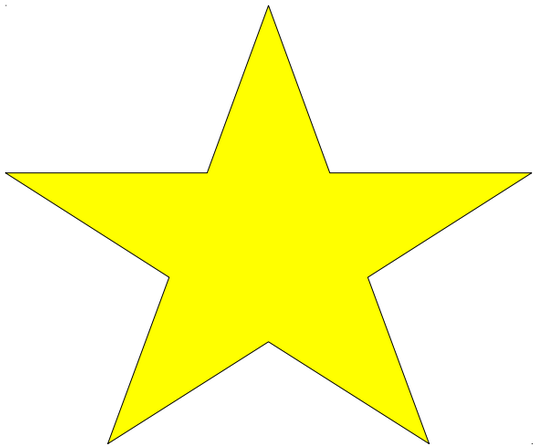
- An ***alphabet*** is a set, usually denoted Σ, consisting of elements called ***characters***.
- A ***string over Σ*** is a finite sequence of zero or more characters taken from Σ.
- The ***empty string*** has no characters and is denoted ε.
- A ***language over Σ*** is a set of strings over Σ.
- The language **Σ*** is the set of all strings over Σ.

DFAs

- A ***DFA*** is a
  - ***D***eterministic
  - ***F***inite
  - ***A***utomaton
- DFAs are the simplest type of automaton that we will see in this course.

DFAs, formally defined

- A DFA consists of:
  - A set of **states** S
  - Exactly one element of the set of states designated as a **start state**
    - (as a consequence, the set of states must be nonempty)
  - A subset of the states designated as **accepting states**
  - A set of characters, the **alphabet** Σ
  - A **transition** function that maps (state, character) ordered pairs to states $\delta : (S \times \Sigma) \rightarrow S$



See how DFA formal definition draws on concepts we've learned about so far!

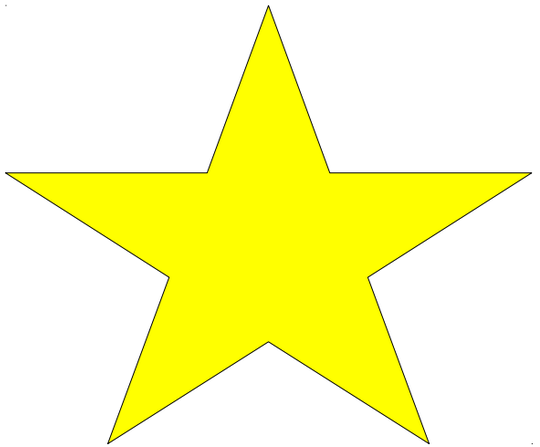
- A DFA consists of:
  - A **set** of states *S*
  - Exactly one **element of the set** of states designated as a start state
    - (as a consequence, the set of states must be nonempty)
  - A **subset** of the states designated as accepting states
  - A **set** of characters, the alphabet Σ
  - A transition **function** that maps (state, character) ordered pairs to states

DFAs, formally defined

- A DFA consists of:
  - A set of states S
  - Exactly one element of the set of as a start state
    - (as a consequence, the set of states must be nonempty)
  - A subset of the states designated as accepting states
  - A set of characters, the alphabet Σ
  - A transition function that maps (state, character) ordered pairs to states $\delta : (S \times \Sigma) \rightarrow S$

The Cartesian product of two sets *A* and *B*, denoted $A \times B$, is the set of all ordered pairs drawn from *A* and *B*, as shown in this set builder notation:

$A \times B = \{ (a,b) \mid a \in A \text{ and } b \in B \}$



DFAs, formally defined

- A DFA consists of:
  - A set of states S
  - Exactly one element of the set of as a start state
    - (as a consequence, the set of states must be nonempty)
  - A subset of the states designated as accepting states
  - A set of characters, the alphabet Σ
  - A transition function that maps (state, character) ordered pairs to states $\delta : (S \times \Sigma) \to S$

The Cartesian product of two sets A and B, denoted $A \times B$, is the set of all ordered pairs drawn from A and B, as shown in this set builder notation:

$A \times B = \{ (a,b) \mid a \in A \text{ and } b \in B \}$

If $S = \{q_0, q_1, q_2\}$ and $\Sigma = \{0, 1\}$, what is the **domain** of the function δ?

PollEv.com/cs103spr26

DFAs, formally defined

- A DFA consists of:
  - A set of states S
  - Exactly one element of the set of as a start state
    - (as a consequence, the set of states must be nonempty)
  - A subset of the states designated as accepting states
  - A set of characters, the alphabet Σ
  - A transition function that maps (state, character) ordered pairs to states $\delta : (S \times \Sigma) \to S$

The Cartesian product of two sets A and B, denoted $A \times B$, is the set of all ordered pairs drawn from A and B, as shown in this set builder notation:

$A \times B = \{ (a,b) \mid a \in A \text{ and } b \in B \}$

If $S = \{q_0, q_1, q_2\}$ and $\Sigma = \{0, 1\}$, then
$\delta : \{(q_0, 0), (q_0, 1), (q_1, 0), (q_1, 1), (q_2, 0), (q_2, 1)\} \to \{q_0, q_1, q_2\}$.

PollEv.com/
cs103spr26

The Language of an Automaton

- If D is a DFA that processes strings over Σ, the ***language of D***, denoted $\mathscr{L}(D)$, is the set of all strings D accepts.
- Formally:

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

New Stuff!

The Regular Languages

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

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

The Complement of a Language

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

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

The Complement of a Language

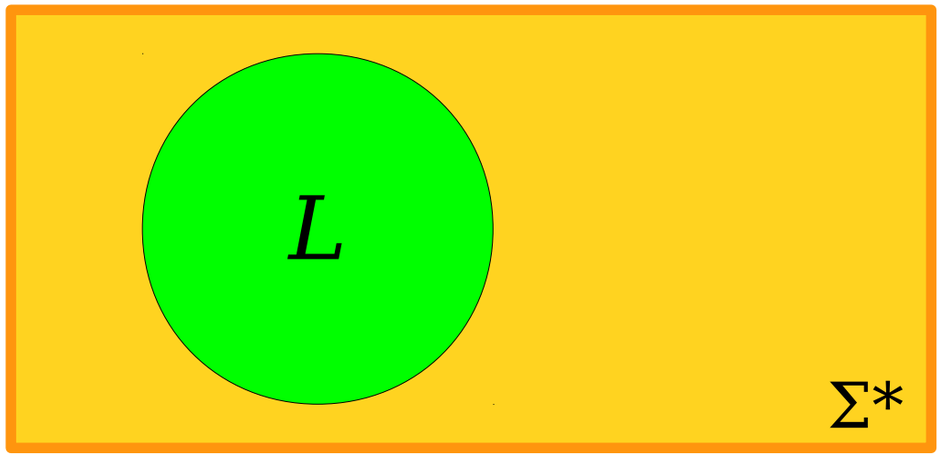
- Given a language $L \subseteq \Sigma^*$, the ***complement*** of that language (denoted $\overline{L}$) is the language of all strings in Σ^* that aren't in L.
- Formally:

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

The Complement of a Language

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

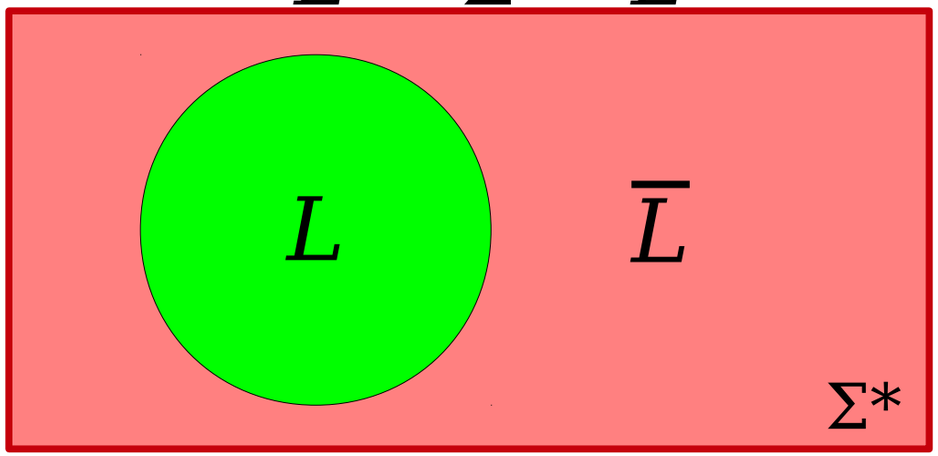
$$\overline{L} = \Sigma^* - L$$



The Complement of a Language

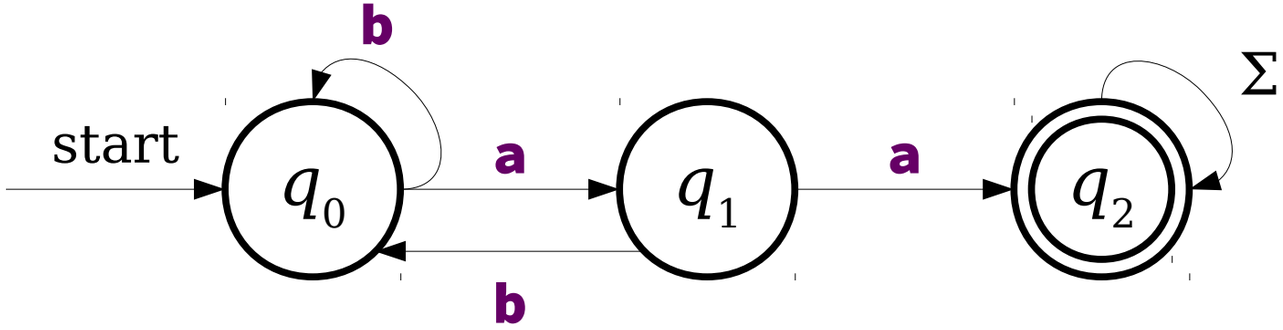
- Given a language $L \subseteq \Sigma^*$, the ***complement*** of that language (denoted $\overline{L}$) is the set of all strings in Σ^* that aren't in L. (The complement of a language is also a language.)
- Formally:

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

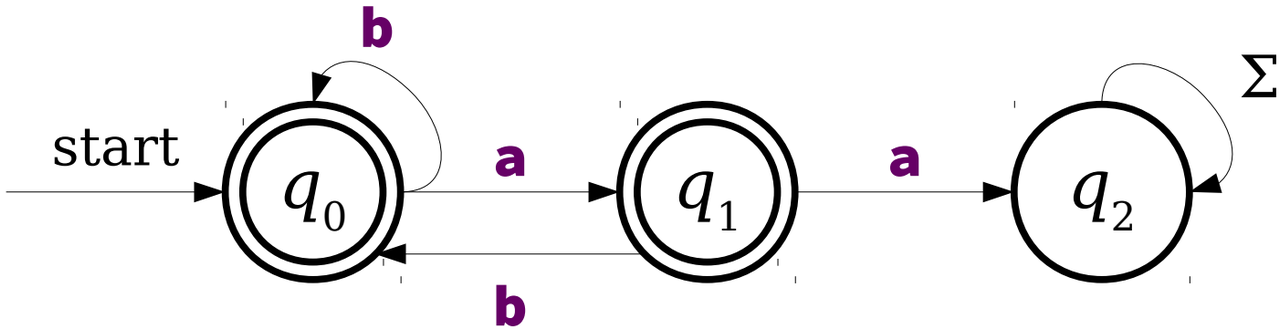


Complementing Regular Languages

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

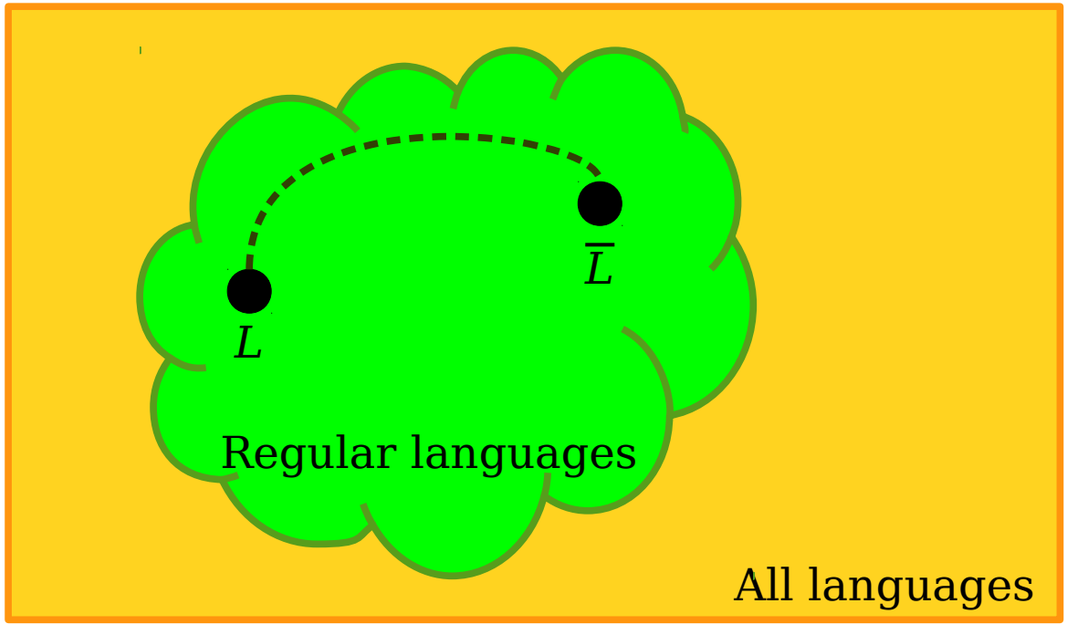


$\overline{L} = \{ w \in \{\mathbf{a}, \mathbf{b}\}^* \mid w$ ***does not*** contain **aa** as a substring $\}$



Closure Properties

- ***Theorem:*** If L is a regular language, then $\overline{L}$ is also a regular language.
  - (We haven't formally proved this, but you may assume it's true in this class.)
- As a result, we say that the regular languages are ***closed under complementation***.



Closure Properties

- ***Theorem:*** If L is a regular language, then $\overline{L}$ is also a regular language.
- ***Possible theorem:*** If L is not a regular language, then $\overline{L}$ is also not a regular language.
  - Answer in pollev: Is this true?
  - In discussion: prove or disprove.

PollEv.com/
cs103spr26

Closure Properties

- ***Theorem:*** If L is a regular language, then $\overline{L}$ is also a regular language.
- ***Theorem:*** For all languages L, if L is not a regular language, then $\overline{L}$ is also not a regular language.
- ***Proof:*** Assume for the sake of contradiction that ...

What is the negation that we are assuming?

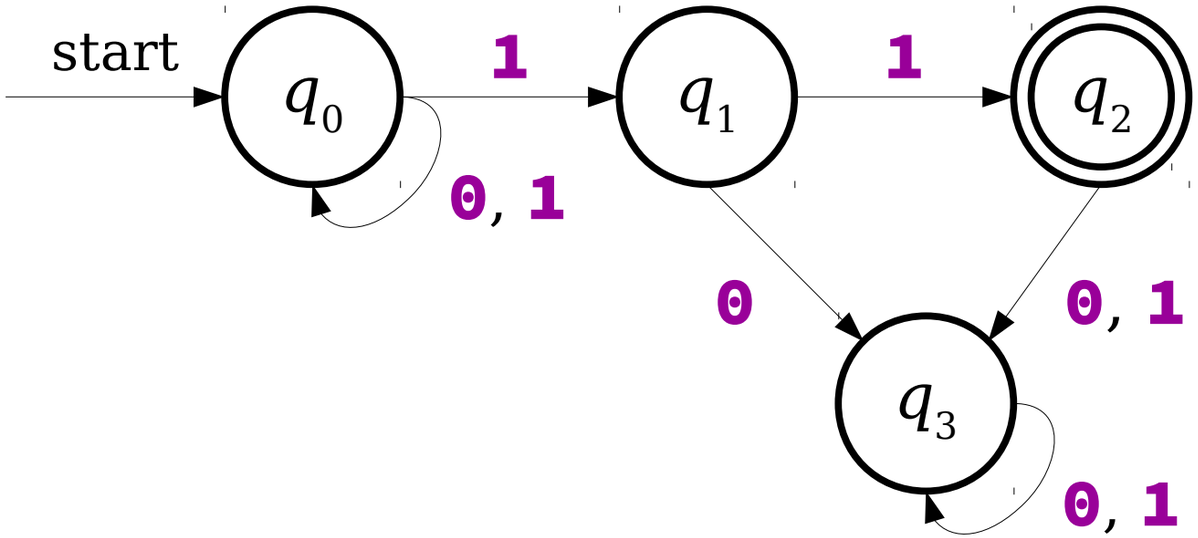
PollEv.com/
cs103spr26

Closure Properties

- ***Theorem:*** If L is a regular language, then $\overline{L}$ is also a regular language.
- ***Theorem:*** For all languages L, if L is not a regular language, then $\overline{L}$ is also not a regular language.
- ***Proof:*** Assume for the sake of contradiction that **there exists a language L that is not regular, but its complement $\overline{L}$ is regular.** Since $\overline{L}$ is regular, the complement of $\overline{L}$ must also be regular, by the first theorem. But the complement of $\overline{L}$ is L, which by assumption is not regular, a contradiction. So the assumption must be false and the set of non-regular languages is closed under complement.

NFAs

Revisiting a Problem



NFAs

- An ***NFA*** is a
  - ***N***ondeterministic
  - ***F***inite
  - ***A***utomaton
- 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.
- Represents a fundamental shift in how we'll think about computation.

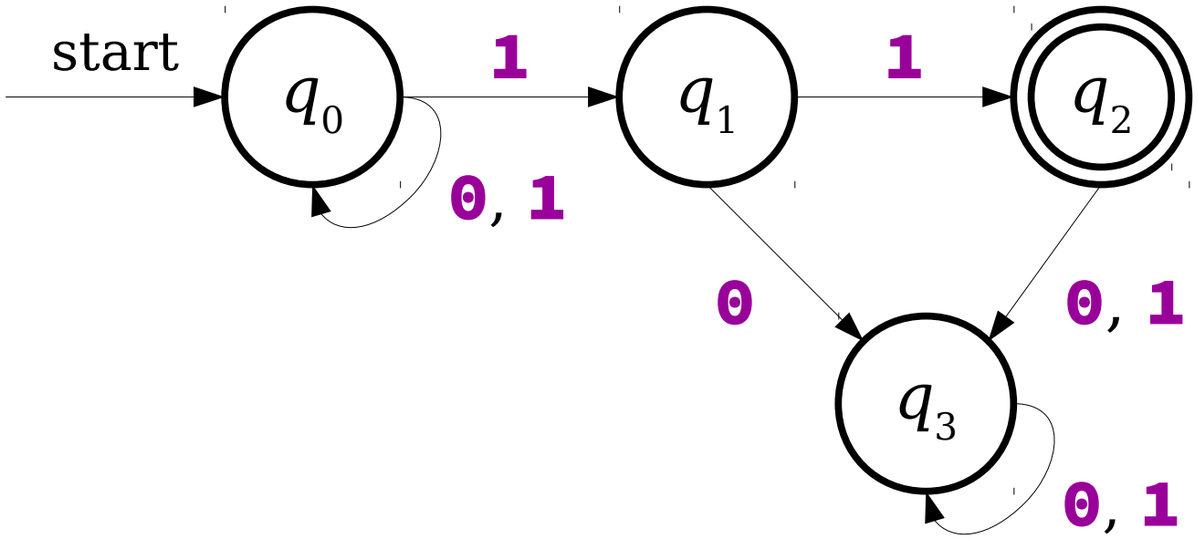
NFA

- State structure similar to a DFA
- Different **transition function**:
  - DFA: $\delta : (S \times \Sigma) \rightarrow S$
    - *always go to exactly one state*
  - NFA: $\delta : (\wp(S) \times \Sigma) \rightarrow \wp(S)$
    - *could go to one, many, or none!*
- Different accept condition:
  - Accepts if ***there exists*** a series of choices that ends in an accepting state.

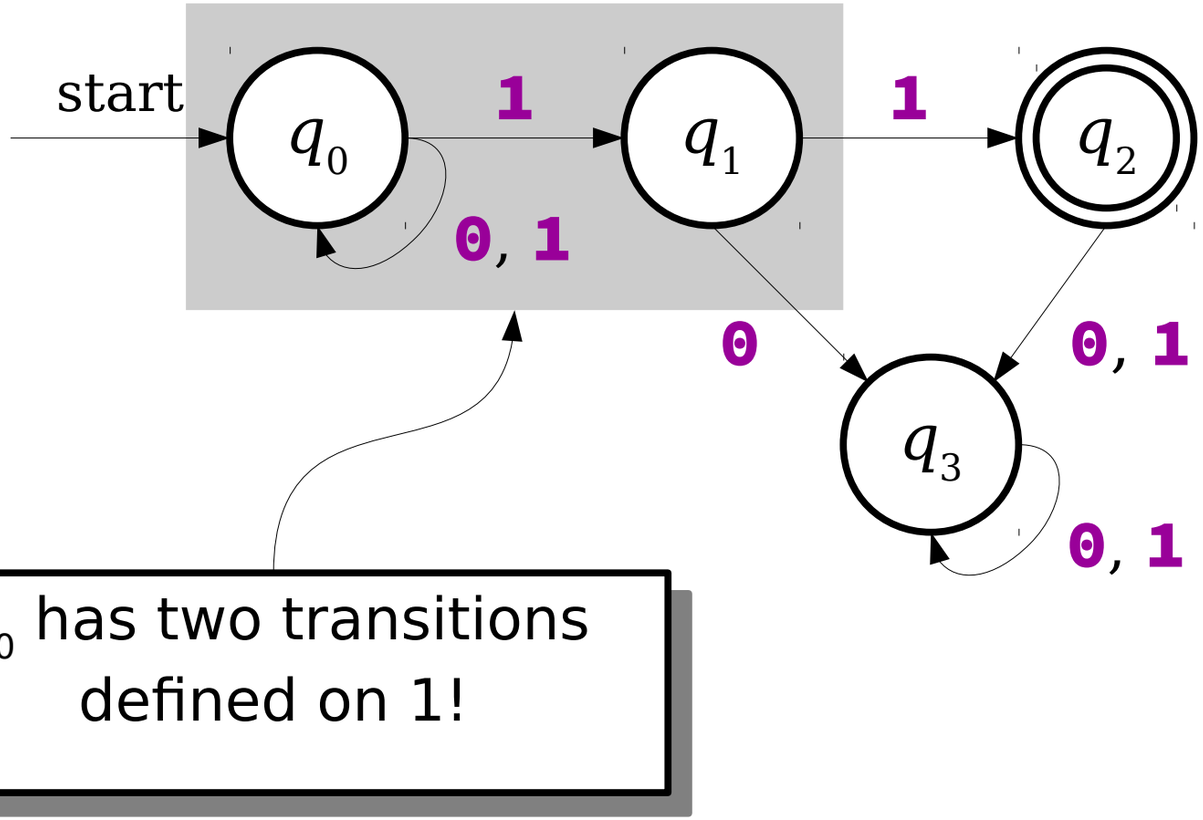
PollEv.com/
cs103spr26

If $S = \{q_0, q_1\}$ and $\Sigma = \{0, 1\}$, what is the **co-domain** of the function δ for **NFA**?

A Simple NFA

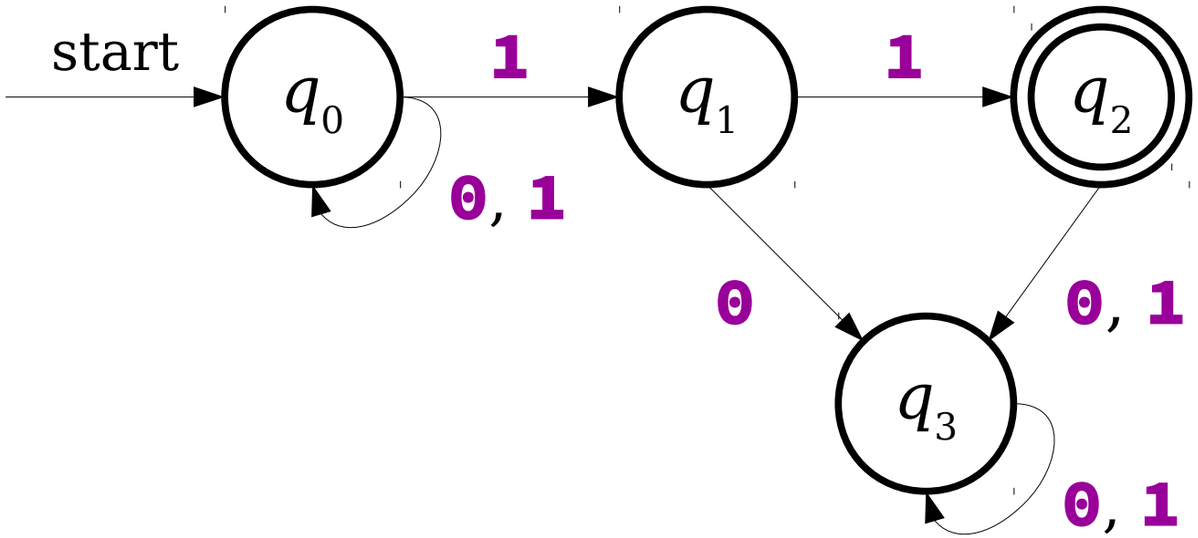


A Simple NFA



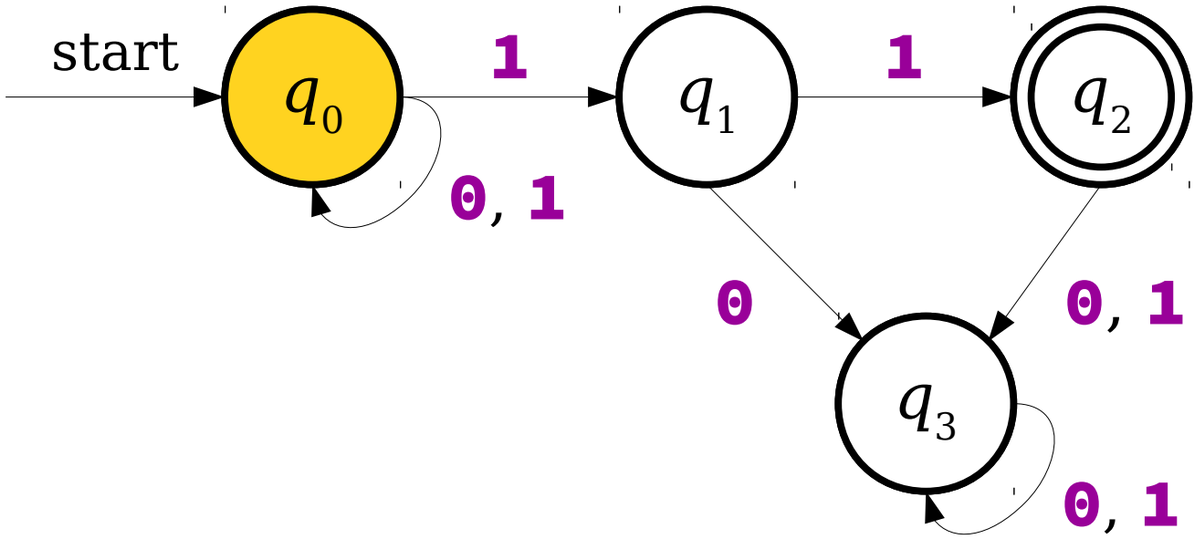
q_0 has two transitions defined on 1!

A Simple NFA



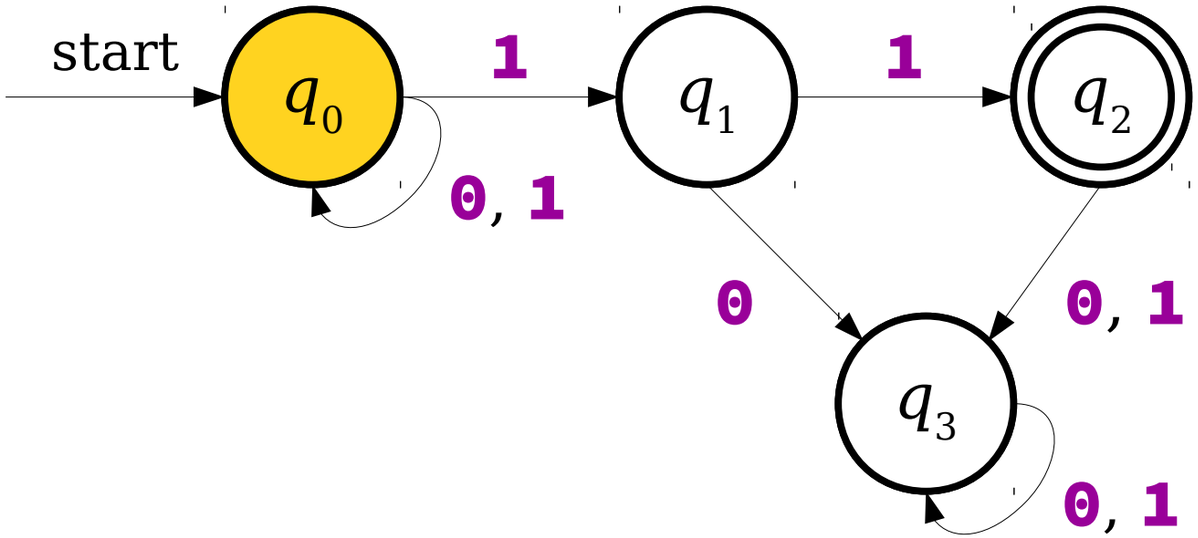
0	1	0	1	1

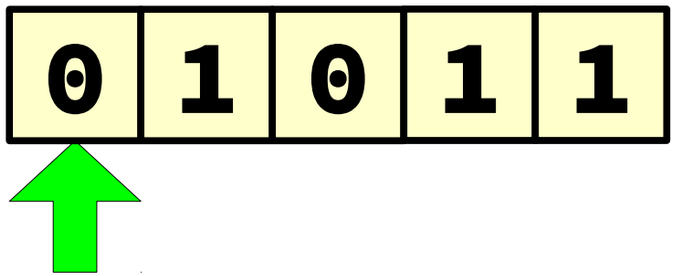
A Simple NFA



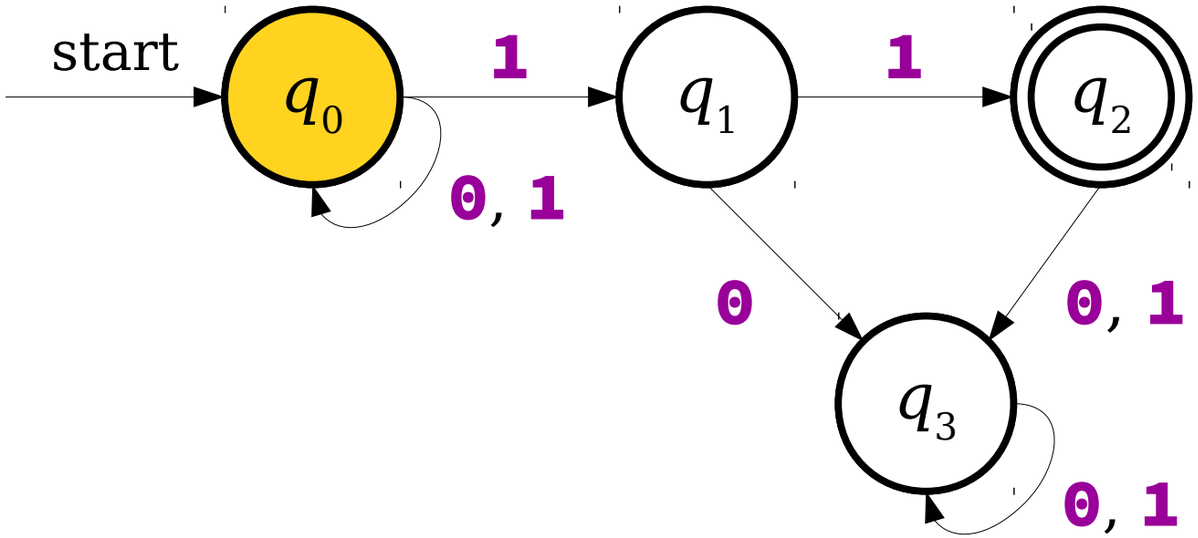
0	1	0	1	1

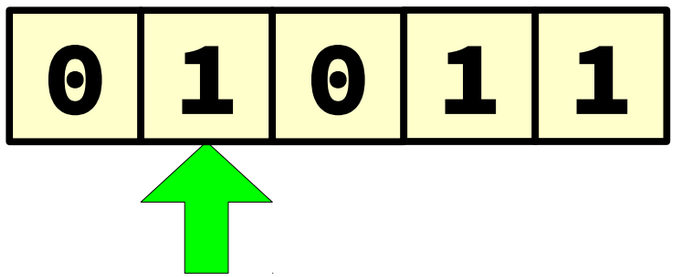
A Simple NFA



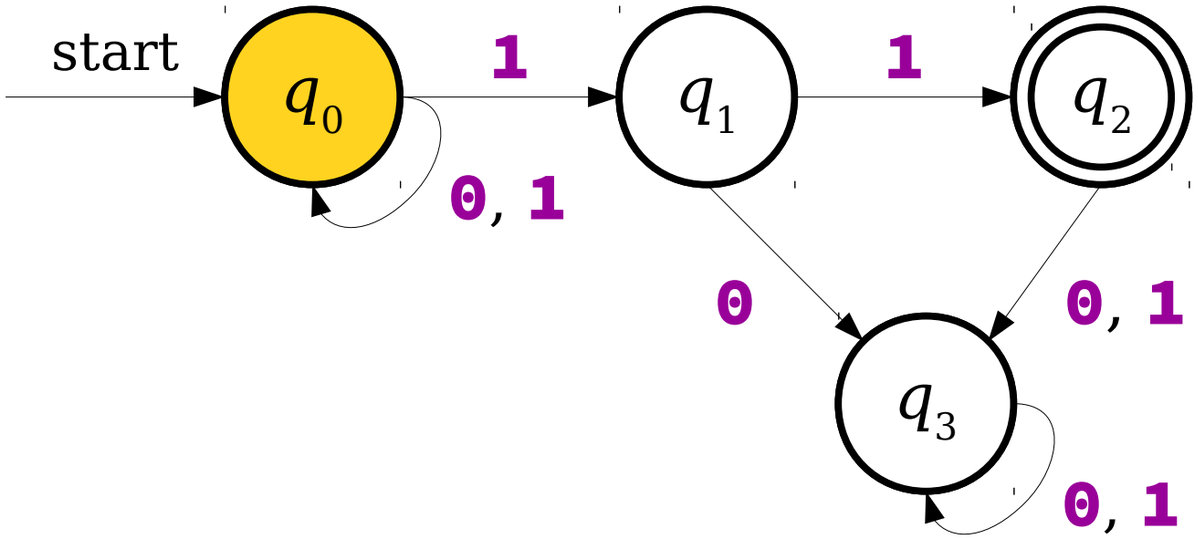


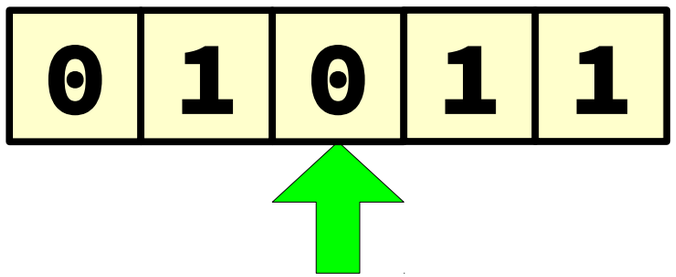
A Simple NFA



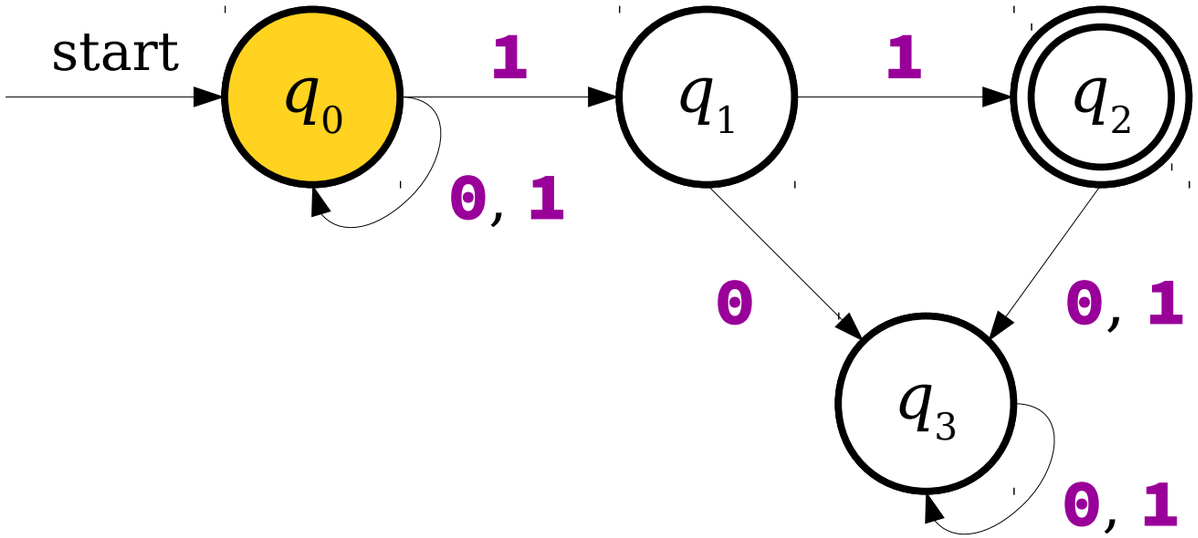


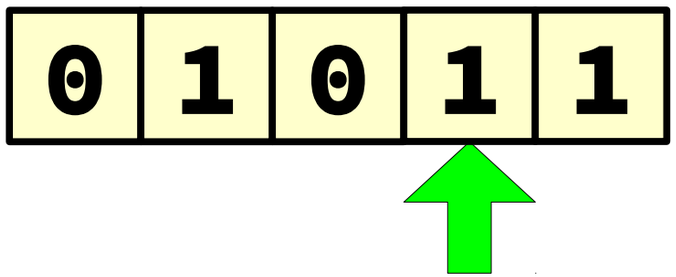
A Simple NFA



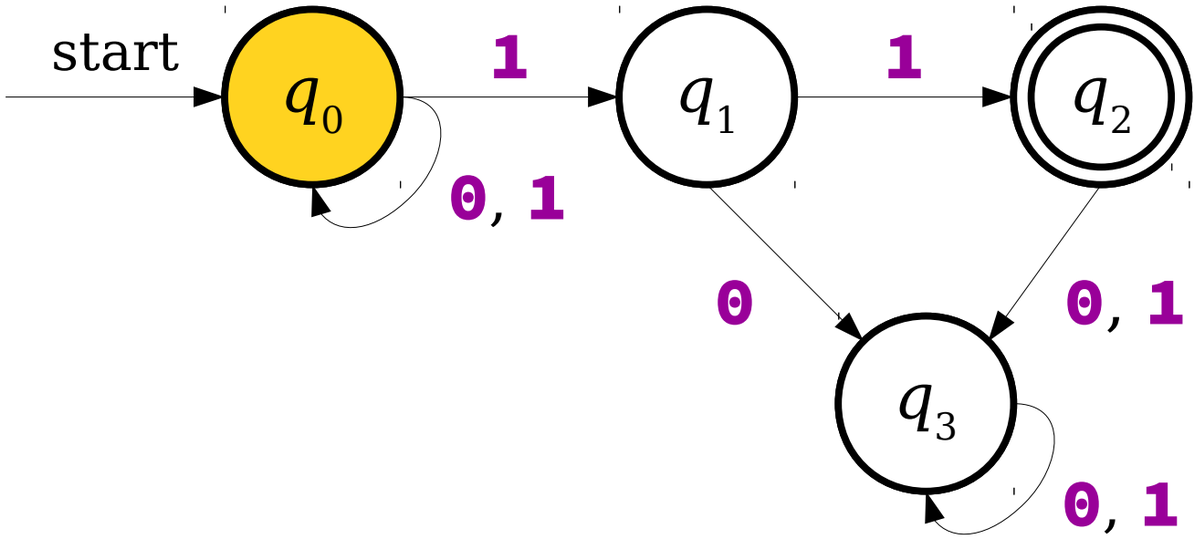


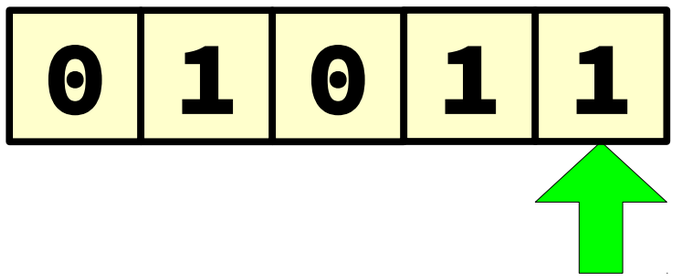
A Simple NFA



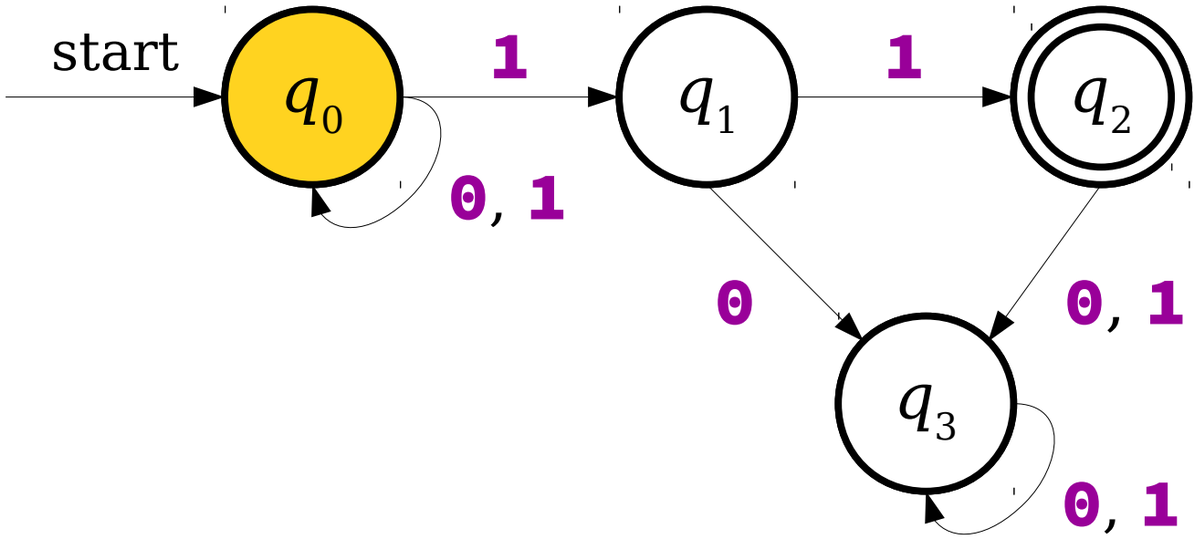


A Simple NFA



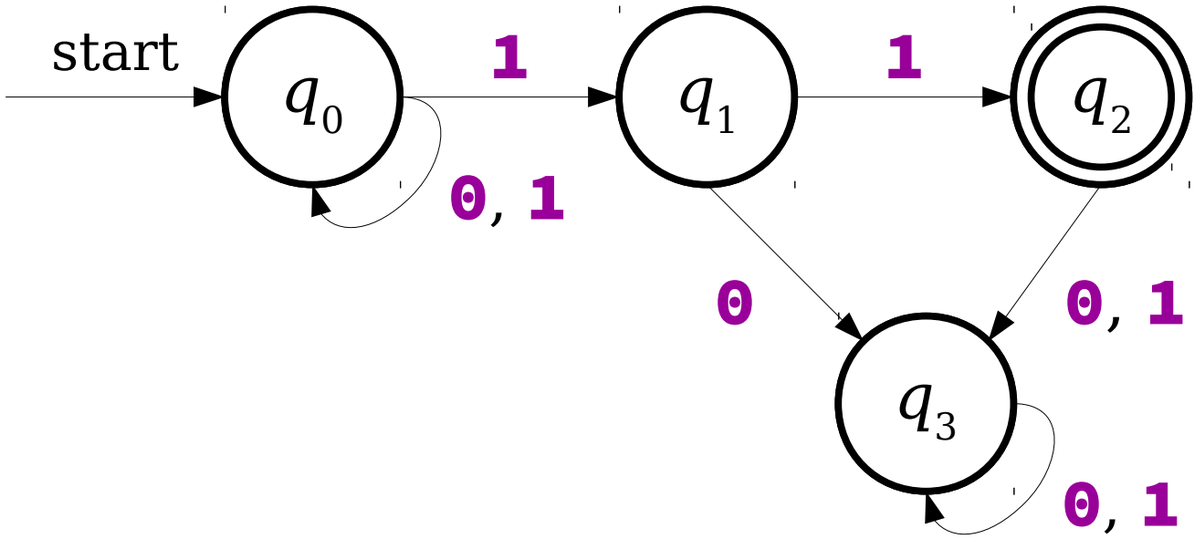


A Simple NFA



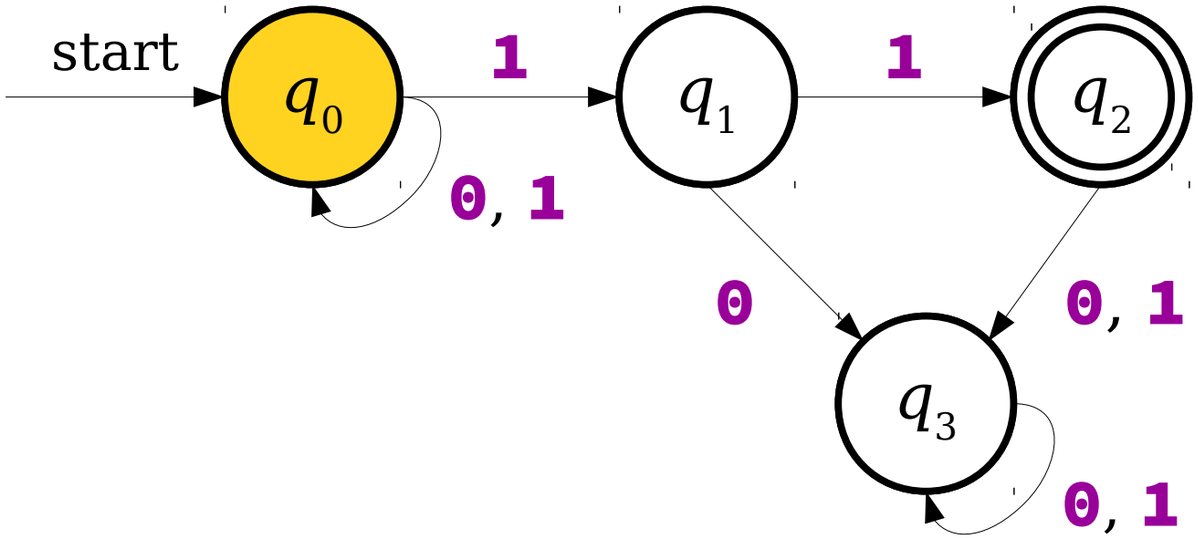
0	1	0	1	1

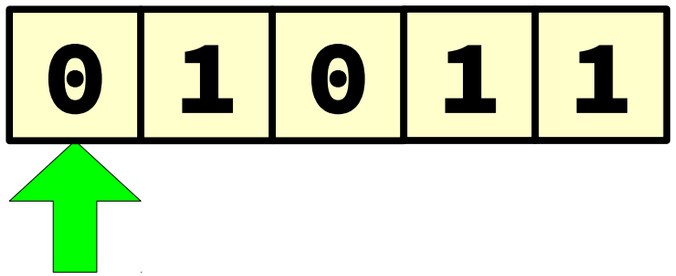
A Simple NFA



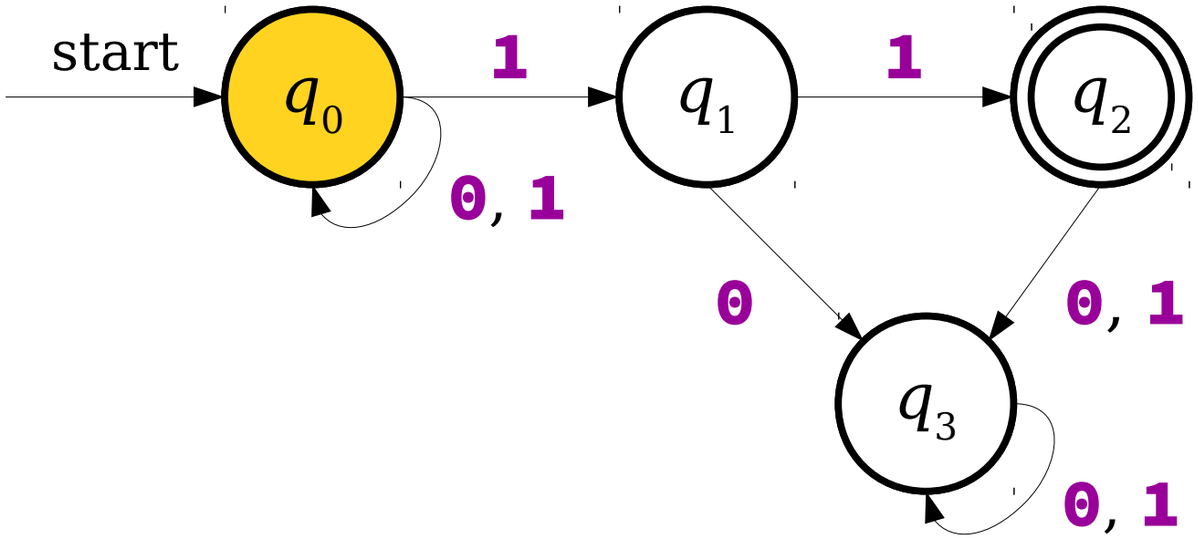
0	1	0	1	1

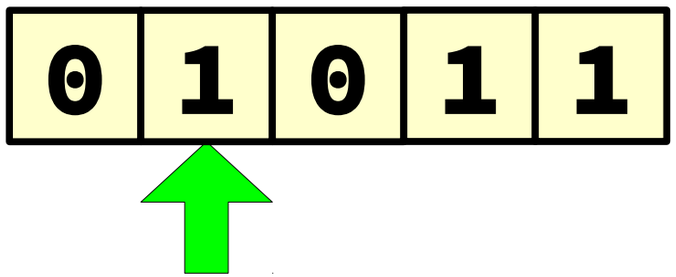
A Simple NFA



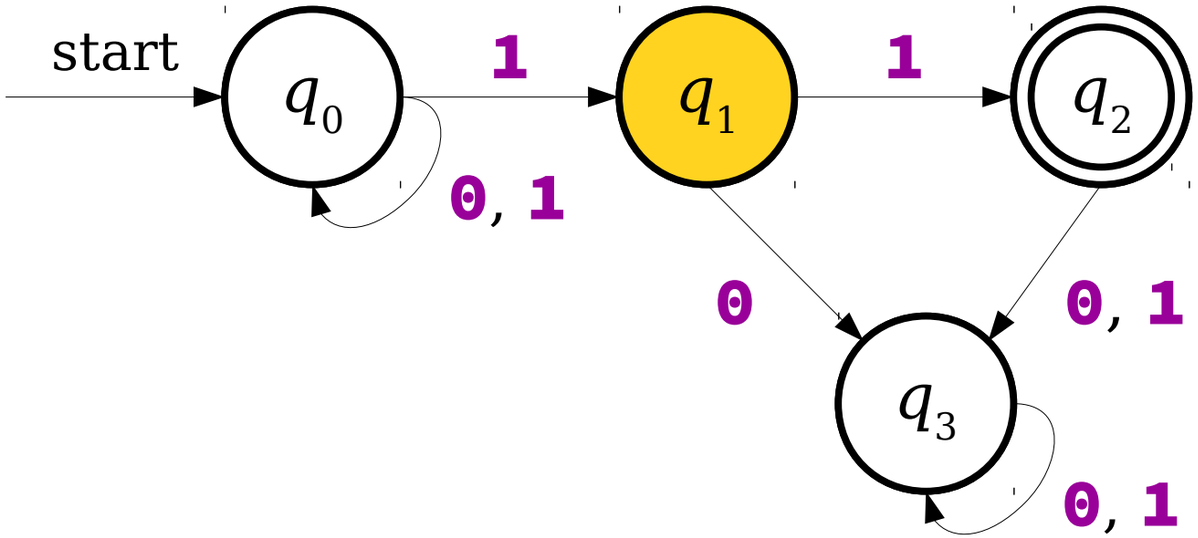


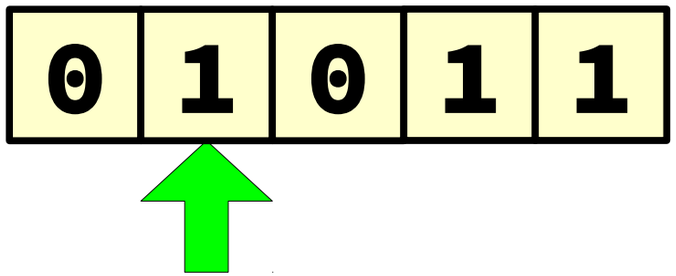
A Simple NFA



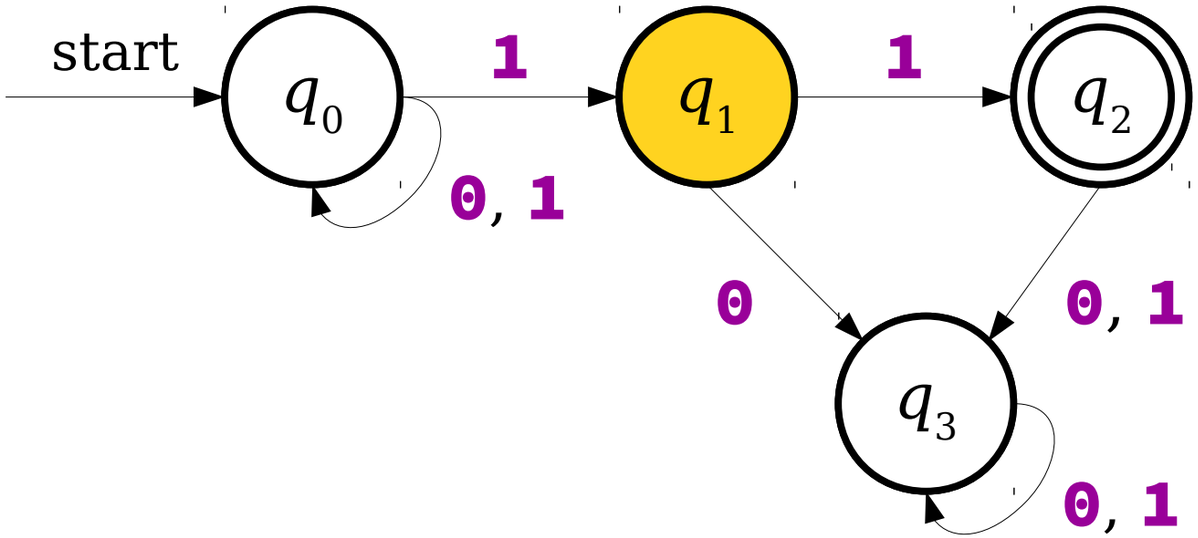


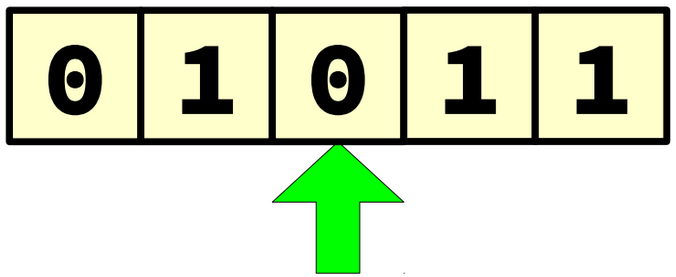
A Simple NFA



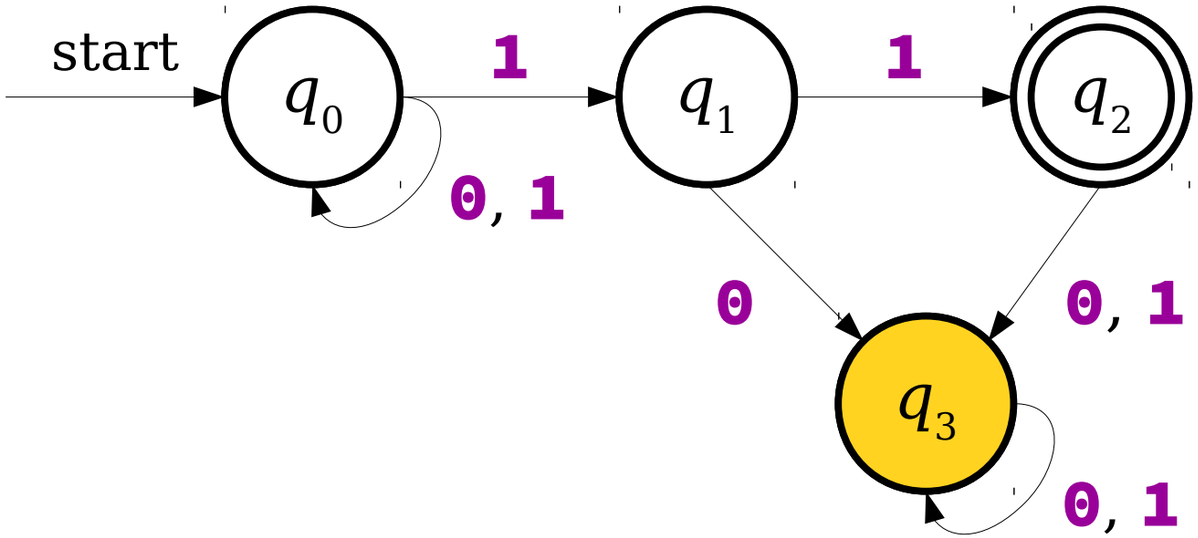


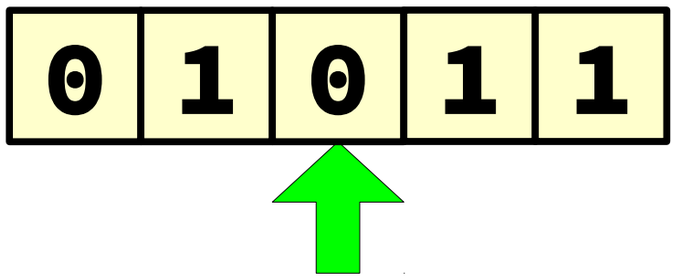
A Simple NFA



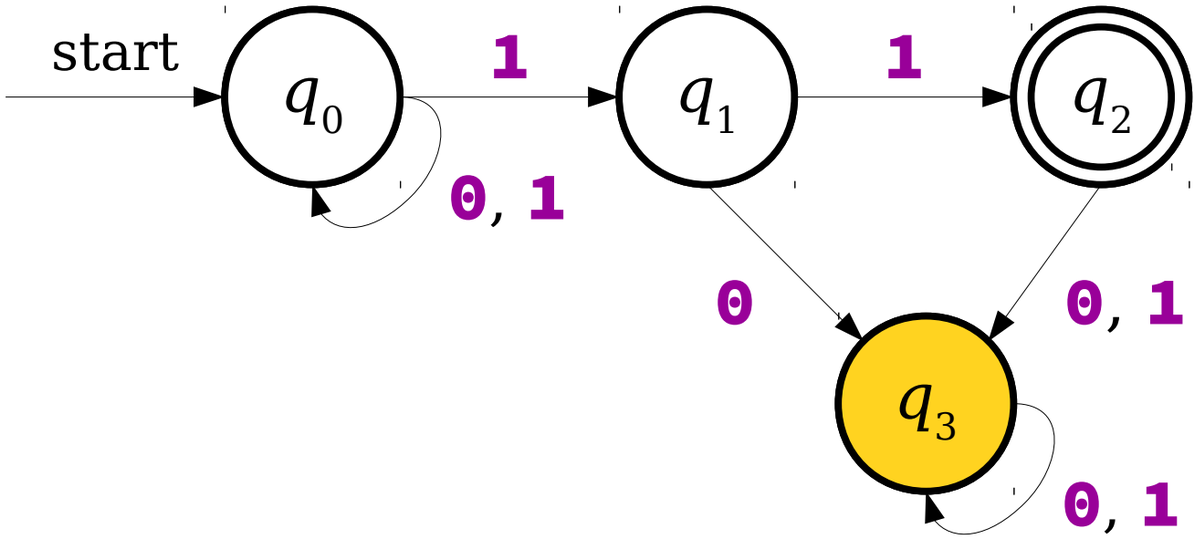


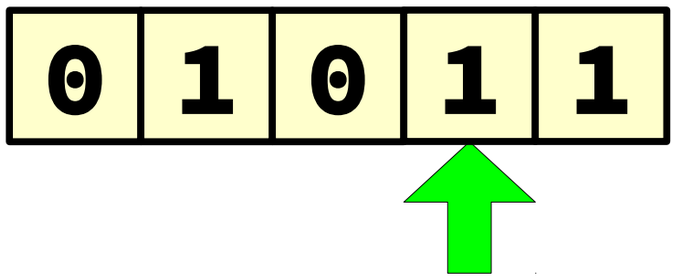
A Simple NFA



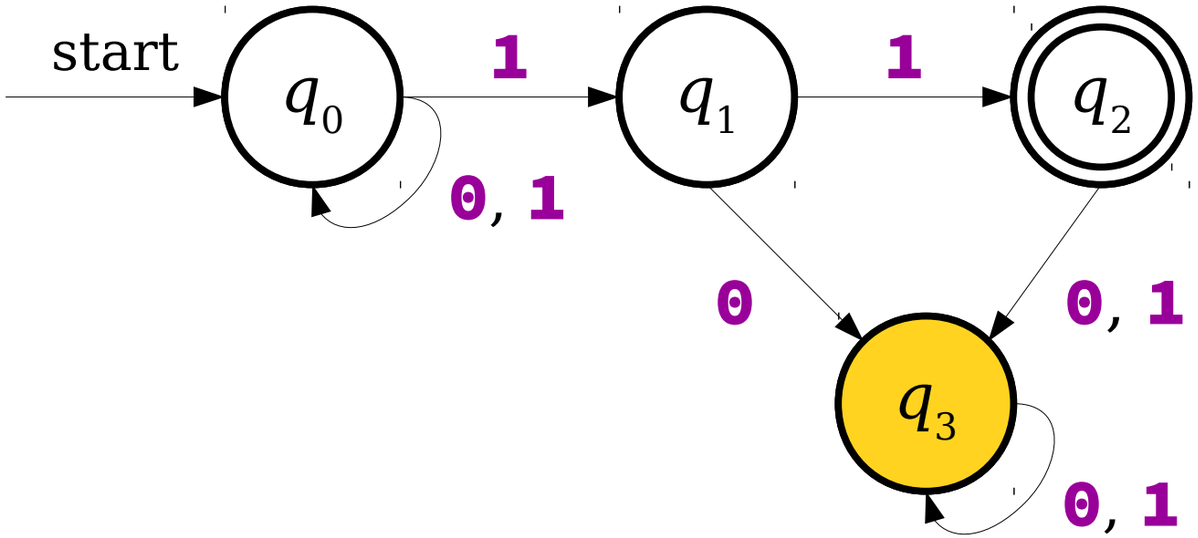


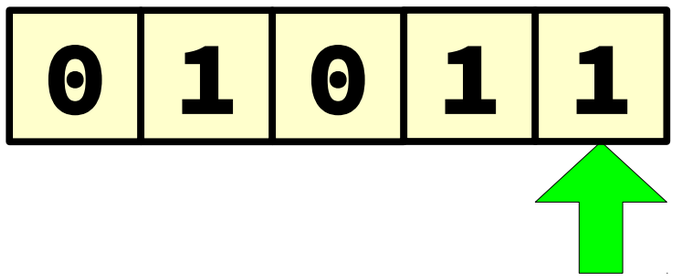
A Simple NFA



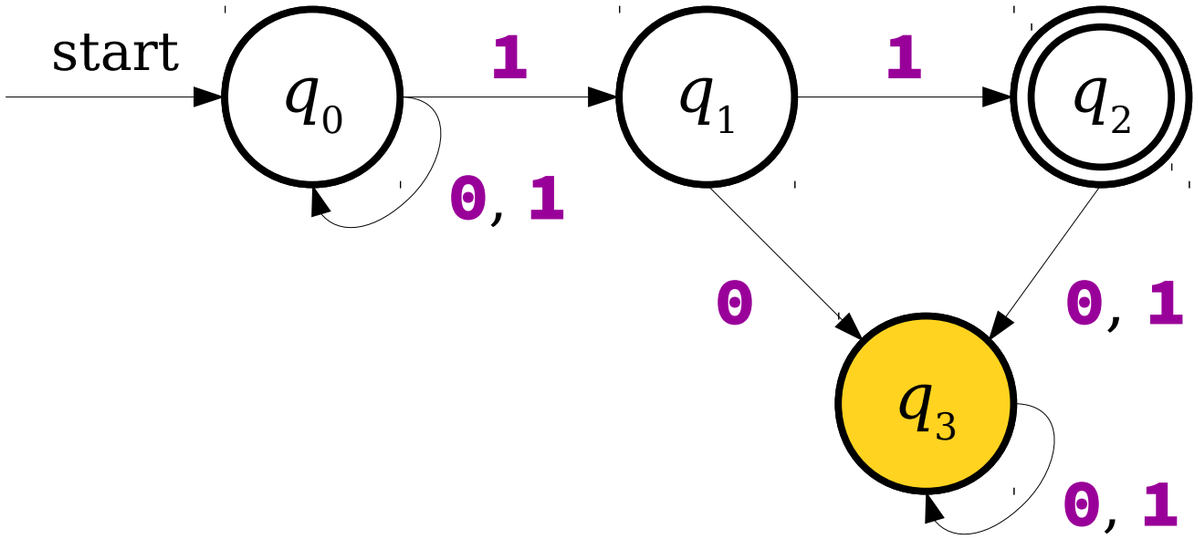


A Simple NFA



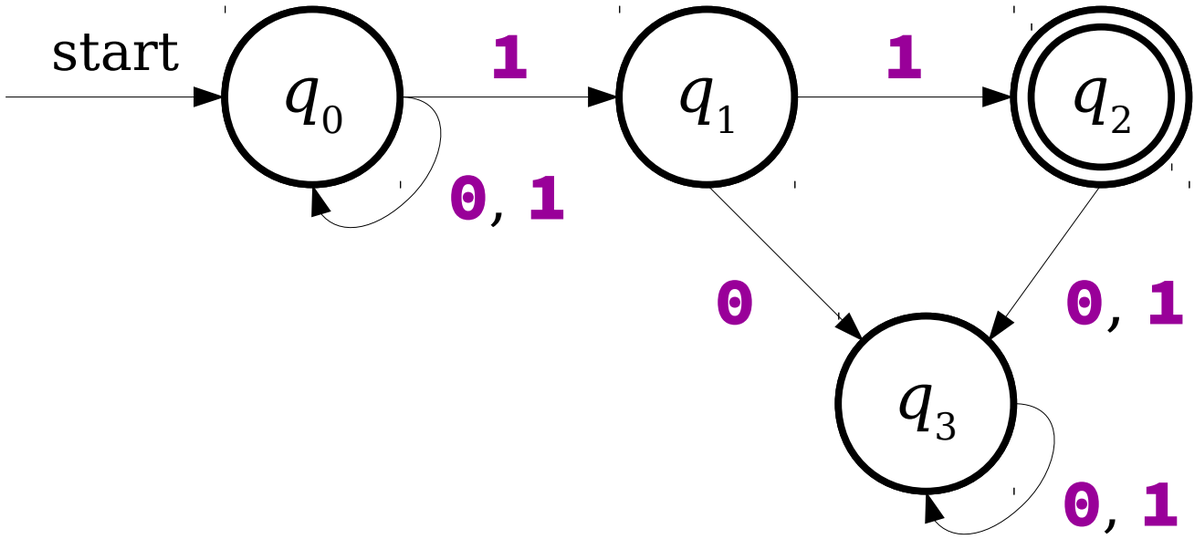


A Simple NFA



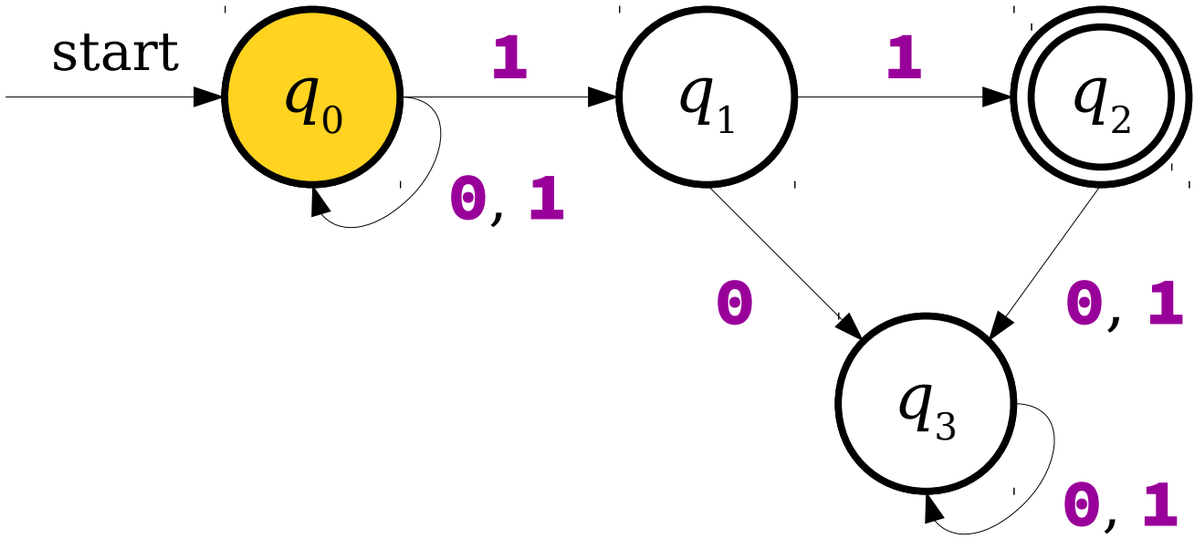
0	1	0	1	1

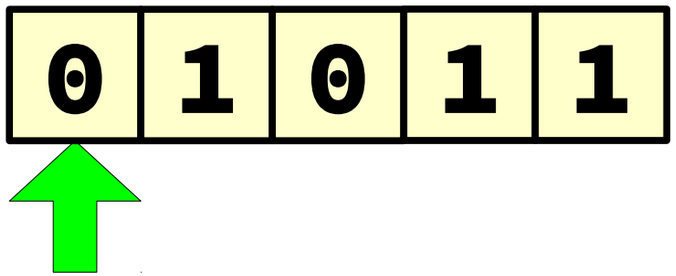
A Simple NFA



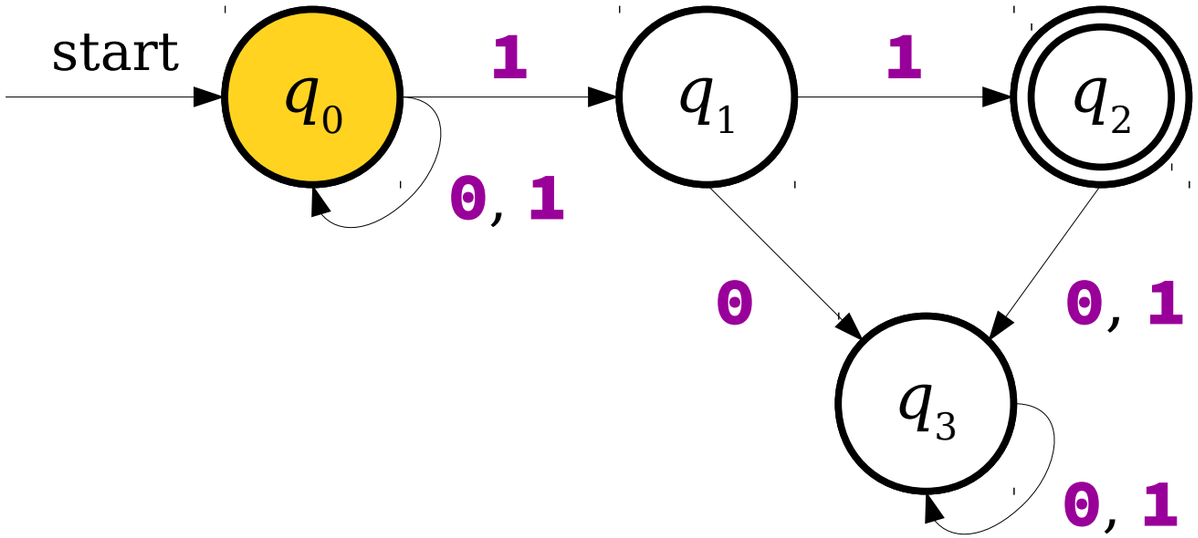
0	1	0	1	1

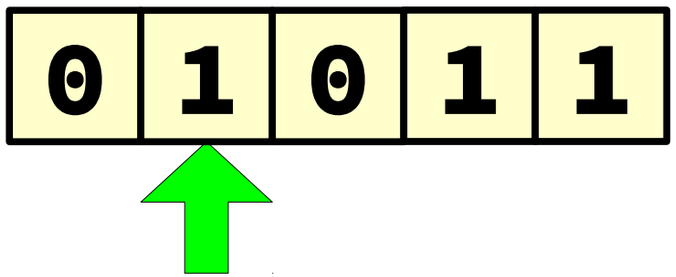
A Simple NFA



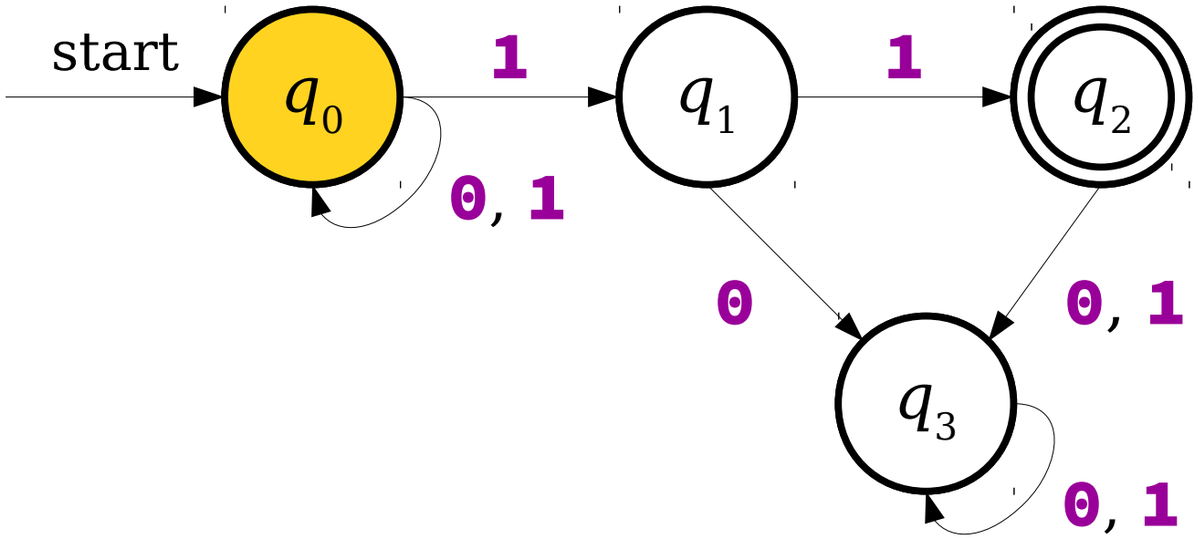


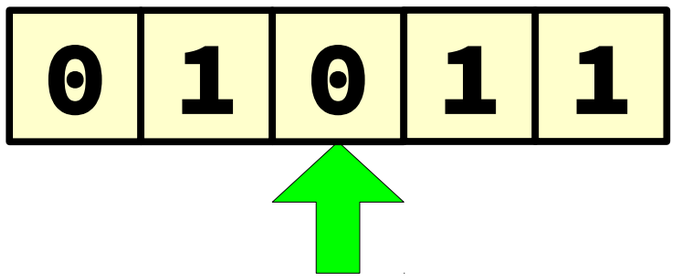
A Simple NFA



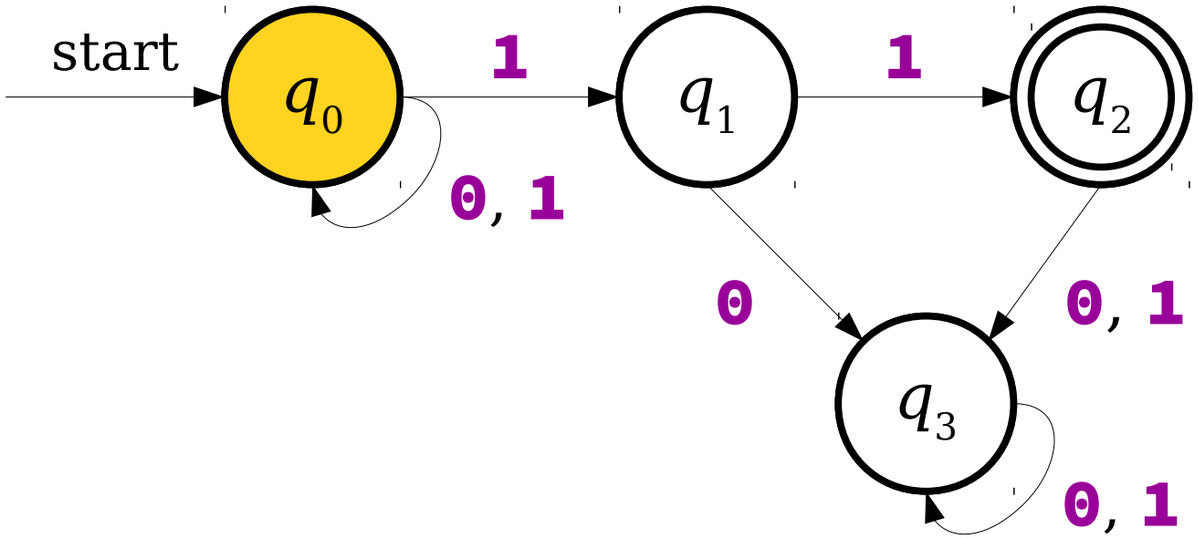


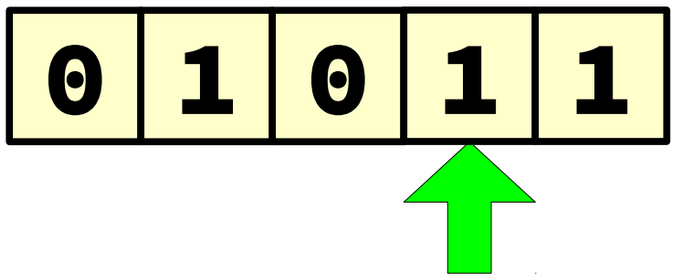
A Simple NFA



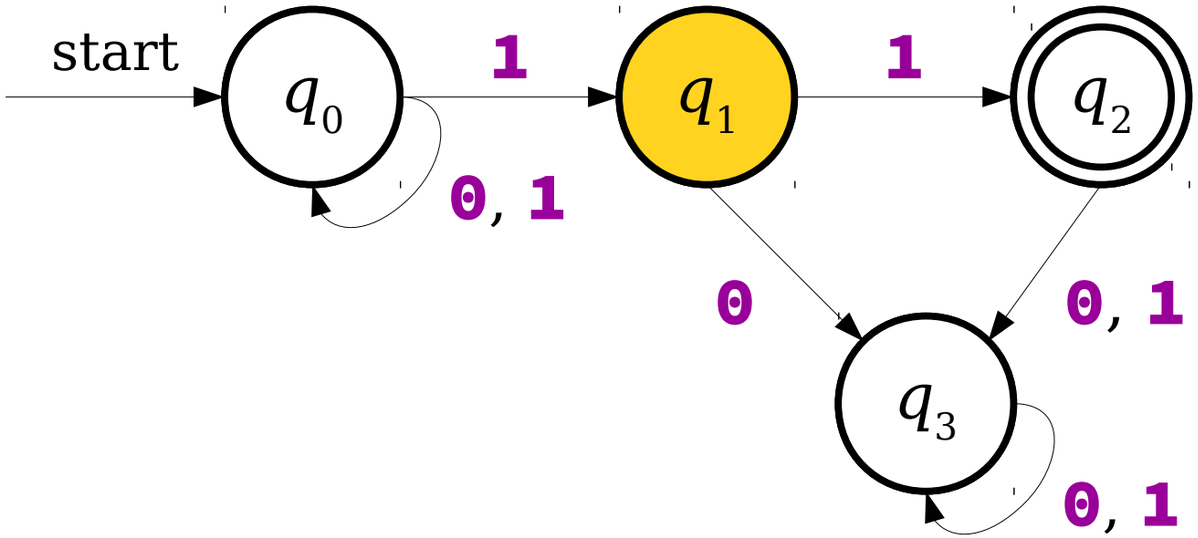


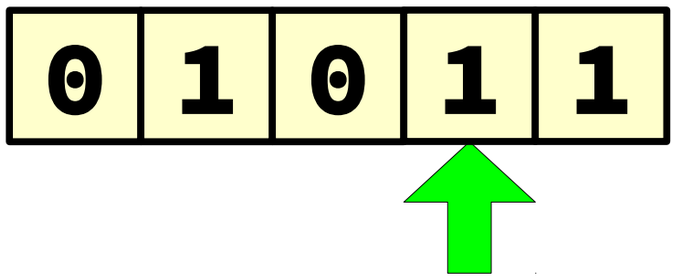
A Simple NFA



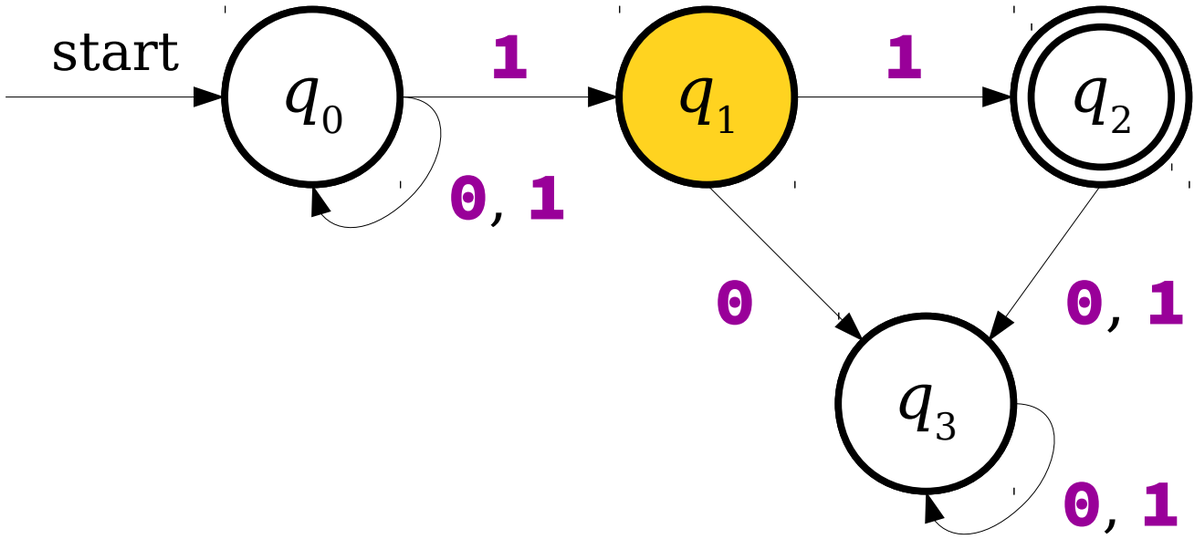


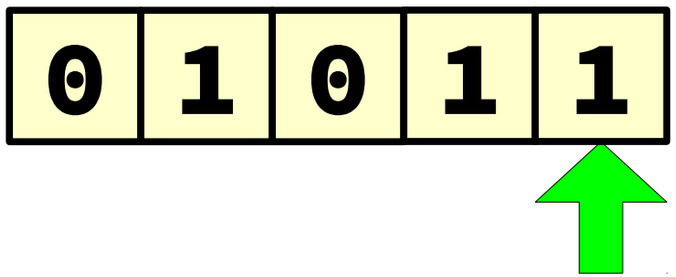
A Simple NFA



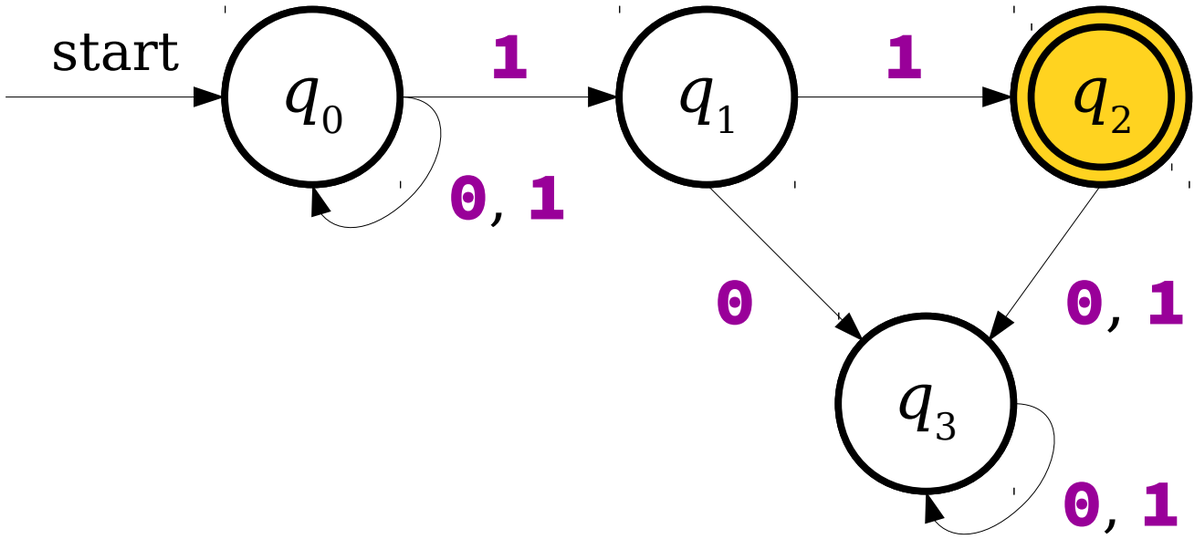


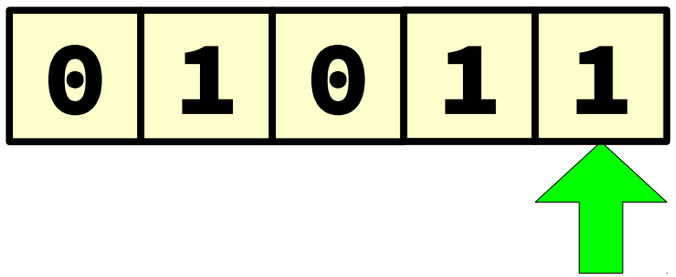
A Simple NFA



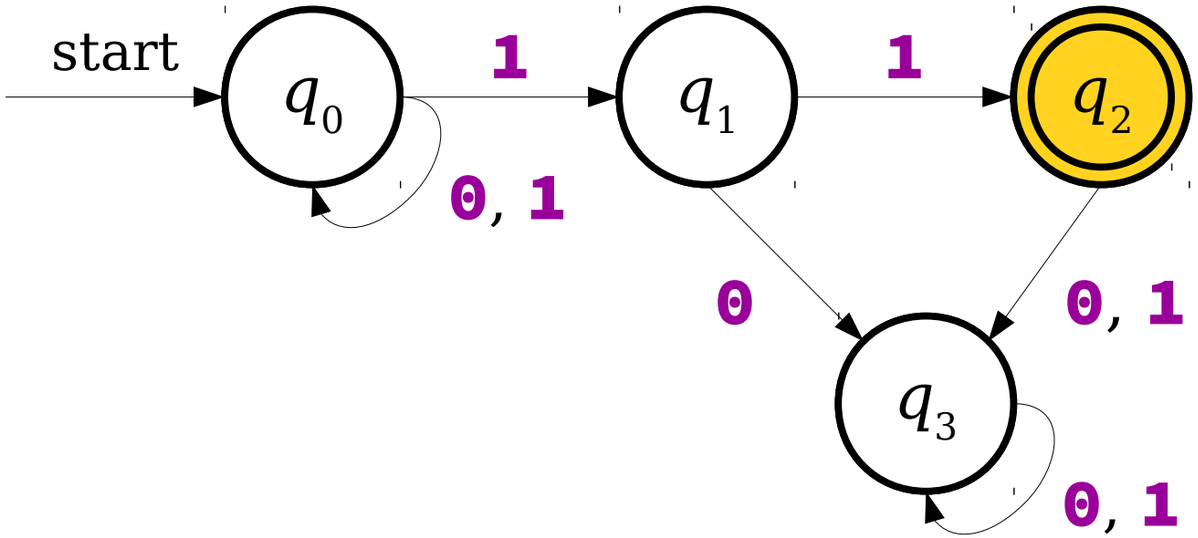


A Simple NFA



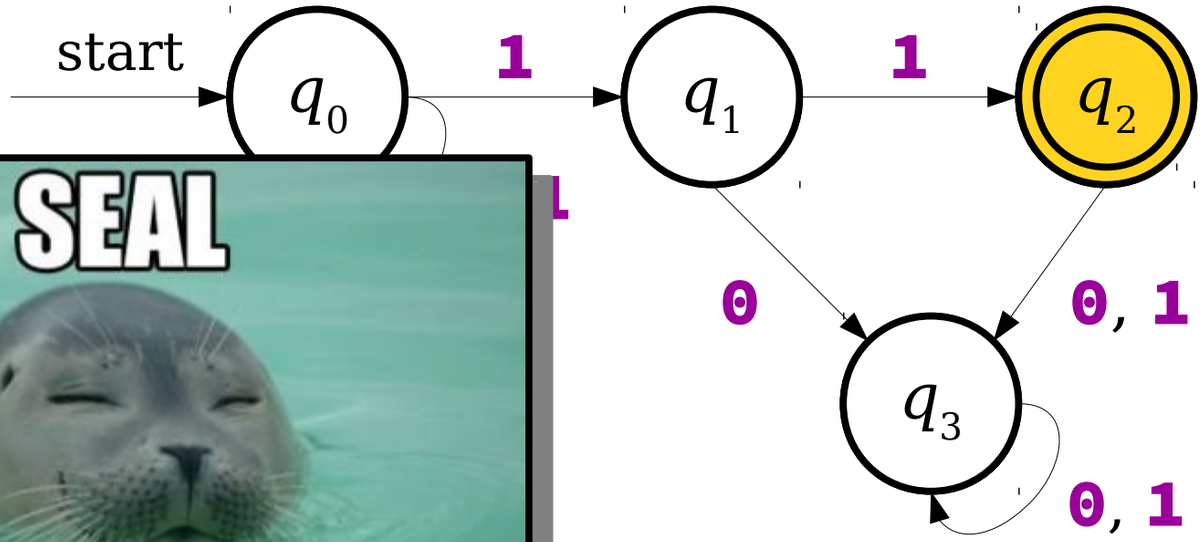


A Simple NFA



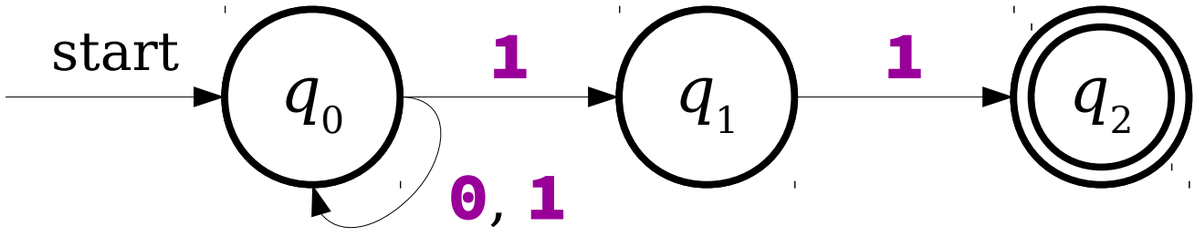
0	1	0	1	1

A Simple NFA

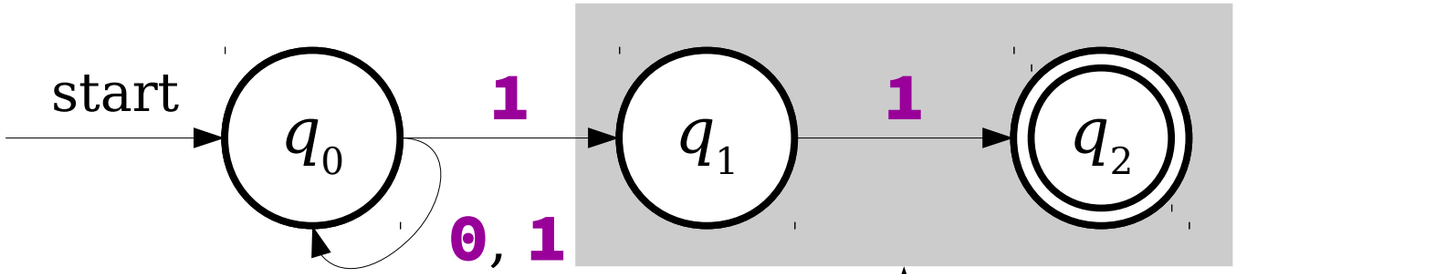


0 1 0 1 1

A More Complex NFA

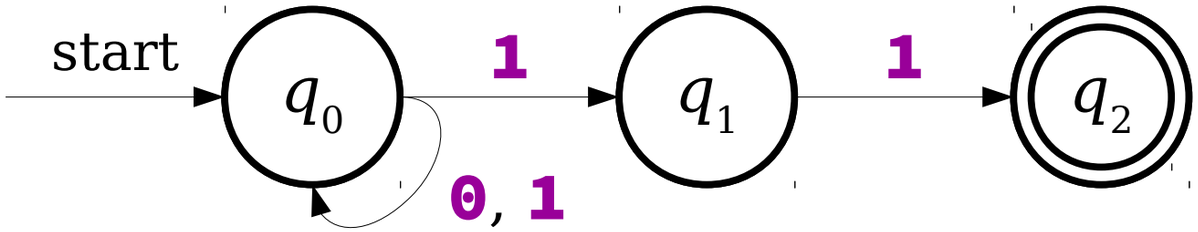


A More Complex NFA



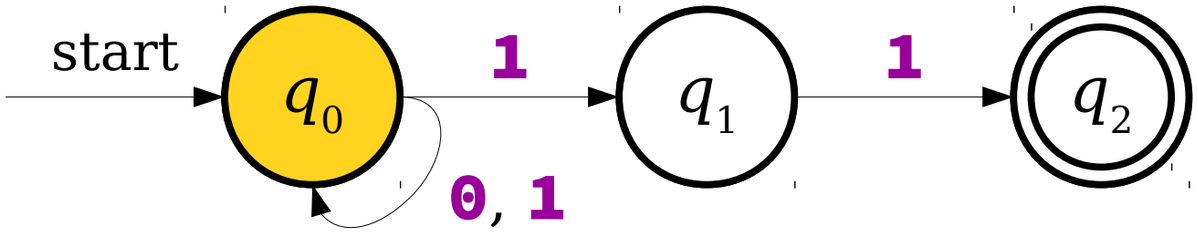
If a NFA needs to make a transition when no transition exists, the automaton **dies** and that particular path does not accept.

A More Complex NFA



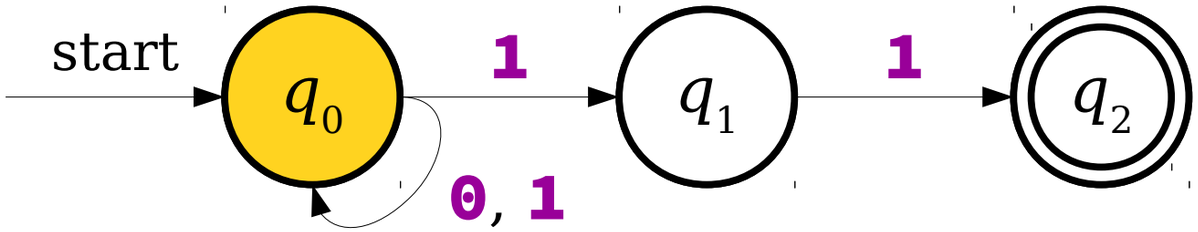
0	1	0	1	1

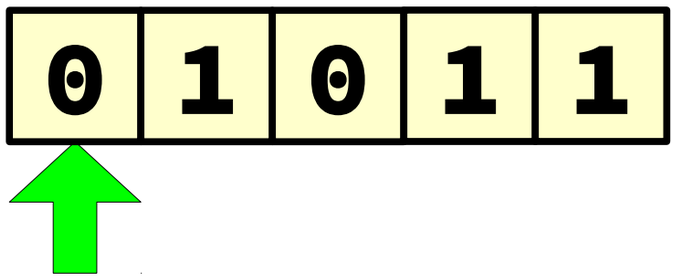
A More Complex NFA



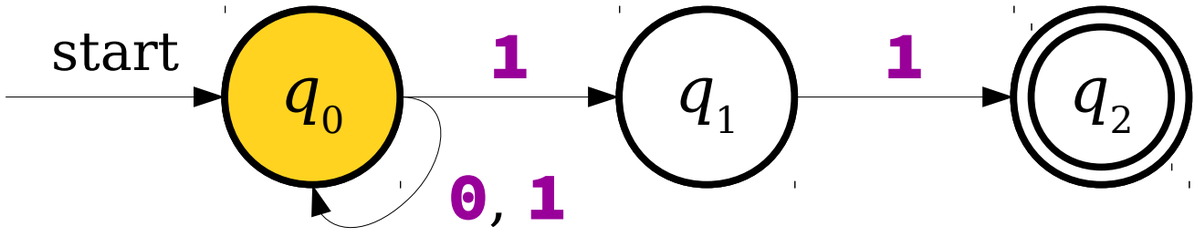
0	1	0	1	1

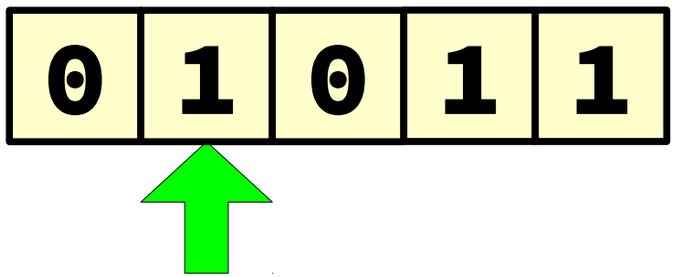
A More Complex NFA



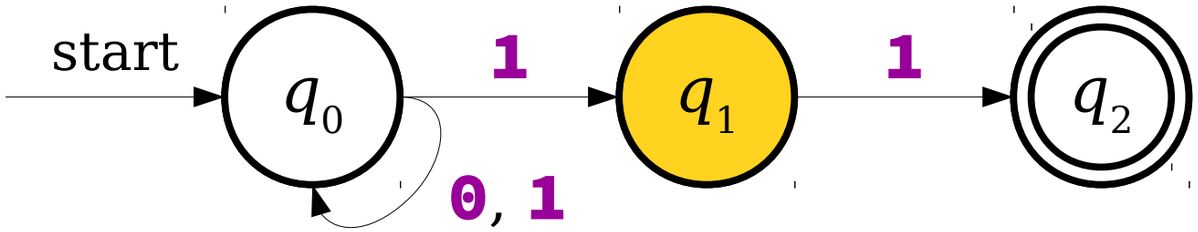


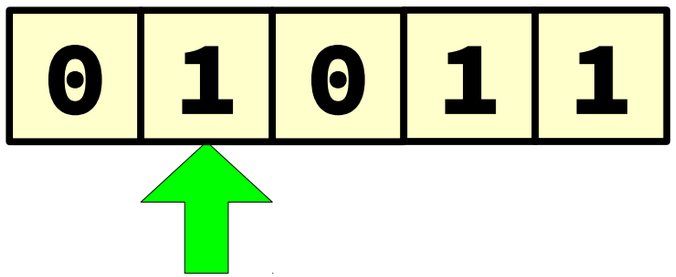
A More Complex NFA



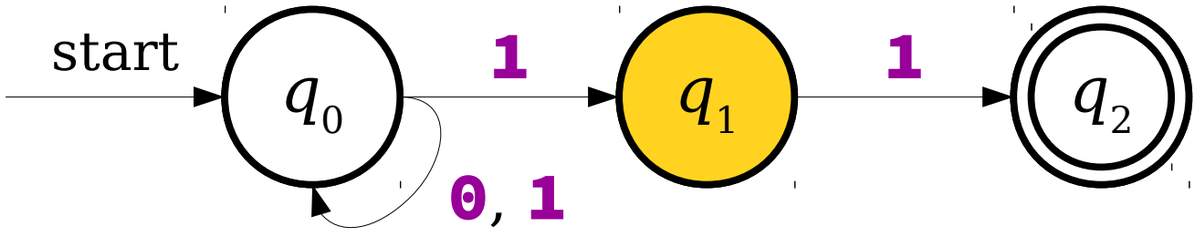


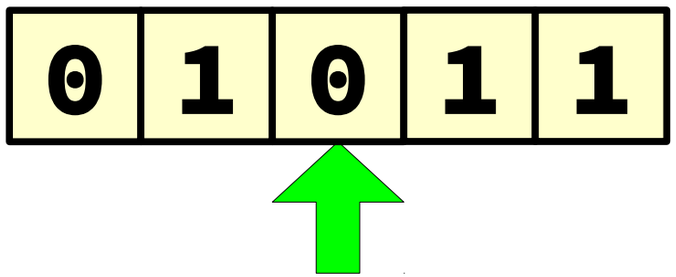
A More Complex NFA



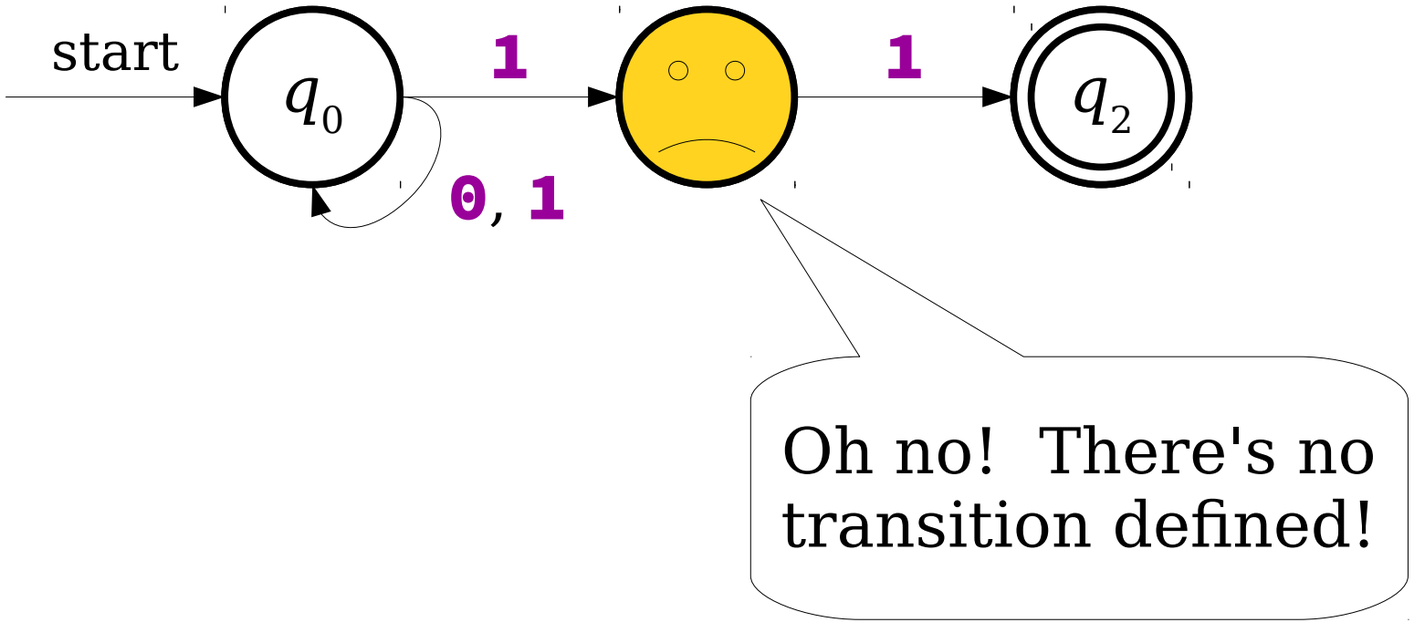


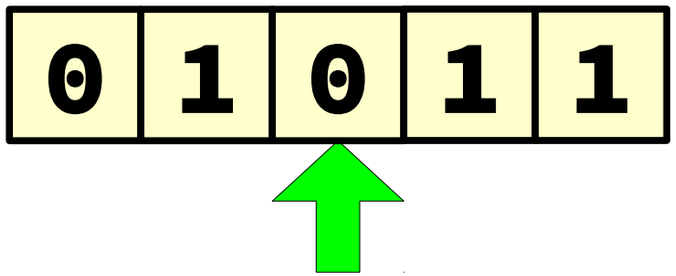
A More Complex NFA



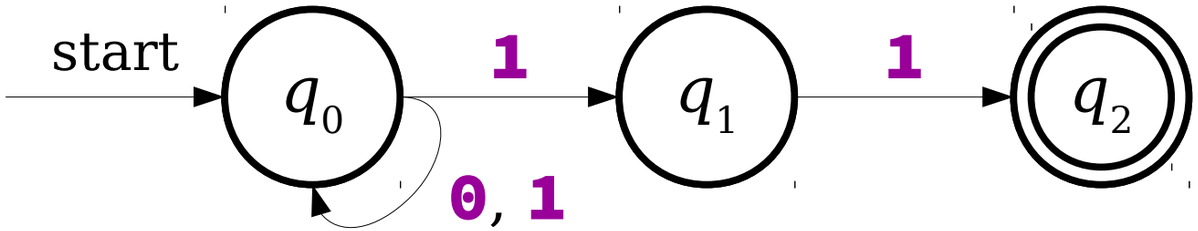


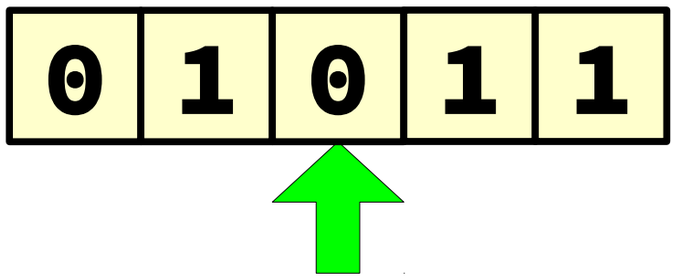
A More Complex NFA



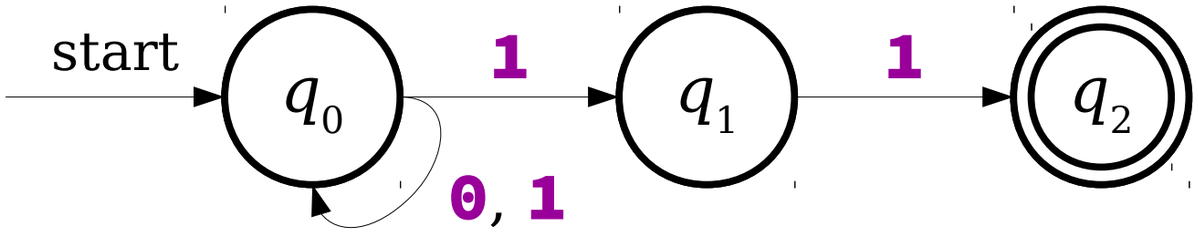


A More Complex NFA



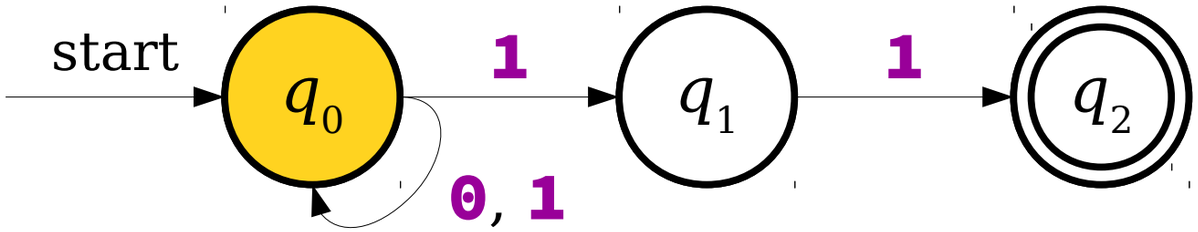


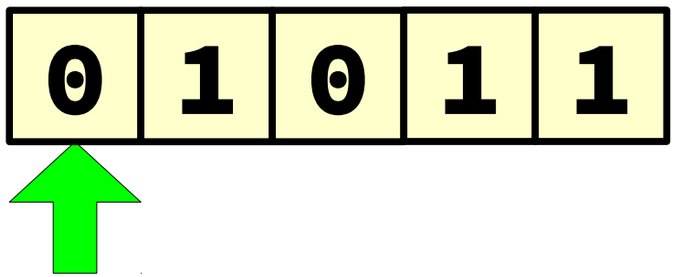
A More Complex NFA



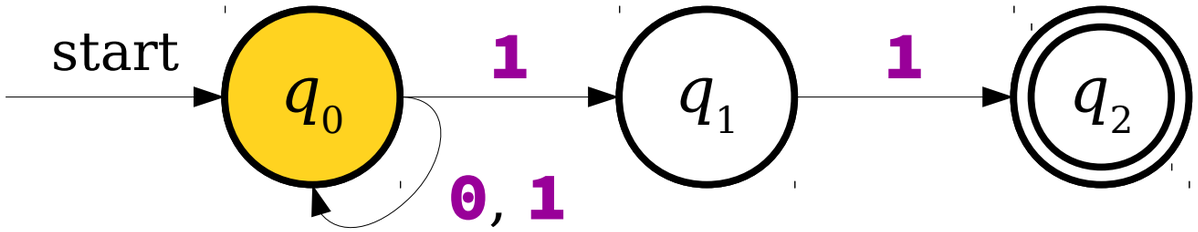
0	1	0	1	1

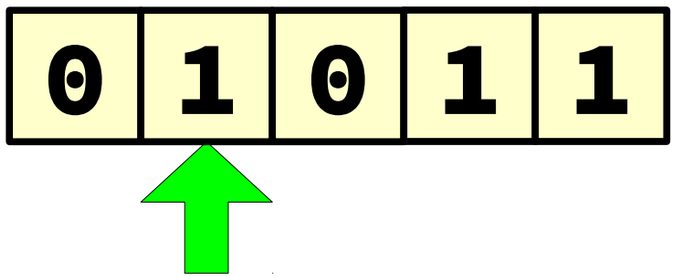
A More Complex NFA



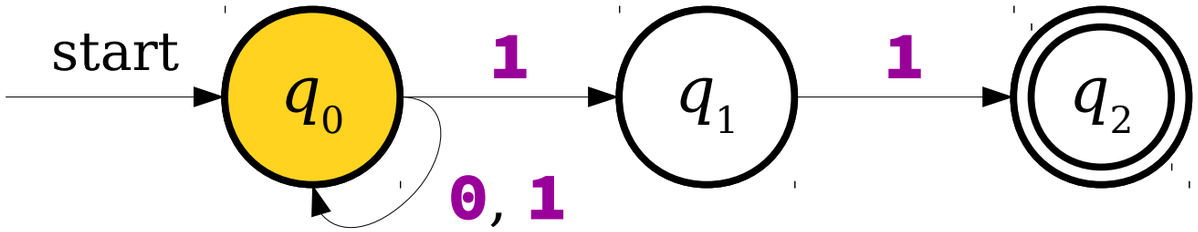


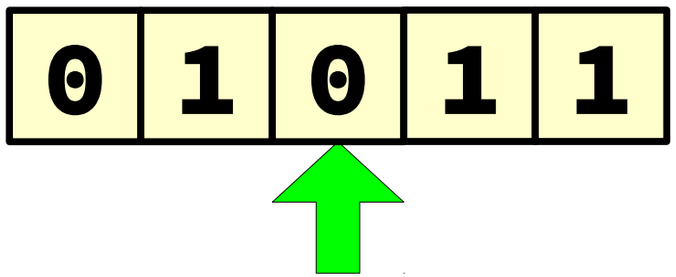
A More Complex NFA



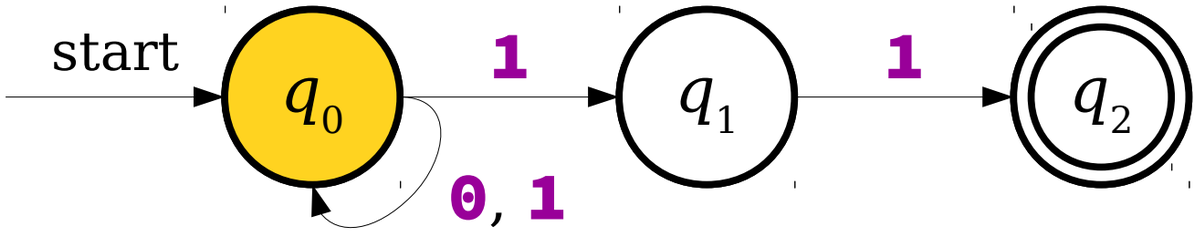


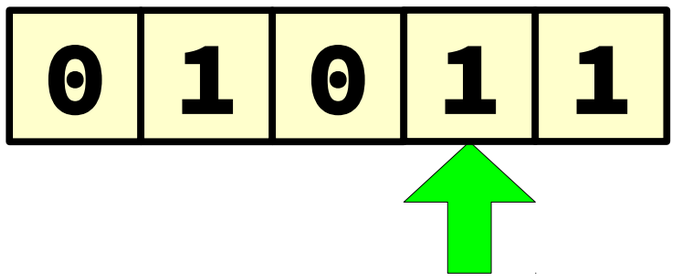
A More Complex NFA



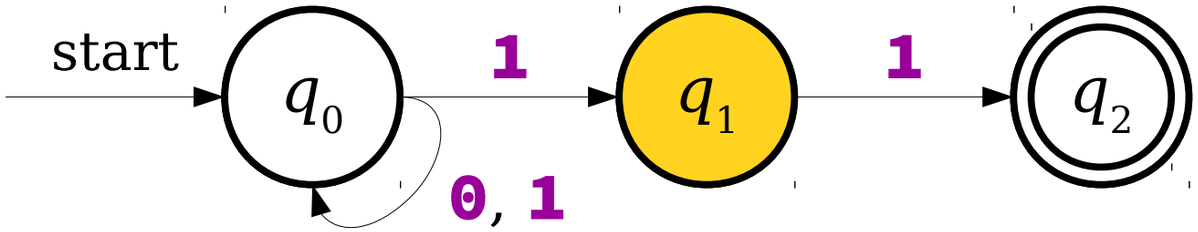


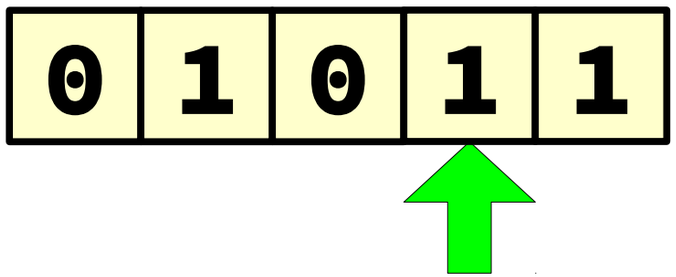
A More Complex NFA



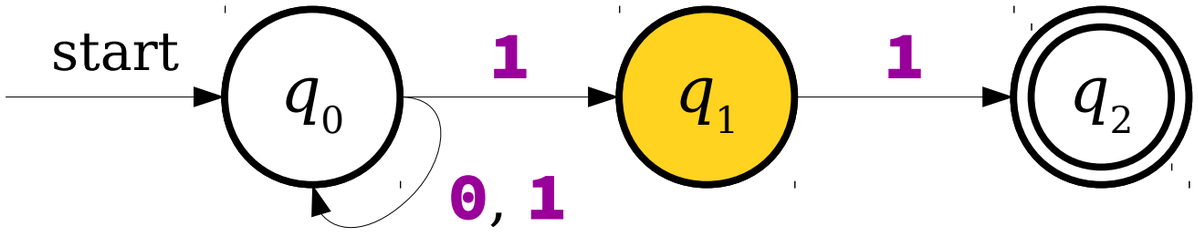


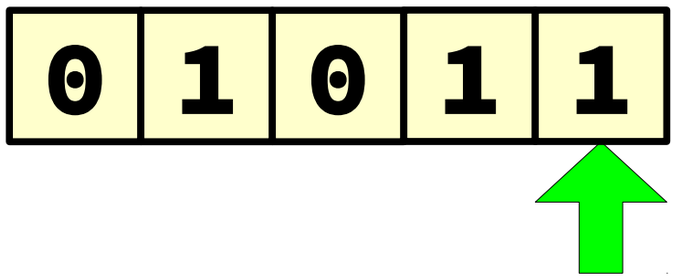
A More Complex NFA



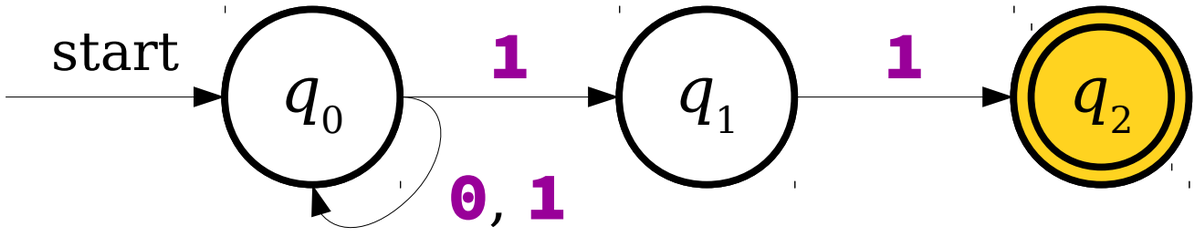


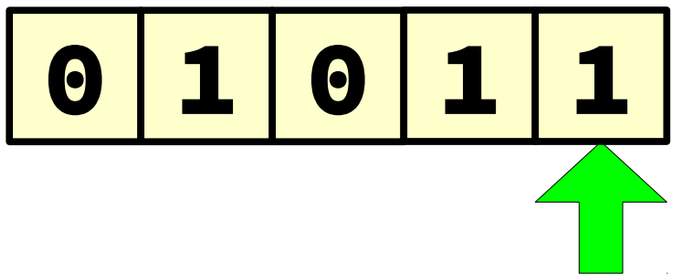
A More Complex NFA



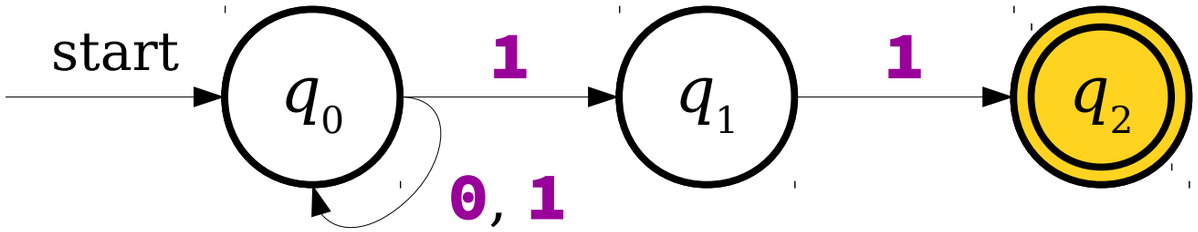


A More Complex NFA



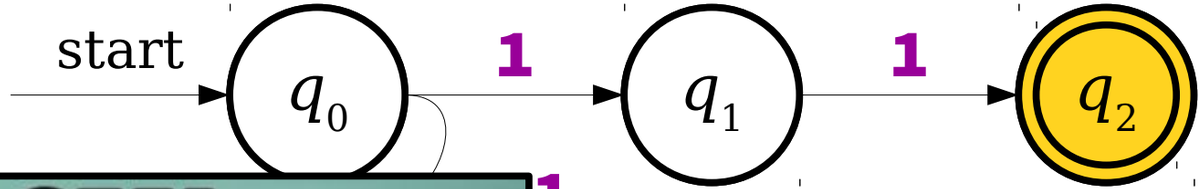


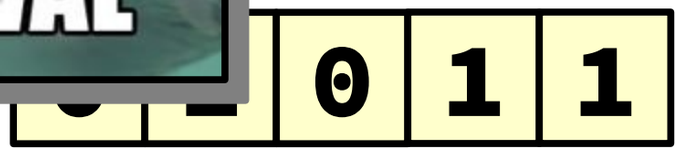
A More Complex NFA



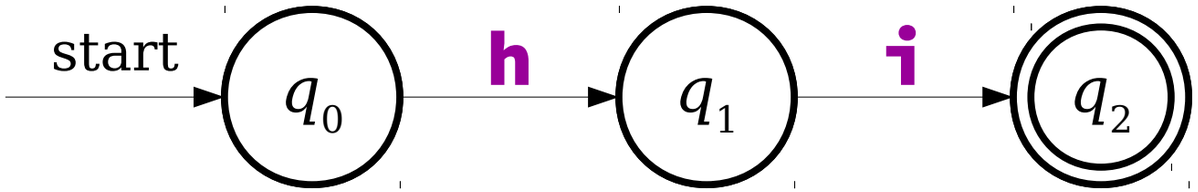
0	1	0	1	1

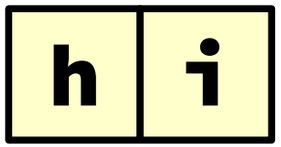
A More Complex NFA



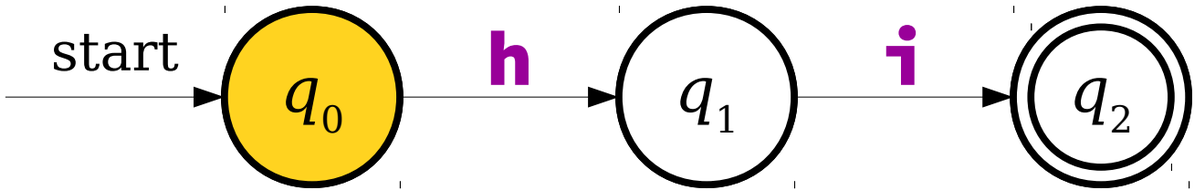


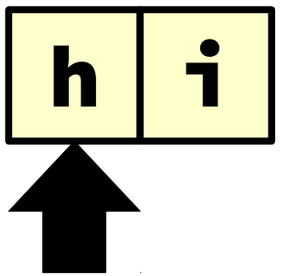
Hello, NFA!



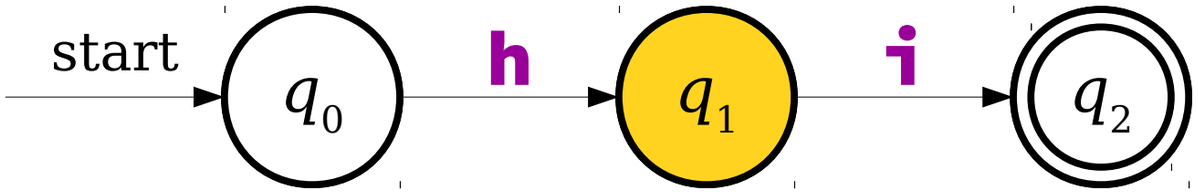


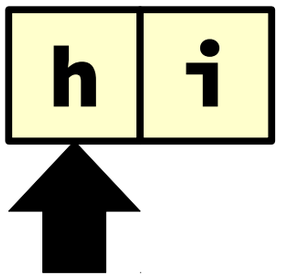
Hello, NFA!



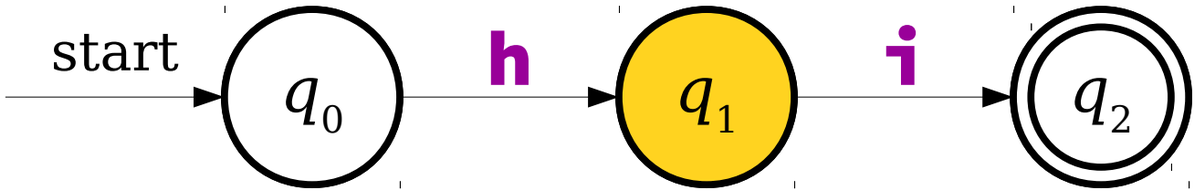


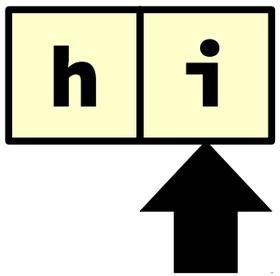
Hello, NFA!



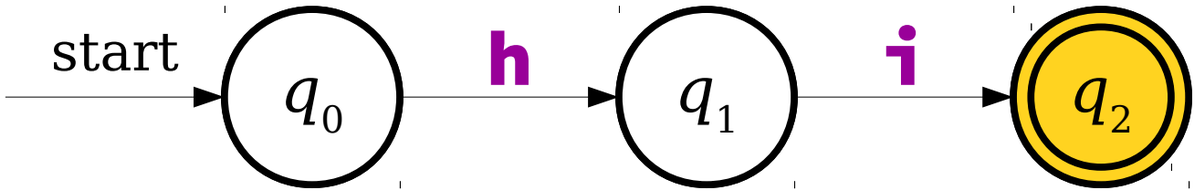


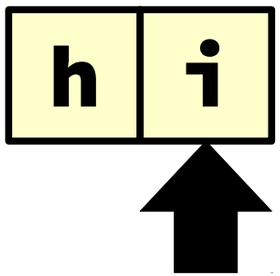
Hello, NFA!



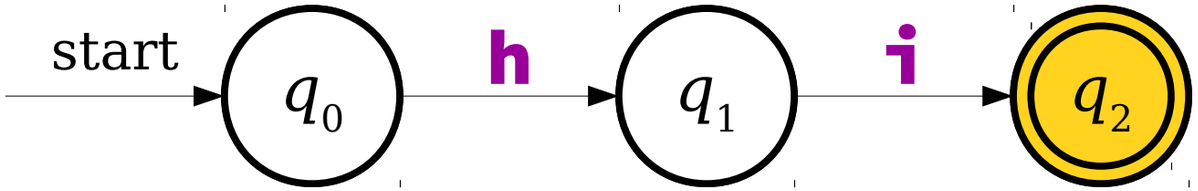


Hello, NFA!



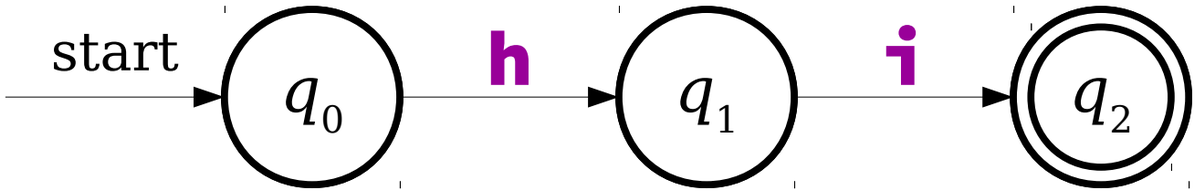


Hello, NFA!



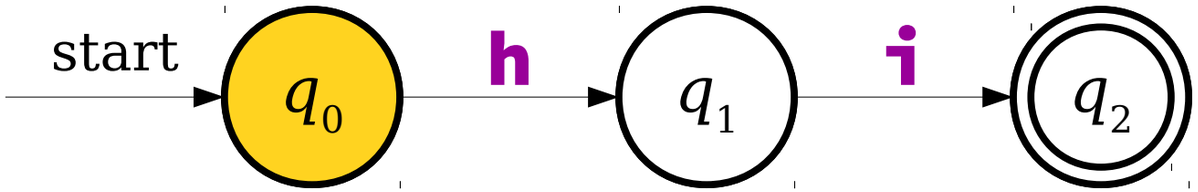
h	i

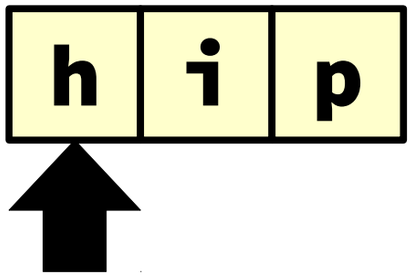
Tragedy in Paradise



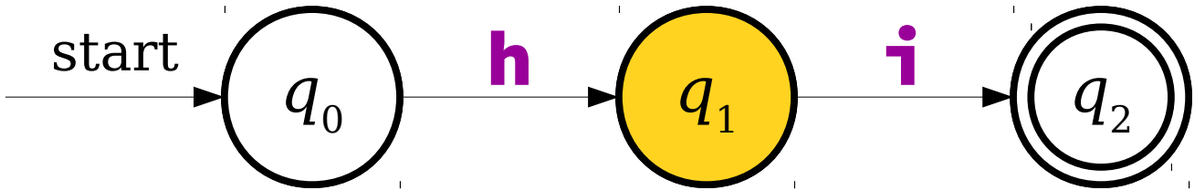
h	i	p

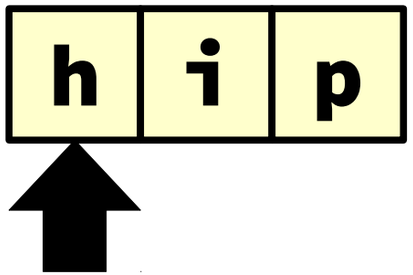
Tragedy in Paradise



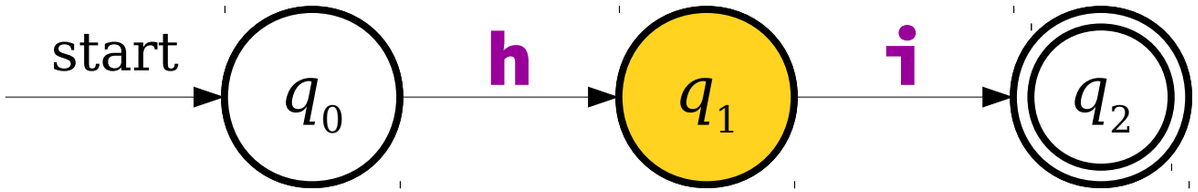


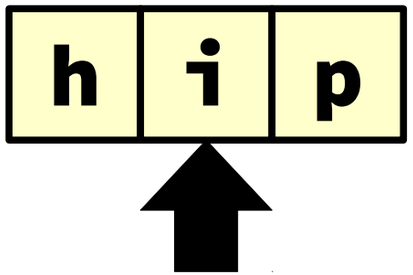
Tragedy in Paradise



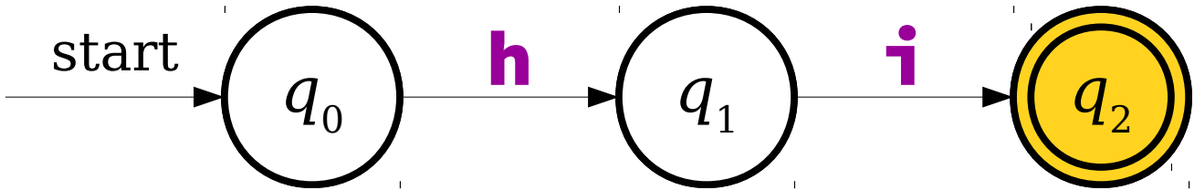


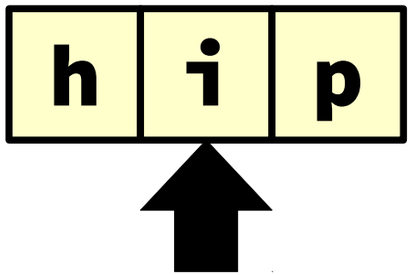
Tragedy in Paradise



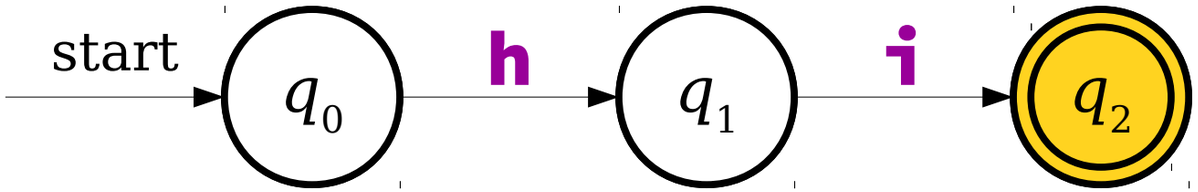


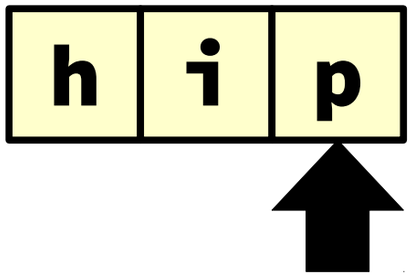
Tragedy in Paradise



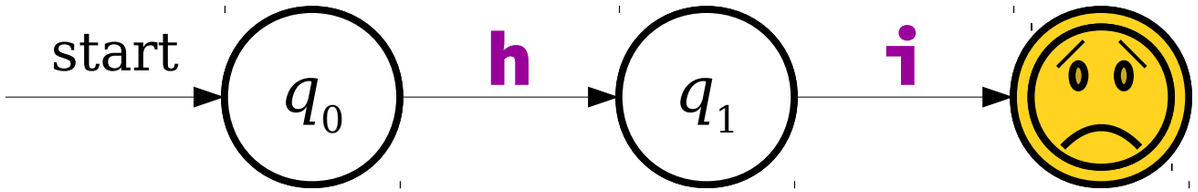


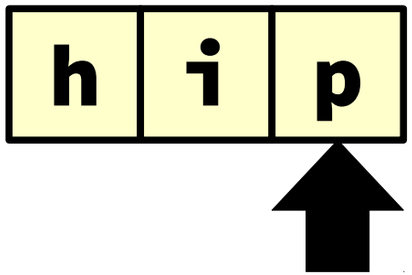
Tragedy in Paradise



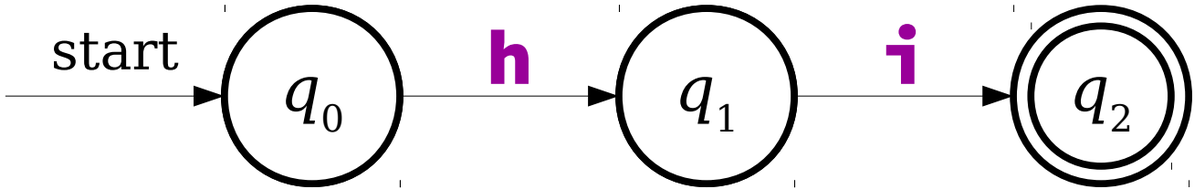


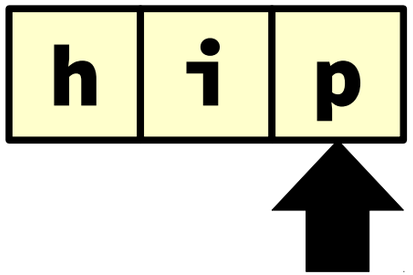
Tragedy in Paradise





Tragedy in Paradise



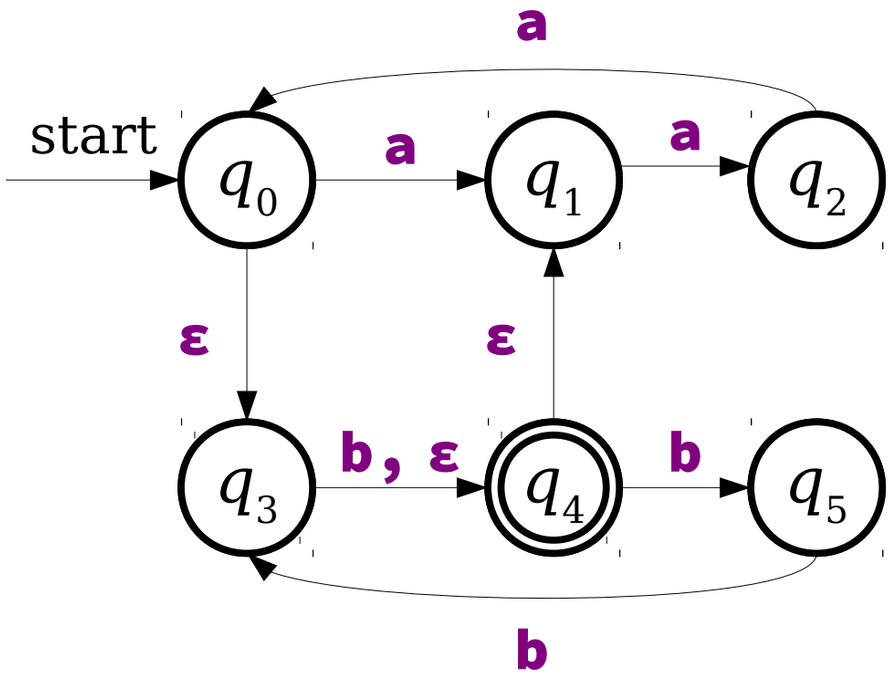


ε-Transitions

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

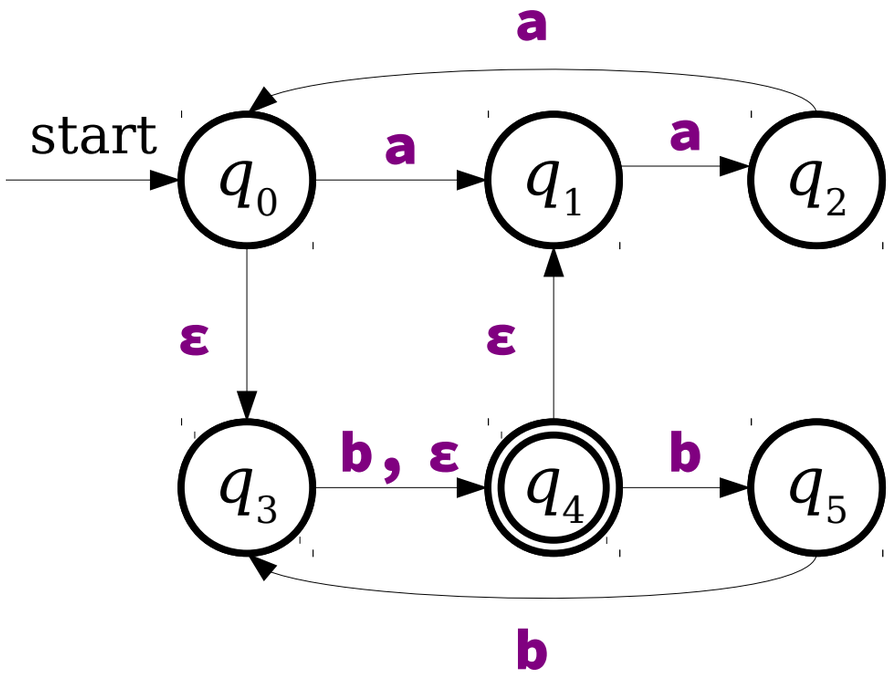
ε-Transitions

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



ε-Transitions

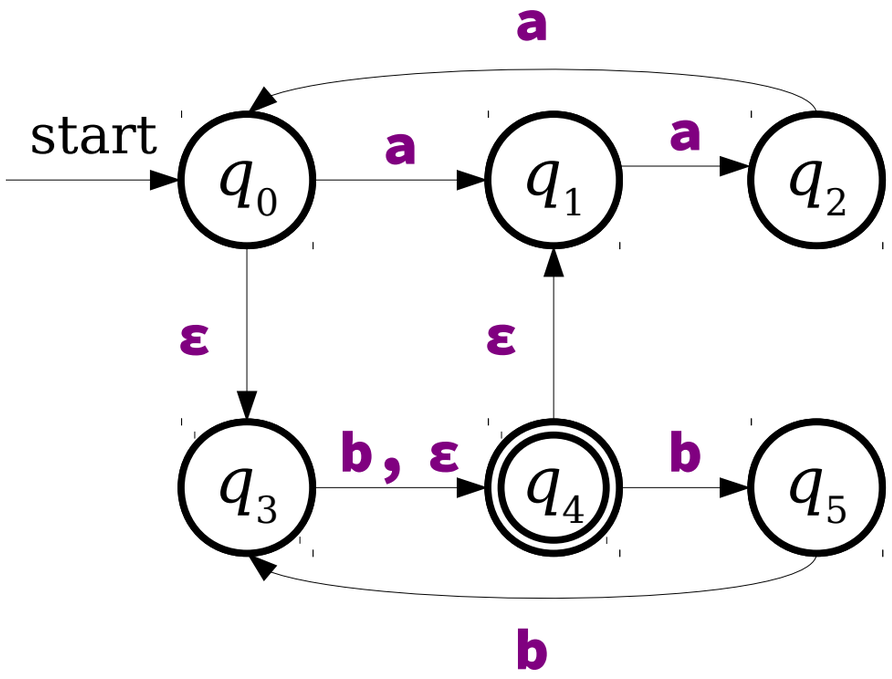
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

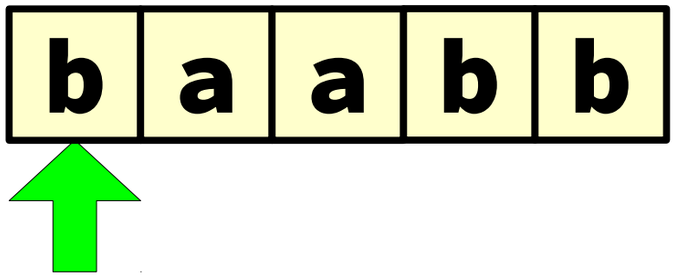


b	a	a	b	b

ε-Transitions

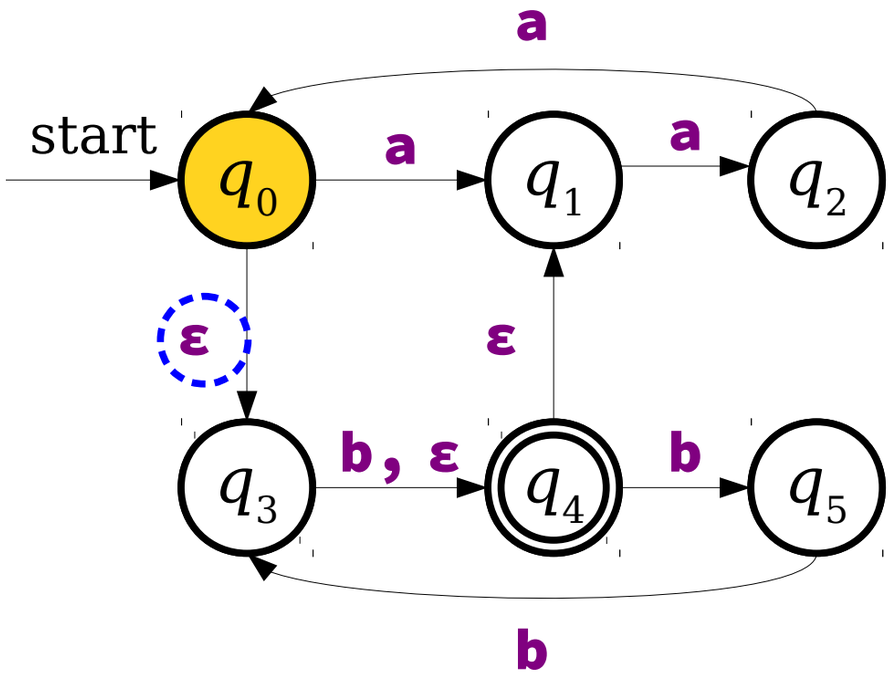
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

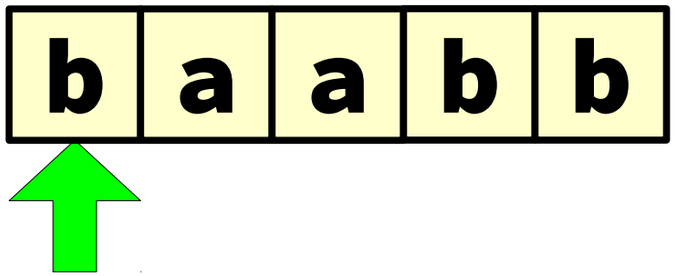




ε-Transitions

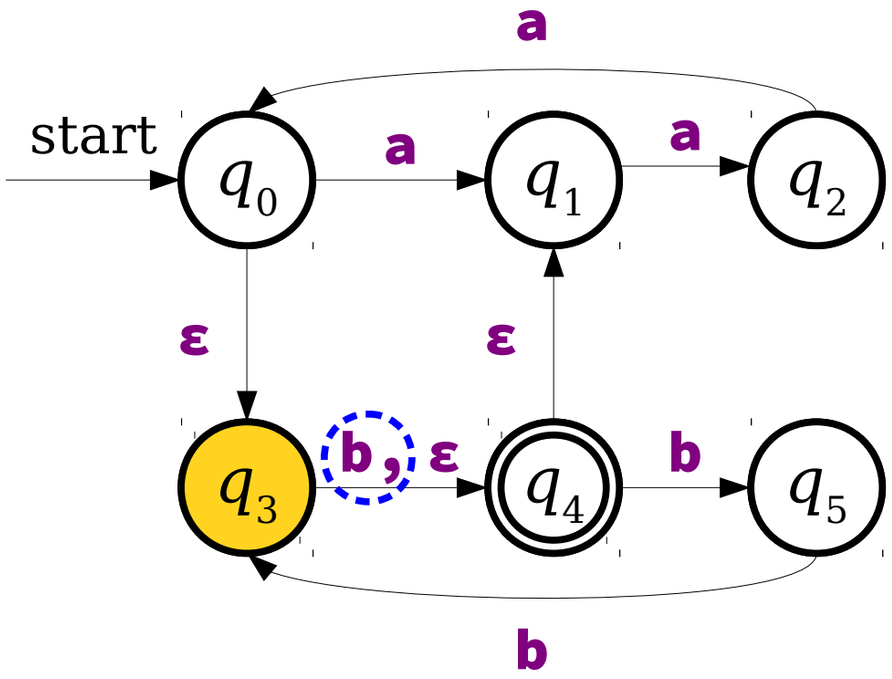
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

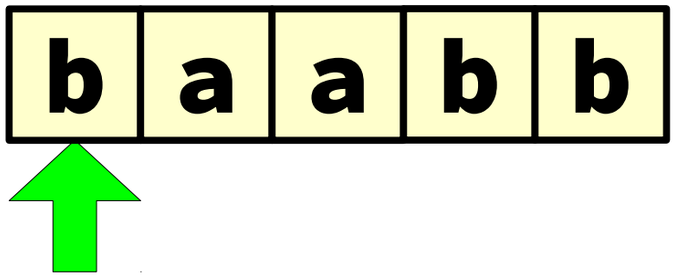




ε-Transitions

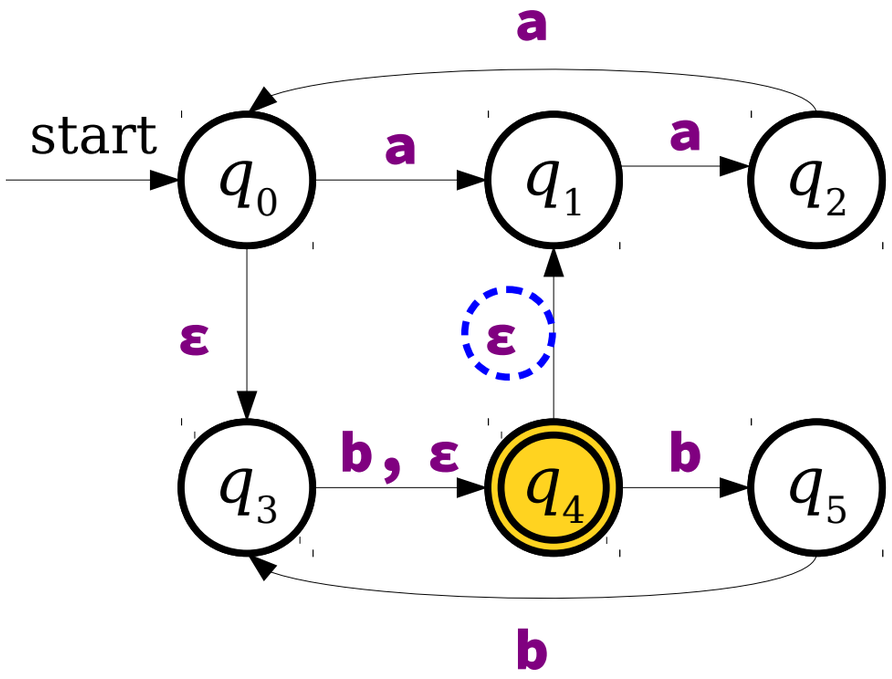
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

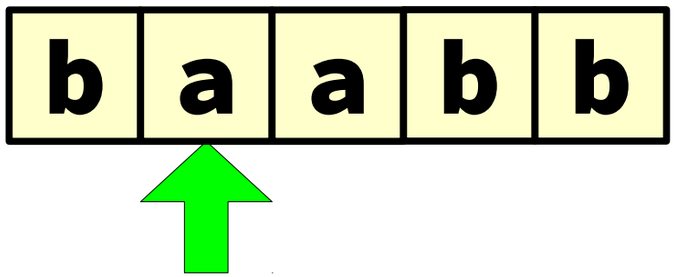




ε-Transitions

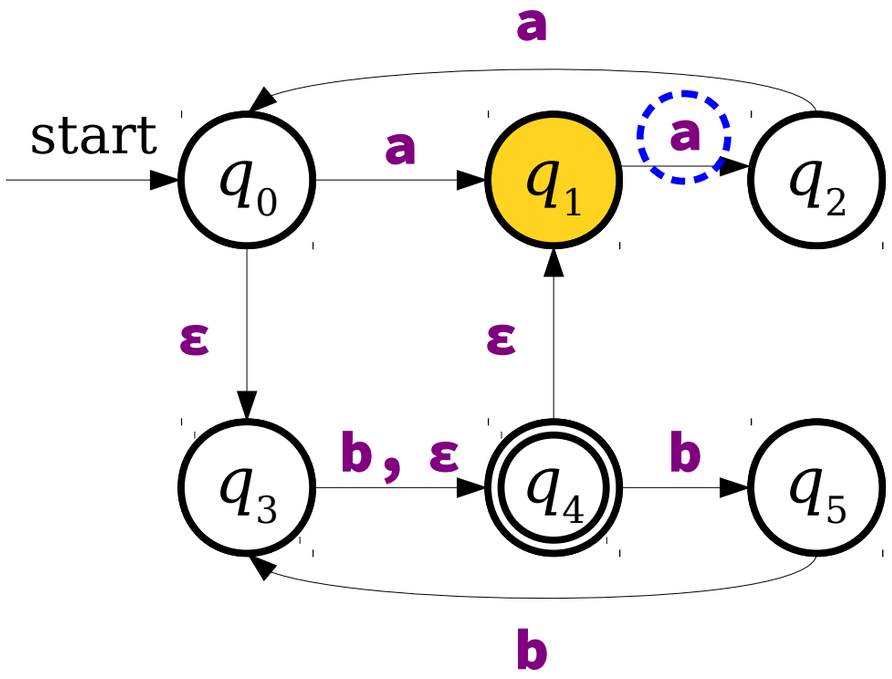
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

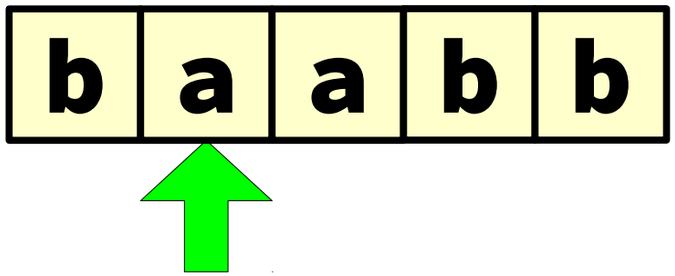




ε-Transitions

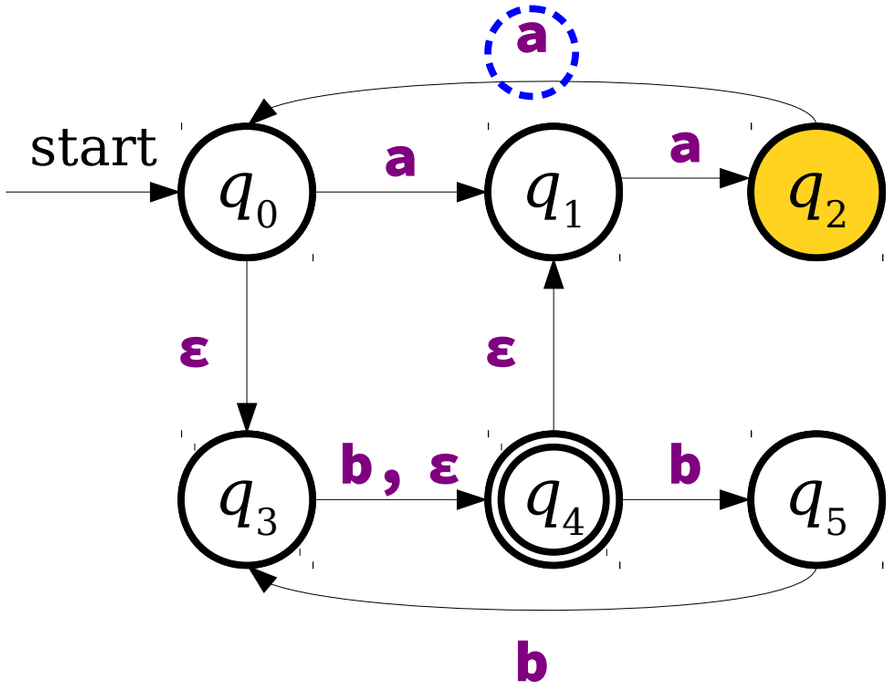
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

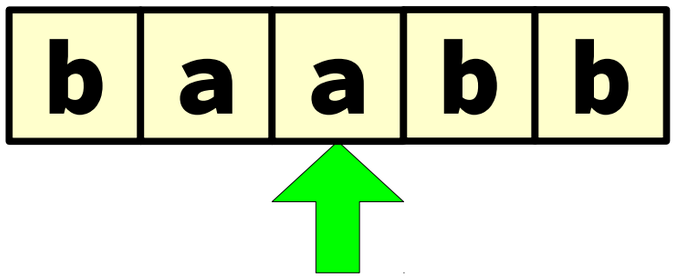




ε-Transitions

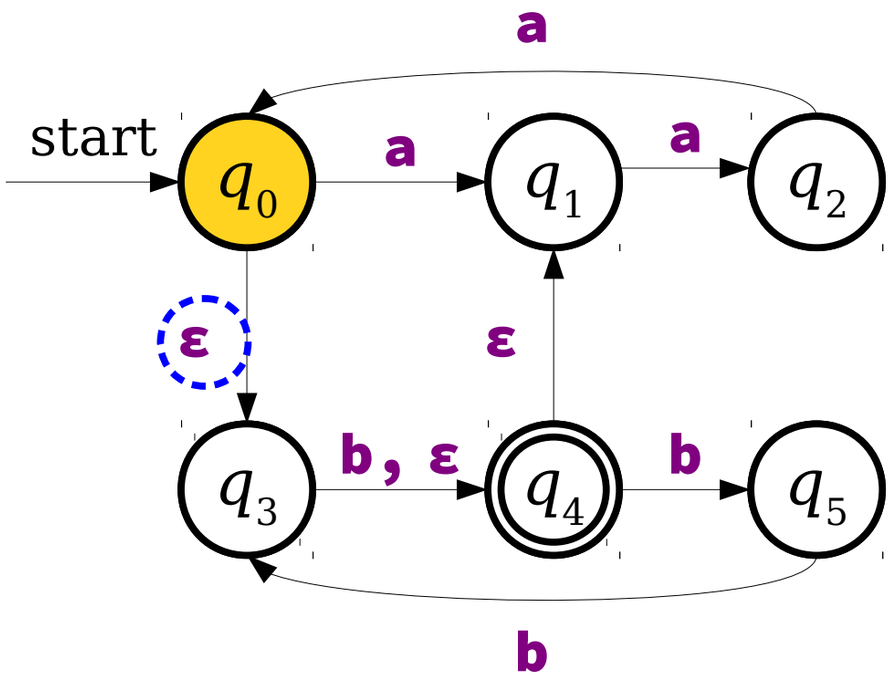
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

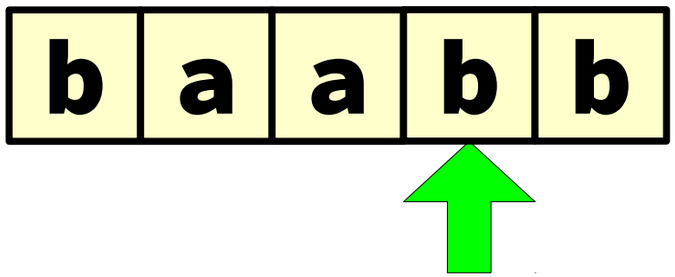




ε-Transitions

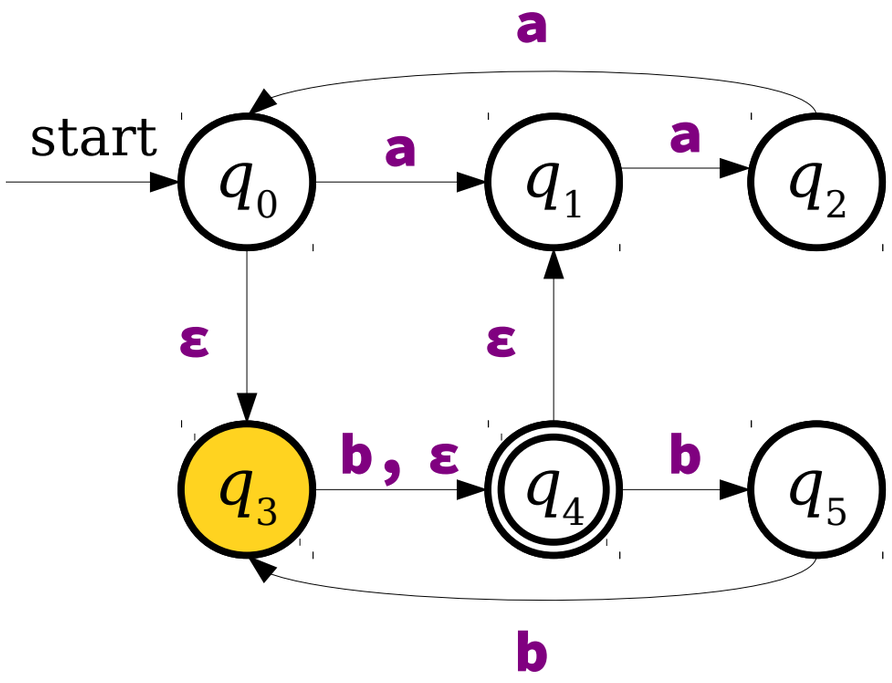
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



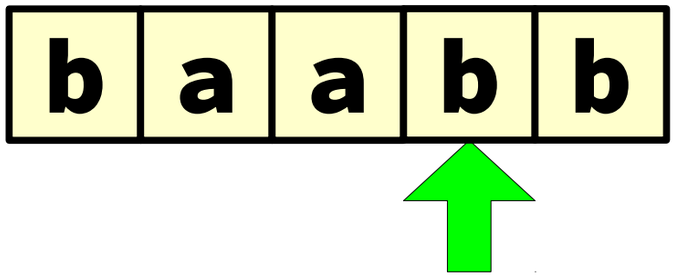


ε-Transitions

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



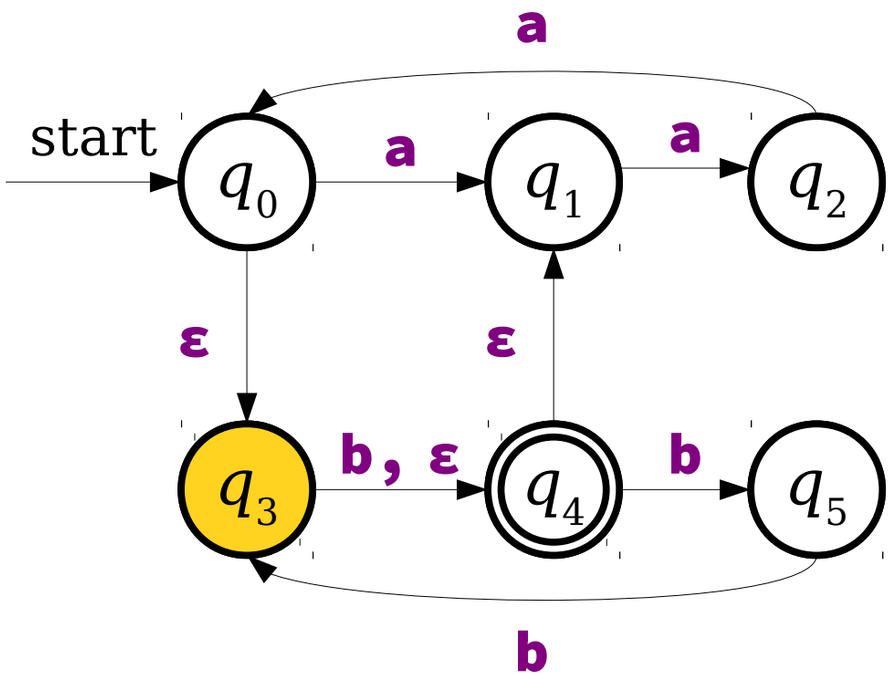
Give a sequence of states (like q_0, q_1, …) to follow from here that would process the rest of the string, but **not** result in accept.



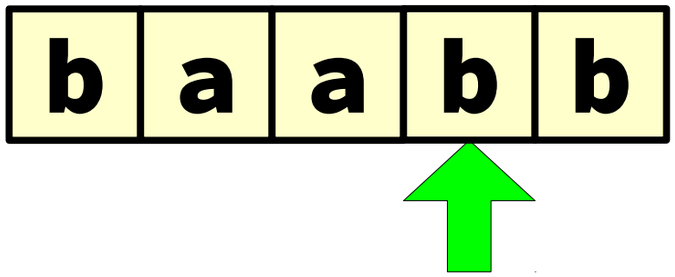
PollEv.com/
cs103spr26

ε-Transitions

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



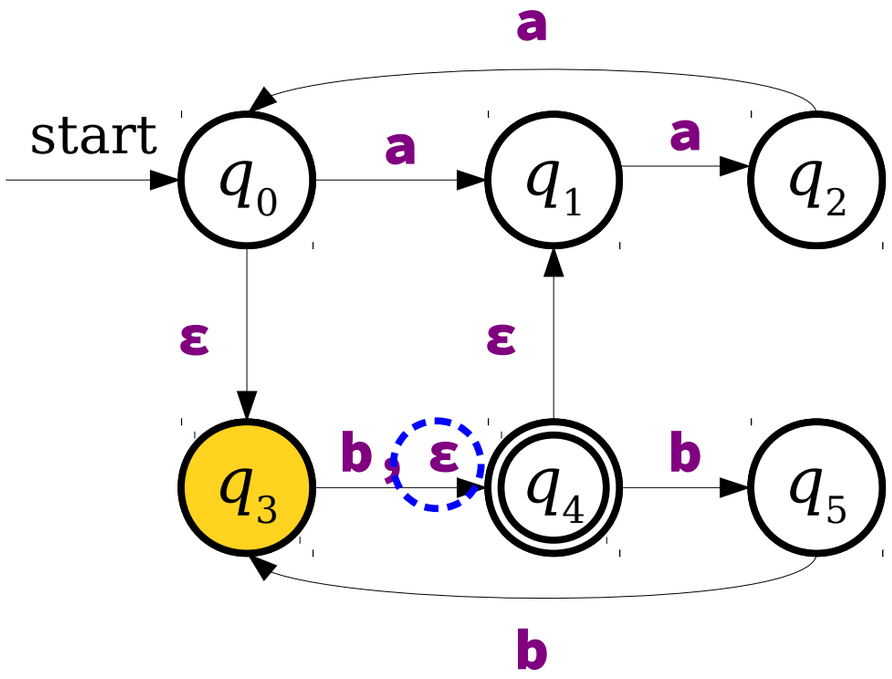
Give a sequence of states (like q_0, q_1, …) to follow from here that would process the rest of the string, and **would** result in accept.

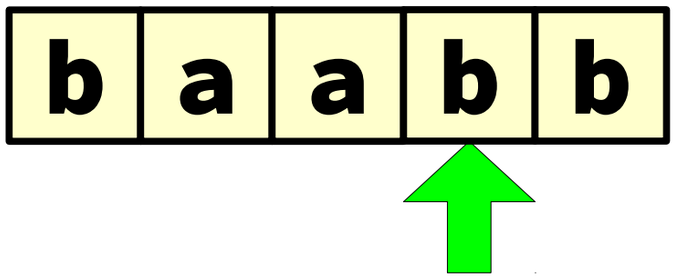


PollEv.com/
cs103spr26

ε-Transitions

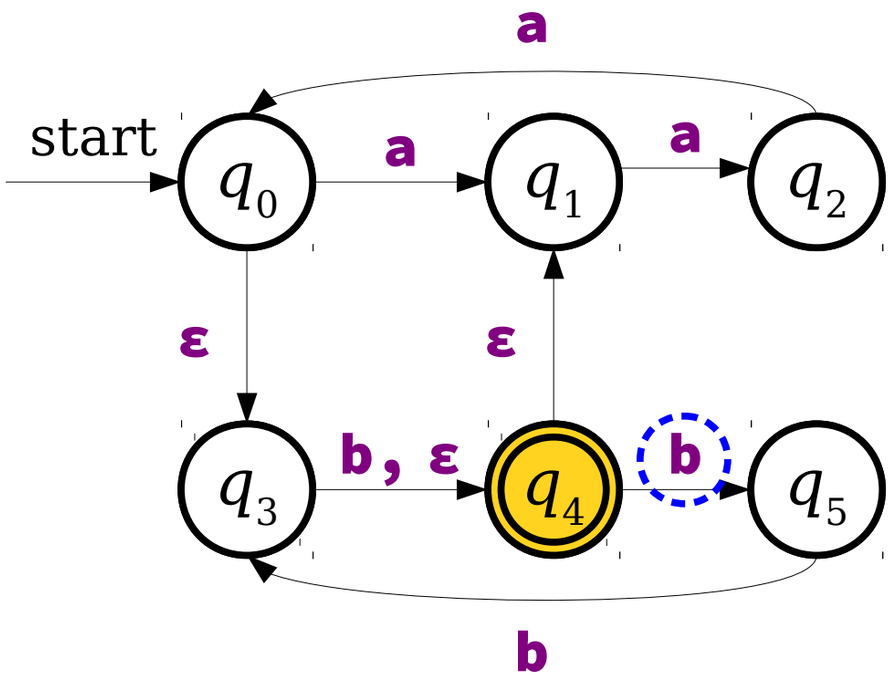
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

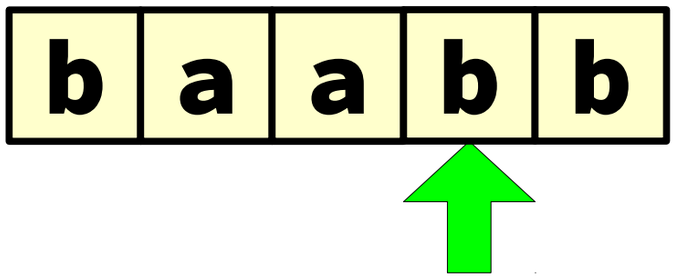




ε-Transitions

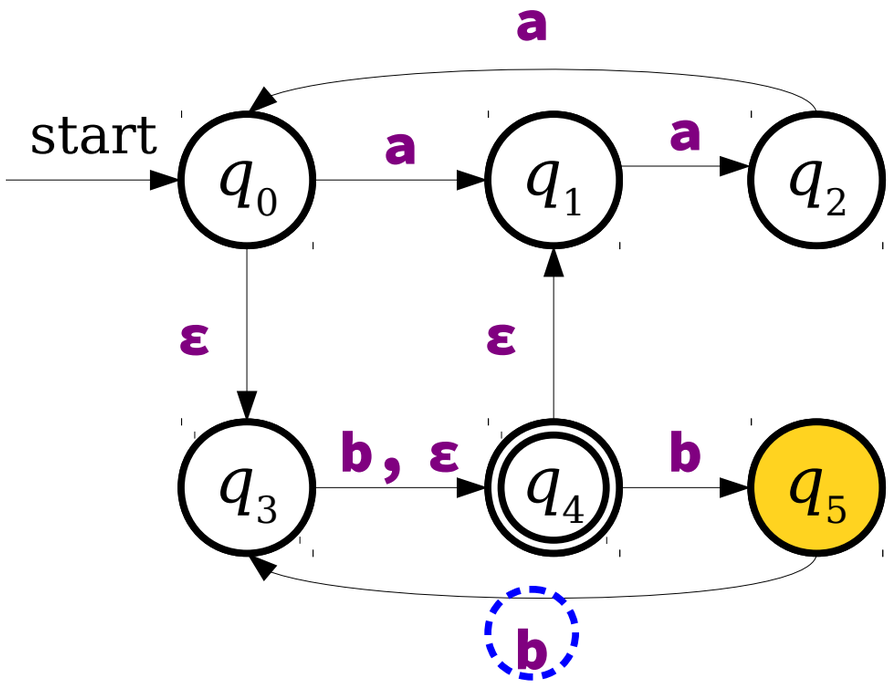
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

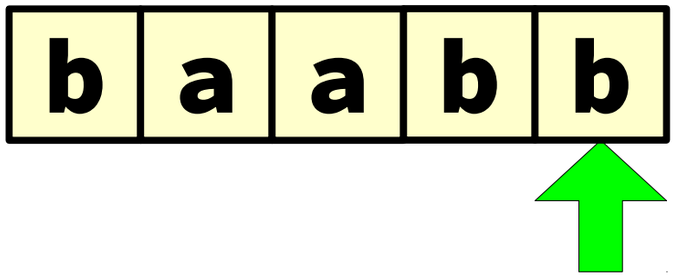




ε-Transitions

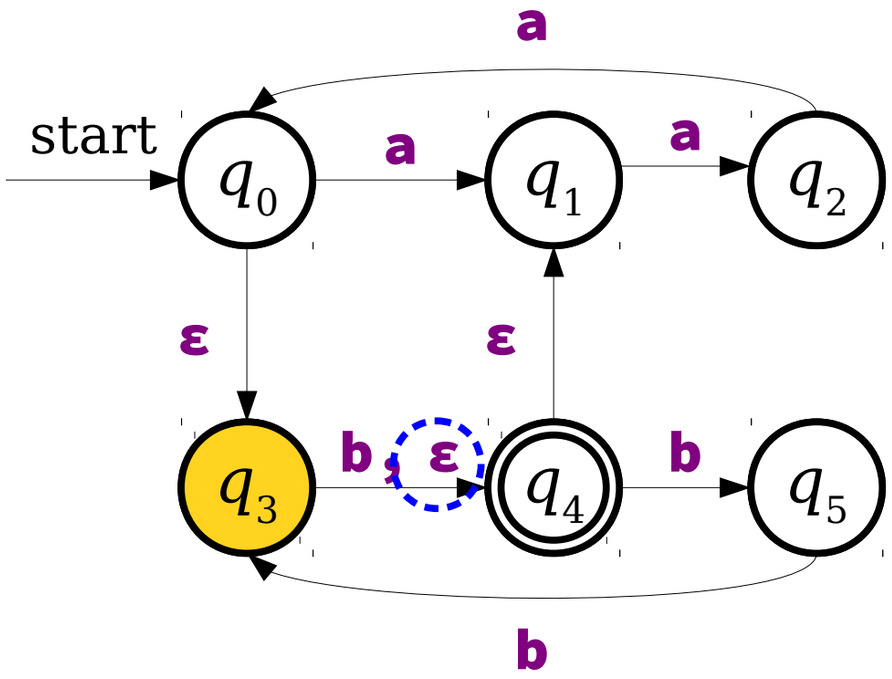
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.





ε-Transitions

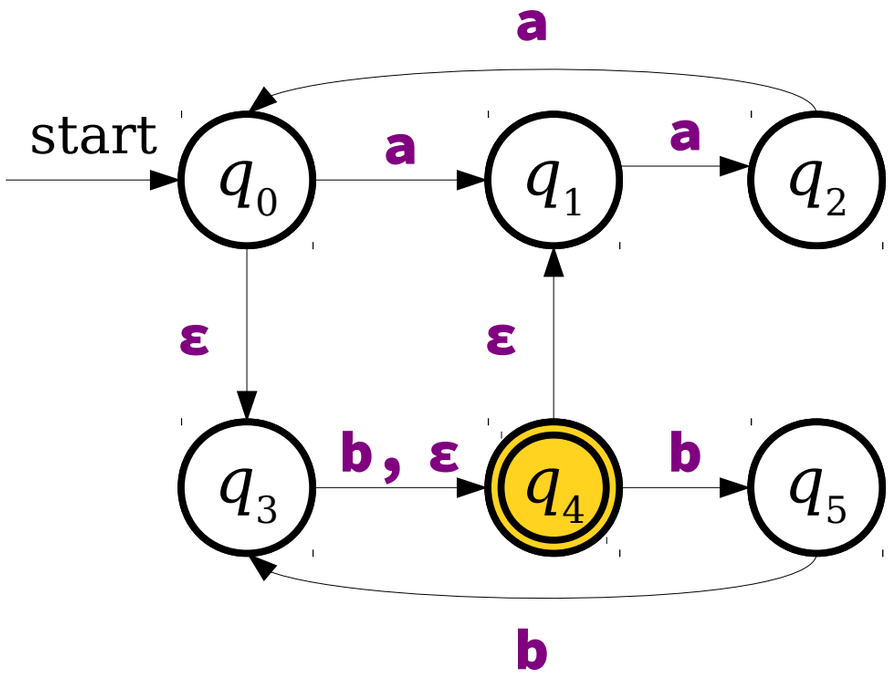
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



b	a	a	b	b

ε-Transitions

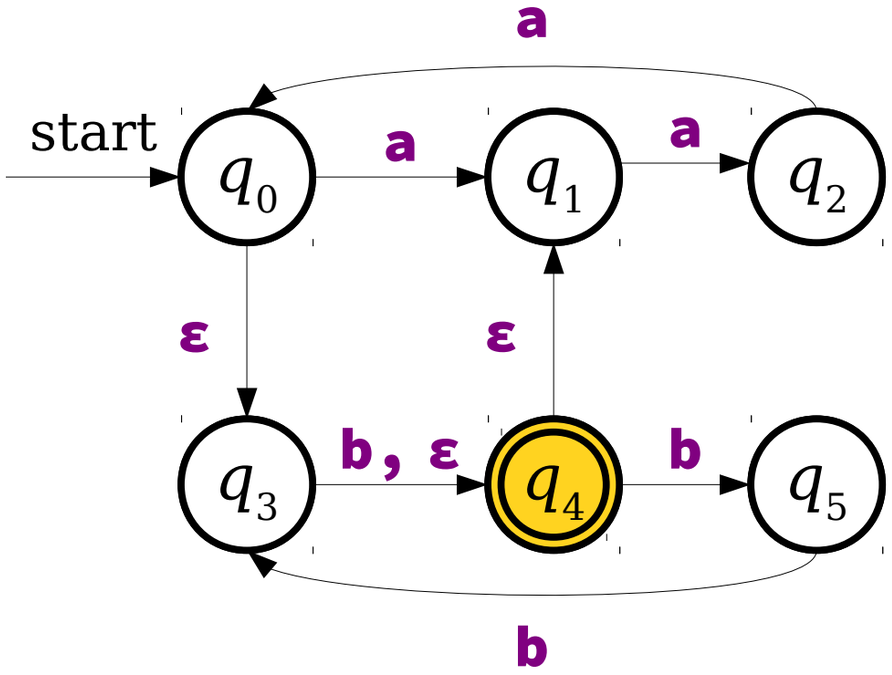
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



b	a	a	b	b

ε-Transitions

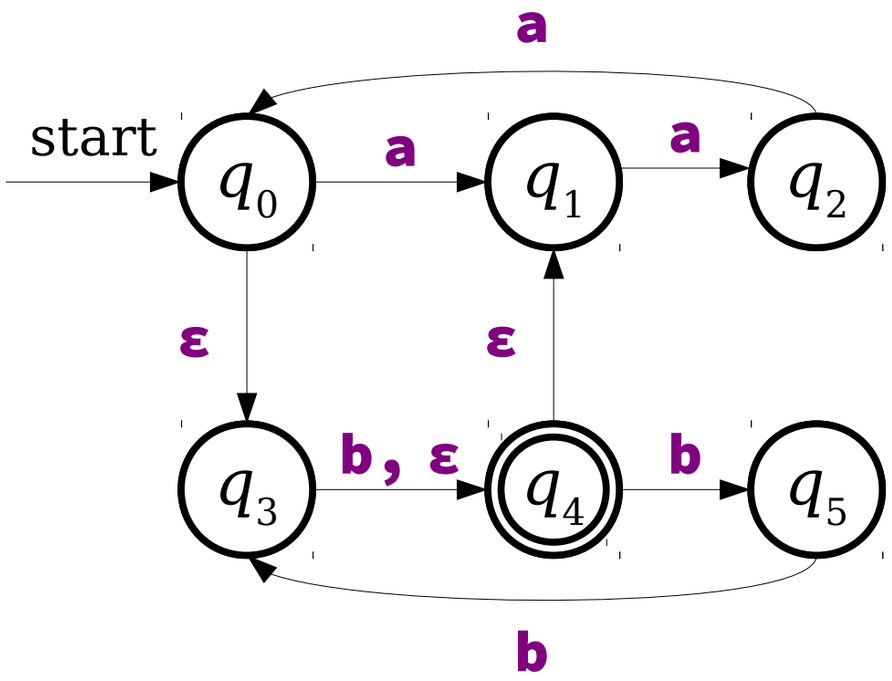
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



b	a	a	b	b

ε-Transitions

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



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

ε-Transitions

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

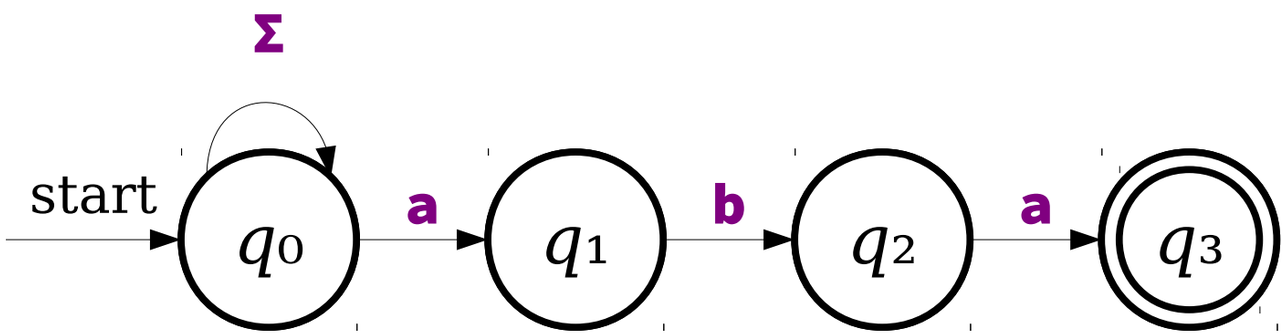
Intuiting Nondeterminism

- Nondeterministic machines are a serious departure from physical computers. How can we build up an intuition for them?
- There are two particularly useful frameworks for interpreting nondeterminism:
  - ***Perfect positive guessing***
  - ***Massive parallelism***

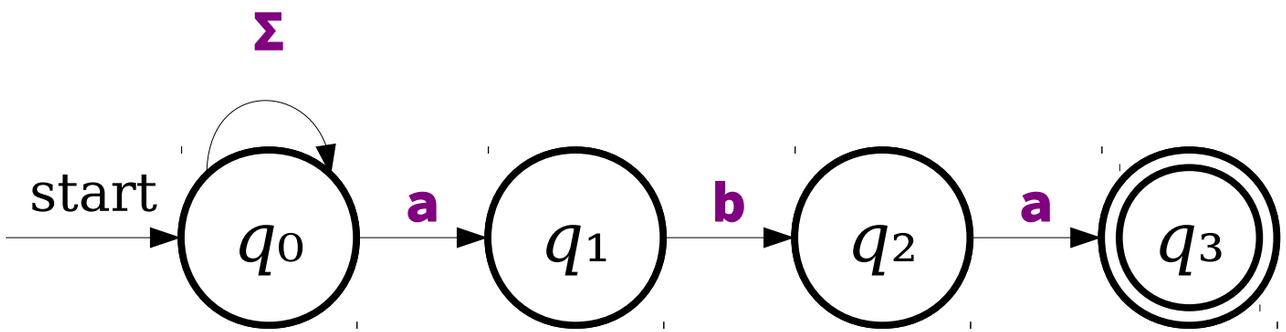
Perfect Positive Guessing

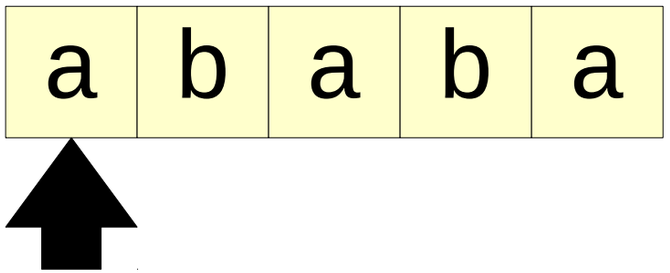
- We can view nondeterministic machines as having ***Magic Superpowers*** that enable them to guess choices that lead to an accepting state.
  - If there is at least one choice that leads to an accepting state, the machine will guess it.
  - If there are no choices, the machine guesses any one of the wrong guesses.
- There is no known way to physically model this intuition of nondeterminism – this is quite a departure from reality!

Perfect Positive Guessing

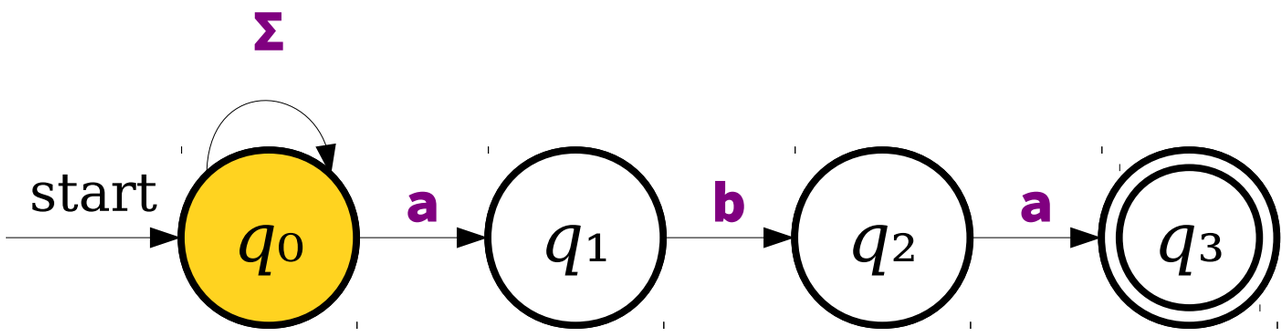


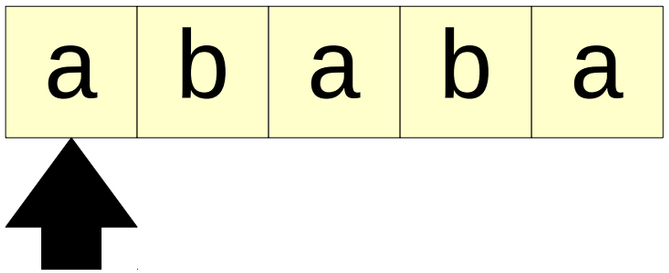
Perfect Positive Guessing



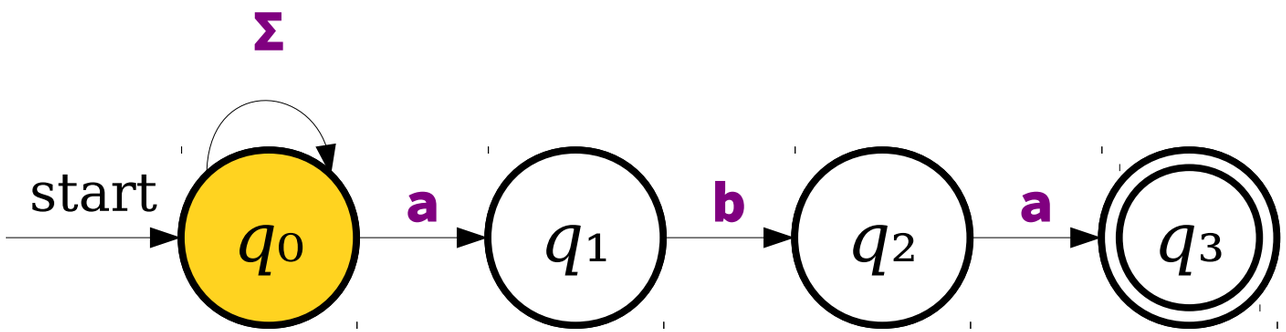


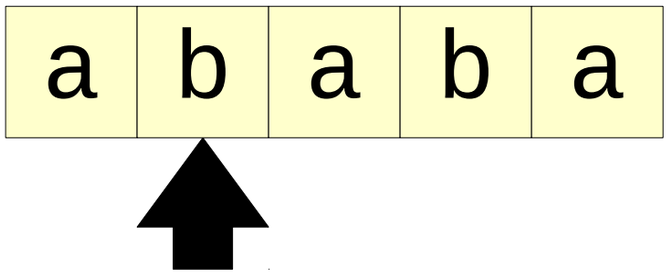
Perfect Positive Guessing



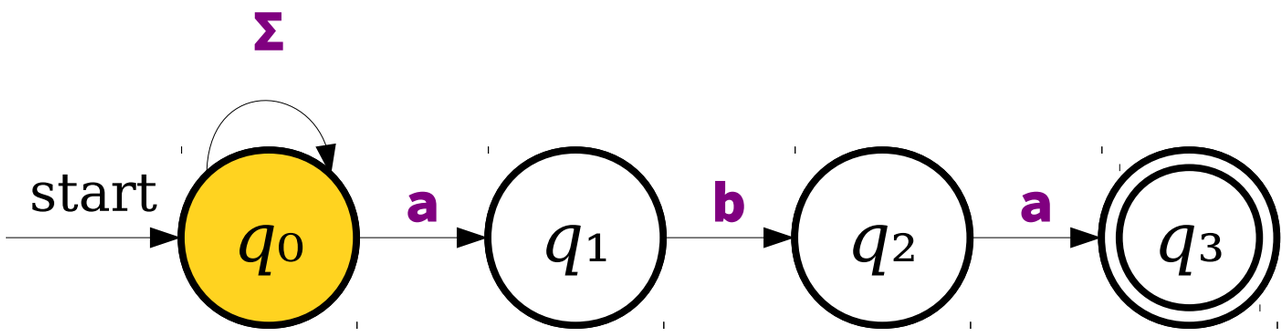


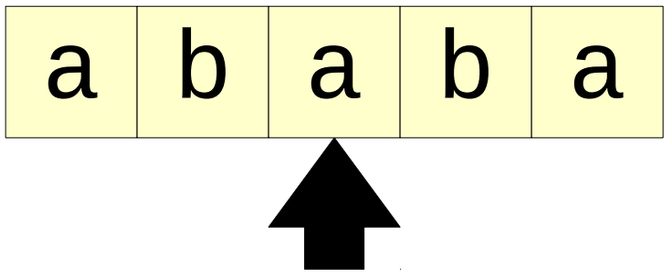
Perfect Positive Guessing



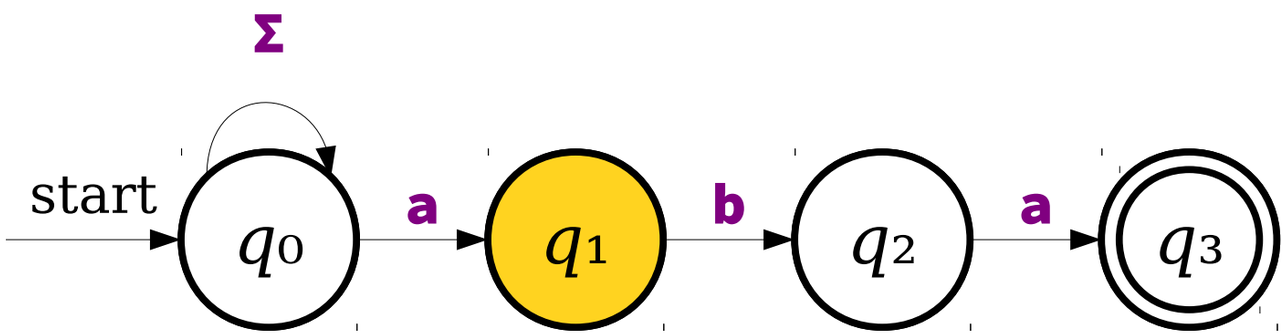


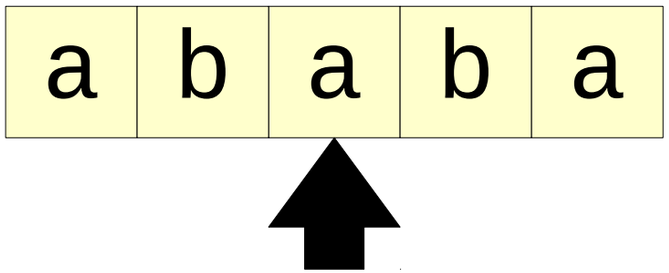
Perfect Positive Guessing



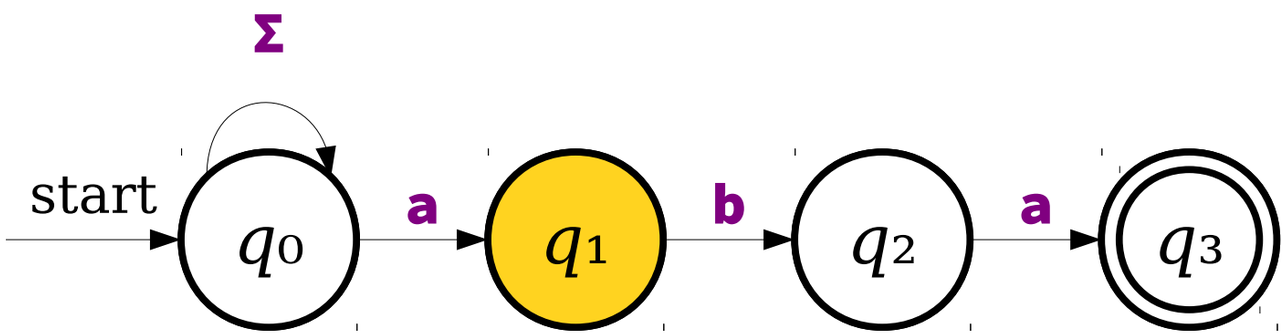


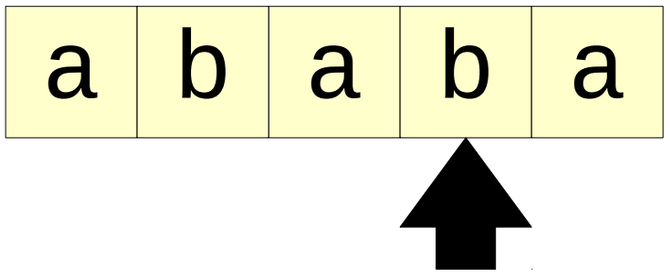
Perfect Positive Guessing



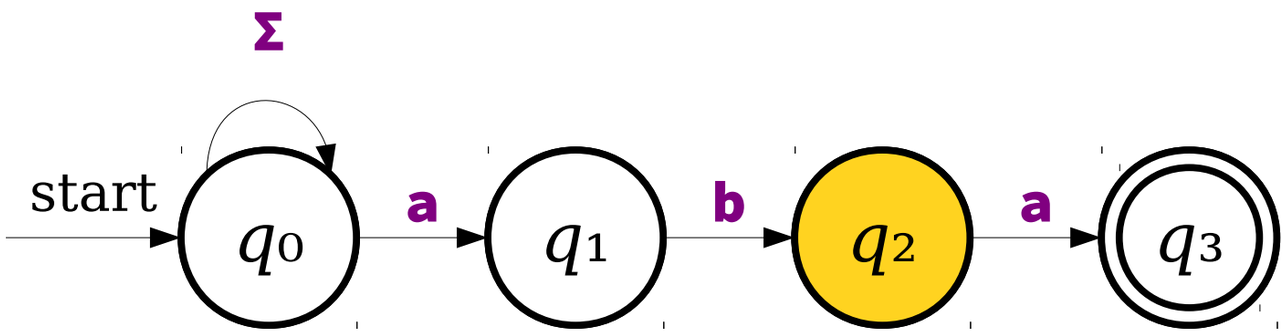


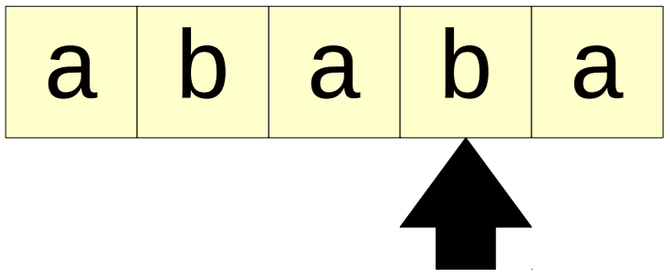
Perfect Positive Guessing



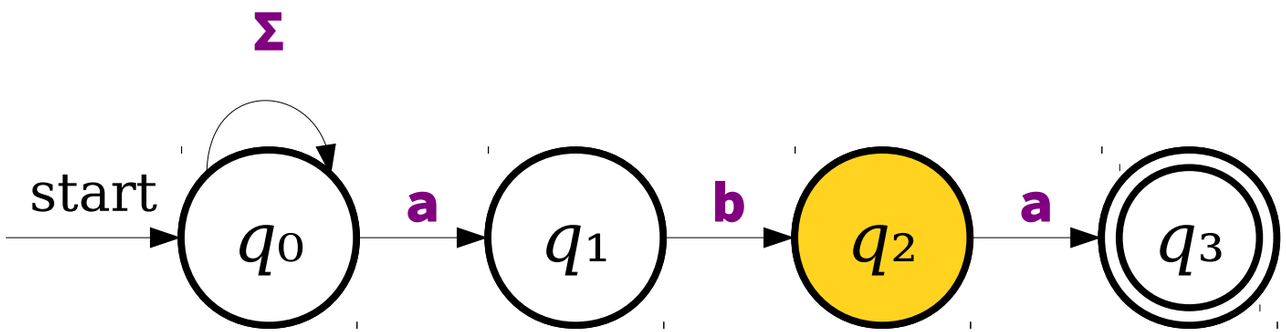


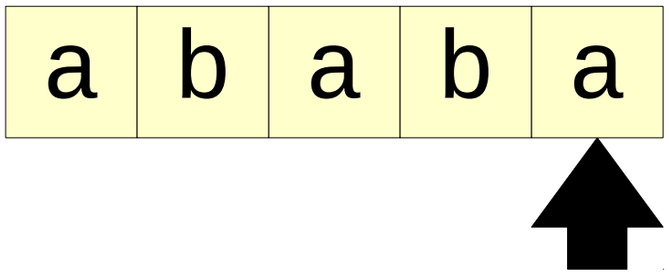
Perfect Positive Guessing



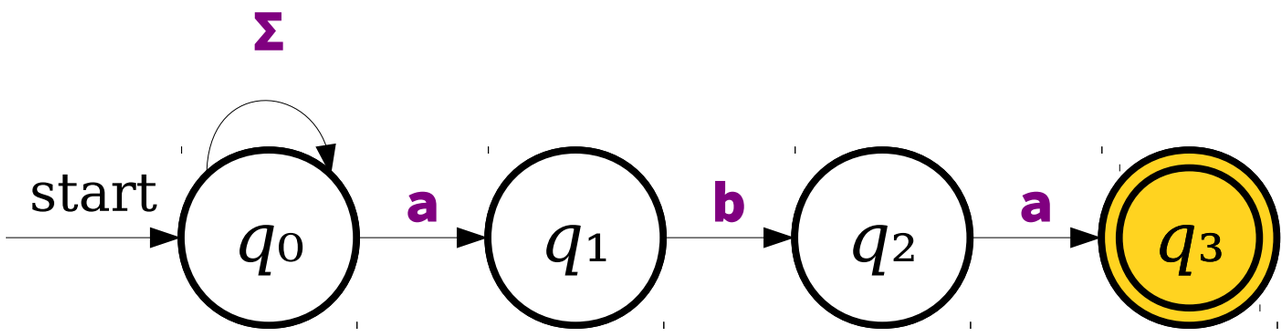


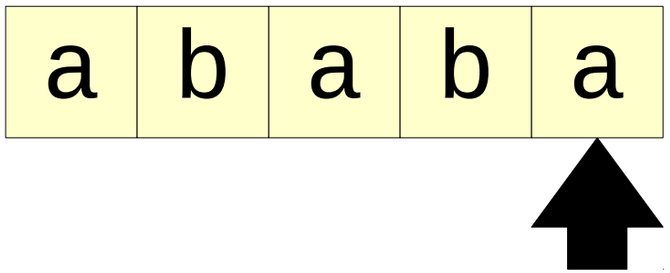
Perfect Positive Guessing



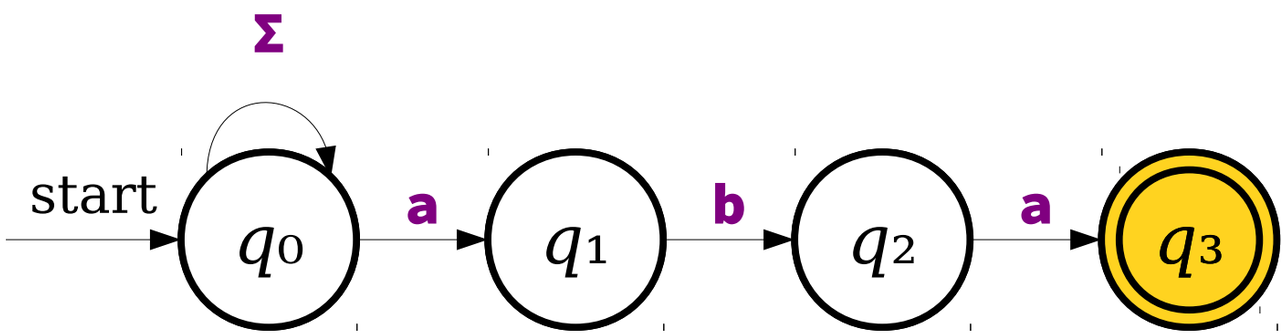


Perfect Positive Guessing





Perfect Positive Guessing

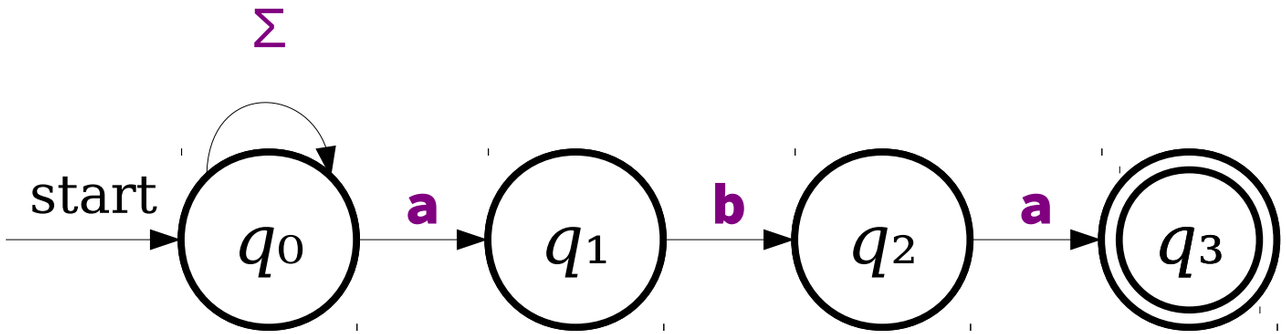


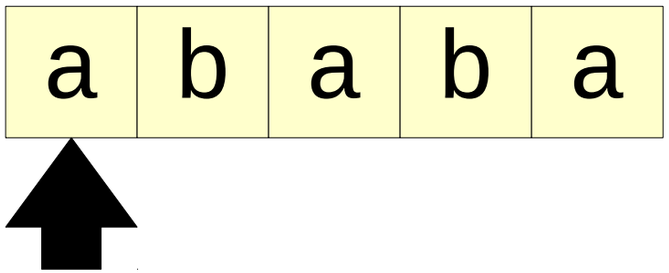
a	b	a	b	a

Massive Parallelism

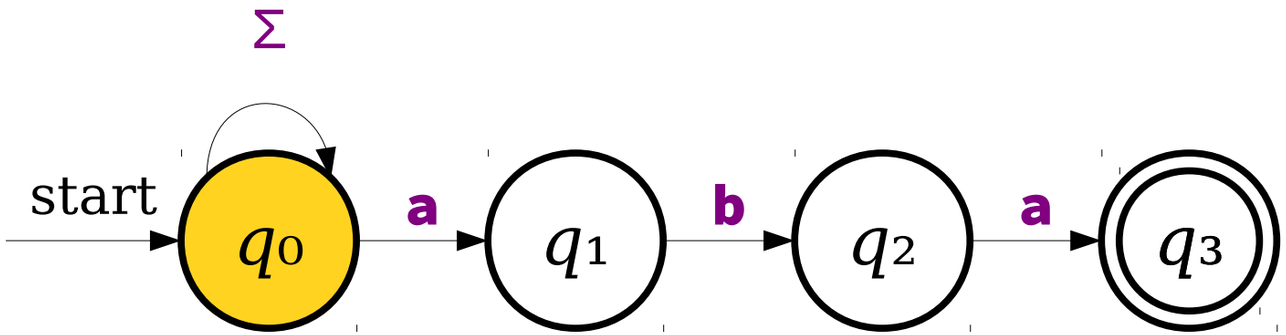
- An NFA can be thought of as a DFA that can be in many states at once.
- At each point in time, when the NFA needs to follow a transition, it tries all the options at the same time.

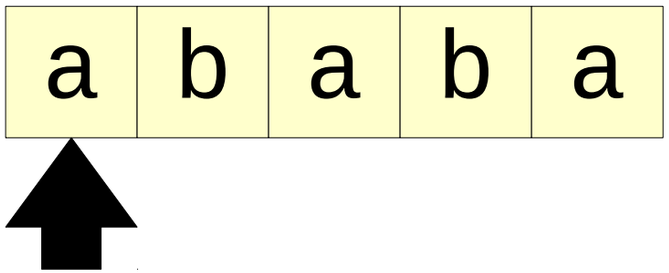
Massive Parallelism



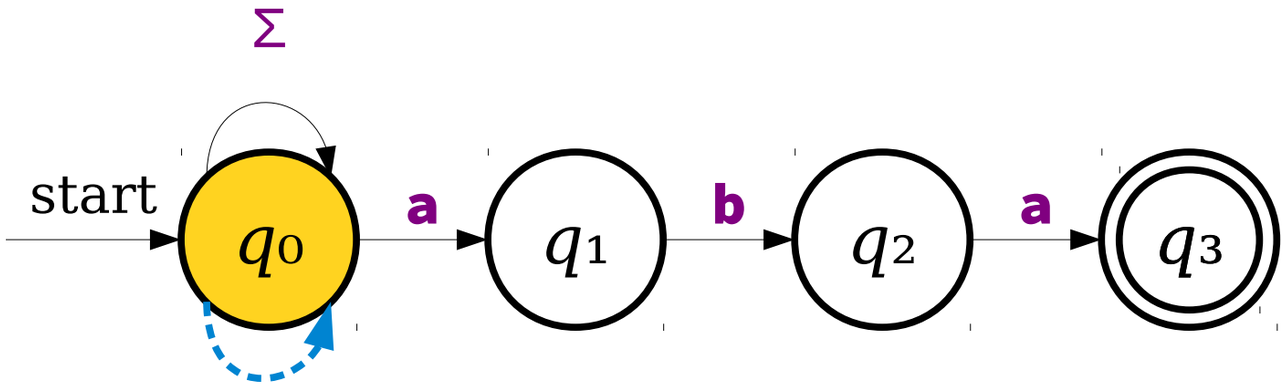


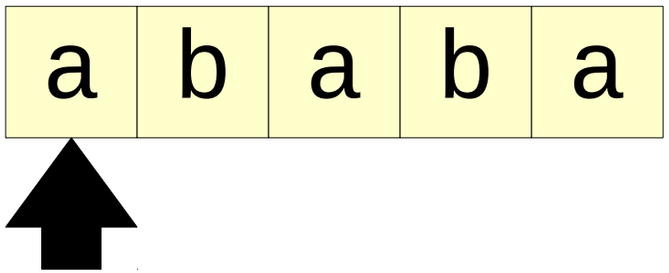
Massive Parallelism



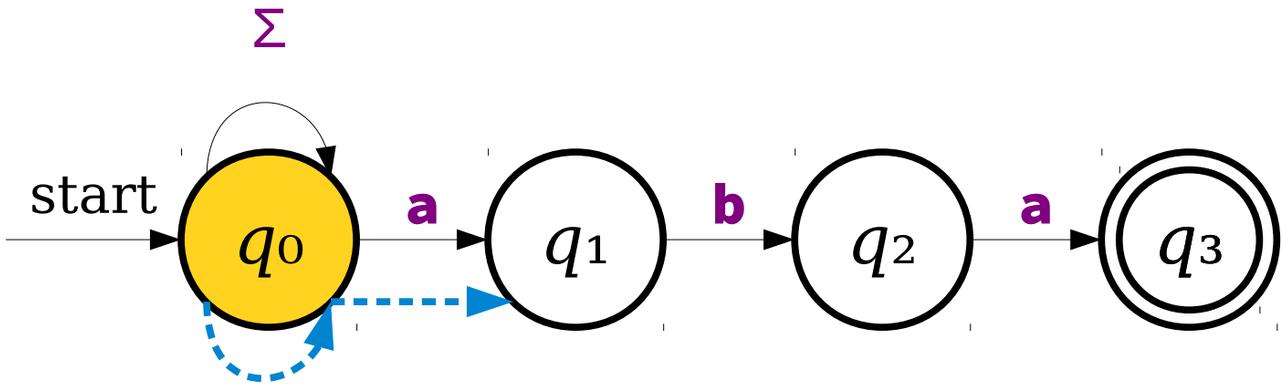


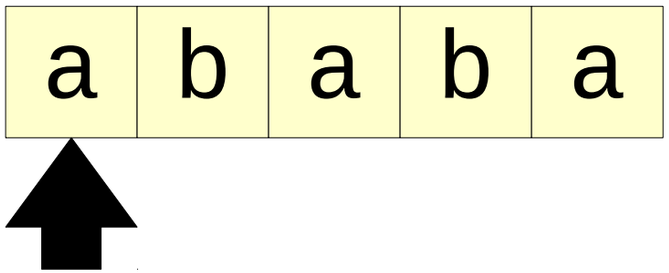
Massive Parallelism



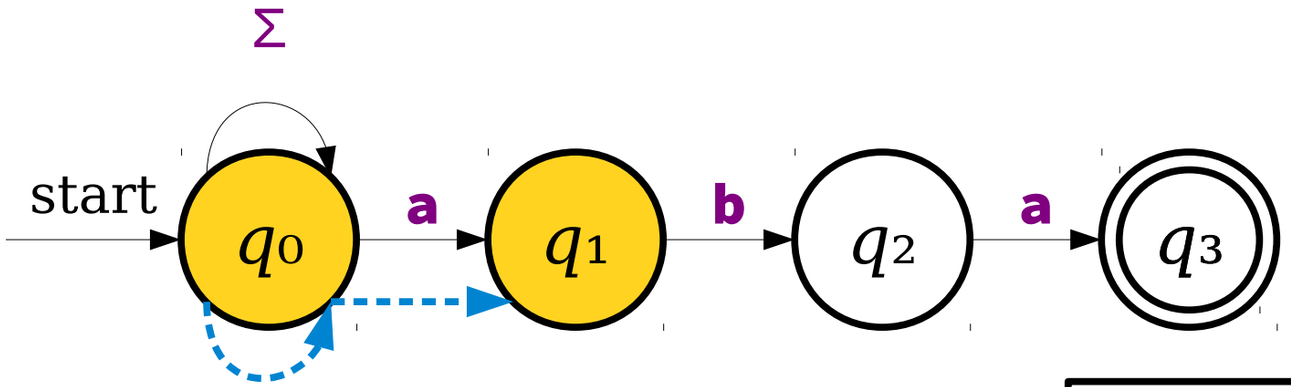


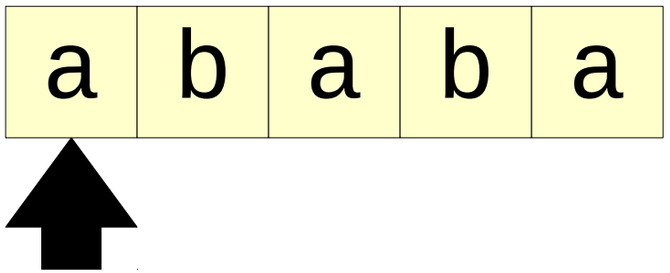
Massive Parallelism





Massive Parallelism

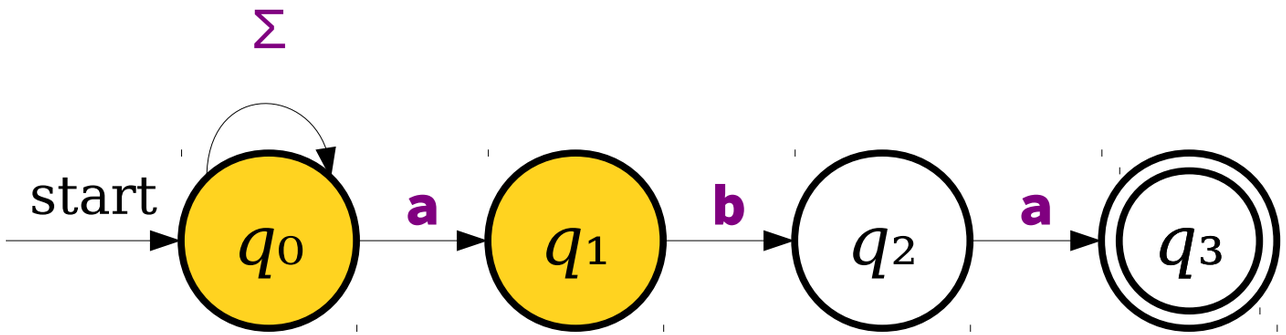


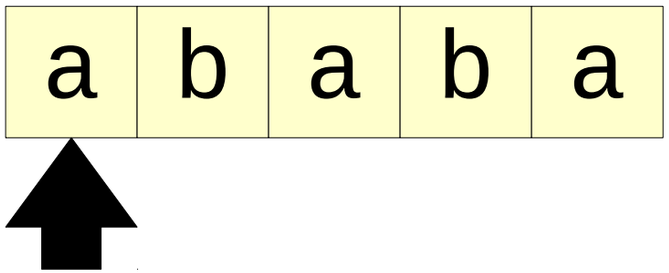


"Massive parallelism" matches how we defined the NFA transition function—we think of "where I am right now" as not just one state, but simultaneously all of some subset of the states.

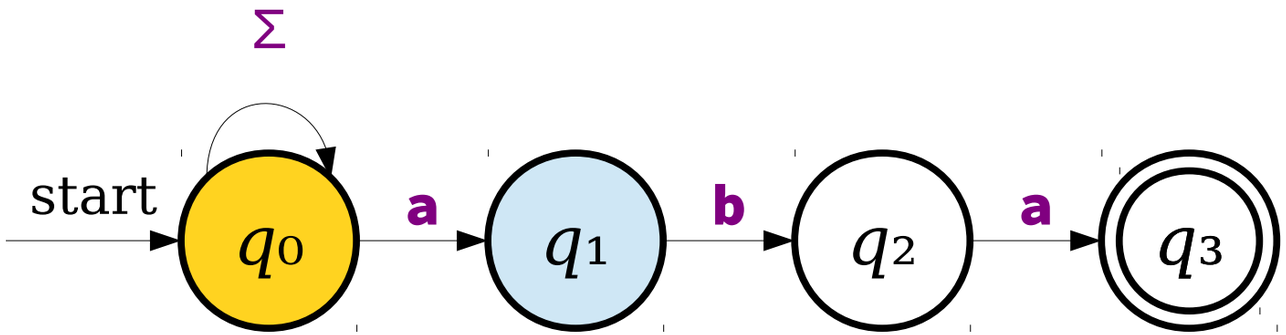
NFA: $\delta : (\wp(S) \times \Sigma) \rightarrow \wp(S)$

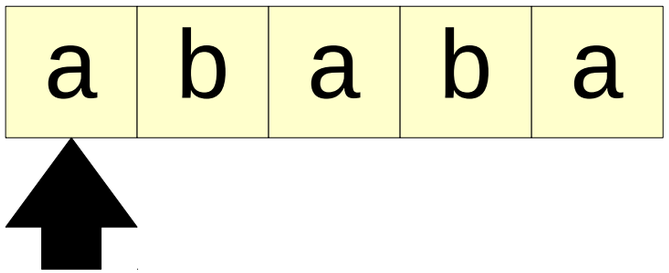
Massive Parallelism



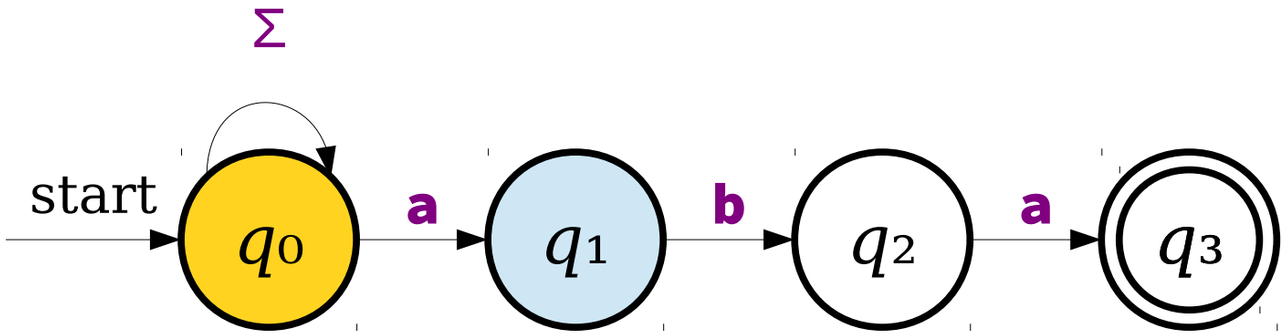


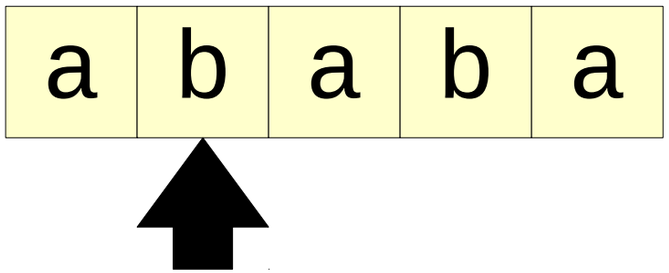
Massive Parallelism



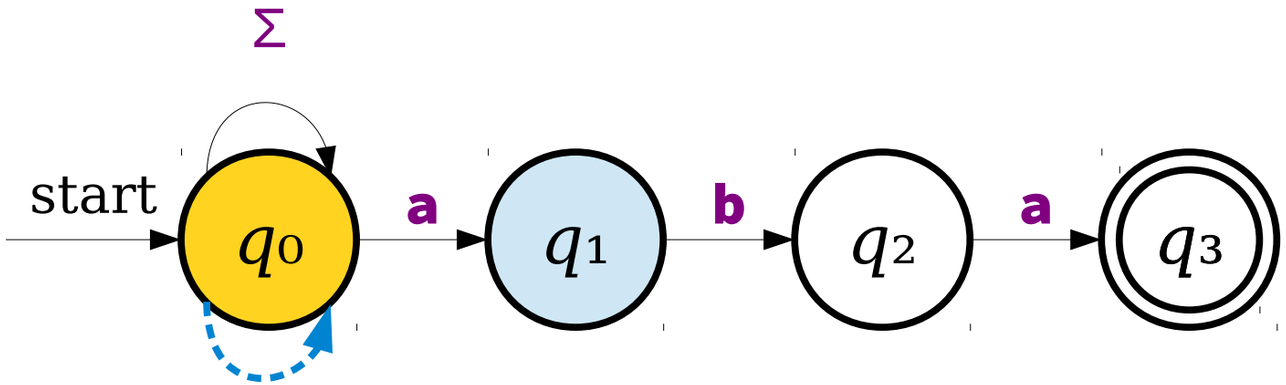


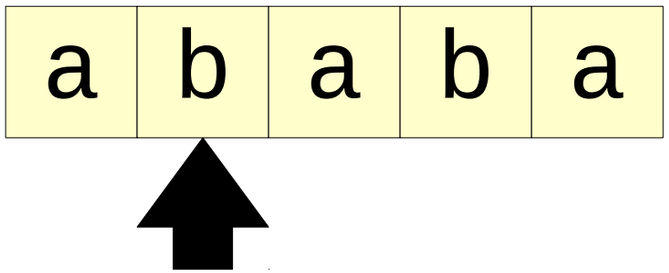
Massive Parallelism



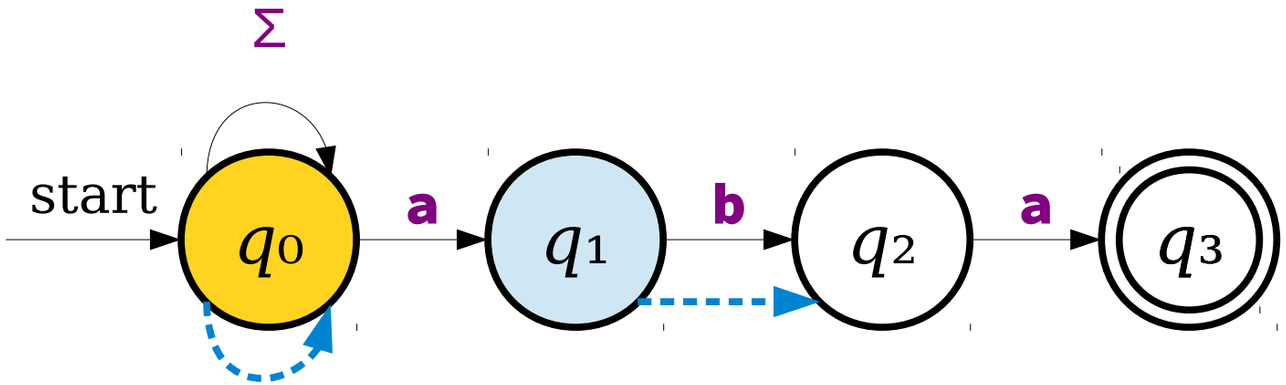


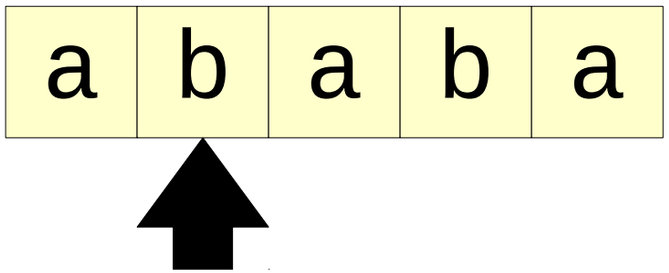
Massive Parallelism



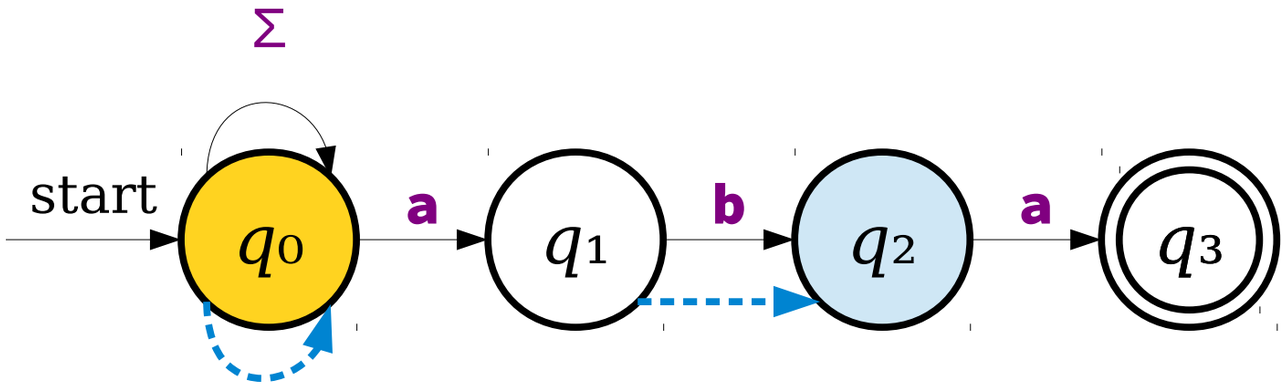


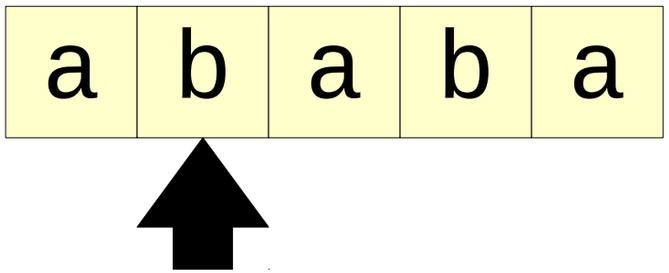
Massive Parallelism



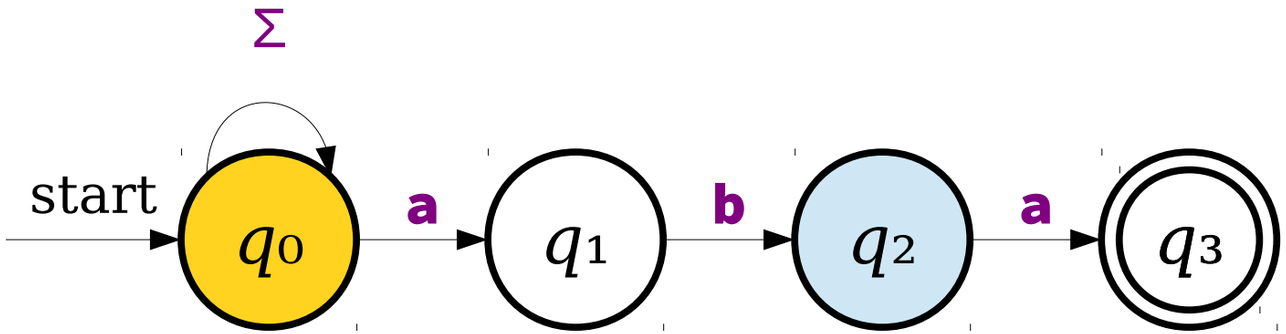


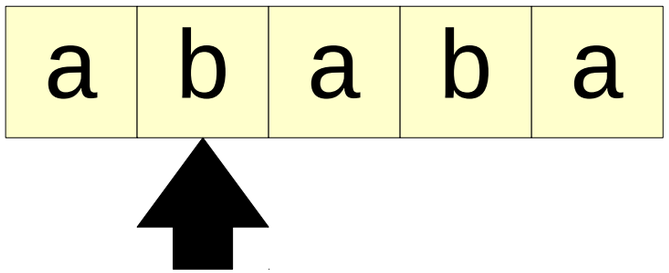
Massive Parallelism



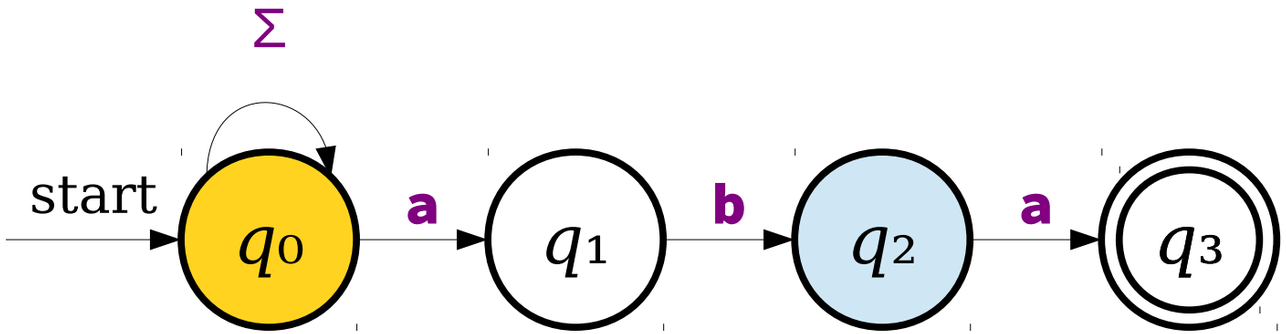


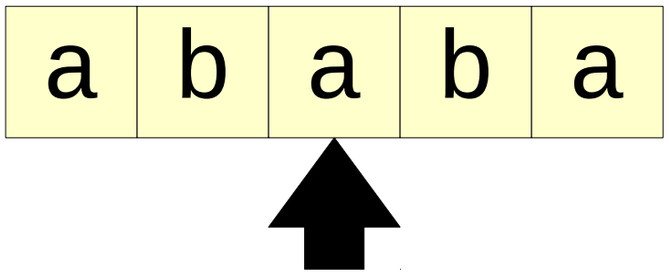
Massive Parallelism



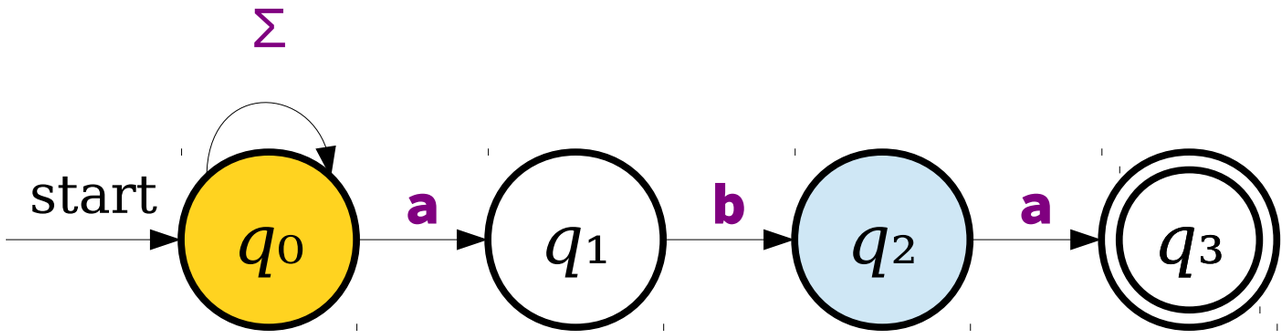


Massive Parallelism

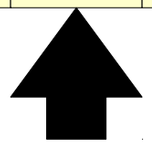




Massive Parallelism



a	b	a	b	a

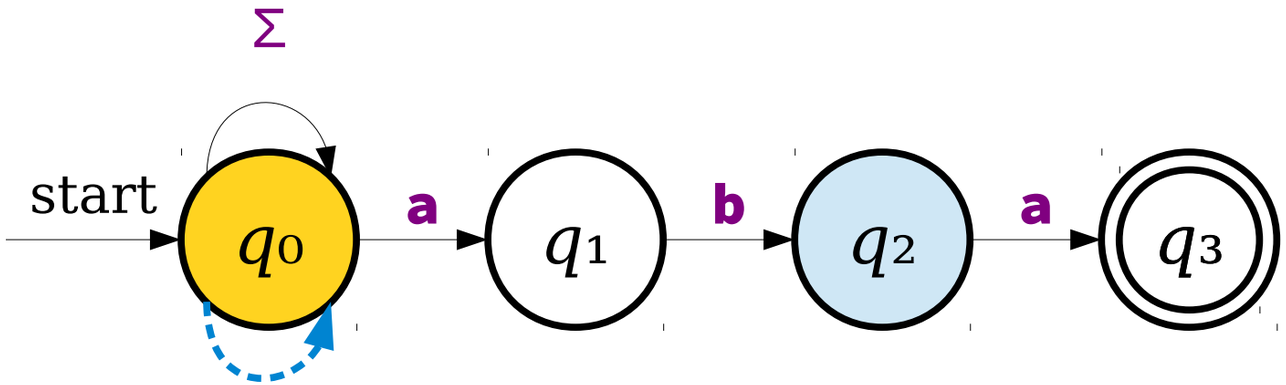


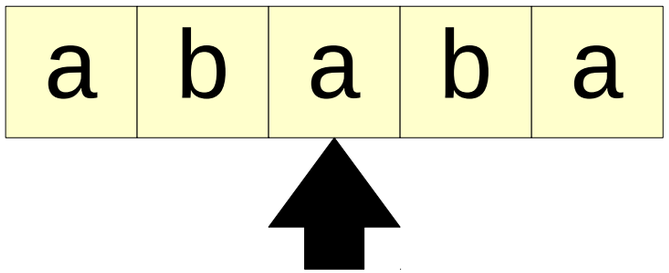
Reminder:
NFA: $\delta : (\wp(S) \times \Sigma) \rightarrow \wp(S)$

The output of $\delta((\{q_0, q_2\}, \mathbf{a}))$ is where we go next. What is that output?

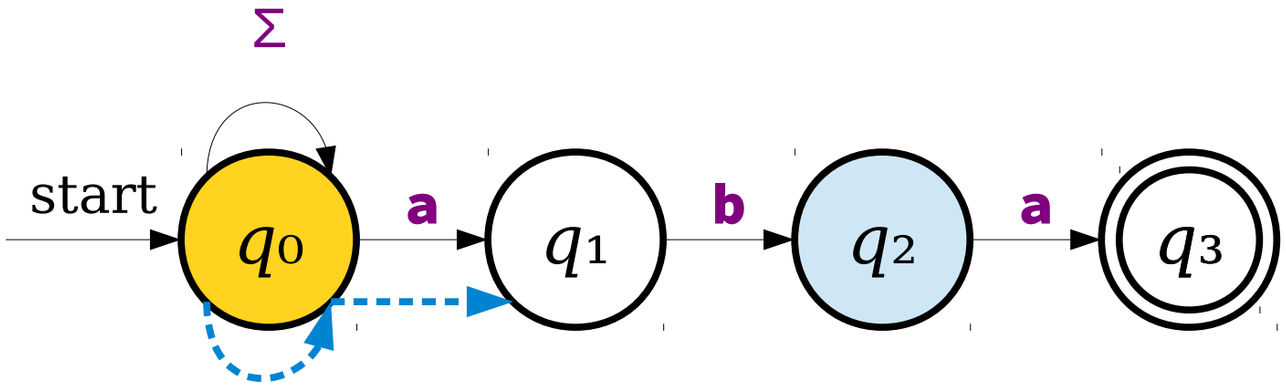
PollEv.com/
cs103spr26

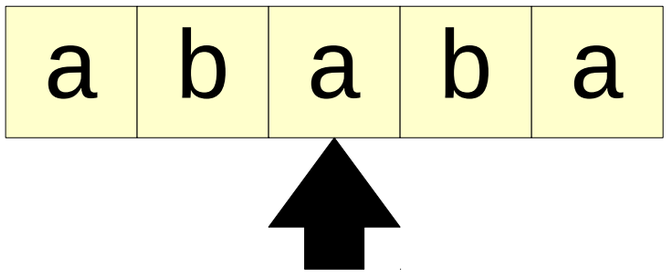
Massive Parallelism



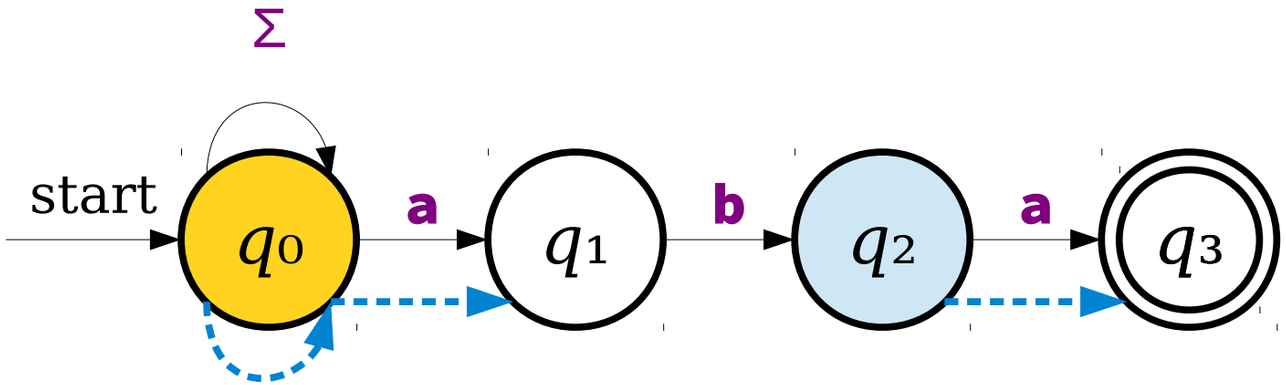


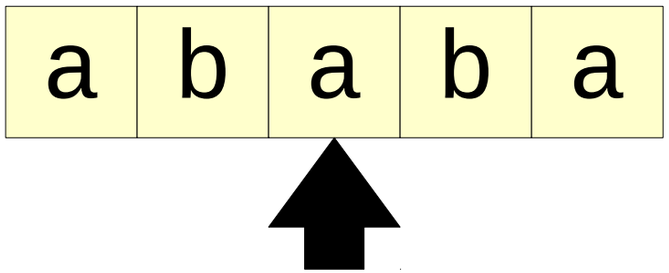
Massive Parallelism



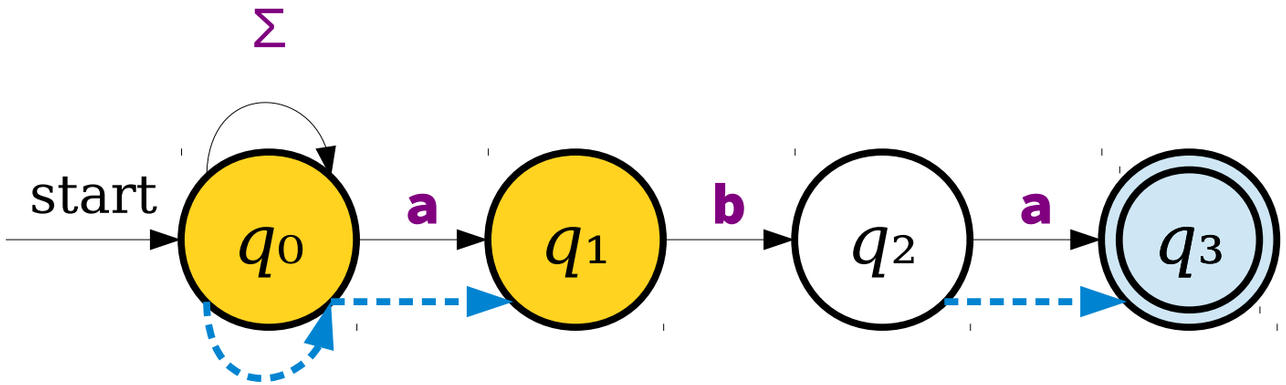


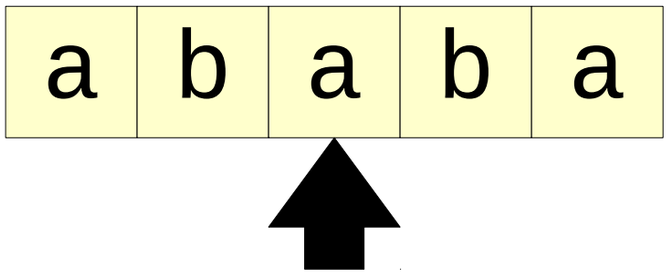
Massive Parallelism



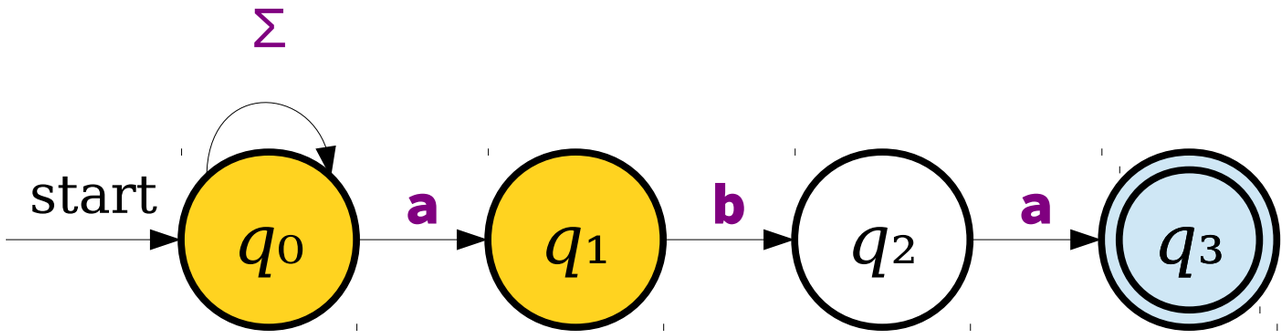


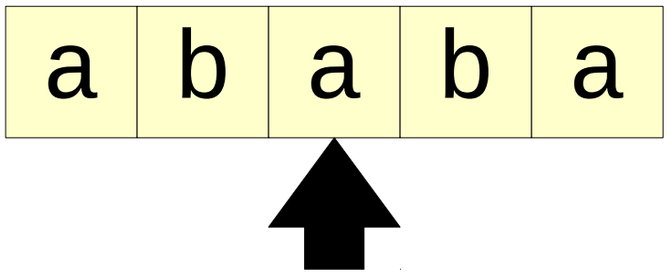
Massive Parallelism



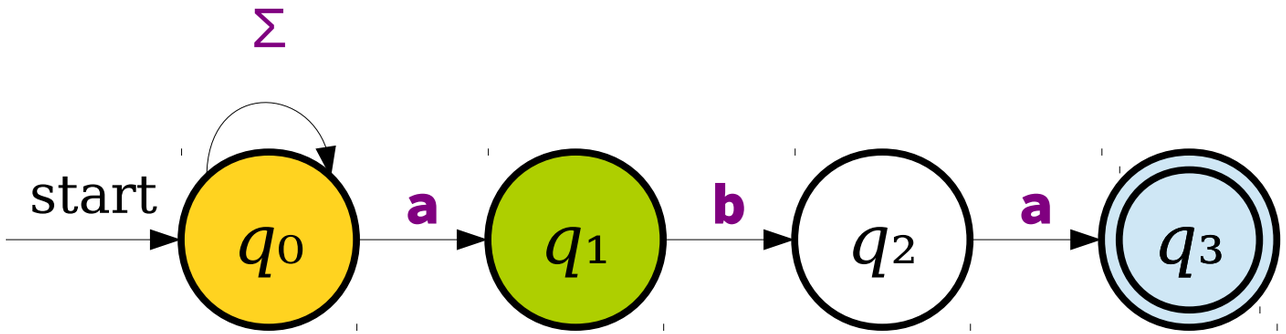


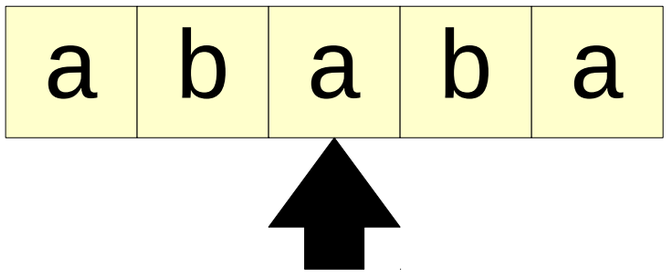
Massive Parallelism



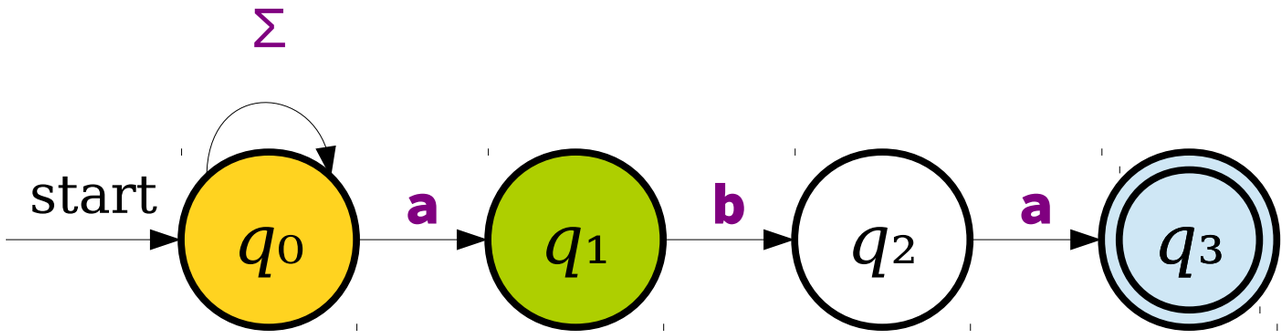


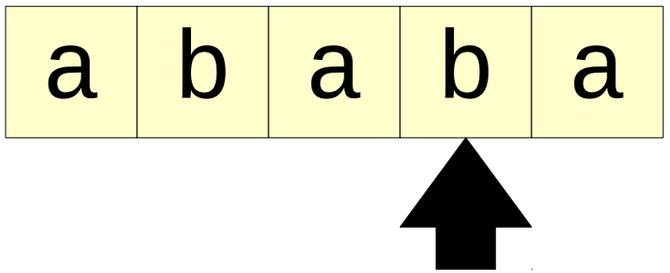
Massive Parallelism



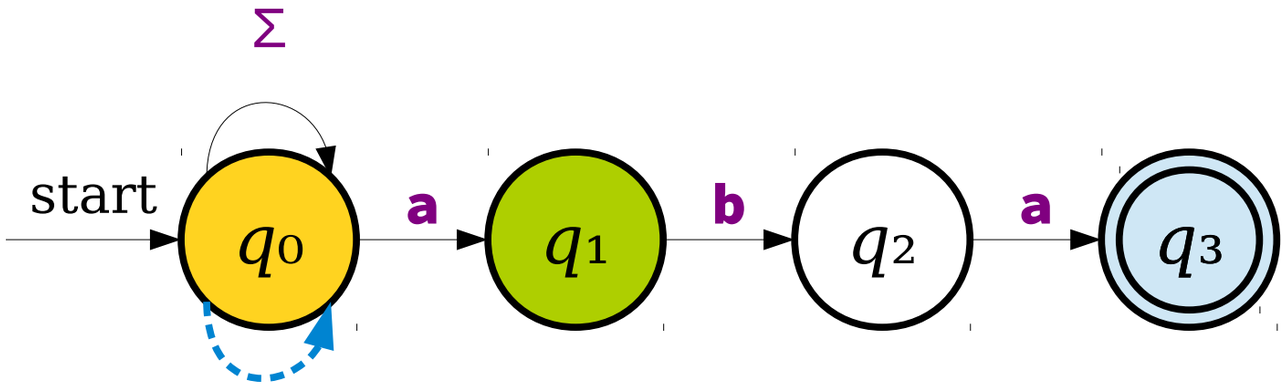


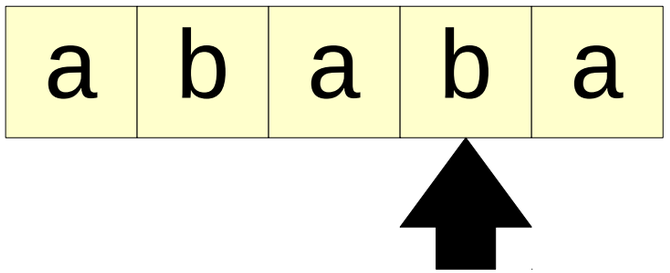
Massive Parallelism



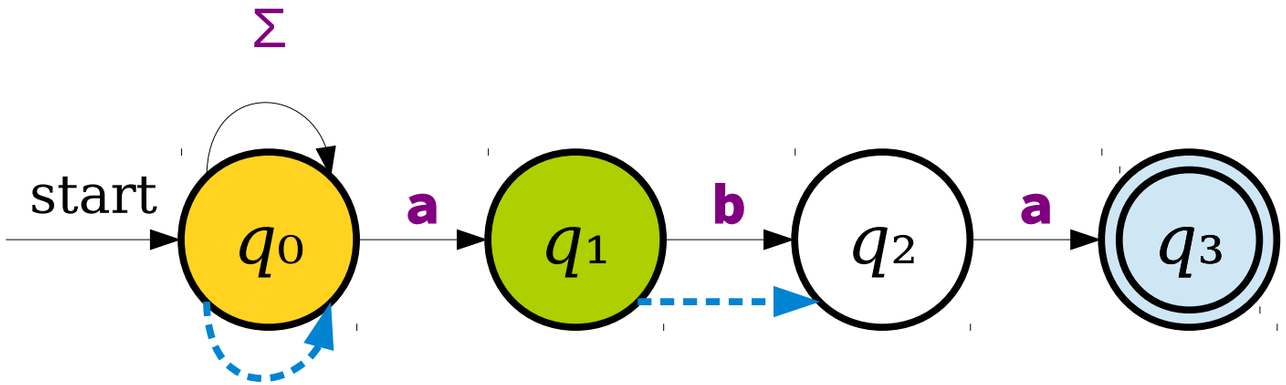


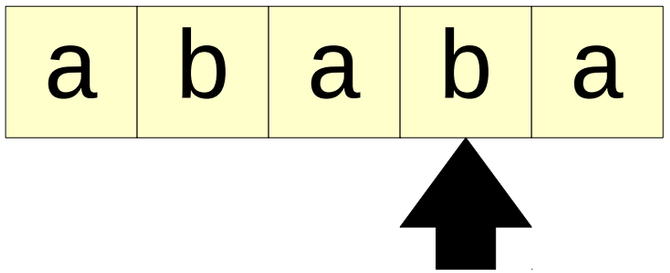
Massive Parallelism



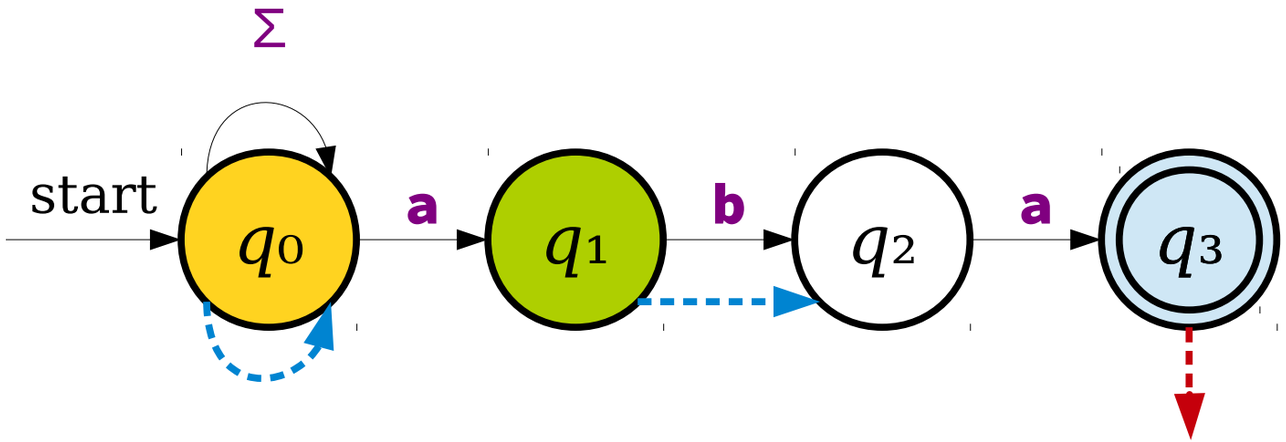


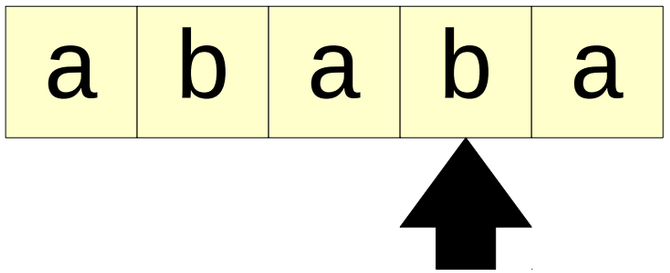
Massive Parallelism



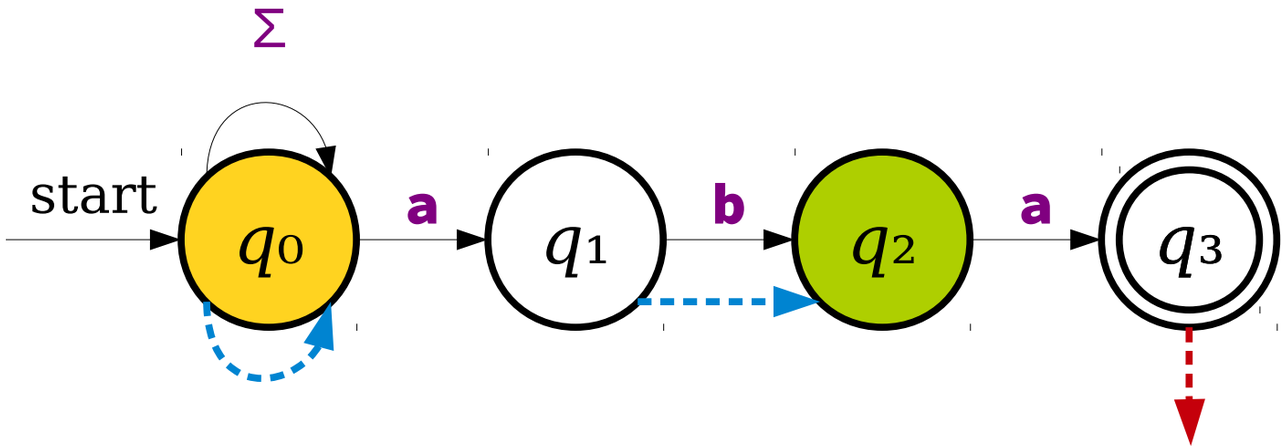


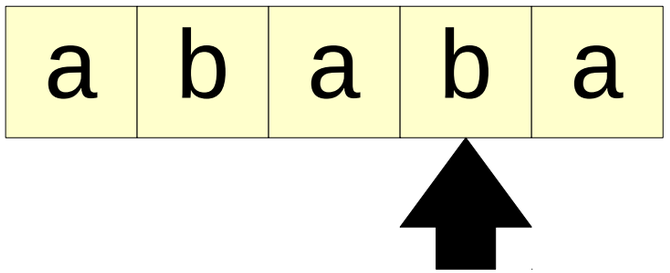
Massive Parallelism



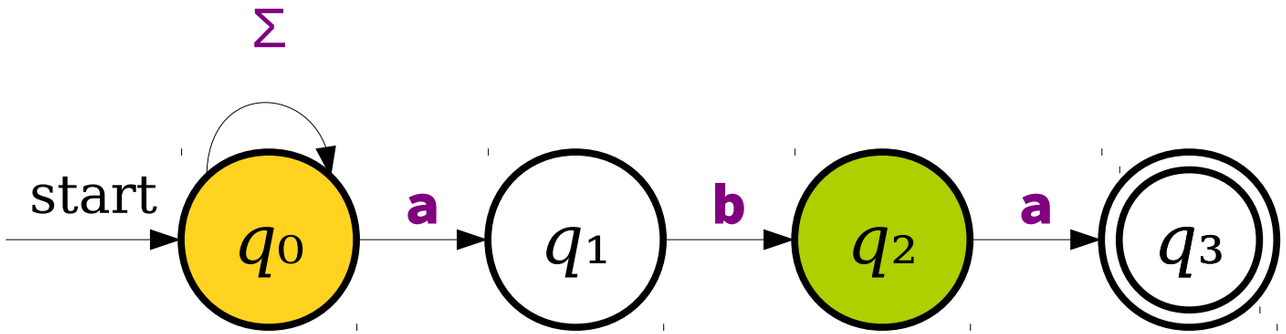


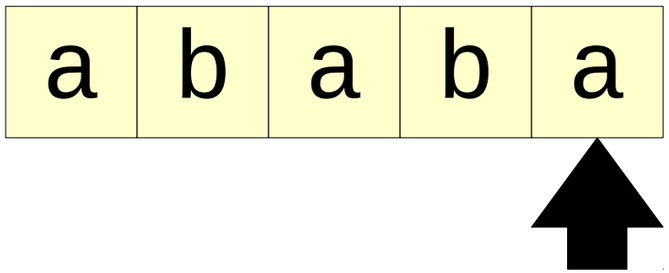
Massive Parallelism



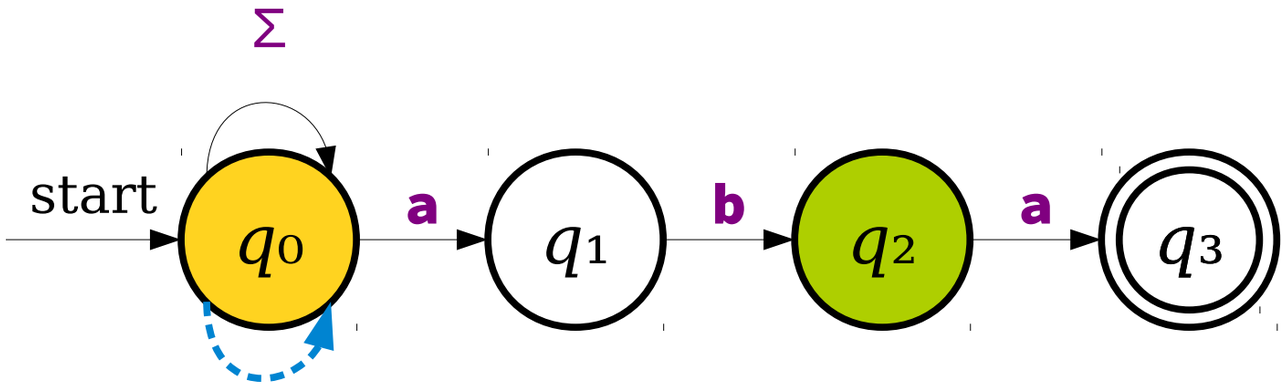


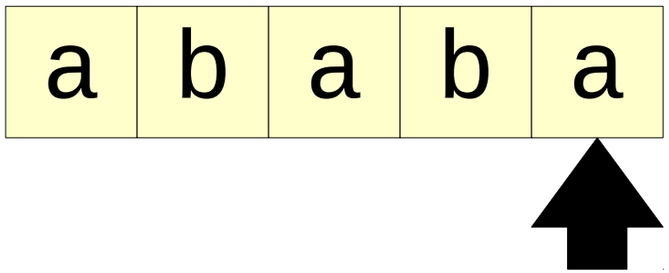
Massive Parallelism



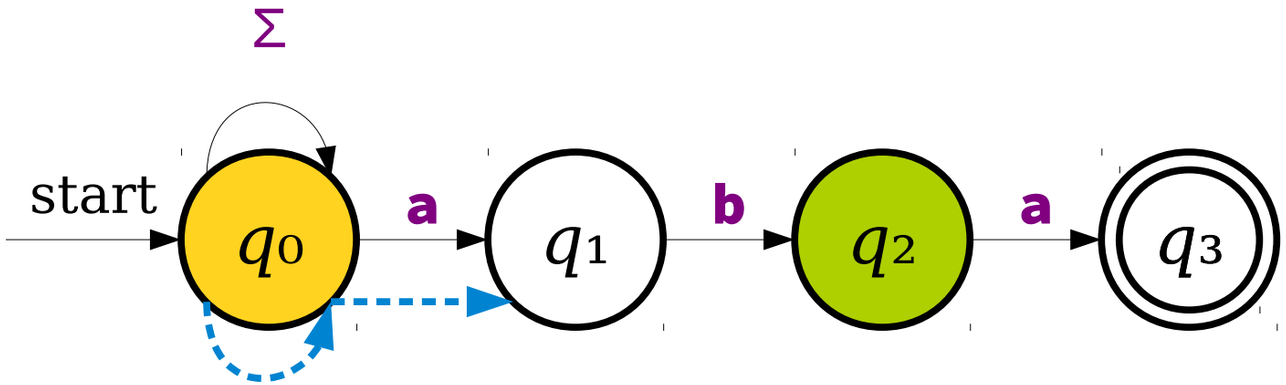


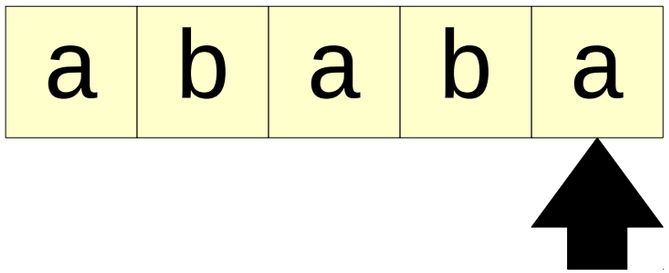
Massive Parallelism



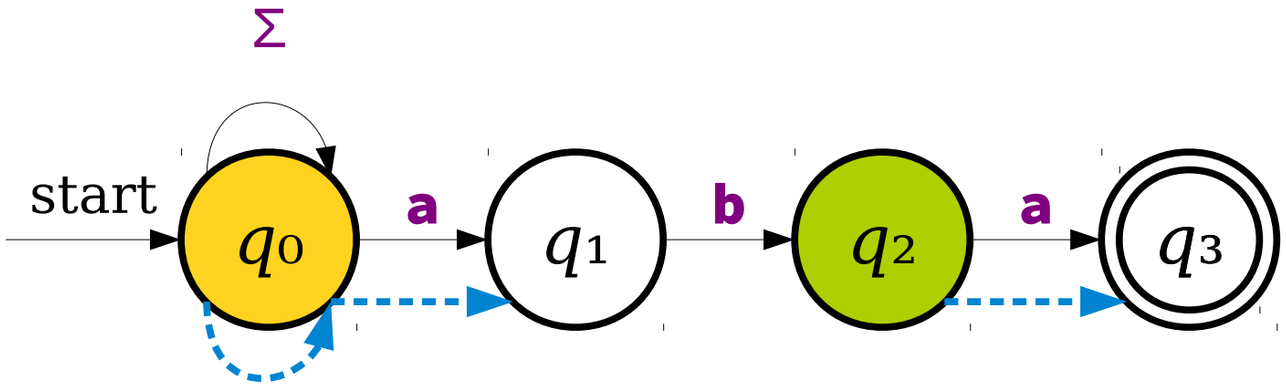


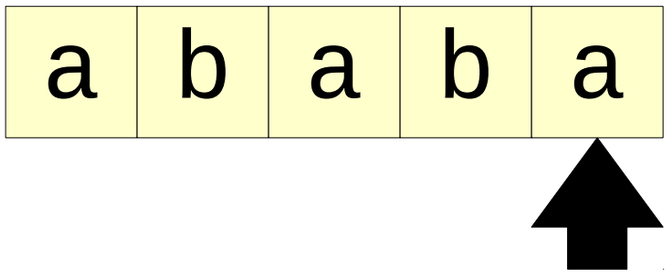
Massive Parallelism



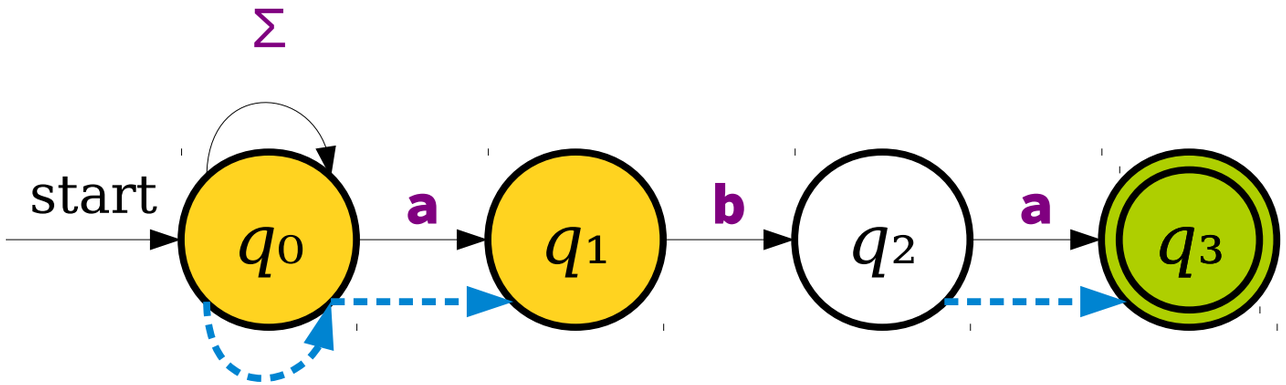


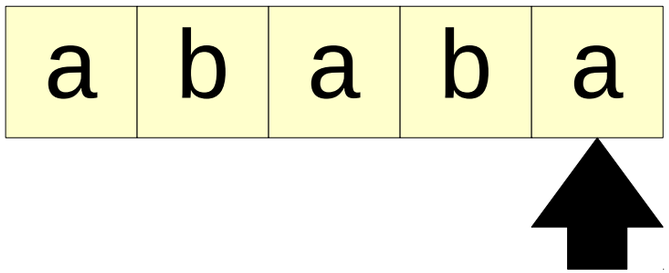
Massive Parallelism



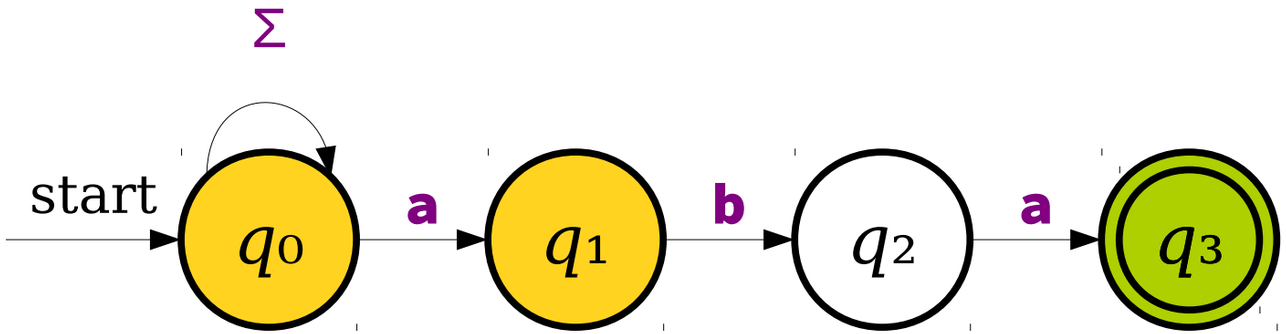


Massive Parallelism



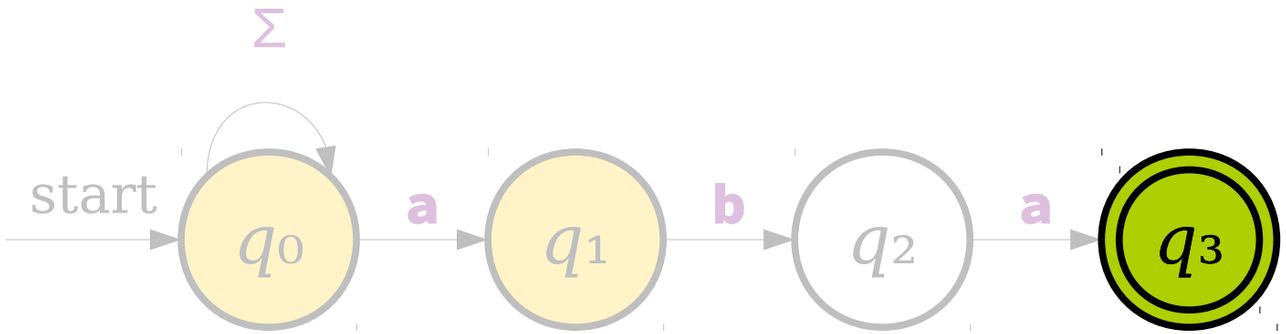


Massive Parallelism



a	b	a	b	a

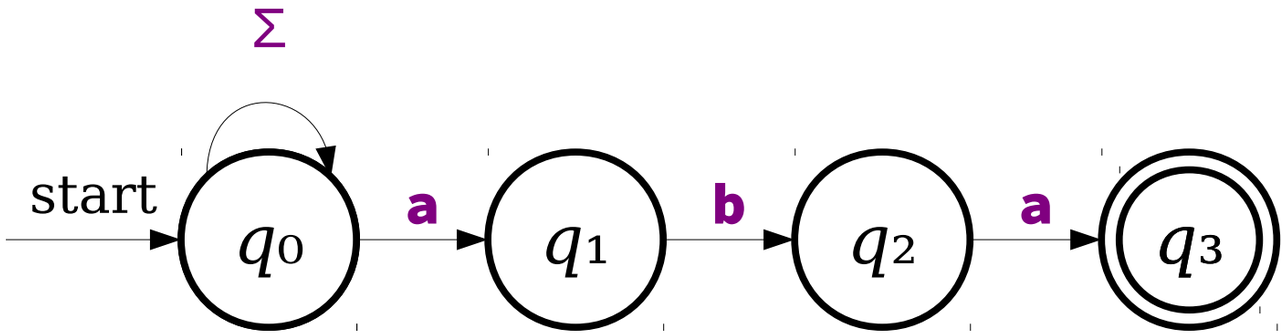
Massive Parallelism



The set of states we are in after processing the string contains at least one accepting state, there exists some path that gets us to an accepting state.

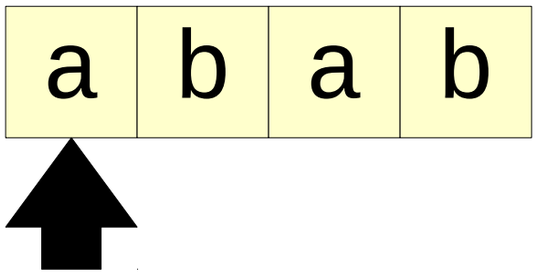
a	b	a	b	a

Massive Parallelism



If the set of states we are in after processing the string does not contain ANY accepting state, there is no path that gets us to an accepting state, so we reject the string.

(not going to go through it, but the string **abab** *is one such string for this NFA)*



Designing NFAs

- ***Embrace the nondeterminism!***
- Good model: ***Guess-and-check***:
  - Is there some fact about the later part of the string that you'd really like to know early on?
    - Have the machine *nondeterministically guess* that information!
  - Then, have the machine *deterministically check* that the guess was correct.

Guess-and-Check

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

Guess-and-Check

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

Ask yourself these design questions:

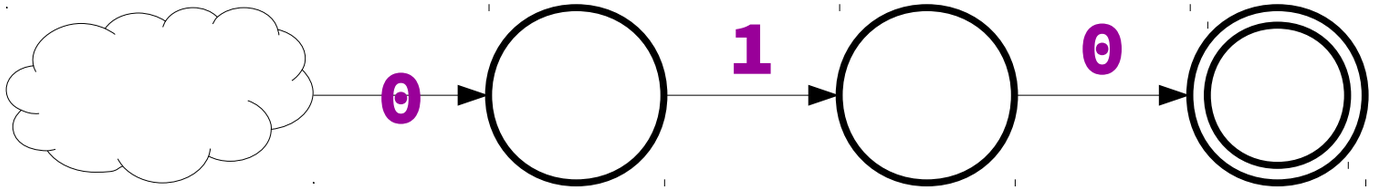
Would it be really easy to design an NFA to detect the substring 010 at the end, if you knew that's what you were looking for, and when you'd reached the near-end?

Would it be really easy to design an NFA to detect the substring 101, if you knew that's what you were looking for, and when you'd reached the near-end?

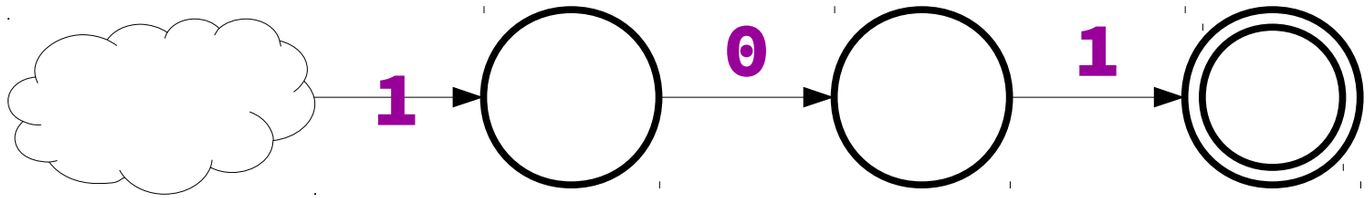
Would it be really convenient if you could just ***magically guess*** that?

Guess-and-Check

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



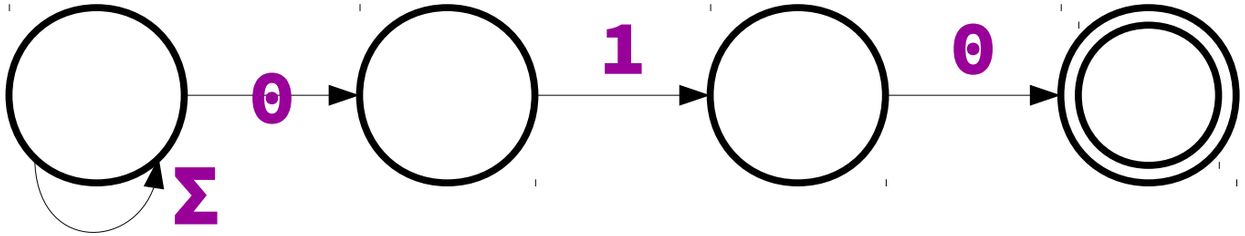
Detemrinistic machine for “substring 010 at the end”



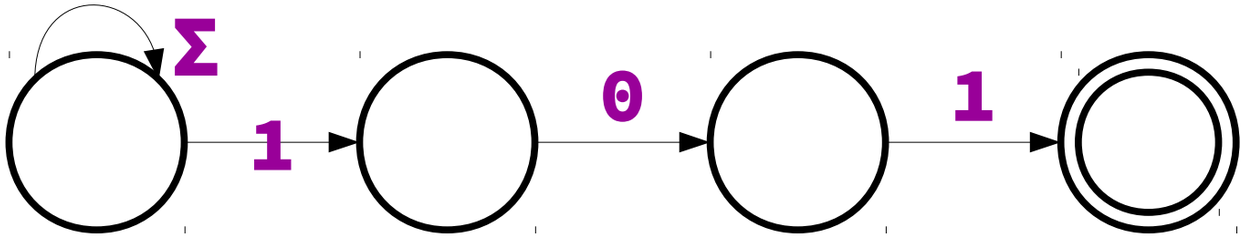
Deterministic machine for “substring 101 at the end”

Guess-and-Check

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



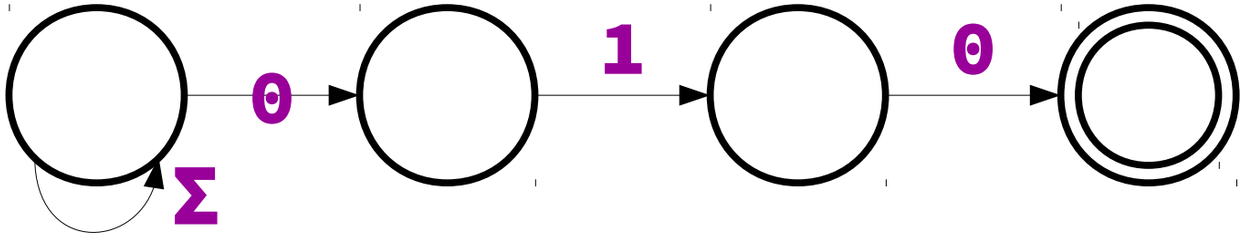
machine for “substring 010 at the end”



machine for “substring 101 at the end”

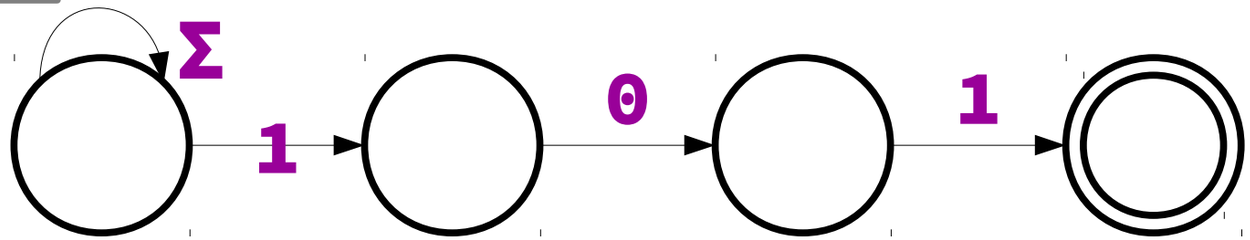
Guess-and-Check

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



machine for "substring 010 at the end"

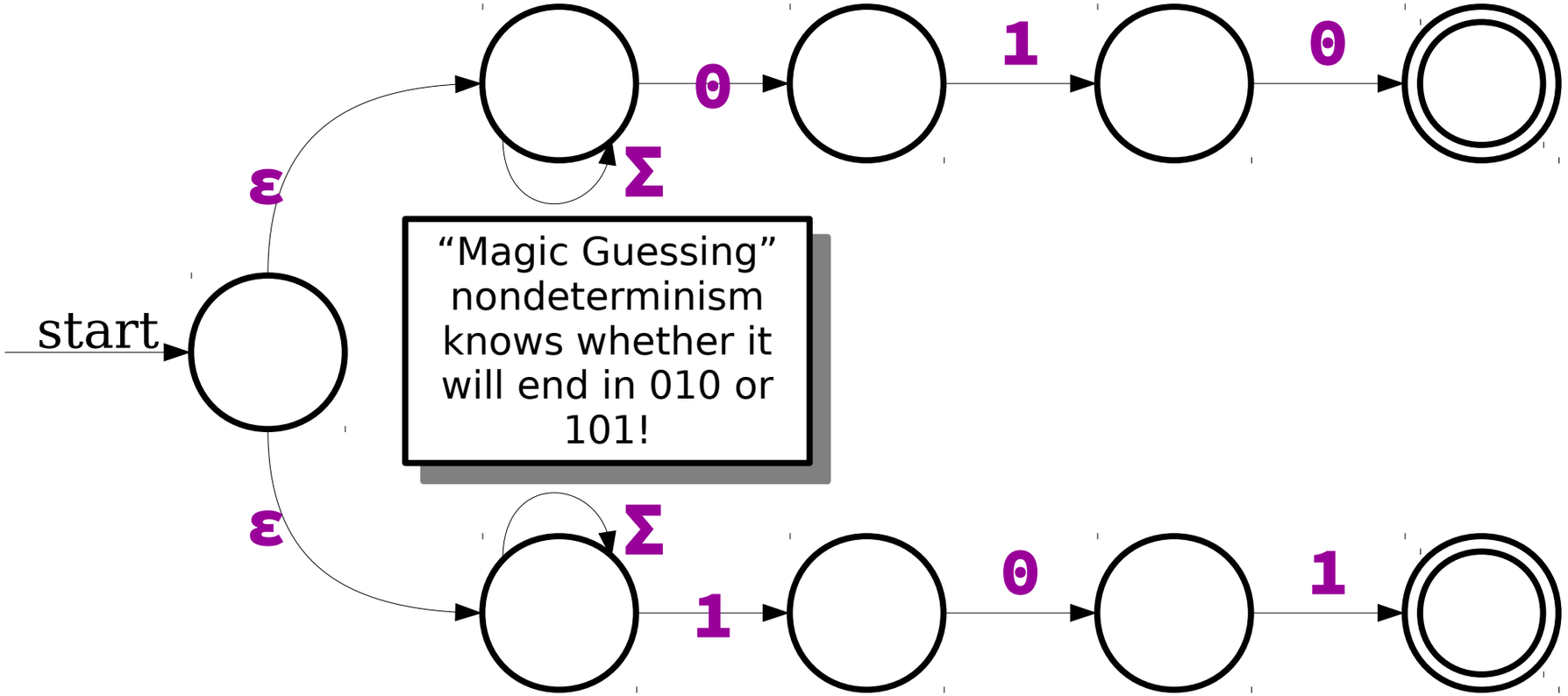
"Magic Guessing" nondeterminism knows when we are nearing end of string!



machine for "substring 101 at the end"

Guess-and-Check

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



Guess-and-Check

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

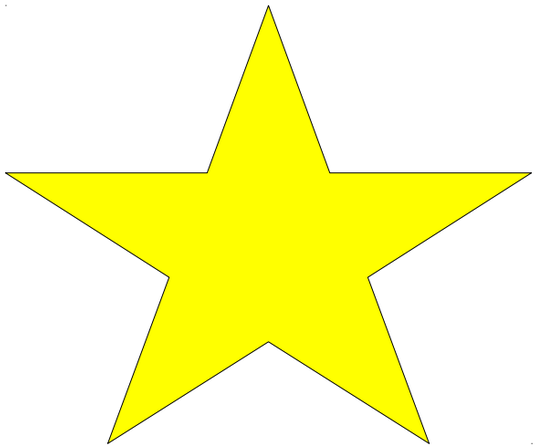
$= L_1 \cup L_2$ where:

$L_1 = \{ w \in \{\texttt{0}, \texttt{1}\}^* \mid w \text{ ends in } \texttt{010} \}$

$L_2 = \{ w \in \{\texttt{0}, \texttt{1}\}^* \mid w \text{ ends in } \texttt{101} \}$

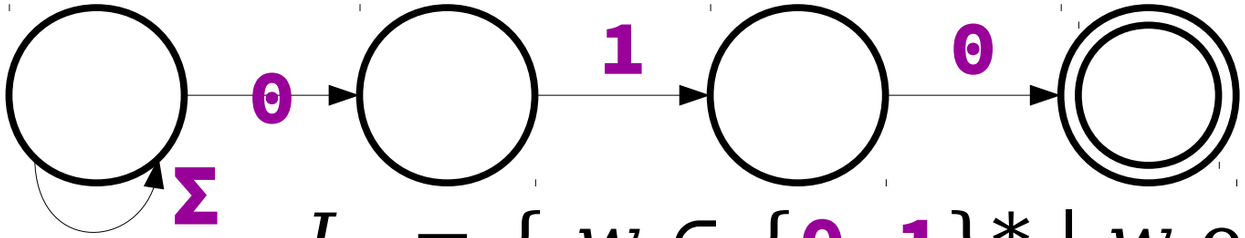
NFA Design Hack!

If you can write the language as the union of two or more very simple languages:
(1) make simple DFA/NFAs for those simple languages
(2) a single start state dispatches to the simple DFA/NFAs using epsilon transitions



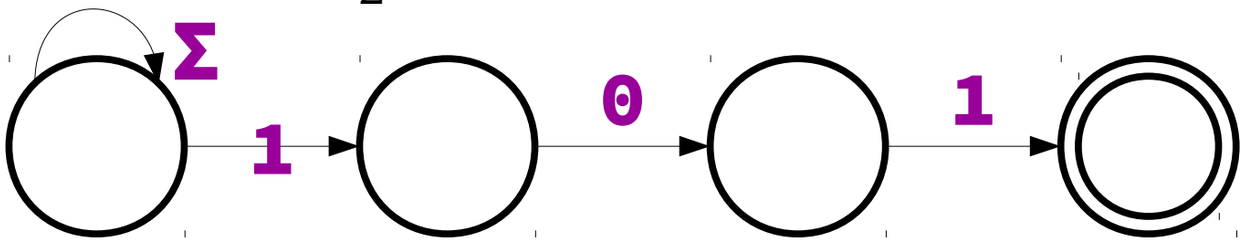
Guess-and-Check

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



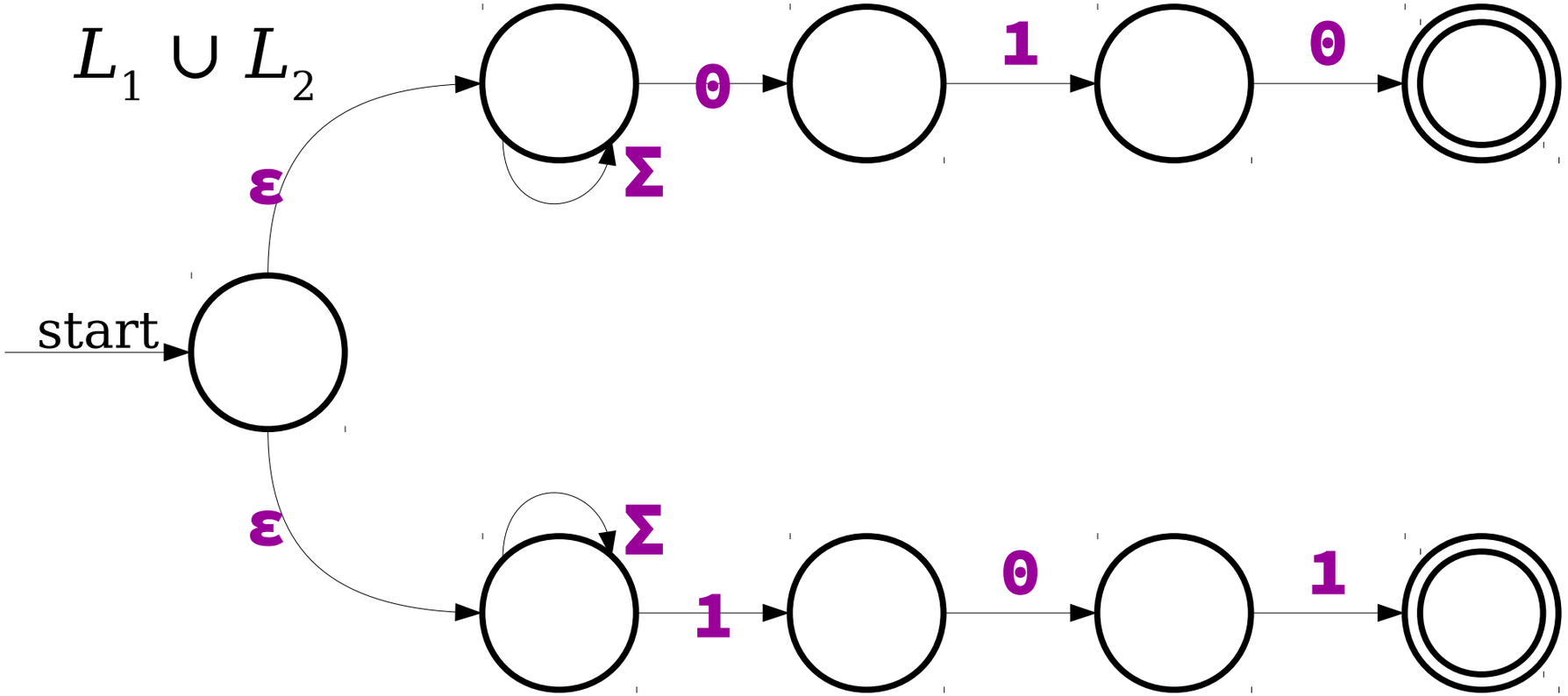
$L_1 = \{ w \in \{\mathbf{0}, \mathbf{1}\}^* \mid w \text{ ends in } \mathbf{010} \}$

$L_2 = \{ w \in \{\mathbf{0}, \mathbf{1}\}^* \mid w \text{ ends in } \mathbf{101} \}$



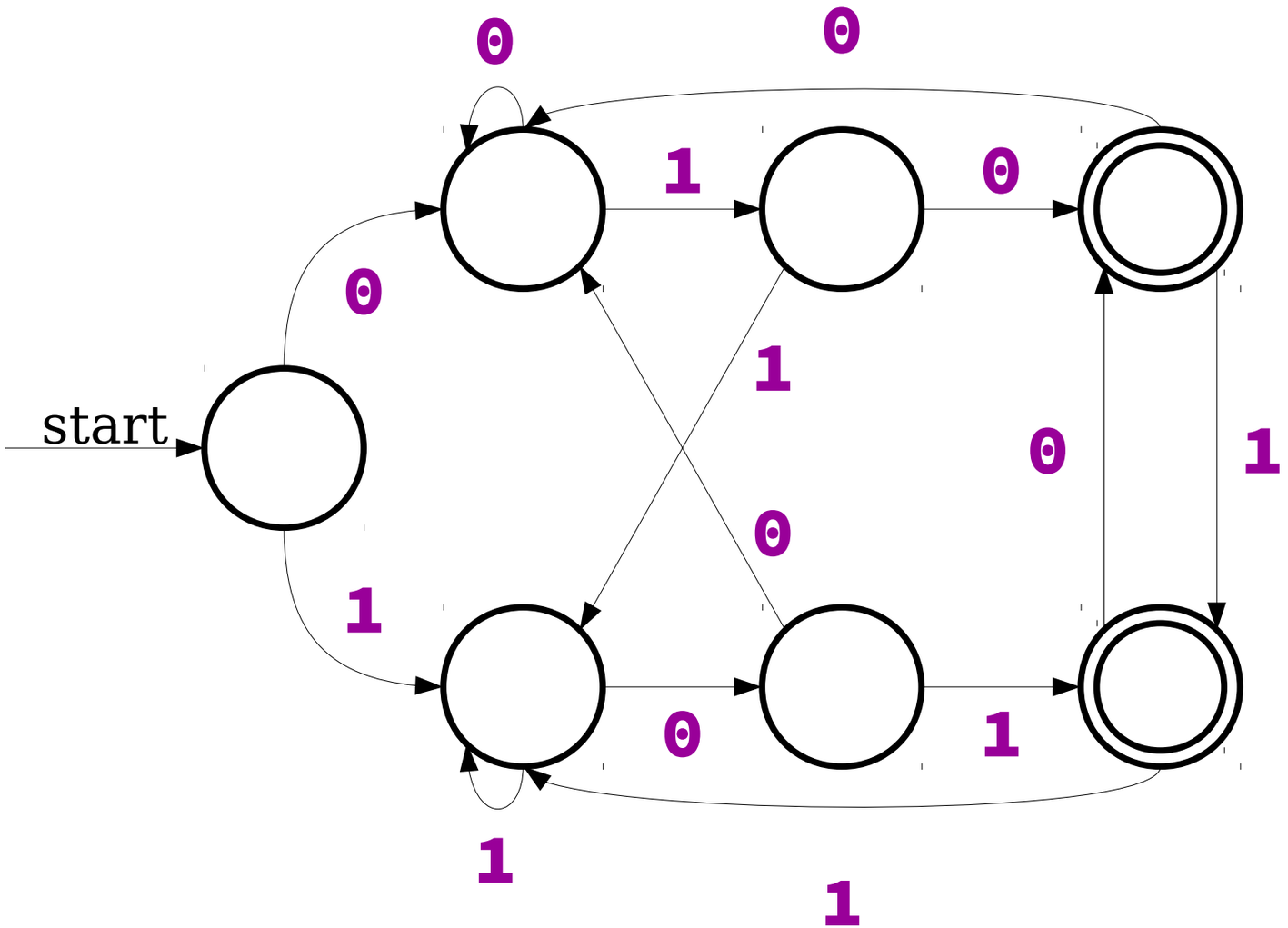
Guess-and-Check

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



Guess-and-Check

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



Guess-and-Check

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

Guess-and-Check

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

Ask yourself these design questions:

Would it be really easy to design an NFA to detect the string...

...has no a's in it, if you knew that's what you were looking for?

...has no b's in it, if you knew that's what you were looking for?

..has no c's in it, if you knew that's what you were looking for?

Would it be really convenient if you could just ***magically guess*** which letter is the missing one this time?

Guess-and-Check

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

$L_1 = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid \mathbf{a} \text{ is not in } w \}$

$L_2 = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid \mathbf{b} \text{ is not in } w \}$

$L_3 = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid \mathbf{c} \text{ is not in } w \}$

$$L = L_1 \cup L_2 \cup L_3$$

Ask yourself these design questions:

Would it be really easy to design an NFA to detect the string...

...has no a's in it, if you knew that's what you were looking for?

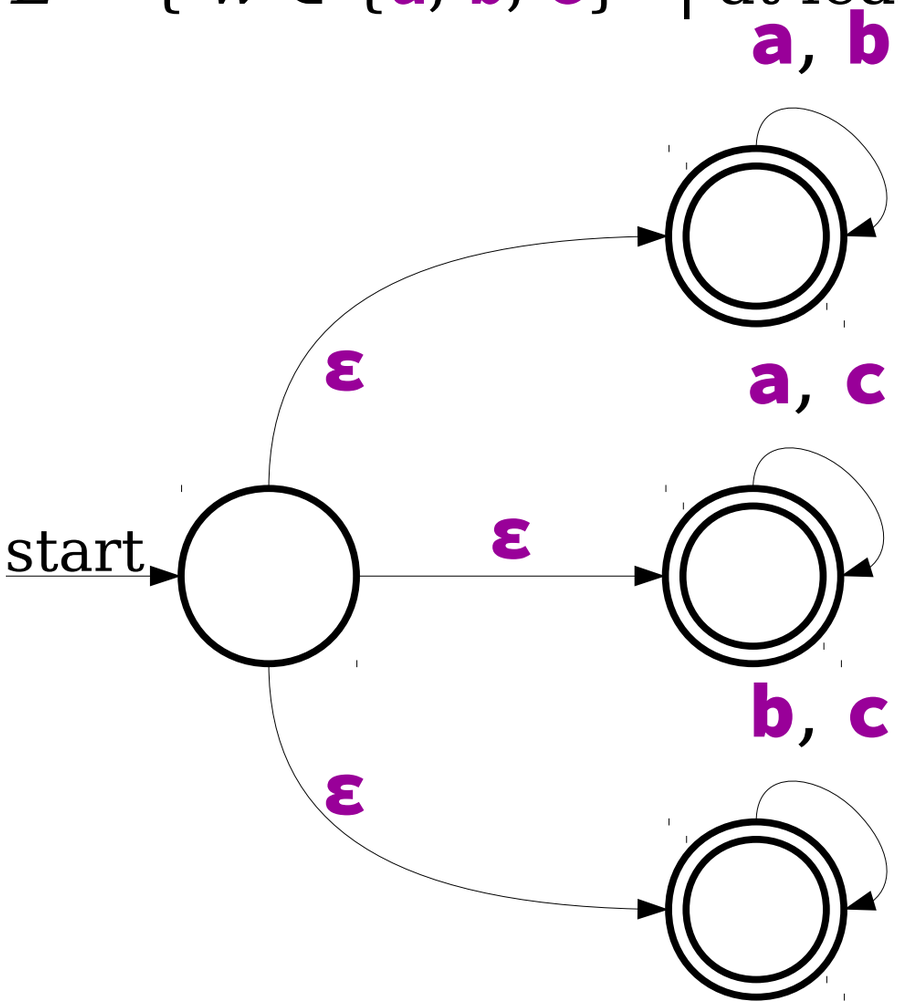
...has no b's in it, if you knew that's what you were looking for?

..has no c's in it, if you knew that's what you were looking for?

Would it be really convenient if you could just ***magically guess*** which letter is the missing one this time?

Guess-and-Check

$L = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid$ at least one of **a**, **b**, or **c** is not in $w \} = L_1 \cup L_2 \cup L_3$



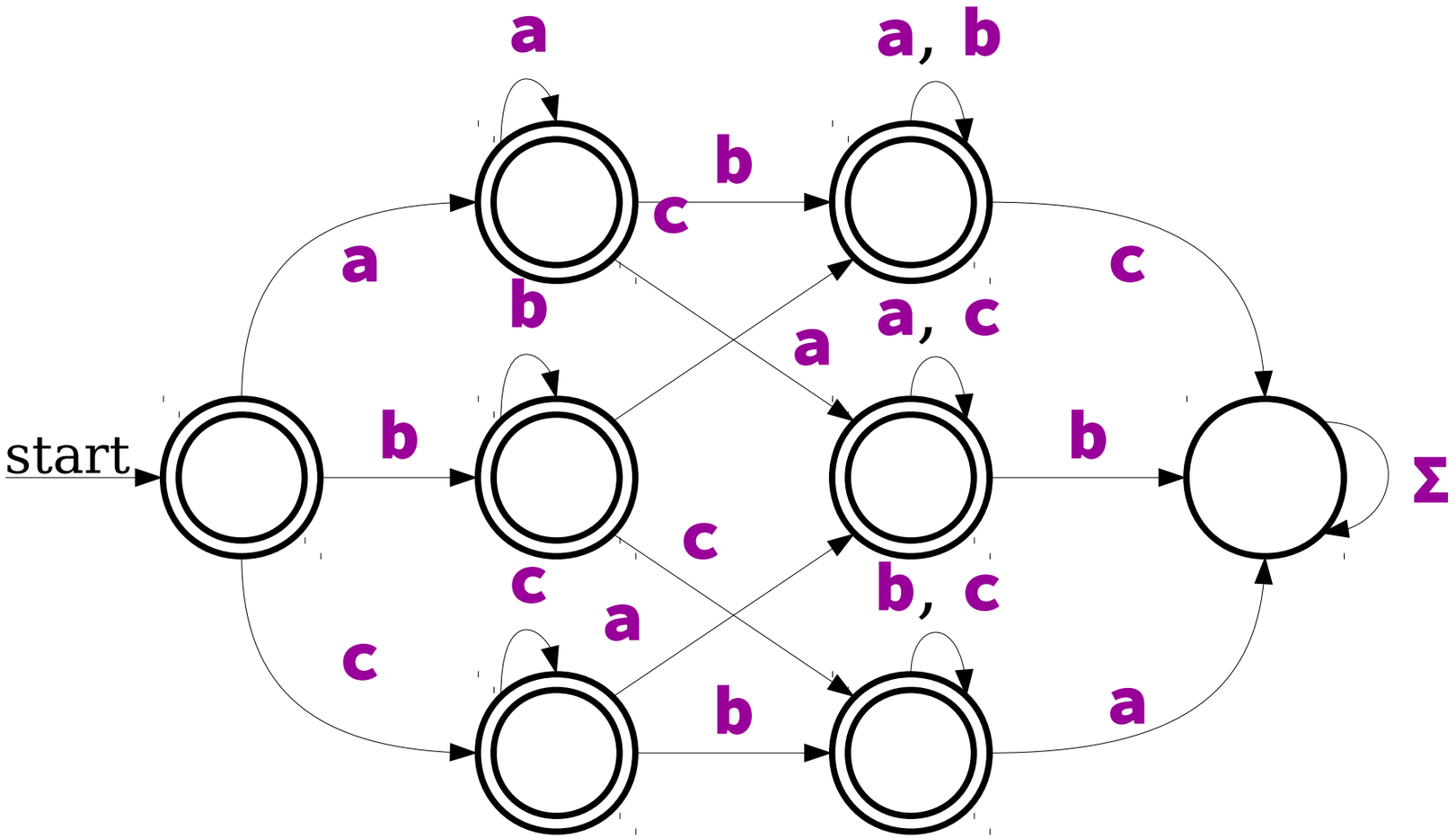
$L_3 = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid \mathbf{c}$ is not in $w \}$

$L_2 = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid \mathbf{b}$ is not in $w \}$

$L_1 = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid \mathbf{a}$ is not in $w \}$

Guess-and-Check

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



Just how powerful are NFAs?

Next Time

- ***The Powerset Construction***
  - So beautiful. So elegant. So cool!
- ***More Closure Properties***
  - Other set-theoretic operations.
- ***Language Transformations***
  - What's the deal with the notation Σ*?