Finite Automata

Part Two

Recap from Last Time

Strings

- An ***alphabet*** is a finite, nonempty set of symbols called ***characters***.
  - Typically, we use the symbol **Σ** to refer to an alphabet.
- A ***string over an alphabet Σ*** is a finite sequence of characters drawn from Σ.
- Example: If Σ = {**a**, **b**}, here are some valid strings over Σ:

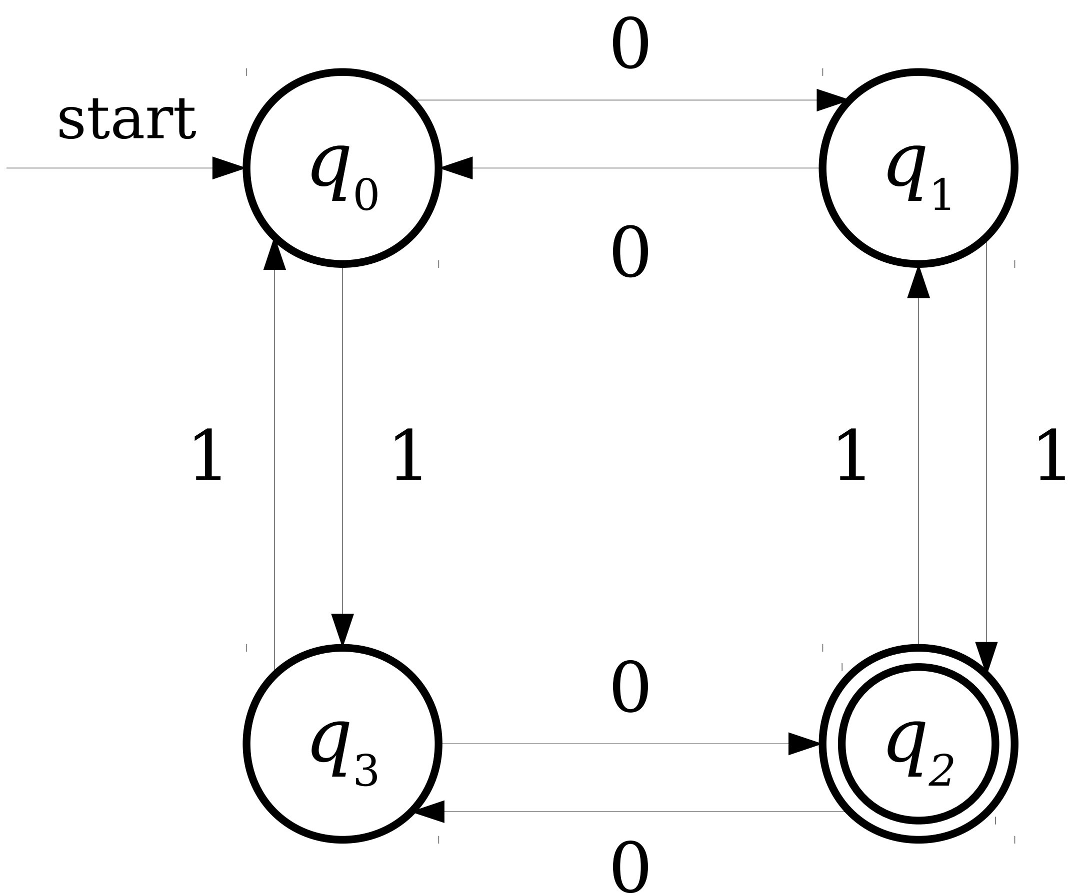
  a   **aabaaabbabaaabaaaabbb**   **abbababba**

- The ***empty string*** has no characters and is denoted **ε**.
- Calling attention to an earlier point: since all strings are finite sequences of characters from Σ, you cannot have a string of infinite length.

Languages

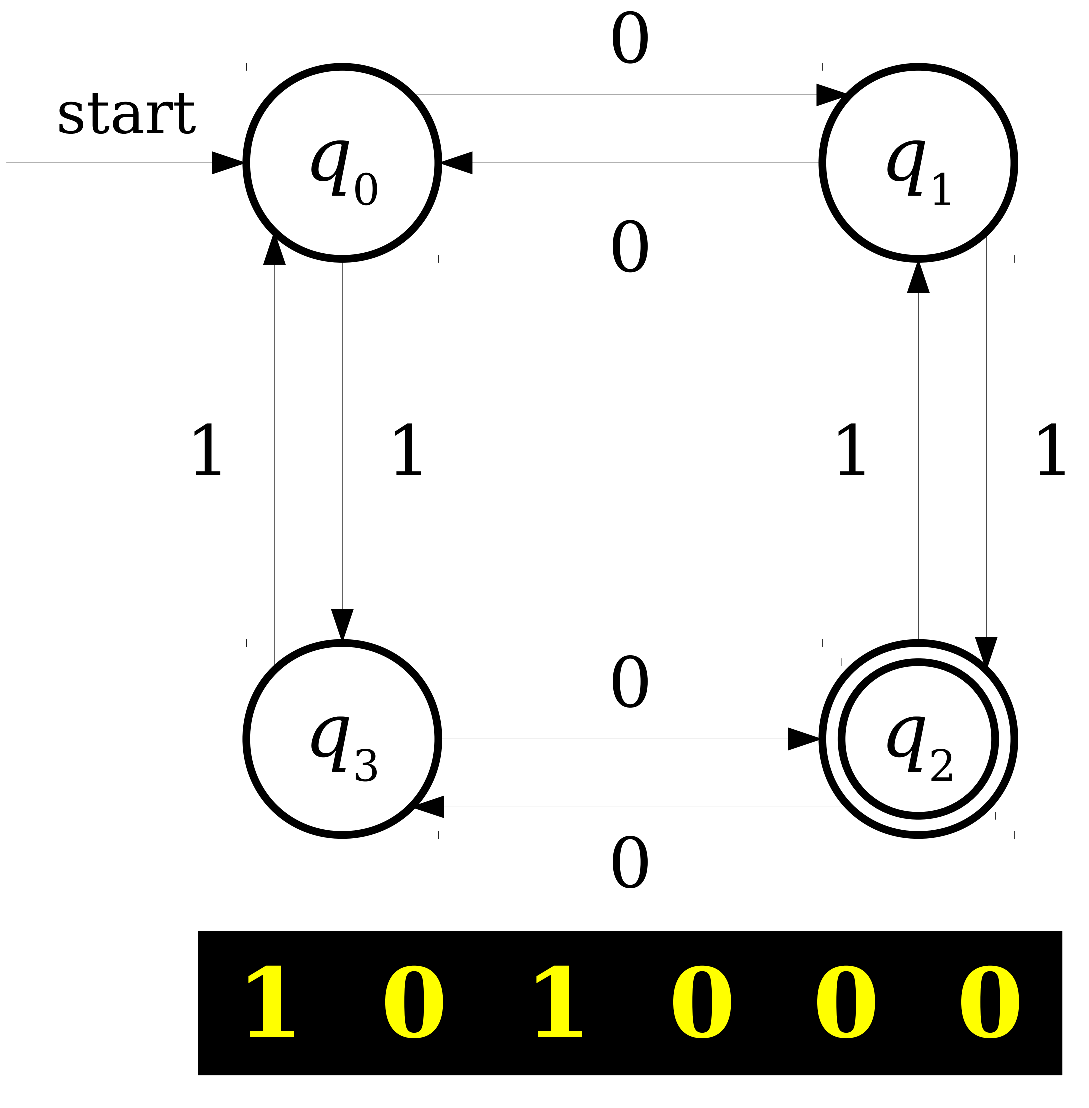
- A ***formal language*** is a set of strings.
- We say that L is a ***language over Σ*** if it is a set of strings over Σ.
- Example: The language of palindromes over Σ = {**a**, **b**, **c**} is the set
  - {ε, **a**, **b**, **c**, **aa**, **bb**, **cc**, **aaa**, **aba**, **aca**, **bab**, … }
- The set of all strings composed from letters in Σ is denoted **Σ***.
- Formally, we say that L is a language over Σ if $L \subseteq \Sigma^*$.

A Simple Finite Automaton



0 1 0 1 1 0

A Simple Finite Automaton



The ***language of an automaton*** is the set of strings that it accepts.

If D is an automaton, we denote the language of D as $\mathscr{L}(\boldsymbol{D})$.

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

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, Informally

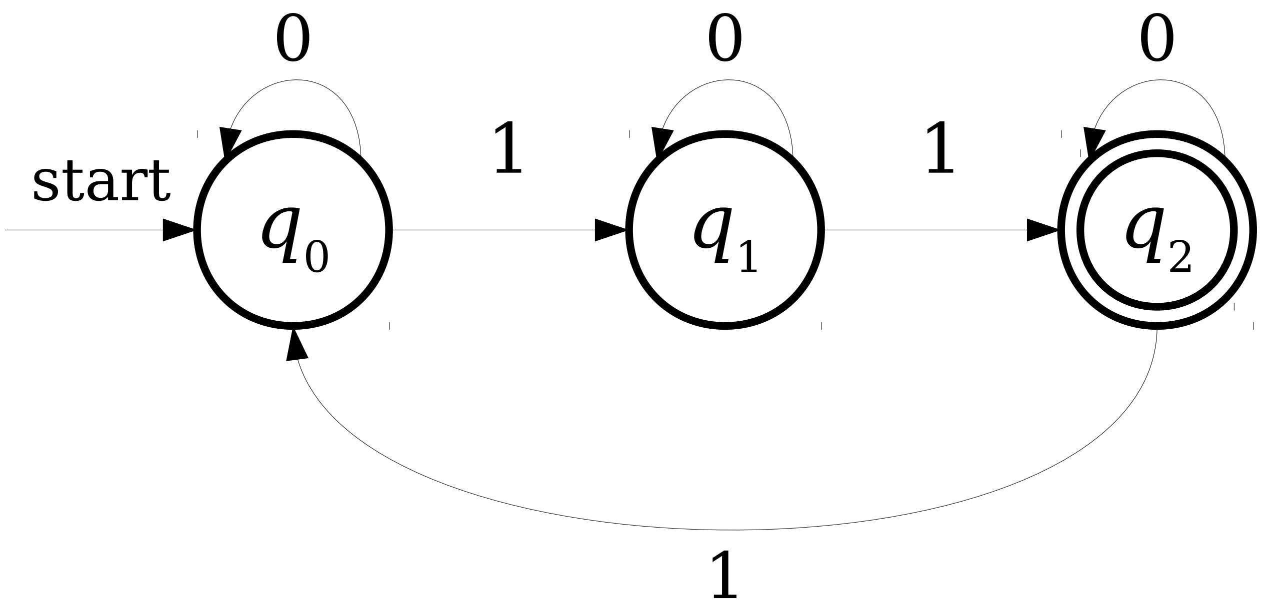
- A DFA is defined relative to some alphabet Σ.
- For each state in the DFA, there must be ***exactly one*** transition defined for each symbol in Σ.
  - This is the "deterministic" part of DFA.
- There is a unique start state.
- There are zero or more accepting states.

Designing DFAs

- At each point in its execution, the DFA can only remember what state it is in.
- **DFA Design Tip:** Build each state to correspond to some piece of information you need to remember.
  - Each state acts as a “memento” of what you're supposed to do next.
  - Only finitely many different states ≈ only finitely many different things the machine can remember.

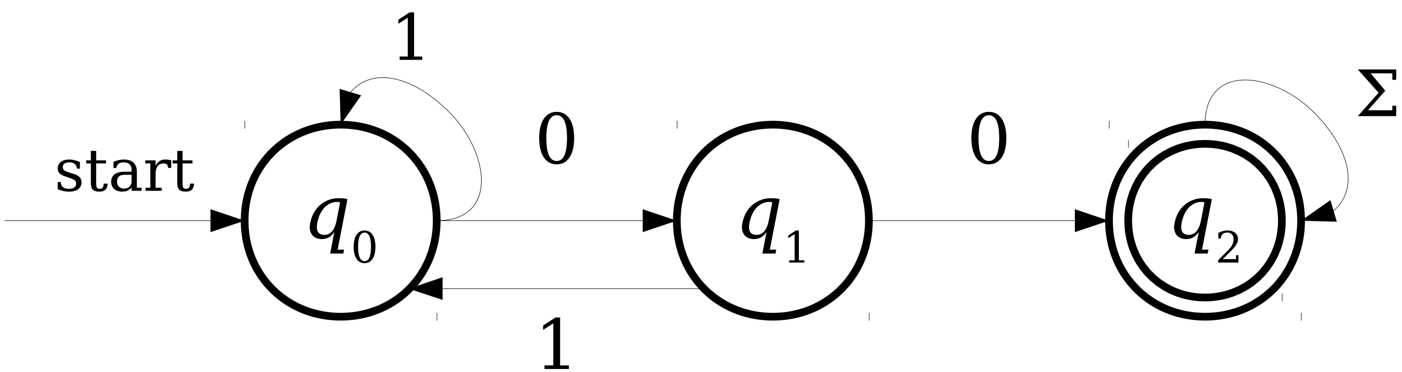
Recognizing Languages with DFAs

$L = \{ w \in \{\mathtt{0}, \mathtt{1}\}^* \mid$ the number of $\mathtt{1}$'s in w is congruent to two modulo three $\}$



Recognizing Languages with DFAs

$L = \{\ w \in \{\mathtt{0}, \mathtt{1}\}^* \mid w \text{ contains } \mathtt{00} \text{ as a substring } \}$



More Elaborate DFAs

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

Suppose the alphabet is

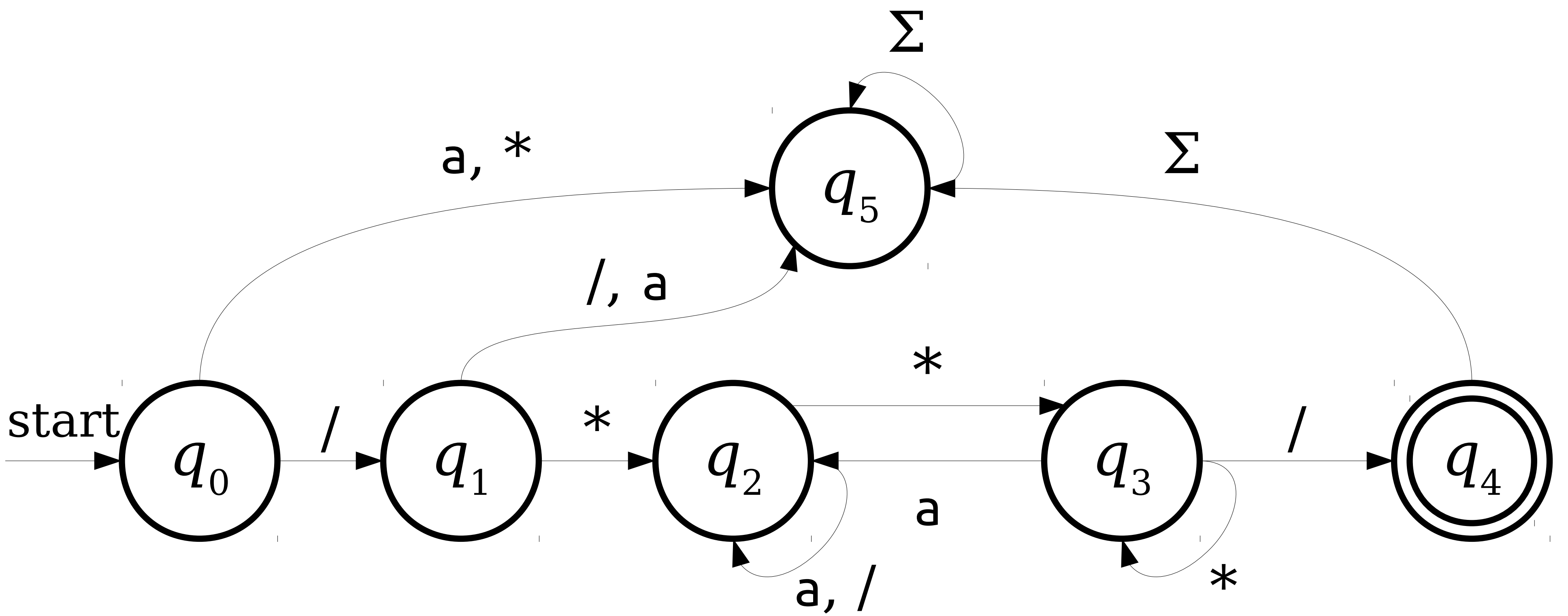
$\Sigma = \{ \mathtt{a}, \mathtt{*}, \mathtt{/} \}$

Try designing a DFA for comments! Some test cases:

ACCEPTED	REJECTED
`/*a*/`	`/**`
`/**/`	`/**/a/*aa*/`
`/***/`	`aaa/**/`
`/*aaa*aaa*/`	`/*/`
`/*a/a*/`	`/**a/`

More Elaborate DFAs

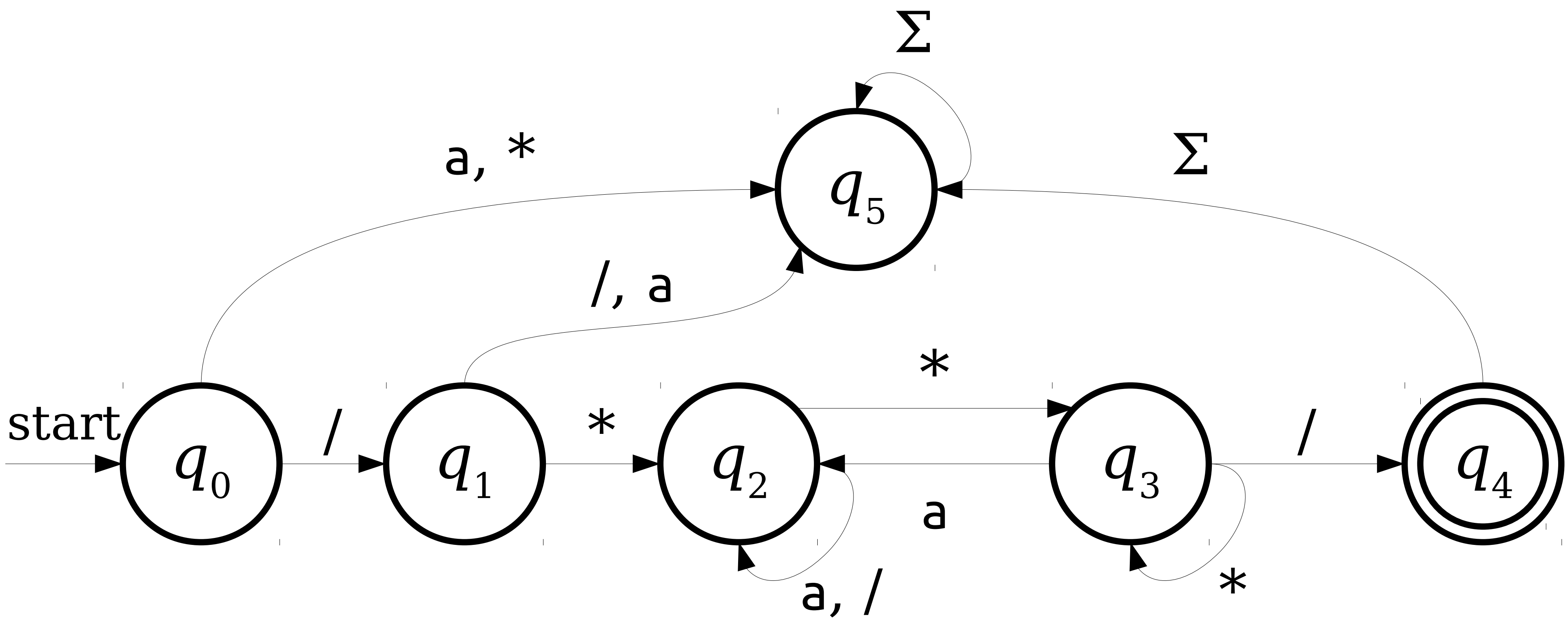
$L = \{ w \in \{$a, *, /$\}^* \mid w$ represents a C-style comment $\}$



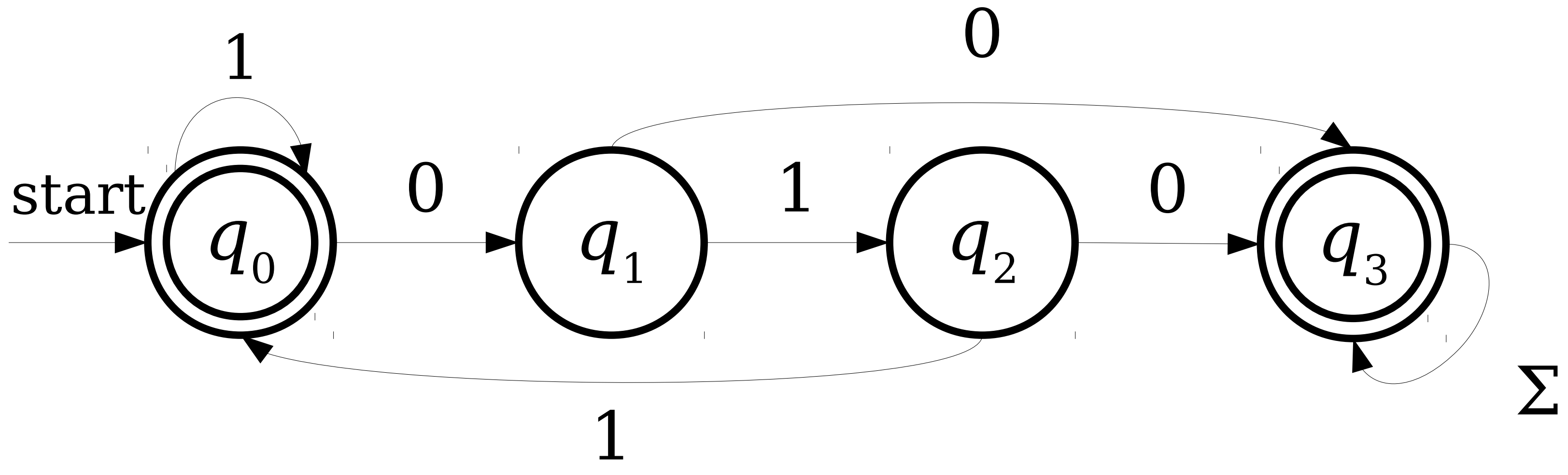
New Stuff!

More Elaborate DFAs

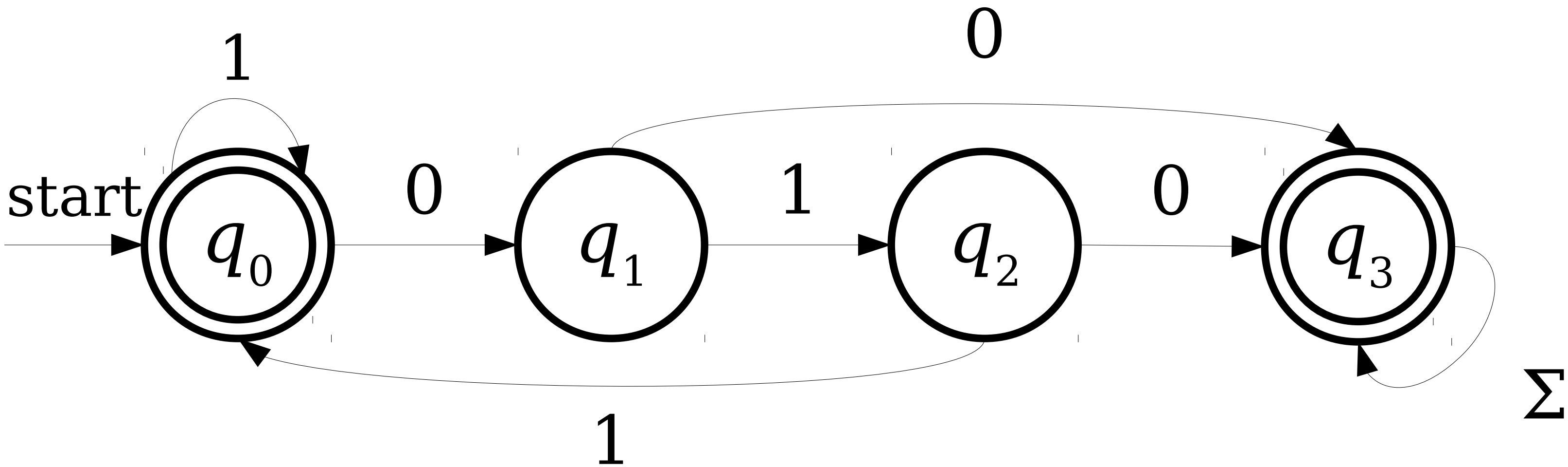
$L = \{ w \in \{\texttt{a}, \texttt{*}, \texttt{/}\}^* \mid w \text{ represents a C-style comment } \}$



Tabular DFAs

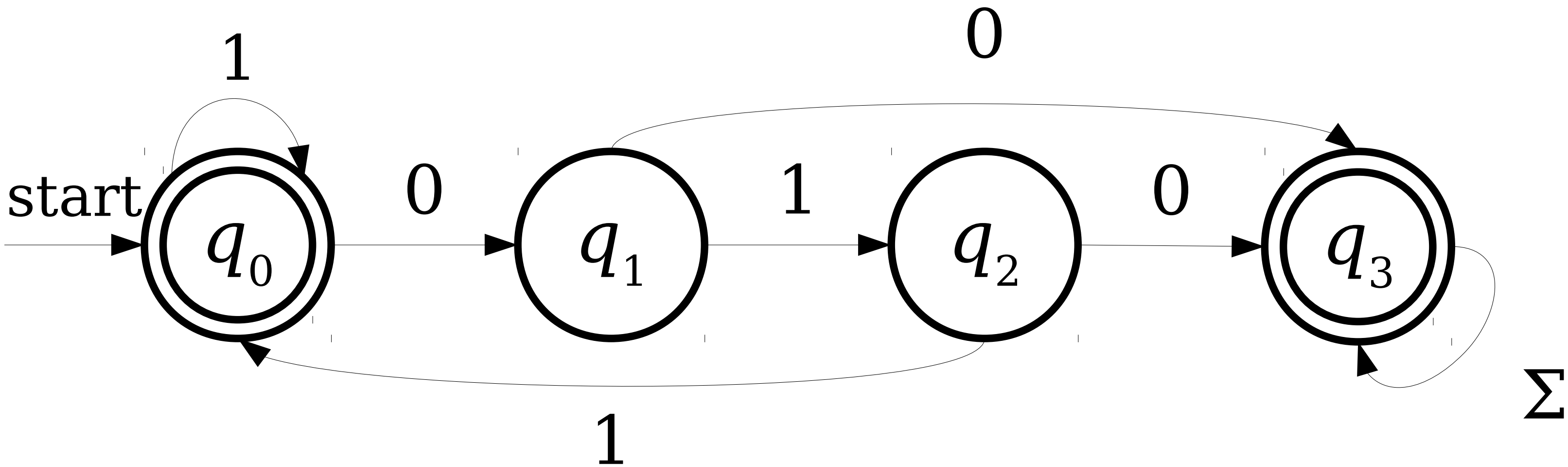


Tabular DFAs



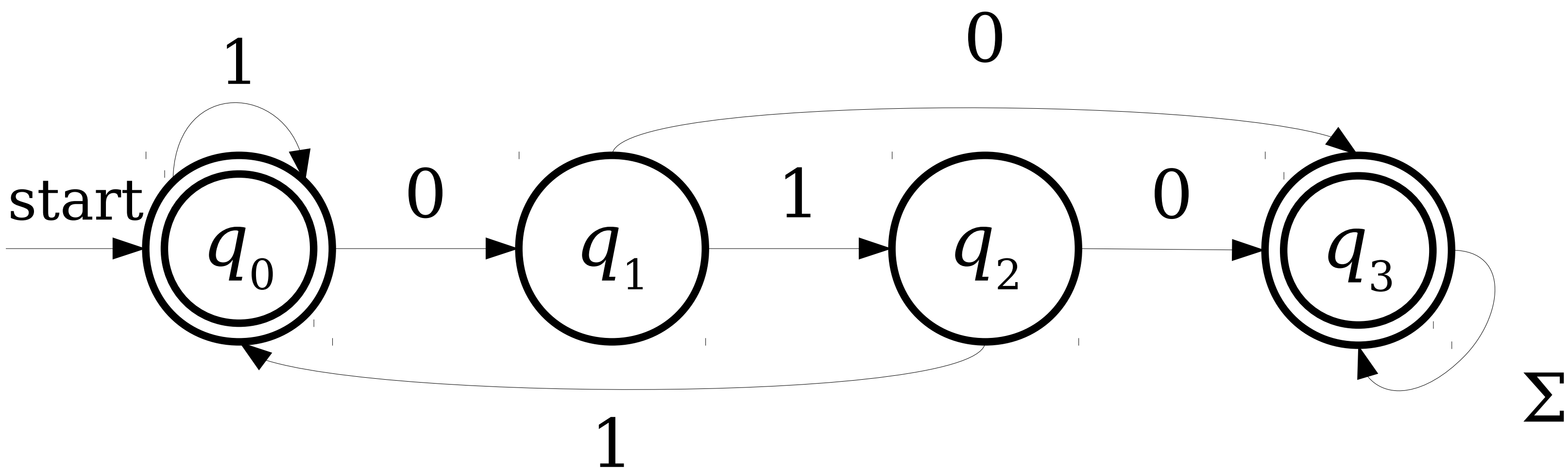
	0	1
q_0		
q_1		
q_2		
q_3		

Tabular DFAs



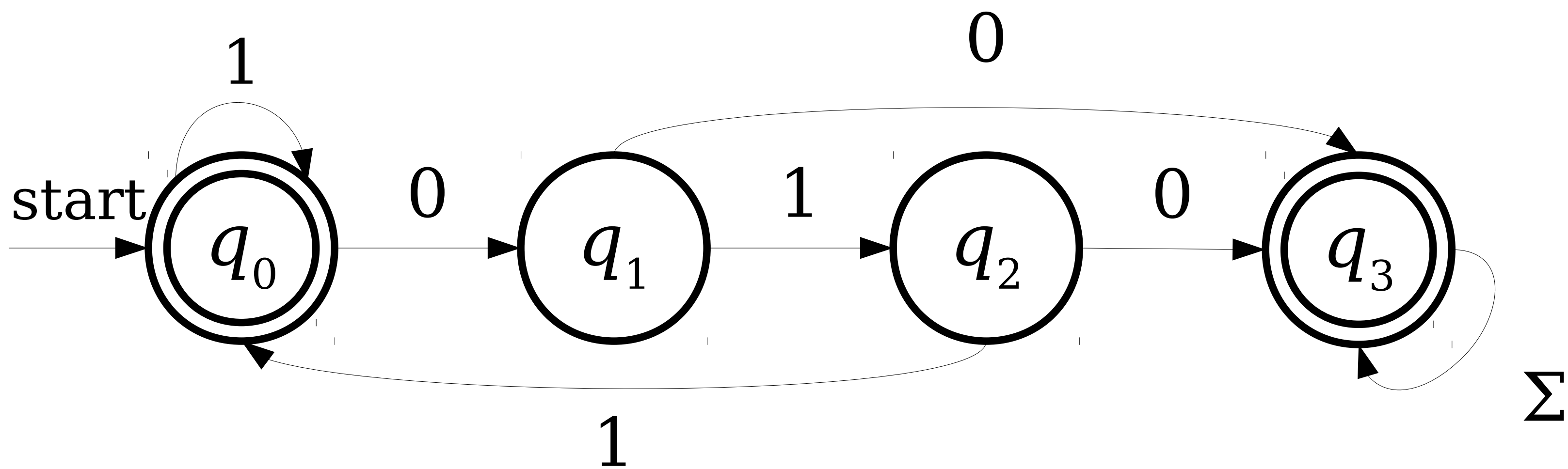
	0	1
q_0	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
q_3	q_3	q_3

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

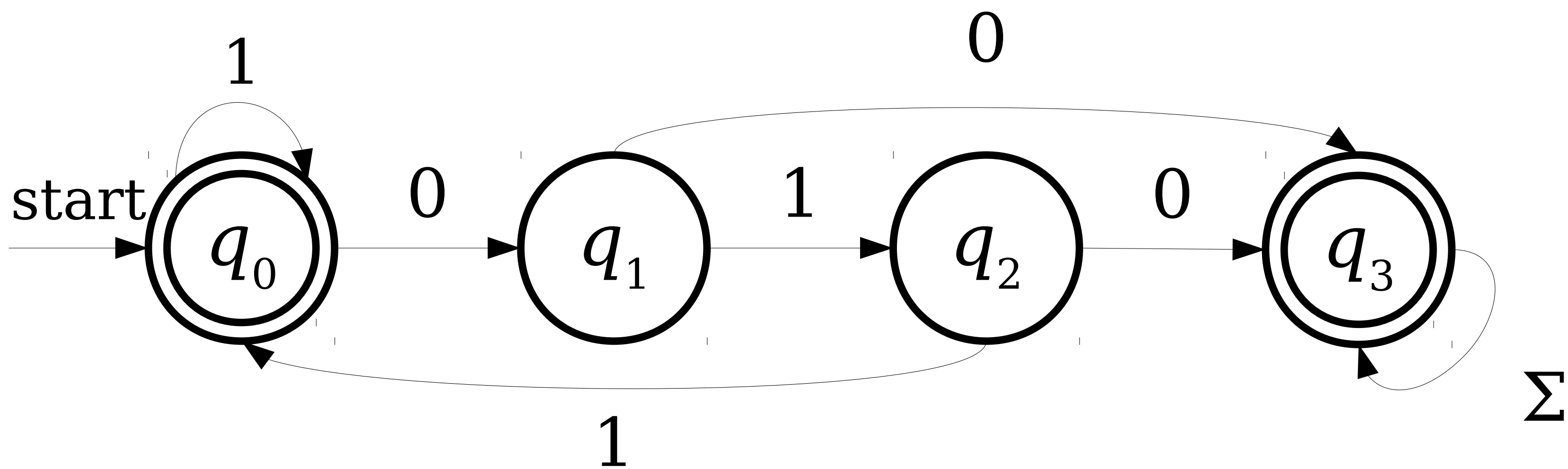
Tabular DFAs



	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



	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

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

Code? In a Theory Course?

```
int kTransitionTable[kNumStates][kNumSymbols] = {
     {0, 0, 1, 3, 7, 1, …},
      …
};
bool kAcceptTable[kNumStates] = {
    false,
    true,
    true,
    …
};
bool SimulateDFA(string input) {
    int state = 0;
    for (char ch: input)
        state = kTransitionTable[state][ch];
    return kAcceptTable[state];
}
```

The Regular Languages

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

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:

$$\overline{L} = \{ w \mid w \in \Sigma^* \land w \notin L \}$$

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:

$$\overline{L} = \{ w \mid w \in \Sigma^* \land w \notin L \}$$

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

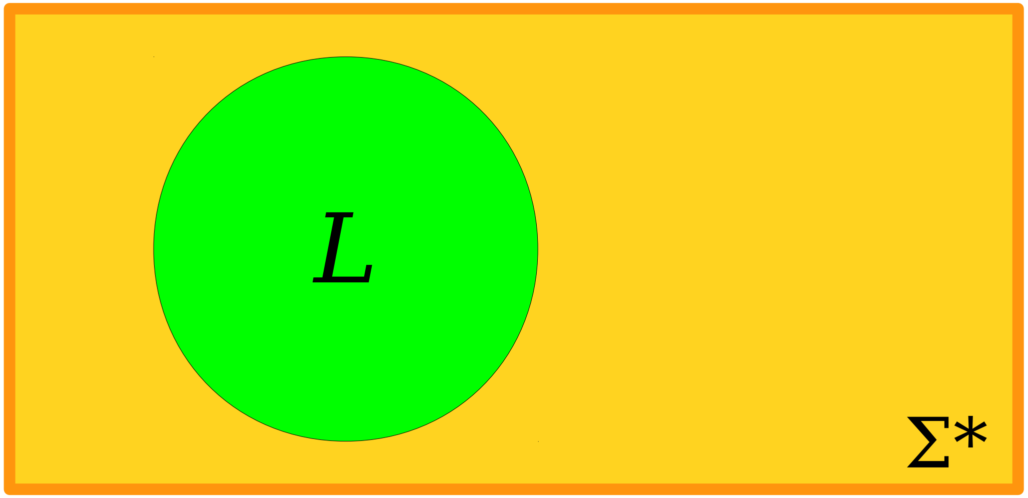
- Given a language $L \subseteq \Sigma^*$, the ***complement*** of that language (denoted $\boldsymbol{\overline{L}}$) is the language of all strings in Σ^* that aren't in L.
- Formally:

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

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:

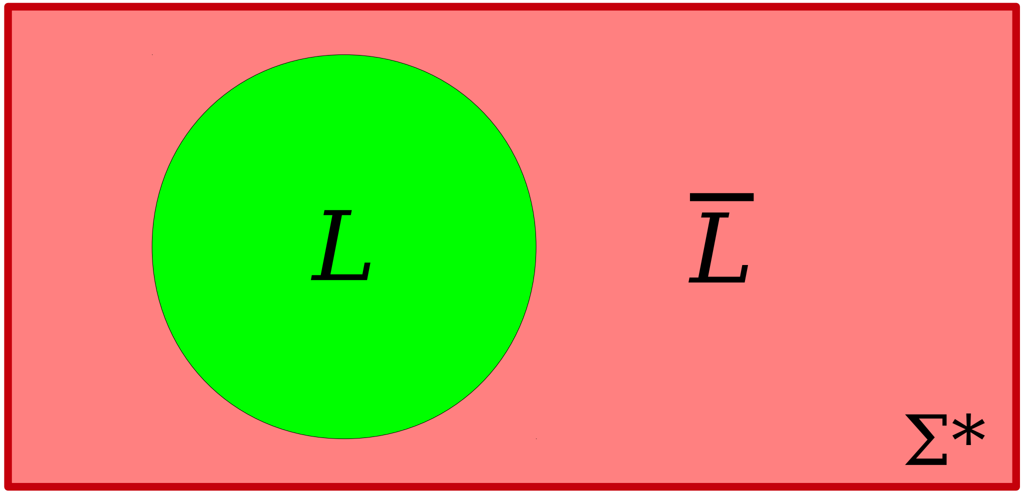
$$\overline{L} = \Sigma^* - L$$



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:

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

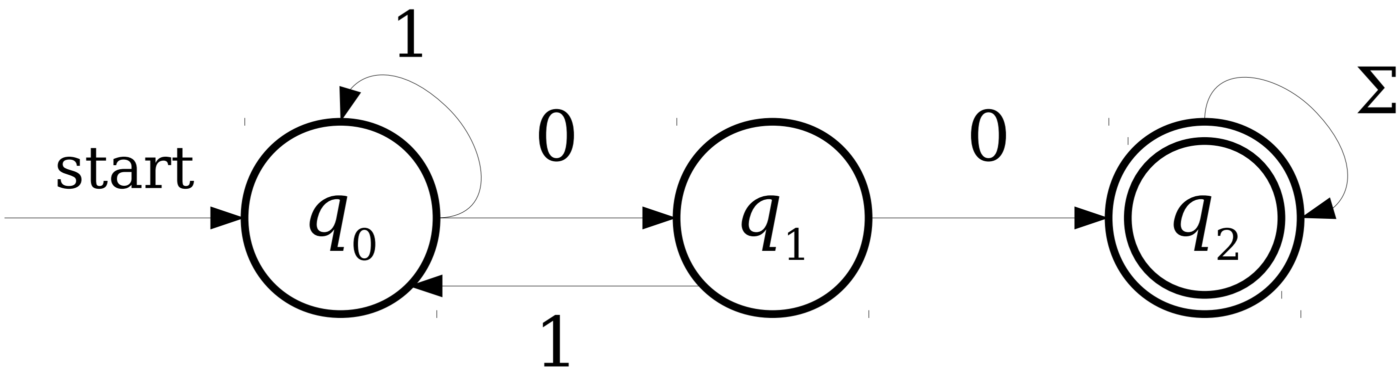


Complements of Regular Languages

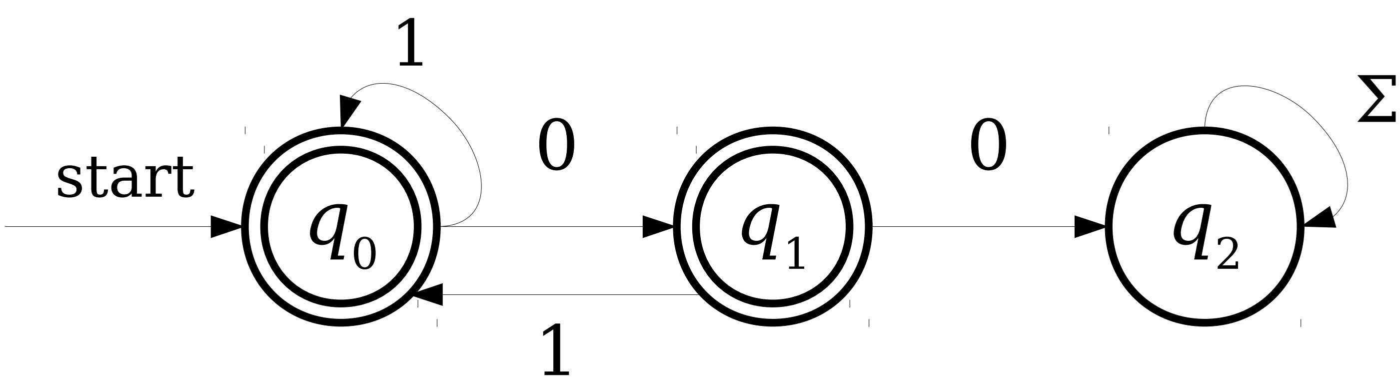
- As we saw a few minutes ago, a ***regular language*** is a language accepted by some DFA.
- ***Question:*** If L is a regular language, is $\overline{L}$ necessarily a regular language?
- If the answer is "yes," then if there is a way to construct a DFA for L, there must be some way to construct a DFA for $\overline{L}$.
- If the answer is "no," then some language L can be accepted by some DFA, but $\overline{L}$ cannot be accepted by any DFA.

Complementing Regular Languages

$L = \{ w \in \{0, 1\}^* \mid w \text{ contains } 00 \text{ as a substring } \}$

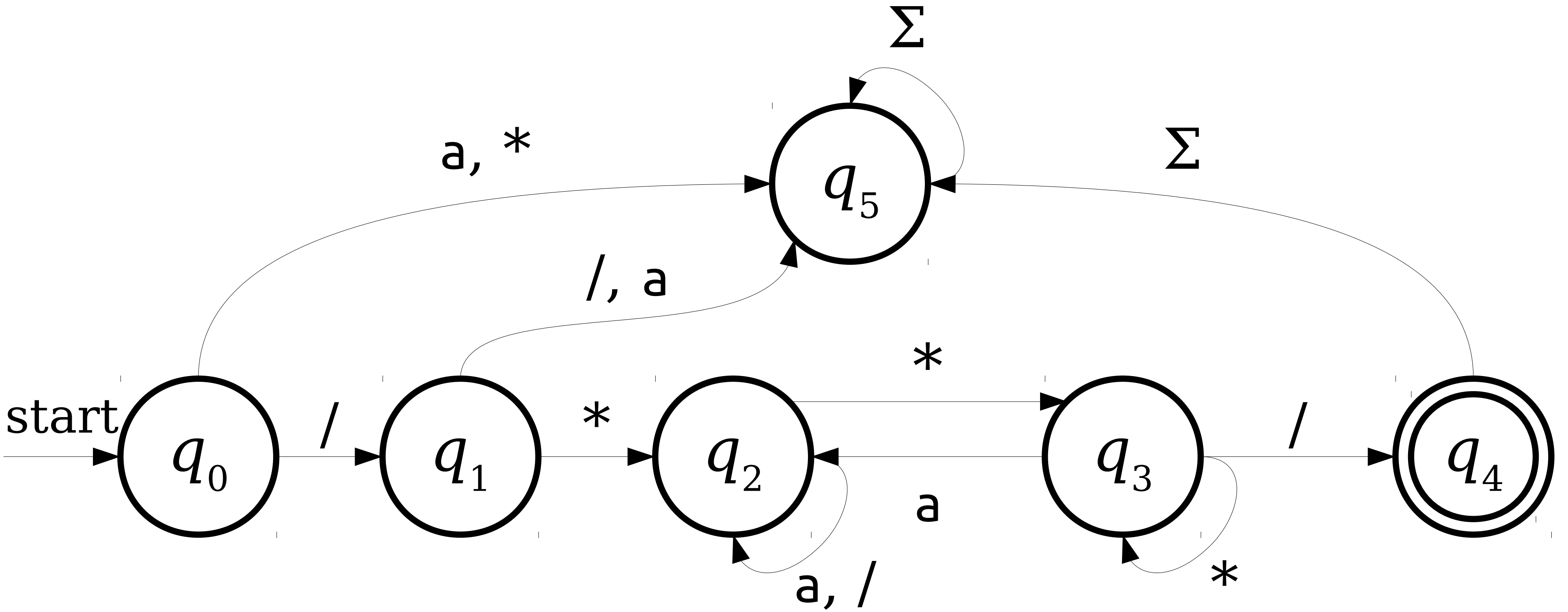


$\overline{L} = \{ w \in \{0, 1\}^* \mid w \textbf{\textit{ does not}} \text{ contain } 00 \text{ as a substring } \}$



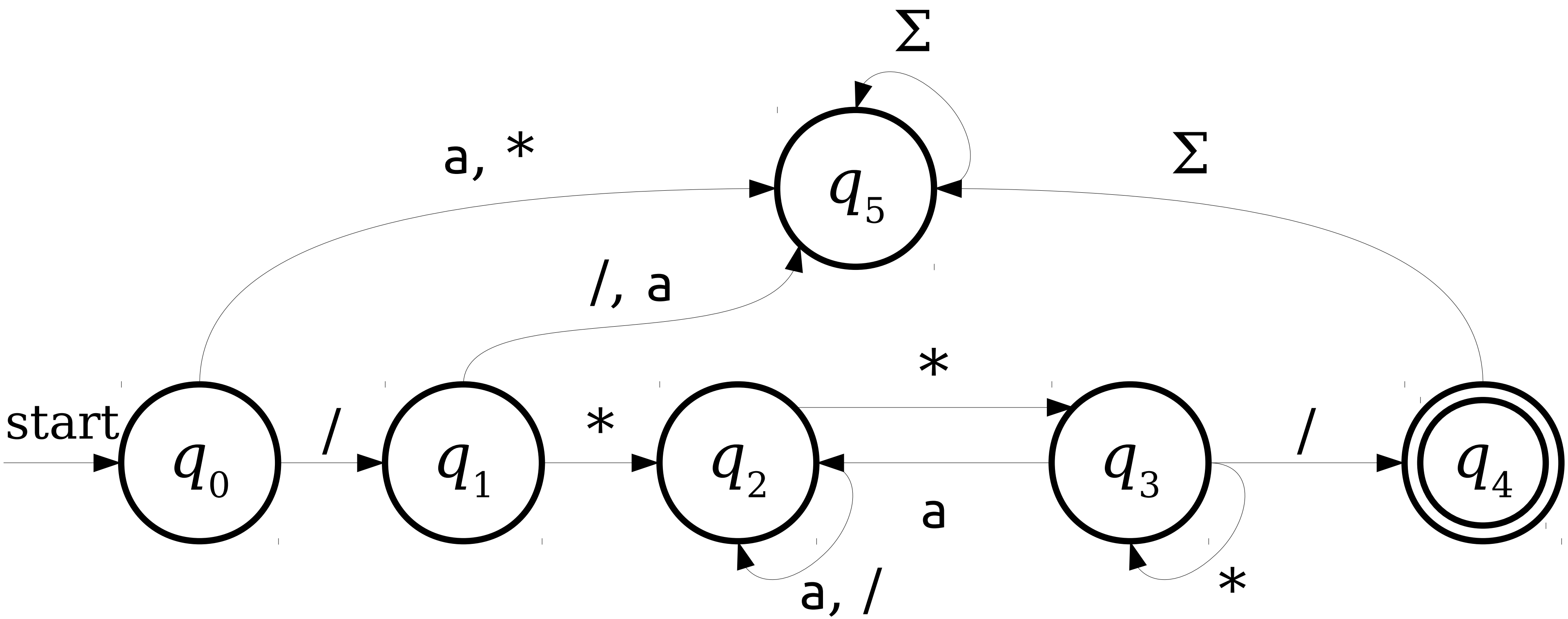
More Elaborate DFAs

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



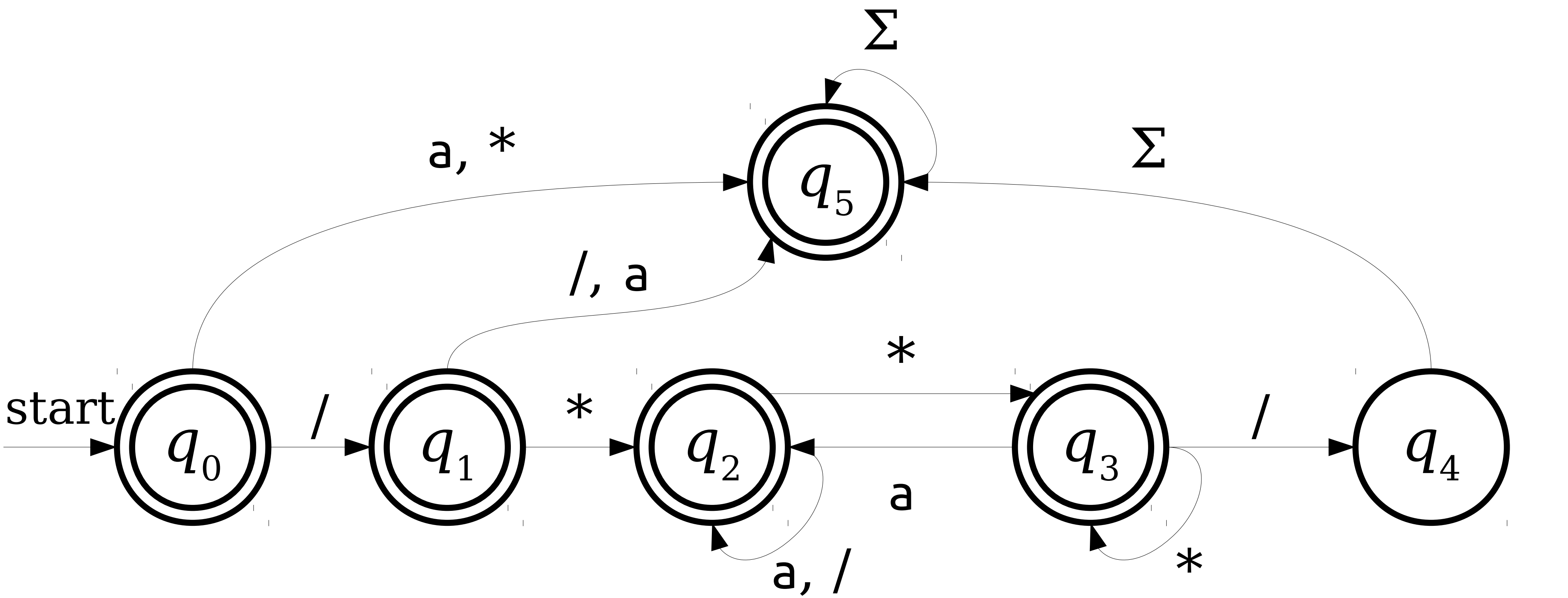
More Elaborate DFAs

$\overline{L}$ = { w ∈ {a, *, /}* | *w doesn't* represent a C-style comment }



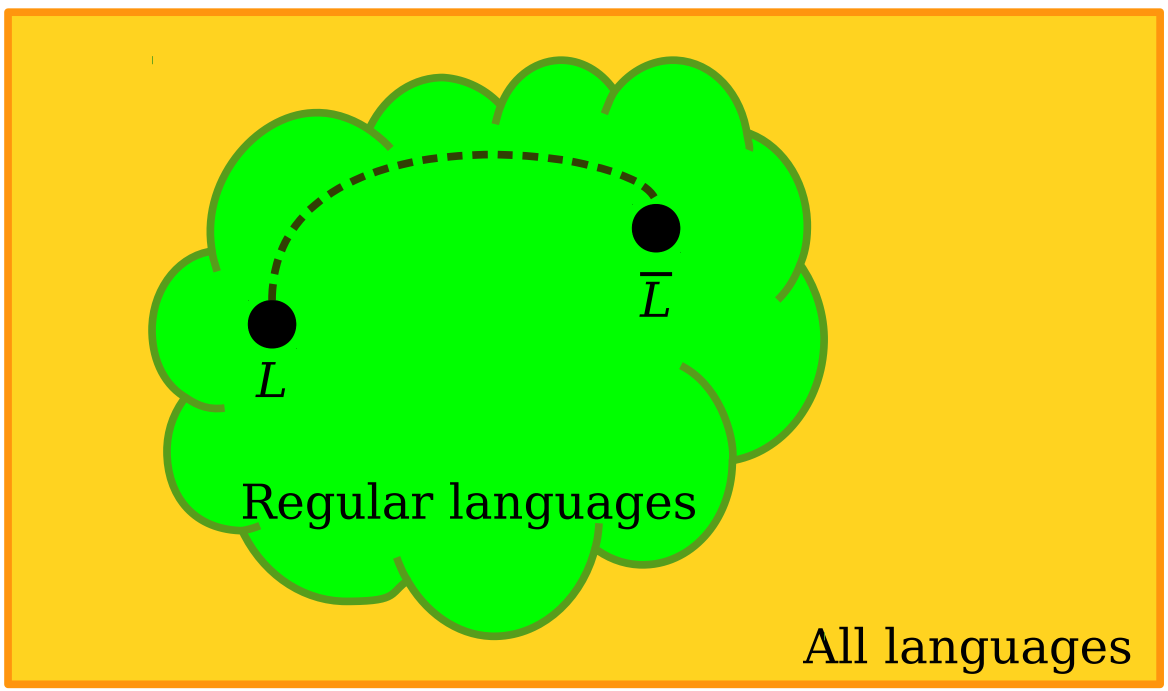
More Elaborate DFAs

$\overline{L}$ = { w ∈ {a, *, /}* | *w doesn't* represent a C-style comment }



Closure Properties

- ***Theorem:*** If L is a regular language, then $\overline{L}$ is also a regular language.
- As a result, we say that the regular languages are ***closed under complementation***.



Time-Out For Announcements!

Midterm Logistics

- Our first midterm is tonight from 7PM – 10PM.
- Room locations divvied up by last (family) name:
  - `Abd – Lin`: Go to **Cubberly Auditorium**.
  - `Liu – Raj`: Go to **370-370**.
  - `Ram – Zhu`: Go to **420-040**.
- Closed-book, closed-computer, limited-notes.
  - You can have a double-sided 8.5” × 11” sheet of notes when you take the exam.
- Topic coverage is PS1 – PS3 and Lectures 00–08.

Solution Sets

- We've moved solution sets for everything midterm-related down to the basement of Gates. They're still there for pickup between now and the exam.
- ***We will be recycling all unclaimed solution sets on Wednesday***. If you'd like copies of anything, please take them soon!

Your Questions

“Any areas of CS that have application in theater, film, art and literature?”

Did you see Stanford's production of *Hairspray* last year? Matt Lathrop and his team did a phenomenal job putting together these crazy cool LED boards that they used to make the scene evolve and change in real-time.

About a year ago, two students put together a program that made it easier to compare multiple translations of a piece of literature side-by-side. About three years ago a student did a large-scale analysis of writings about cities over hundreds of years to learn how cities looked and sounded through the years and how people described them.

Also a few years back, someone analyzed the works of a famous writer (I *think* it was Virginia Woolf, but I might be wrong) and figured out that the conventional explanation of how she explored new writing styles was wrong. That came as a real shock to a lot of people.

Also, did you see *Frozen* or *Inside Out*? I rest my case. ☺

Back to CS103!

NFAs

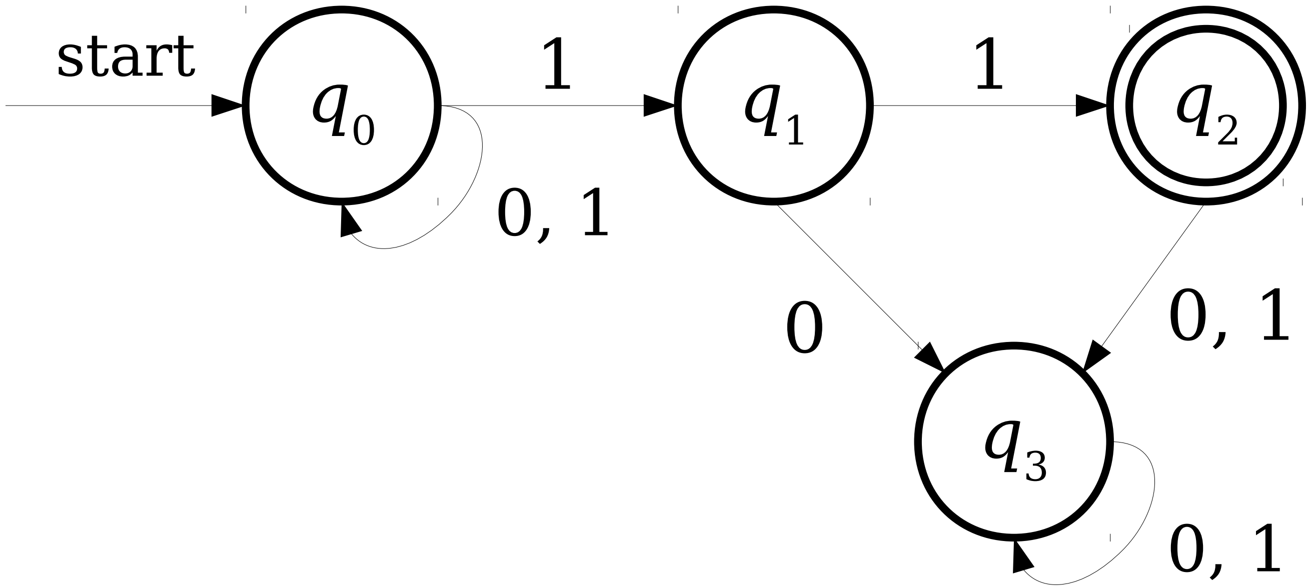
NFAs

- An **NFA** is a
  - **N**ondeterministic
  - **F**inite
  - **A**utomaton
- Structurally similar to a DFA, but represents a fundamental shift in how we'll think about computation.

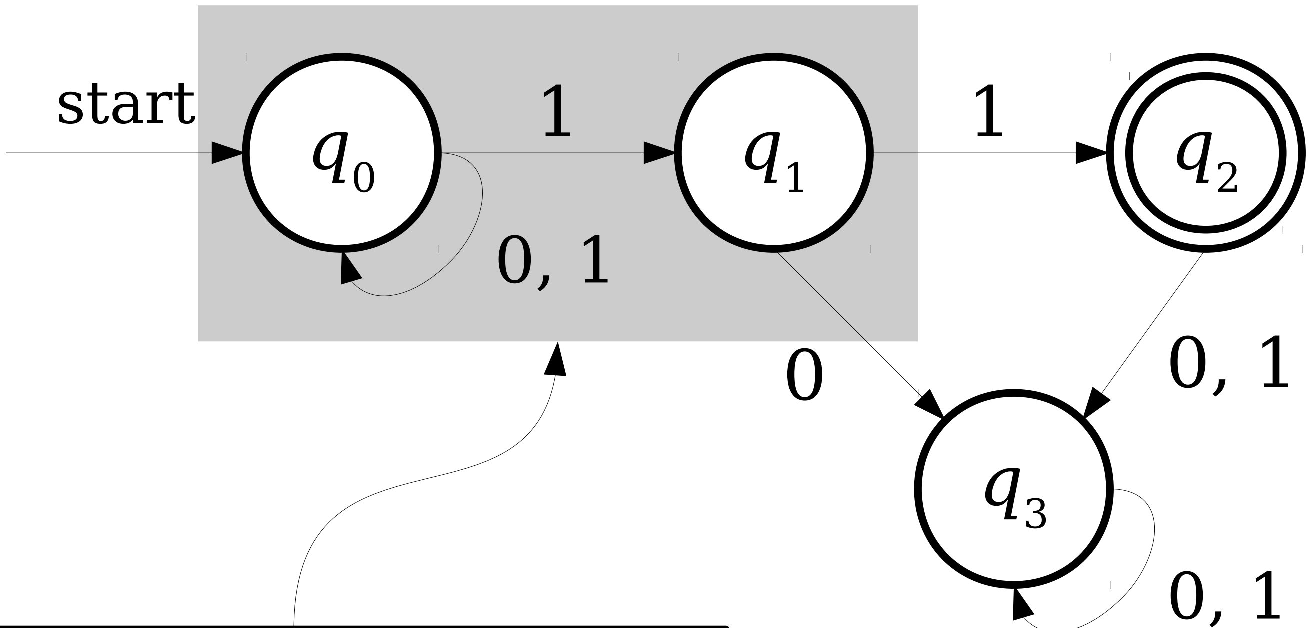
(Non)determinism

- 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 may have multiple decisions that it can make at one point.
- The machine accepts if ***any*** series of choices leads to an accepting state.

A Simple NFA

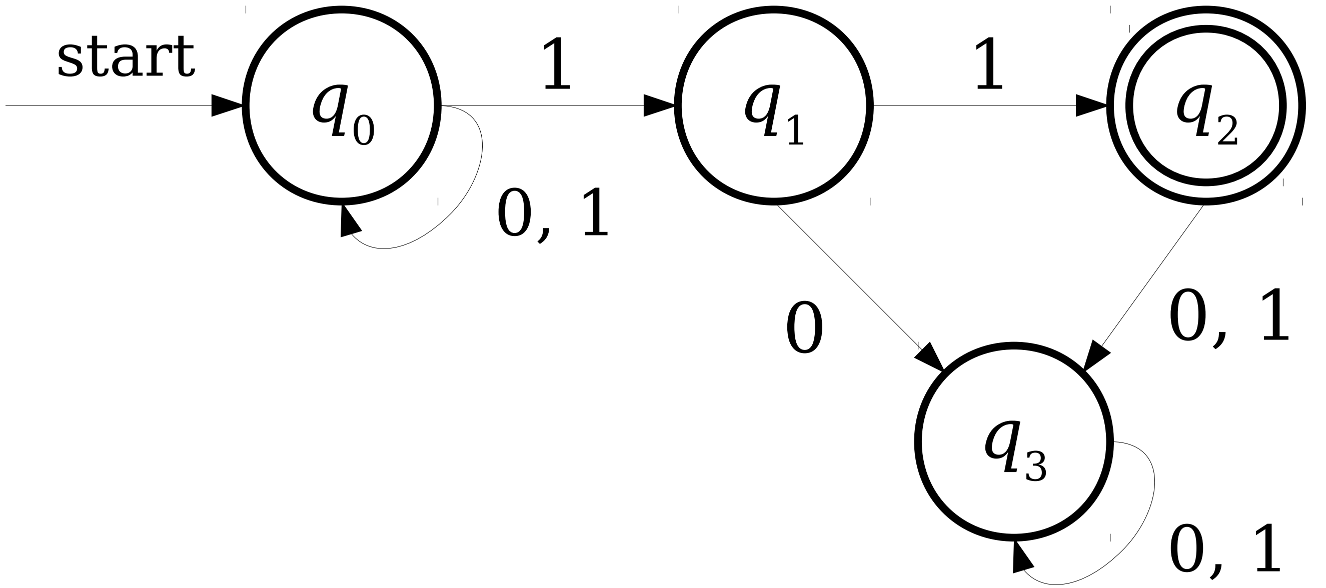


A Simple NFA



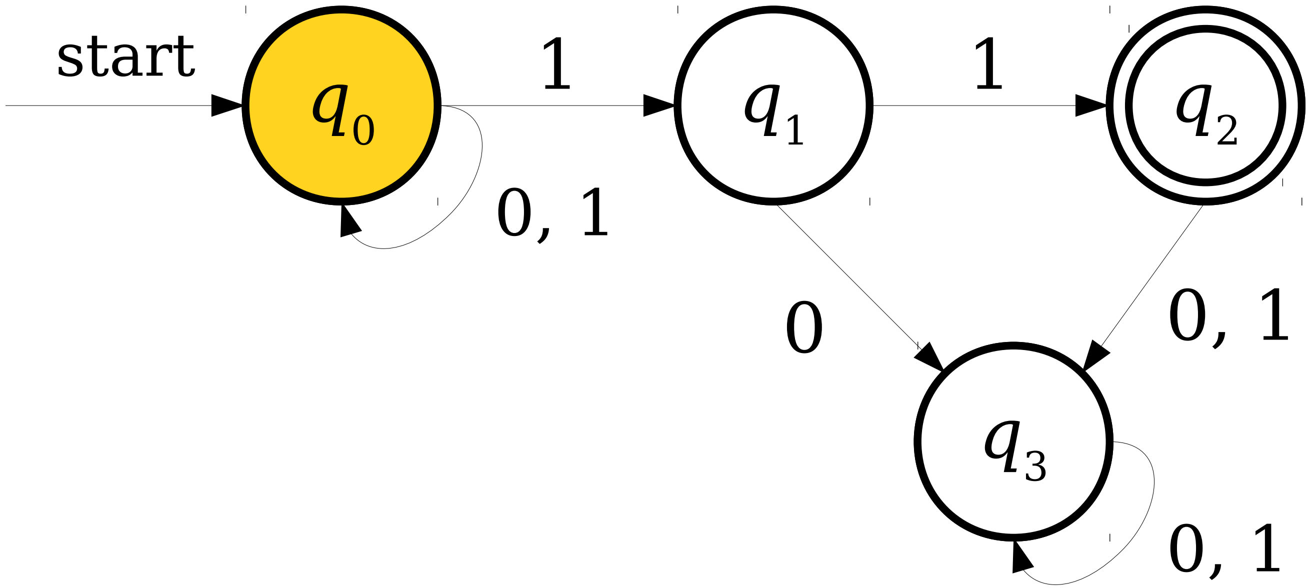
q_0 has two transitions defined on 1!

A Simple NFA



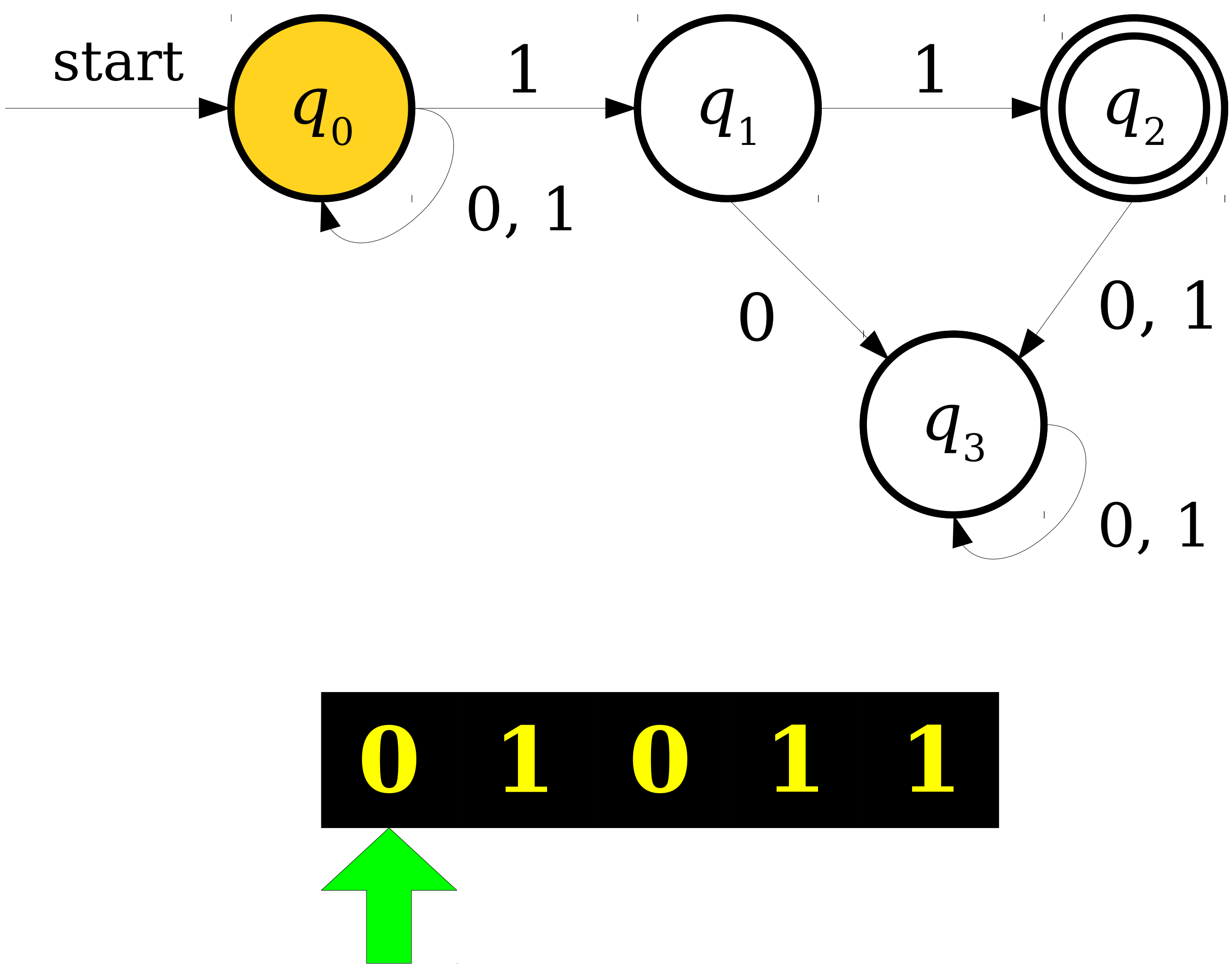
0 1 0 1 1

A Simple NFA

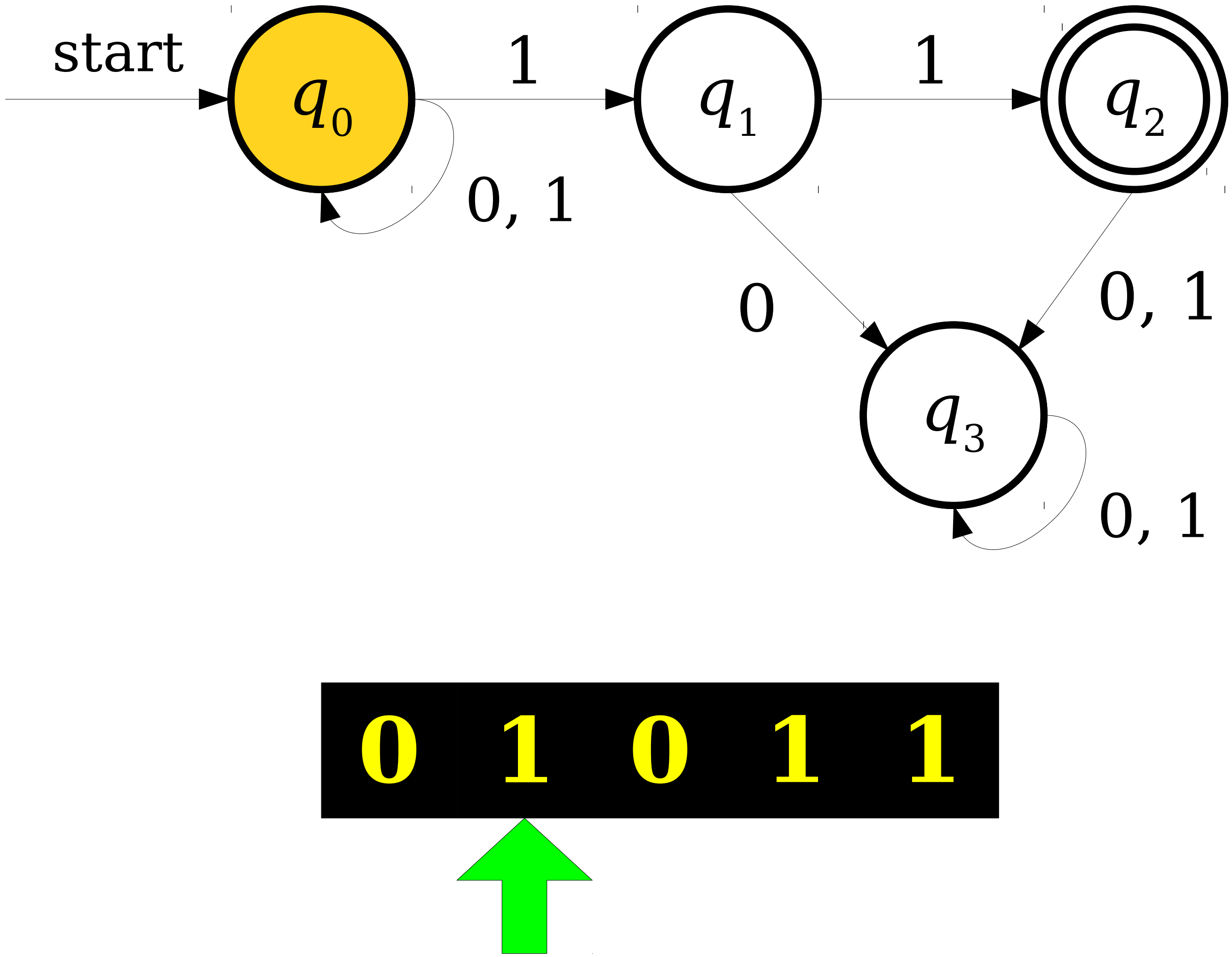


0 1 0 1 1

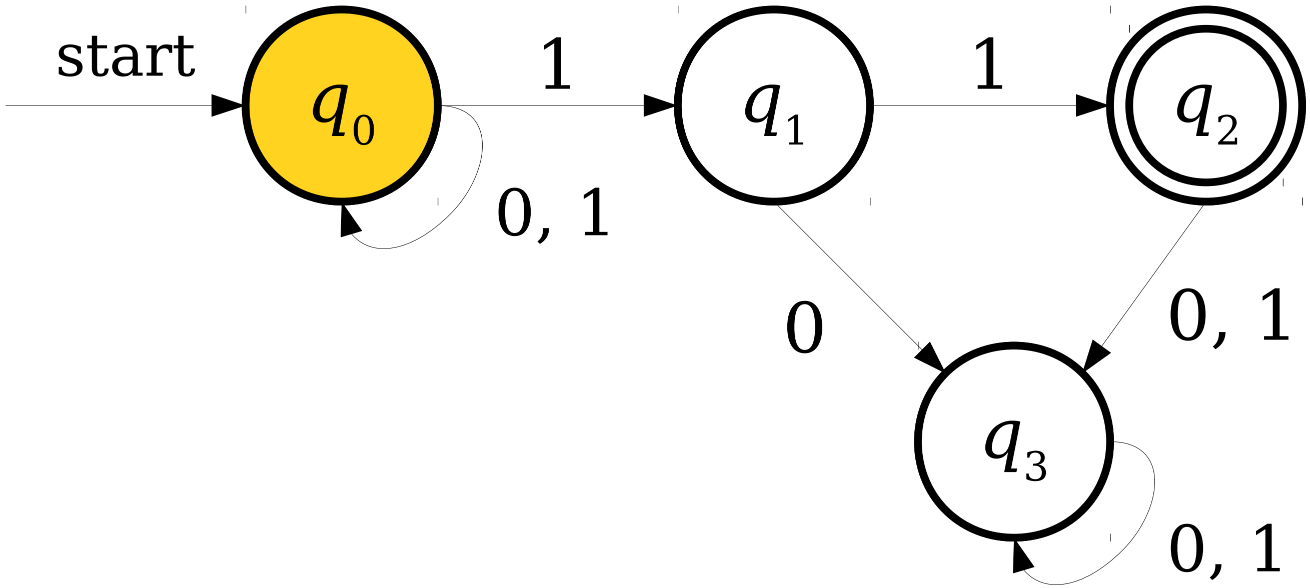
A Simple NFA

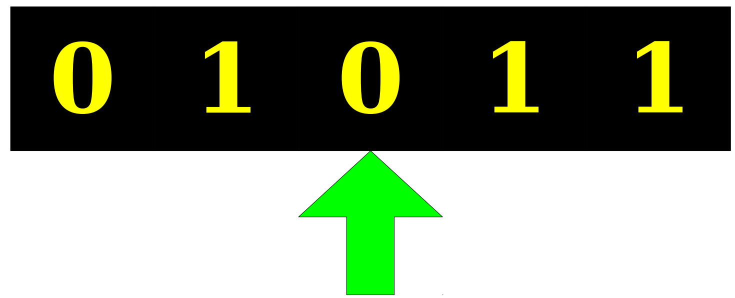


A Simple NFA

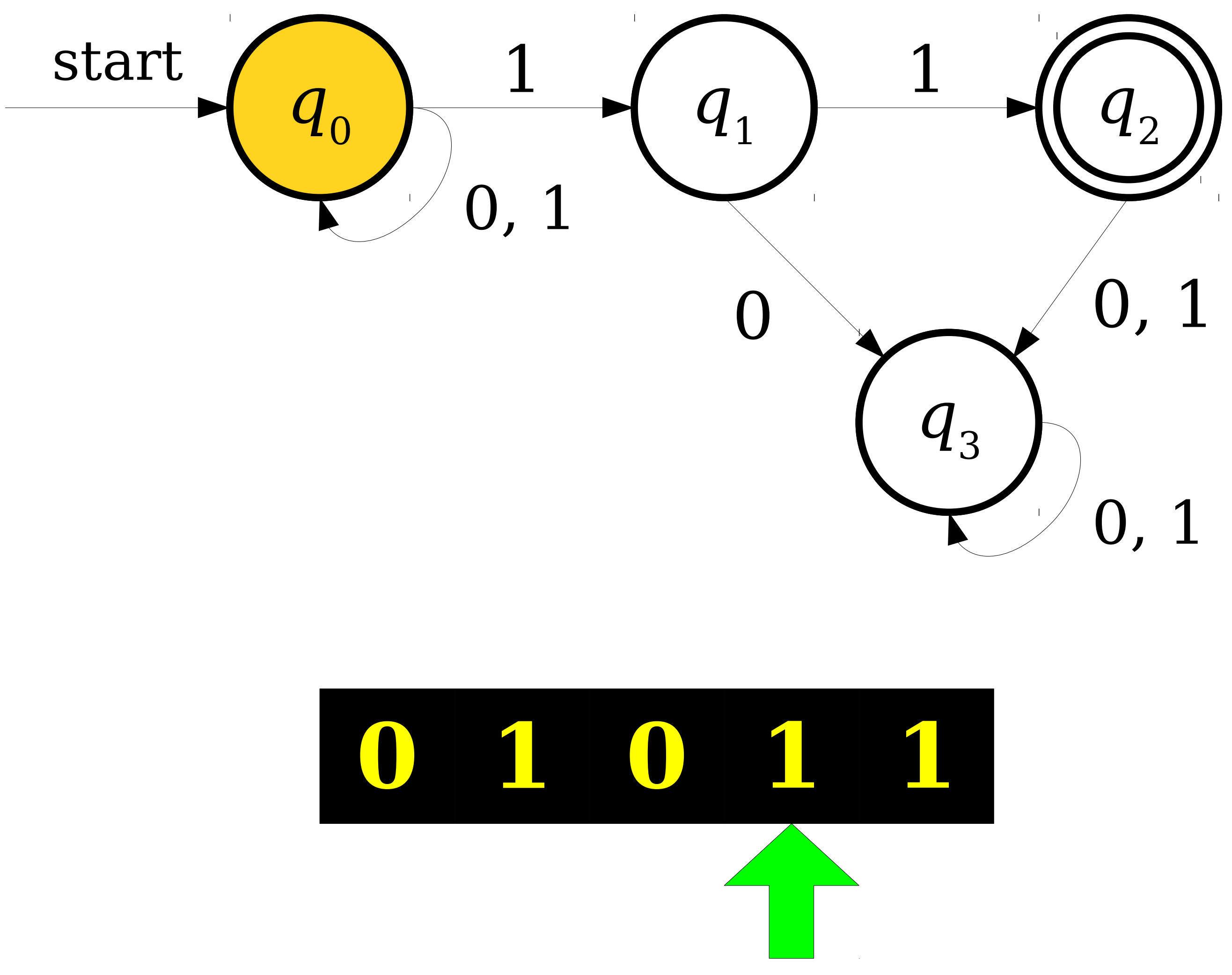


A Simple NFA

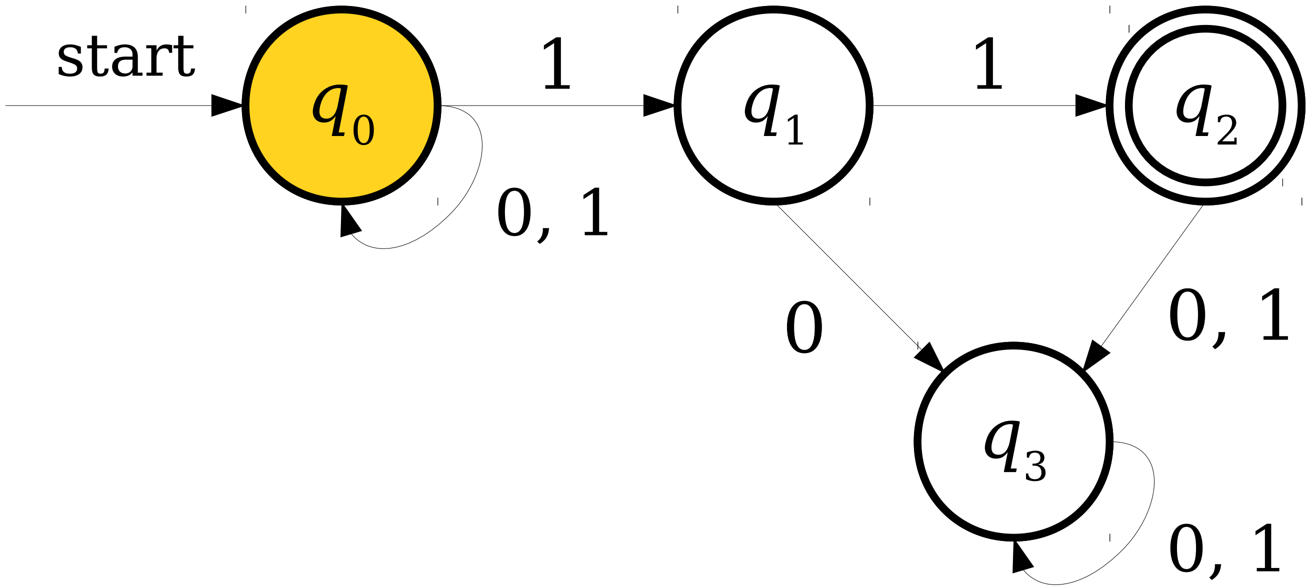




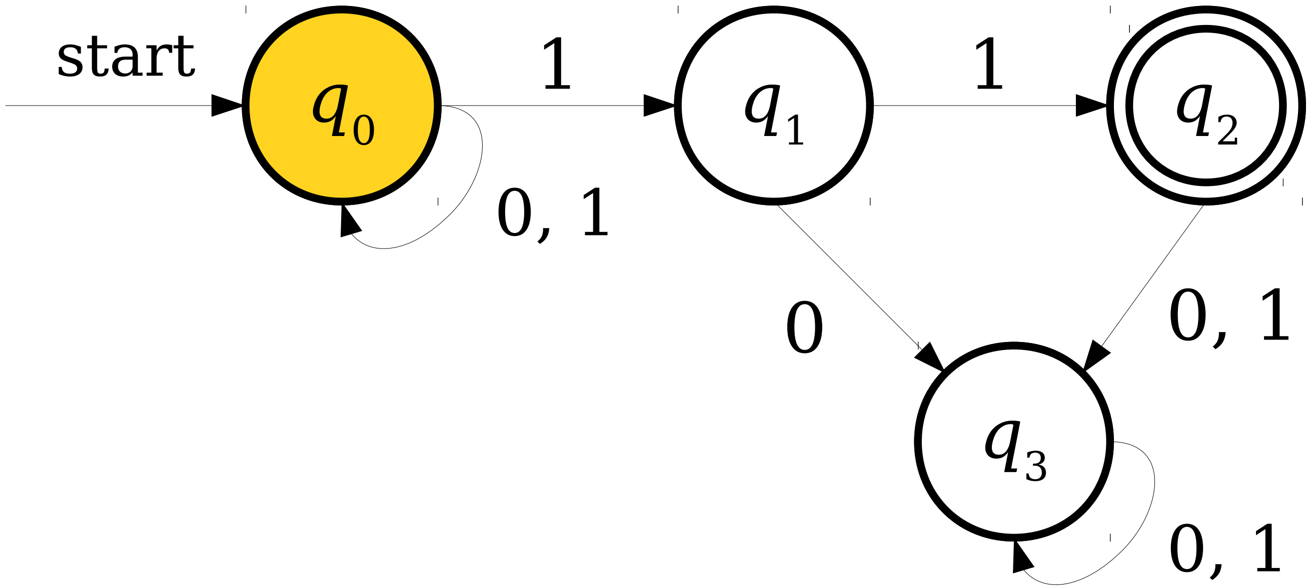
A Simple NFA



A Simple NFA

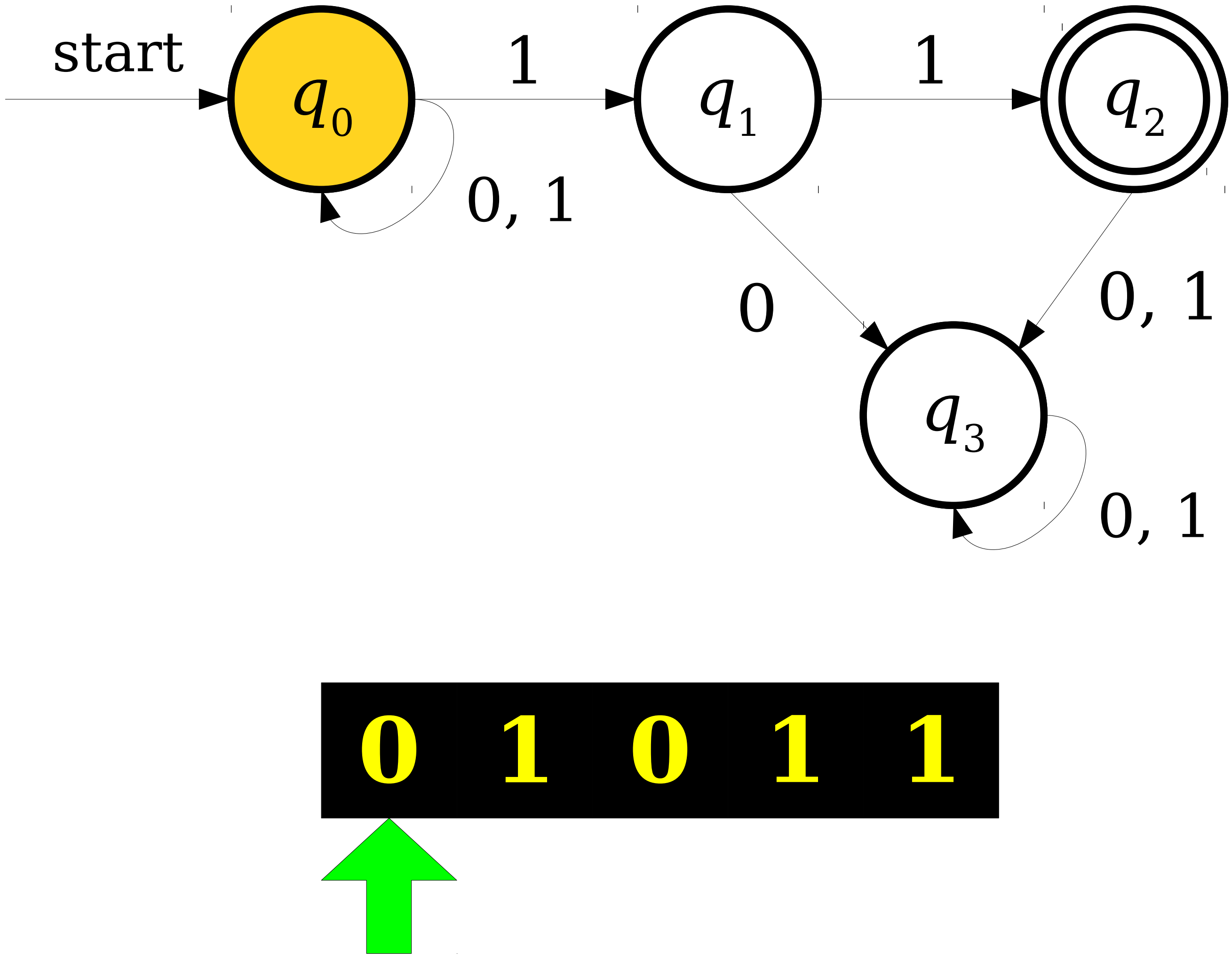


A Simple NFA

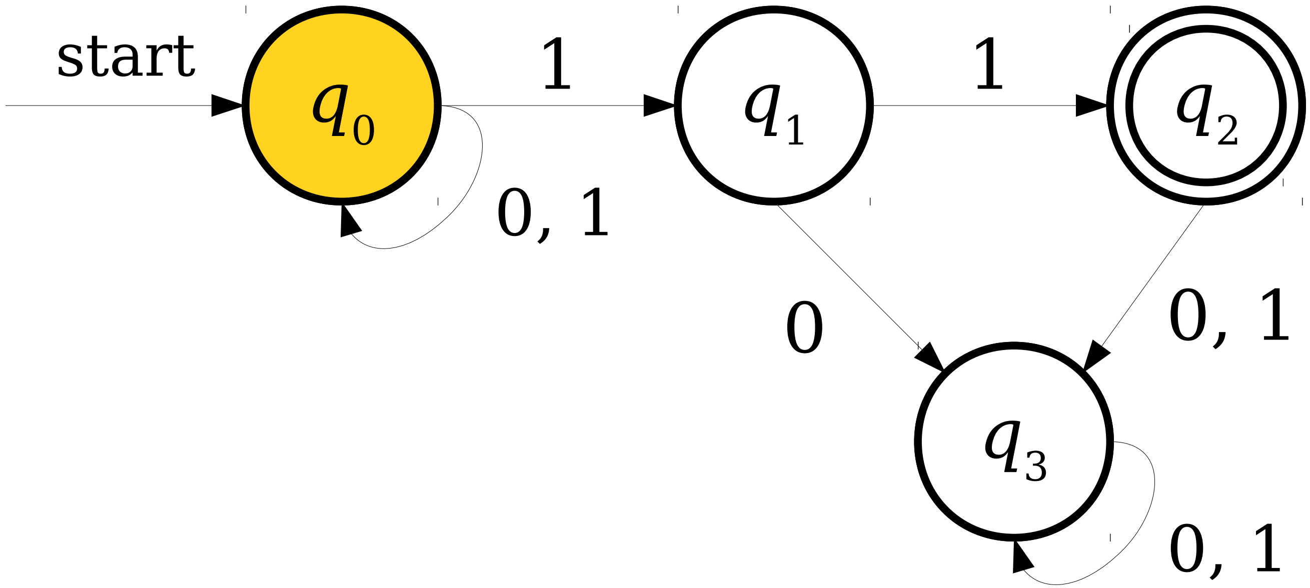


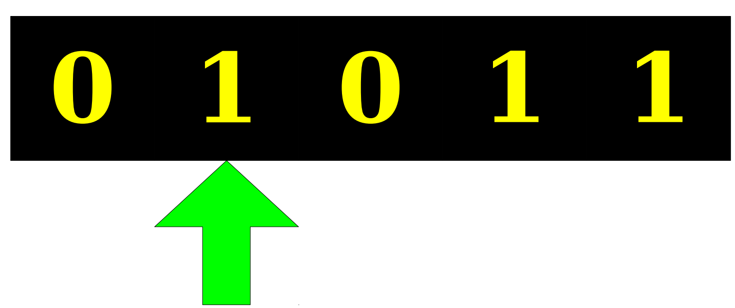
0 1 0 1 1

A Simple NFA

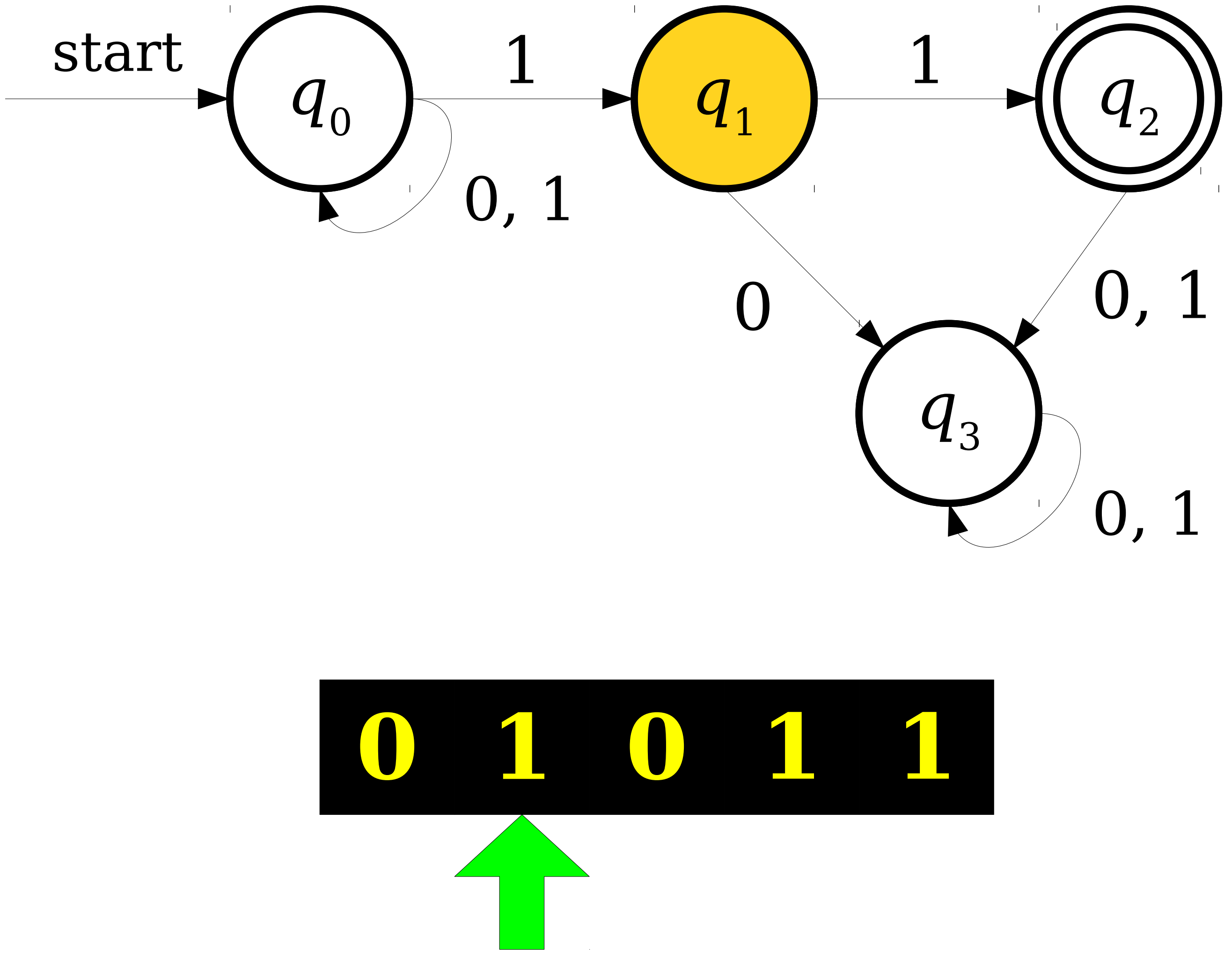


A Simple NFA

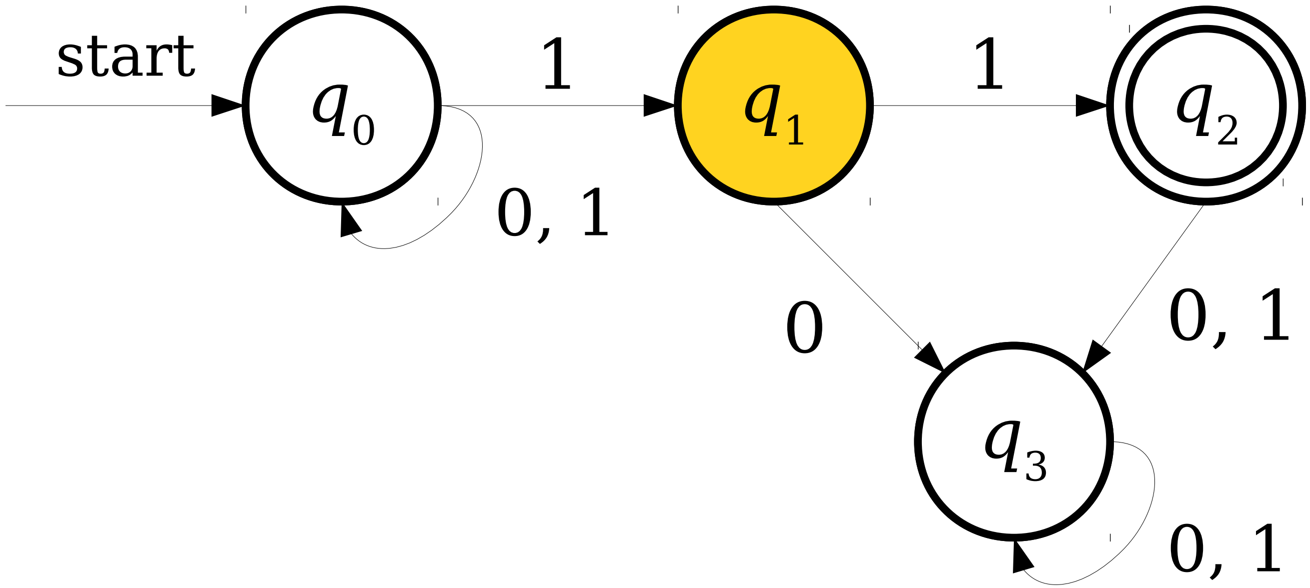


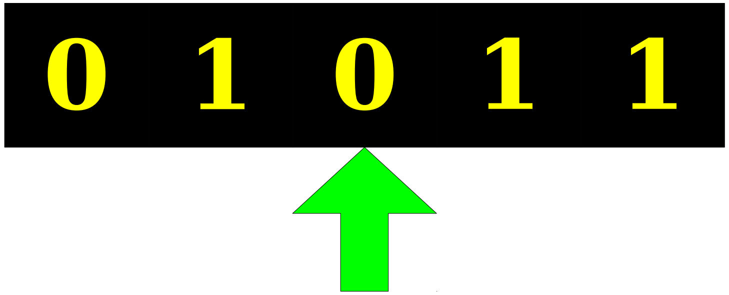


A Simple NFA

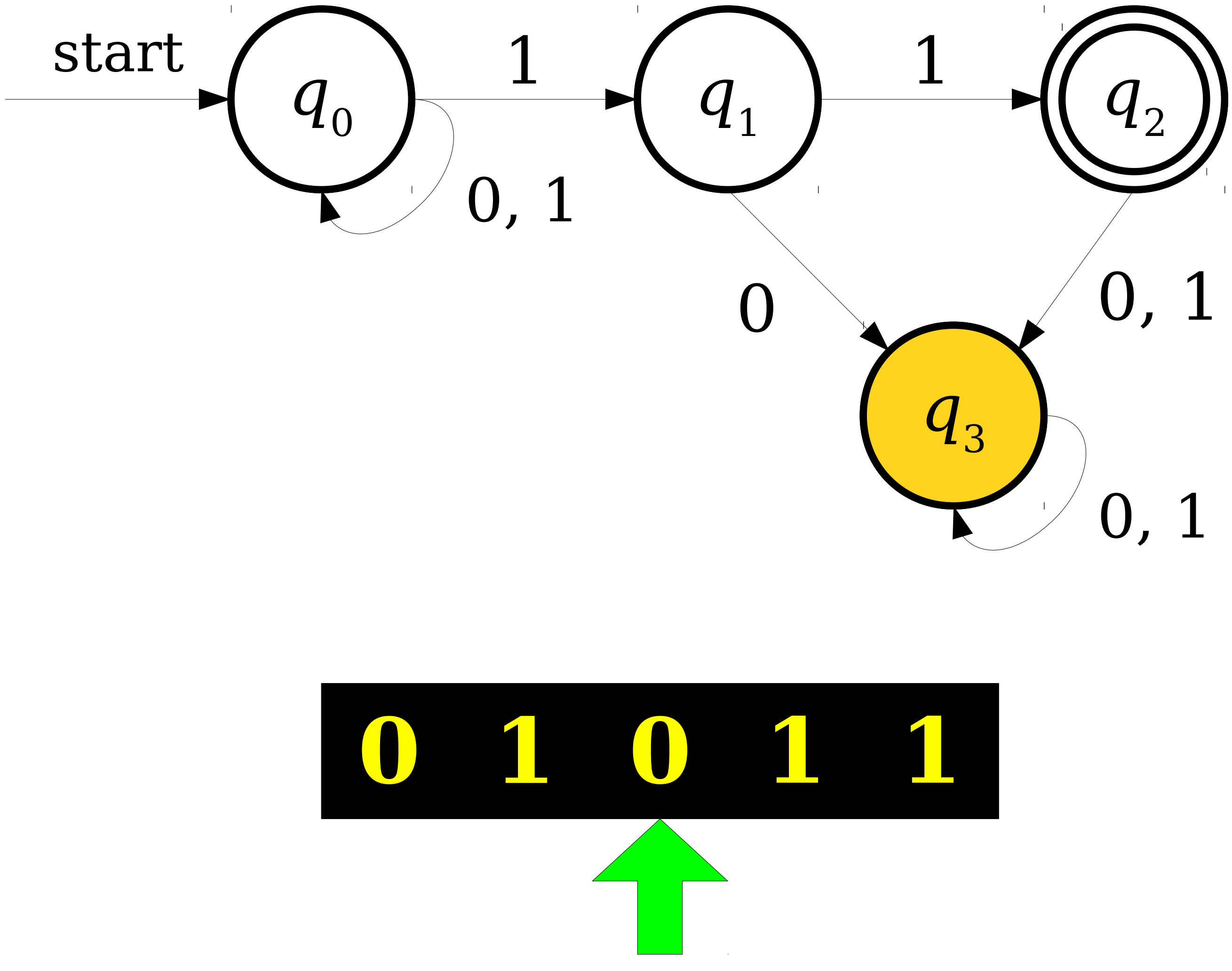


A Simple NFA

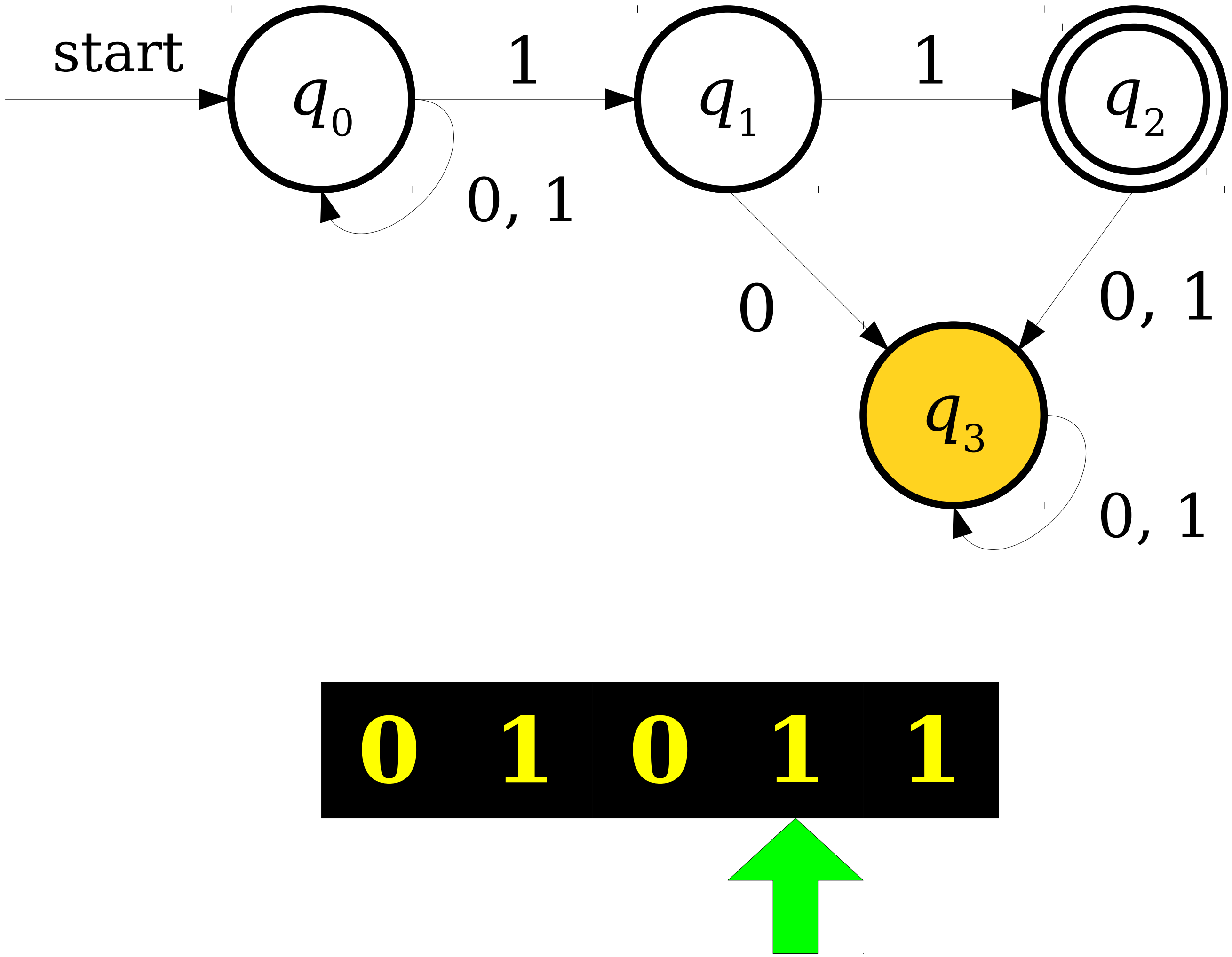




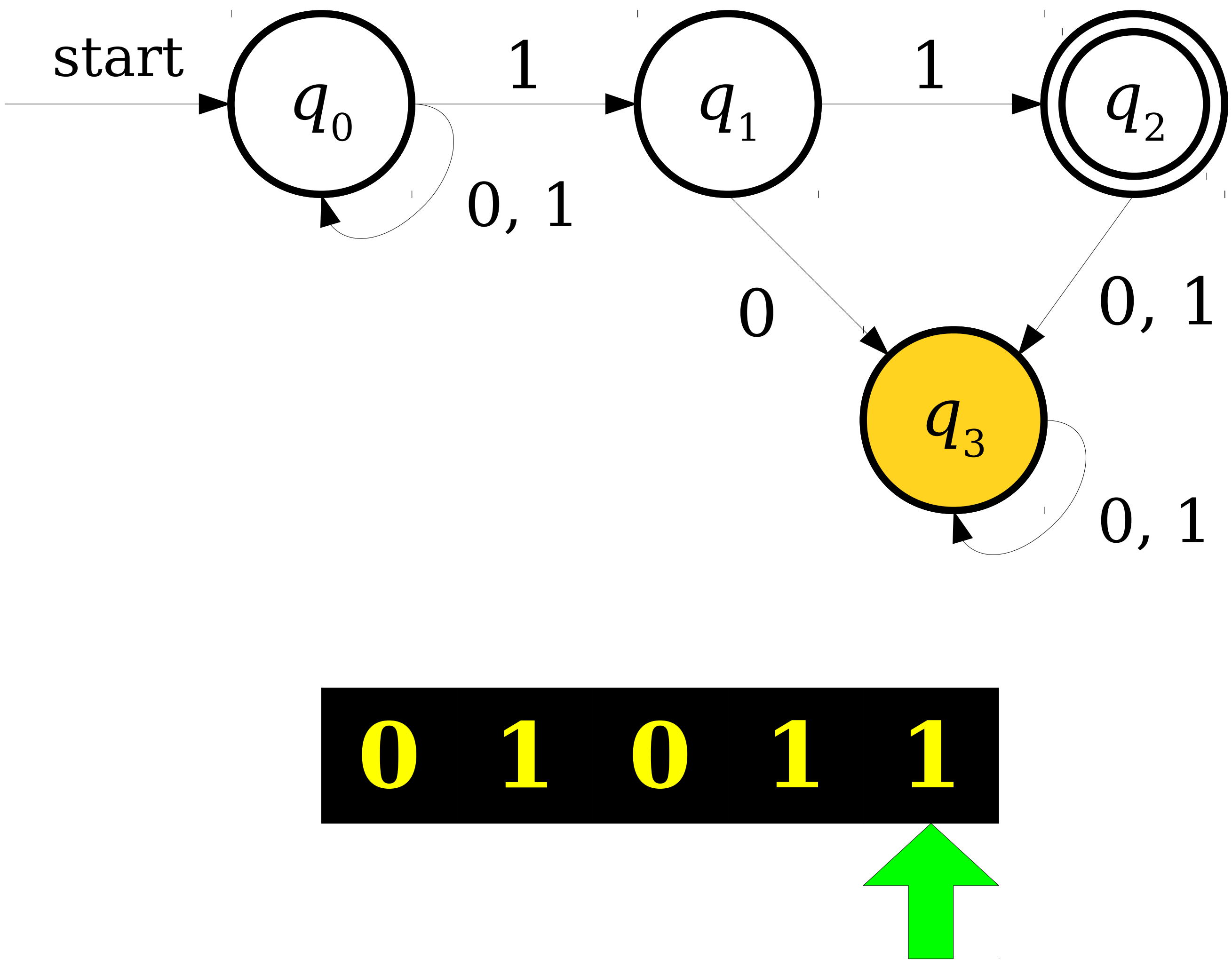
A Simple NFA



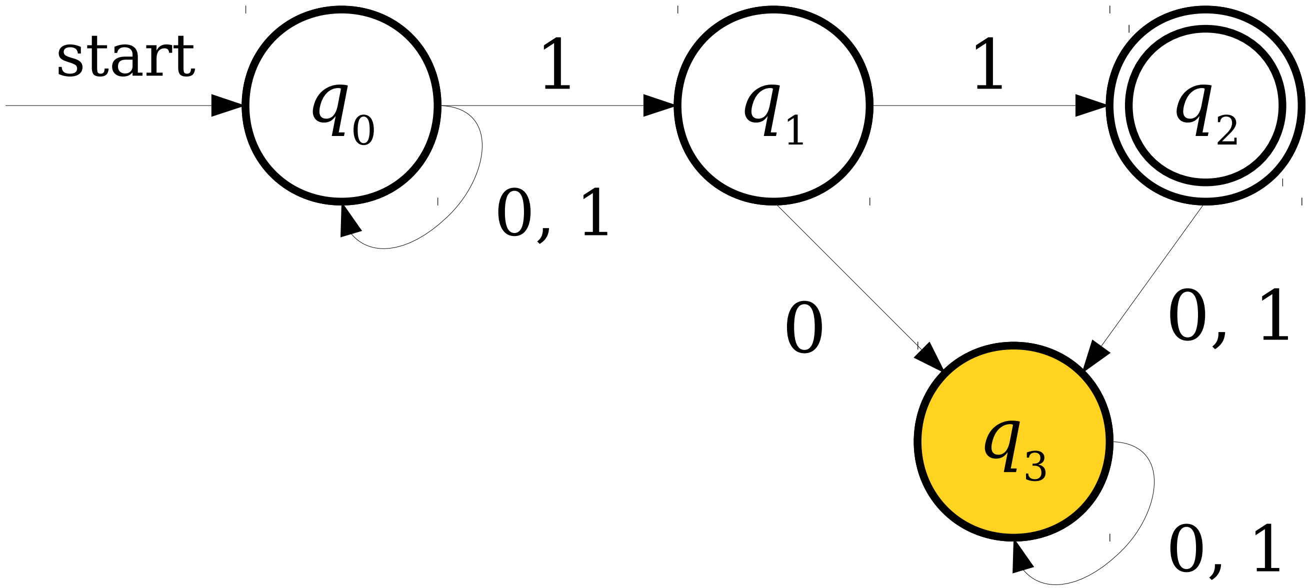
A Simple NFA



A Simple NFA

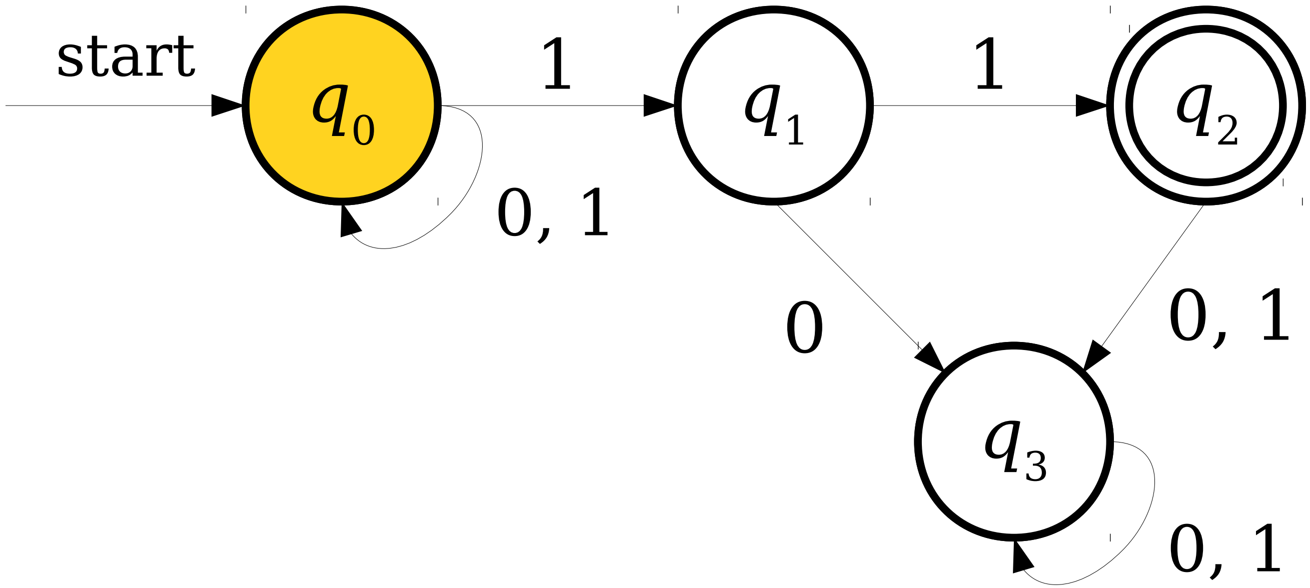


A Simple NFA

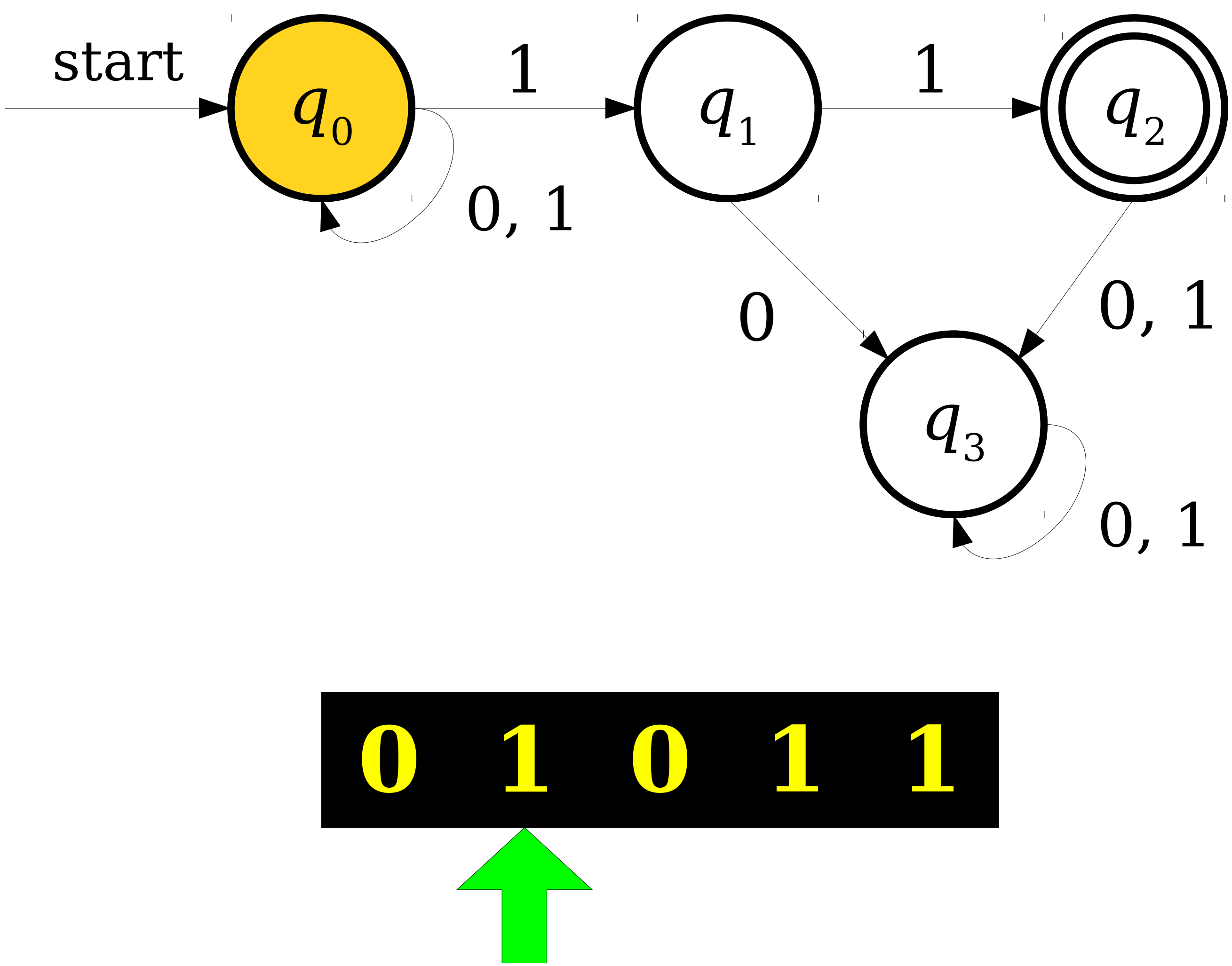


0 1 0 1 1

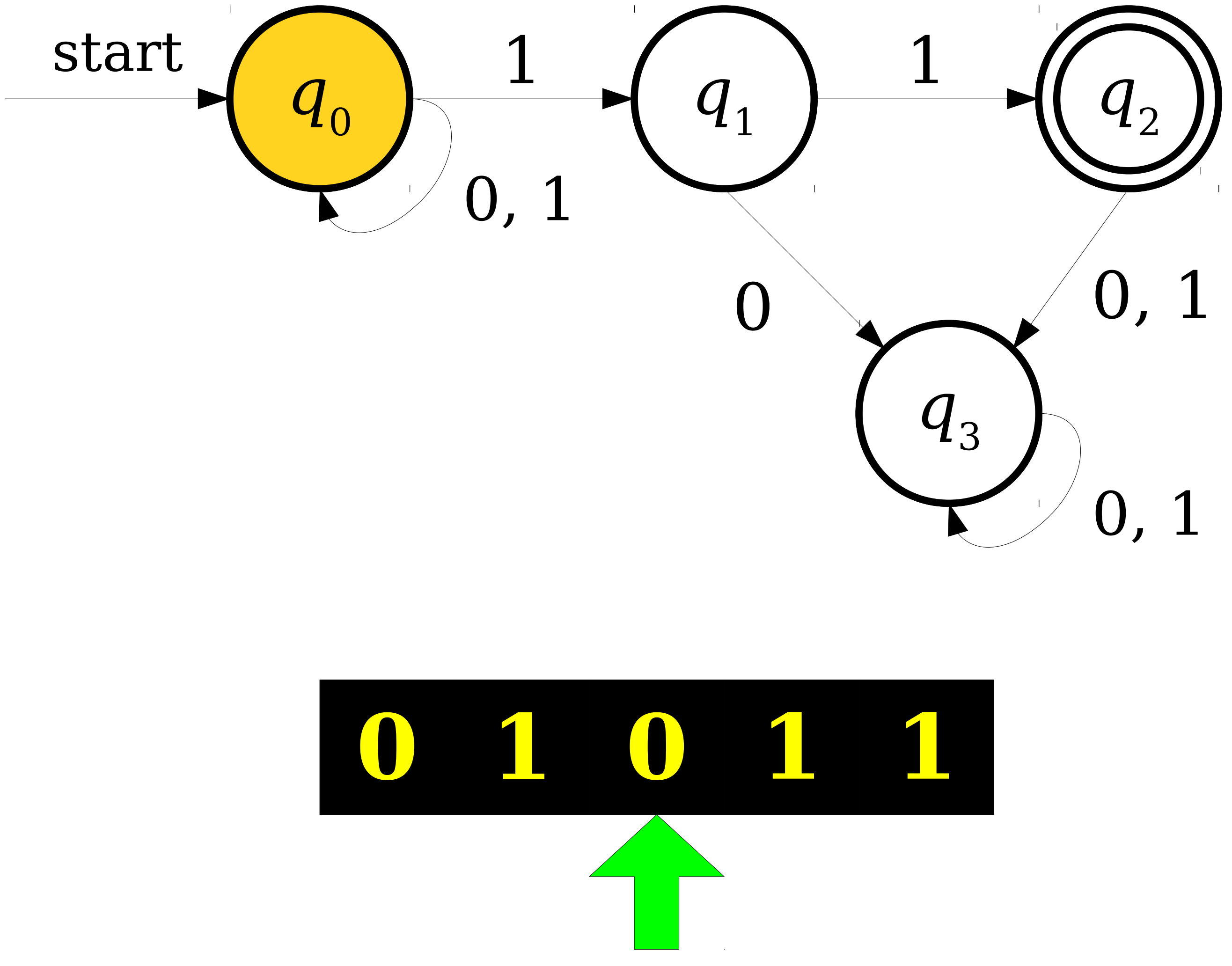
A Simple NFA



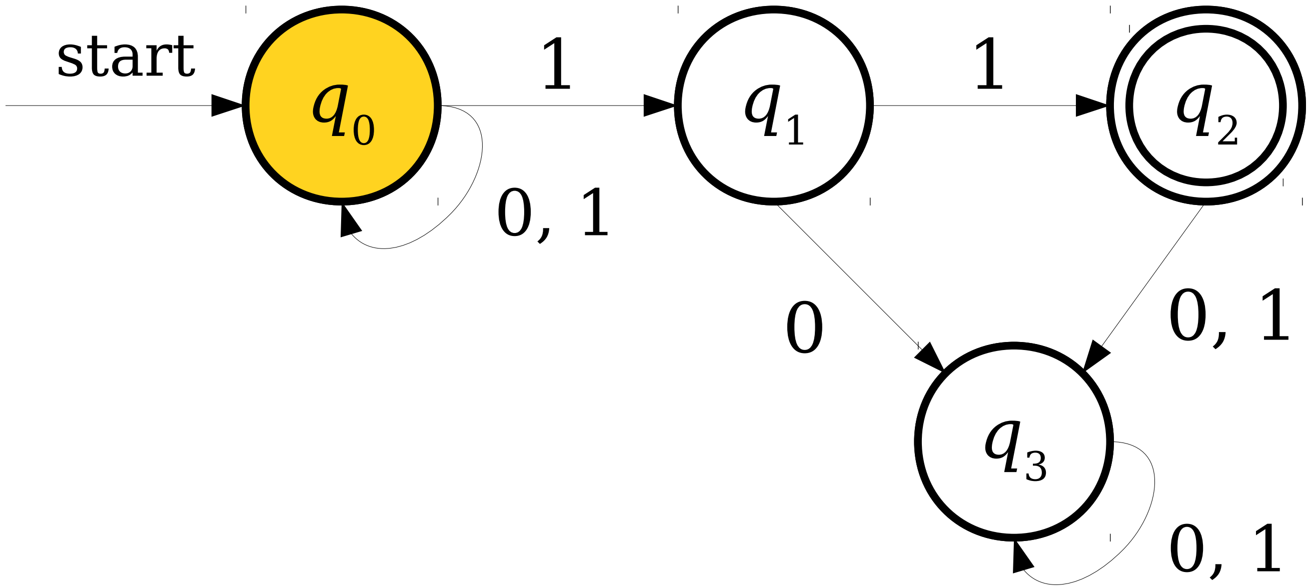
A Simple NFA

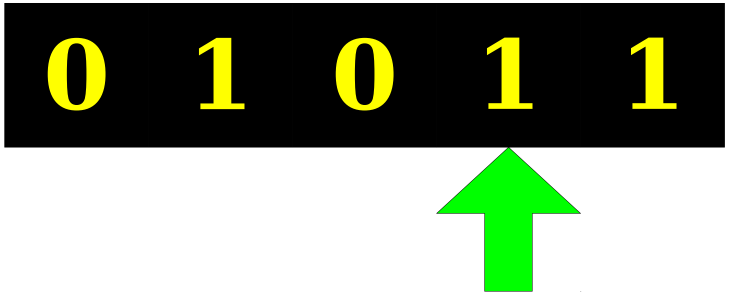


A Simple NFA

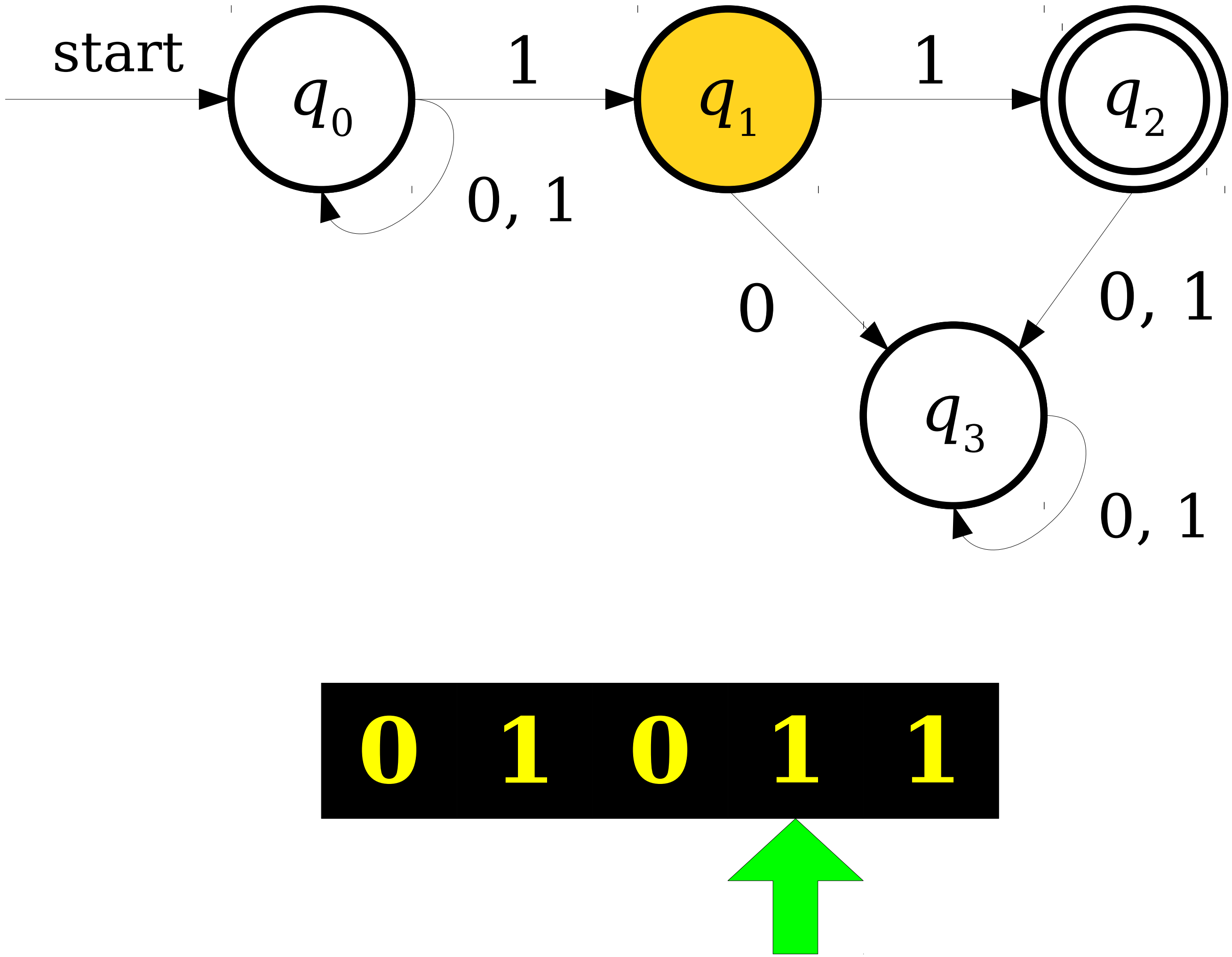


A Simple NFA

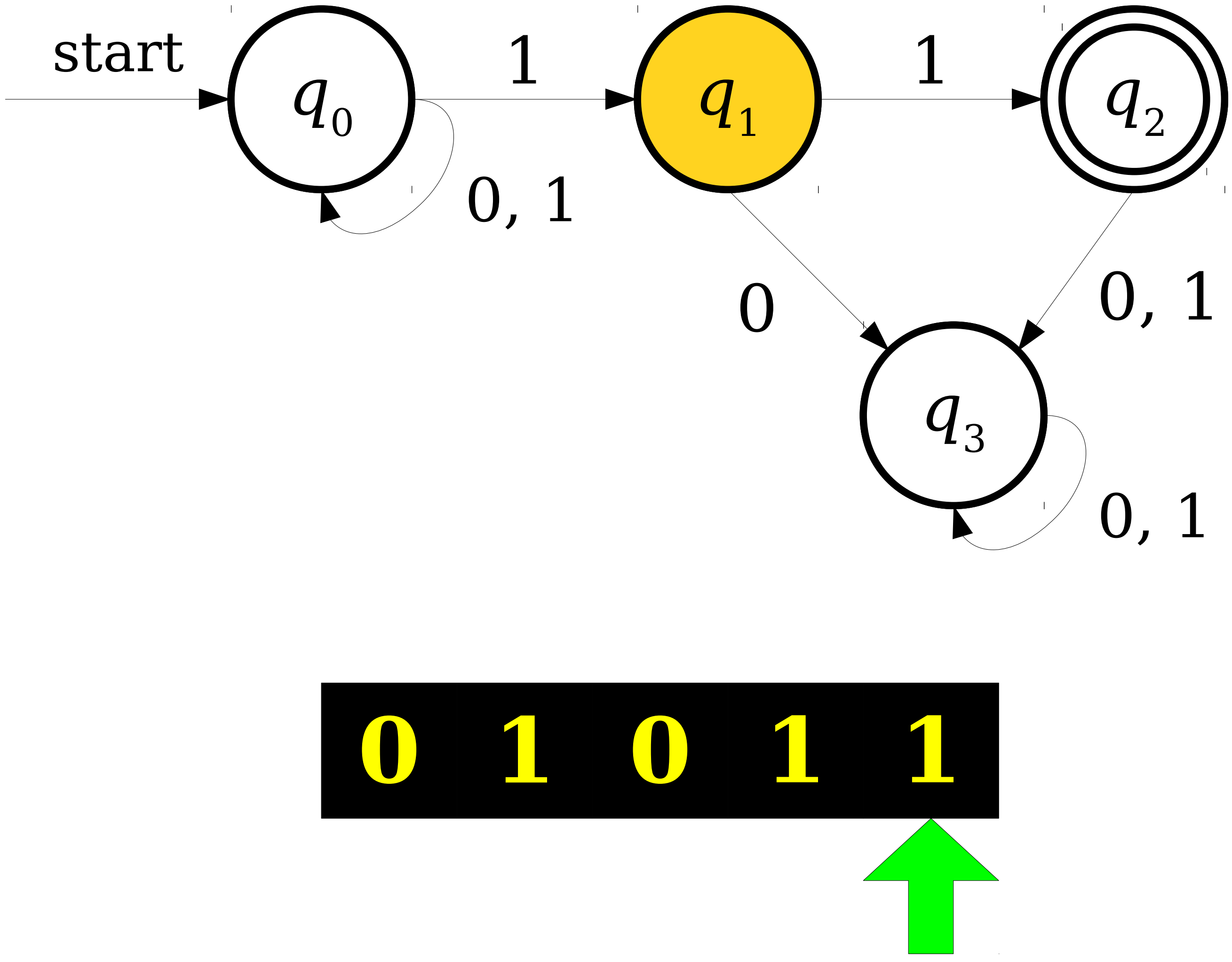




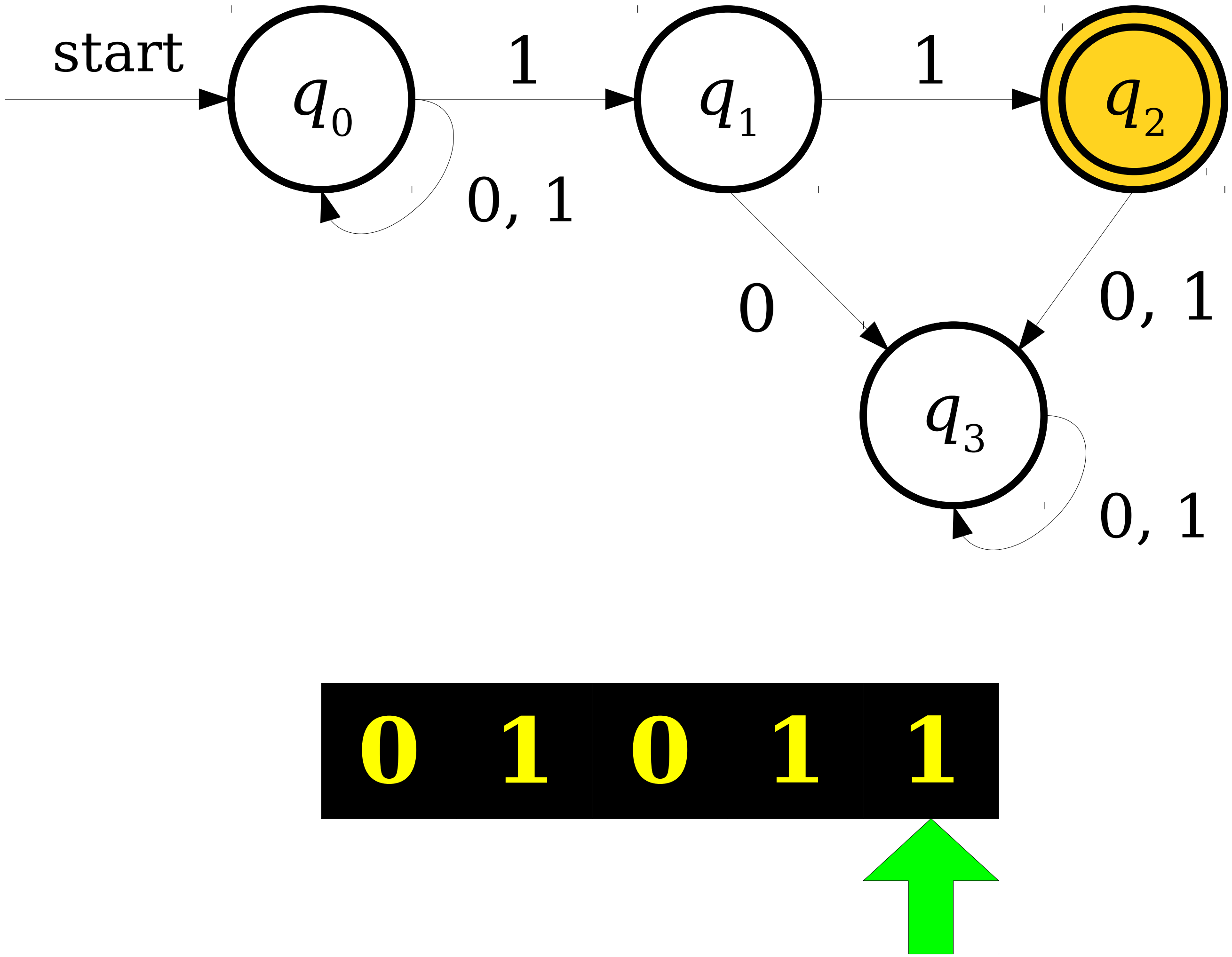
A Simple NFA



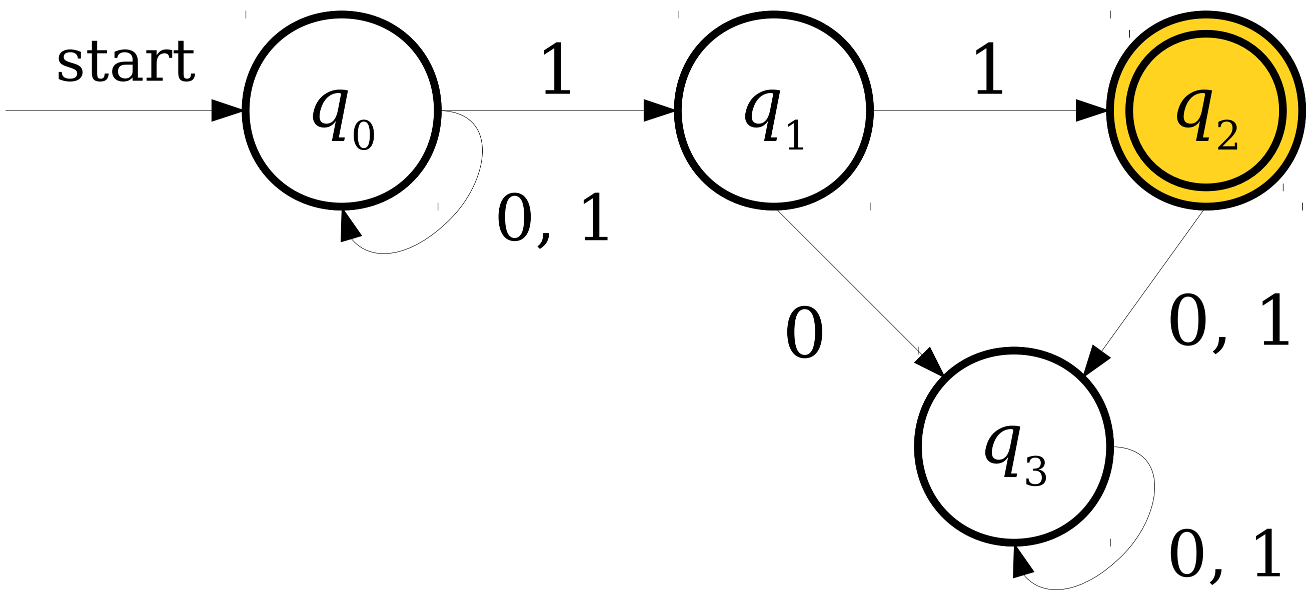
A Simple NFA



A Simple NFA

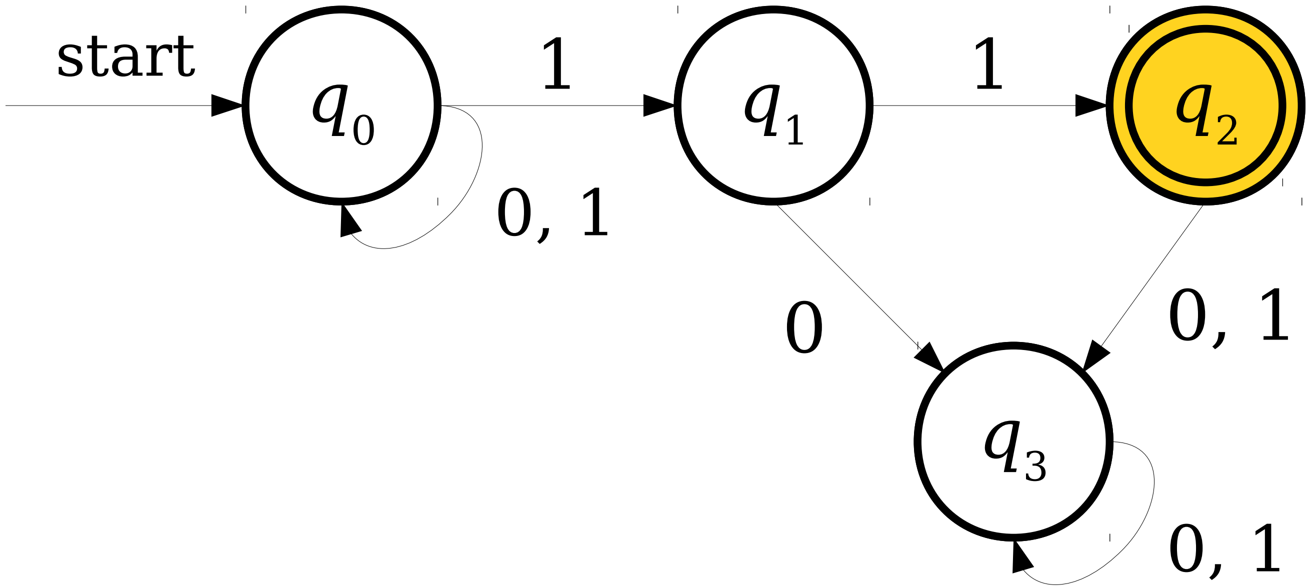


A Simple NFA



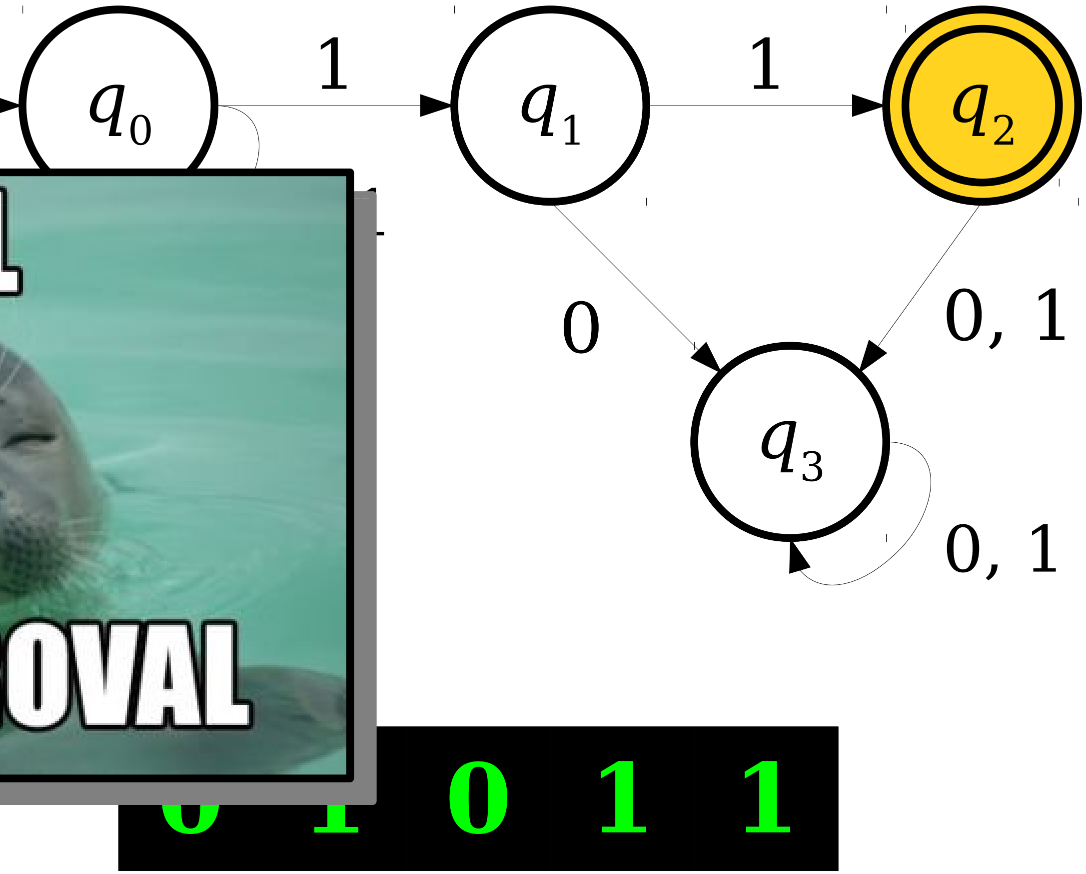
0 1 0 1 1

A Simple NFA

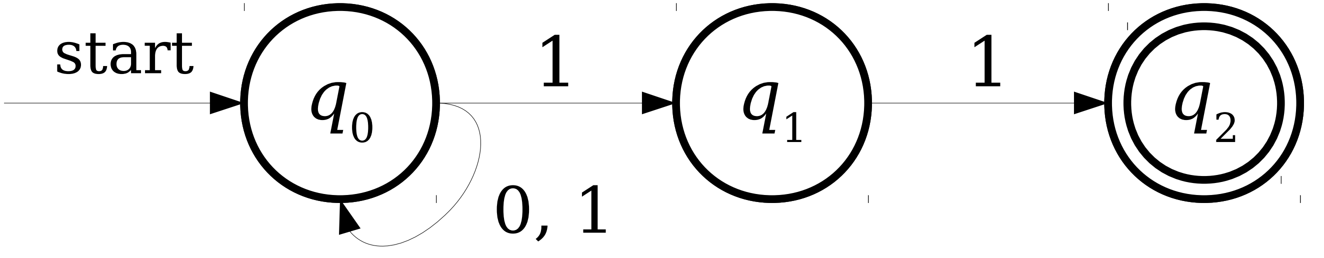


0 1 0 1 1

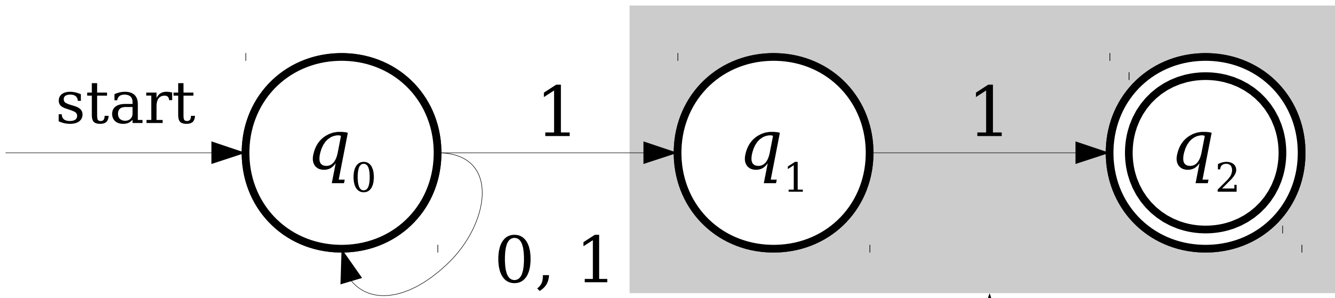
A Simple NFA



A More Complex NFA

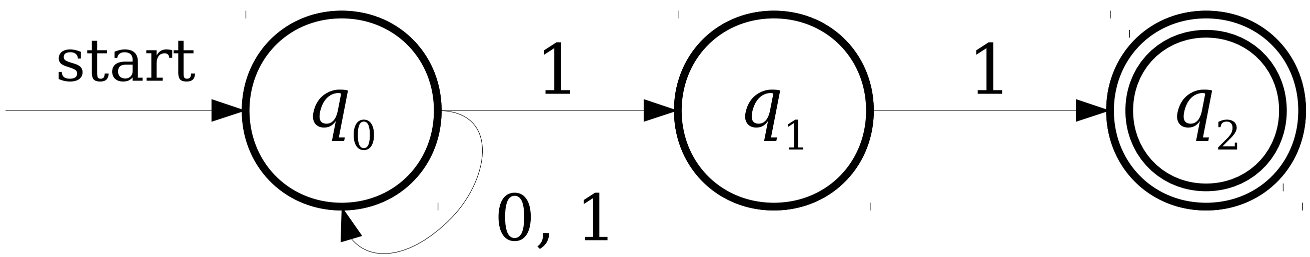


A More Complex NFA



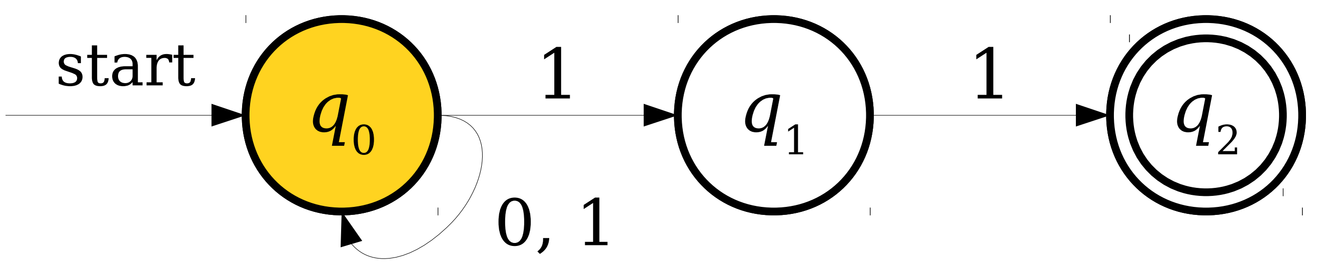
If a NFA needs to make a transition when no transition exists, the automaton dies and that particular path rejects.

A More Complex NFA



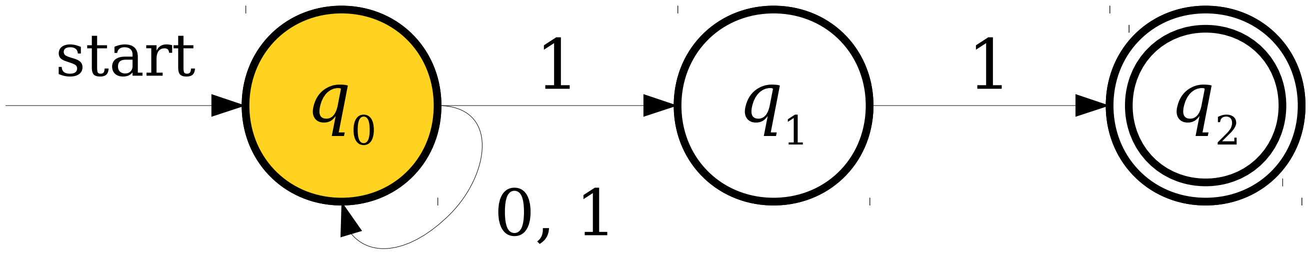
0 1 0 1 1

A More Complex NFA

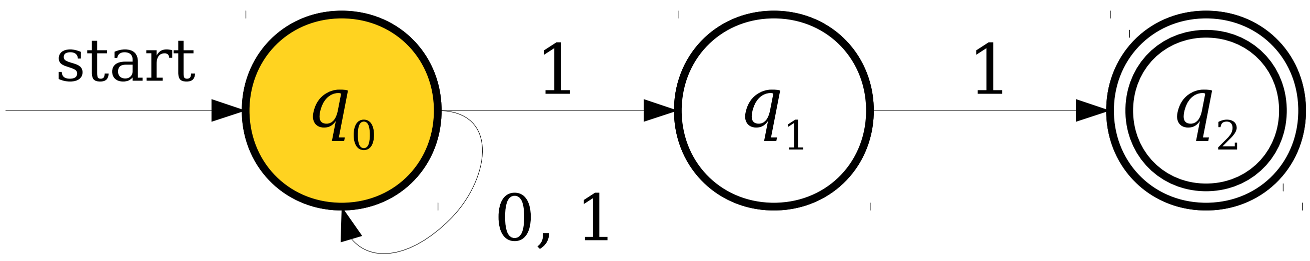


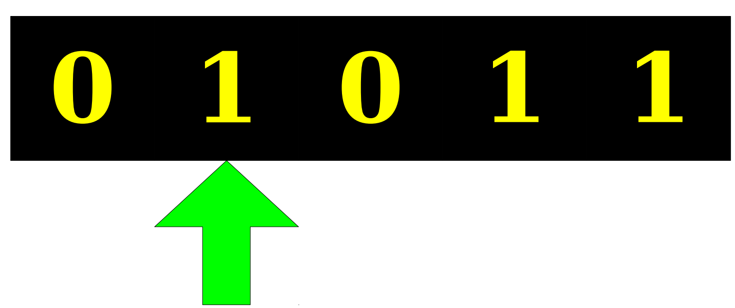
0 1 0 1 1

A More Complex NFA

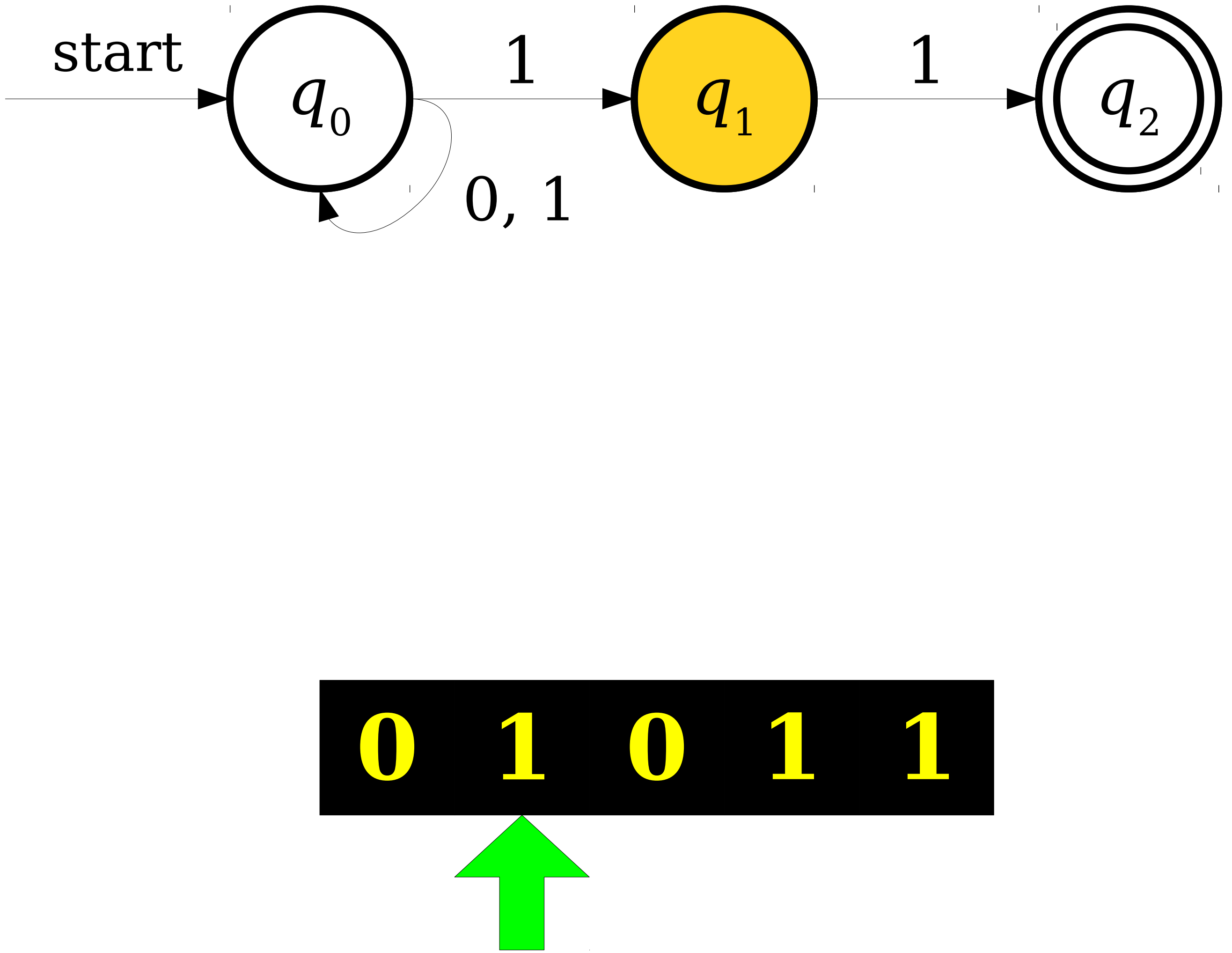


A More Complex NFA

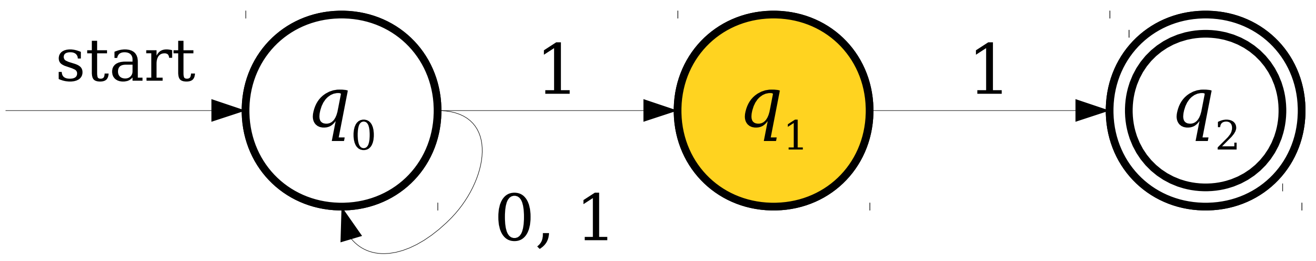


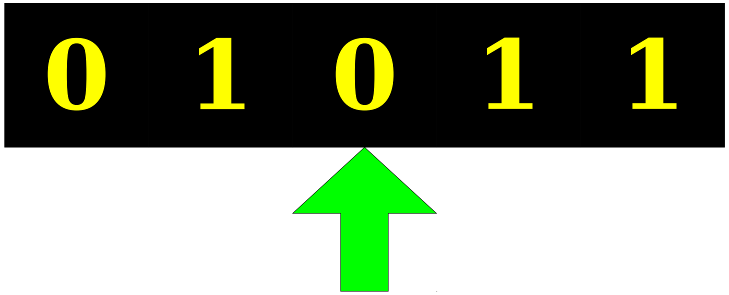


A More Complex NFA

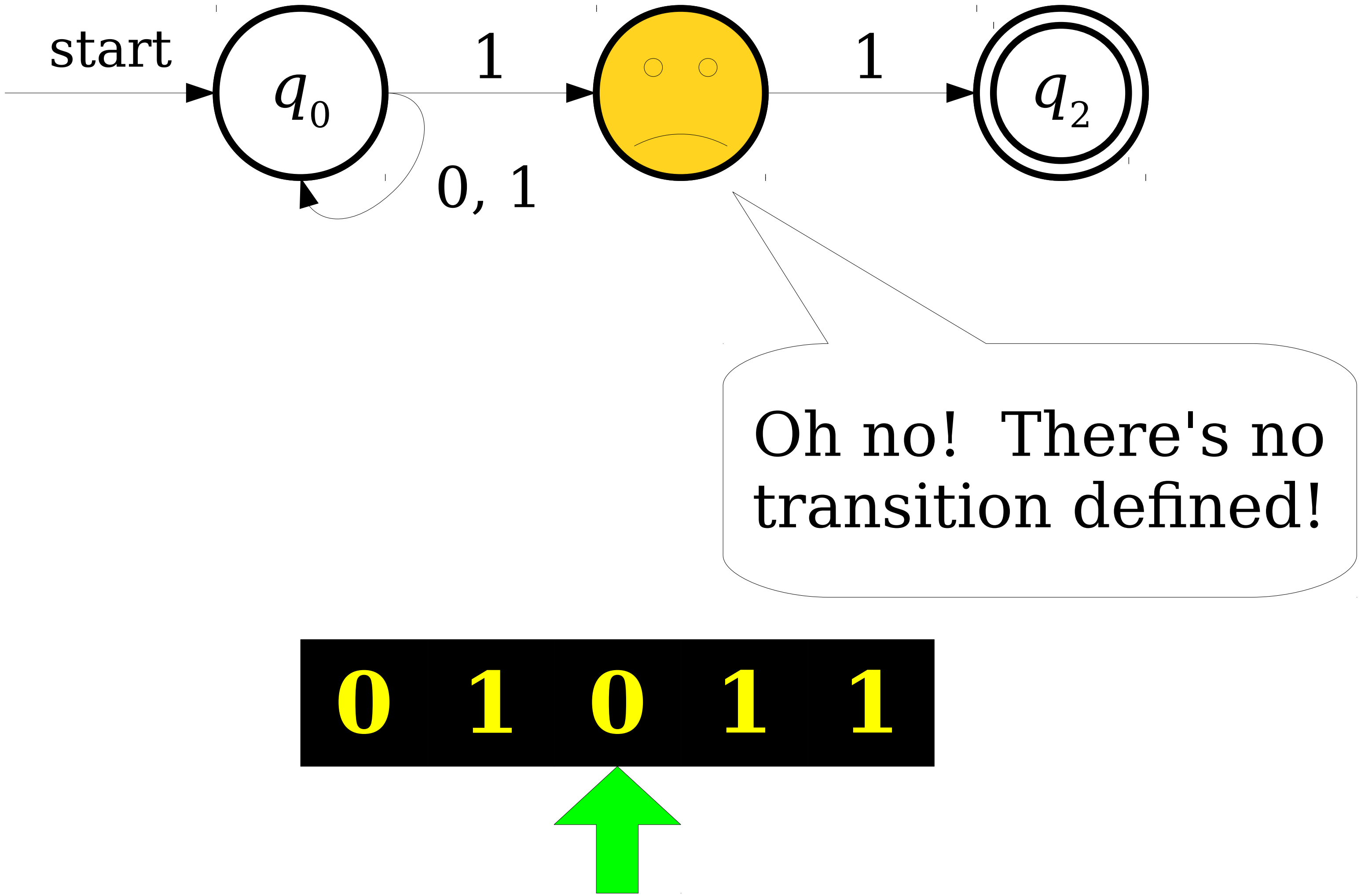


A More Complex NFA

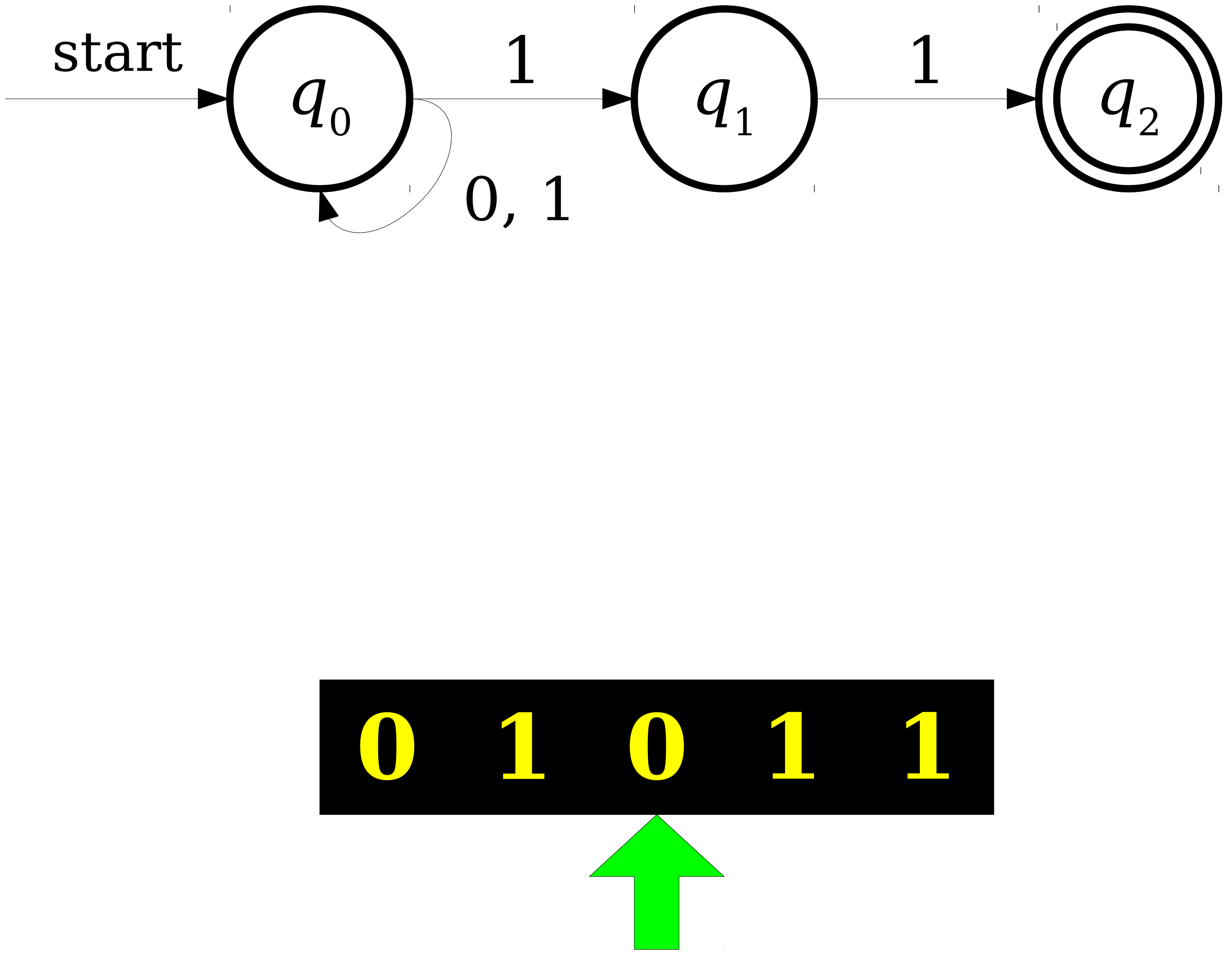




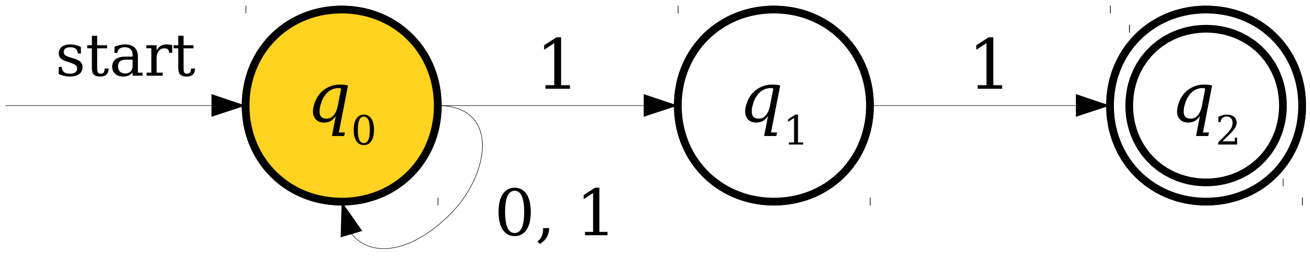
A More Complex NFA



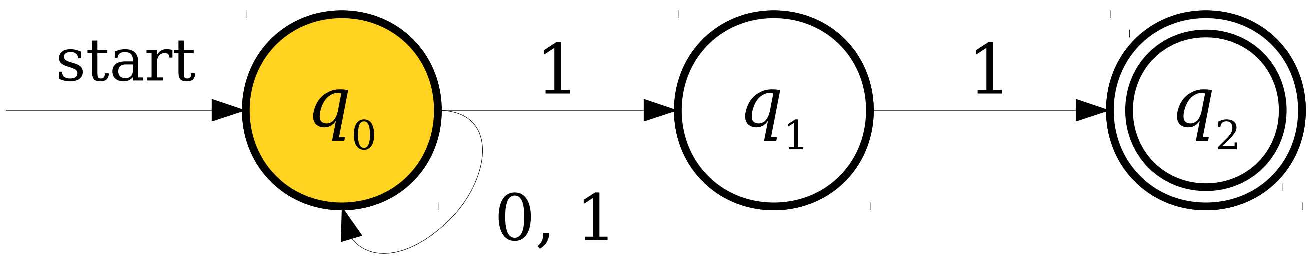
A More Complex NFA

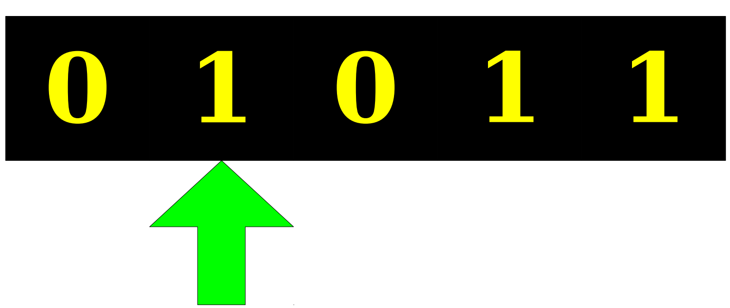


A More Complex NFA

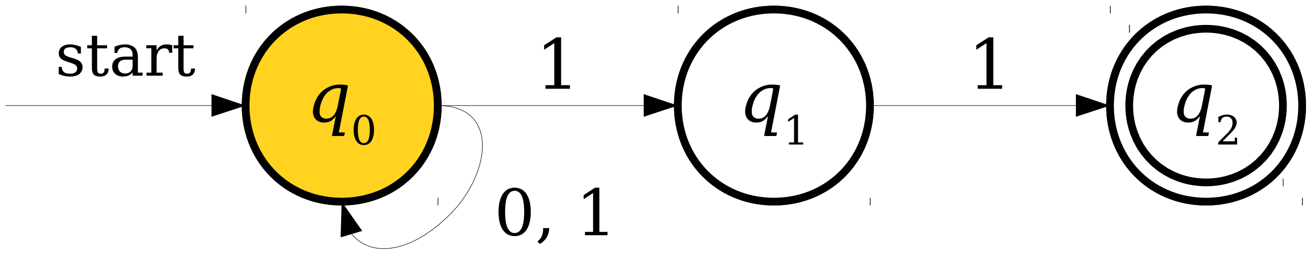


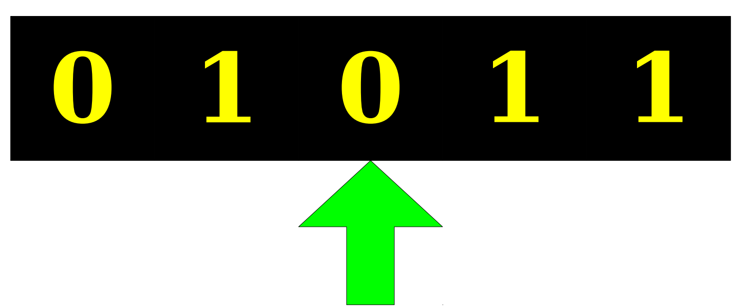
A More Complex NFA



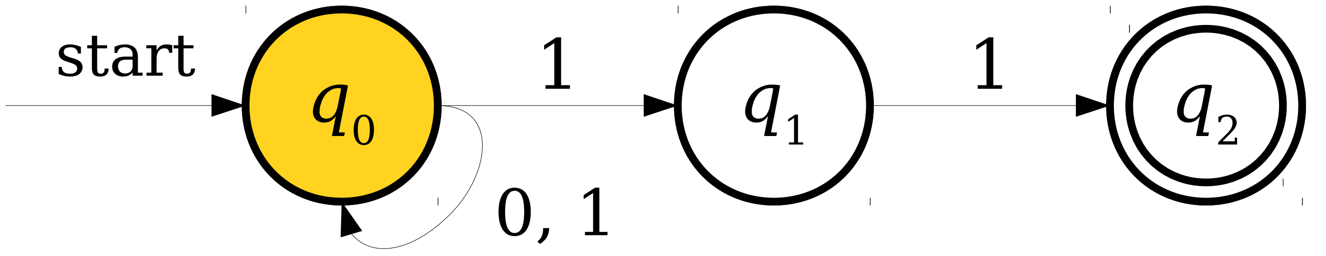


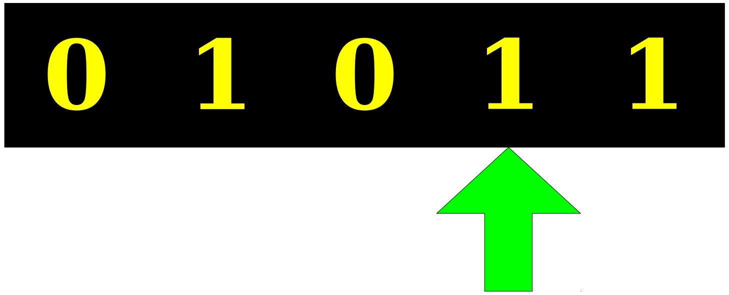
A More Complex NFA



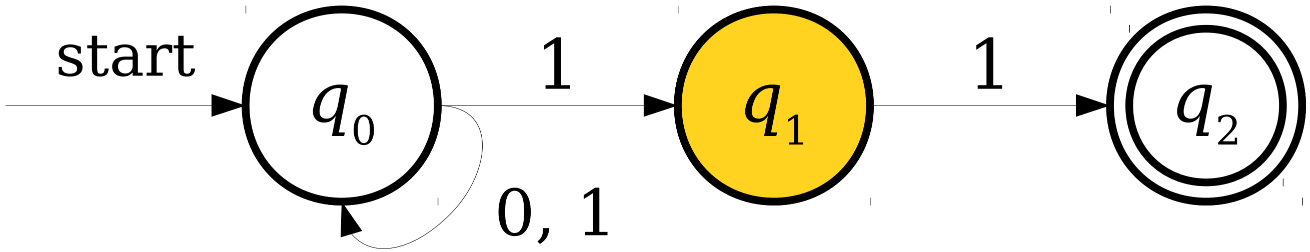


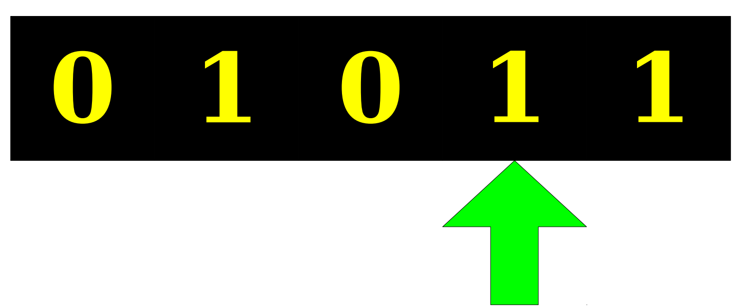
A More Complex NFA



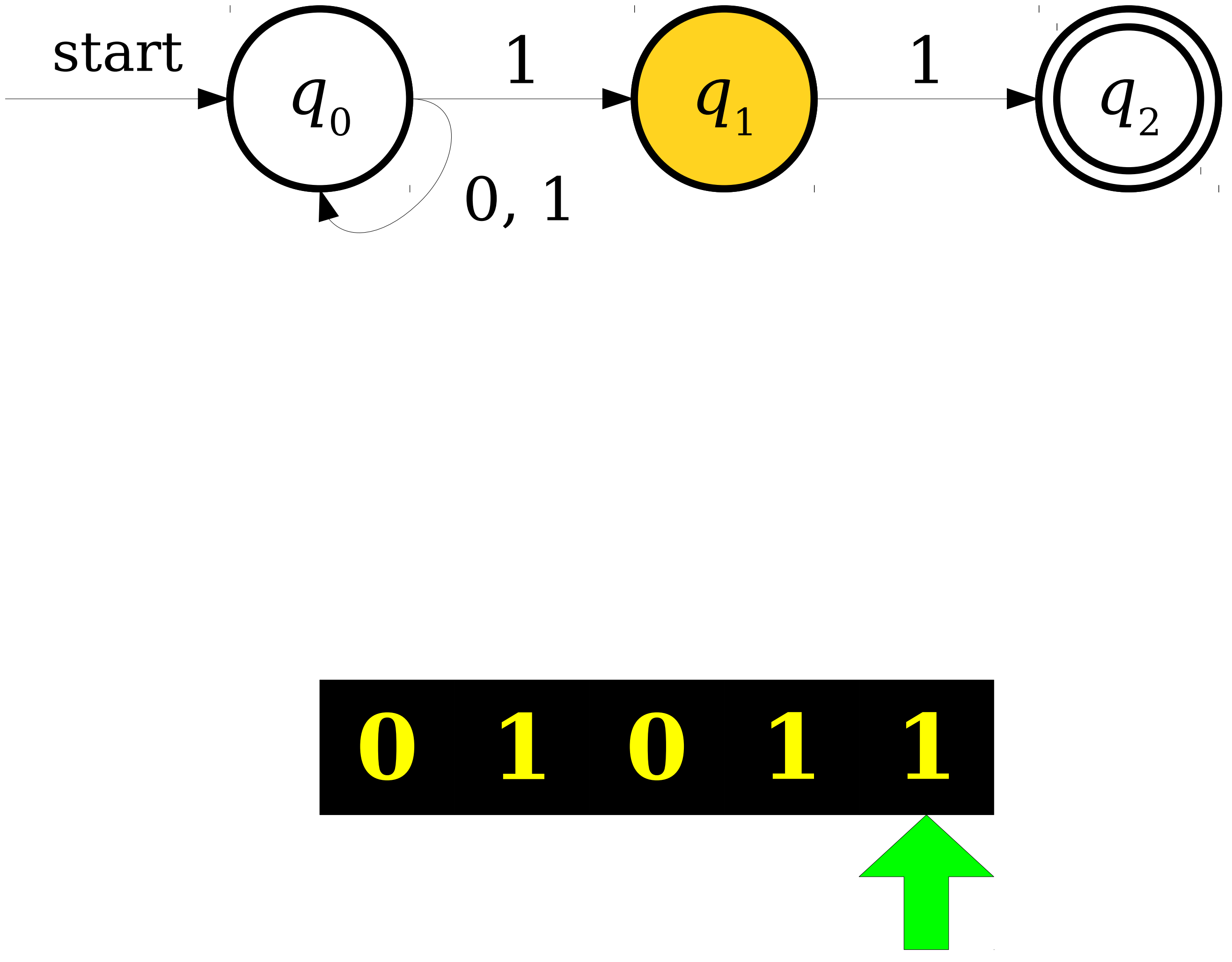


A More Complex NFA

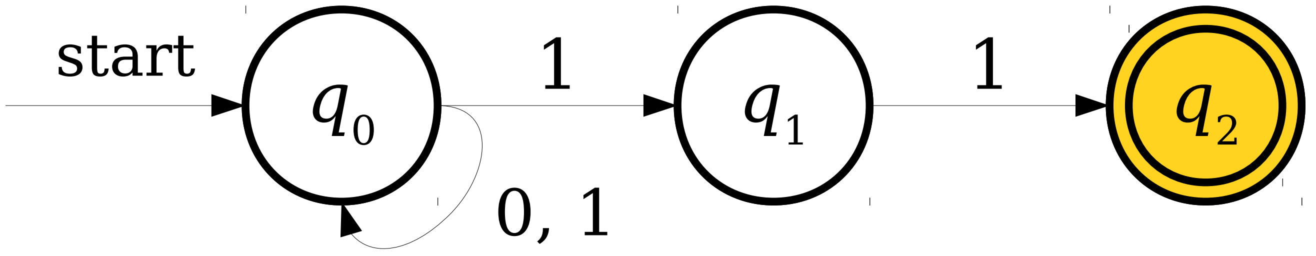




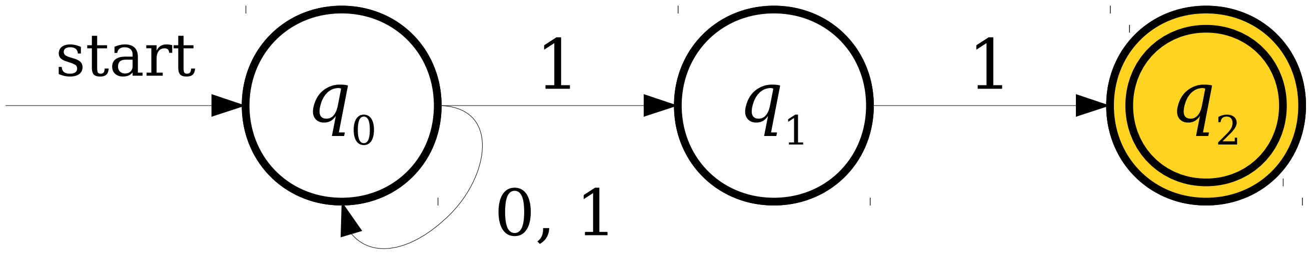
A More Complex NFA



A More Complex NFA

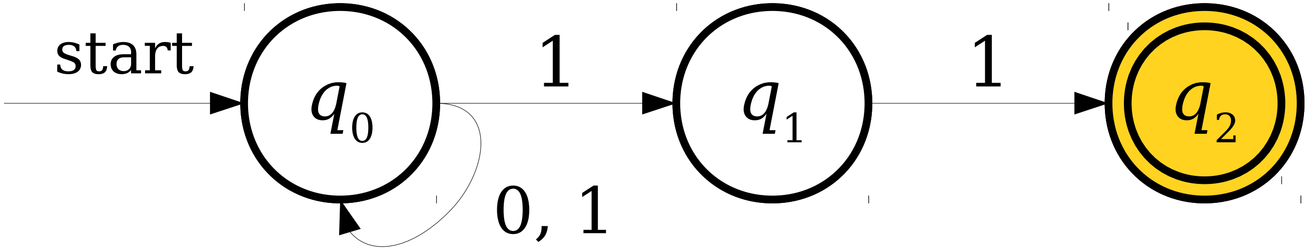


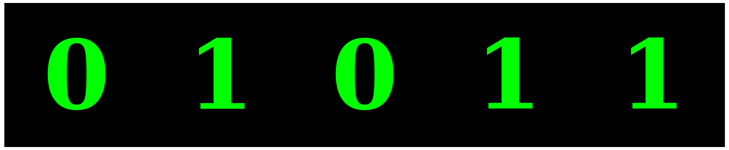
A More Complex NFA



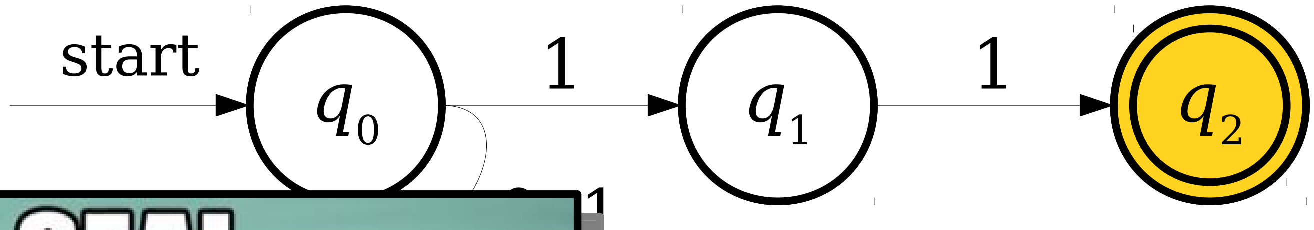
0 1 0 1 1

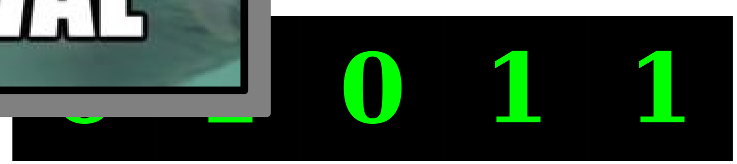
A More Complex NFA





A More Complex NFA





NFA Acceptance

- An NFA N accepts a string w if there is some series of choices that lead to an accepting state.
- Let $LeadsToAccept(N, c, w)$ mean "the series of choices c takes N into an accept state when run on w."
- Then

$$\boldsymbol{N \text{ accepts } w \leftrightarrow \exists c.\ LeadsToAccept(N, c, w)}$$

- Consequently,

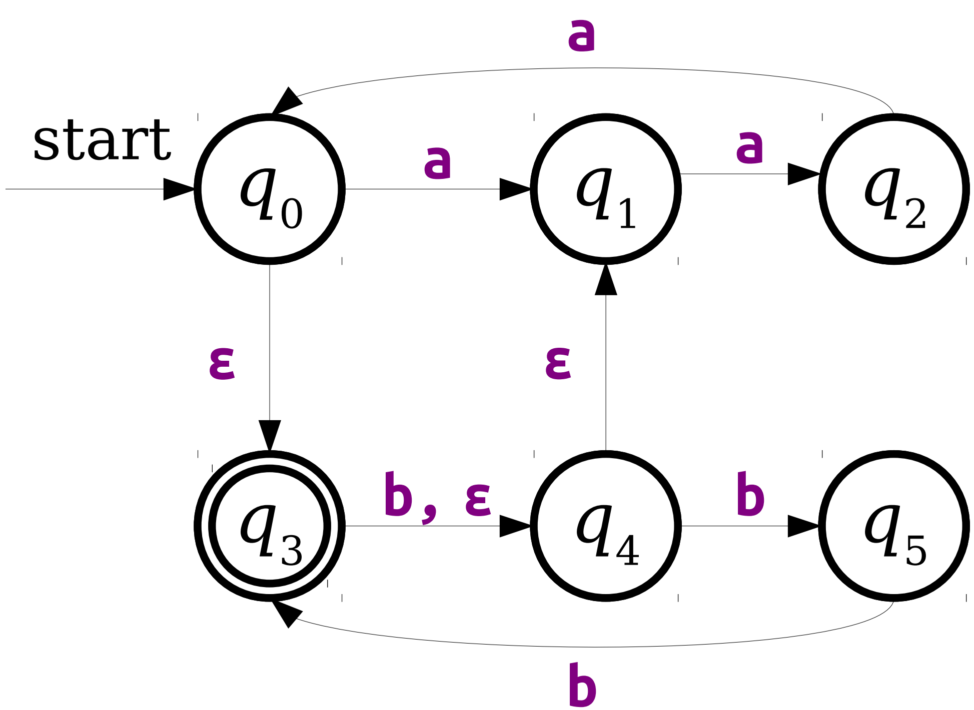
$$\boldsymbol{N \text{ rejects } w \leftrightarrow \forall c.\ \neg LeadsToAccept(N, c, w)}$$

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.

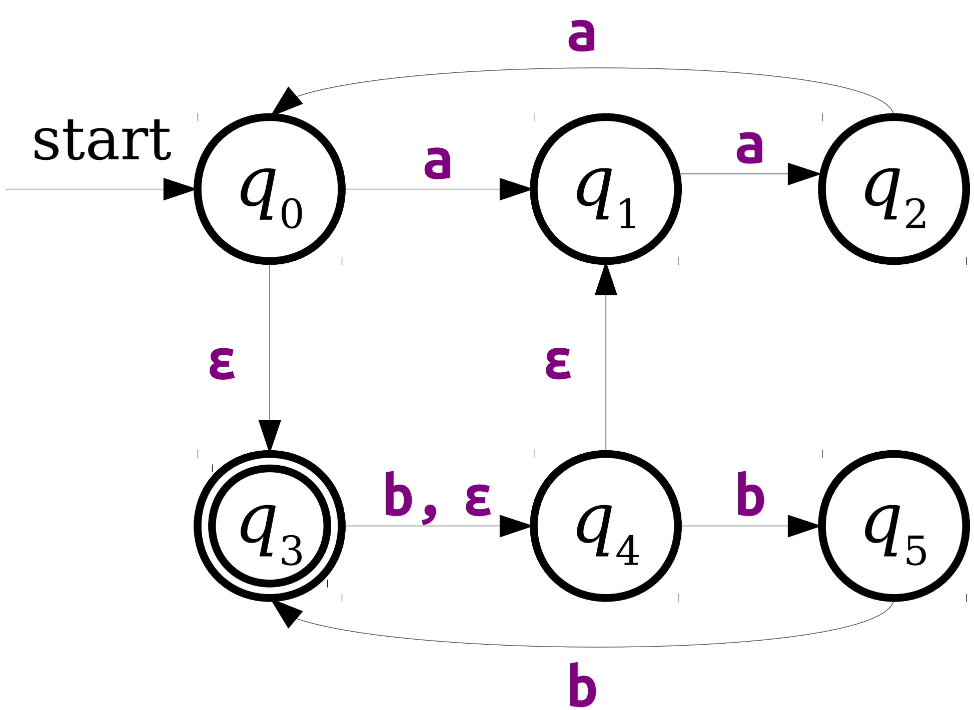
ε-Transitions

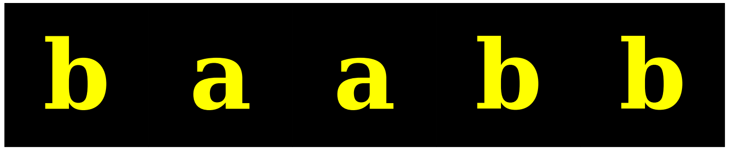
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



ε-Transitions

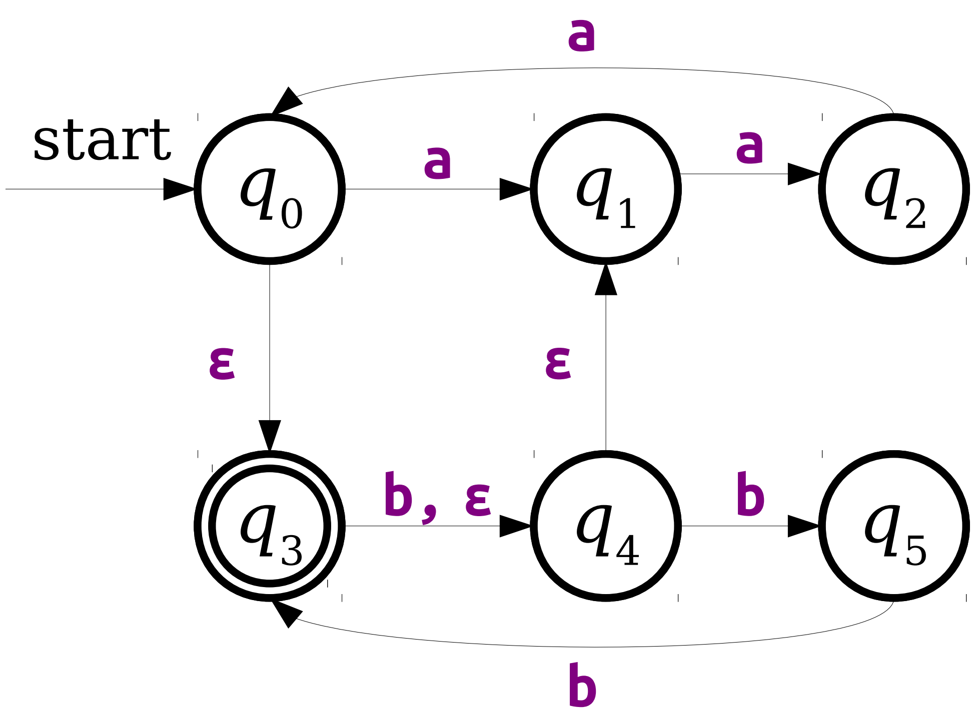
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.





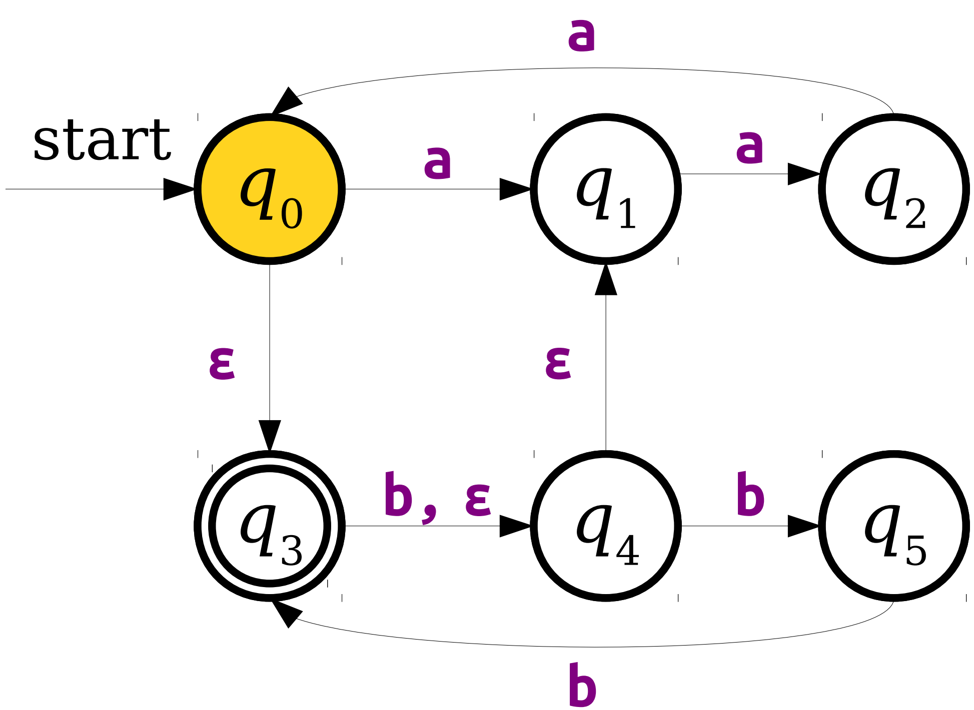
ε-Transitions

- 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.



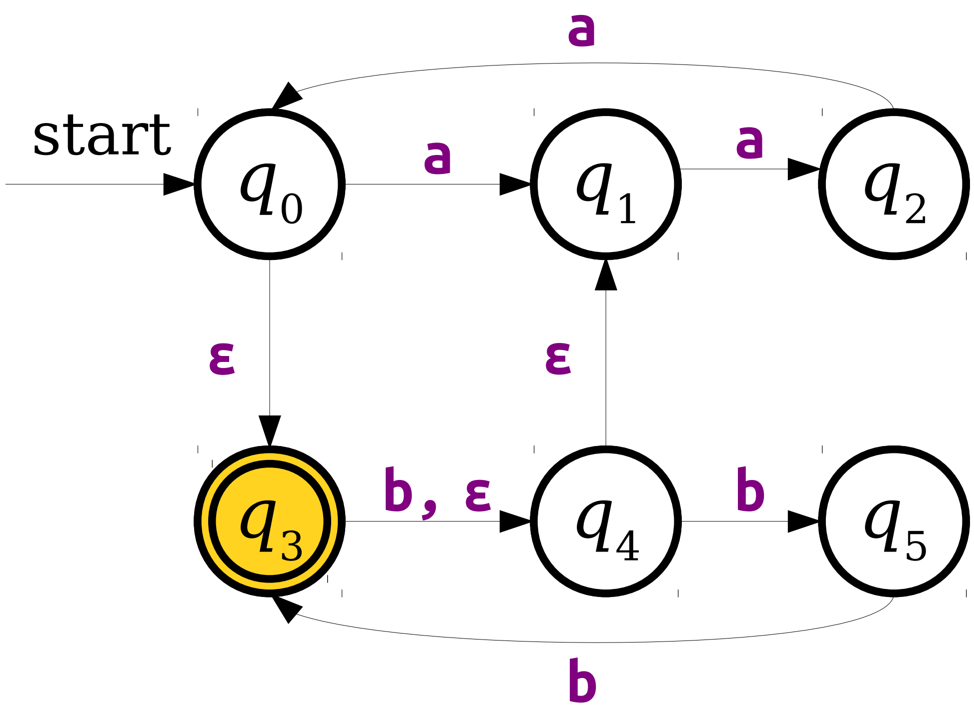
ε-Transitions

- 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.



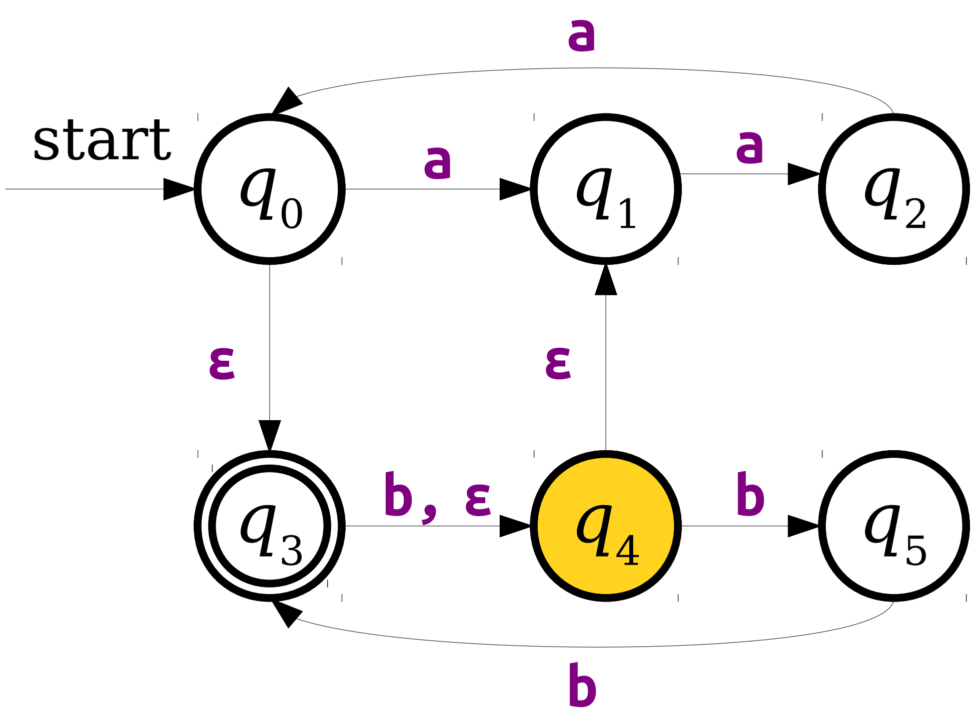
ε-Transitions

- 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.



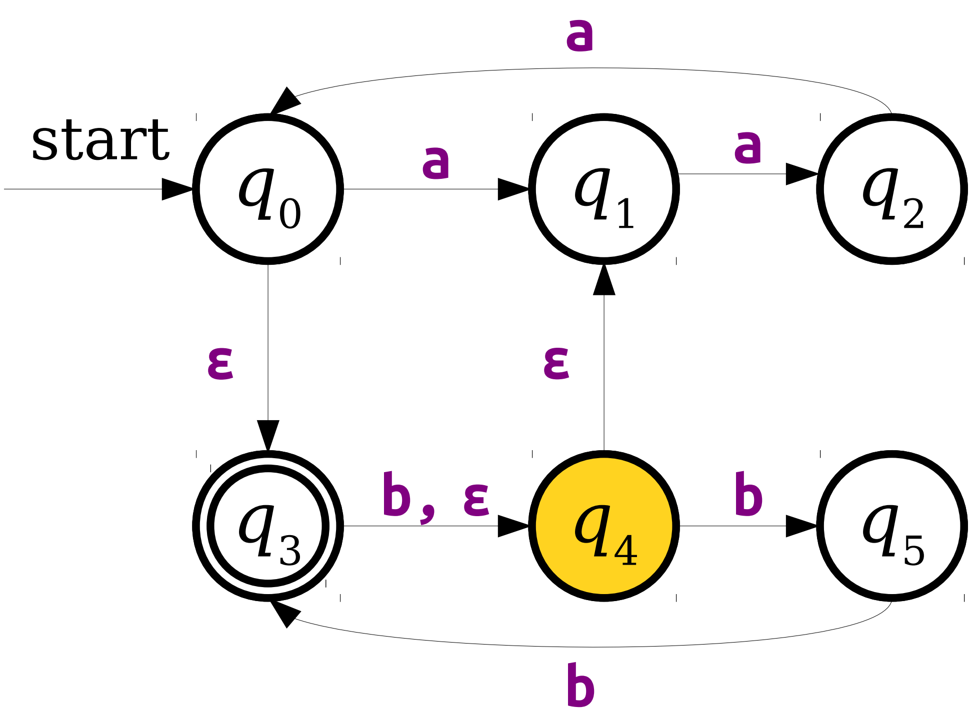
ε-Transitions

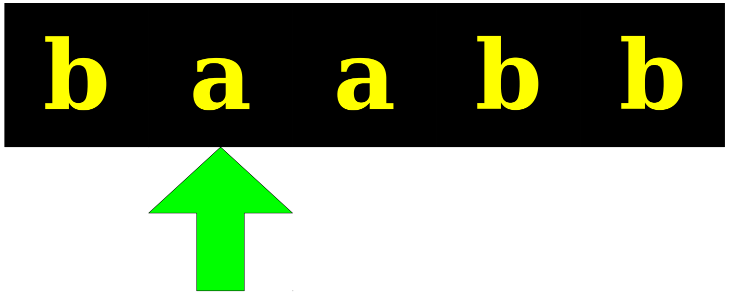
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



ε-Transitions

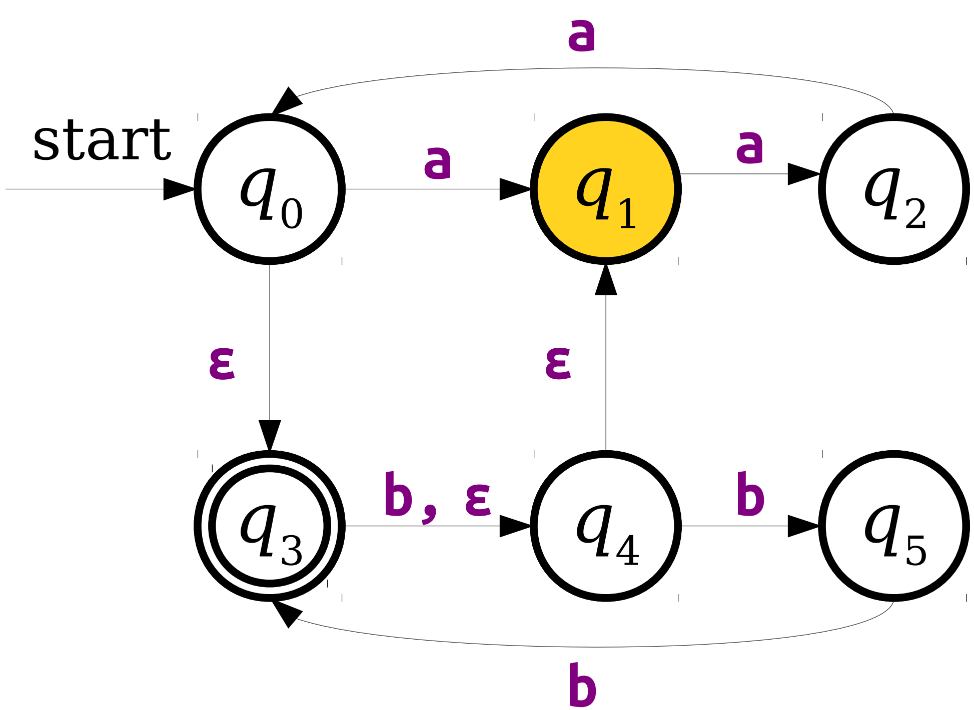
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

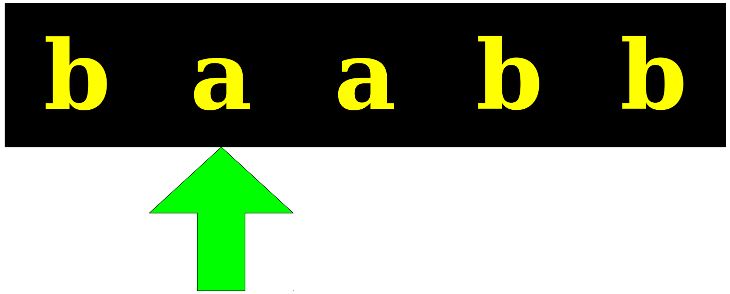




ε-Transitions

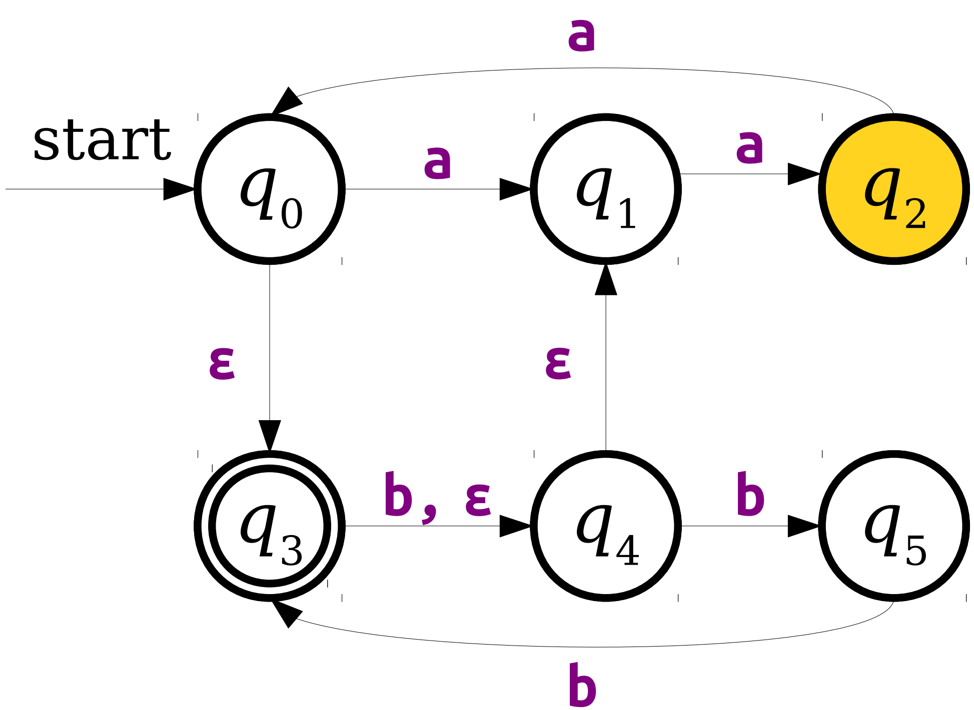
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

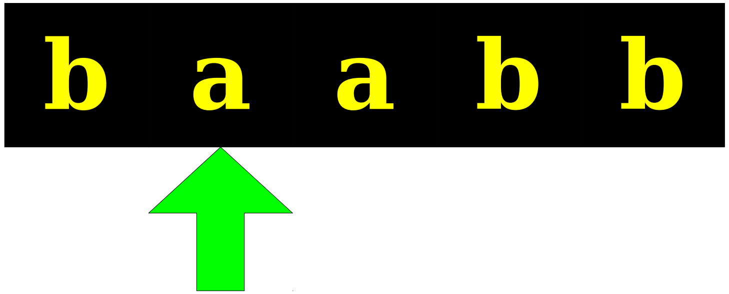




ε-Transitions

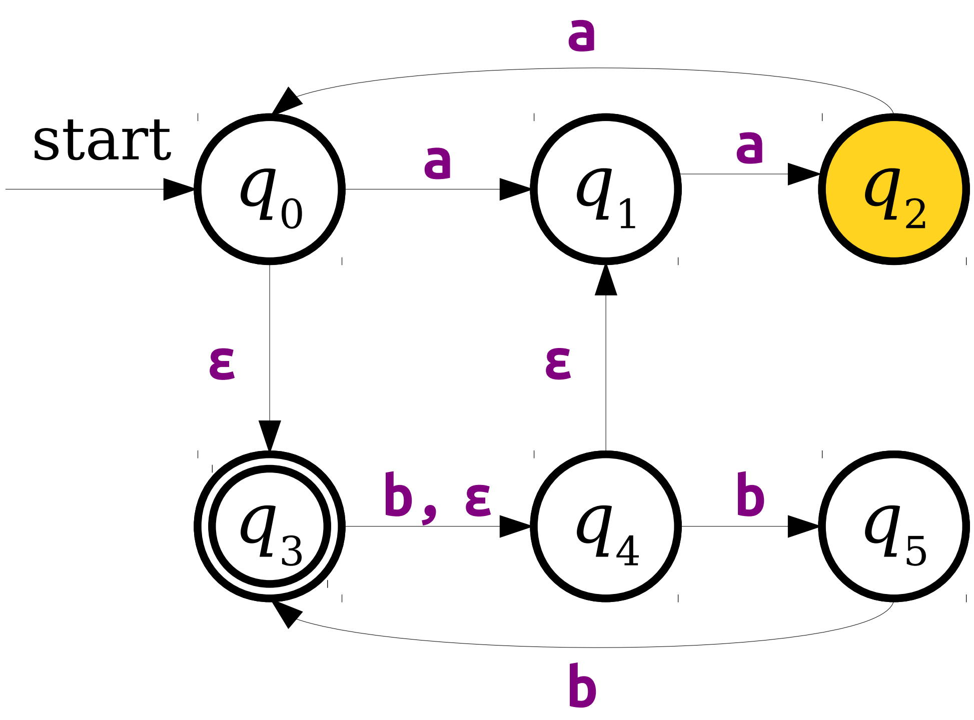
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

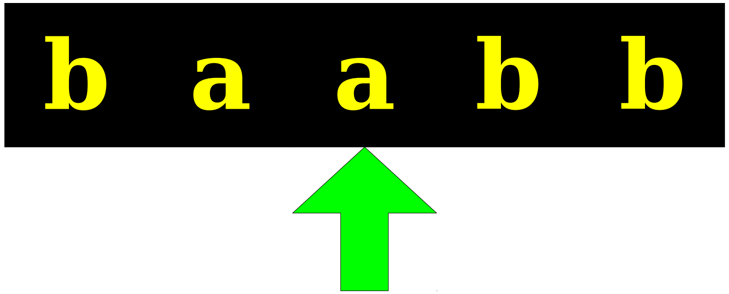




ε-Transitions

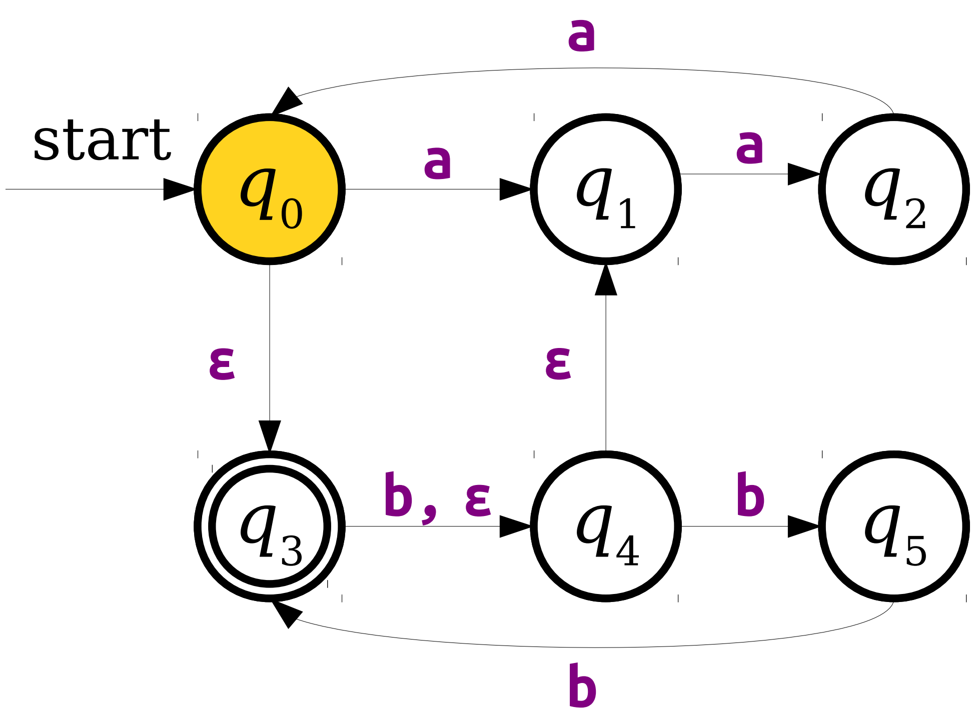
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

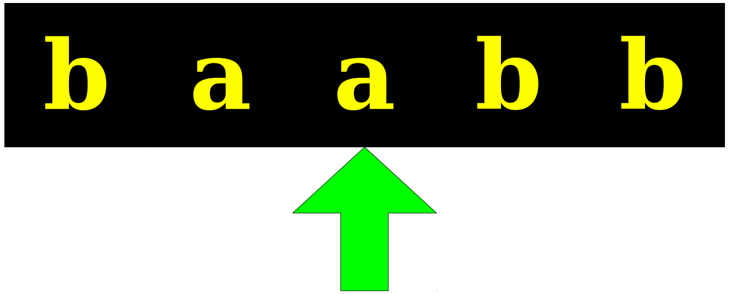




ε-Transitions

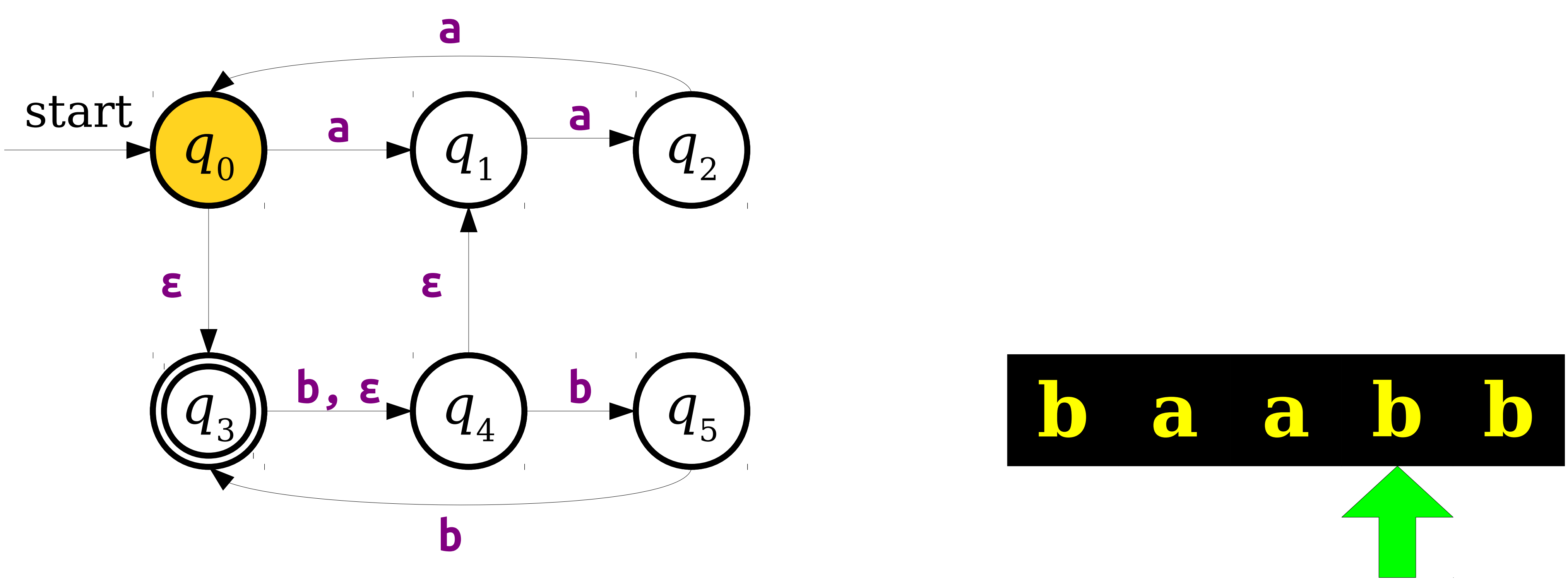
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.





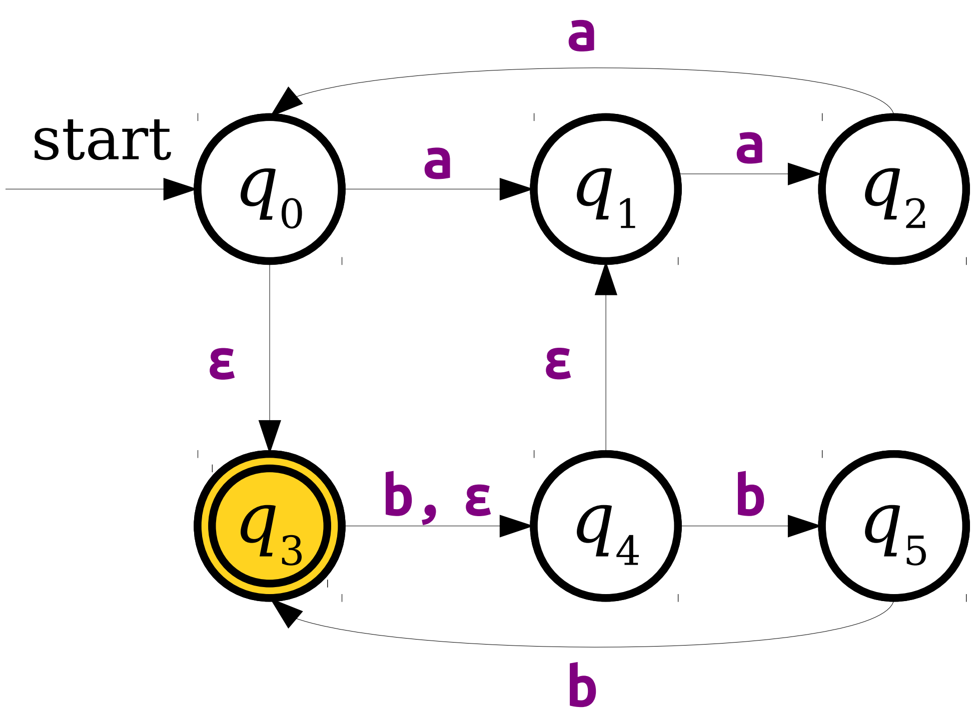
ε-Transitions

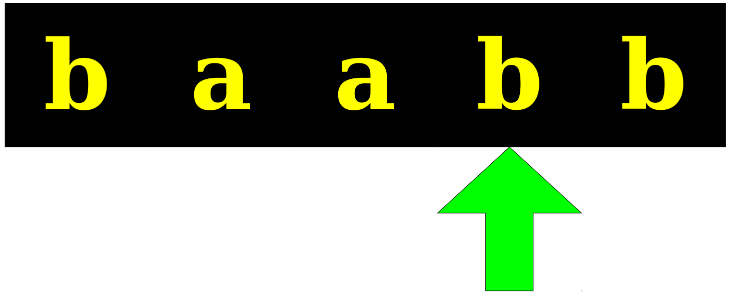
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



ε-Transitions

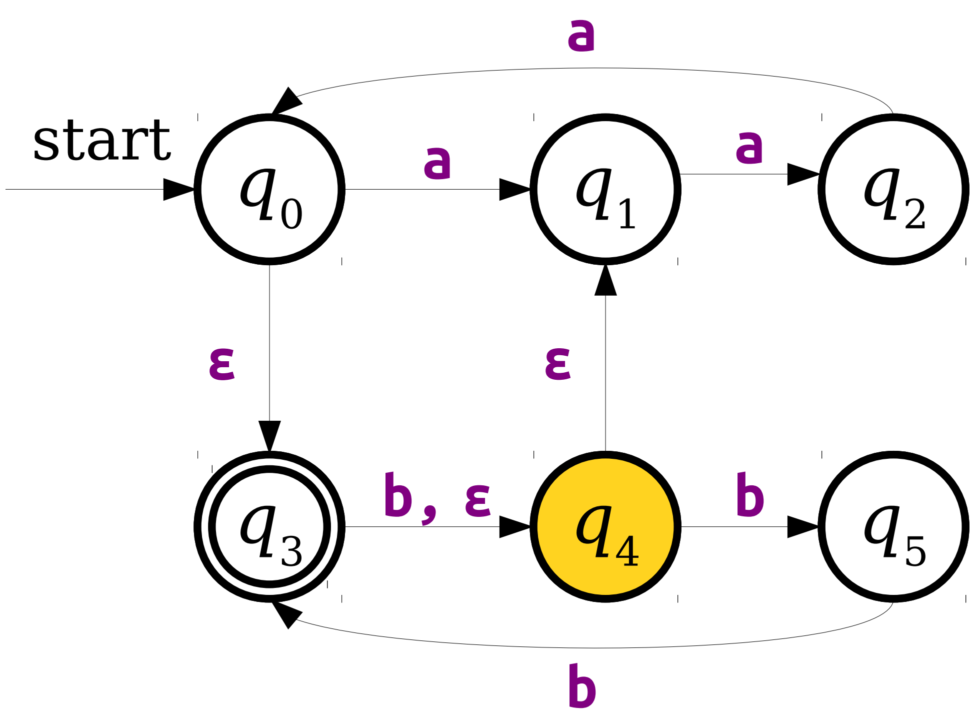
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

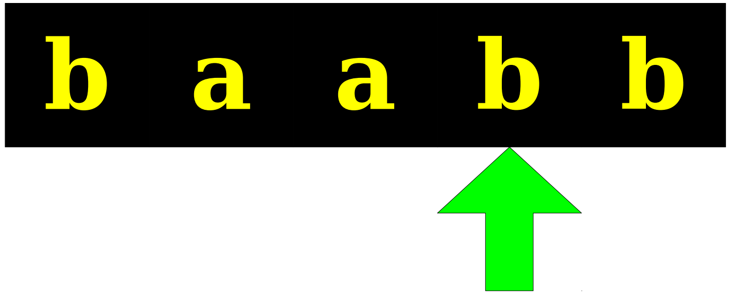




ε-Transitions

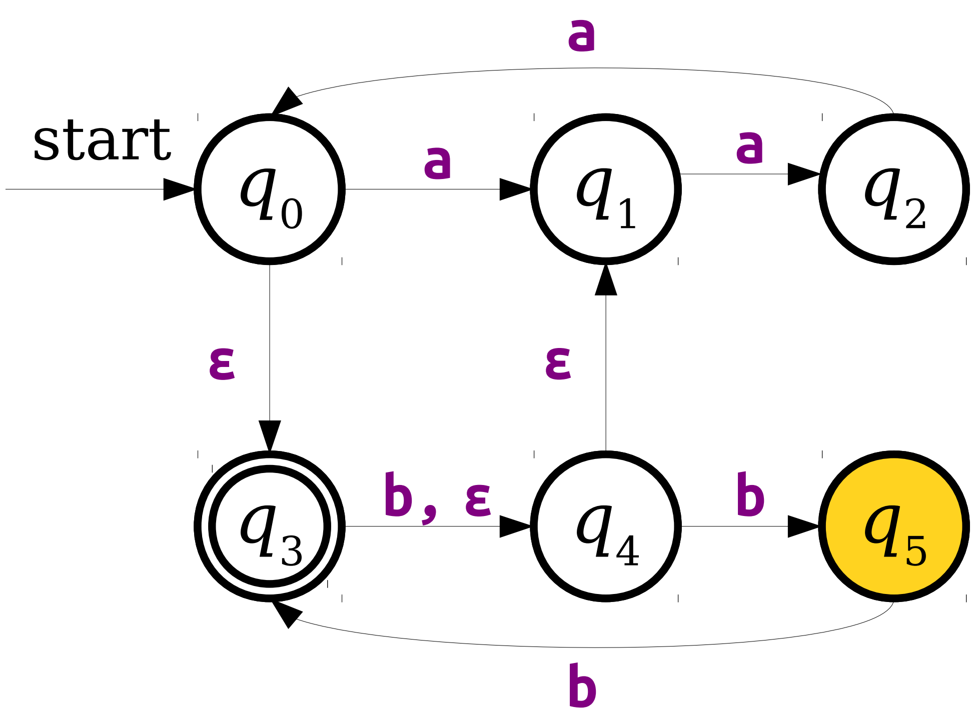
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

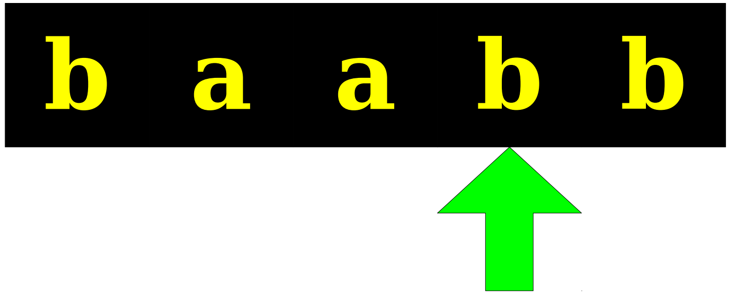




ε-Transitions

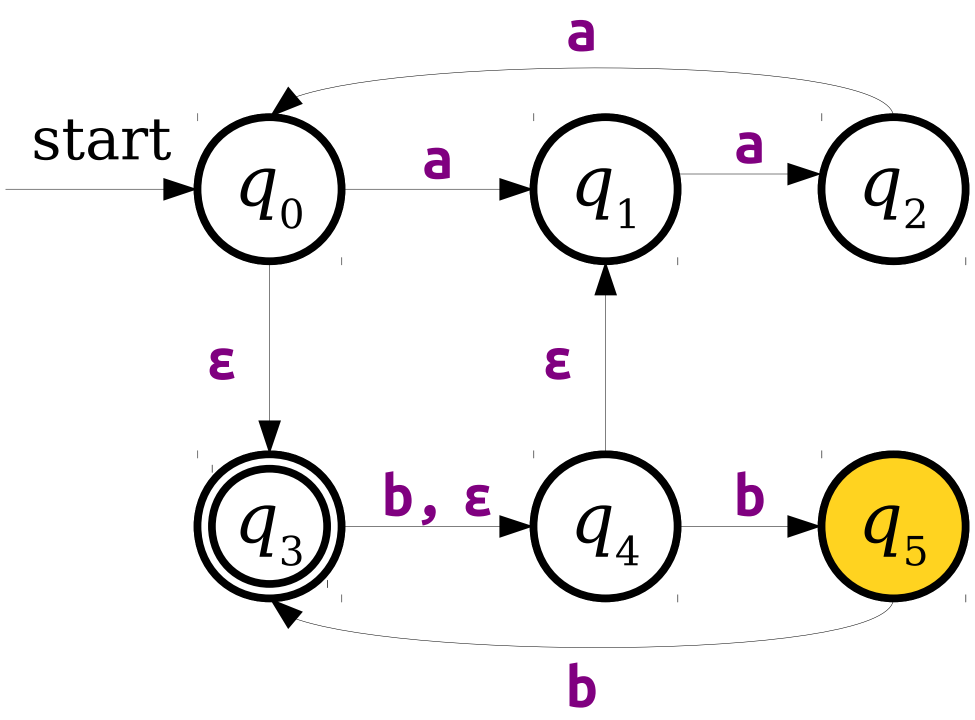
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.





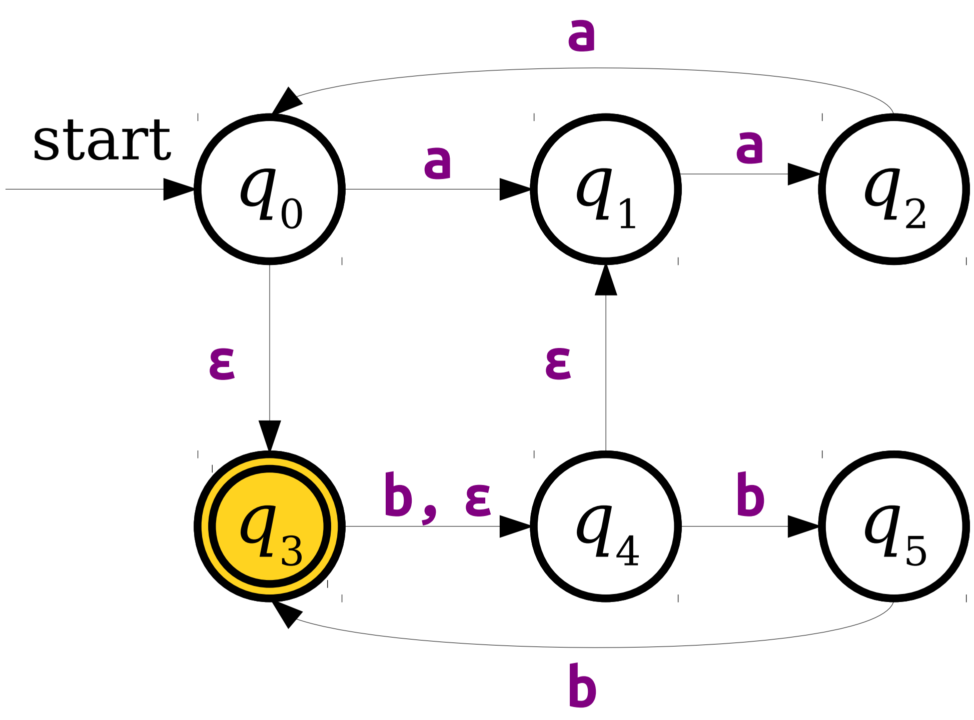
ε-Transitions

- 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.



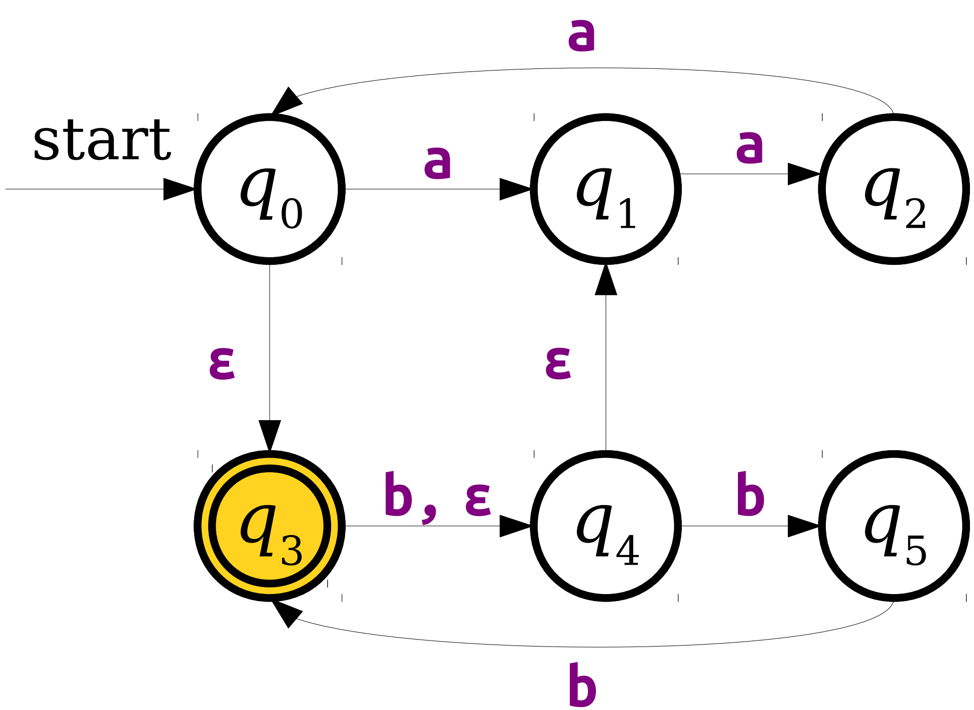
ε-Transitions

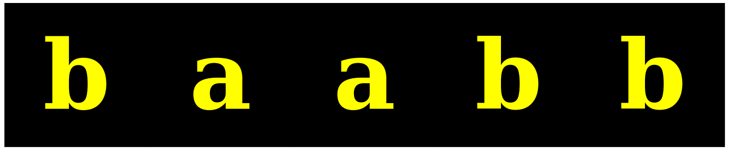
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



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.





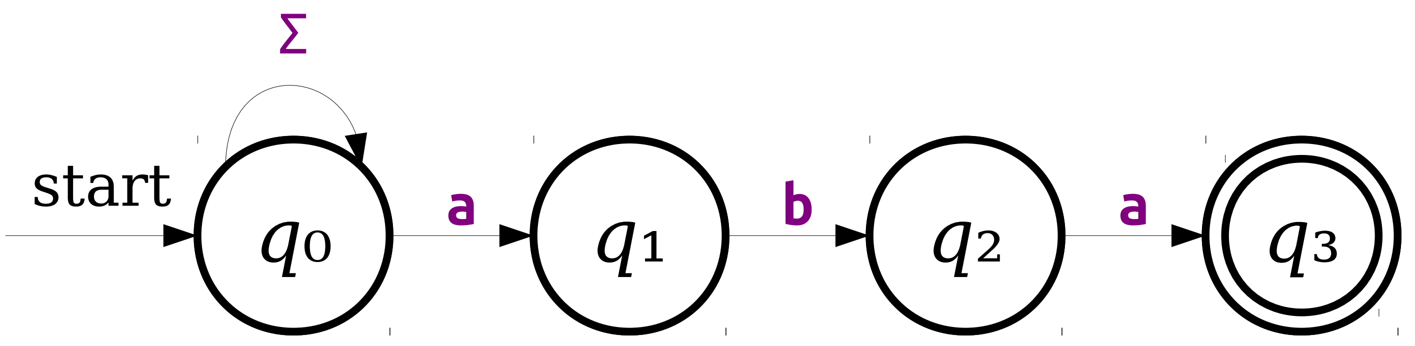
ε-Transitions

- 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.
- NFAs are not *required* to follow ε-transitions. It's simply another option at the machine's disposal.

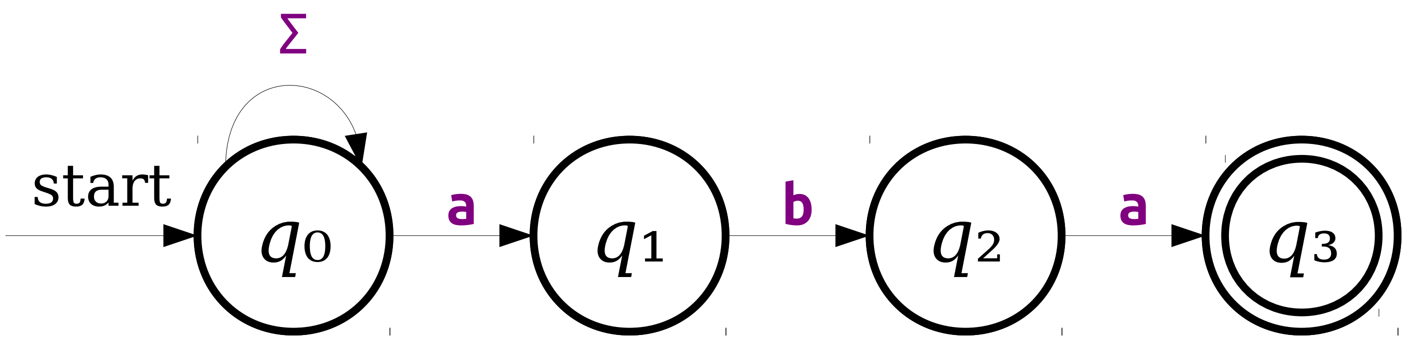
Intuiting Nondeterminism

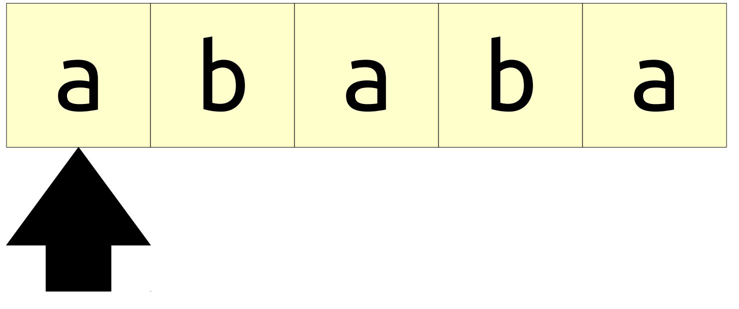
- Nondeterministic machines are a serious departure from physical computers. How can we build up an intuition for them?
- There are two particularly useful frameworks for interpreting nondeterminism:
  - **Perfect guessing**
  - **Massive parallelism**

Perfect Guessing

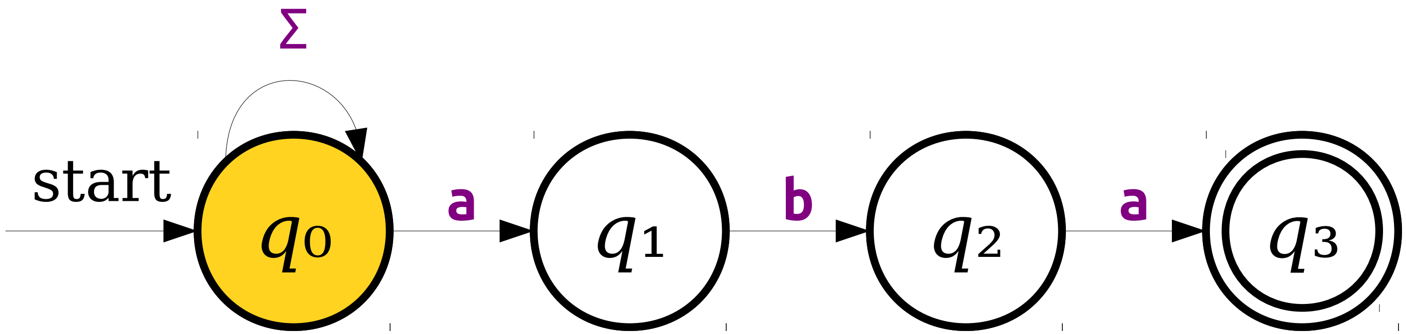


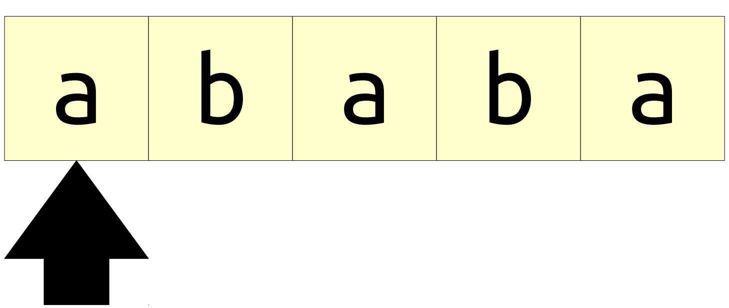
Perfect Guessing



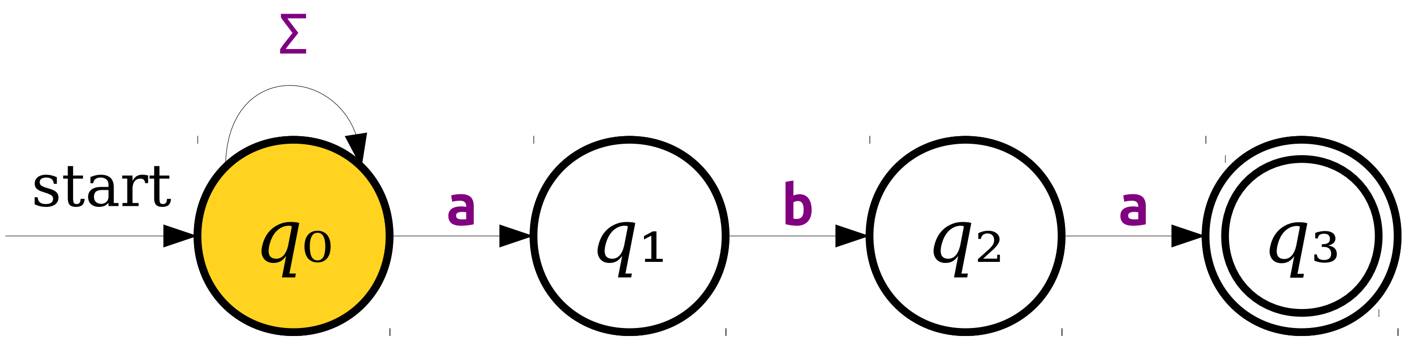


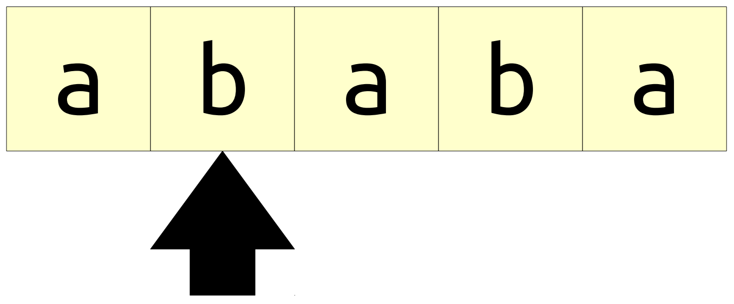
Perfect Guessing



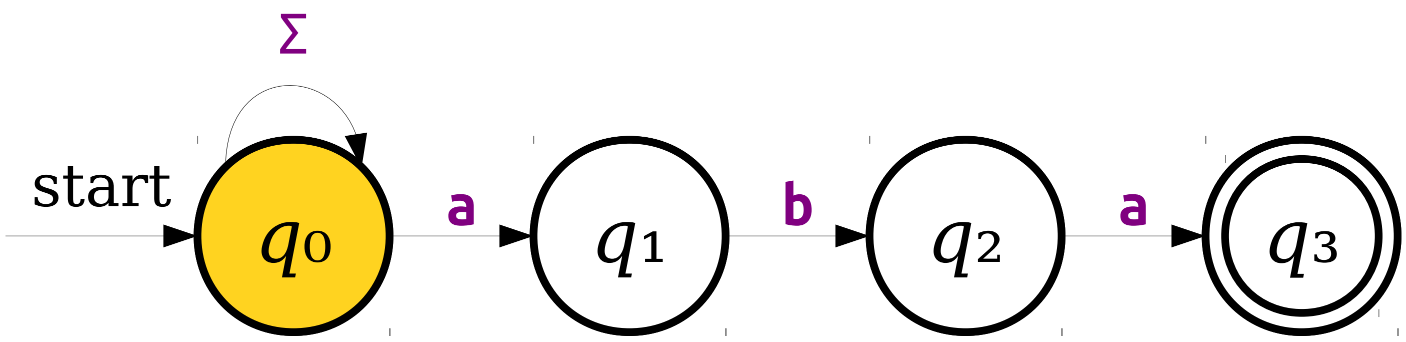


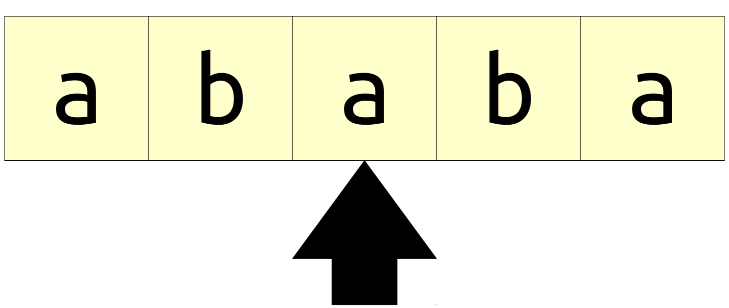
Perfect Guessing



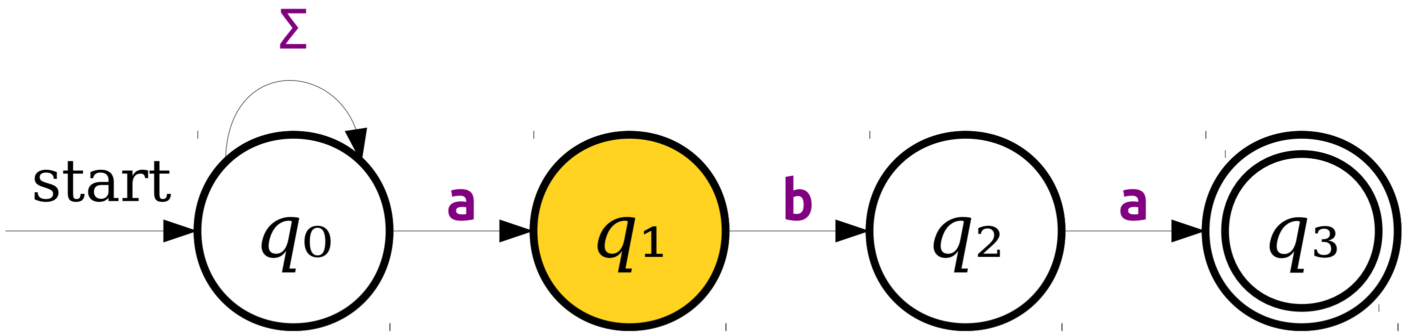


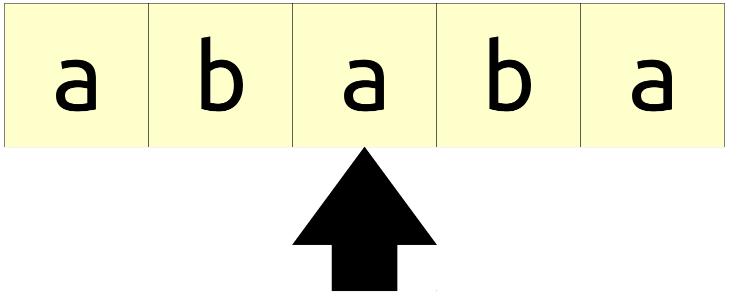
Perfect Guessing



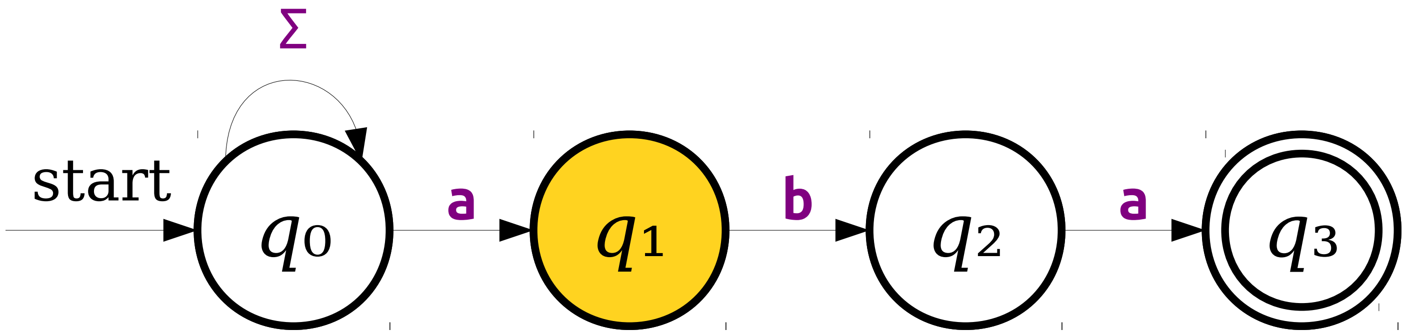


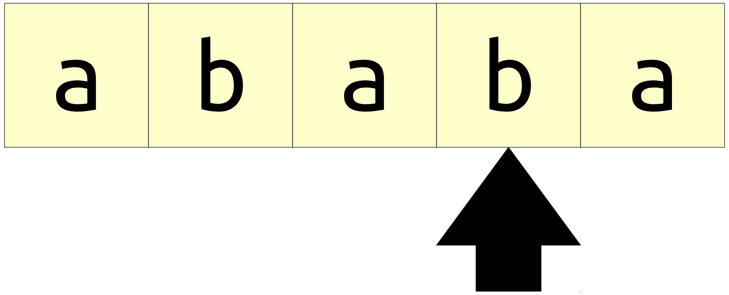
Perfect Guessing



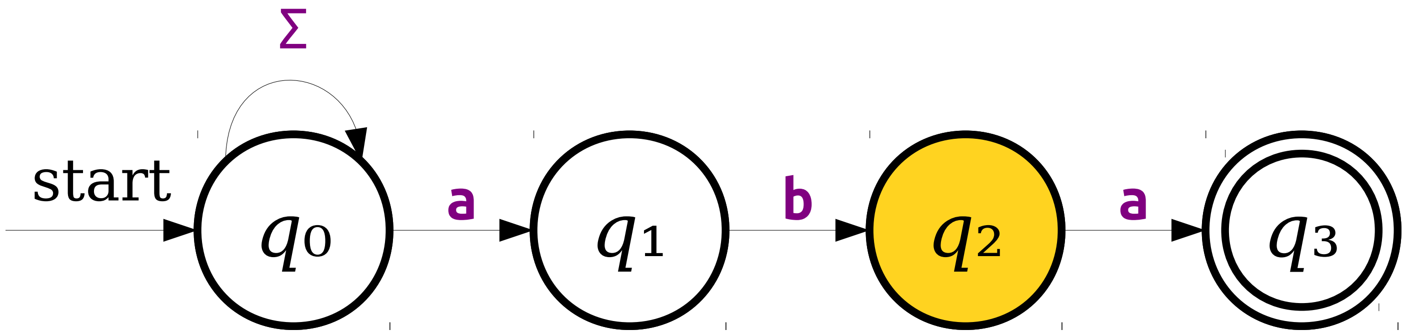


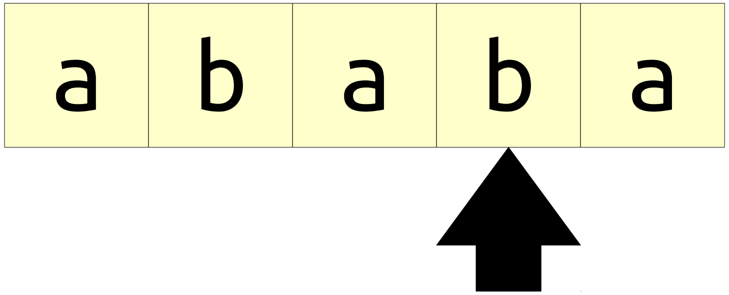
Perfect Guessing



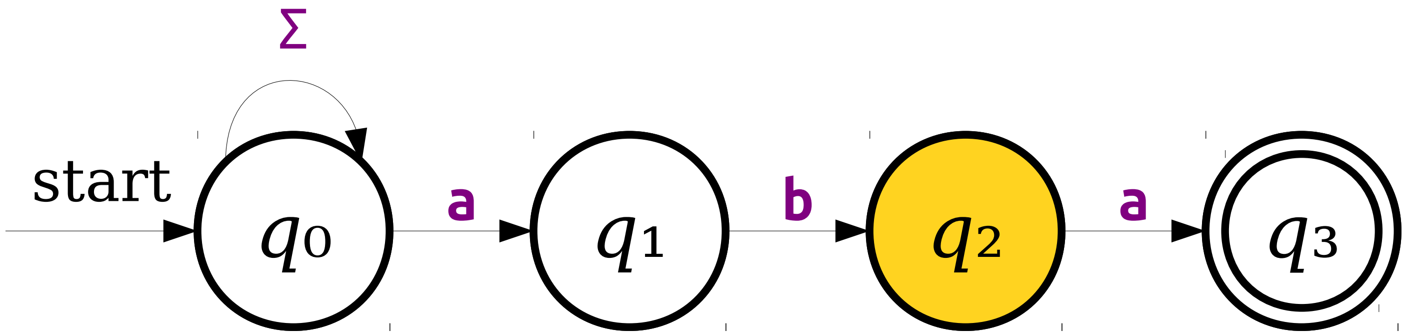


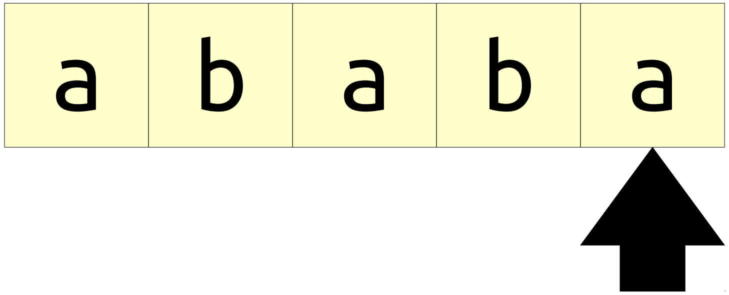
Perfect Guessing



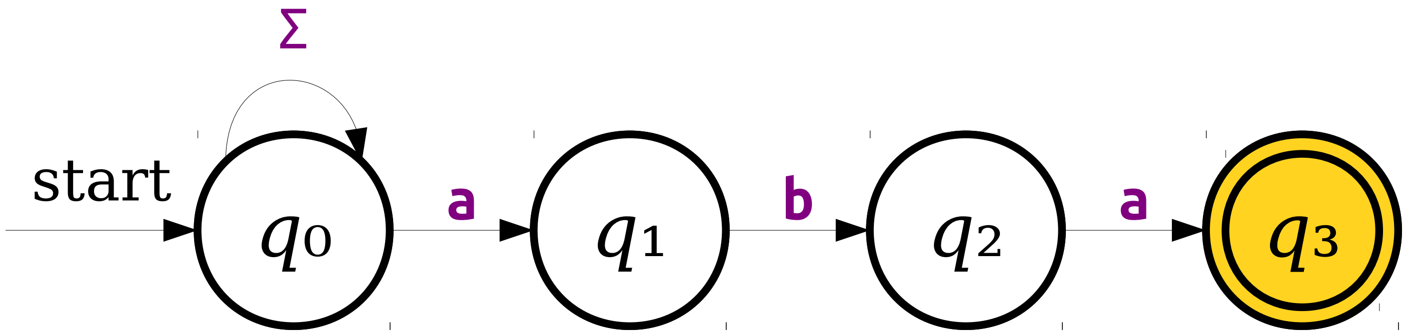


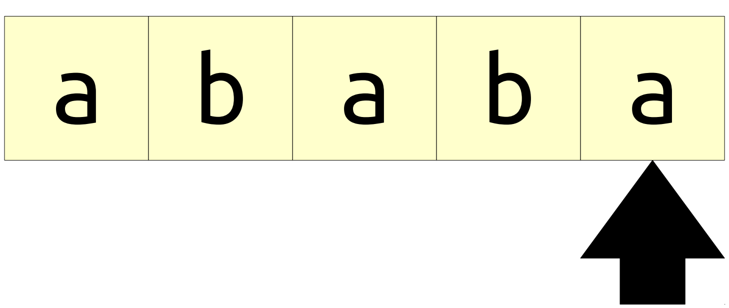
Perfect Guessing



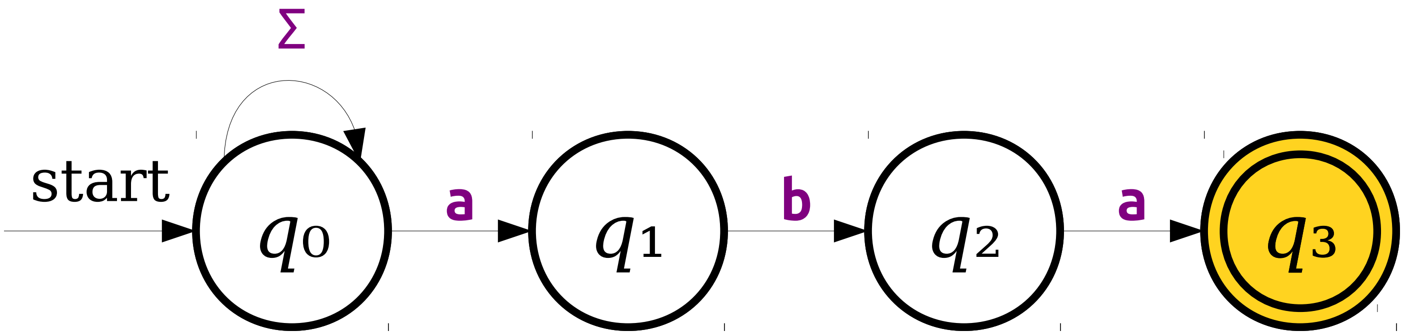


Perfect Guessing





Perfect Guessing

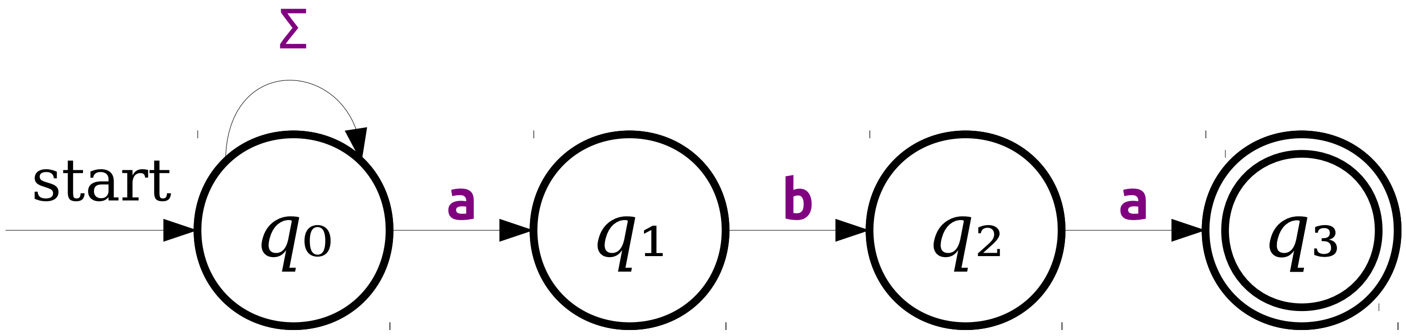


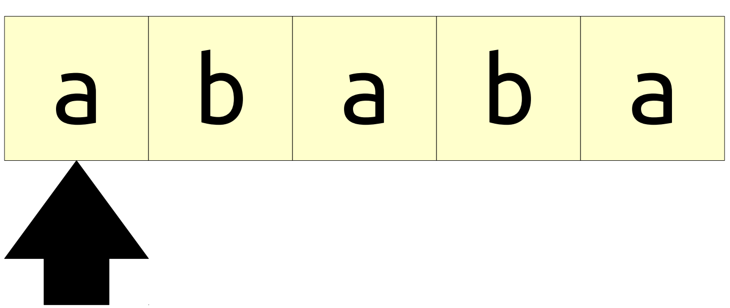
a	b	a	b	a

Perfect Guessing

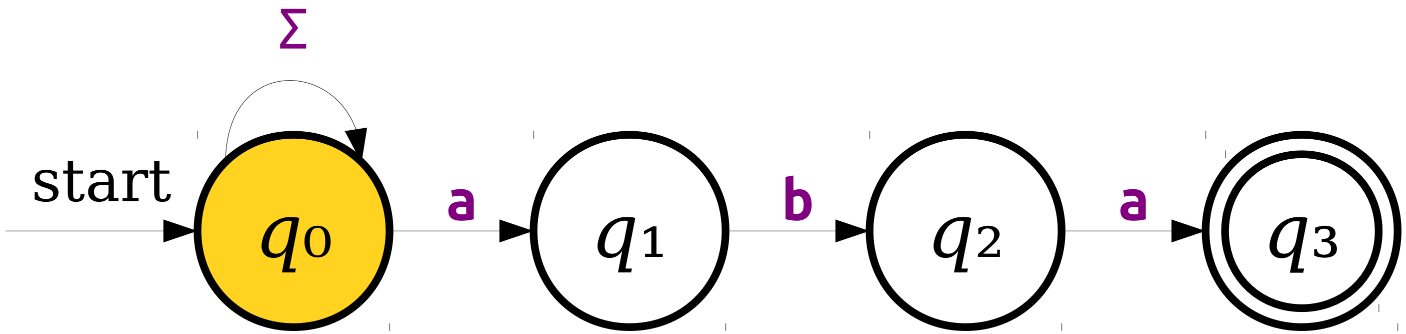
- We can view nondeterministic machines as having ***Magic Superpowers*** that enable them to guess the correct choice of moves to make.
  - If there is at least one choice that leads to an accepting state, the machine will guess it.
  - If there are no choices, the machine guesses any one of the wrong guesses.
- No known physical analog for this style of computation – this is totally new!

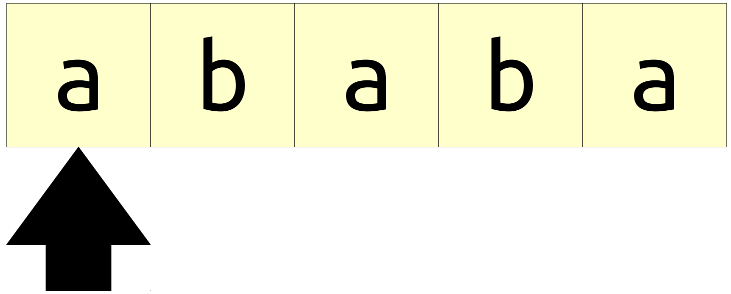
Massive Parallelism



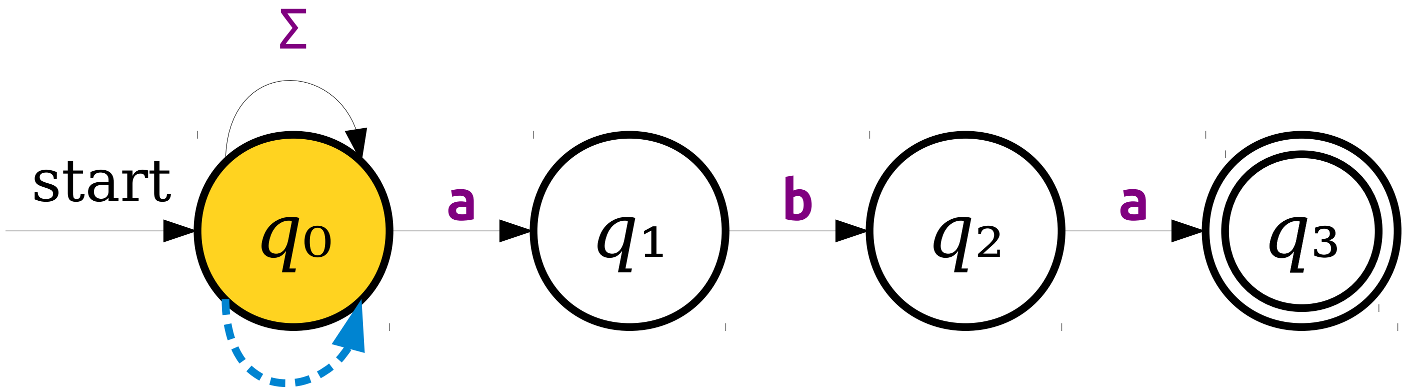


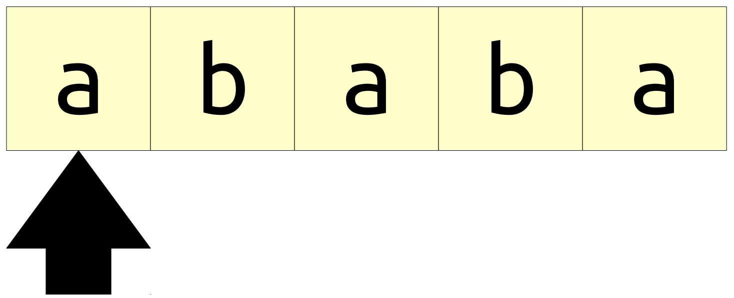
Massive Parallelism



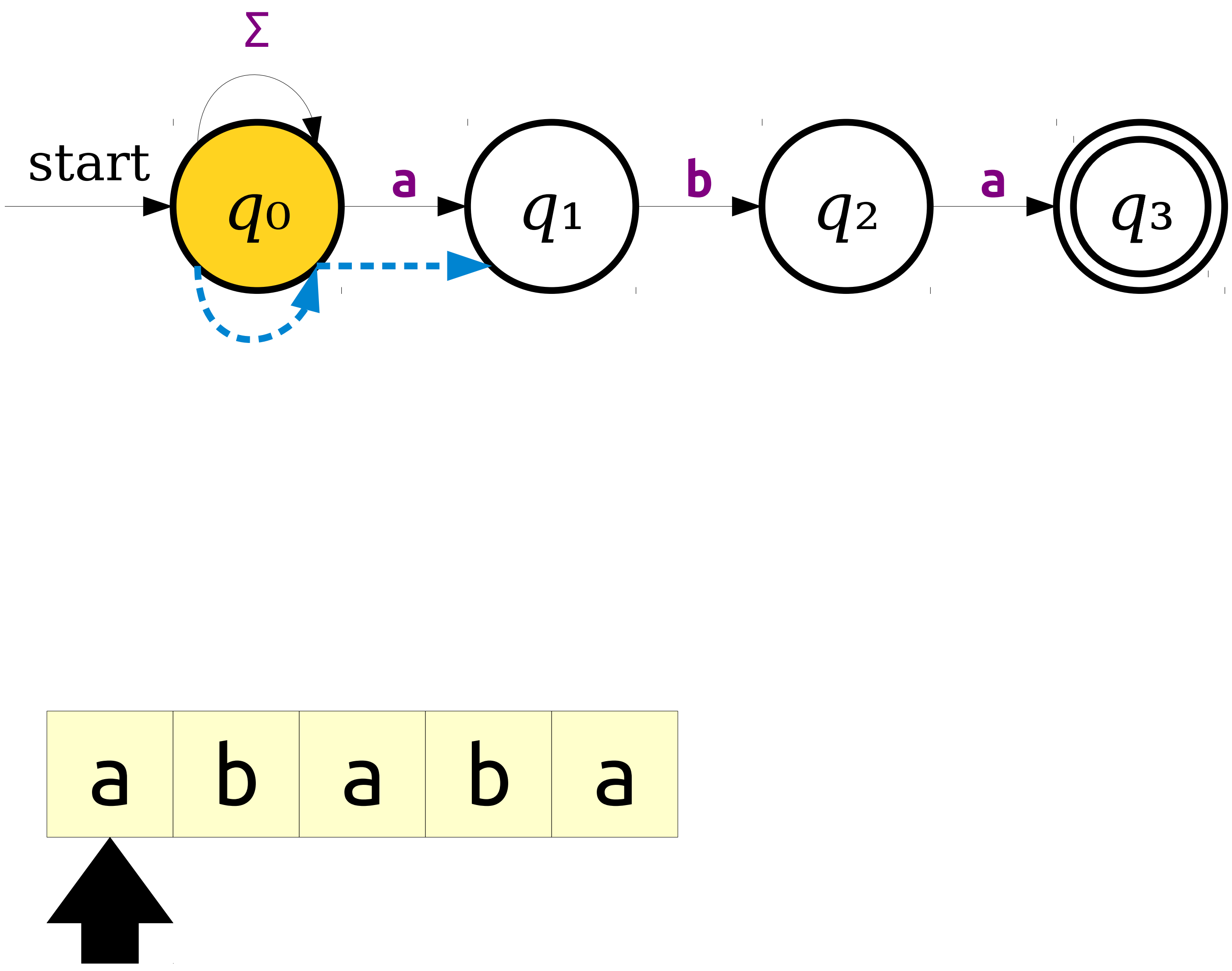


Massive Parallelism

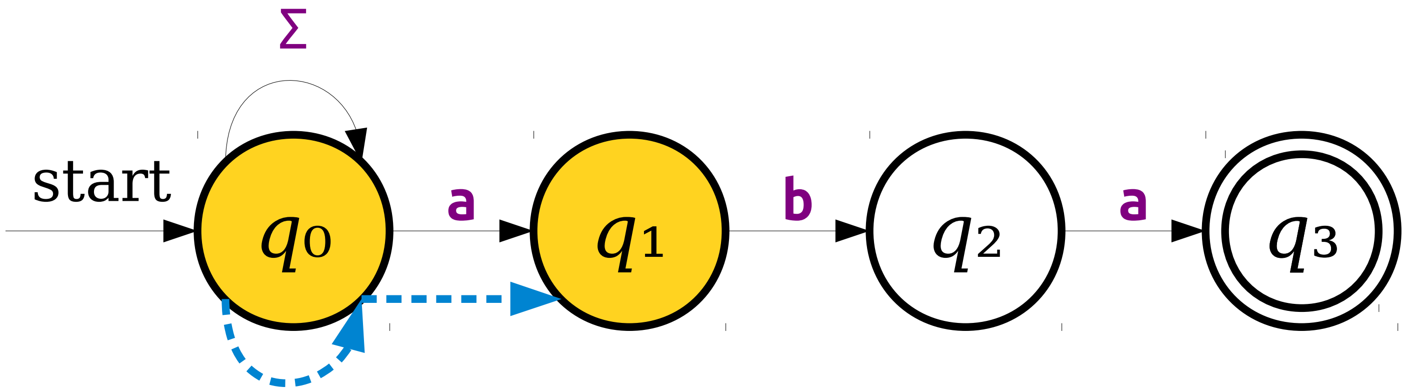


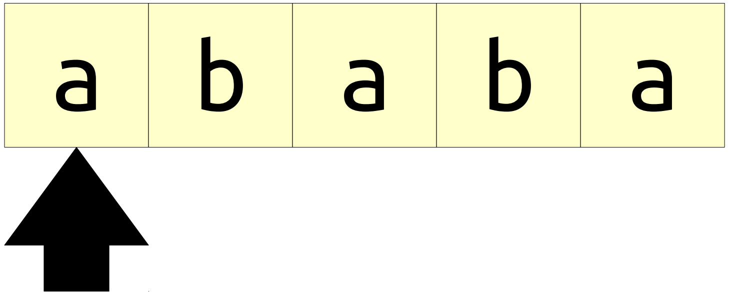


Massive Parallelism

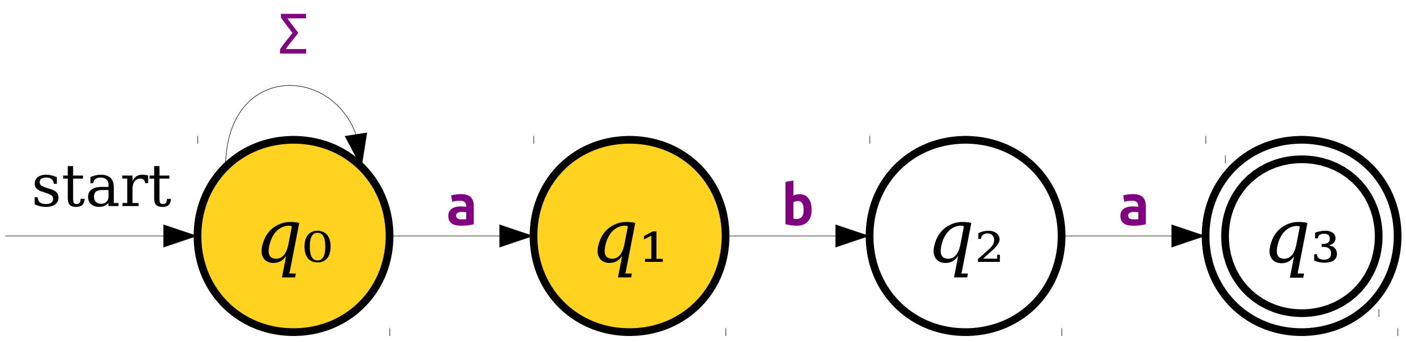


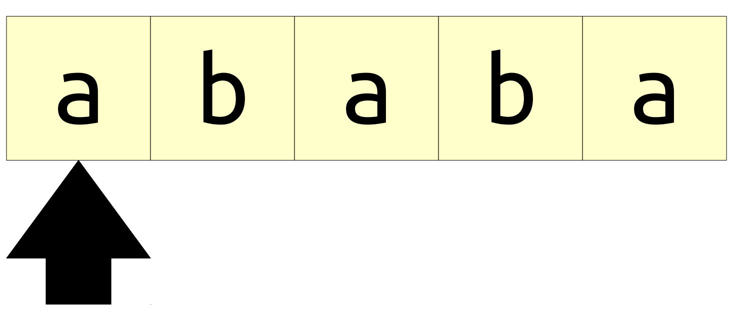
Massive Parallelism



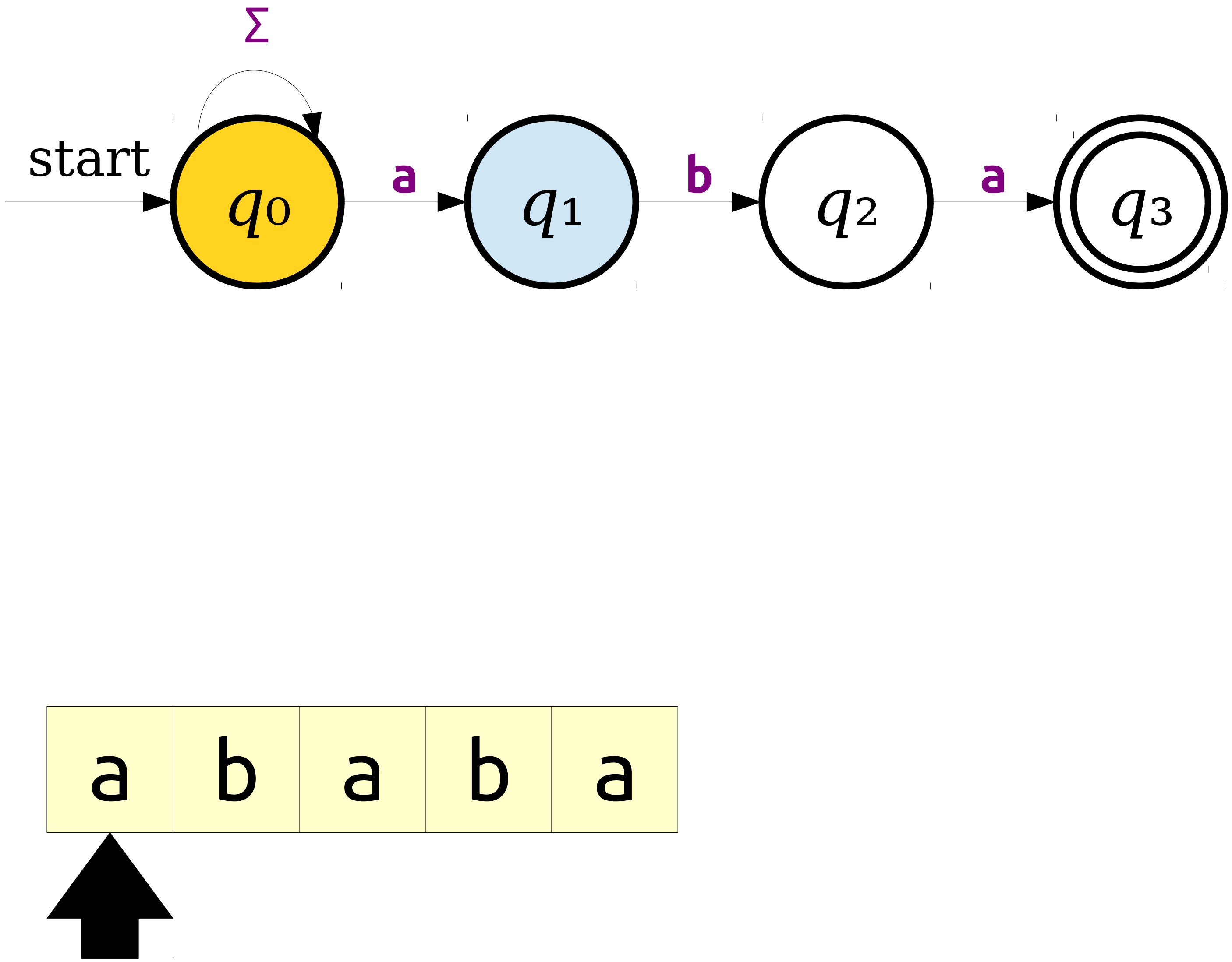


Massive Parallelism

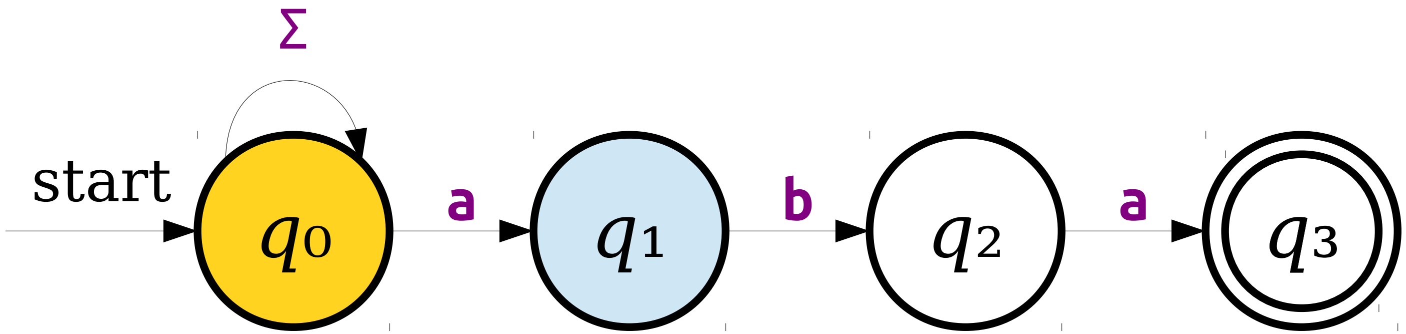


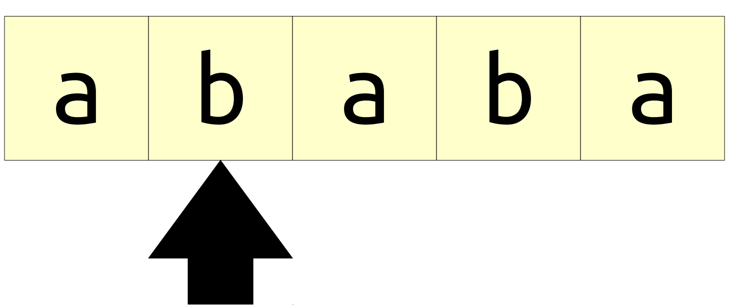


Massive Parallelism

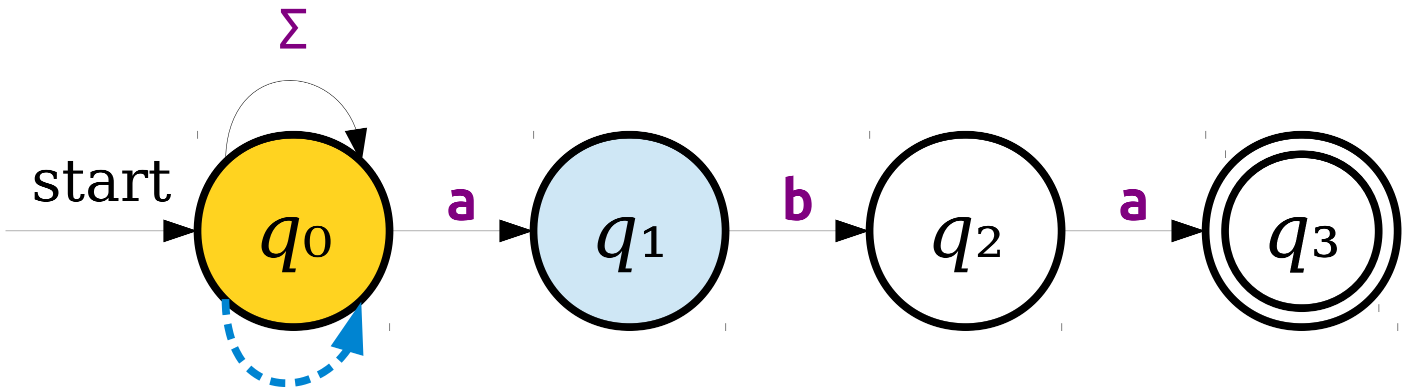


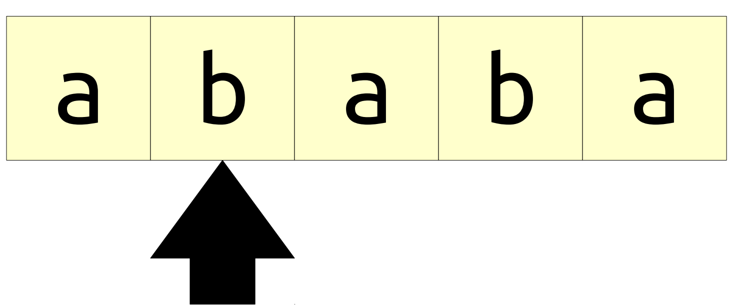
Massive Parallelism



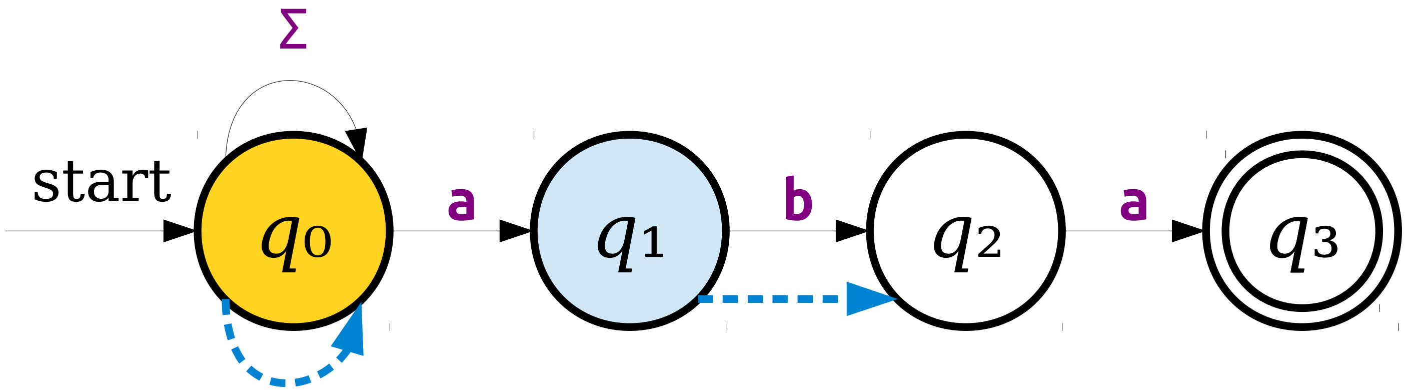


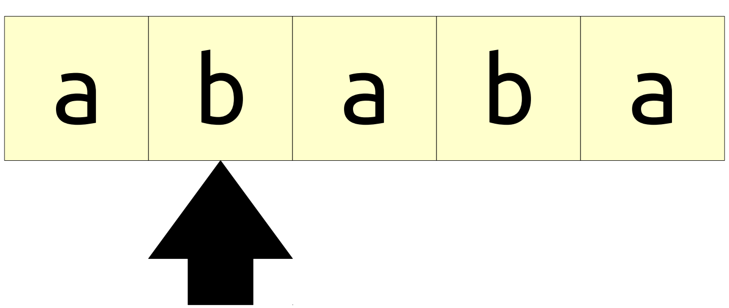
Massive Parallelism



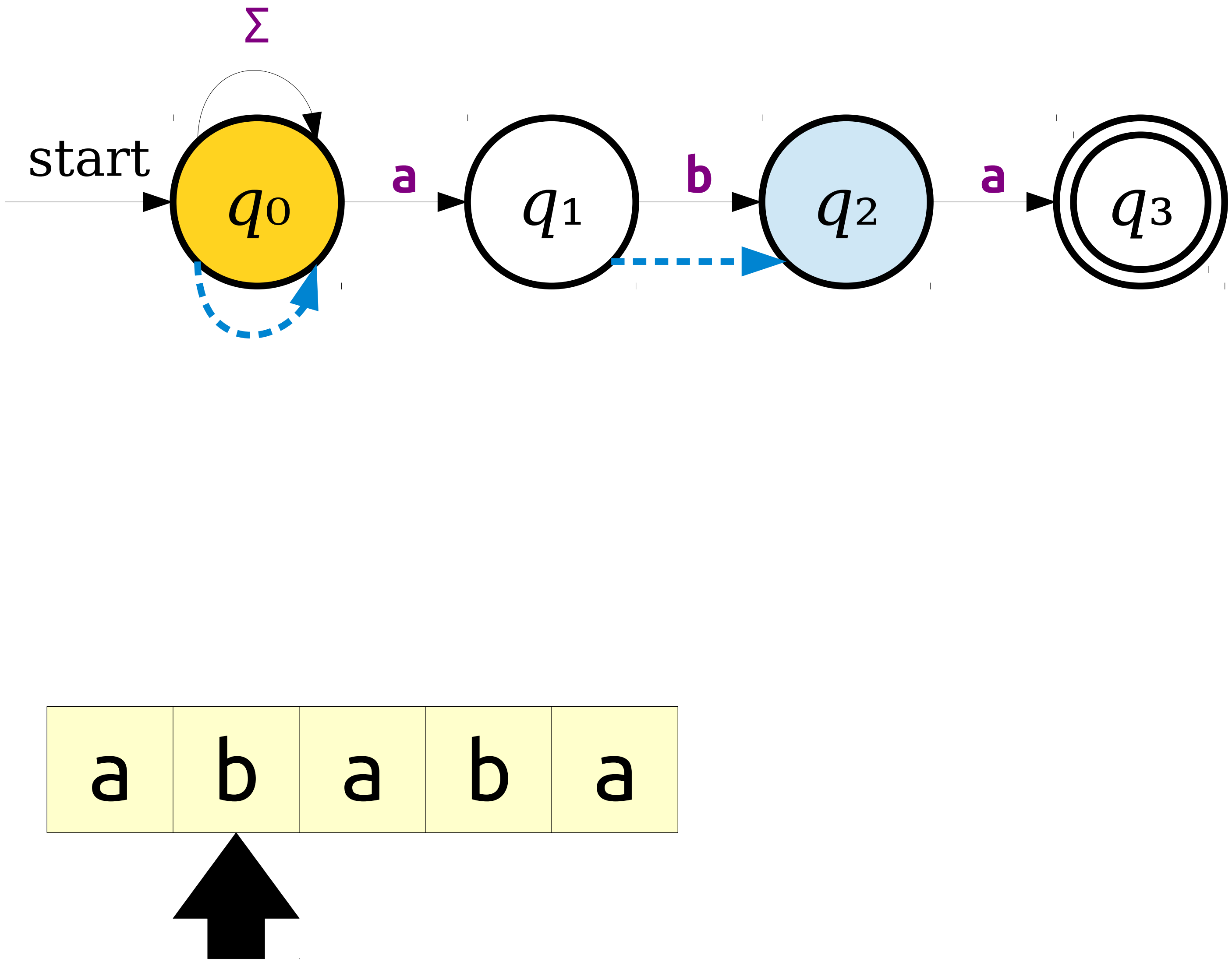


Massive Parallelism

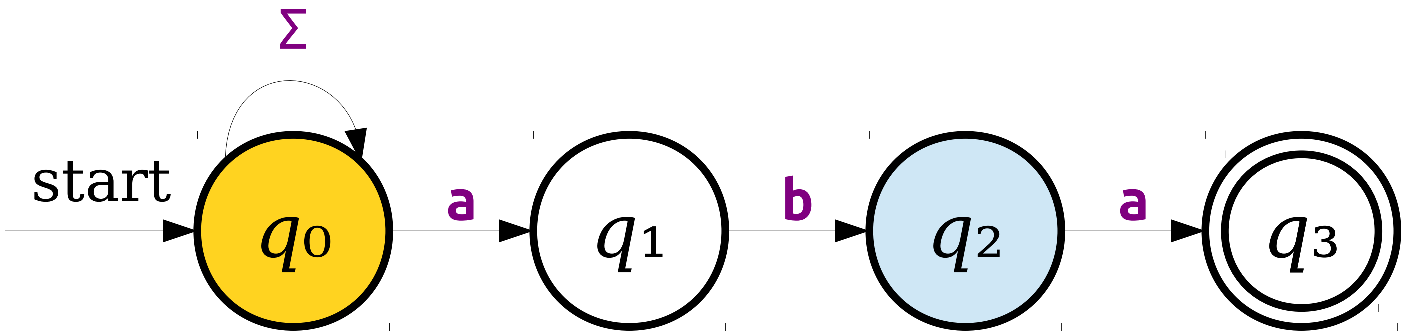


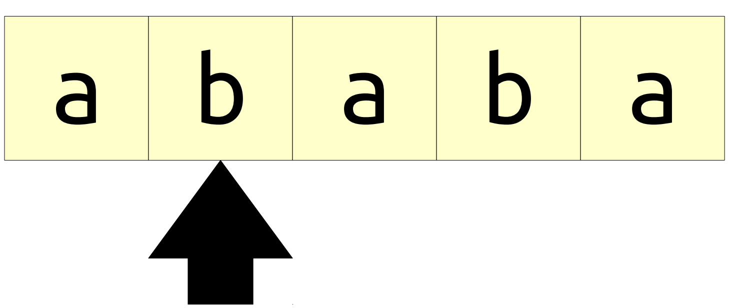


Massive Parallelism

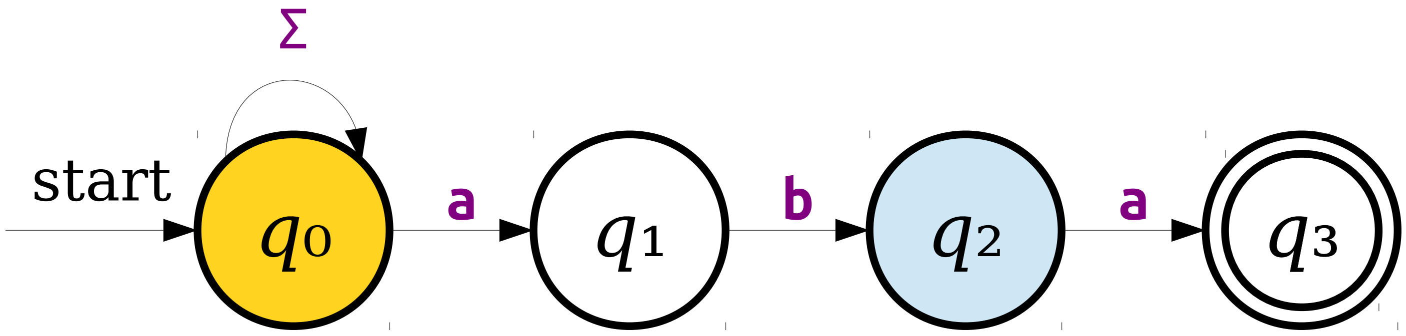


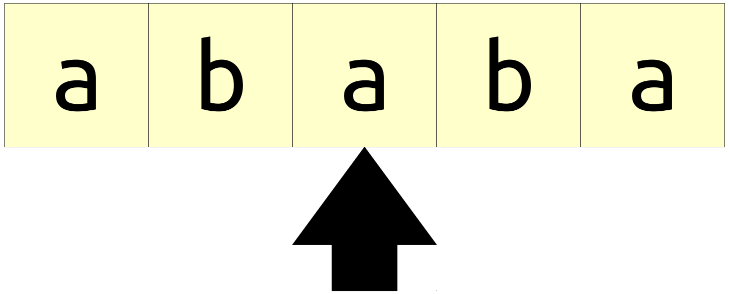
Massive Parallelism



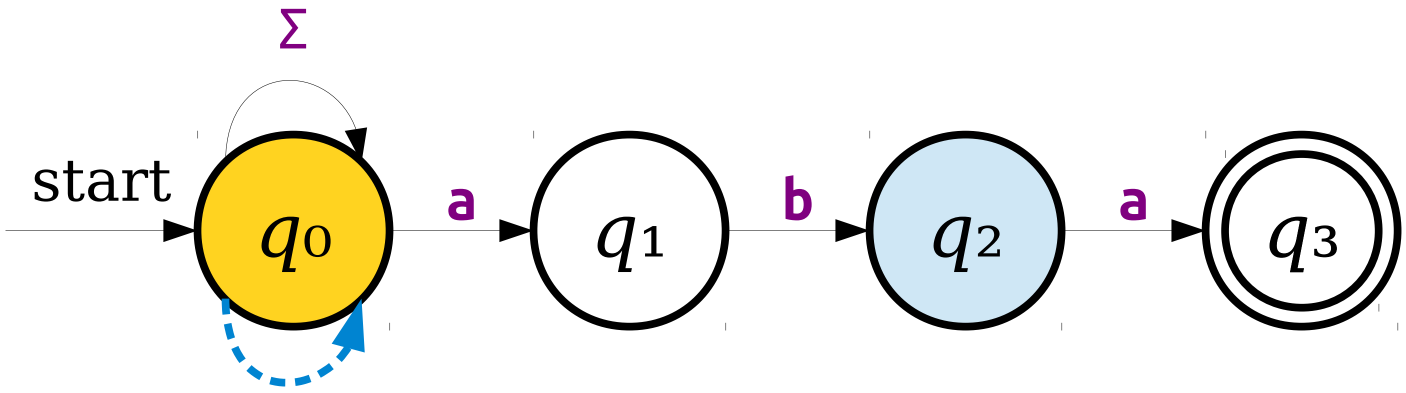


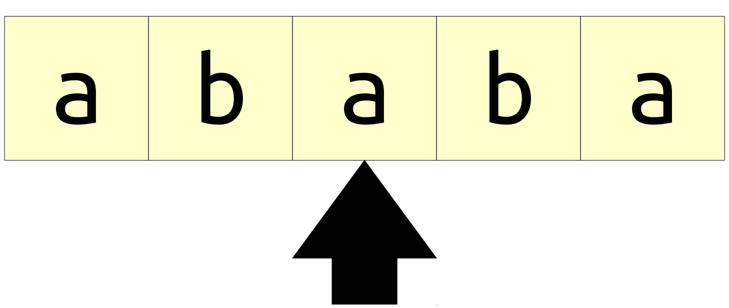
Massive Parallelism



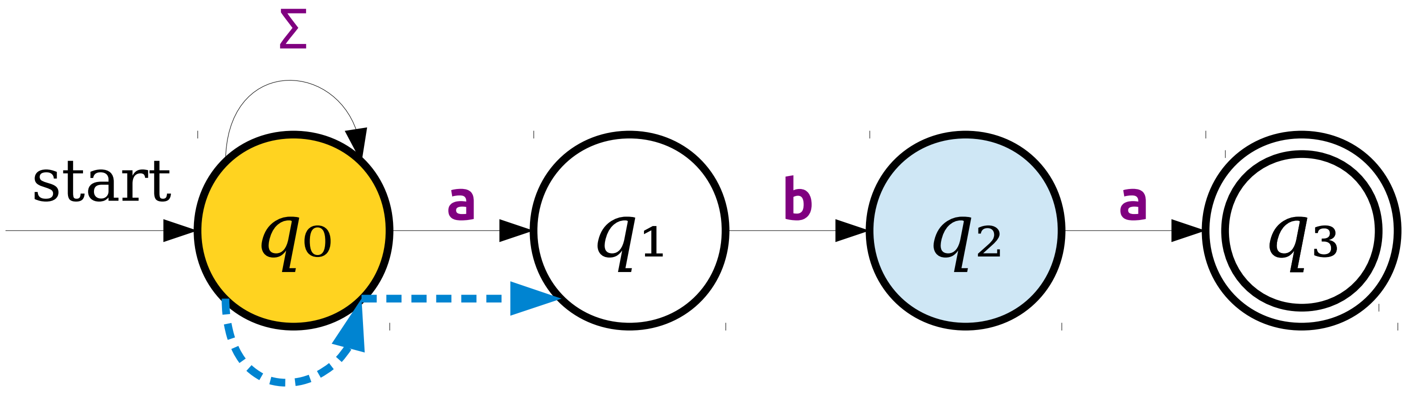


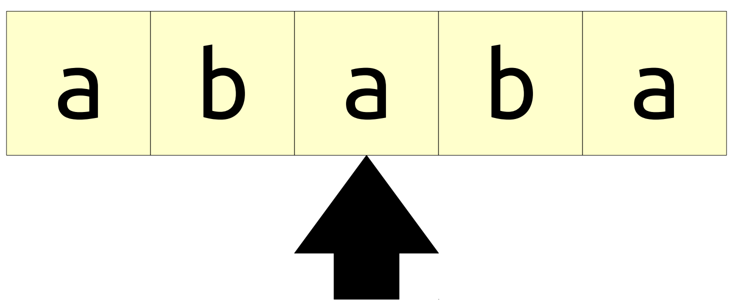
Massive Parallelism



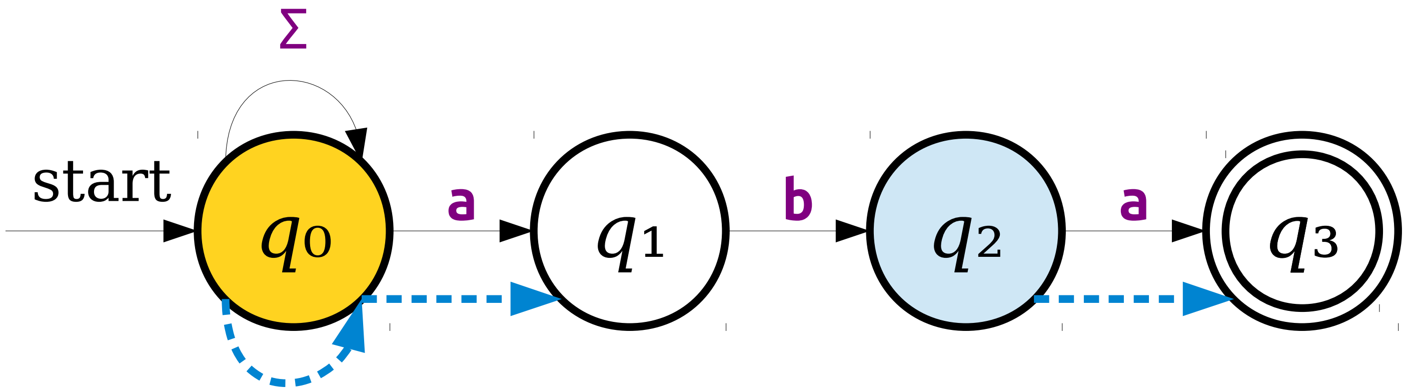


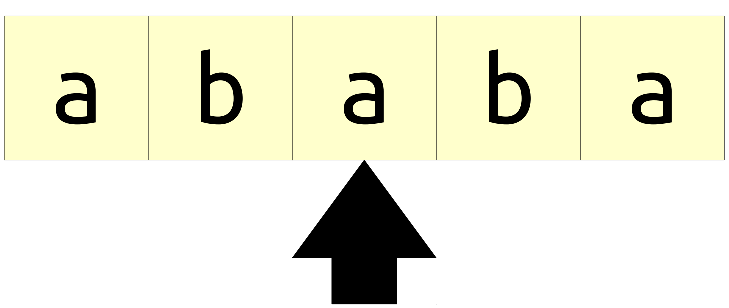
Massive Parallelism



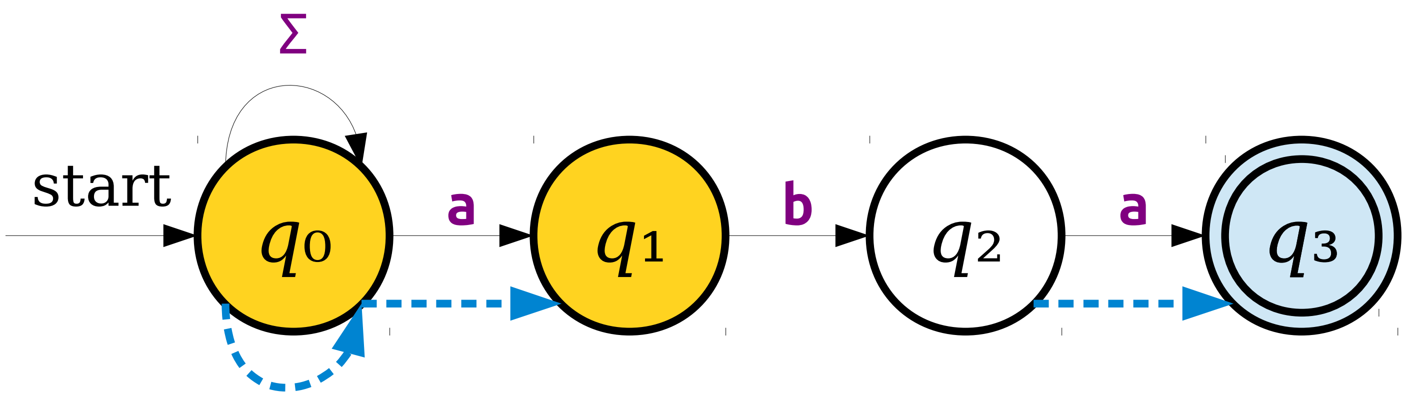


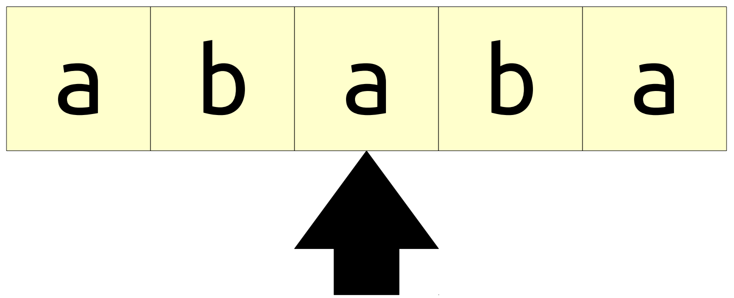
Massive Parallelism



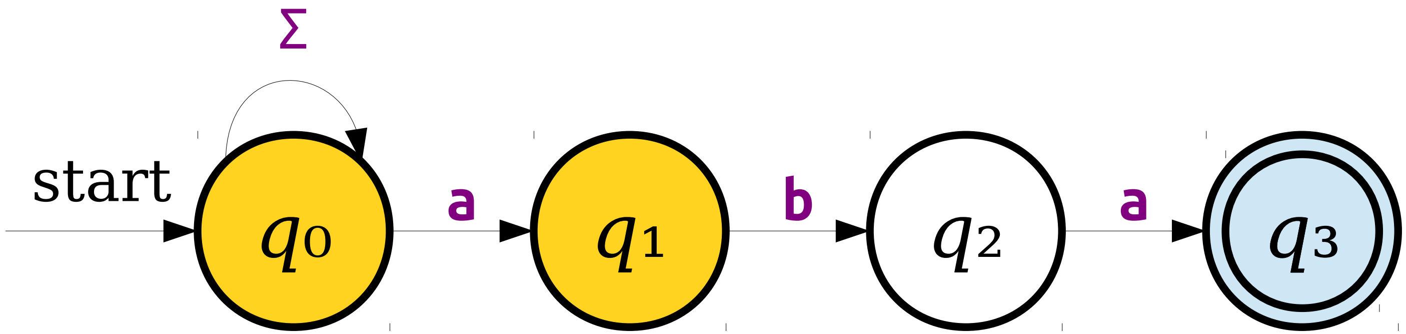


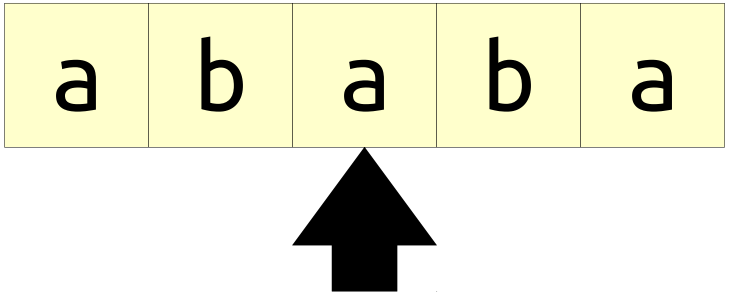
Massive Parallelism



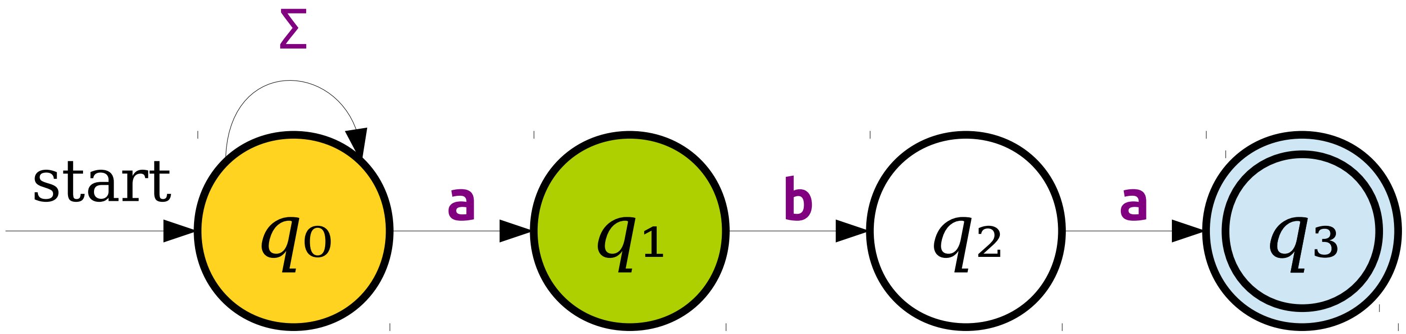


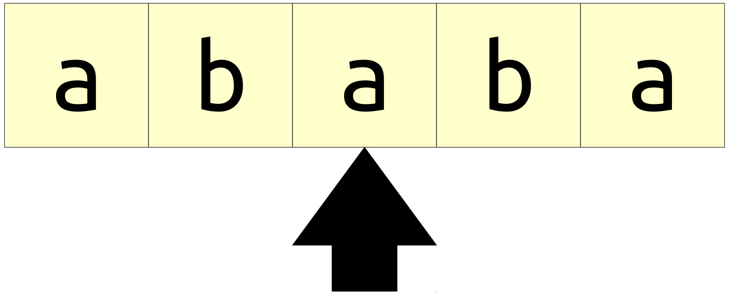
Massive Parallelism



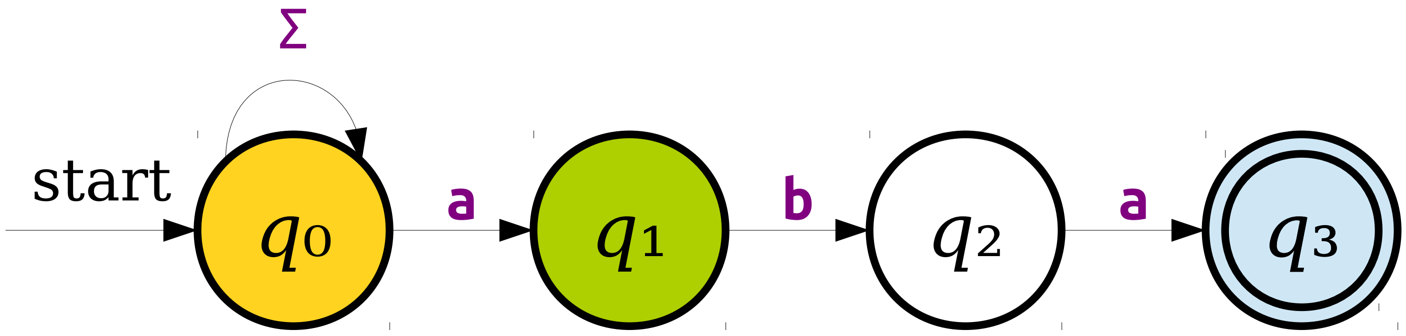


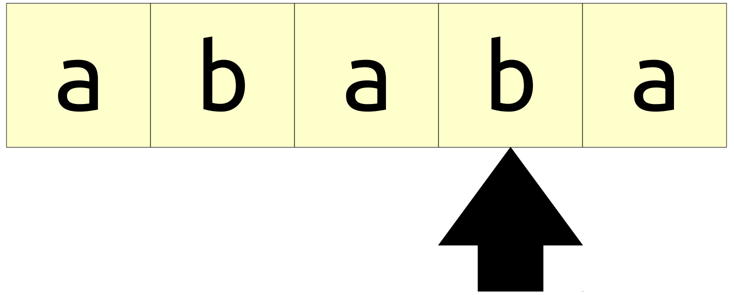
Massive Parallelism



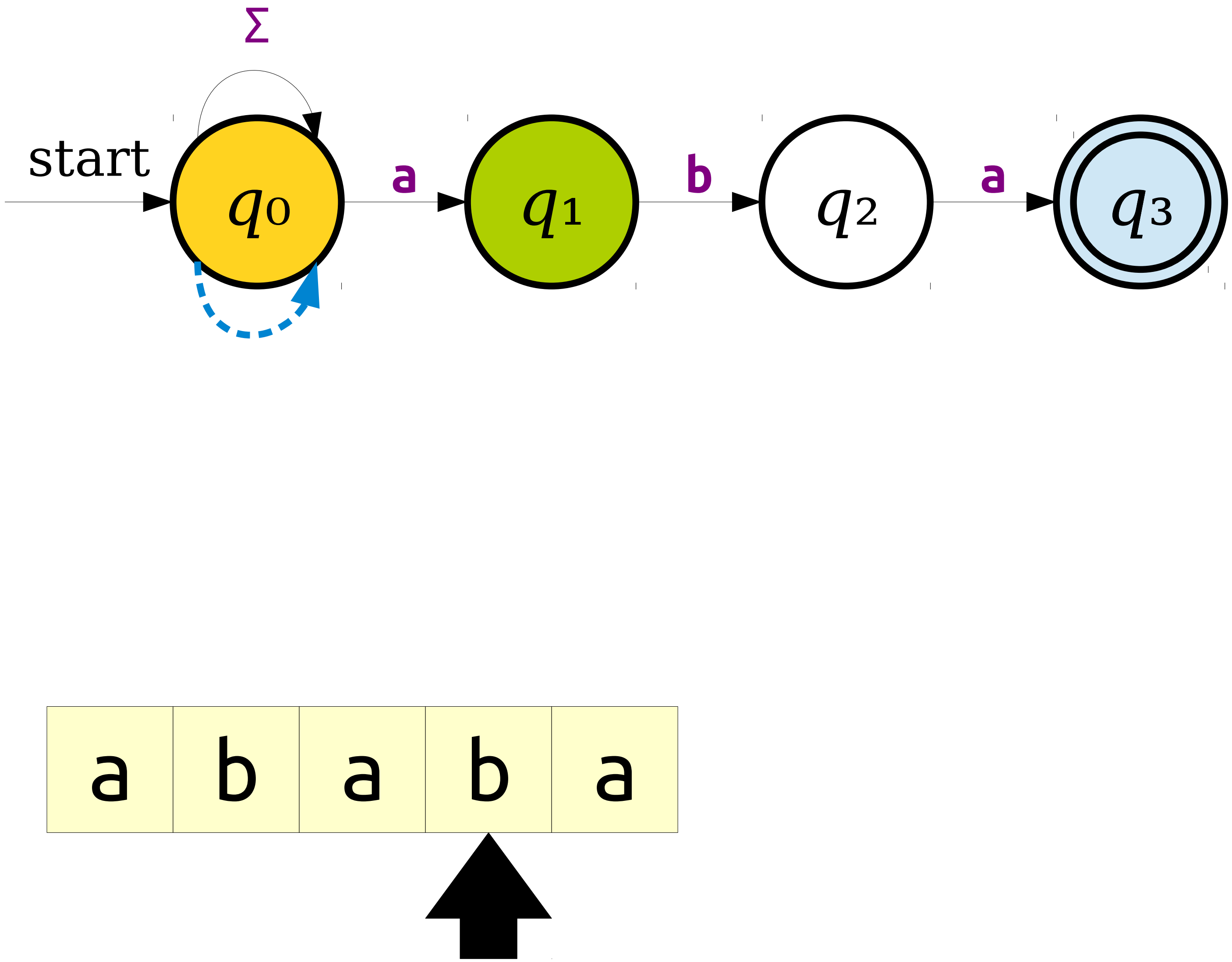


Massive Parallelism

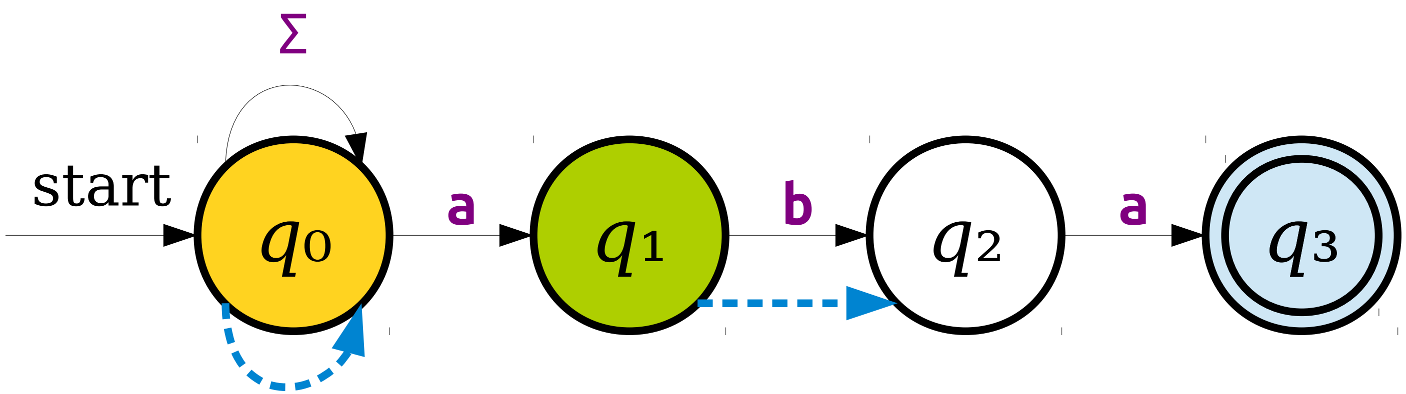


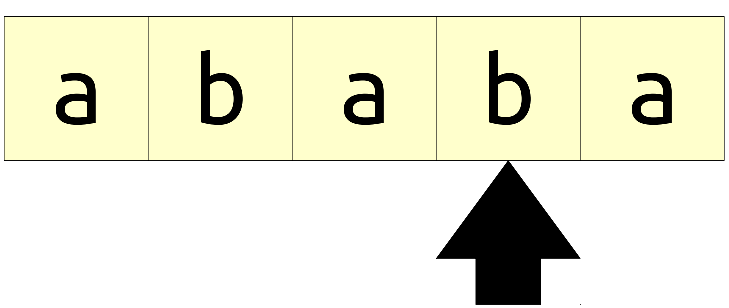


Massive Parallelism

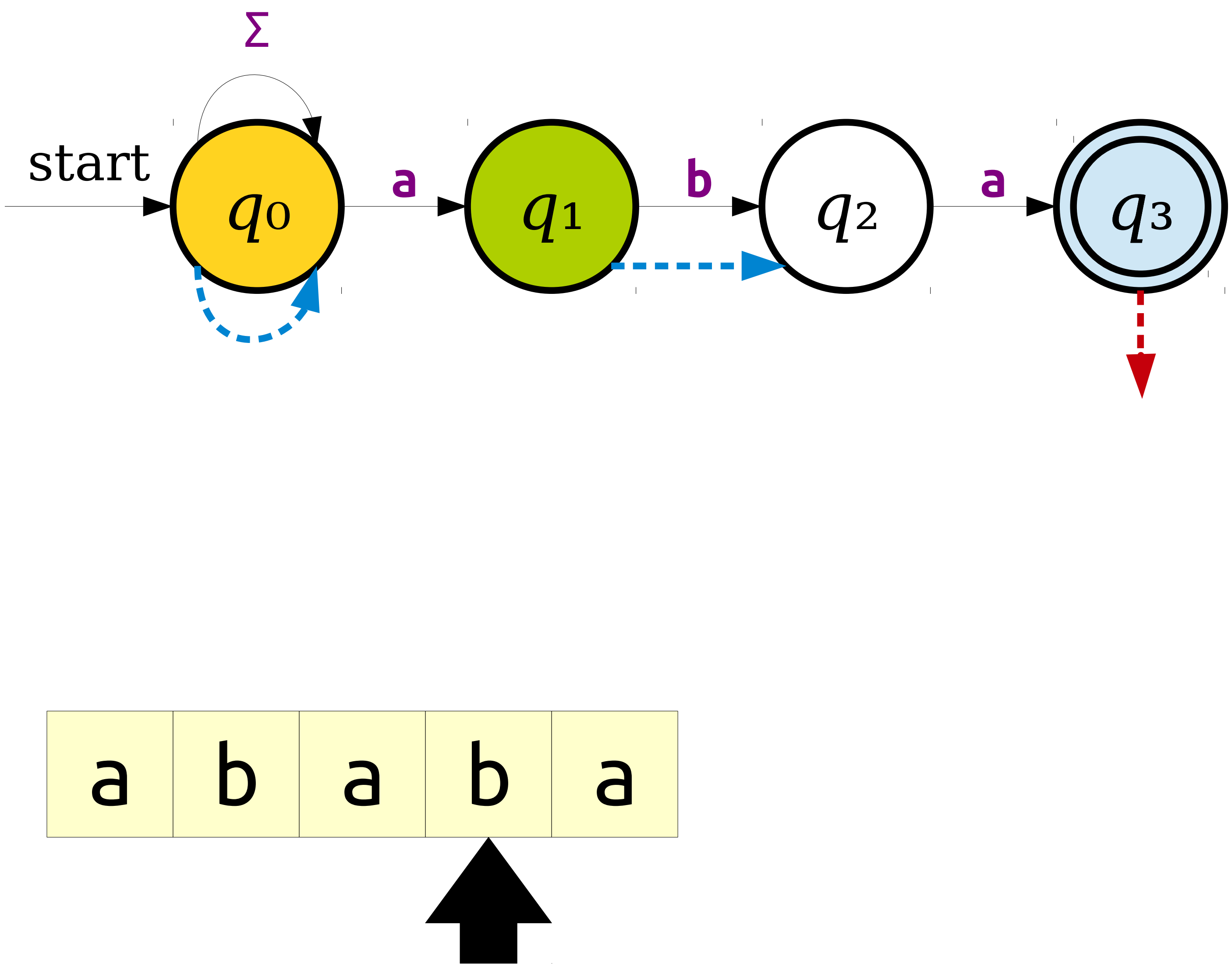


Massive Parallelism

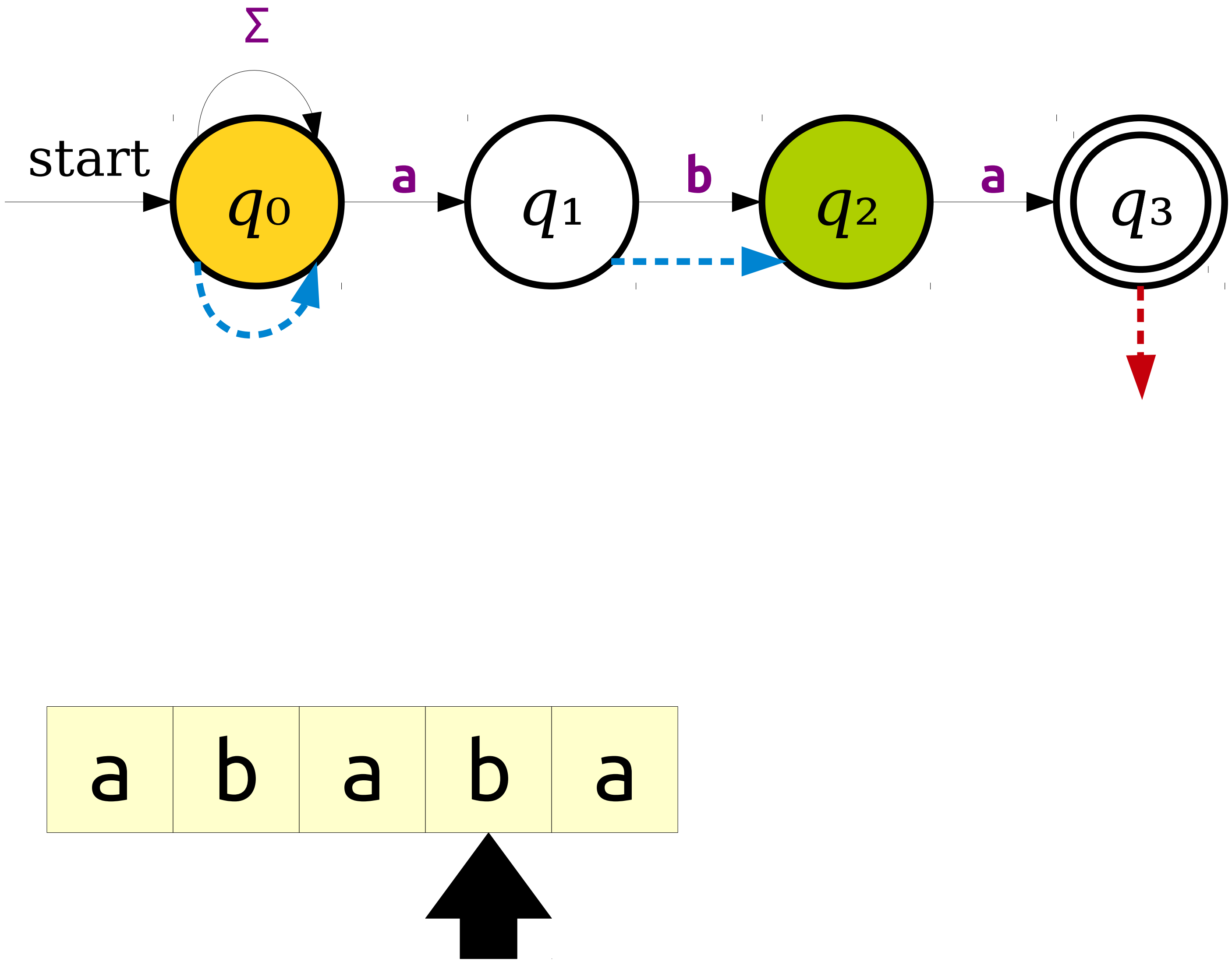




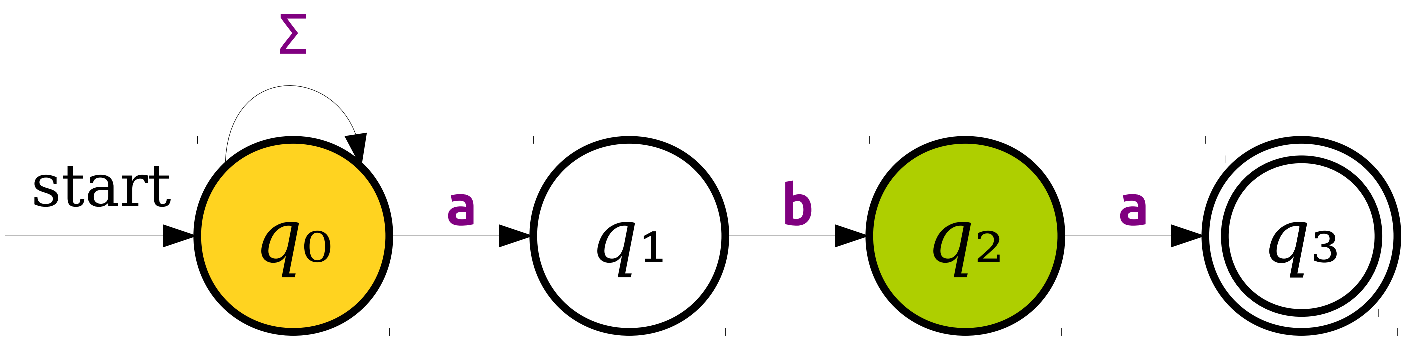
Massive Parallelism

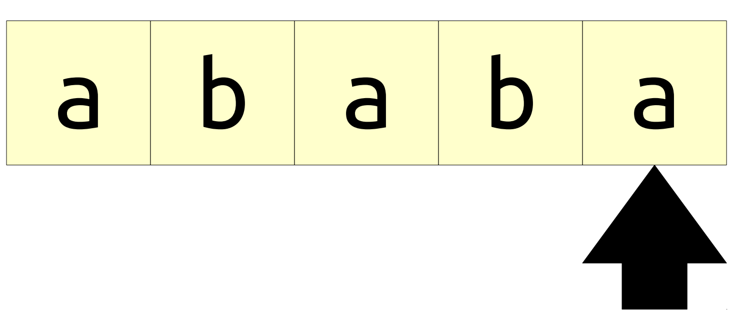


Massive Parallelism

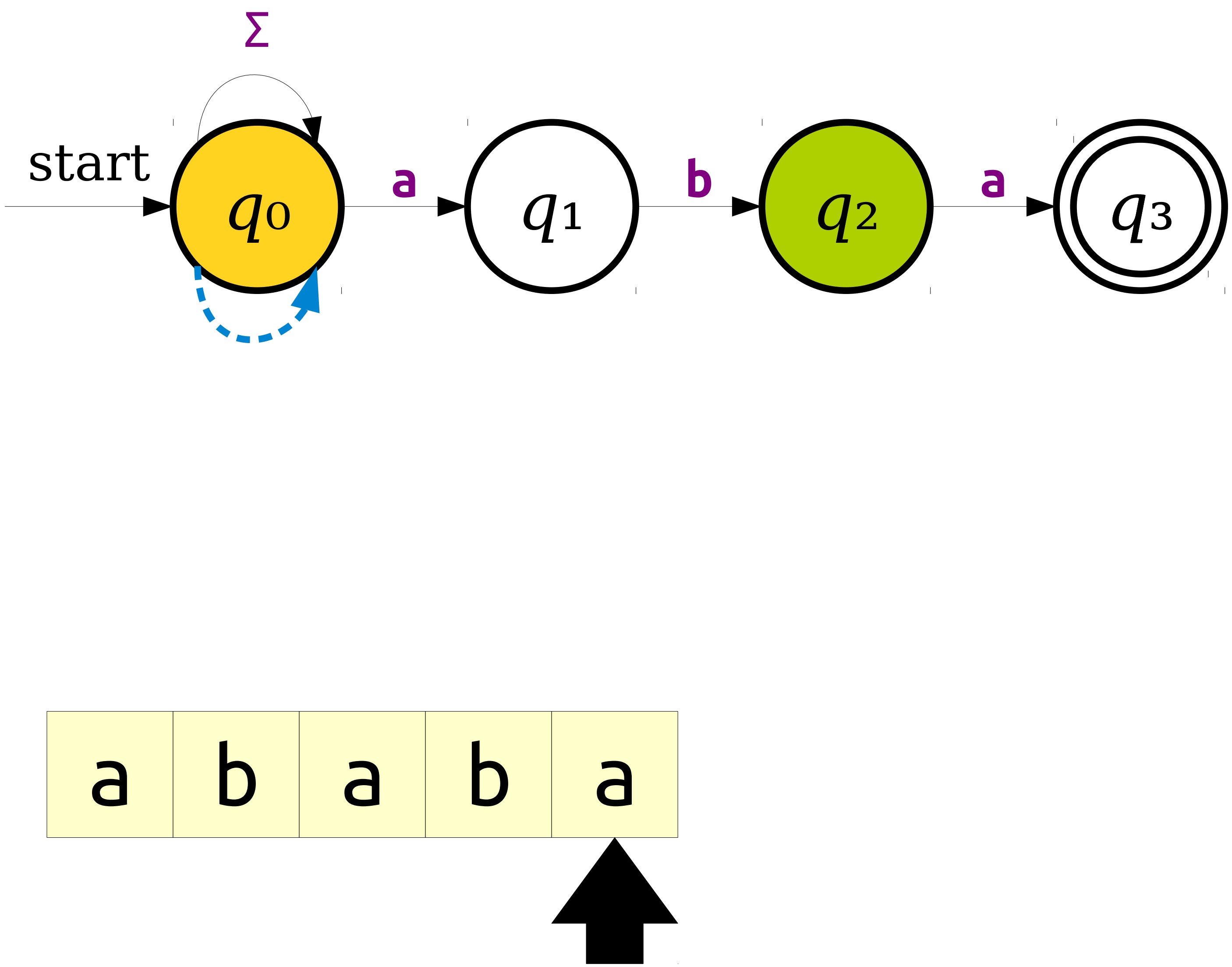


Massive Parallelism

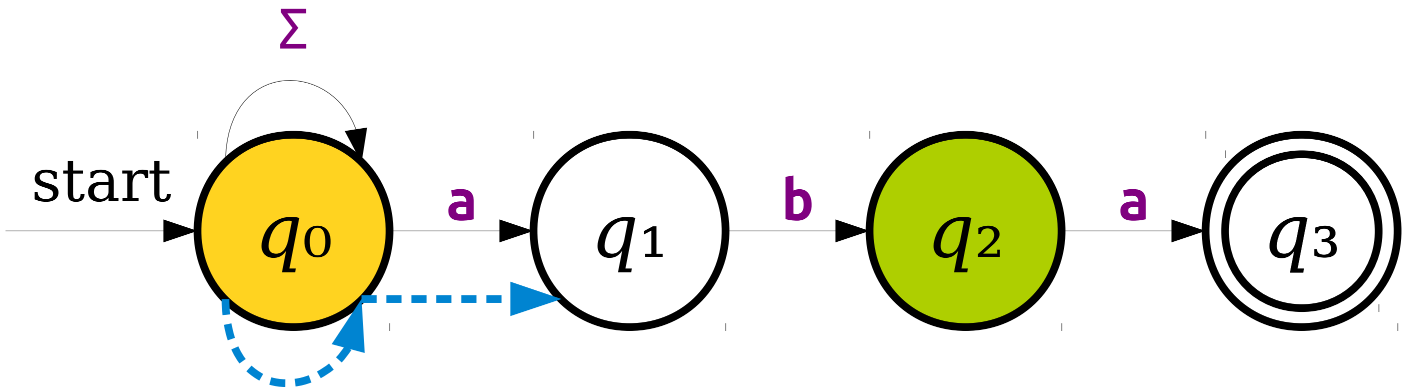


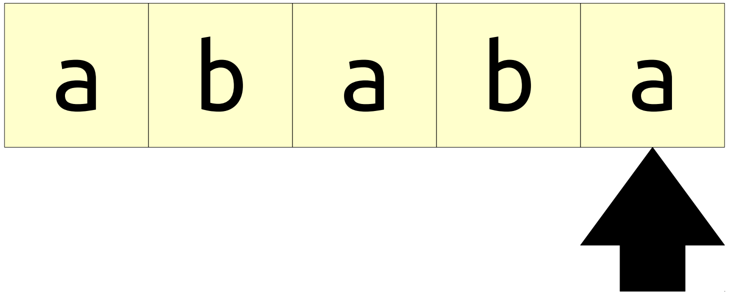


Massive Parallelism

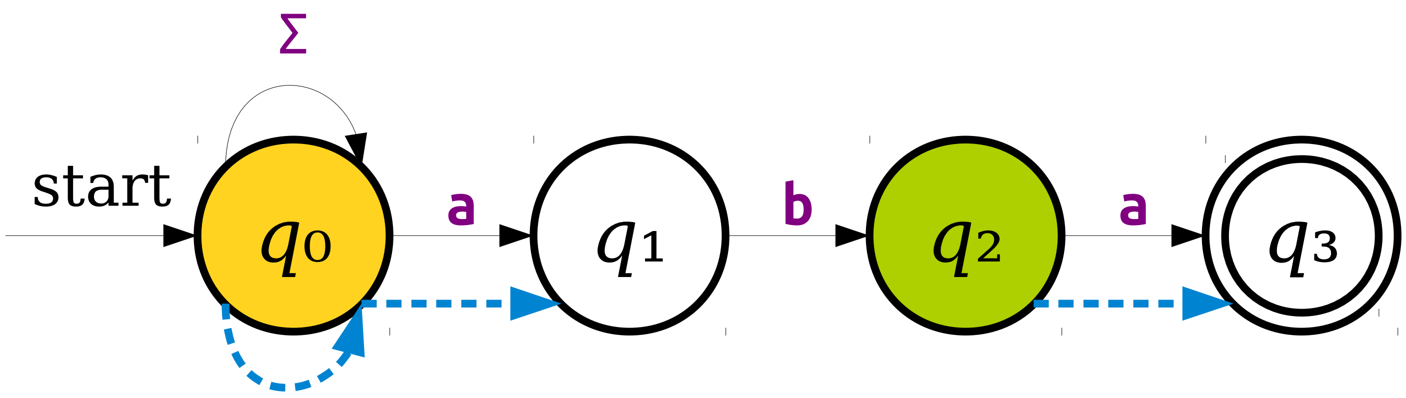


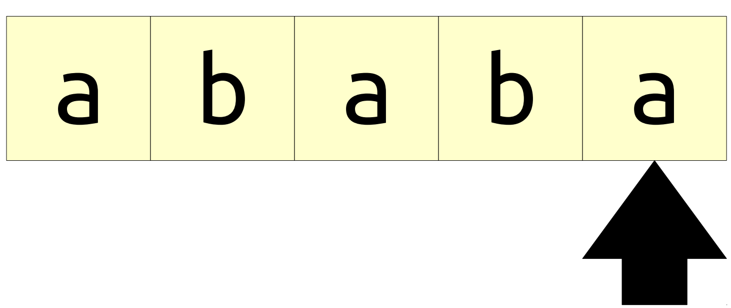
Massive Parallelism



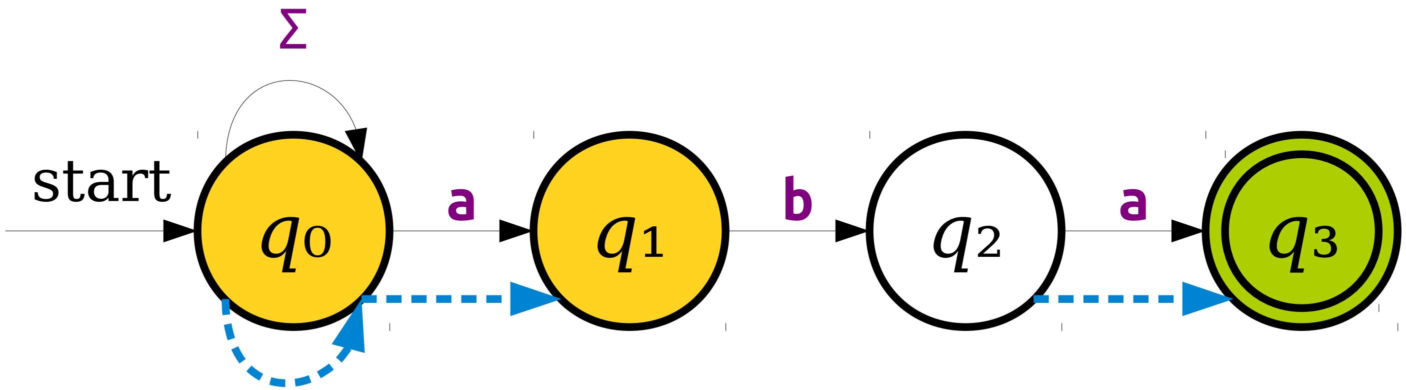


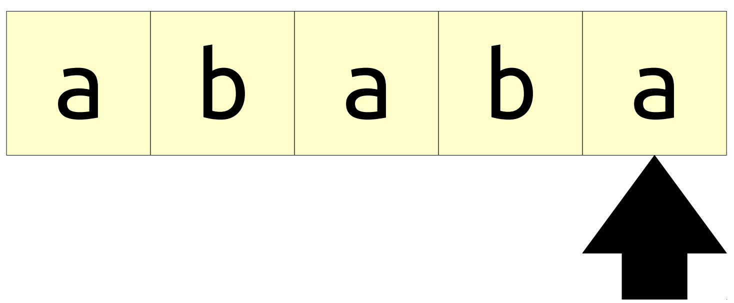
Massive Parallelism



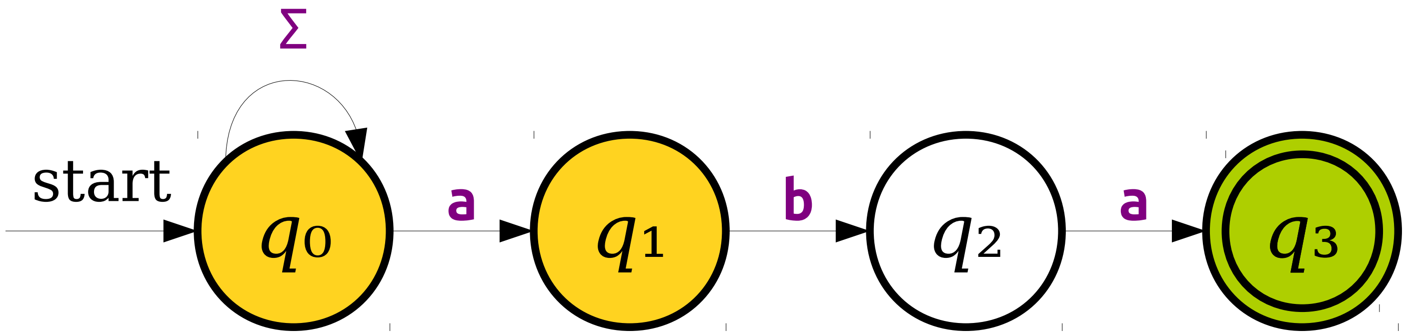


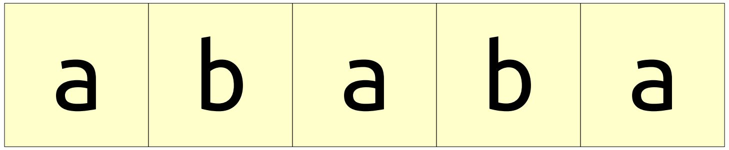
Massive Parallelism



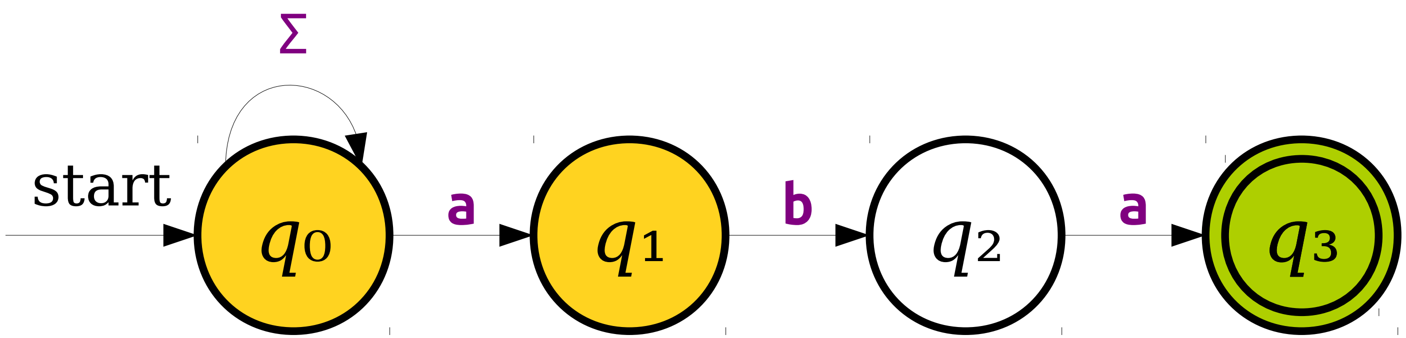


Massive Parallelism





Massive Parallelism



a	b	a	b	a

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

Therefore, we accept!

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.
- (Here's a rigorous explanation about how this works; read this on your own time). If you're in an initial set of states S, to determine the next set of states S', do the following:
  - For each active state, find all the transitions leaving the state on the current symbol.
  - Follow all those transitions and add the endpoints to S'.
  - Follow all ε-transitions out of S' and add them to S'.
  - Your new set of states is the resulting set S'.

So What?

- Each intuition of nondeterminism is useful in a different setting:
  - Perfect guessing is a great way to think about how to design a machine.
  - Massive parallelism is a great way to test machines – and has nice theoretical implications.
- Nondeterministic machines may not be feasible, but they give a great basis for interesting questions:
  - Can any problem that can be solved by a nondeterministic machine be solved by a deterministic machine?
  - Can any problem that can be solved by a nondeterministic machine be solved ***efficiently*** by a deterministic machine?
- The answers vary from automaton to automaton.

Designing NFAs

Designing NFAs

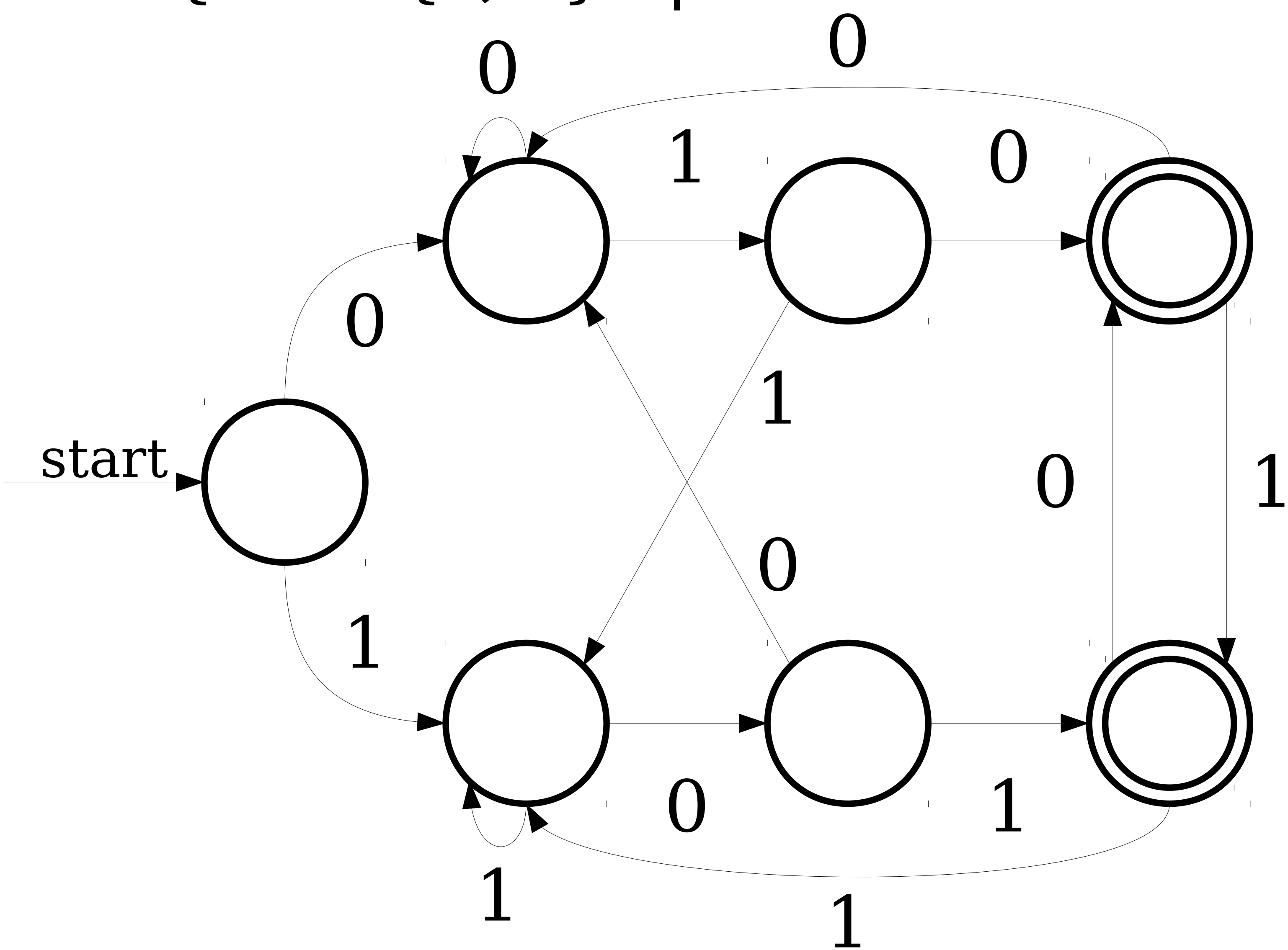
- When designing NFAs, *embrace the nondeterminism!*
- Good model: ***Guess-and-check***:
  - Is there some information that you'd really like to have? Have the machine *nondeterministically guess* that information.
  - Then, have the machine *deterministically check* that the choice was correct.
- The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

Guess-and-Check

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

Guess-and-Check

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

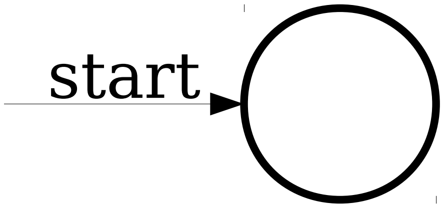


Guess-and-Check

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

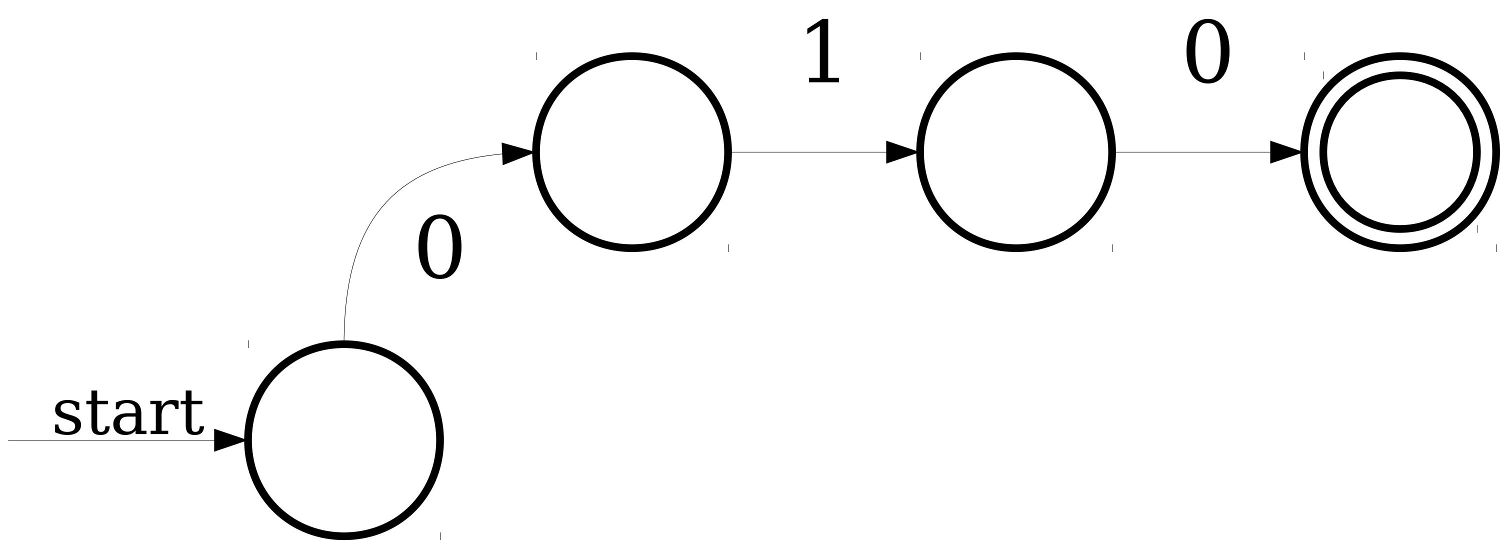
Guess-and-Check

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



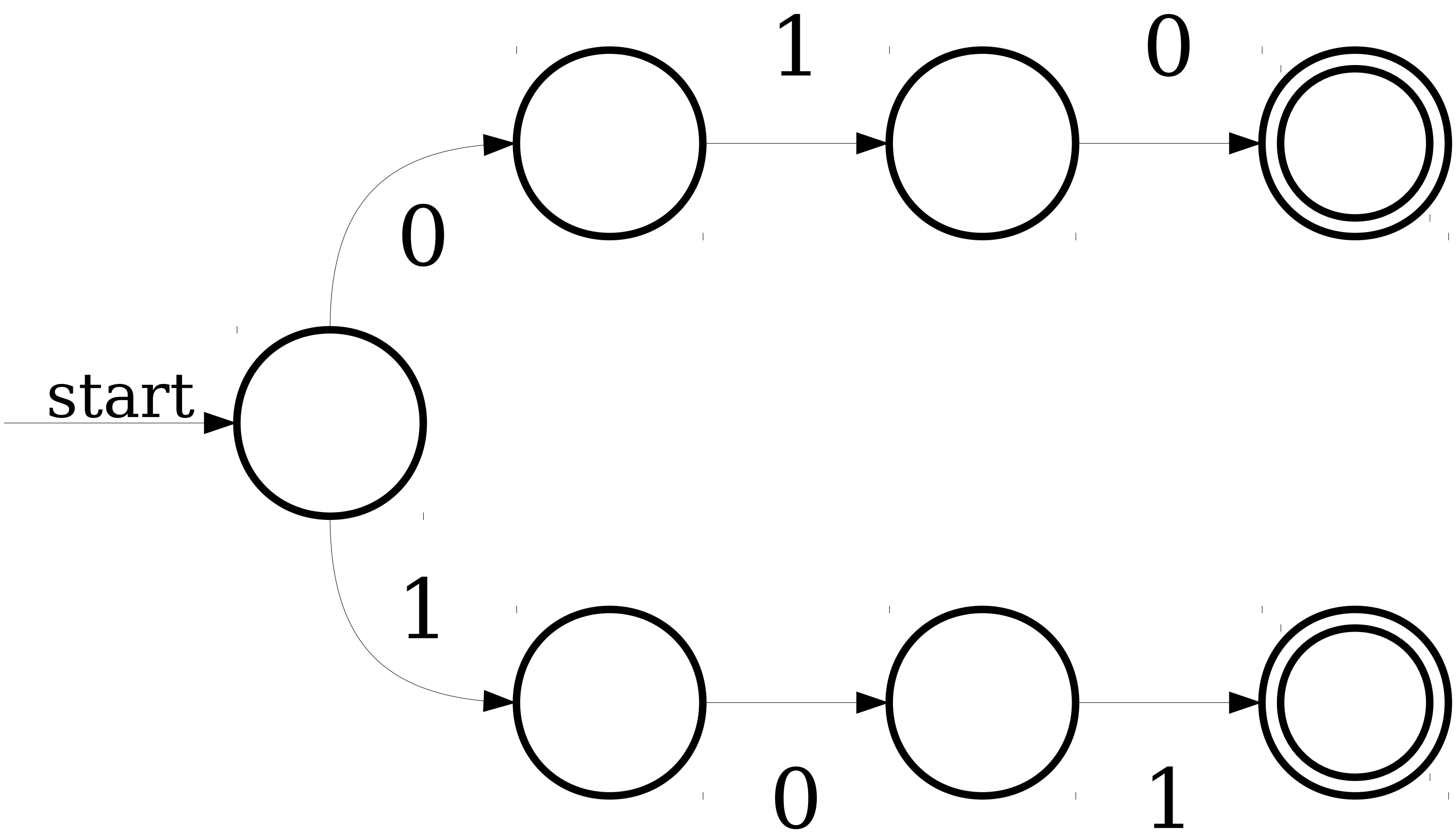
Guess-and-Check

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



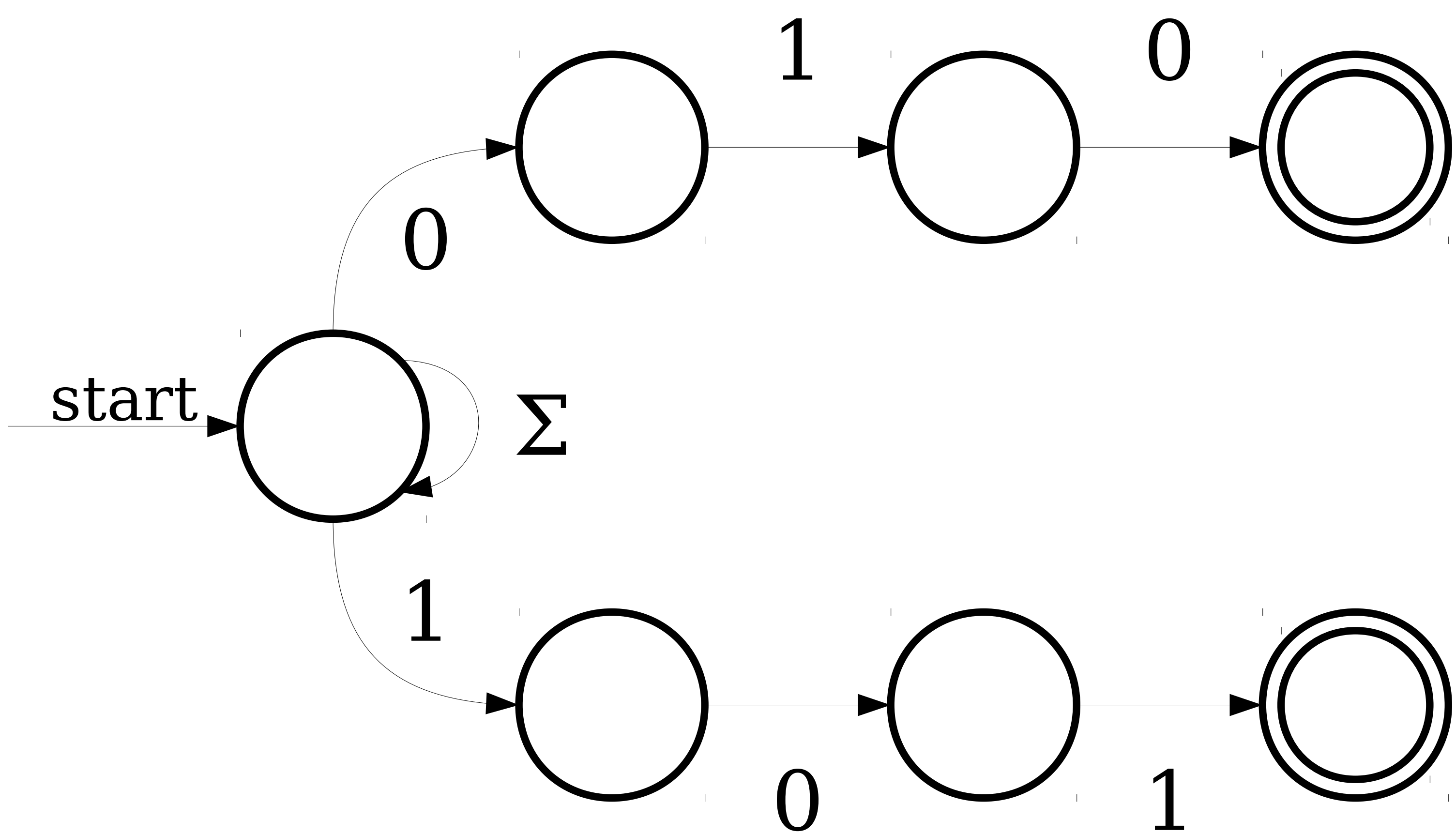
Guess-and-Check

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



Guess-and-Check

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

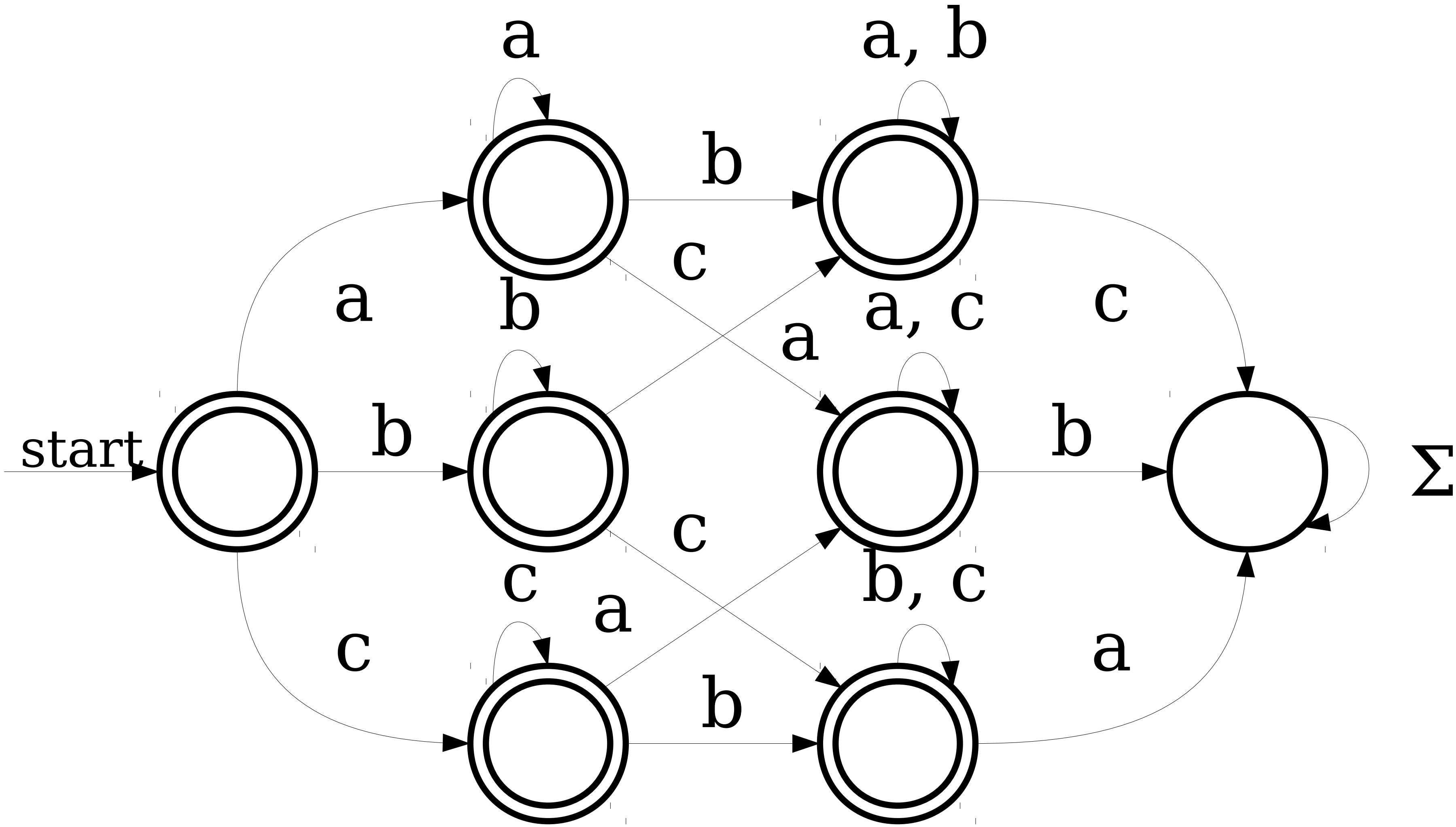


Guess-and-Check

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

Guess-and-Check

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

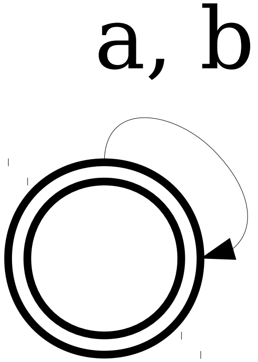


Guess-and-Check

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

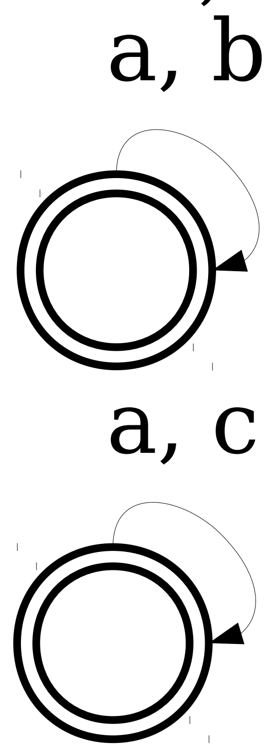
Guess-and-Check

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



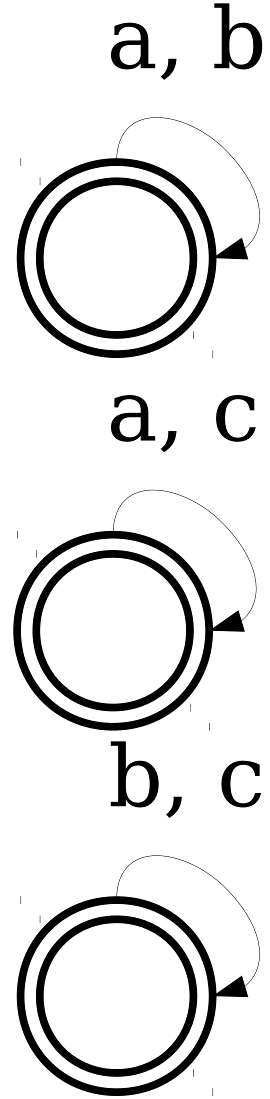
Guess-and-Check

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



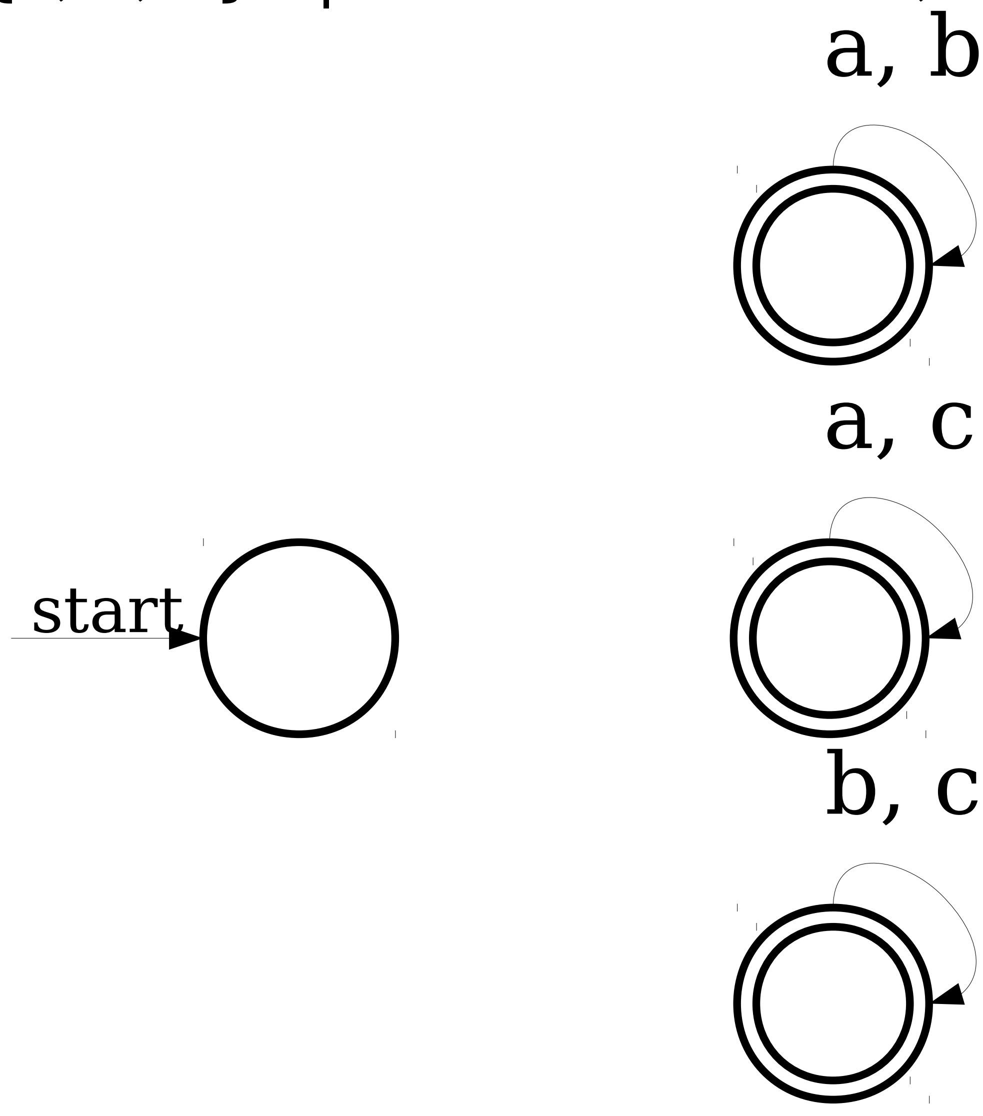
Guess-and-Check

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



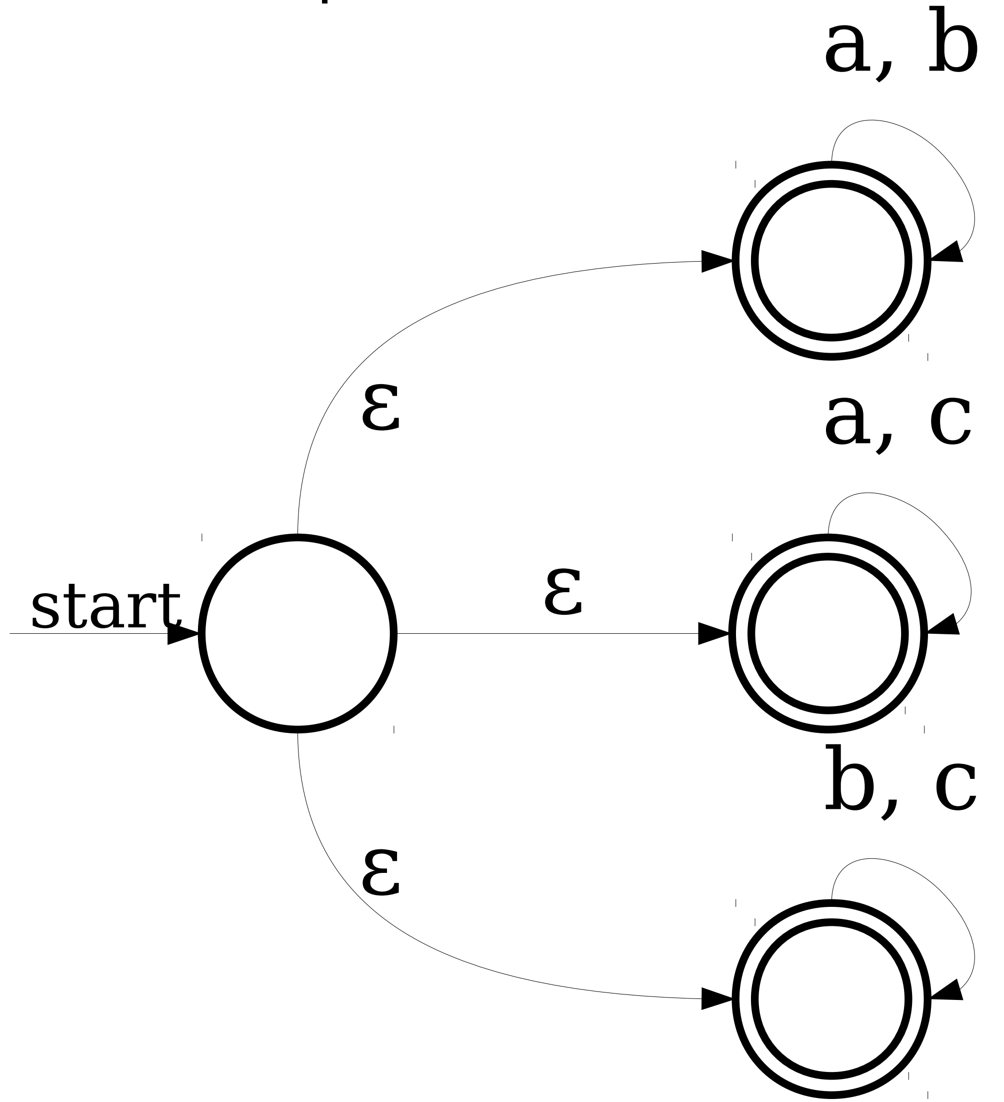
Guess-and-Check

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



Guess-and-Check

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



Just how powerful are NFAs?

Some Words of Encouragement