Finite Automata

Part Two

Recap from Last Time

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

- 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.

If D is a DFA, the ***language of D***, denoted $\mathscr{L}(D)$, is $\{ w \in \Sigma^* \mid D \text{ accepts } w \}$.

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

NFAs

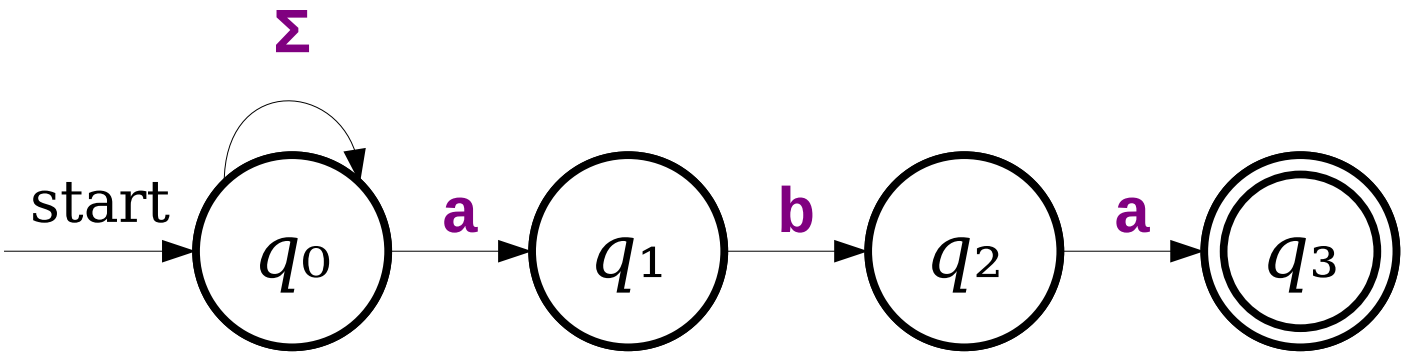
- An ***NFA*** is a
  - ***N***ondeterministic
  - ***F***inite
  - ***A***utomaton
- Can have missing transitions or multiple transitions defined on the same input symbol.
- Accepts if *any possible series of choices* leads to an accepting state.

New Stuff!

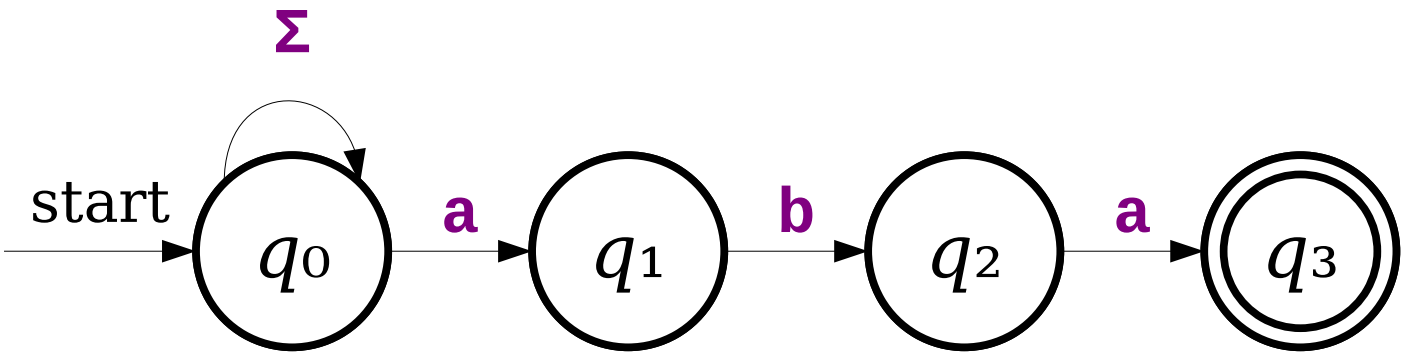
Intuiting Nondeterminism

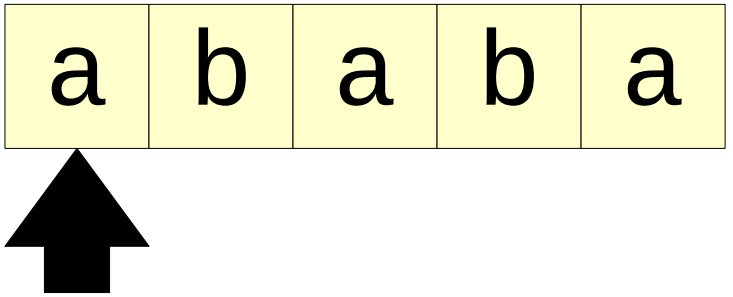
- Nondeterministic machines are a serious departure from physical computers. How can we build up an intuition for them?
- There are two particularly useful frameworks for interpreting nondeterminism:
  - ***Perfect positive guessing***
  - ***Massive parallelism***

Perfect Positive Guessing

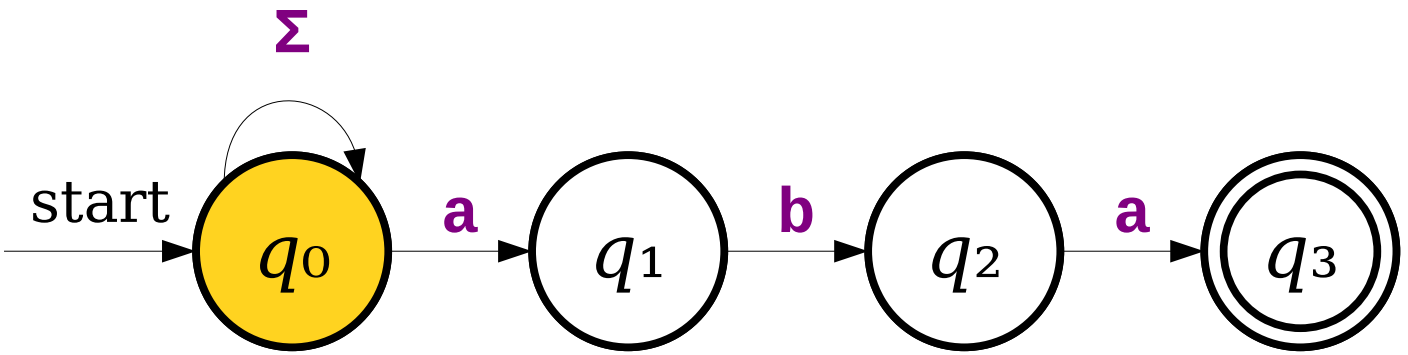


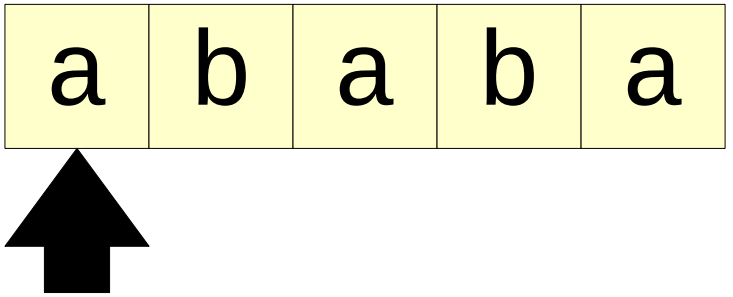
Perfect Positive Guessing



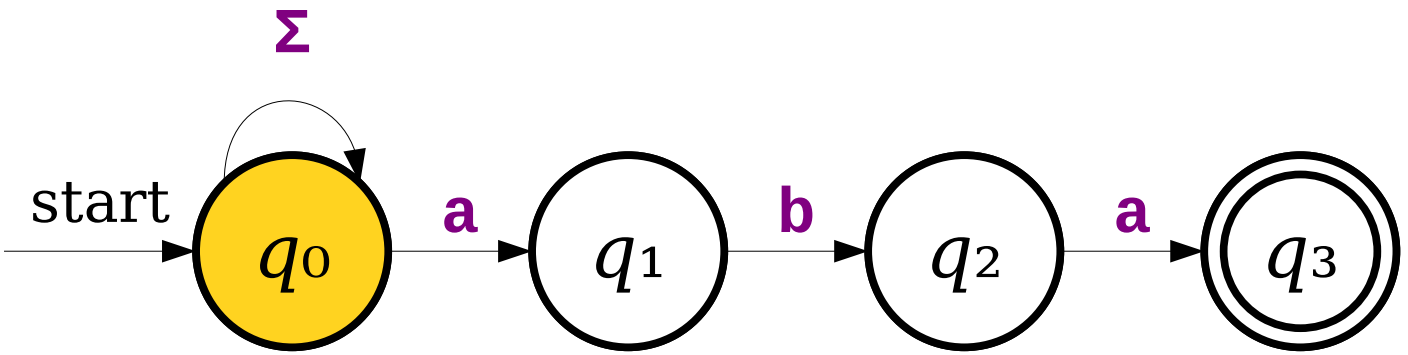


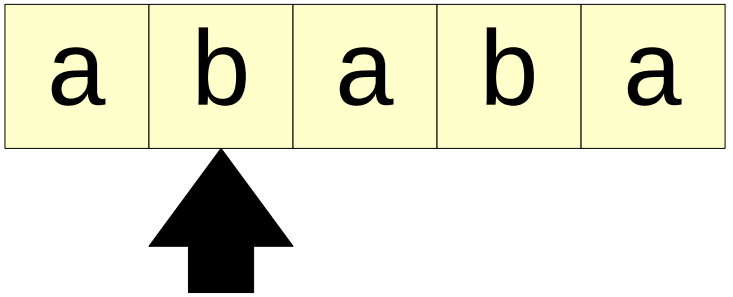
Perfect Positive Guessing



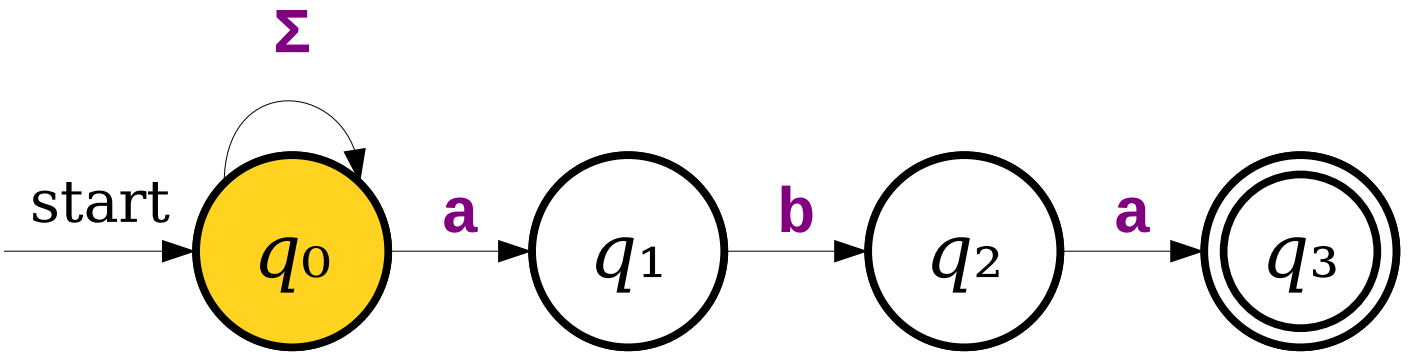


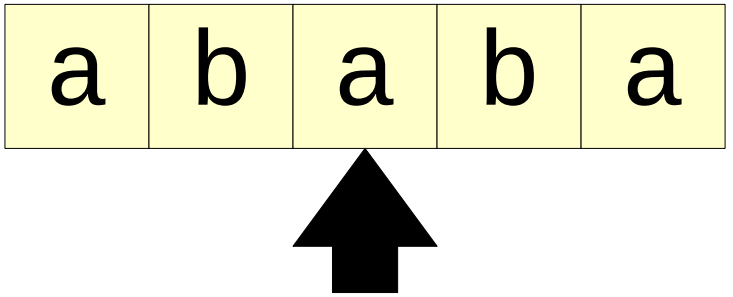
Perfect Positive Guessing



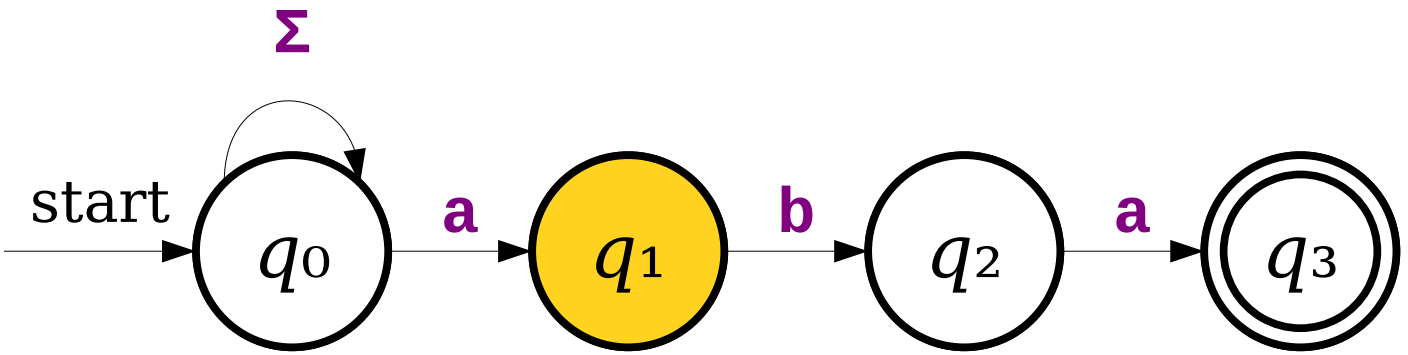


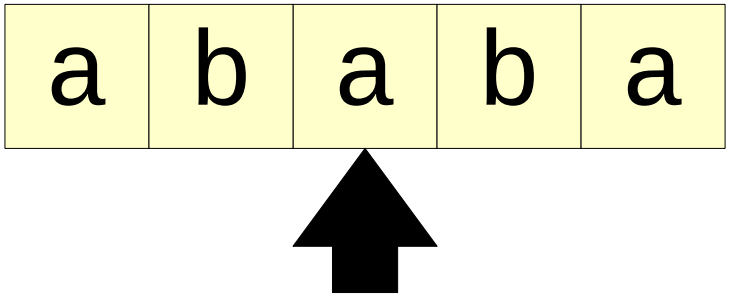
Perfect Positive Guessing



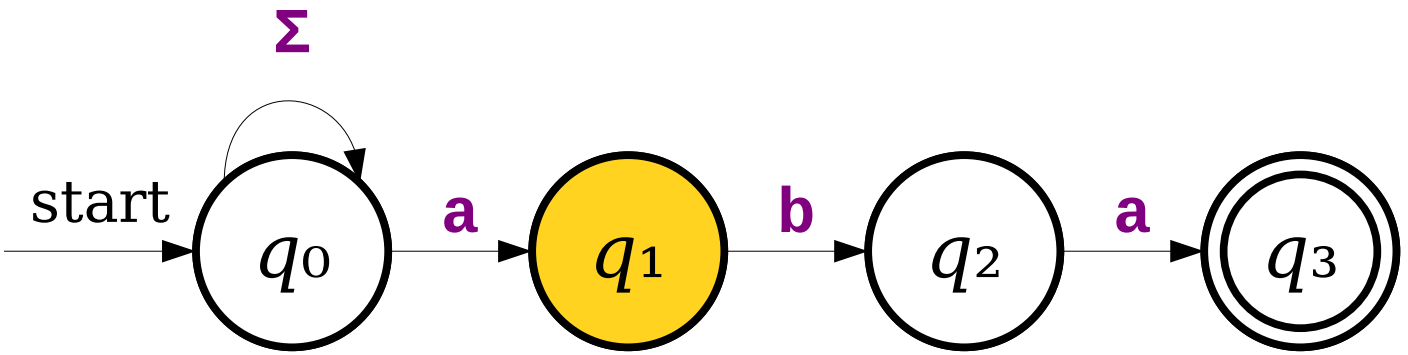


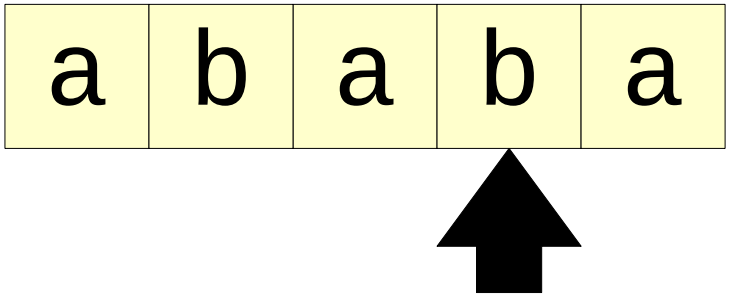
Perfect Positive Guessing



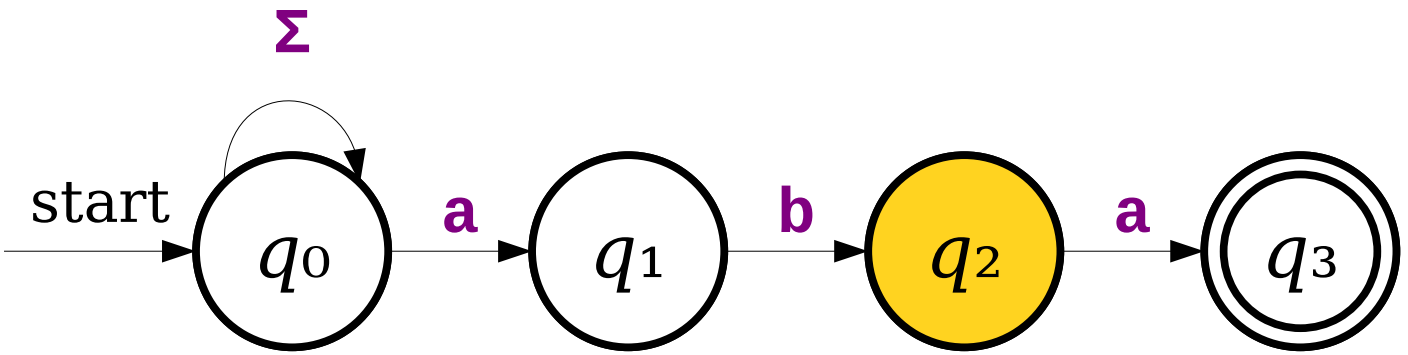


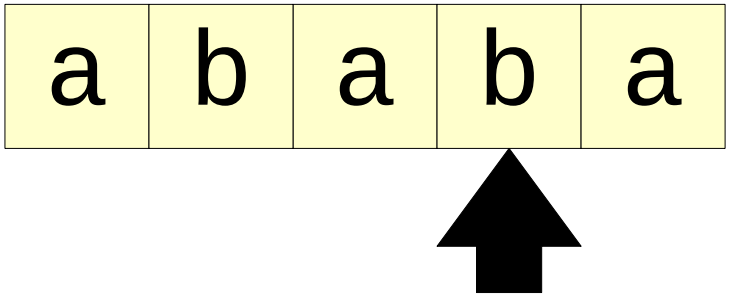
Perfect Positive Guessing



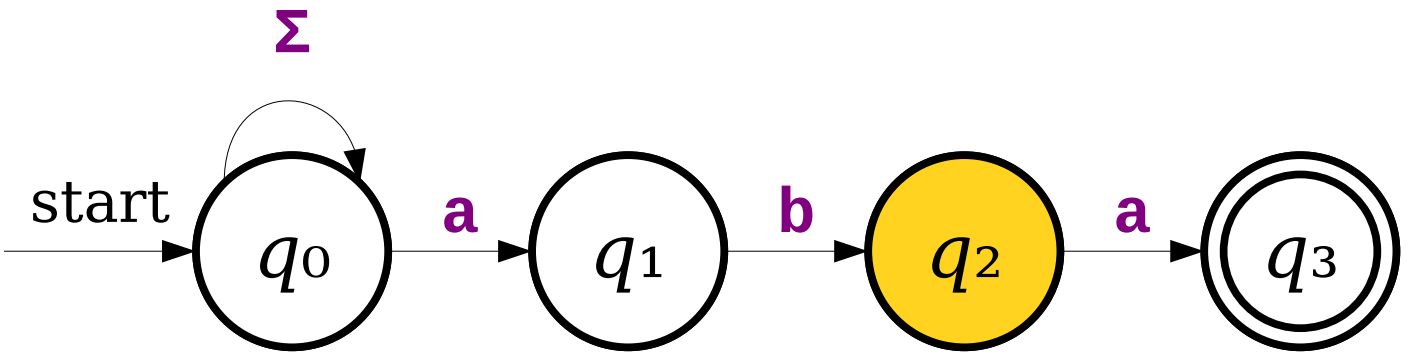


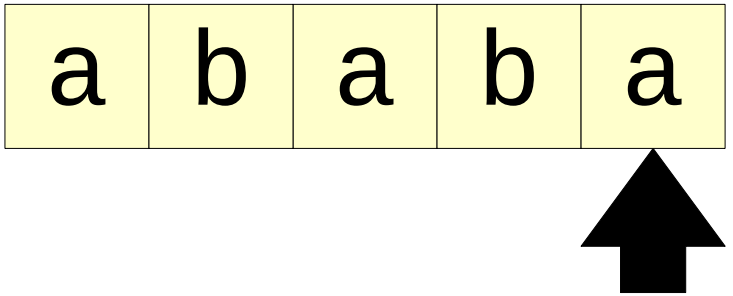
Perfect Positive Guessing



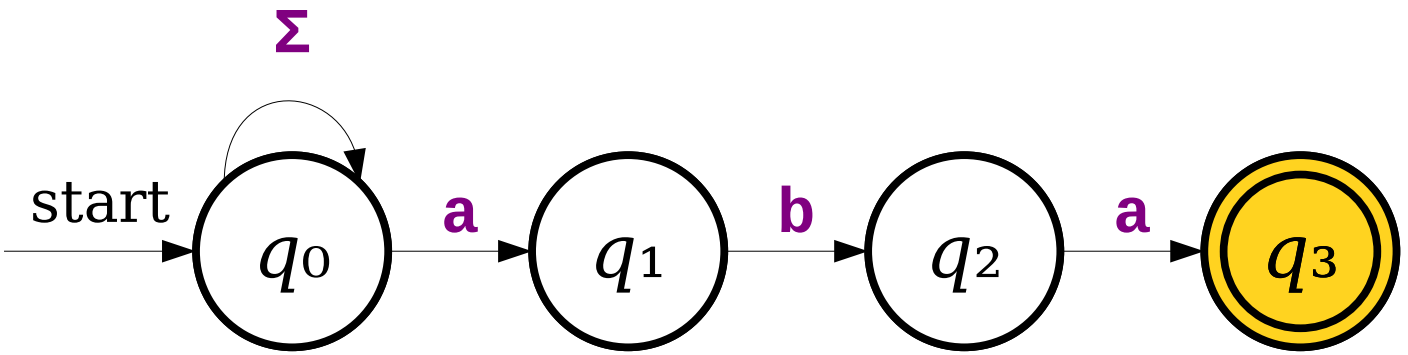


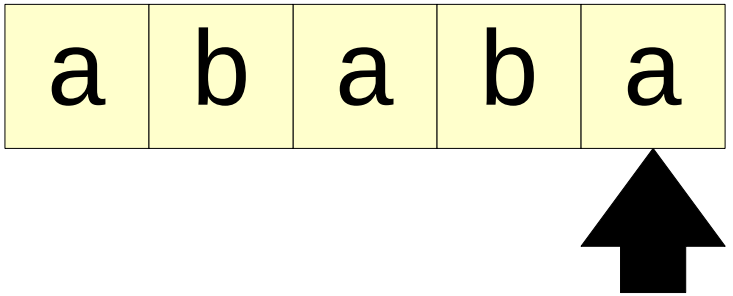
Perfect Positive Guessing



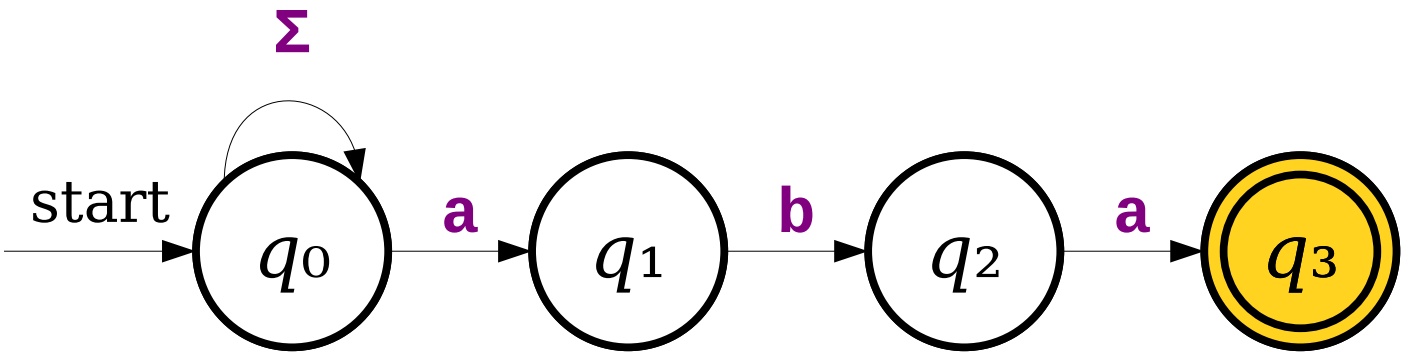


Perfect Positive Guessing





Perfect Positive Guessing

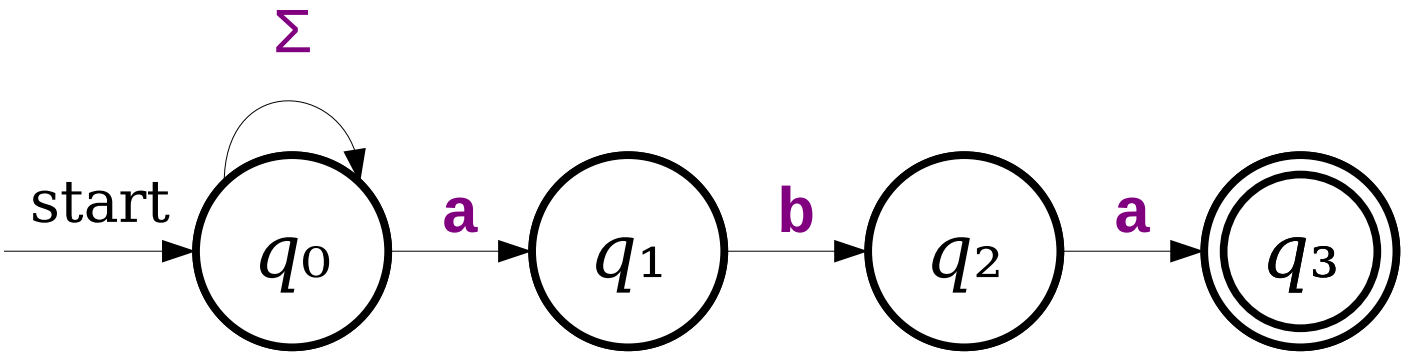


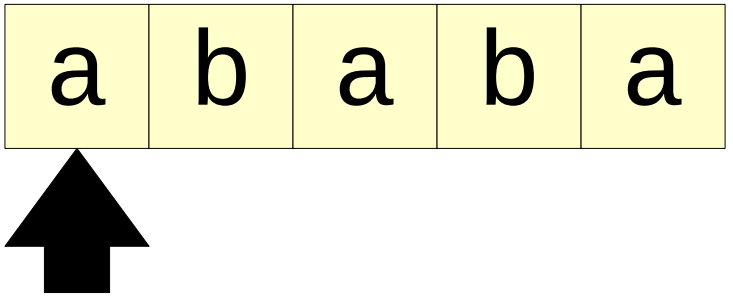
a	b	a	b	a

Perfect Positive Guessing

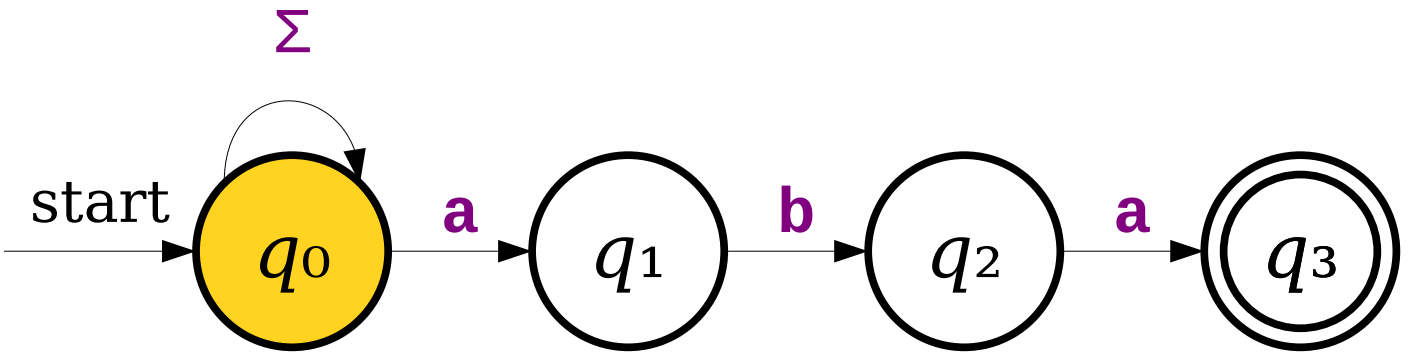
- We can view nondeterministic machines as having ***Magic Superpowers*** that enable them to guess choices that lead to an accepting state.
  - If there is at least one choice that leads to an accepting state, the machine will guess it.
  - If there are no choices, the machine guesses any one of the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

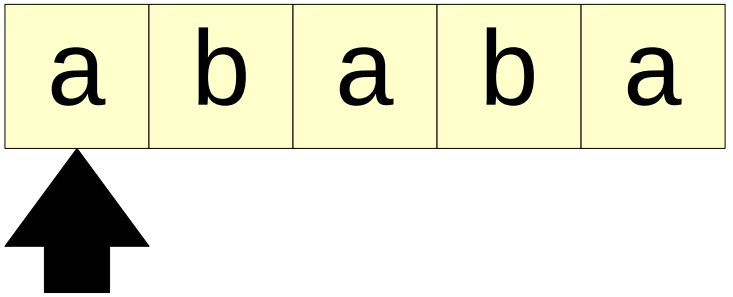
Massive Parallelism



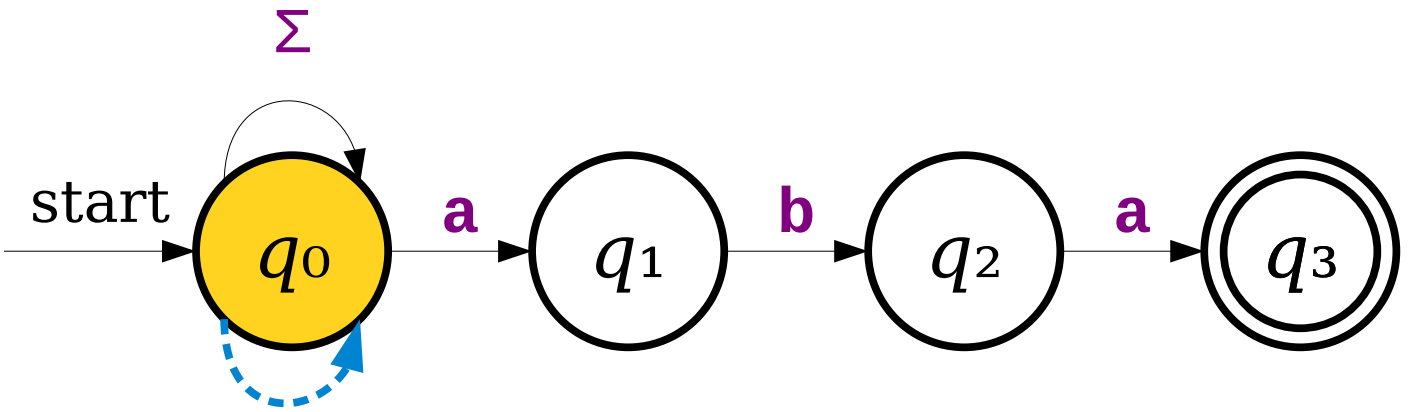


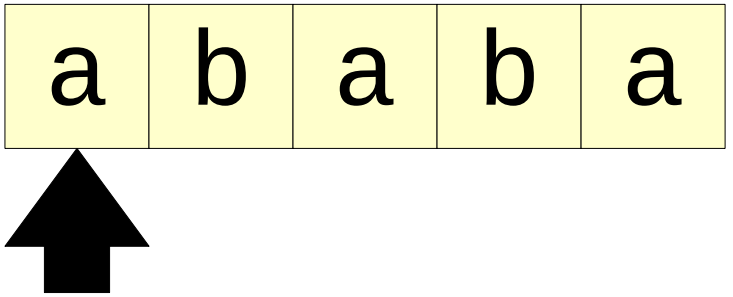
Massive Parallelism



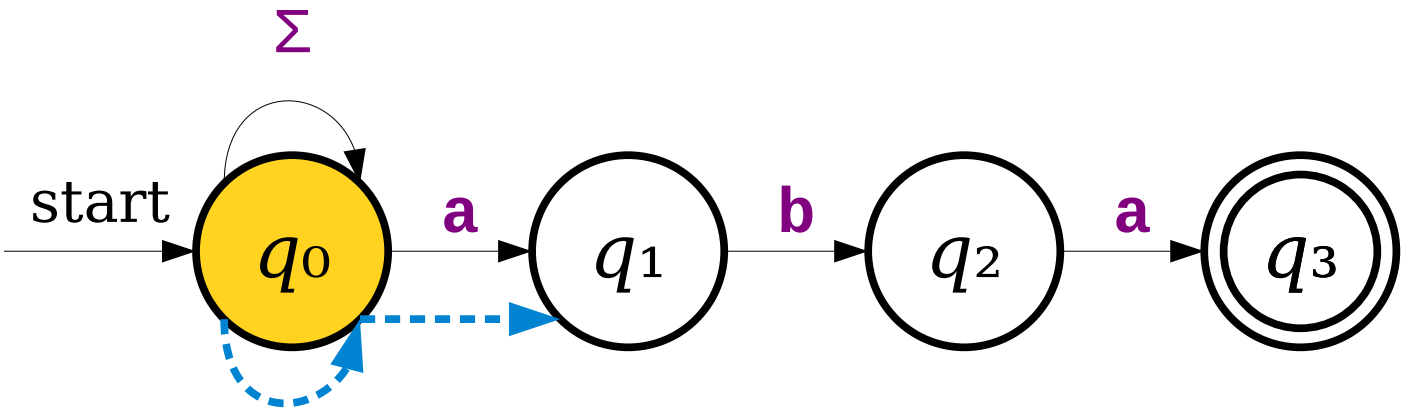


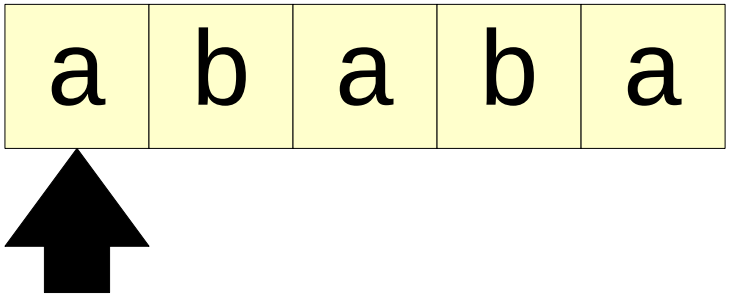
Massive Parallelism



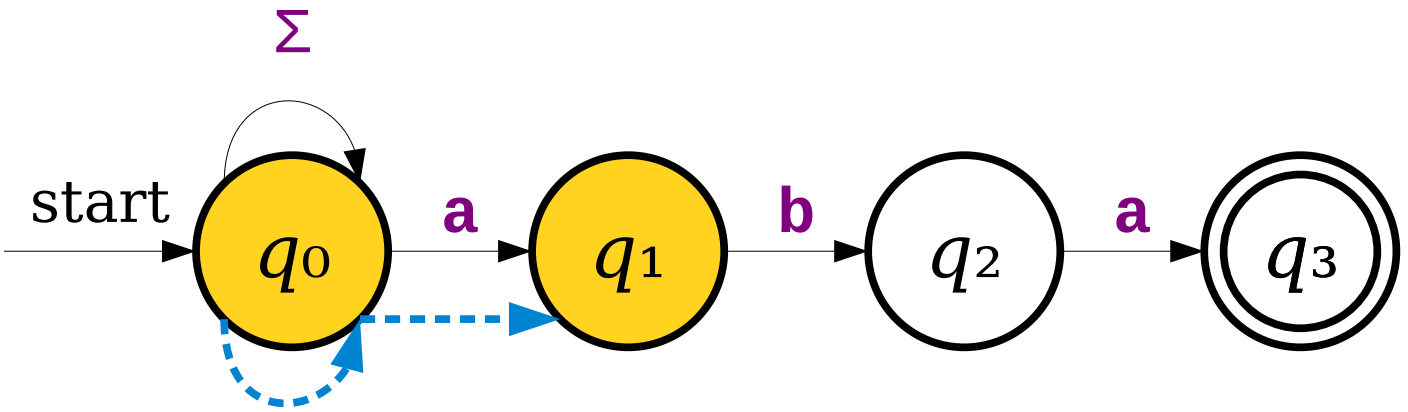


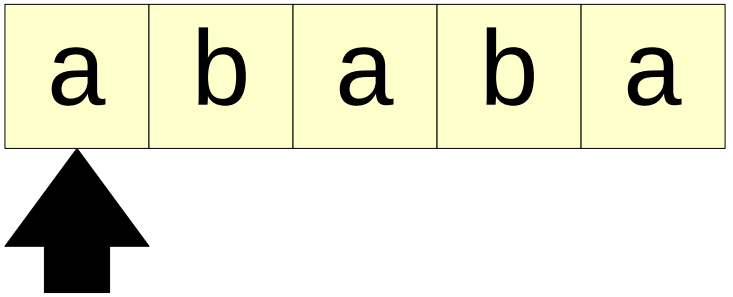
Massive Parallelism



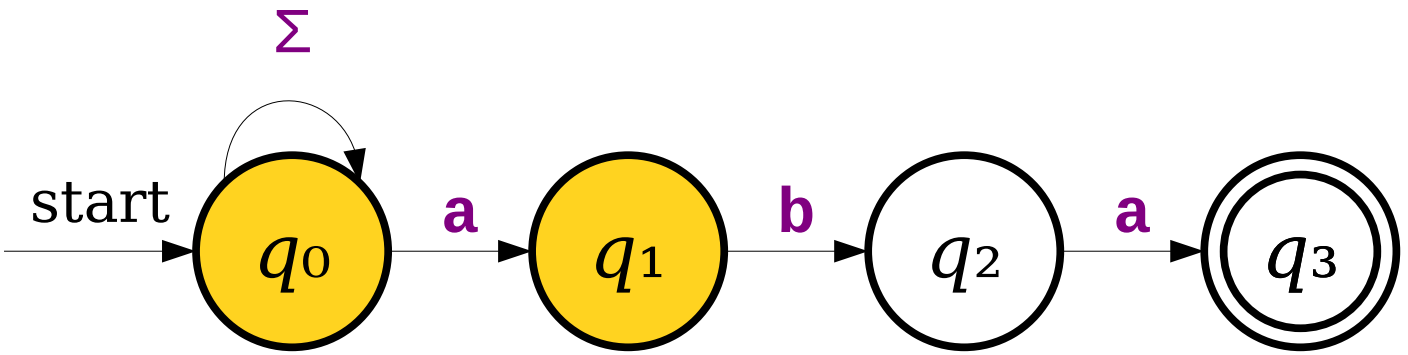


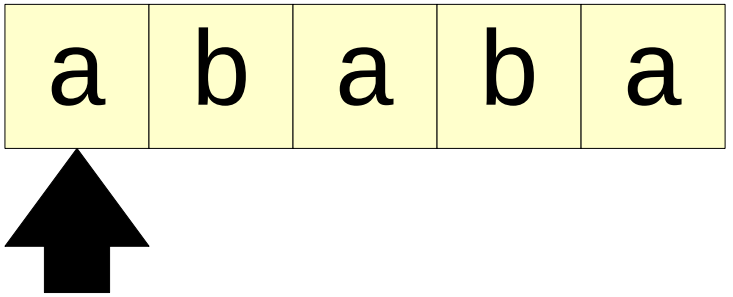
Massive Parallelism



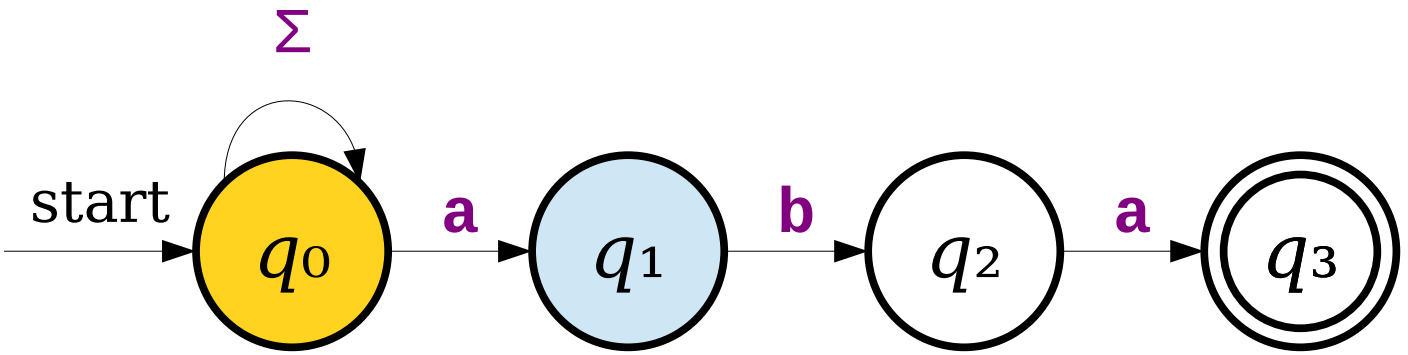


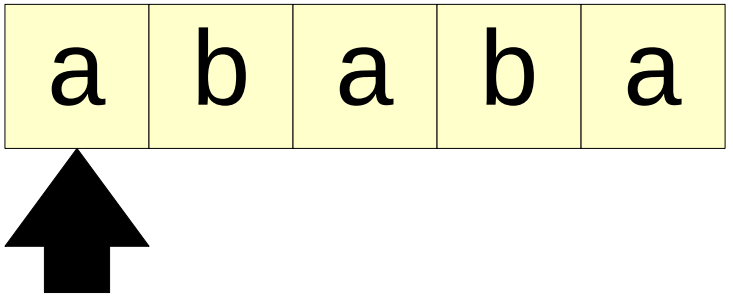
Massive Parallelism



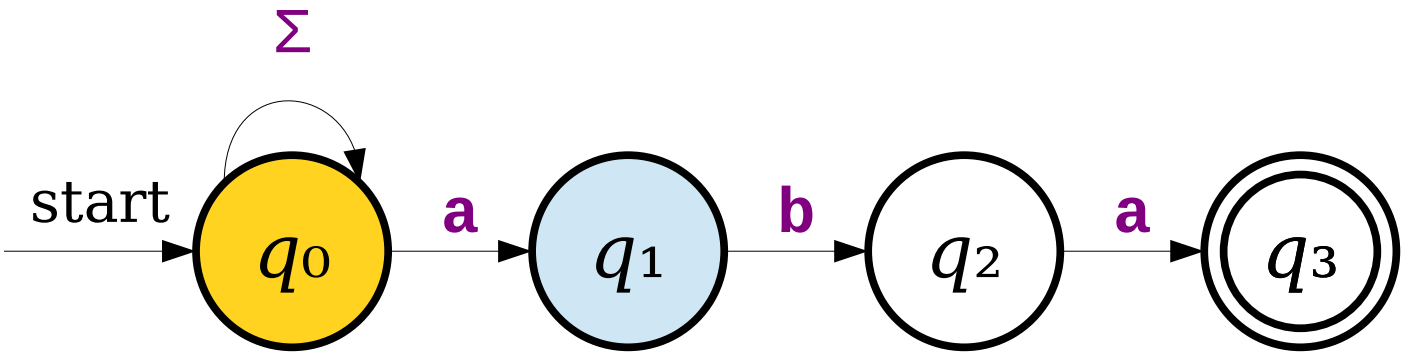


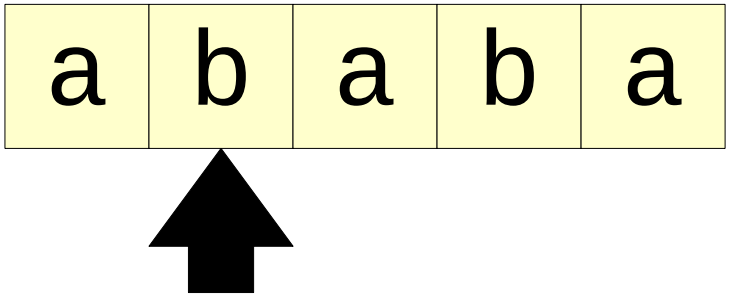
Massive Parallelism



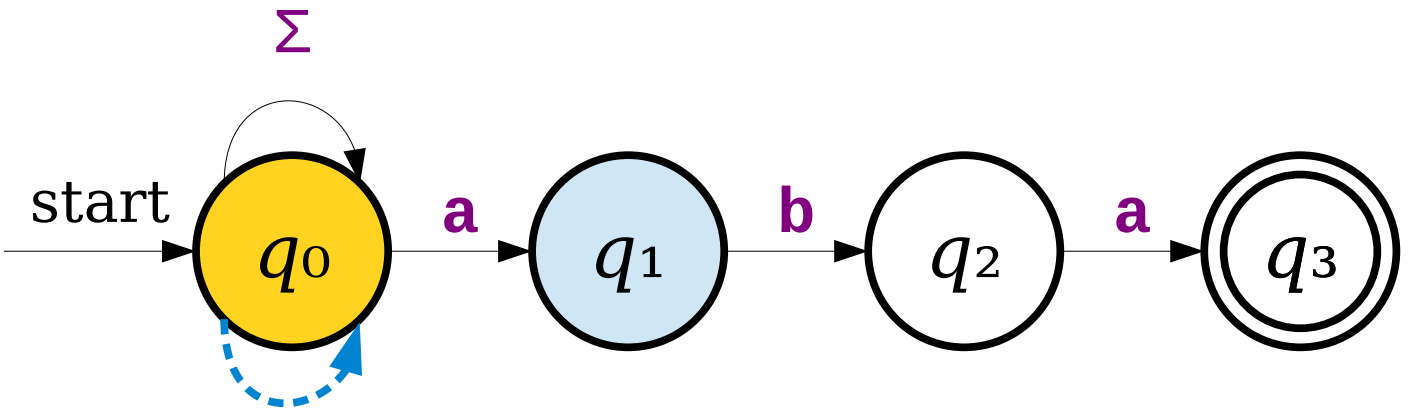


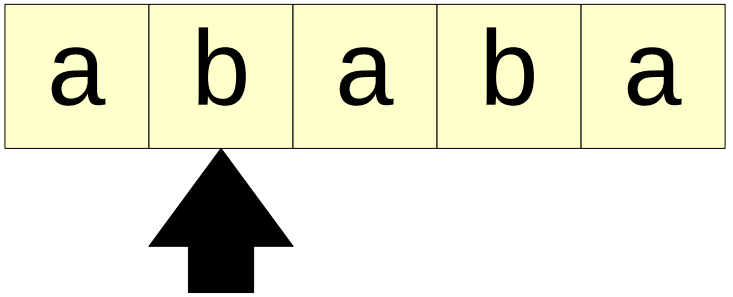
Massive Parallelism



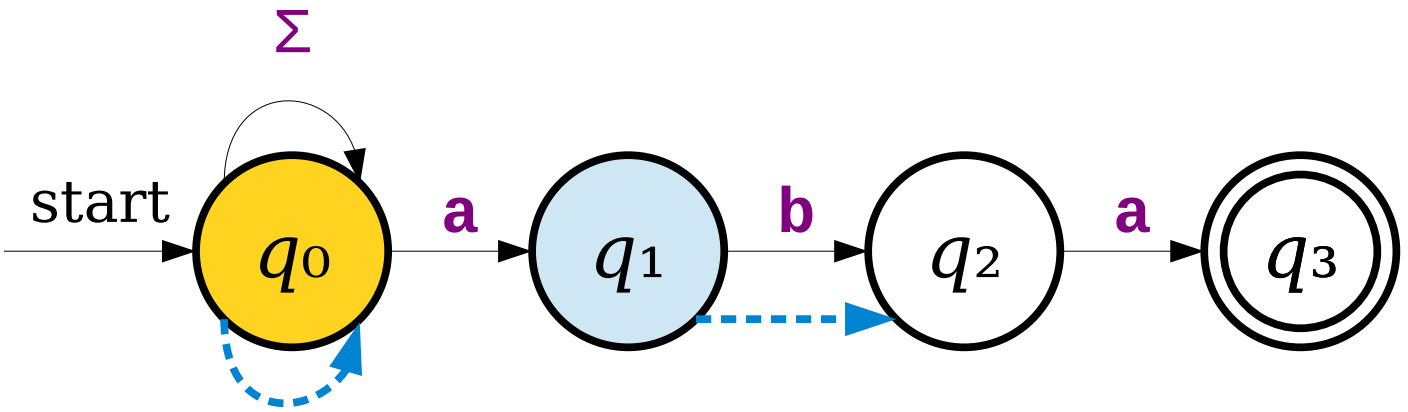


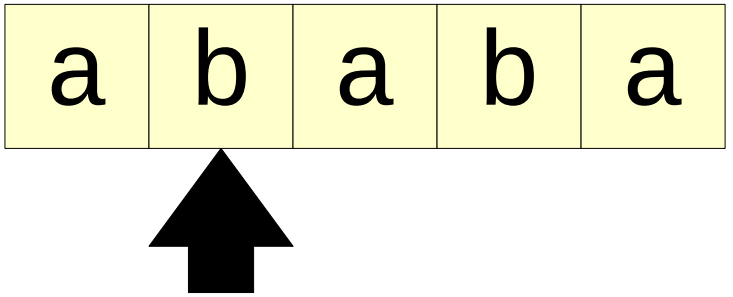
Massive Parallelism



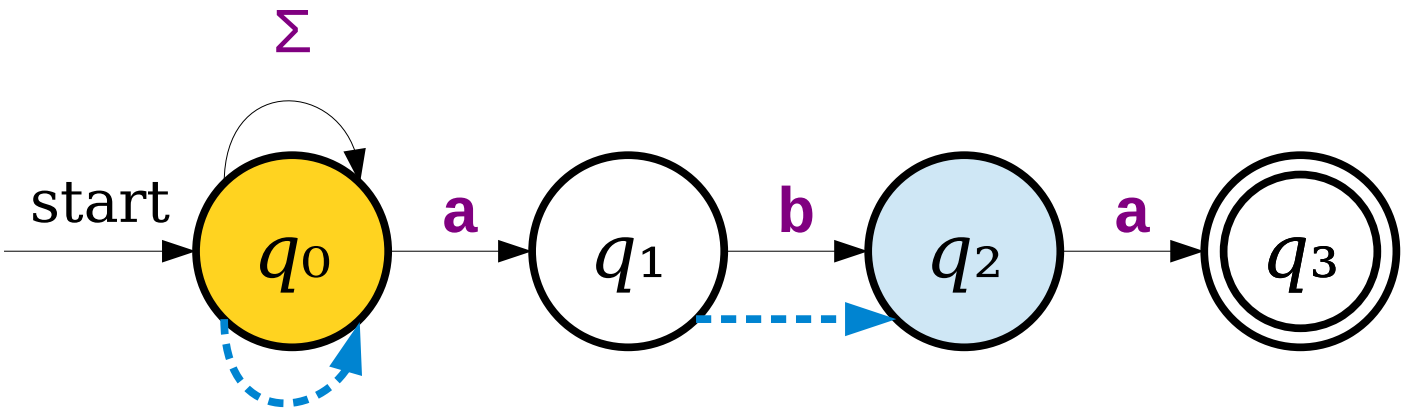


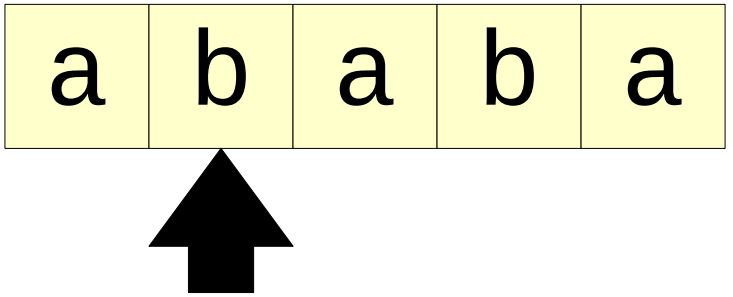
Massive Parallelism



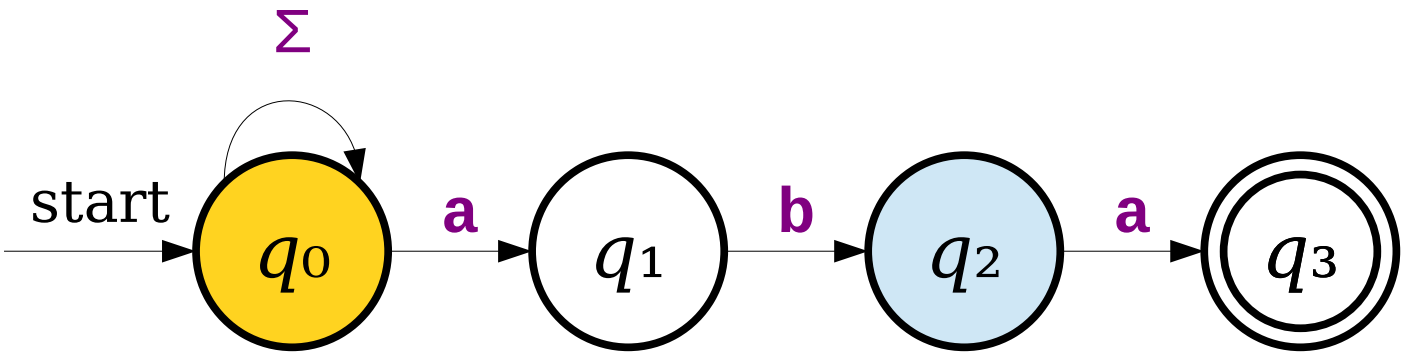


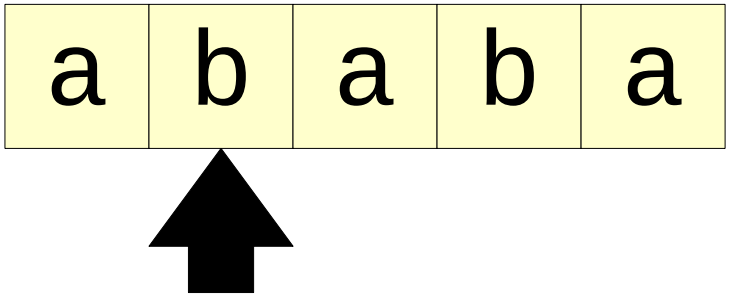
Massive Parallelism



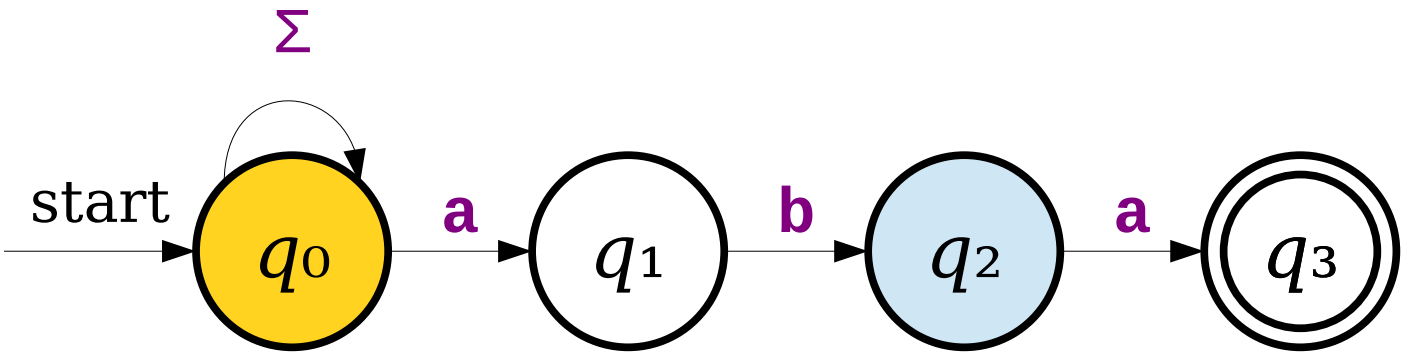


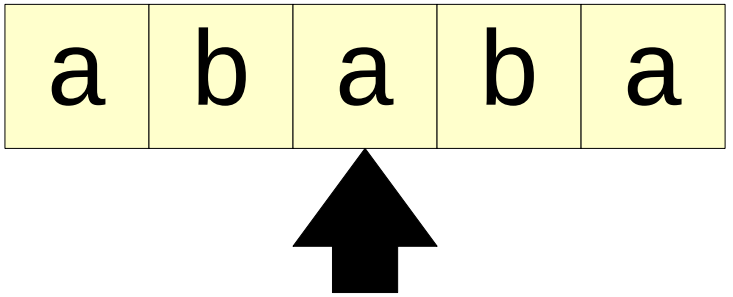
Massive Parallelism



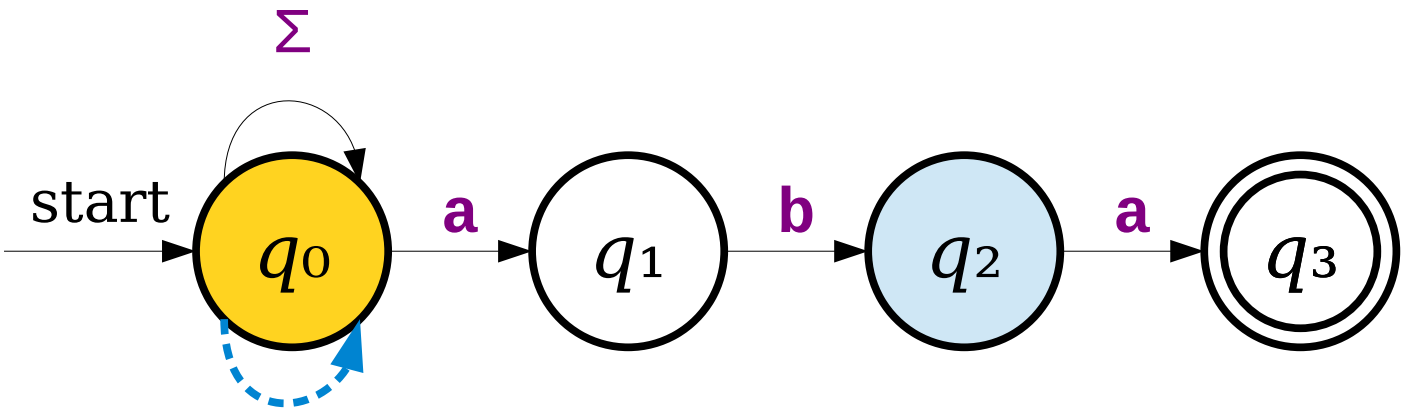


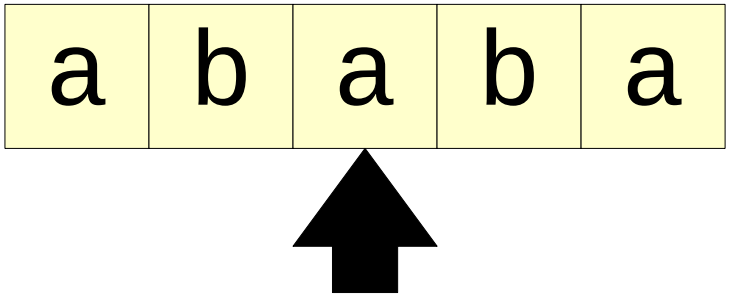
Massive Parallelism



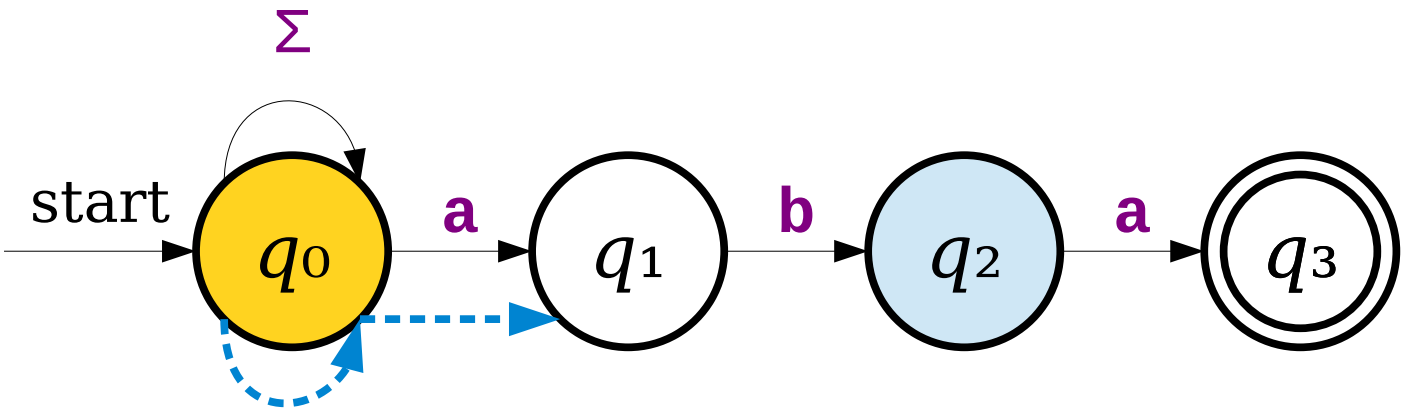


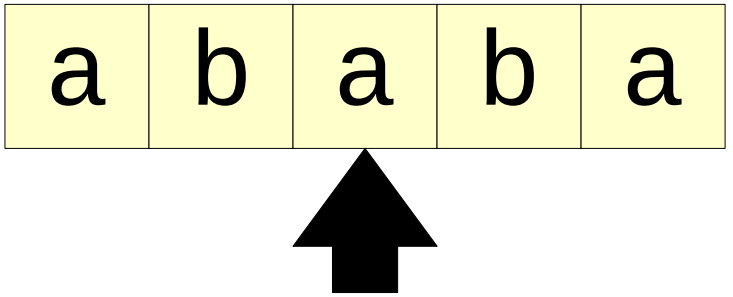
Massive Parallelism



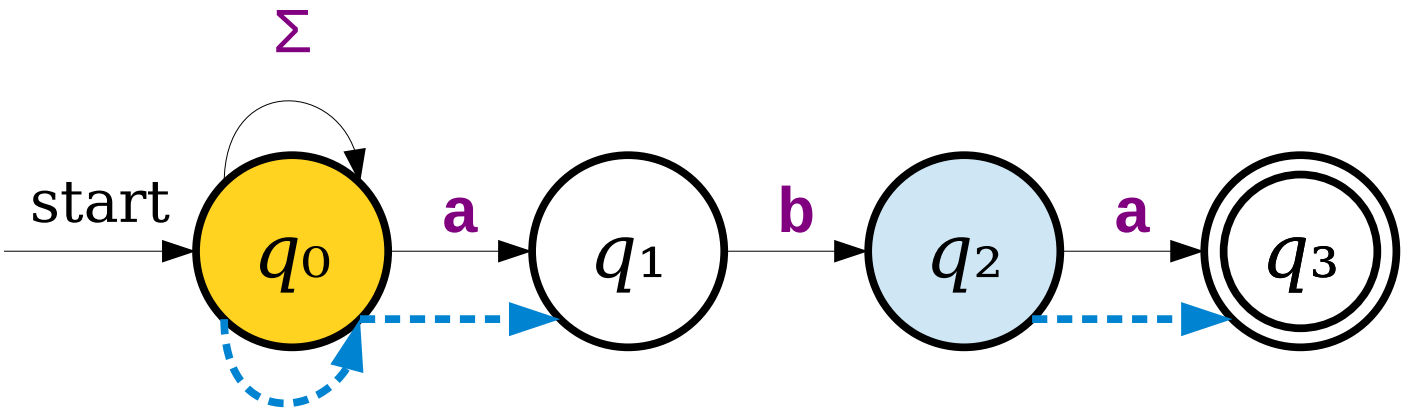


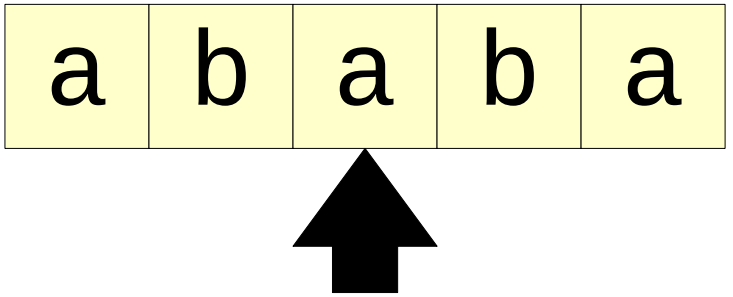
Massive Parallelism



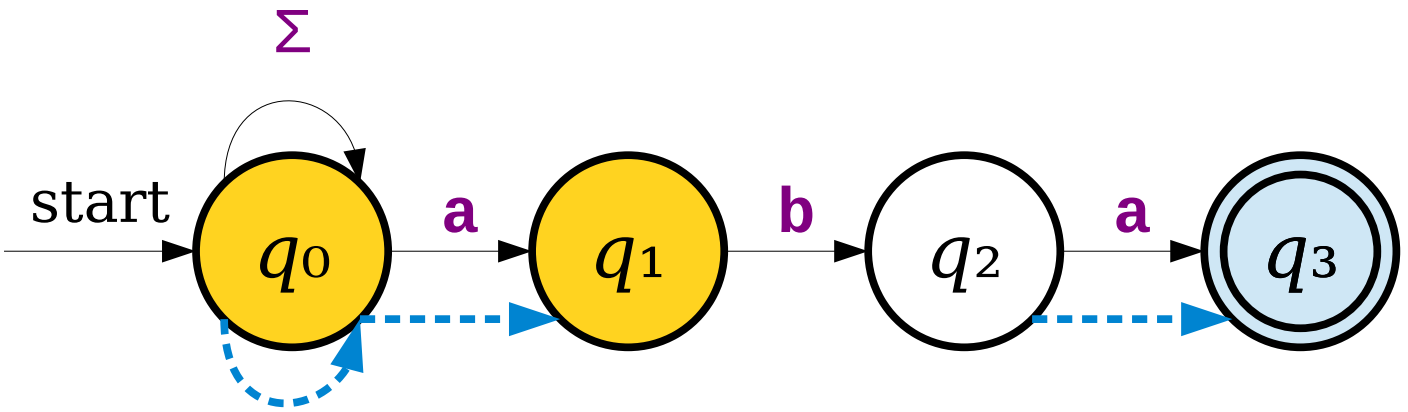


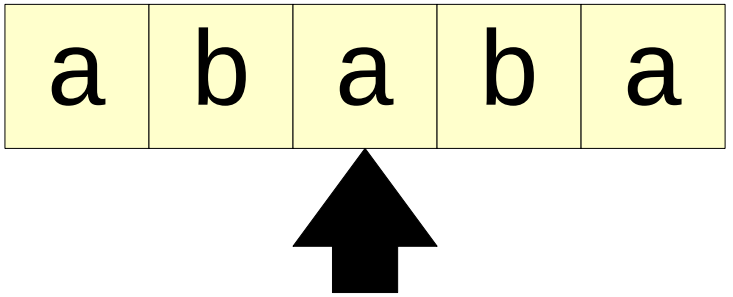
Massive Parallelism



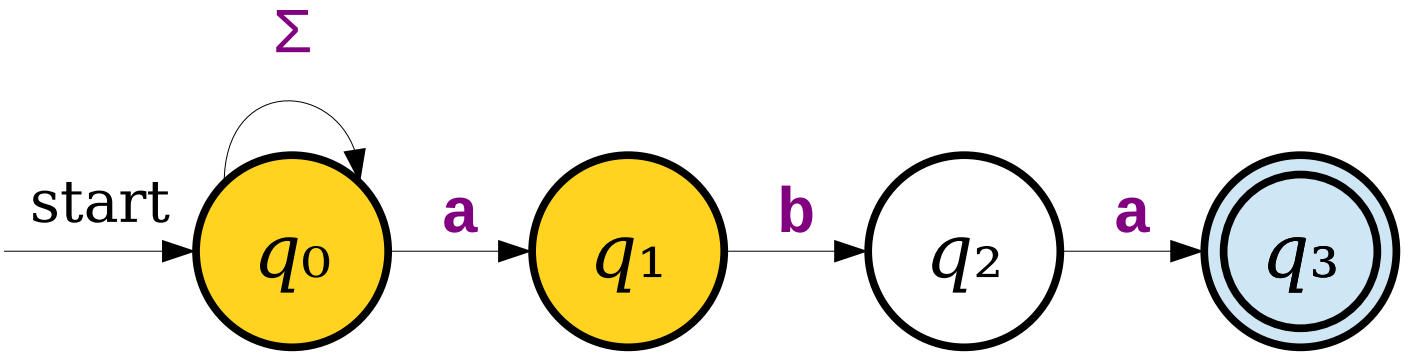


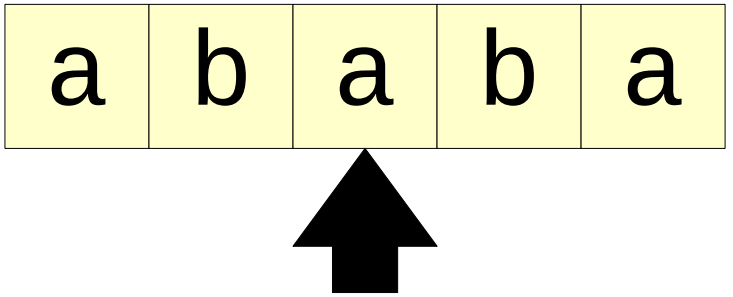
Massive Parallelism



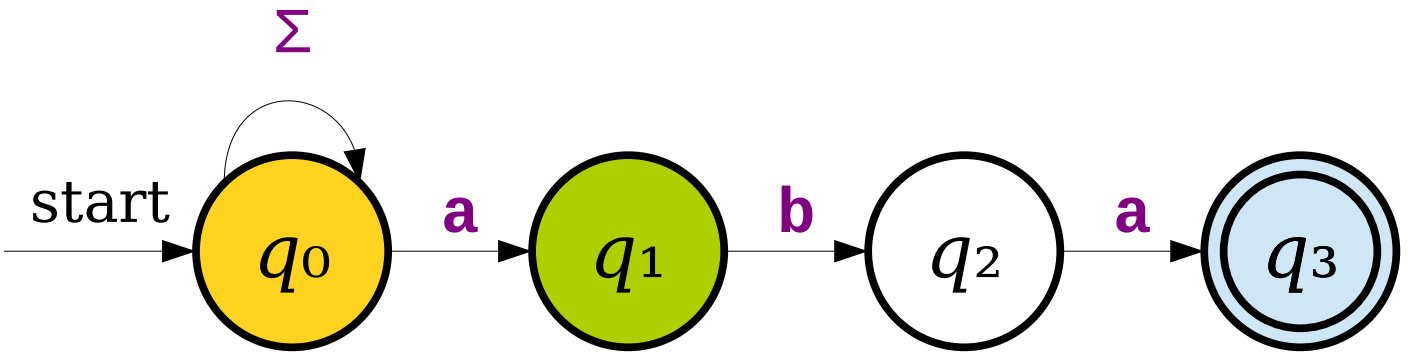


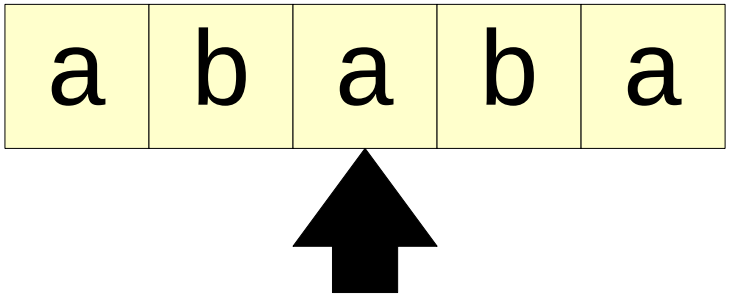
Massive Parallelism



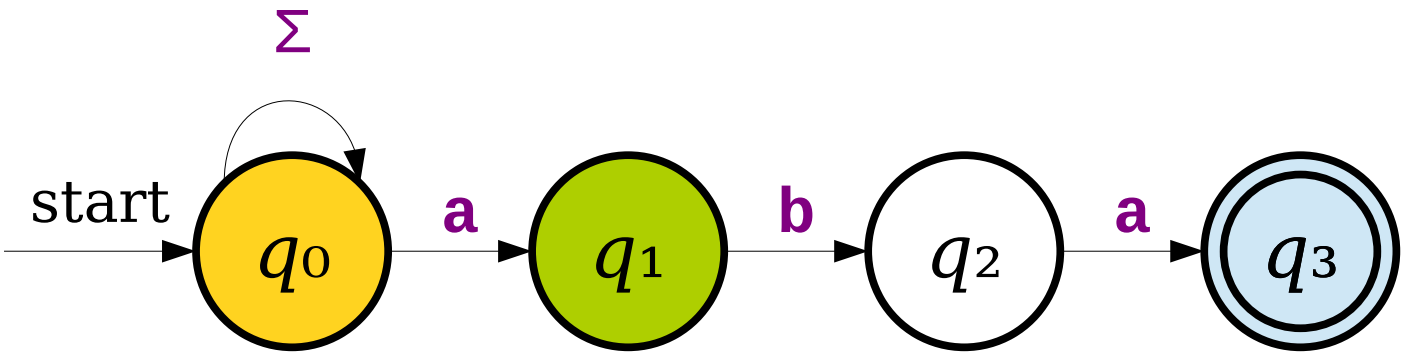


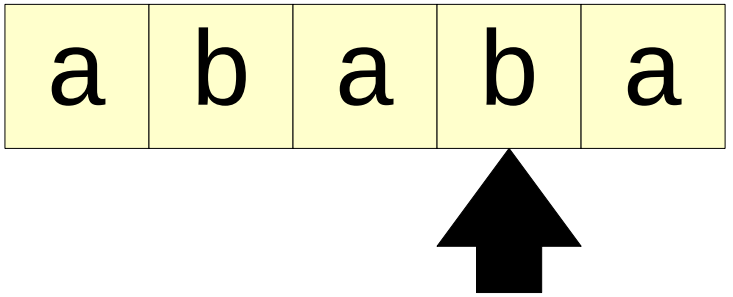
Massive Parallelism



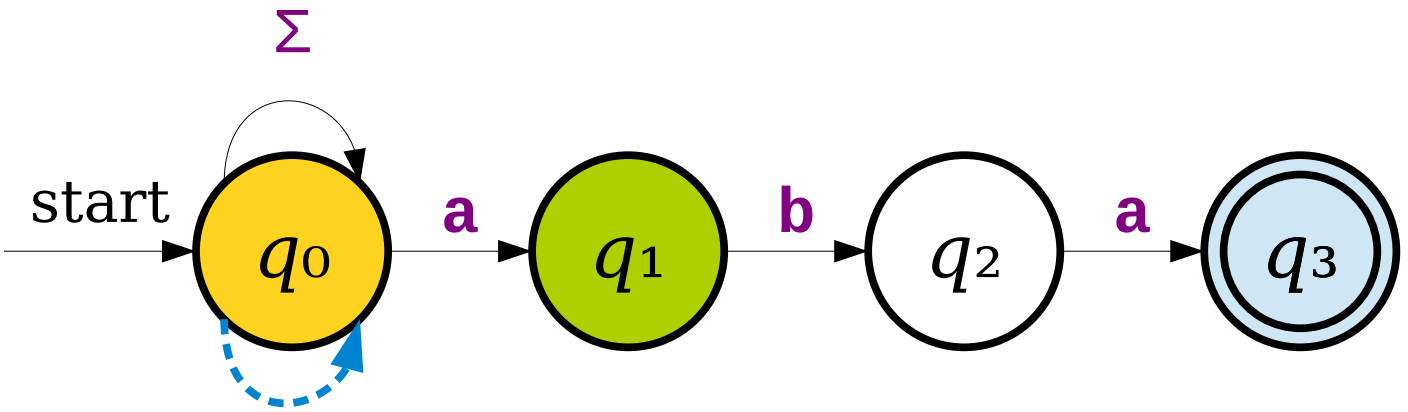


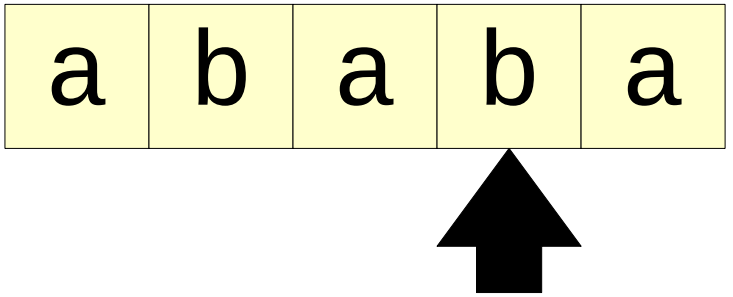
Massive Parallelism



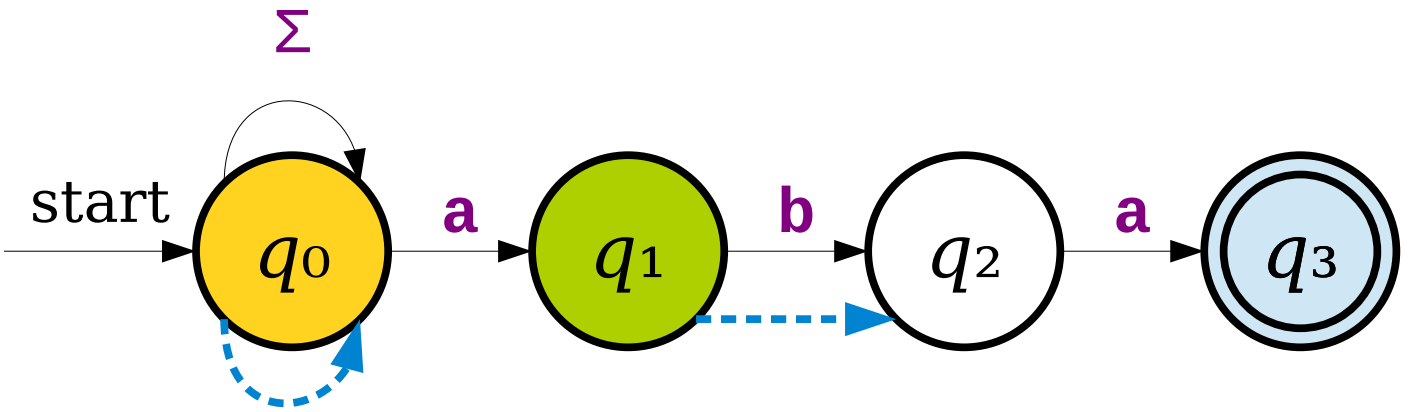


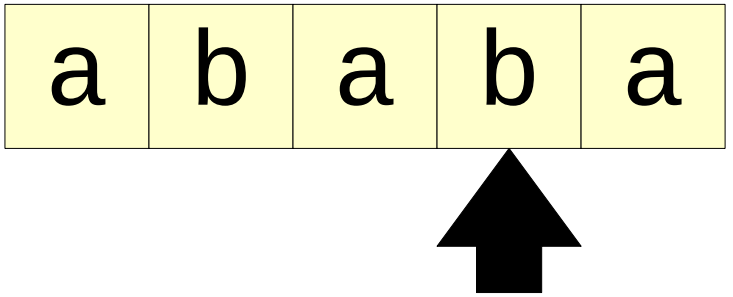
Massive Parallelism



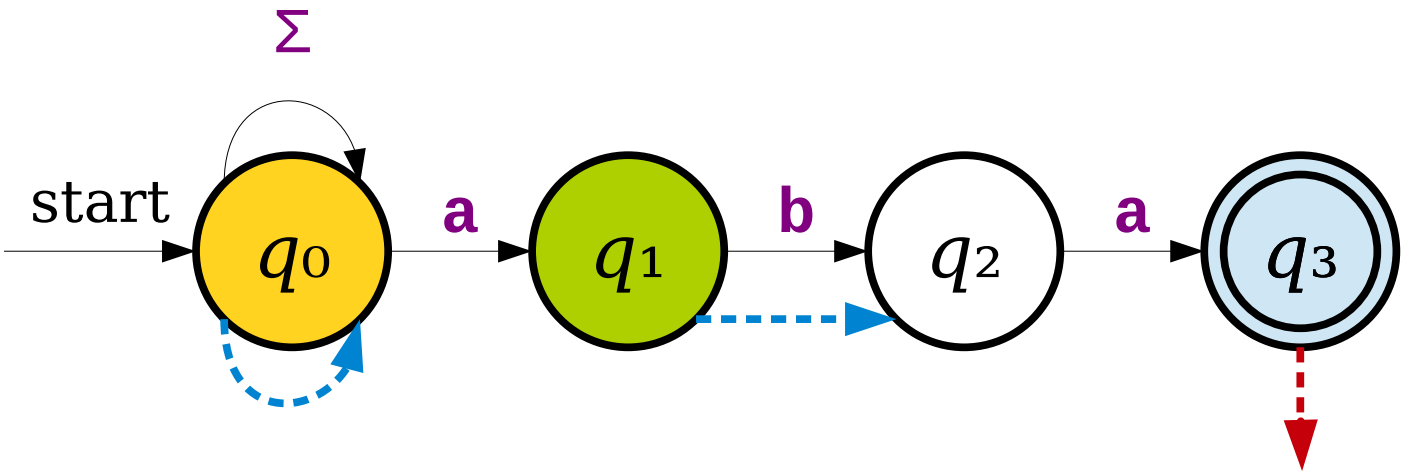


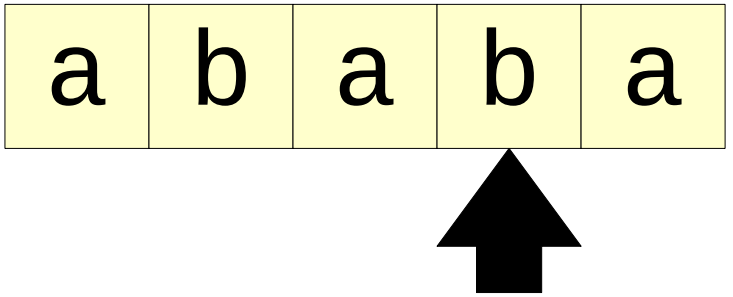
Massive Parallelism



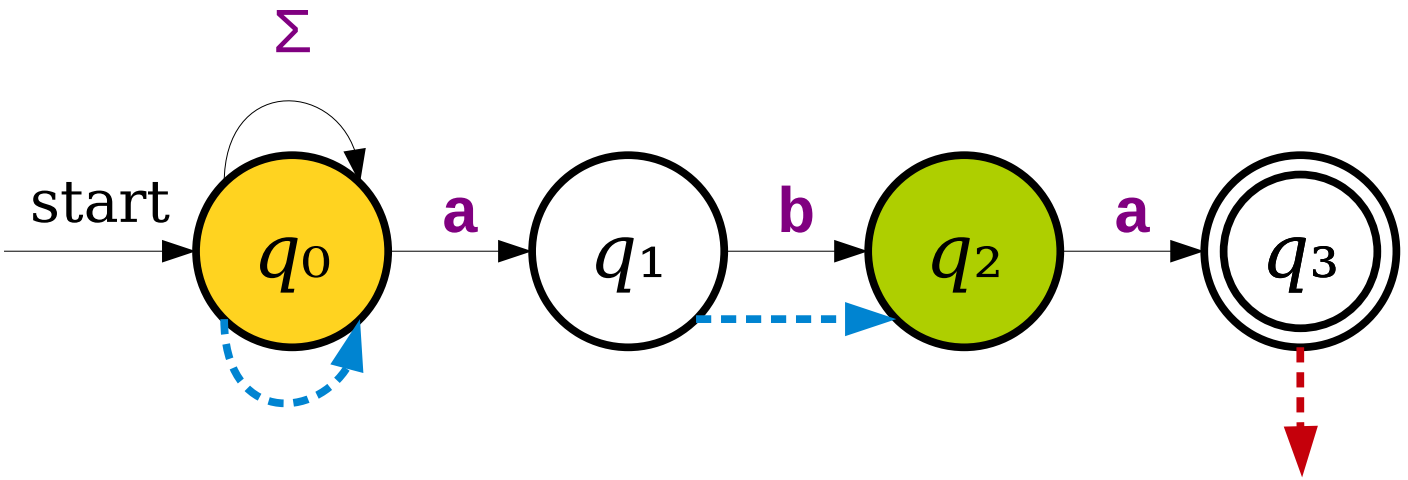


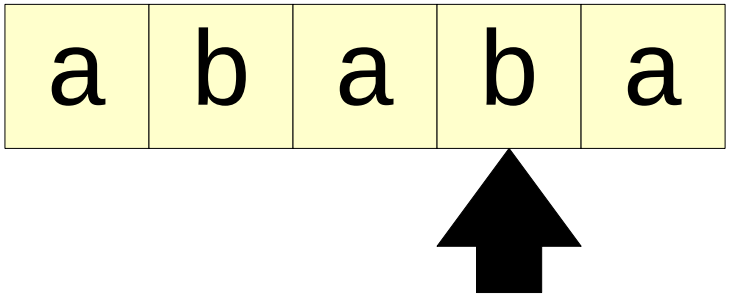
Massive Parallelism



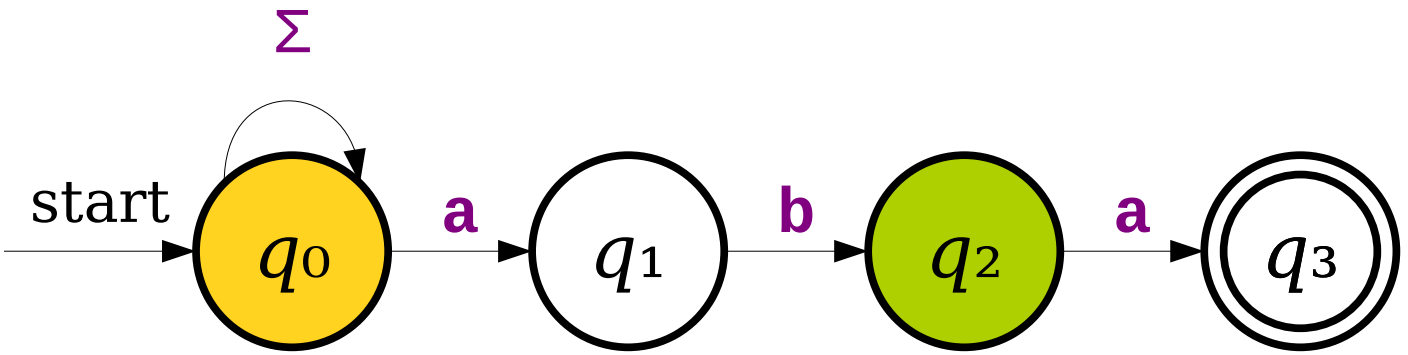


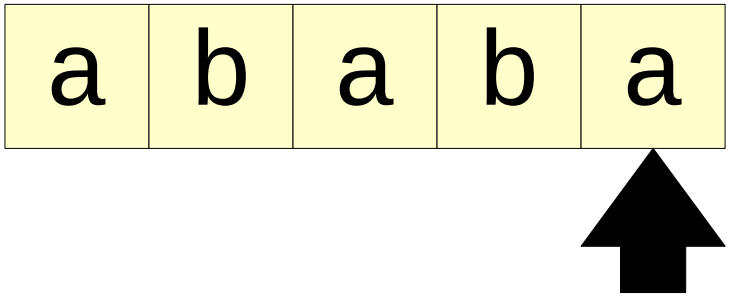
Massive Parallelism





Massive Parallelism

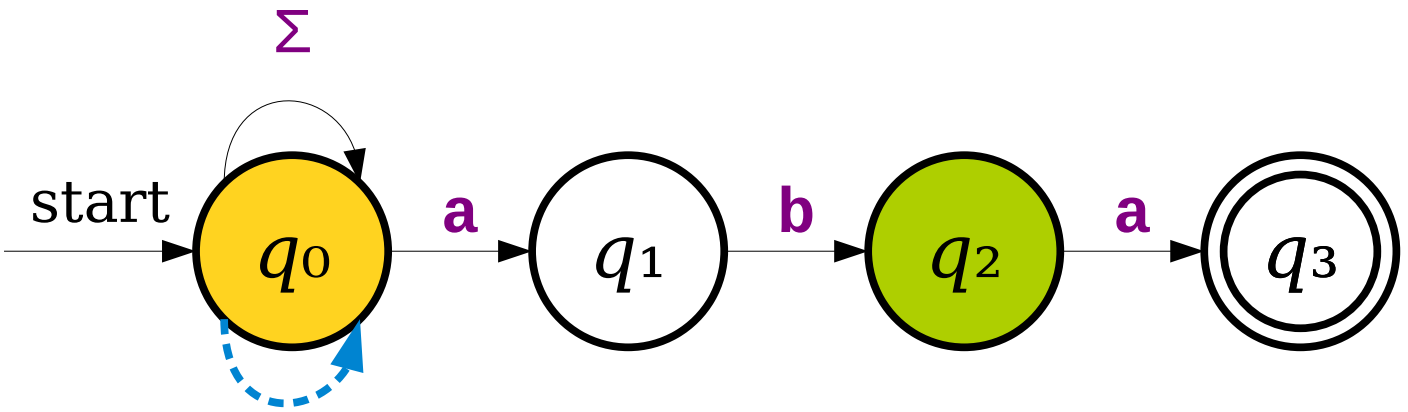


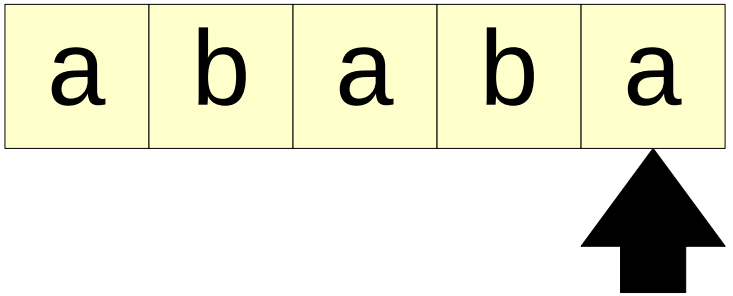


Using the massive parallelism intuition, if we are in the states q_0 and q_2, what set of states will we be in after reading the character **a**?

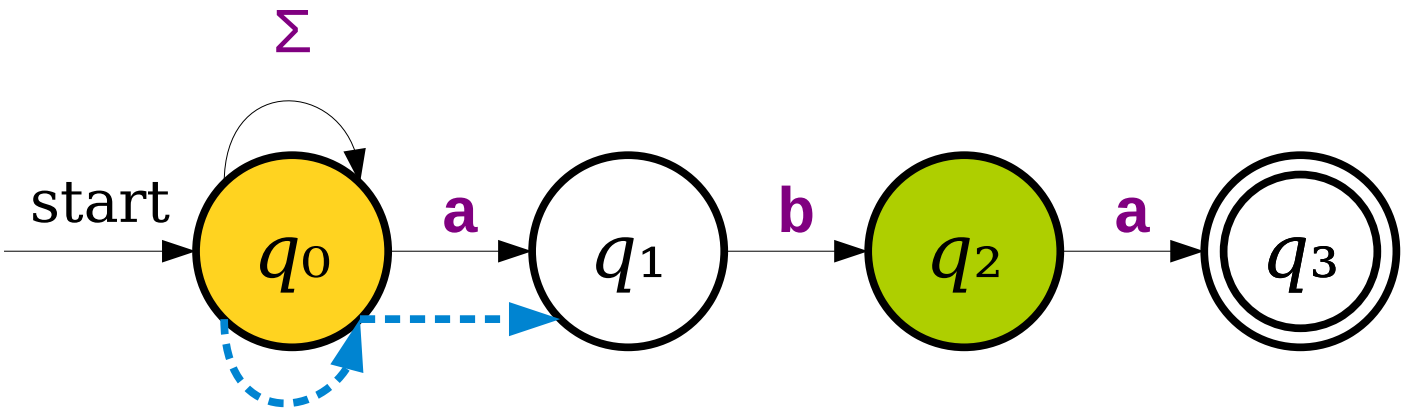
Respond at pollev.com/zhenglian740

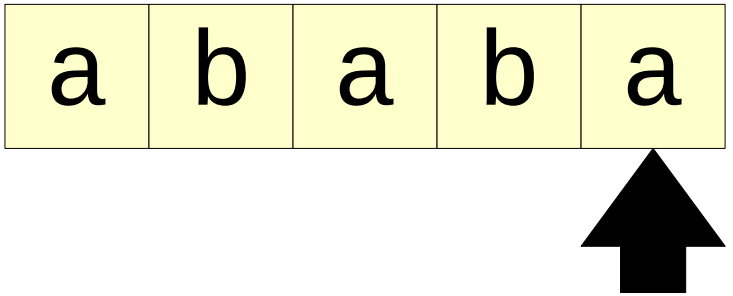
Massive Parallelism



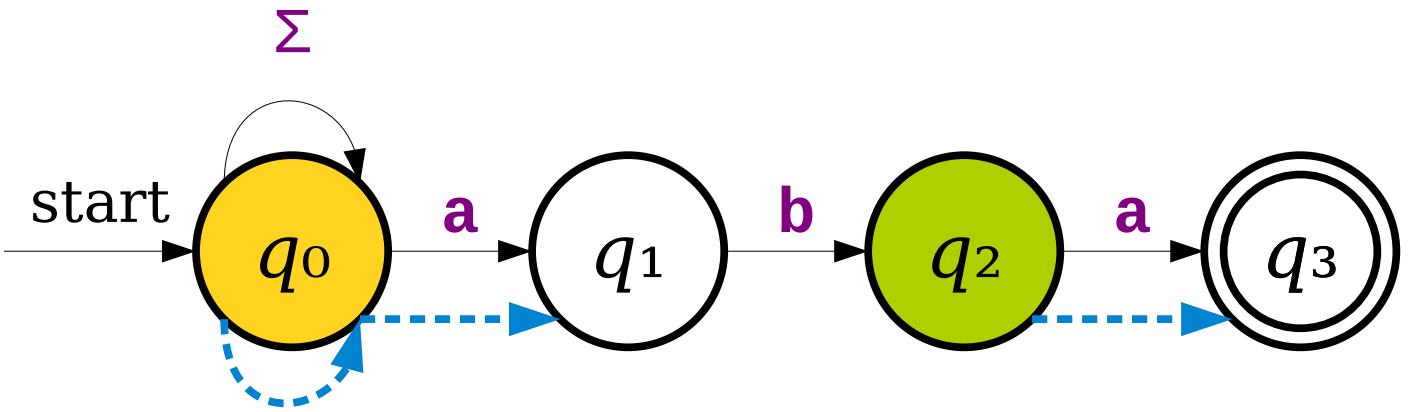


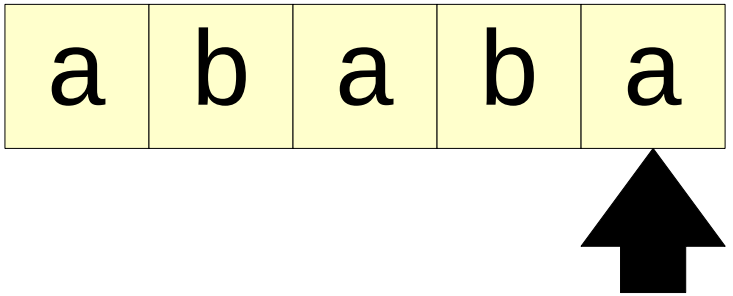
Massive Parallelism



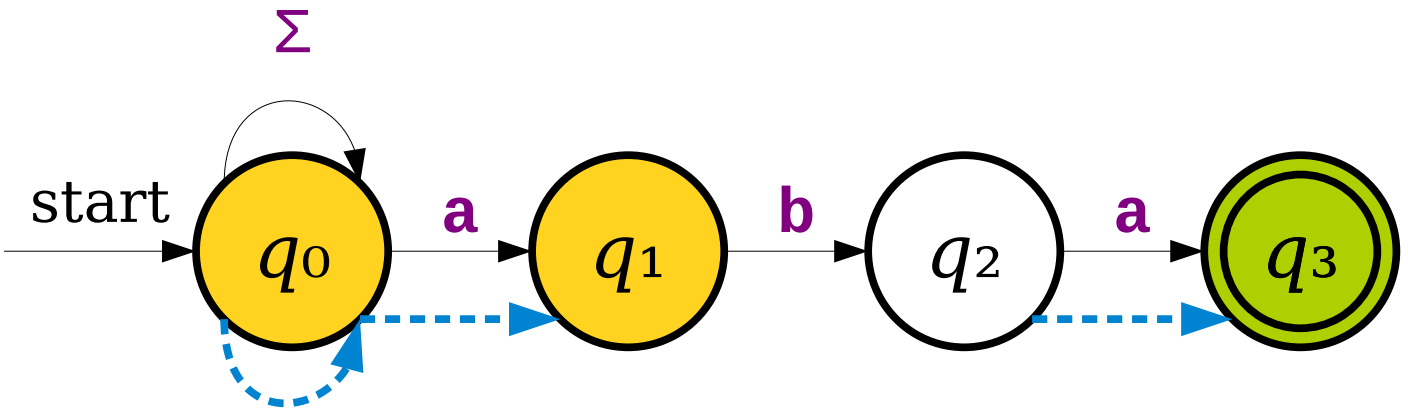


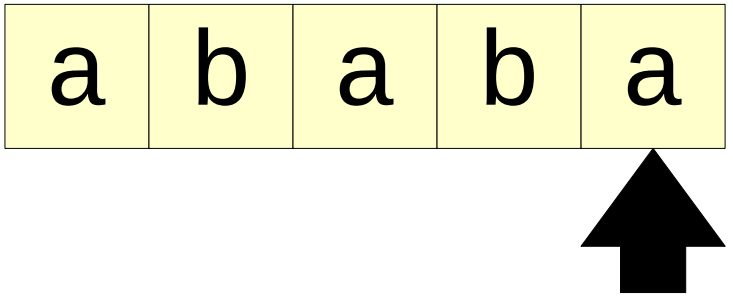
Massive Parallelism



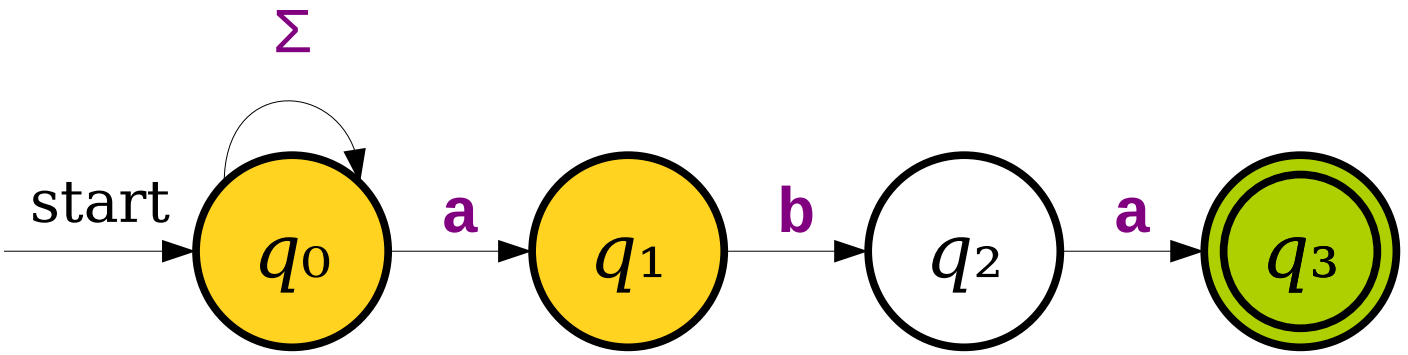


Massive Parallelism



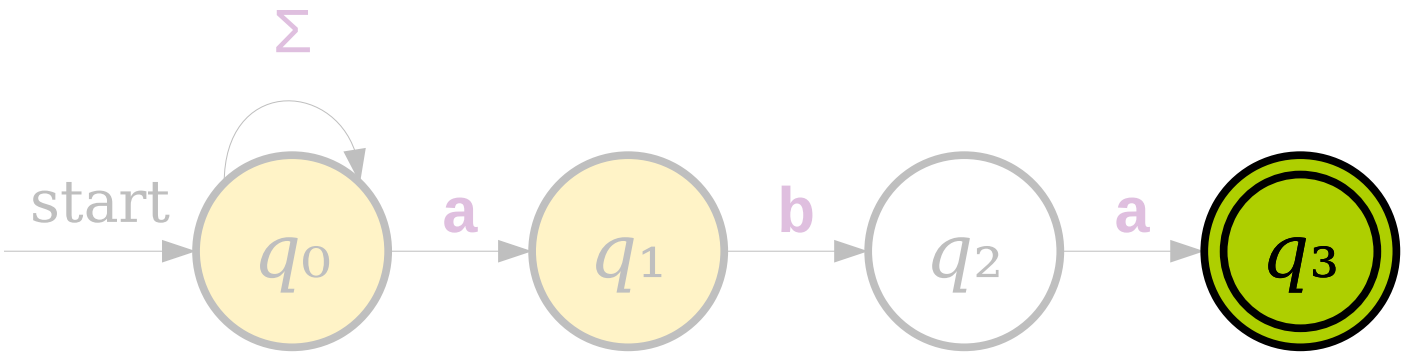


Massive Parallelism



a	b	a	b	a

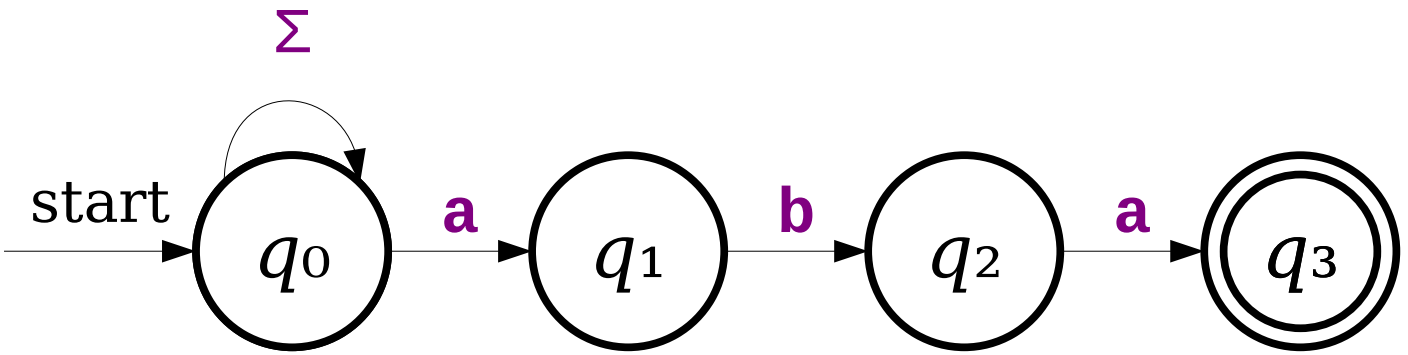
Massive Parallelism

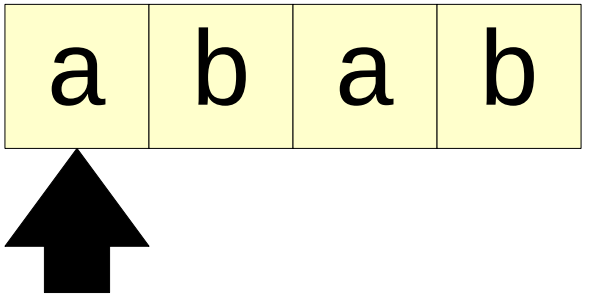


We're in at least one accepting state, so there's some path that gets us to an accepting state.

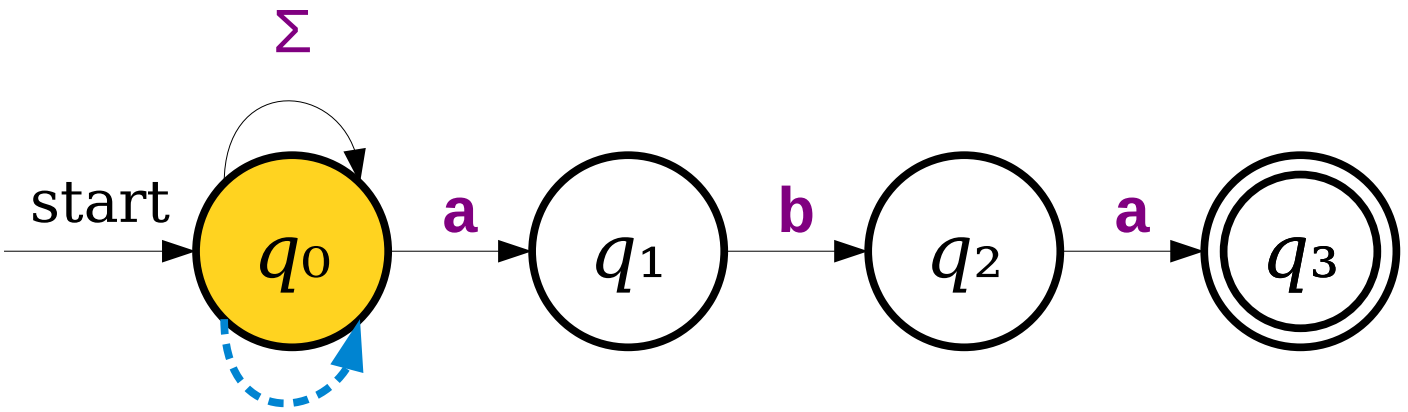
a	b	a	b	a

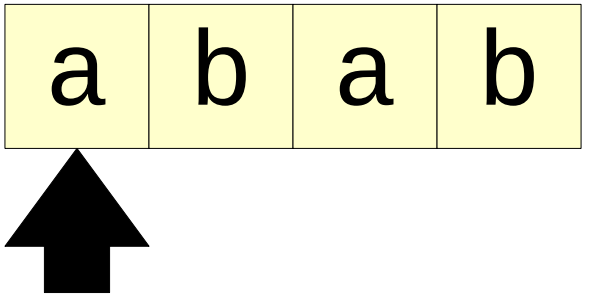
Massive Parallelism



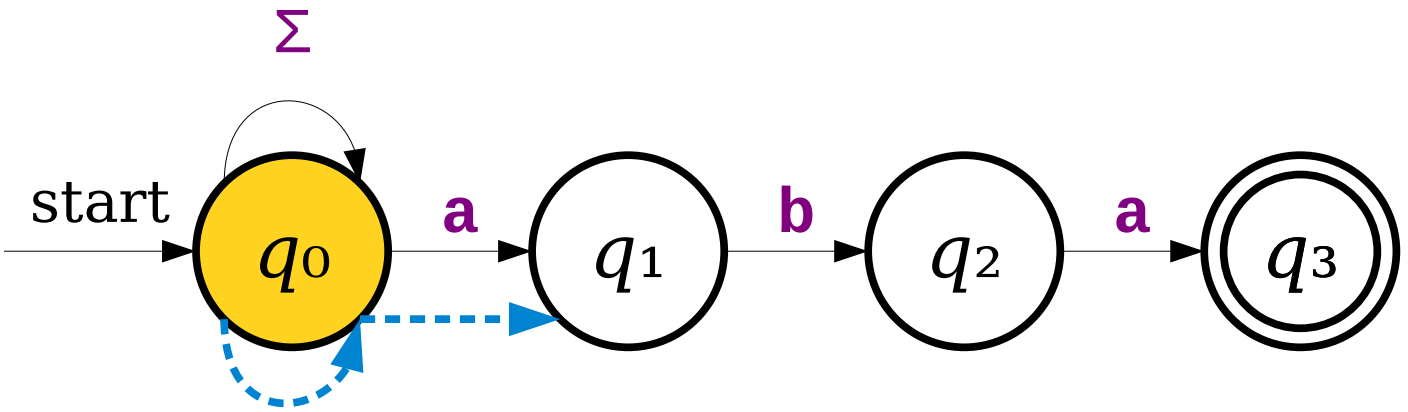


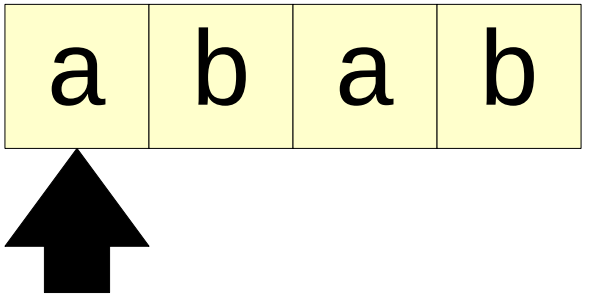
Massive Parallelism



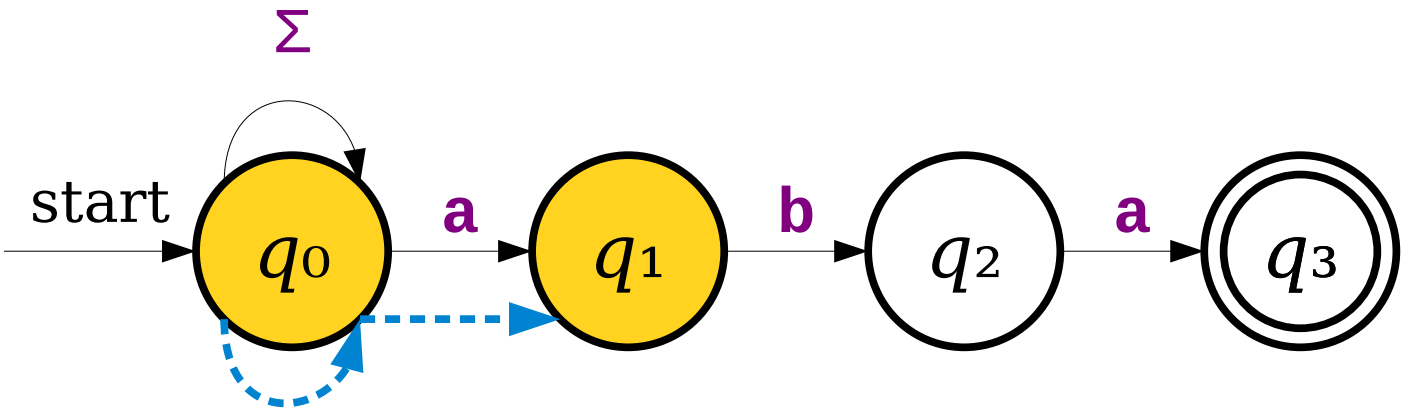


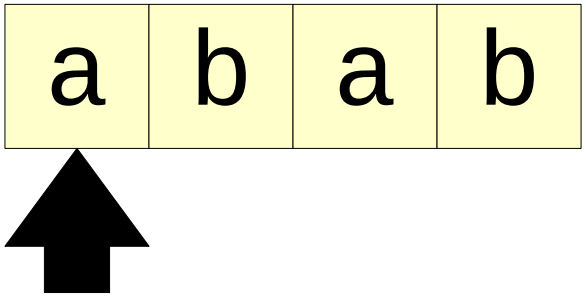
Massive Parallelism



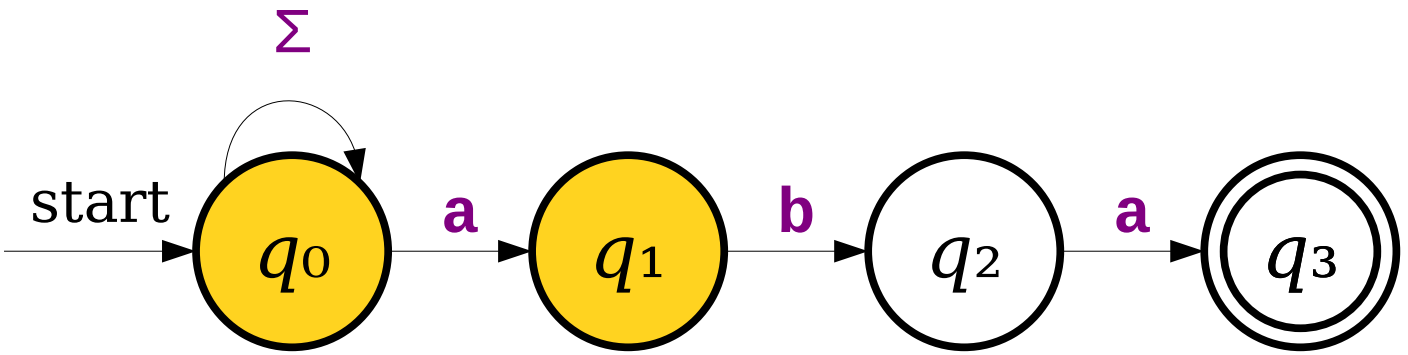


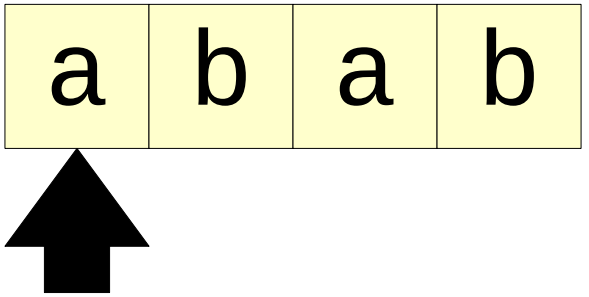
Massive Parallelism



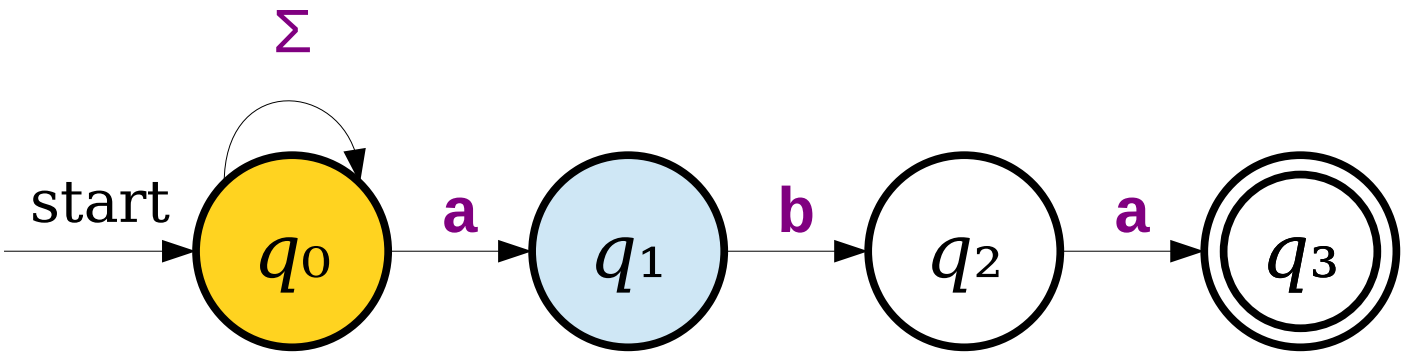


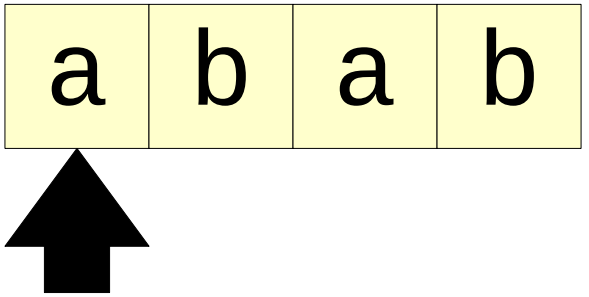
Massive Parallelism



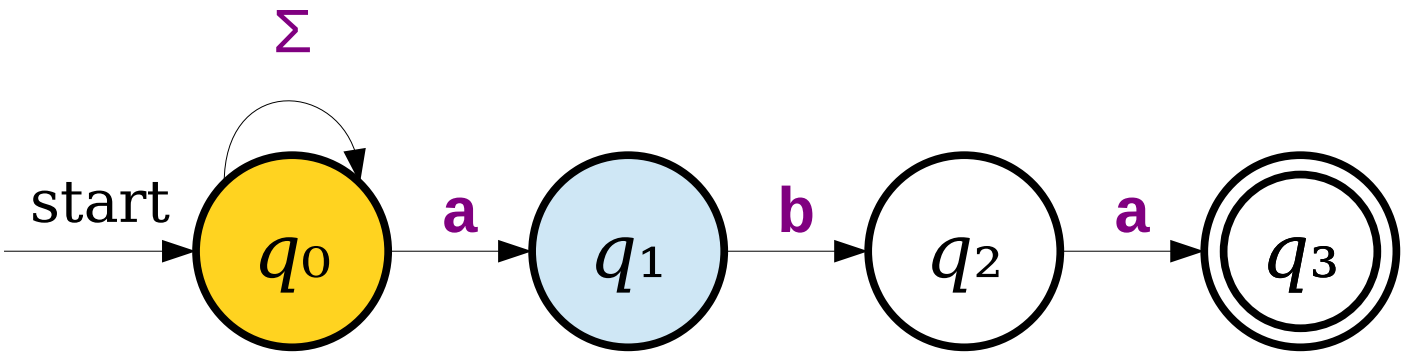


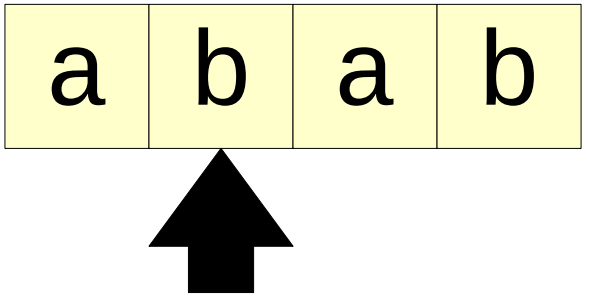
Massive Parallelism



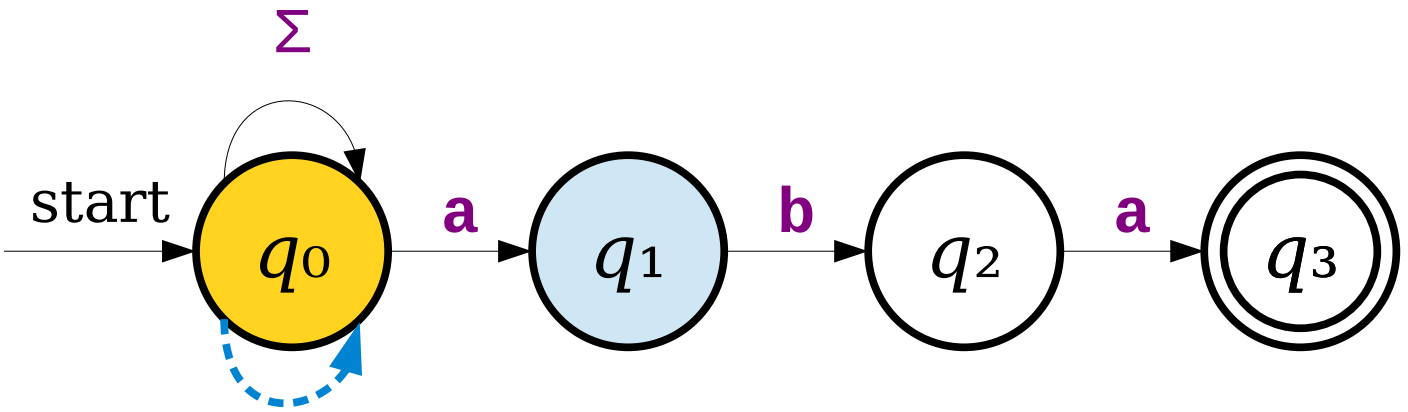


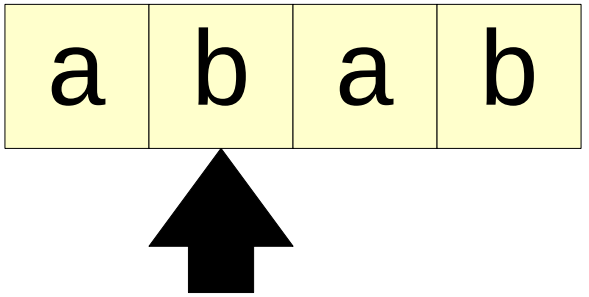
Massive Parallelism



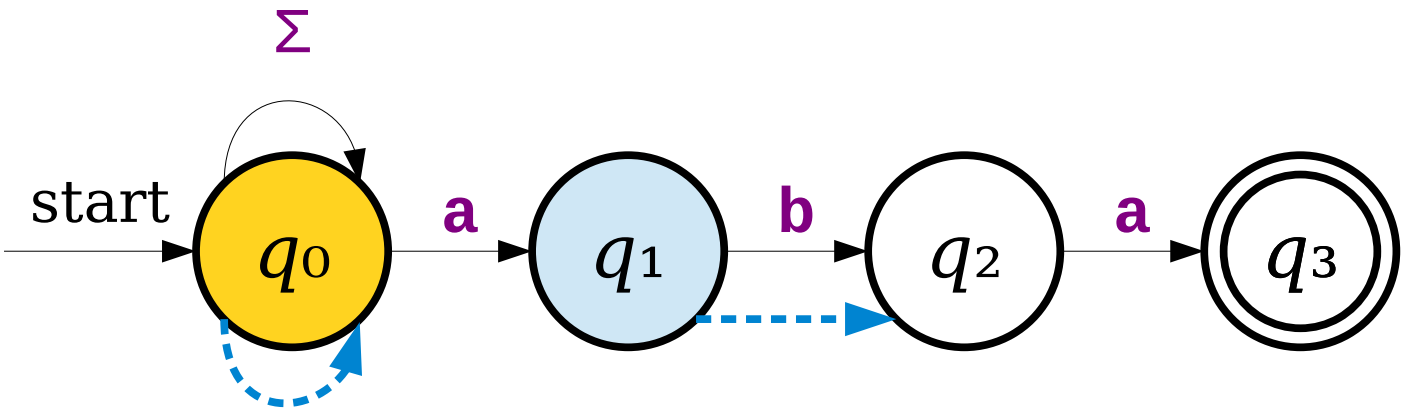


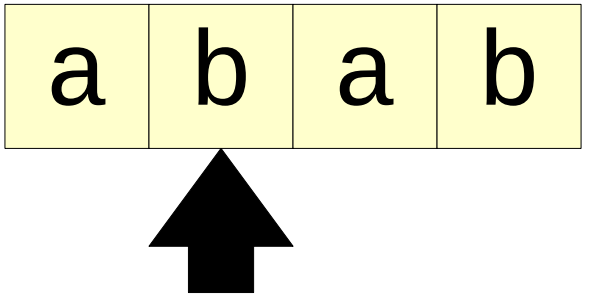
Massive Parallelism



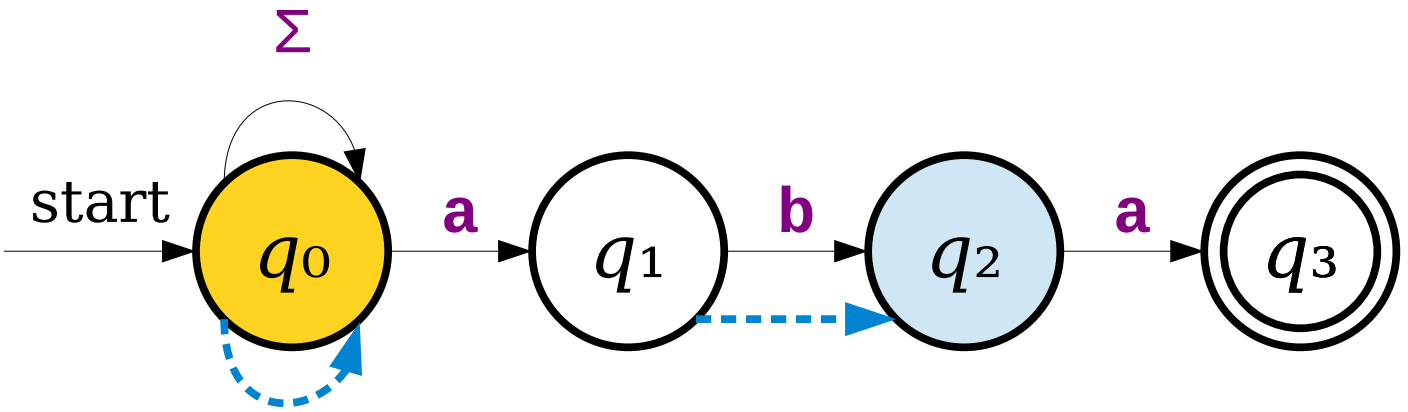


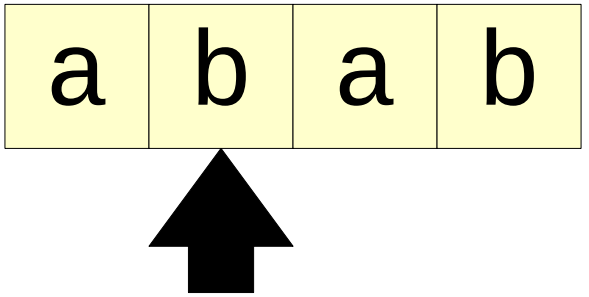
Massive Parallelism



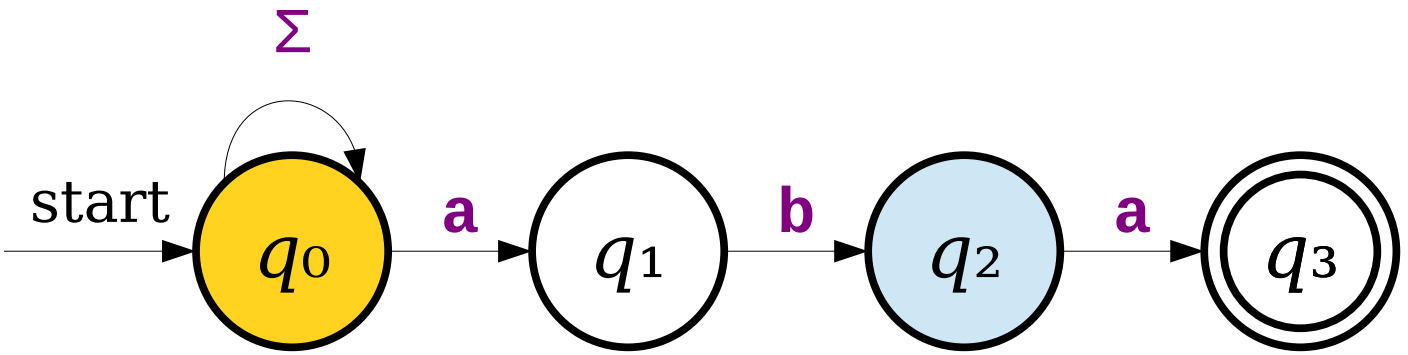


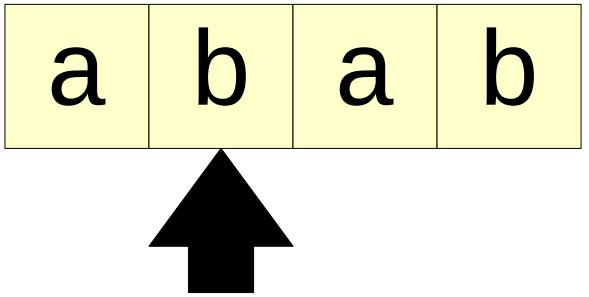
Massive Parallelism



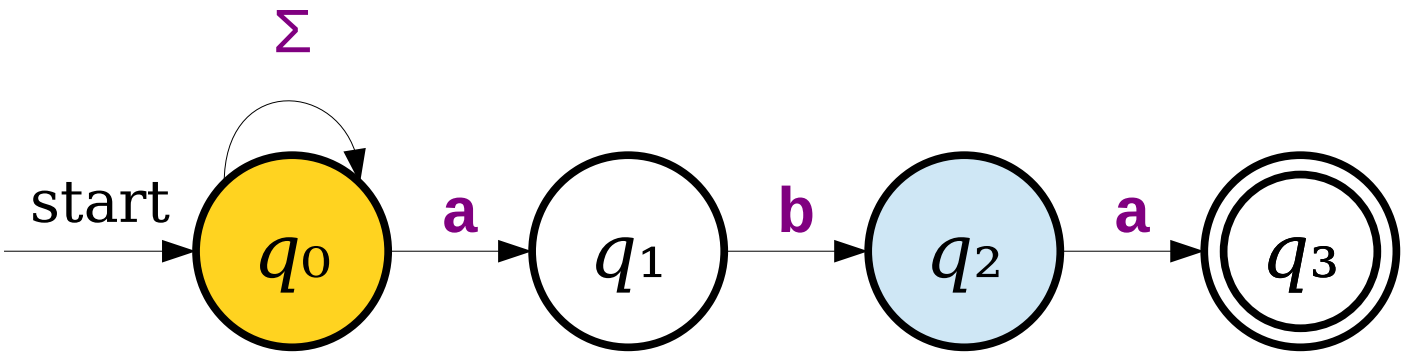


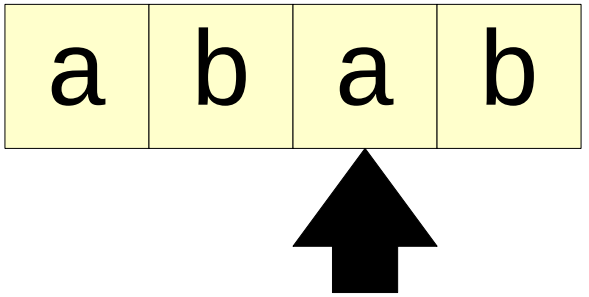
Massive Parallelism



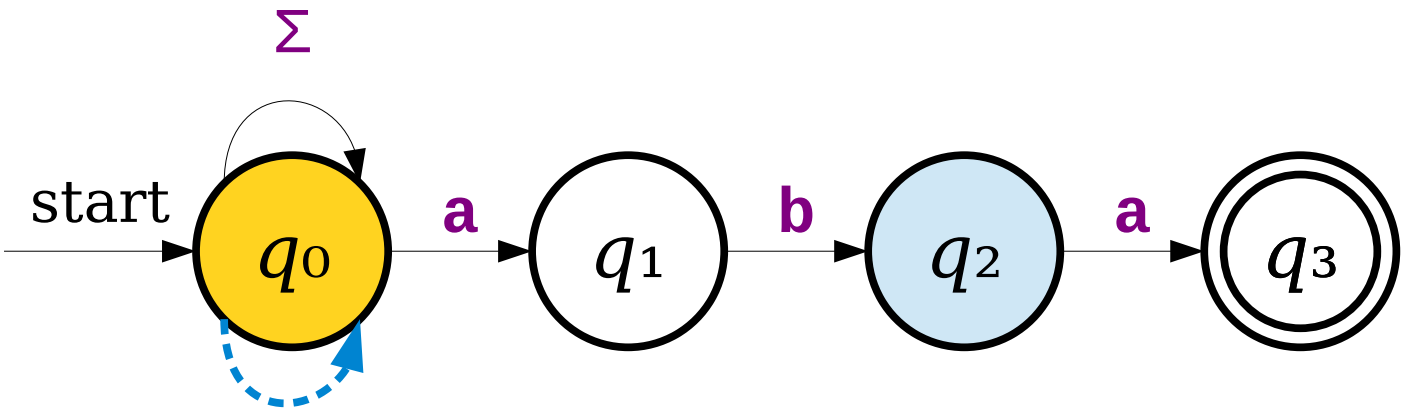


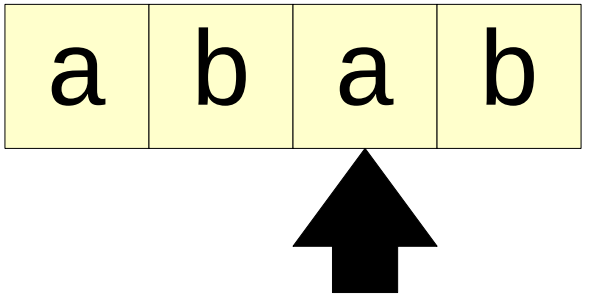
Massive Parallelism



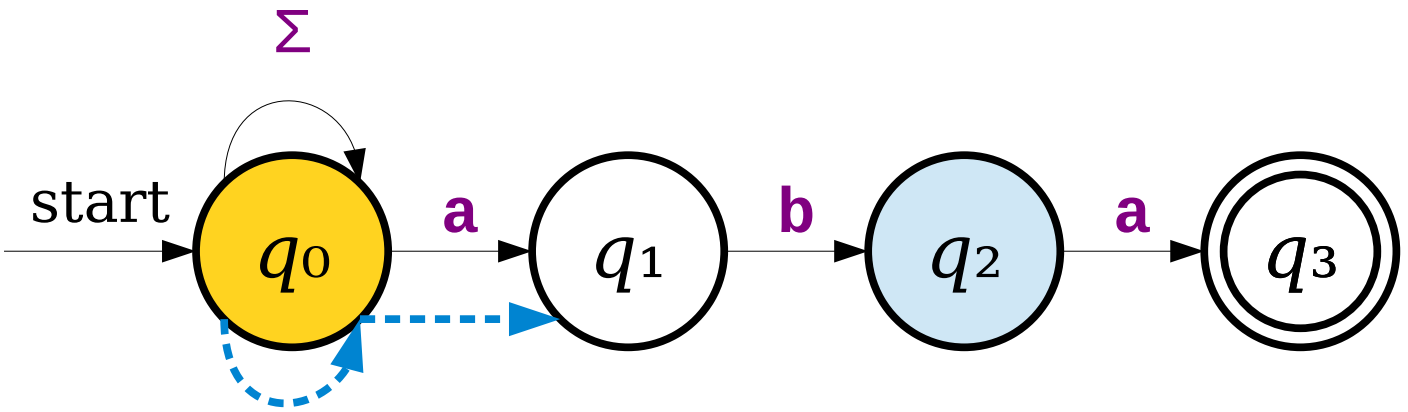


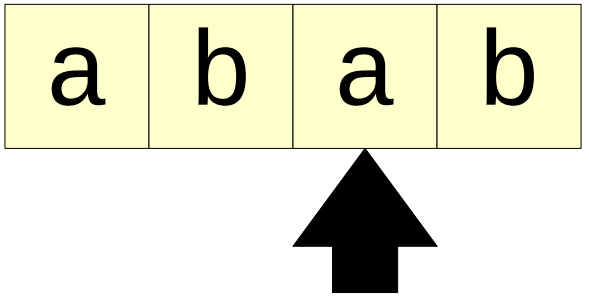
Massive Parallelism



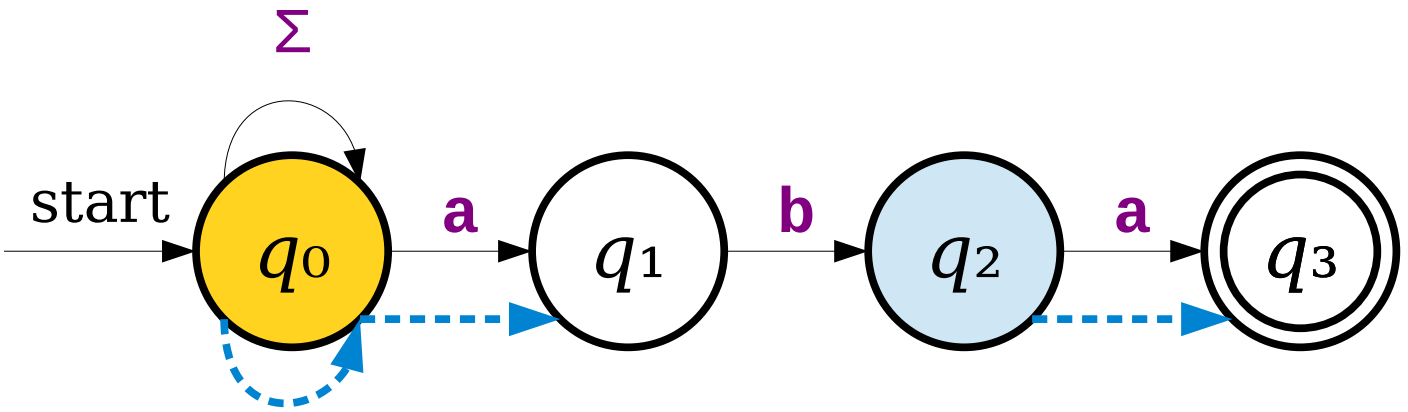


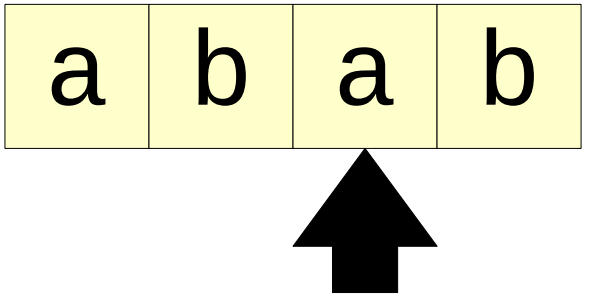
Massive Parallelism



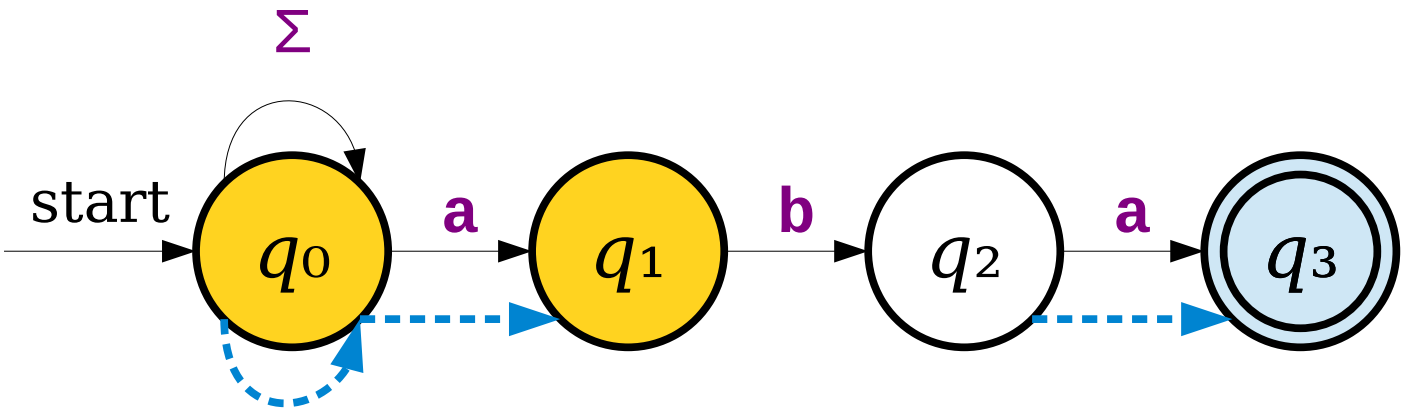


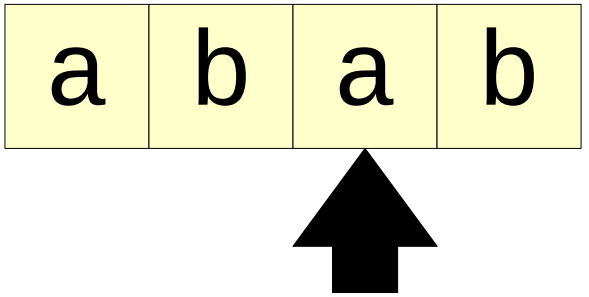
Massive Parallelism



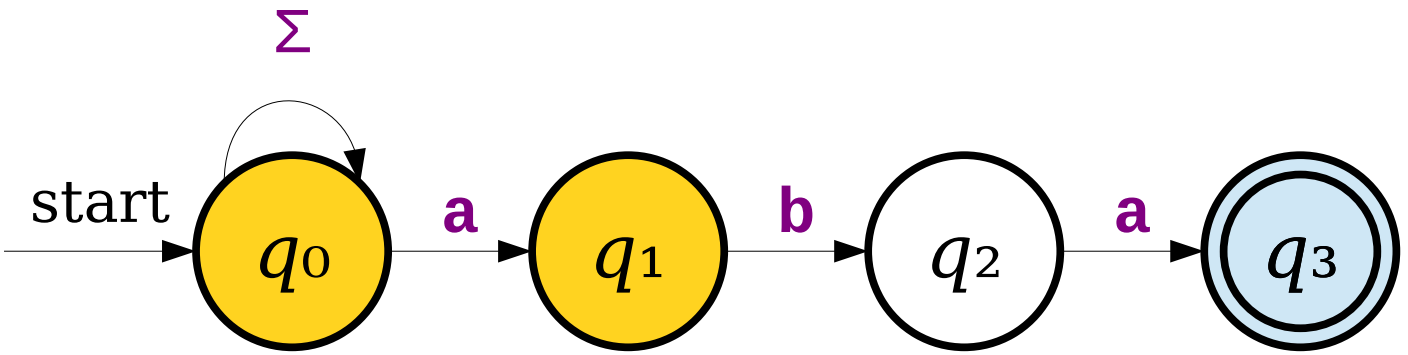


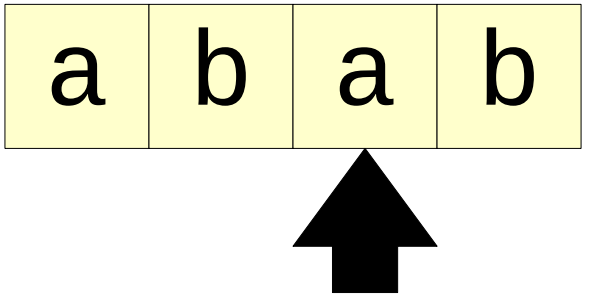
Massive Parallelism



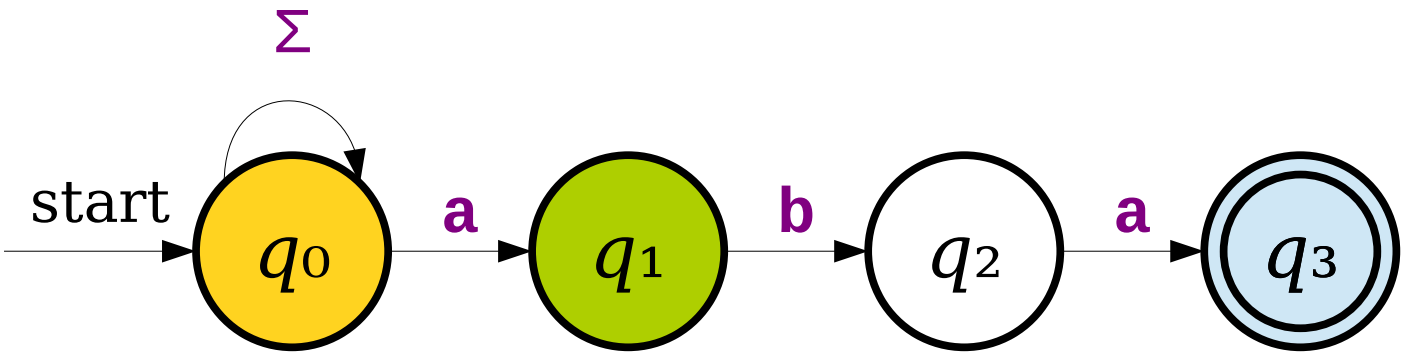


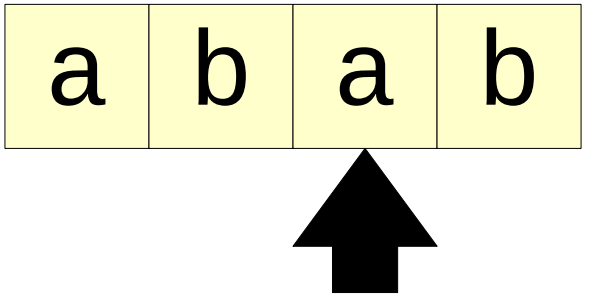
Massive Parallelism



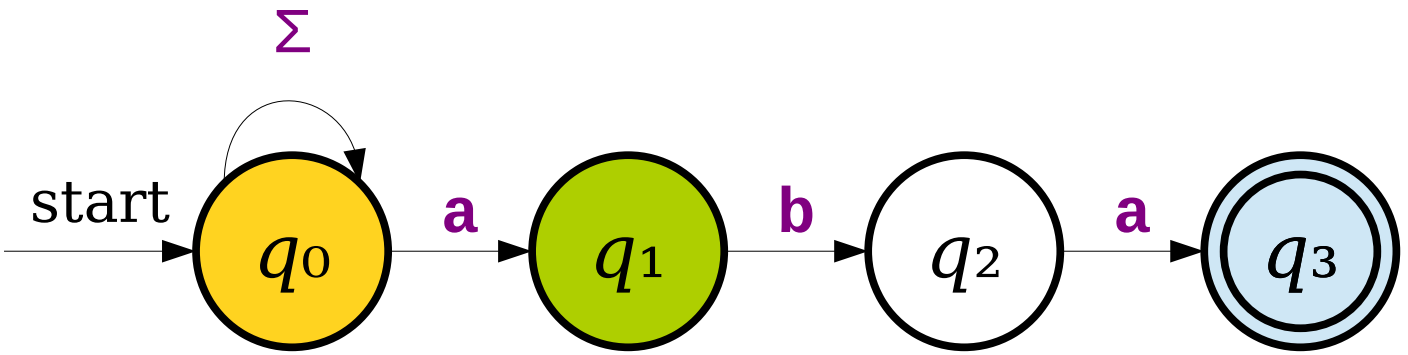


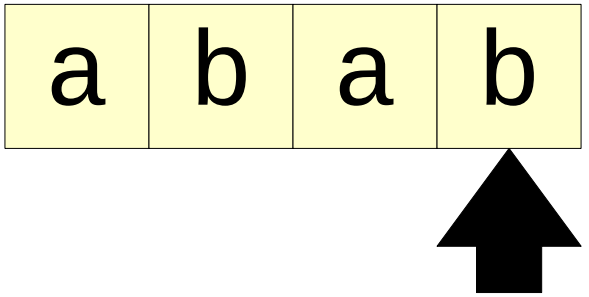
Massive Parallelism



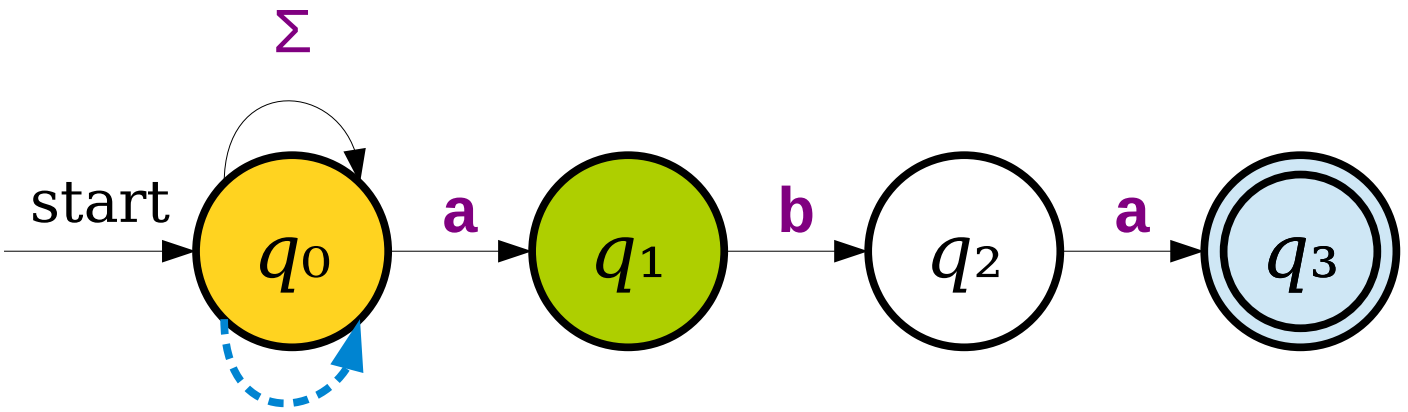


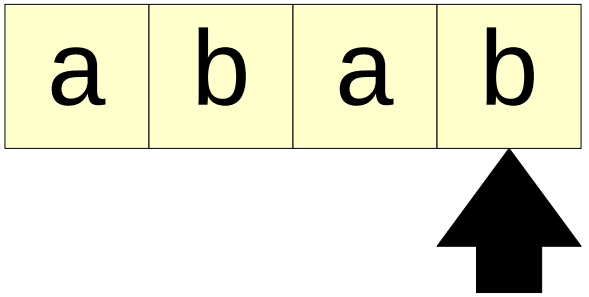
Massive Parallelism



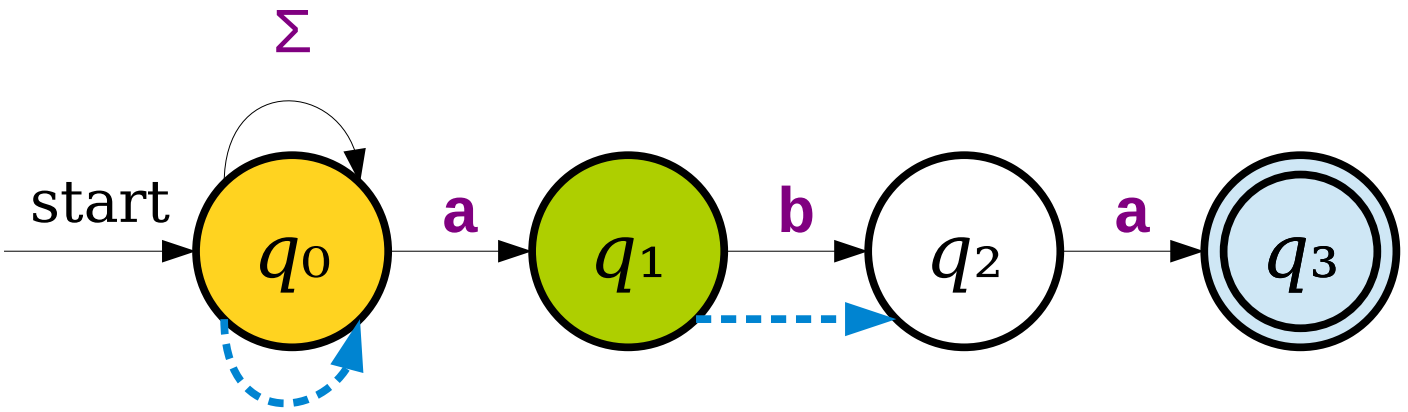


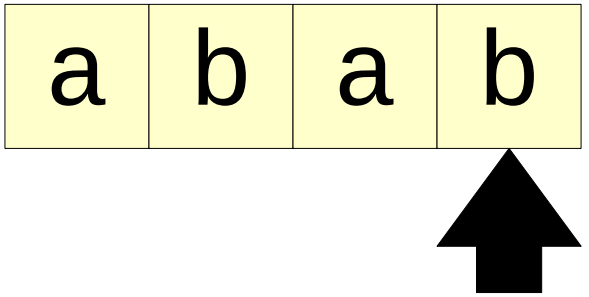
Massive Parallelism



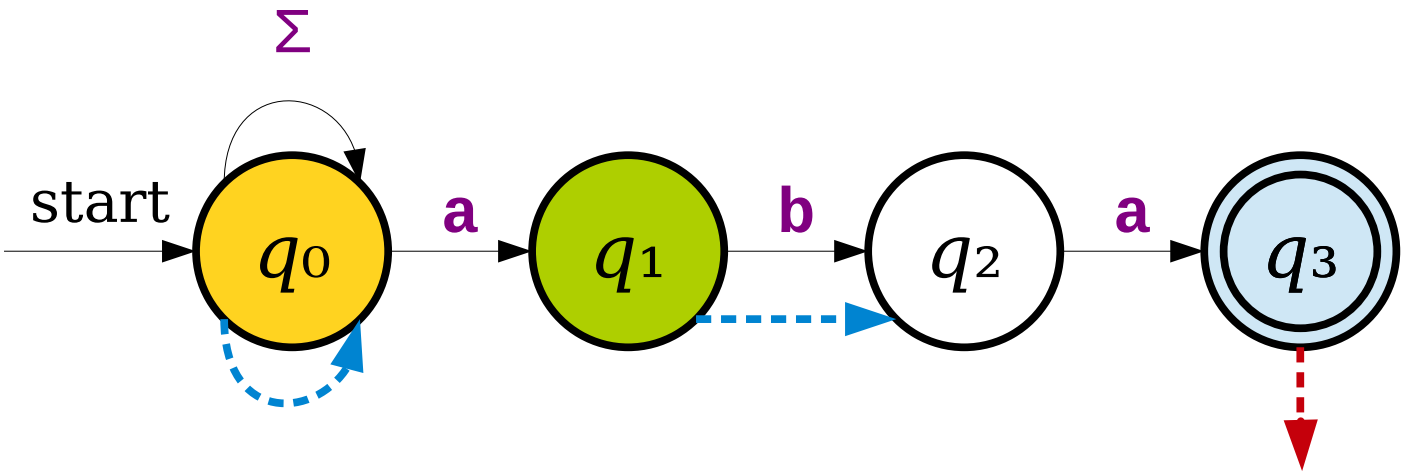


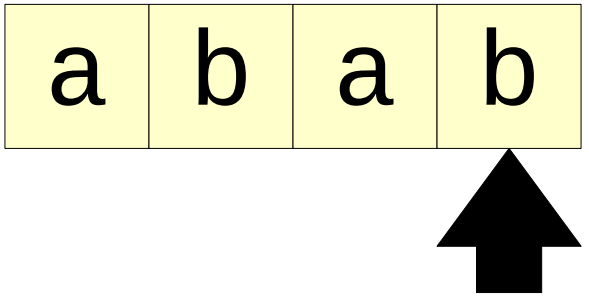
Massive Parallelism



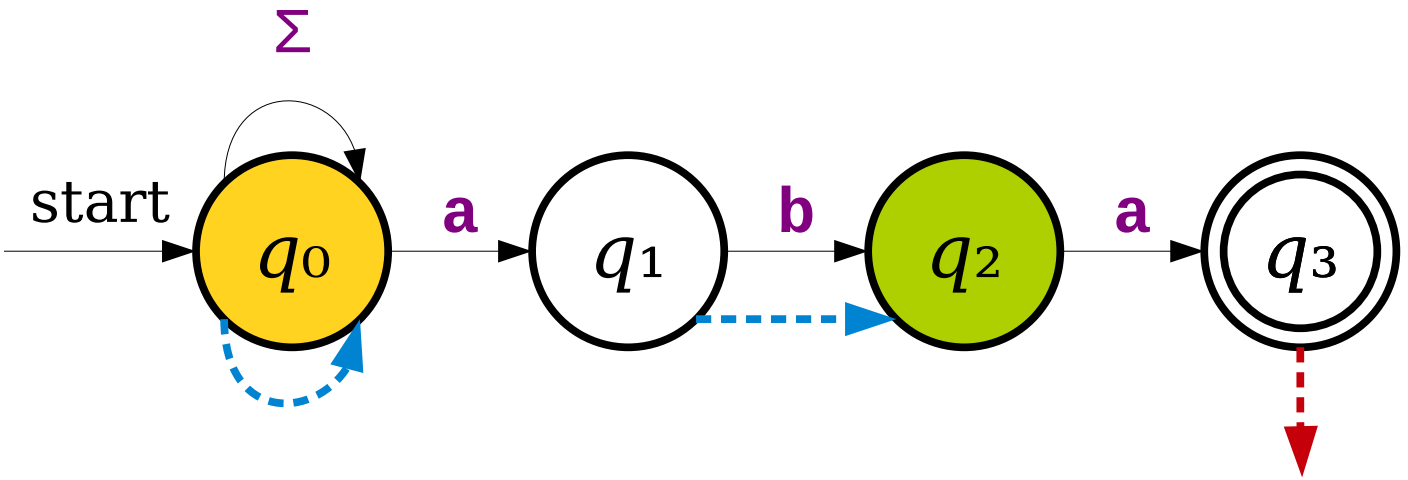


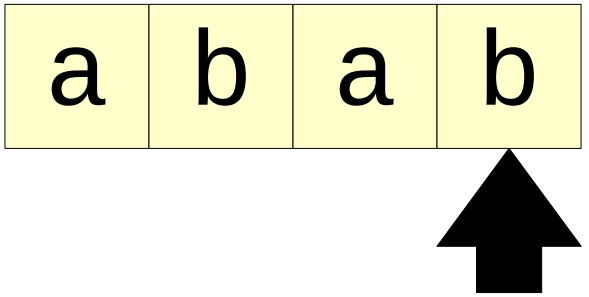
Massive Parallelism



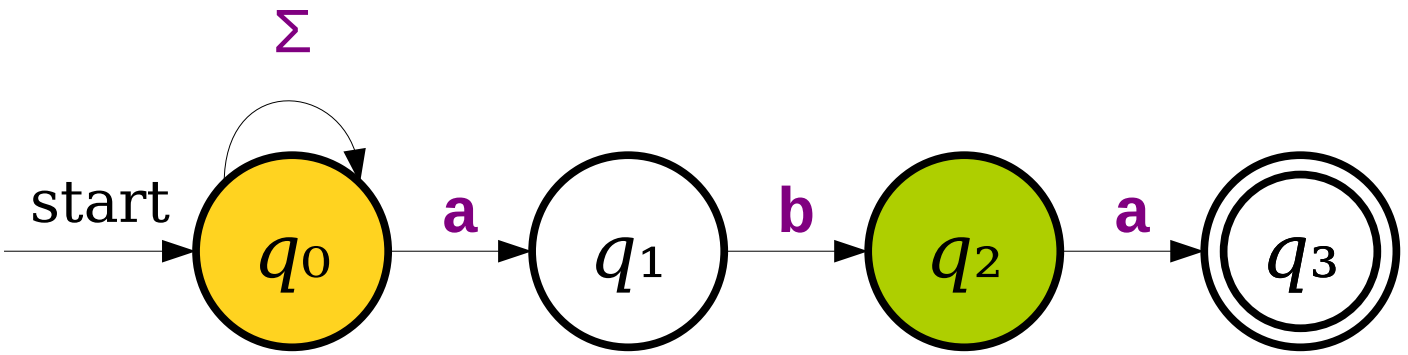


Massive Parallelism



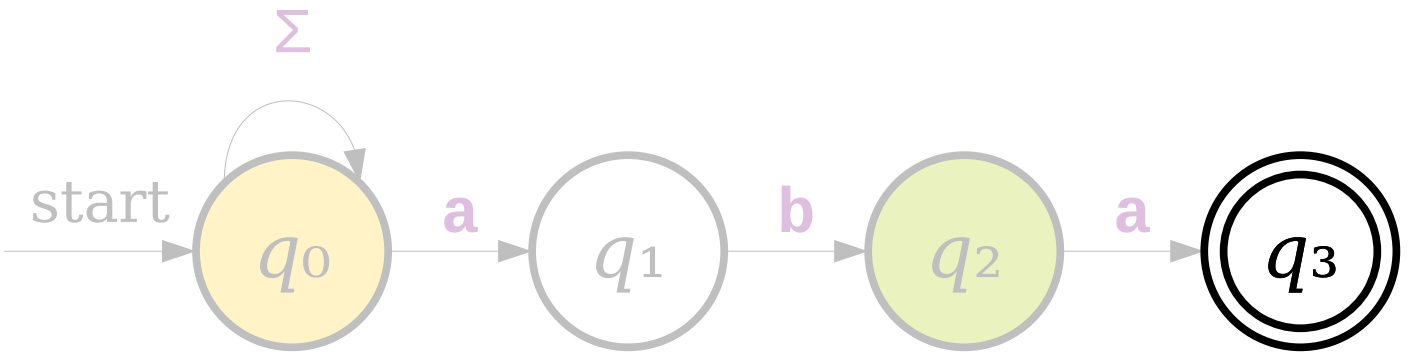


Massive Parallelism



a	b	a	b

Massive Parallelism



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

a	b	a	b

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).
  - Start off in the set of all states formed by taking the start state and including each state that can be reached by zero or more ε-transitions.
  - When you read a symbol **a** in a set of states *S*:
    - Form the set *S'* of states that can be reached by following a single **a** transition from some state in *S*.
    - Your new set of states is the set of states in *S'*, plus the states reachable from *S'* by following zero or more ε-transitions.

Designing NFAs

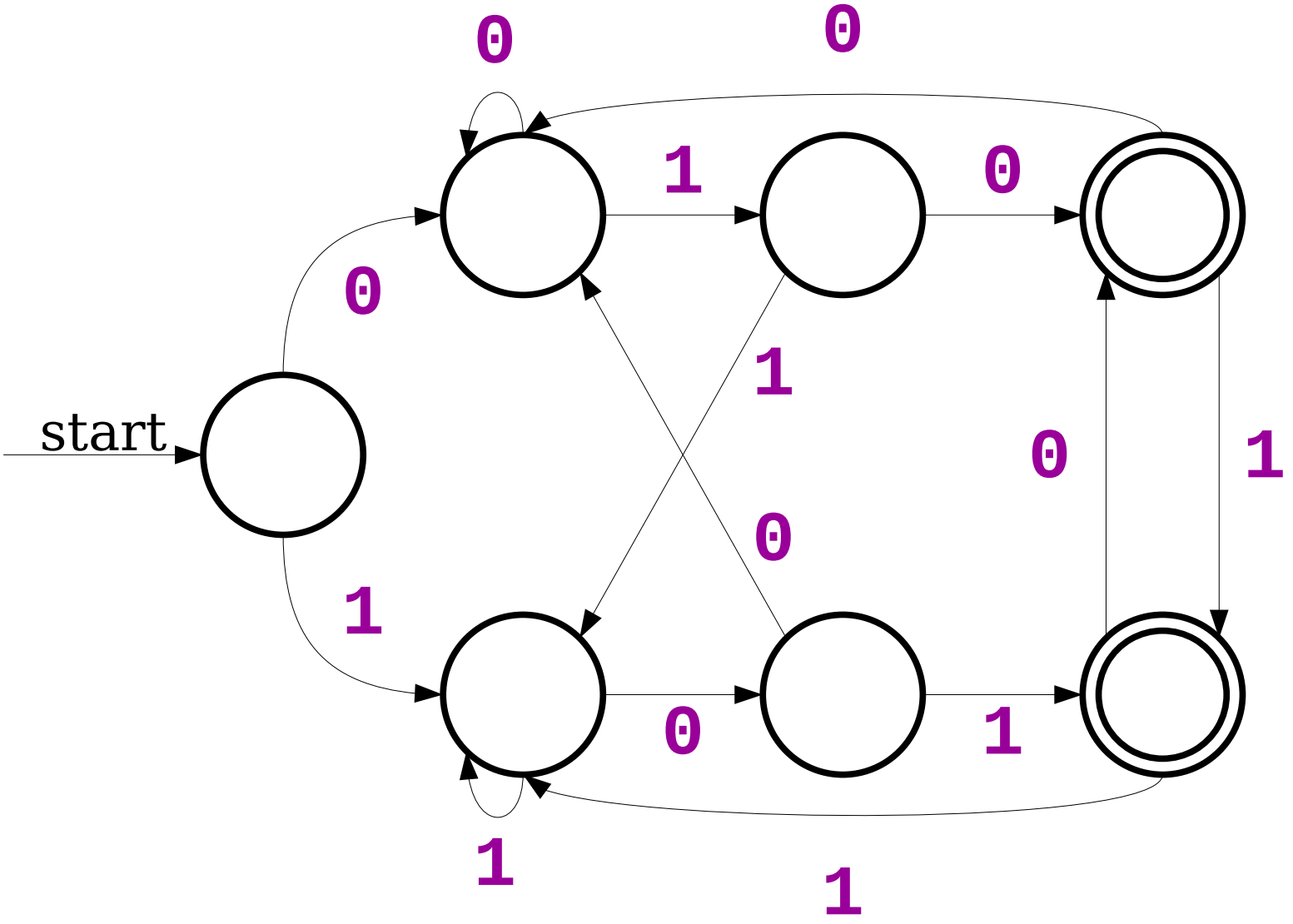
- ***Embrace the nondeterminism!***
- Good model: ***Guess-and-check***:
  - Is there some information that you'd really like to have? Have the machine *nondeterministically guess* that information.
  - Then, have the machine *deterministically check* that the choice was correct.
- The *guess* phase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

Guess-and-Check

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

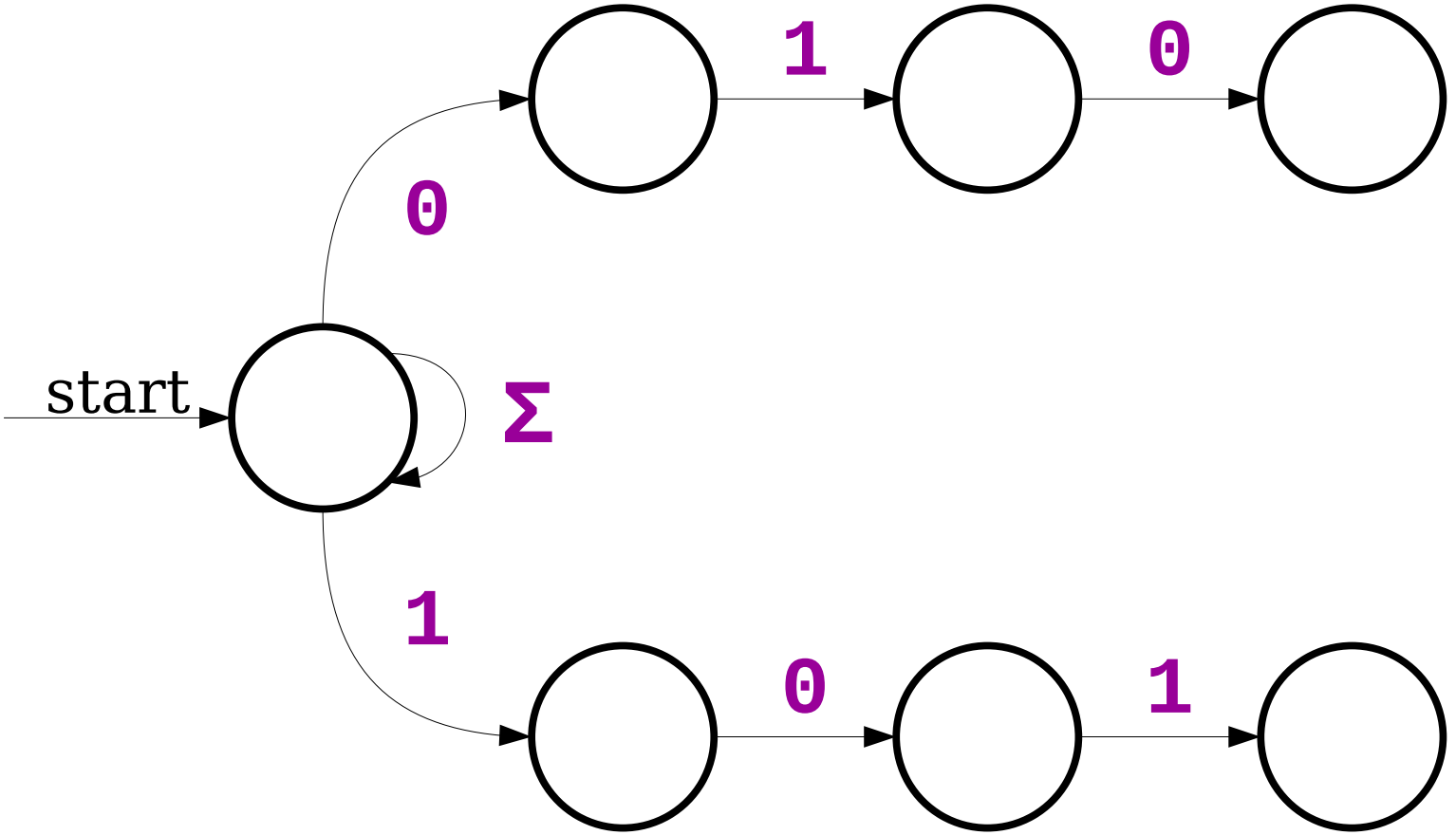
Guess-and-Check

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



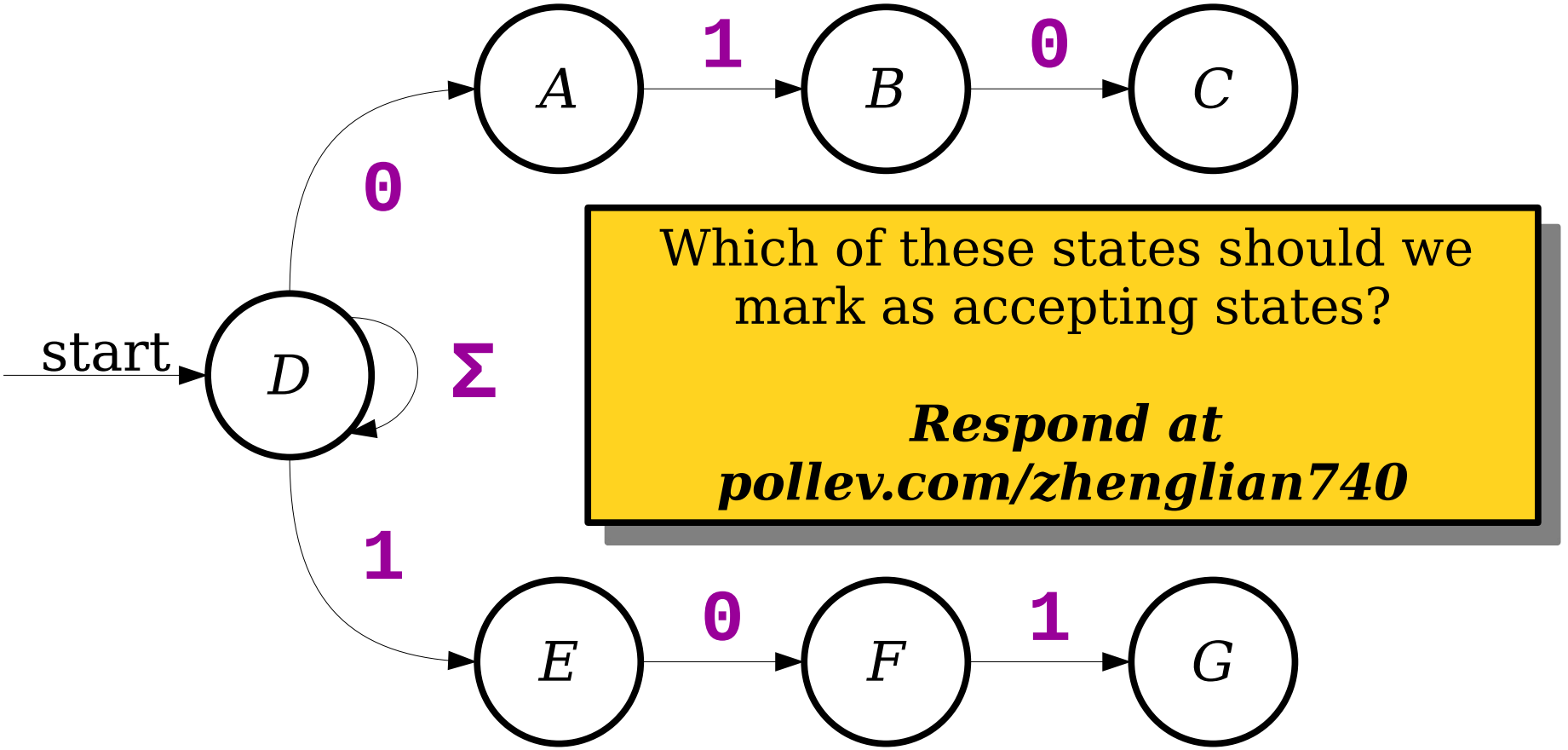
Guess-and-Check

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



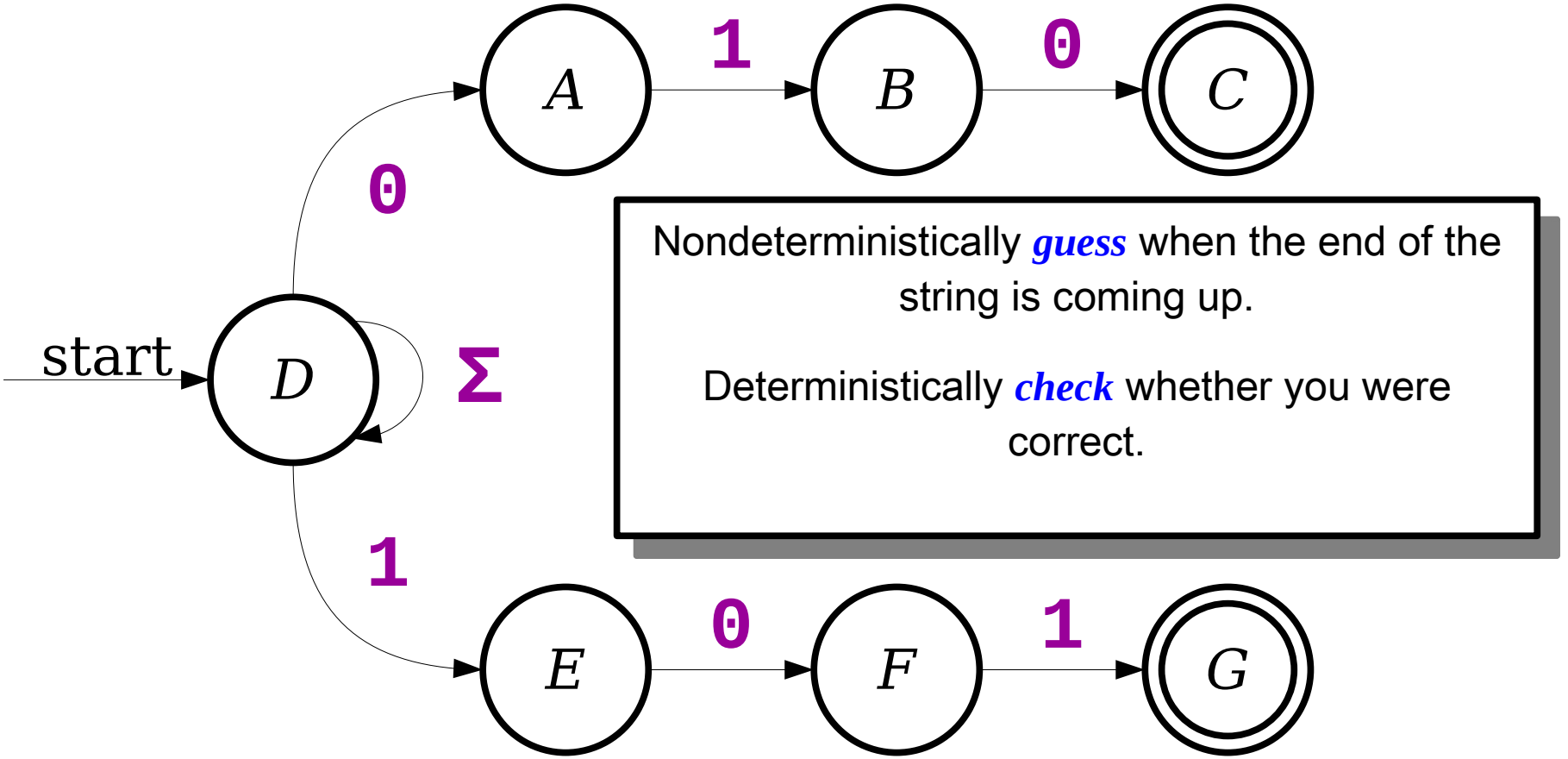
Guess-and-Check

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



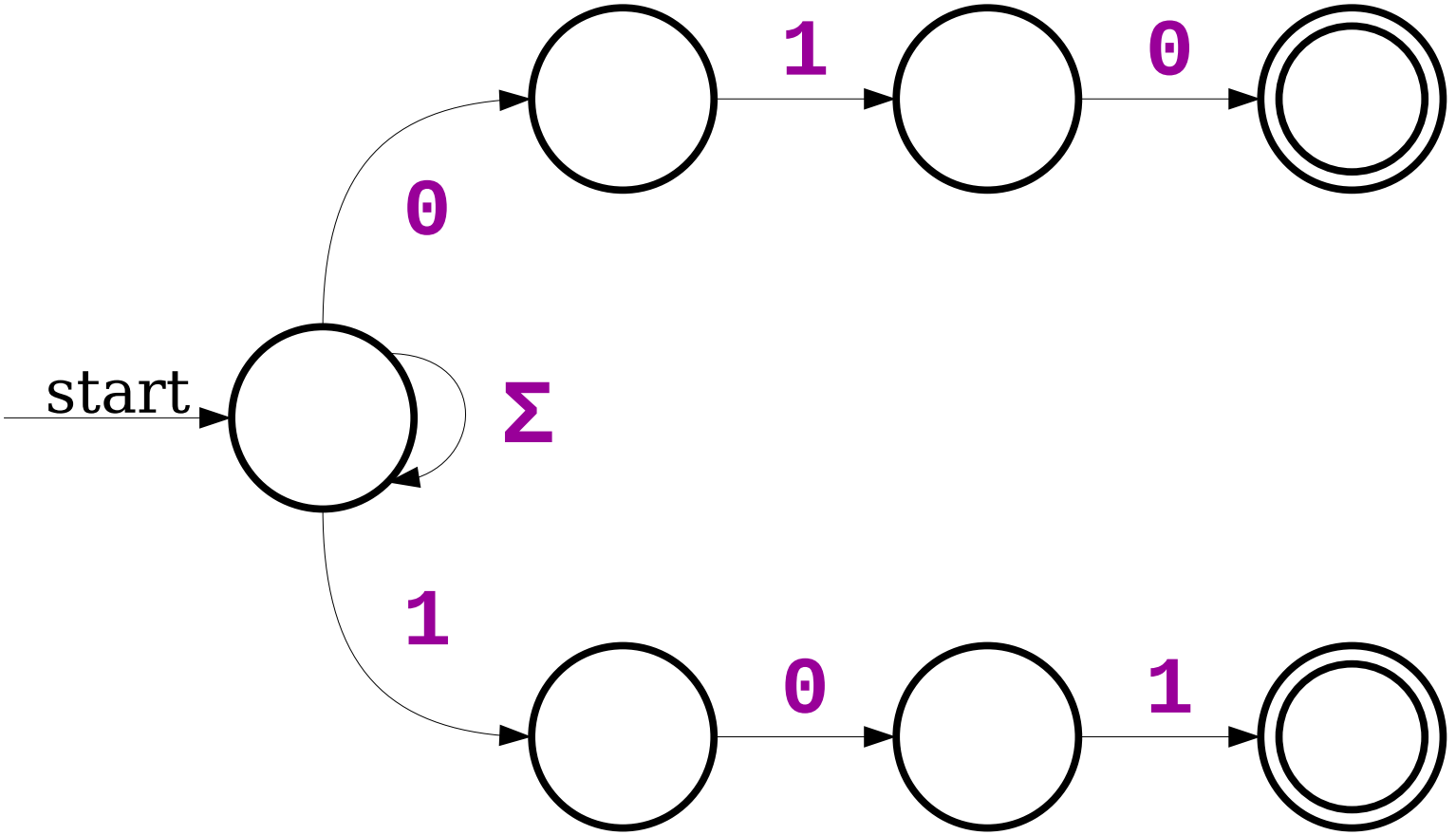
Guess-and-Check

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



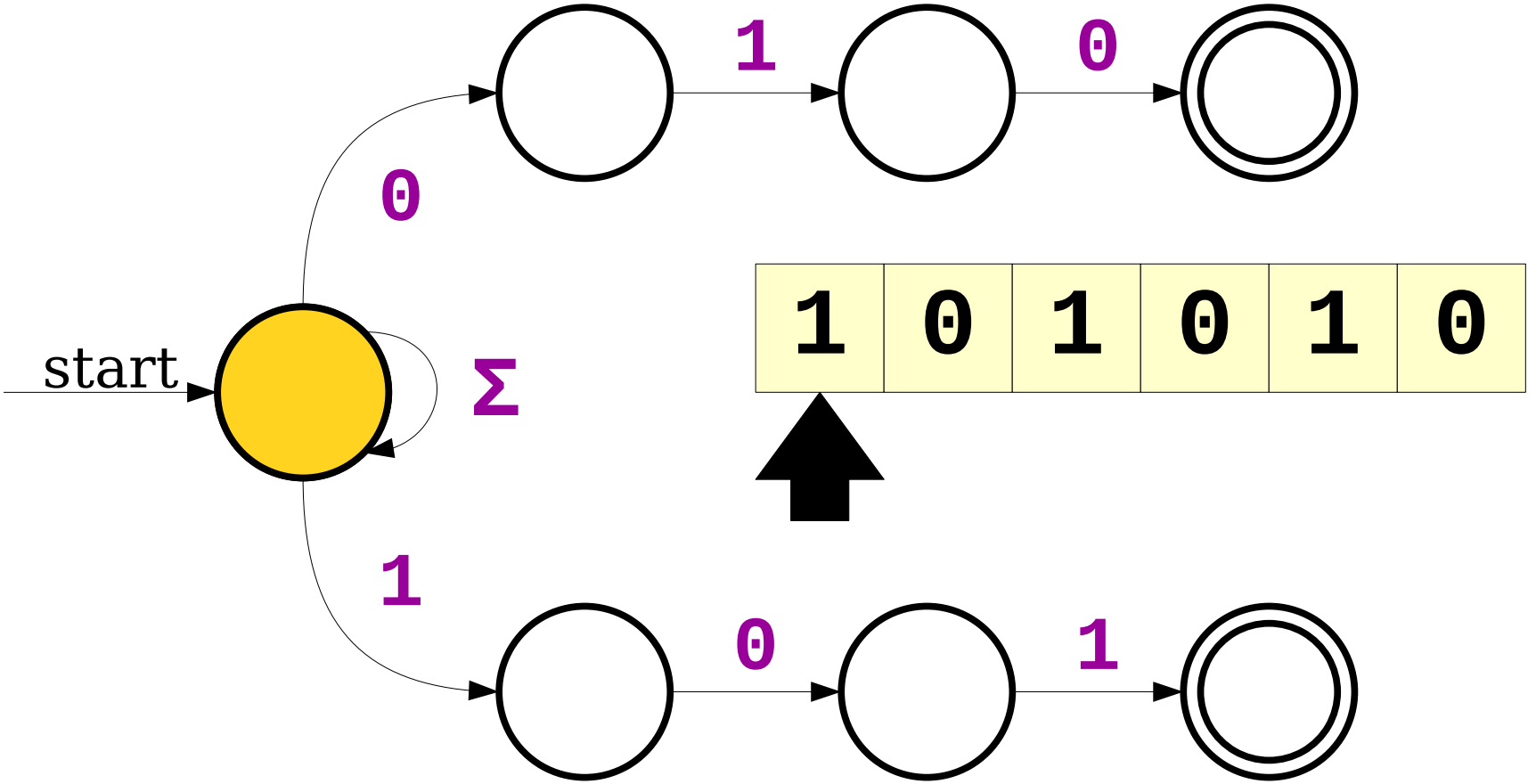
Guess-and-Check

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



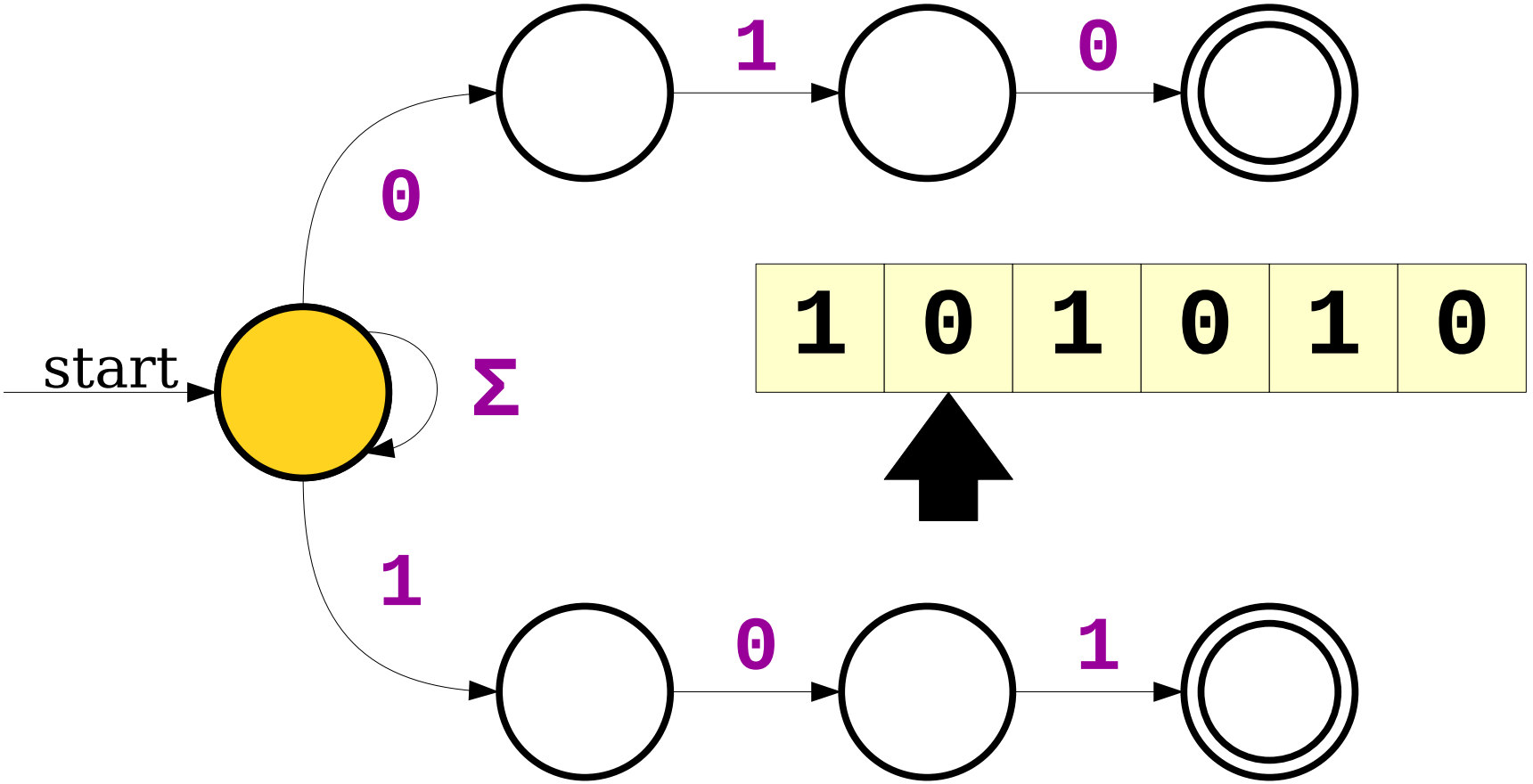
Guess-and-Check

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



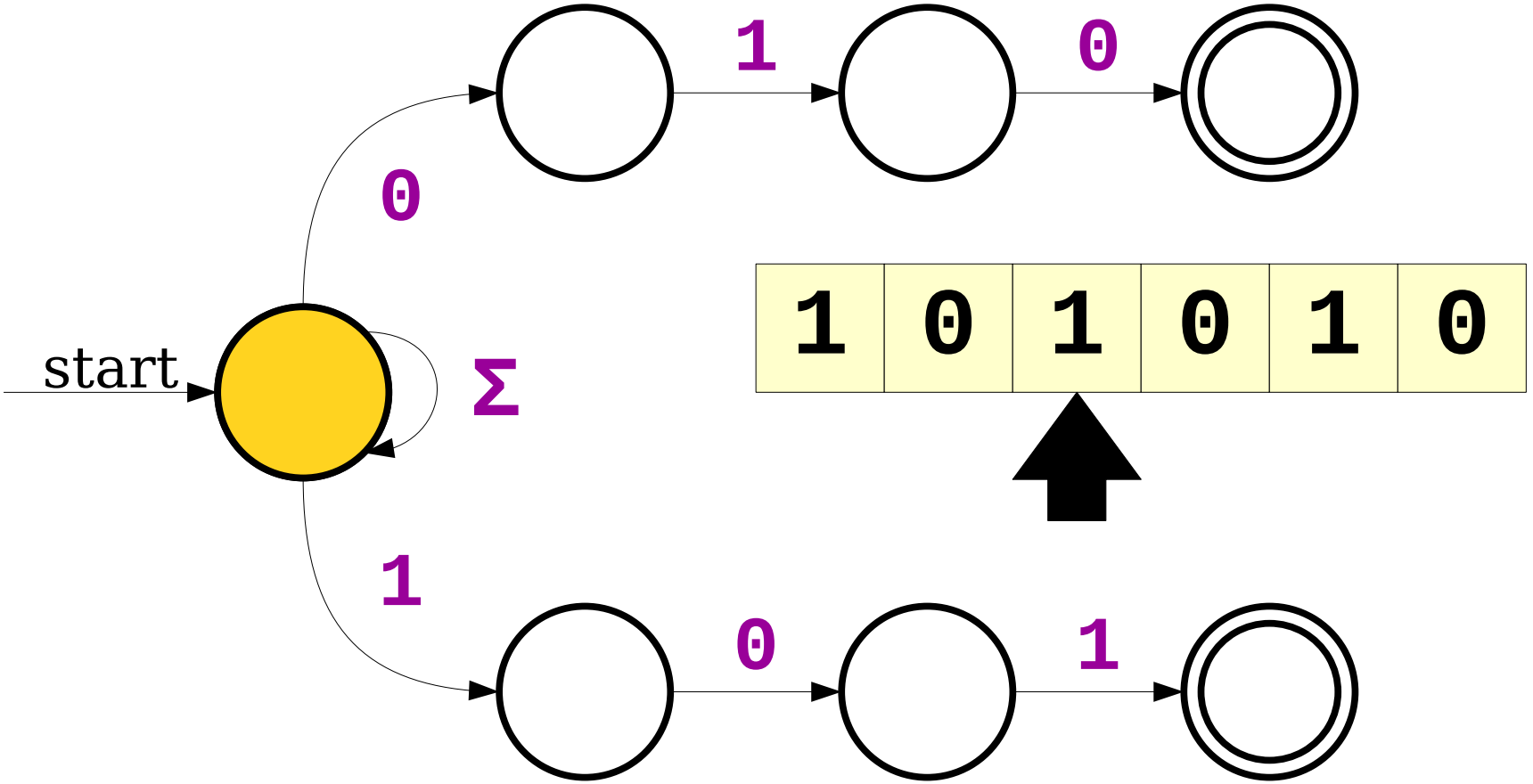
Guess-and-Check

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



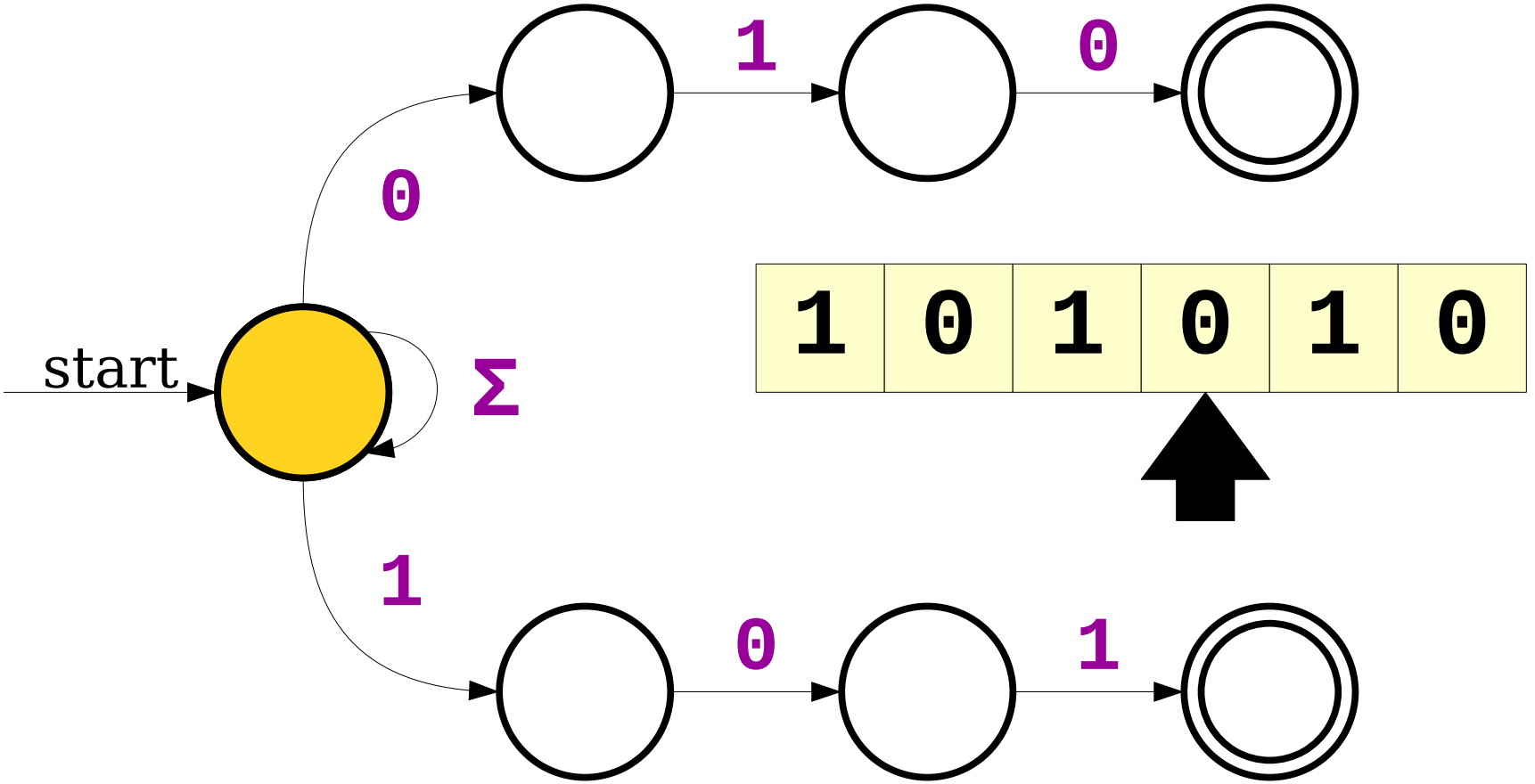
Guess-and-Check

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



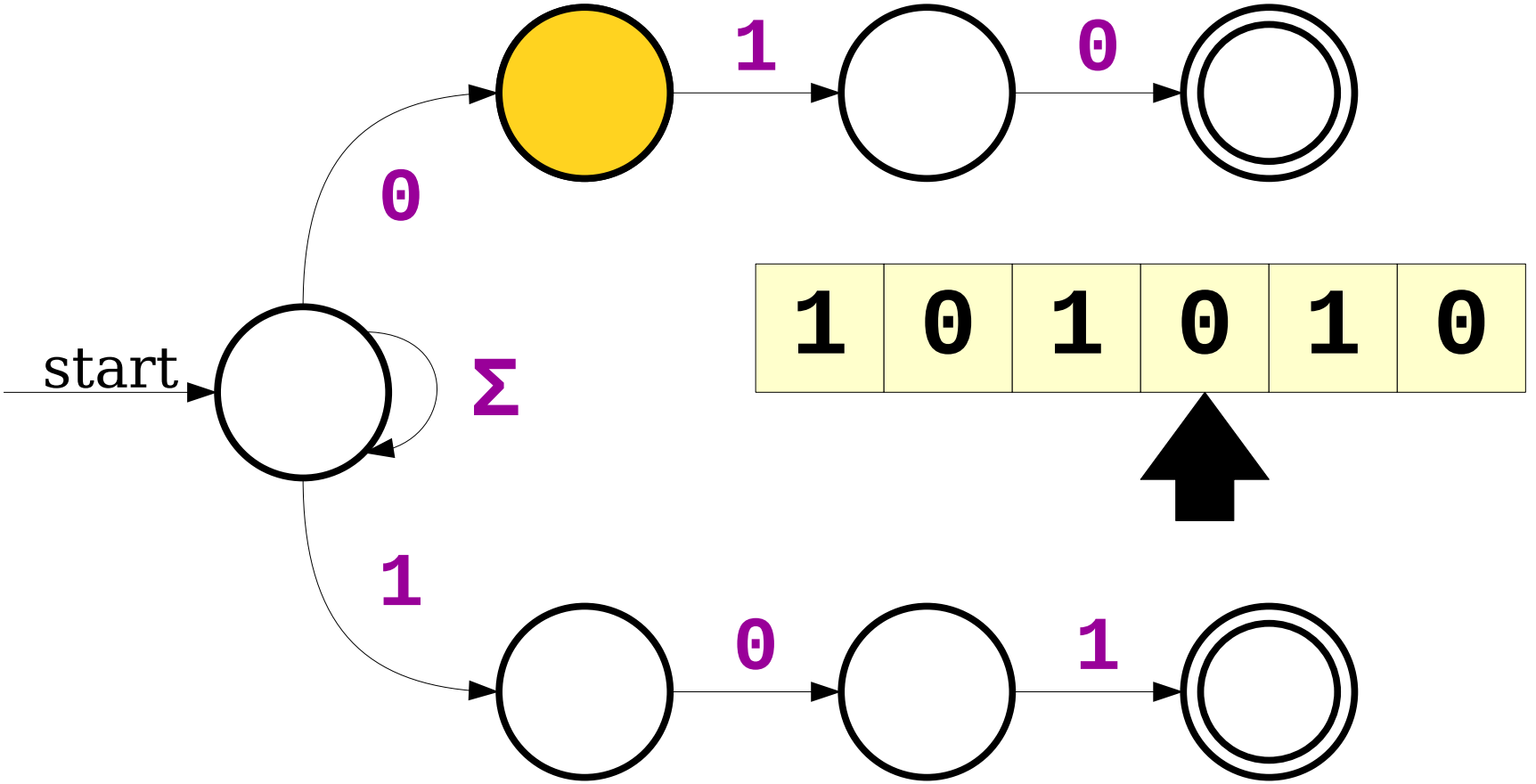
Guess-and-Check

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



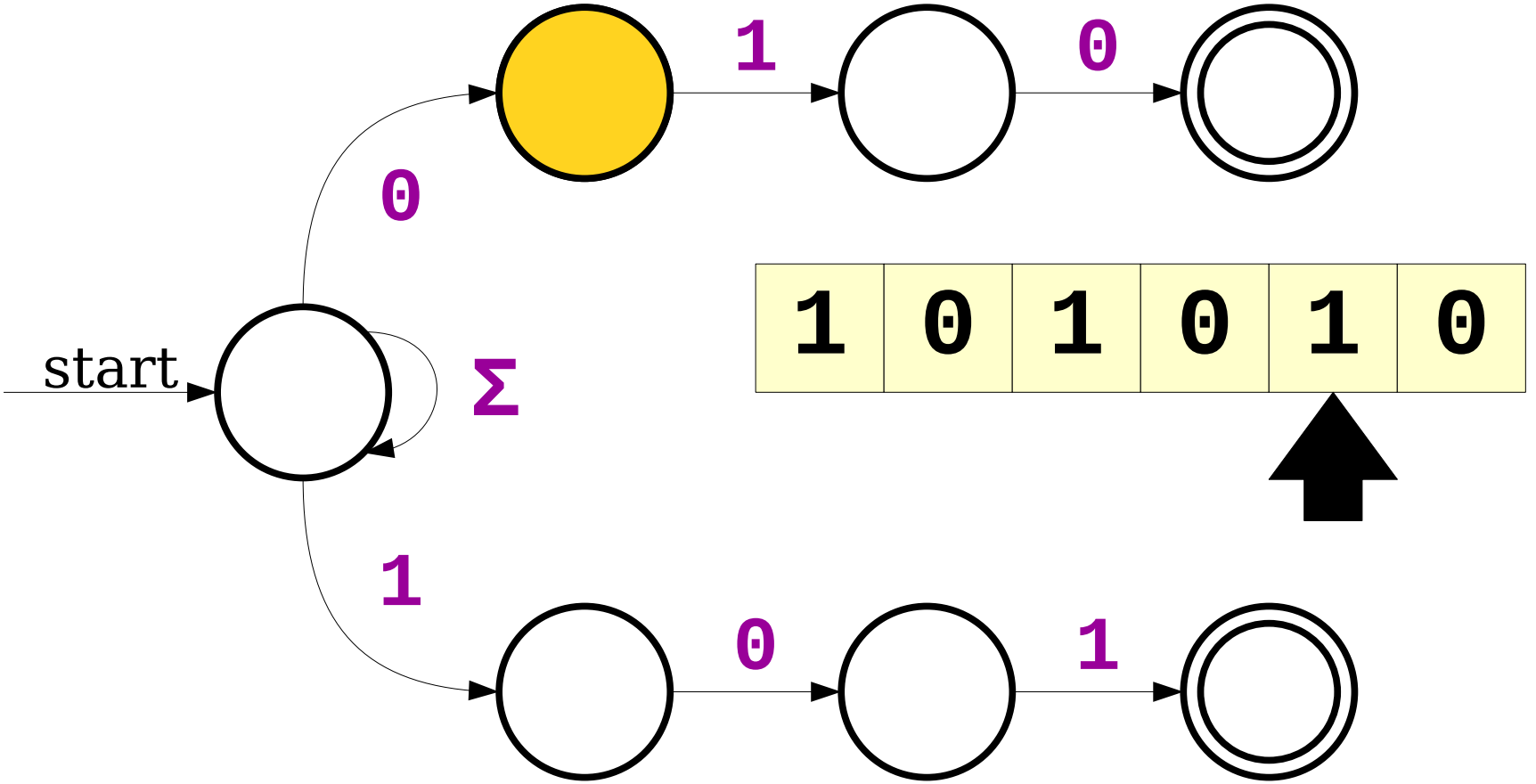
Guess-and-Check

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



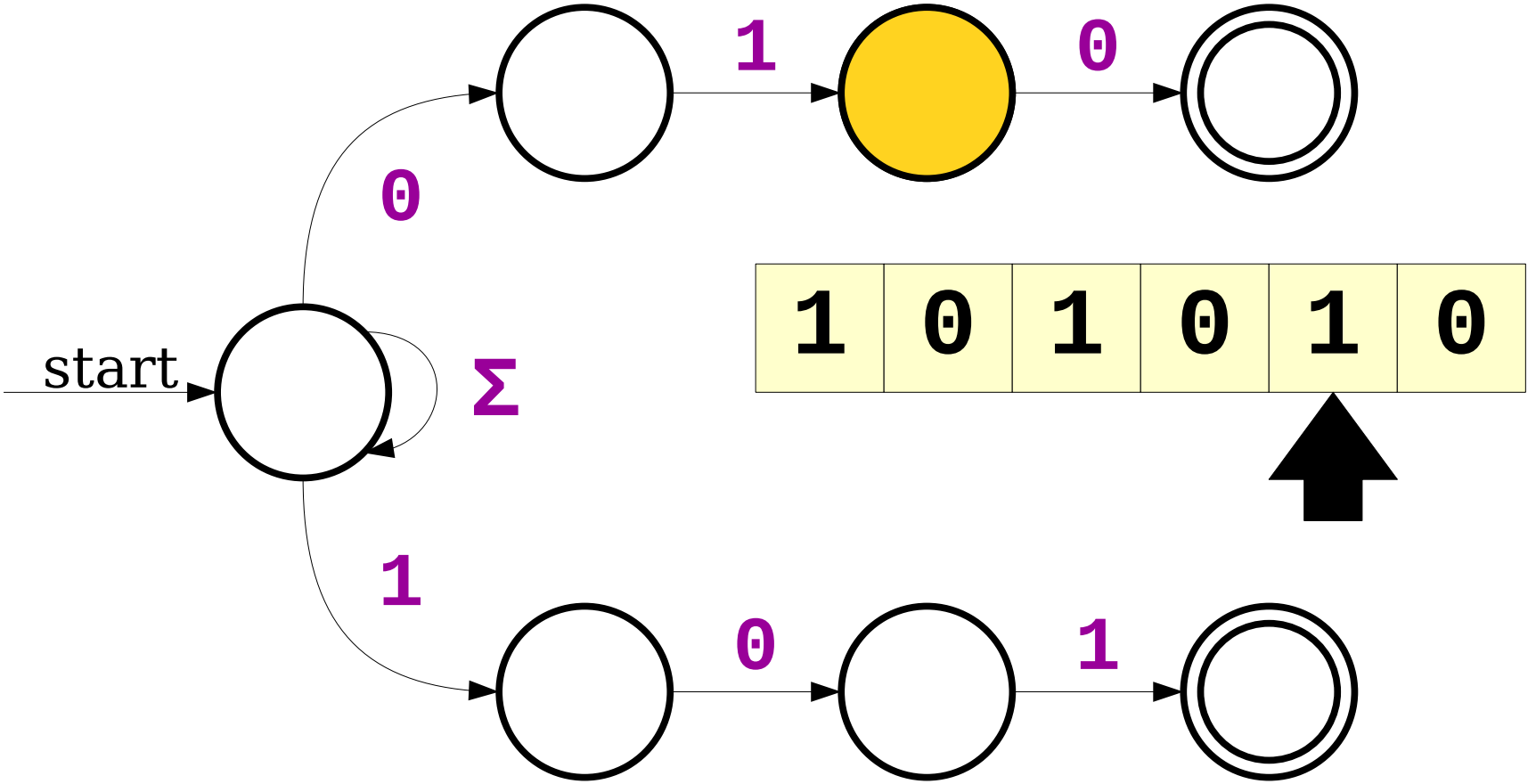
Guess-and-Check

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



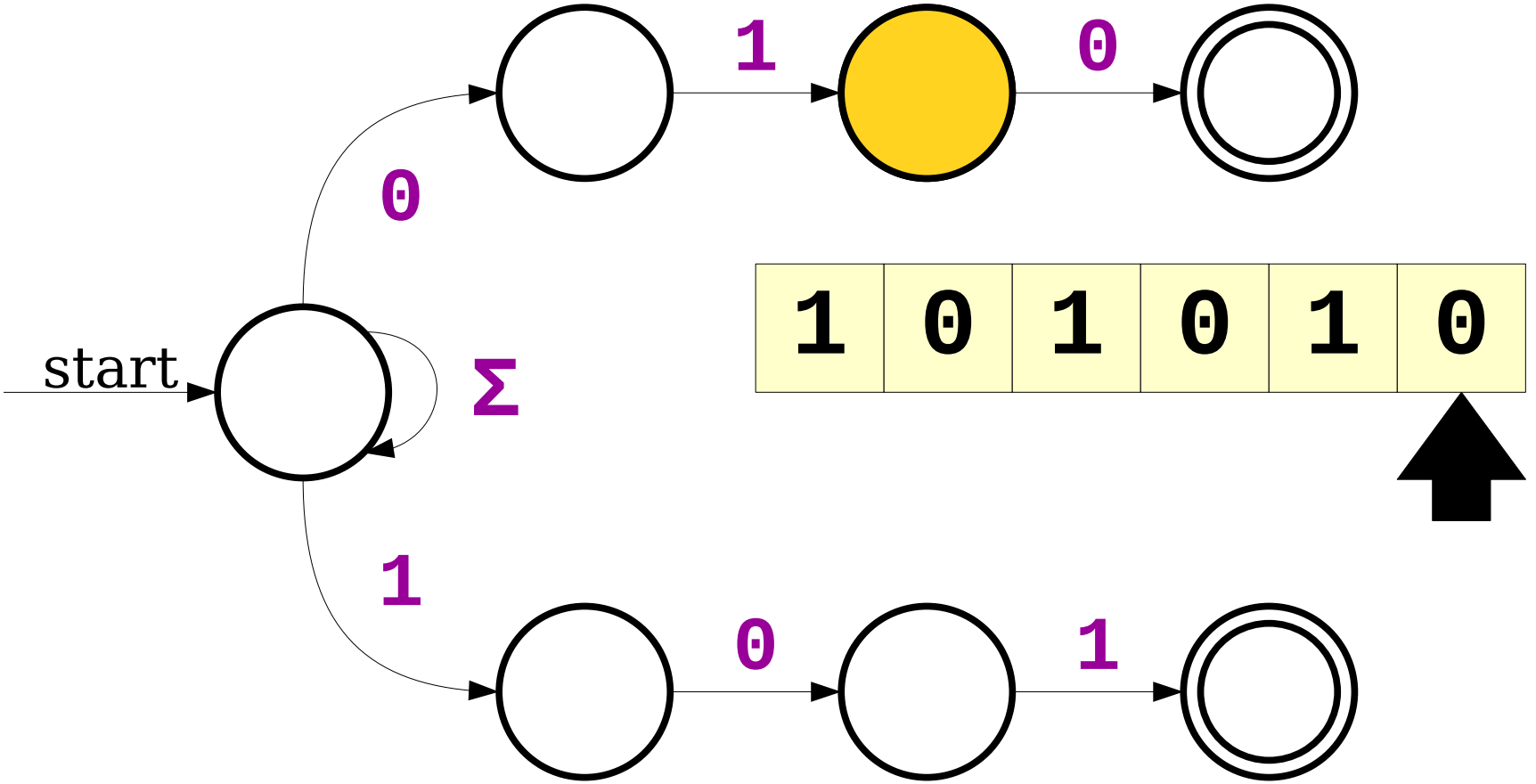
Guess-and-Check

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



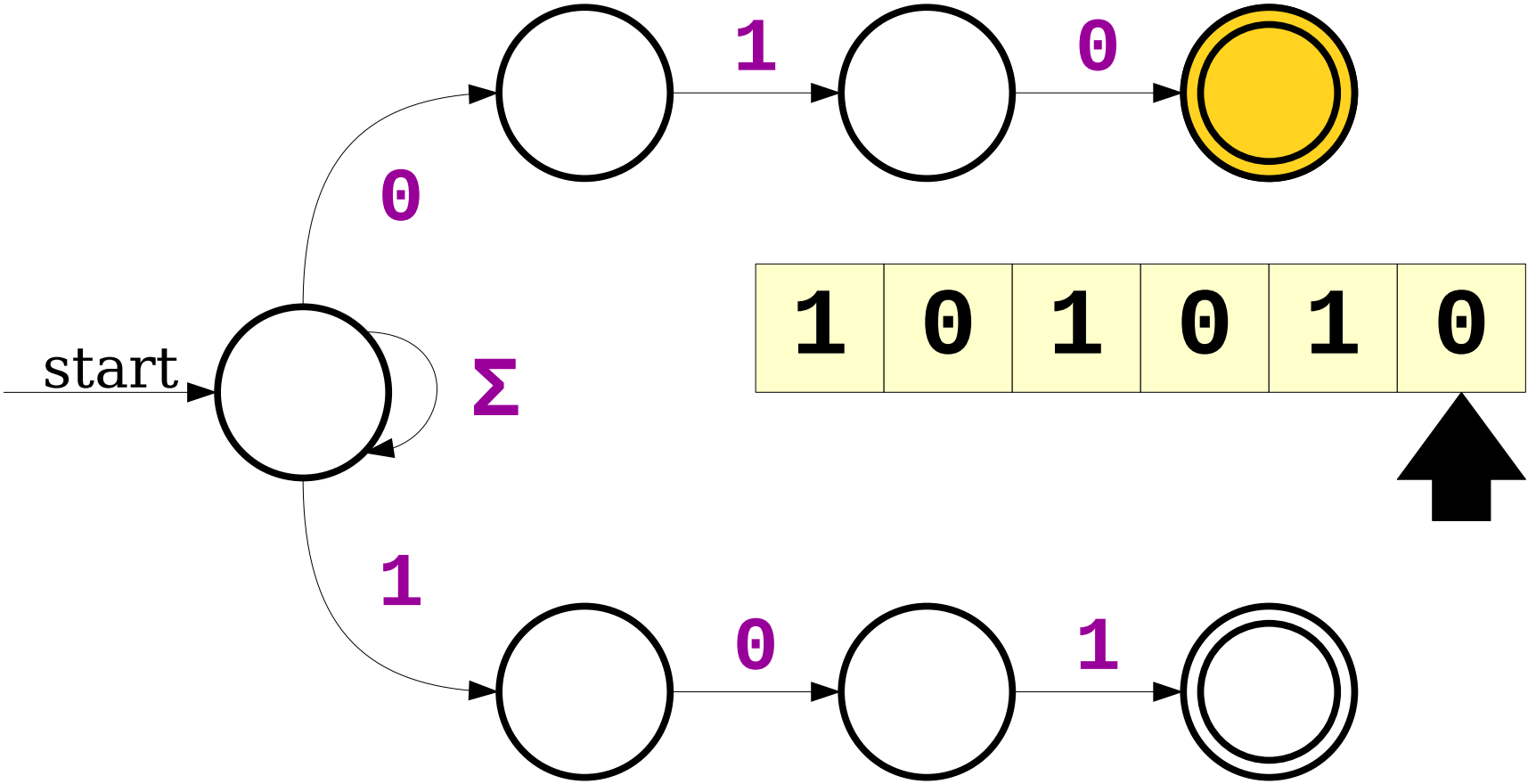
Guess-and-Check

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



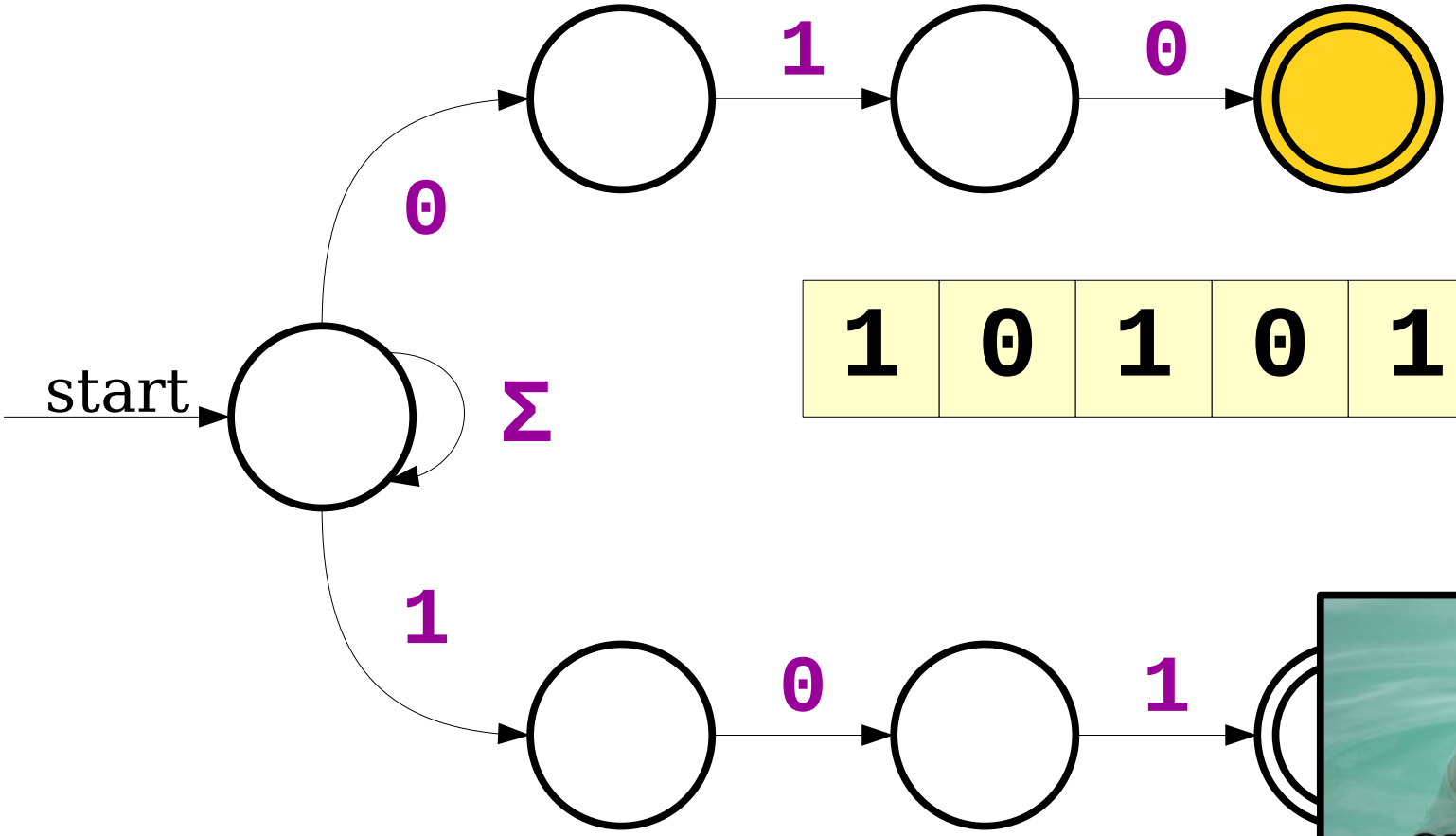
Guess-and-Check

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



Guess-and-Check

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

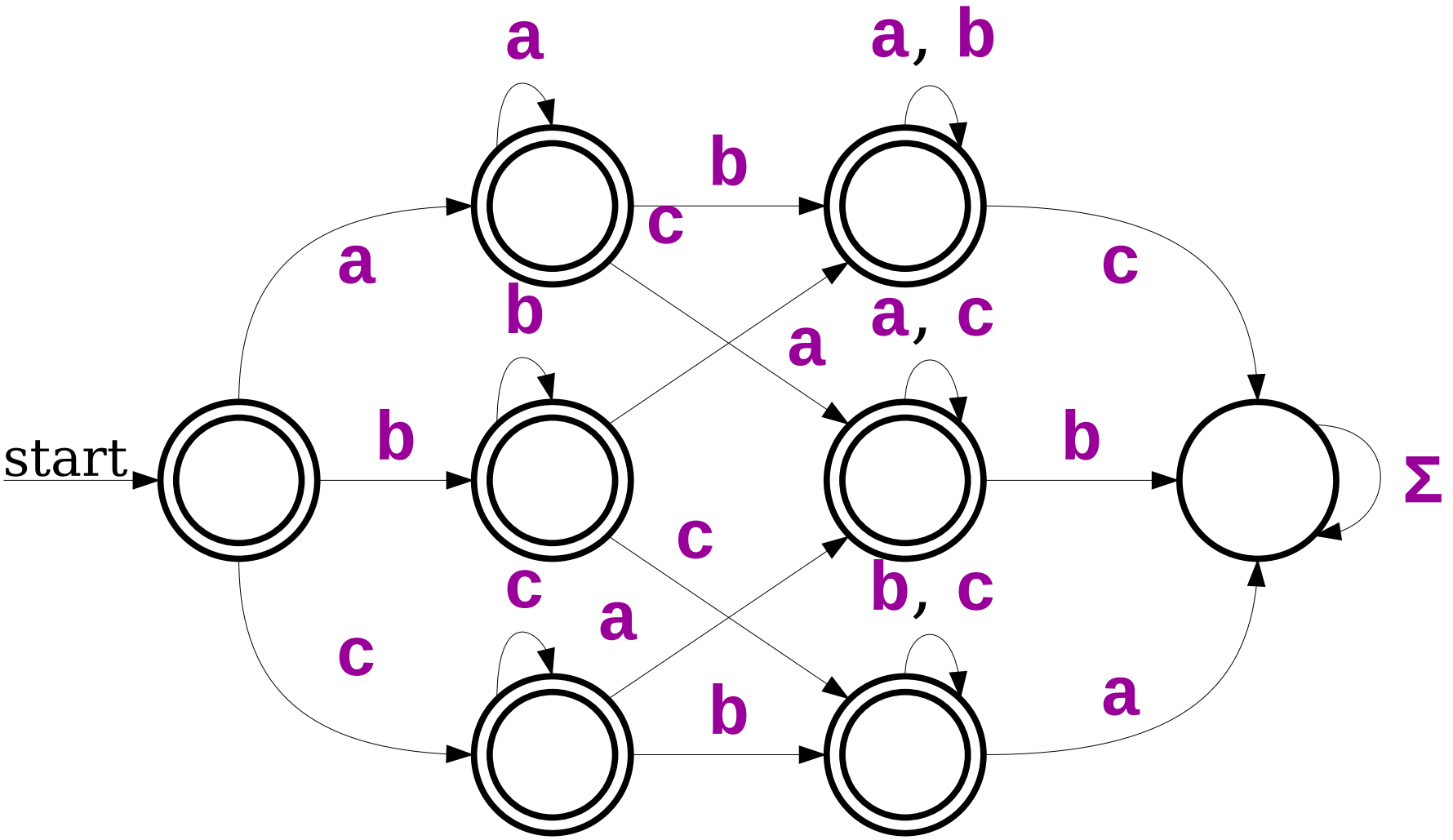


Guess-and-Check

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

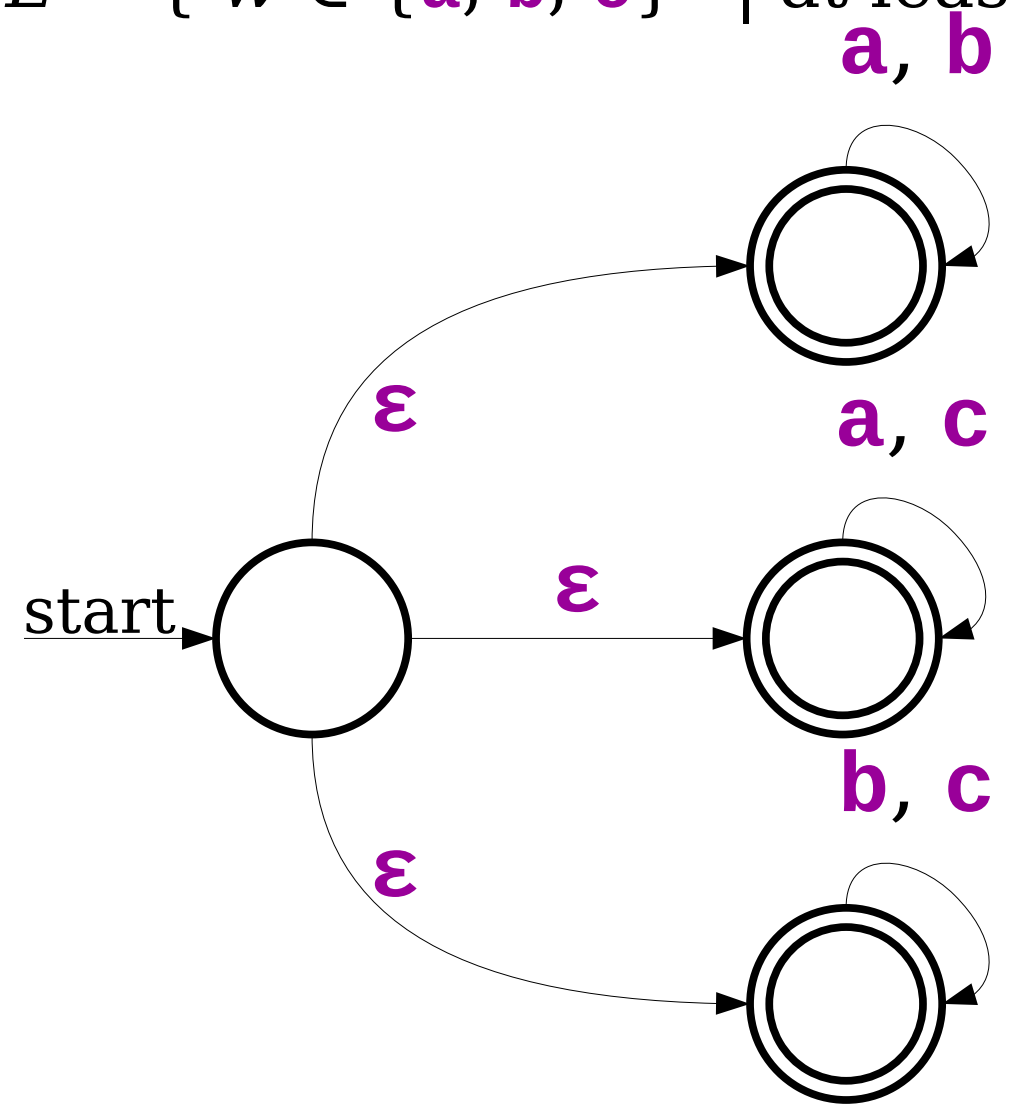
Guess-and-Check

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



Guess-and-Check

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

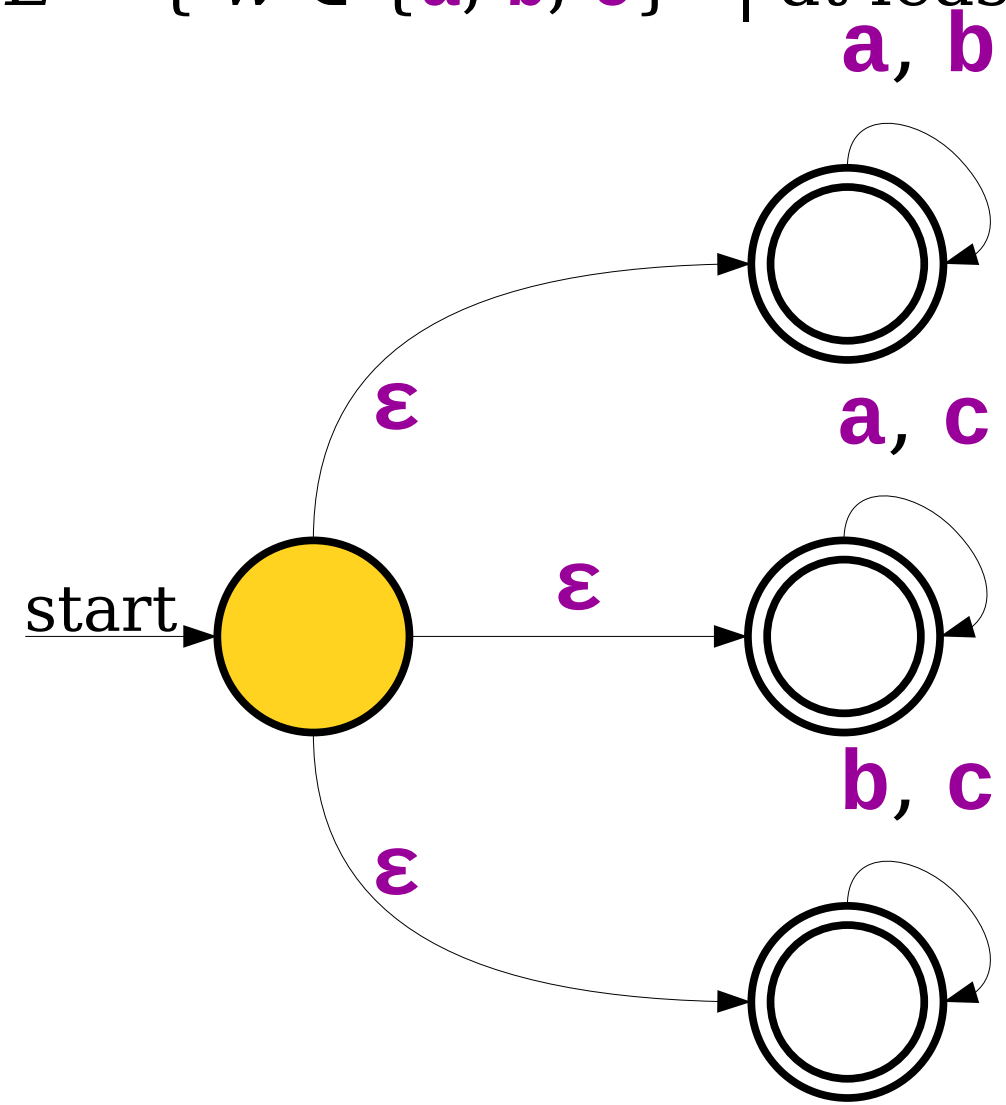


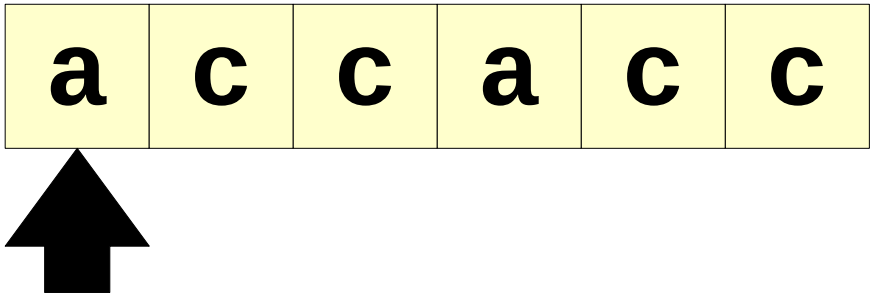
Nondeterministically ***guess*** which character is missing.

Deterministically ***check*** whether that character is indeed missing.

Guess-and-Check

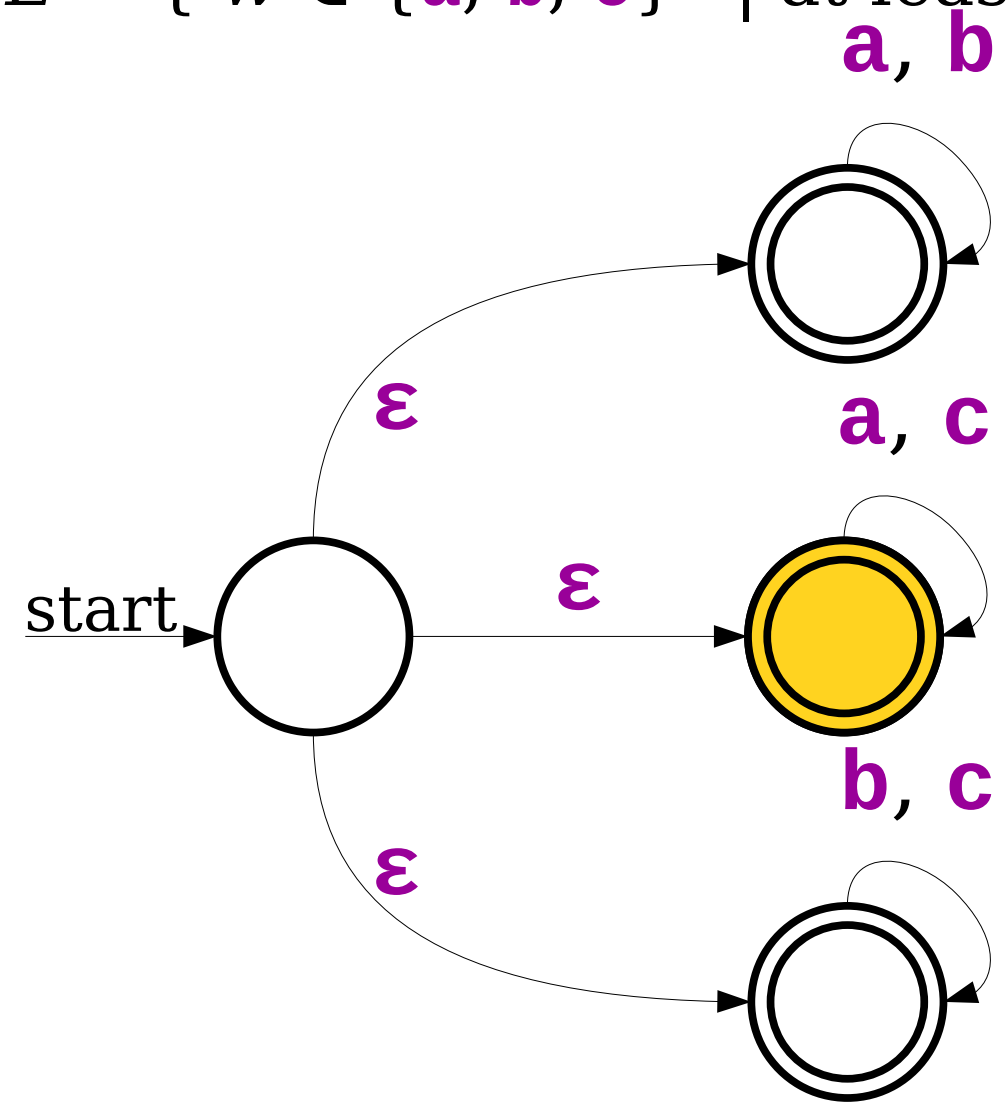
$L = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid$ at least one of **a**, **b**, or **c** is not in $w \}$

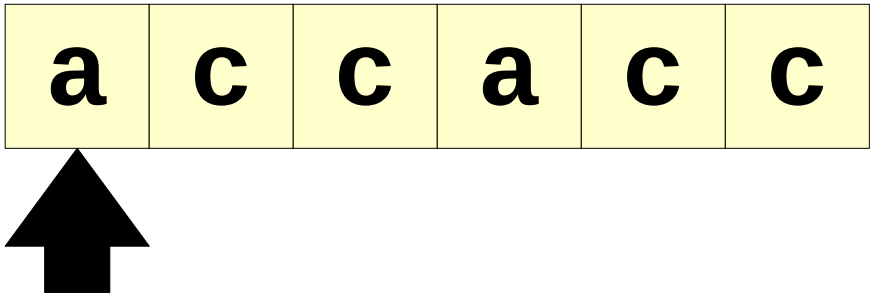




Guess-and-Check

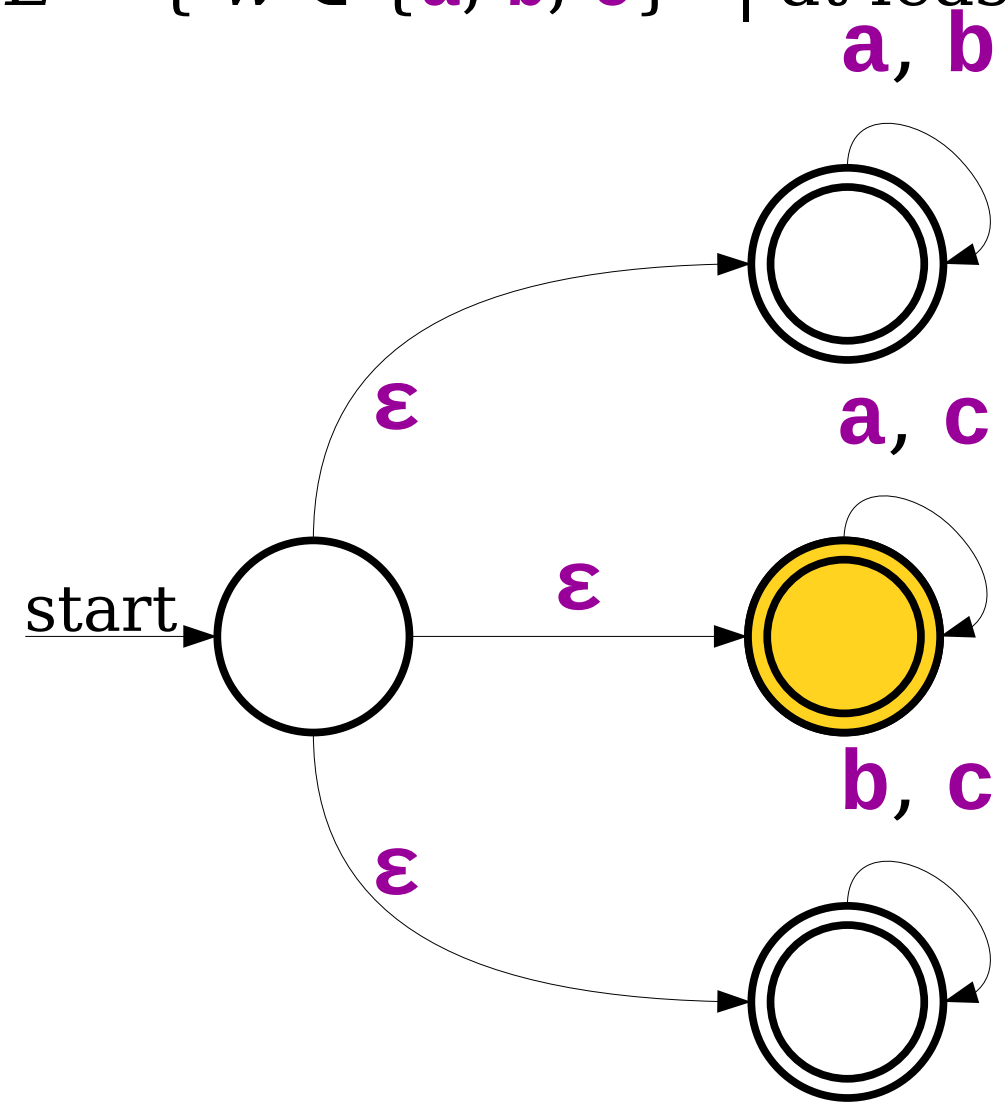
$L = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid$ at least one of **a**, **b**, or **c** is not in $w \}$

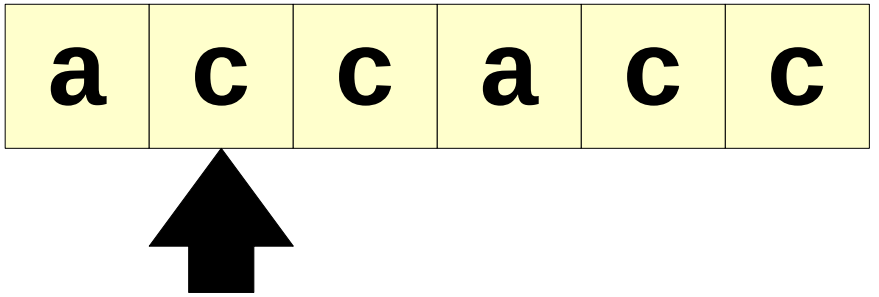




Guess-and-Check

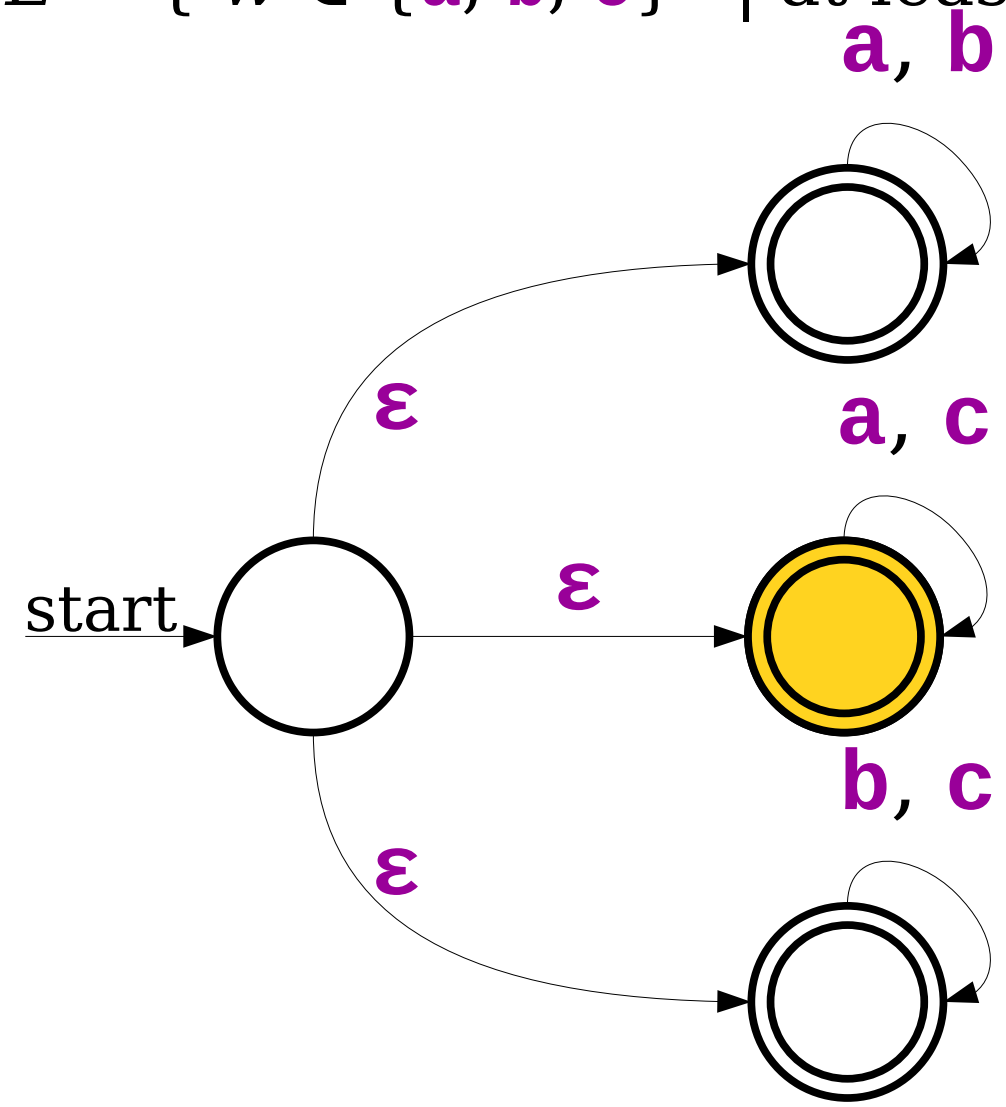
$L = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid$ at least one of **a**, **b**, or **c** is not in $w \}$

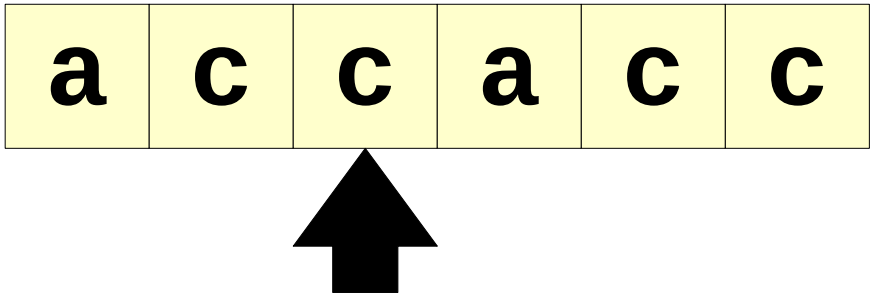




Guess-and-Check

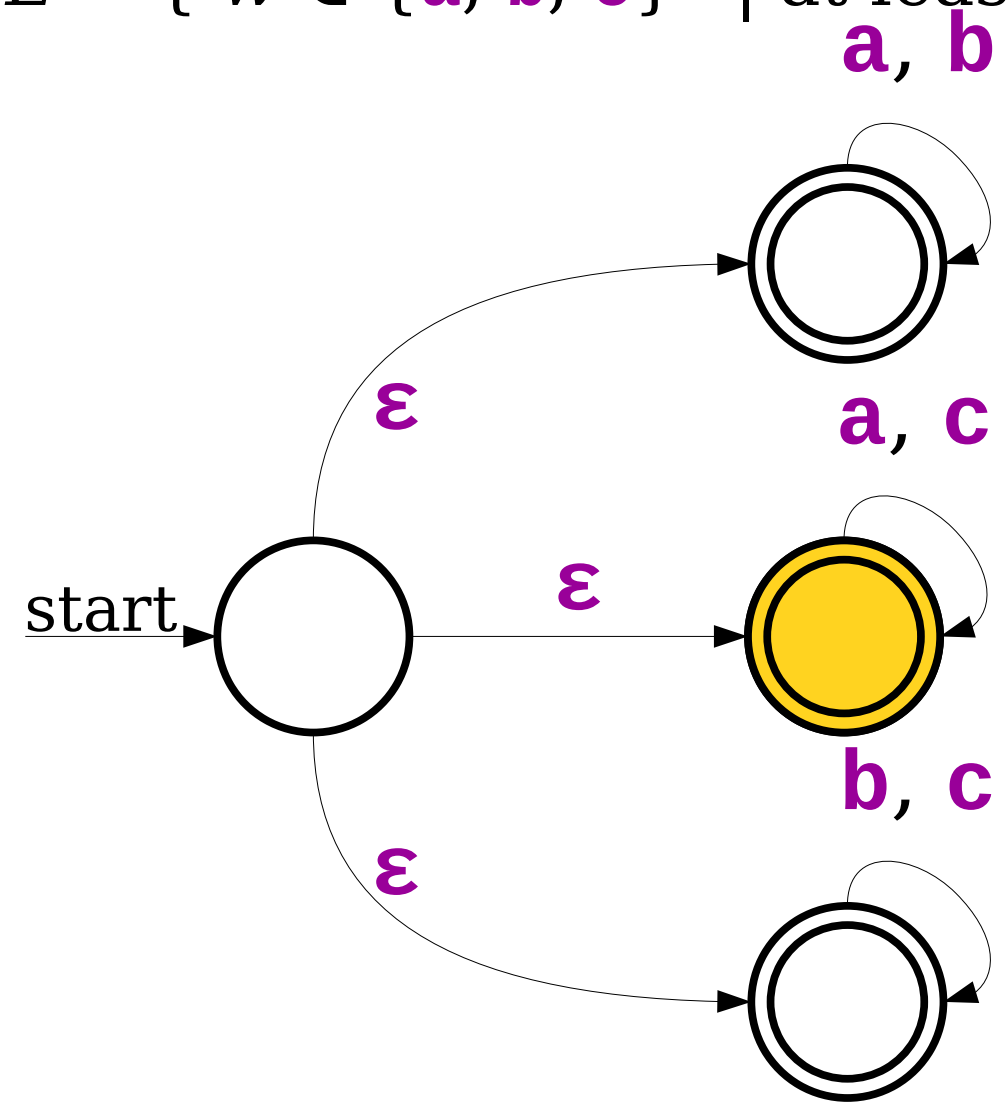
$L = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid$ at least one of **a**, **b**, or **c** is not in $w \}$

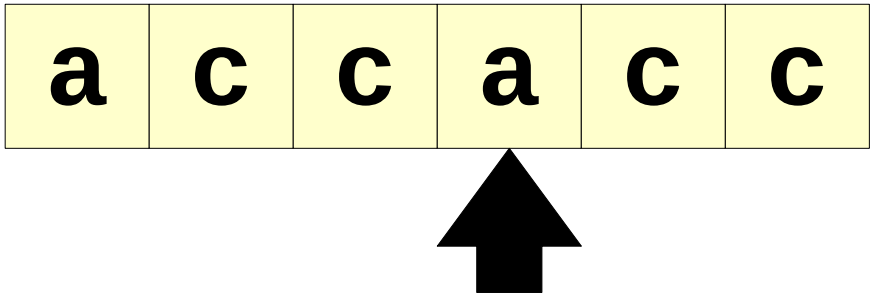




Guess-and-Check

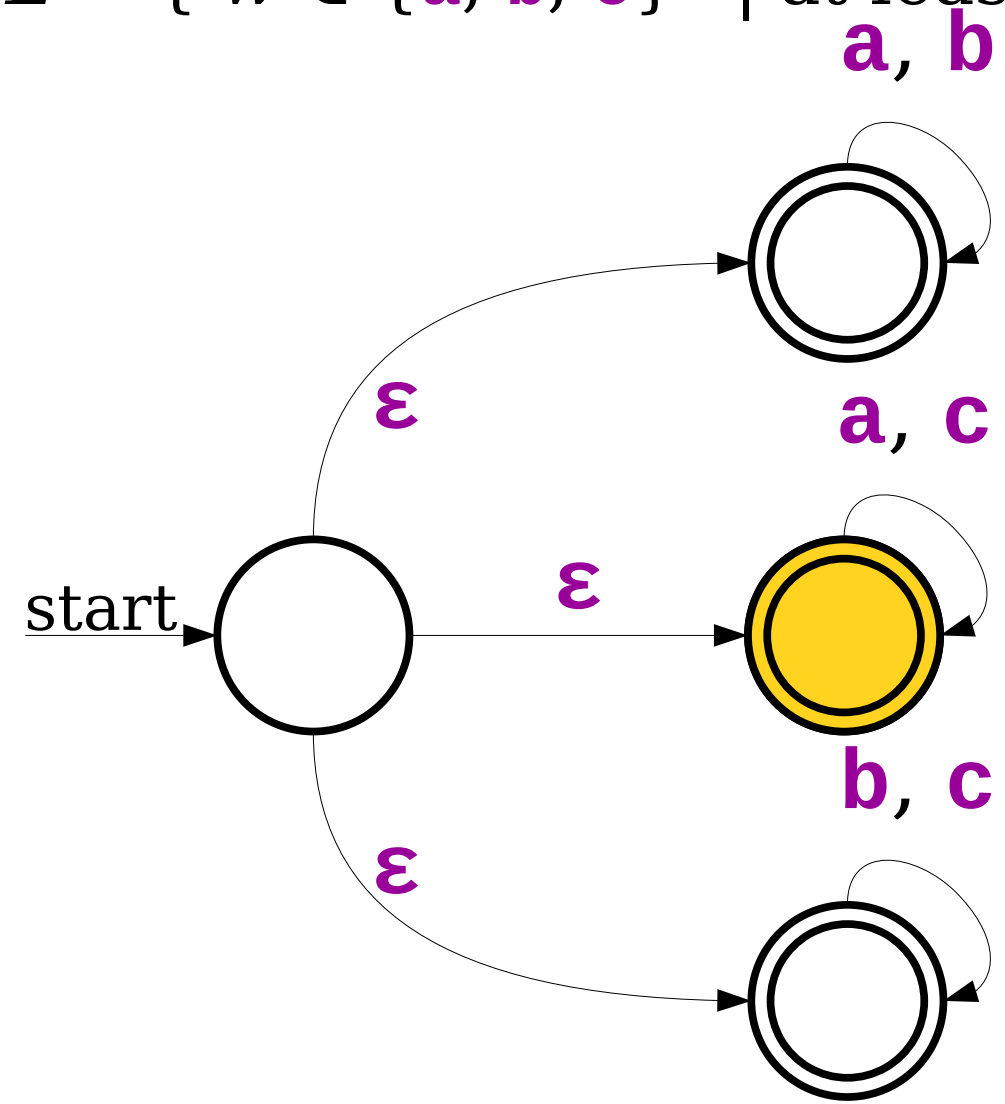
$L = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid$ at least one of **a**, **b**, or **c** is not in $w \}$

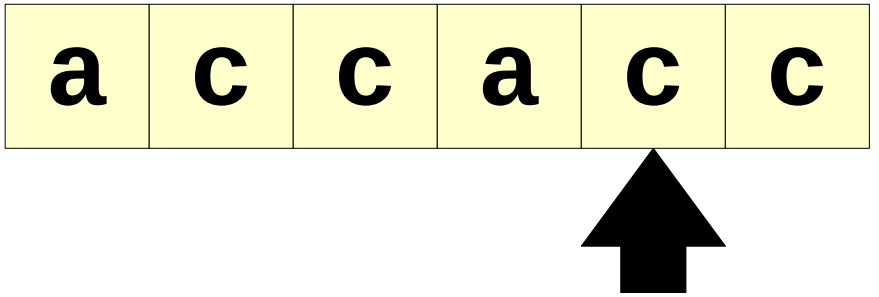




Guess-and-Check

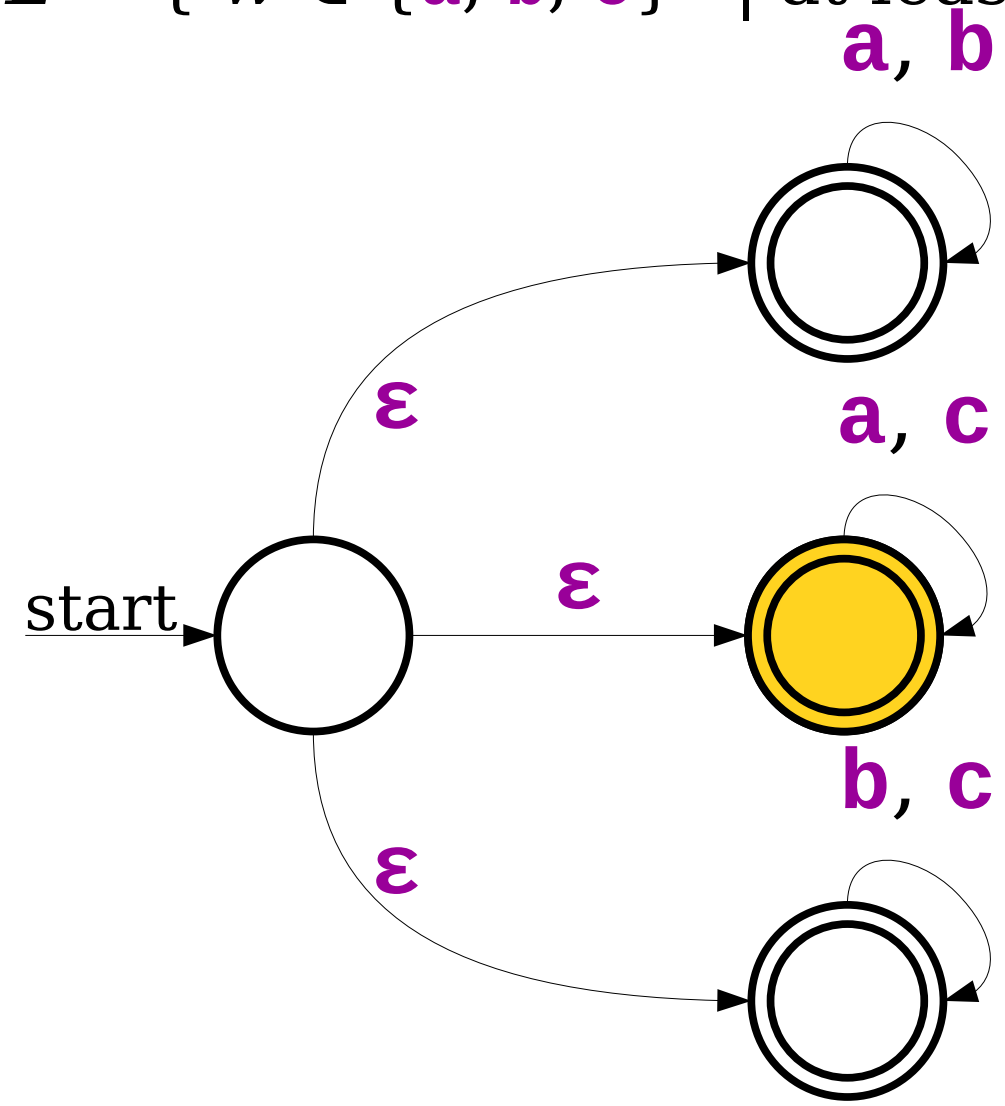
$L = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid$ at least one of **a**, **b**, or **c** is not in $w \}$

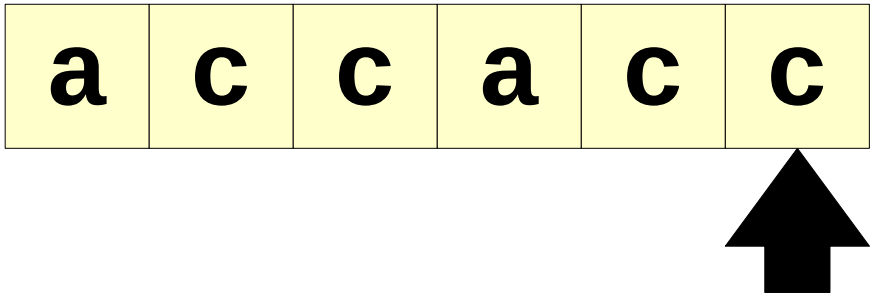




Guess-and-Check

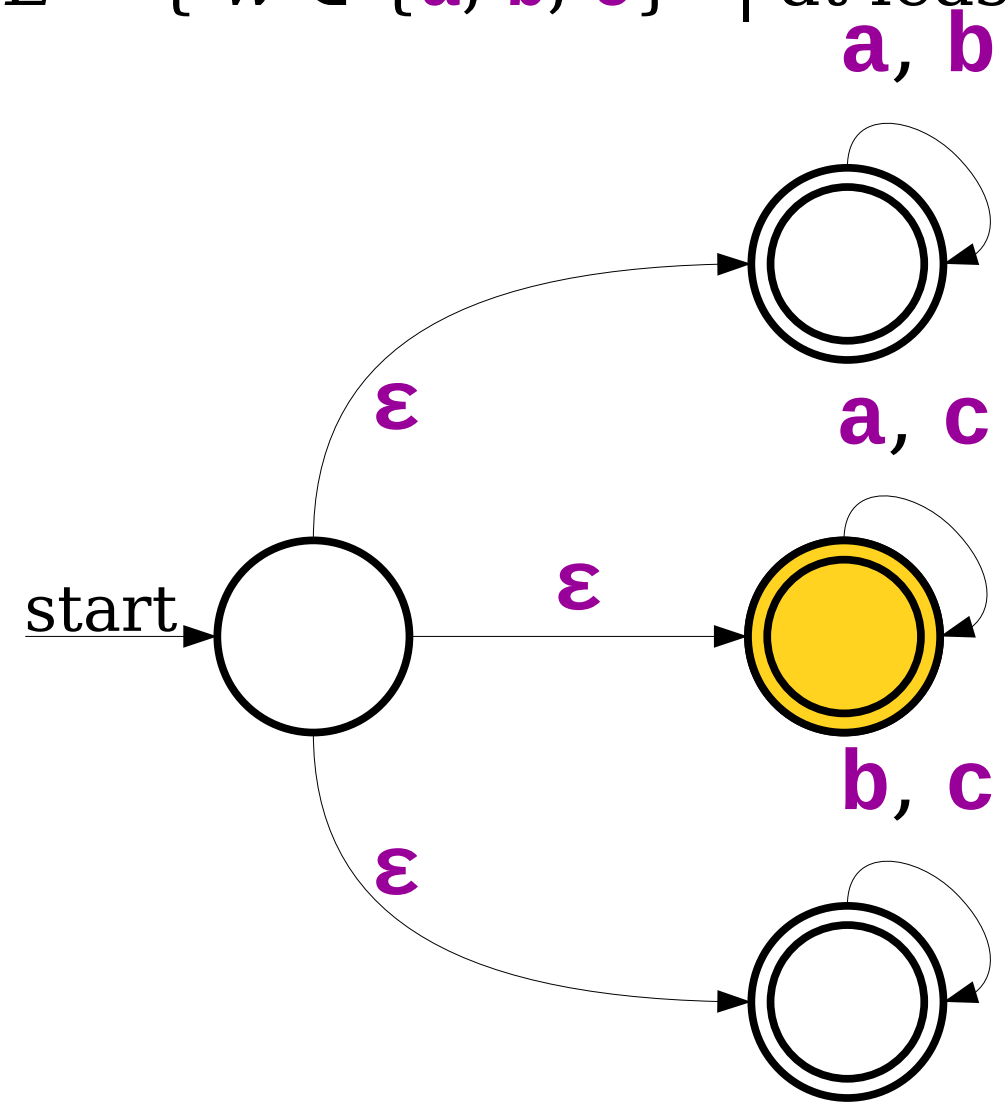
$L = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid$ at least one of **a**, **b**, or **c** is not in $w \}$





Guess-and-Check

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



a	c	c	a	c	c

Time-Out For Announcements!

Midterm Exam on Friday!

- Our midterm exam will be on Friday, July 26th from 5:00 – 8:00 PM in Hewlett 201 (our normal lecture room).
- You're responsible for lectures up to the end of week 3 and topics from PS1 – PS3. Later lectures and problem sets won't be tested here. Exam problems may build on the written or coding components from the problem sets.
- The exam is open-book, open-note, and closed-other-humans/AI.

Back to CS103!

Just how powerful are NFAs?

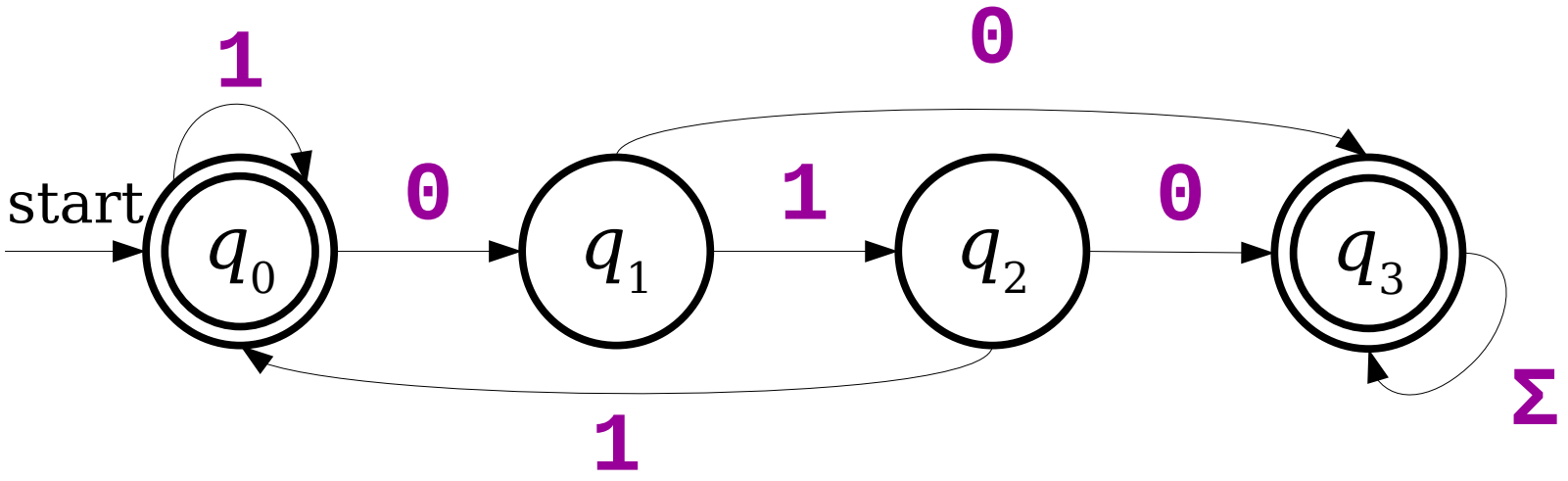
NFAs and DFAs

- Any language that can be accepted by a DFA can be accepted by an NFA.
- Why?
  - Every DFA essentially already *is* an NFA!
- ***Question:*** Can any language accepted by an NFA also be accepted by a DFA?
- Surprisingly, the answer is ***yes***!

Thought Experiment:

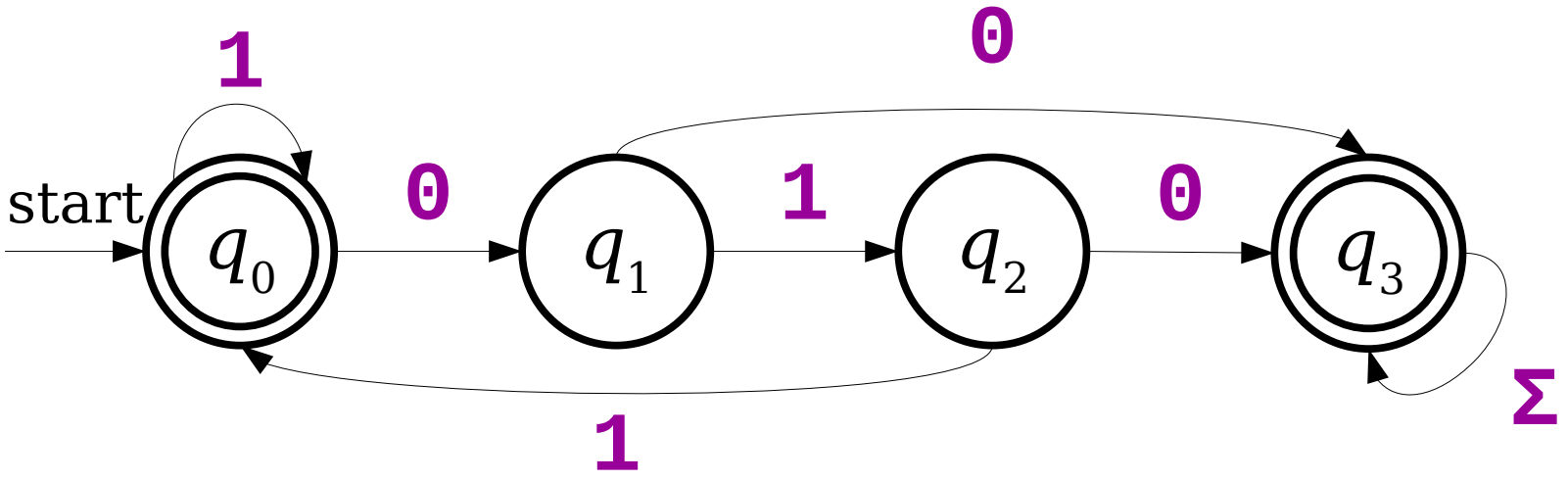
How would you simulate a finite automata in software?

Tabular DFAs



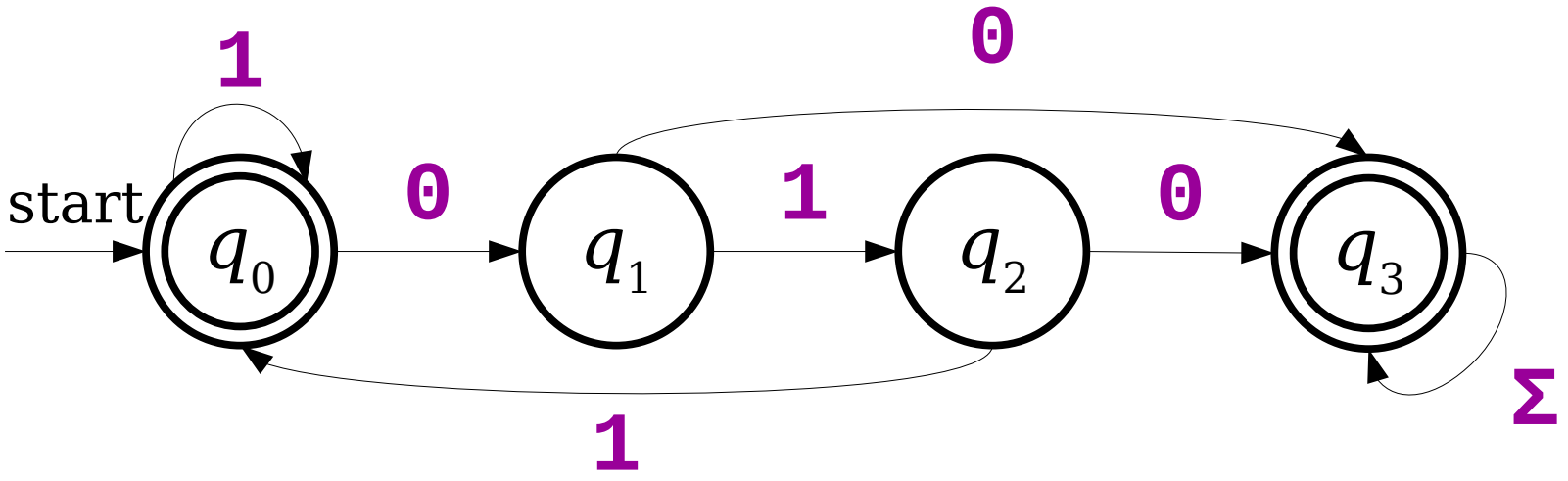
	0	1
q_0		
q_1		
q_2		
q_3		

Tabular DFAs



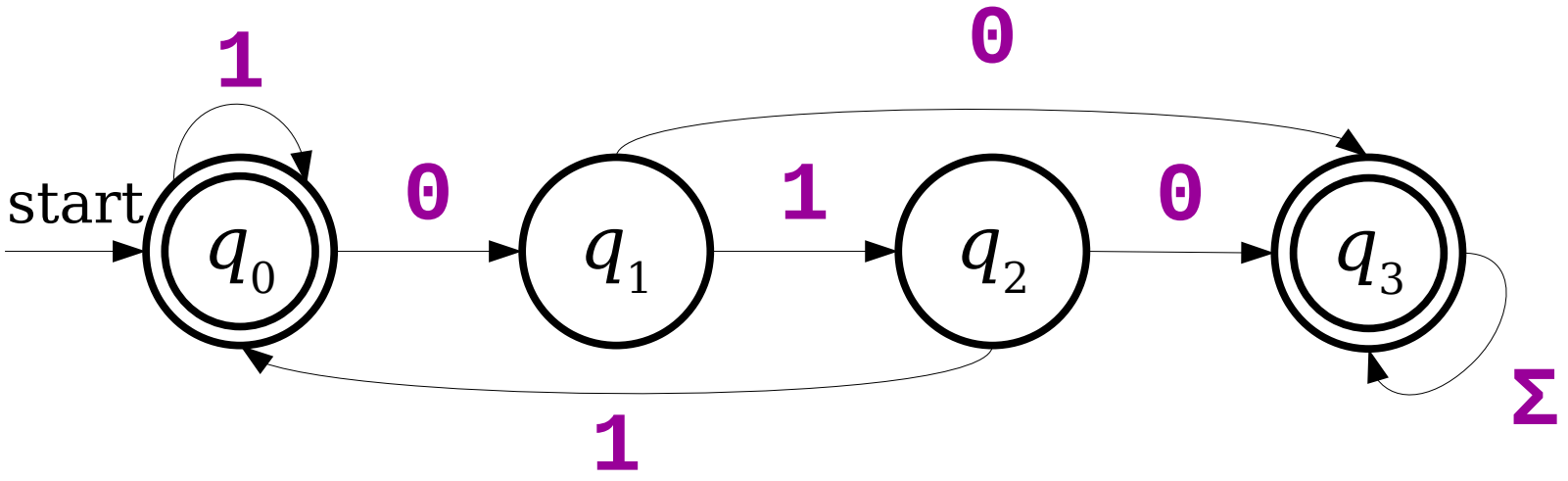
	0	1
q_0	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
q_3	q_3	q_3

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

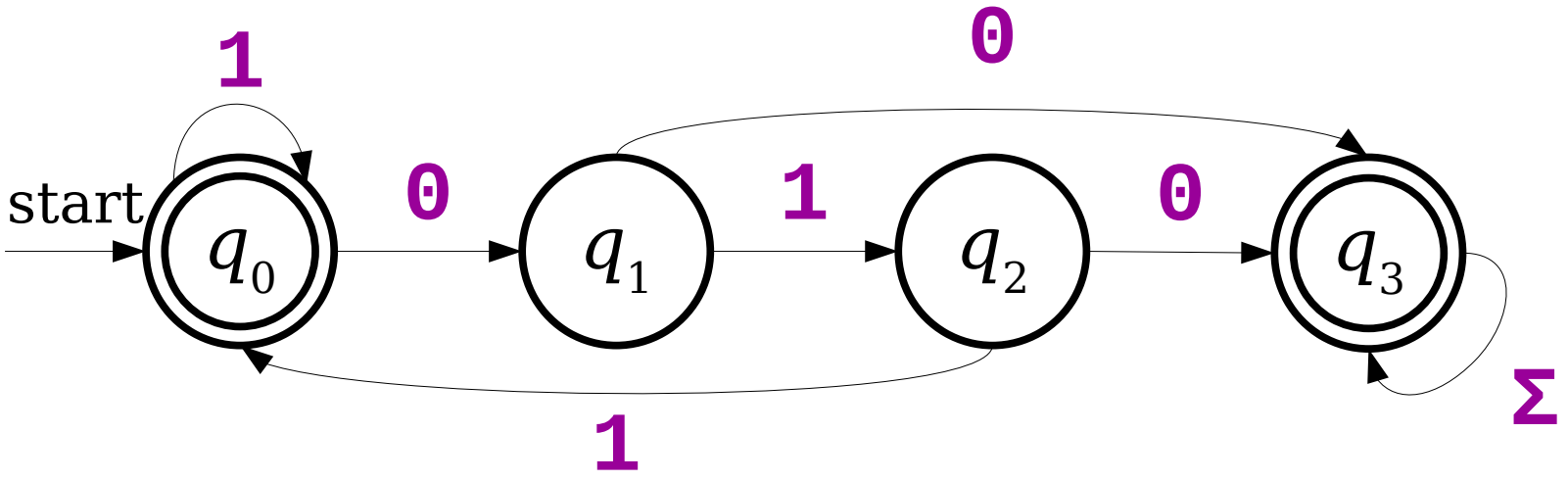
Tabular DFAs



	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.

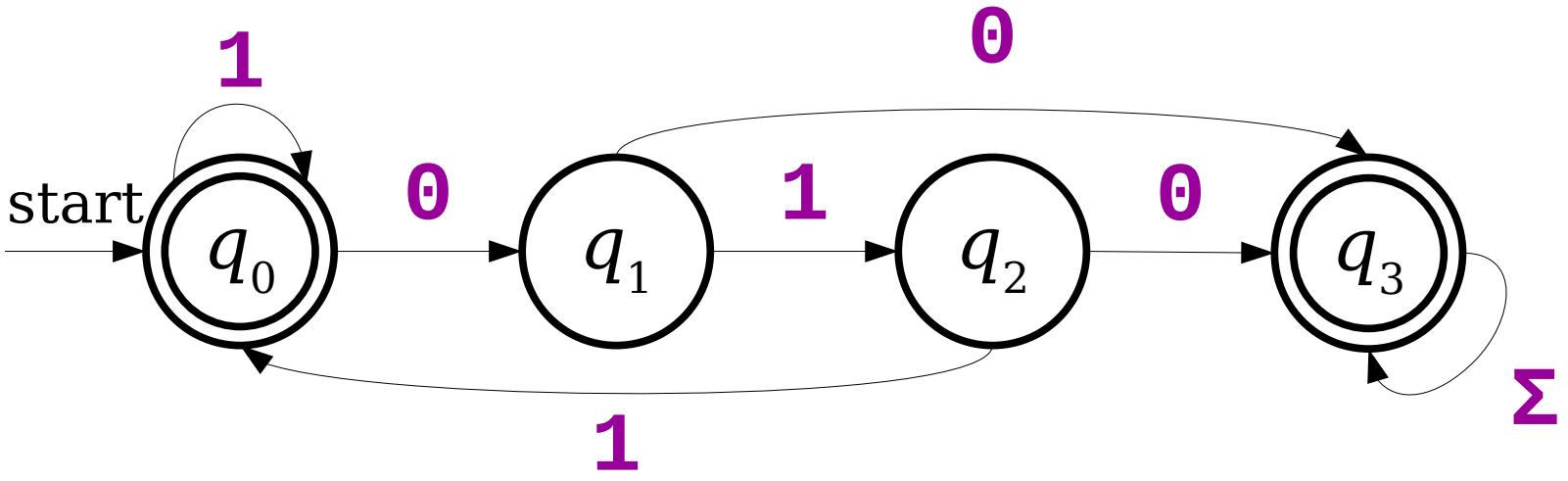
Tabular DFAs



	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.

Tabular DFAs



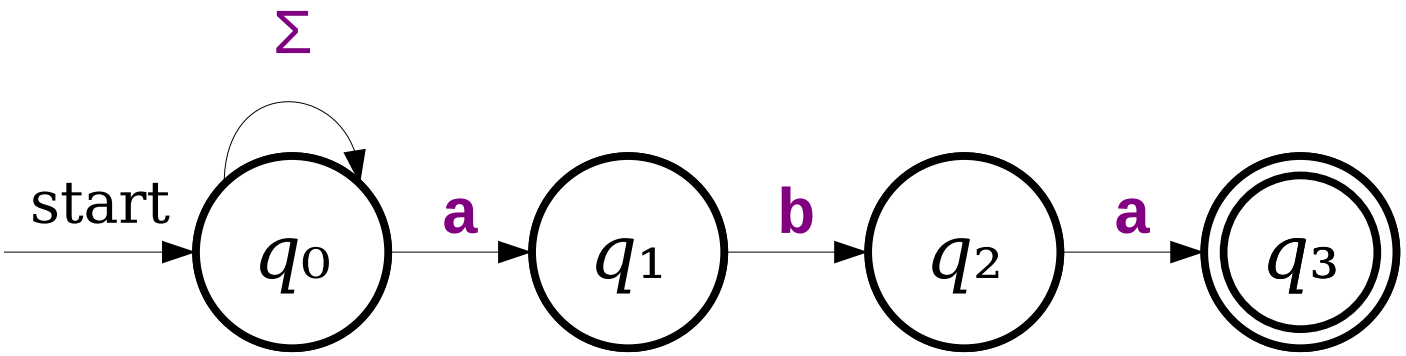
	0	1
$*q_0$	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
$*q_3$	q_3	q_3

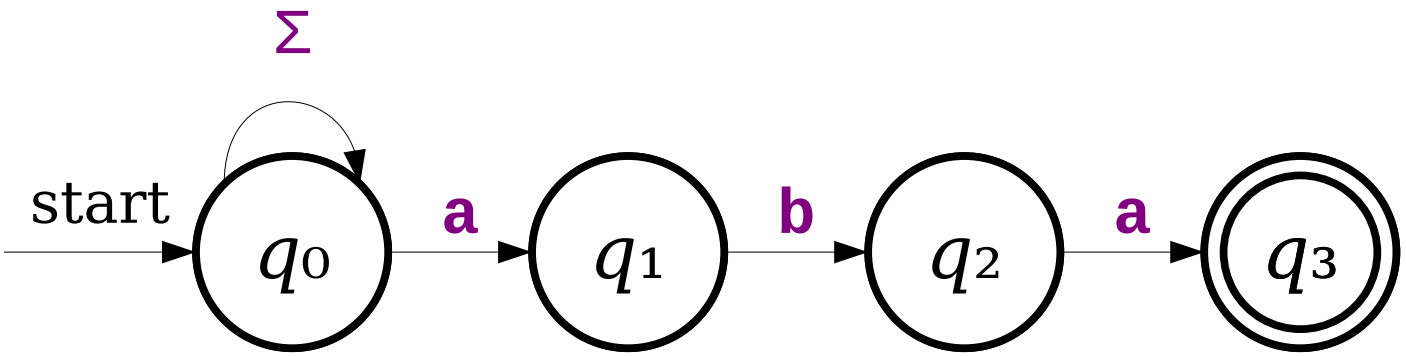
Question to ponder:
Why isn't there a column here for Σ?

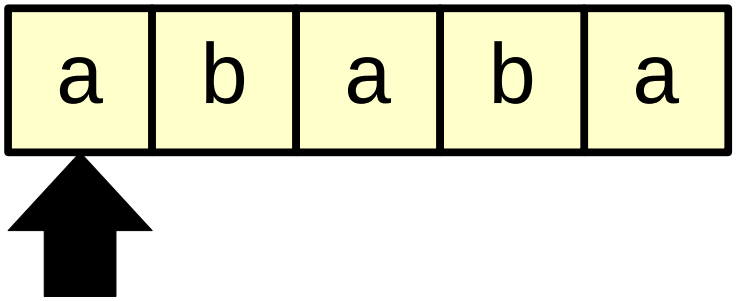
Code? In a Theory Class?

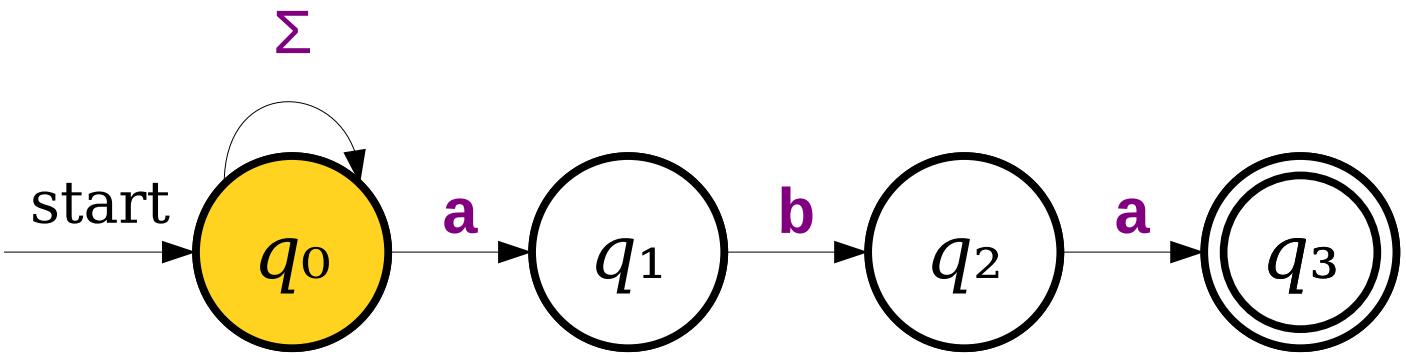
```
int kTransitionTable[kNumStates][kNumSymbols] = {
      {0, 0, 1, 3, 7, 1, …},
       …
};
bool kAcceptTable[kNumStates] = {
     false,
     true,
     true,
     …
};
bool SimulateDFA(string input) {
     int state = 0;
     for (char ch: input) {
          state = kTransitionTable[state][ch];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

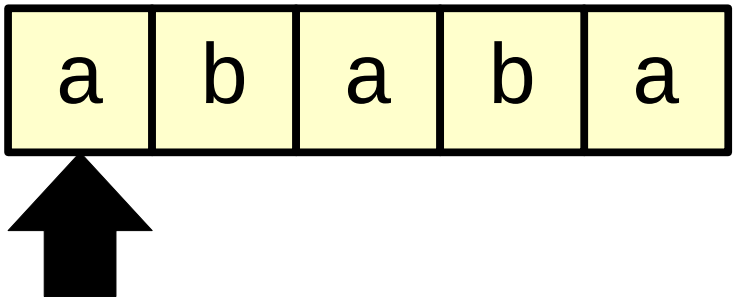
Can we do something similar for NFAs?

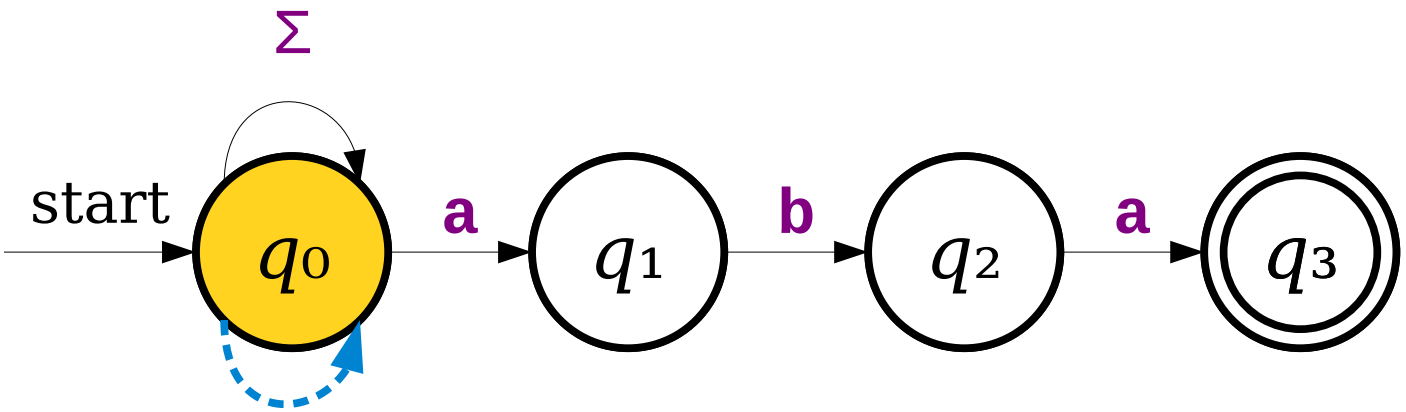

Σ
start
q_0
a
q_1
b
q_2
a
q_3

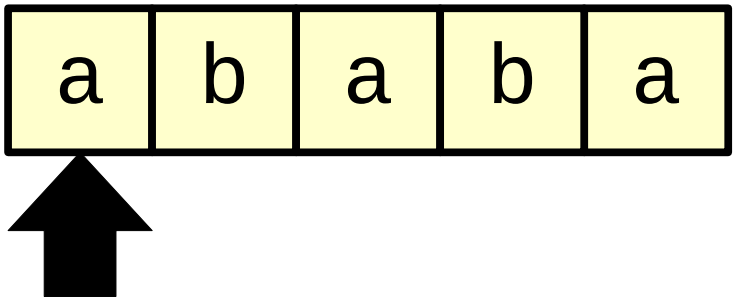

Σ
start
q_0
a
q_1
b
q_2
a
q_3

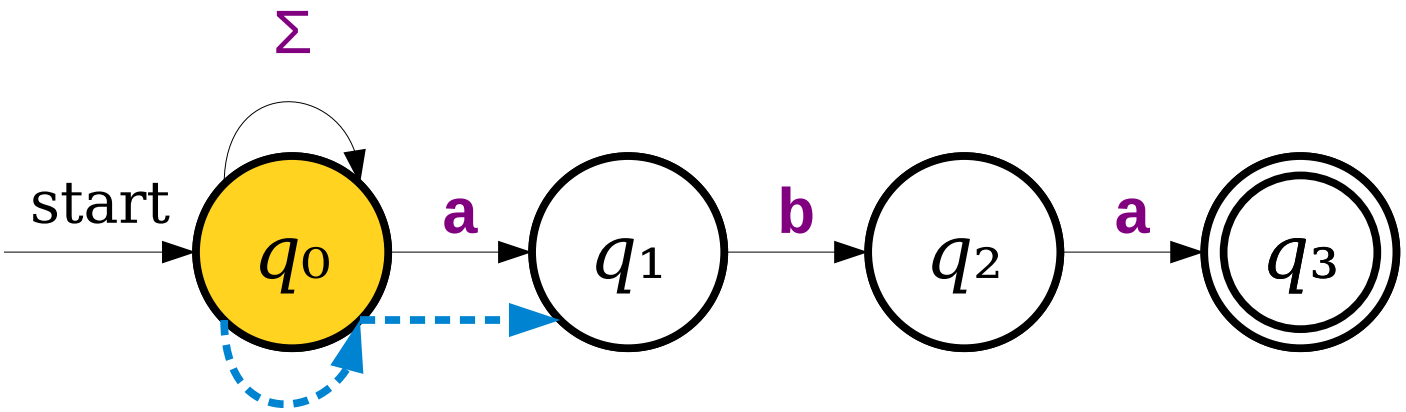

a
b
a
b
a

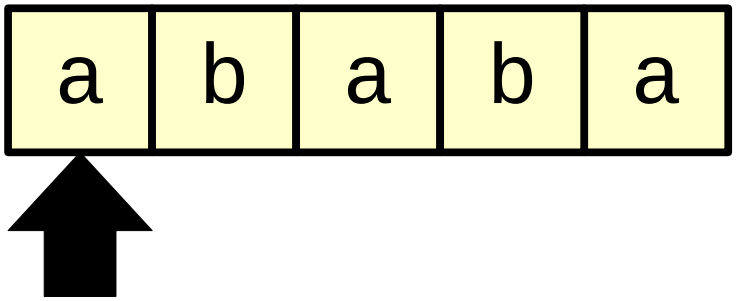

Σ
start
q_0
a
q_1
b
q_2
a
q_3

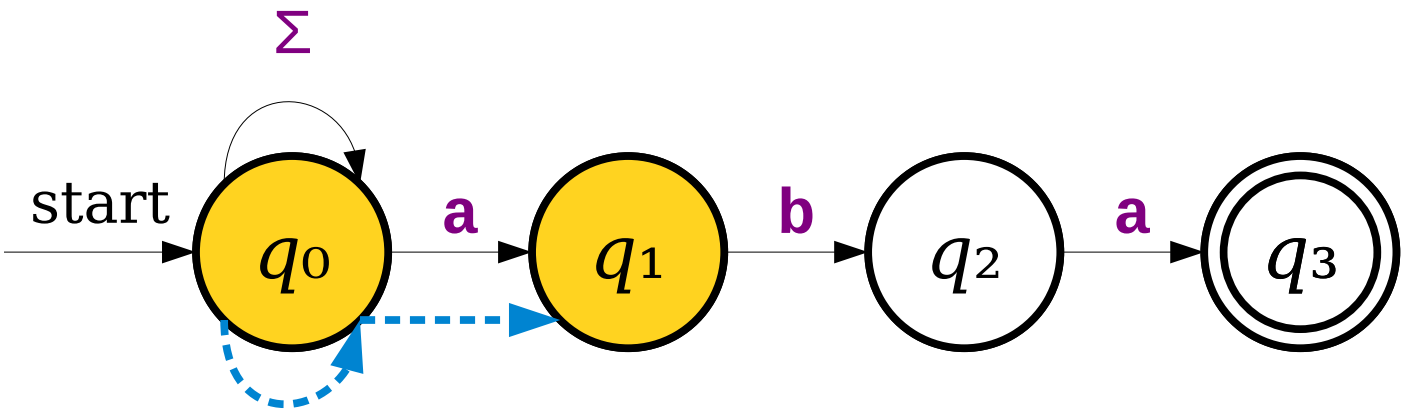

a
b
a
b
a

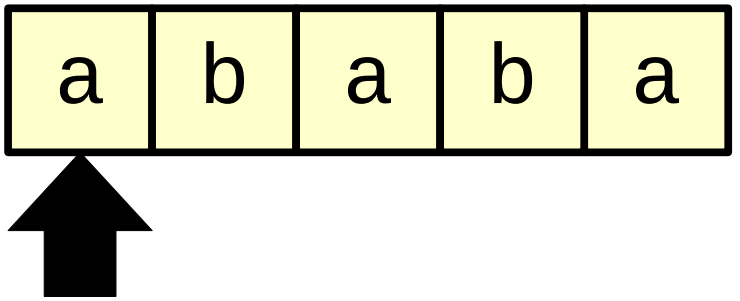

Σ
start
q_0
a
q_1
b
q_2
a
q_3

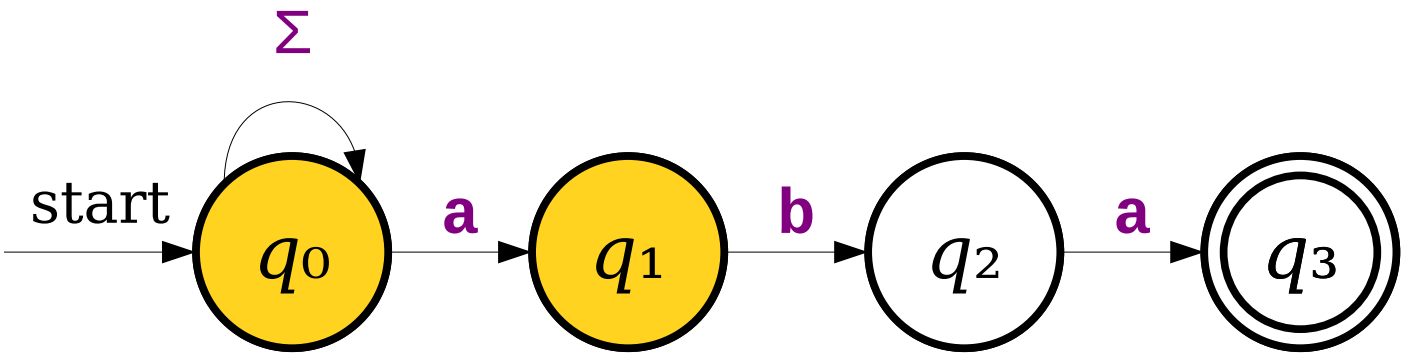

a
b
a
b
a

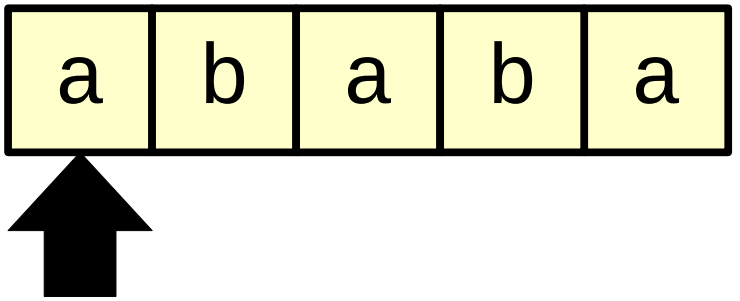

Σ
start
q_0
a
q_1
b
q_2
a
q_3

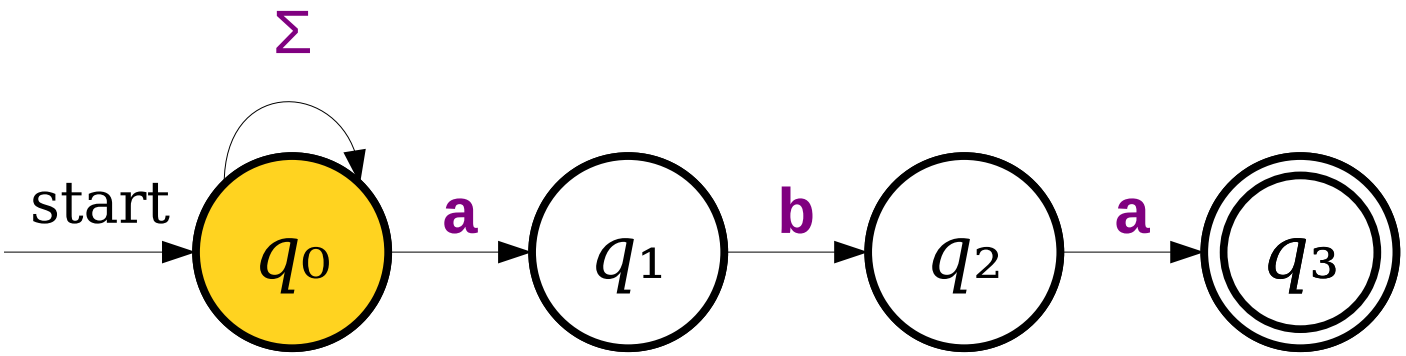

a
b
a
b
a

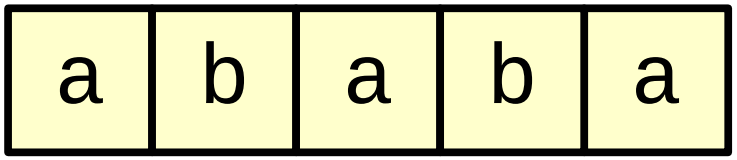

Σ
start
q_0
a
q_1
b
q_2
a
q_3

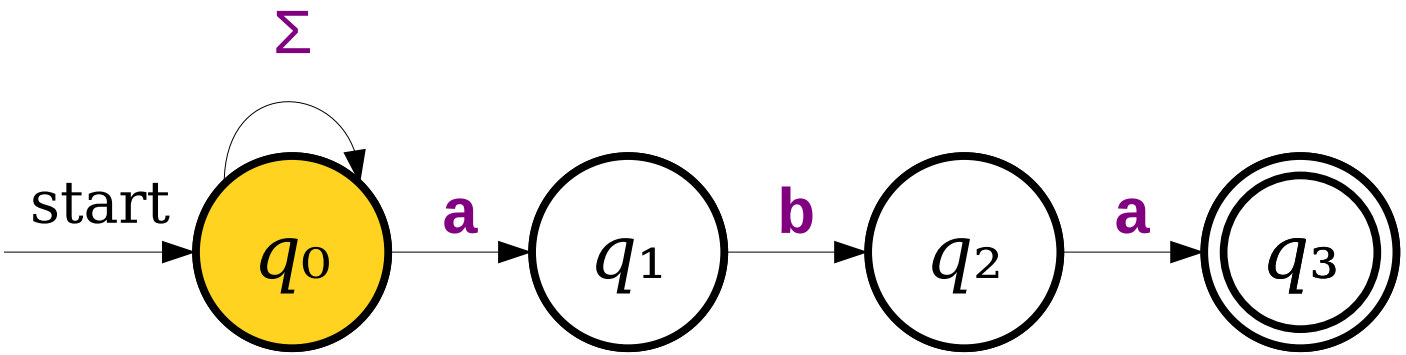

a
b
a
b
a

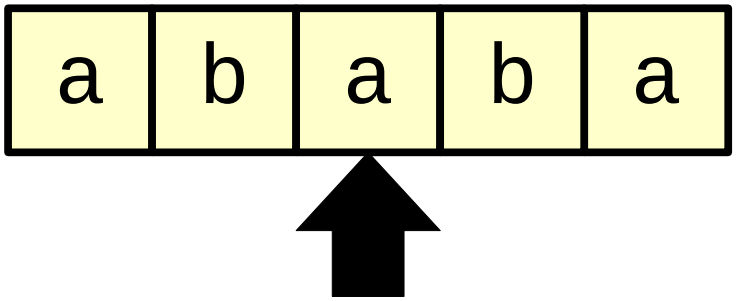

Σ
start
q_0
a
q_1
b
q_2
a
q_3

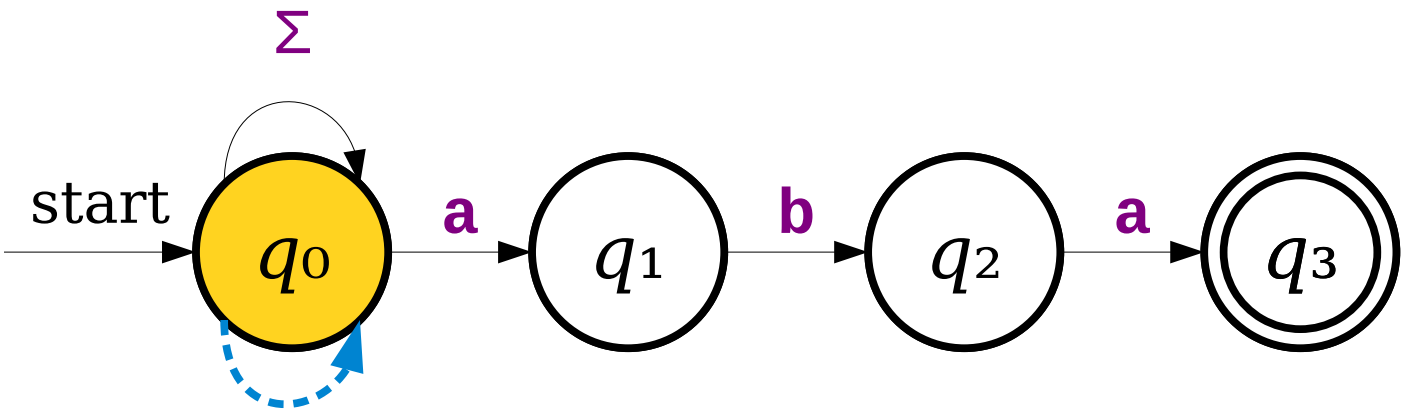

a
b
a
b
a

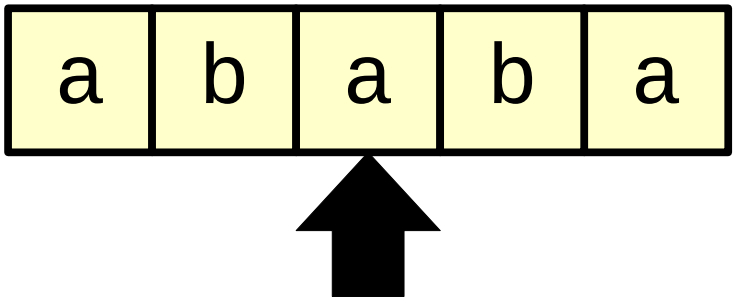

Σ
start
q_0
a
q_1
b
q_2
a
q_3

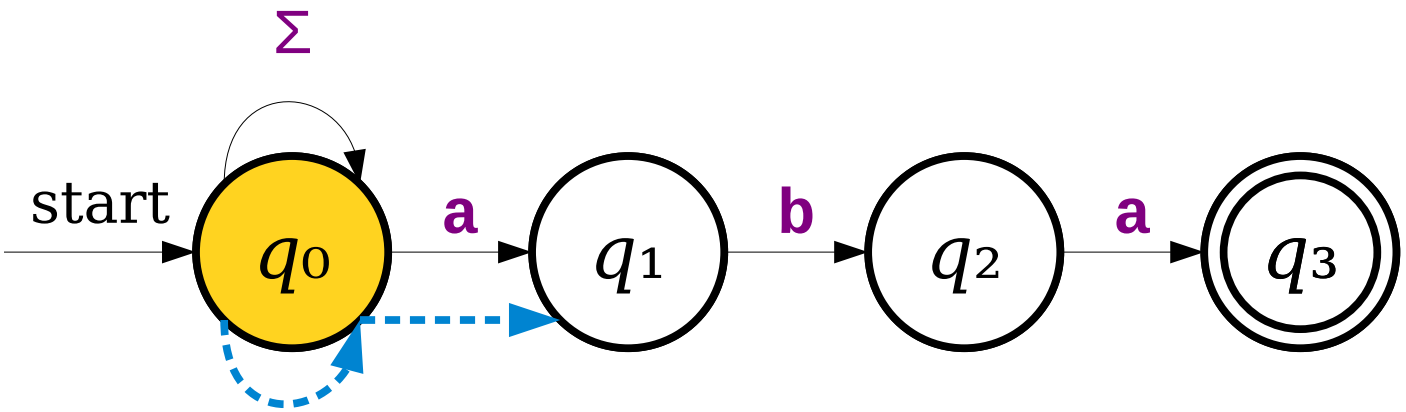

a
b
a
b
a

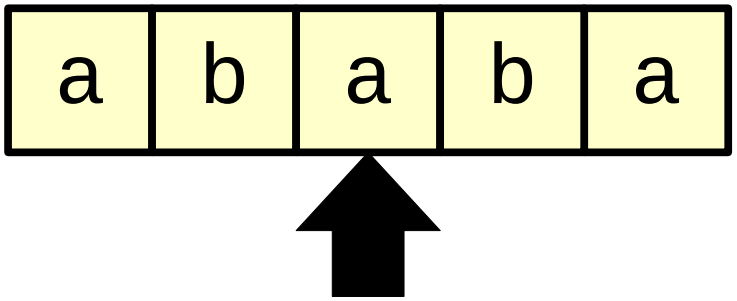

Σ
start
q_0
a
q_1
b
q_2
a
q_3

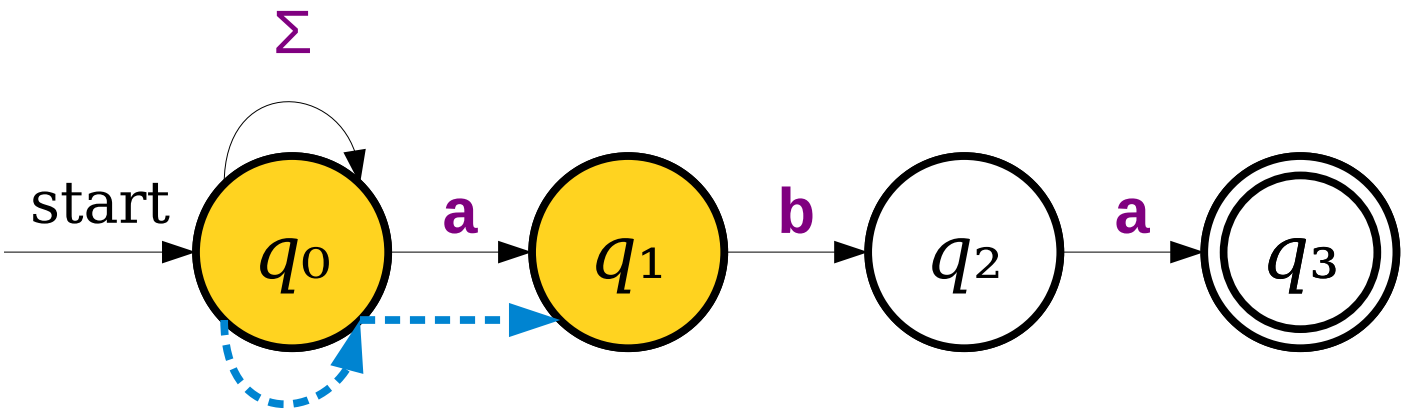

a
b
a
b
a

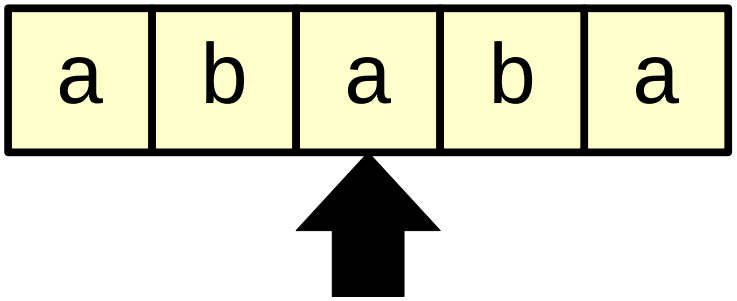

Σ
start
q_0
a
q_1
b
q_2
a
q_3

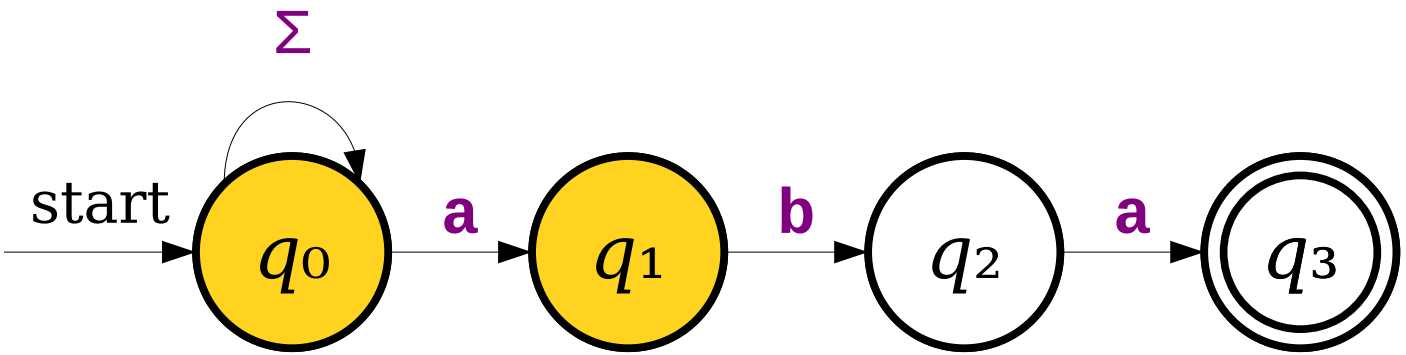

a
b
a
b
a

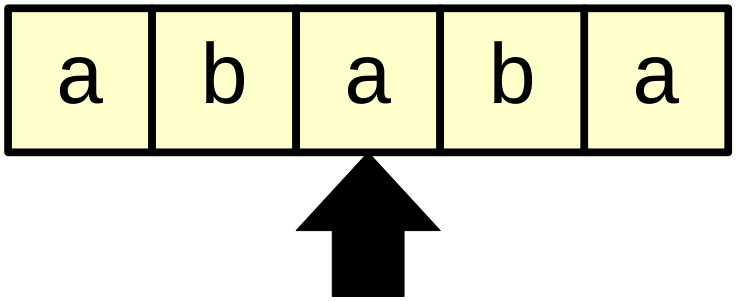

Σ
start
q_0
a
q_1
b
q_2
a
q_3

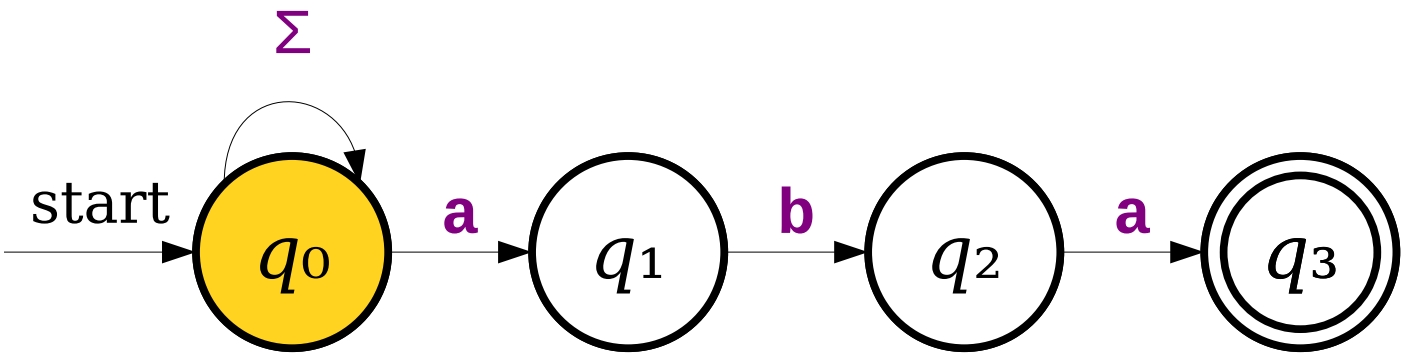

a
b
a
b
a

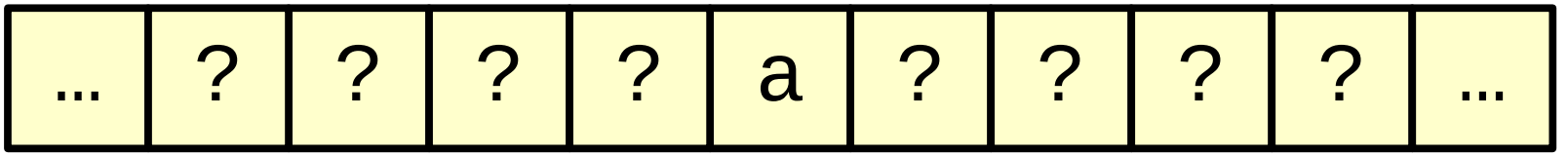

Σ
start
q_0
a
q_1
b
q_2
a
q_3

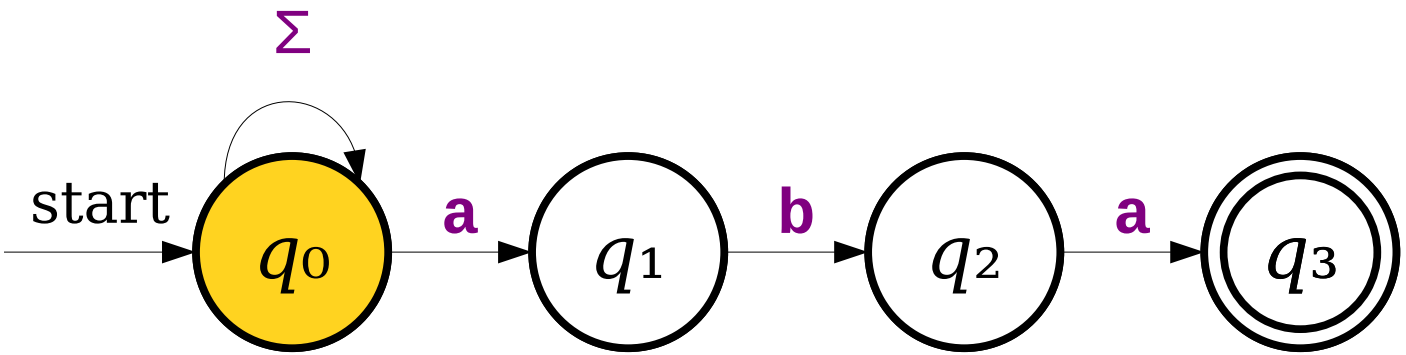

a
b
a
b
a

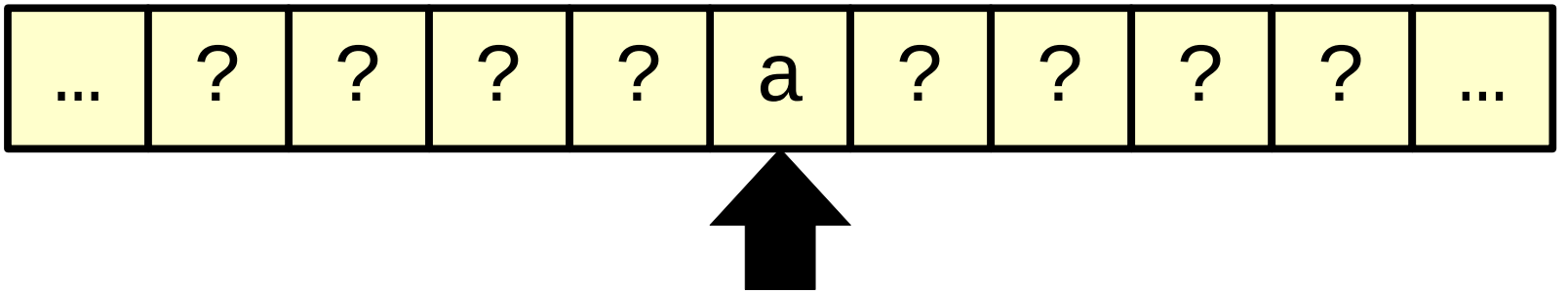

Σ
start
q_0
a
q_1
b
q_2
a
q_3

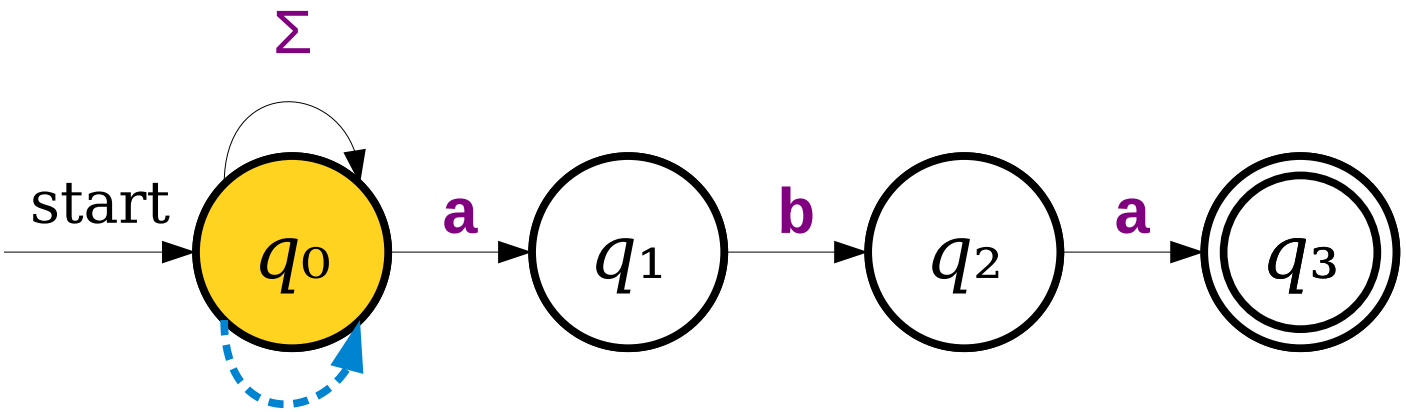

a
b
a
b
a

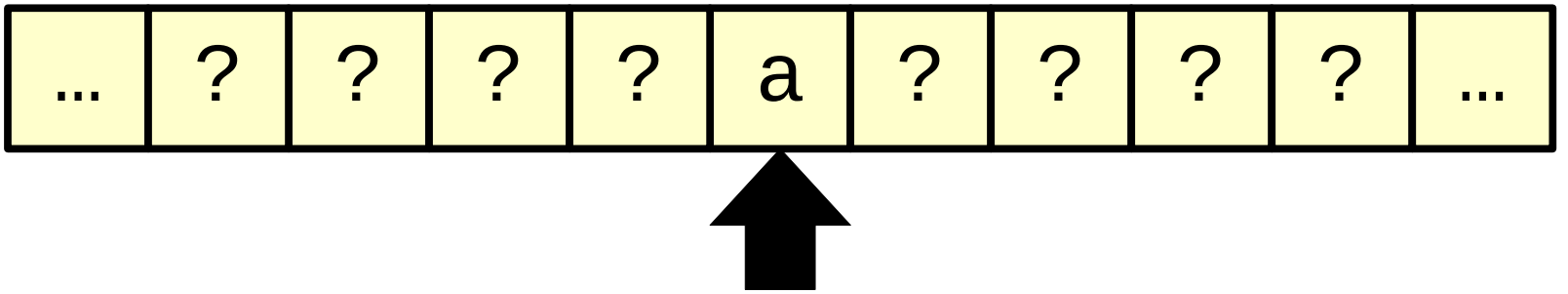

Σ
start
q_0
a
q_1
b
q_2
a
q_3

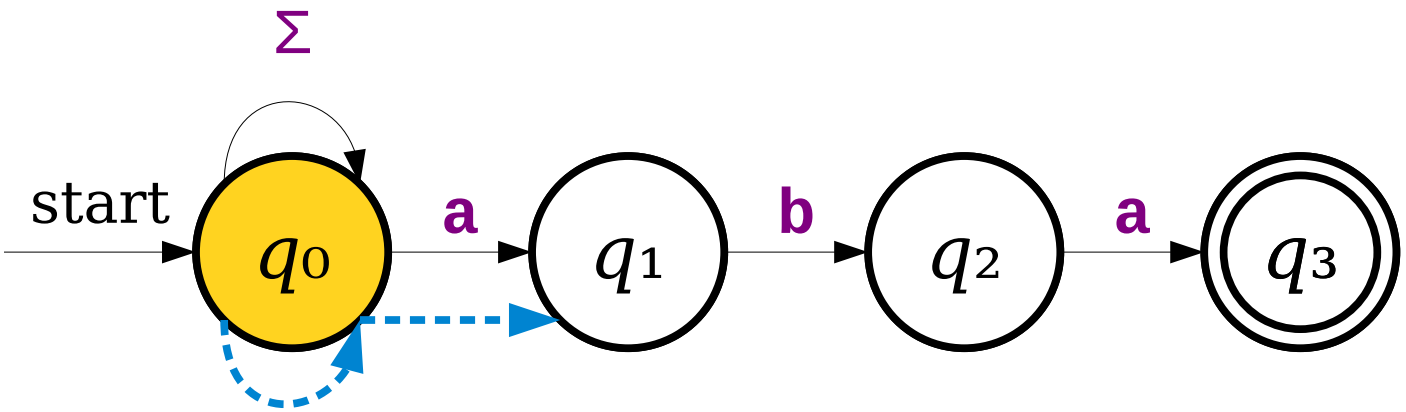

...
?
?
?
?
a
?
?
?
?
...

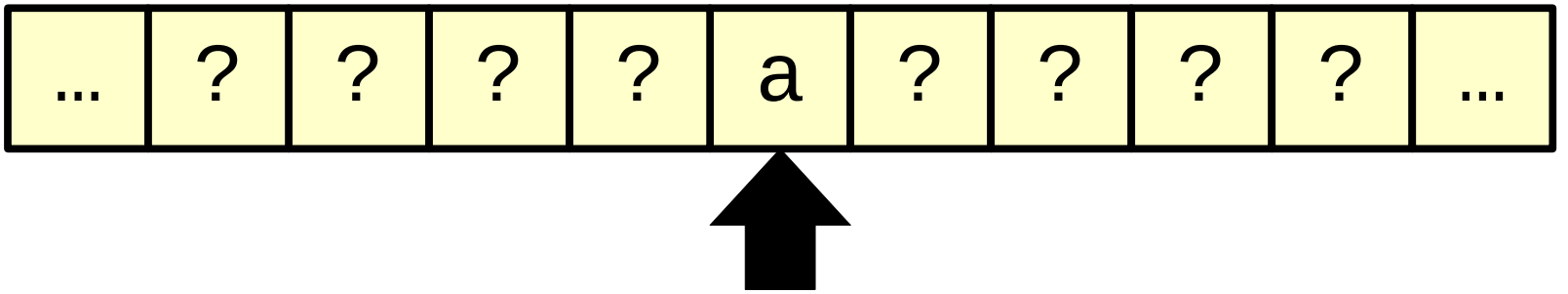

Σ
start
q_0
a
q_1
b
q_2
a
q_3

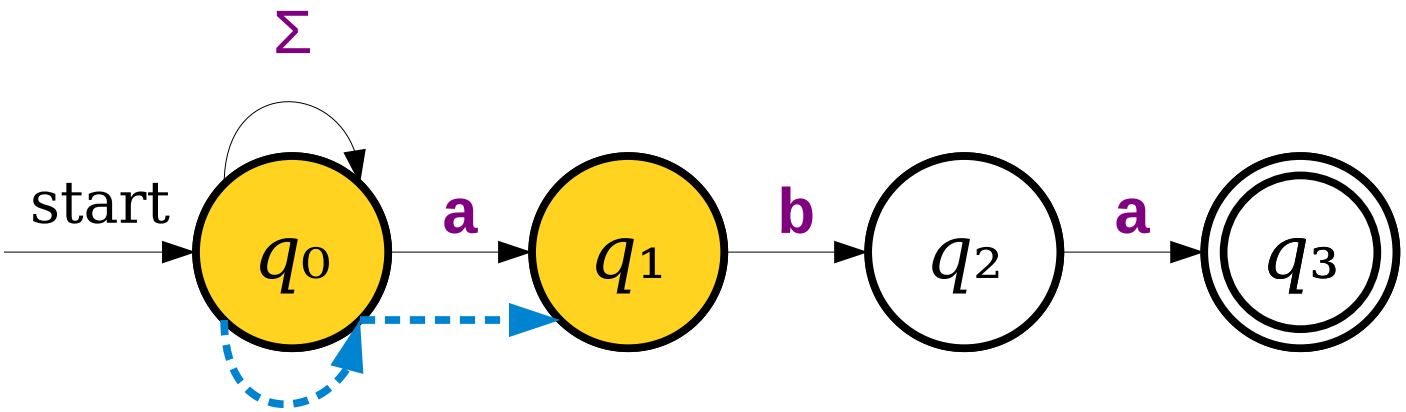

...
?
?
?
?
a
?
?
?
?
...

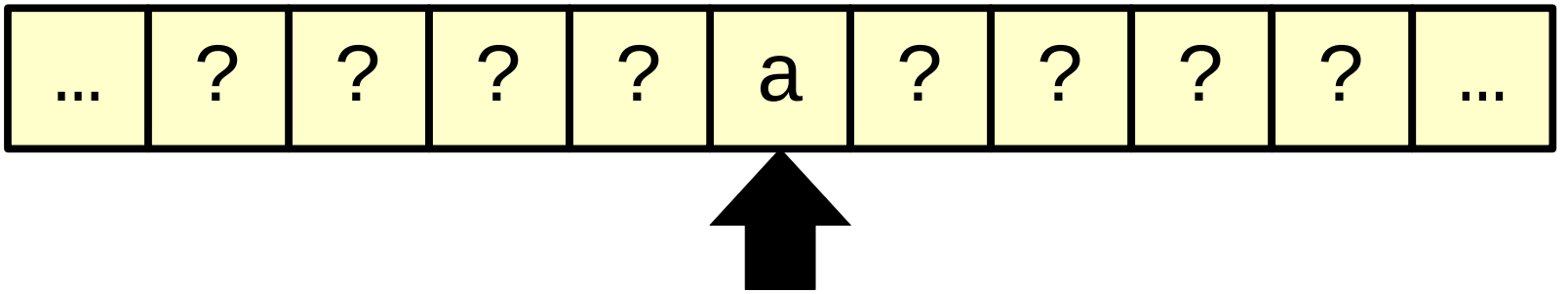

Σ
start
q_0
a
q_1
b
q_2
a
q_3

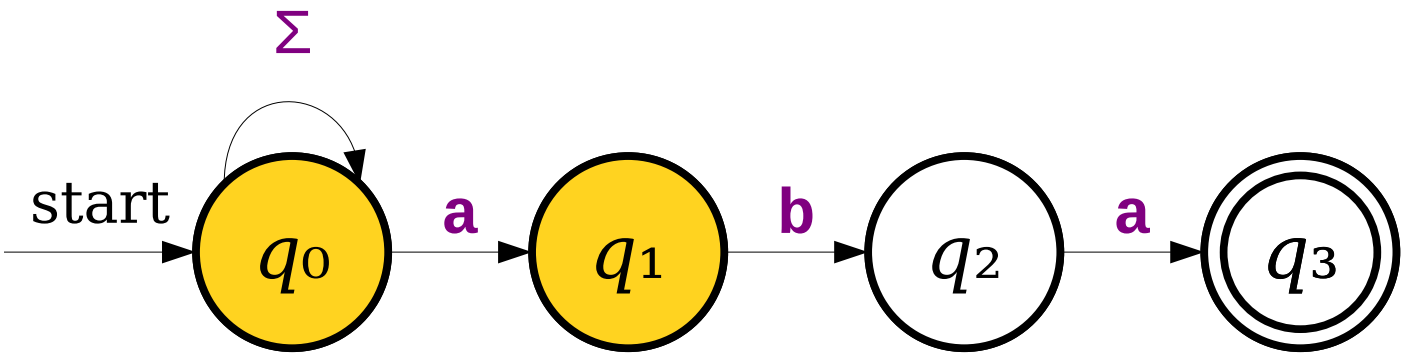

...
?
?
?
?
a
?
?
?
?
...

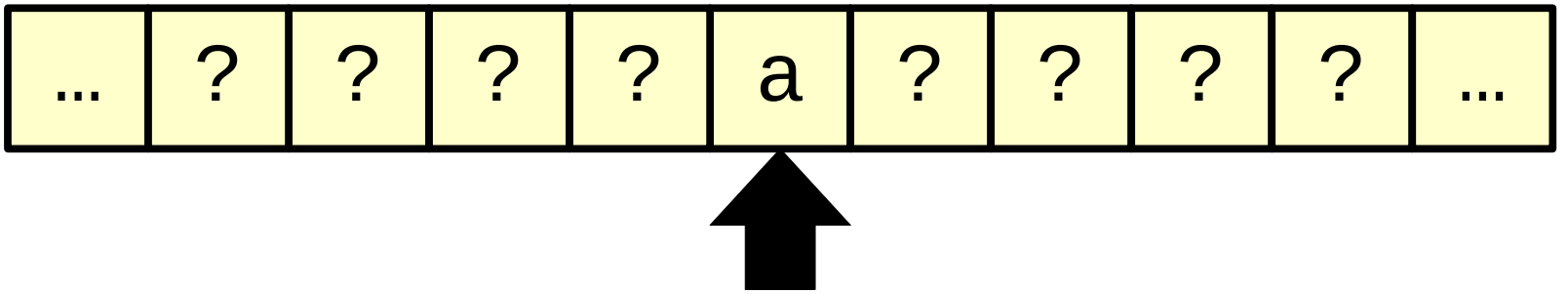

Σ
start
q_0
a
q_1
b
q_2
a
q_3

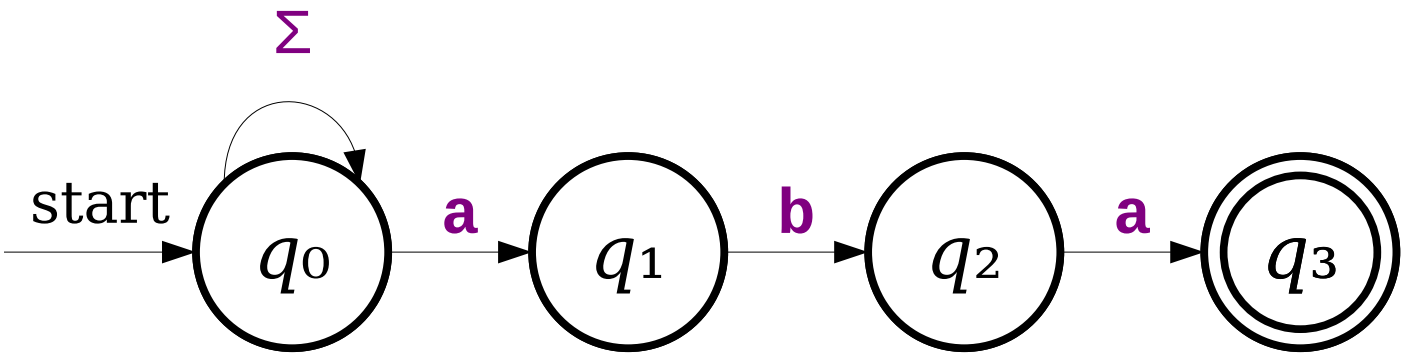

...
?
?
?
?
a
?
?
?
?
...


Σ
start
q_0
a
q_1
b
q_2
a
q_3

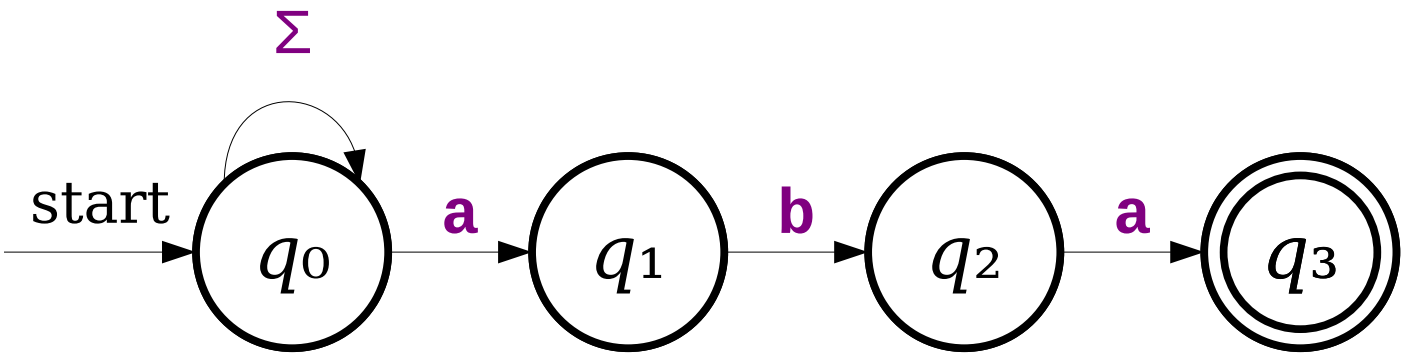

...
?
?
?
?
a
?
?
?
?
...


Σ
start
q_0
a
q_1
b
q_2
a
q_3

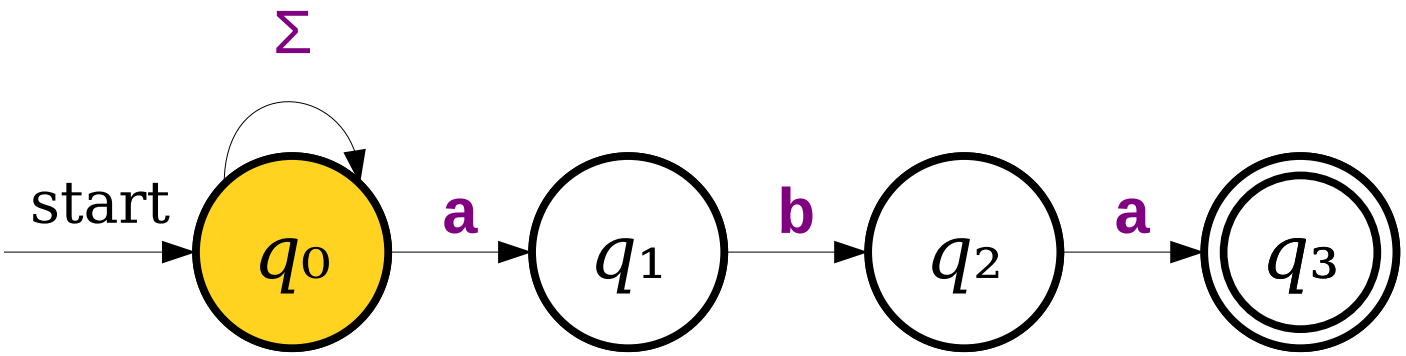

...
?
?
?
?
a
?
?
?
?
...



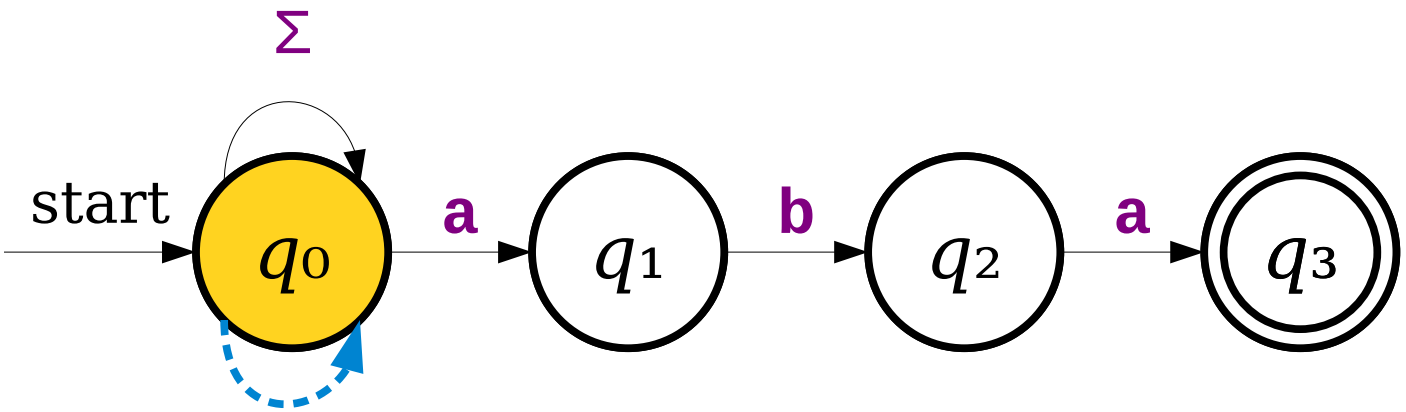
	a
$\{q_0\}$	$\{q_0, q_1\}$



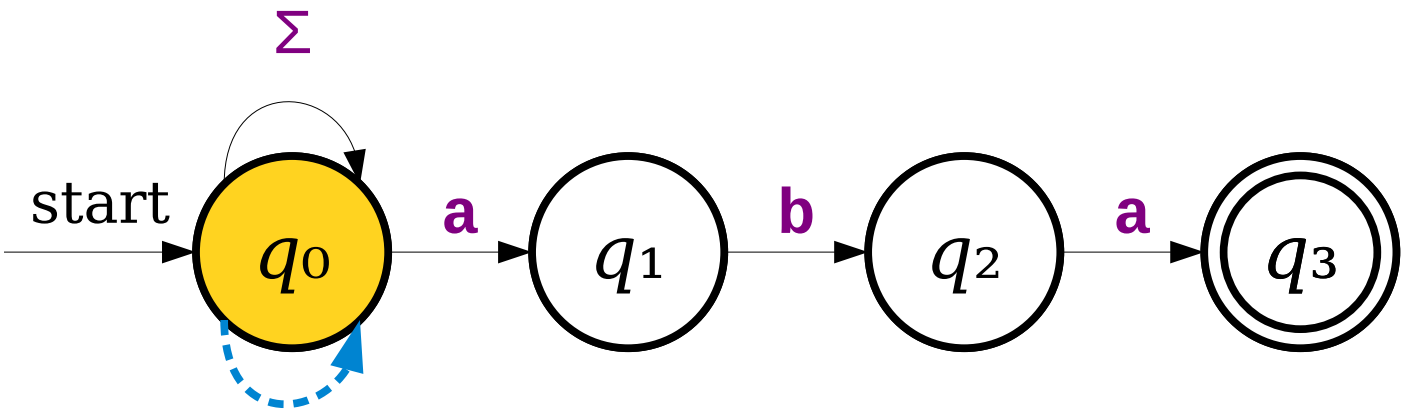
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	



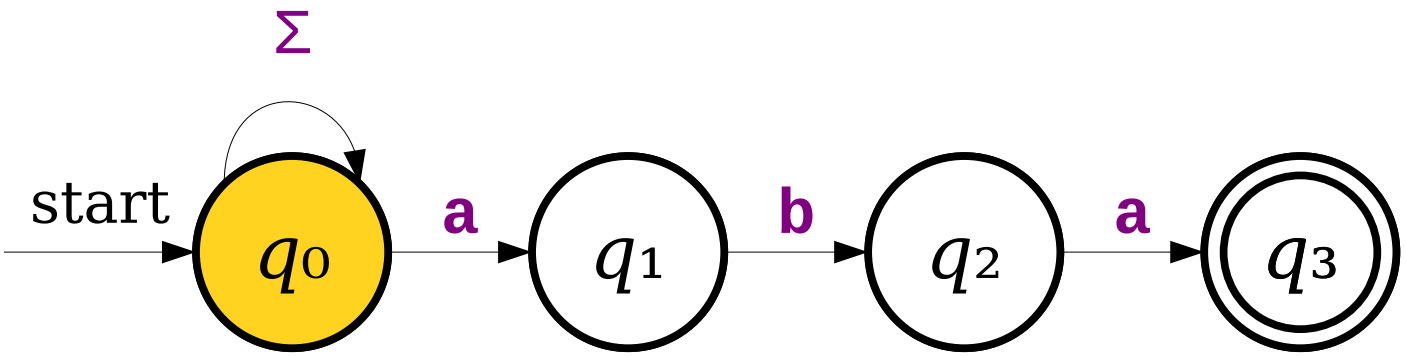
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	



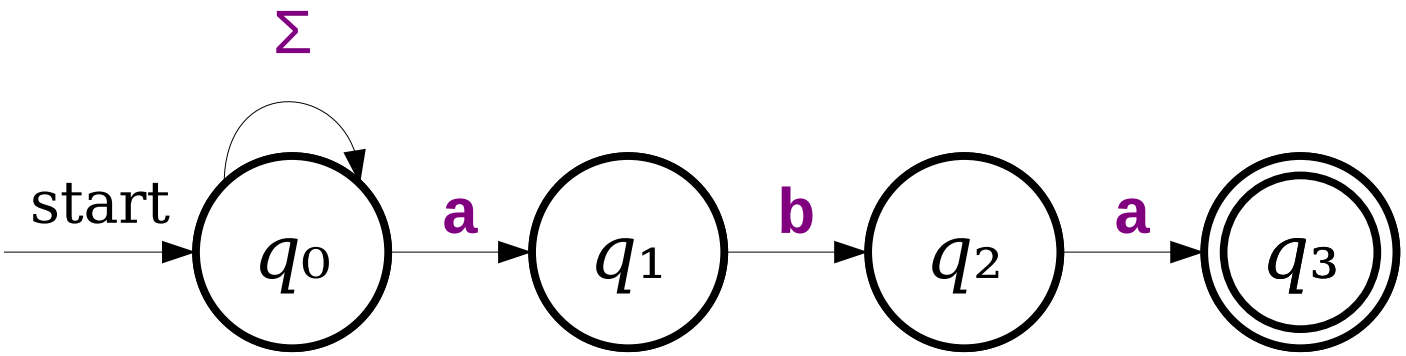
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	



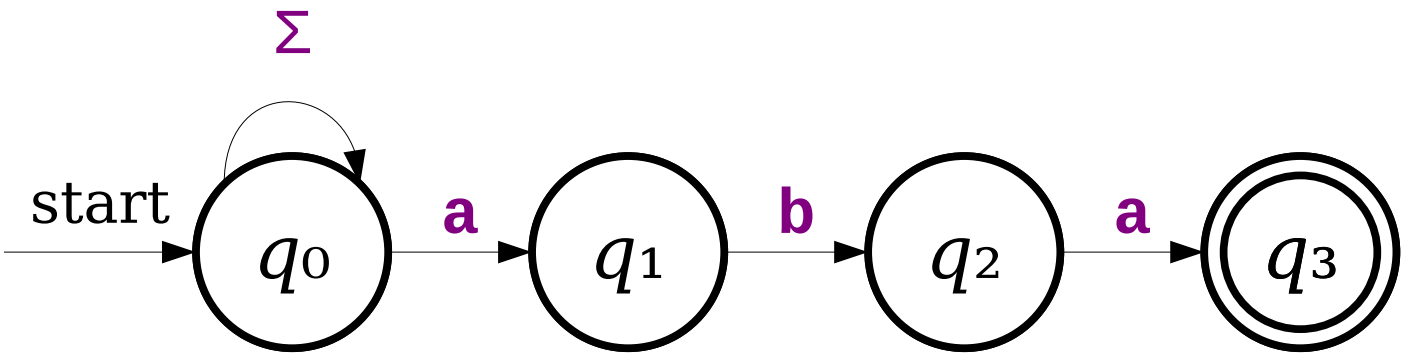
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$



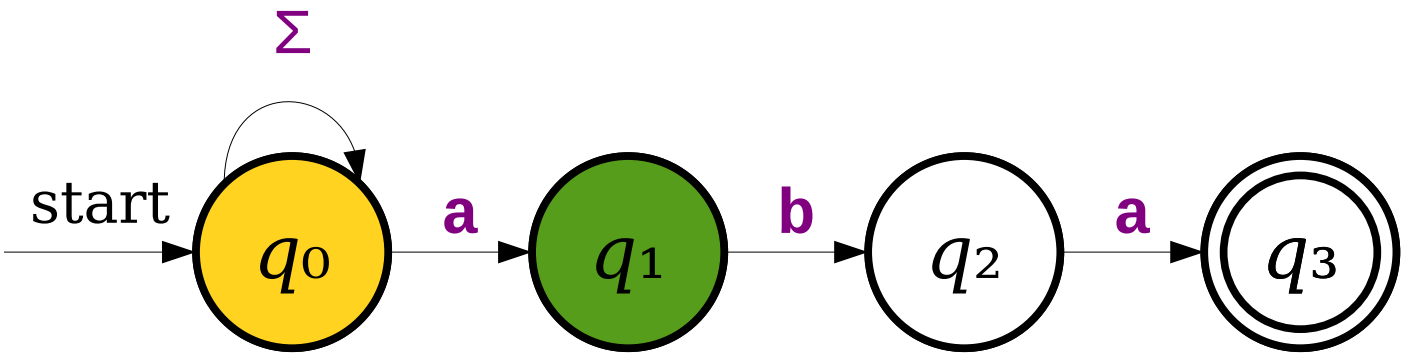
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$



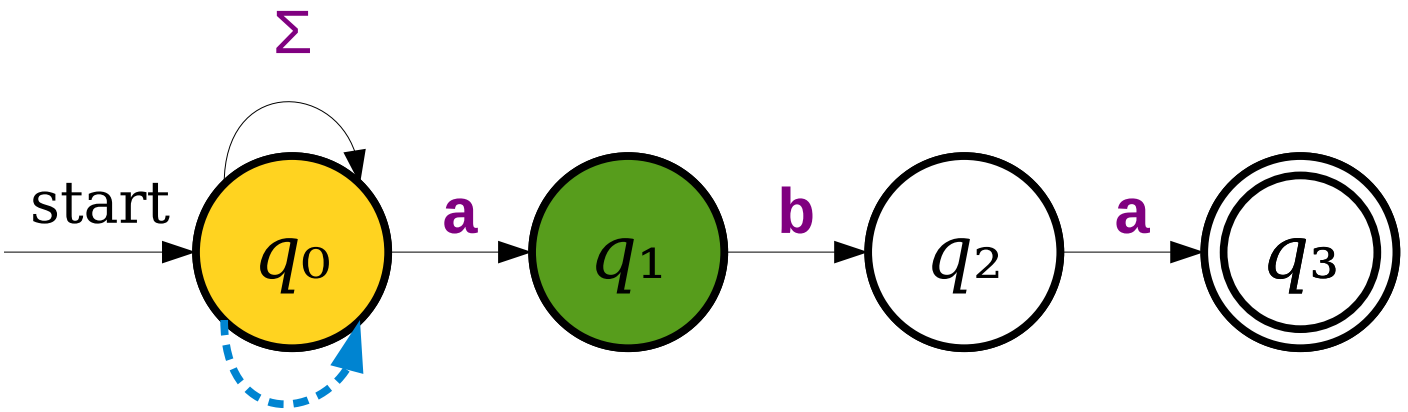
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$



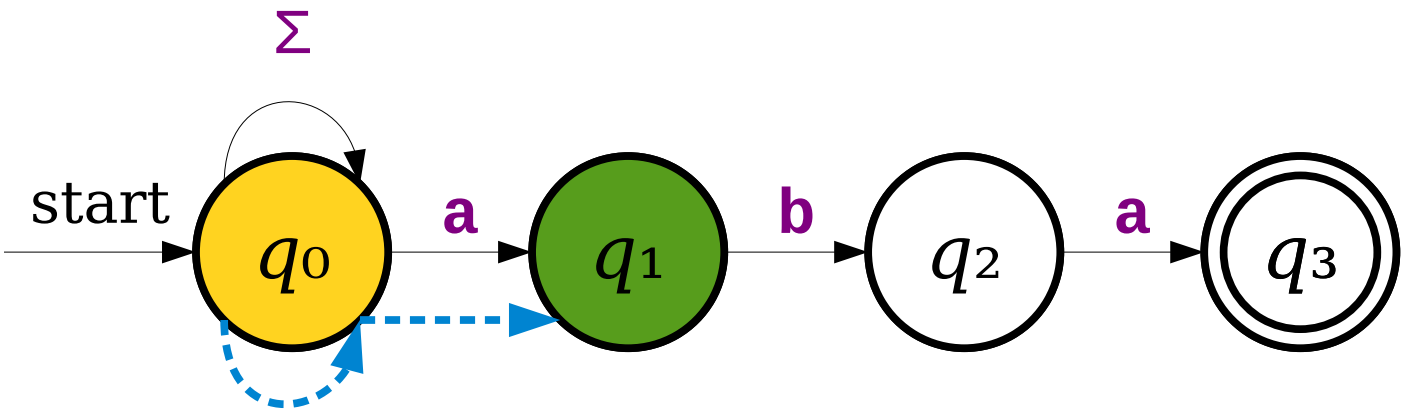
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



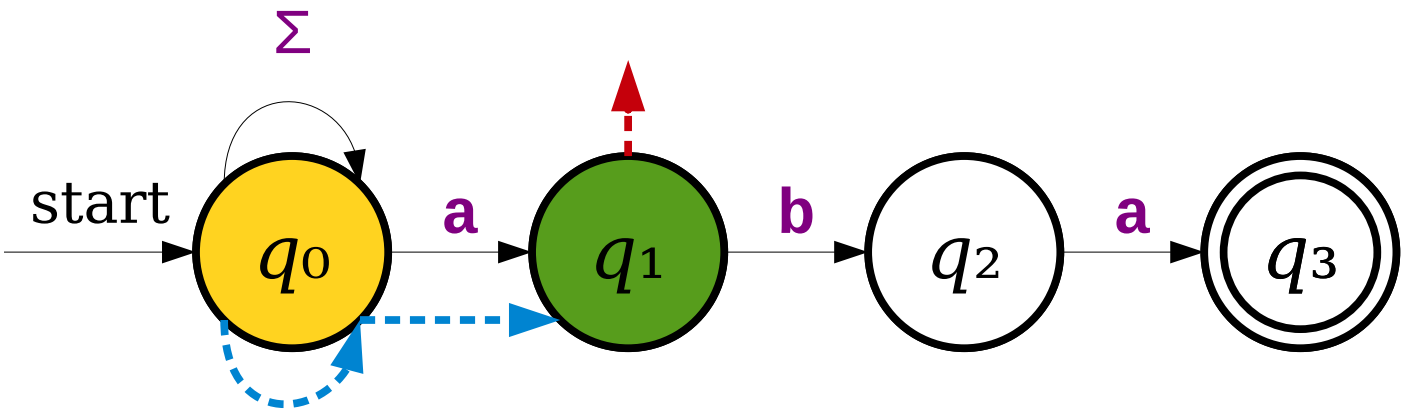
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



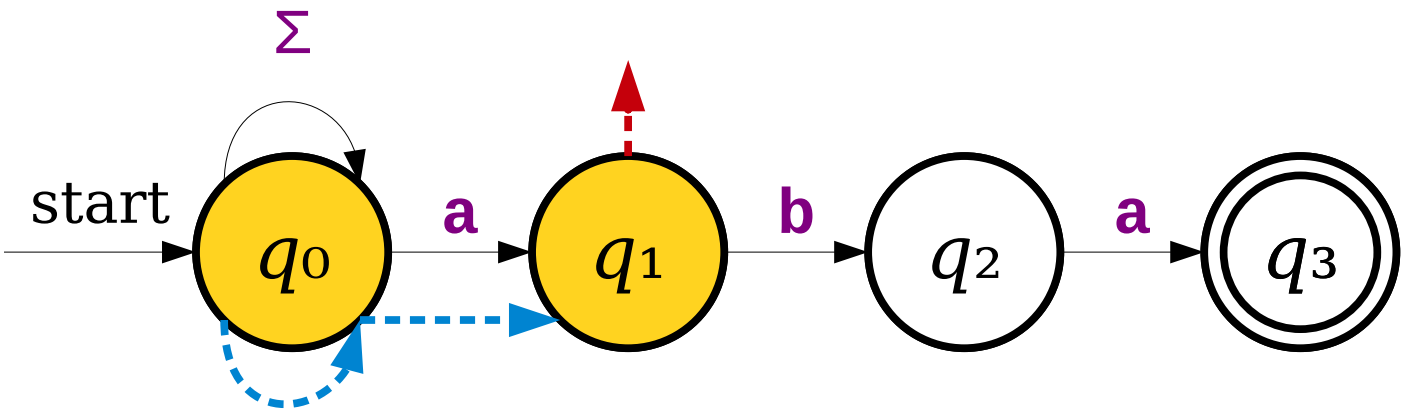
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



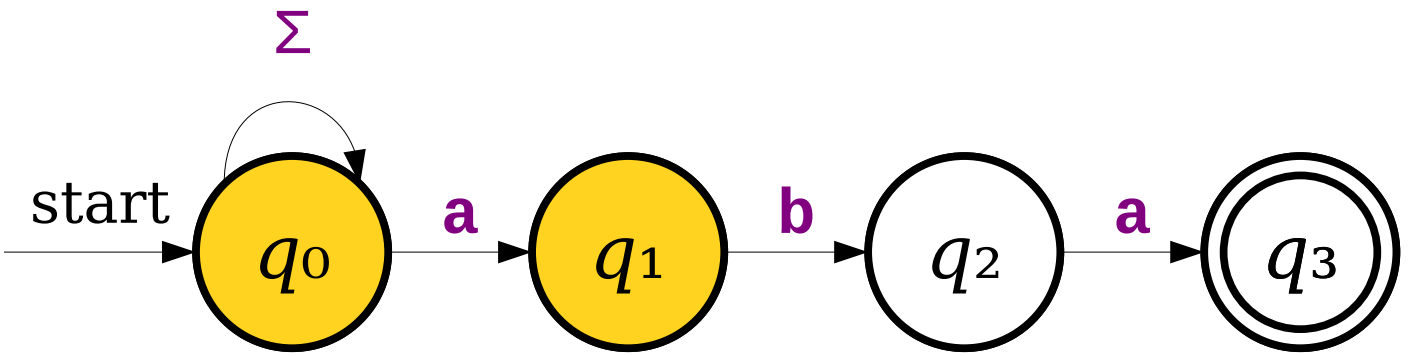
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



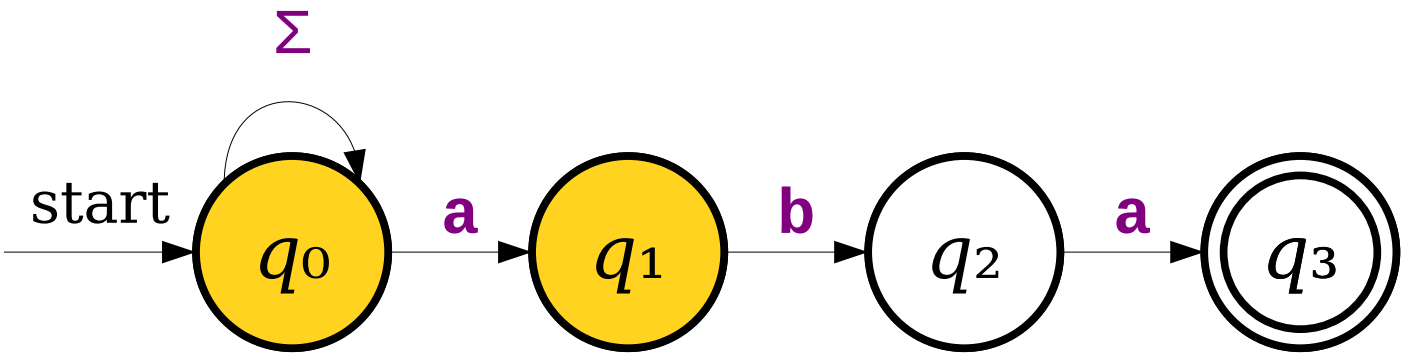
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



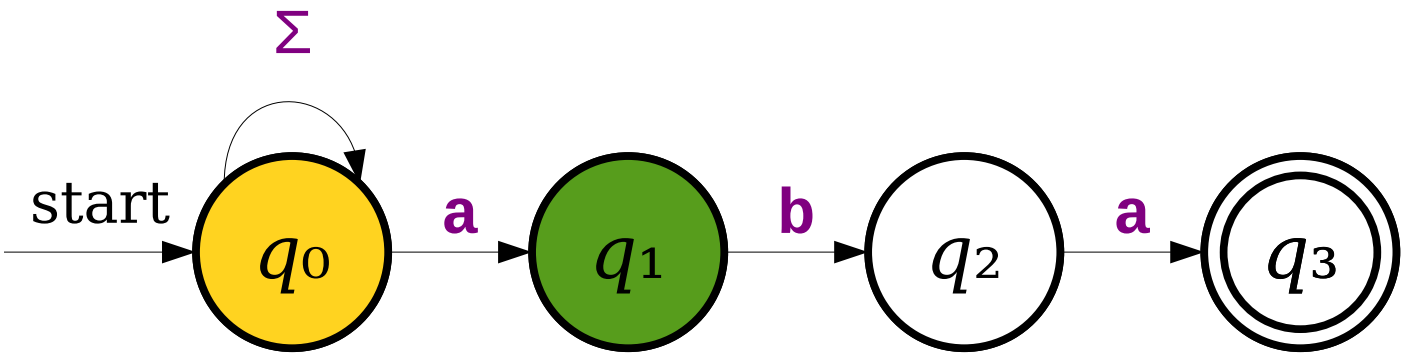
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



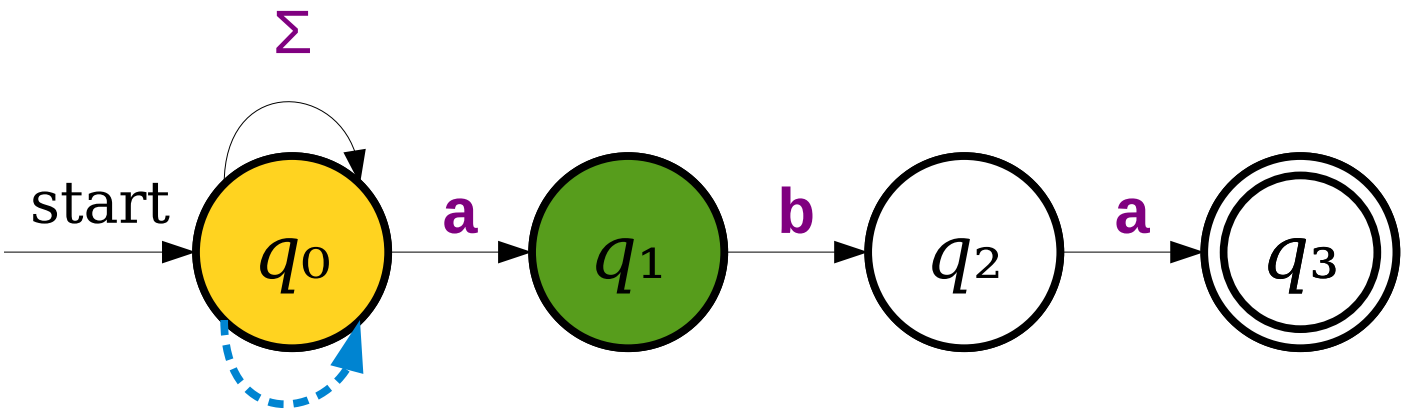
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



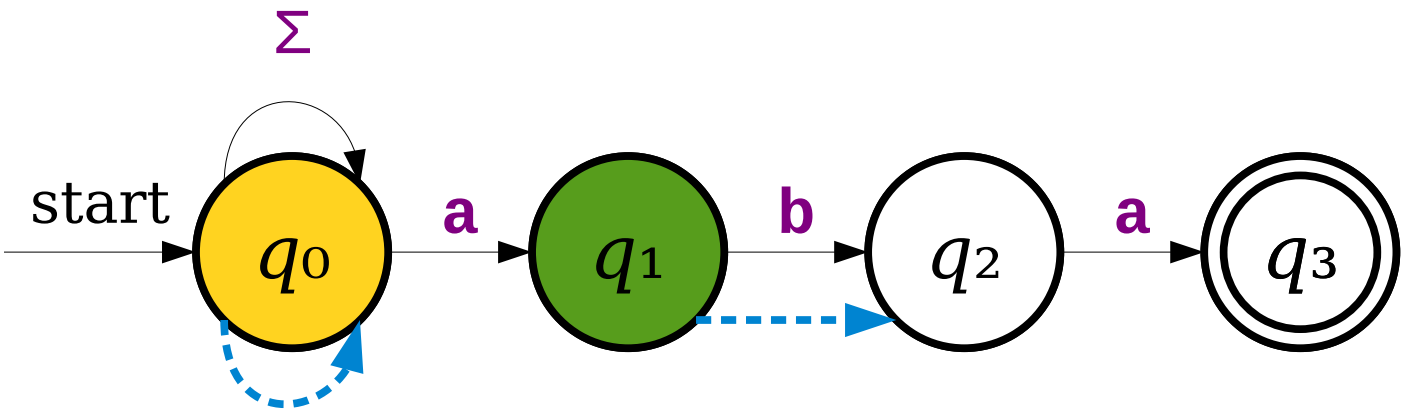
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	



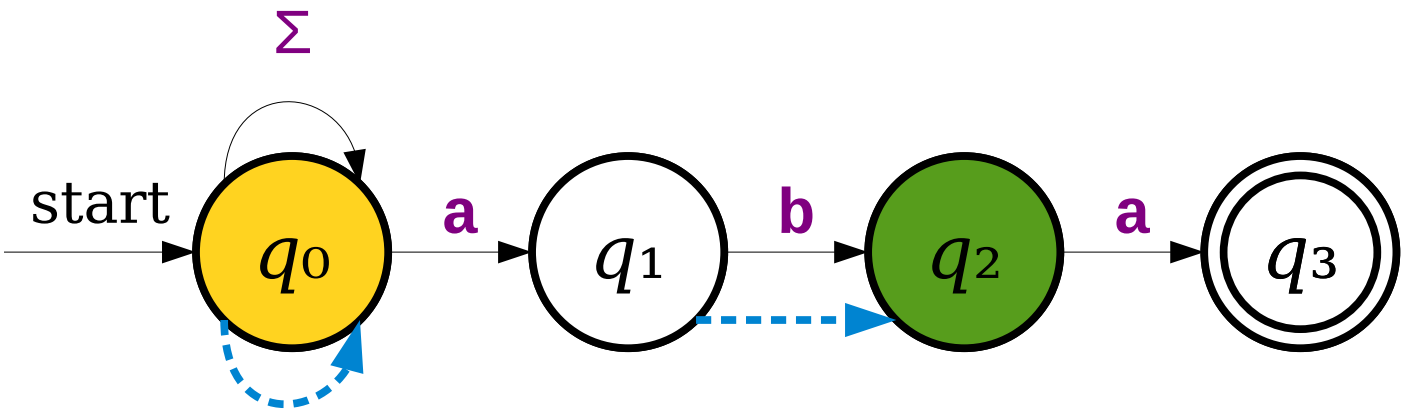
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	



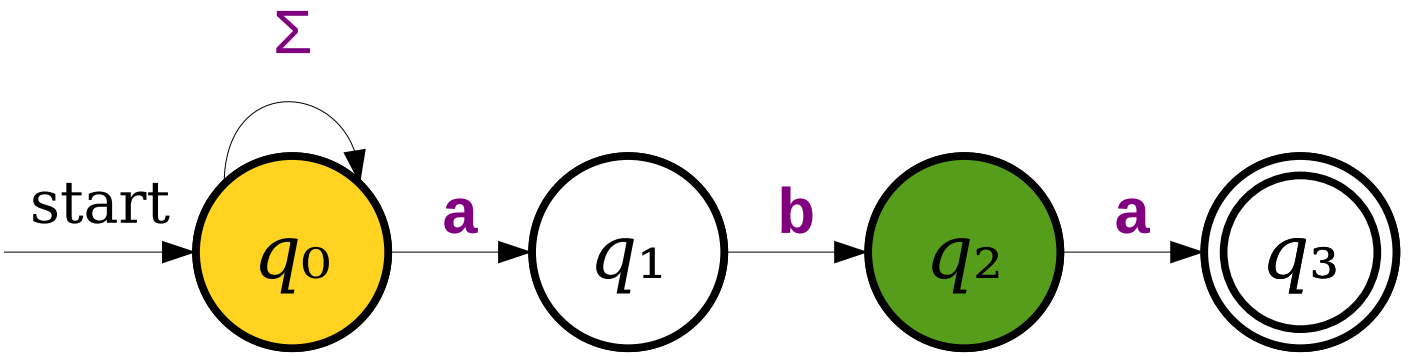
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	



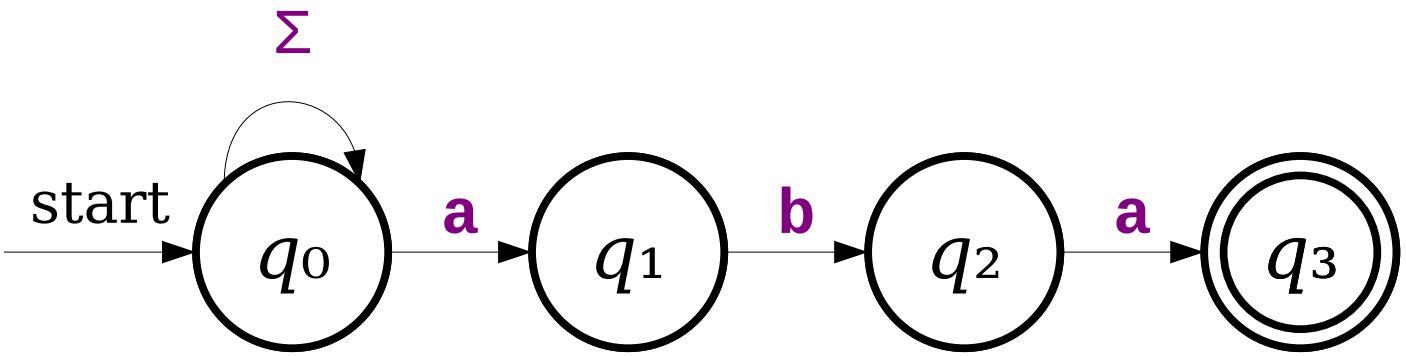
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	



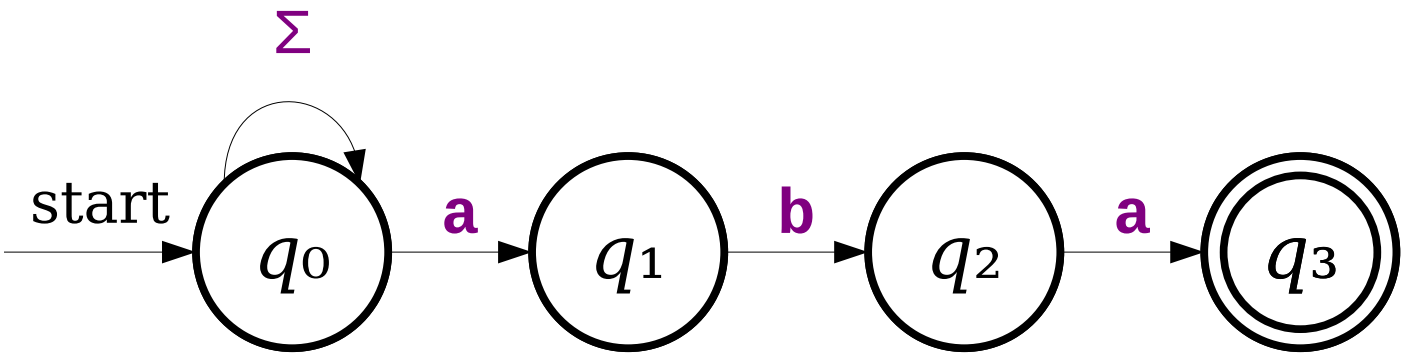
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	



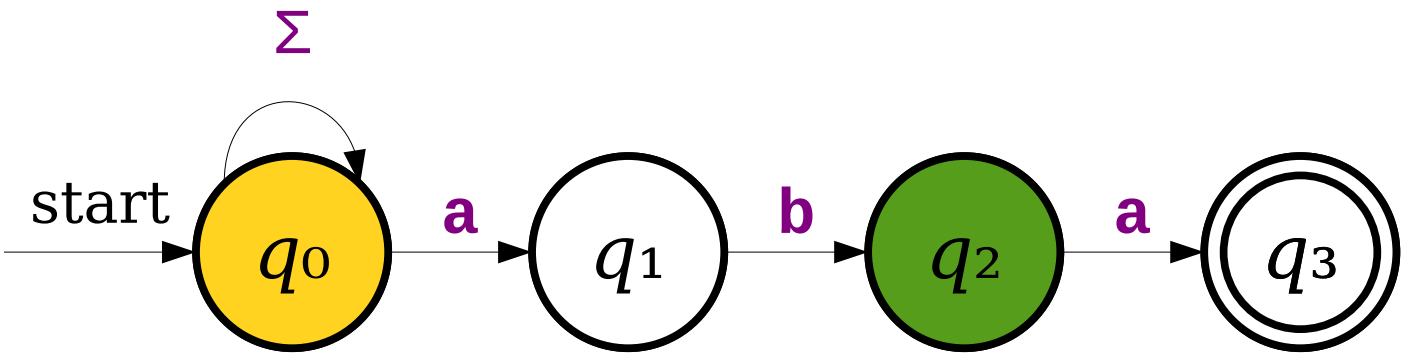
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



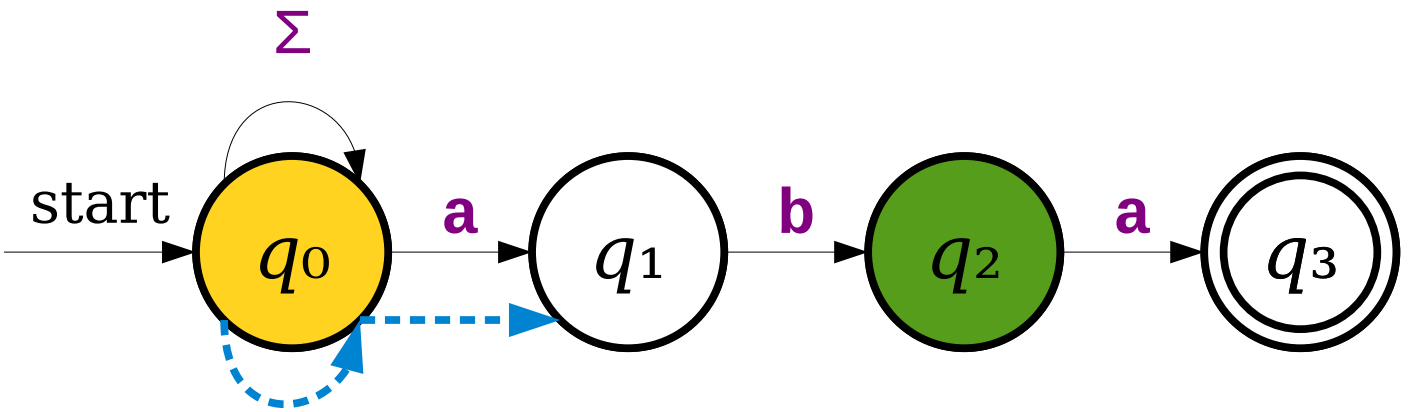
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



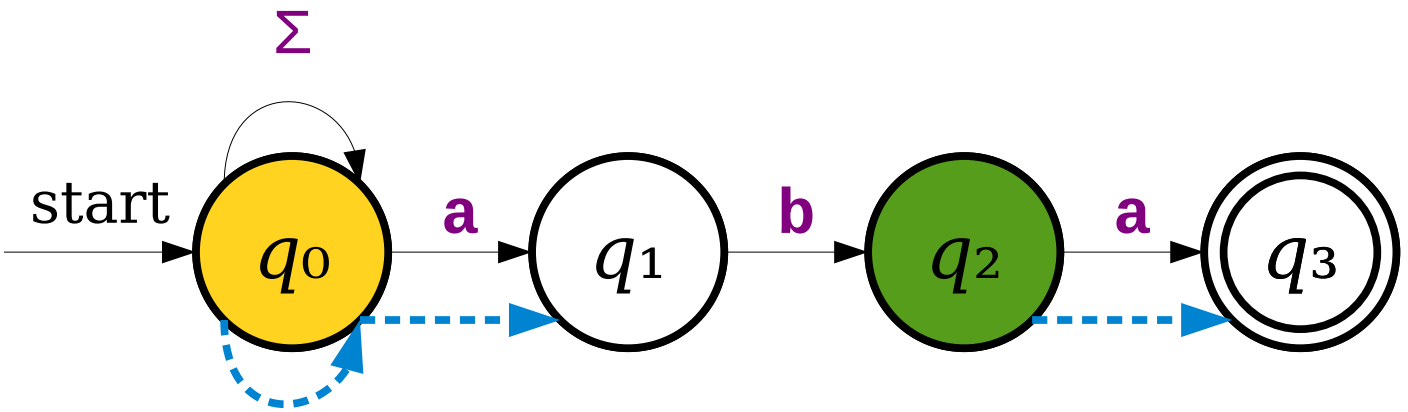
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		



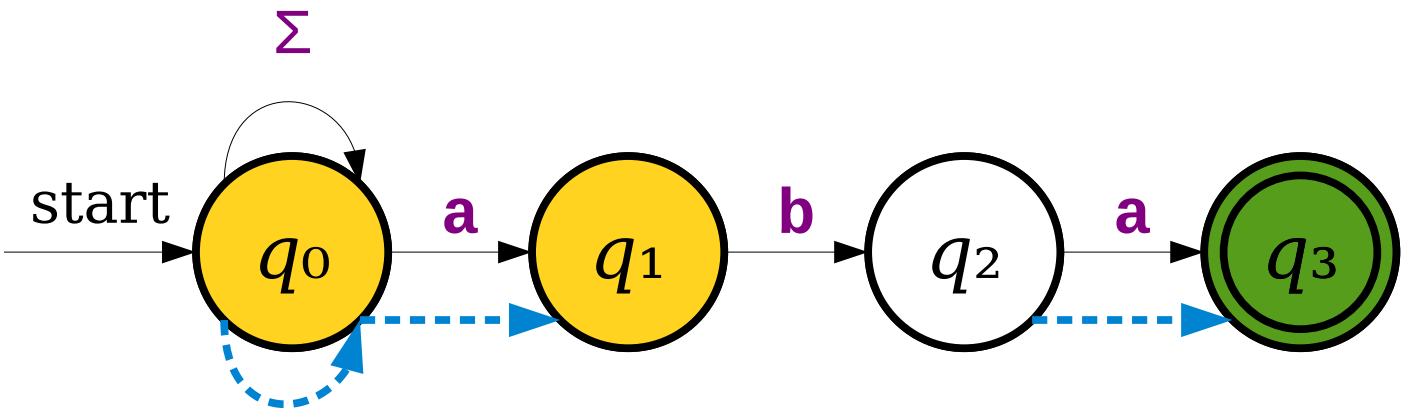
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		



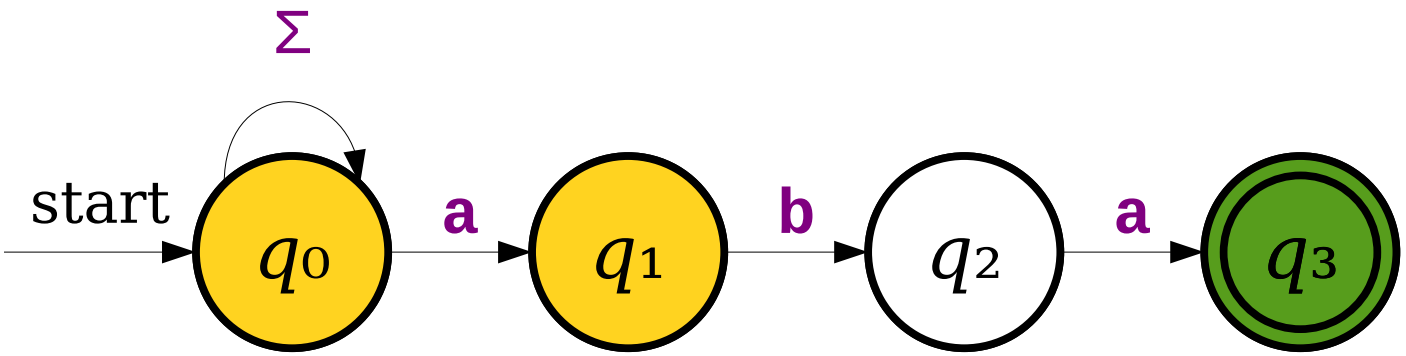
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		



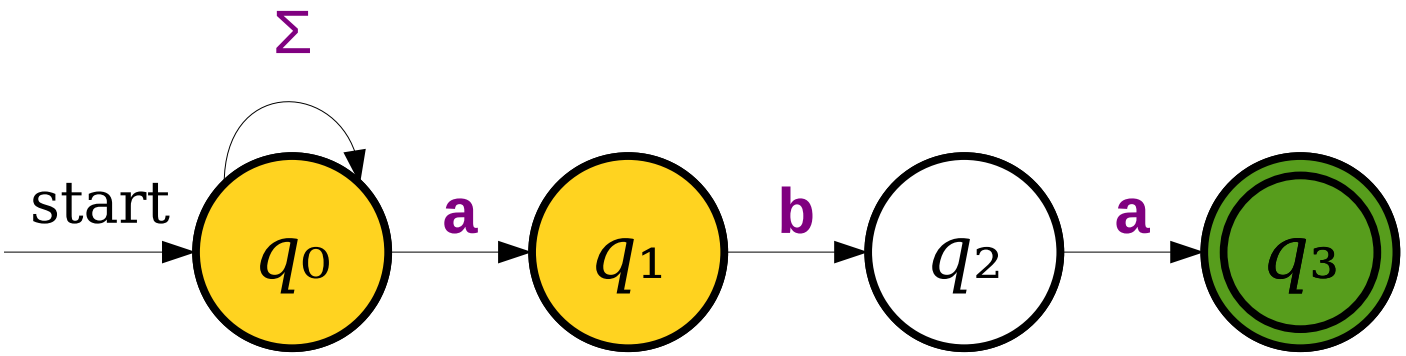
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		



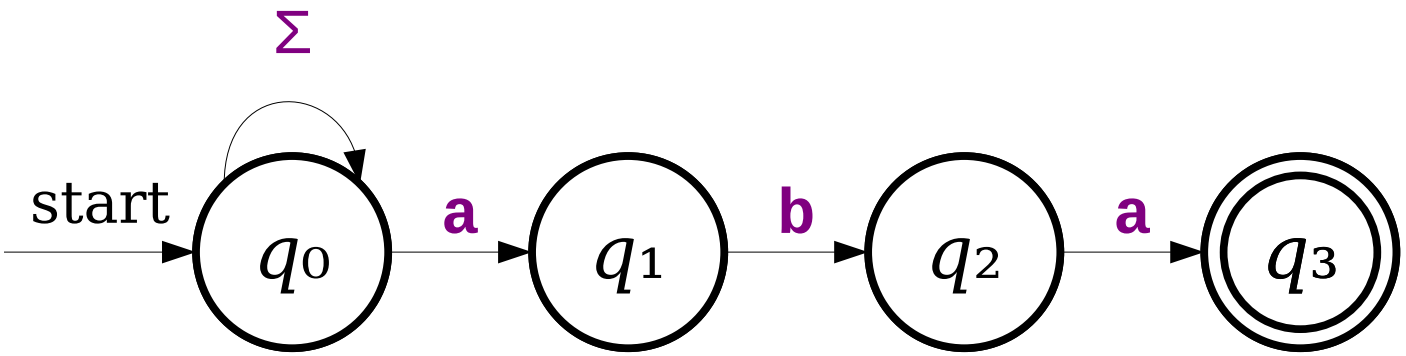
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		



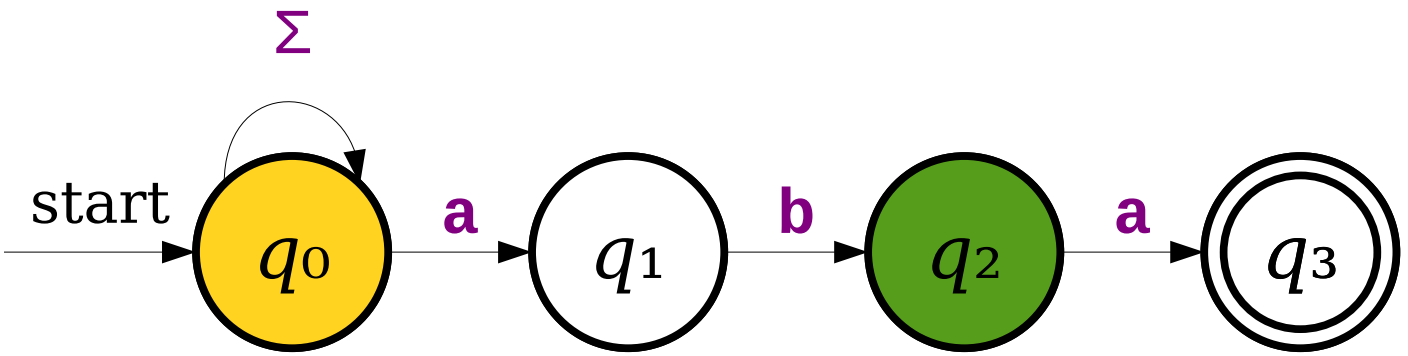
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		



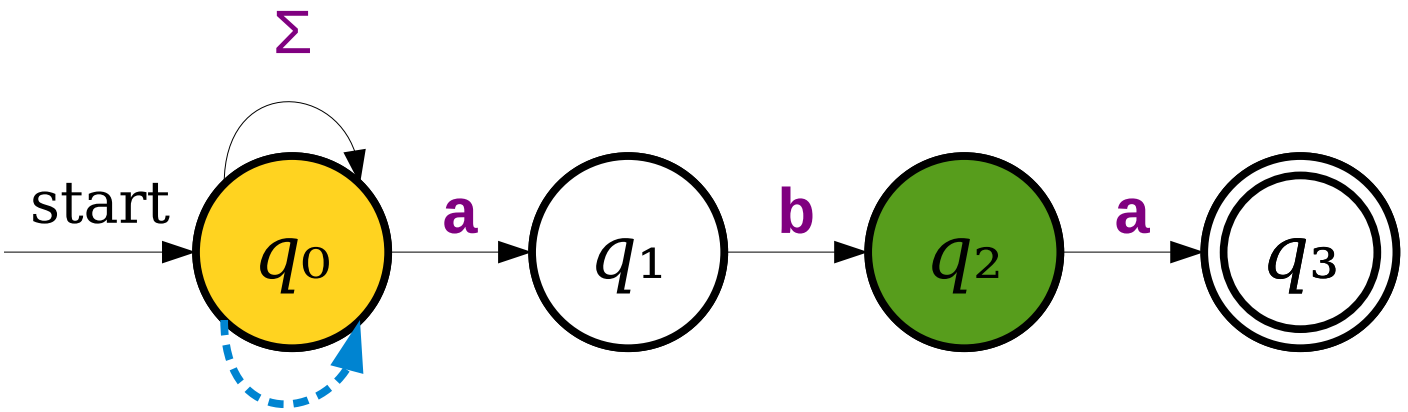
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



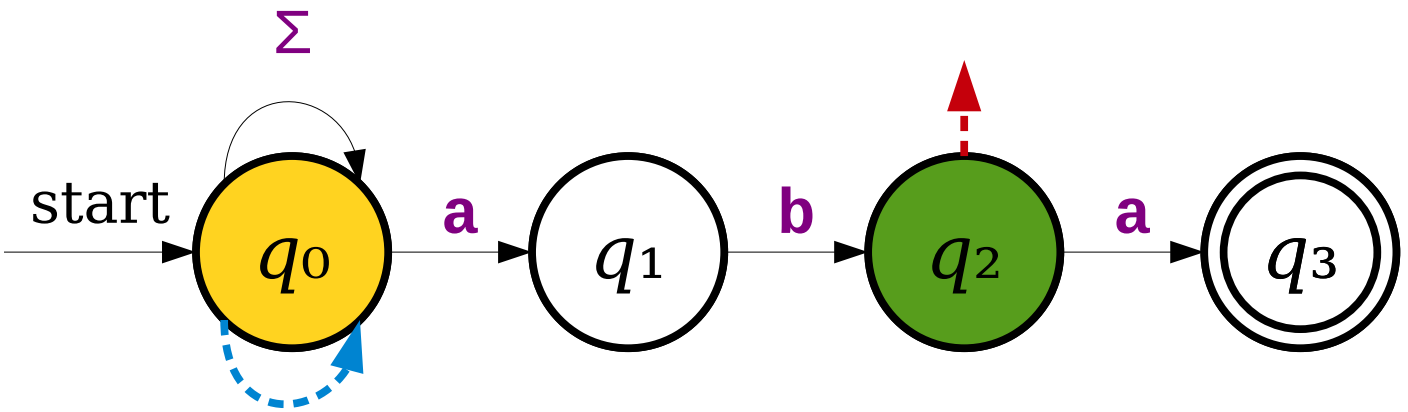
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



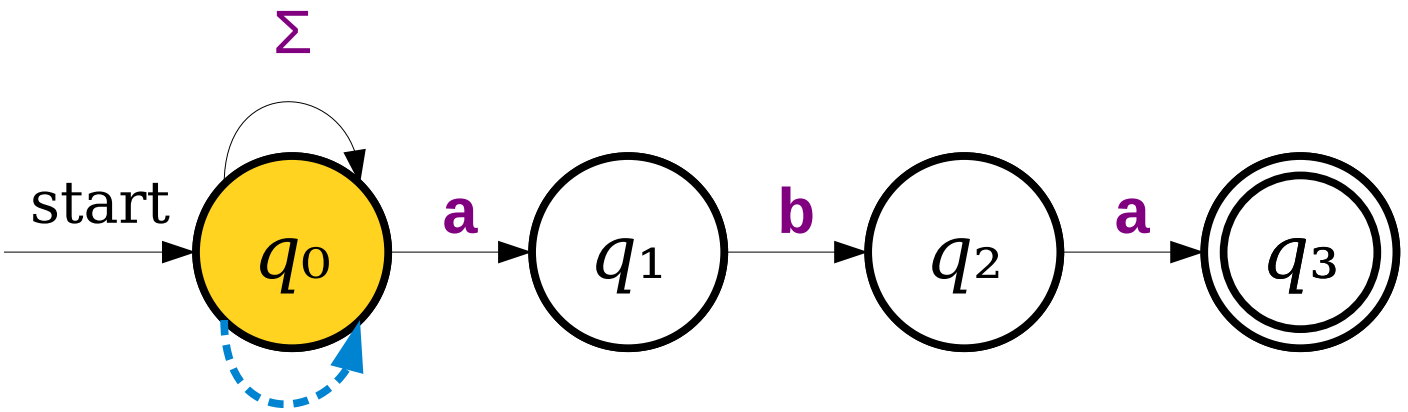
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



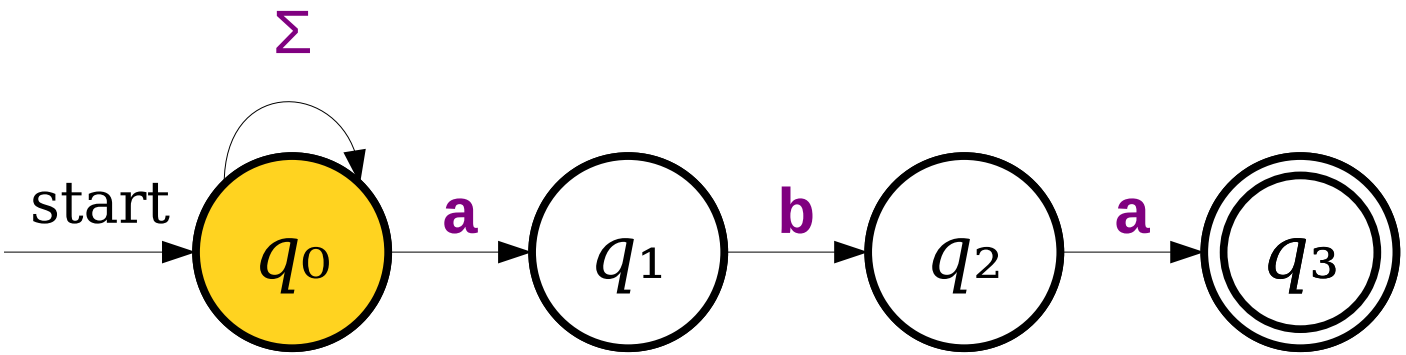
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



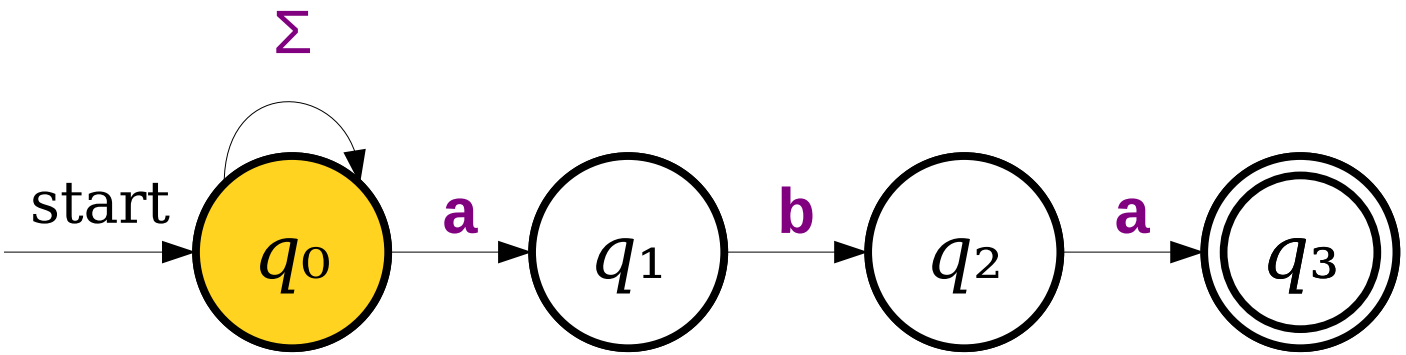
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



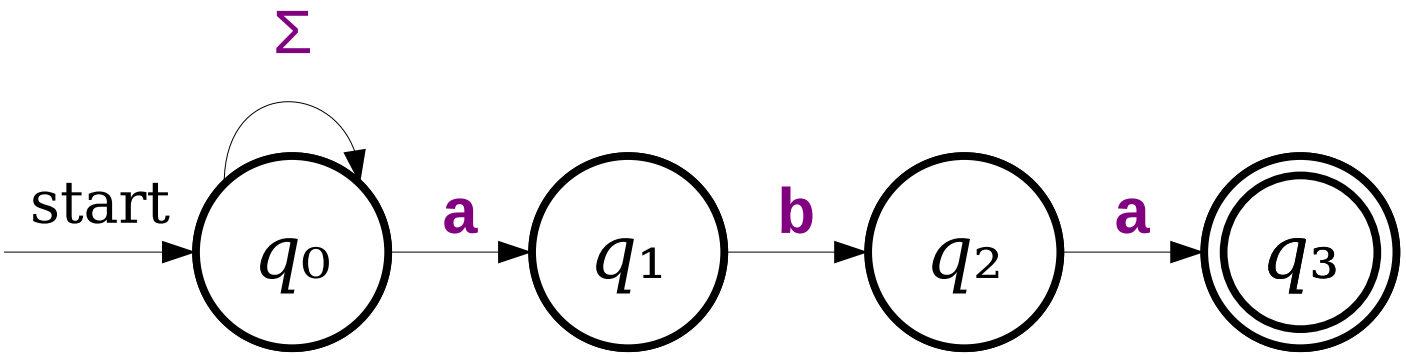
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



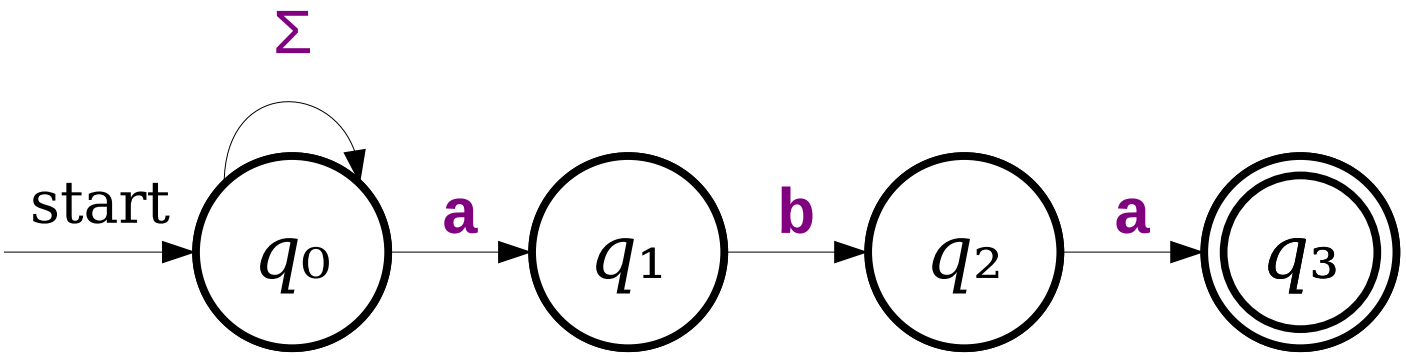
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



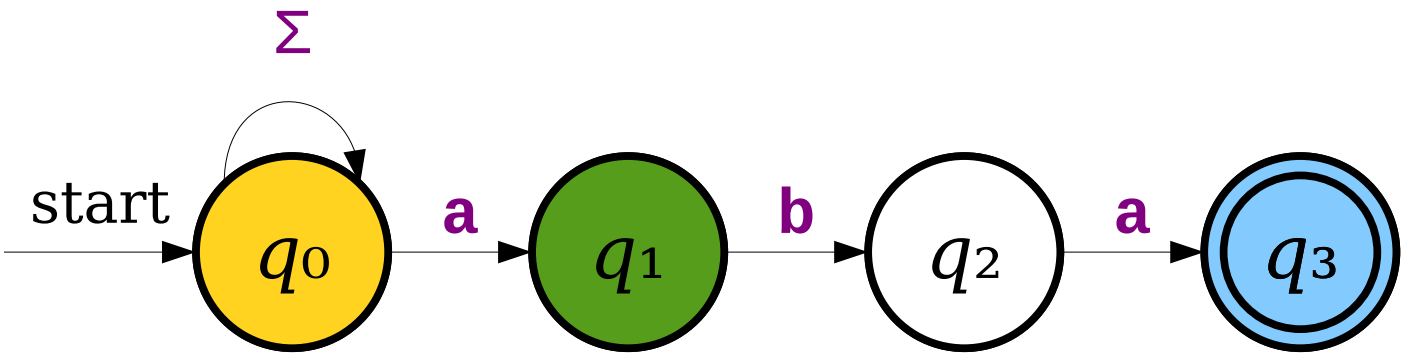
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$



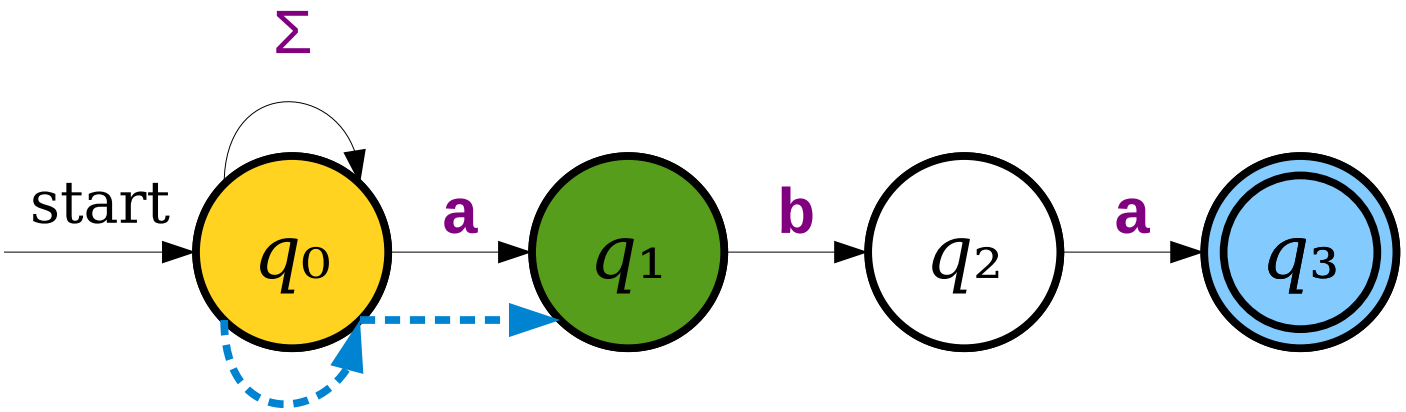
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$



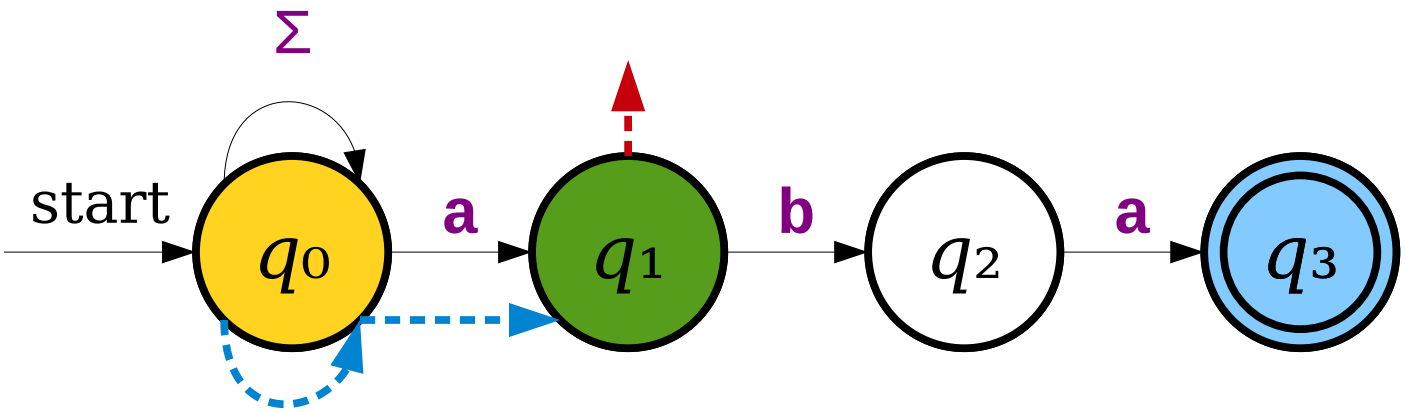
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



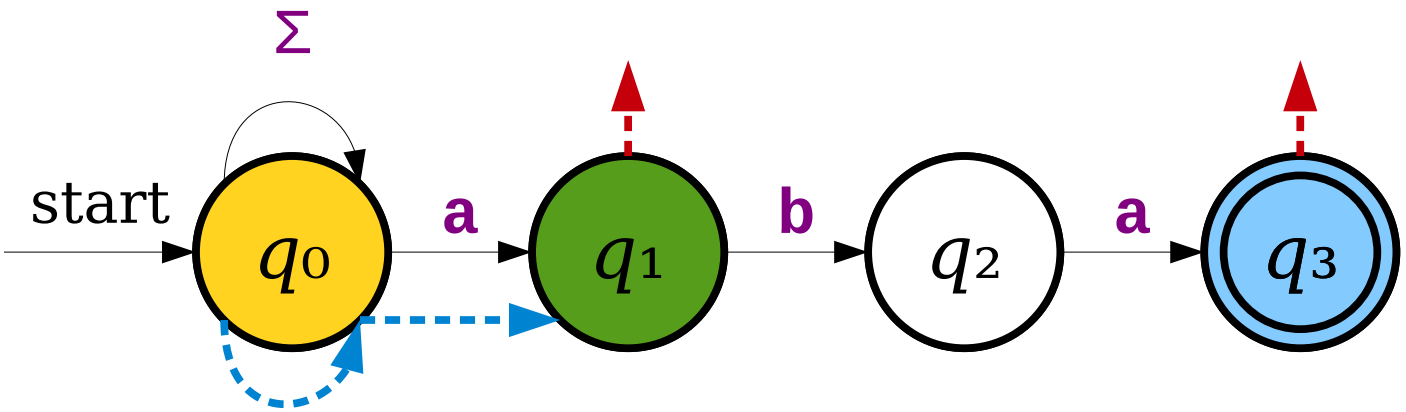
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



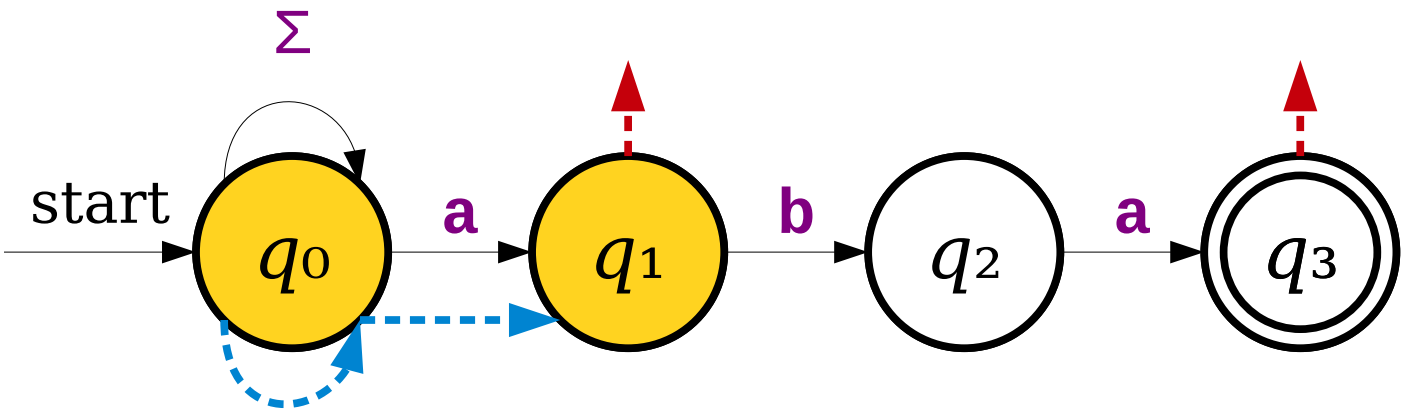
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



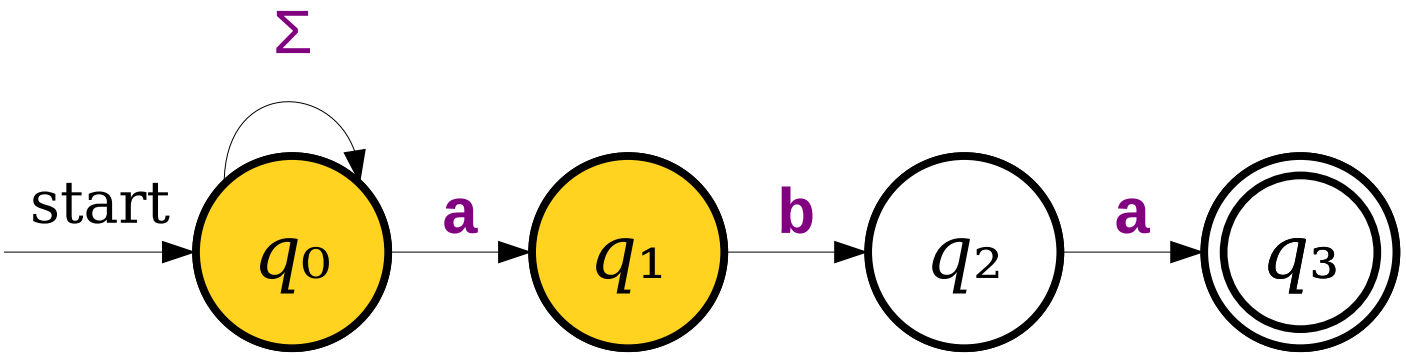
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



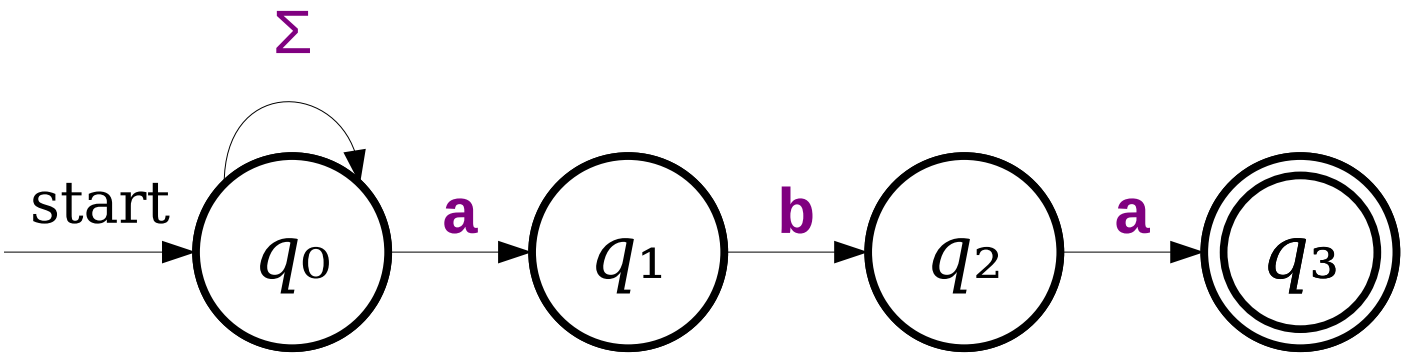
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



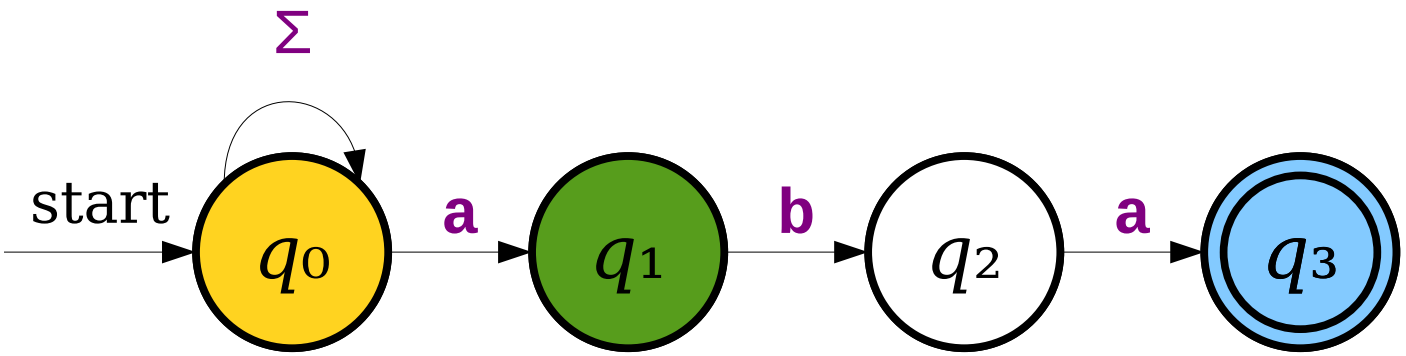
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



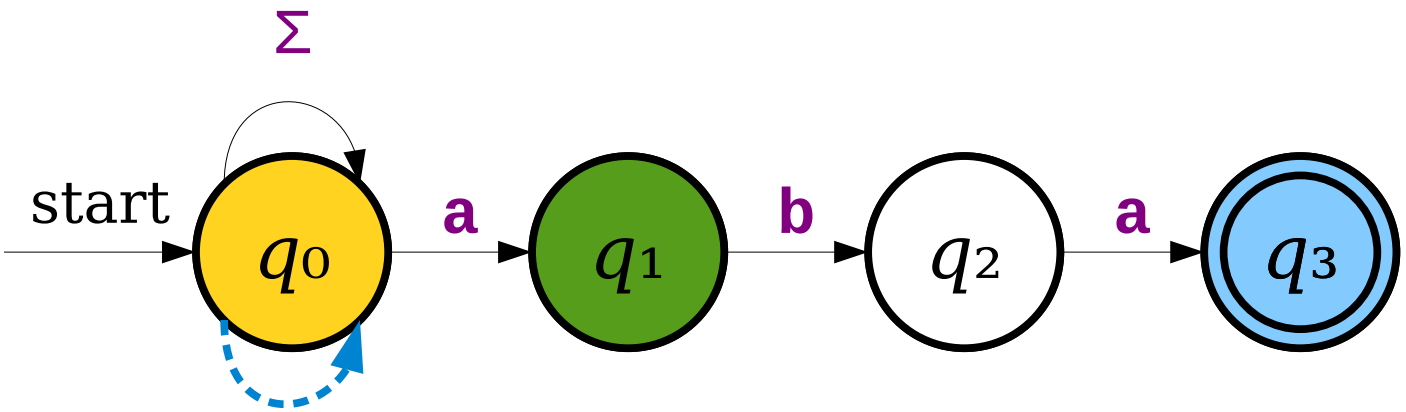
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



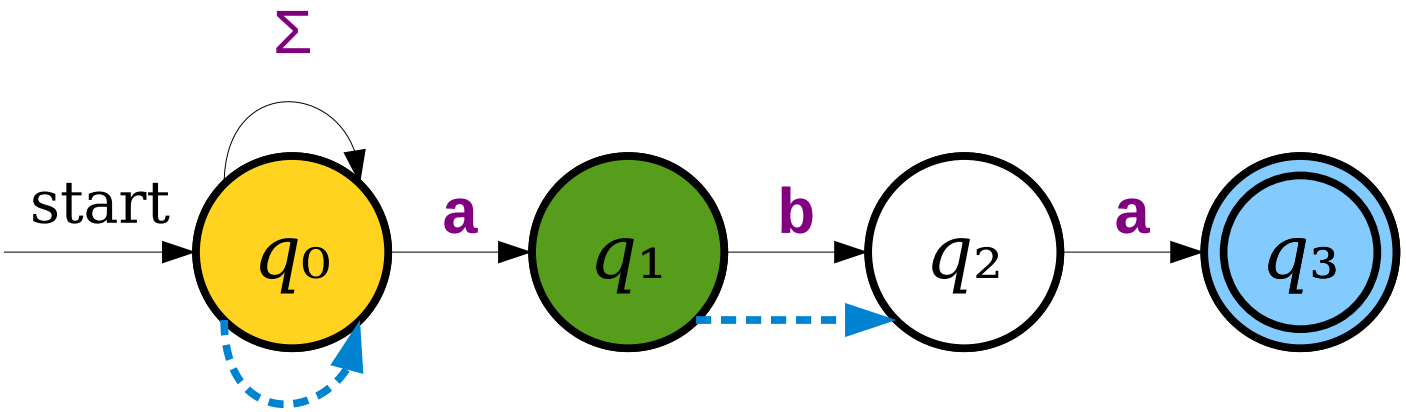
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



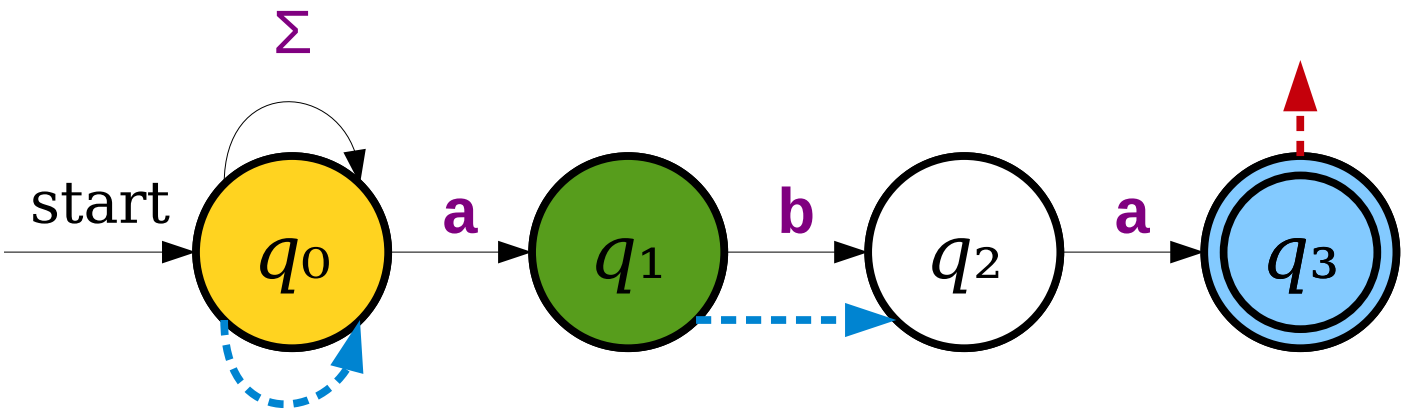
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



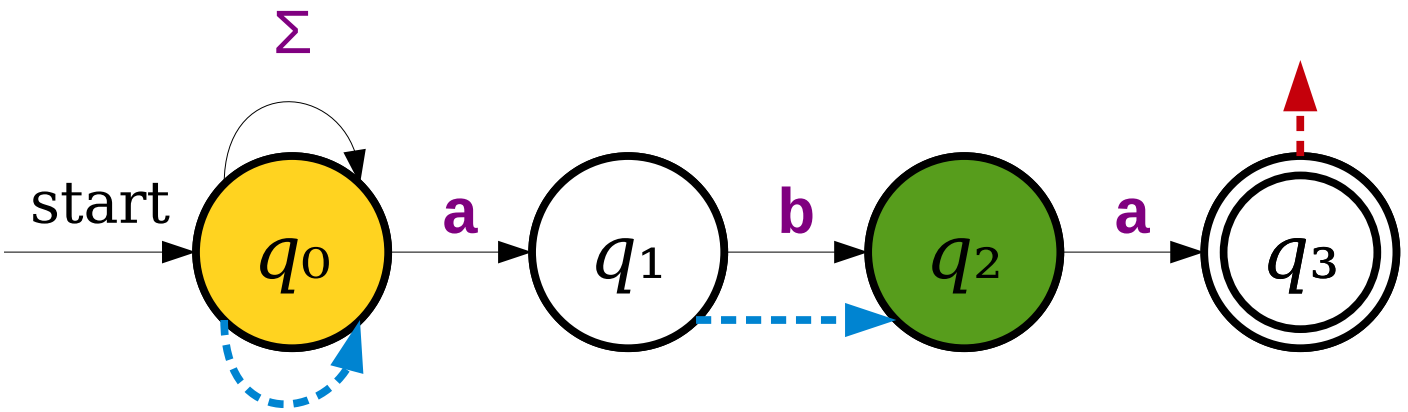
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



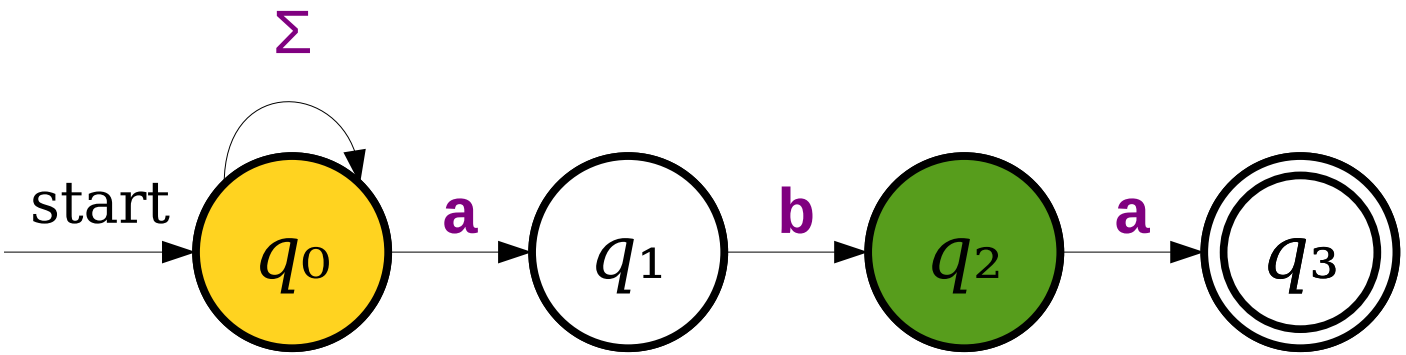
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



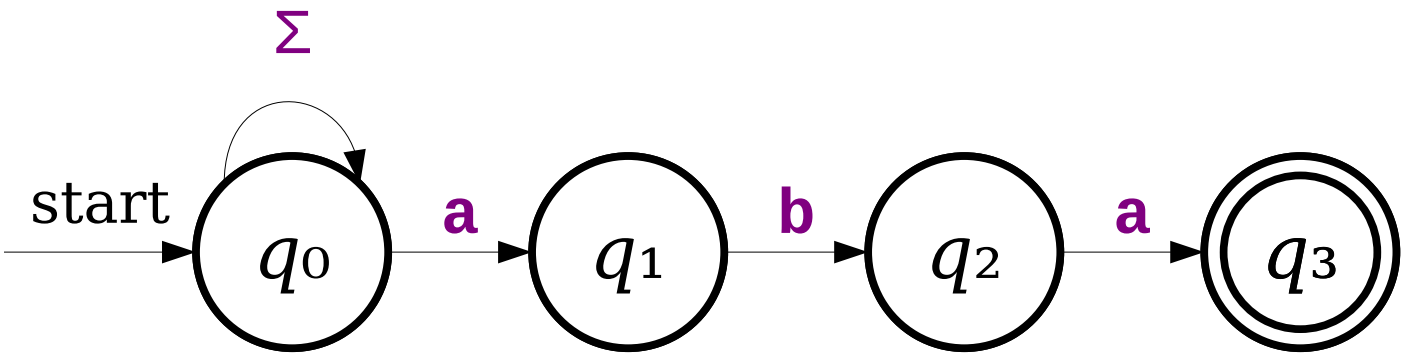
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



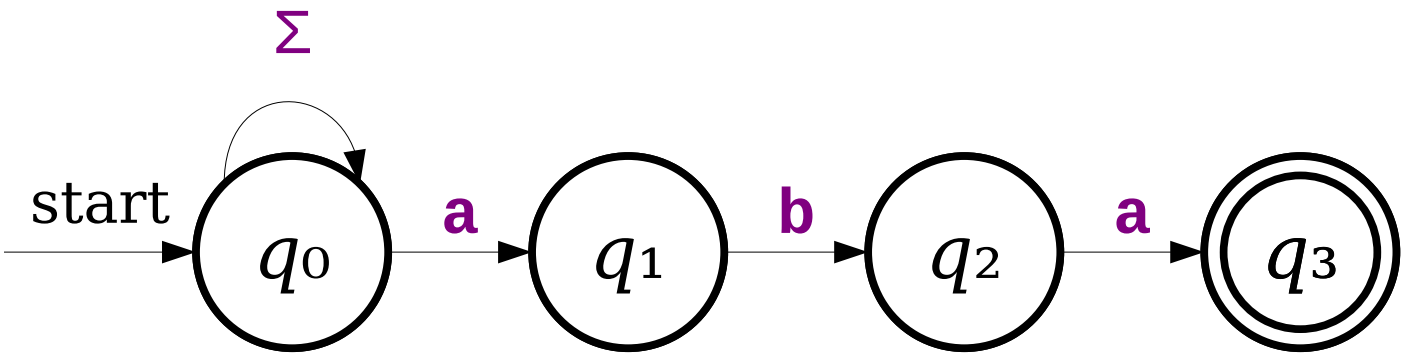
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



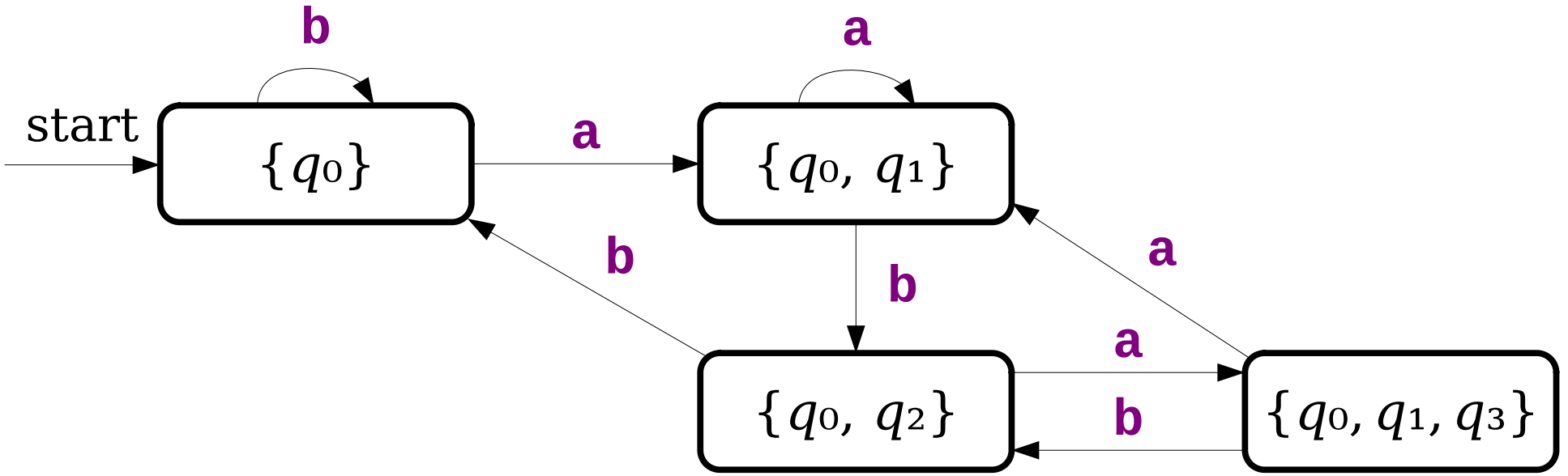
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

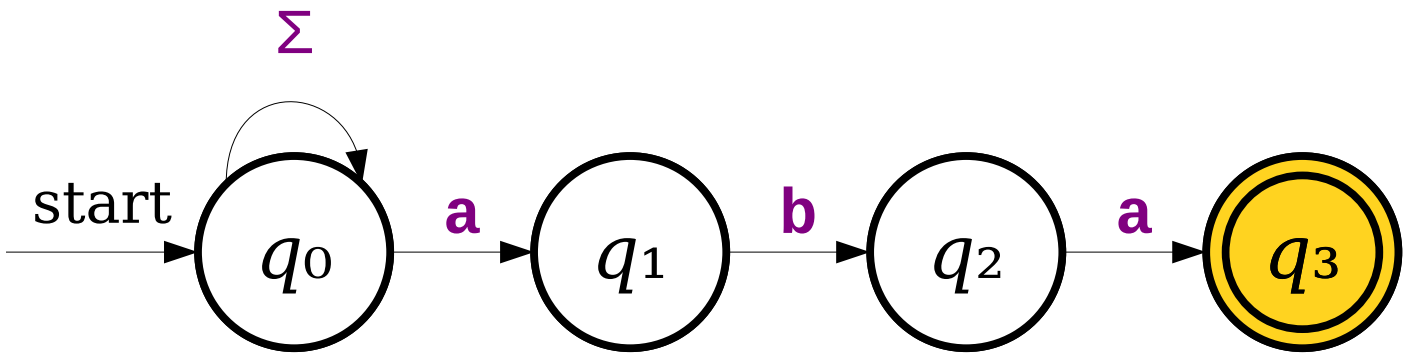


	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

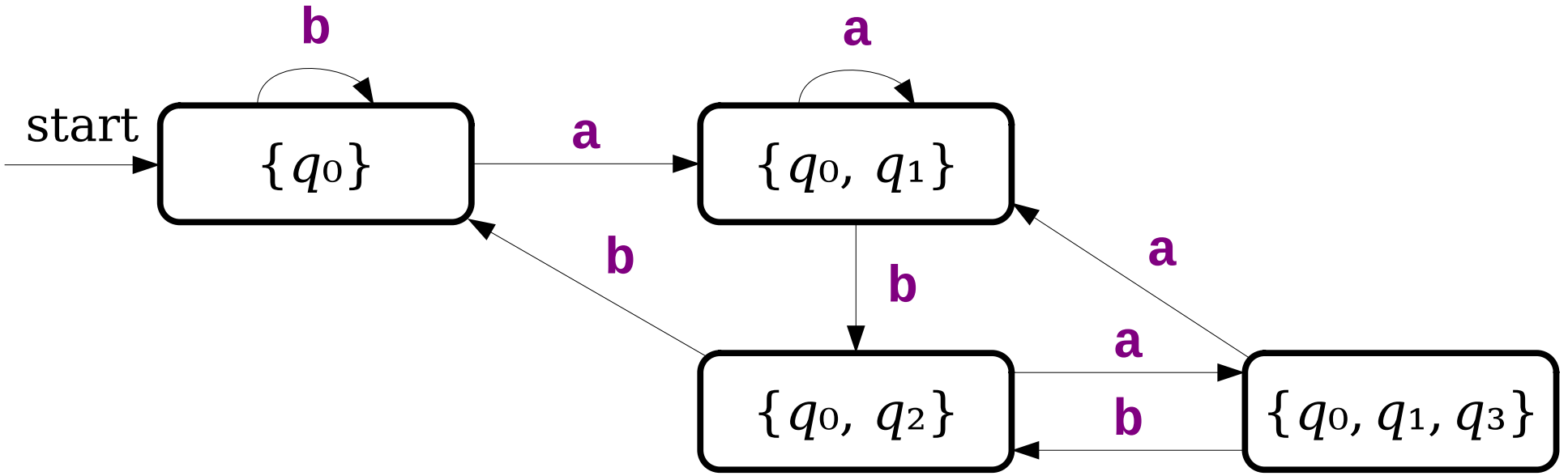


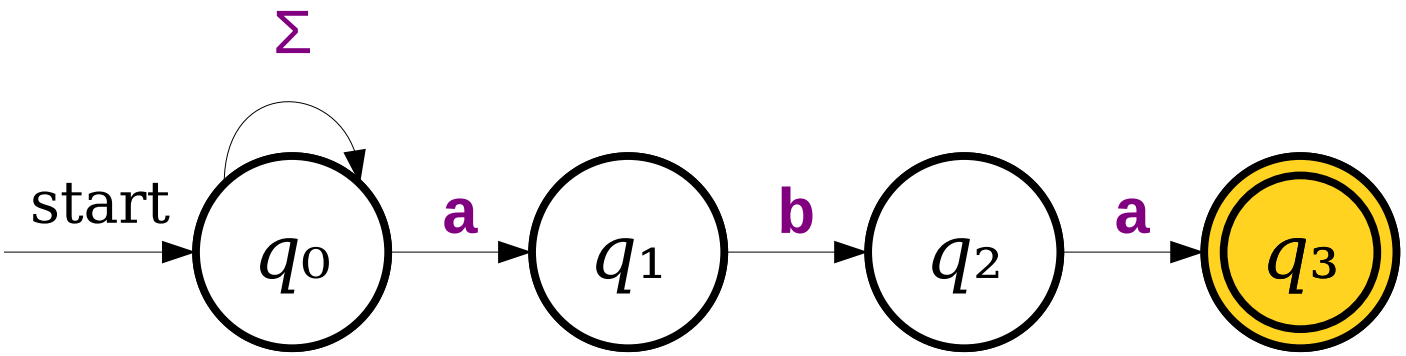
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



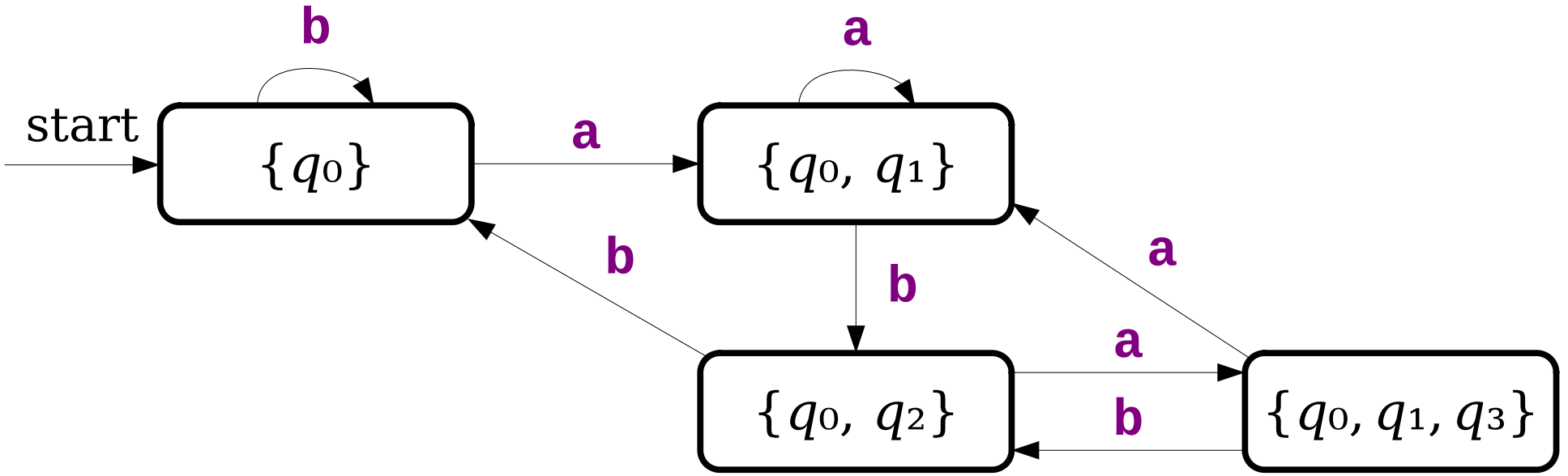


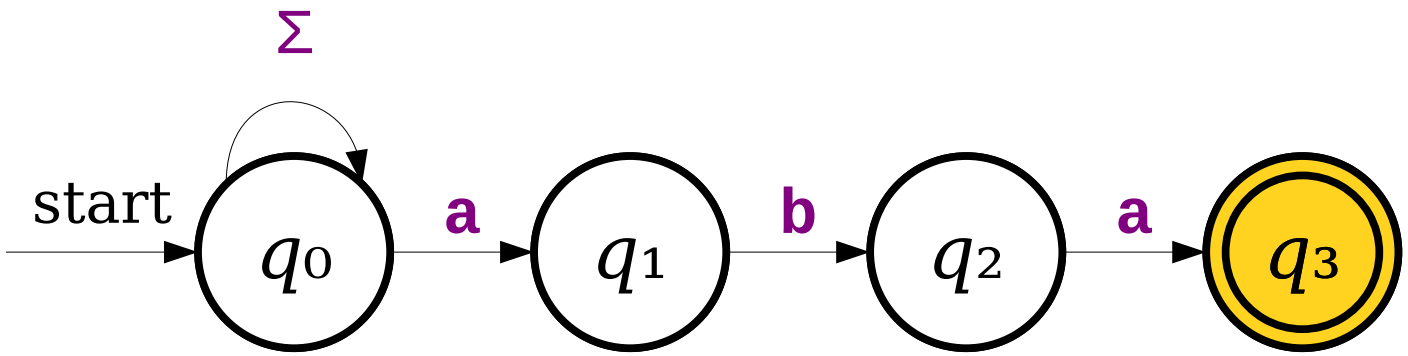
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



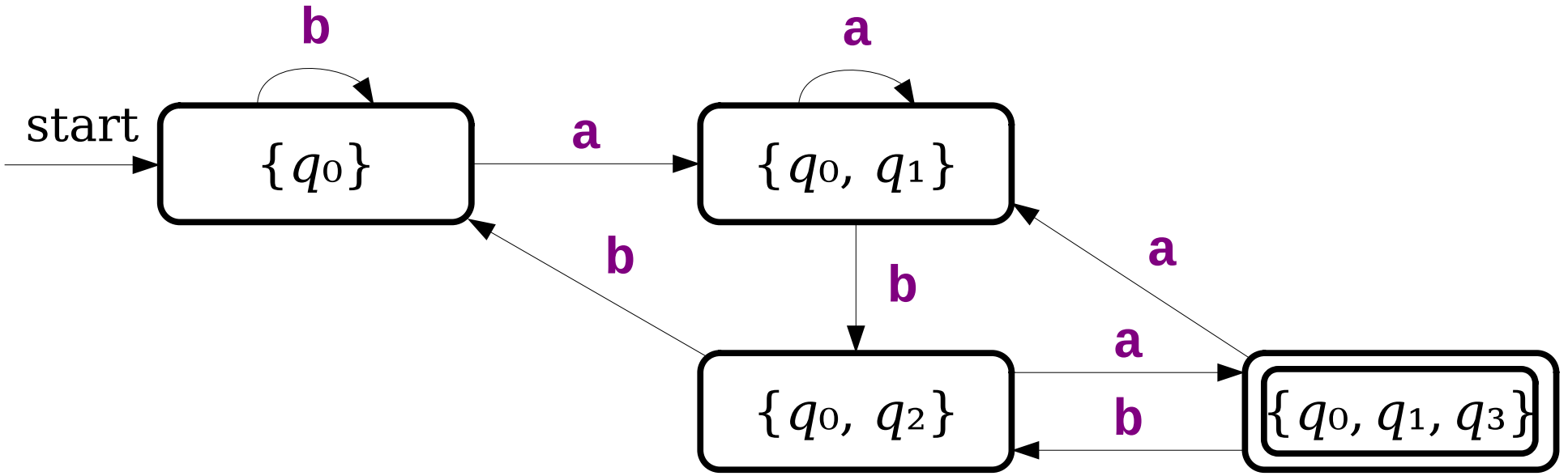


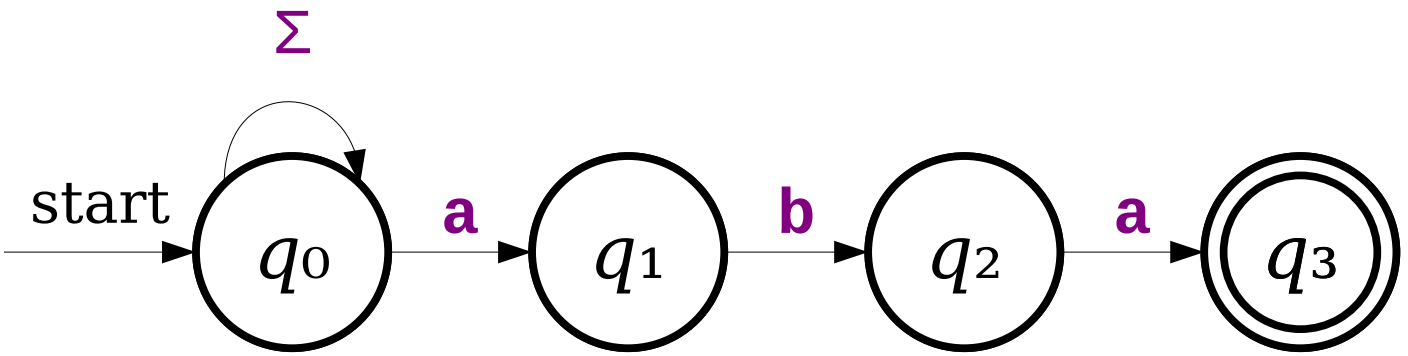
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$*\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



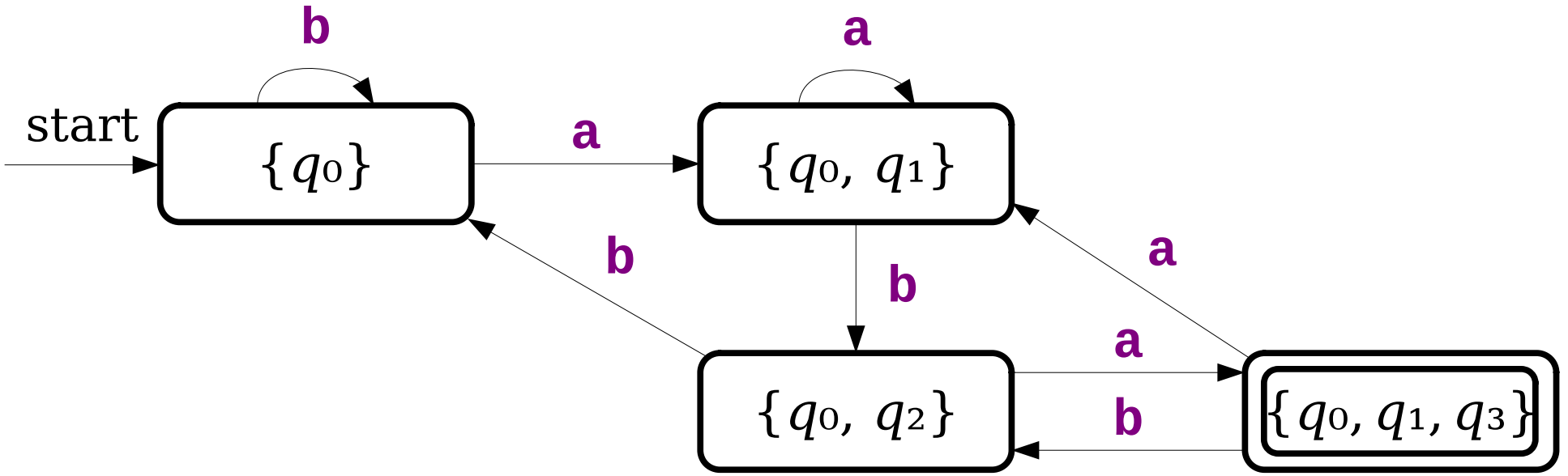


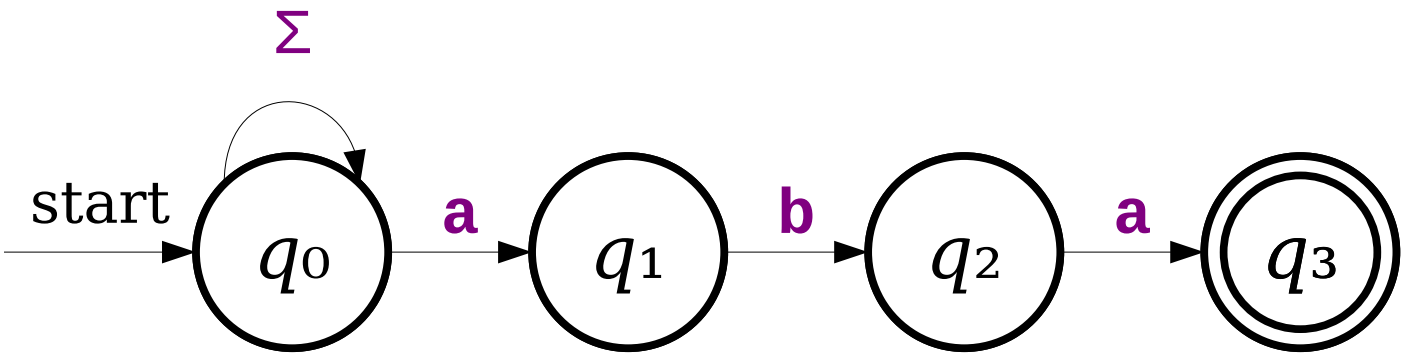
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$*\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



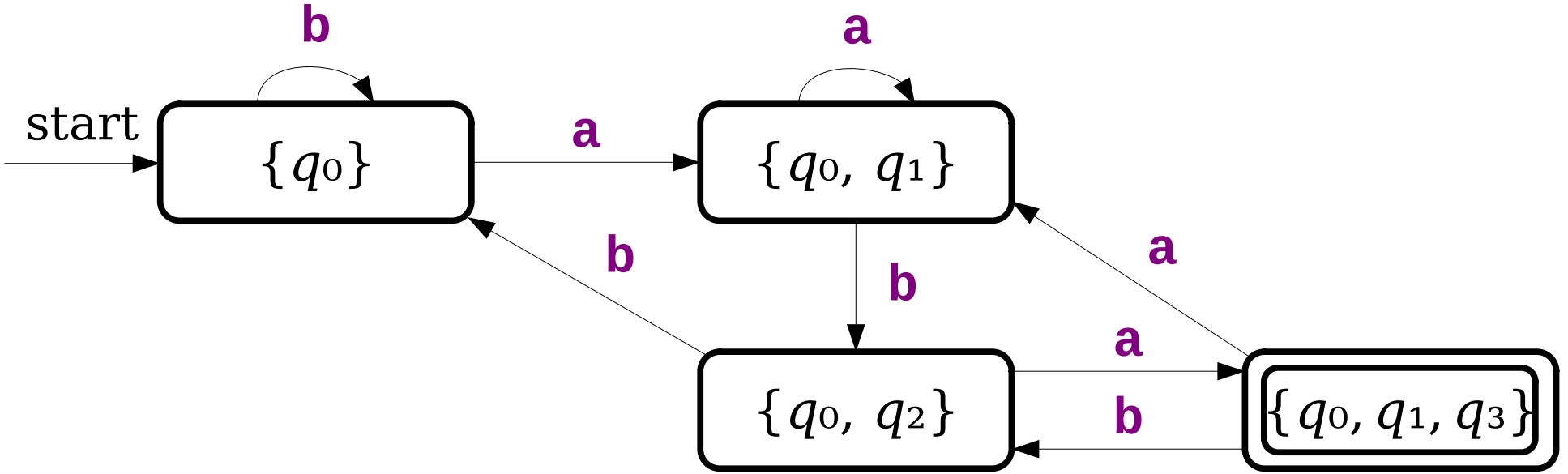


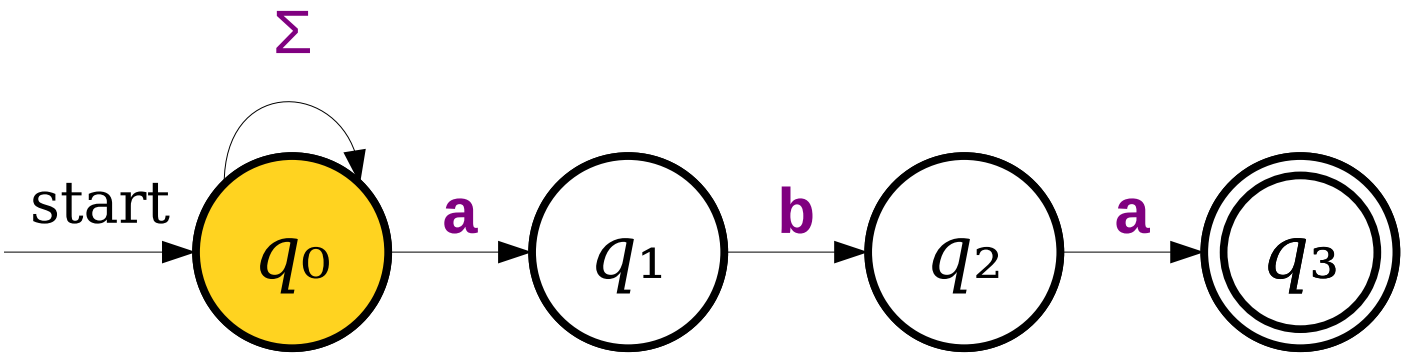
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$*\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



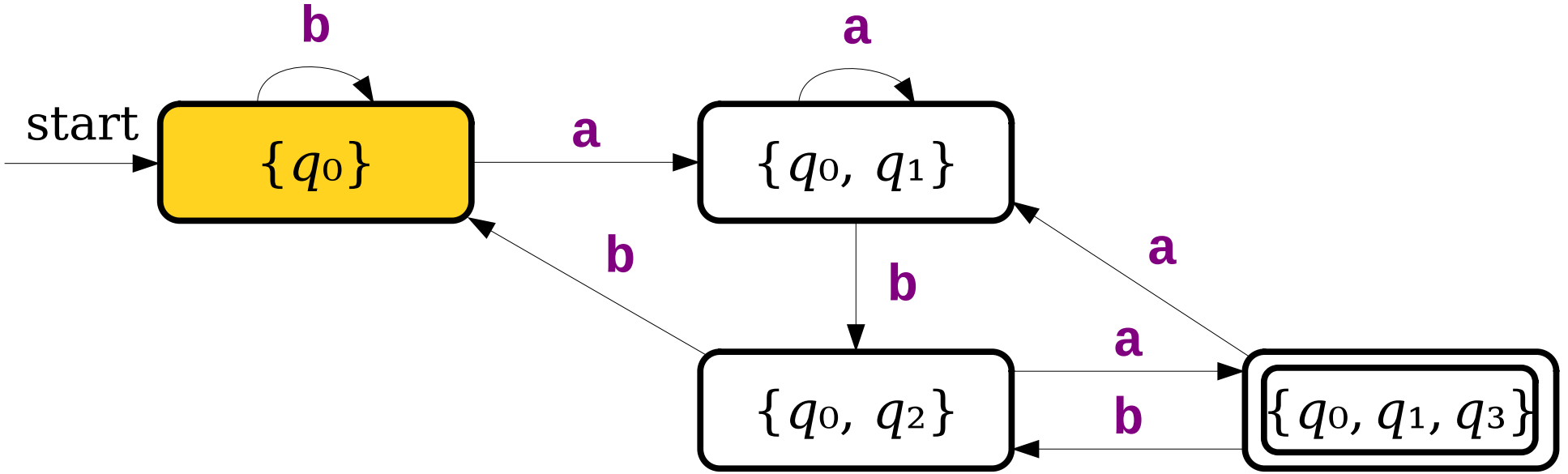


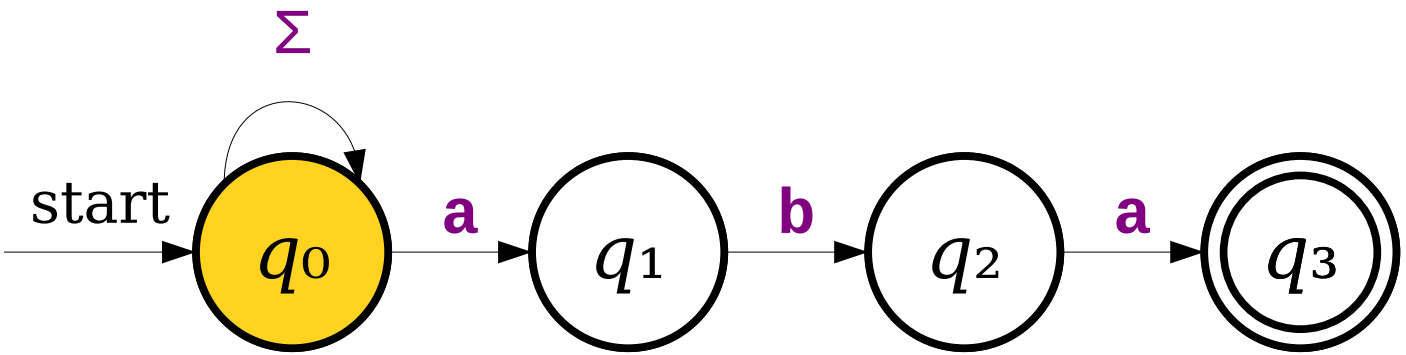
a	b	a	a	b	a

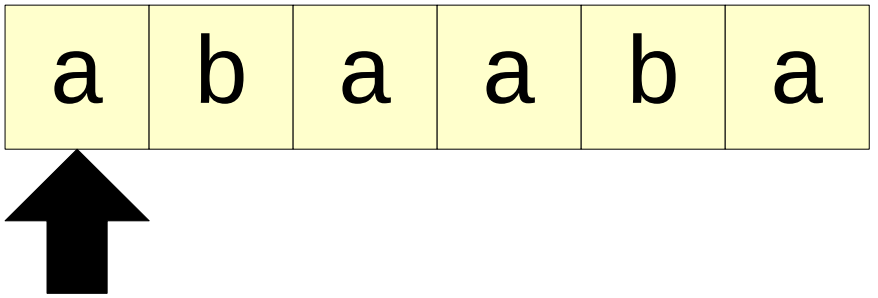


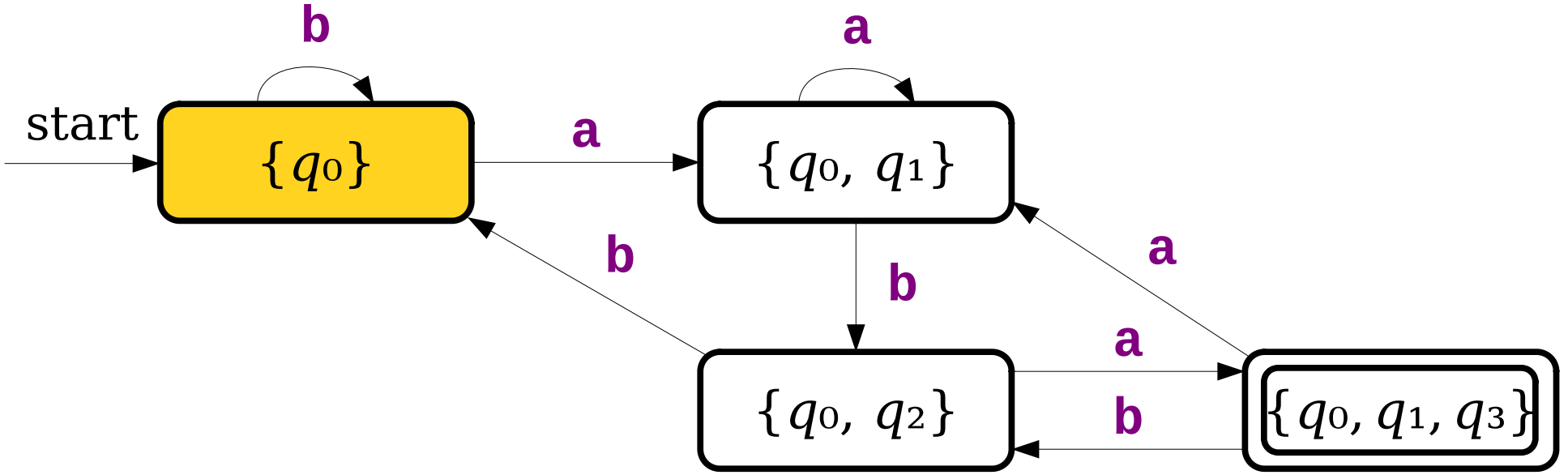


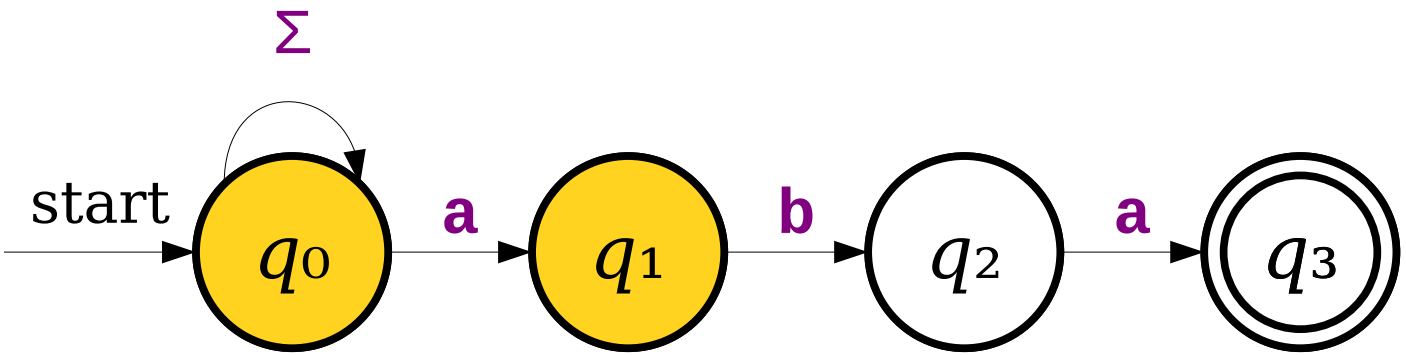
a	b	a	a	b	a

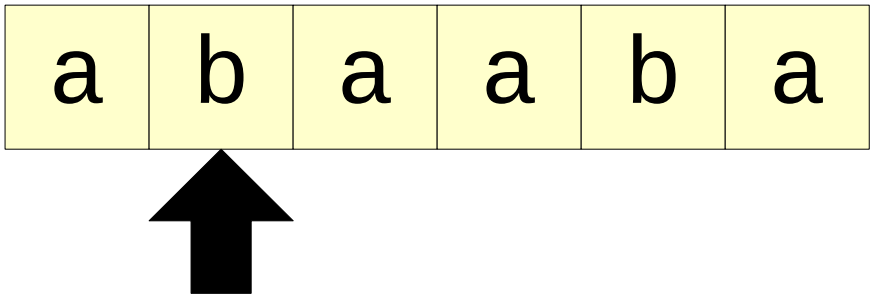


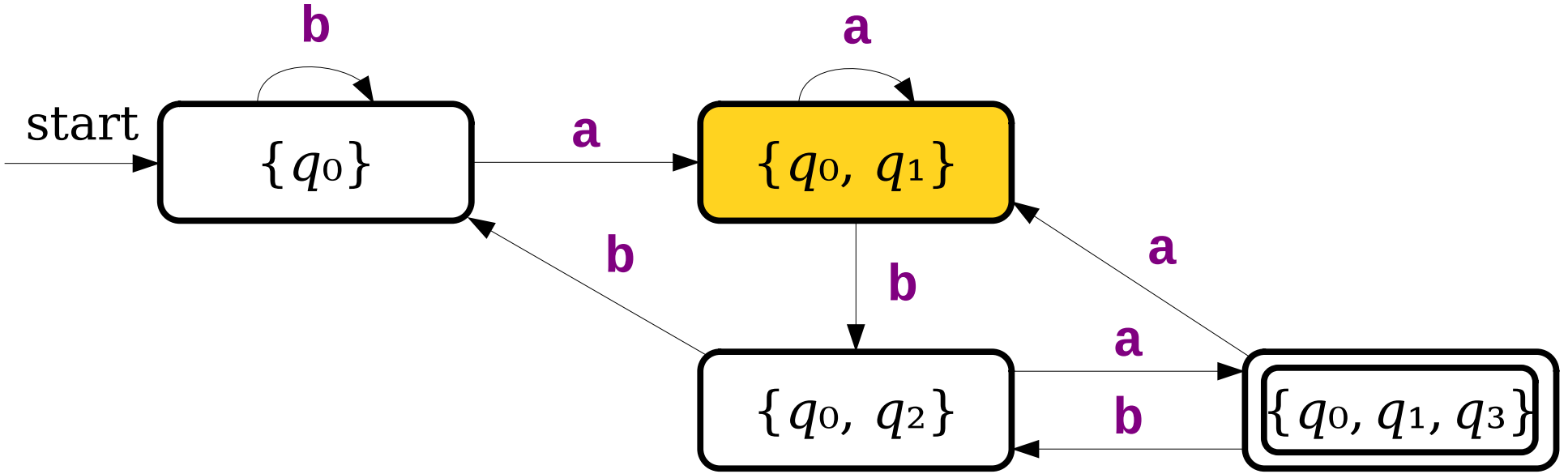

Σ
start
q_0
a
q_1
b
q_2
a
q_3

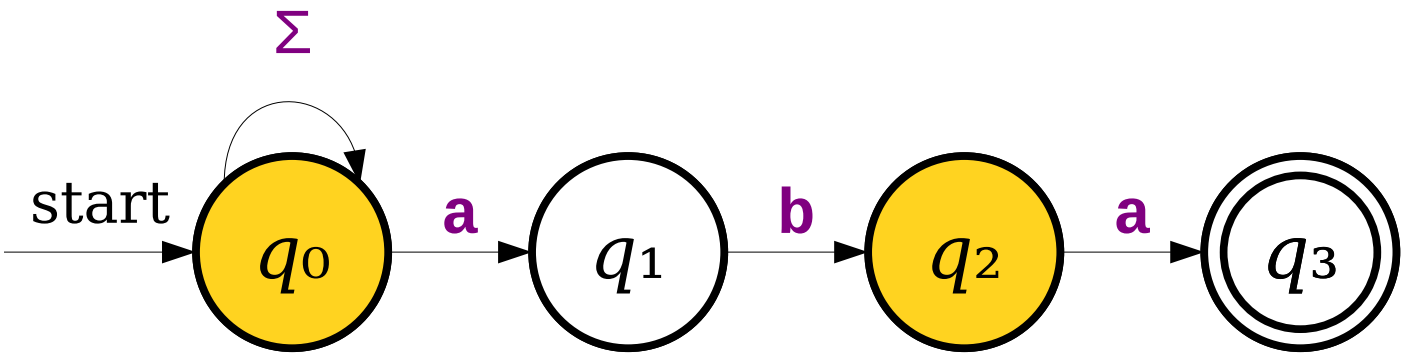

a
b
a
a
b
a

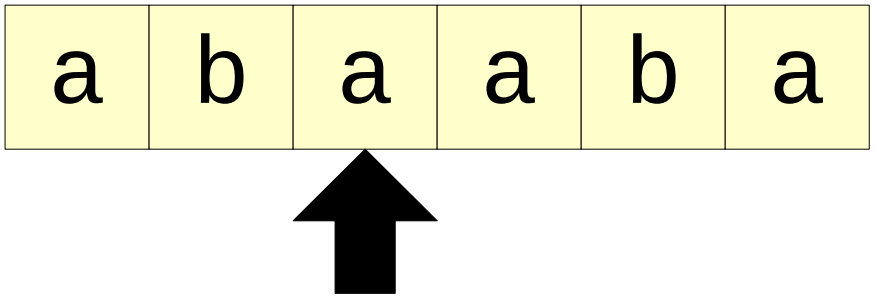

b
a
start
$\{q_0\}$
a
$\{q_0, q_1\}$
b
b
a
a
b
$\{q_0, q_2\}$
$\{q_0, q_1, q_3\}$

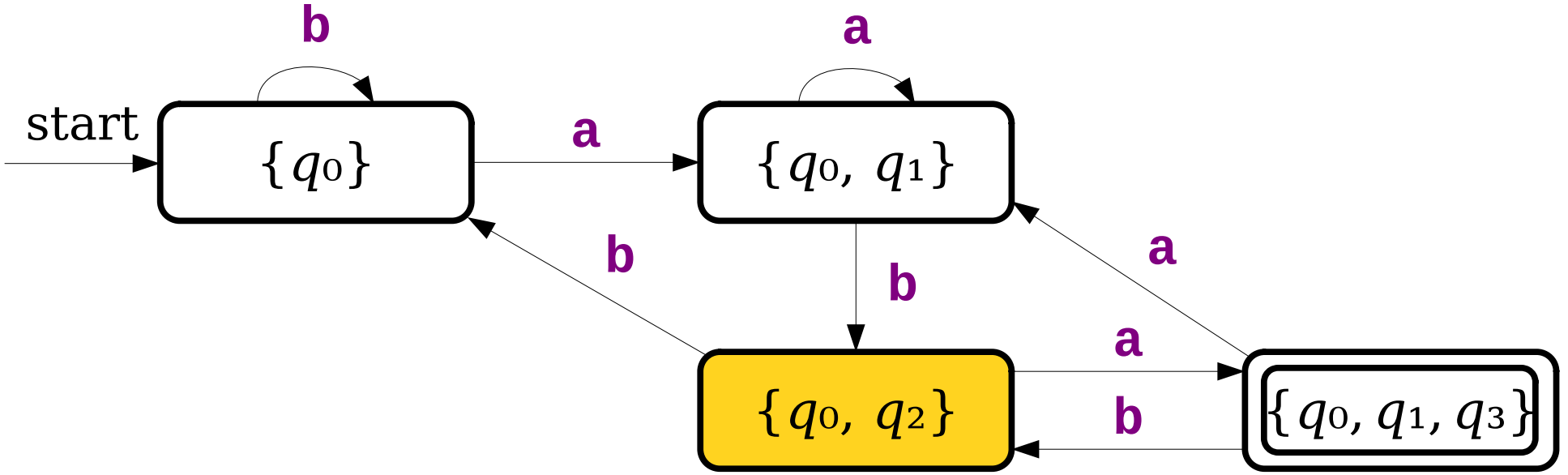

Σ
start
q_0
a
q_1
b
q_2
a
q_3

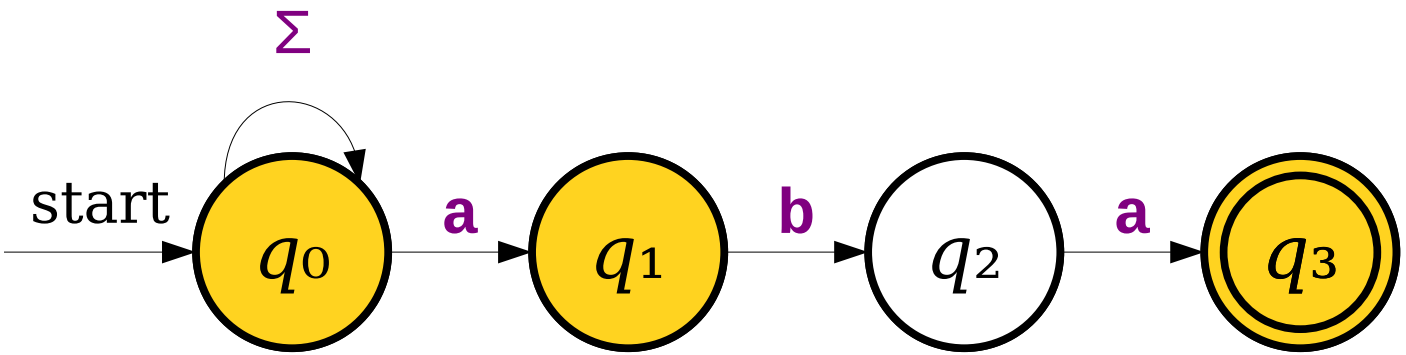

a
b
a
a
b
a

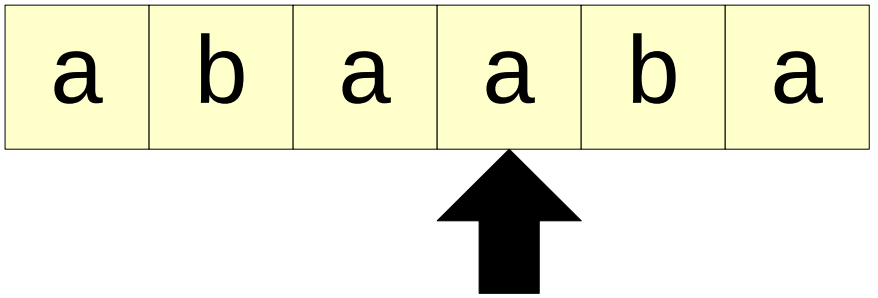

b
a
start
$\{q_0\}$
a
$\{q_0, q_1\}$
b
b
a
a
b
$\{q_0, q_2\}$
$\{q_0, q_1, q_3\}$

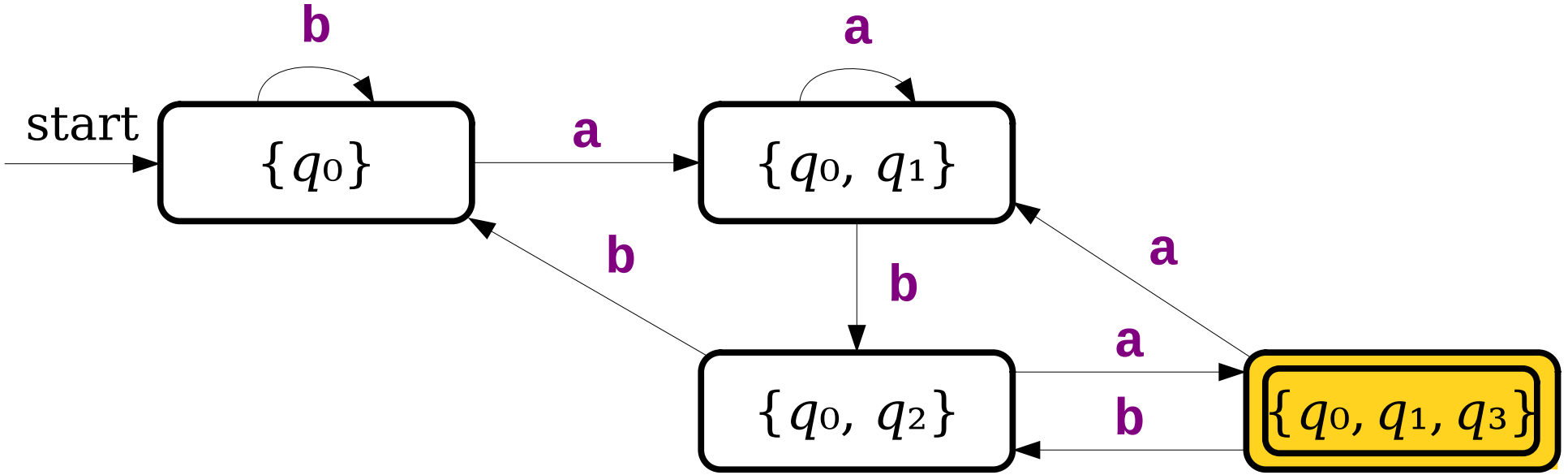

Σ
start
q_0
a
q_1
b
q_2
a
q_3

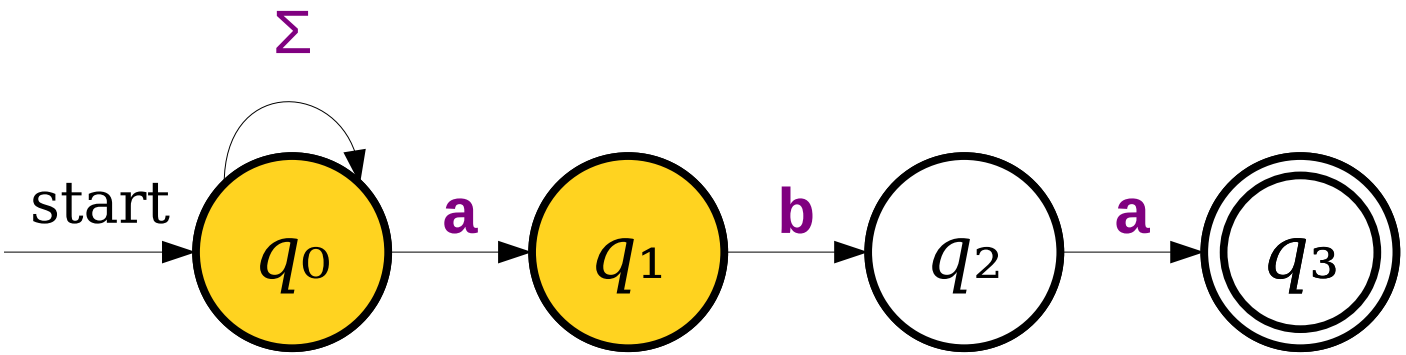

a b a a b a

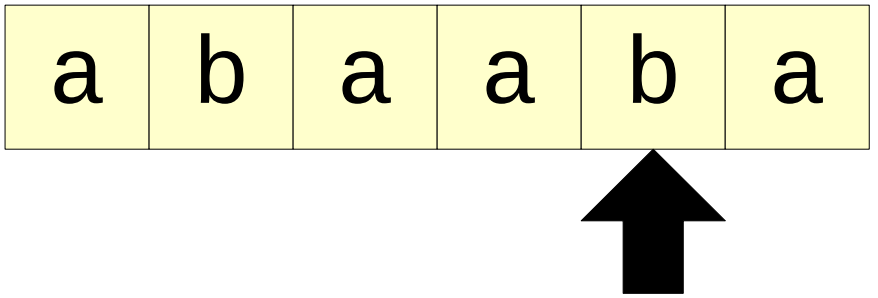

b
a
start
$\{q_0\}$
a
$\{q_0, q_1\}$
b
b
a
a
b
$\{q_0, q_2\}$
$\{q_0, q_1, q_3\}$

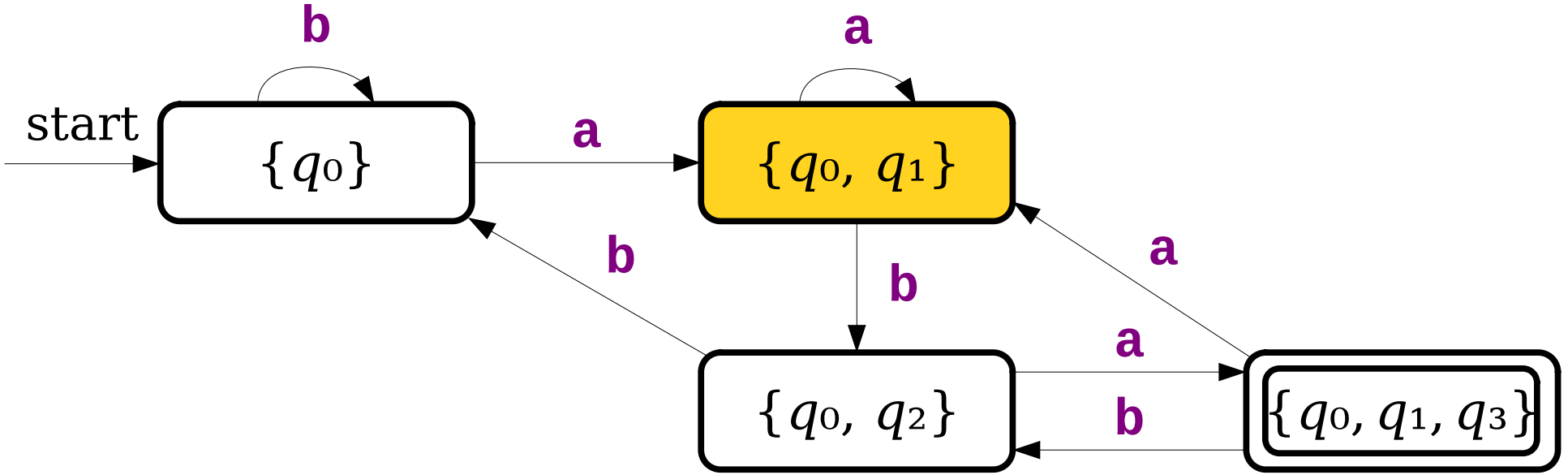

Σ
start
q_0
a
q_1
b
q_2
a
q_3

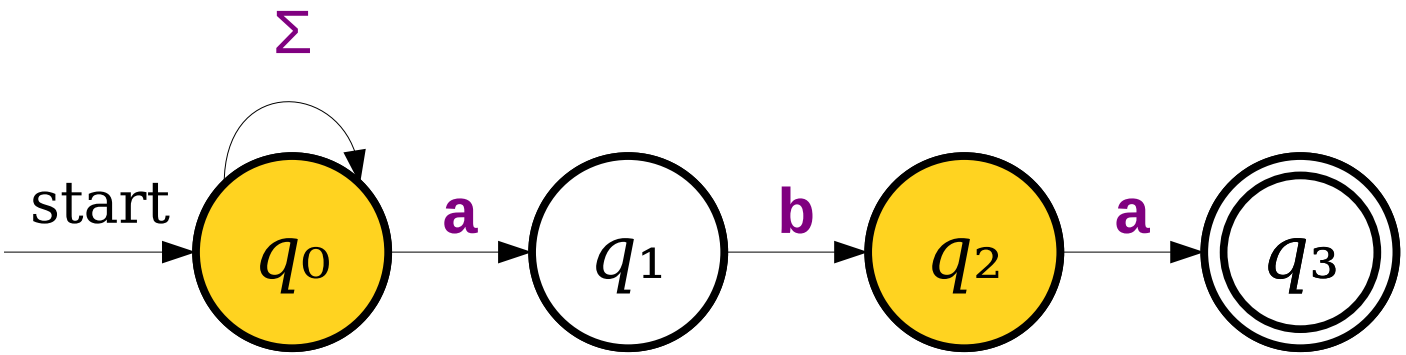

a b a a b a

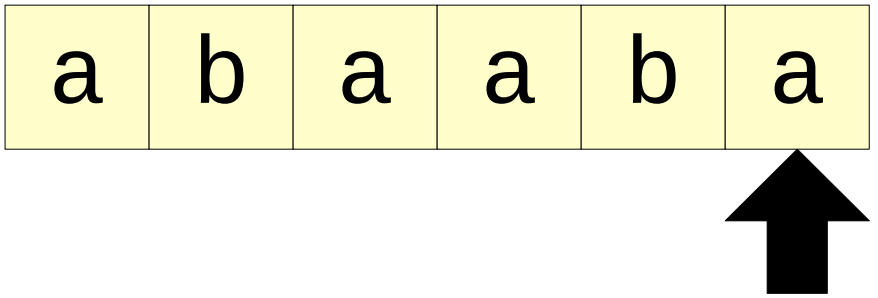

b
a
start
$\{q_0\}$
a
$\{q_0, q_1\}$
b
b
a
a
b
$\{q_0, q_2\}$
$\{q_0, q_1, q_3\}$

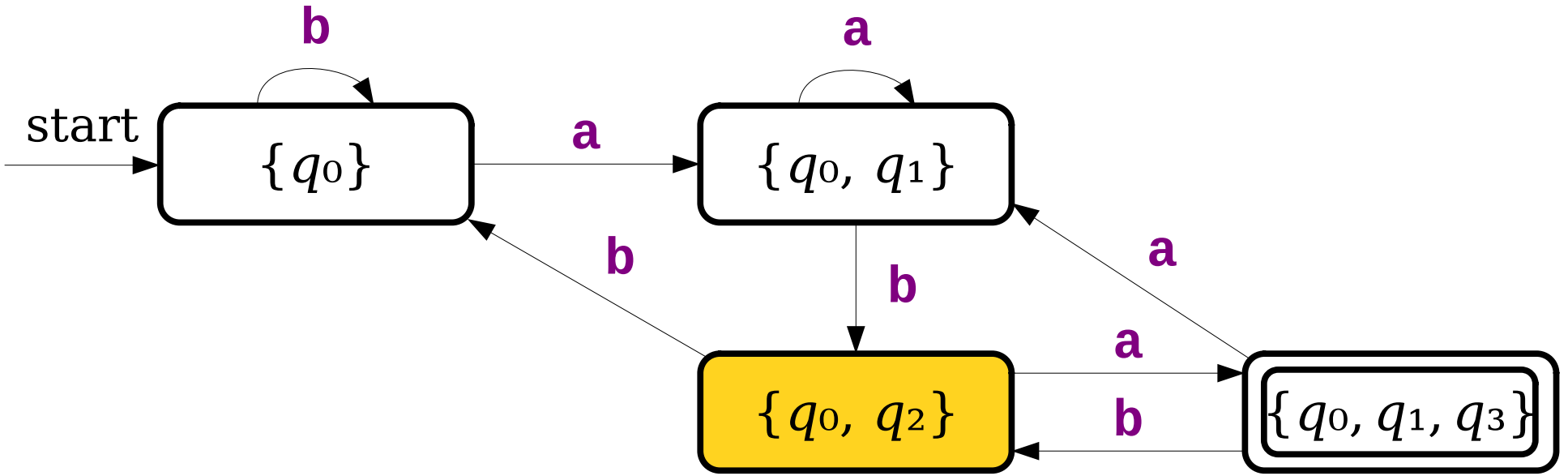

Σ
start
q_0
a
q_1
b
q_2
a
q_3

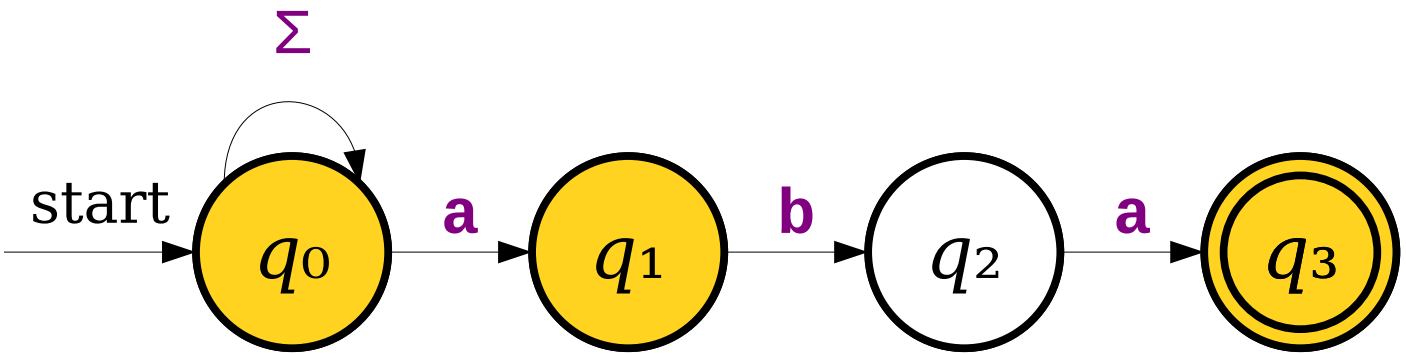

a b a a b a


b
a
start
$\{q_0\}$
a
$\{q_0, q_1\}$
b
b
a
a
b
$\{q_0, q_2\}$
$\{q_0, q_1, q_3\}$

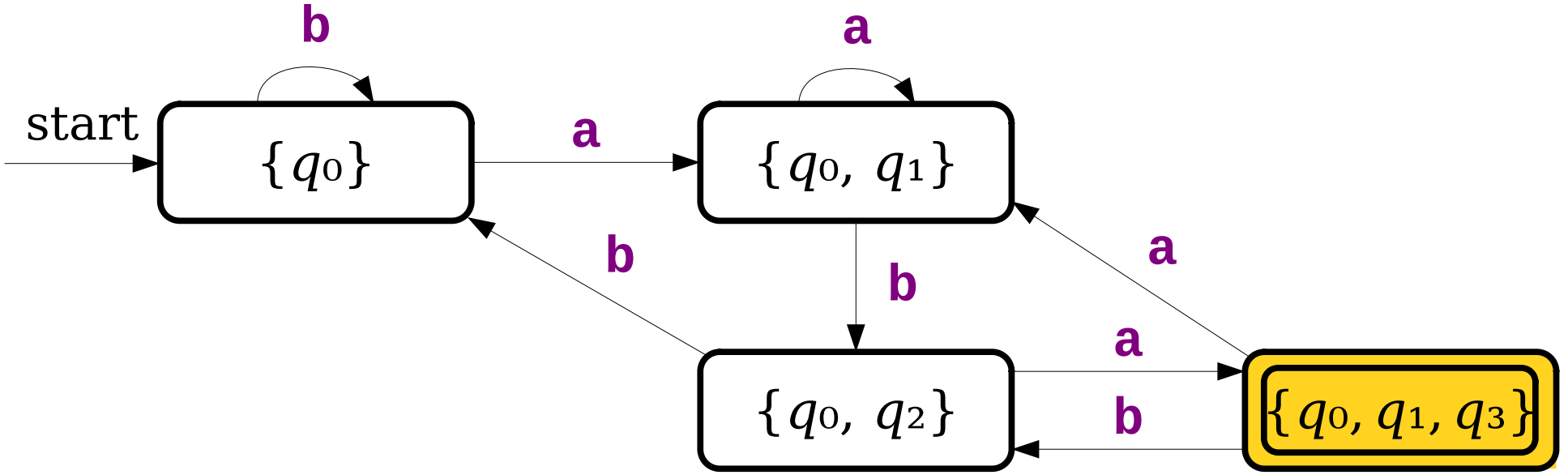

Σ
start
q_0
a
q_1
b
q_2
a
q_3


a b a a b a


b
a
start
$\{q_0\}$
a
$\{q_0, q_1\}$
b
b
a
a
b
$\{q_0, q_2\}$
$\{q_0, q_1, q_3\}$



a	b	a	a	b	a



The Subset Construction

- This procedure for turning an NFA for a language L into a DFA for a language L is called the ***subset construction***.
  - It's sometimes called the ***powerset construction***; it's different names for the same thing!
- Intuitively:
  - Each state in the DFA corresponds to a set of states from the NFA.
  - Each transition in the DFA corresponds to what transitions would be taken in the NFA when using the massive parallel intuition.
  - The accepting states in the DFA correspond to which sets of states would be considered accepting in the NFA when using the massive parallel intuition.

The Subset Construction

- In converting an NFA to a DFA, the DFA's states correspond to sets of NFA states.
- ***Useful fact:*** $|\wp(S)| = 2^{|S|}$ for any finite set S.
- In the worst-case, the construction can result in a DFA that is *exponentially larger* than the original NFA.
- ***Question to ponder:*** Can you find a family of languages that have NFAs of size n, but no DFAs of size less than 2^n?

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

An Important Result

Theorem: A language L is regular if and only if there is some NFA N such that $\mathscr{L}(N) = L$.

Proof Sketch: Pick a language L. First, assume L is regular. That means there's a DFA D where $\mathscr{L}(D) = L$. Every DFA is "basically" an NFA, so there's an NFA (D) whose language is L.

Next, assume there's an NFA N such that $\mathscr{L}(N) = L$. Using the subset construction, we can build a DFA D where $\mathscr{L}(N) = \mathscr{L}(D)$. Then we have that $\mathscr{L}(D) = L$, so L is regular. ■-*ish*

Why This Matters

- We now have two perspectives on regular languages:
  - Regular languages are languages accepted by DFAs.
  - Regular languages are languages accepted by NFAs.
- We can now reason about the regular languages in two different ways.

Properties of Regular Languages

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

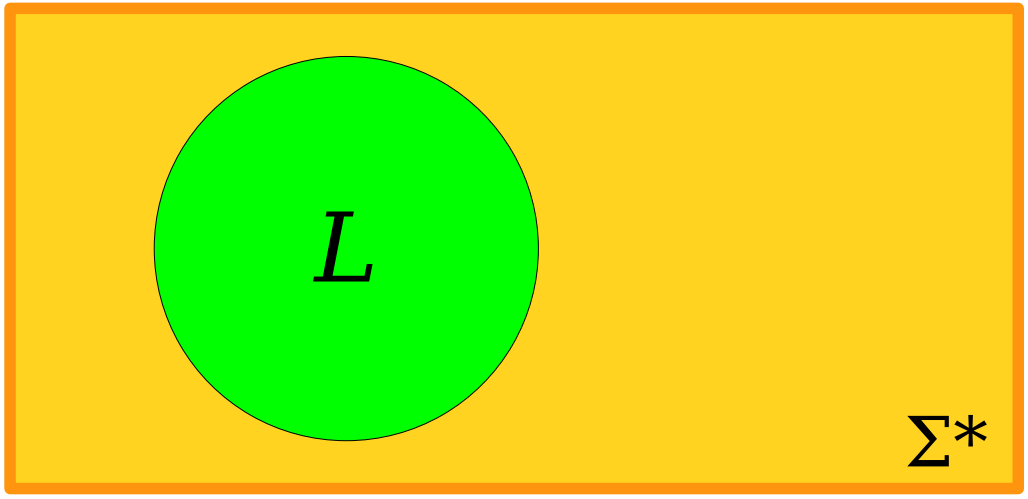
- Given a language $L \subseteq \Sigma^*$, the ***complement*** of that language (denoted $\overline{L}$) is the language of all strings in Σ^* that aren't in L.
- Formally:

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

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:

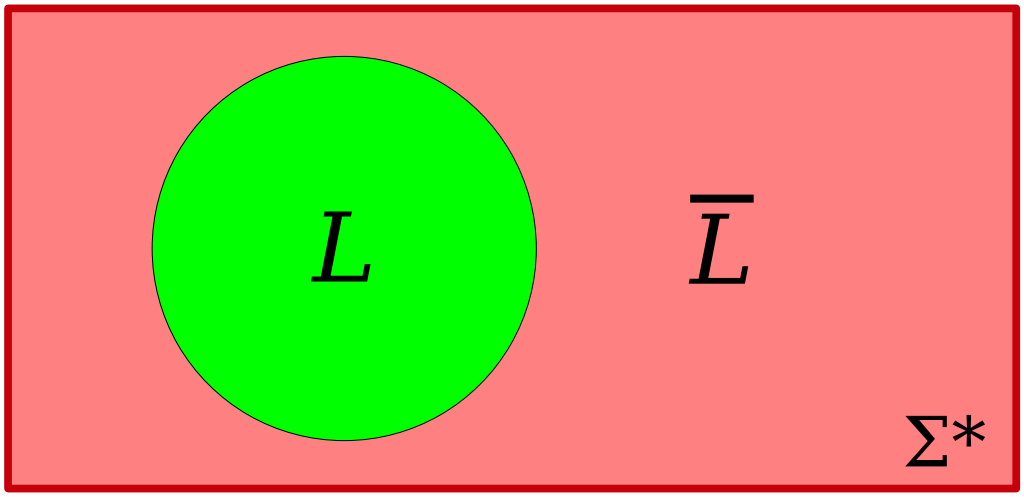
$$\overline{L} = \Sigma^* - L$$



The Complement of a Language

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

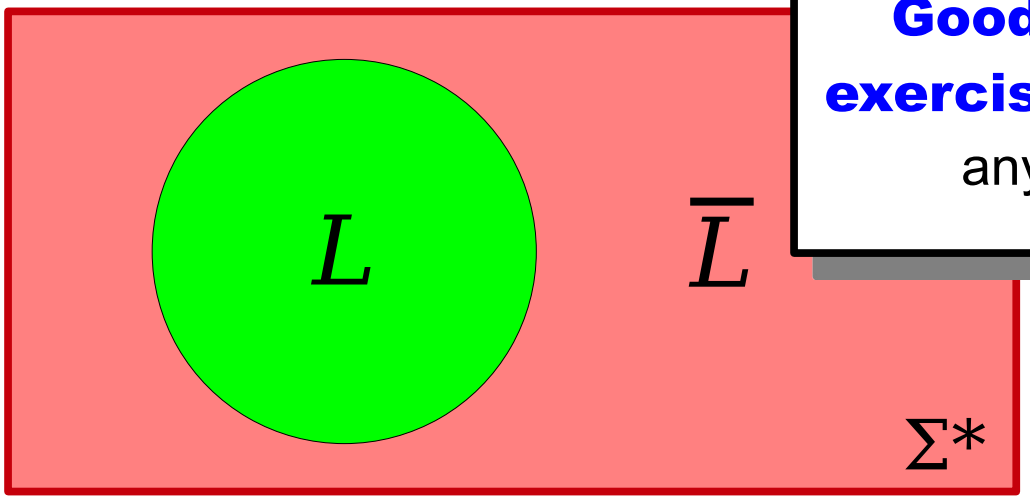
$$\overline{L} = \Sigma^* - L$$



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:

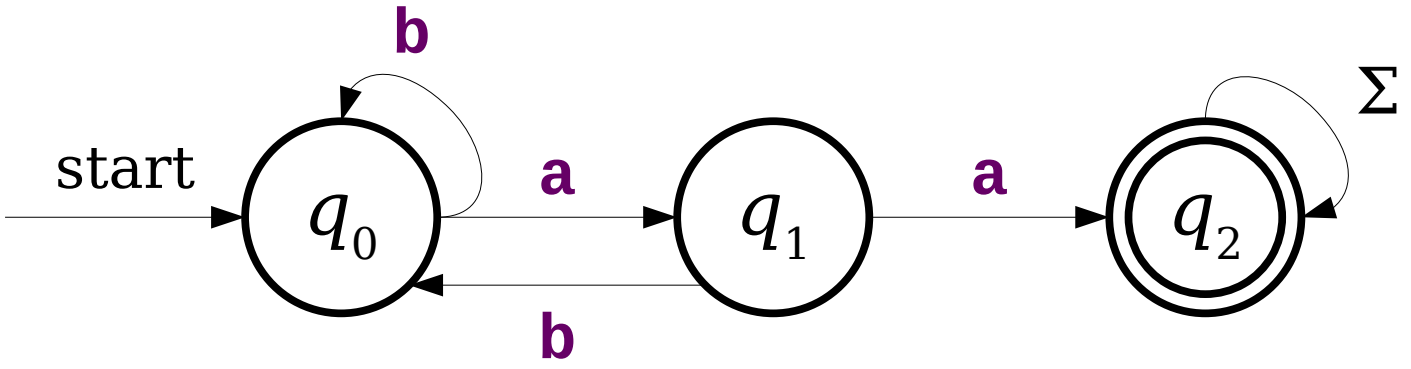
$$\overline{L} = \Sigma^* - L$$



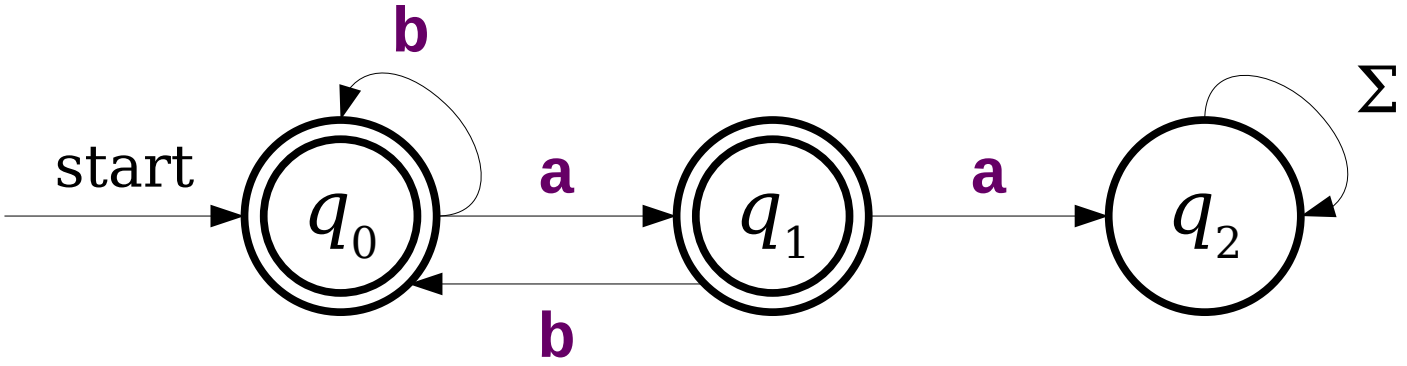
Good proofwriting exercise: prove $\overline{\overline{L}} = L$ for any language L.

Complementing Regular Languages

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

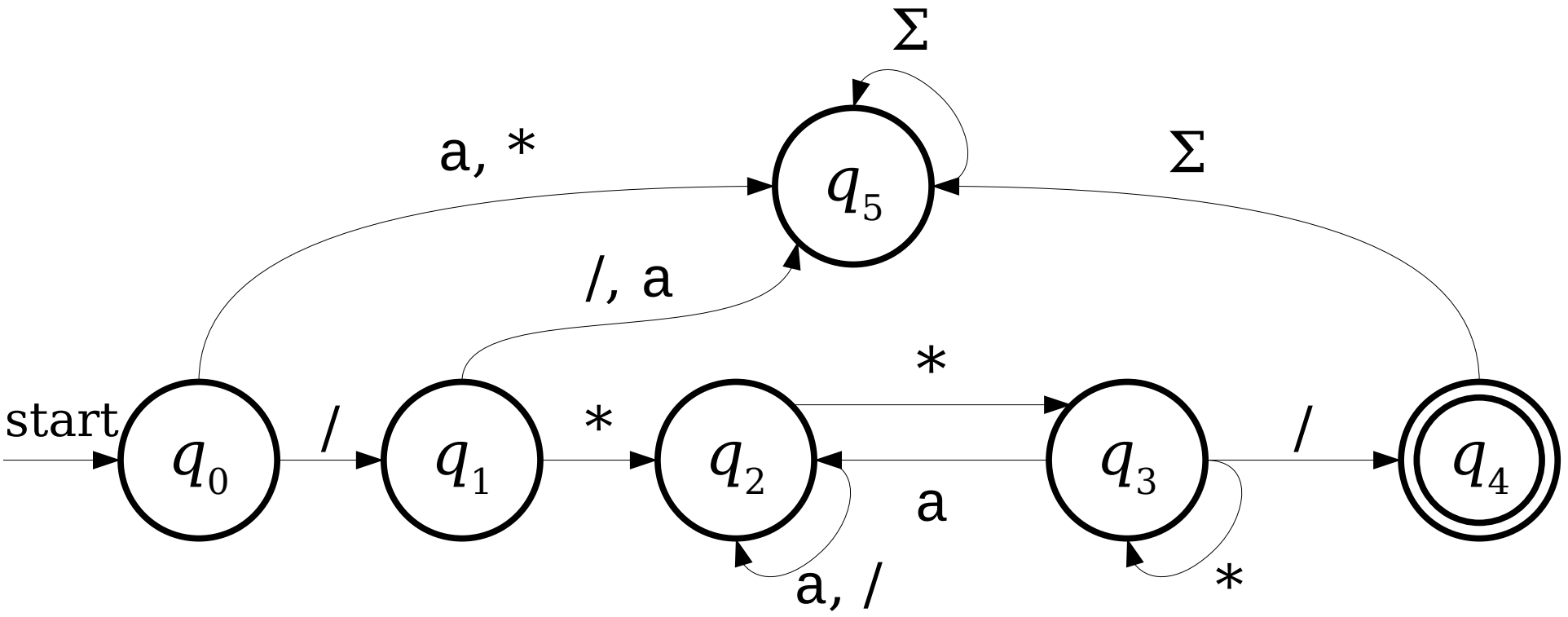


$\overline{L} = \{ w \in \{\mathbf{a}, \mathbf{b}\}^* \mid w$ ***does not*** contain **aa** as a substring $\}$



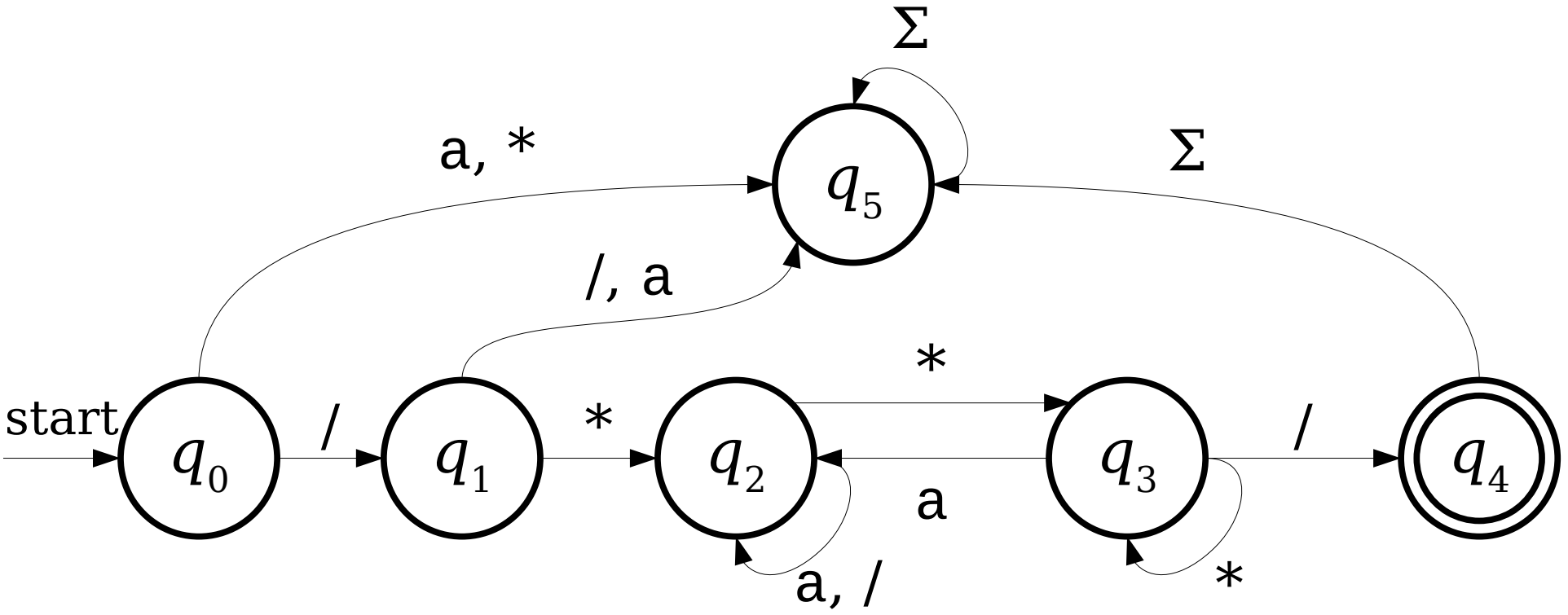
Complementing Regular Languages

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



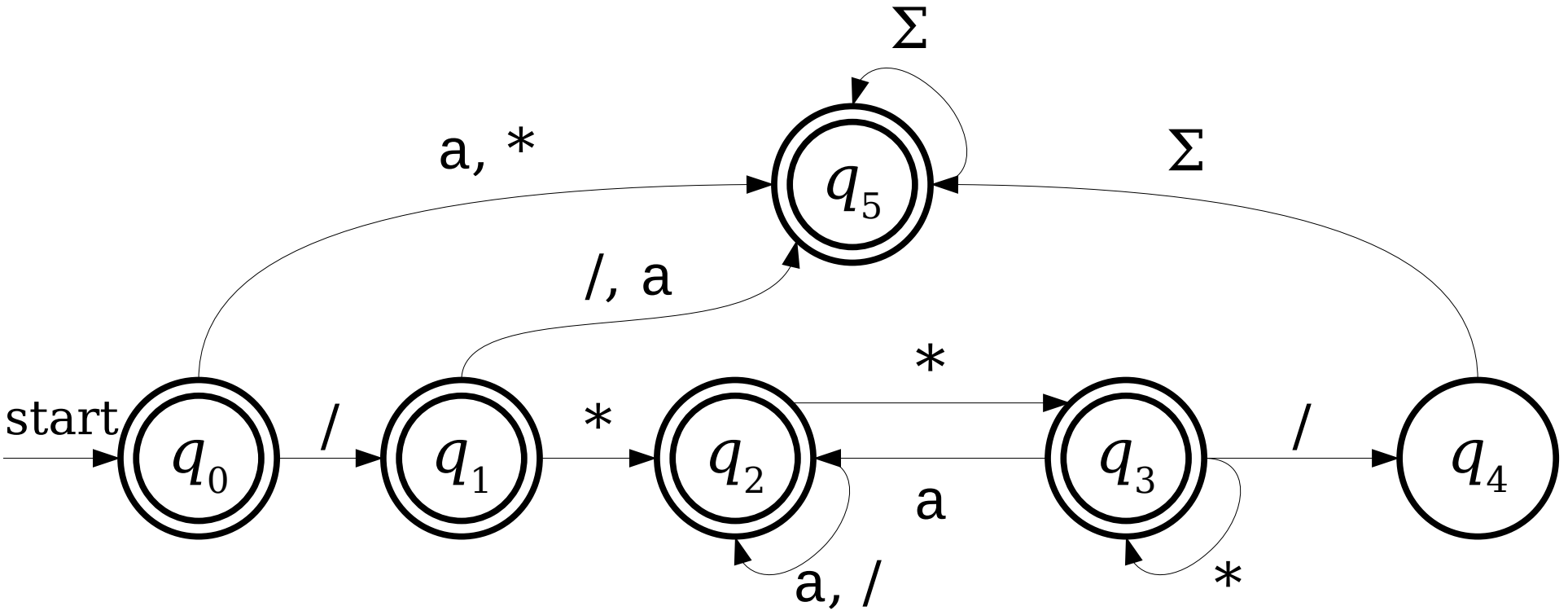
Complementing Regular Languages

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



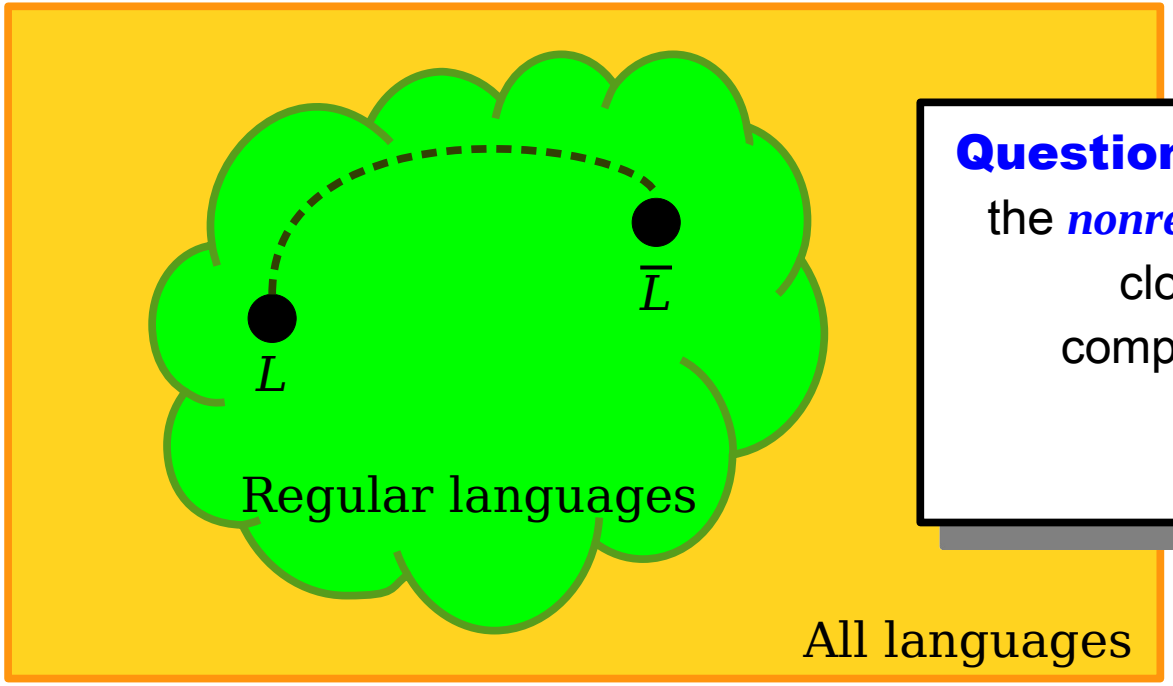
Complementing Regular Languages

$\overline{L} = \{ w \in \{\mathbf{a}, \mathbf{*}, \mathbf{/}\}^* \mid w$ ***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***.



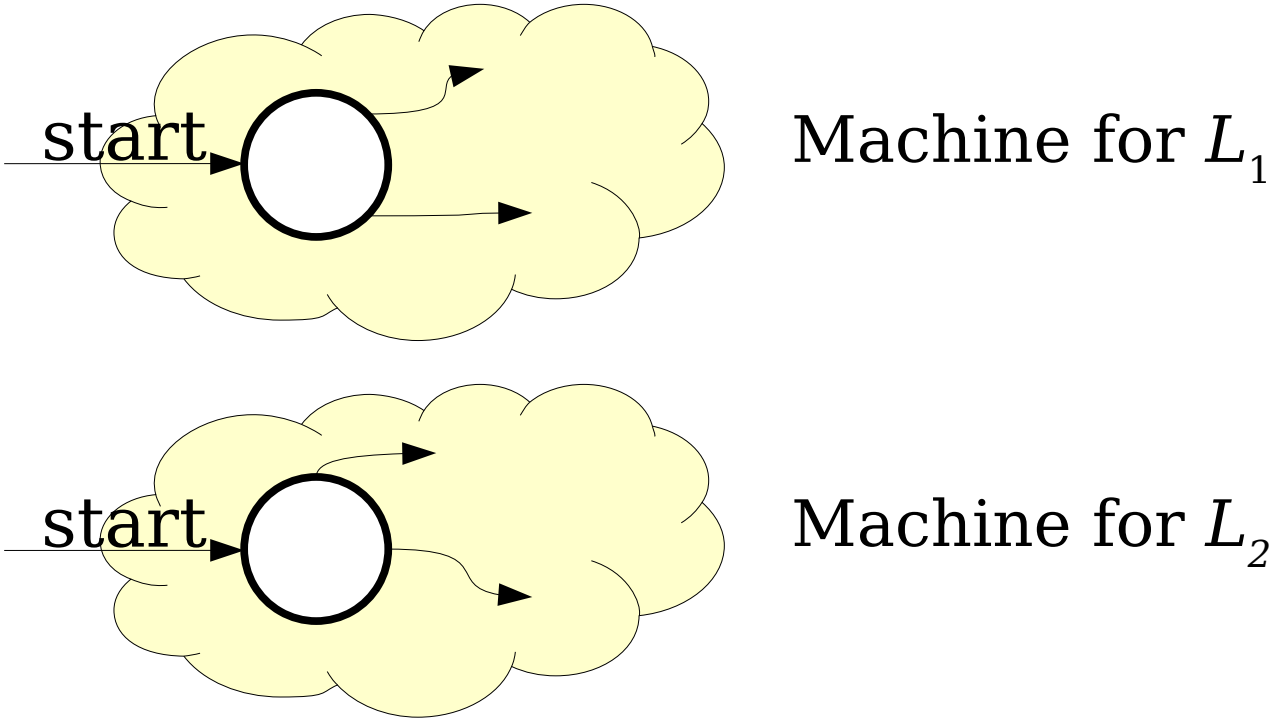
Question to ponder: are the ***nonregular*** languages closed under complementation?

The Union of Two Languages

- If L_1 and L_2 are languages over the alphabet Σ, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.
- If L_1 and L_2 are regular languages, is $L_1 \cup L_2$?

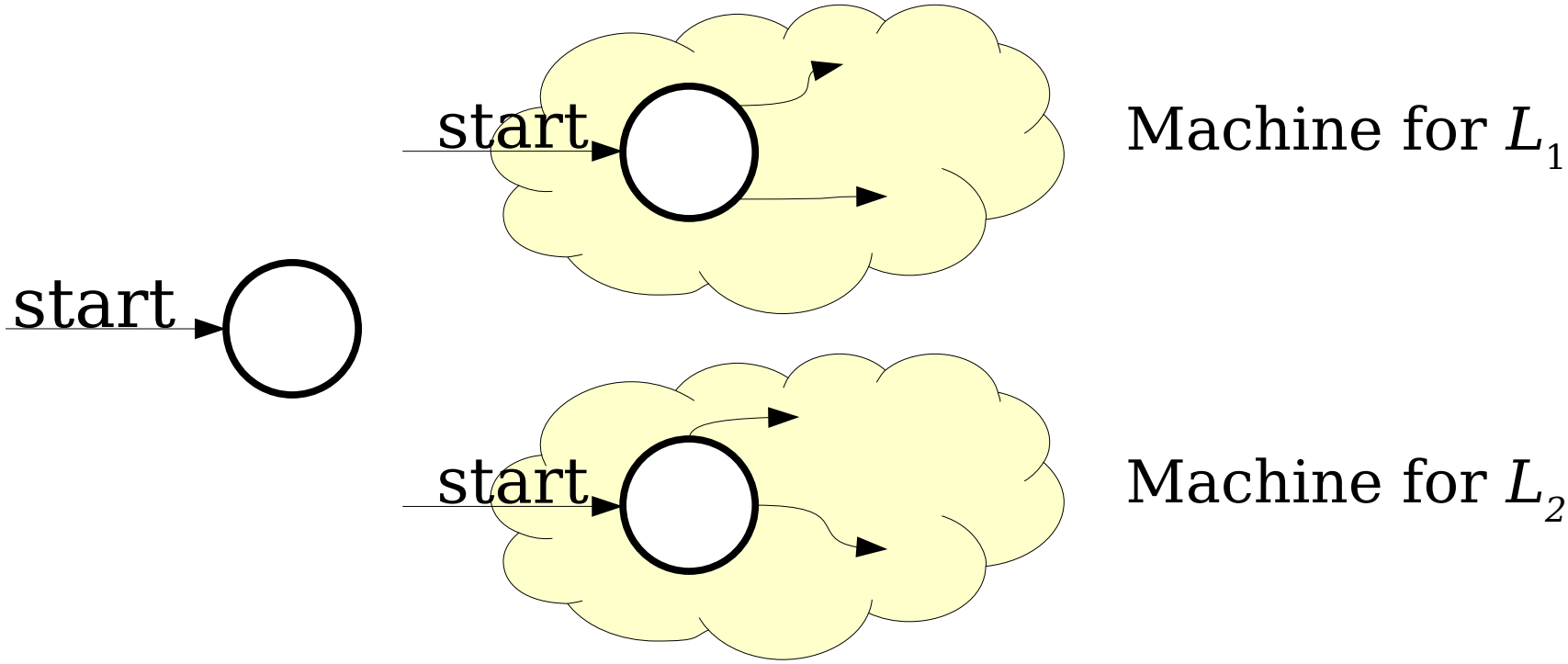
The Union of Two Languages

- If L_1 and L_2 are languages over the alphabet Σ, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.
- If L_1 and L_2 are regular languages, is $L_1 \cup L_2$?



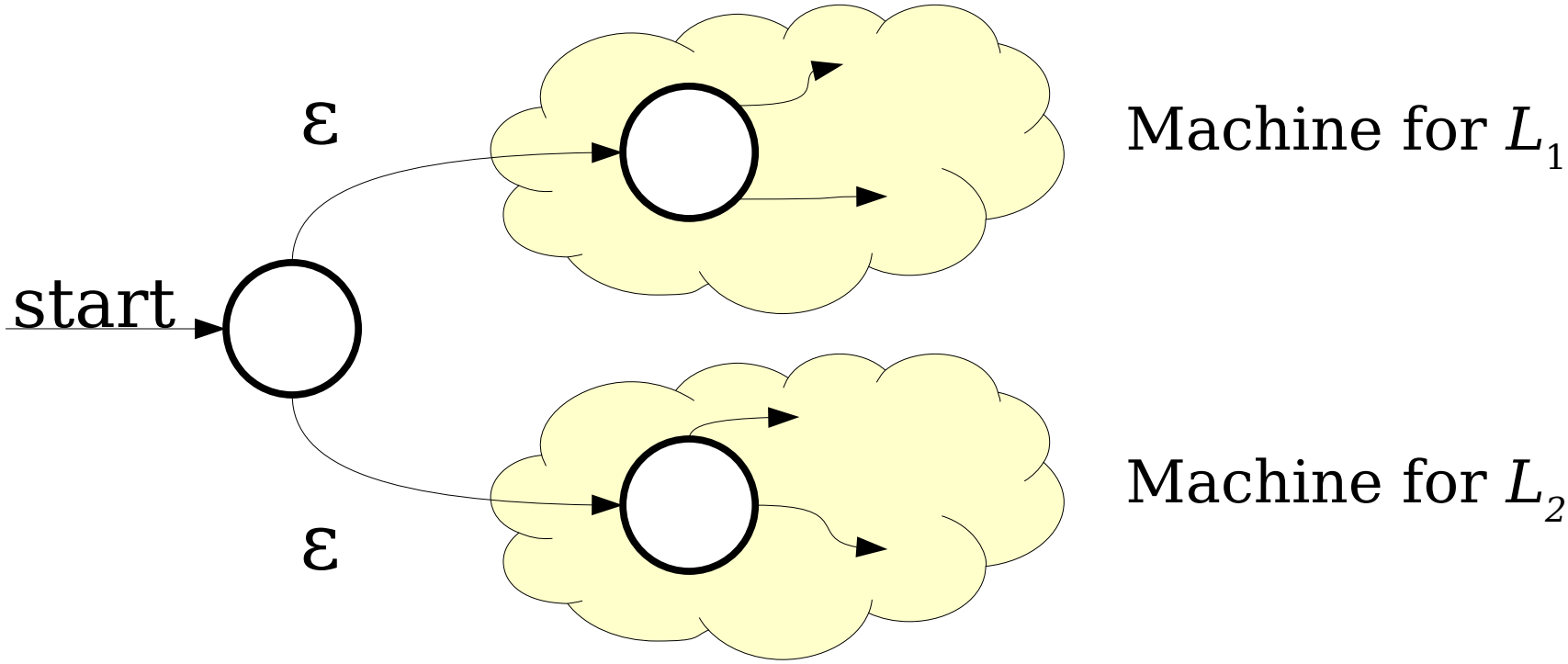
The Union of Two Languages

- If L_1 and L_2 are languages over the alphabet Σ, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.
- If L_1 and L_2 are regular languages, is $L_1 \cup L_2$?



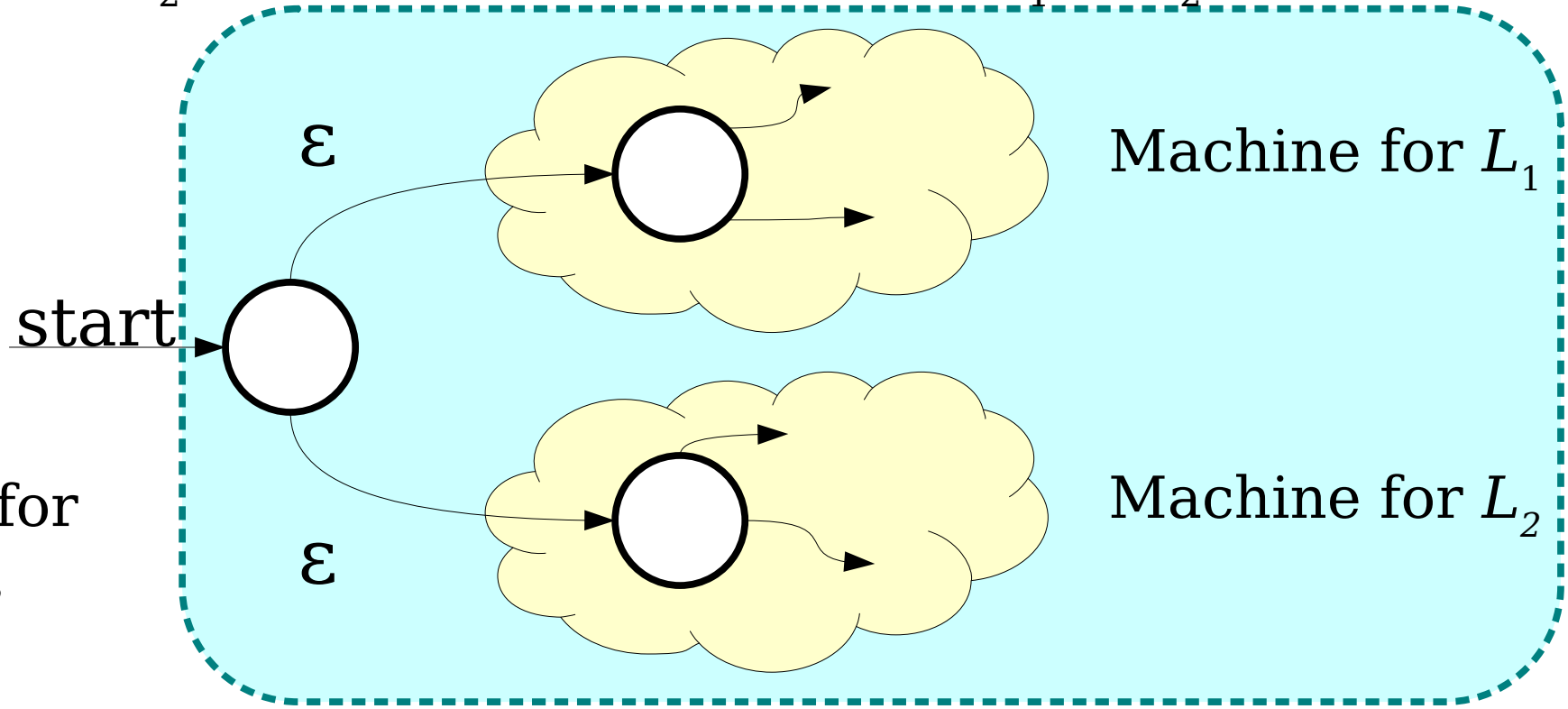
The Union of Two Languages

- If L_1 and L_2 are languages over the alphabet Σ, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.
- If L_1 and L_2 are regular languages, is $L_1 \cup L_2$?



The Union of Two Languages

- If L_1 and L_2 are languages over the alphabet Σ, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.
- If L_1 and L_2 are regular languages, is $L_1 \cup L_2$?



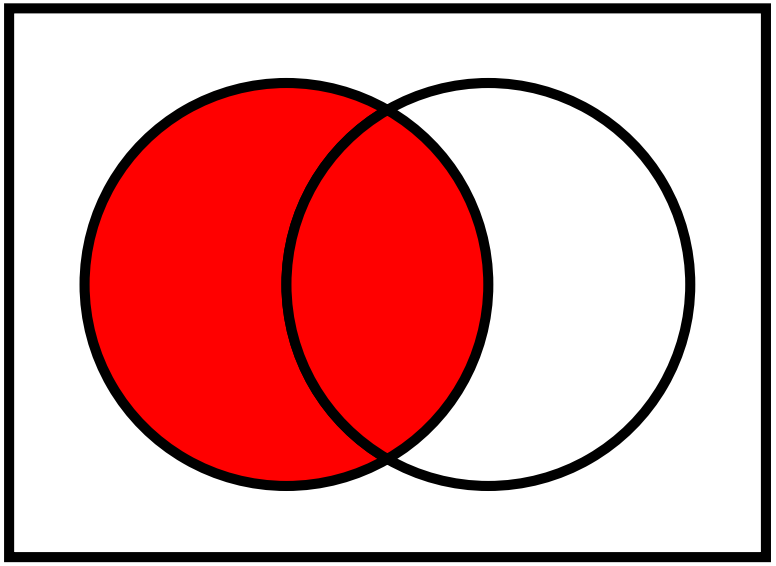
Machine for $L_1 \cup L_2$

The Intersection of Two Languages

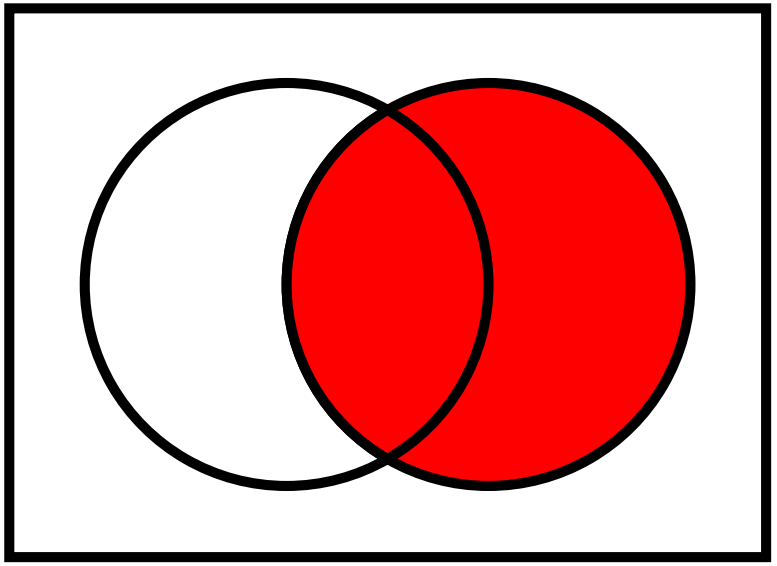
- If L_1 and L_2 are languages over Σ, then $L_1 \cap L_2$ is the language of strings in both L_1 and L_2.
- Question: If L_1 and L_2 are regular, is $L_1 \cap L_2$ regular as well?

The Intersection of Two Languages

- If L_1 and L_2 are languages over Σ, then $L_1 \cap L_2$ is the language of strings in both L_1 and L_2.
- Question: If L_1 and L_2 are regular, is $L_1 \cap L_2$ regular as well?



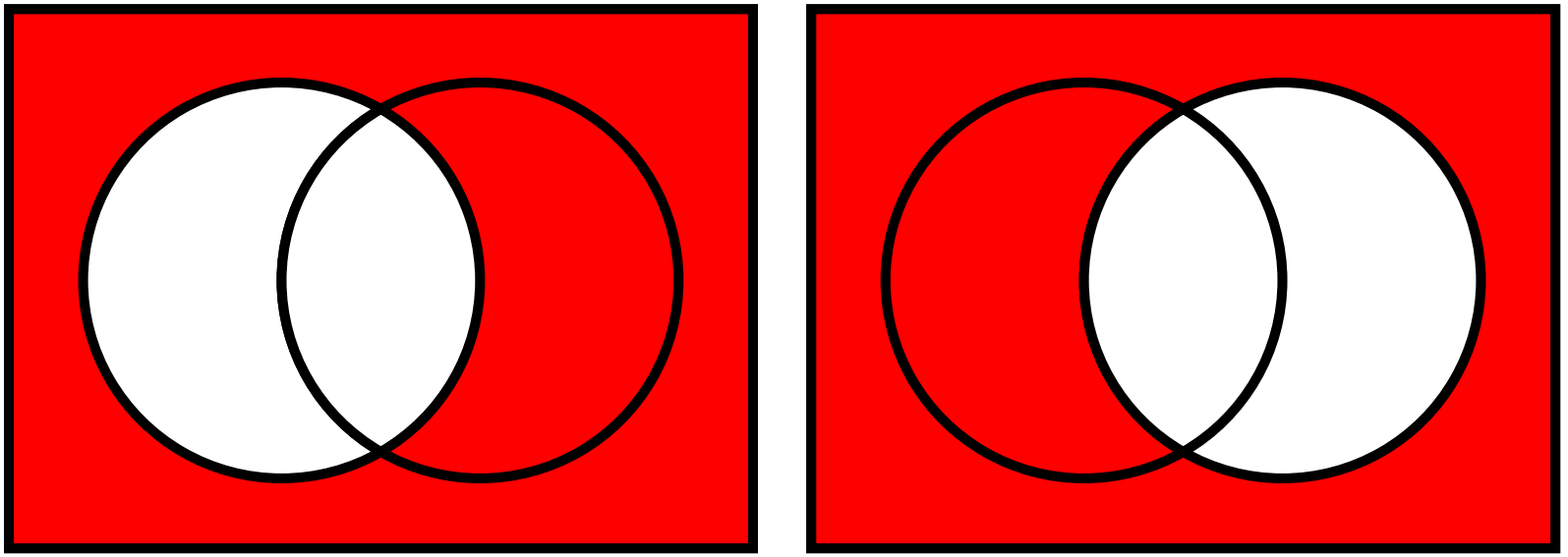
L_1



L_2

The Intersection of Two Languages

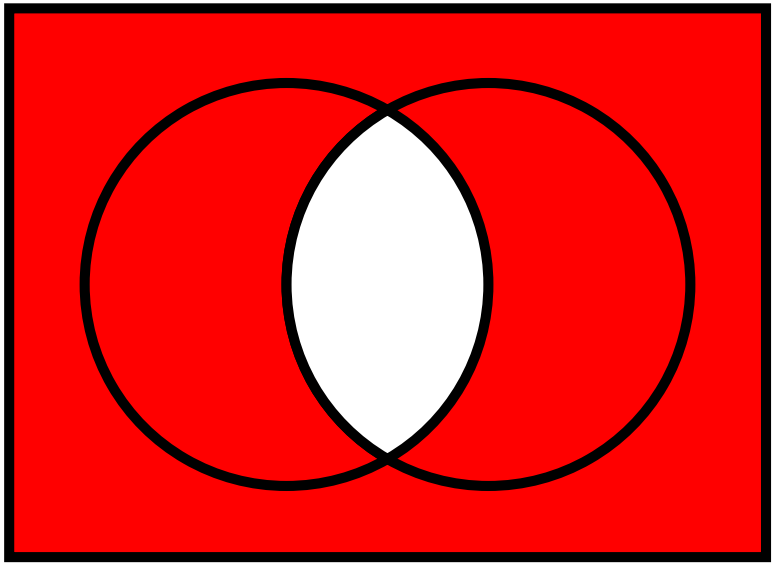
- If L_1 and L_2 are languages over Σ, then $L_1 \cap L_2$ is the language of strings in both L_1 and L_2.
- Question: If L_1 and L_2 are regular, is $L_1 \cap L_2$ regular as well?



$\overline{L}_1$ $\overline{L}_2$

The Intersection of Two Languages

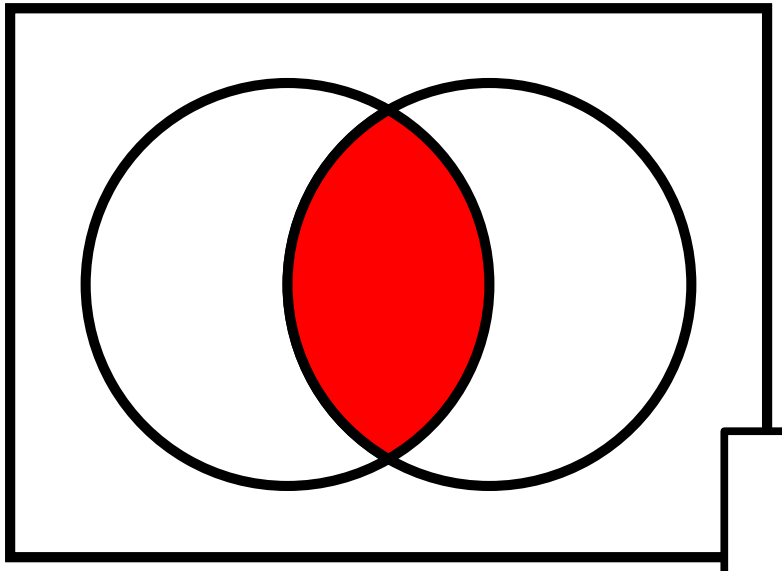
- If L_1 and L_2 are languages over Σ, then $L_1 \cap L_2$ is the language of strings in both L_1 and L_2.
- Question: If L_1 and L_2 are regular, is $L_1 \cap L_2$ regular as well?



$$\overline{L}_1 \cup \overline{L}_2$$

The Intersection of Two Languages

- If L_1 and L_2 are languages over Σ, then $L_1 \cap L_2$ is the language of strings in both L_1 and L_2.
- Question: If L_1 and L_2 are regular, is $L_1 \cap L_2$ regular as well?



$$\overline{\overline{L_1} \cup \overline{L_2}}$$

Hey, it's De Morgan's laws!

Concatenation

String Concatenation

- If $w \in \Sigma^*$ and $x \in \Sigma^*$, the ***concatenation*** of w and x, denoted ***wx***, is the string formed by tacking all the characters of x onto the end of w.
- Example: if w = **quo** and x = **kka**, the concatenation wx = **quokka**.
- This is analogous to the + operator for strings in many programming languages.
- Some facts about concatenation:
  - The empty string ε is the ***identity element*** for concatenation:

    $$w\varepsilon = \varepsilon w = w$$

  - Concatenation is ***associative***:

    $$wxy = w(xy) = (wx)y$$

Concatenation

- The ***concatenation*** of two languages L_1 and L_2 over the alphabet Σ is the language

$$L_1L_2 = \{ wx \in \Sigma^* \mid w \in L_1 \wedge x \in L_2 \}$$

Concatenation Example

- Let Σ = { **a**, **b**, …, **z**, **A**, **B**, …, **Z** } and consider these languages over Σ:
  - ***Noun*** = { **Puppy**, **Rainbow**, **Whale**, … }
  - ***Verb*** = { **Hugs**, **Juggles**, **Loves**, … }
  - ***The*** = { **The** }
- The language ***TheNounVerbTheNoun*** is
  - { **ThePuppyHugsTheWhale**, **TheWhaleLovesTheRainbow**, **TheRainbowJugglesTheRainbow**, … }

Concatenation

- The ***concatenation*** of two languages L_1 and L_2 over the alphabet Σ is the language

$$L_1L_2 = \{ wx \in \Sigma^* \mid w \in L_1 \wedge x \in L_2 \}$$

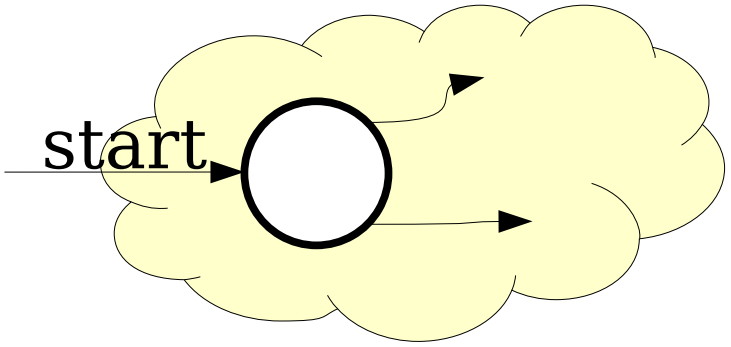
- Two views of L_1L_2:
  - The set of all strings that can be made by concatenating a string in L_1 with a string in L_2.
  - The set of strings that can be split into two pieces: a piece from L_1 and a piece from L_2.

Concatenating Regular Languages

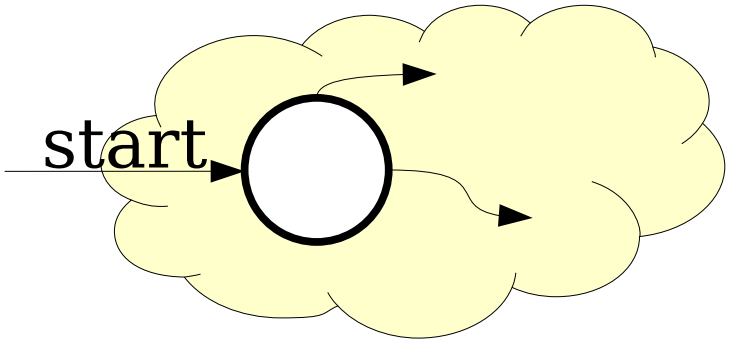
- If L_1 and L_2 are regular languages, is L_1L_2?
- Intuition – can we split a string w into two strings xy such that $x \in L_1$ and $y \in L_2$?

Concatenating Regular Languages

- If L_1 and L_2 are regular languages, is L_1L_2?
- Intuition – can we split a string w into two strings xy such that $x \in L_1$ and $y \in L_2$?



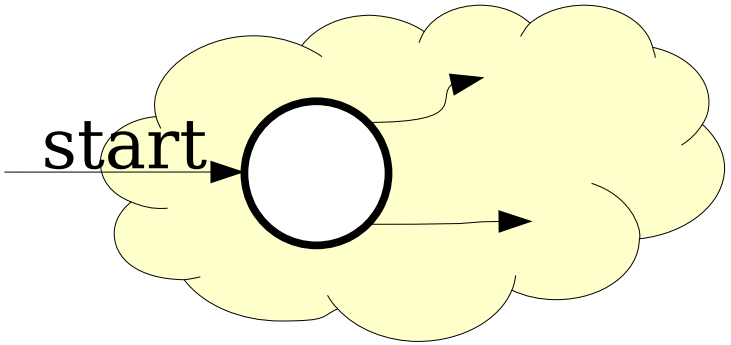
Machine for L_1



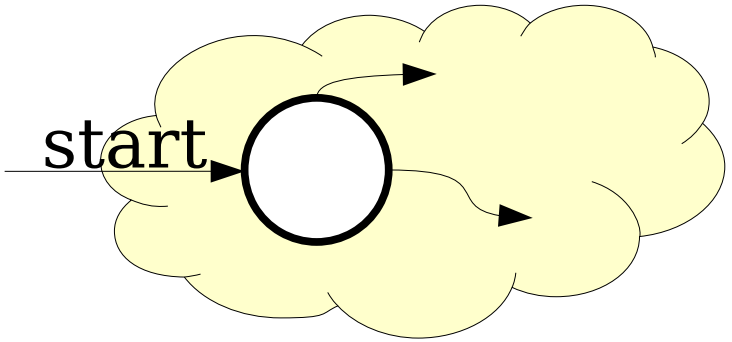
Machine for L_2

Concatenating Regular Languages

- If L_1 and L_2 are regular languages, is L_1L_2?
- Intuition – can we split a string w into two strings xy such that $x \in L_1$ and $y \in L_2$?



Machine for L_1

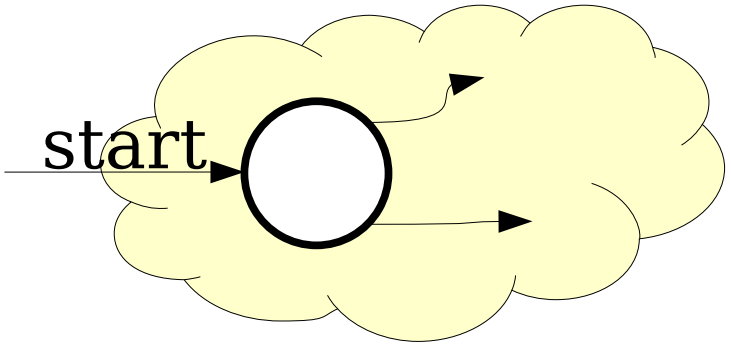


Machine for L_2

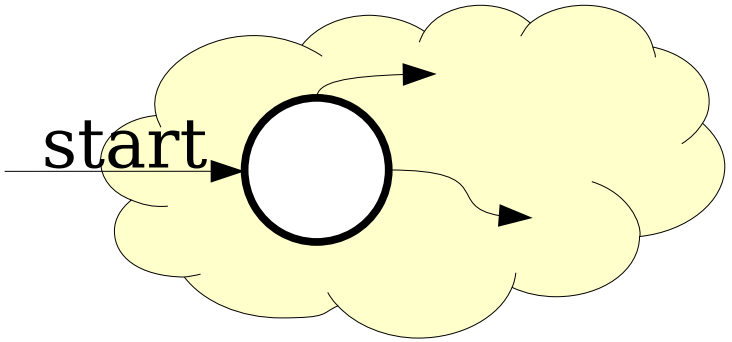
b	**o**	**o**	**k**	**k**	**e**	**e**	**p**	**e**	**r**

Concatenating Regular Languages

- If L_1 and L_2 are regular languages, is L_1L_2?
- Intuition – can we split a string w into two strings xy such that $x \in L_1$ and $y \in L_2$?



Machine for L_1

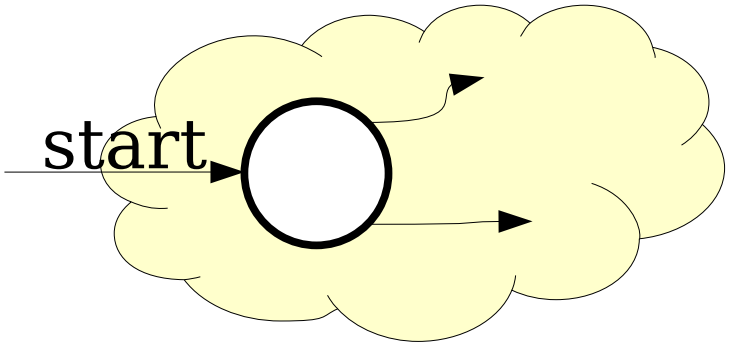


Machine for L_2

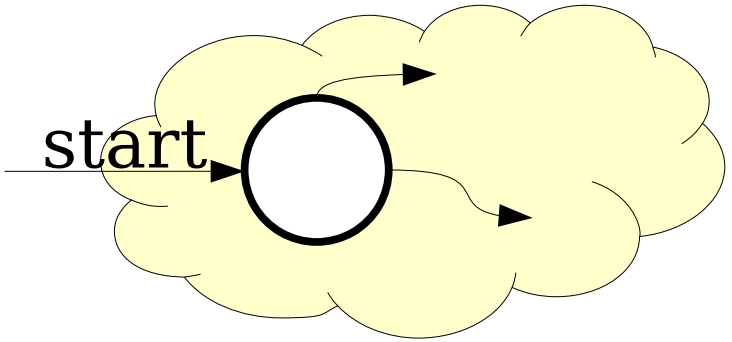
b	**o**	**o**	**k**	**k**	**e**	**e**	**p**	**e**	**r**

Concatenating Regular Languages

- If L_1 and L_2 are regular languages, is L_1L_2?
- Intuition – can we split a string w into two strings xy such that $x \in L_1$ and $y \in L_2$?



Machine for L_1



Machine for L_2

b	o	o	k

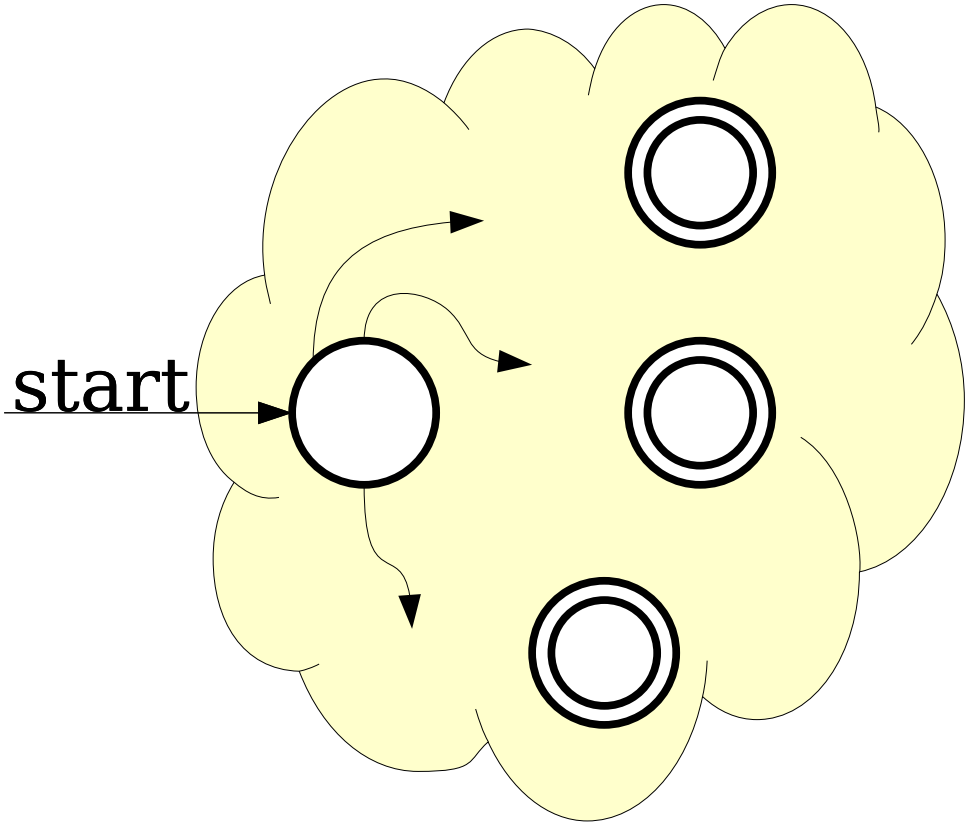
k	e	e	p	e	r

Concatenating Regular Languages

- If L_1 and L_2 are regular languages, is L_1L_2?
- Intuition – can we split a string w into two strings xy such that $x \in L_1$ and $y \in L_2$?
- ***Idea***:
  - Run a DFA/NFA for L_1 on w.
  - Whenever it reaches an accepting state, optionally hand the rest of w to a DFA/NFA for L_2.
  - If the automaton for L_2 accepts the rest, $w \in L_1L_2$.
  - If the automaton for L_2 rejects the remainder, the split was incorrect.

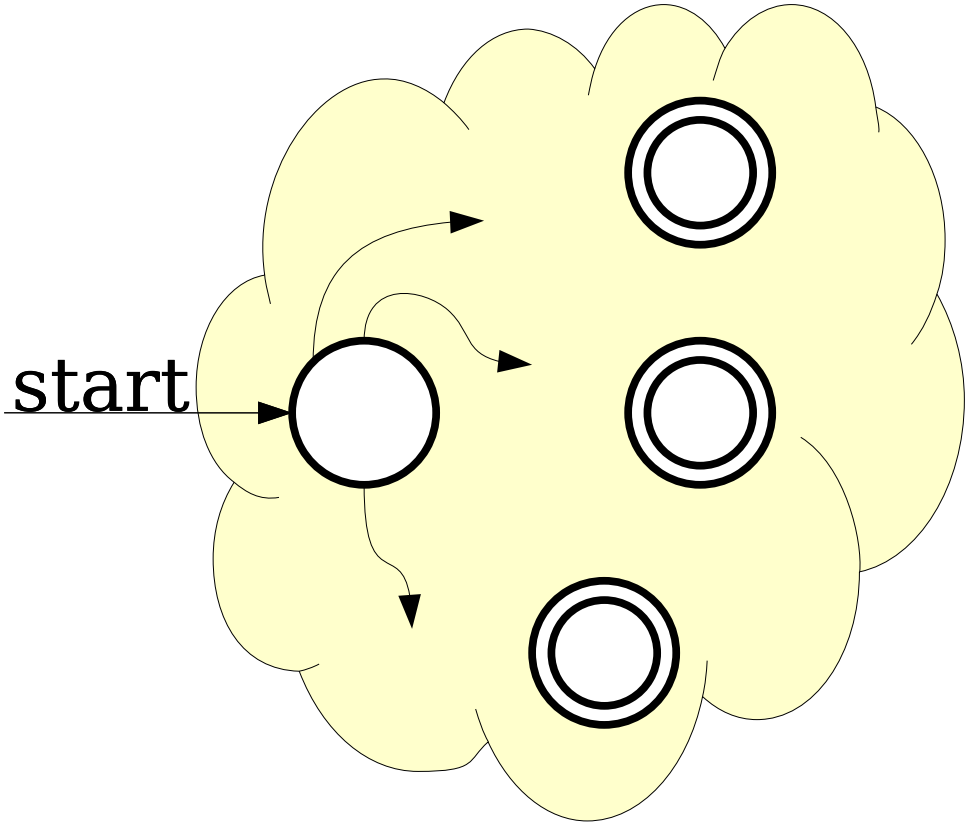
Concatenating Regular Languages

Concatenating Regular Languages

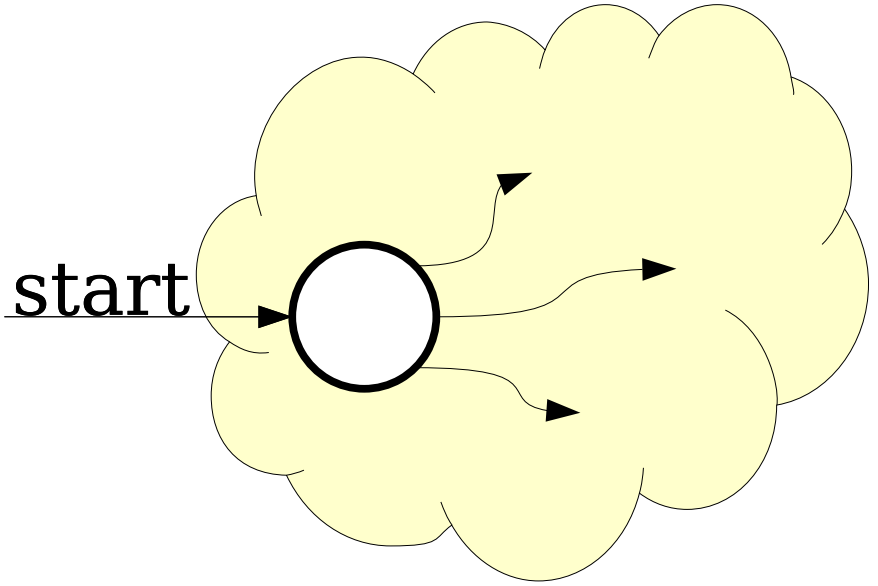


Machine for L_1

Concatenating Regular Languages

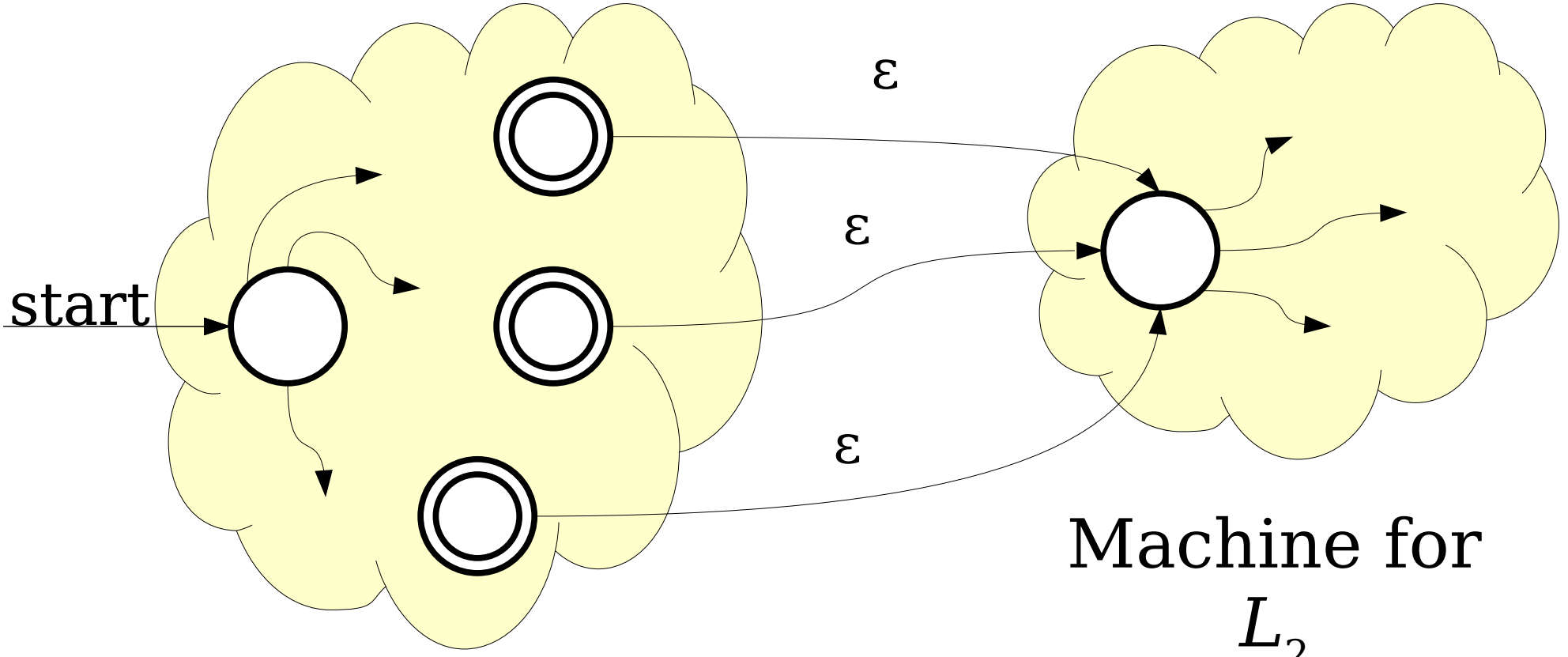


Machine for L_1

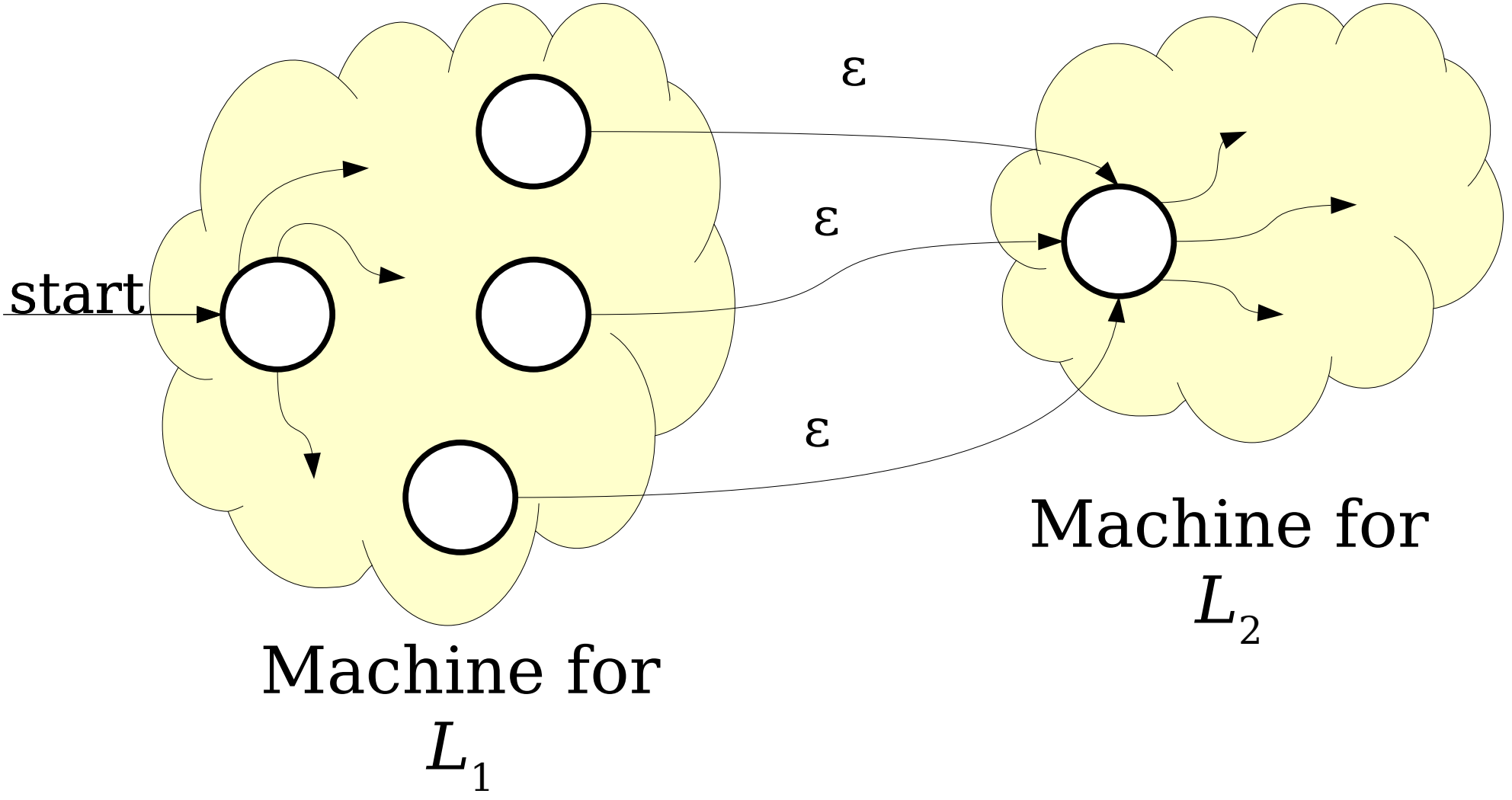


Machine for L_2

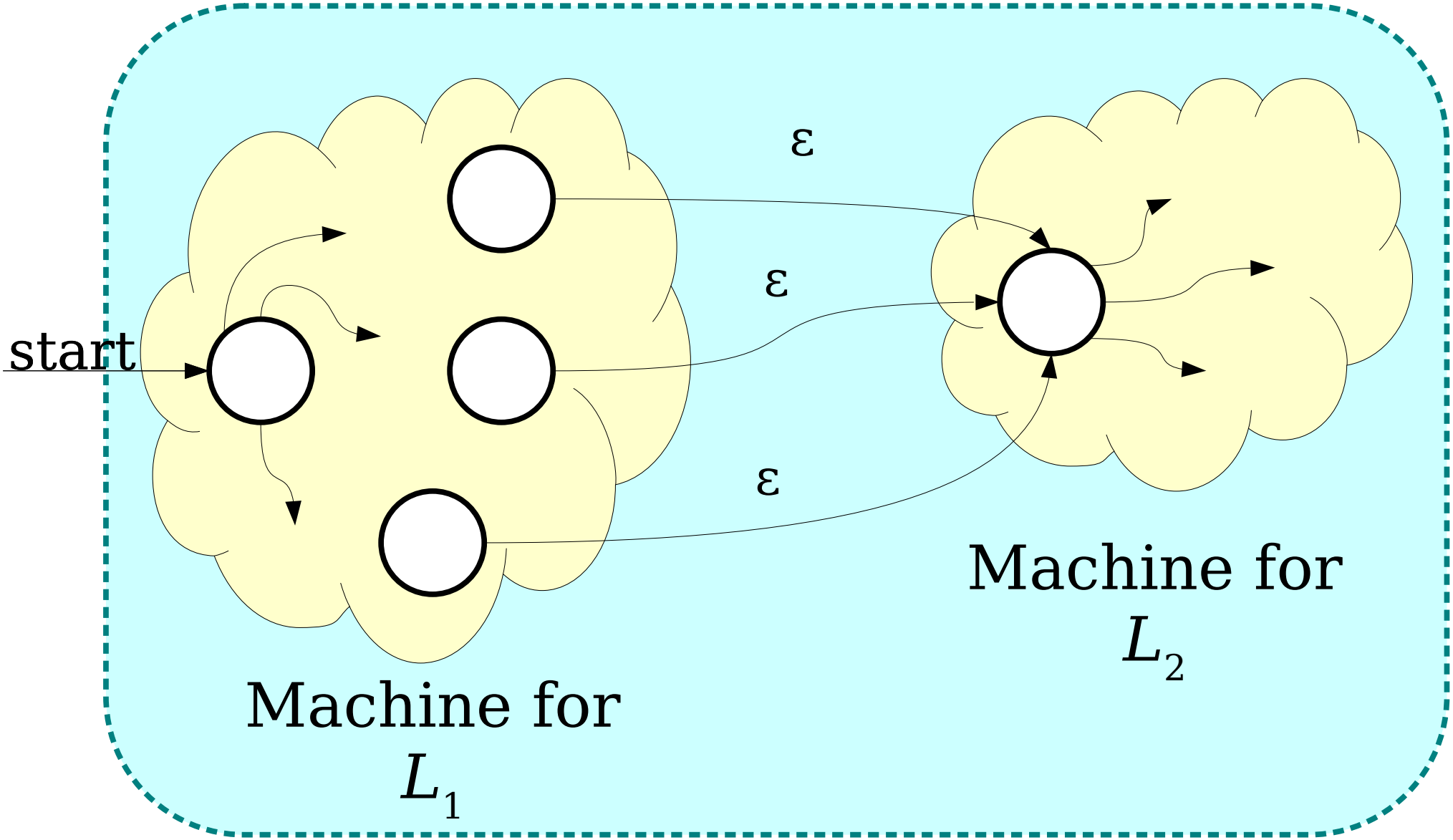
Concatenating Regular Languages



Concatenating Regular Languages



Concatenating Regular Languages



Lots and Lots of Concatenation

- Consider the language L = { **aa**, **b** }
- LL is the set of strings formed by concatenating pairs of strings in L.

{ **aaaa**, **aab**, **baa**, **bb** }

- LLL is the set of strings formed by concatenating triples of strings in L.

{ **aaaaaa**, **aaaab**, **aabaa**, **aabb**, **baaaa**, **baab**, **bbaa**, **bbb**}

- $LLLL$ is the set of strings formed by concatenating quadruples of strings in L.

{ **aaaaaaaa**, **aaaaaab**, **aaaabaa**, **aaaabb**, **aabaaaa**, **aabaab**, **aabbaa**, **aabbb**, **baaaaaa**, **baaaab**, **baabaa**, **baabb**, **bbaaaa**, **bbaab**, **bbbaa**, **bbbb**}

Language Exponentiation

- We can define what it means to “exponentiate” a language as follows:
- $L^0 = \{\varepsilon\}$
  - Intuition: The only string you can form by gluing no strings together is the empty string.
  - Notice that $\{\varepsilon\} \neq \varnothing$. Can you explain why?
- $L^{n+1} = LL^n$
  - Idea: Concatenating (n+1) strings together works by concatenating n strings, then concatenating one more.
- ***Question to ponder:*** Why define $L^0 = \{\varepsilon\}$?
- ***Question to ponder:*** What is $\varnothing^0$?

The Kleene Star

The Kleene Closure

- An important operation on languages is the ***Kleene Closure***, which is defined as

$$L^* = \{ w \in \Sigma^* \mid \exists n \in \mathbb{N}.\ w \in L^n \}$$

- Mathematically:

$$w \in L^* \quad \leftrightarrow \quad \exists n \in \mathbb{N}.\ w \in L^n$$

- Intuitively, L^* is the language all possible ways of concatenating zero or more strings in L together, possibly with repetition.
- ***Question to ponder:*** What is Ø*?

The Kleene Closure

If L = { **a**, **bb** }, then L^* = {

ε,

a, **bb**,

aa, **abb**, **bba**, **bbbb**,

aaa, **aabb**, **abba**, **abbbb**, **bbaa**, **bbabb**, **bbbba**, **bbbbbb**,

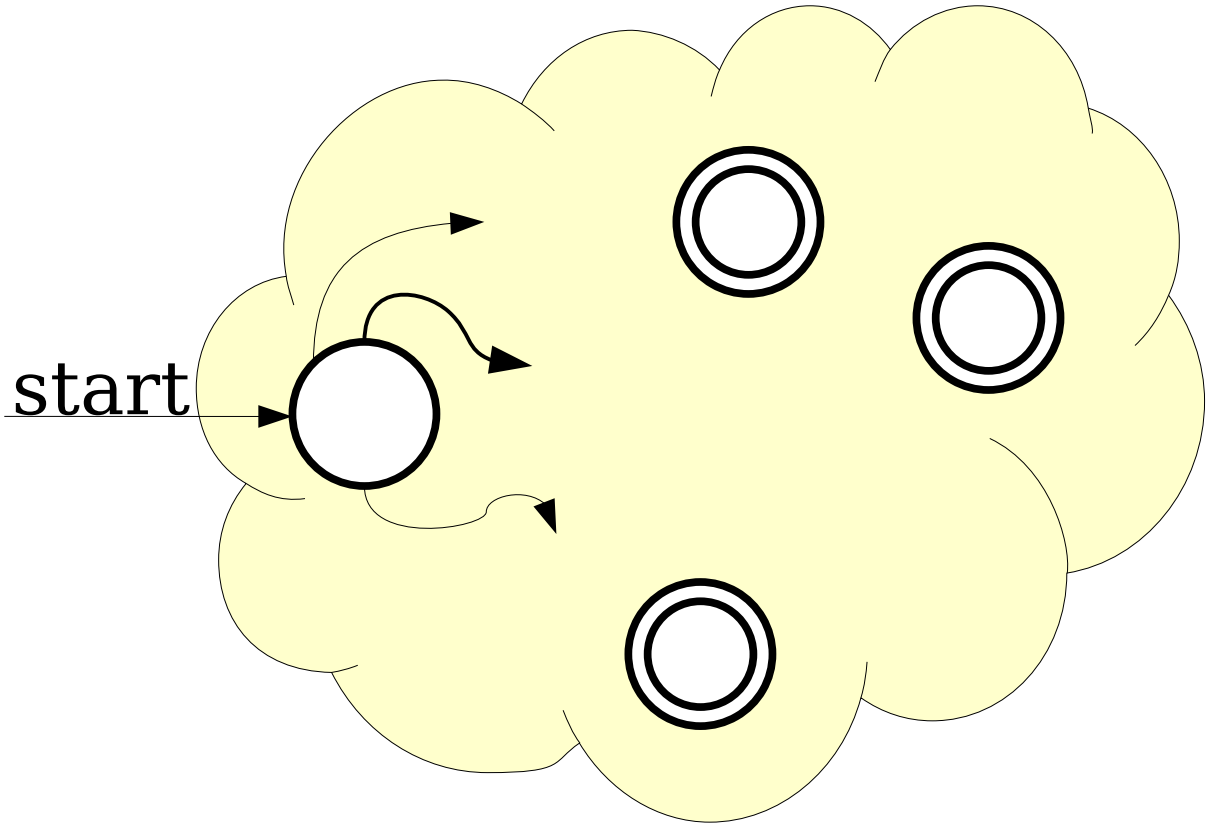
…

}

Think of L^* as the set of strings you can make if you have a collection of stamps – one for each string in L – and you form every possible string that can be made from those stamps.

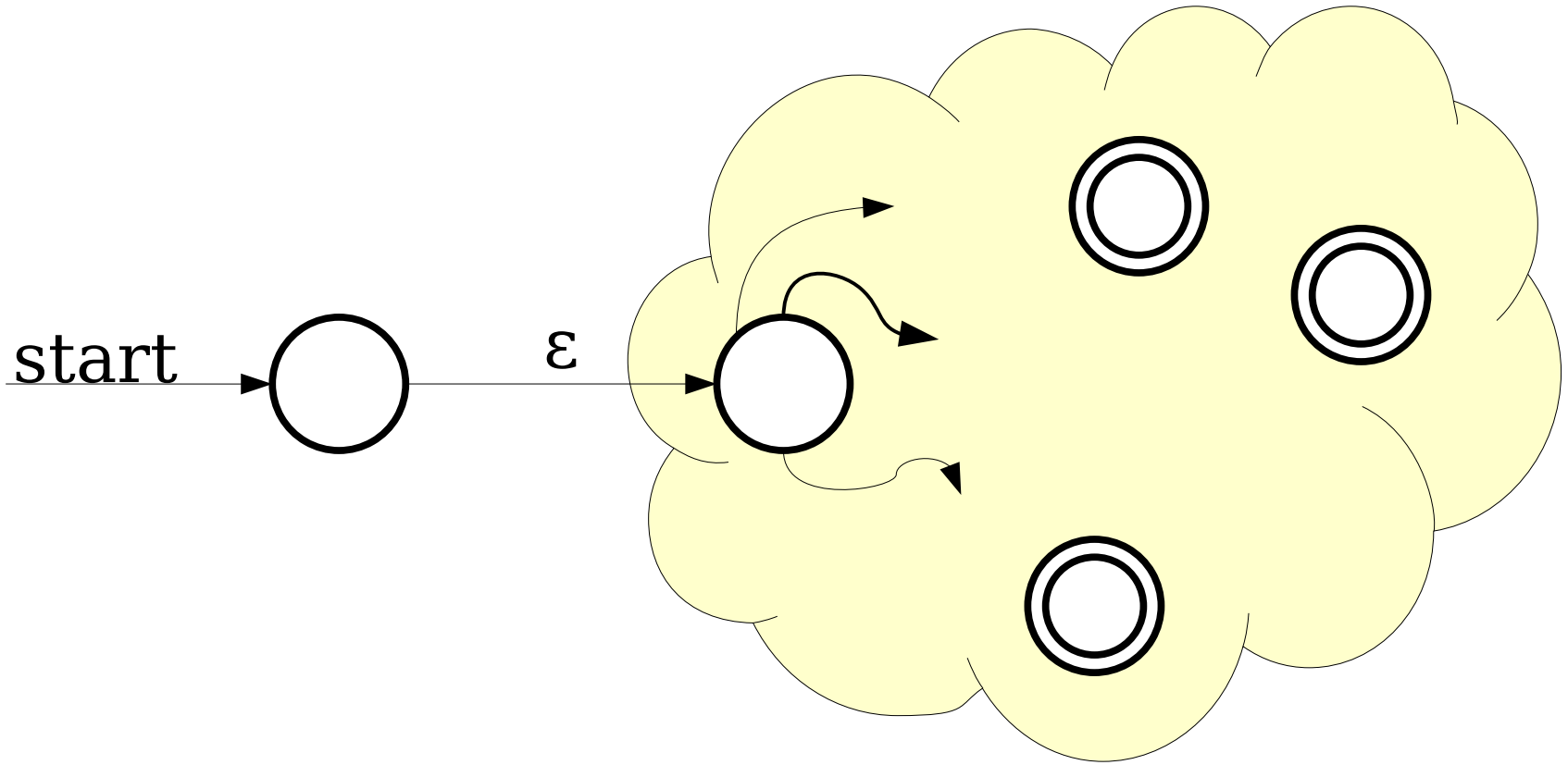
Idea: Can we convert an NFA for language L to an NFA for language L*?

The Kleene Star

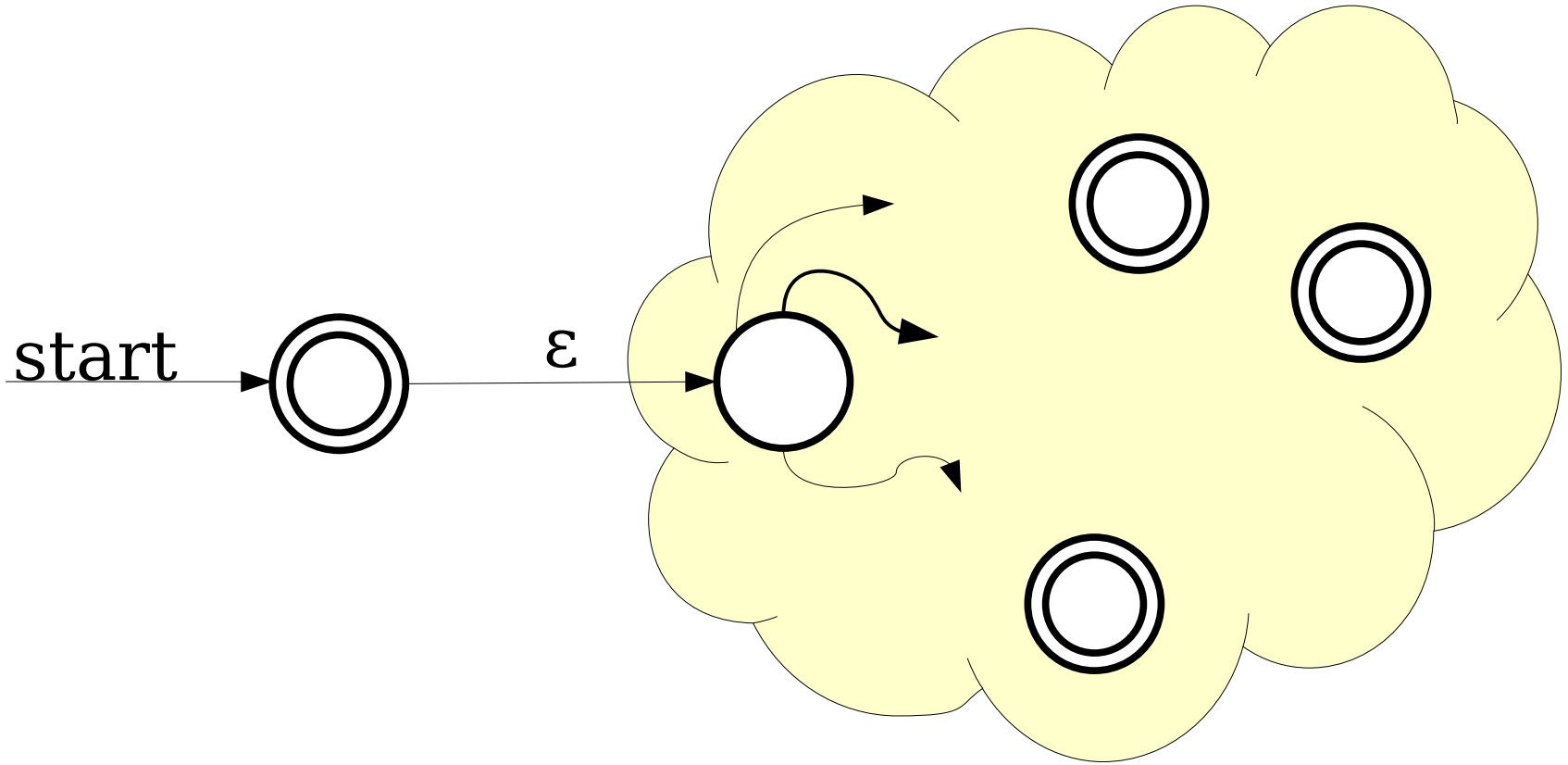


Machine for L

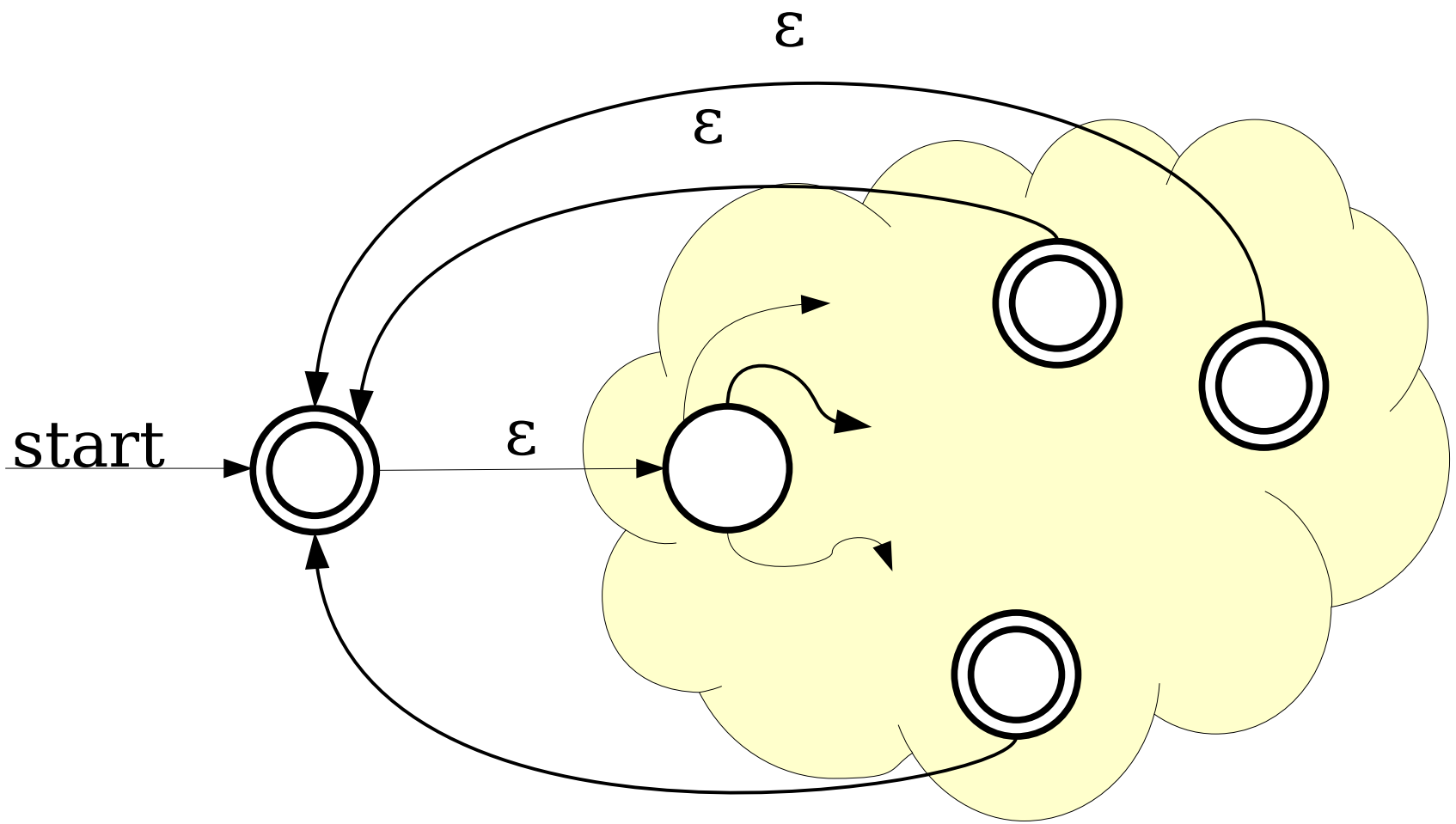
The Kleene Star



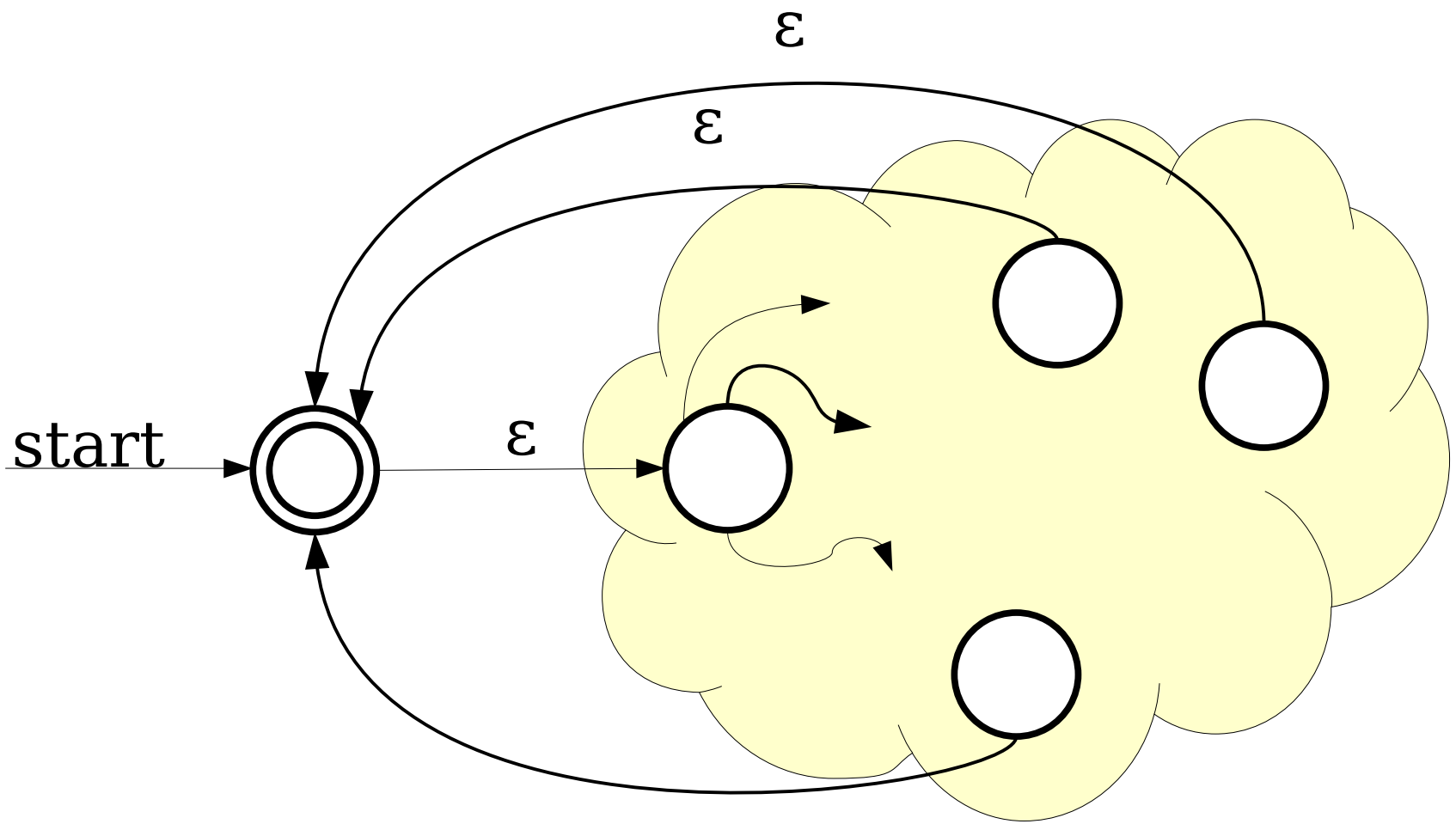
The Kleene Star



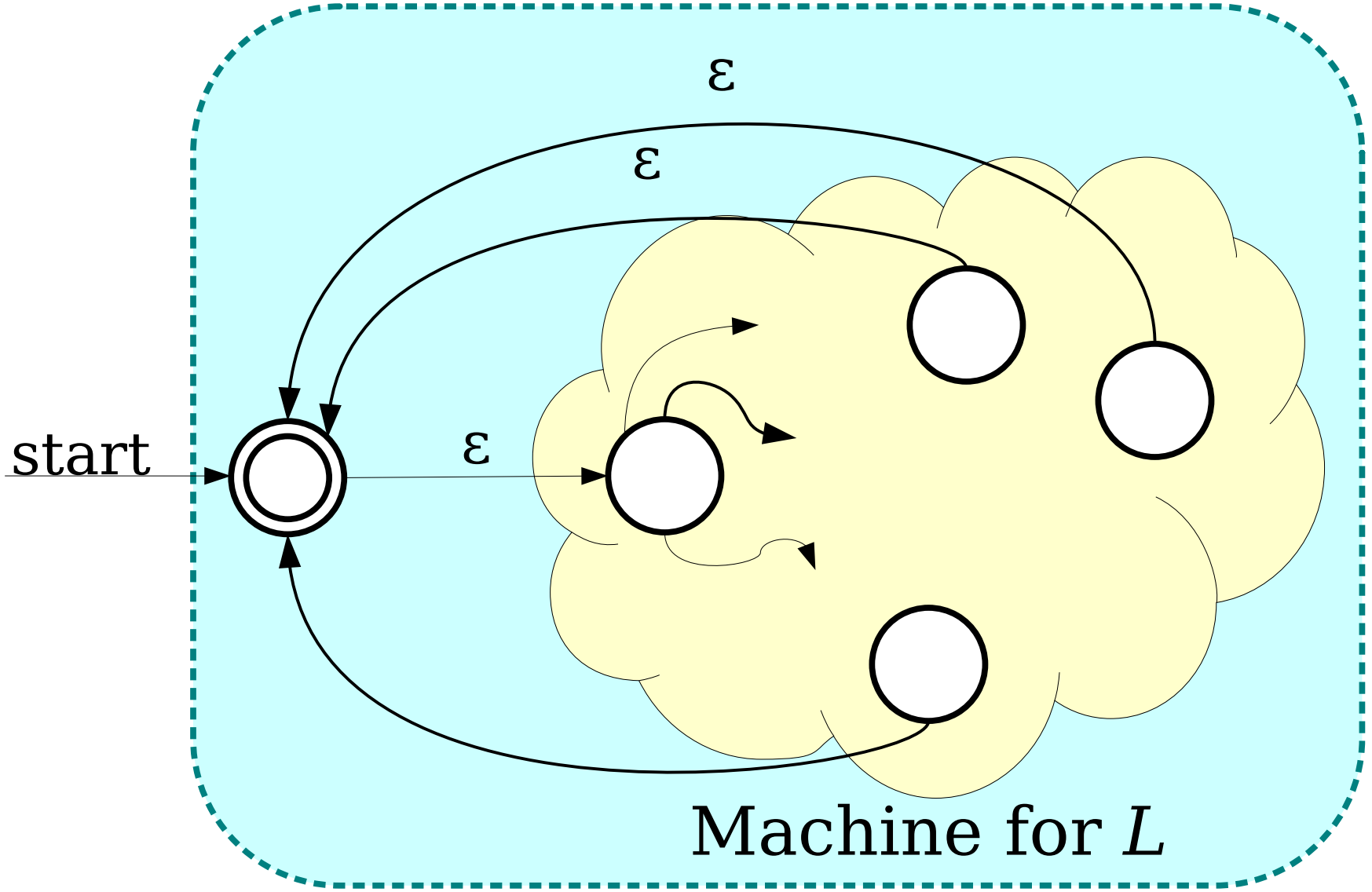
The Kleene Star



The Kleene Star

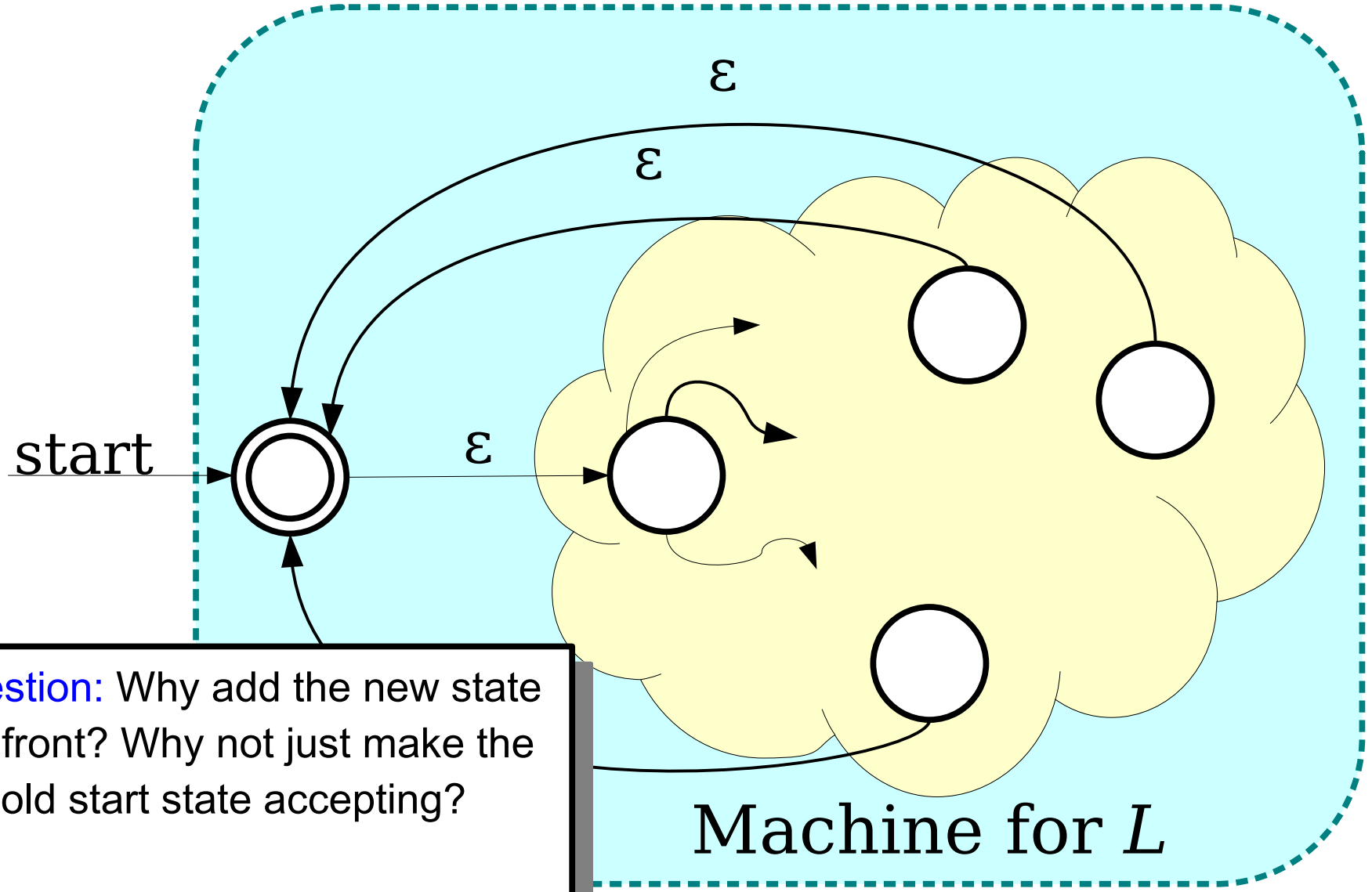


The Kleene Star



Machine for L^*

The Kleene Star



Question: Why add the new state out front? Why not just make the old start state accepting?

achine for L^*

Closure Properties

- ***Theorem:*** If L_1 and L_2 are regular languages over an alphabet Σ, then so are the following languages:
  - $\overline{L}_1$
  - $L_1 \cup L_2$
  - $L_1 \cap L_2$
  - $L_1 L_2$
  - L_1^*
- These properties are called ***closure properties of the regular languages***.

Next Time

- ***Regular Expressions***
  - Building languages from the ground up!
- ***Thompson's Algorithm***
  - A UNIX Programmer in Theoryland.
- ***Kleene's Theorem***
  - From machines to programs!