Finite Automata

Part Two

Recap from Last Time

DFAs

A ***DFA*** is a

Deterministic

Finite

Automaton

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.

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

Nondeterministic

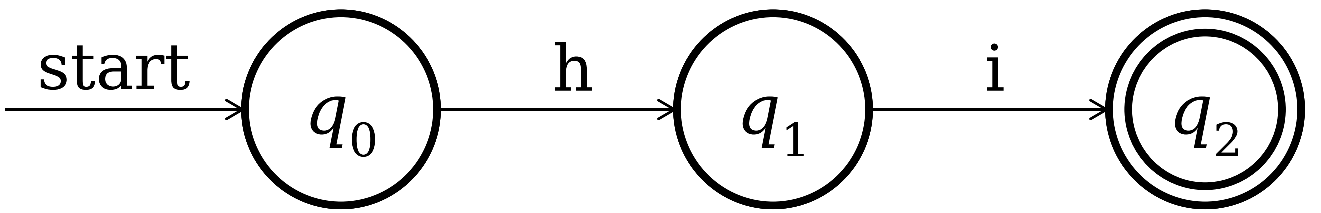
Finite

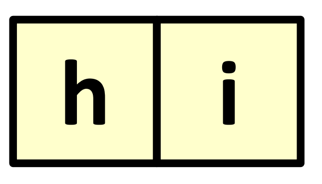
Automaton

Can have missing transitions or multiple transitions defined on the same input symbol.

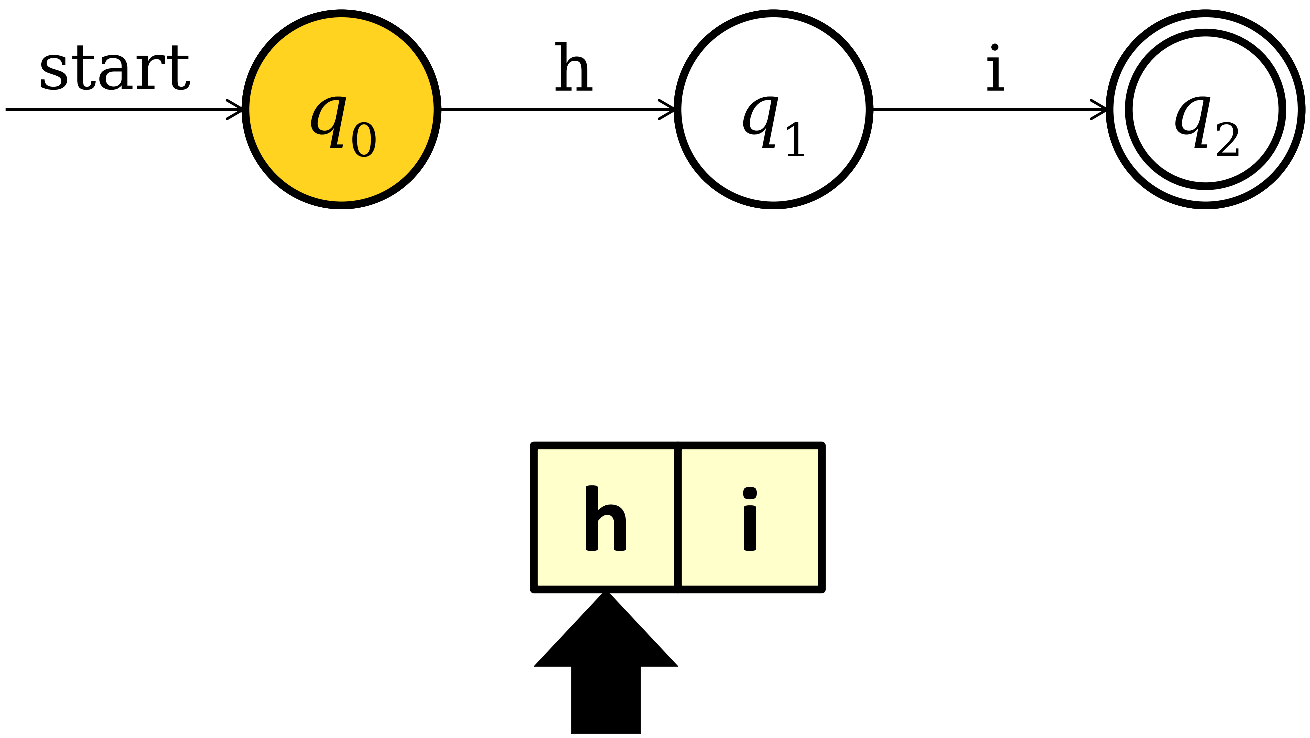
Accepts if *any possible series of choices* leads to an accepting state.

Hello, NFA!

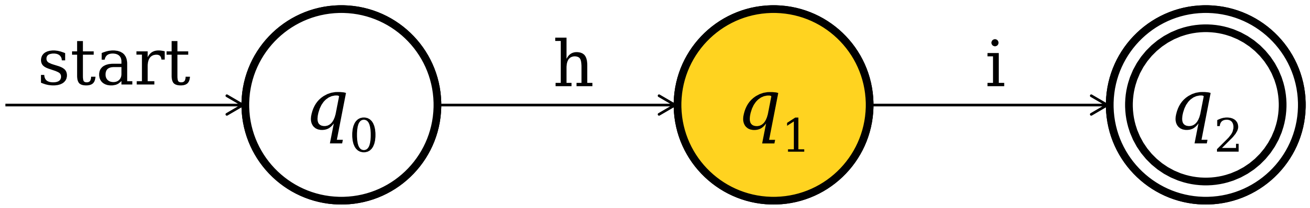


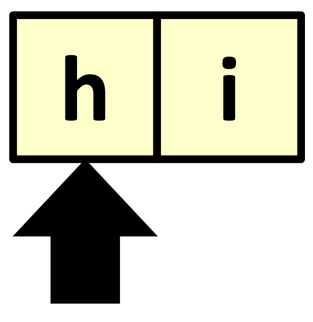


Hello, NFA!

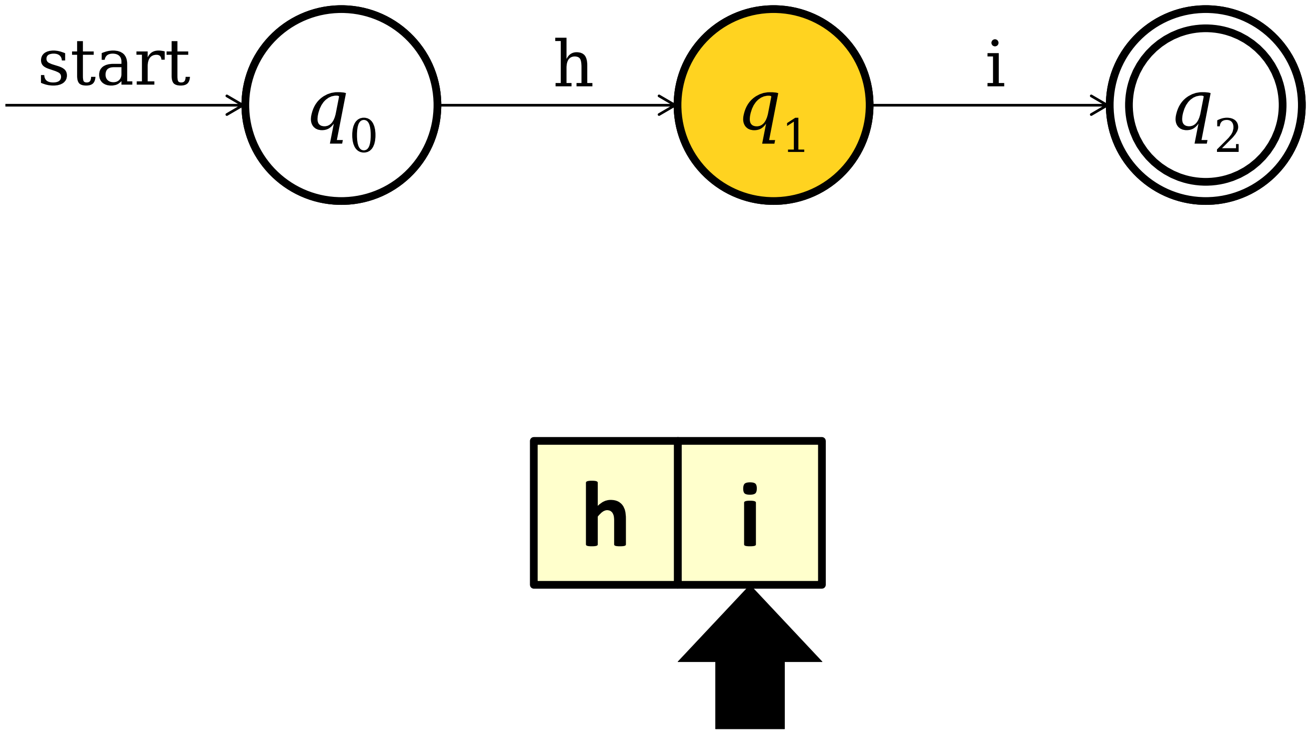


Hello, NFA!

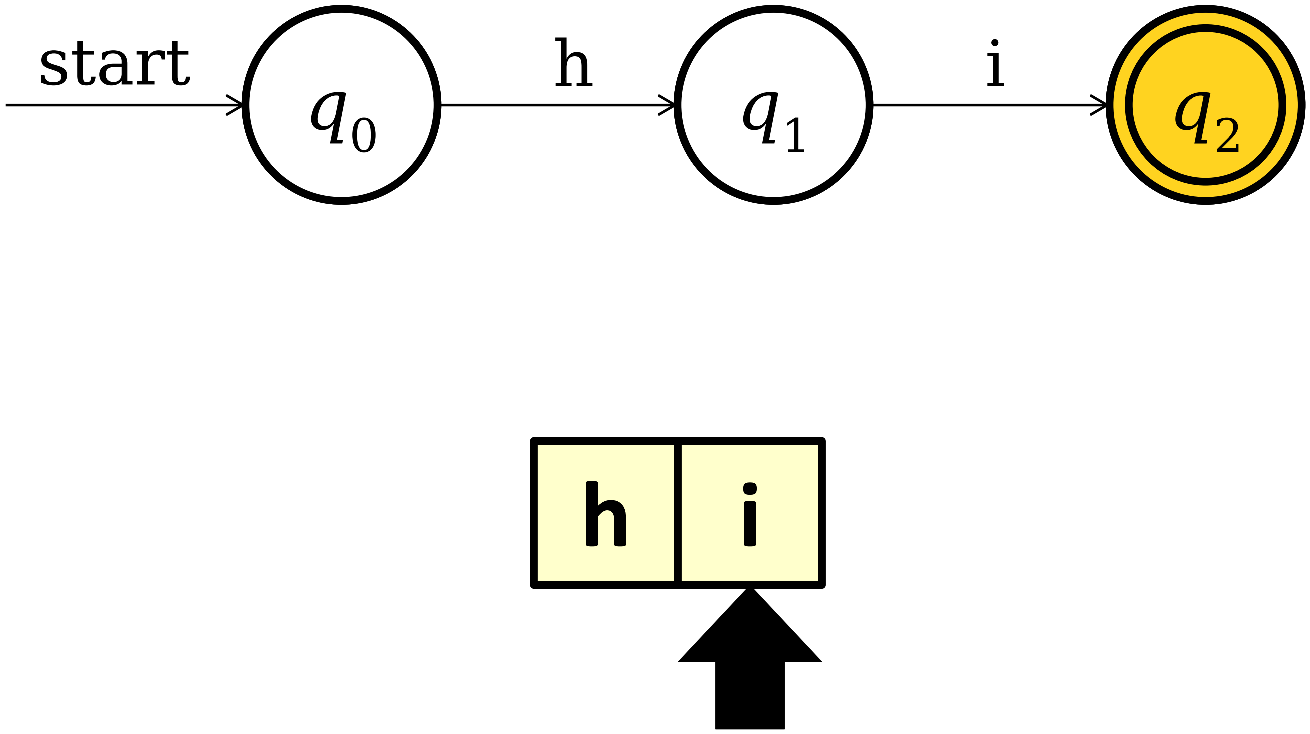




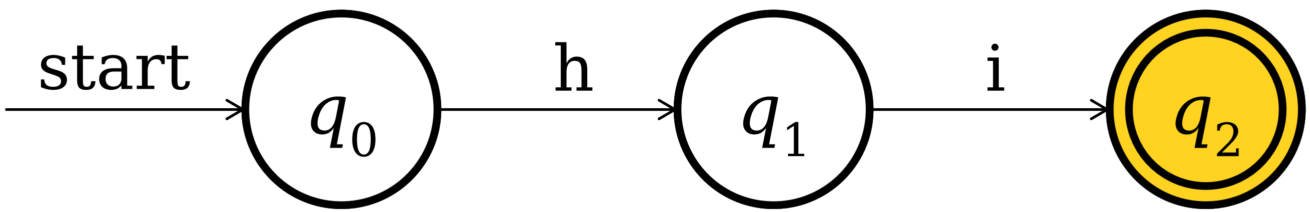
Hello, NFA!



Hello, NFA!

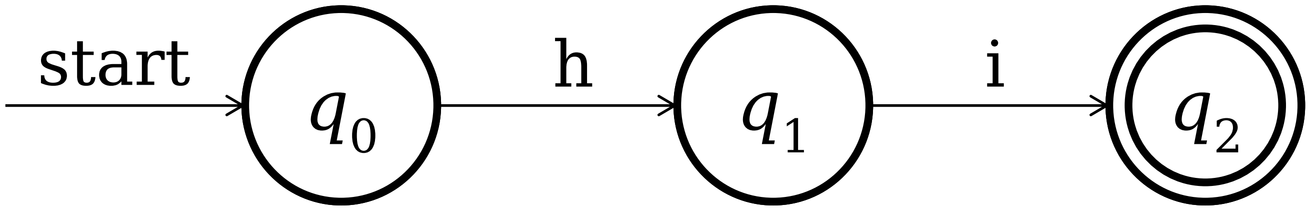


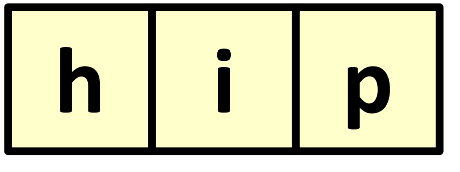
Hello, NFA!



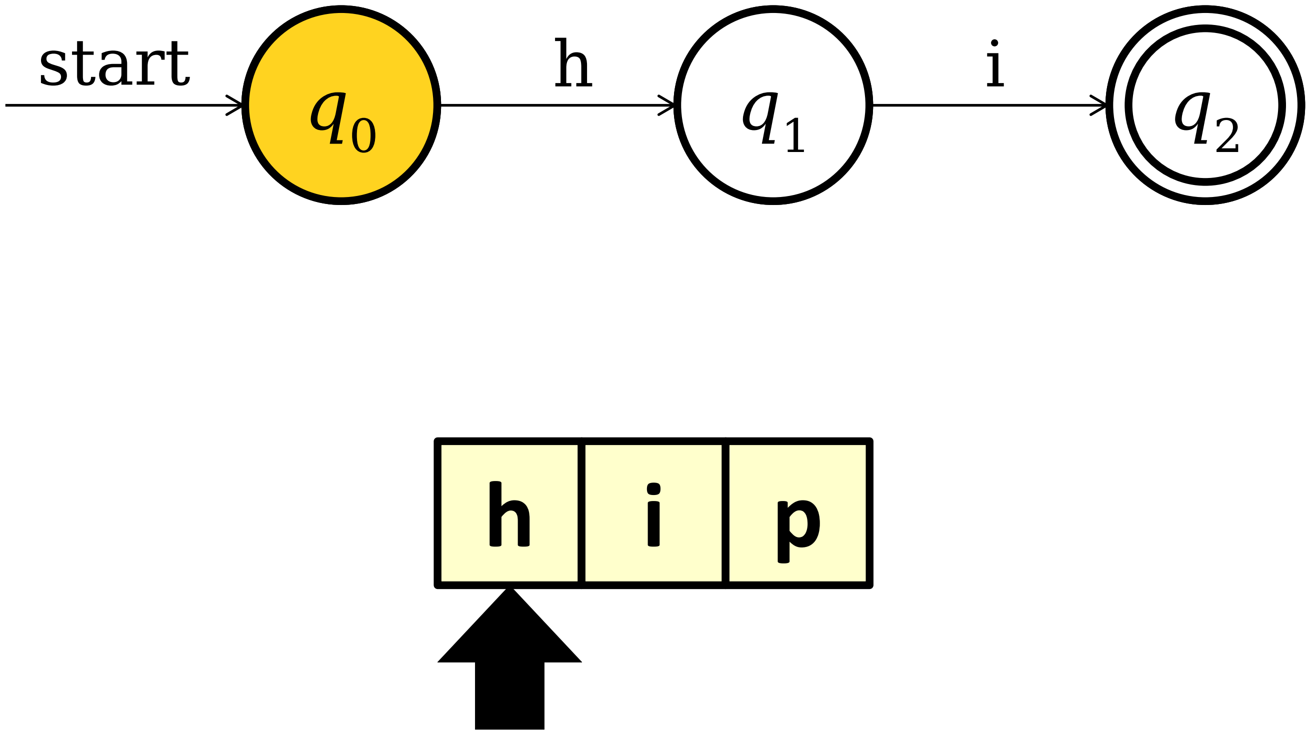
h	i

Tragedy in Paradise

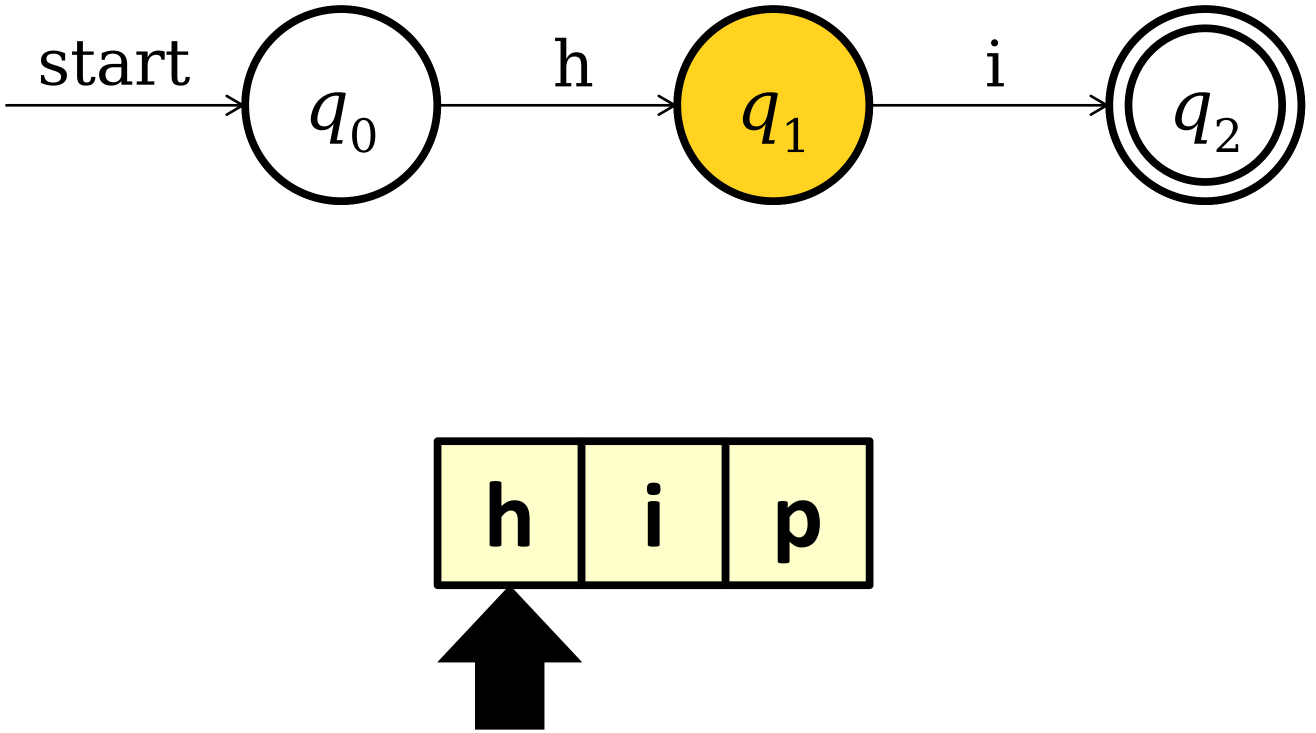




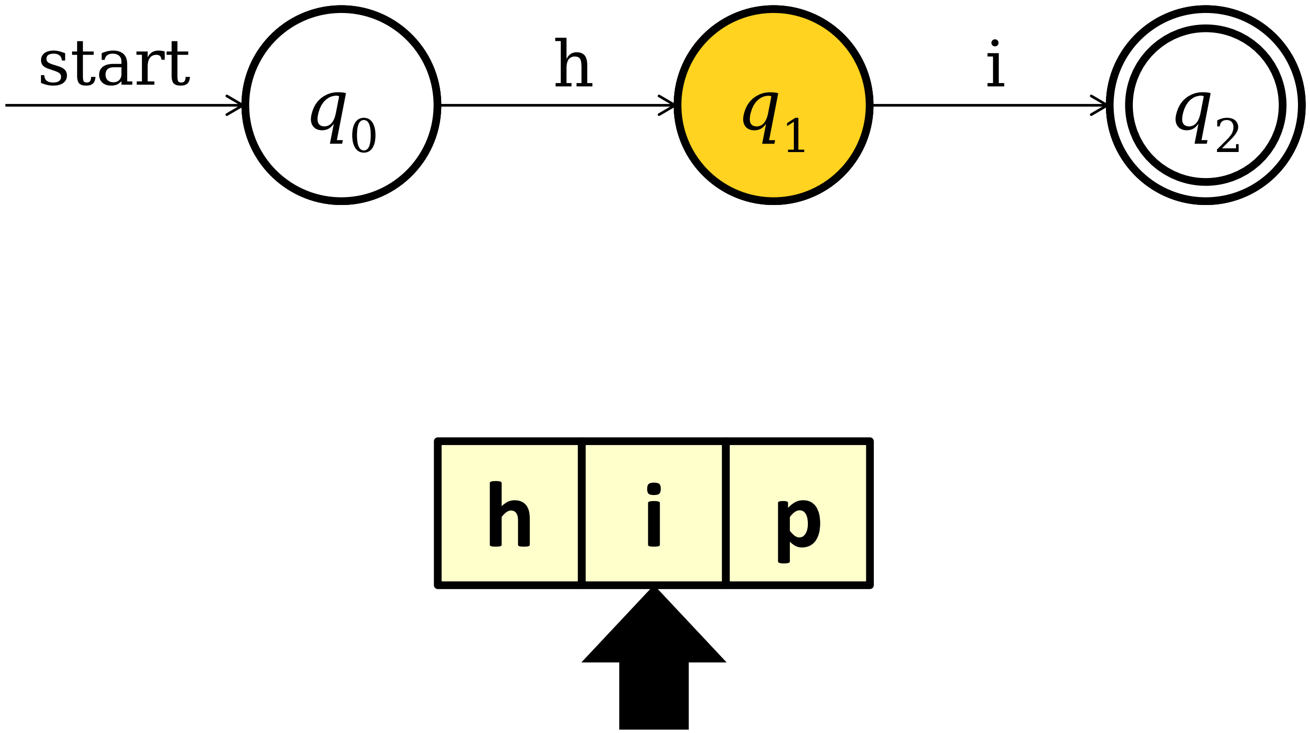
Tragedy in Paradise



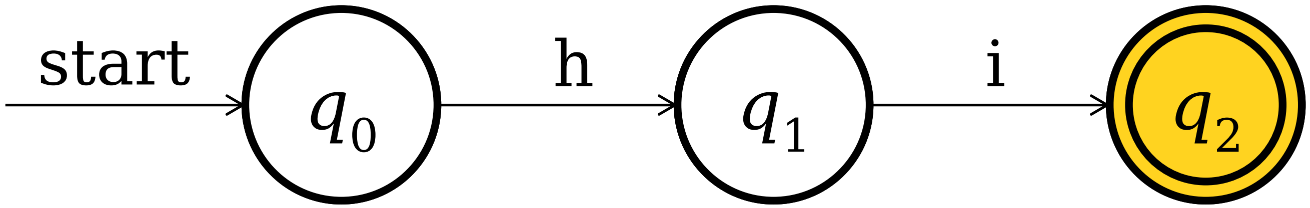
Tragedy in Paradise

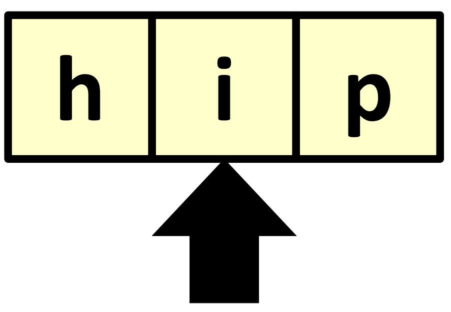


Tragedy in Paradise

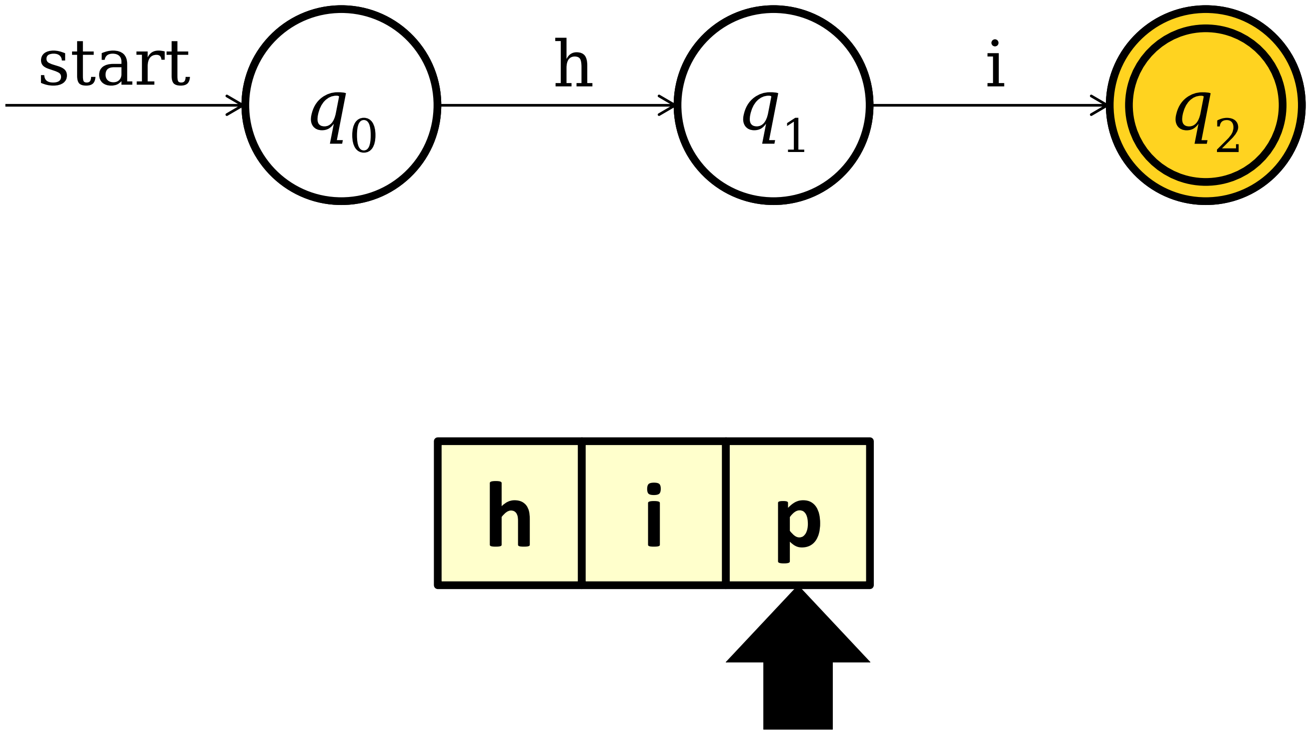


Tragedy in Paradise

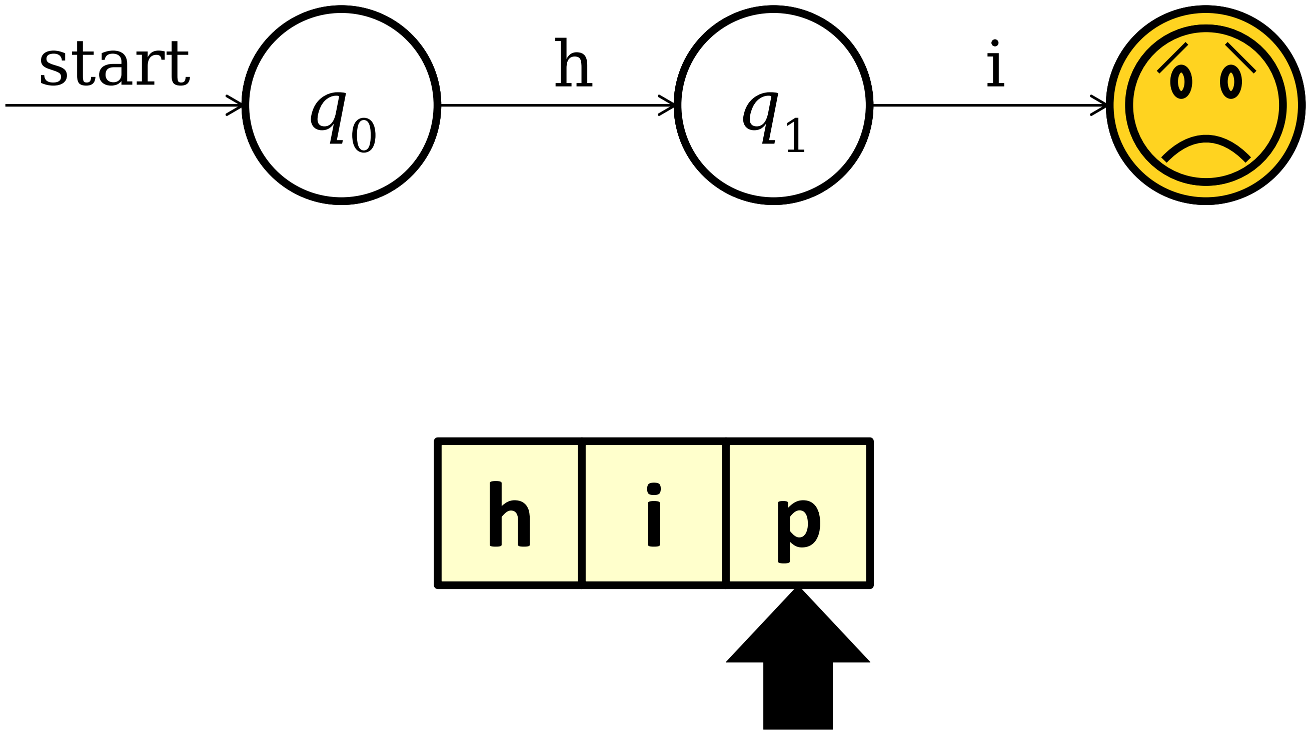




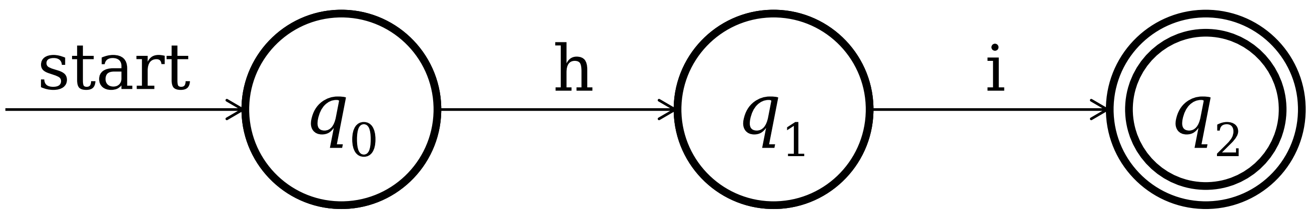
Tragedy in Paradise

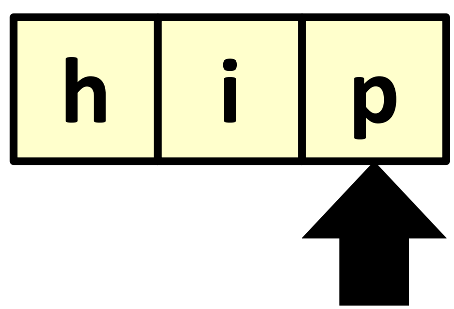


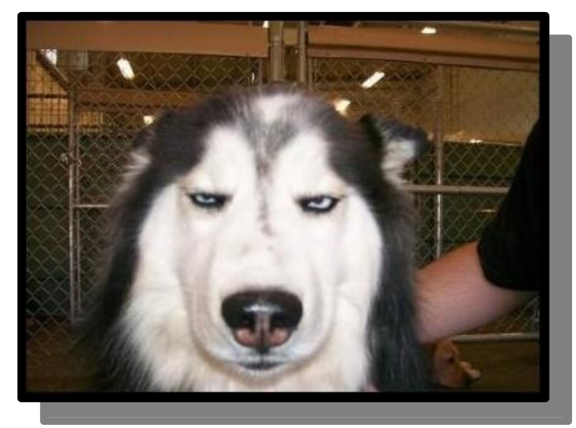
Tragedy in Paradise



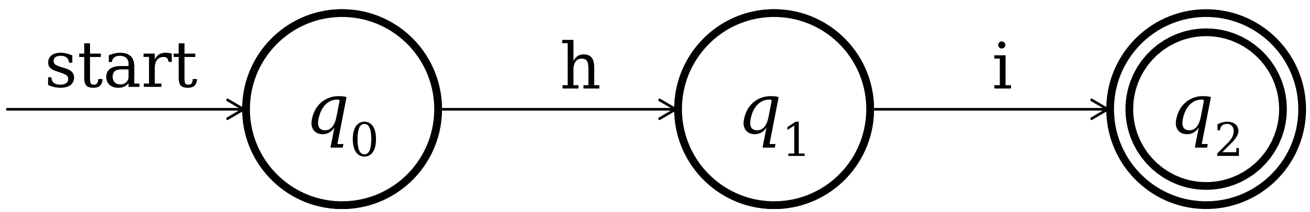
Tragedy in Paradise







NFA Languages



The ***language of an NFA*** is

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

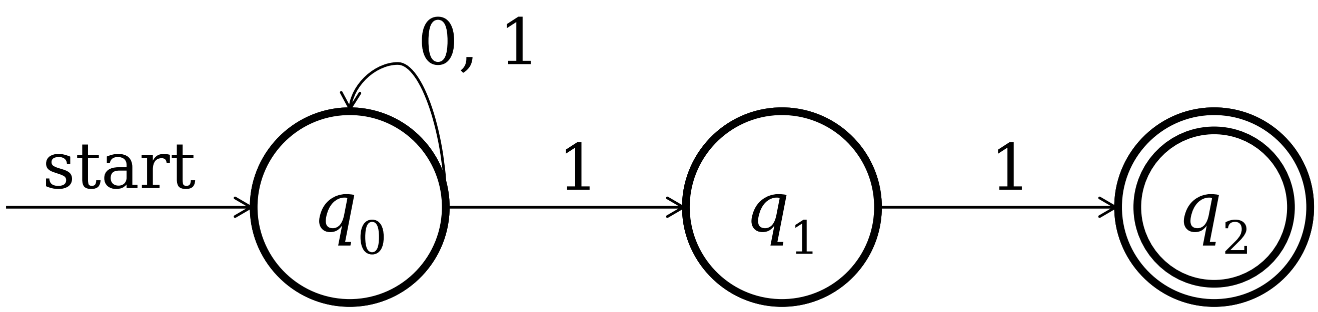
What is the language of this NFA?
(Assume $\Sigma = \{$**h**, **i**$\}$.)

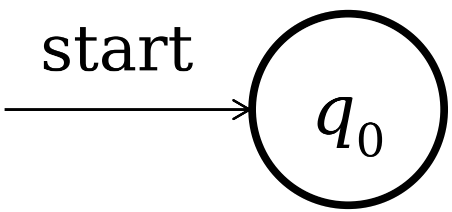
NFA Languages

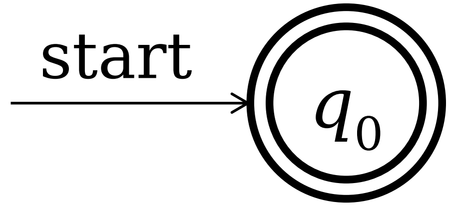
The ***language of an NFA*** is

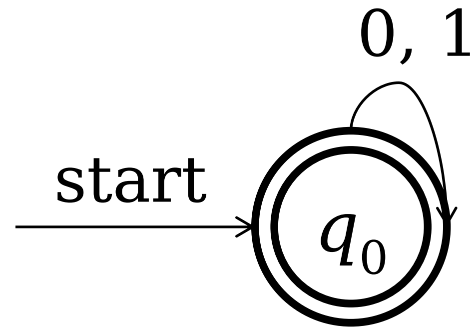
$$\mathscr{L}(N) = \{\, w \in \Sigma^* \mid N \text{ accepts } w \,\}.$$

$\Sigma = \{\mathbf{0}, \mathbf{1}\}$









ε-Transitions

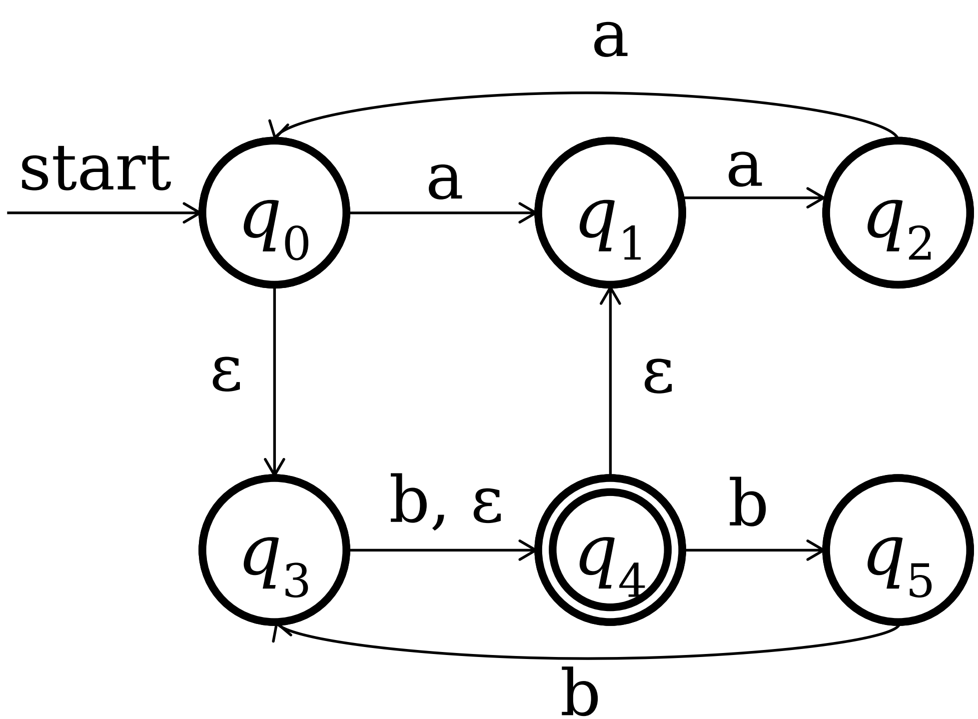
NFAs have a special type of transition called the **ε-transition**.

An NFA may follow any number of ε-transitions at any time without consuming any input.

ε-Transitions

NFAs have a special type of transition called the **ε-transition**.

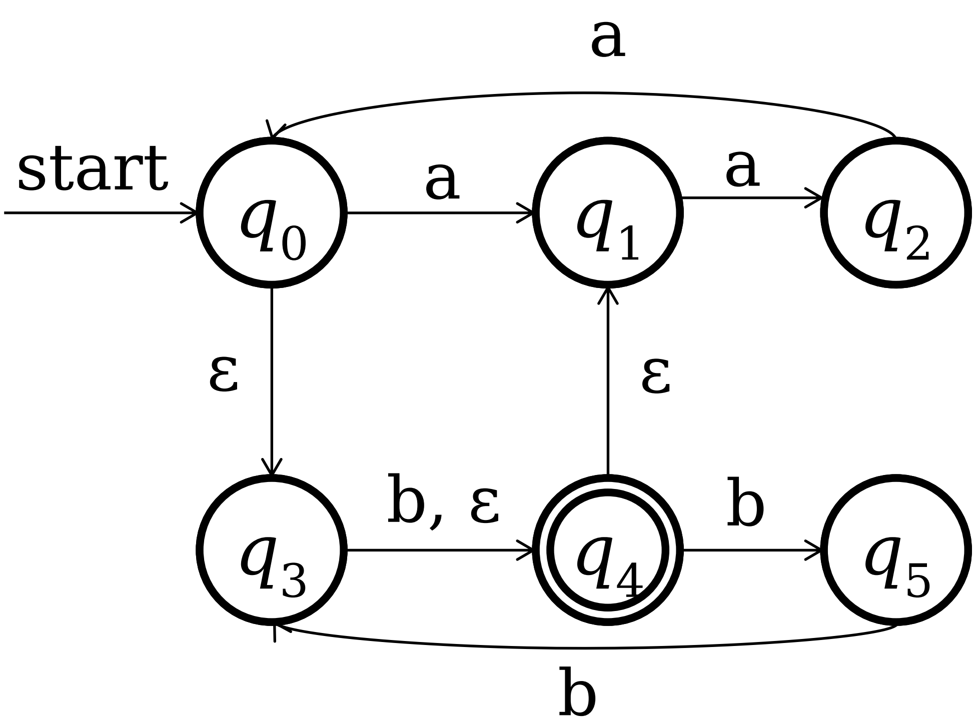
An NFA may follow any number of ε-transitions at any time without consuming any input.

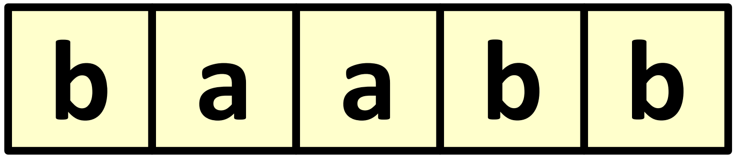


ε-Transitions

NFAs have a special type of transition called the **ε-transition**.

An NFA may follow any number of ε-transitions at any time without consuming any input.

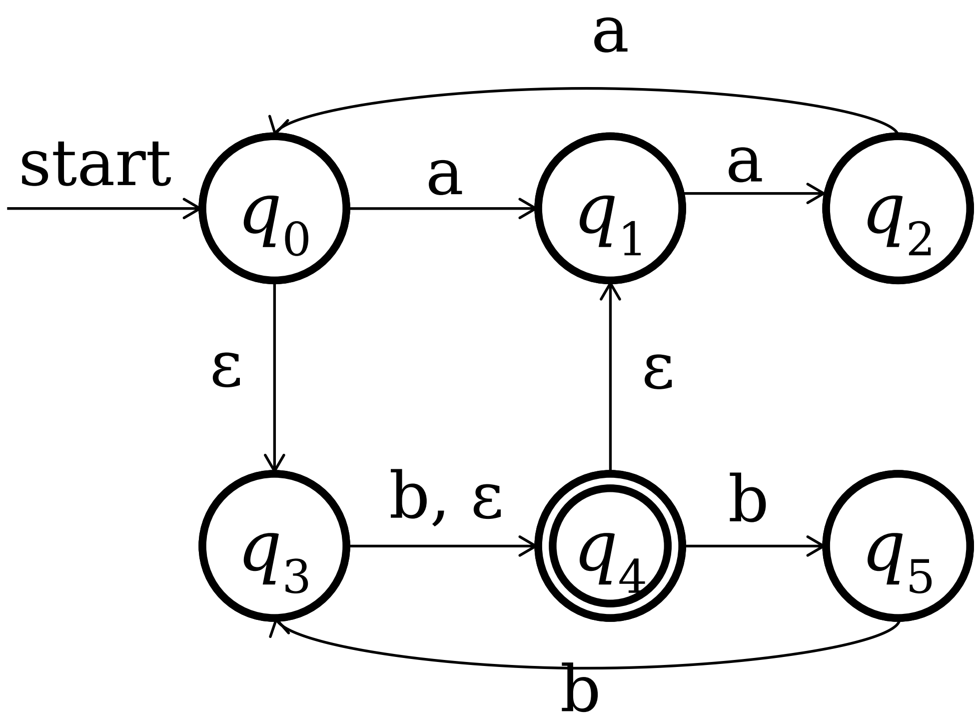


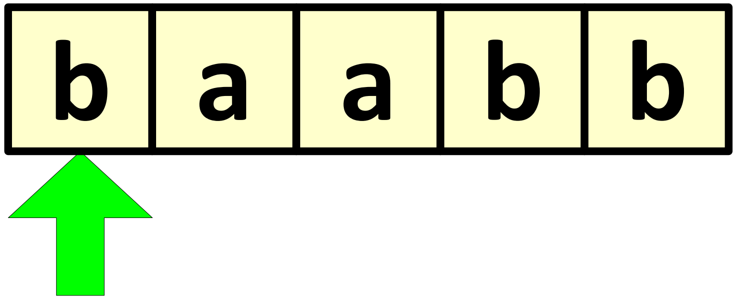


ε-Transitions

NFAs have a special type of transition called the **ε-transition**.

An NFA may follow any number of ε-transitions at any time without consuming any input.

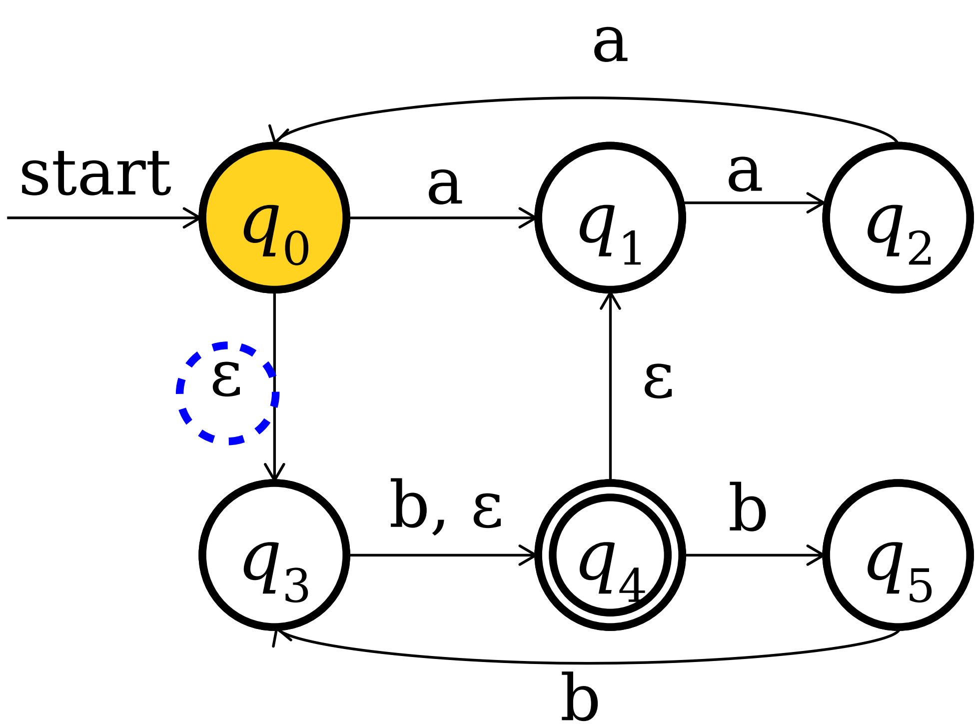


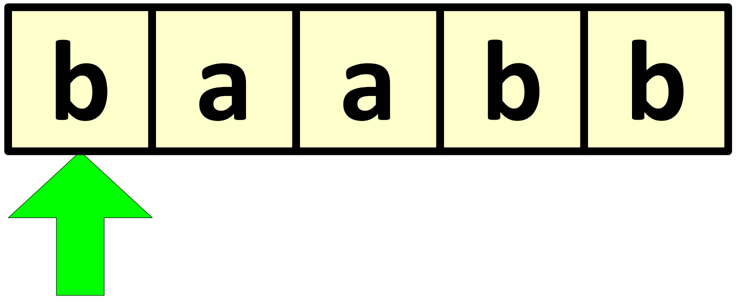


ε-Transitions

NFAs have a special type of transition called the **ε-transition**.

An NFA may follow any number of ε-transitions at any time without consuming any input.

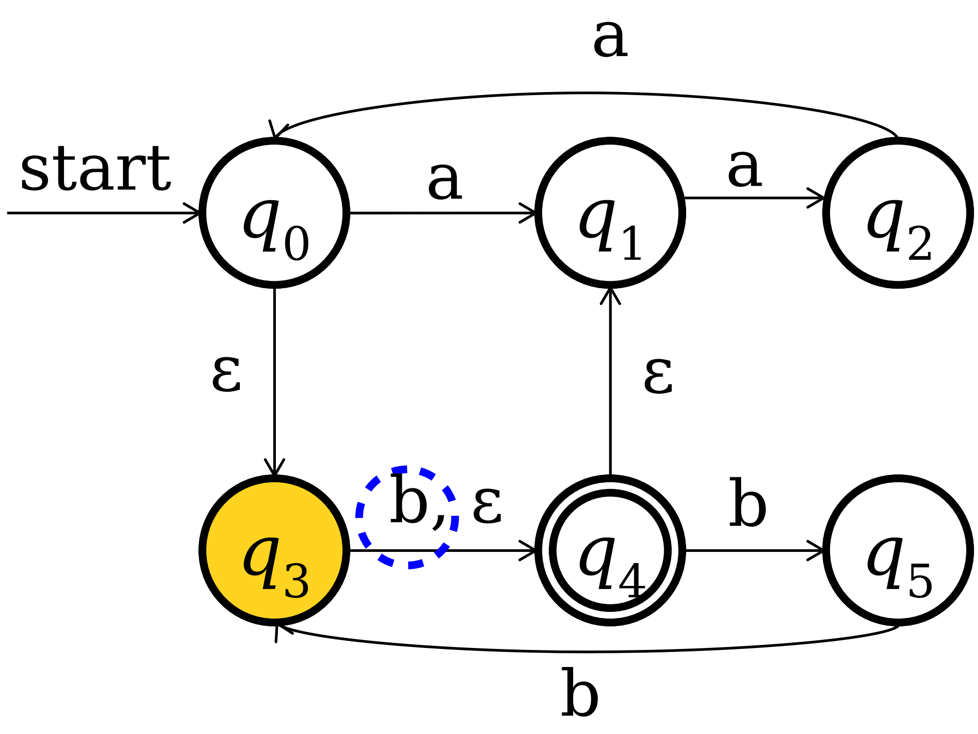


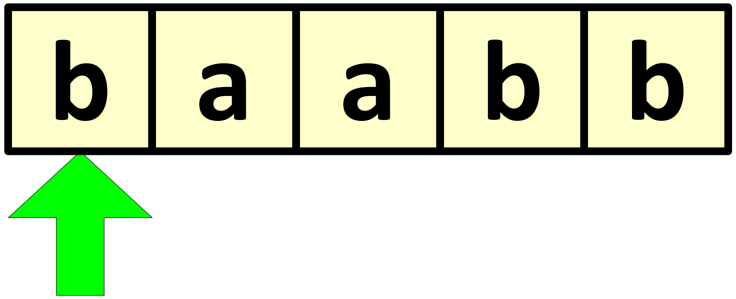


ε-Transitions

NFAs have a special type of transition called the **ε-transition**.

An NFA may follow any number of ε-transitions at any time without consuming any input.

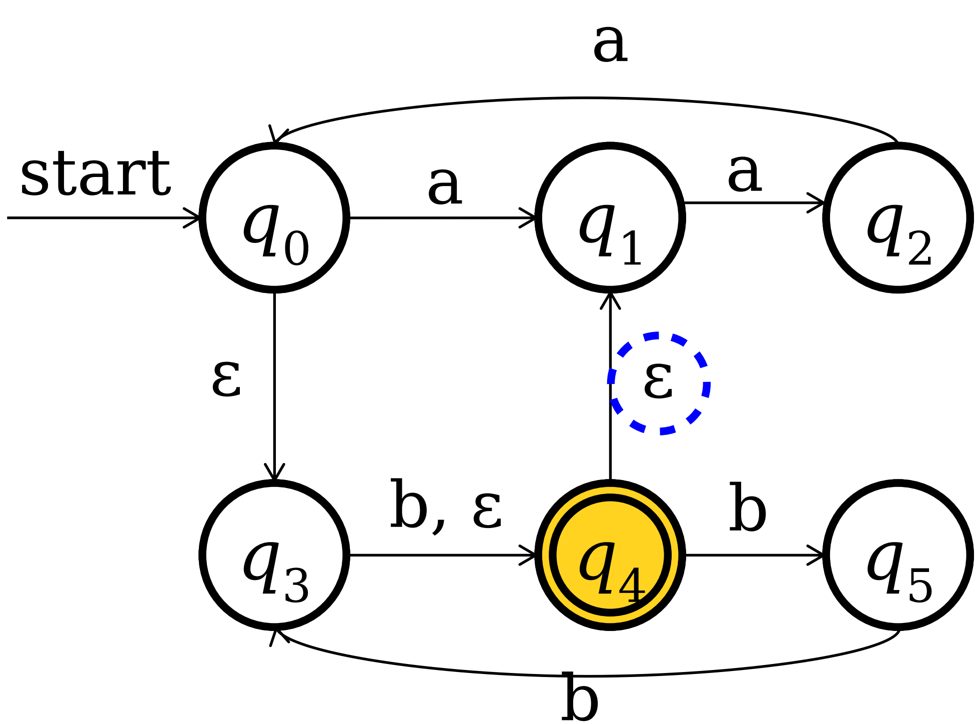


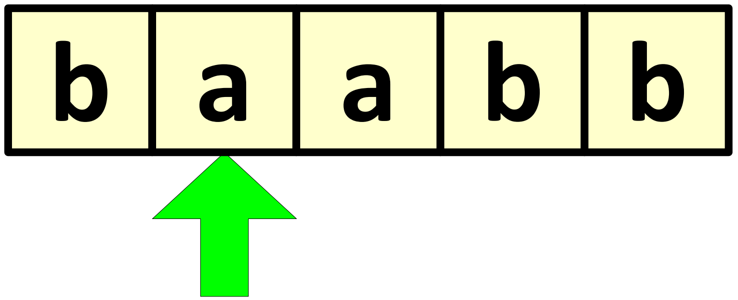


ε-Transitions

NFAs have a special type of transition called the **ε-transition**.

An NFA may follow any number of ε-transitions at any time without consuming any input.

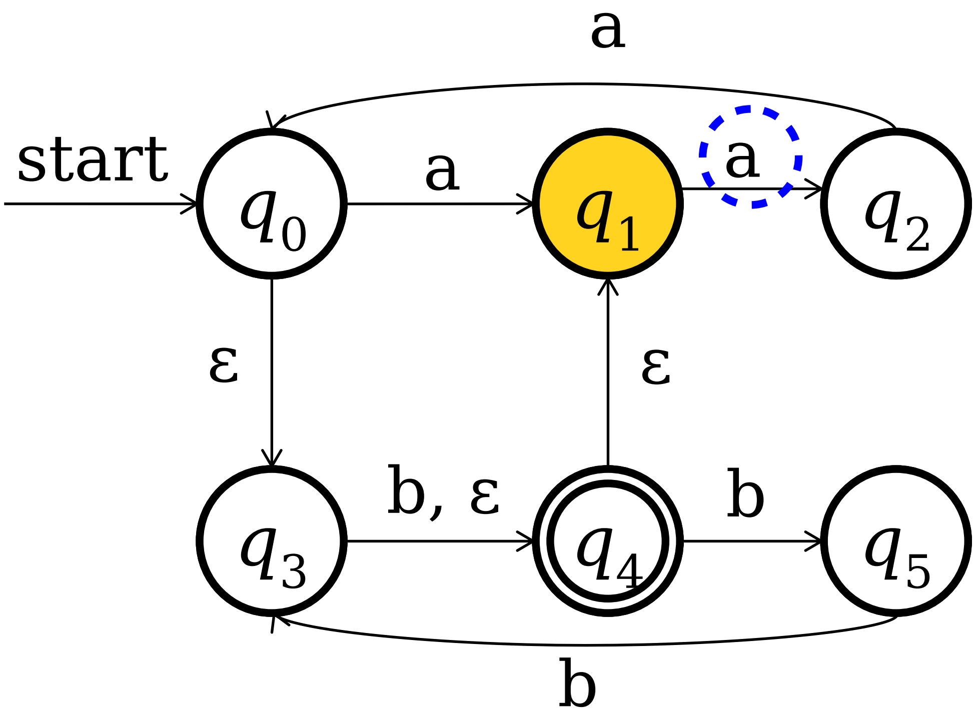


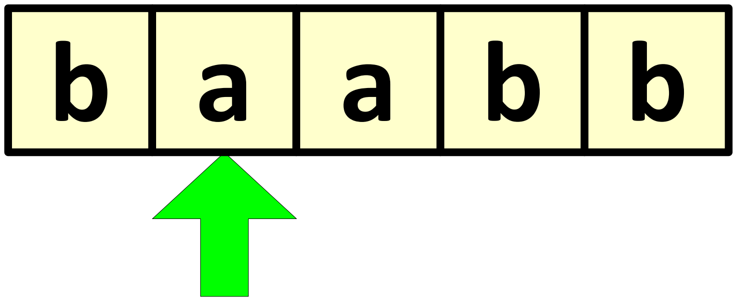


ε-Transitions

NFAs have a special type of transition called the **ε-transition**.

An NFA may follow any number of ε-transitions at any time without consuming any input.

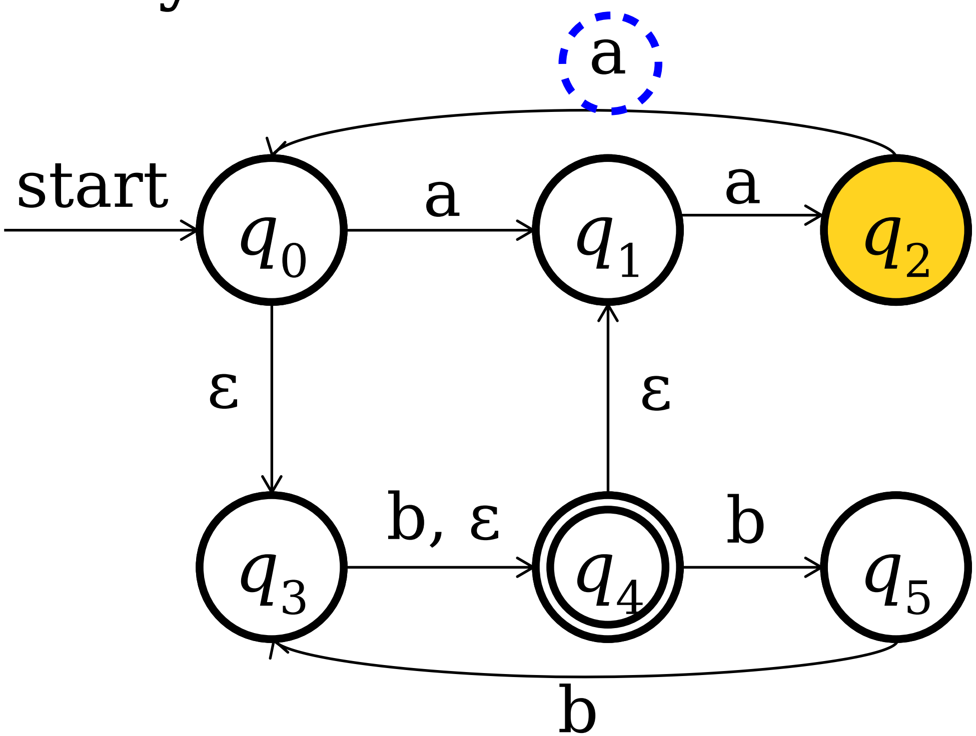


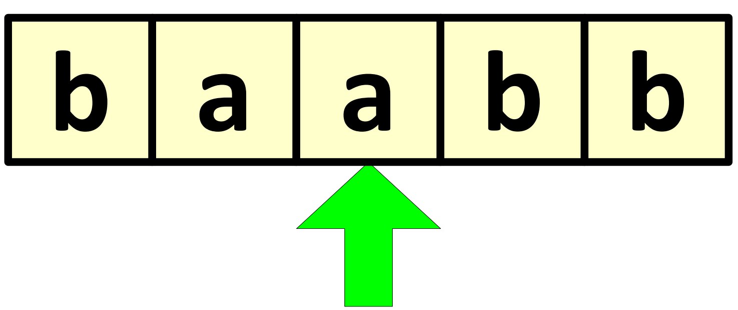


ε-Transitions

NFAs have a special type of transition called the **ε-transition**.

An NFA may follow any number of ε-transitions at any time without consuming any input.

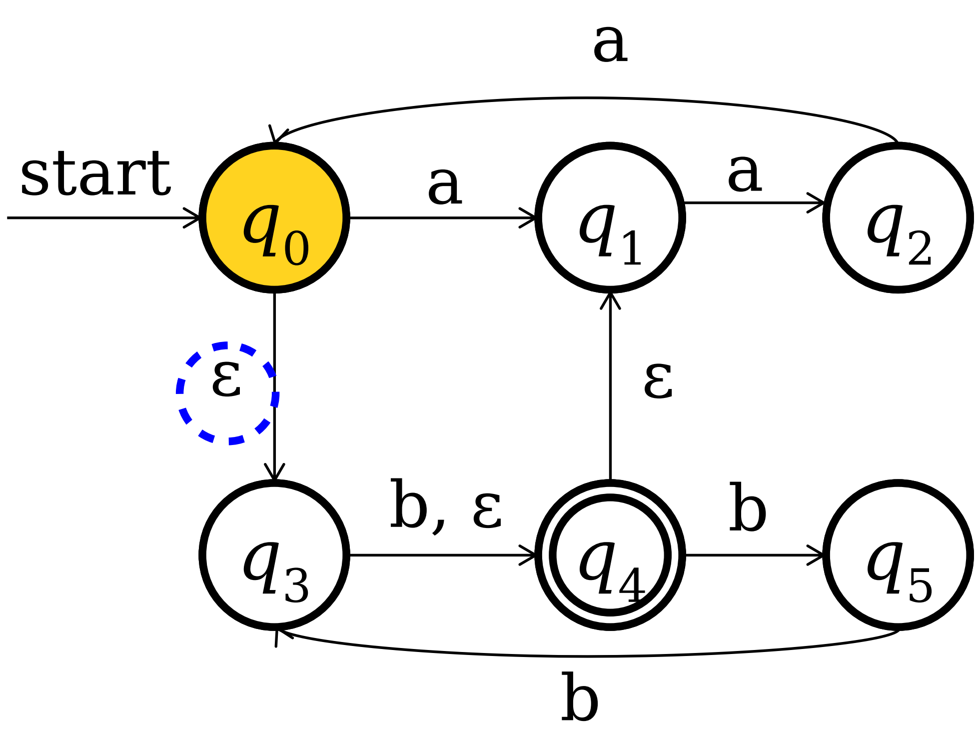


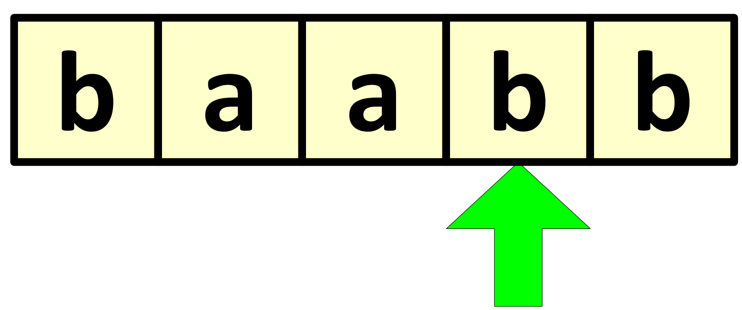


ε-Transitions

NFAs have a special type of transition called the **ε-transition**.

An NFA may follow any number of ε-transitions at any time without consuming any input.

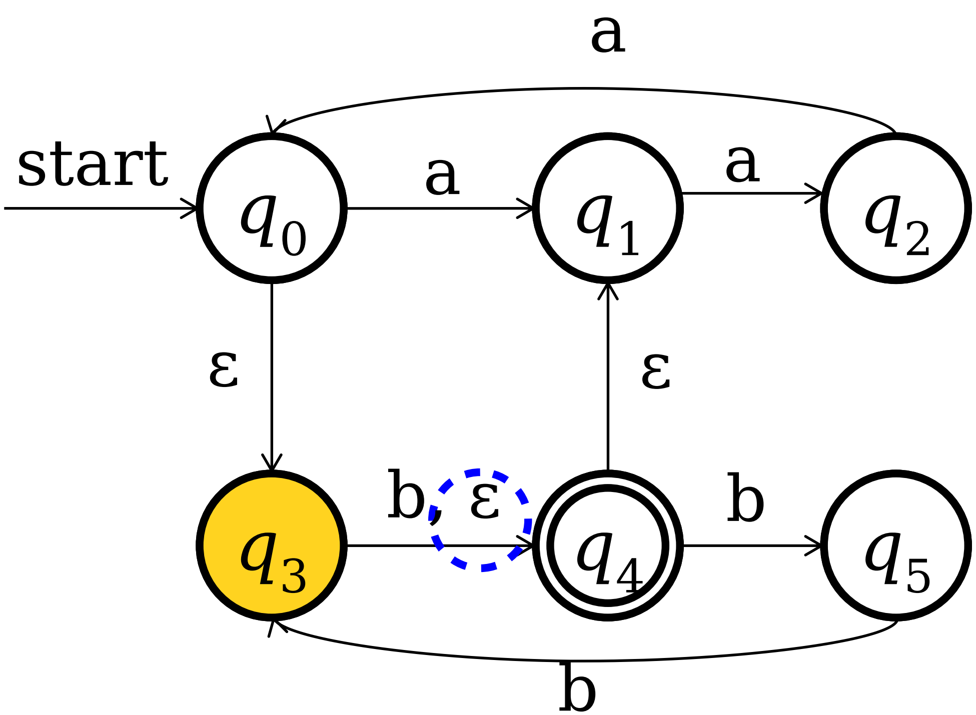


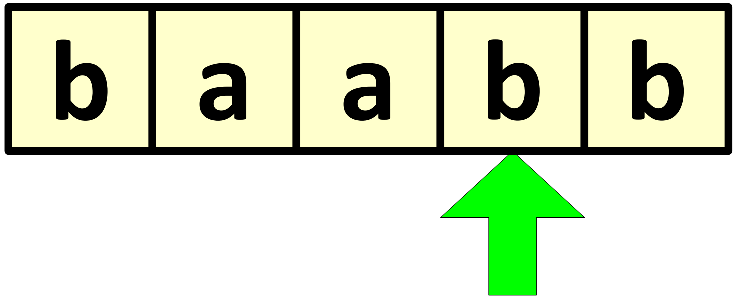


ε-Transitions

NFAs have a special type of transition called the **ε-transition**.

An NFA may follow any number of ε-transitions at any time without consuming any input.

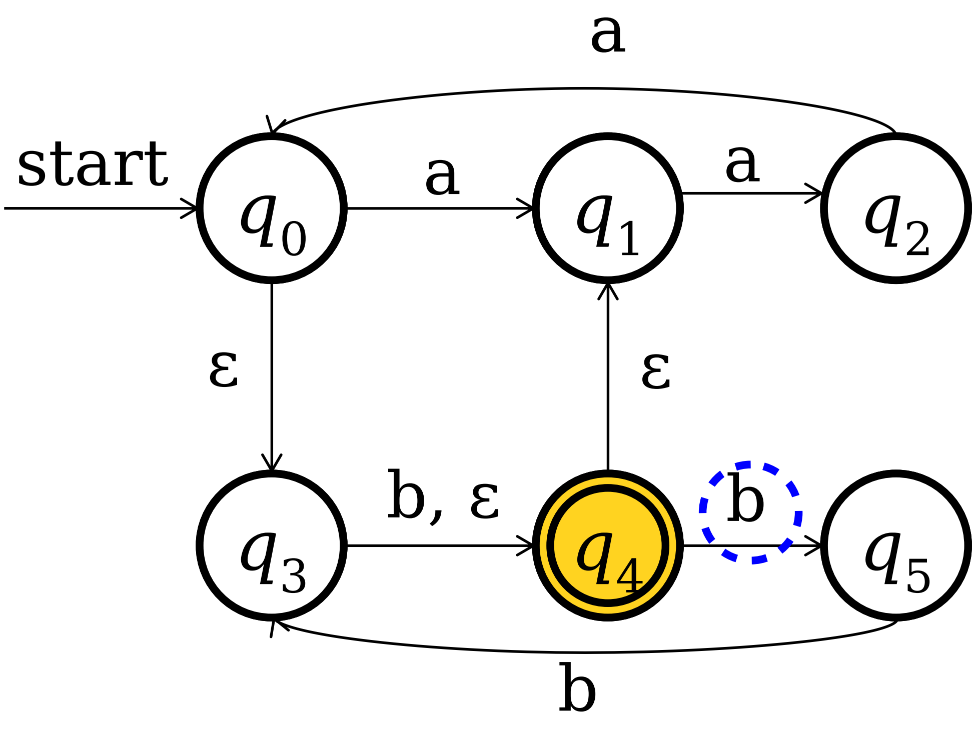


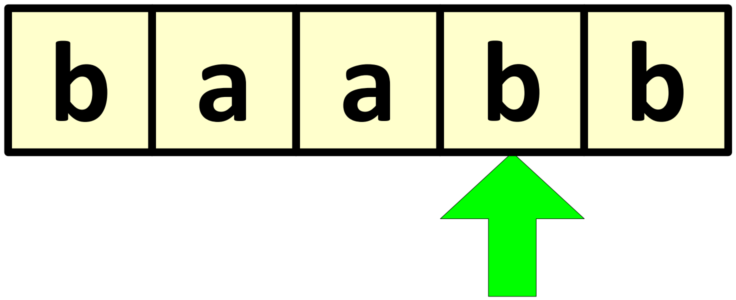


ε-Transitions

NFAs have a special type of transition called the **ε-transition**.

An NFA may follow any number of ε-transitions at any time without consuming any input.

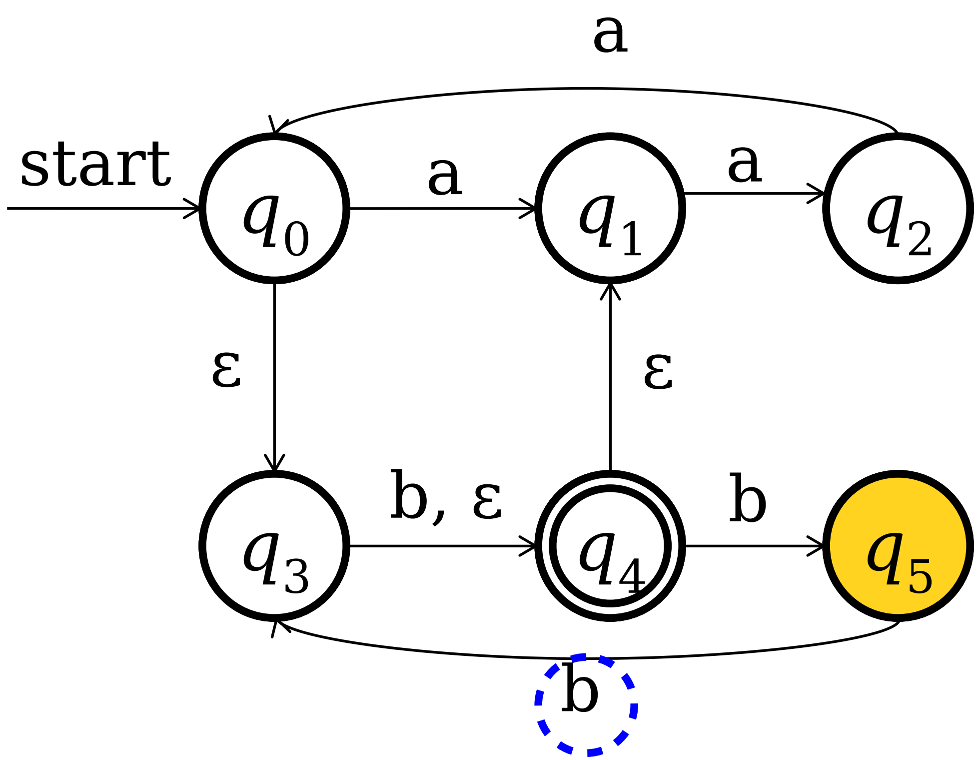


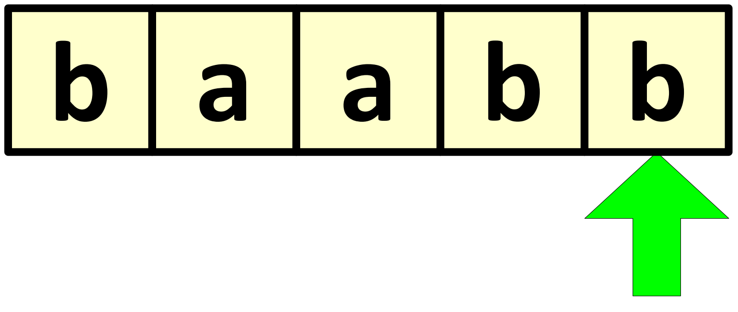


ε-Transitions

NFAs have a special type of transition called the **ε-transition**.

An NFA may follow any number of ε-transitions at any time without consuming any input.

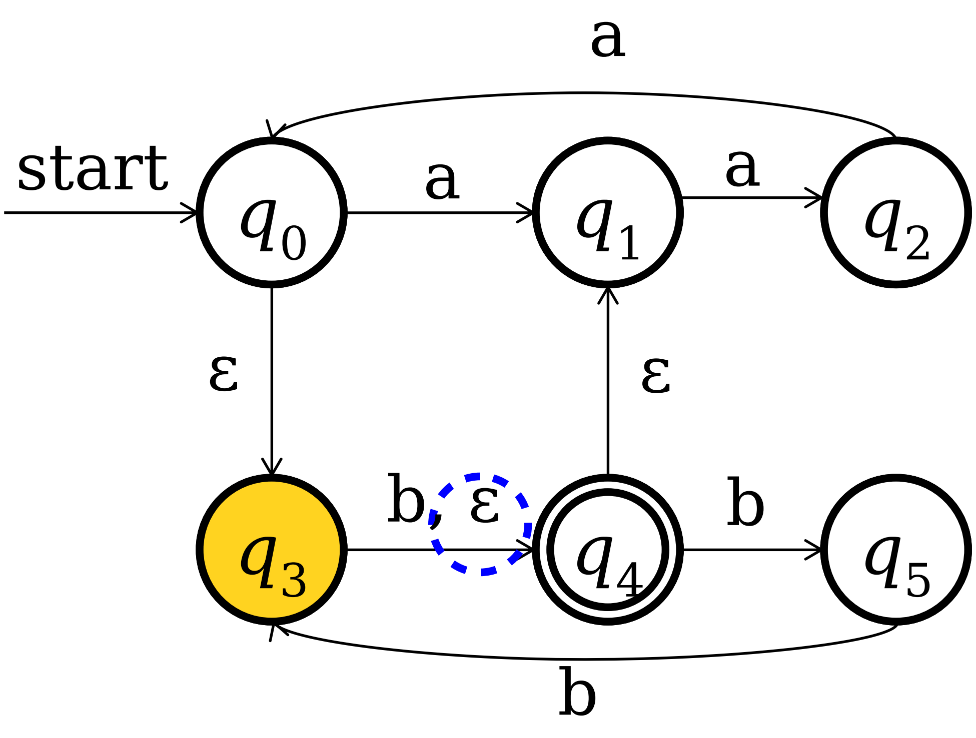


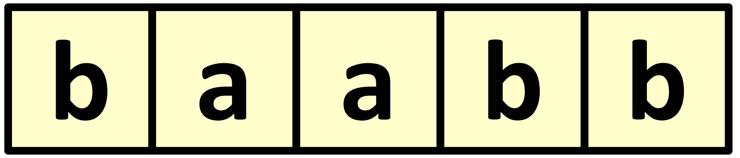


ε-Transitions

NFAs have a special type of transition called the **ε-transition**.

An NFA may follow any number of ε-transitions at any time without consuming any input.

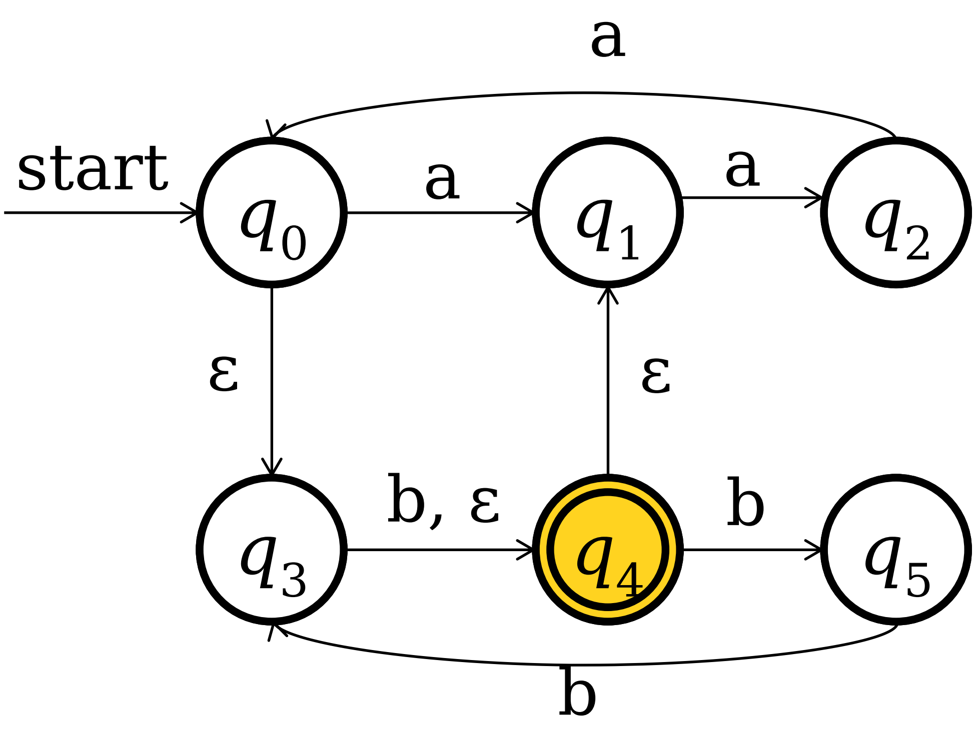


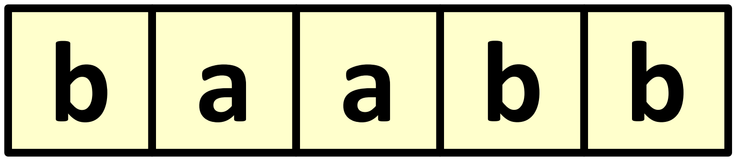


ε-Transitions

NFAs have a special type of transition called the **ε-transition**.

An NFA may follow any number of ε-transitions at any time without consuming any input.

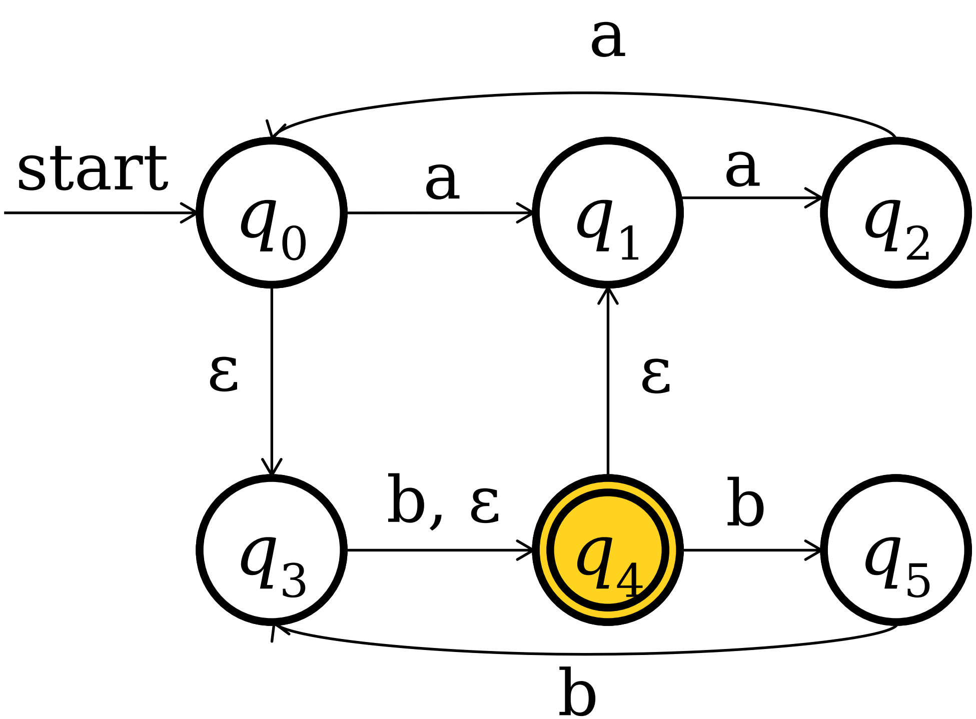


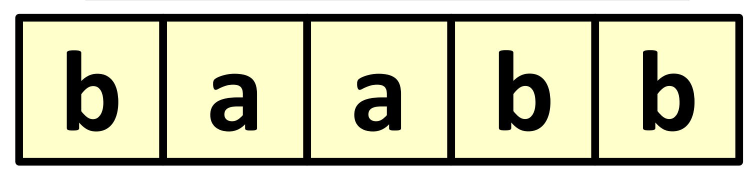


ε-Transitions

NFAs have a special type of transition called the **ε-transition**.

An NFA may follow any number of ε-transitions at any time without consuming any input.





ε-Transitions

NFAs have a special type of transition called the **ε-transition**.

An NFA may follow any number of ε-transitions at any time without consuming any input.

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

Intuiting Nondeterminism

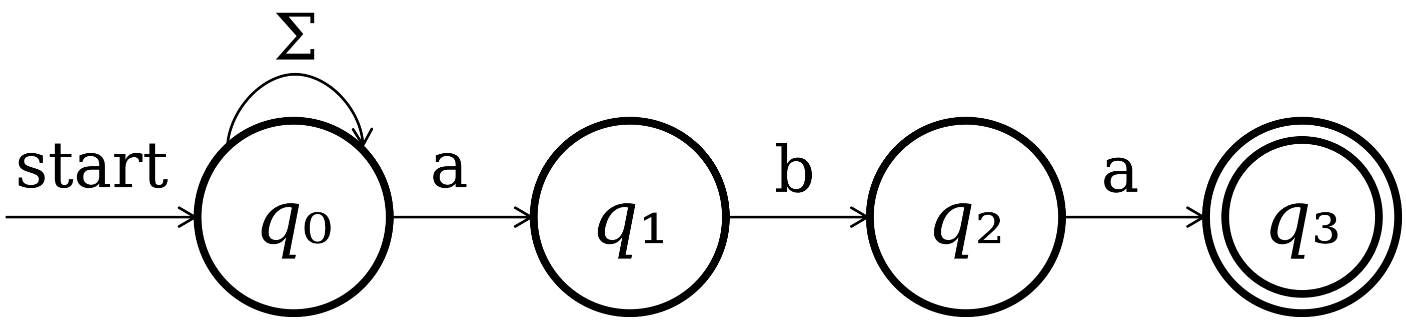
Nondeterministic machines are a serious departure from physical computers. How can we build up an intuition for them?

There are two particularly useful frameworks for interpreting nondeterminism:

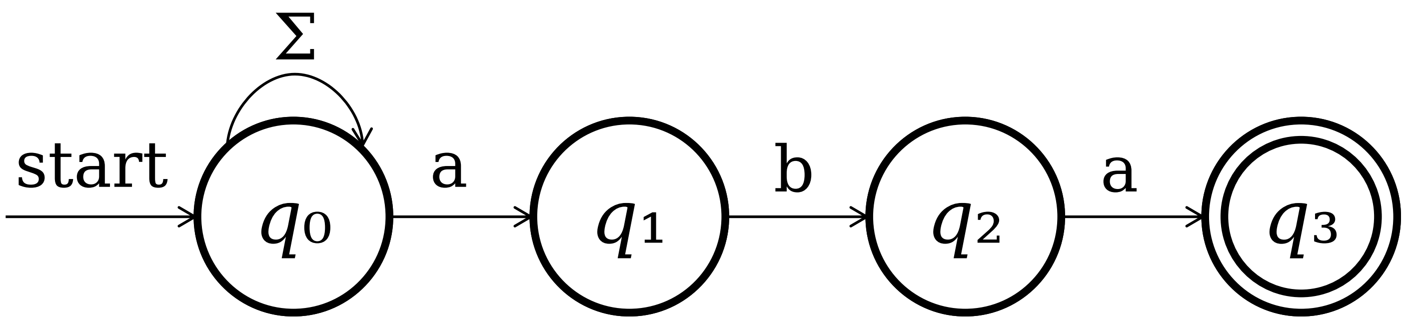
Perfect positive guessing

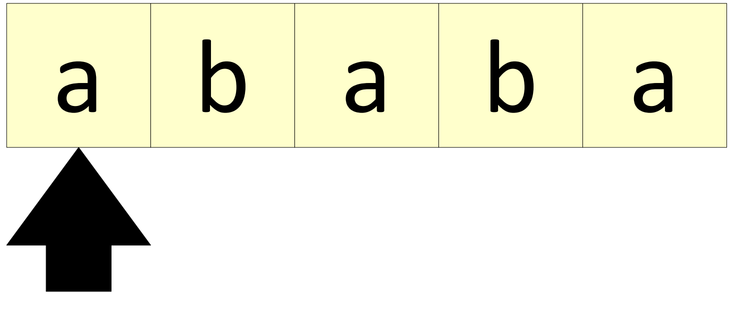
Massive parallelism

Perfect Positive Guessing

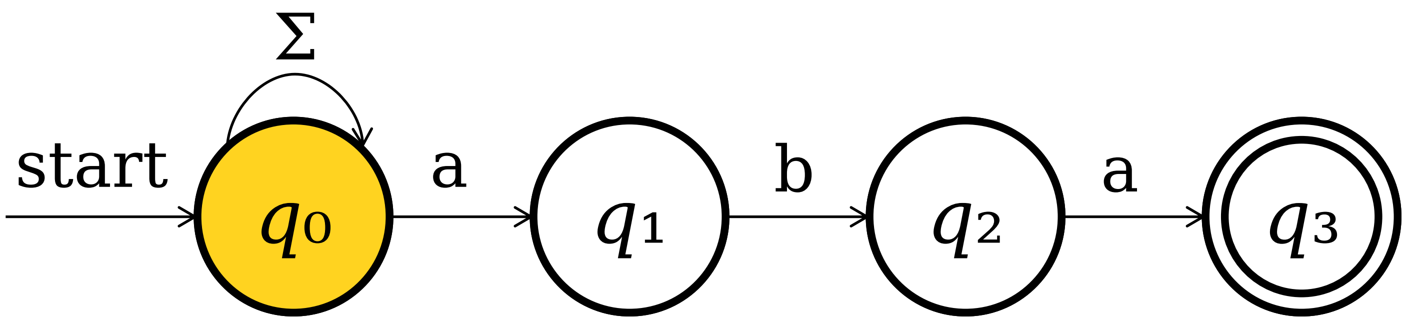


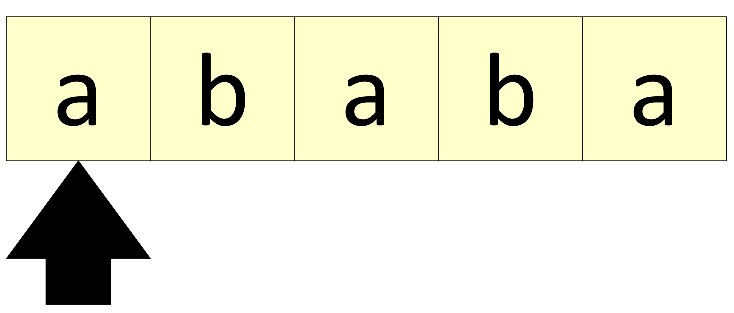
Perfect Positive Guessing



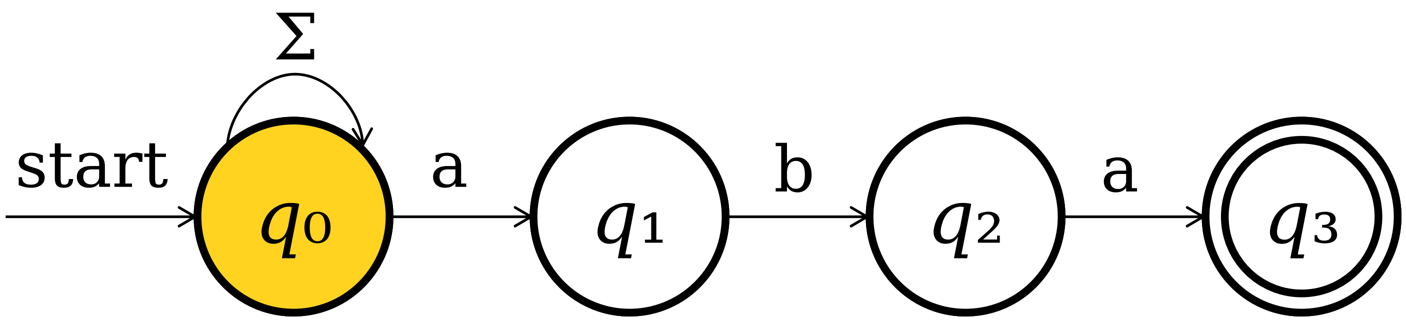


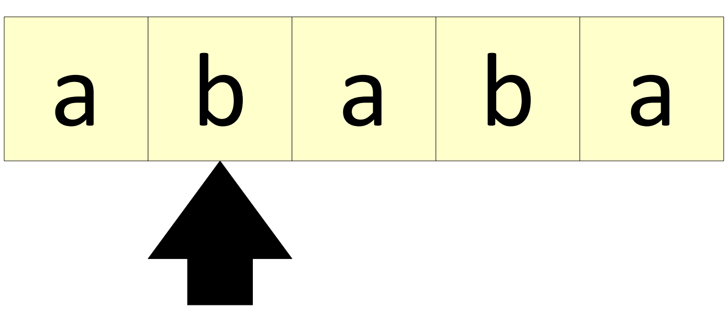
Perfect Positive Guessing



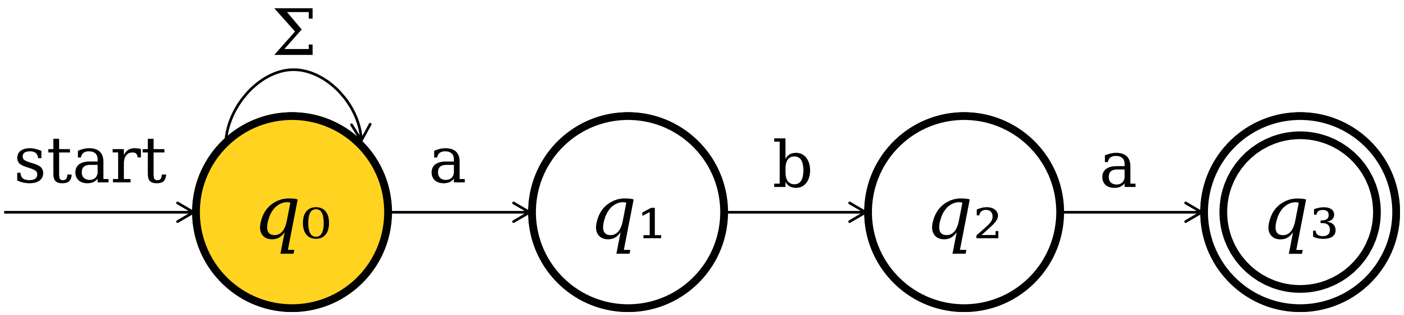


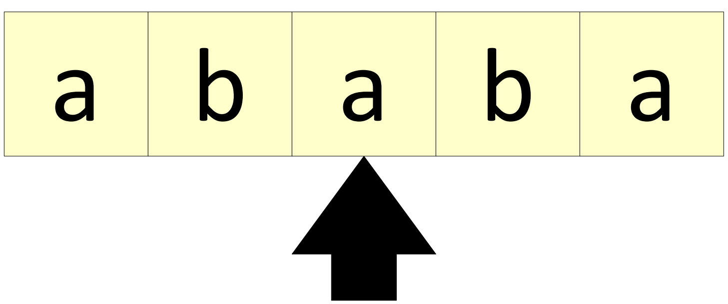
Perfect Positive Guessing



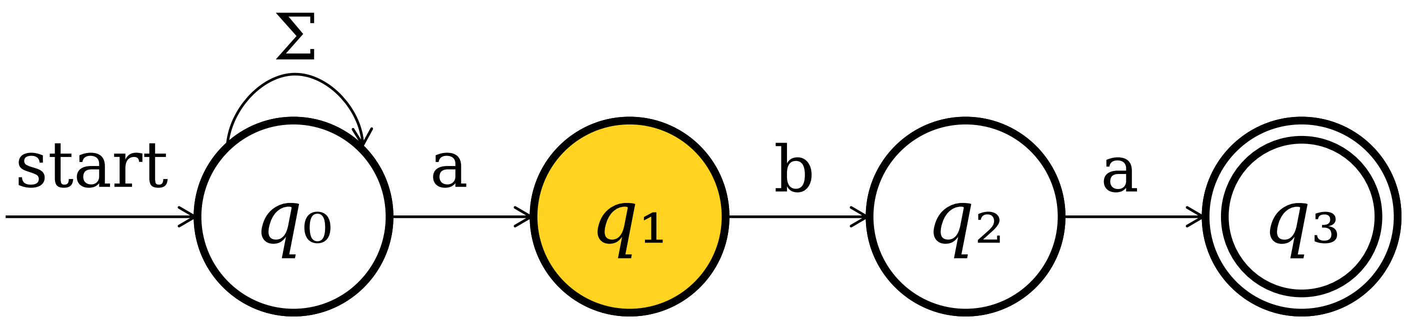


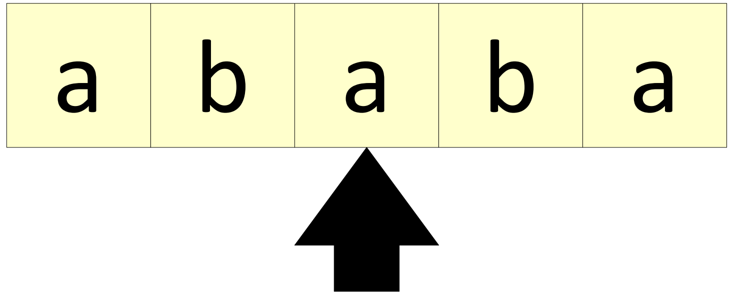
Perfect Positive Guessing



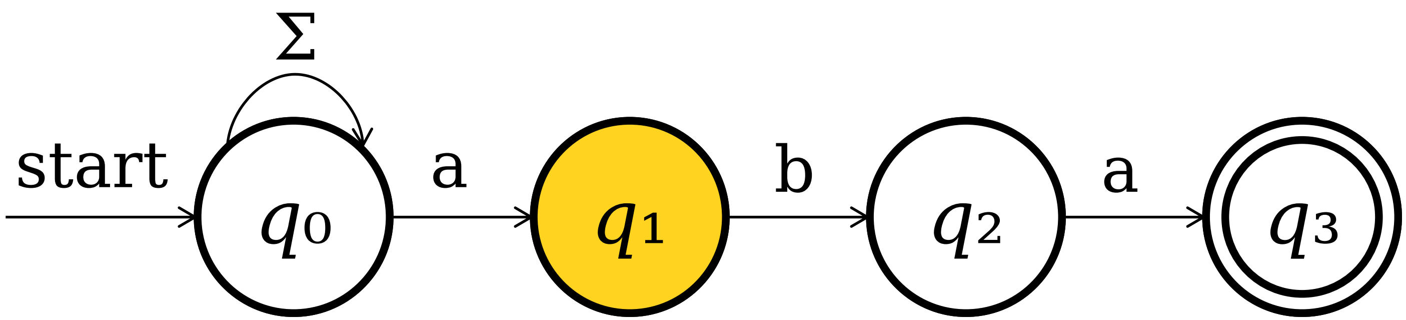


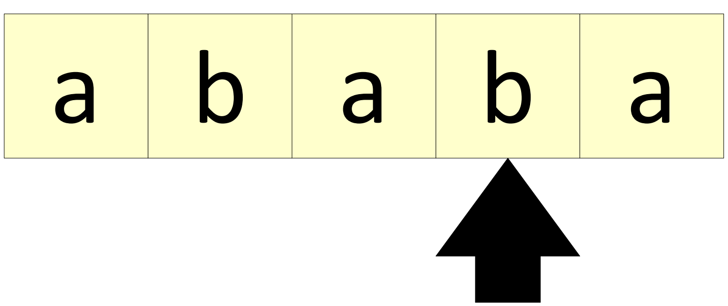
Perfect Positive Guessing



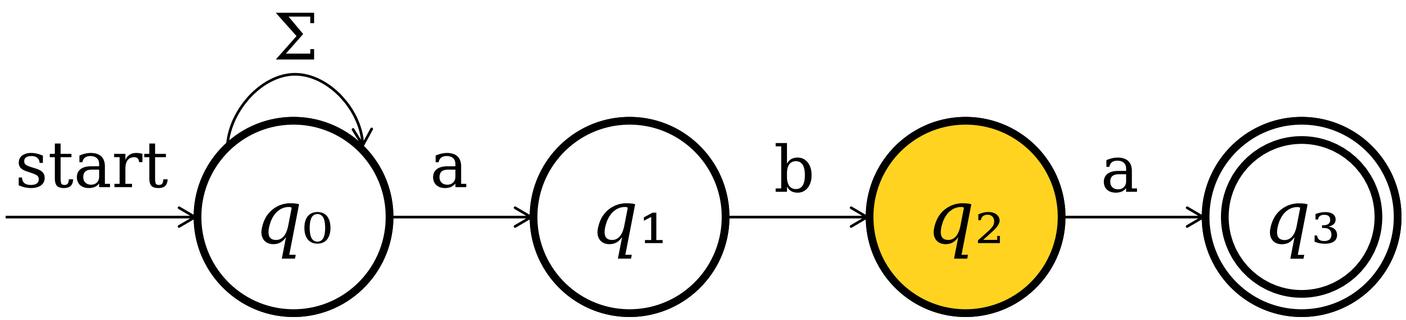


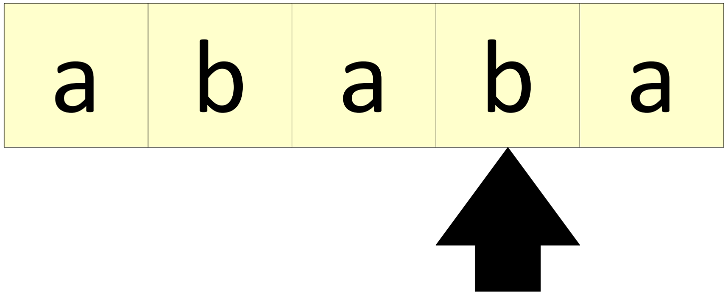
Perfect Positive Guessing



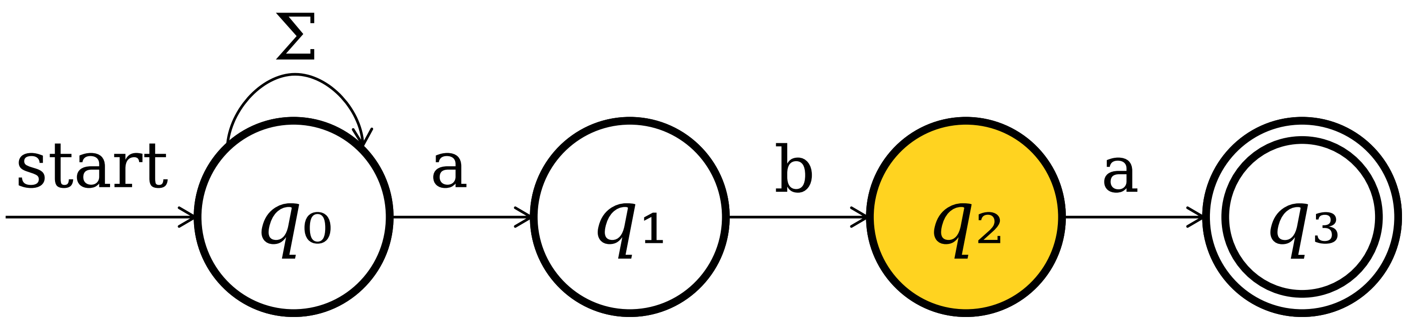


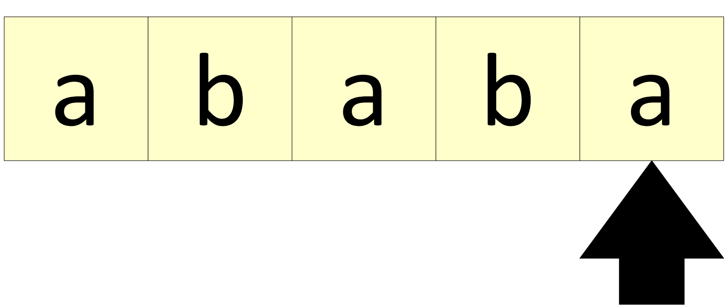
Perfect Positive Guessing



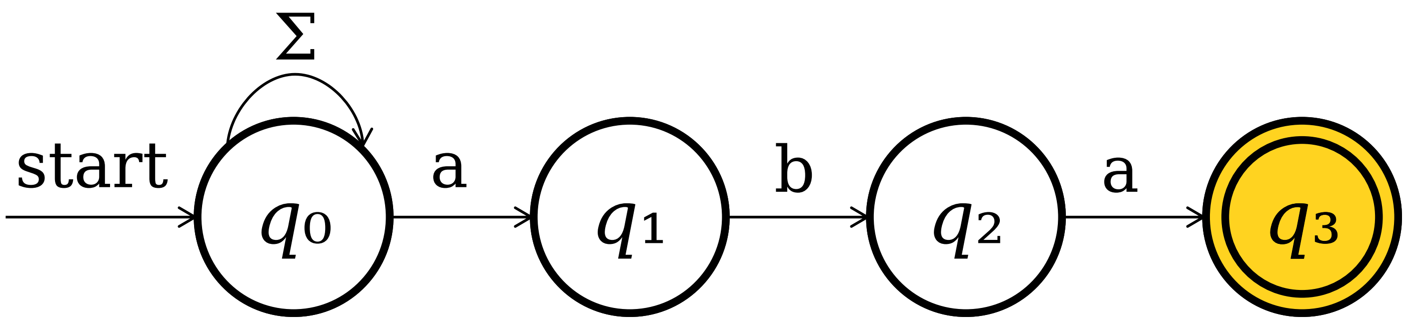


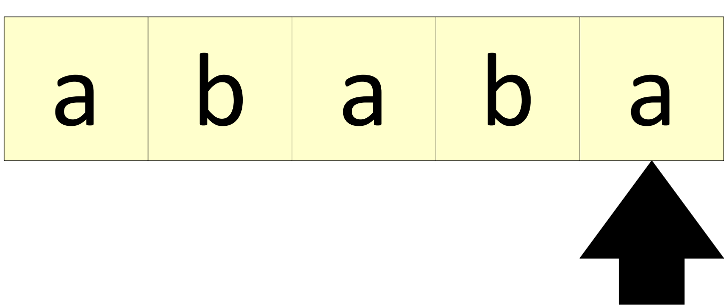
Perfect Positive Guessing



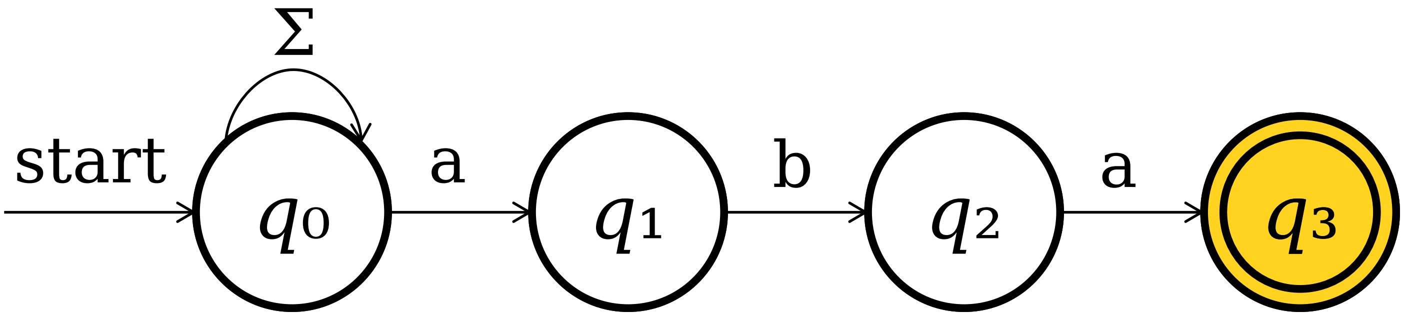


Perfect Positive Guessing





Perfect Positive Guessing



a	b	a	b	a

Perfect Positive Guessing

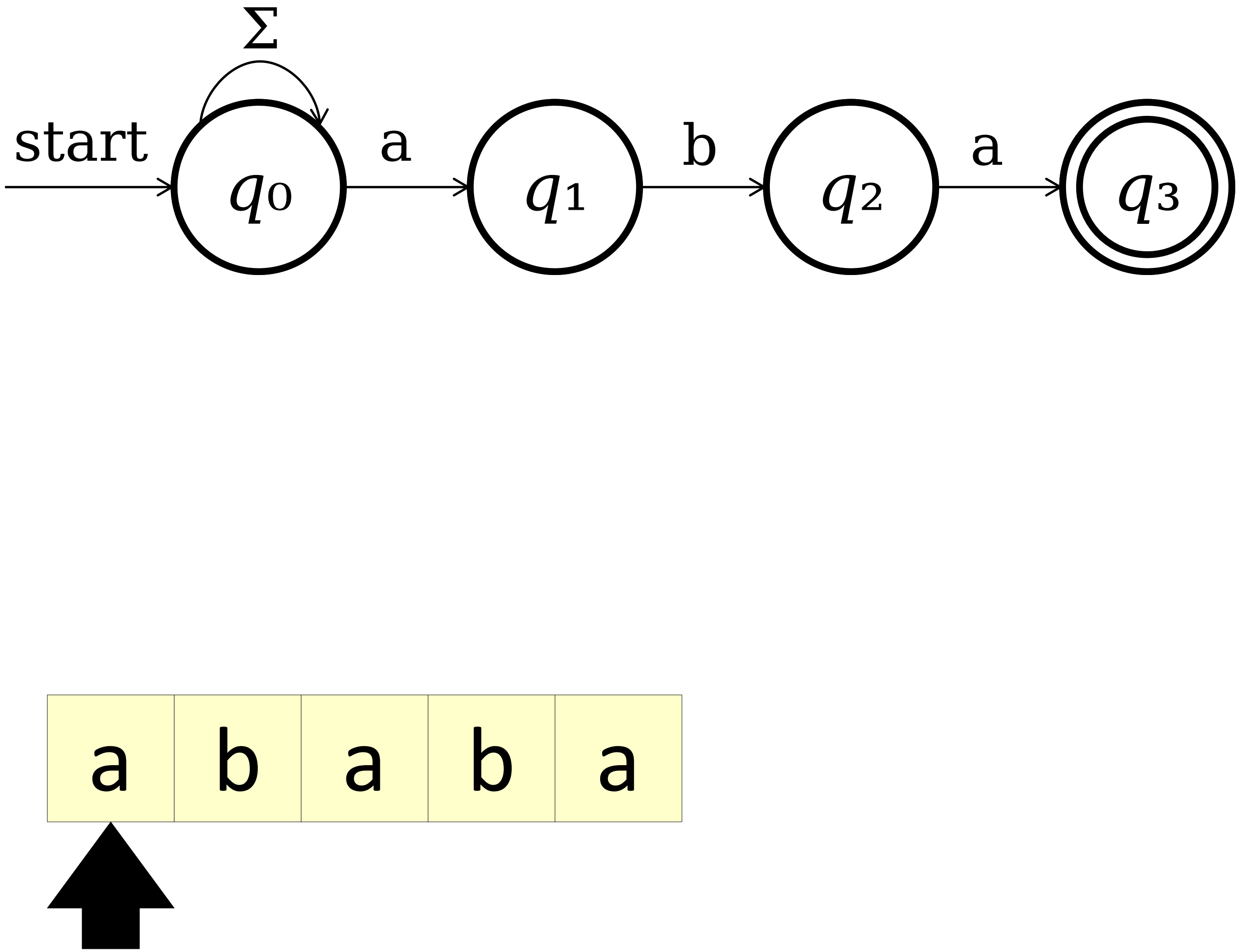
We can view nondeterministic machines as having ***Magic Superpowers*** that enable them to guess choices that lead to an accepting state.

If there is at least one choice that leads to an accepting state, the machine will guess it.

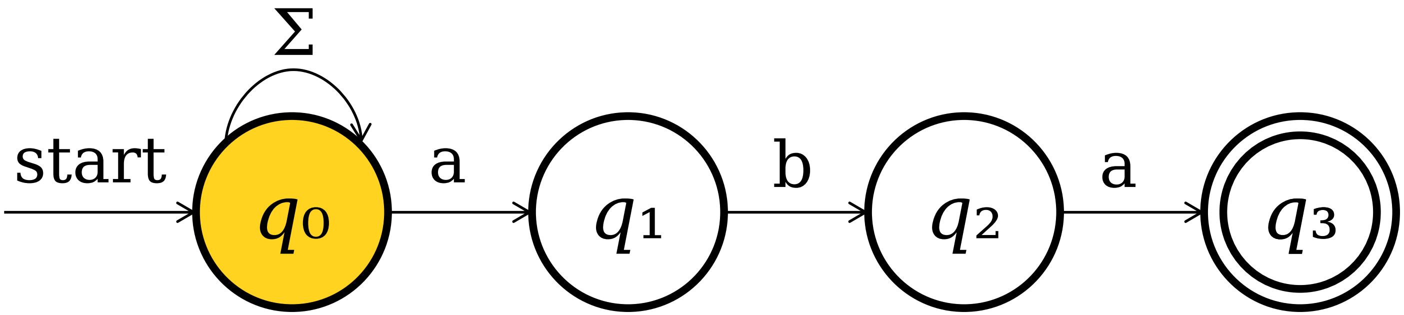
If there are no choices, the machine guesses any one of the wrong guesses.

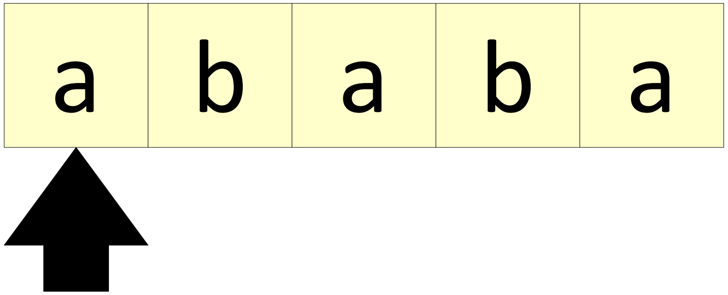
There is no known way to physically model this intuition of nondeterminism – this is quite a departure from reality!

Massive Parallelism

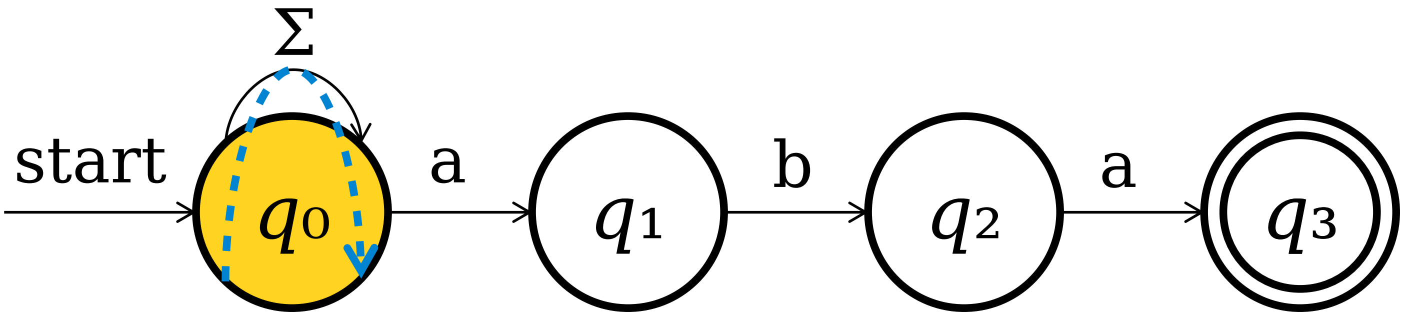


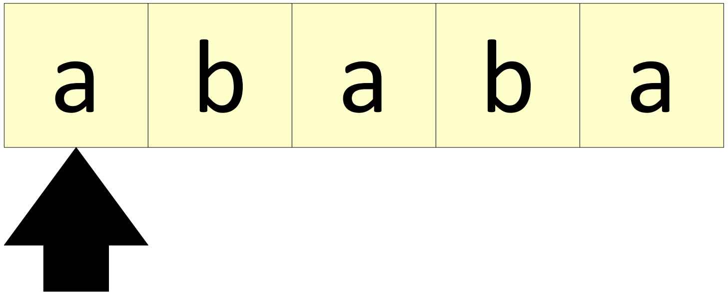
Massive Parallelism



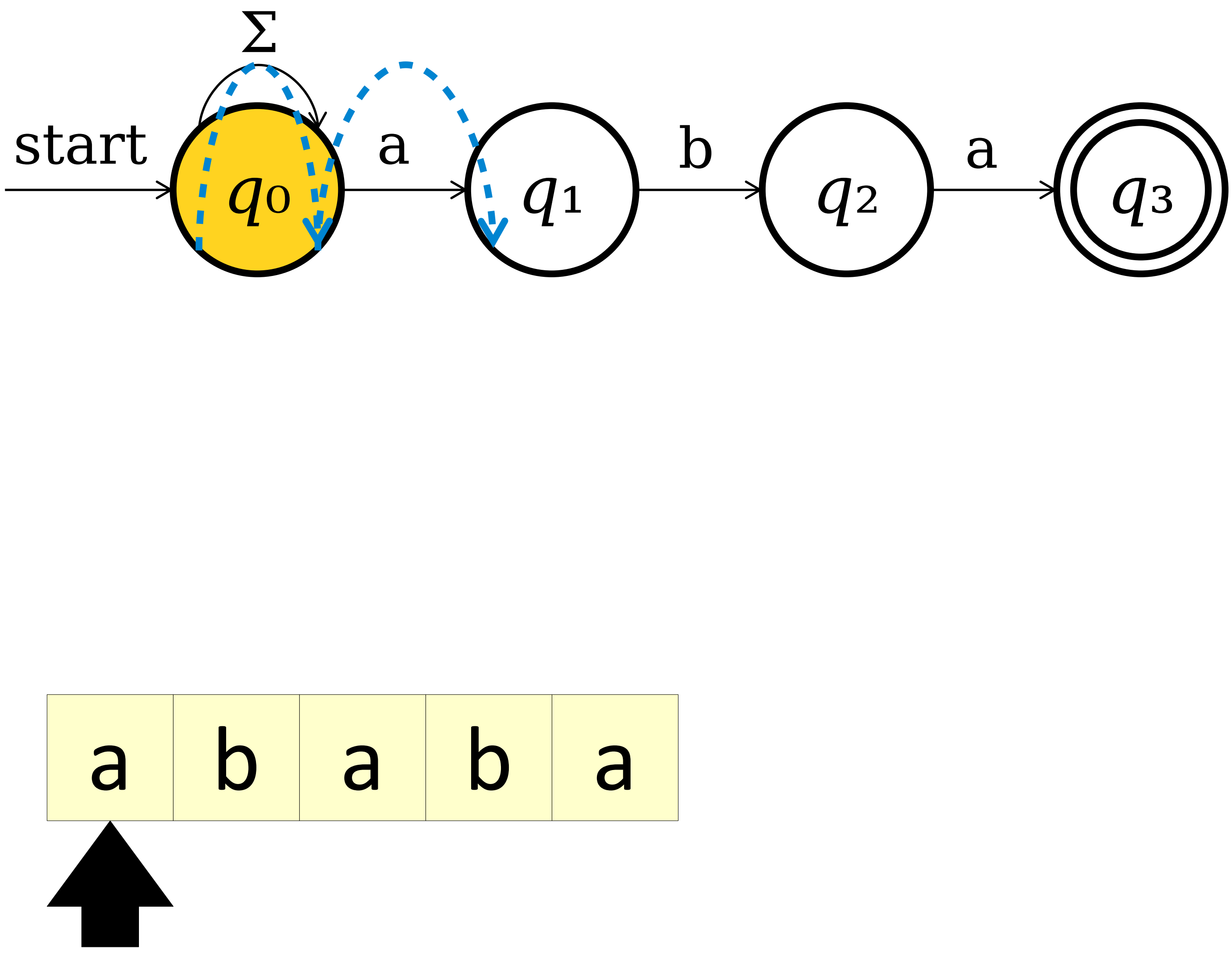


Massive Parallelism

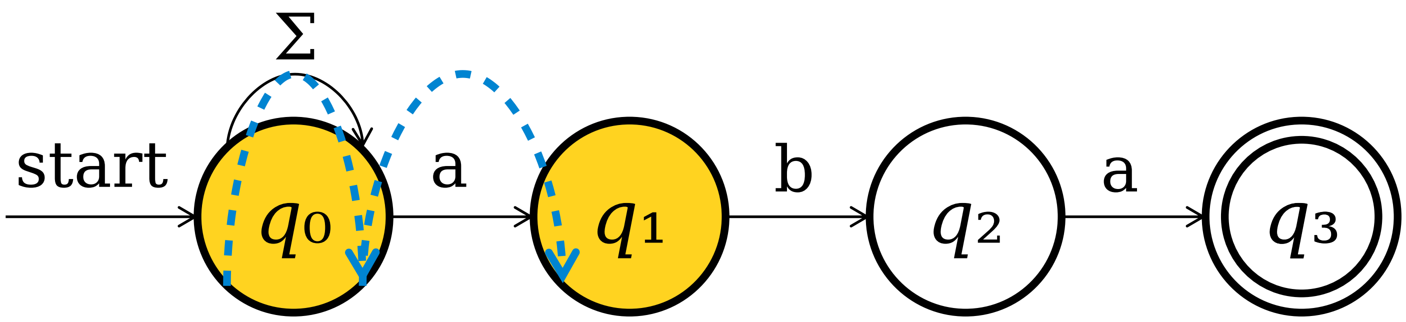


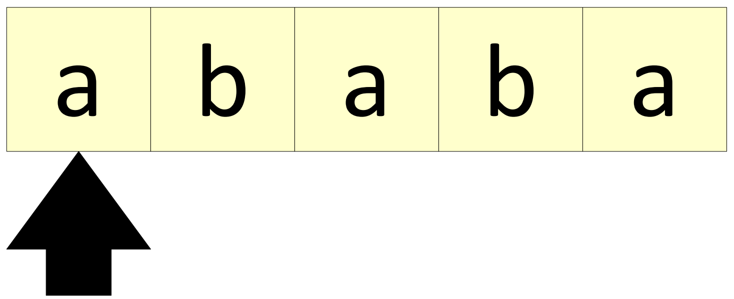


Massive Parallelism

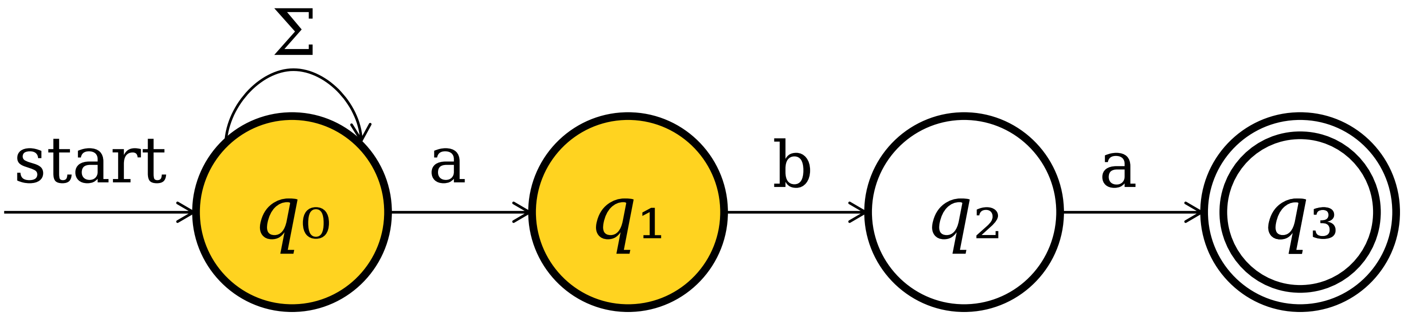


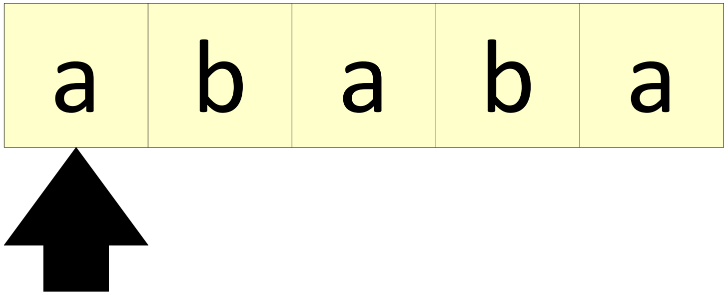
Massive Parallelism



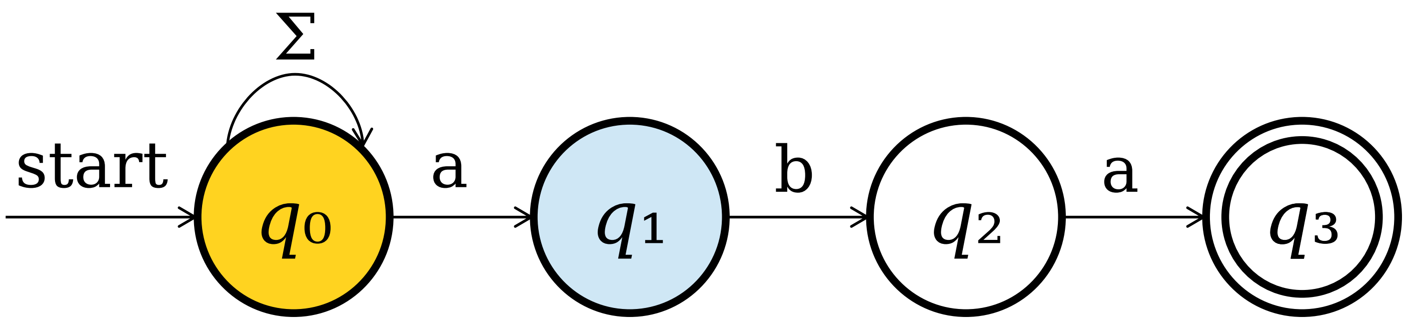


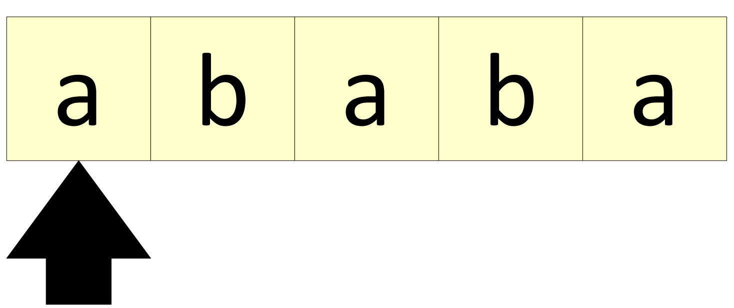
Massive Parallelism



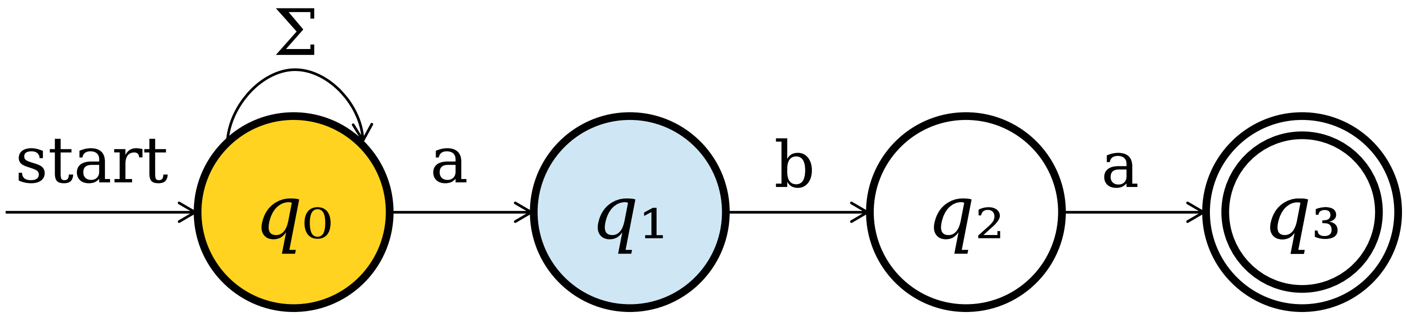


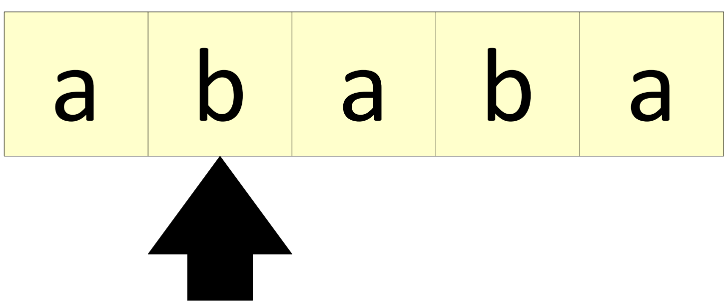
Massive Parallelism



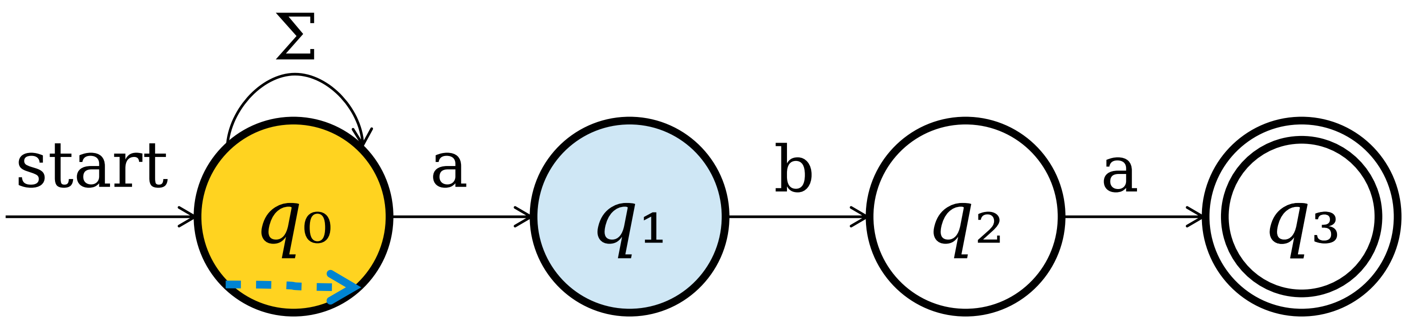


Massive Parallelism



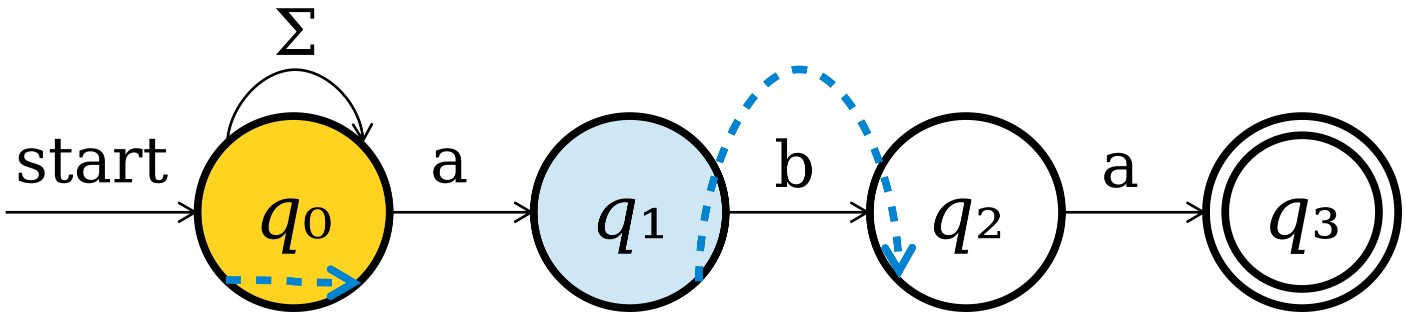


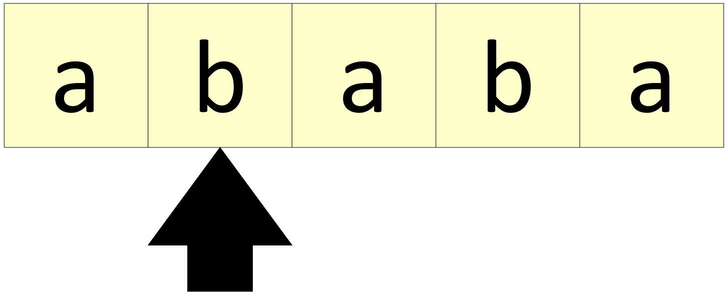
Massive Parallelism



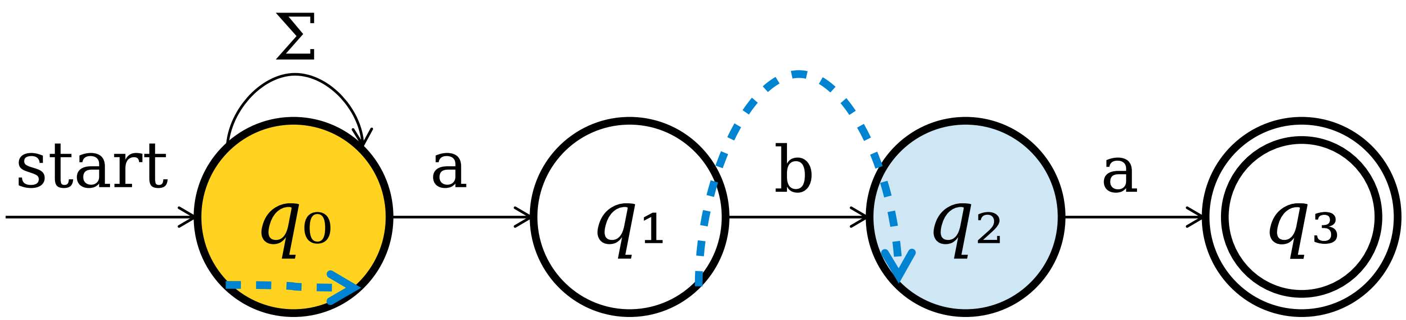
a	b	a	b	a

Massive Parallelism

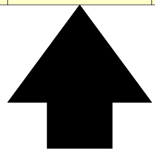




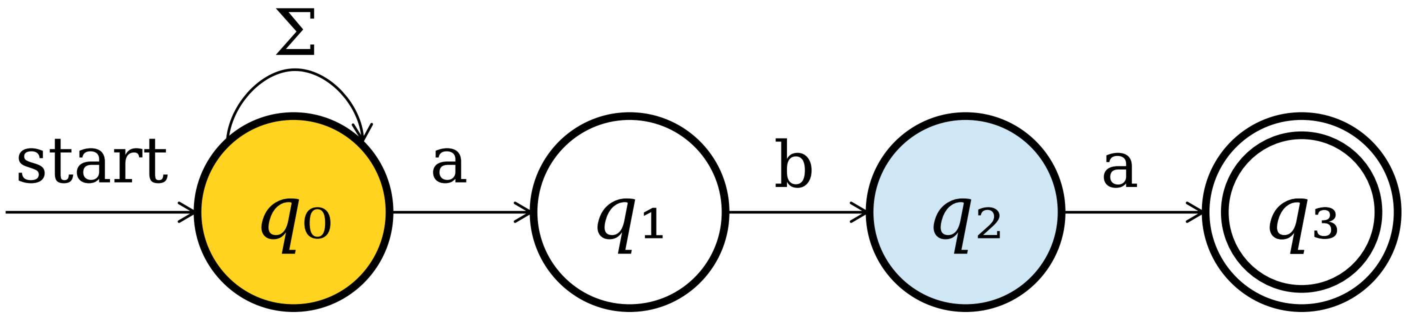
Massive Parallelism

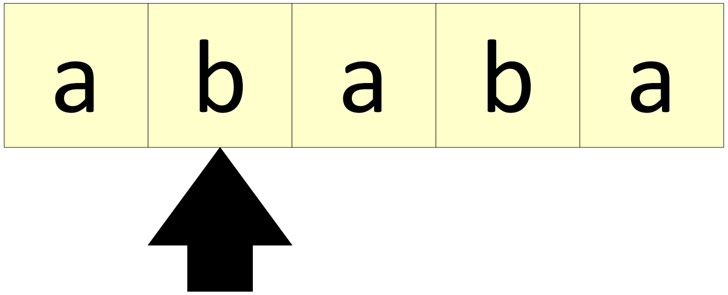


a	b	a	b	a

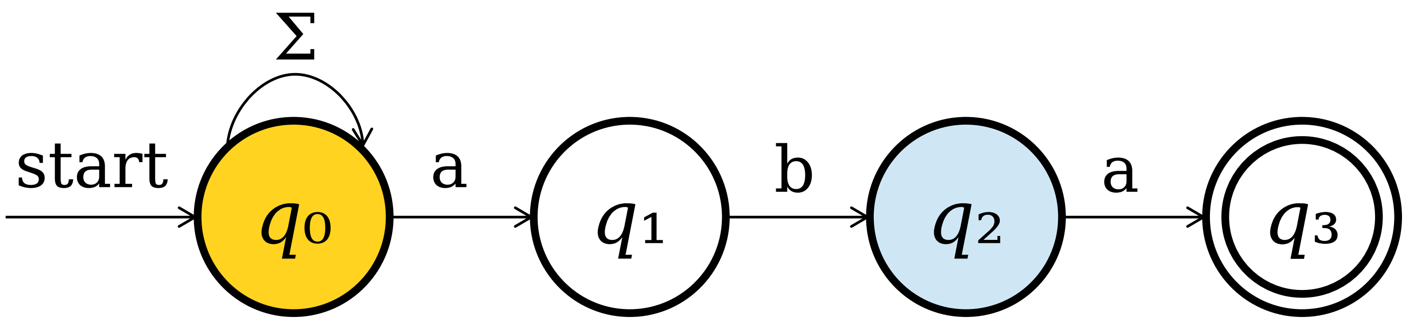


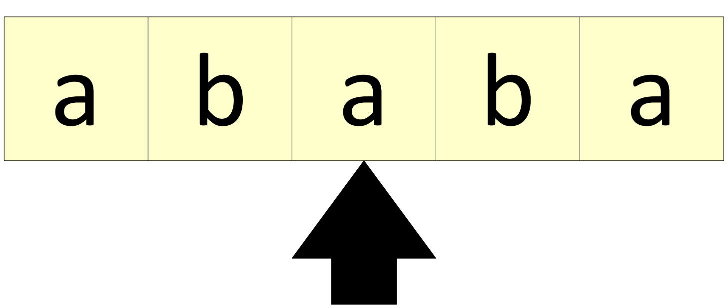
Massive Parallelism



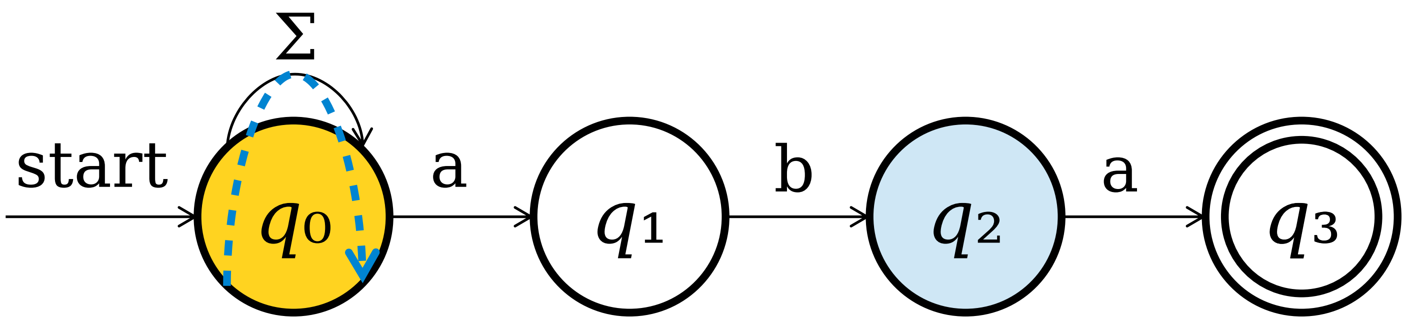


Massive Parallelism



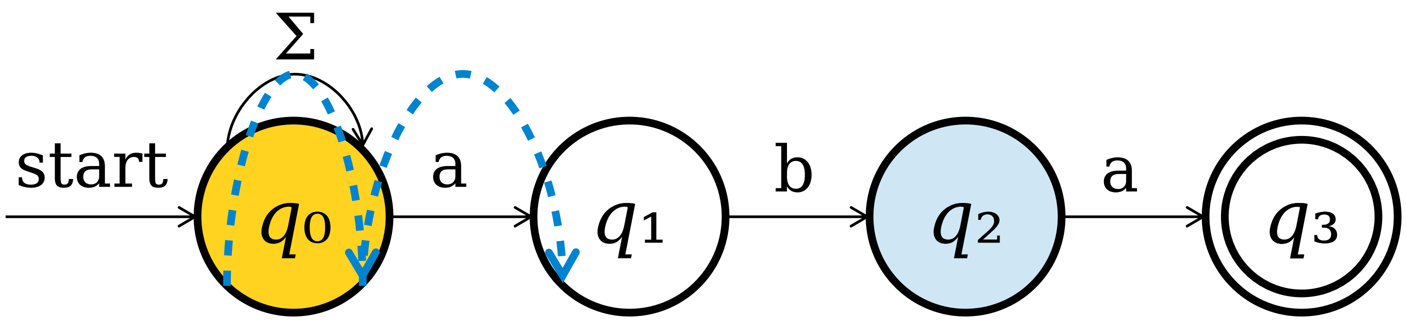


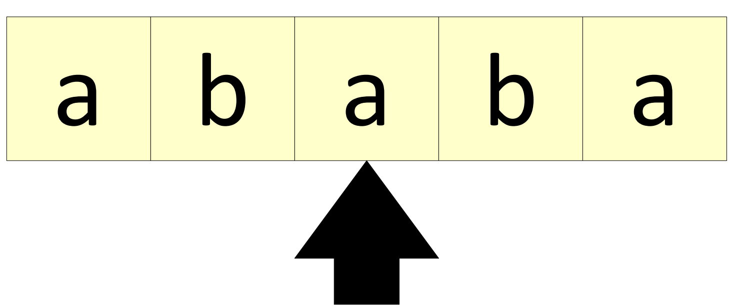
Massive Parallelism



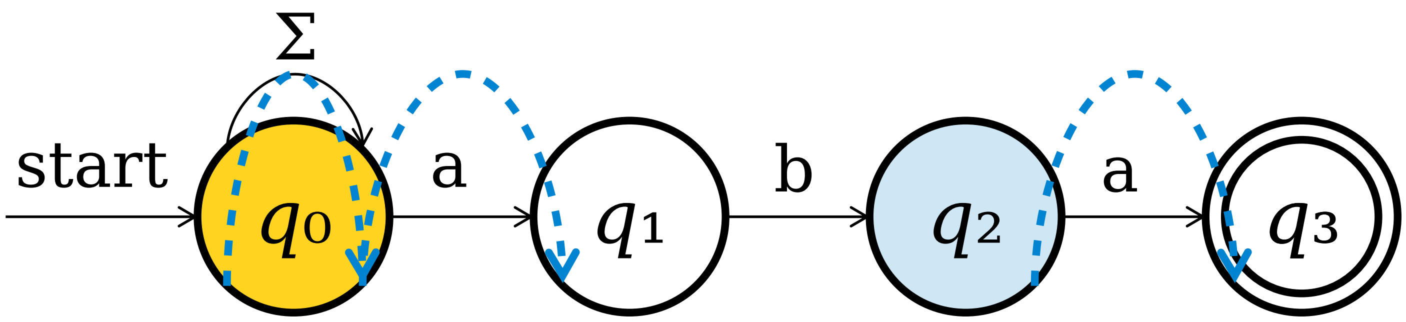
a	b	a	b	a

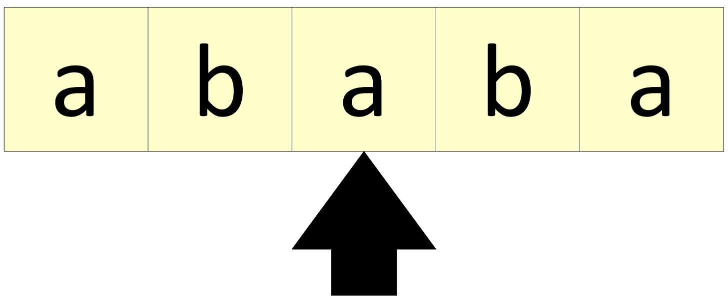
Massive Parallelism



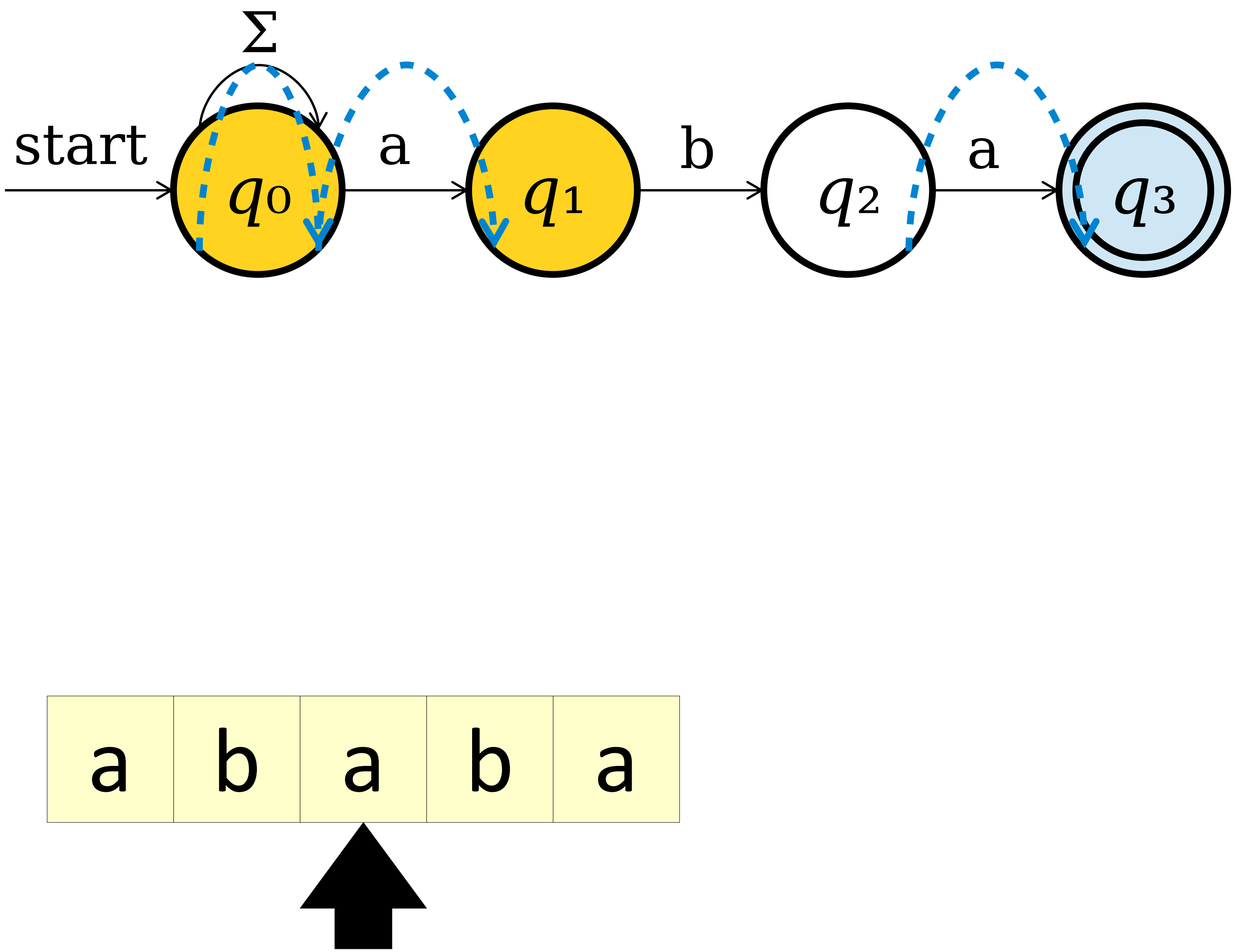


Massive Parallelism

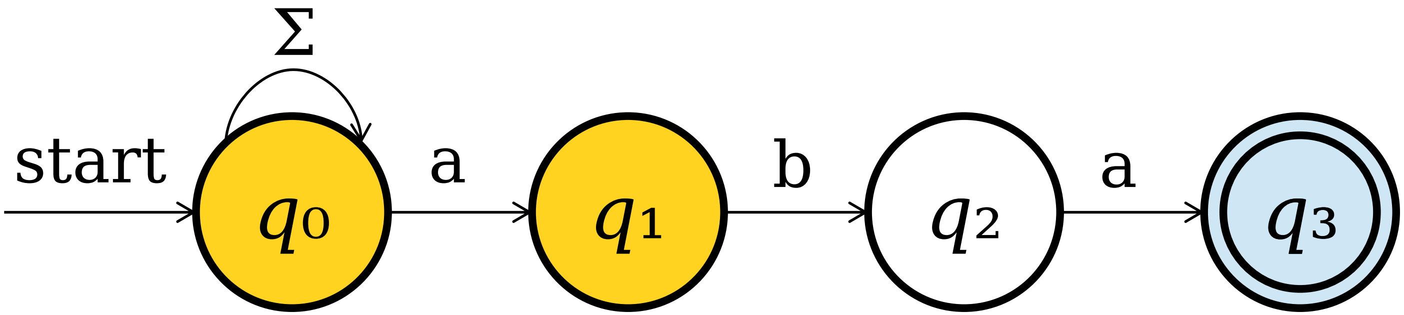


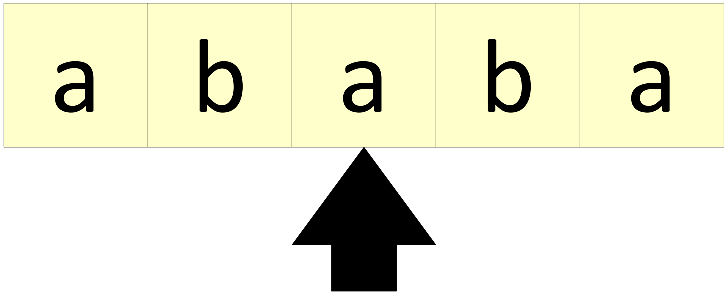


Massive Parallelism

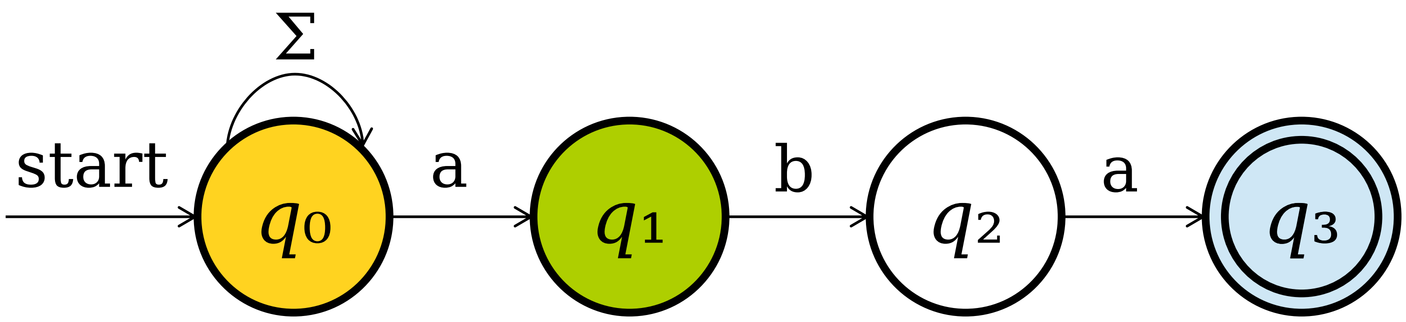


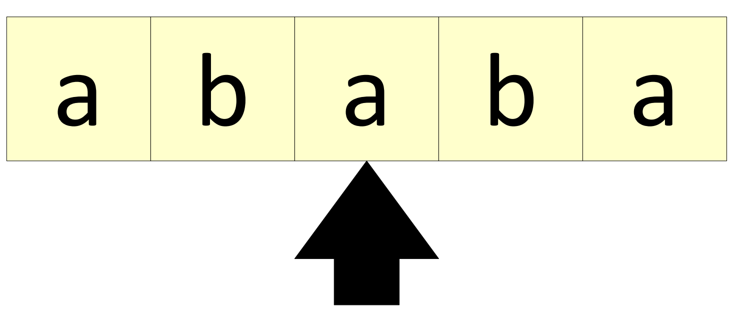
Massive Parallelism



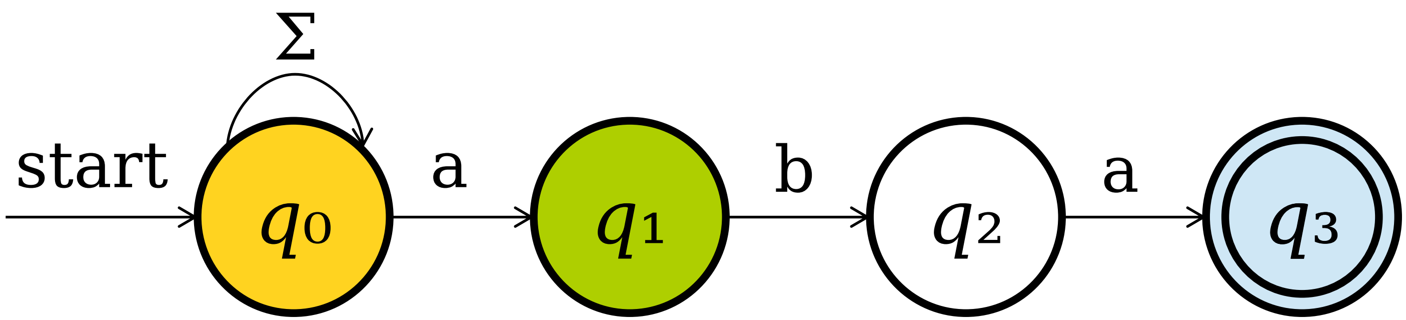


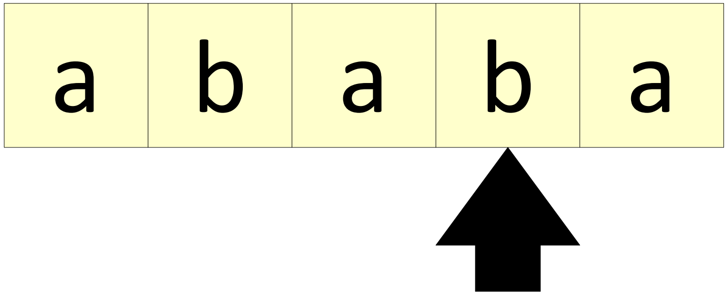
Massive Parallelism



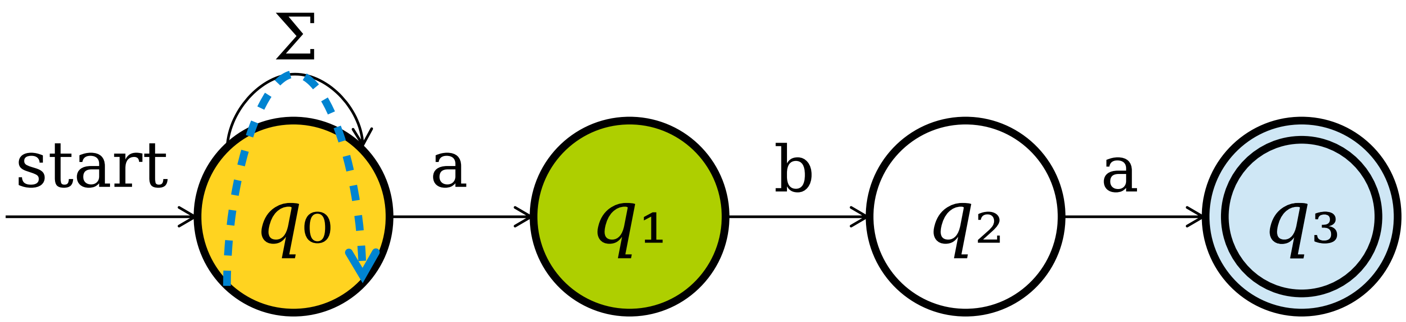


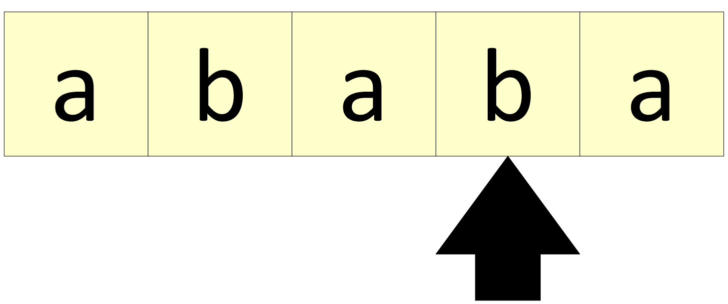
Massive Parallelism



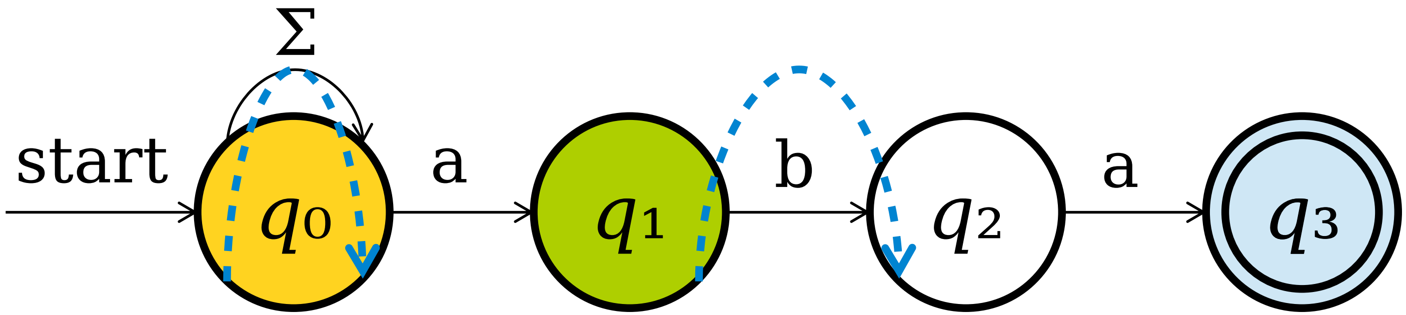


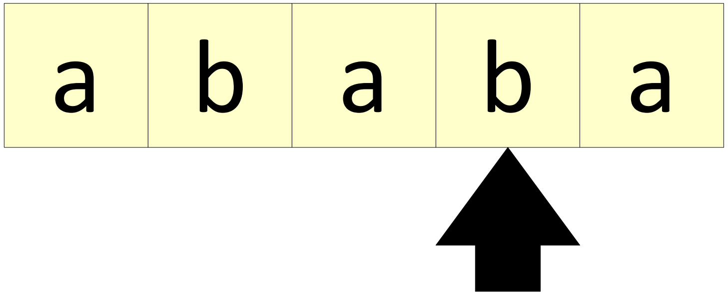
Massive Parallelism



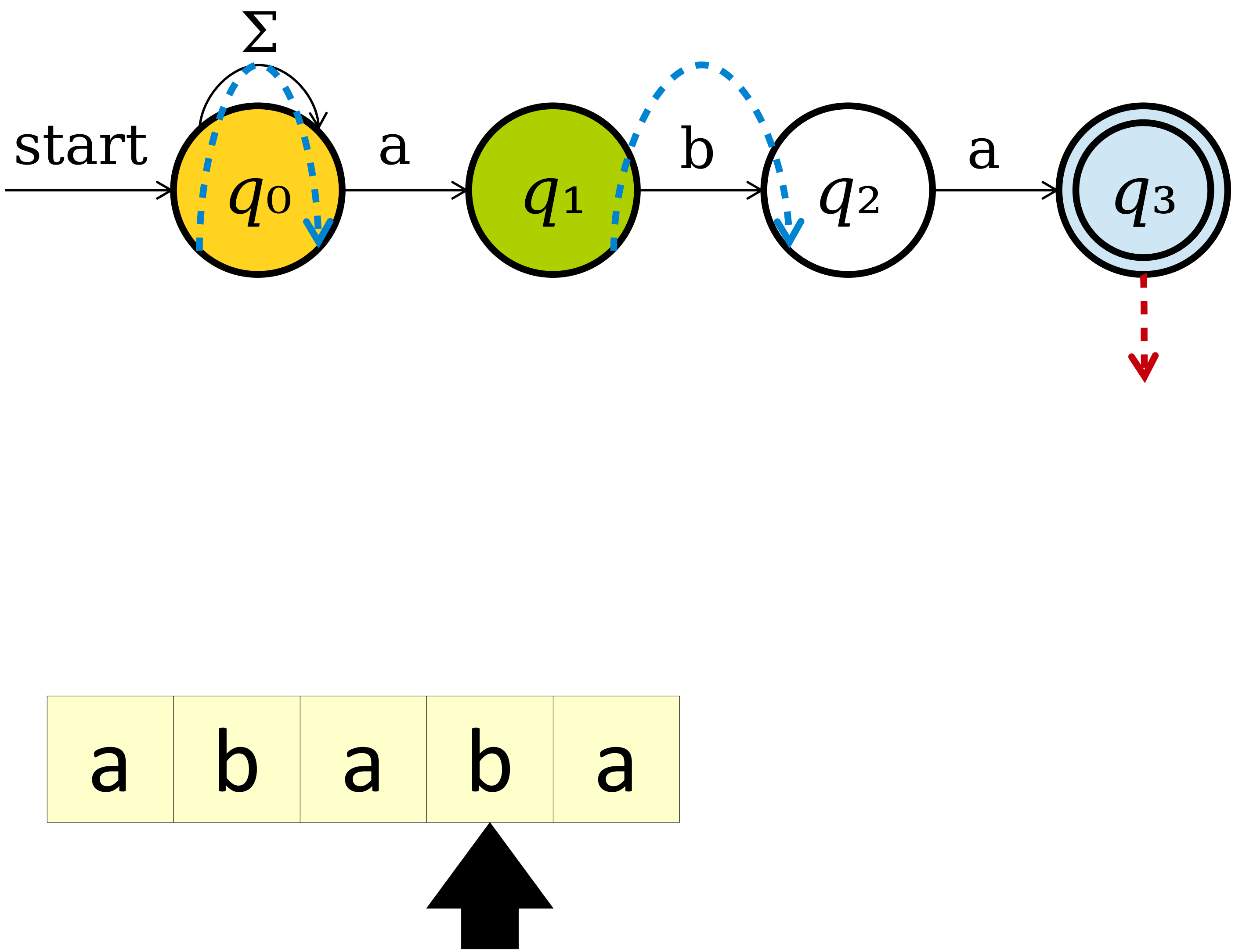


Massive Parallelism

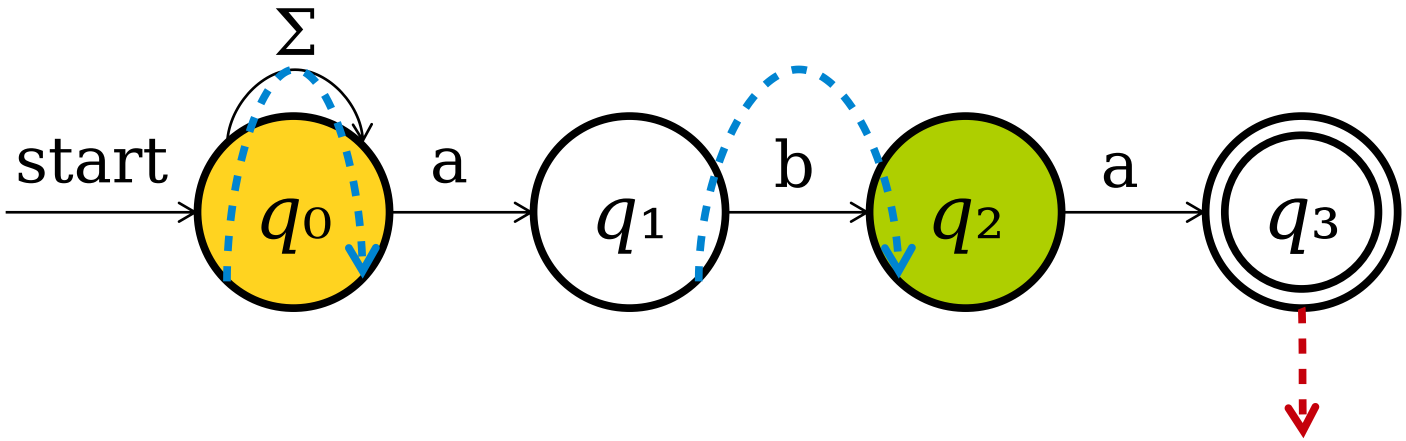


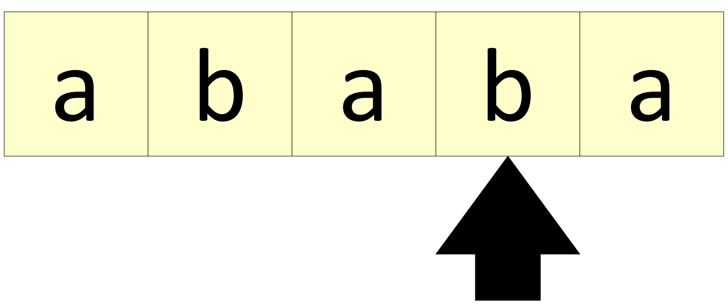


Massive Parallelism

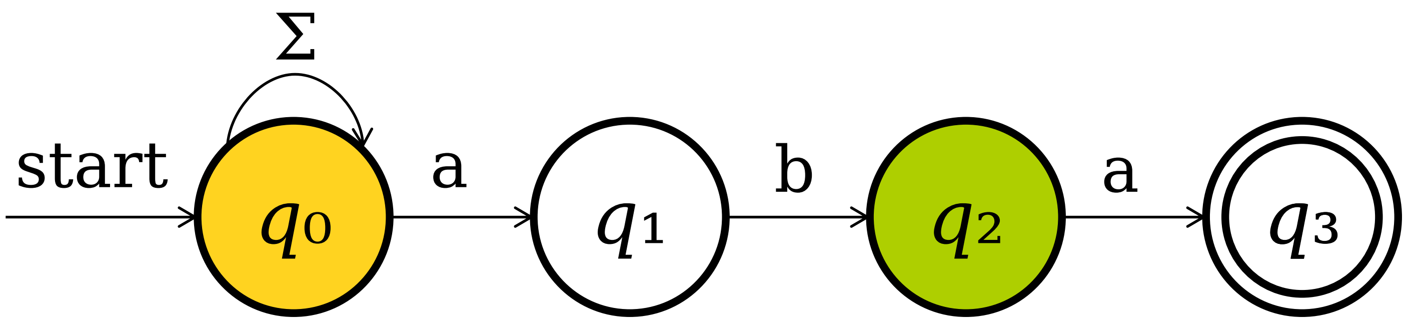


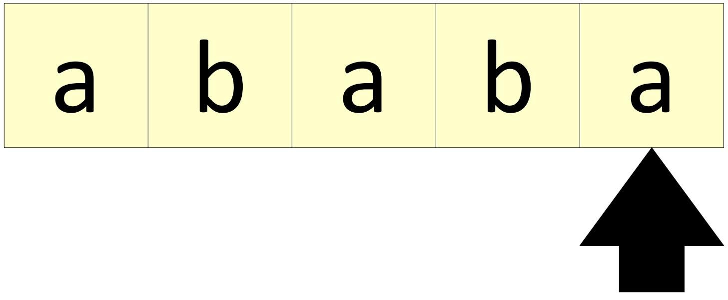
Massive Parallelism



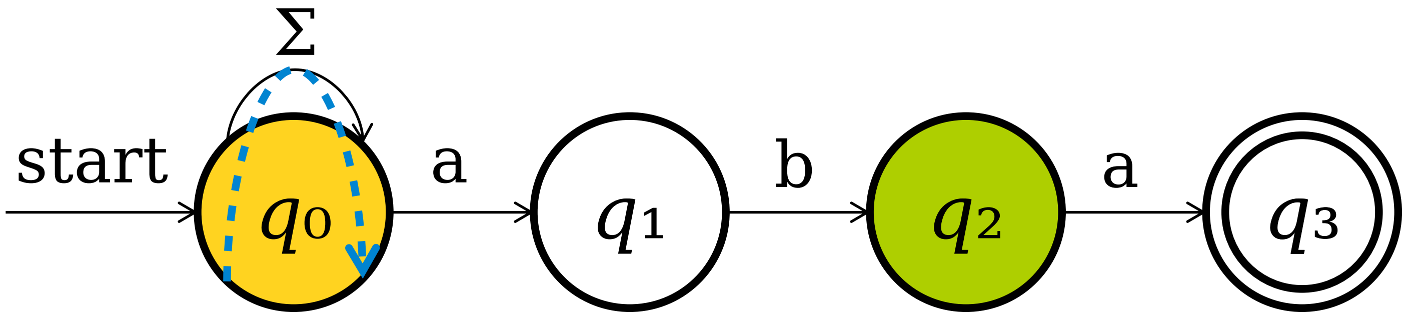


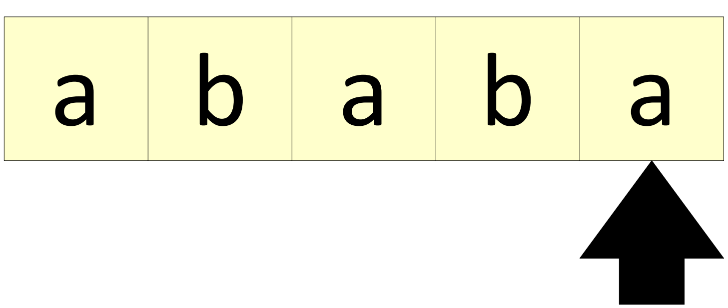
Massive Parallelism



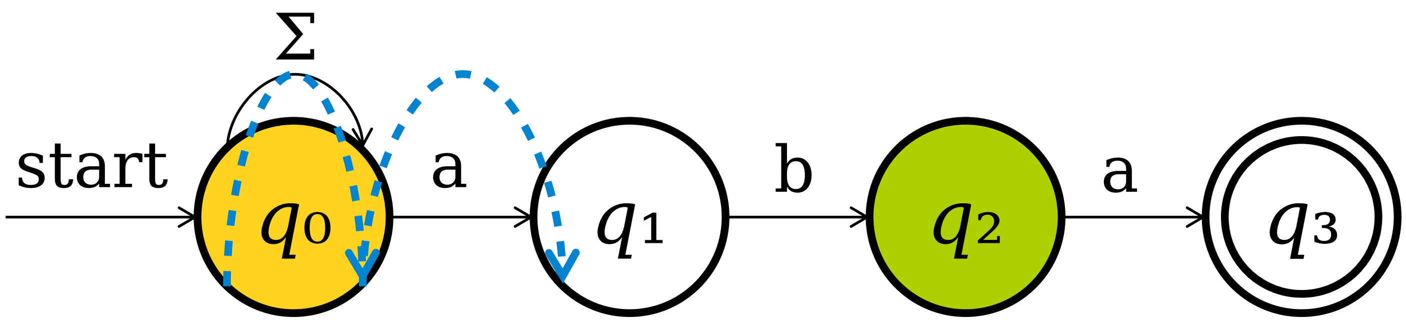


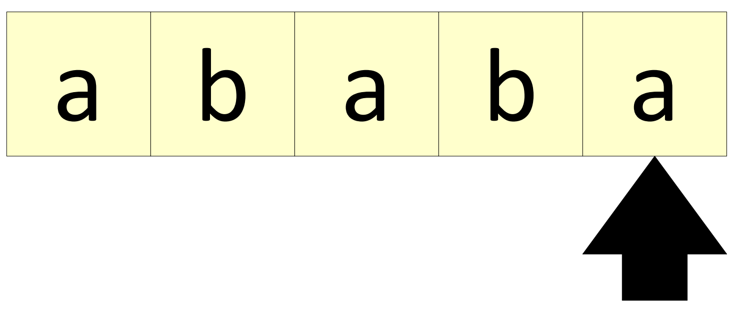
Massive Parallelism



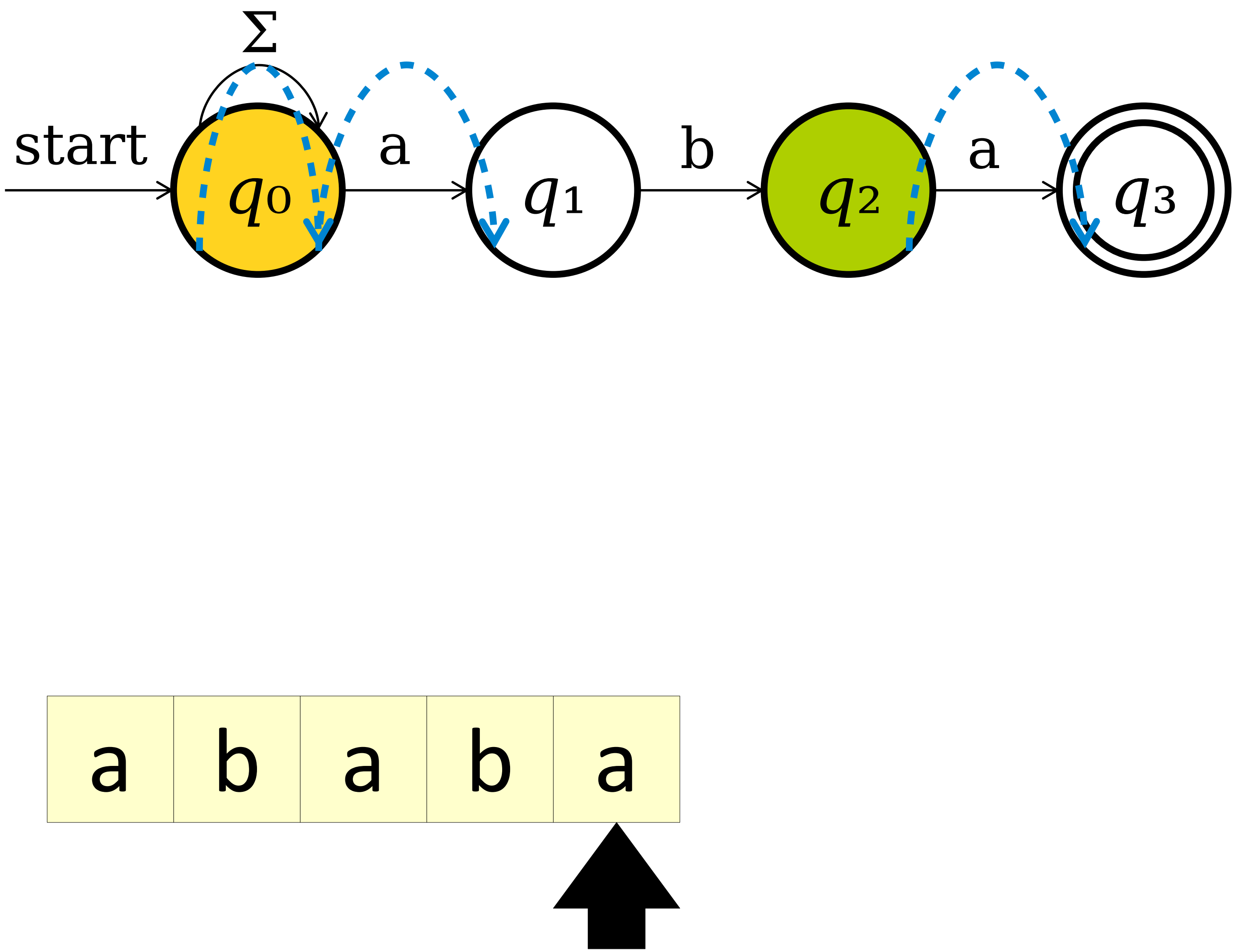


Massive Parallelism

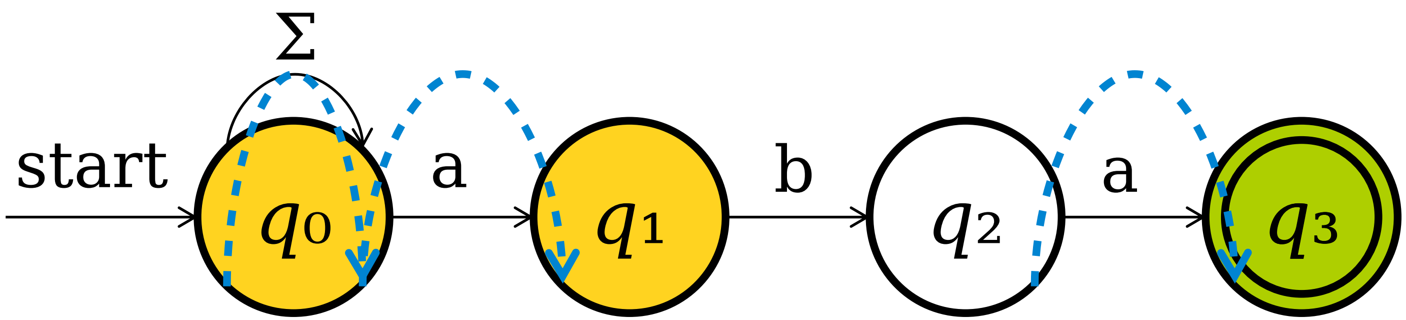


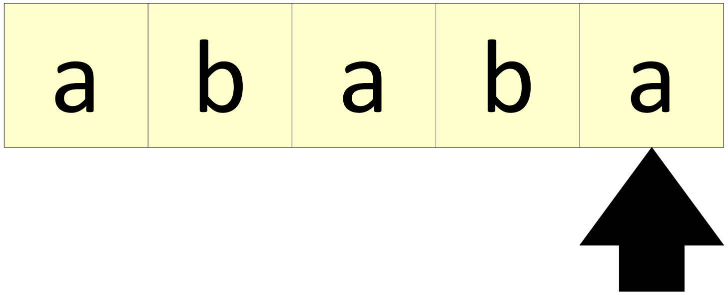


Massive Parallelism

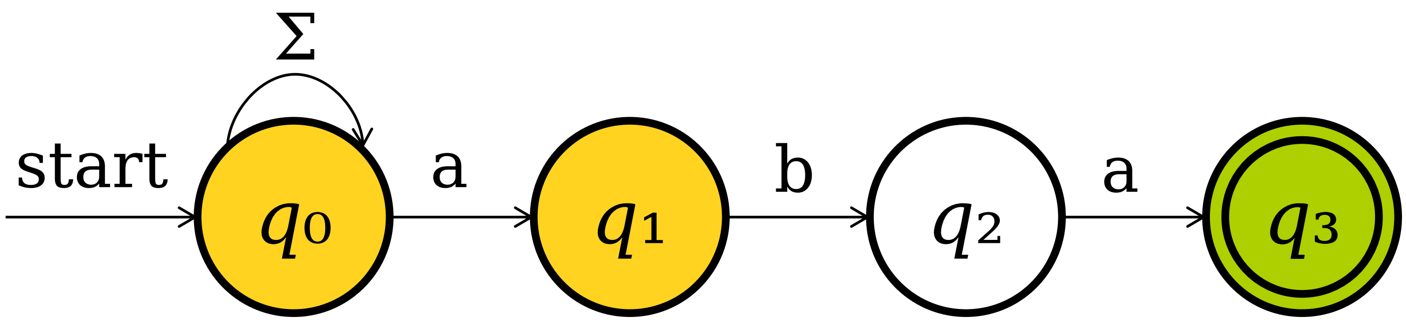


Massive Parallelism



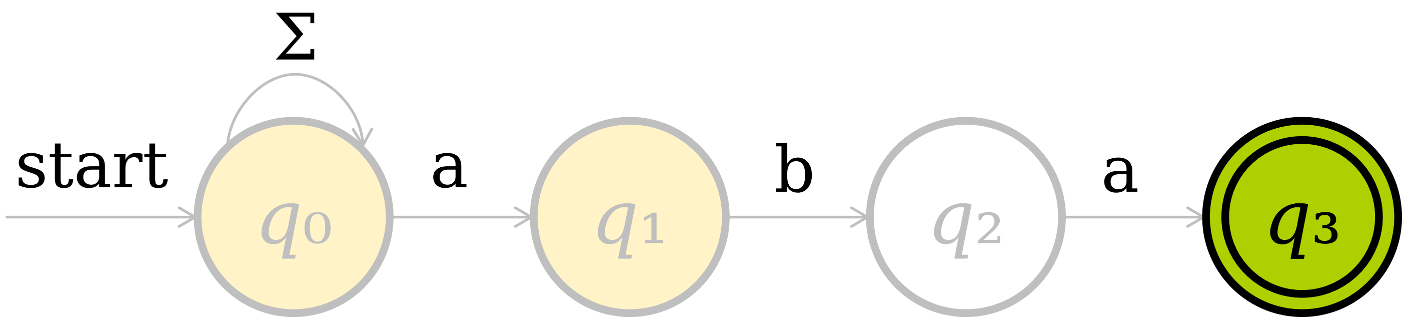


Massive Parallelism



a	b	a	b	a

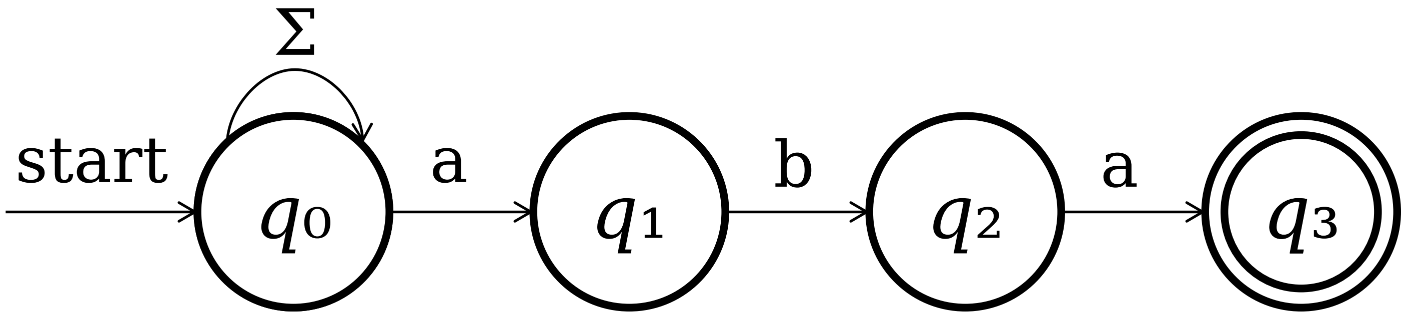
Massive Parallelism



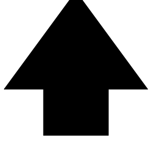
We're in at least one accepting state, so there's some path that gets us to an accepting state.

a	b	a	b	a

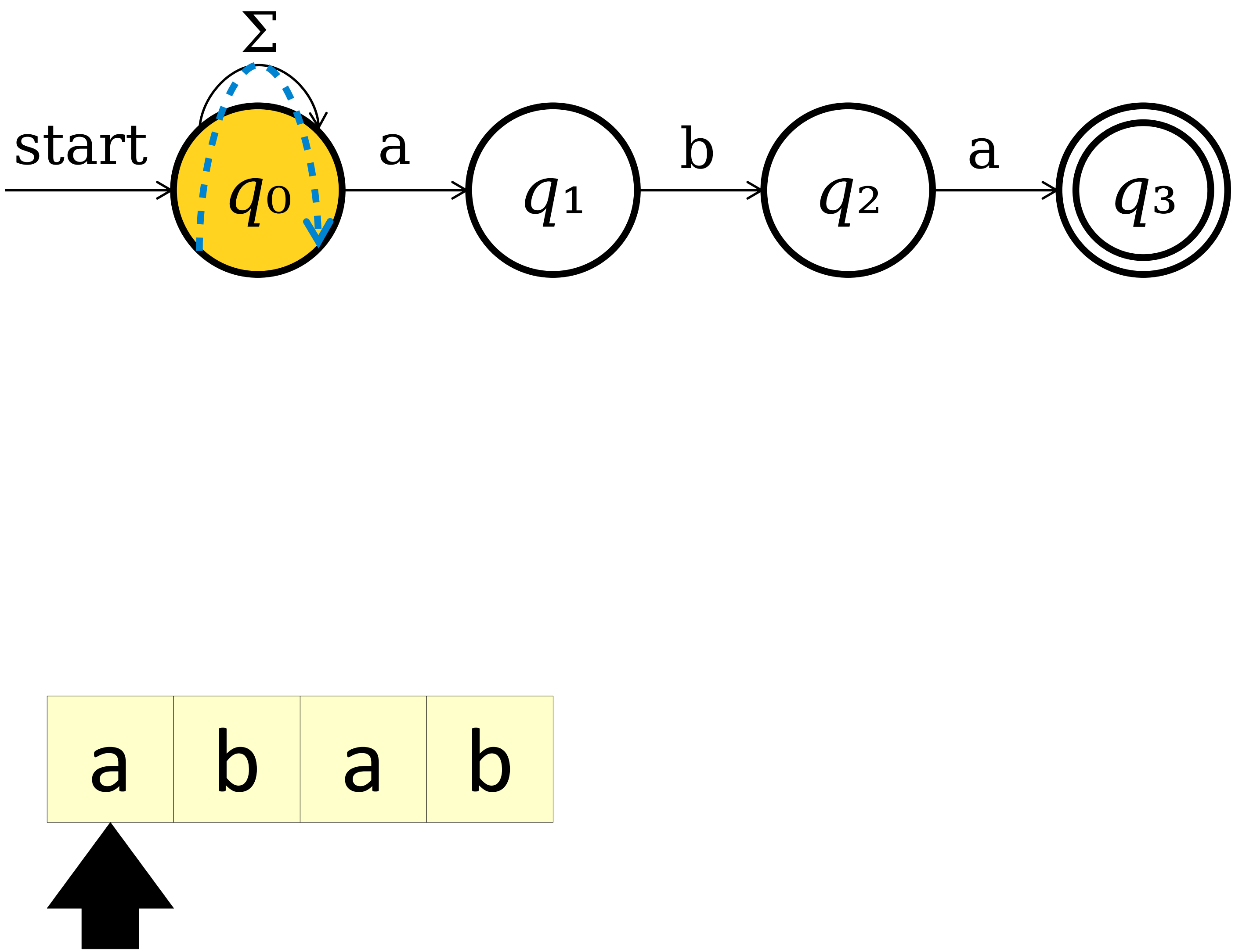
Massive Parallelism



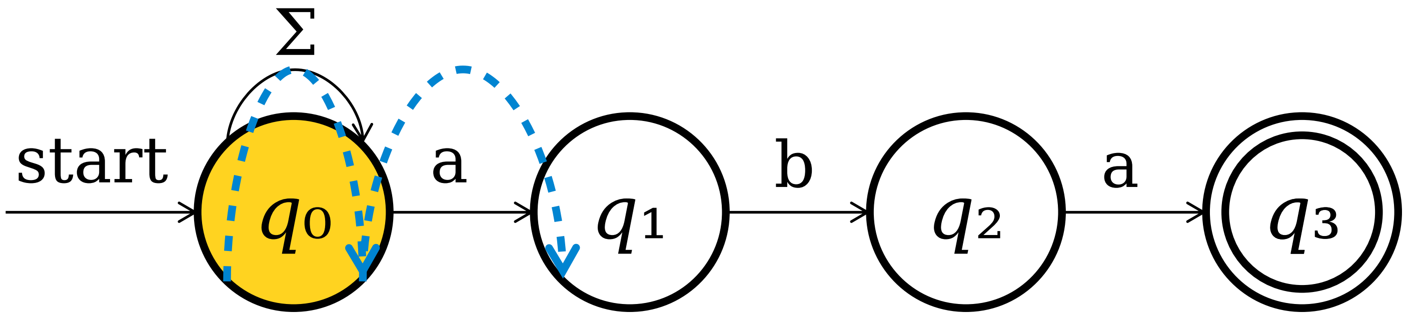
a	b	a	b

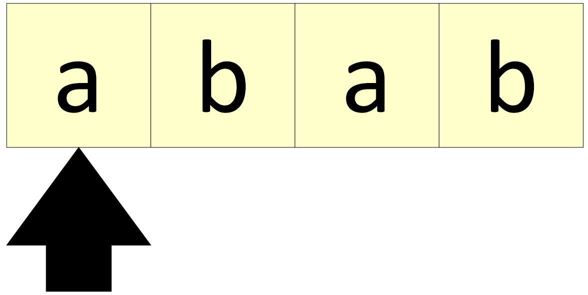


Massive Parallelism

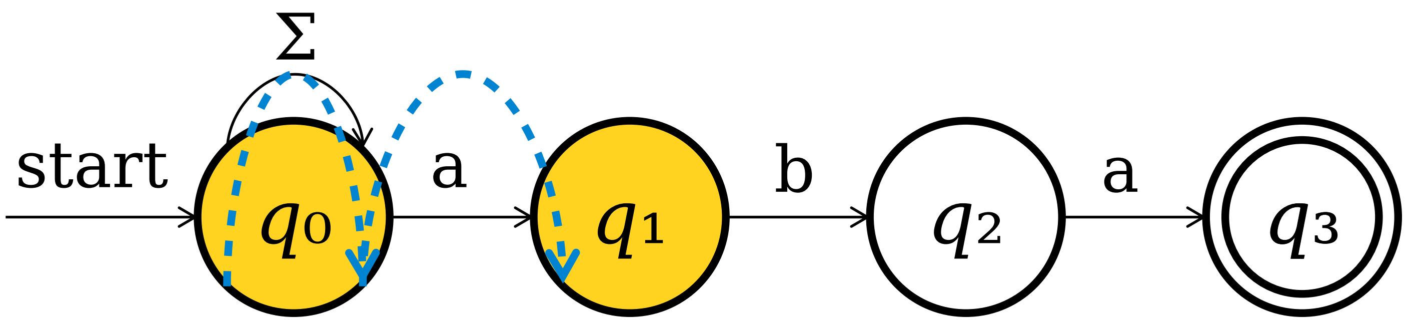


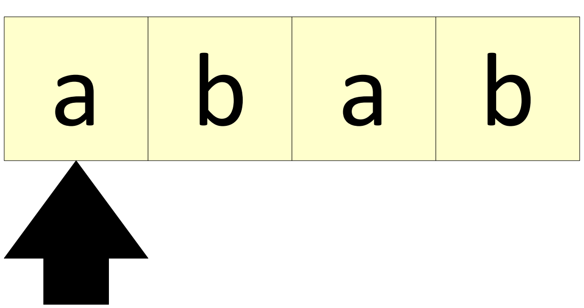
Massive Parallelism



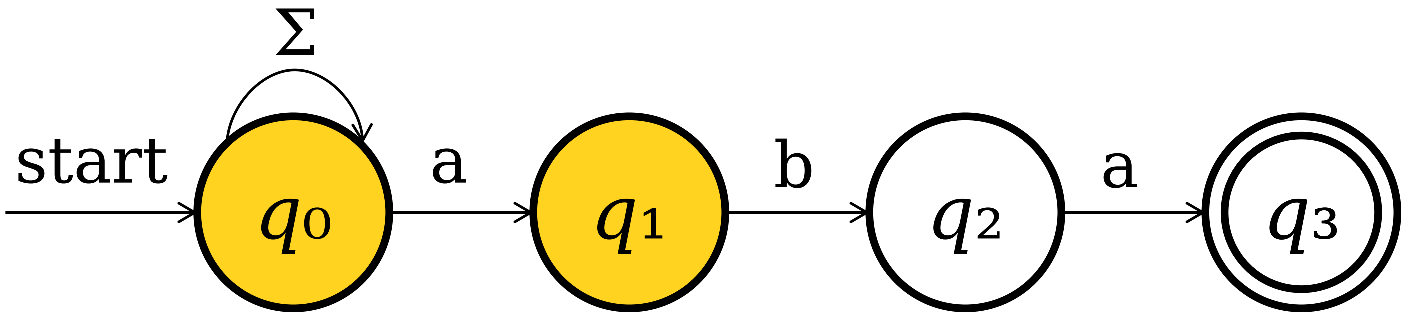


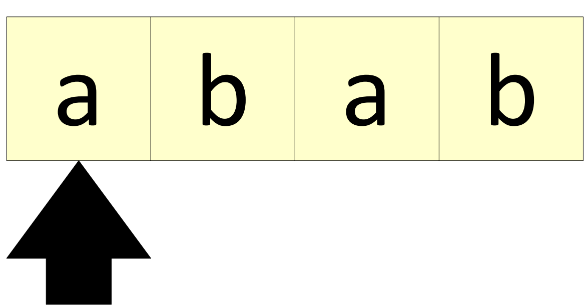
Massive Parallelism



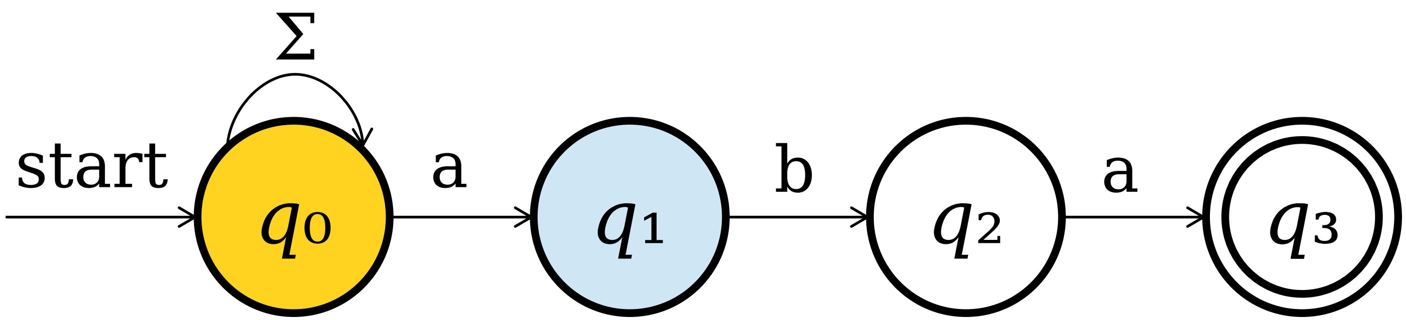


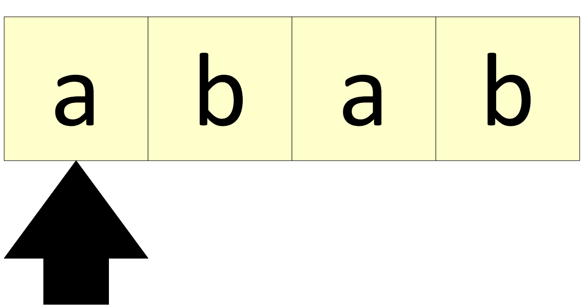
Massive Parallelism



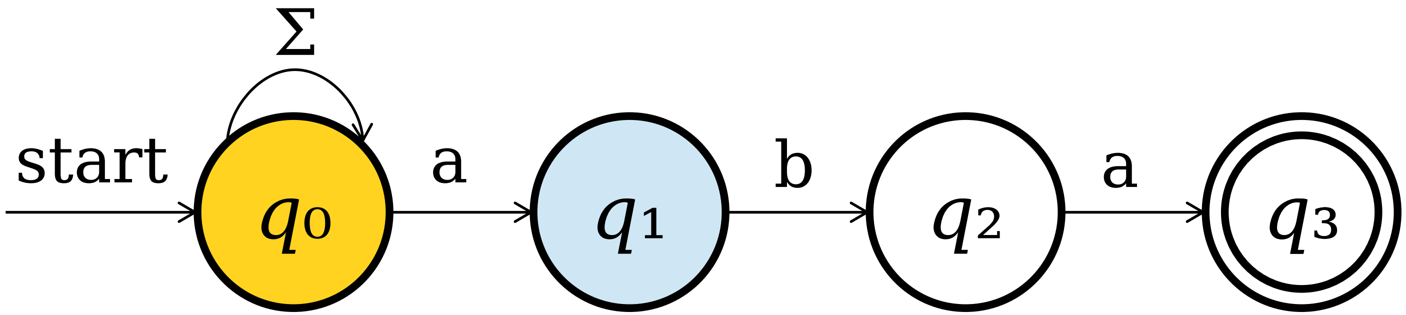


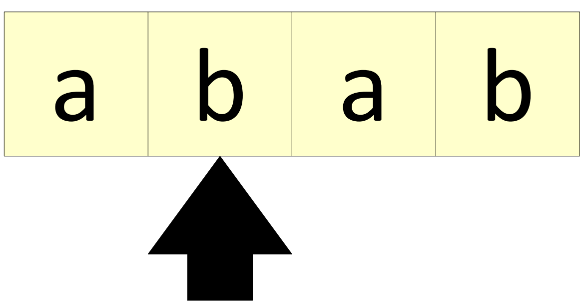
Massive Parallelism



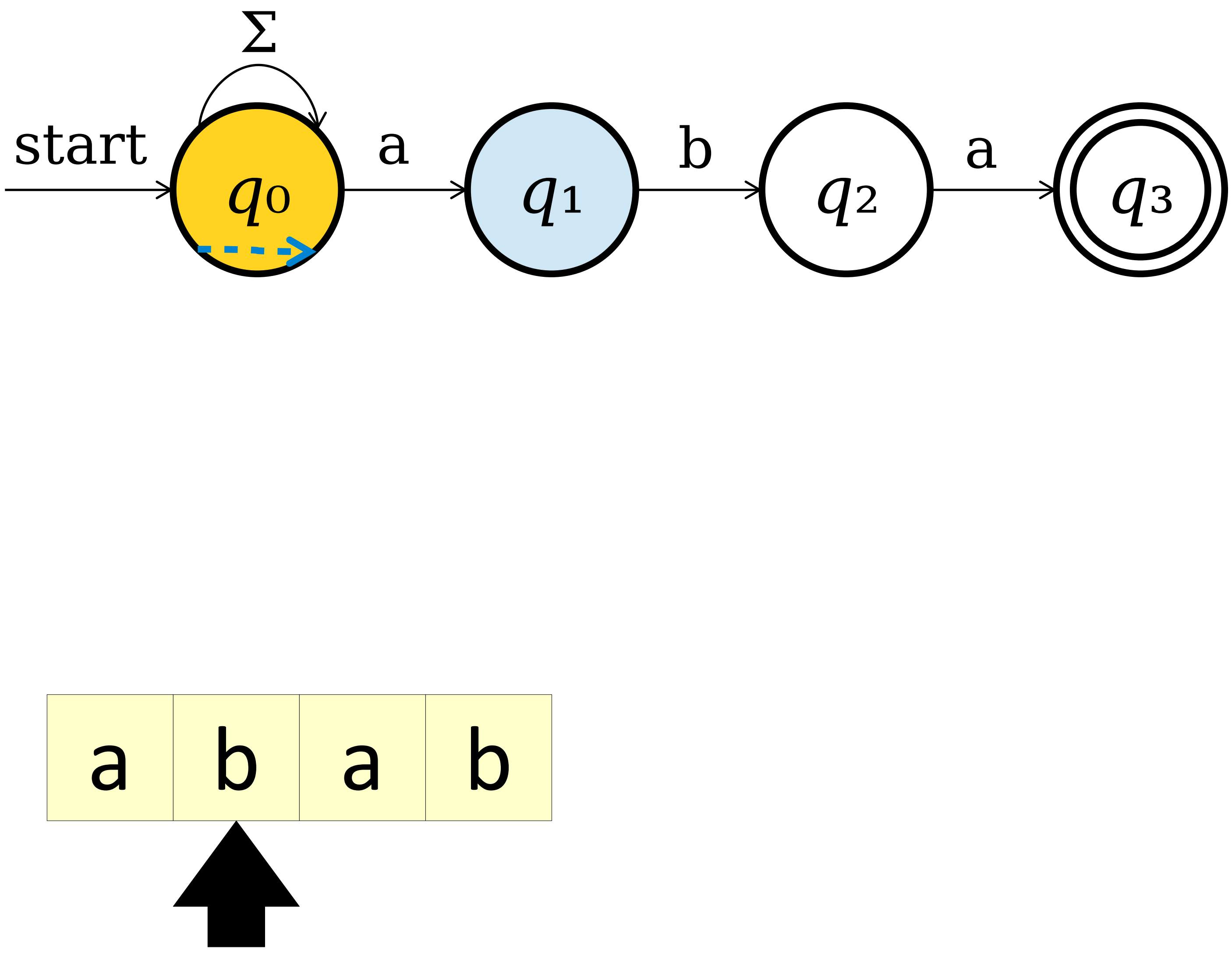


Massive Parallelism

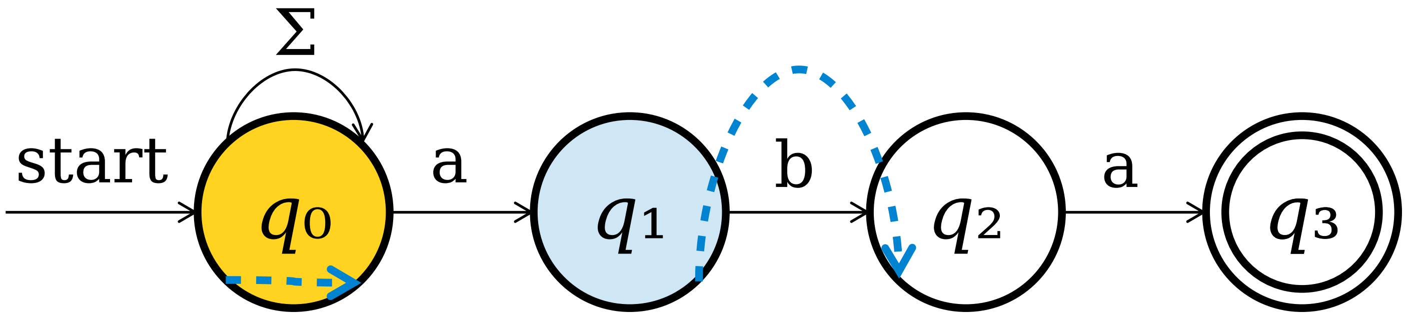


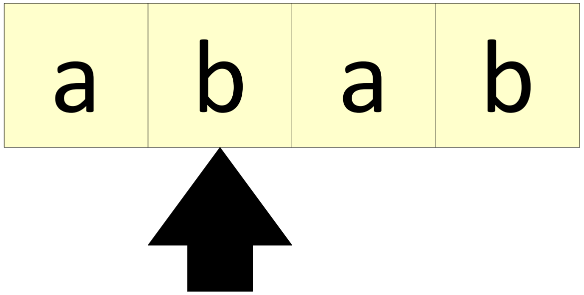


Massive Parallelism

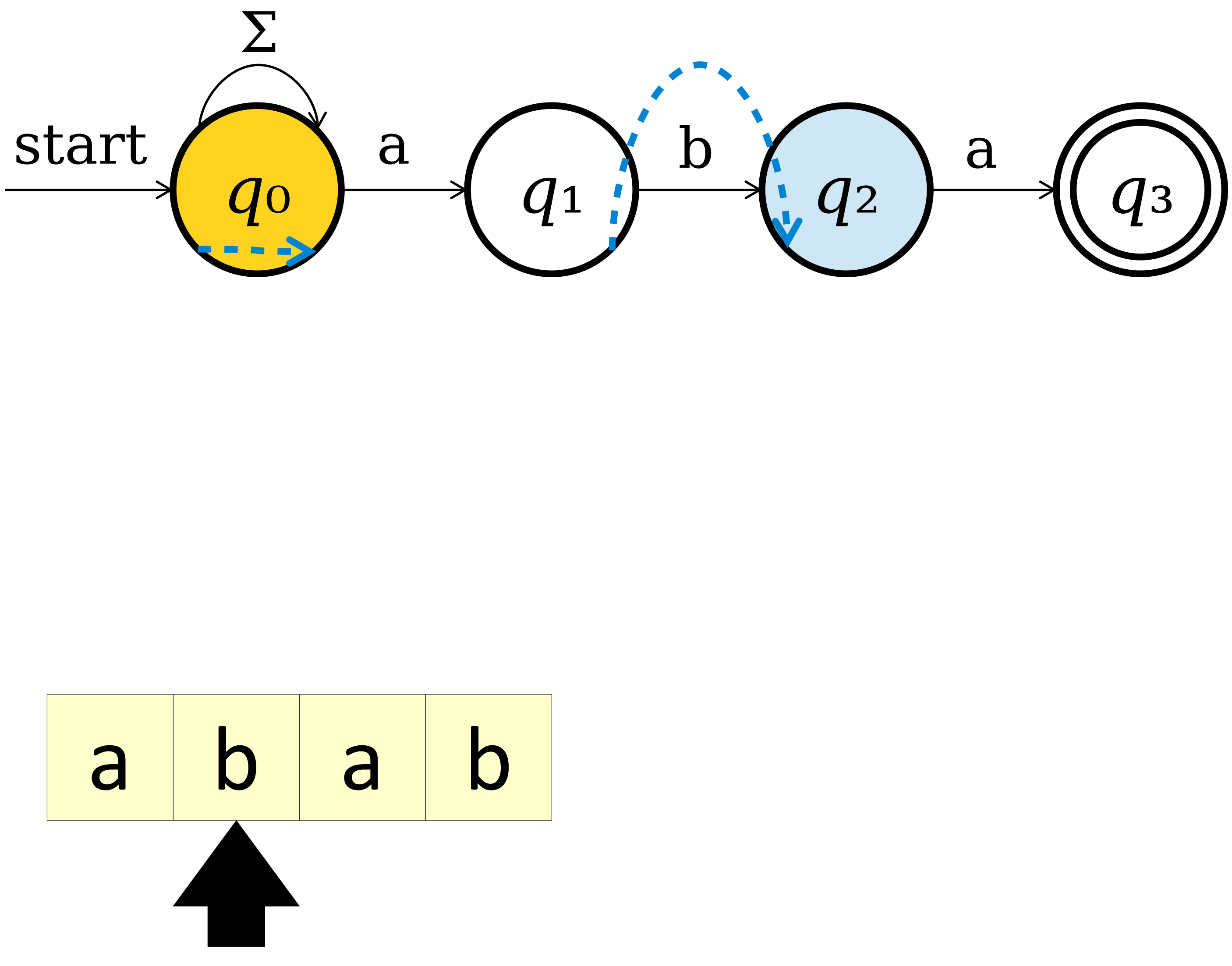


Massive Parallelism

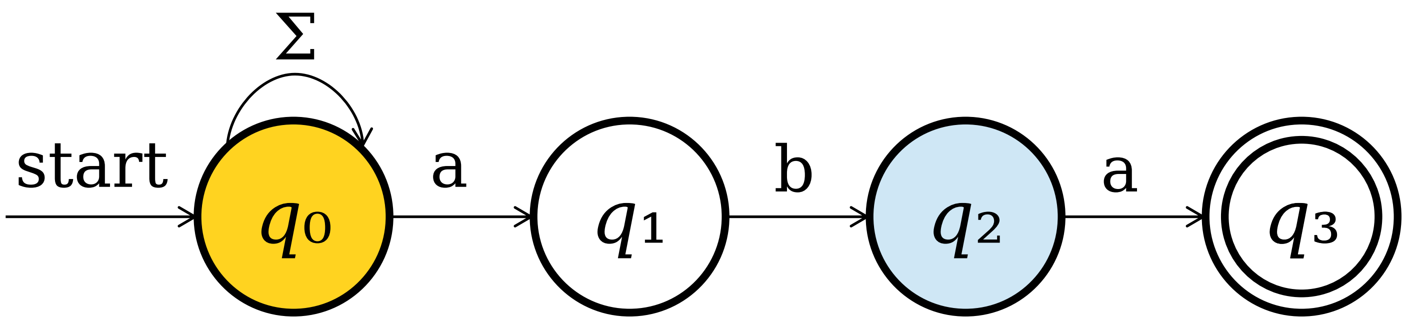


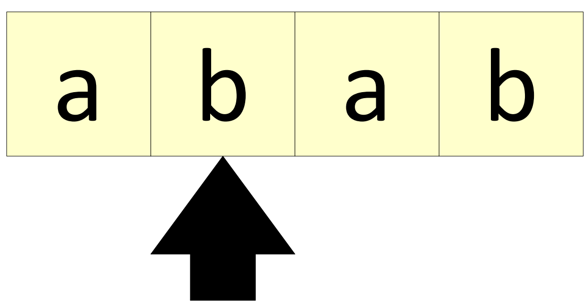


Massive Parallelism

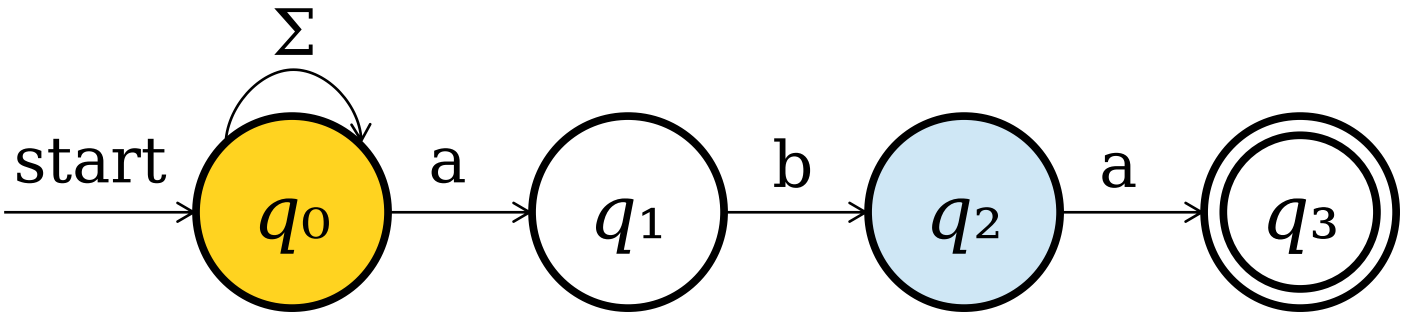


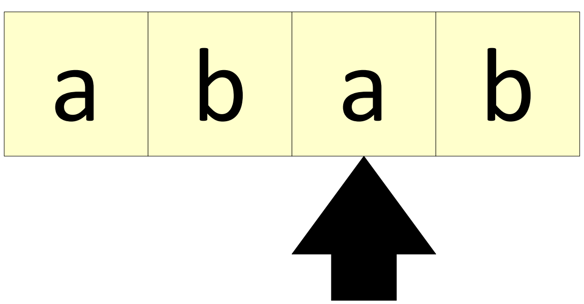
Massive Parallelism



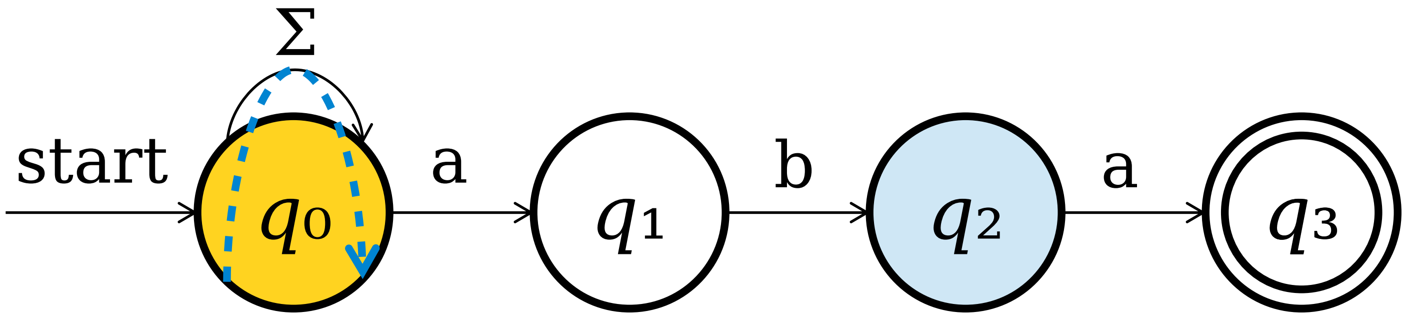


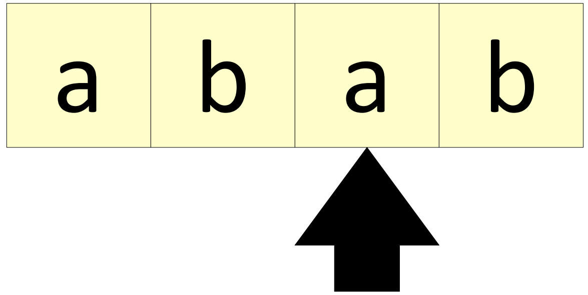
Massive Parallelism



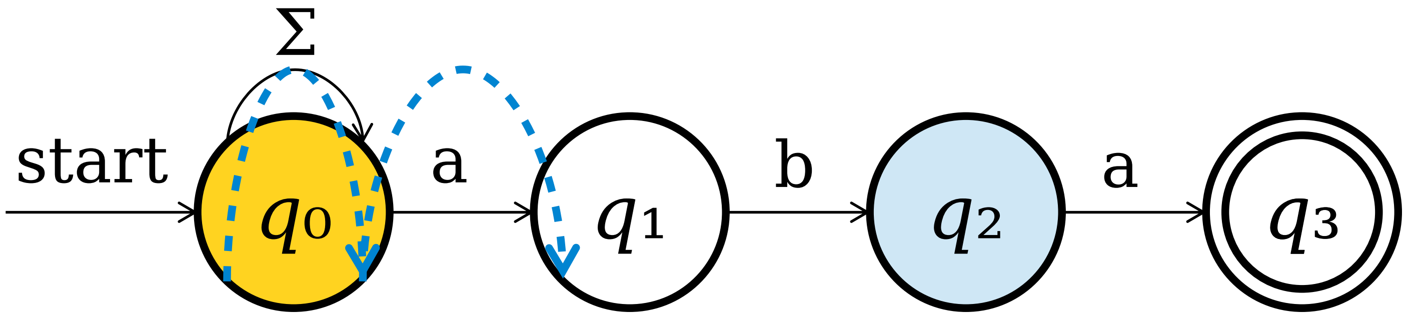


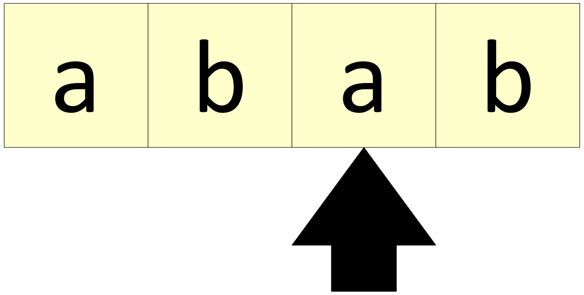
Massive Parallelism



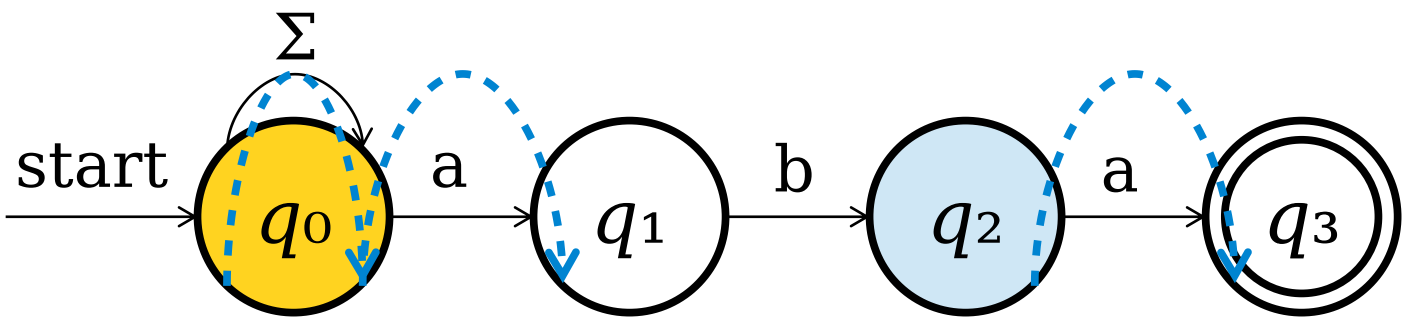


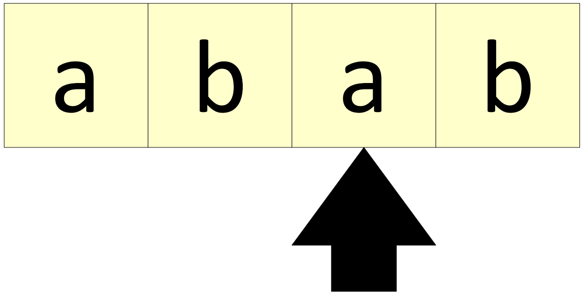
Massive Parallelism



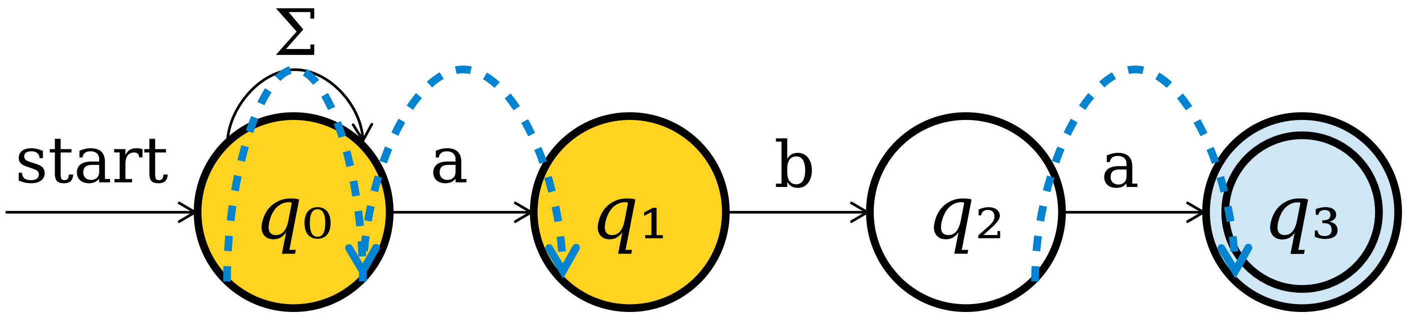


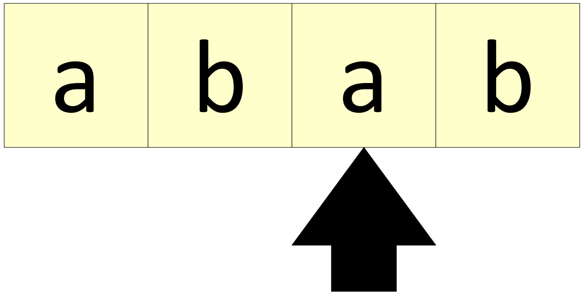
Massive Parallelism



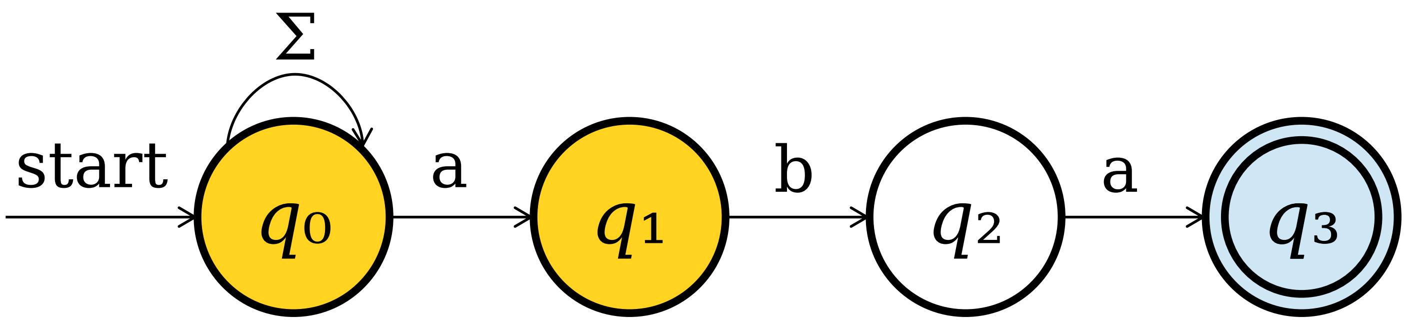


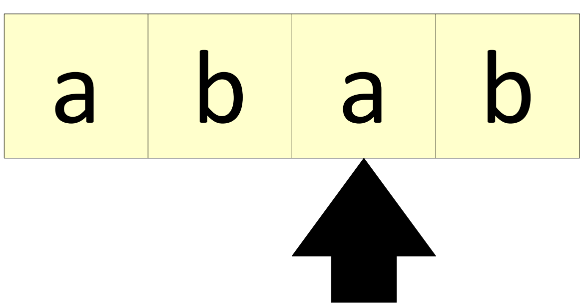
Massive Parallelism



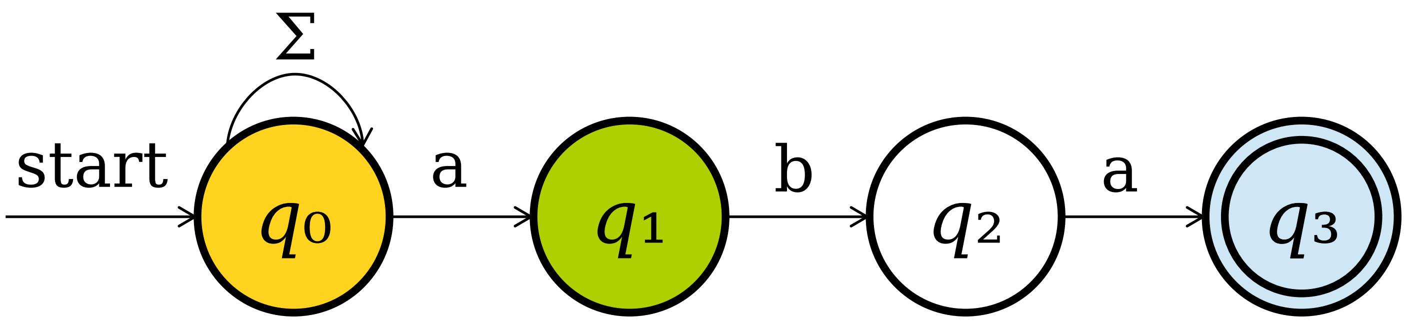


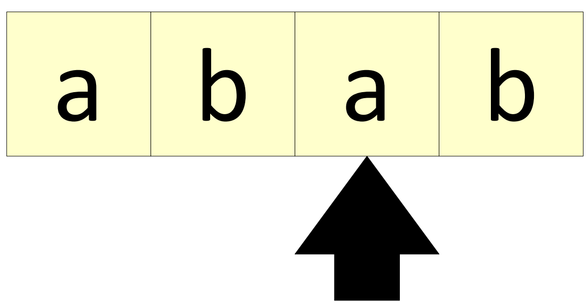
Massive Parallelism



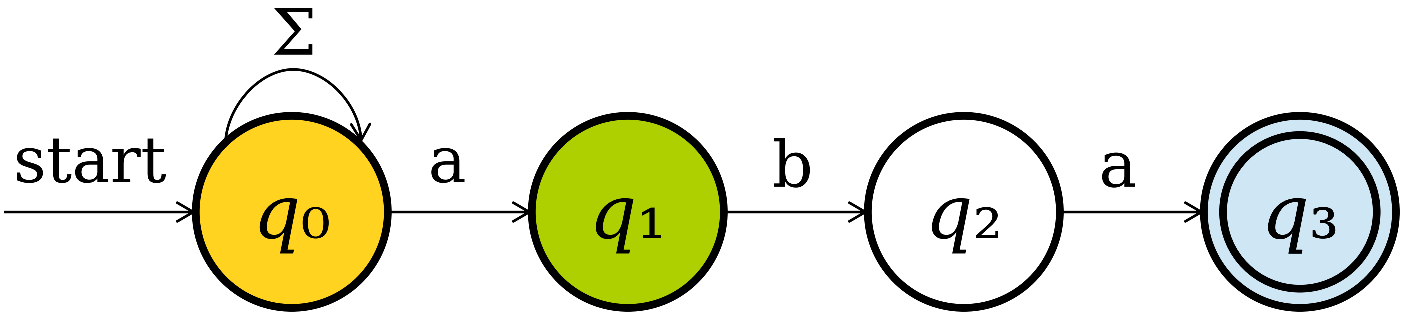


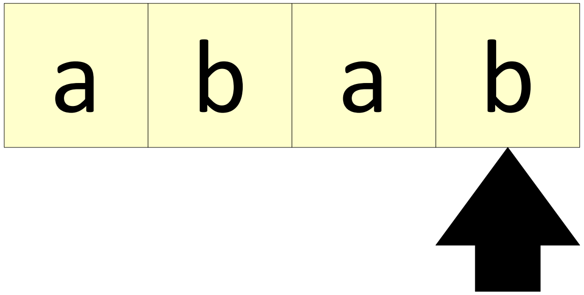
Massive Parallelism



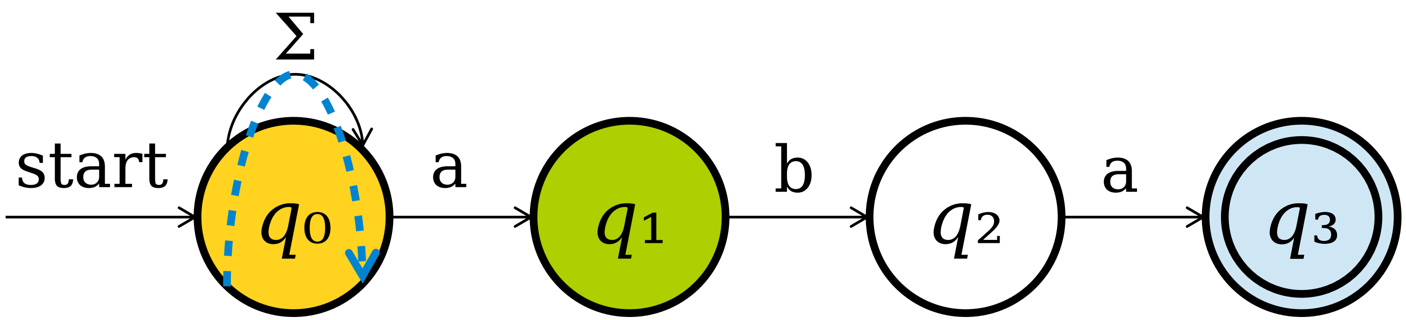


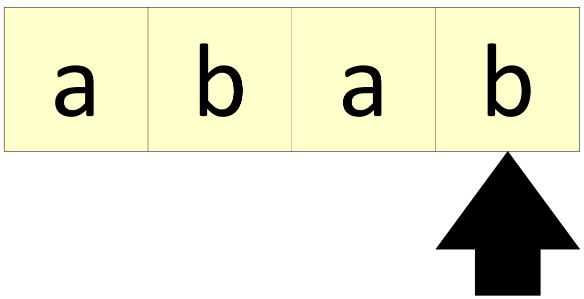
Massive Parallelism



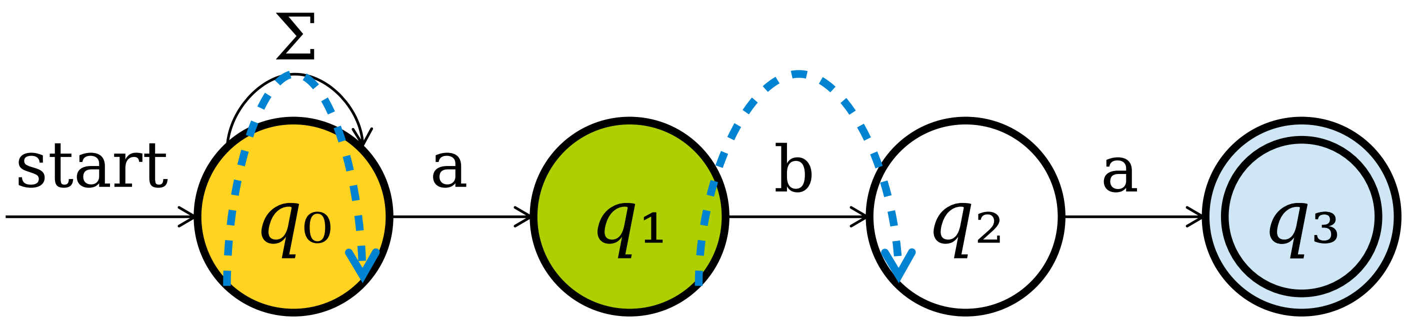


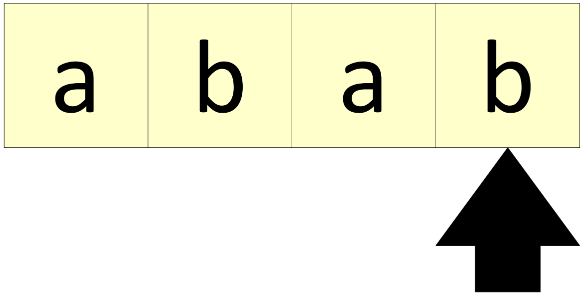
Massive Parallelism



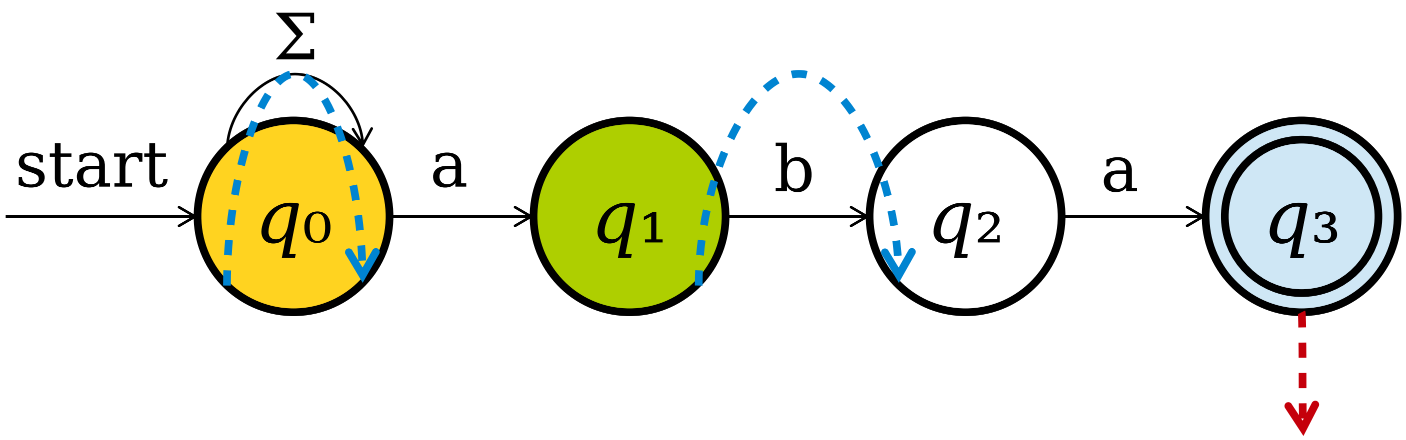


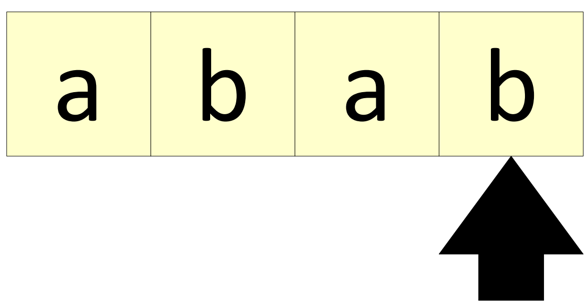
Massive Parallelism



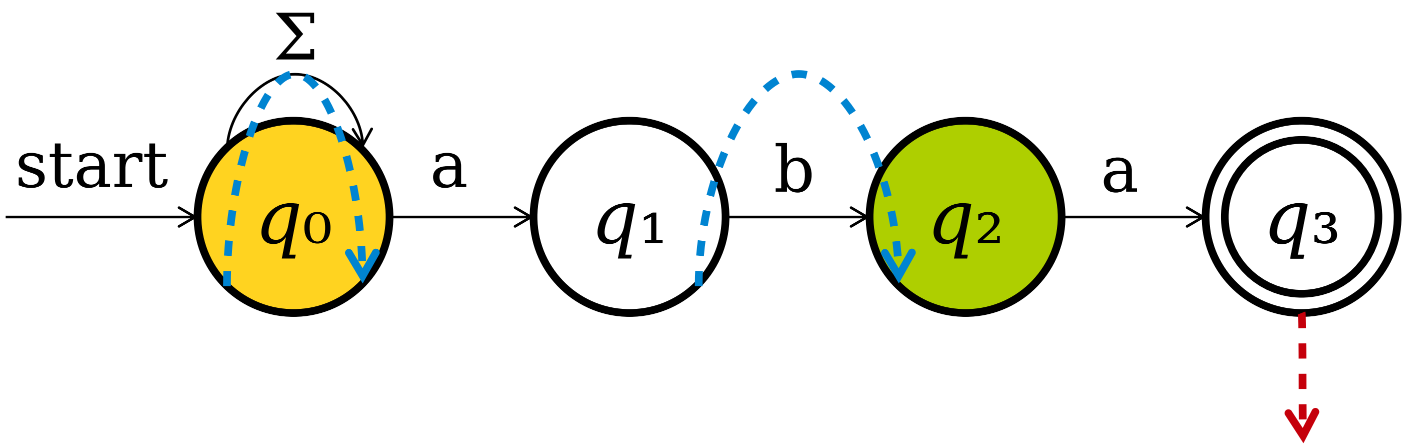


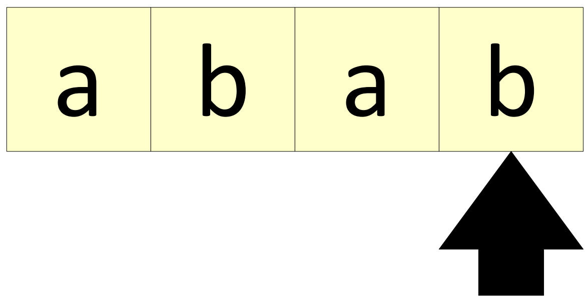
Massive Parallelism



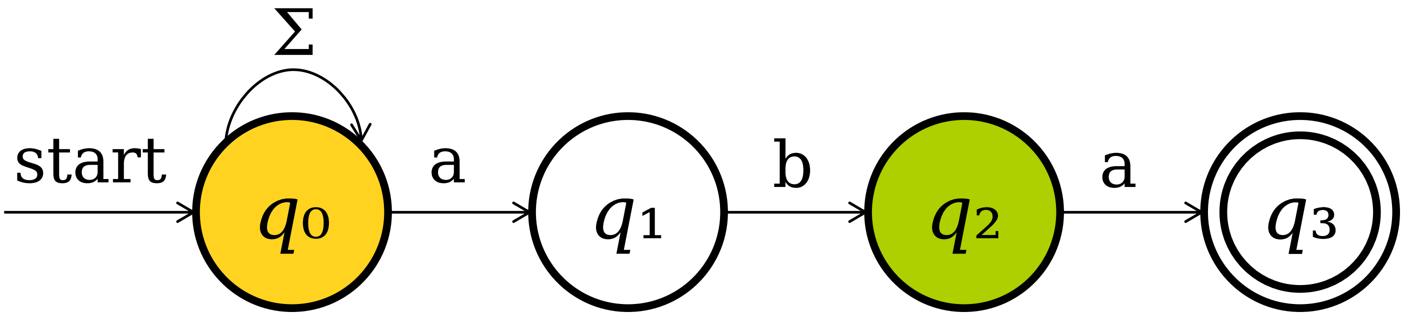


Massive Parallelism



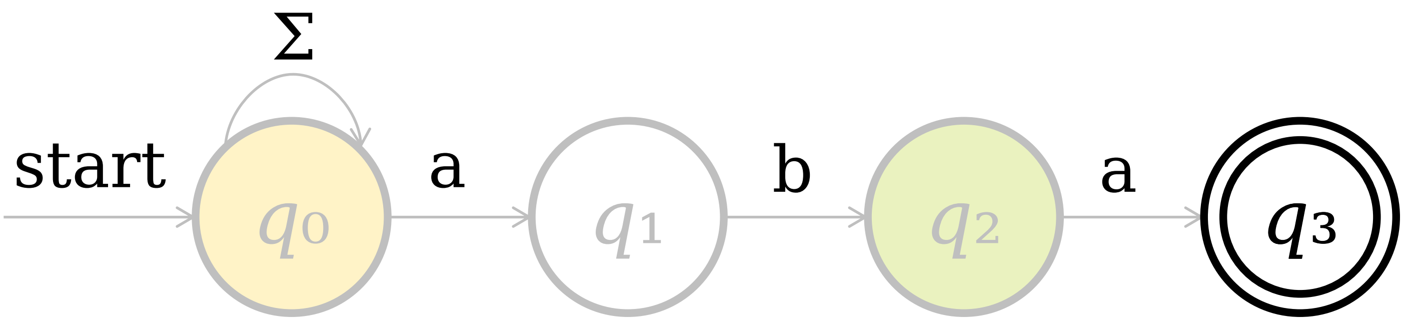


Massive Parallelism



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.

So What?

Each intuition of nondeterminism is useful in a different setting:

Perfect guessing is a great way to think about how to design a machine.

Massive parallelism is a great way to test machines – and has nice theoretical implications.

Nondeterministic machines may not be feasible, but they give a great basis for interesting questions:

Can any problem that can be solved by a nondeterministic machine be solved by a deterministic machine?

Can any problem that can be solved by a nondeterministic machine be solved ***efficiently*** by a deterministic machine?

The answers vary from automaton to automaton.

Designing NFAs

Designing NFAs

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.

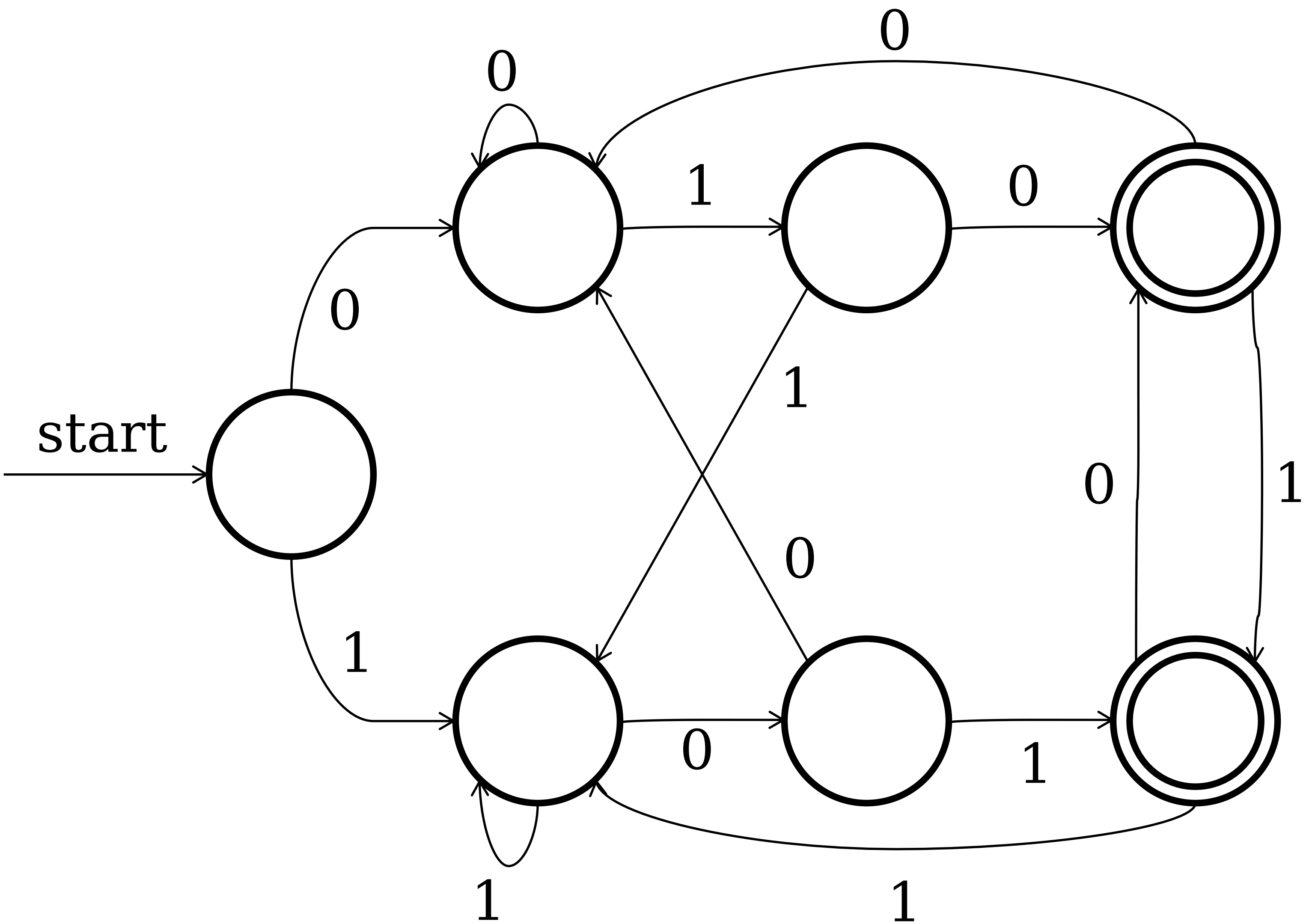
The *check* phase corresponds to filtering out bad guesses or wrong options.

Guess-and-Check

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

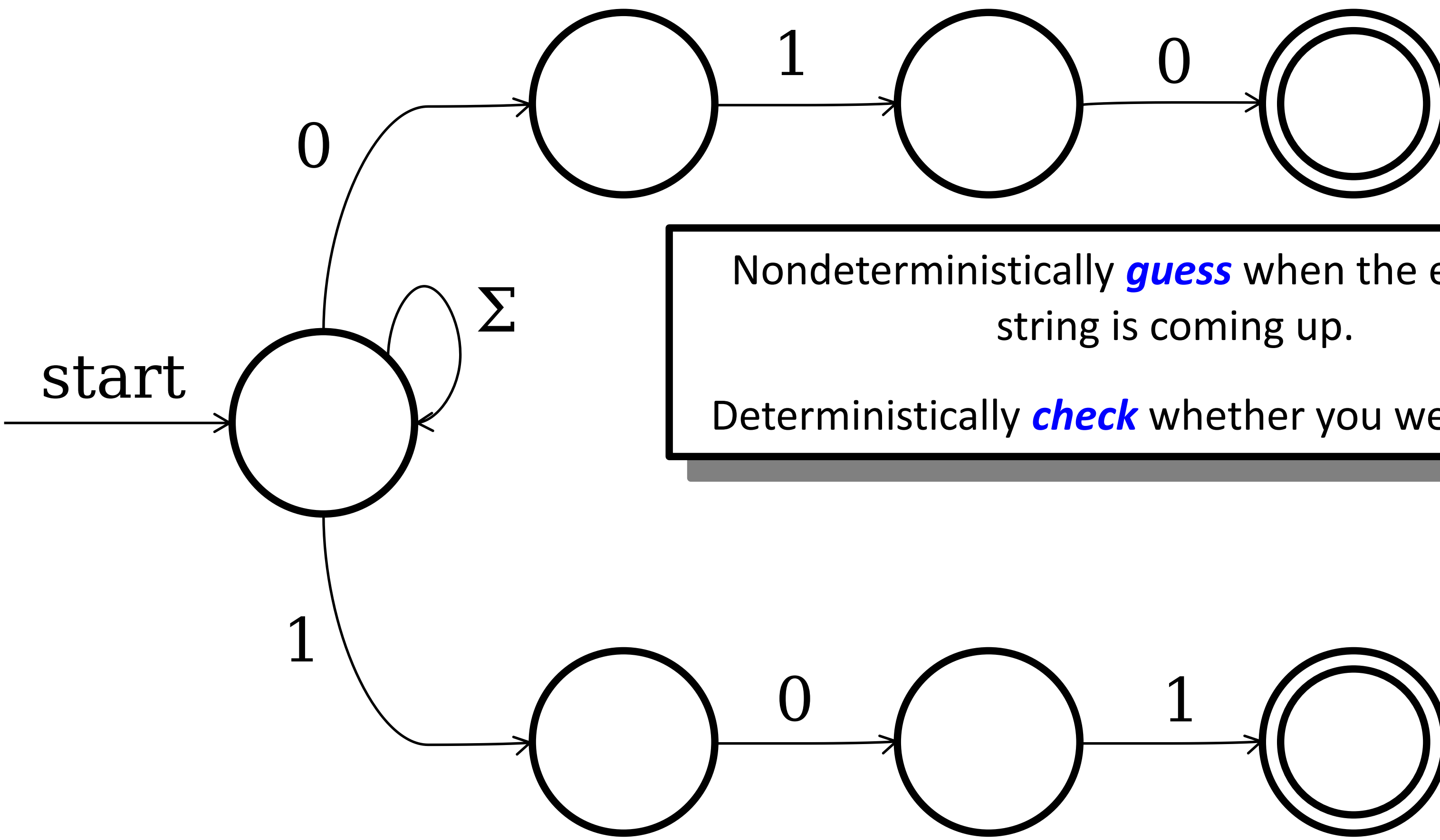
Guess-and-Check

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



Guess-and-Check

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

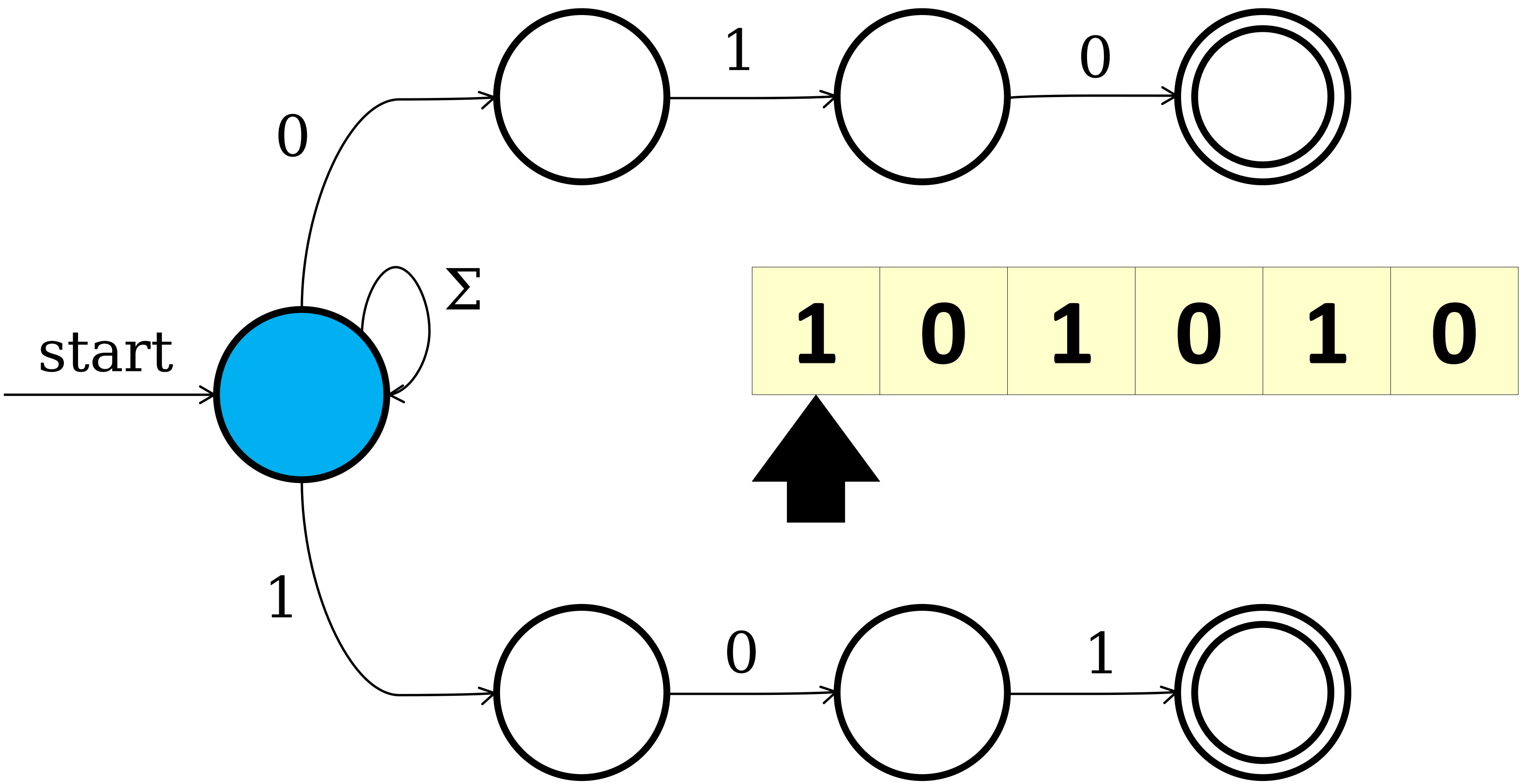


Nondeterministically ***guess*** when the end of the string is coming up.

Deterministically ***check*** whether you were correct.

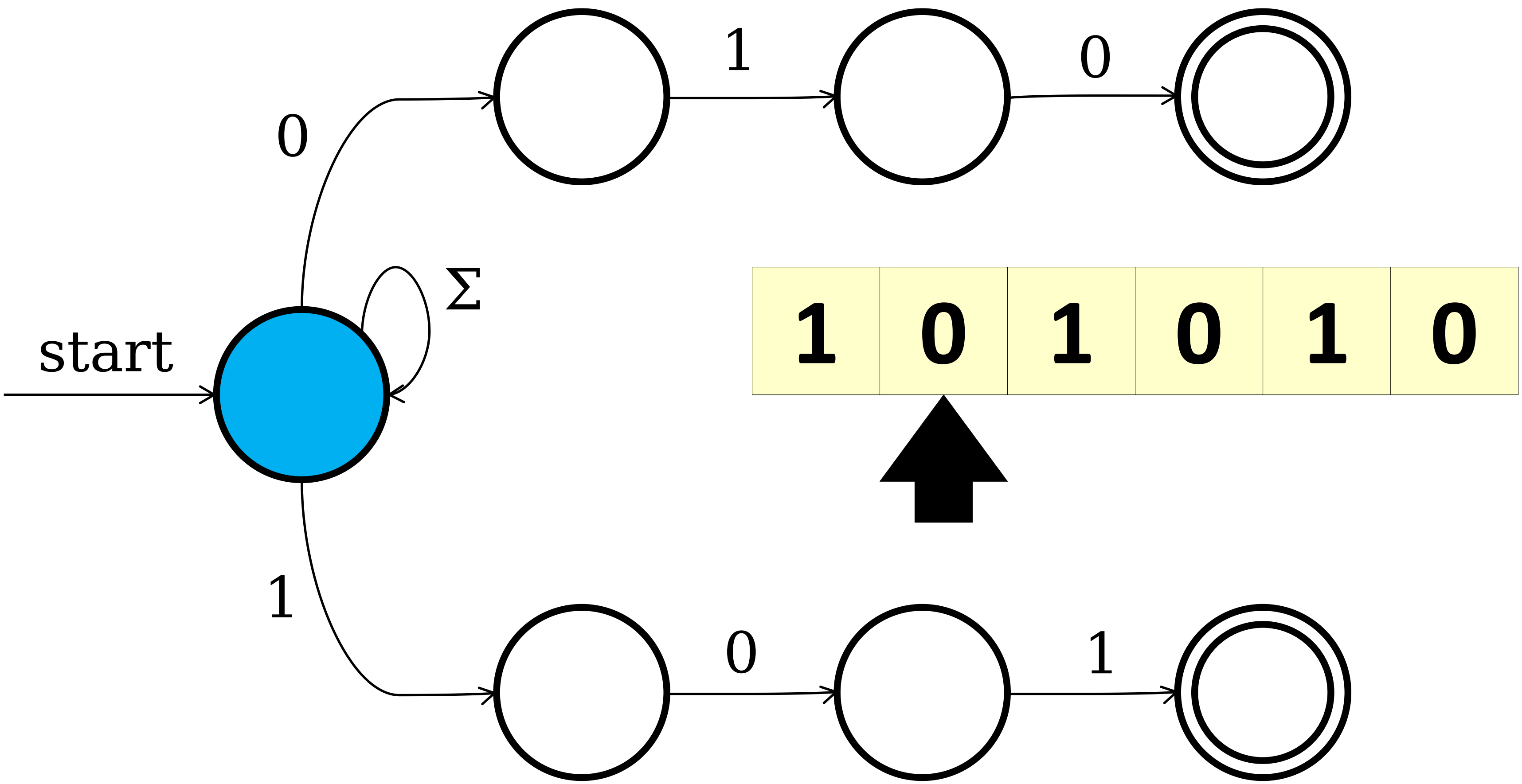
Guess-and-Check

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



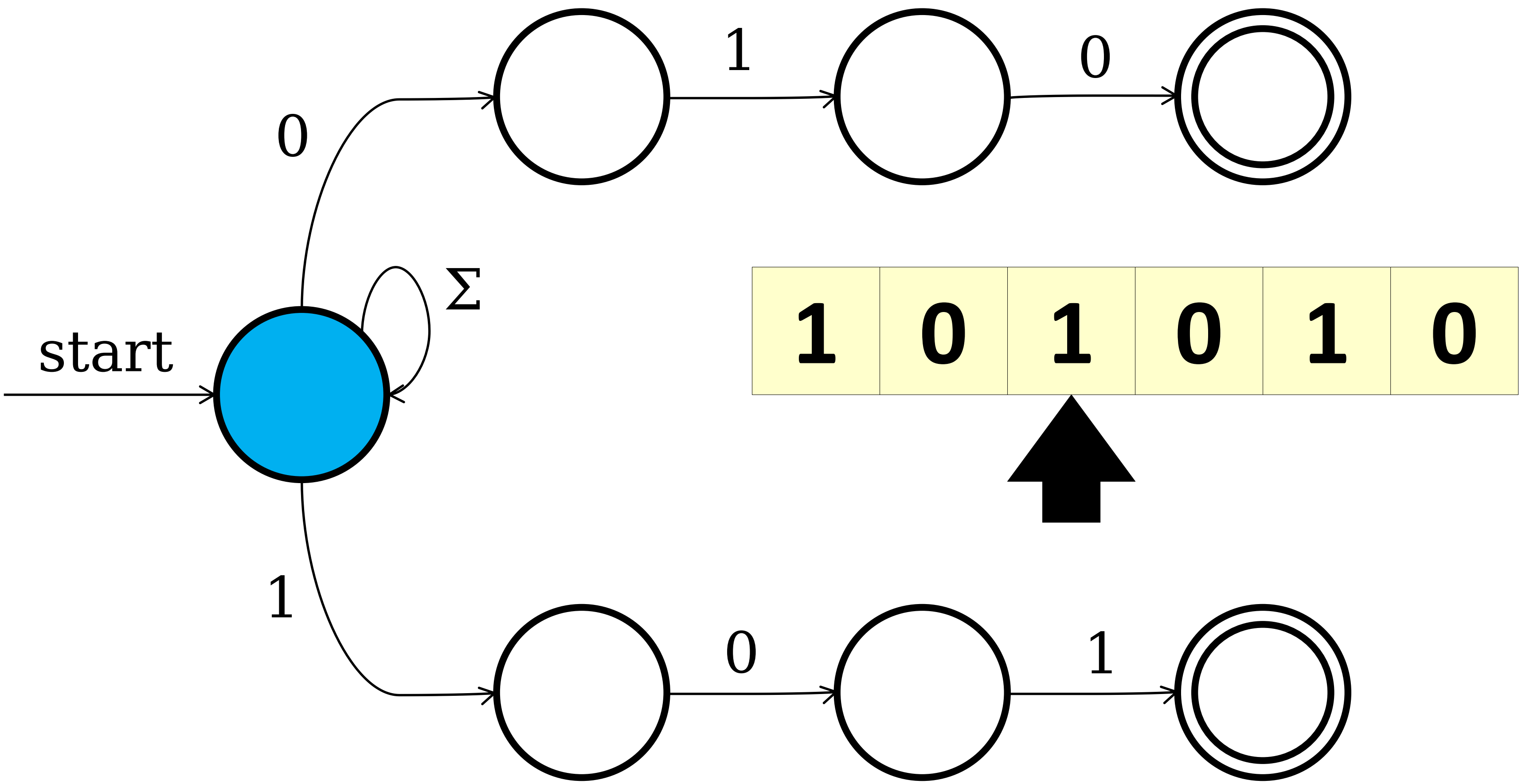
Guess-and-Check

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



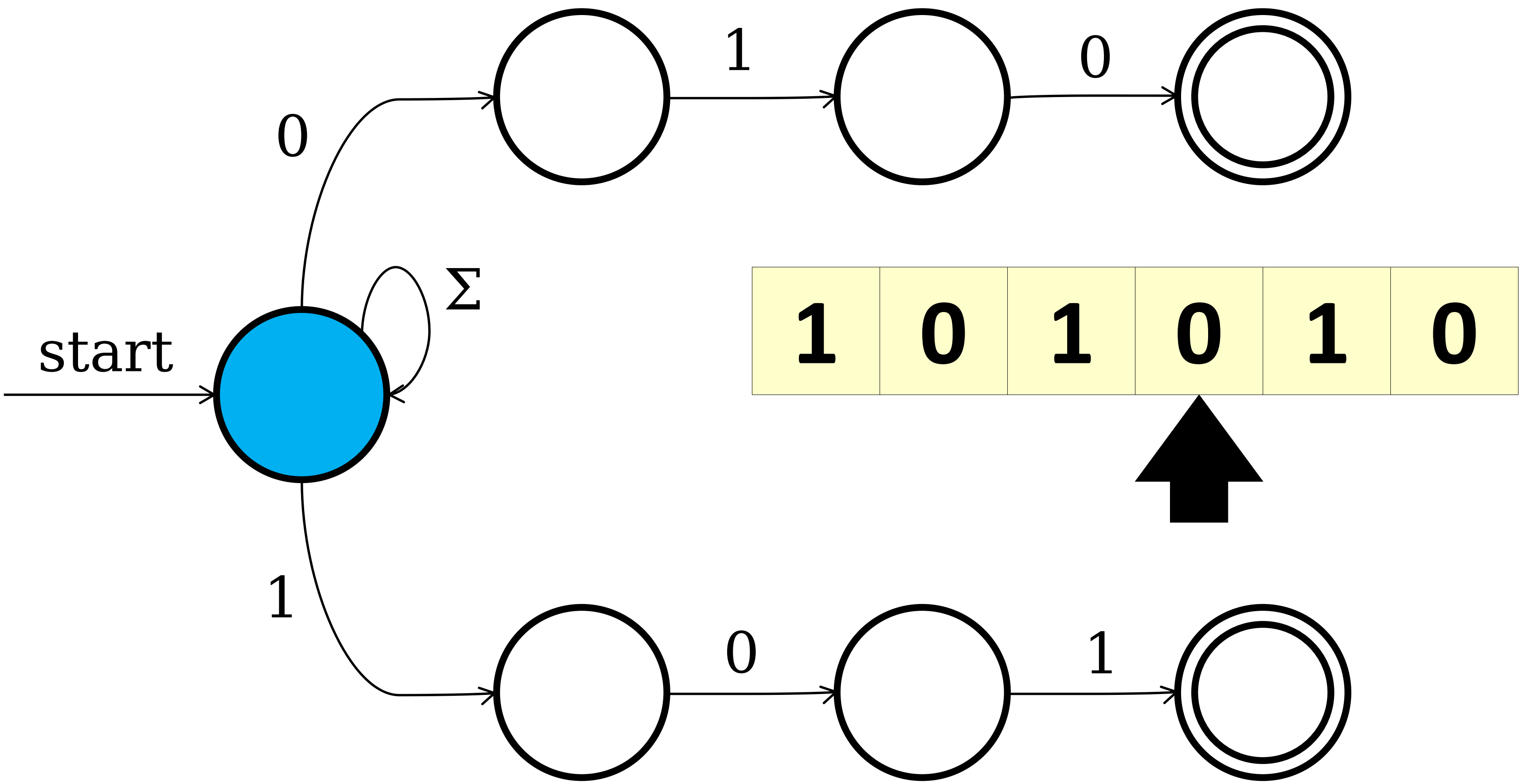
Guess-and-Check

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



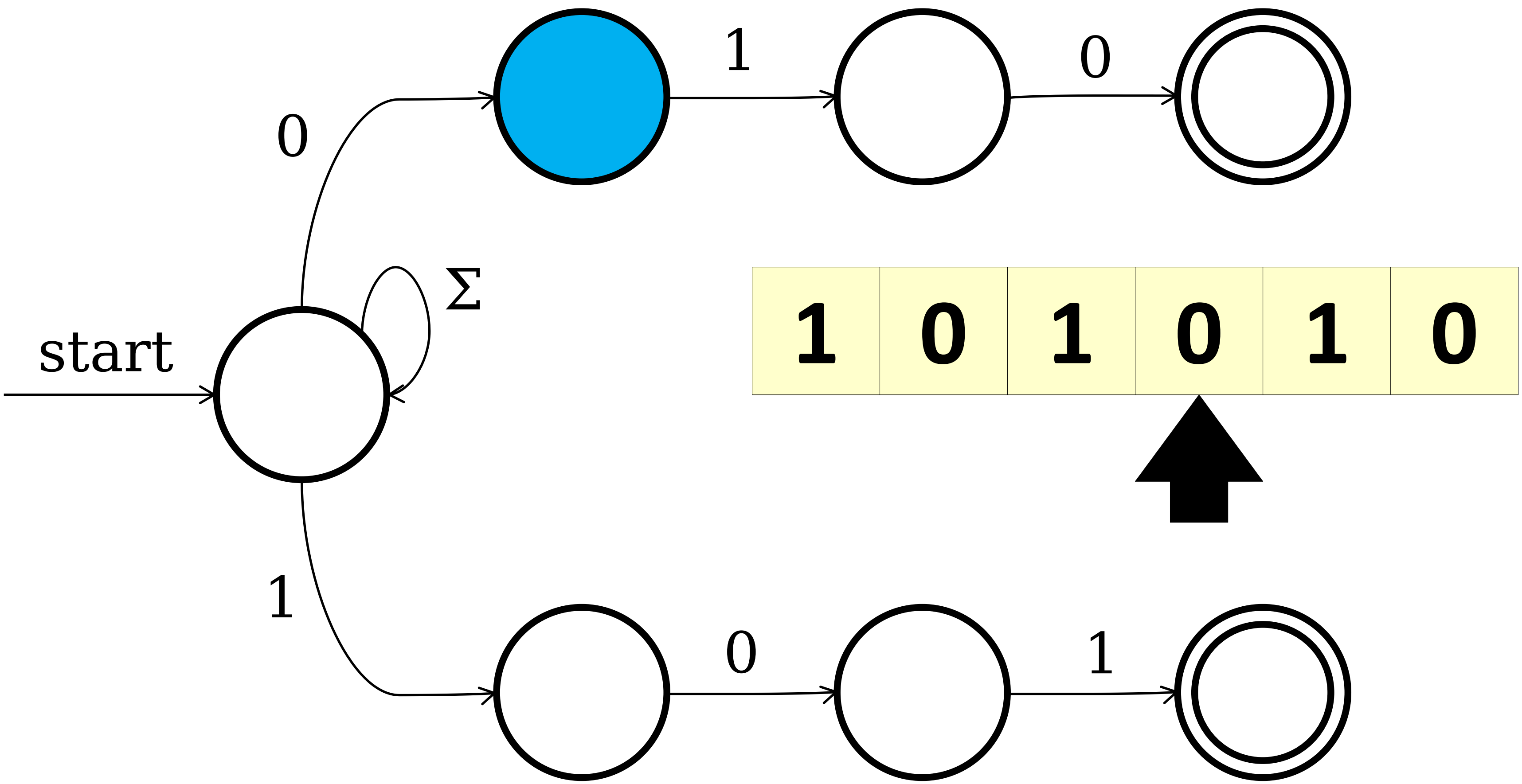
Guess-and-Check

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



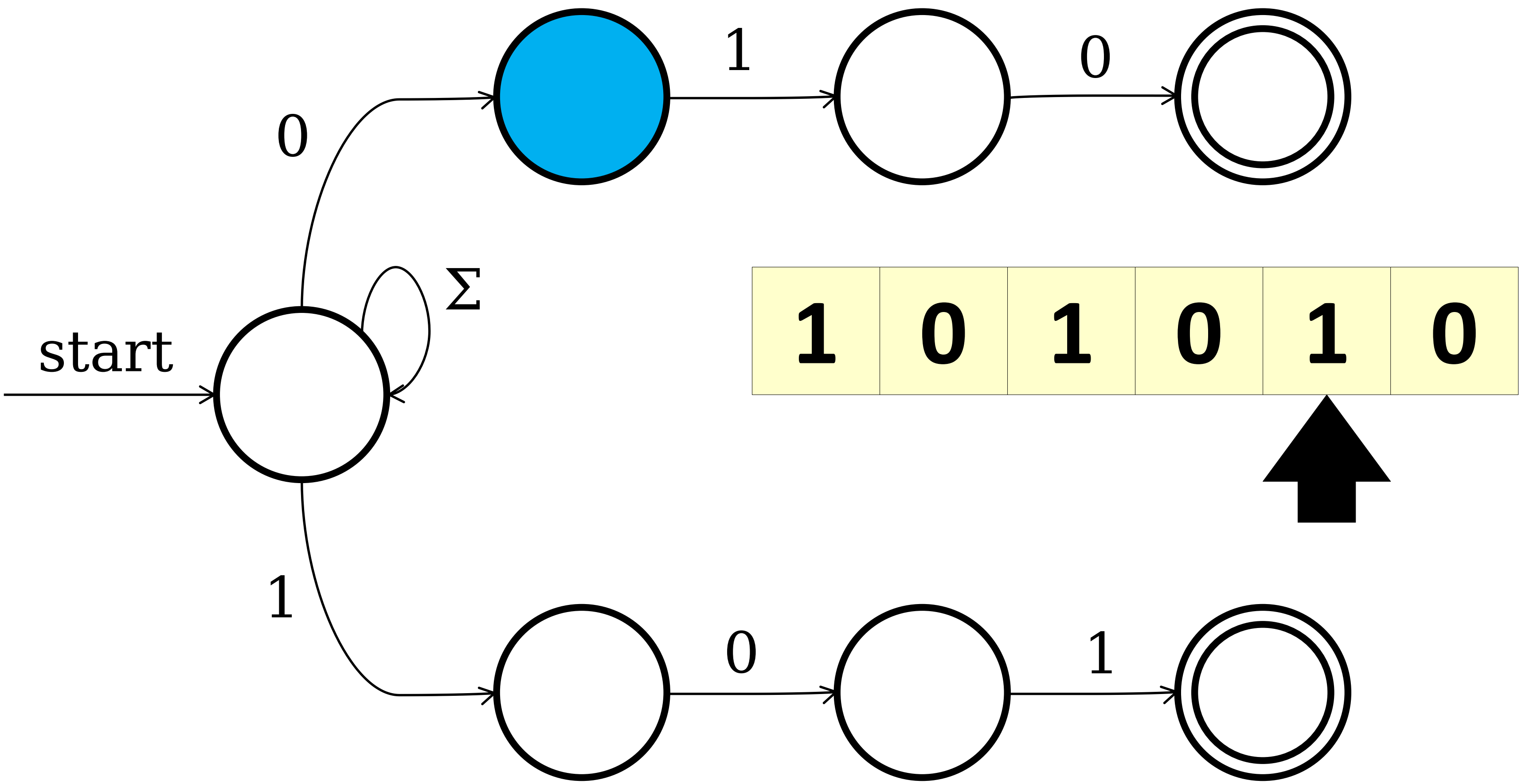
Guess-and-Check

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



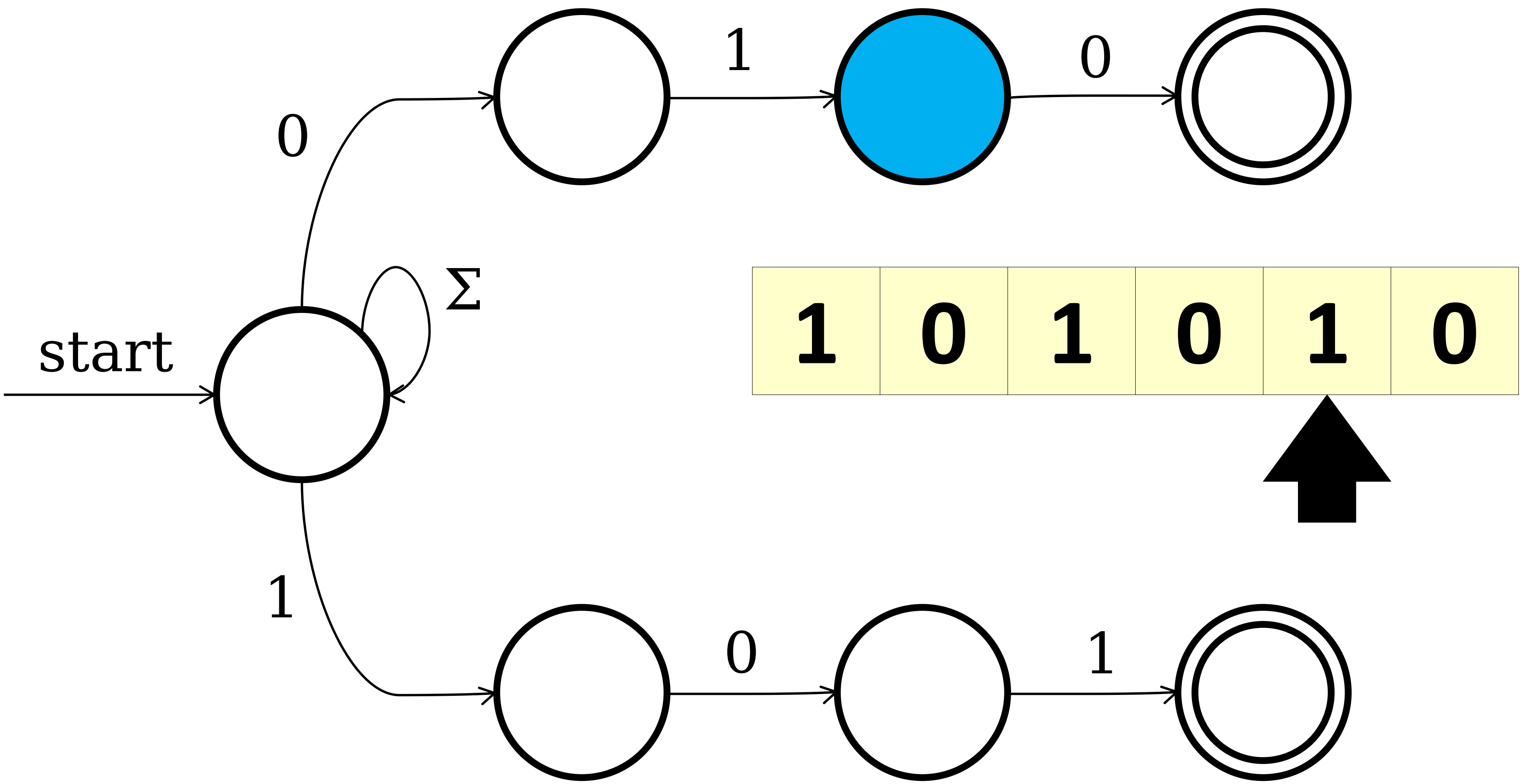
Guess-and-Check

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



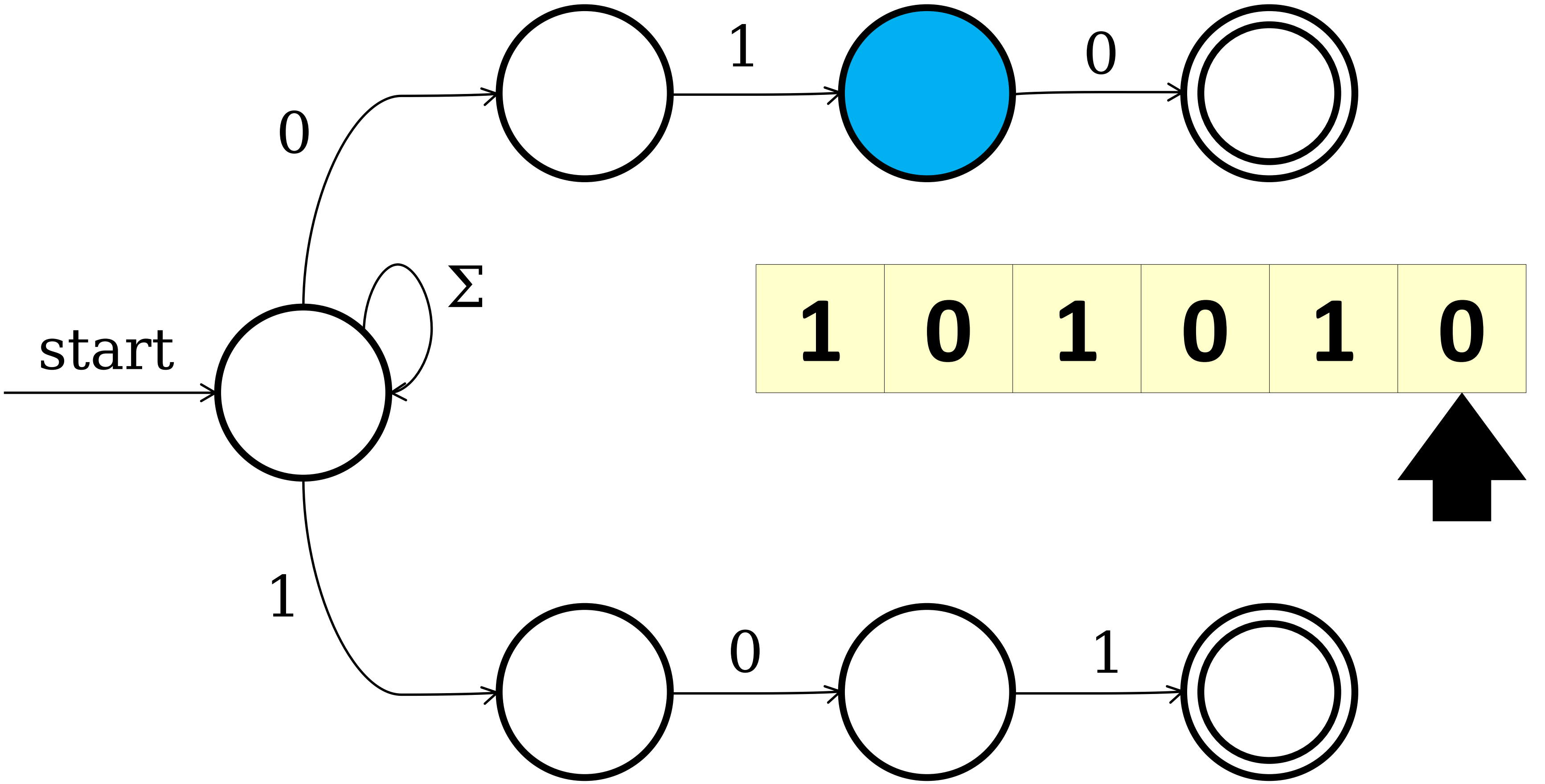
Guess-and-Check

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



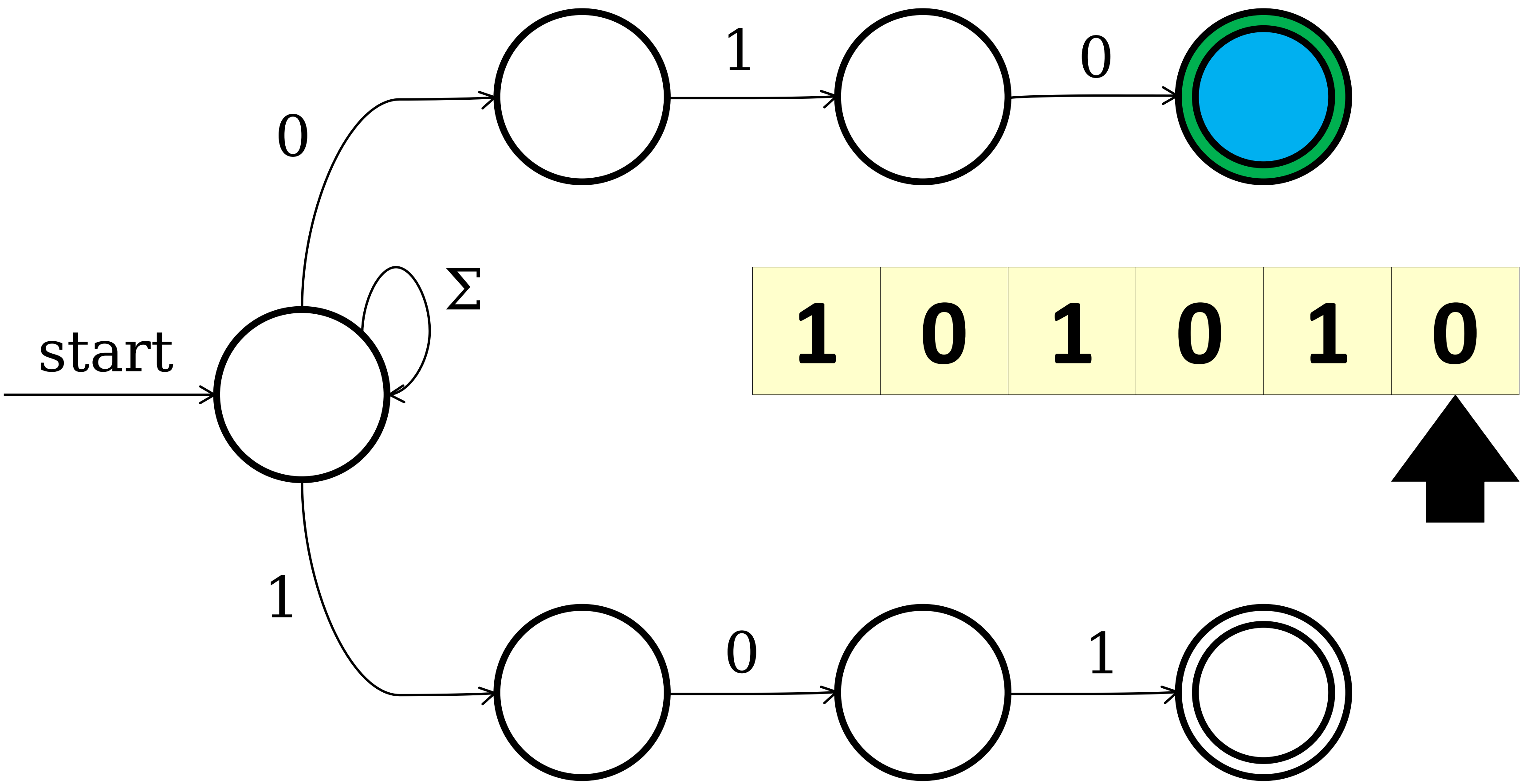
Guess-and-Check

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



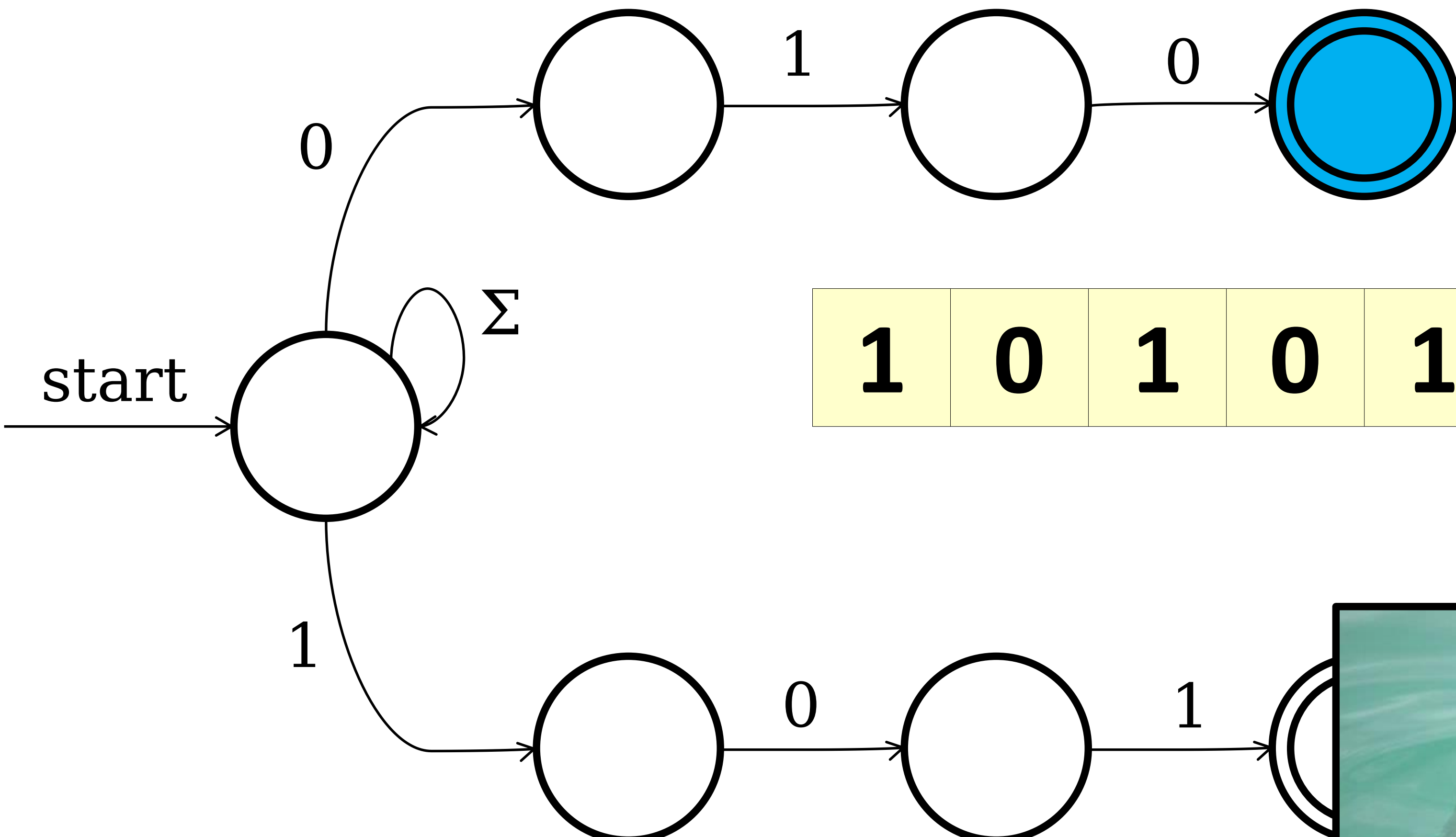
Guess-and-Check

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



Guess-and-Check

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



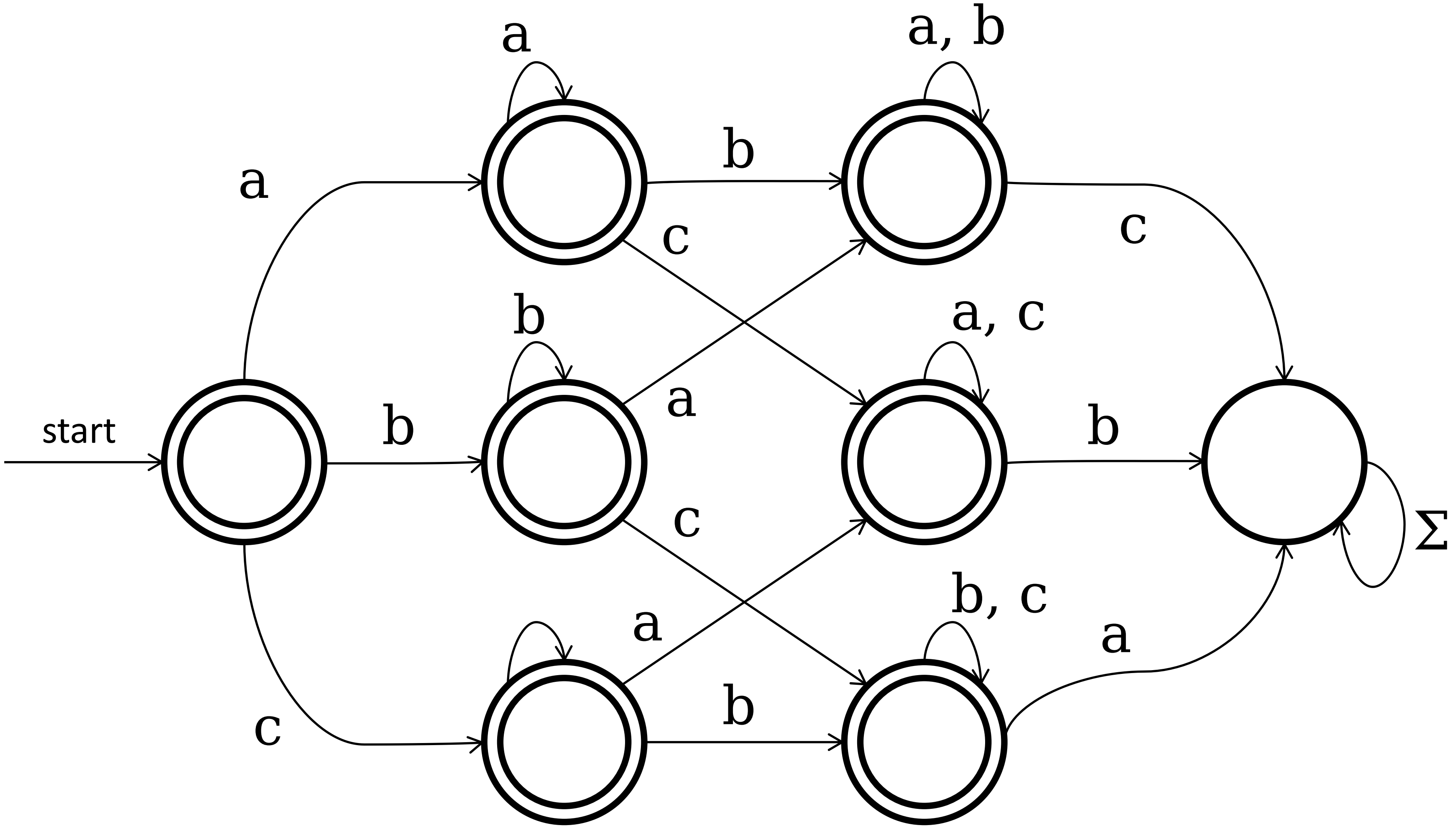
1	**0**	**1**	**0**	**1**	**0**

Guess-and-Check

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

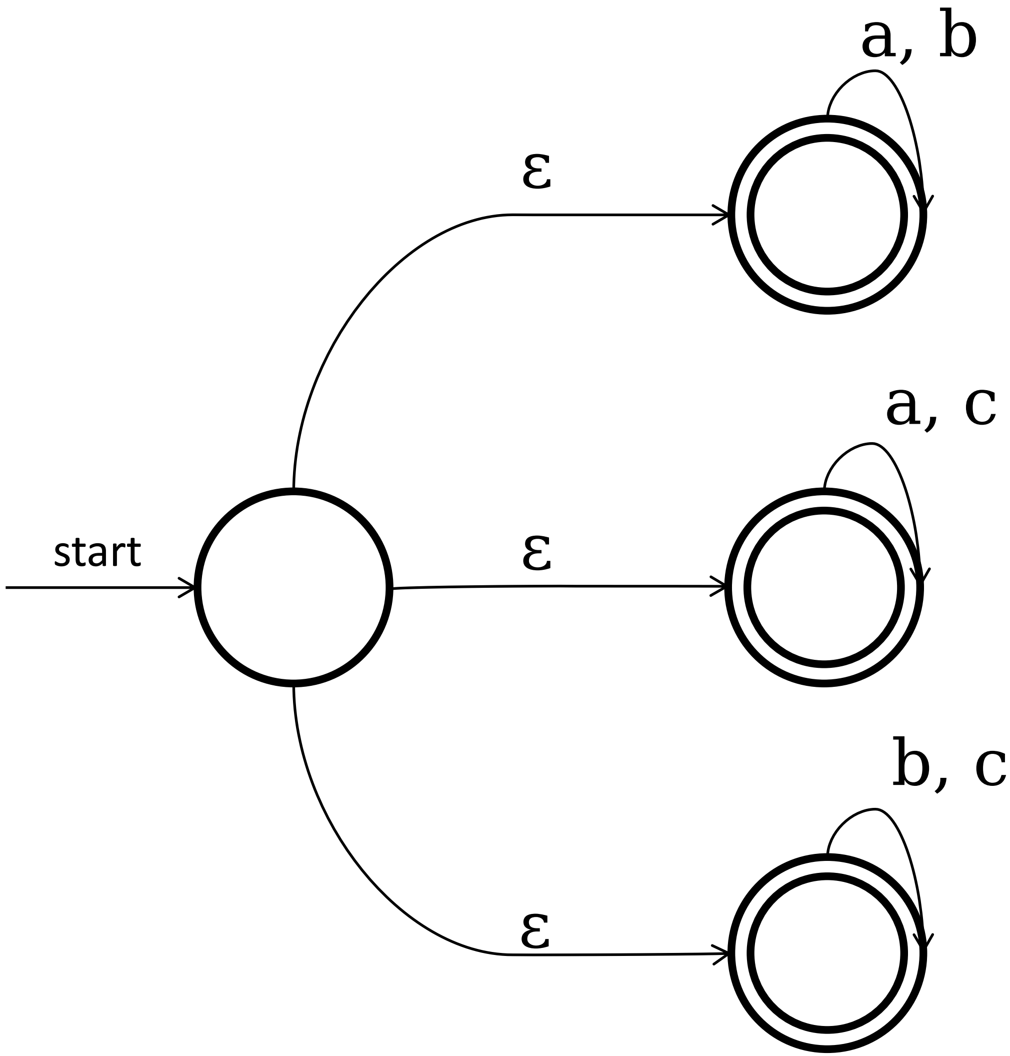
Guess-and-Check

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



Guess-and-Check

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

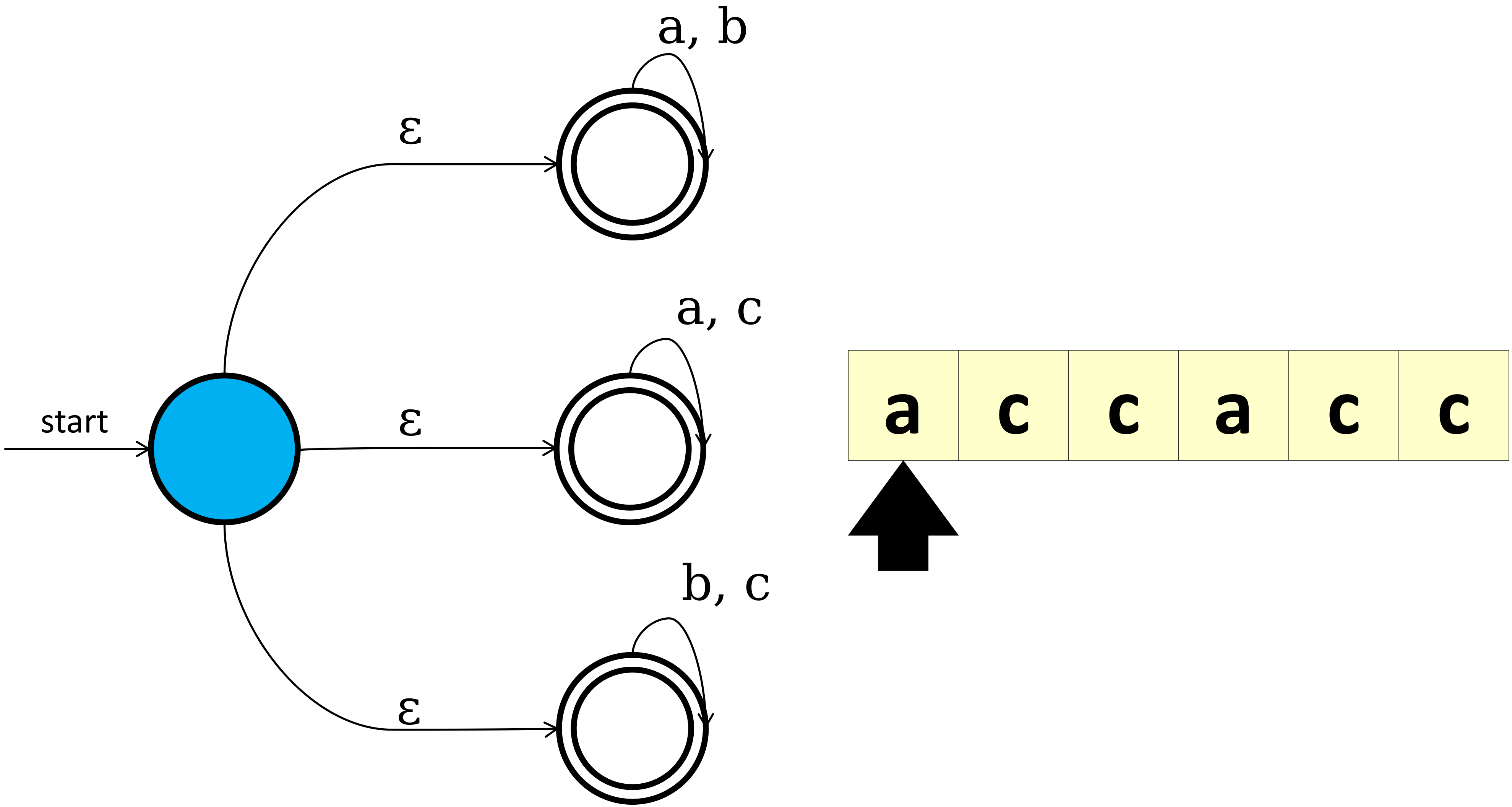


Nondeterministically ***guess*** which character is missing.

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

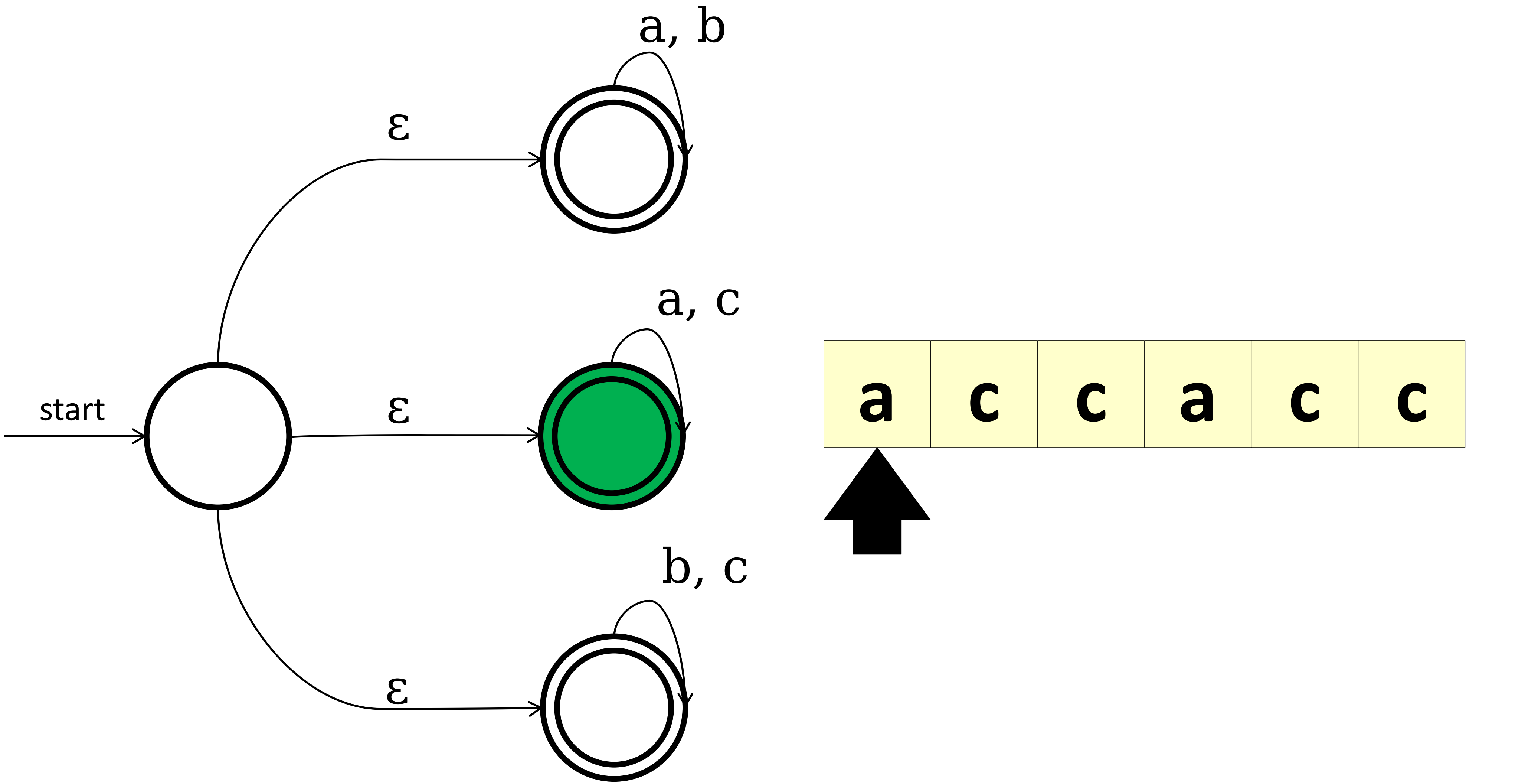
Guess-and-Check

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



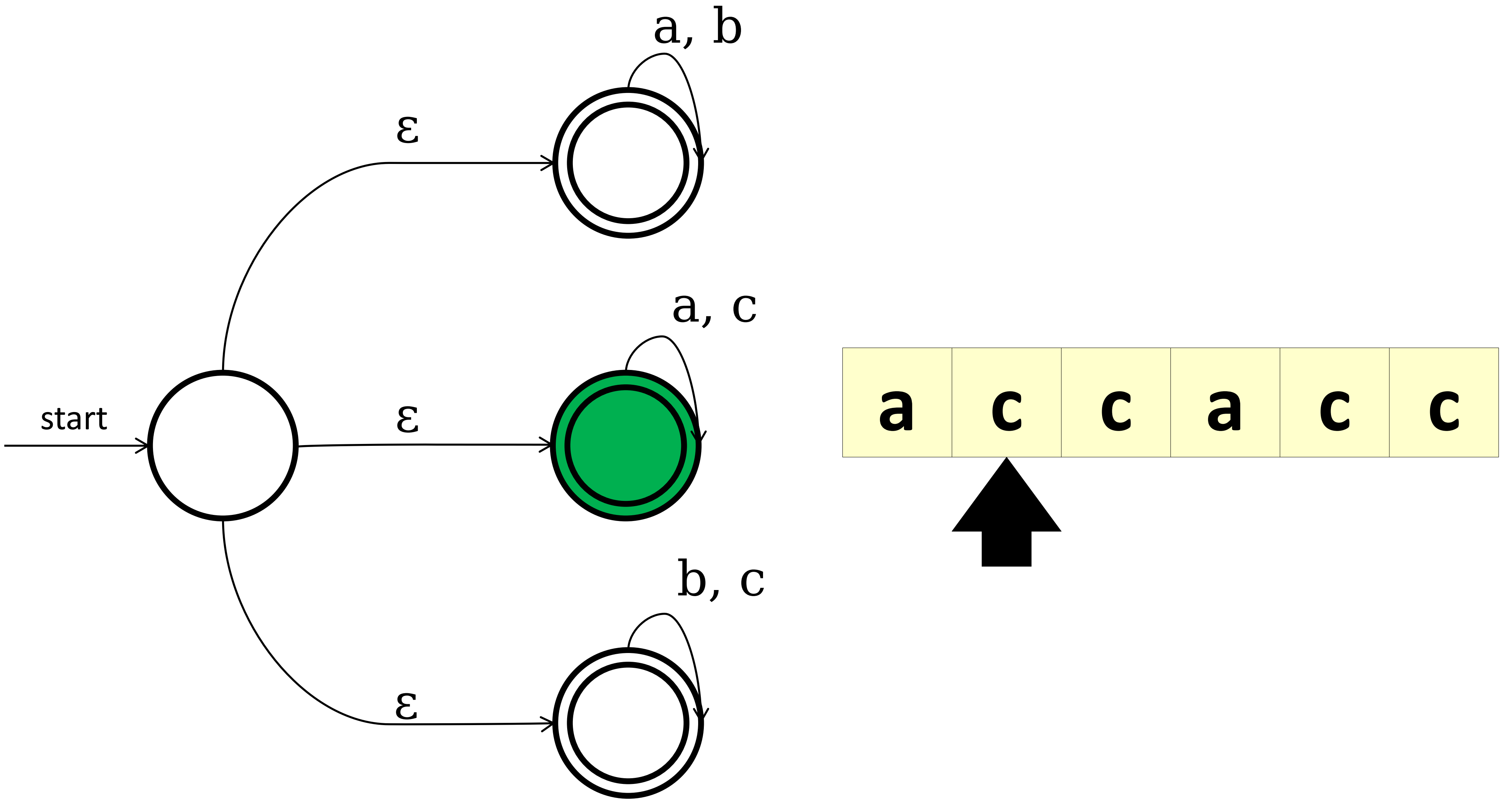
Guess-and-Check

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



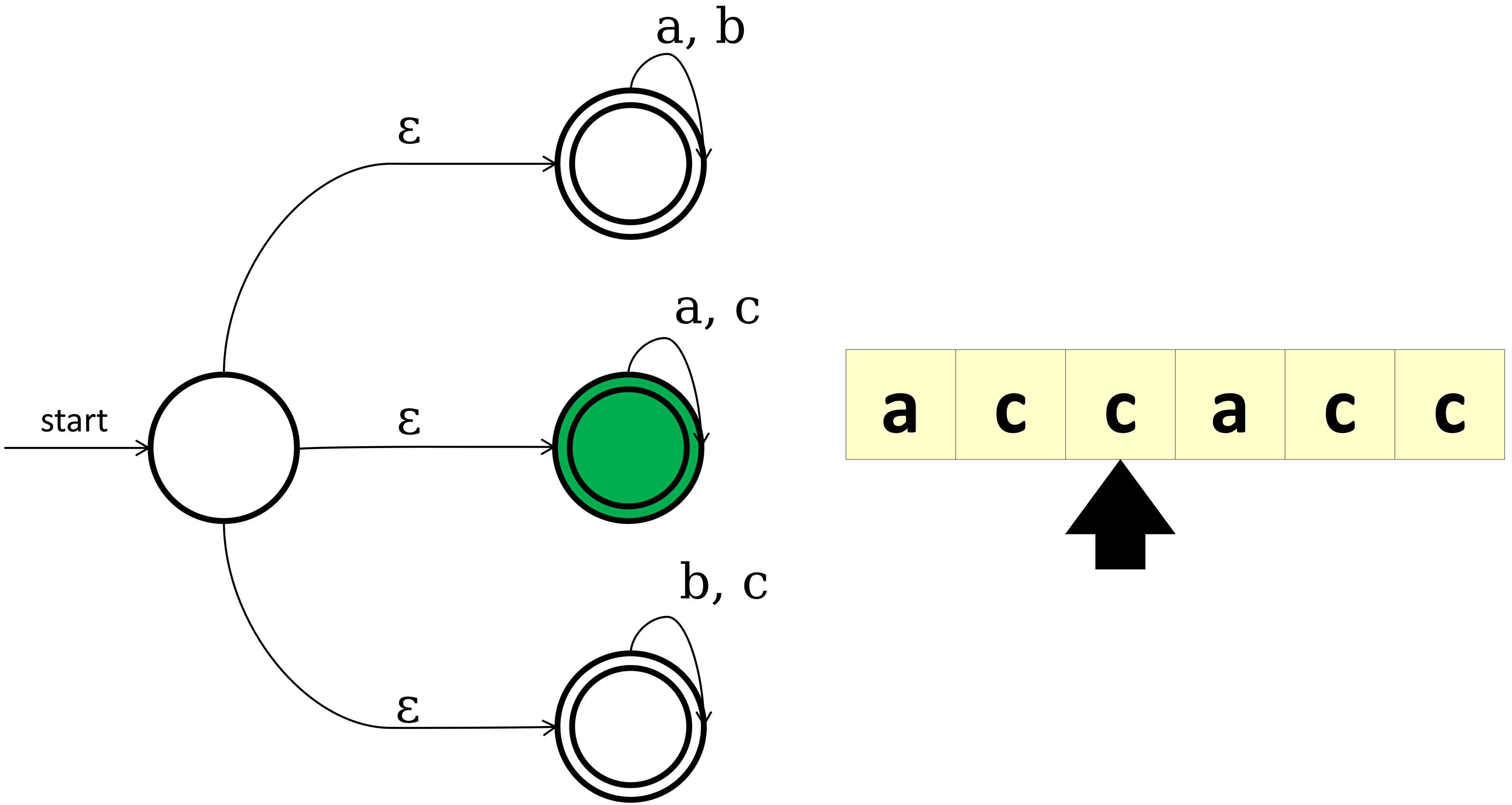
Guess-and-Check

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



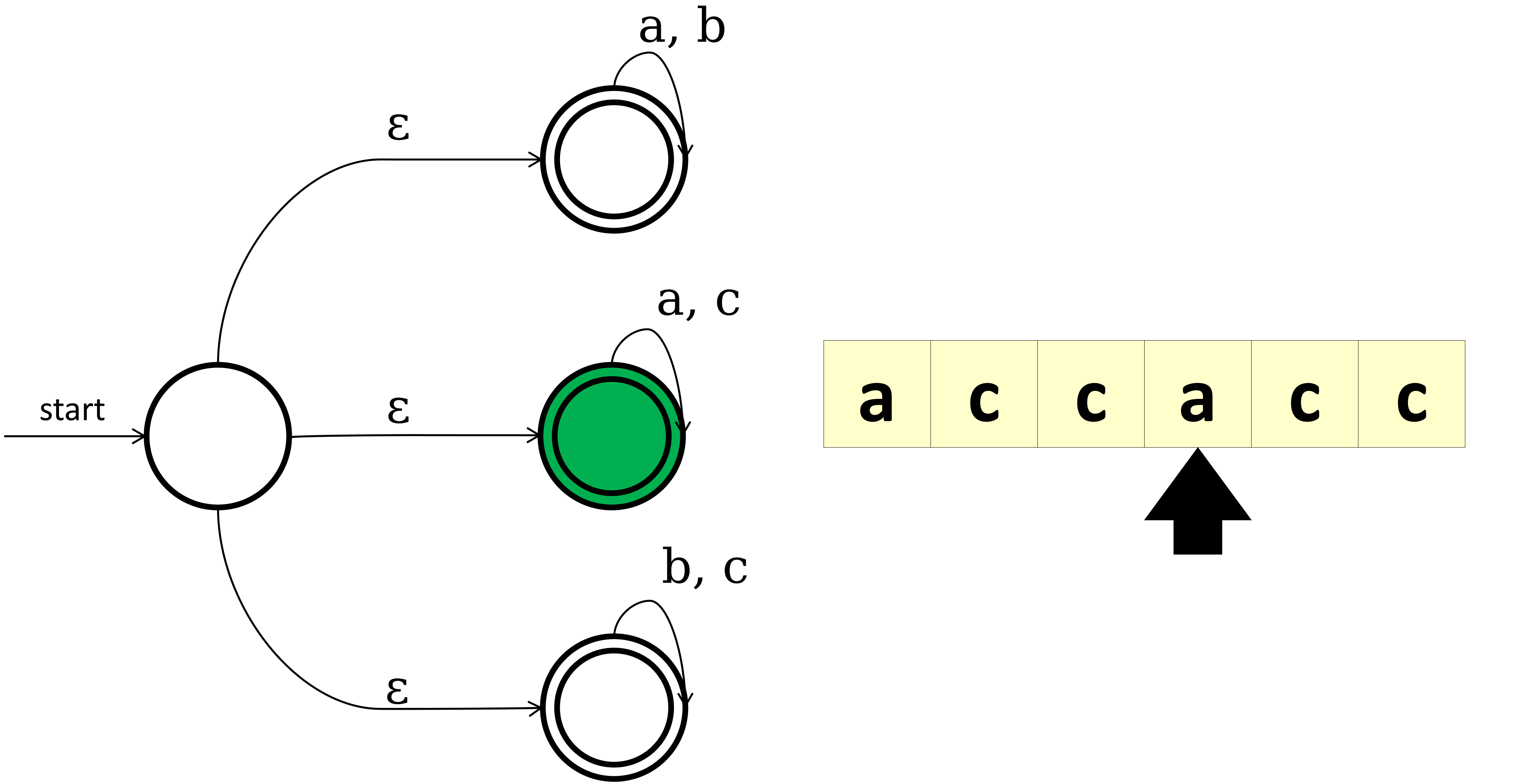
Guess-and-Check

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



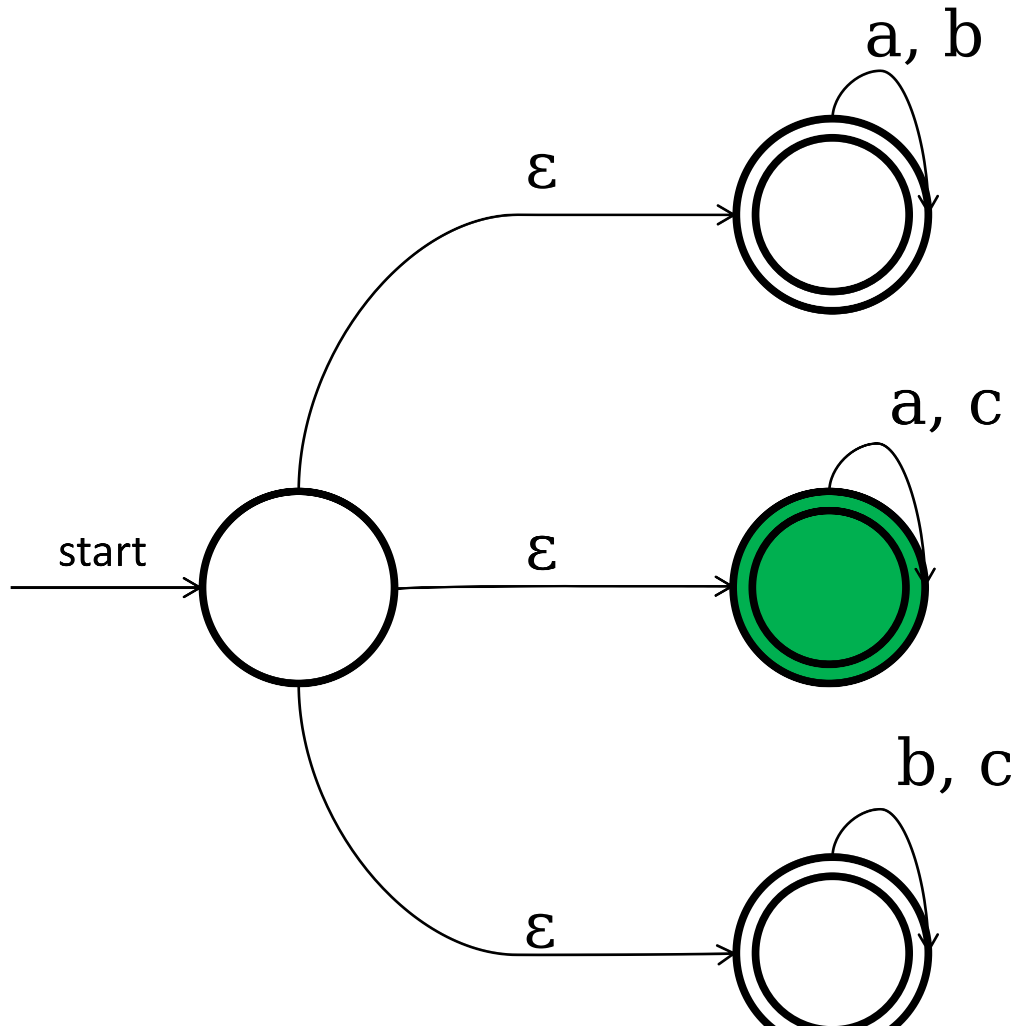
Guess-and-Check

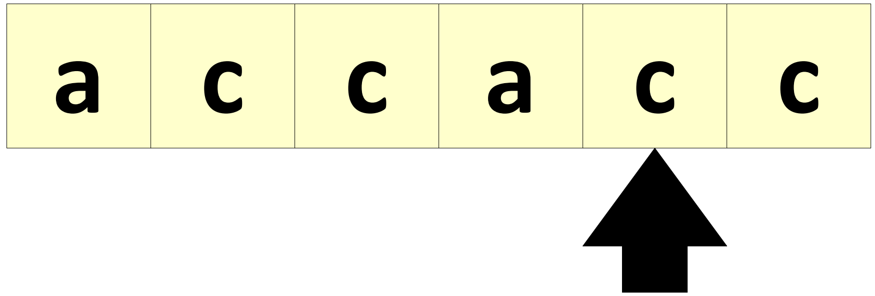
$L = \{ w \in \{$**a**, **b**, **c**$\}^* \mid$ at least one of **a**, **b**, or **c** is not in w $\}$



Guess-and-Check

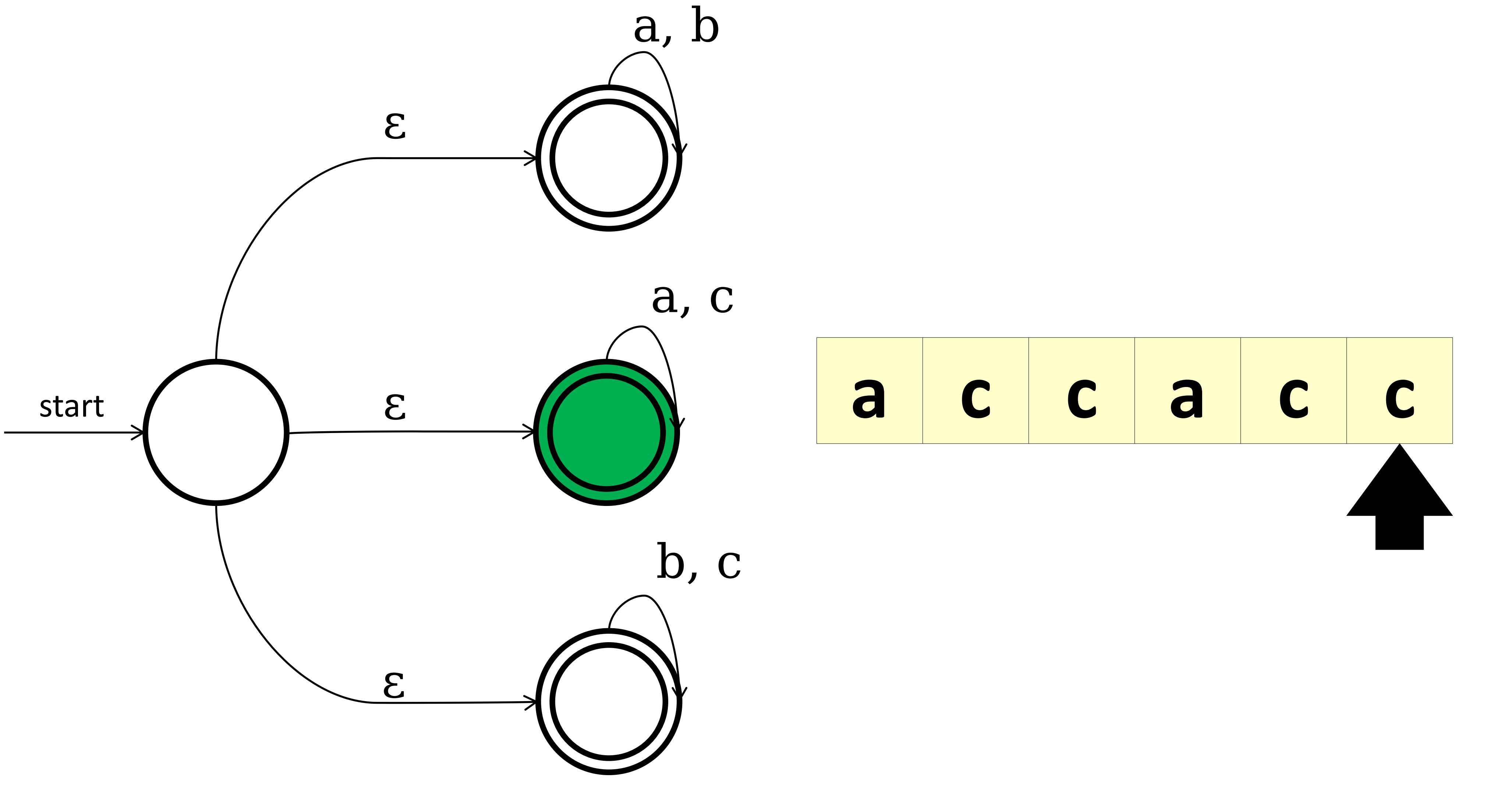
$L = \{\ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid \text{at least one of } \mathbf{a}, \mathbf{b}, \text{ or } \mathbf{c} \text{ is not in } w\ \}$





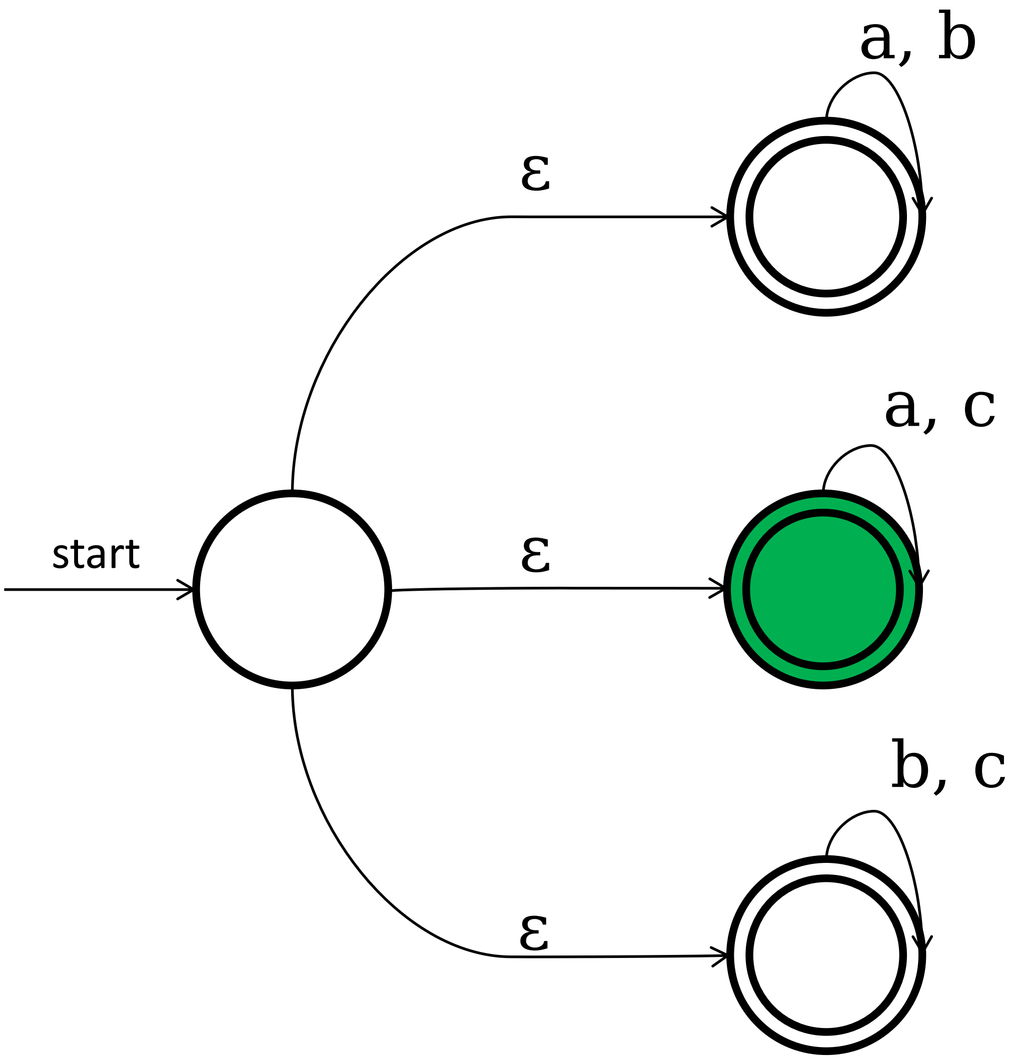
Guess-and-Check

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



Guess-and-Check

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



a	**c**	**c**	**a**	**c**	**c**

Time out for Announcements

Midterm and Pset

- Congratulations on finishing the midterm!
- We’ll have this returned to you on Monday.
- Problem Set 4 is due next Thursday.

Just how powerful are NFAs?

NFAs and DFAs

Any language that can be accepted by a DFA can be accepted by an NFA.

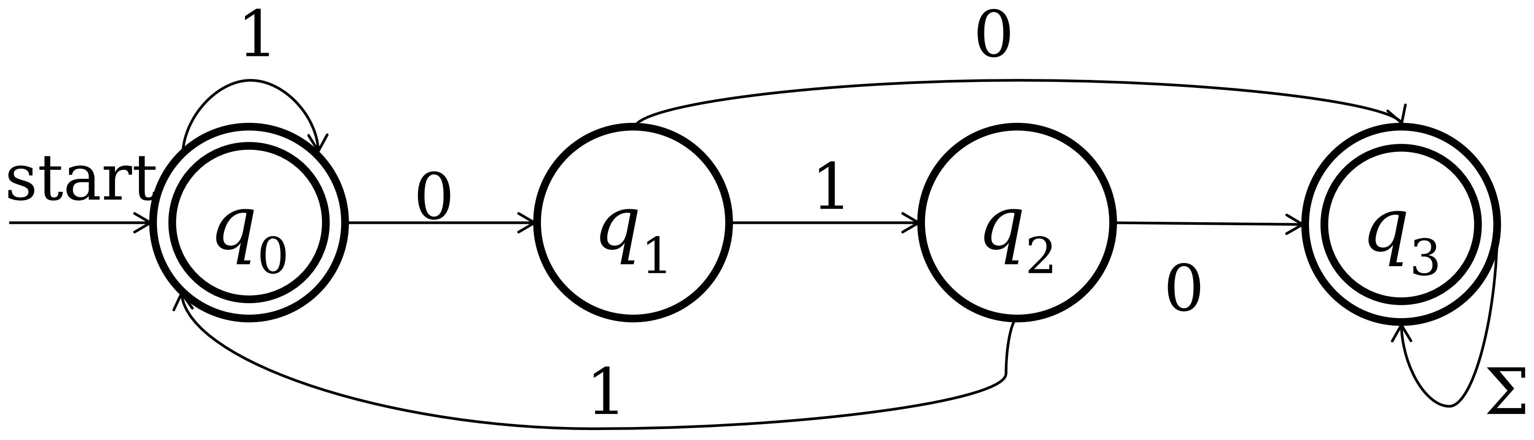
Why?

Every DFA essentially already *is* an NFA!

Question: Can any language accepted by an NFA also be accepted by a DFA?

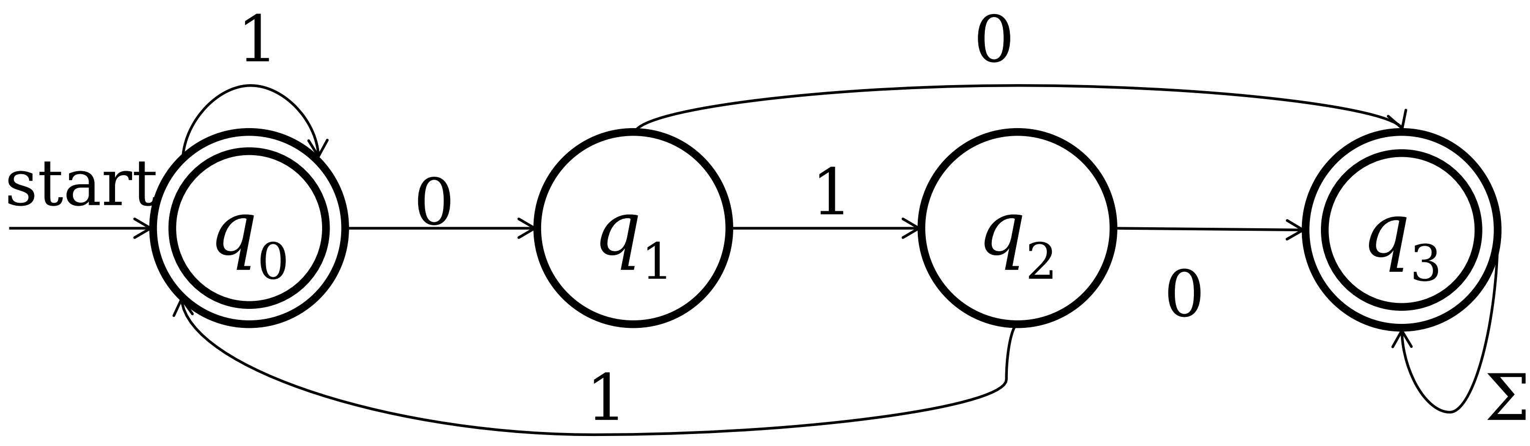
Surprisingly, the answer is ***yes***!

Tabular DFAs



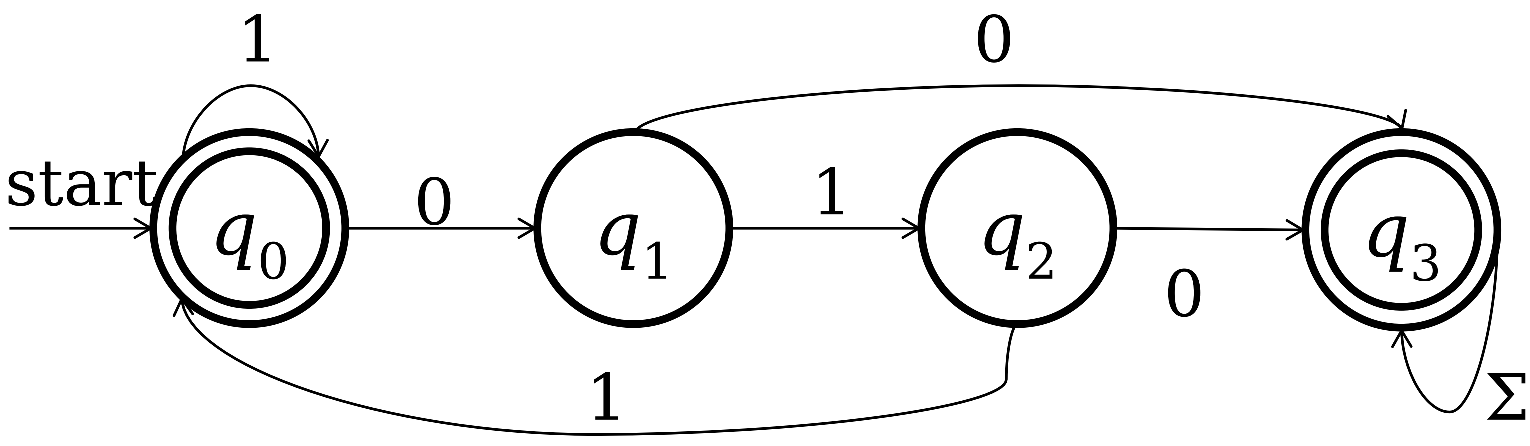
	0	1
q_0		
q_1		
q_2		
q_3		

Tabular DFAs



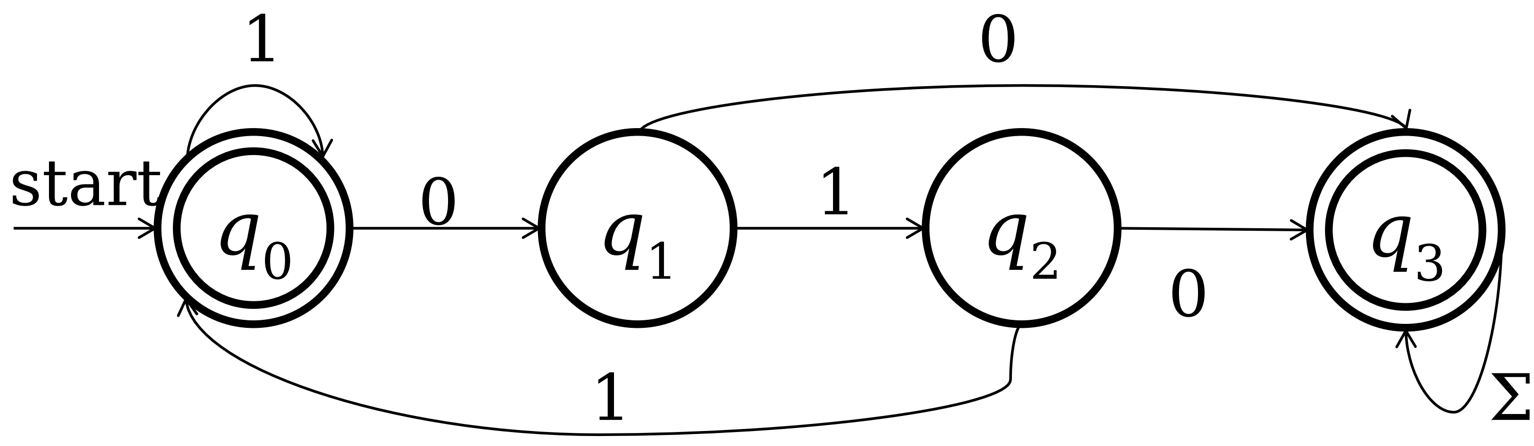
	0	**1**
q_0	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
q_3	q_3	q_3

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

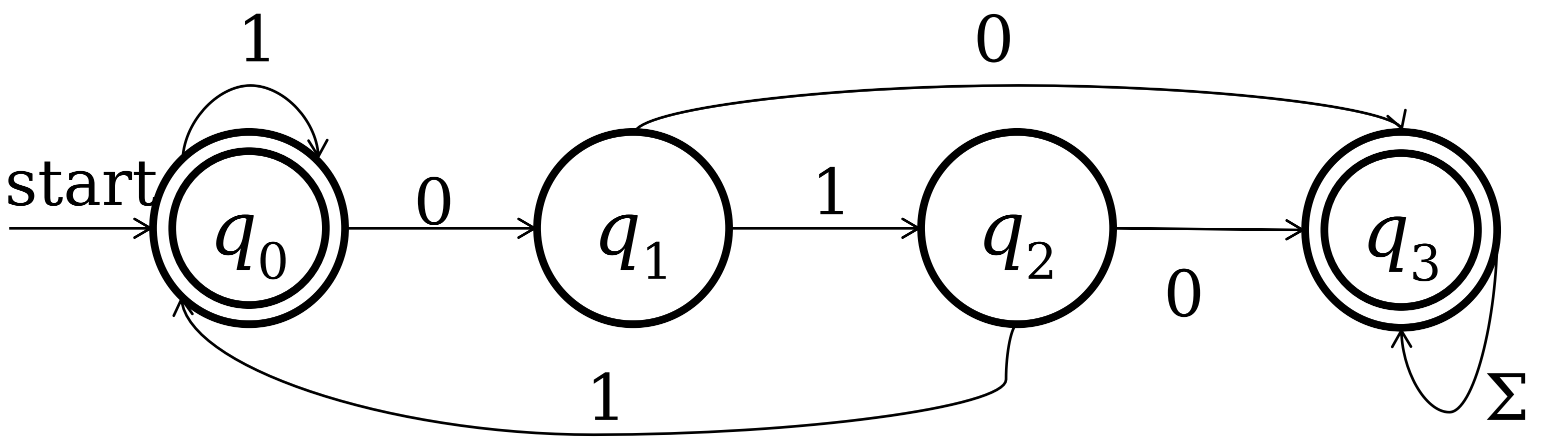
Tabular DFAs



	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.

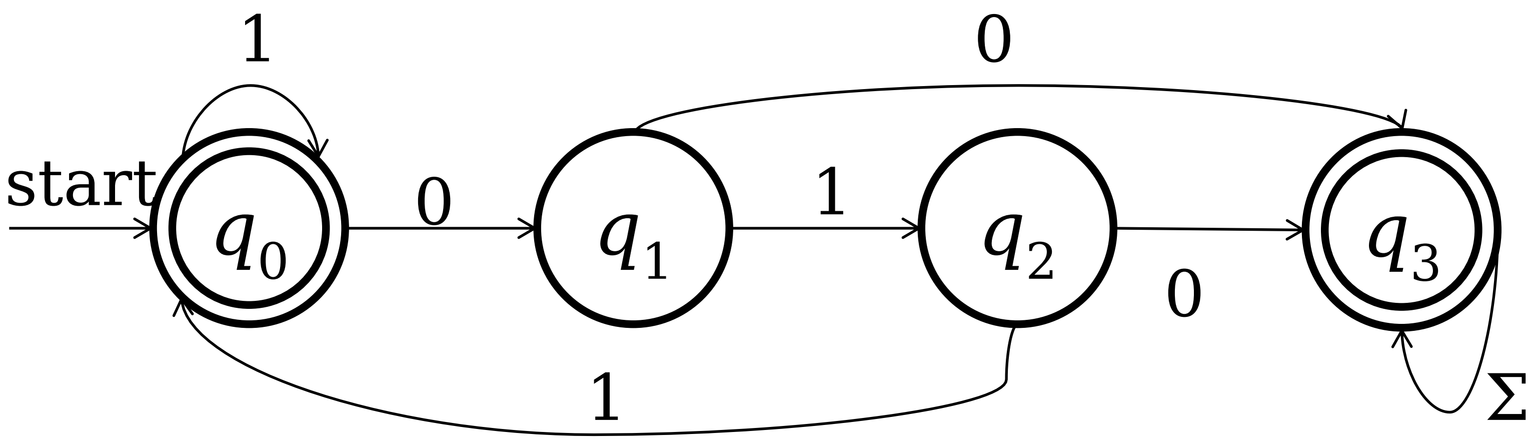
Tabular DFAs



	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

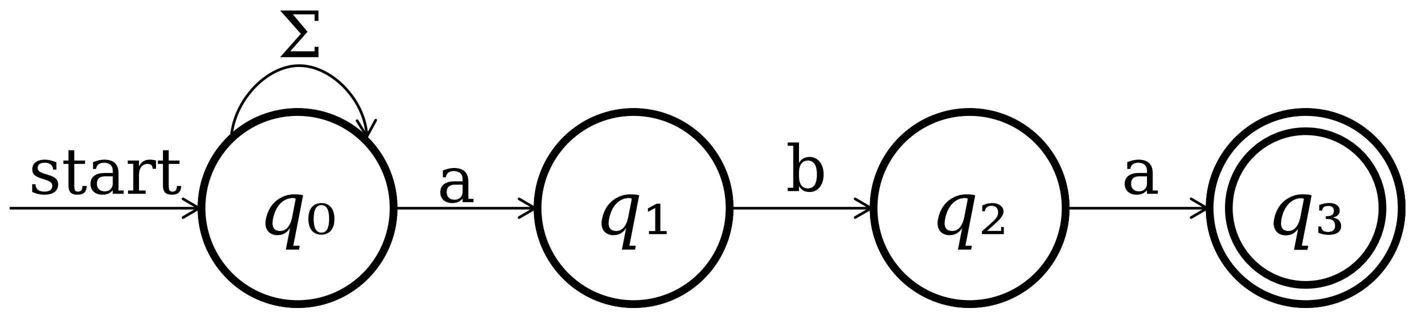
Question to ponder: Why isn't there a column here for Σ?

My Turn to Code Things Up!

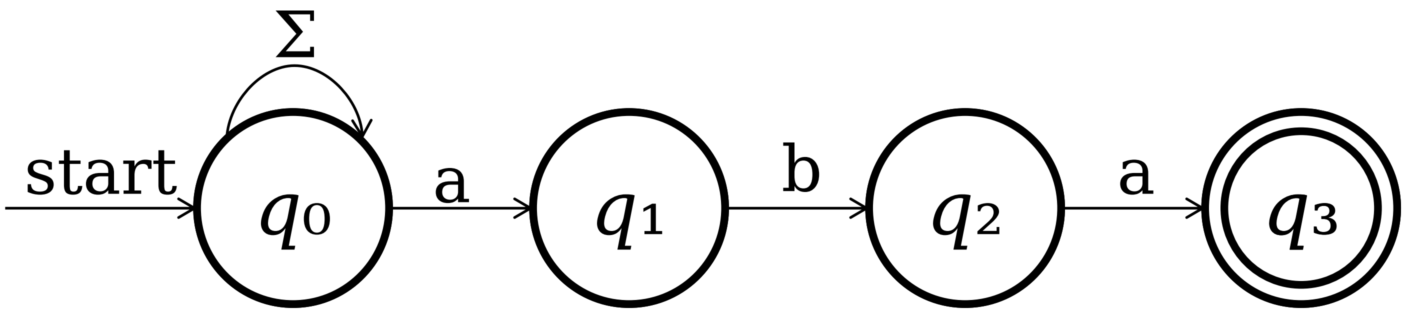
```
int kTransitionTable[kNumStates][kNumSymbols] = {
    {0, 0, 1, 3, 7, 1, …},
     …
};
bool kAcceptTable[kNumStates] = {
   false,
   true,
   true,
   …
};
bool SimulateDFA(string input) {
   int state = 0;
   for (char ch: input) {
      state = kTransitionTable[state][ch];
   }
   return kAcceptTable[state];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

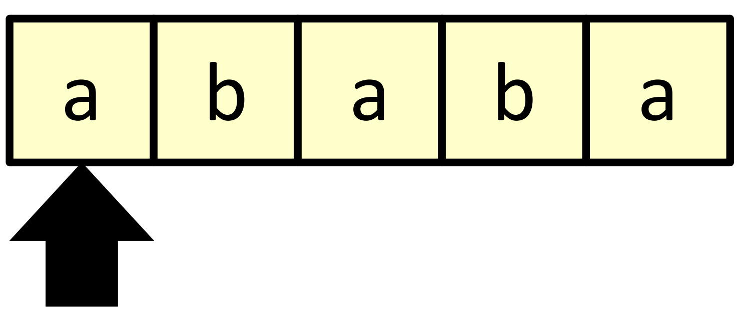
Thought Experiment:

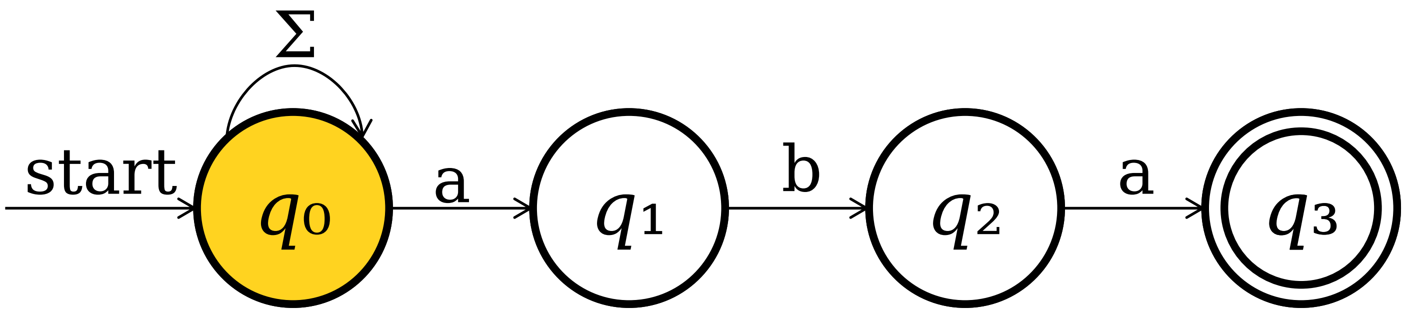
How would you simulate an NFA in software?

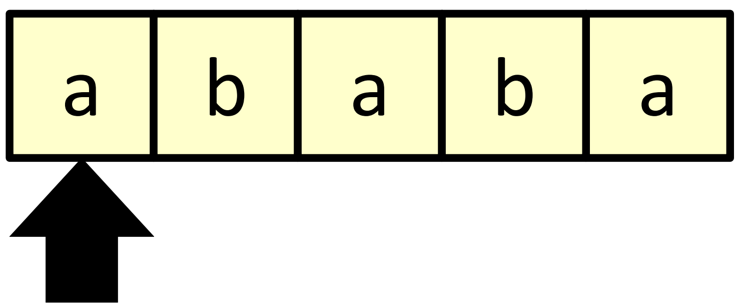


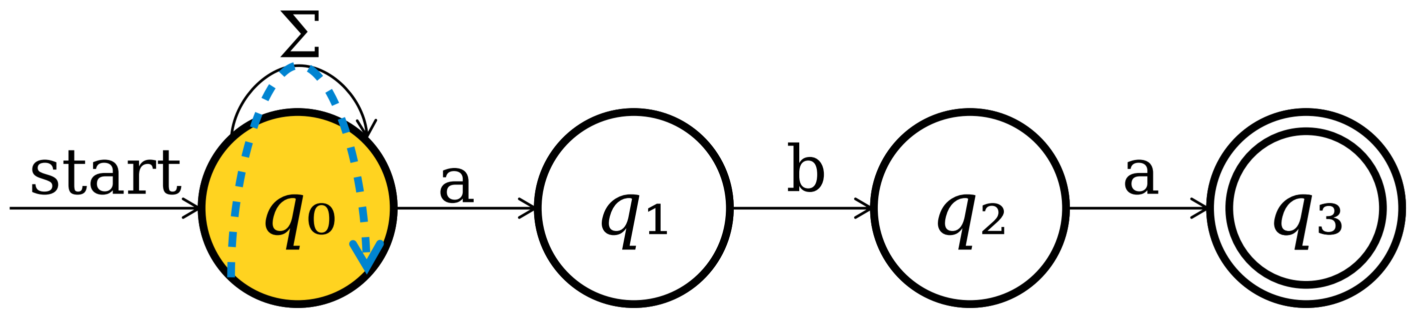
Σ
start
q_0
a
q_1
b
q_2
a
q_3

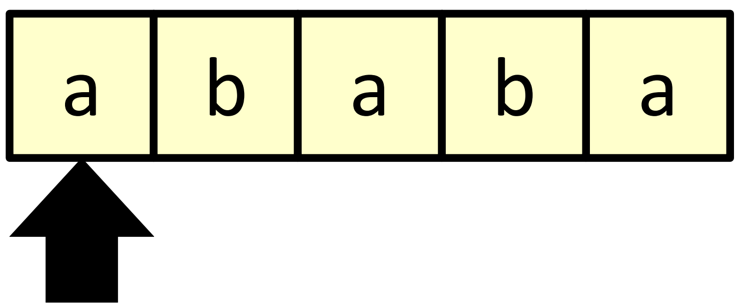

Σ
start
q0
a
q1
b
q2
a
q3

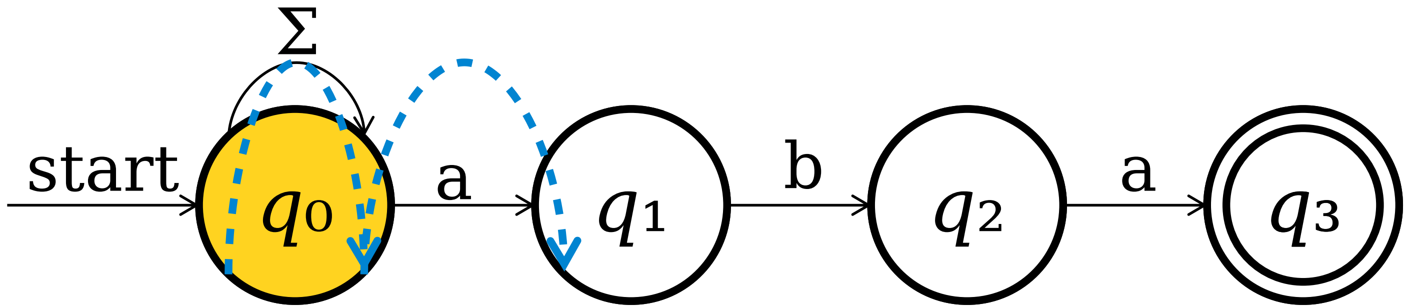

a
b
a
b
a

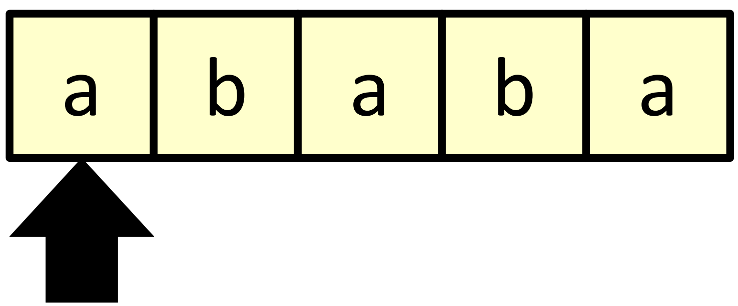

Σ
start
q_0
a
q_1
b
q_2
a
q_3

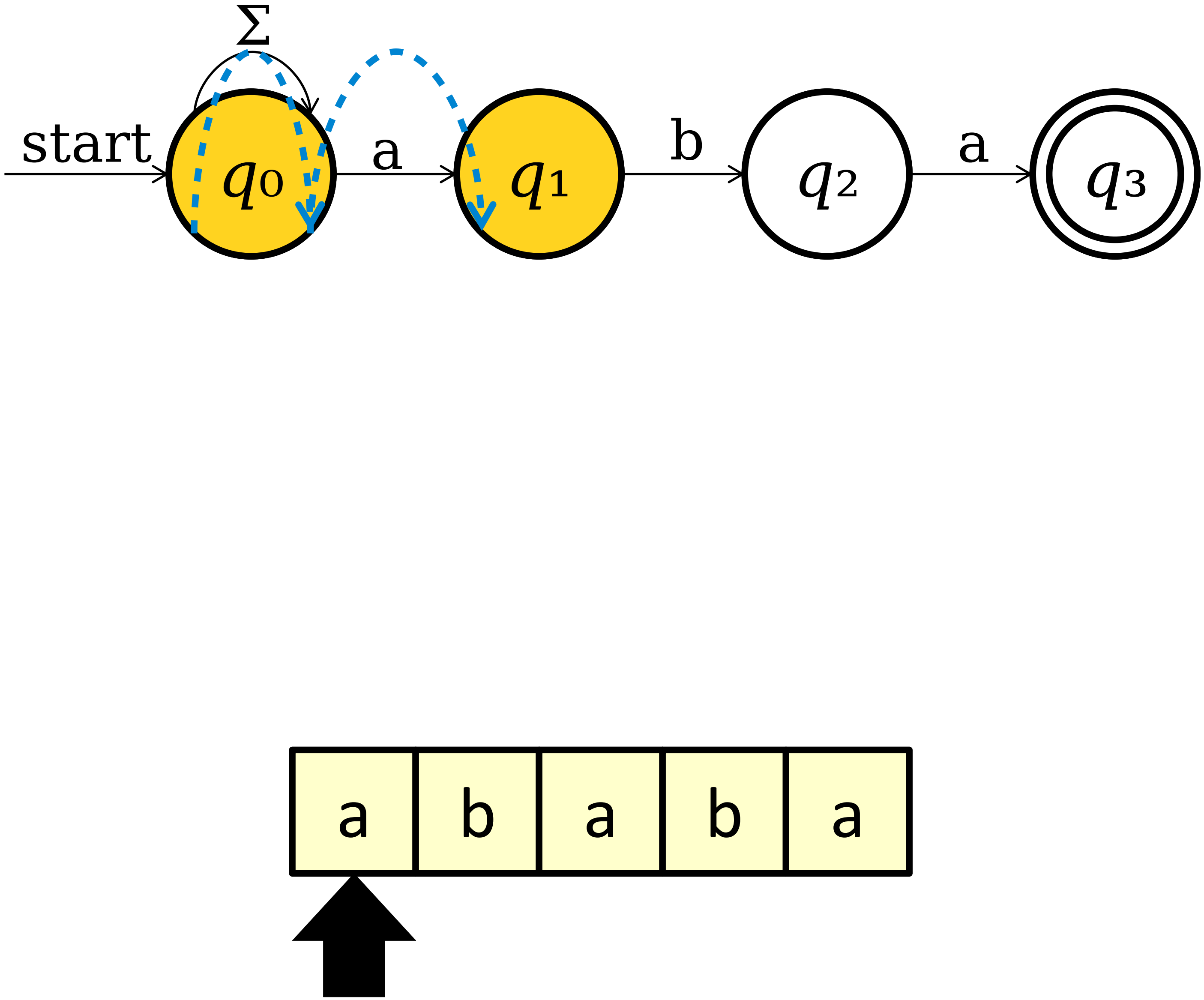

a b a b a

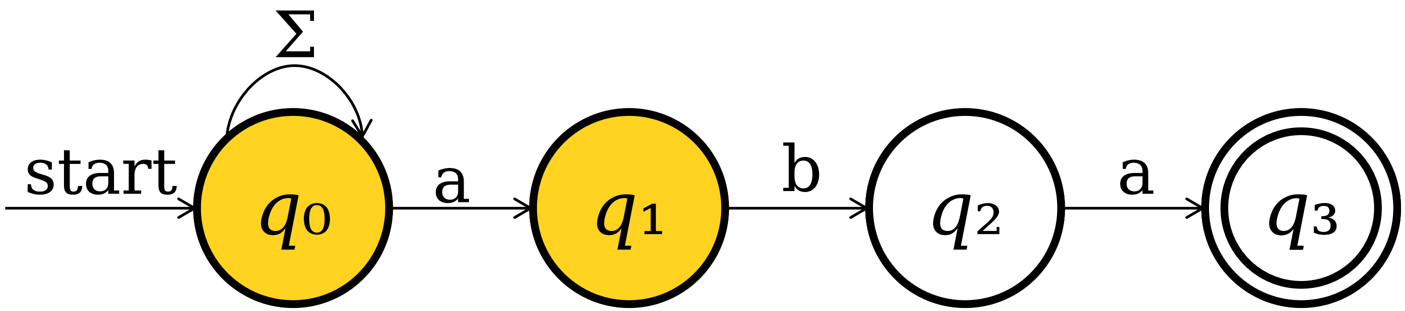

Σ
start
q0
a
q1
b
q2
a
q3

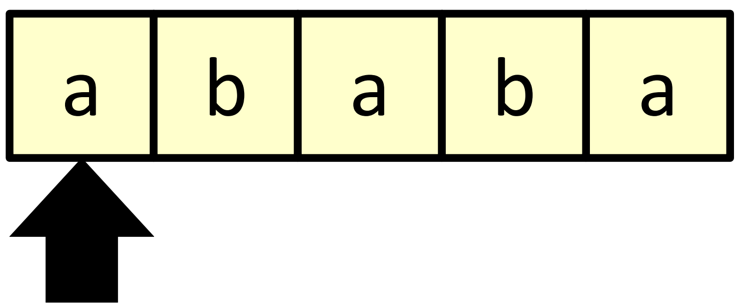

a
b
a
b
a

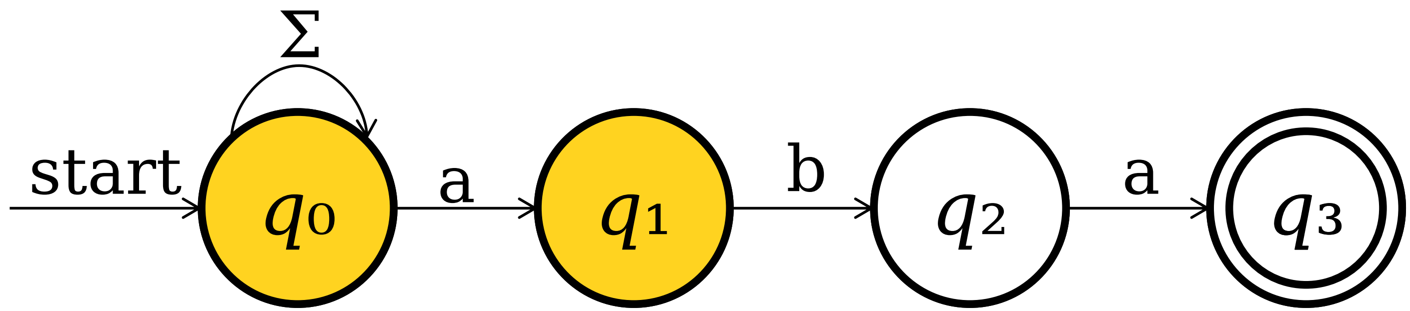

Σ
start
q0
a
q1
b
q2
a
q3

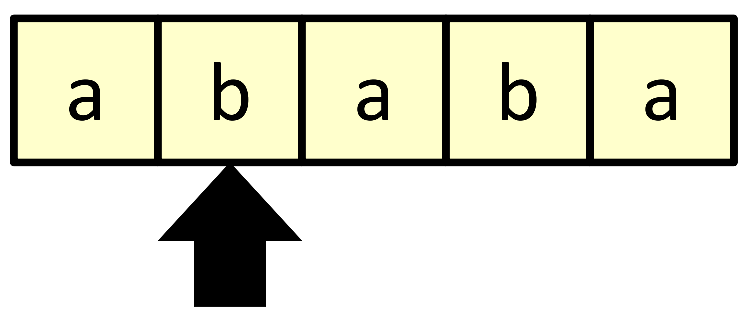

a
b
a
b
a

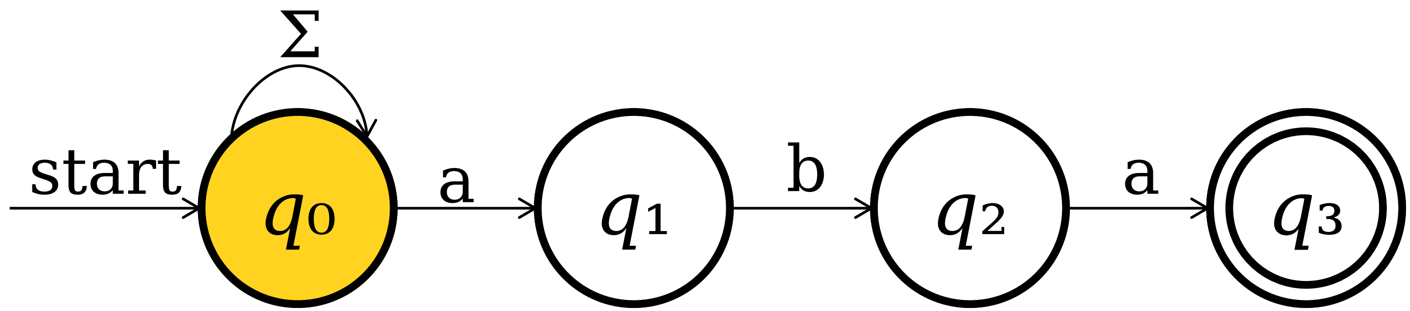

Σ
start
q0
a
q1
b
q2
a
q3
a
b
a
b
a


Σ
start
q_0
a
q_1
b
q_2
a
q_3

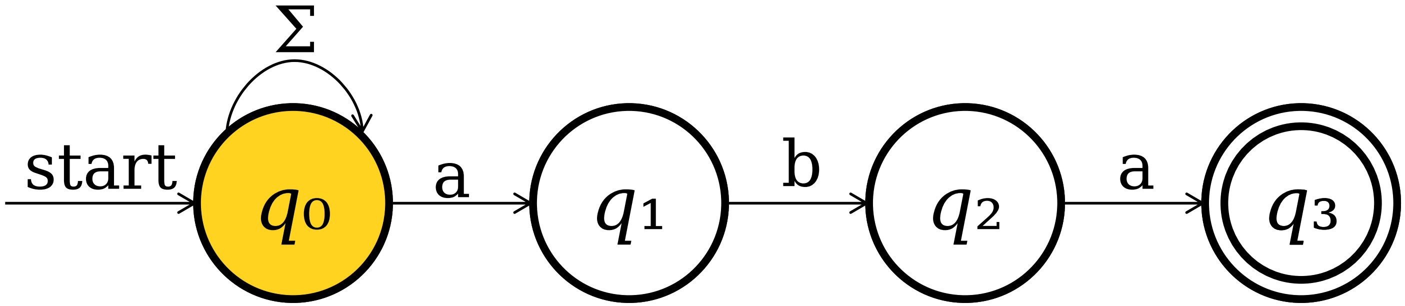

a
b
a
b
a

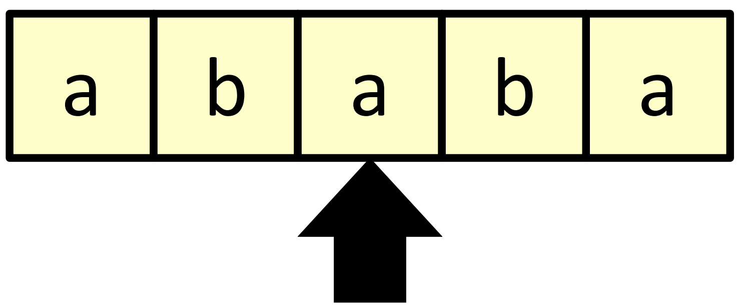

Σ
start
q0
a
q1
b
q2
a
q3

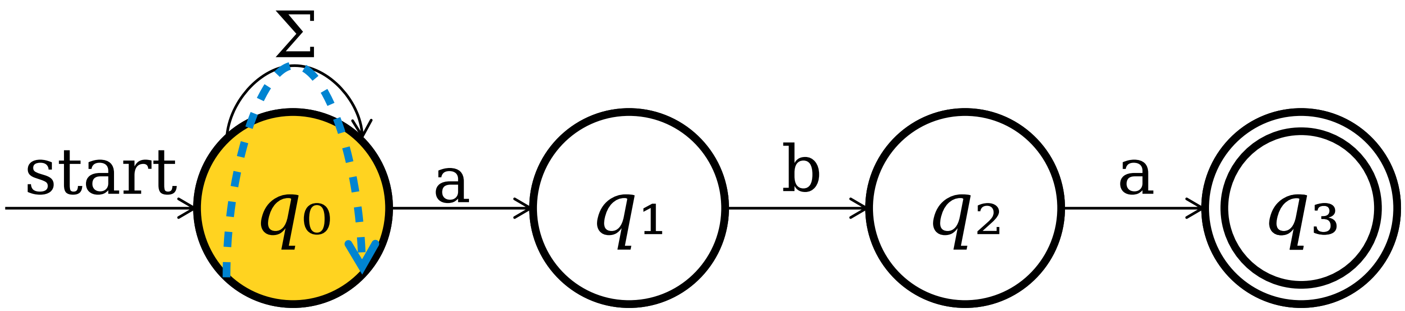

a b a b a



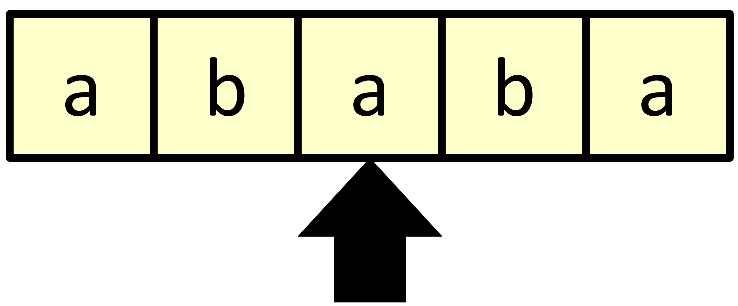
a	b	a	b	a


Σ
start
q_0
a
q_1
b
q_2
a
q_3

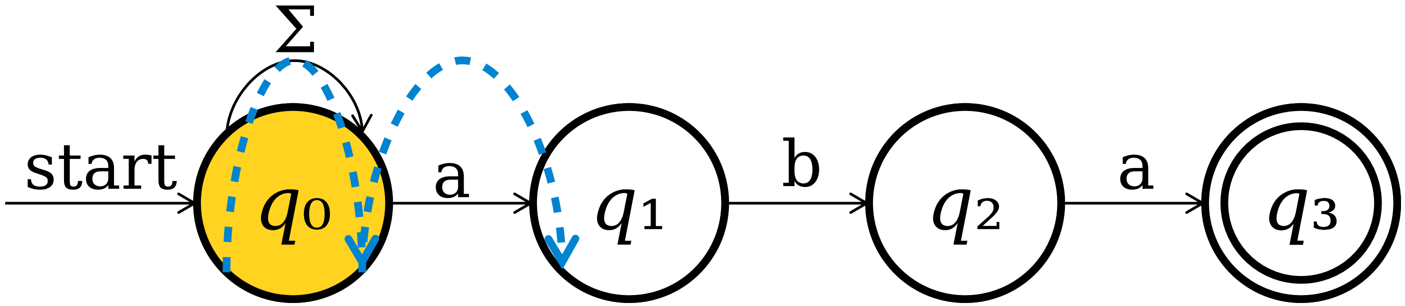

a b a b a



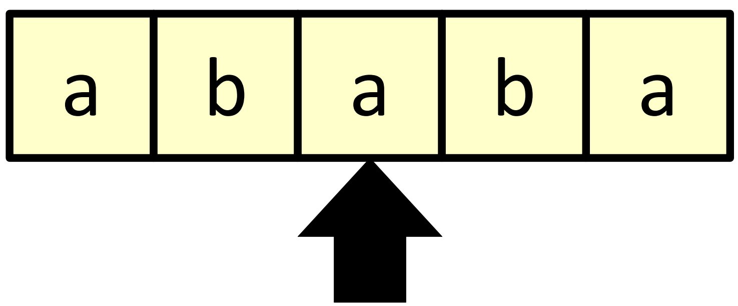
Σ
start
q_0
a
q_1
b
q_2
a
q_3



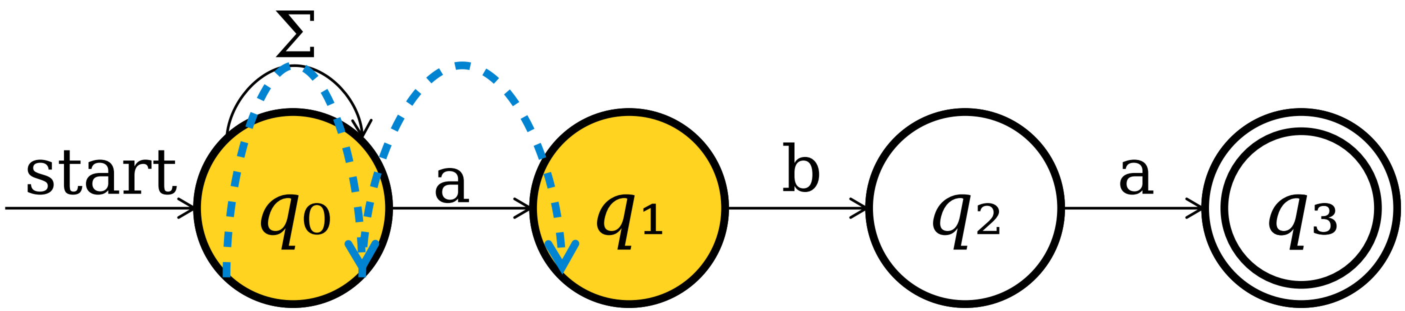
a
b
a
b
a

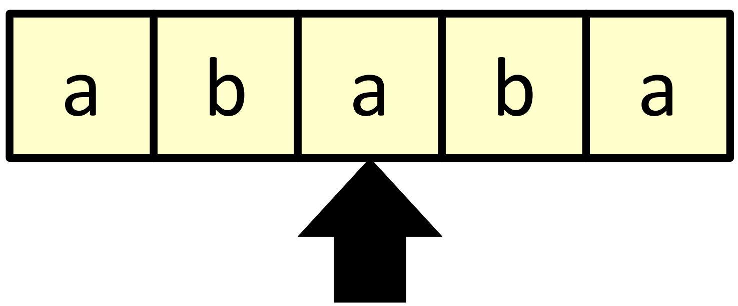


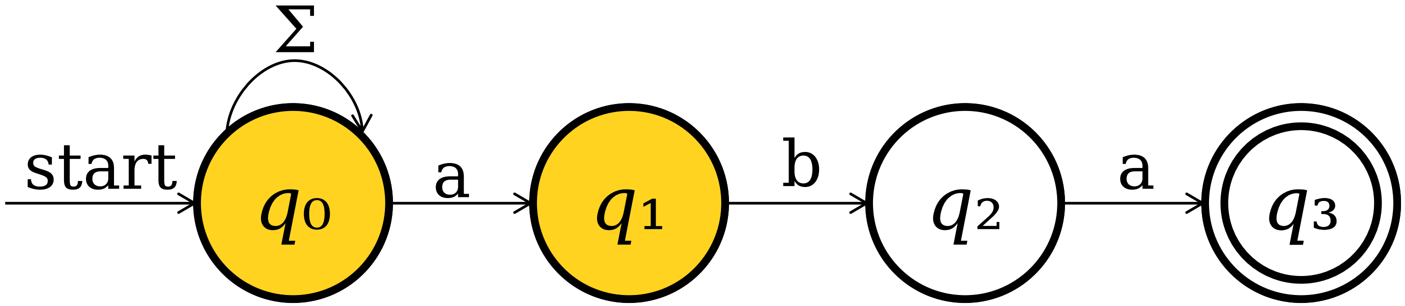
Σ
start
q_0
a
q_1
b
q_2
a
q_3

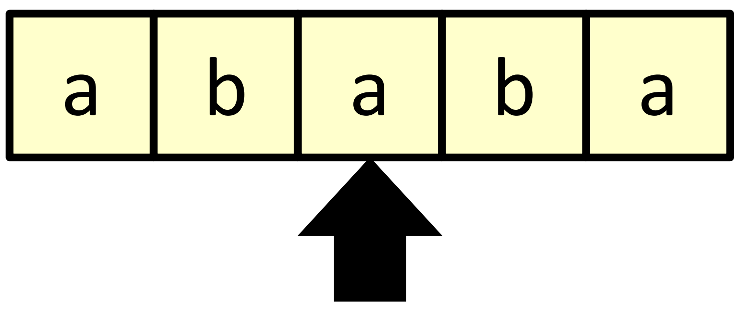


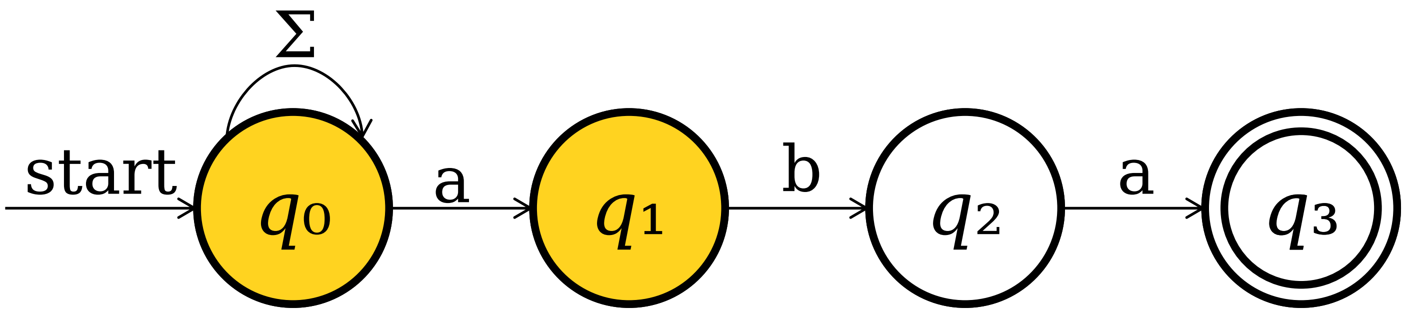
a
b
a
b
a

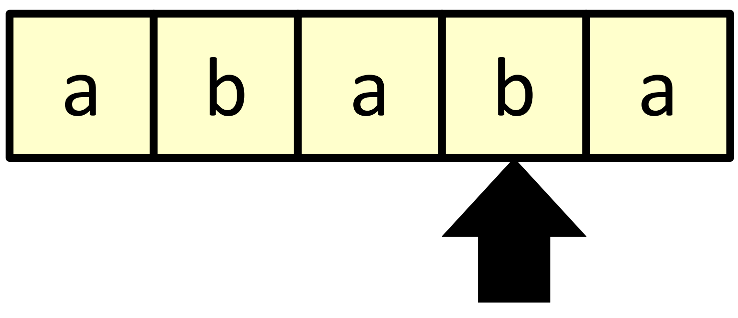

Σ
start
q_0
a
q_1
b
q_2
a
q_3

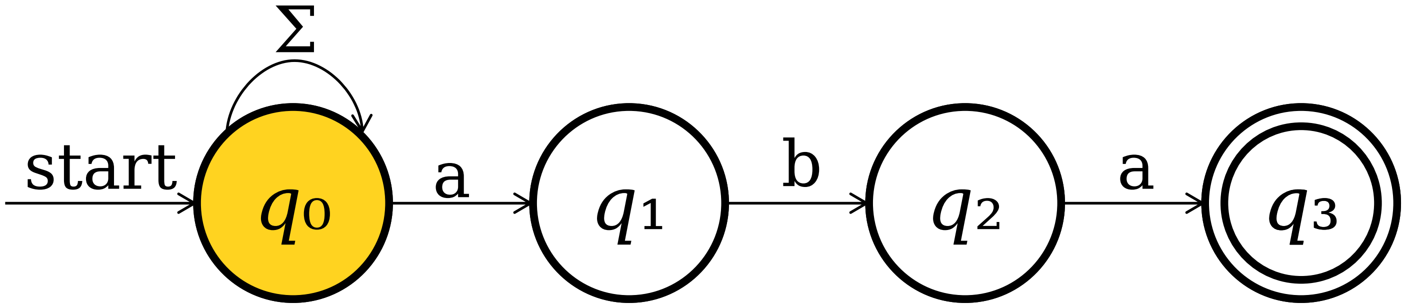

a
b
a
b
a

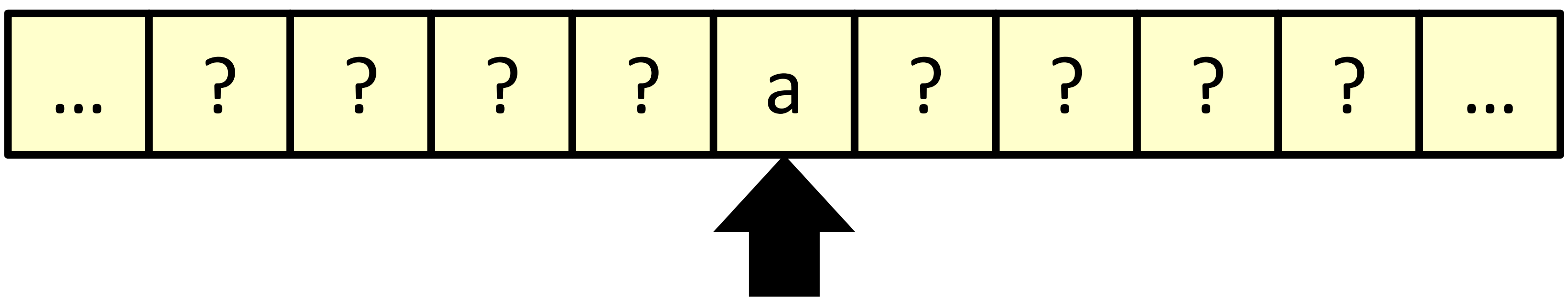

Σ
start
q_0
a
q_1
b
q_2
a
q_3

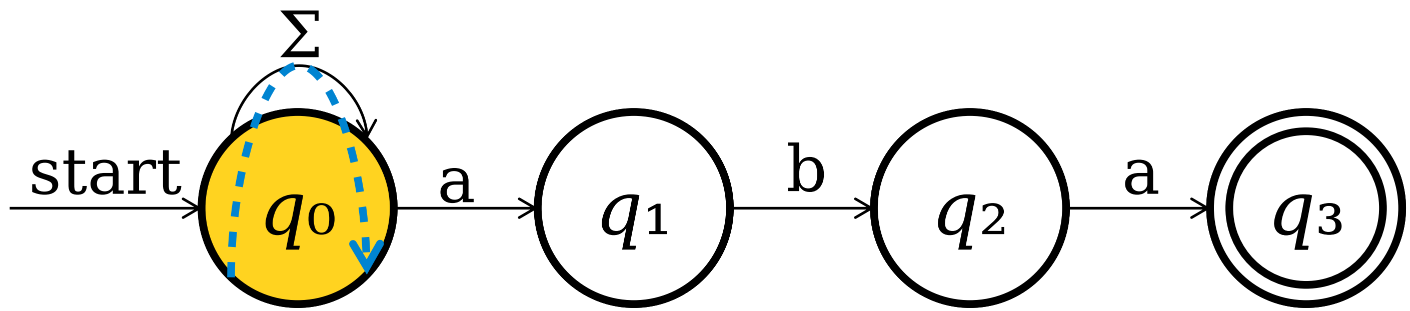

a
b
a
b
a

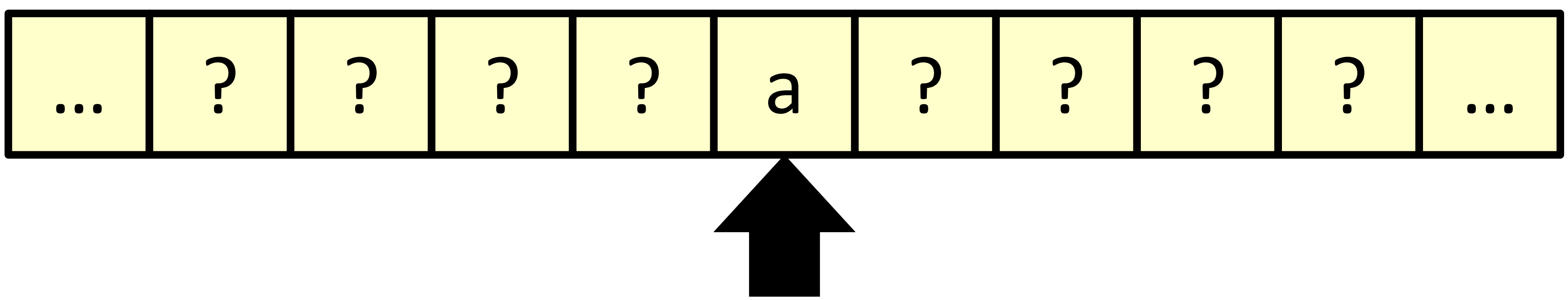

Σ
start
q_0
a
q_1
b
q_2
a
q_3

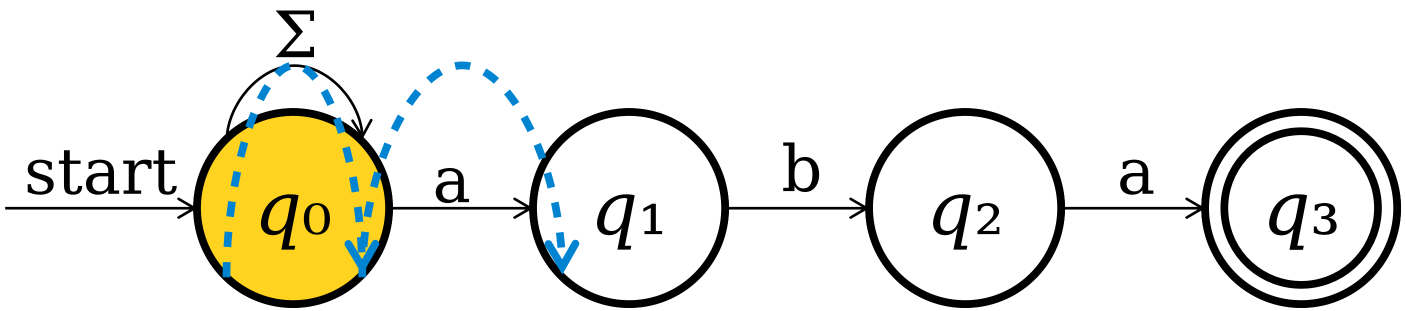

a
b
a
b
a

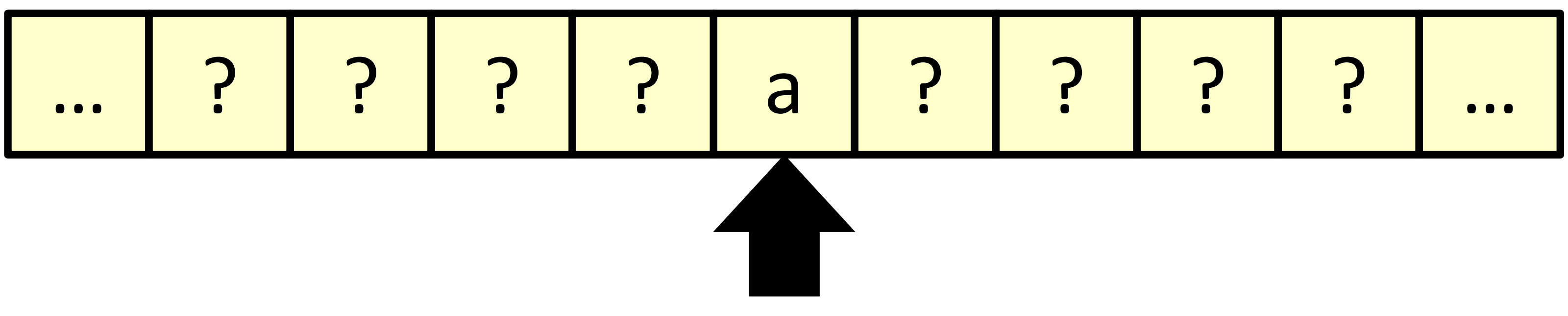

Σ
start
q_0
a
q_1
b
q_2
a
q_3

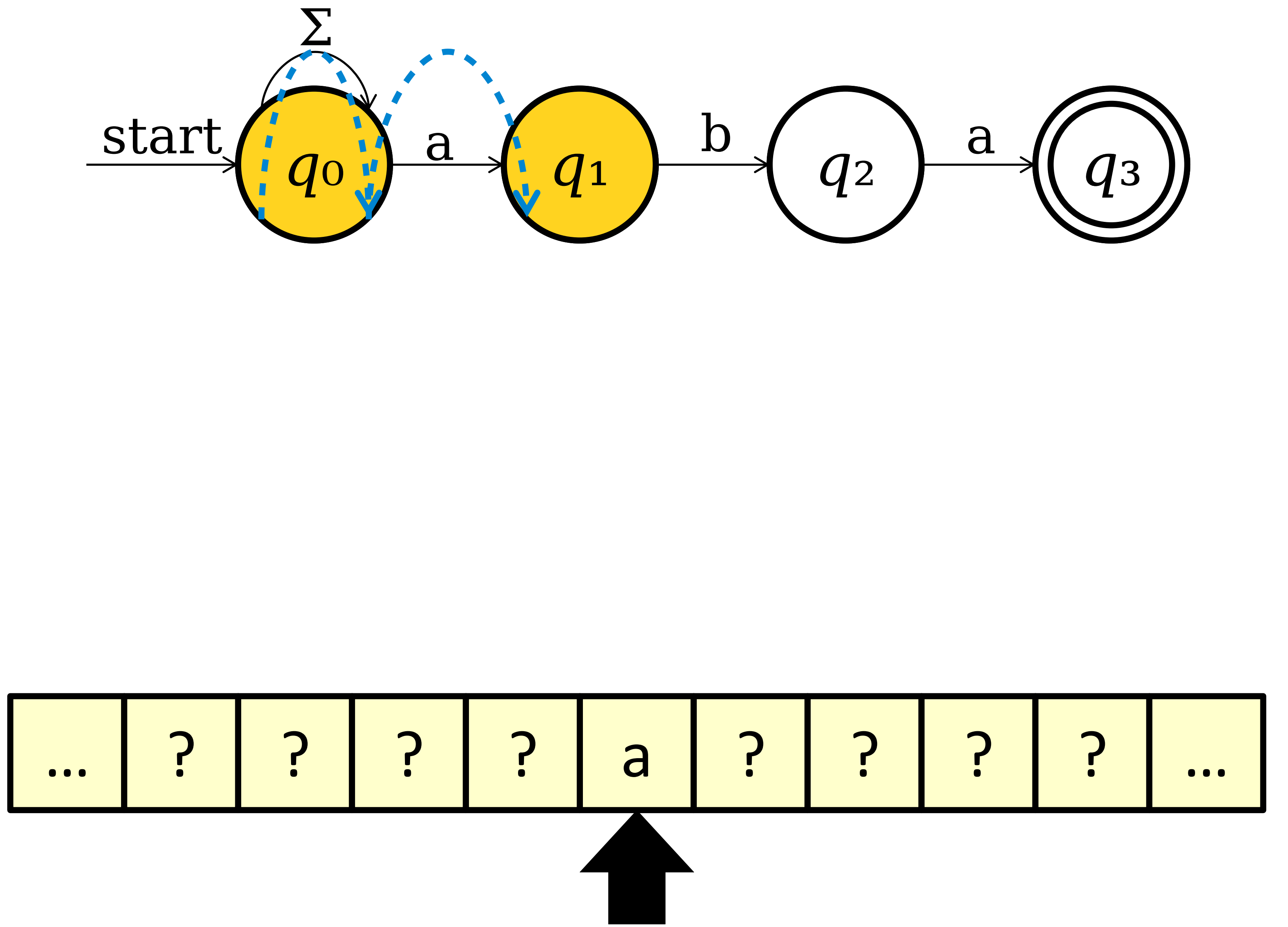

...
?
?
?
?
a
?
?
?
?
...

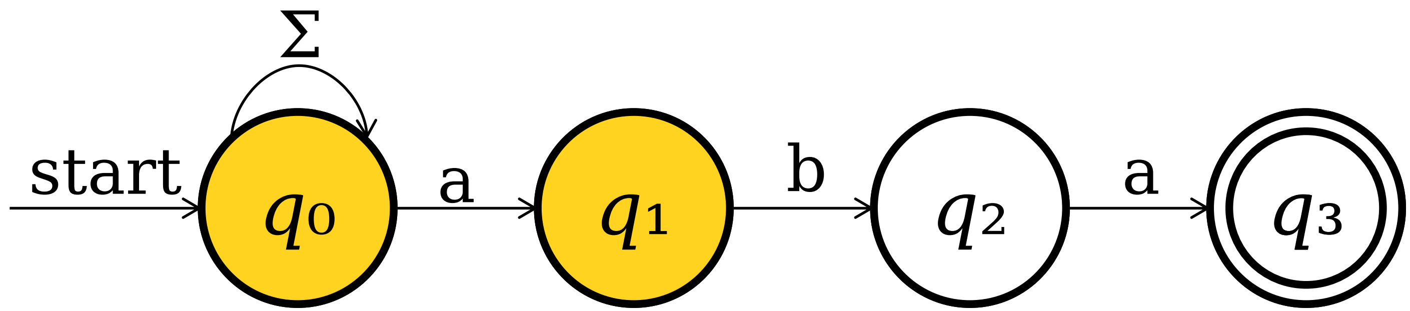

Σ
start
q0
a
q1
b
q2
a
q3

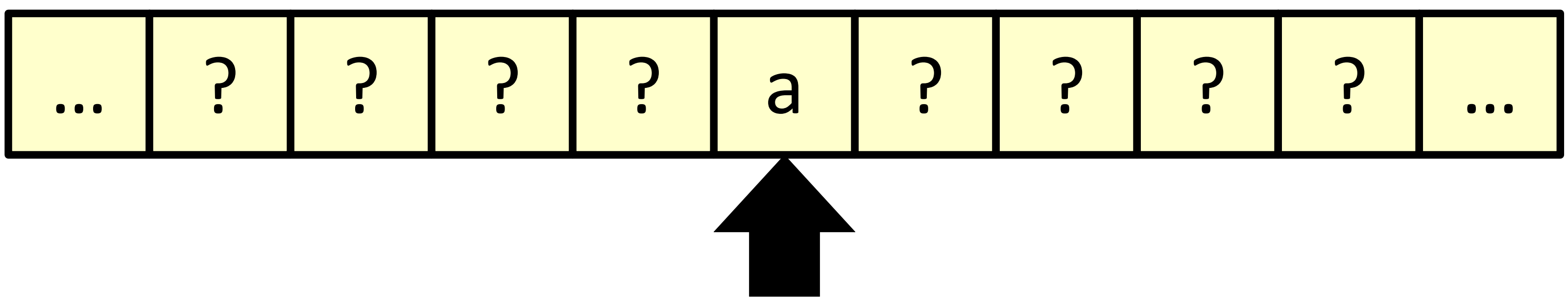

...
?
?
?
?
a
?
?
?
?
...

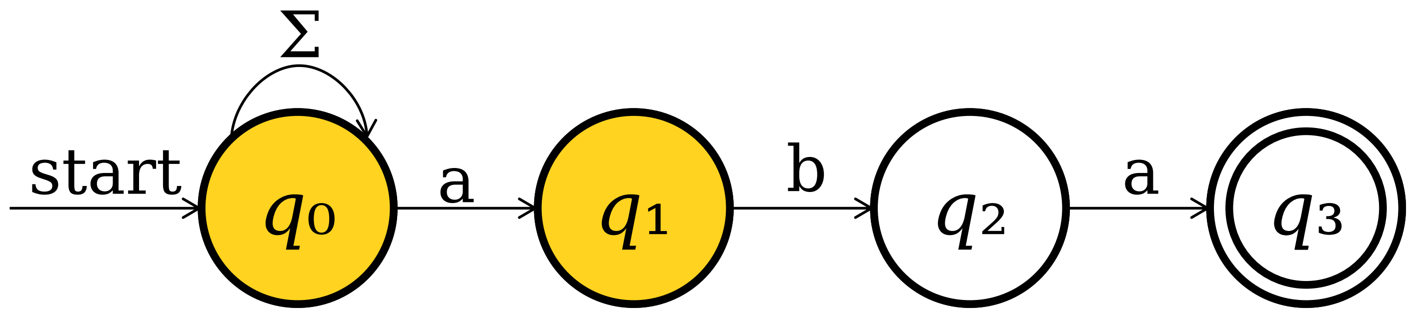

Σ
start
q_0
a
q_1
b
q_2
a
q_3

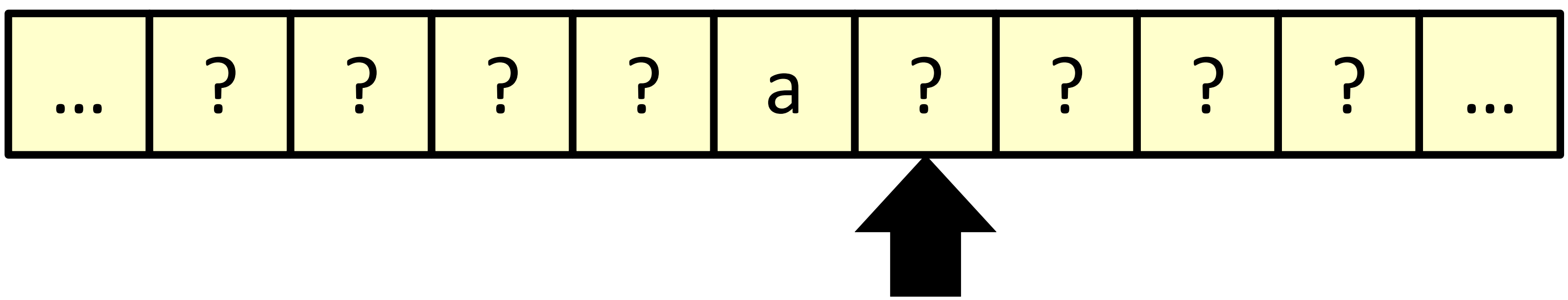

...
?
?
?
?
a
?
?
?
?
...

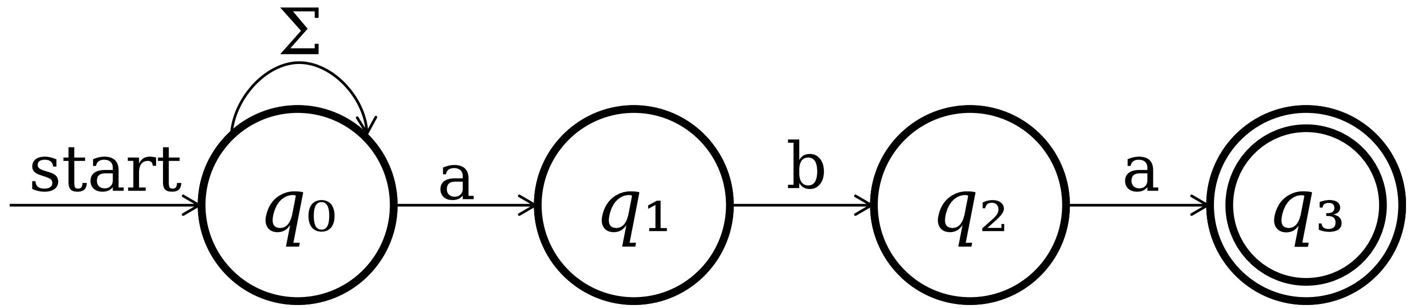

Σ
start
q0
a
q1
b
q2
a
q3
…
?
?
?
?
a
?
?
?
?
…


Σ
start
q_0
a
q_1
b
q_2
a
q_3

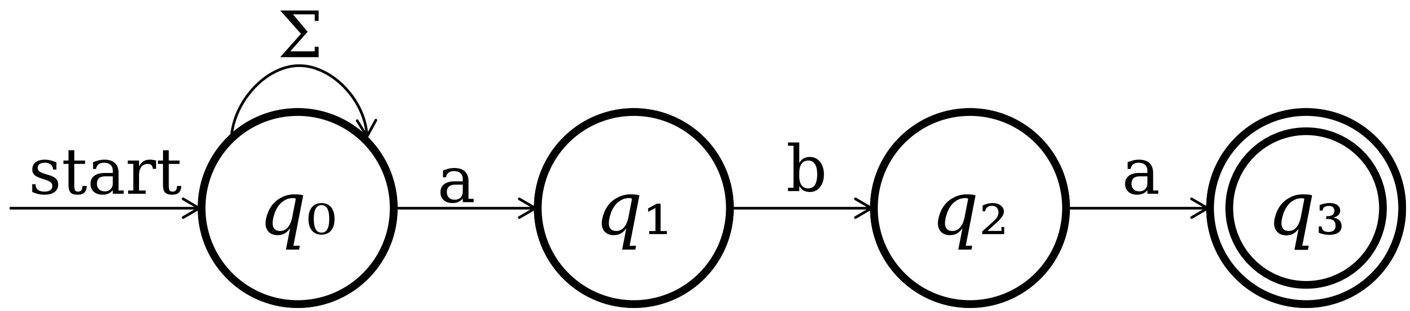

...
?
?
?
?
a
?
?
?
?
...


Σ
start
q0
a
q1
b
q2
a
q3

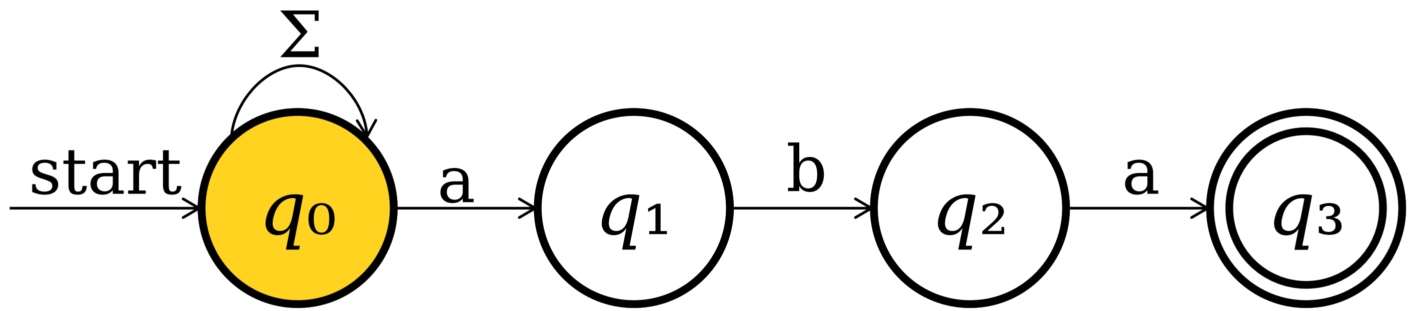

...
?
?
?
?
a
?
?
?
?
...



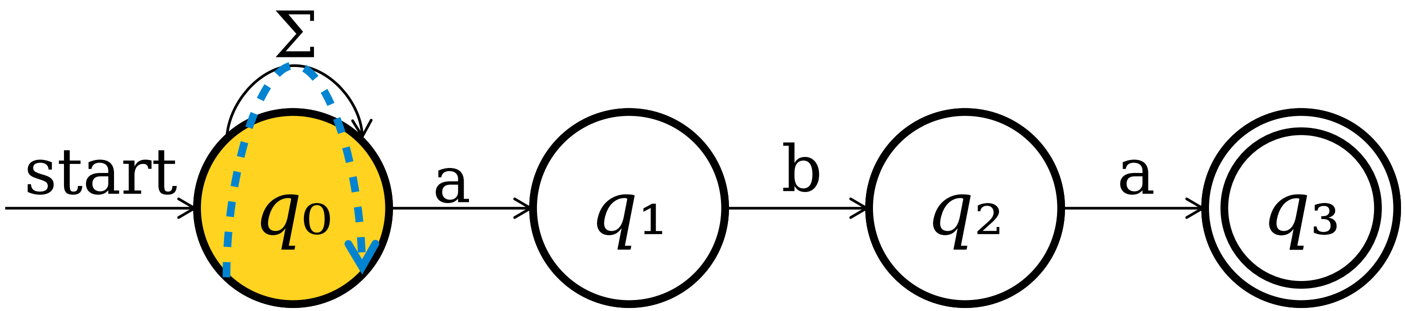
	a
$\{q_0\}$	$\{q_0, q_1\}$



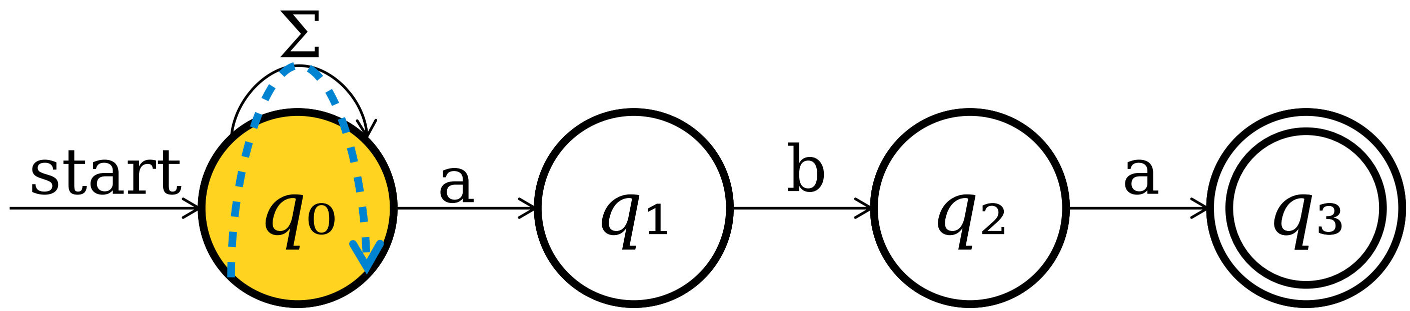
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	



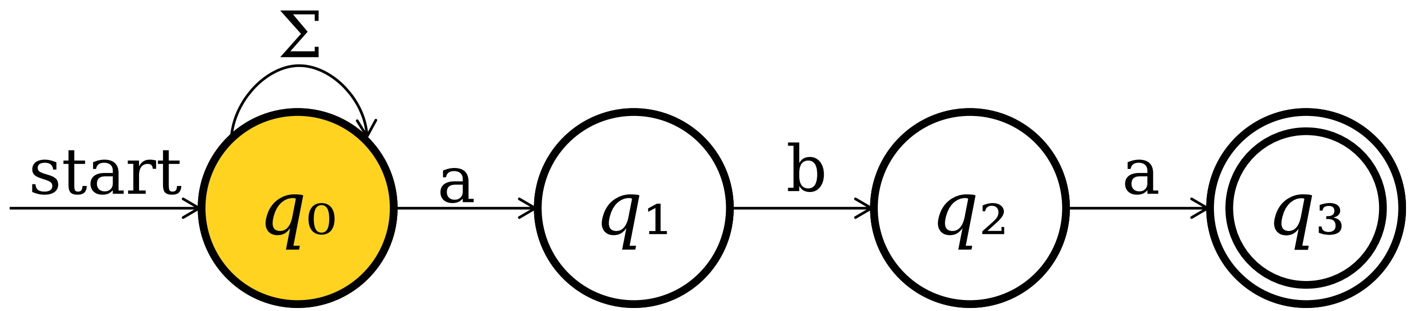
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	



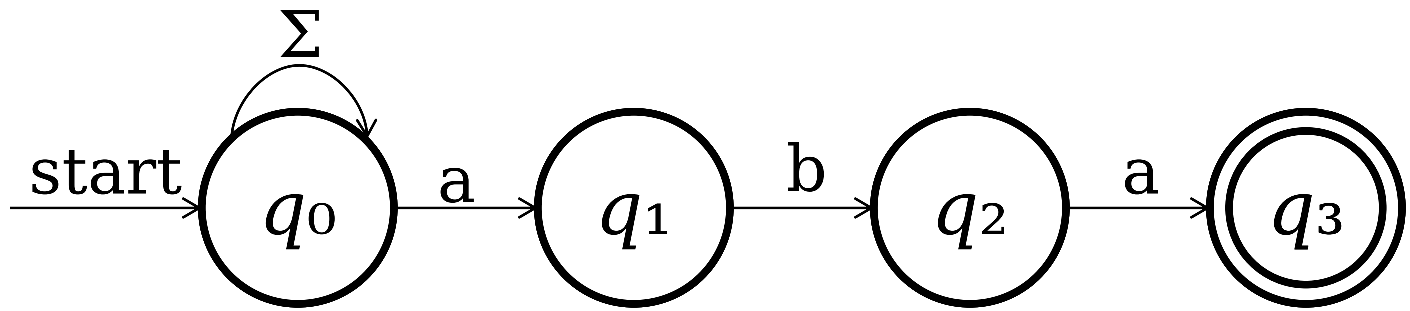
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	



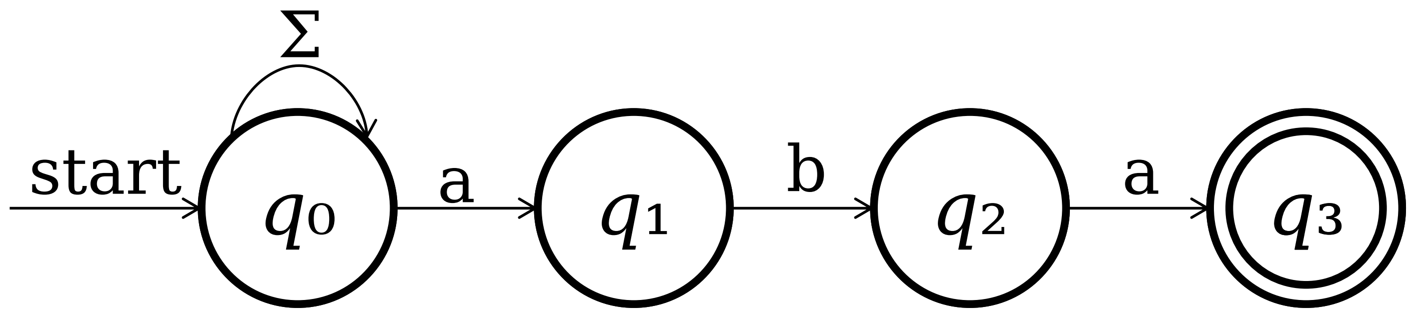
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$



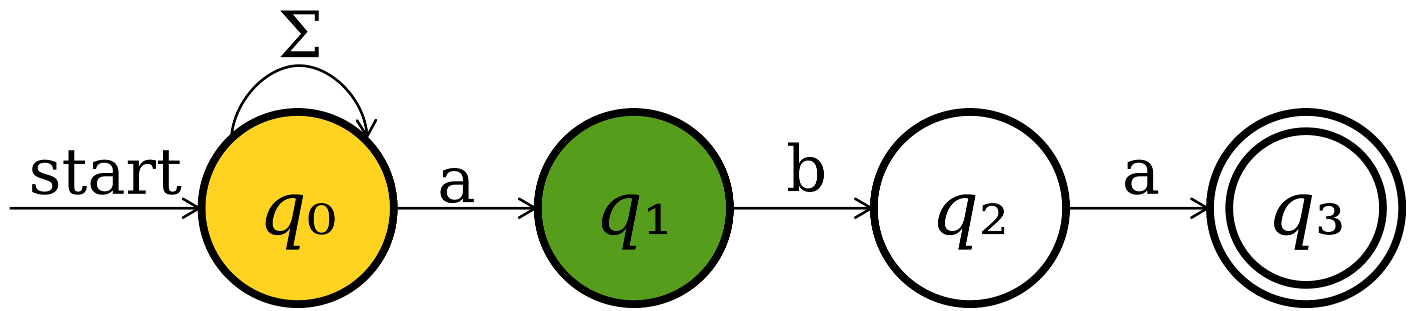
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$



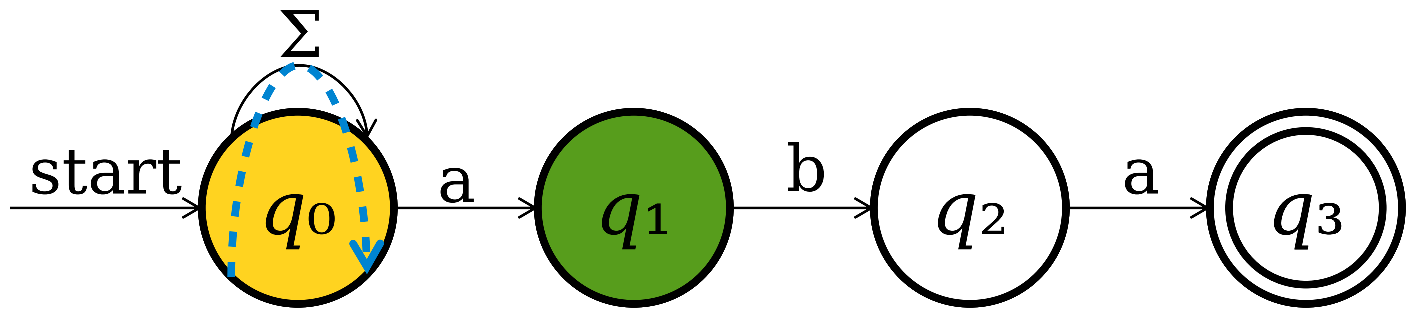
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$



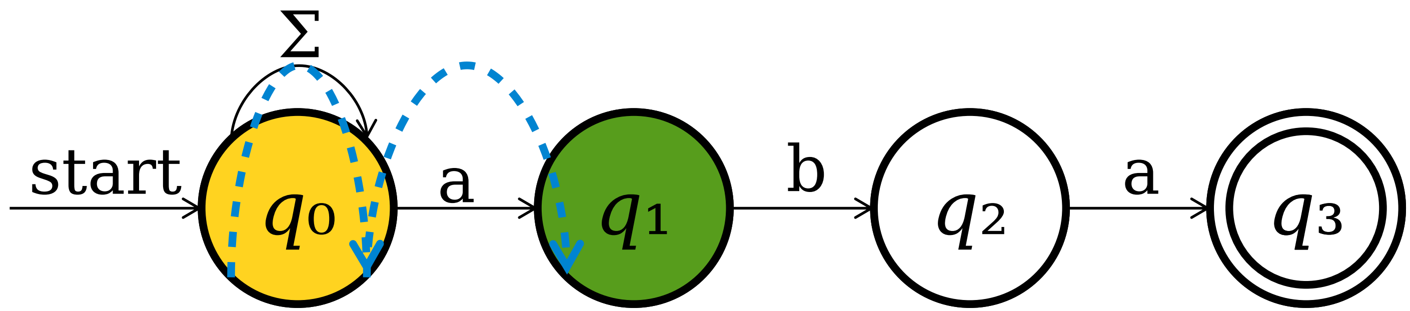
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



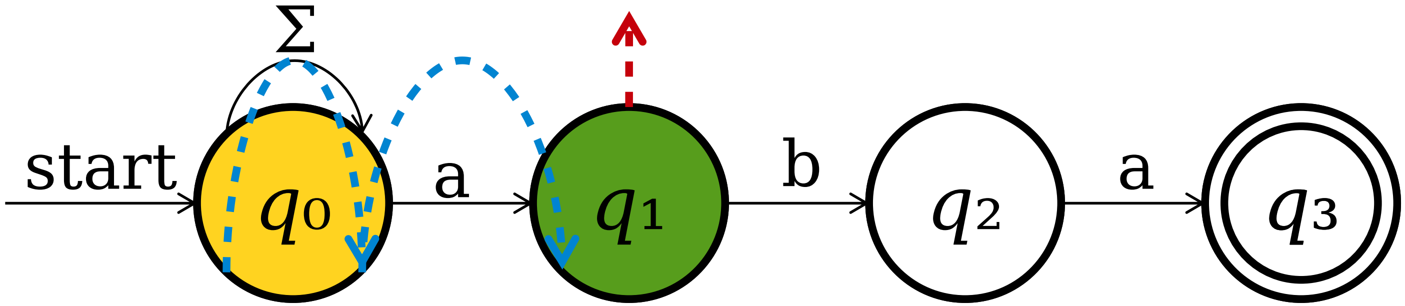
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



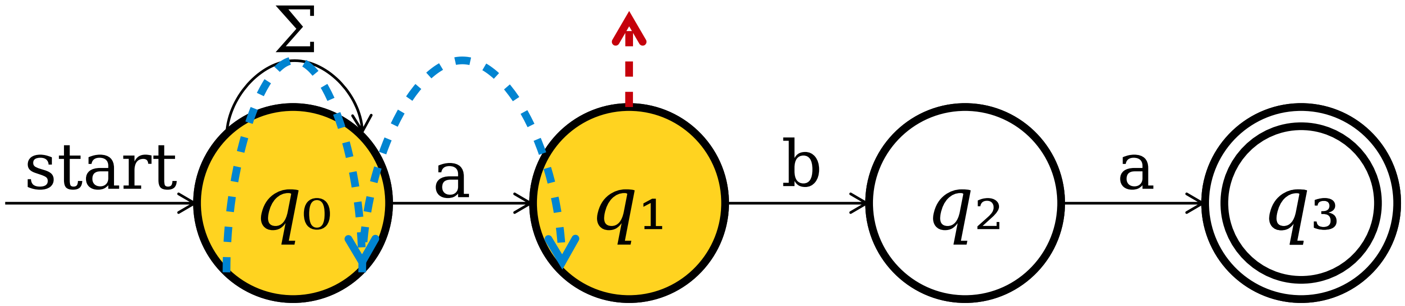
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



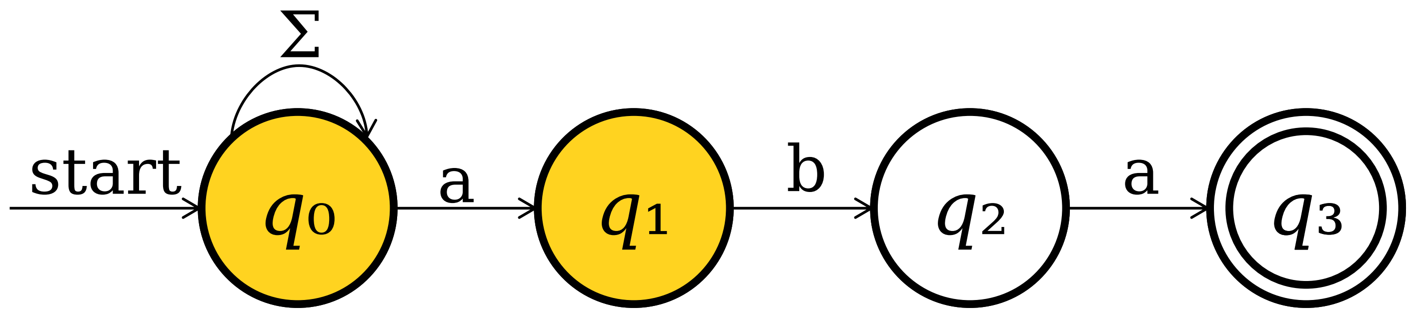
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



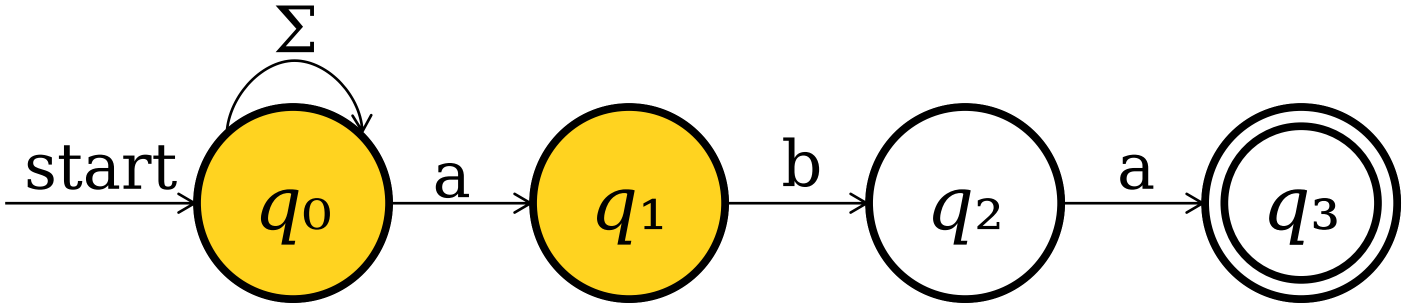
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



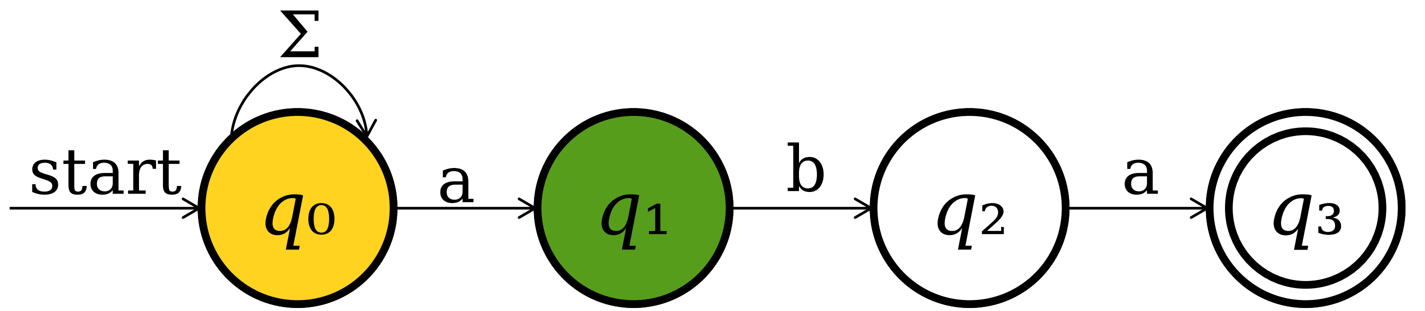
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



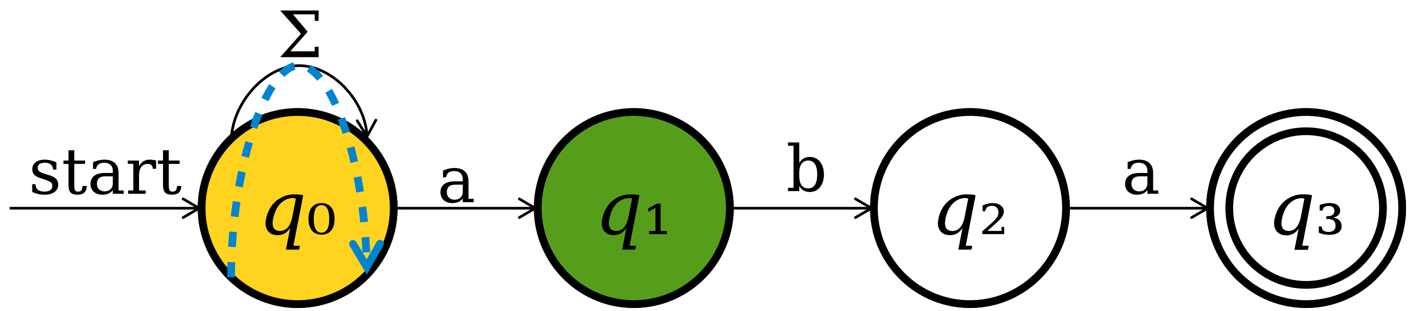
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



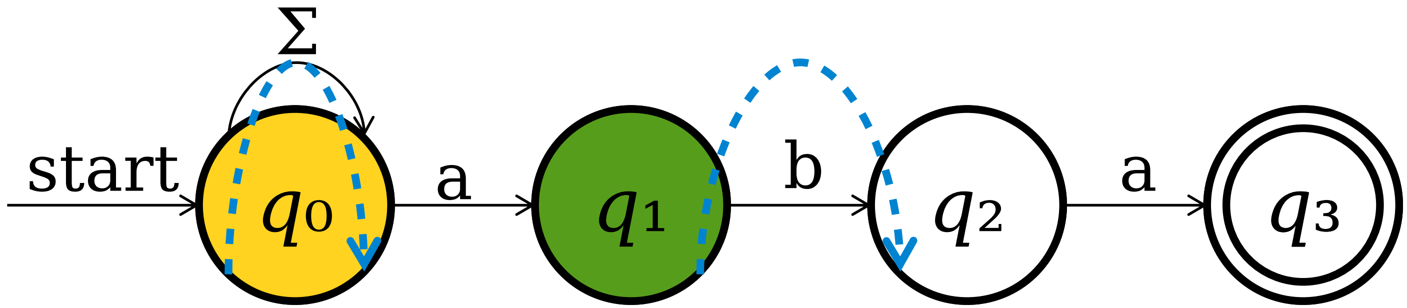
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	



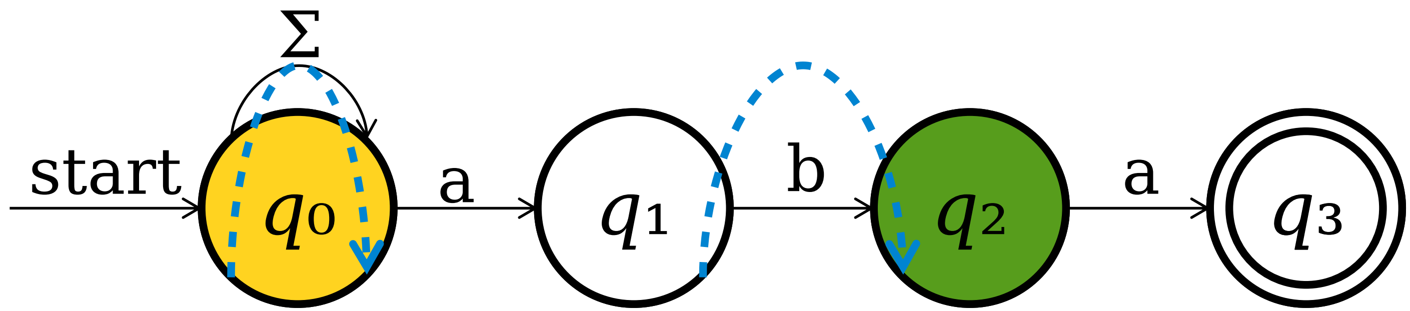
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	



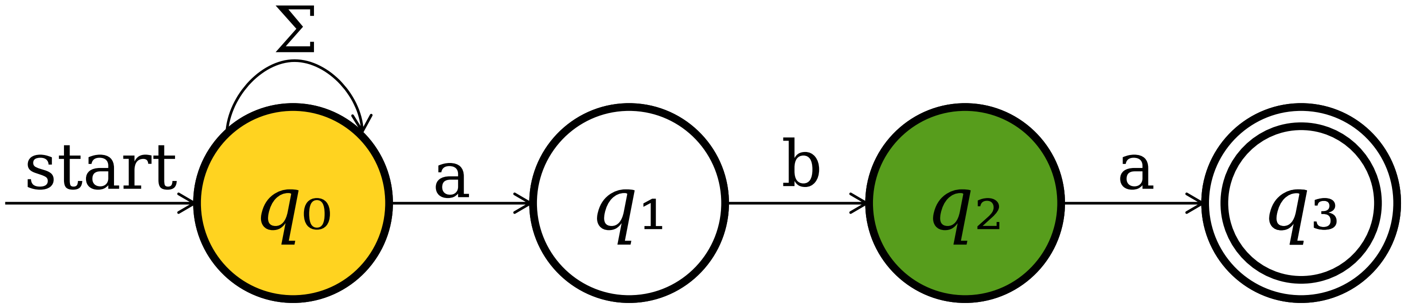
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	



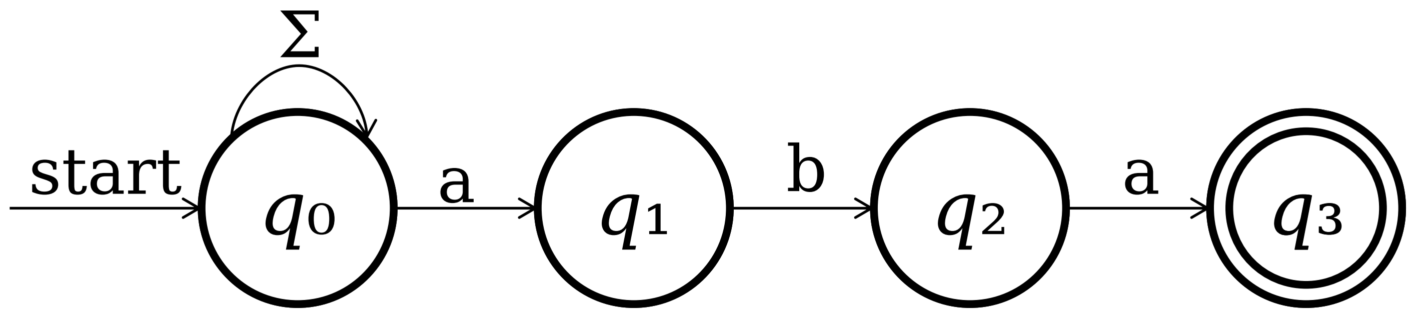
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	



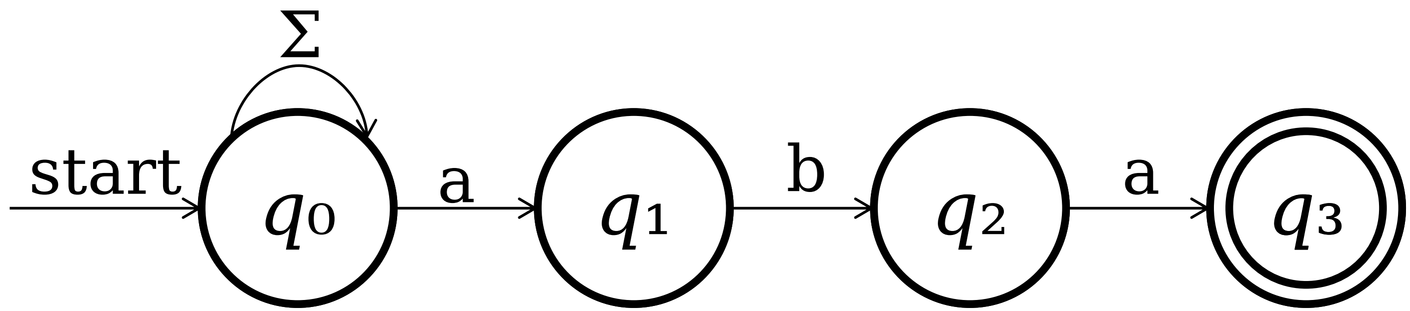
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	



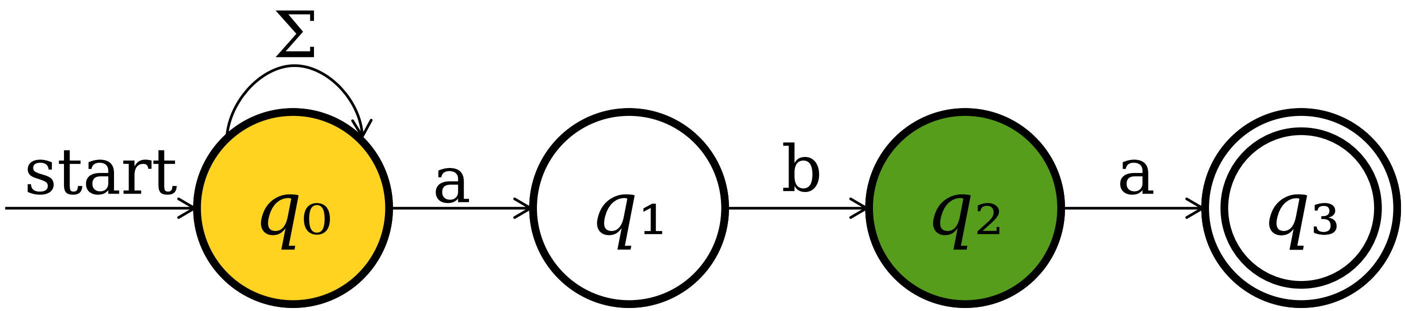
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



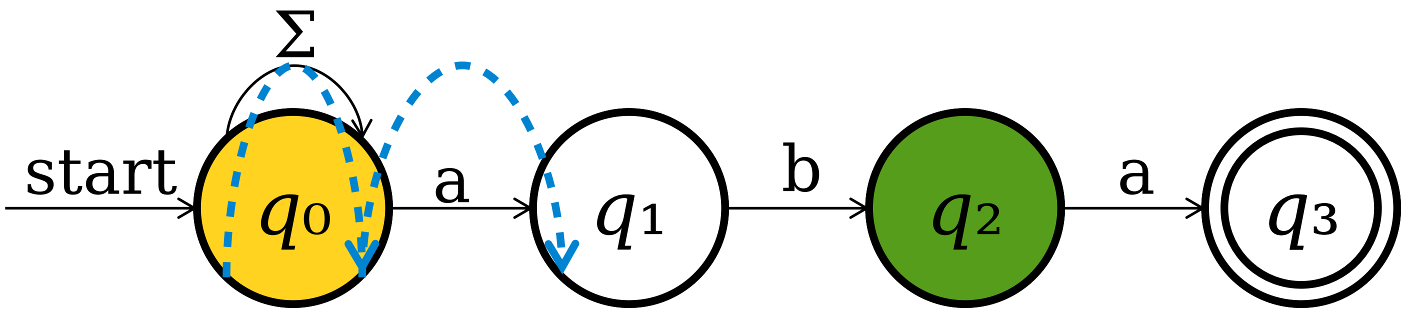
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



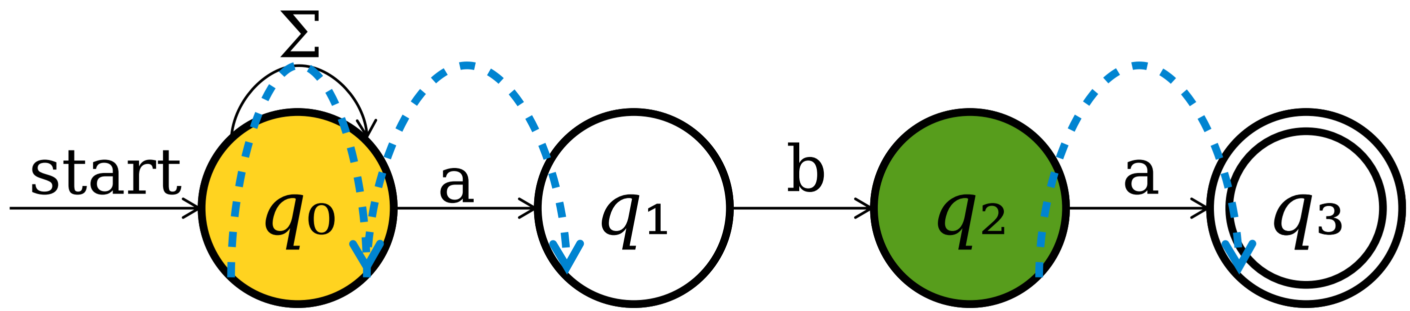
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		



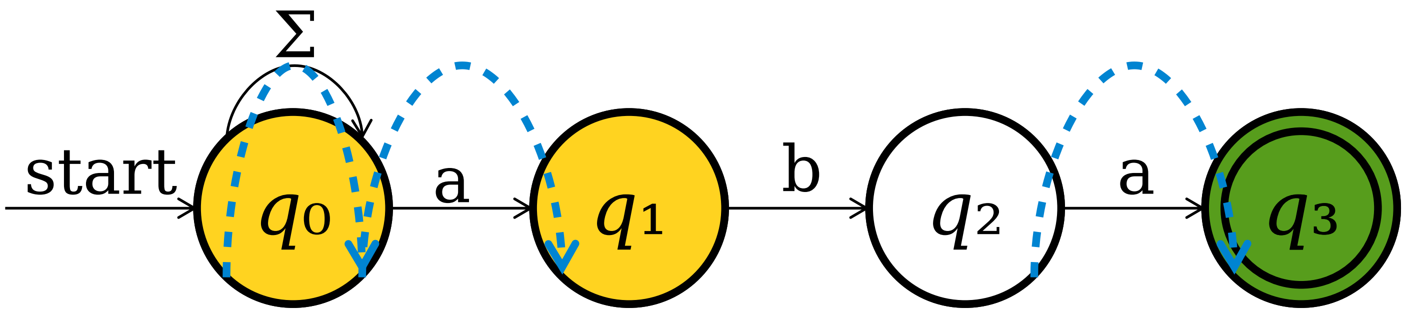
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		



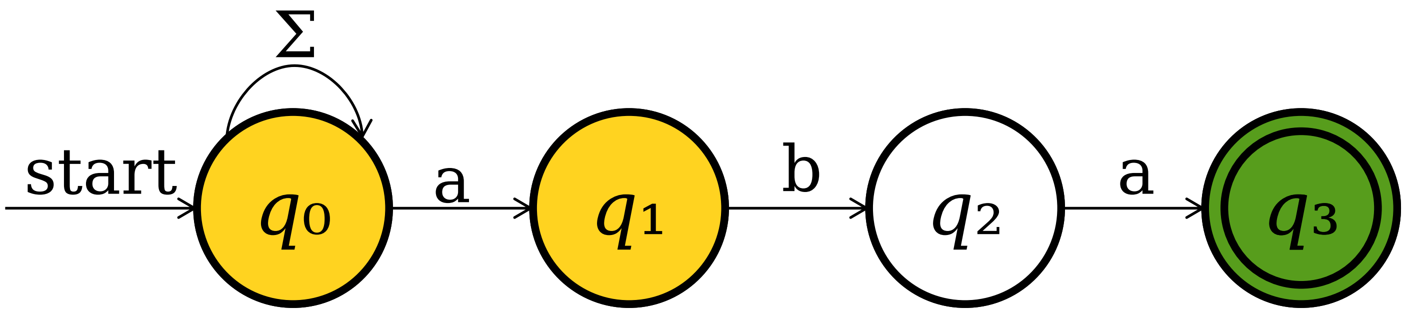
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		



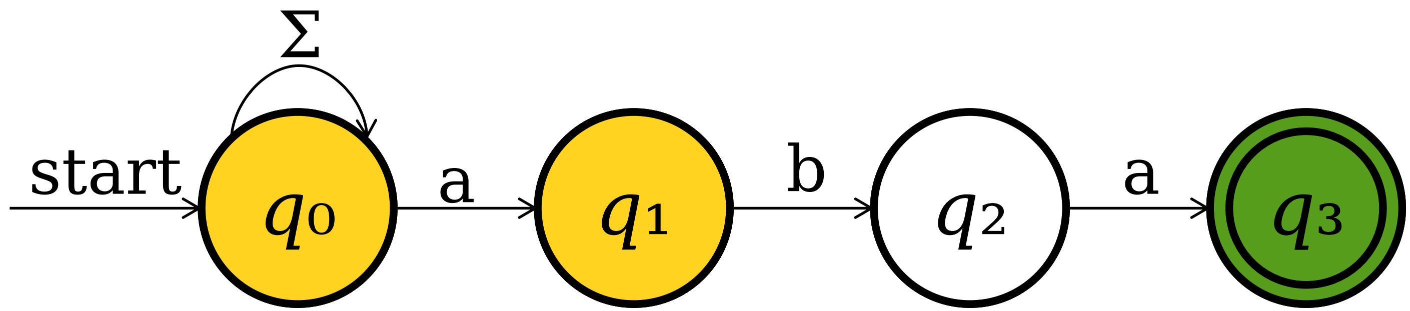
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		



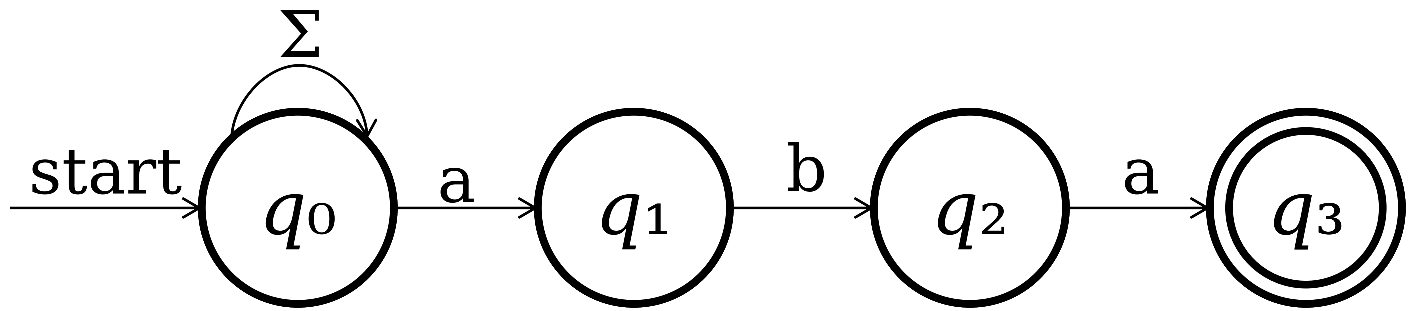
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		



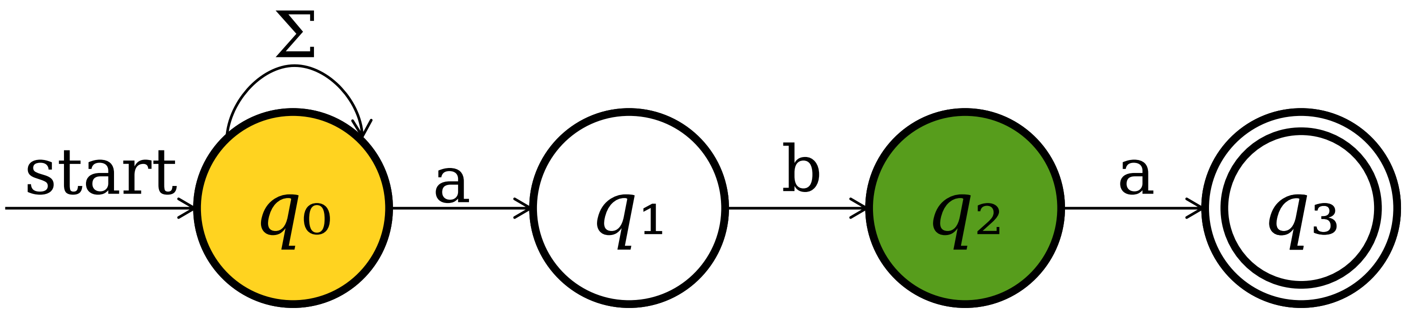
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		



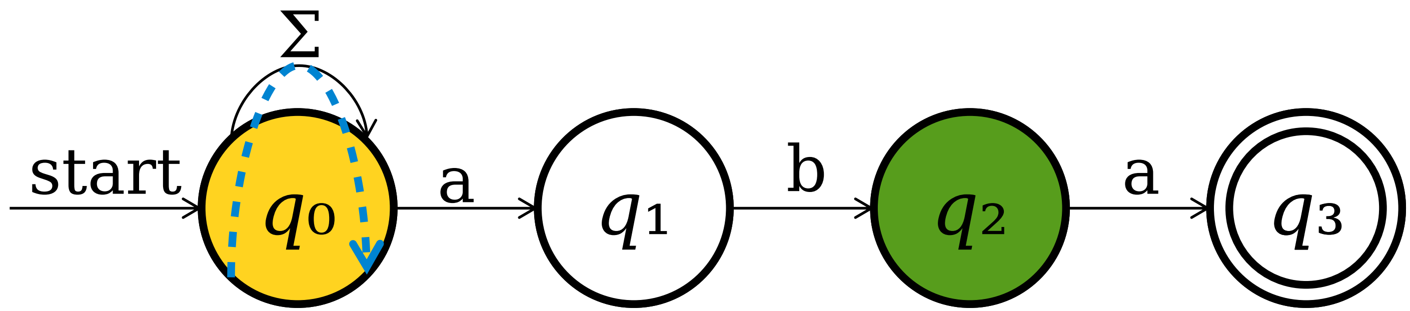
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



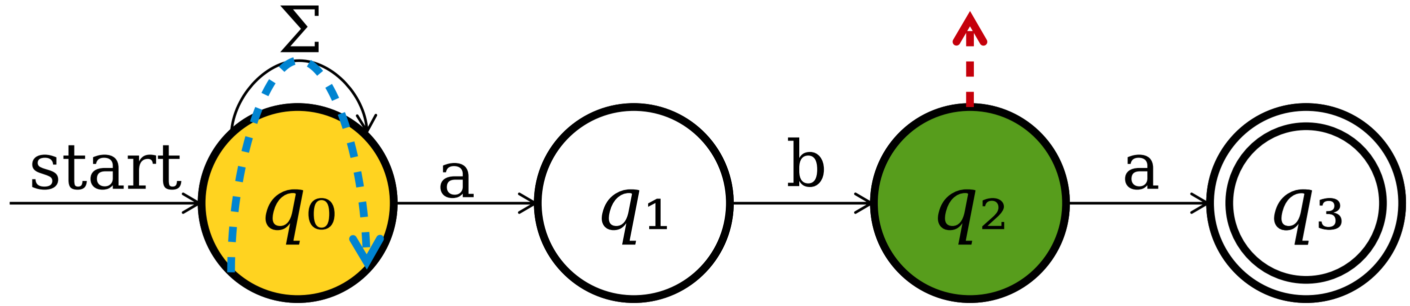
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



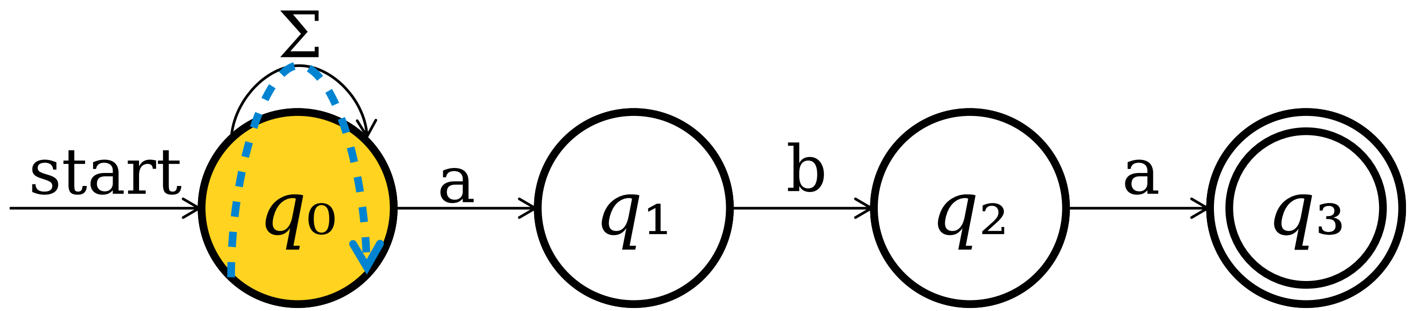
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



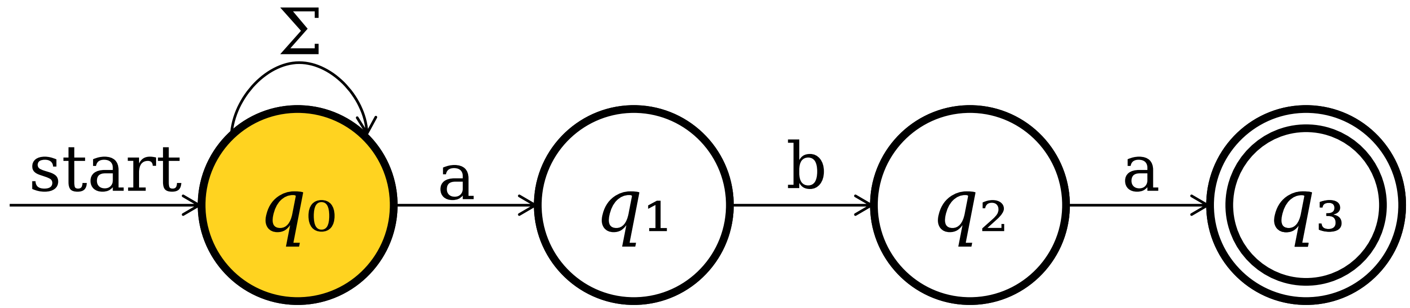
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



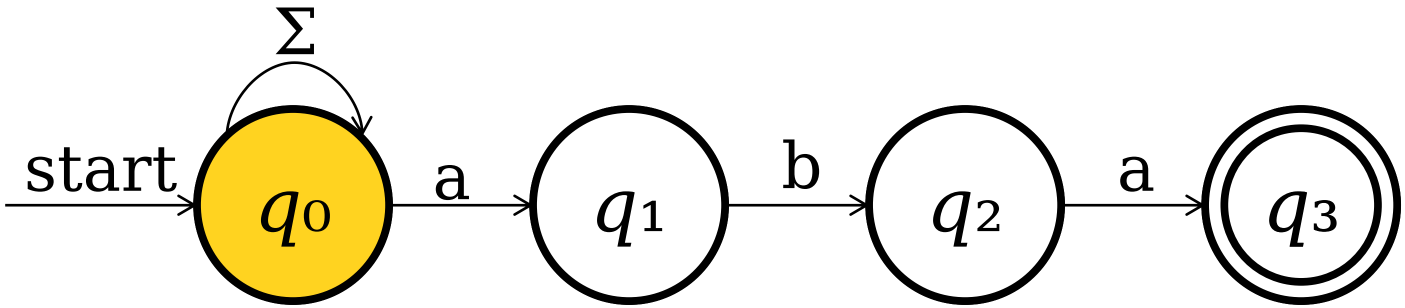
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



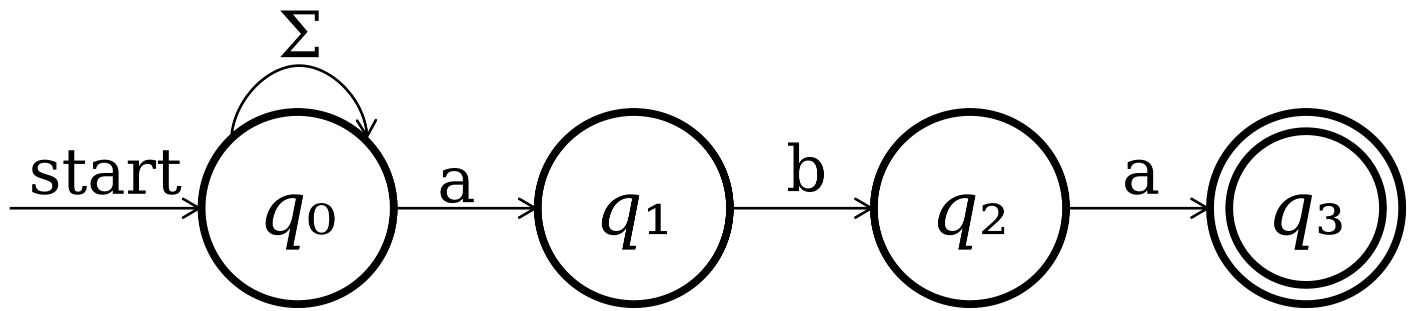
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



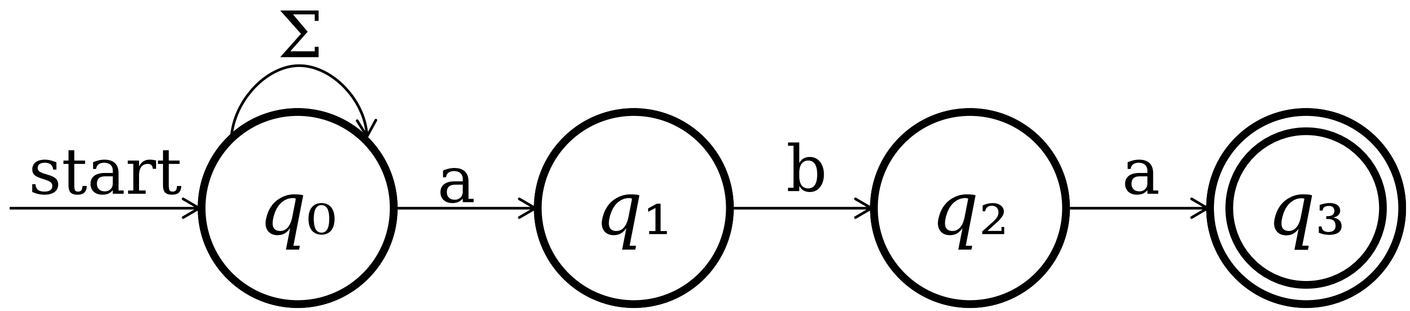
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



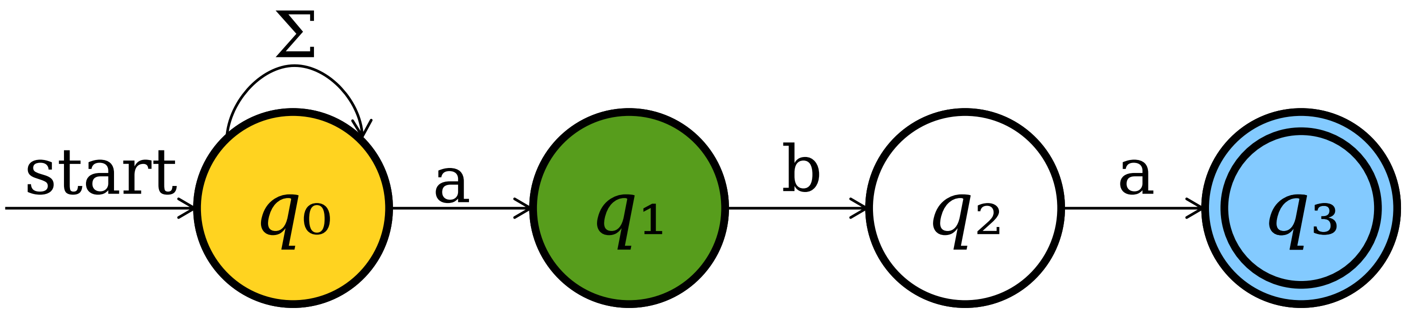
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$



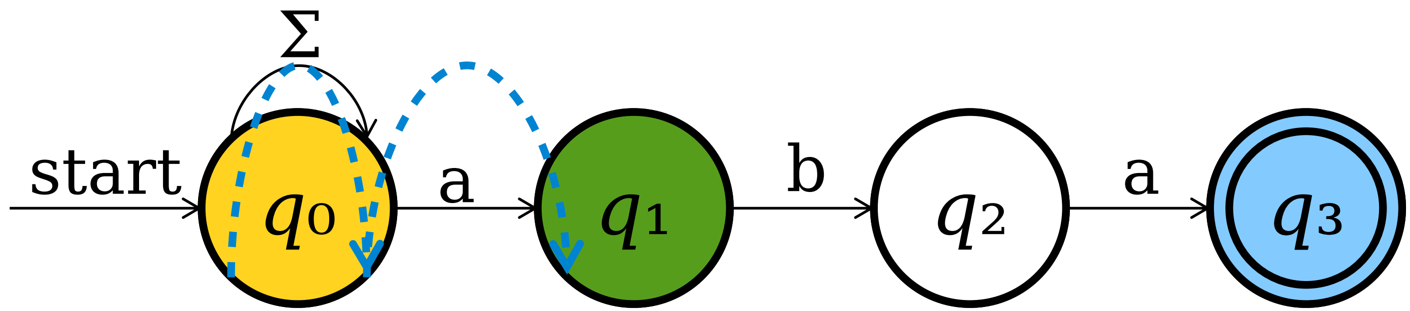
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$



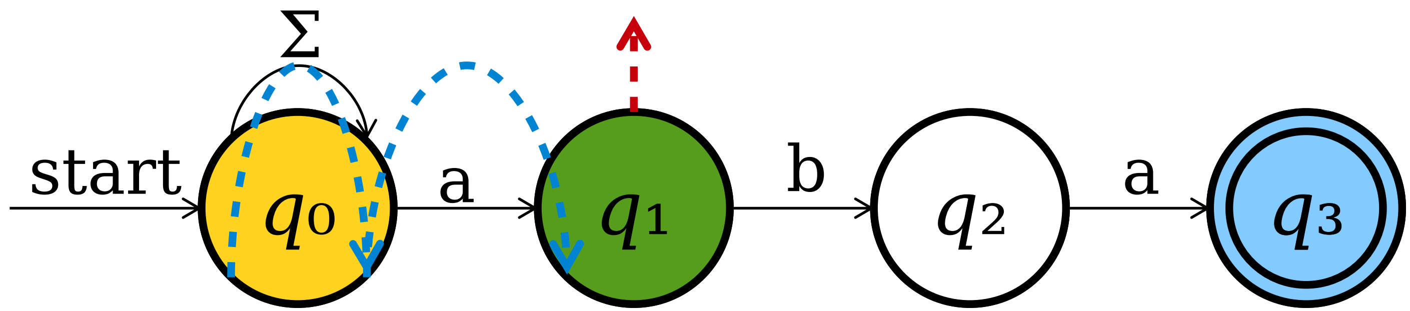
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



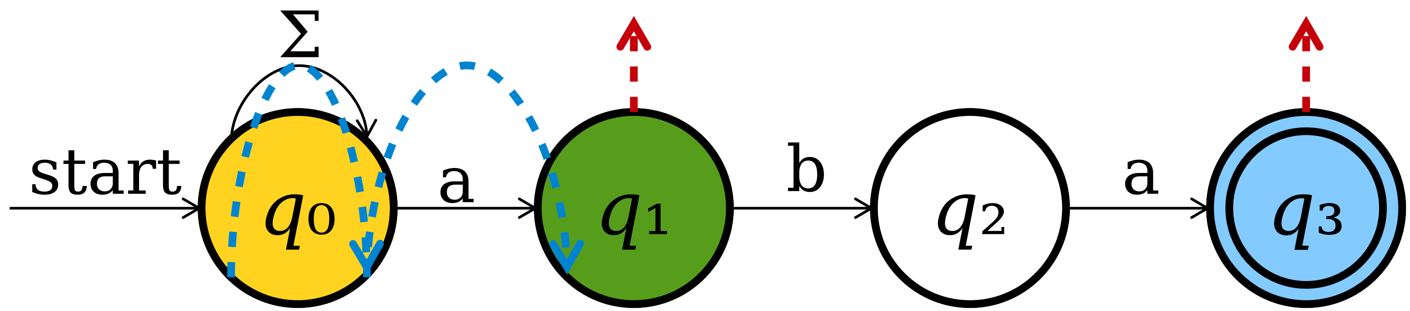
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



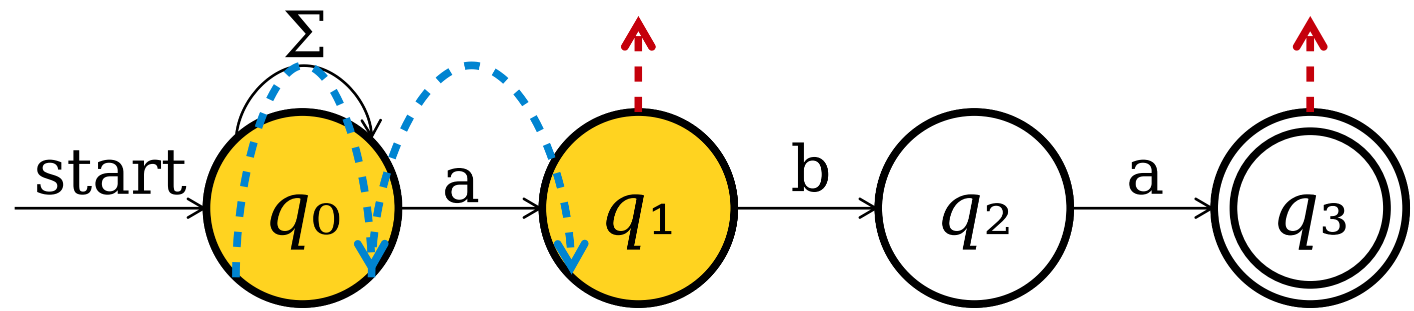
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



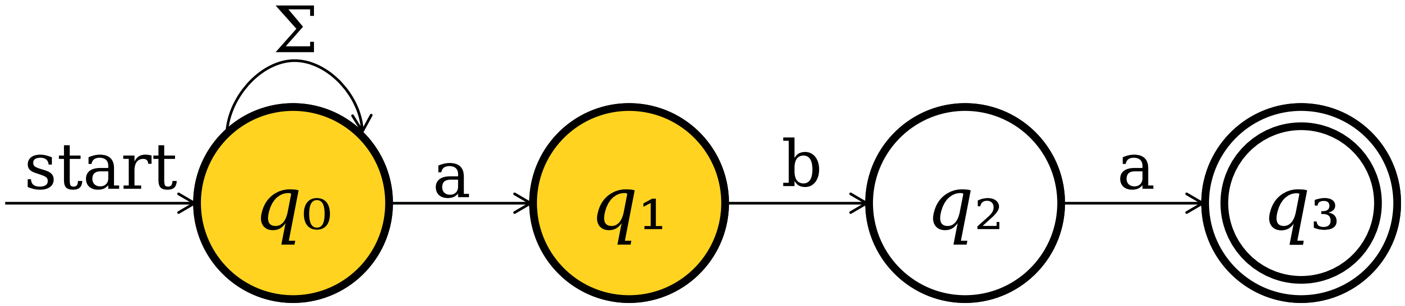
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



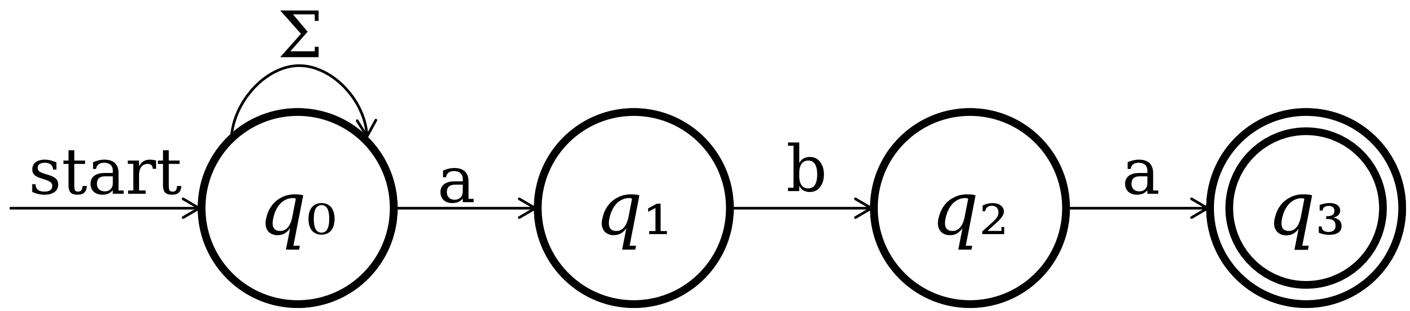
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



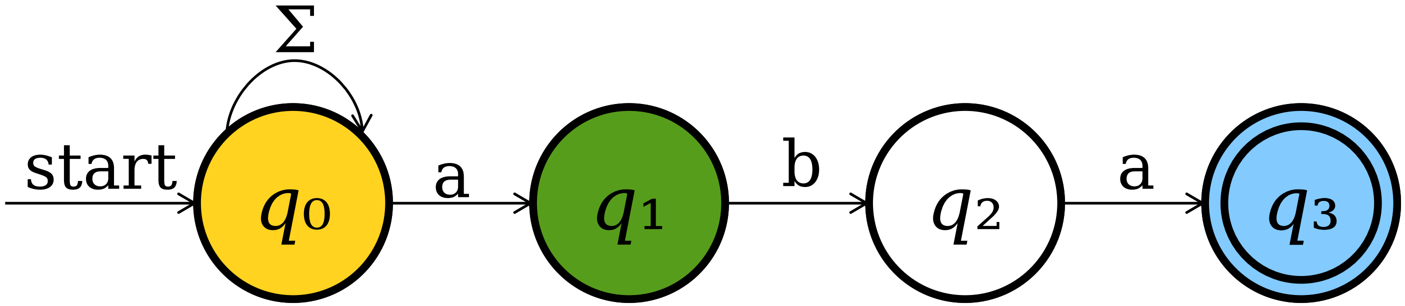
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



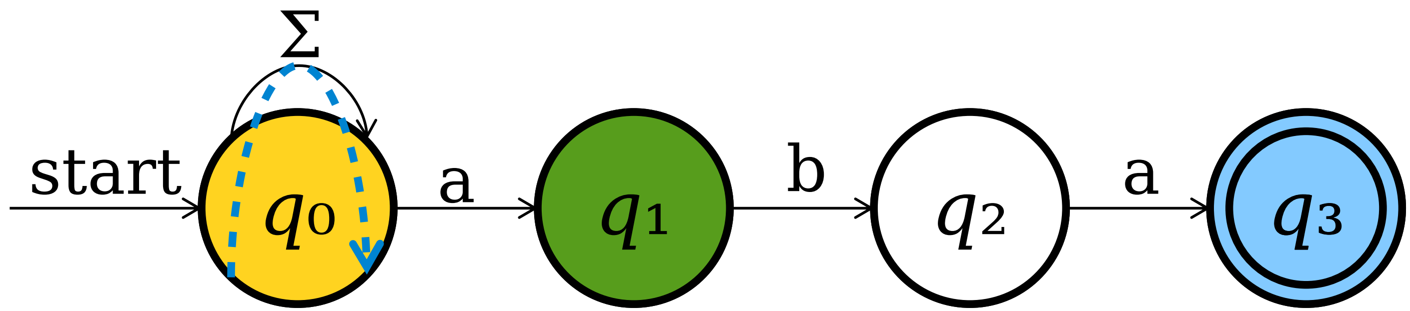
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



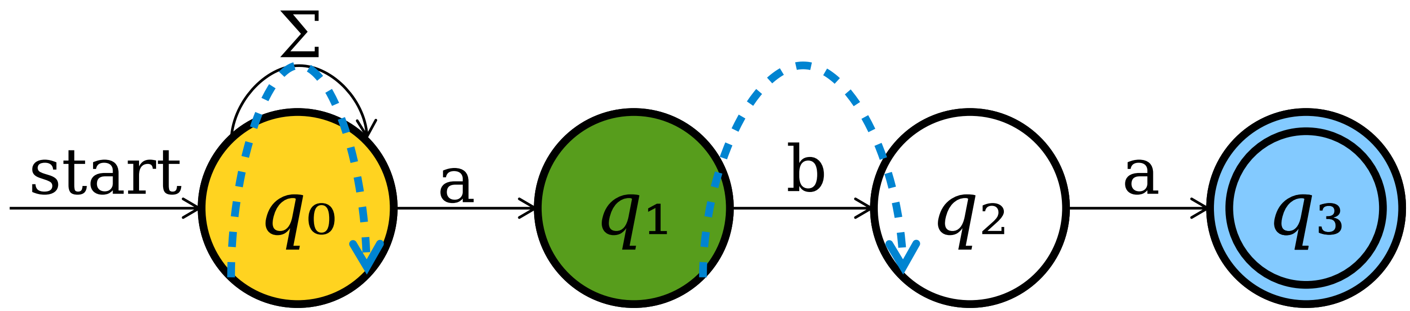
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



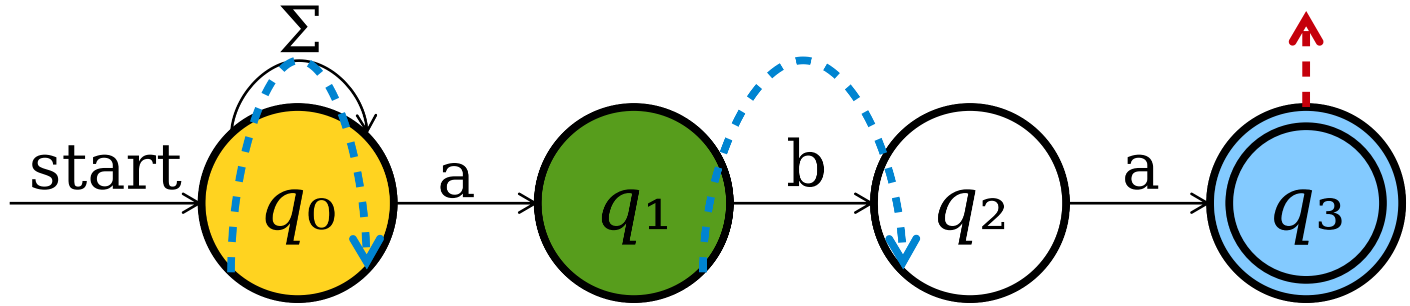
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



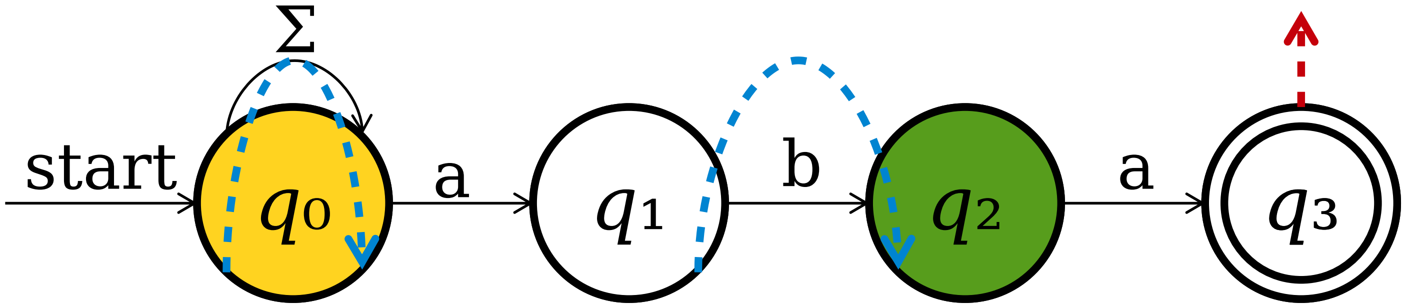
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



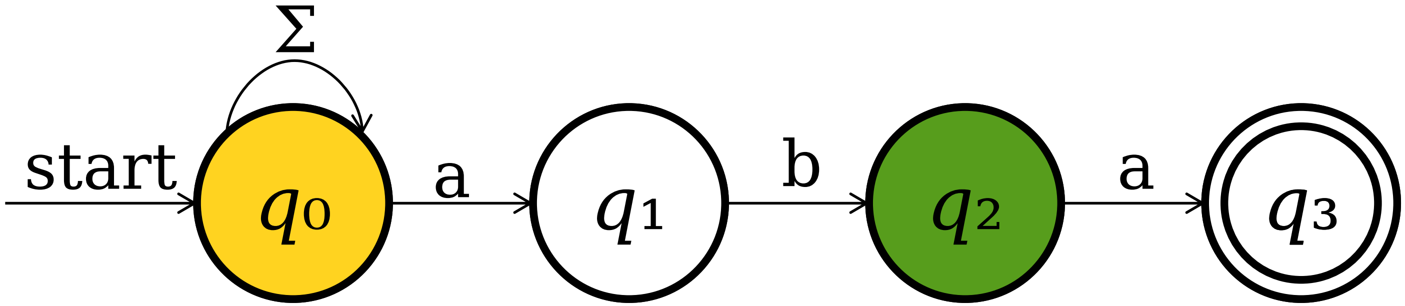
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



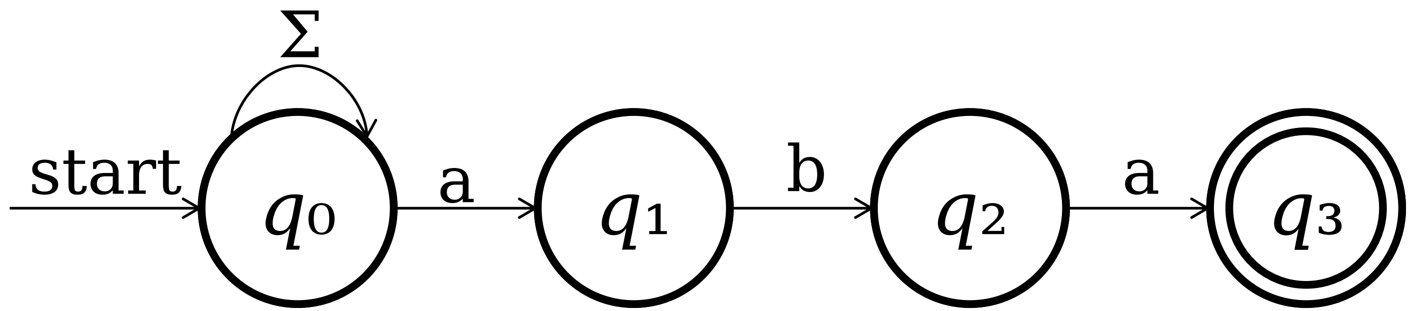
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



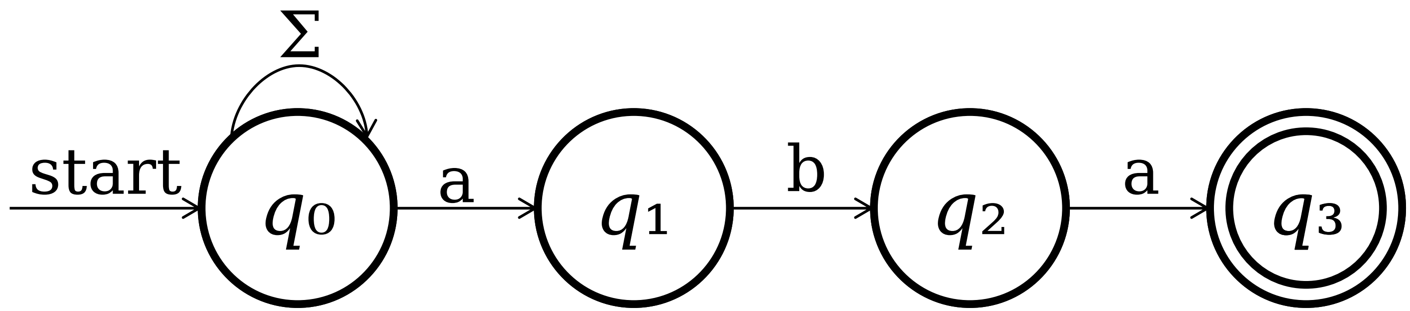
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



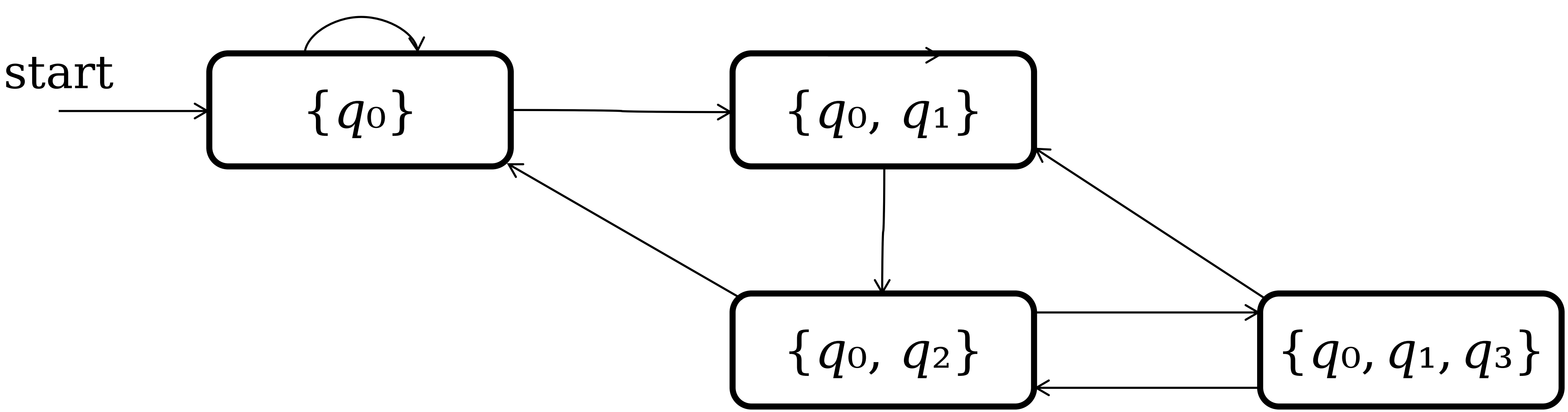
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

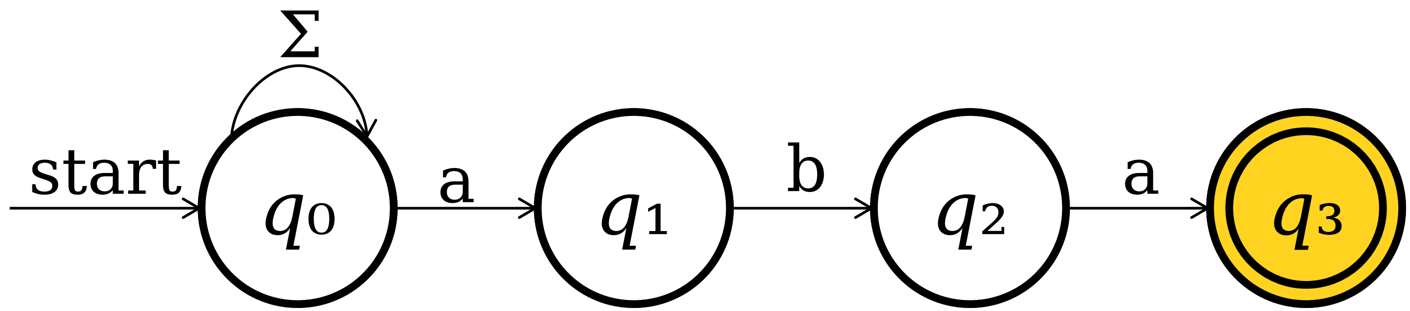


	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

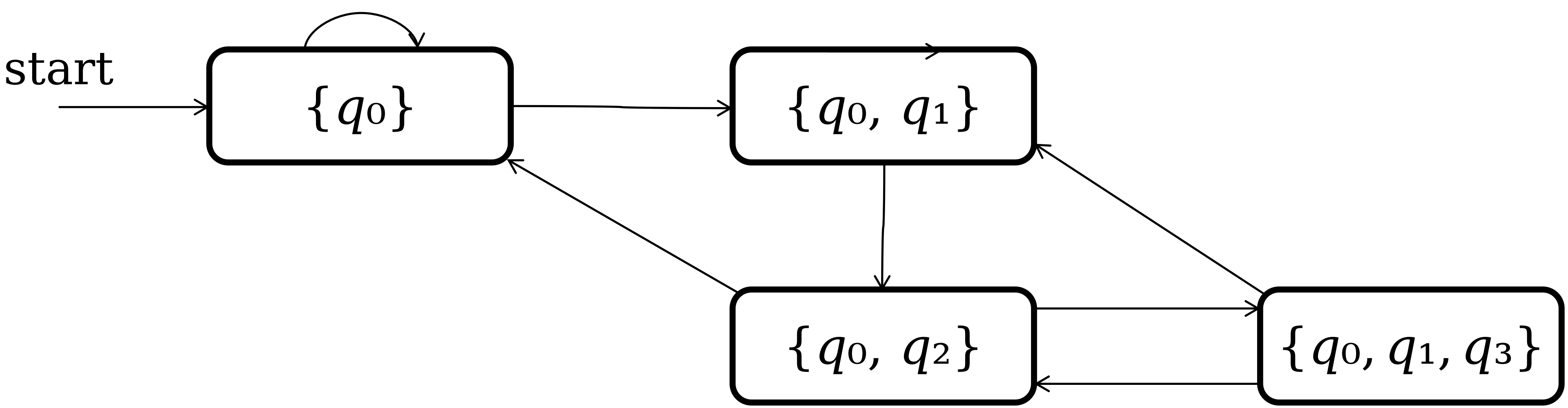


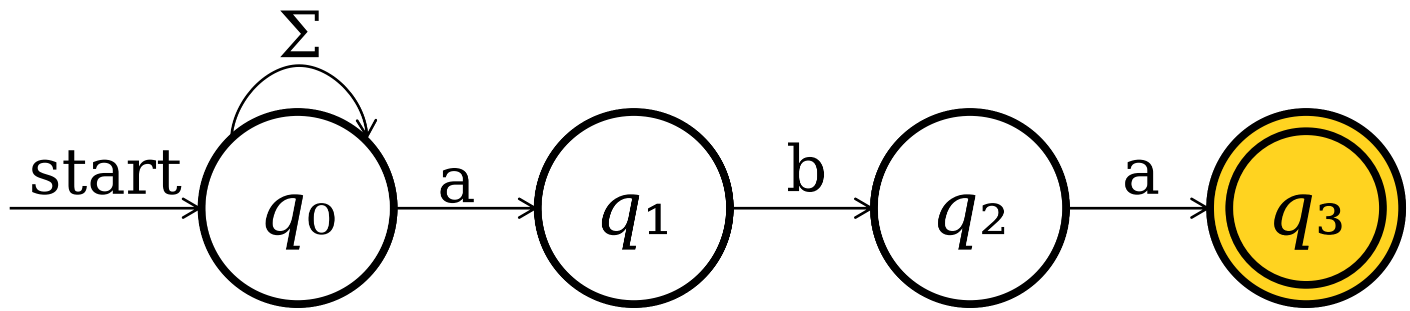
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



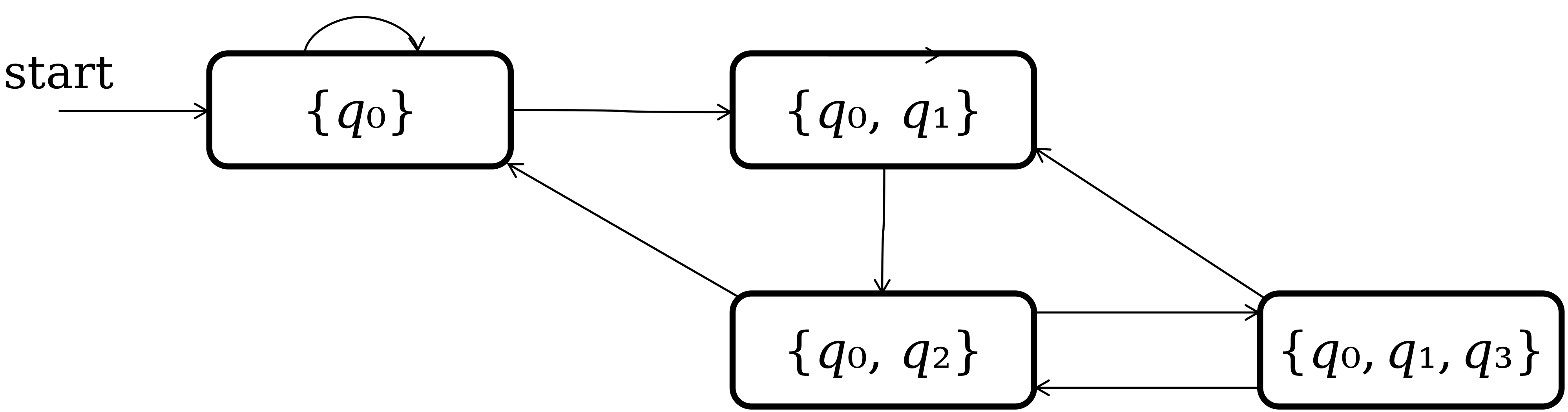


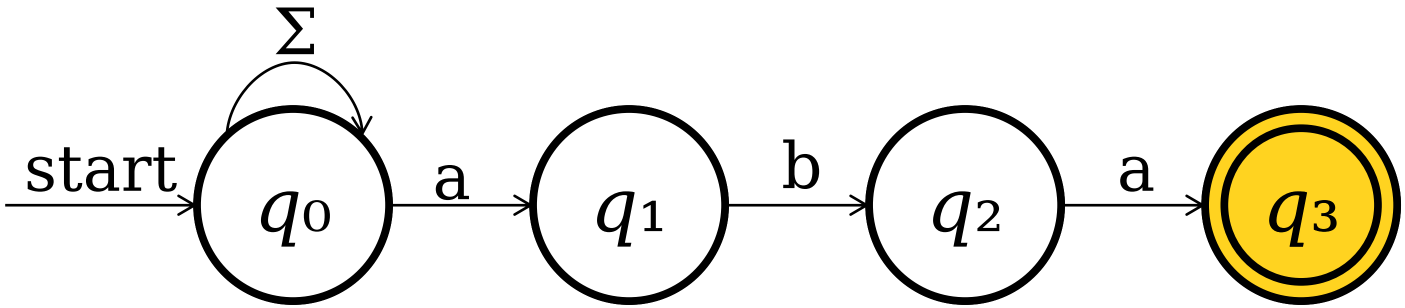
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



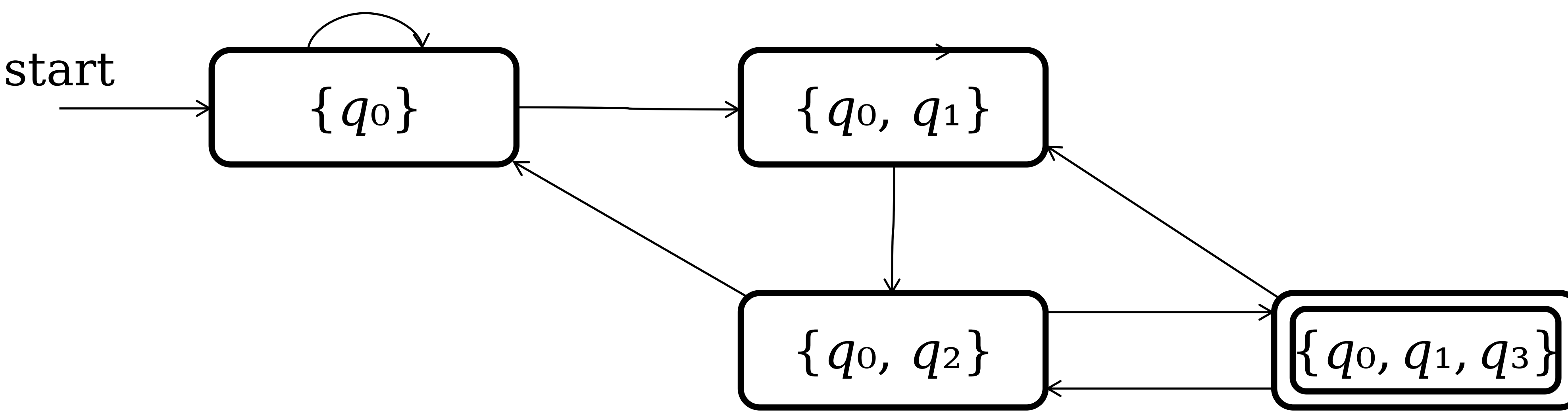


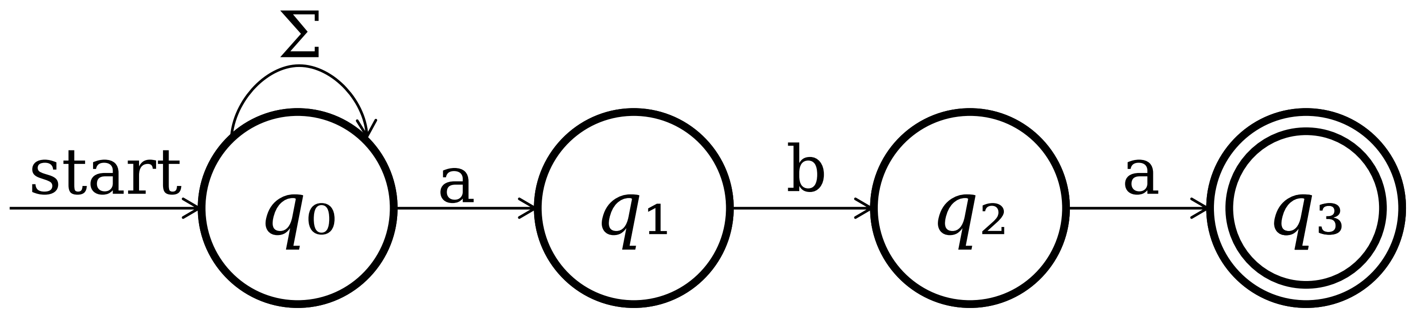
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$*\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



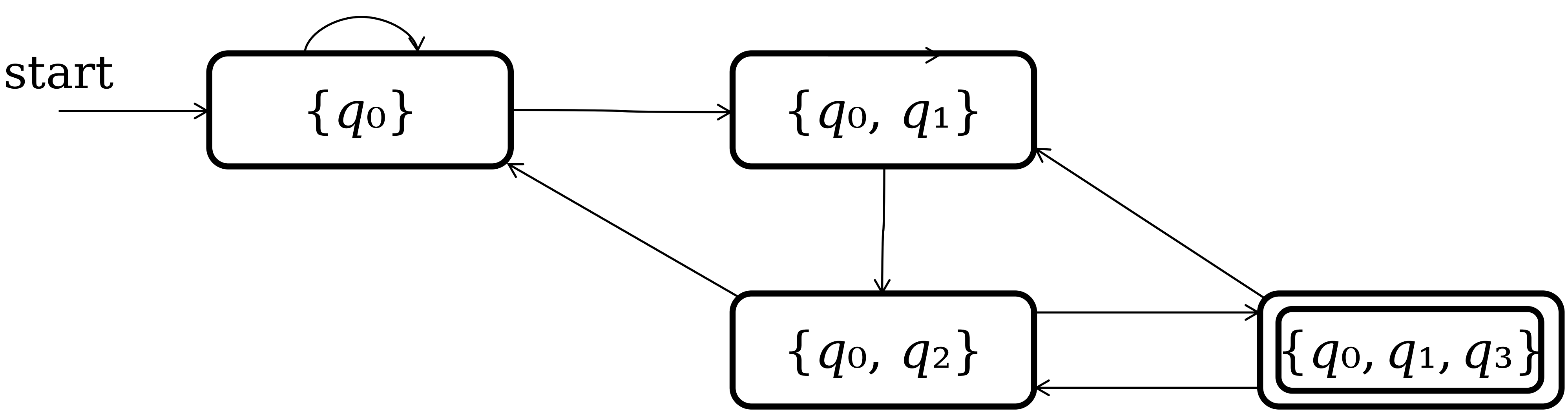


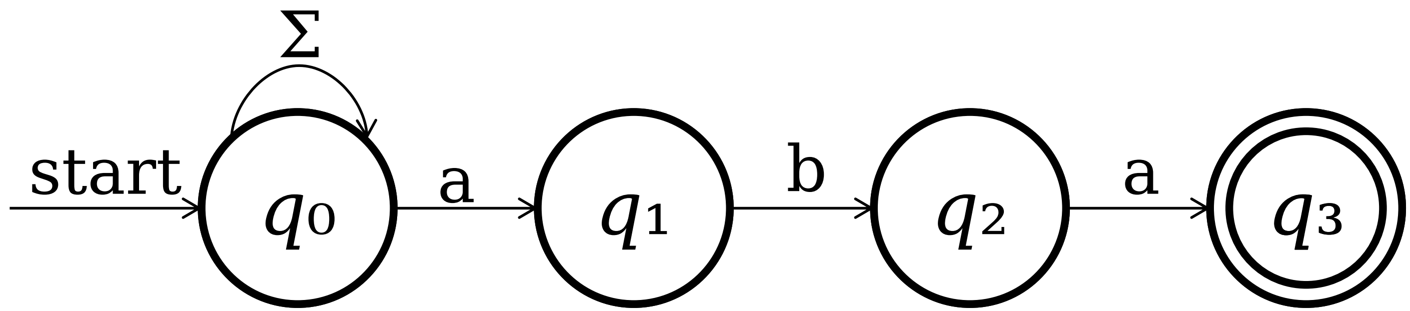
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$*\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



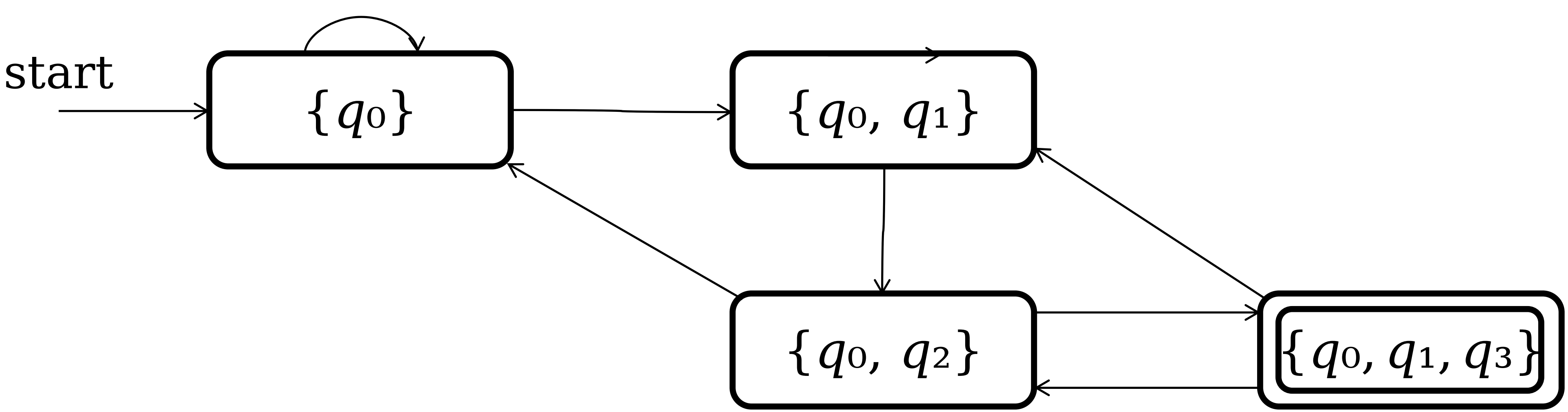


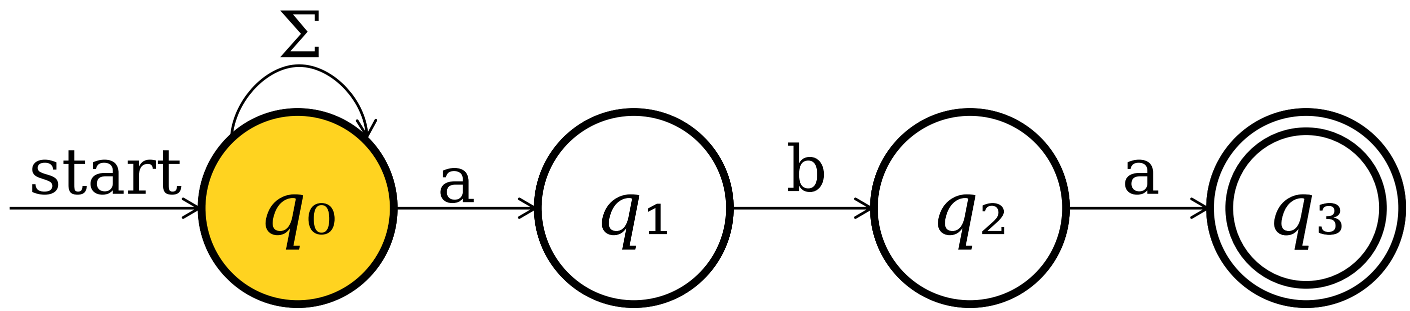
	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$*\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



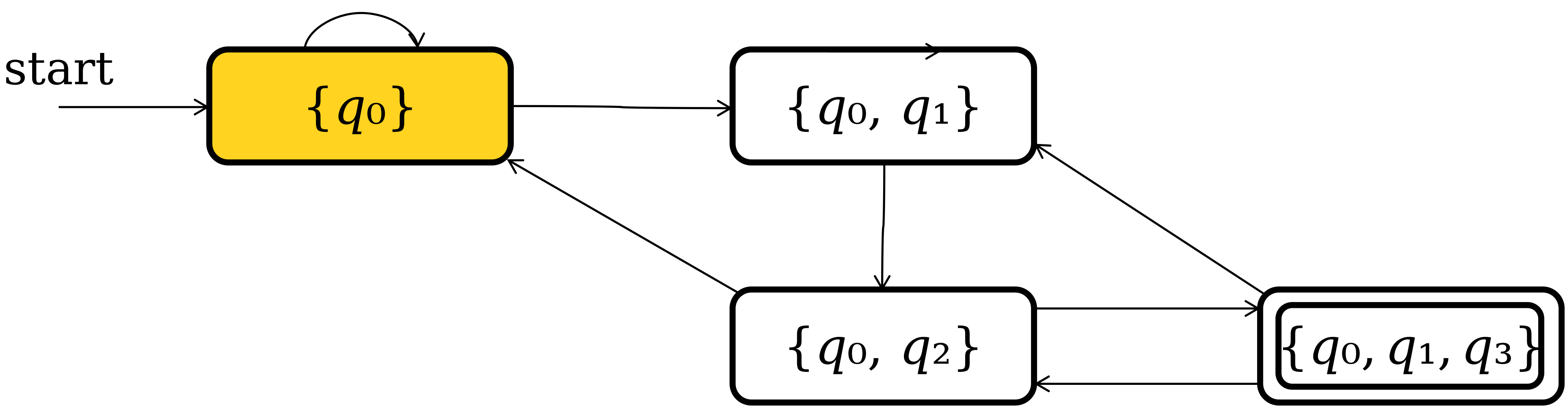


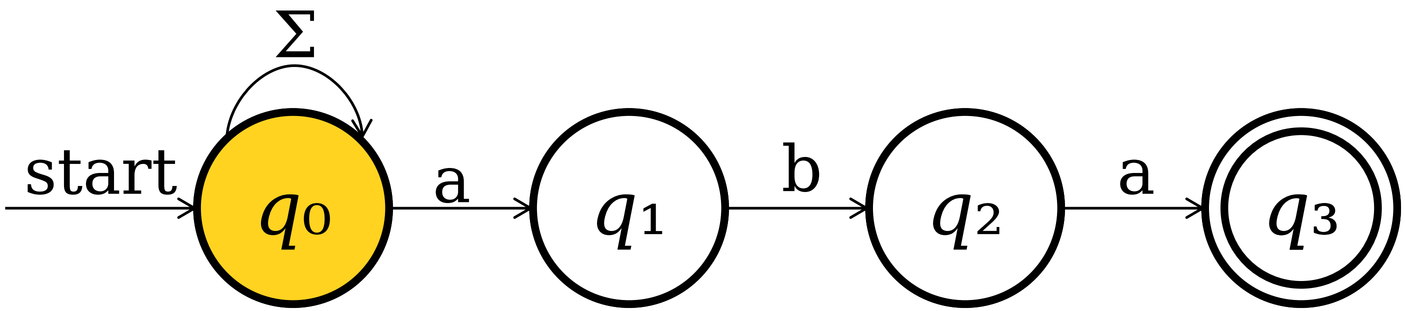
a	b	a	a	b	a



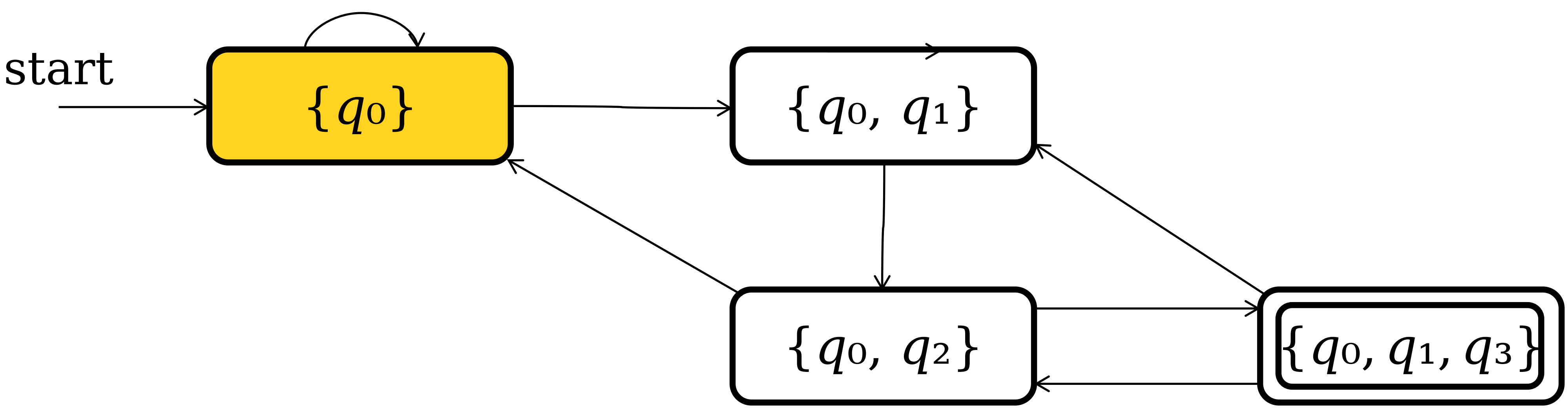


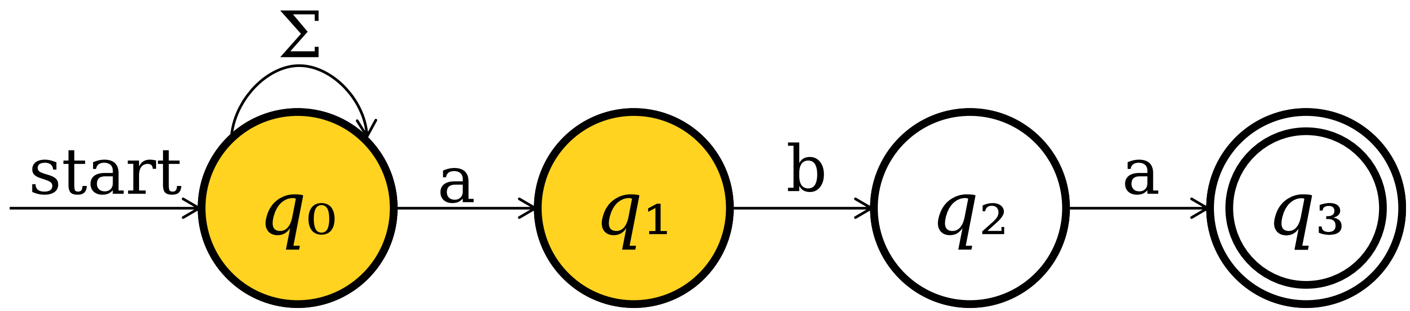
a	b	a	a	b	a

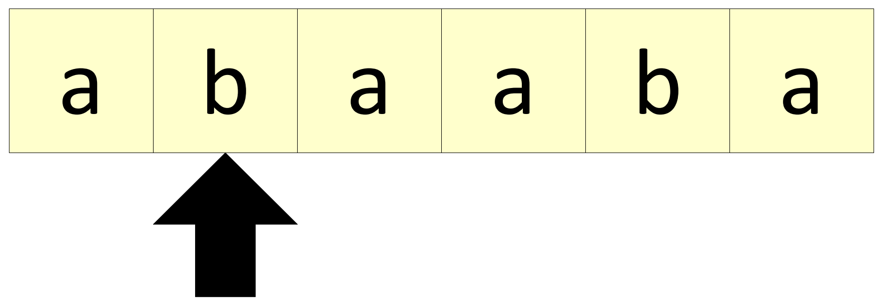


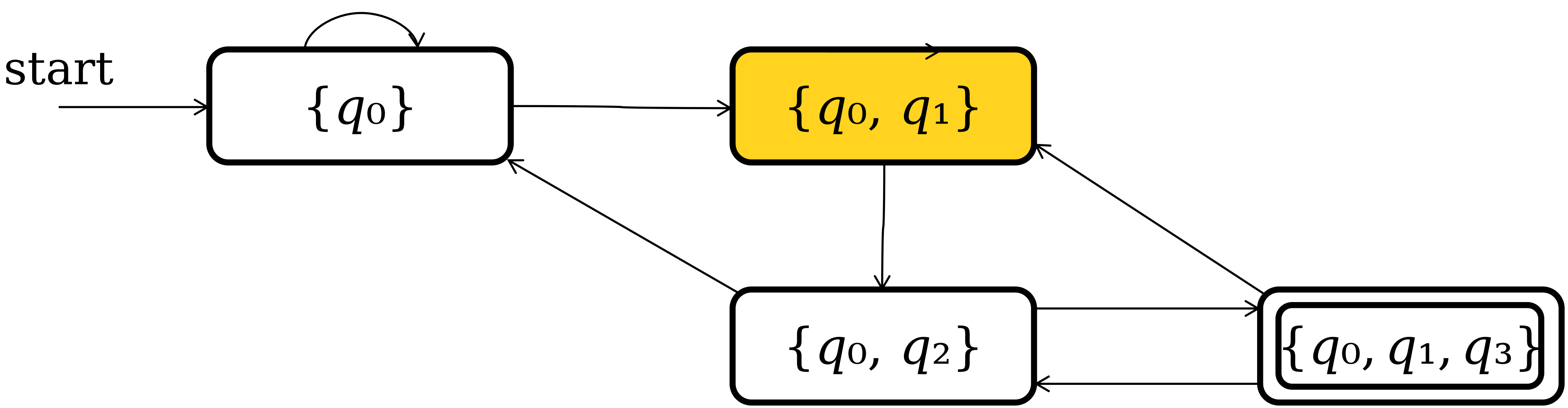


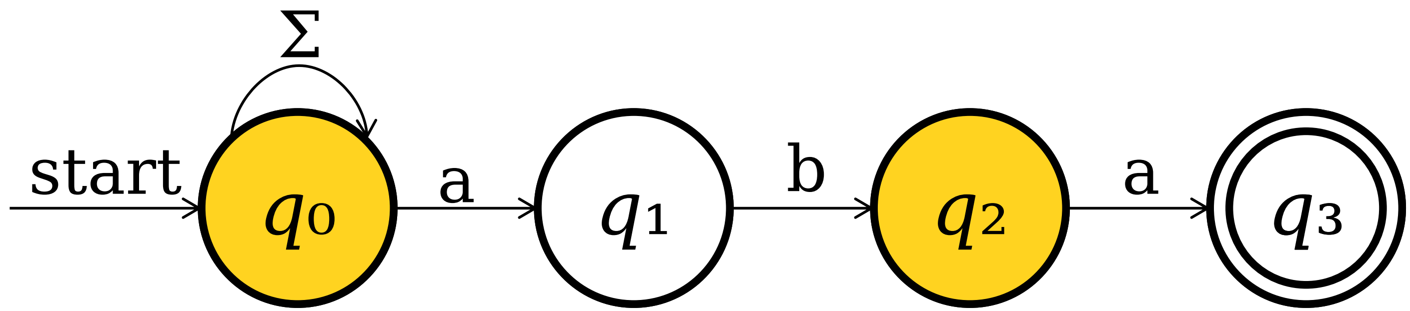
a	b	a	a	b	a

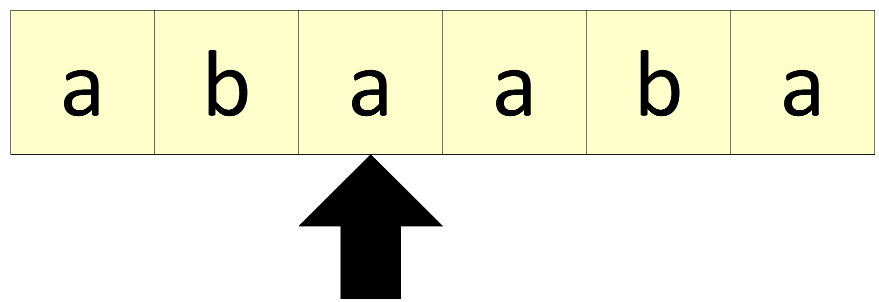


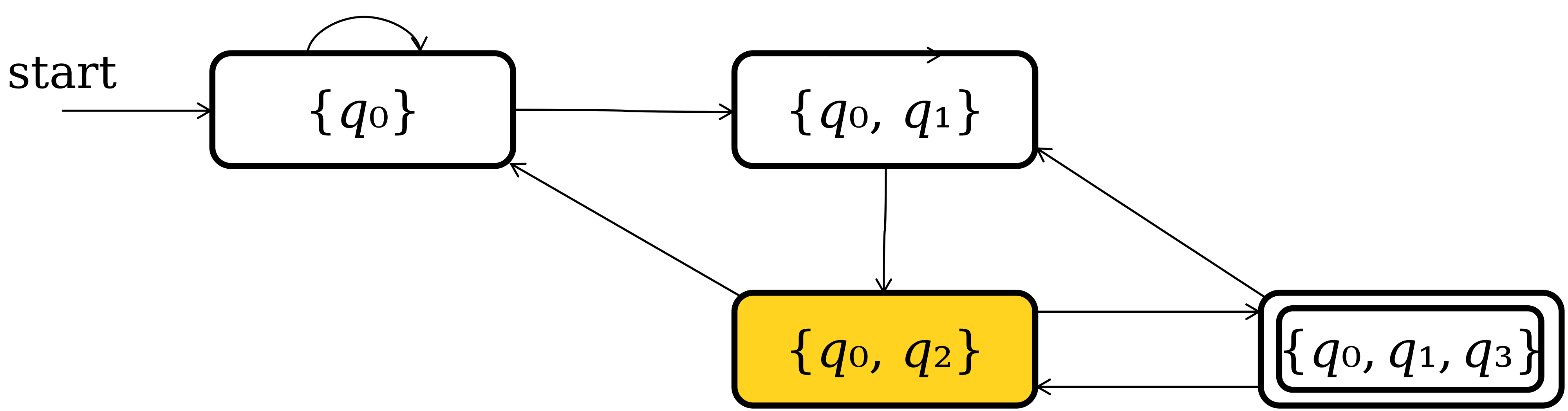

Σ
start
q0
a
q1
b
q2
a
q3

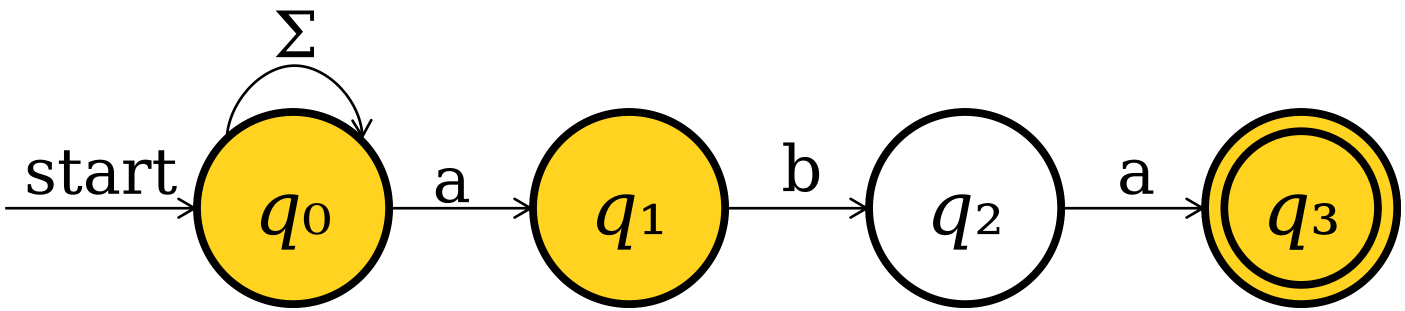

a
b
a
a
b
a


start
{q0}
{q0, q1}
{q0, q2}
{q0, q1, q3}

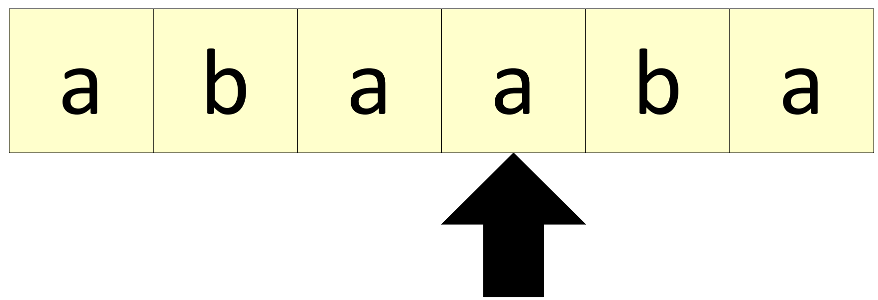

Σ
start
q0
a
q1
b
q2
a
q3


a b a a b a

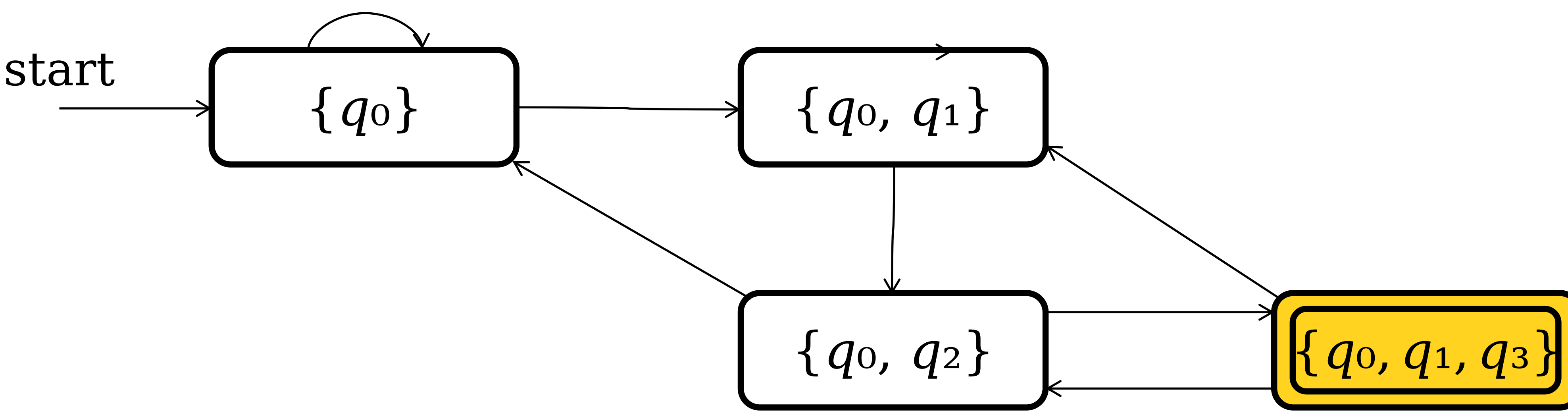

start
{q0}
{q0, q1}
{q0, q2}
{q0, q1, q3}



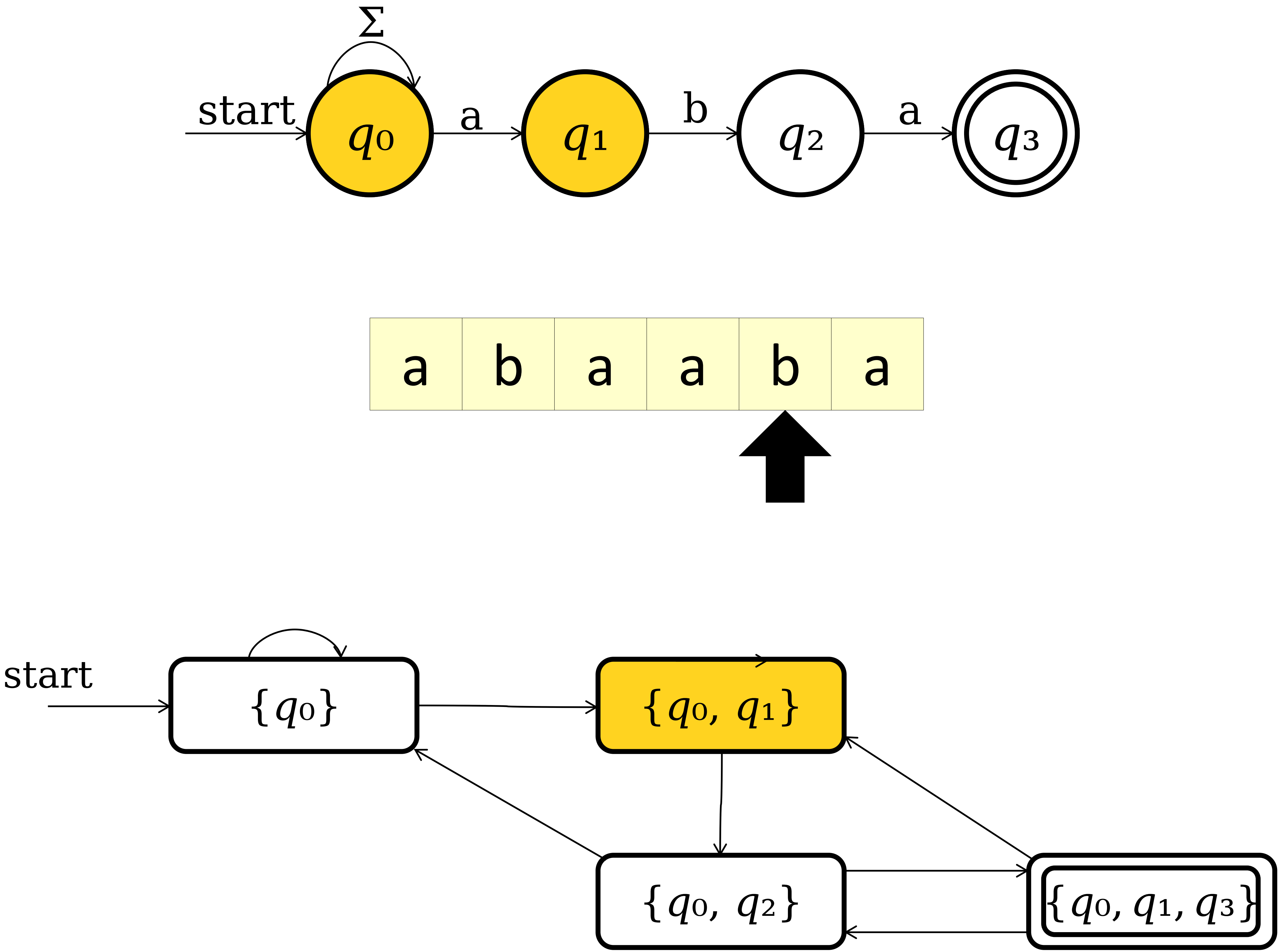
Σ
start
q_0
a
q_1
b
q_2
a
q_3

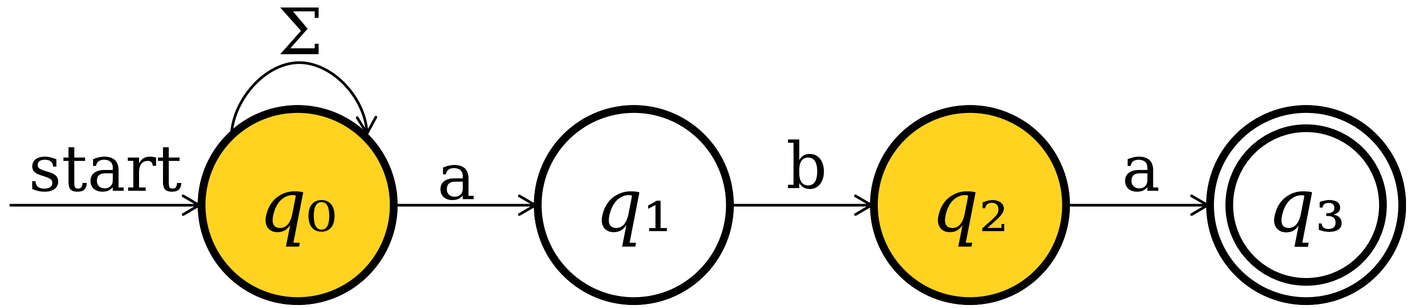


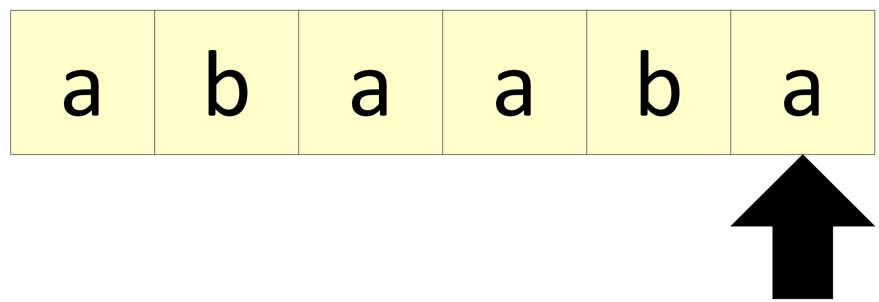
a
b
a
a
b
a

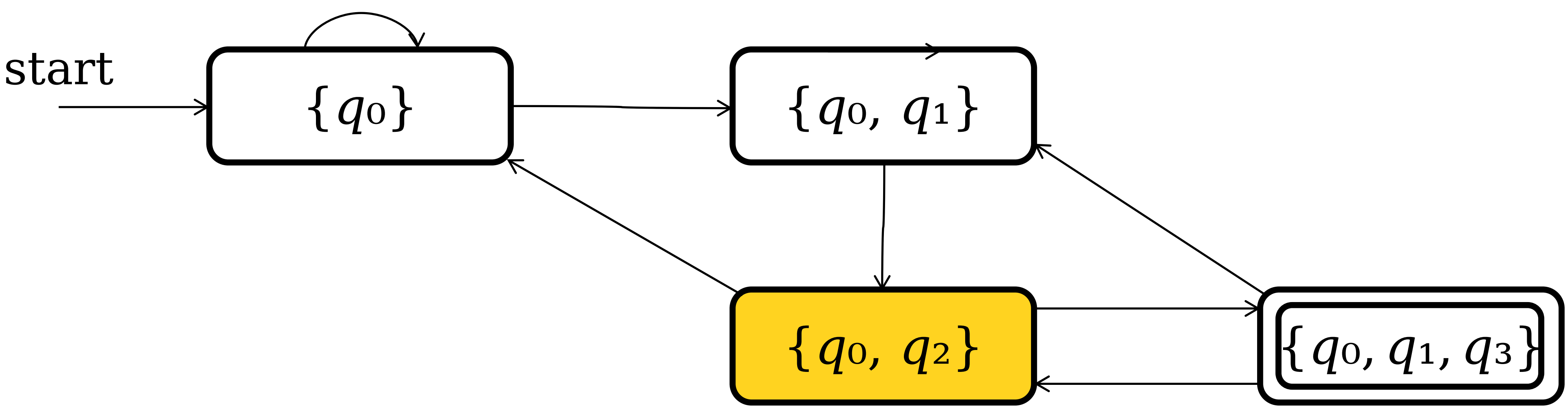


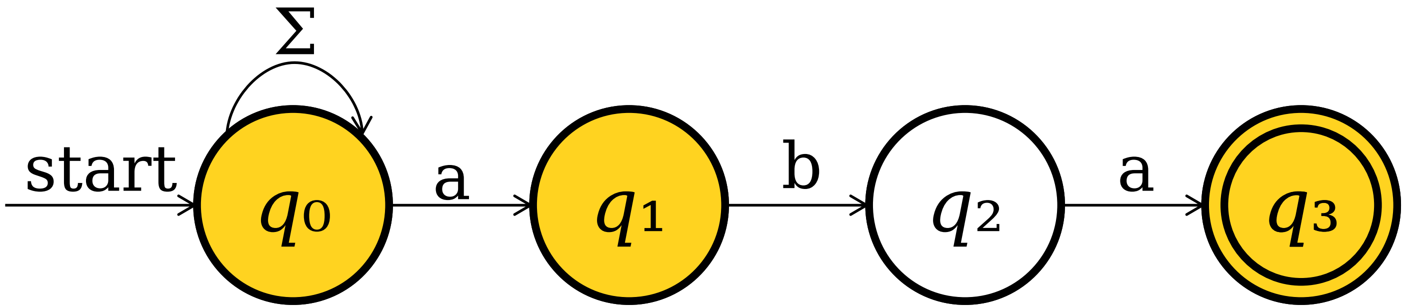
start
$\{q_0\}$
$\{q_0, q_1\}$
$\{q_0, q_2\}$
$\{q_0, q_1, q_3\}$


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

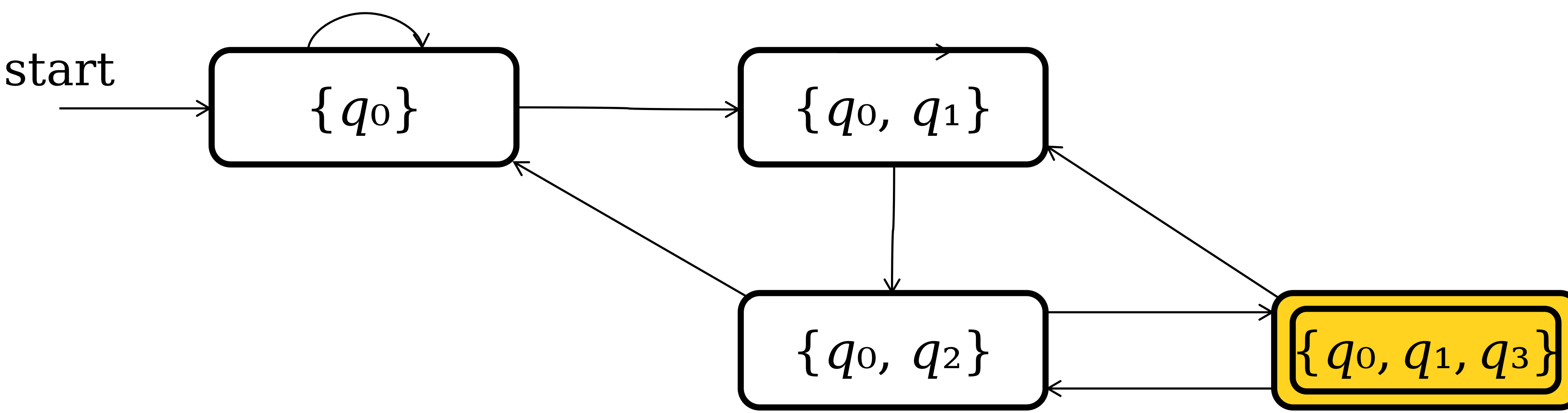

Σ
start
a
b
a
q_0
q_1
q_2
q_3


a b a a b a


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



a	b	a	a	b	a



Some Caveats

Question: what about ε-transitions?

Answer: always include any states you can reach by following ε-transitions.

Question: what happens if there are *no* transitions to follow from a set of states for the character you're trying to fill in?

Answer: then the set of states you can reach is the empty set!

Example included in the appendix of this lecture showing this construction with both of these scenarios.

The Subset Construction

- This construction for transforming an NFA into a DFA is called the ***subset construction*** (or sometimes the ***powerset construction***).
- Each state in the DFA is associated with a set of states in the NFA.
- The start state in the DFA corresponds to the start state of the NFA, plus all states reachable via ε-transitions.
- If a state q in the DFA corresponds to a set of states S in the NFA, then the transition from state q on a character `a` is found as follows:
- Let S' be the set of states in the NFA that can be reached by following a transition labeled `a` from any of the states in S. *(This set may be empty.)*
- Let S'' be the set of states in the NFA reachable from some state in S' by following zero or more epsilon transitions.
- The state q in the DFA transitions on `a` to a DFA state corresponding to the set of states S''.
- ***Read Sipser for a formal account.***

The Subset Construction

For the purposes of this class, we won't ask you to actually perform the subset construction.

Hopefully though, you've been convinced that, in principle, you *could* follow this procedure to turn any NFA into a DFA.

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 iff there is some NFA N such that $\mathscr{L}(N) = L$.

An Important Result

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

Proof Sketch:

An Important Result

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

Proof Sketch: If L is regular, there exists some DFA for it, which we can easily convert into an NFA.

An Important Result

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

Proof Sketch: If L is regular, there exists some DFA for it, which we can easily convert into an NFA.

If L is accepted by some NFA, we can use the subset construction to convert it into a DFA that accepts the same language, so L is regular.

An Important Result

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

Proof Sketch: If L is regular, there exists some DFA for it, which we can easily convert into an NFA.

If L is accepted by some NFA, we can use the subset construction to convert it into a DFA that accepts the same language, so L is regular. ■

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:

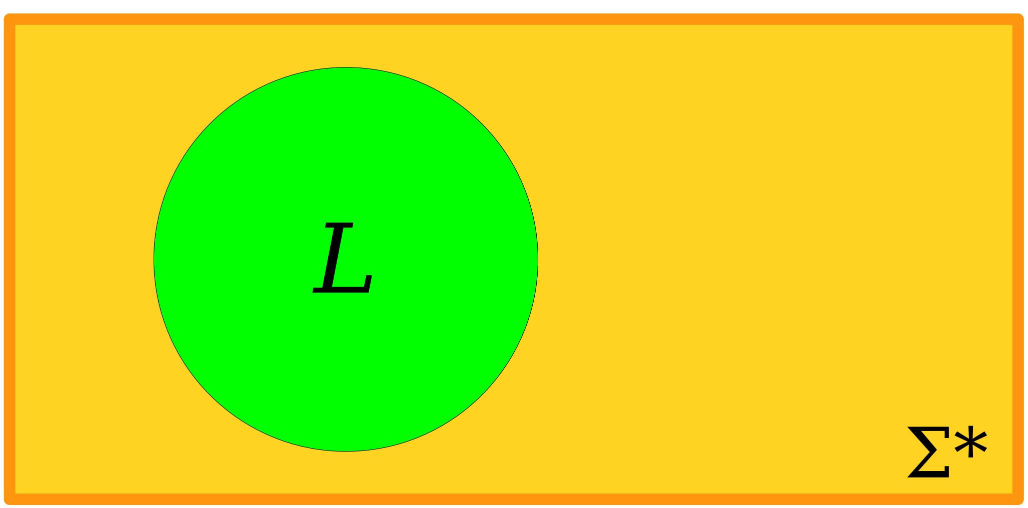
$$\overline{L} = \Sigma^* - L$$

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:

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

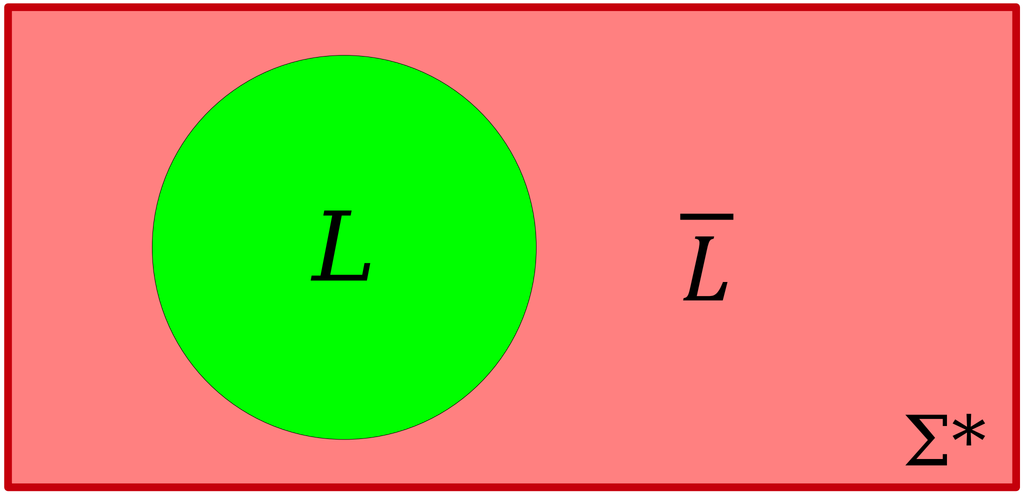


The Complement of a Language

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:

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



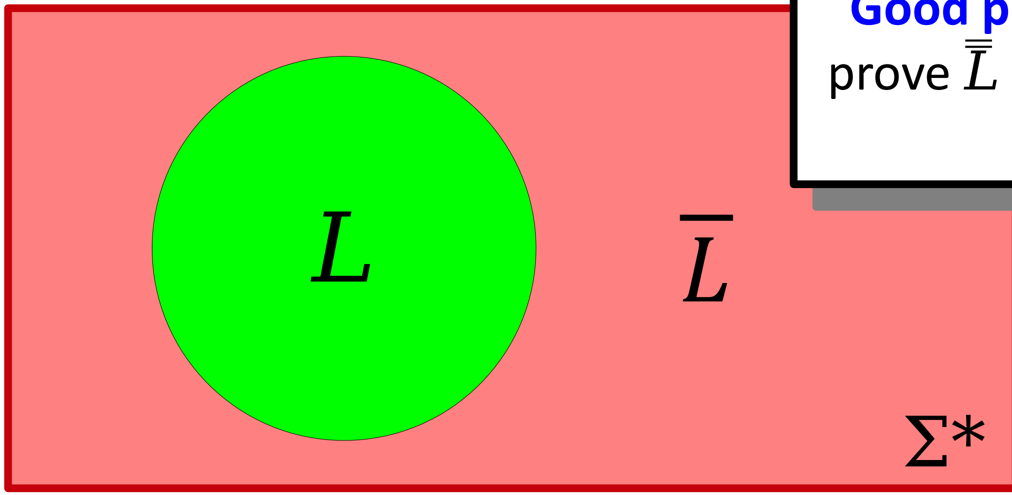
The Complement of a Language

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:

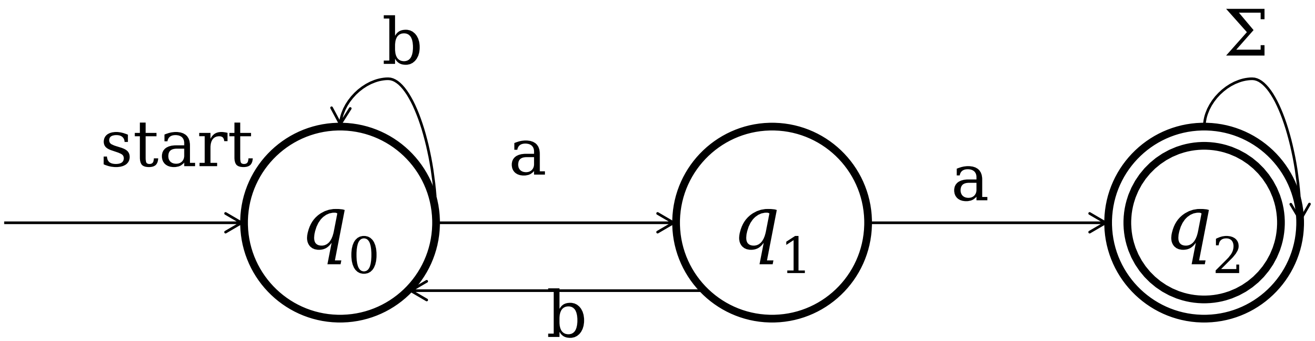
$$\overline{L} = \Sigma^* - L$$

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

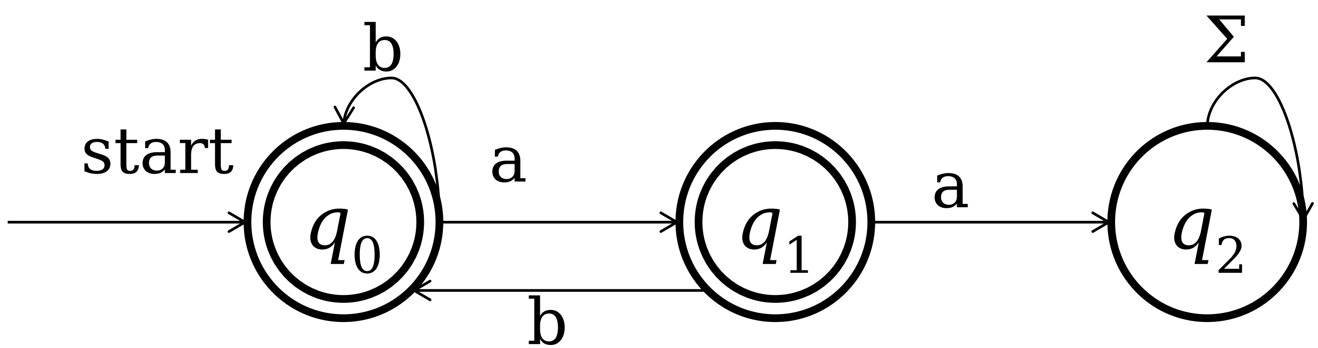


Complementing Regular Languages

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

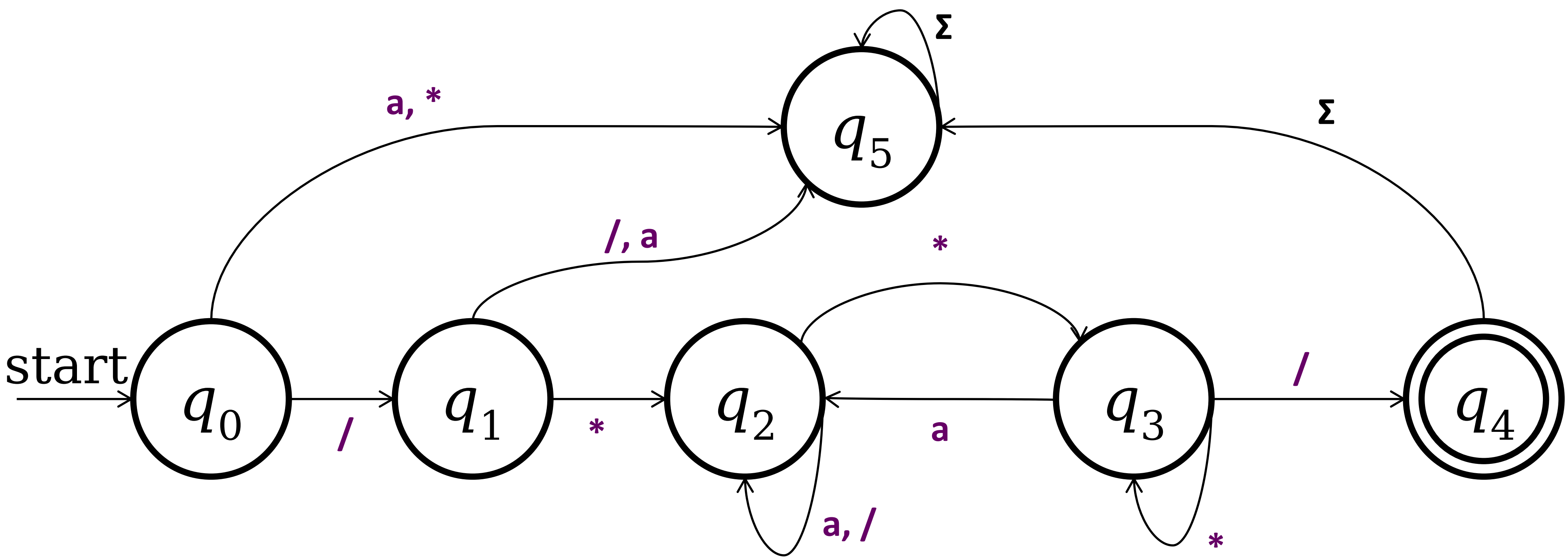


$\overline{L} = \{ w \in \{$**a**, **b**$\}^* \mid w$ ***does not*** contain **aa** as a substring $\}$



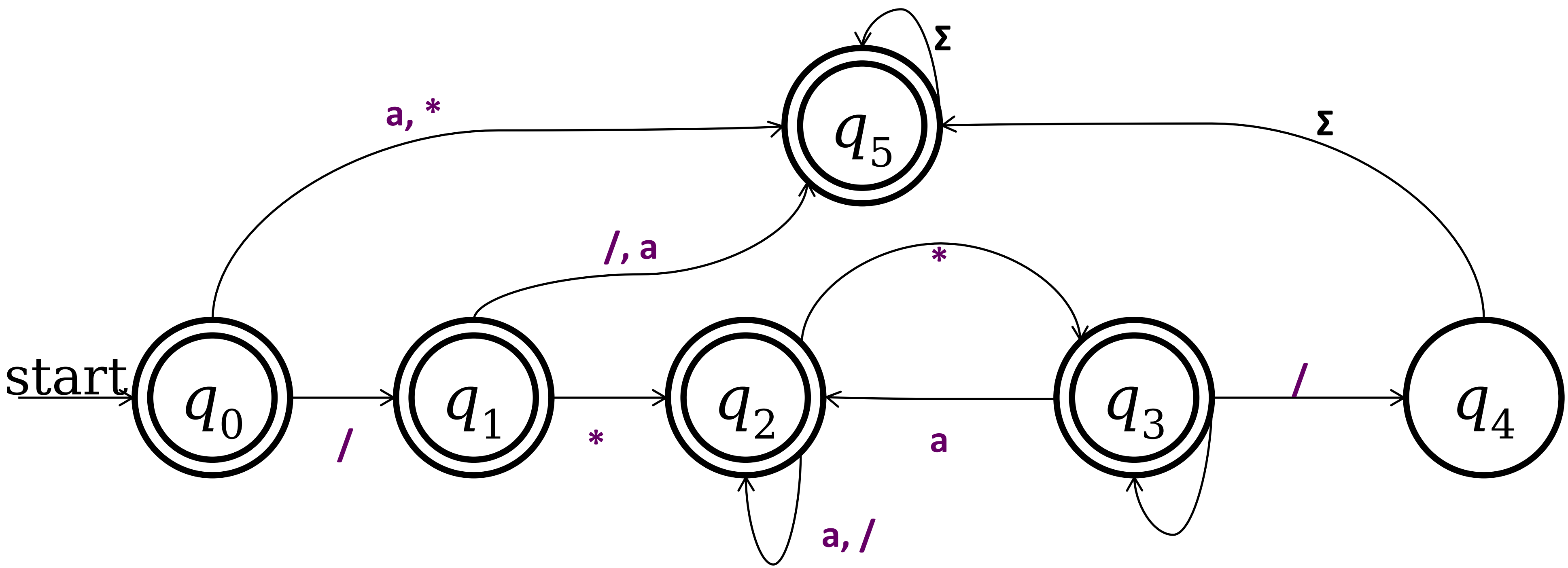
Complementing Regular Languages

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



Complementing Regular Languages

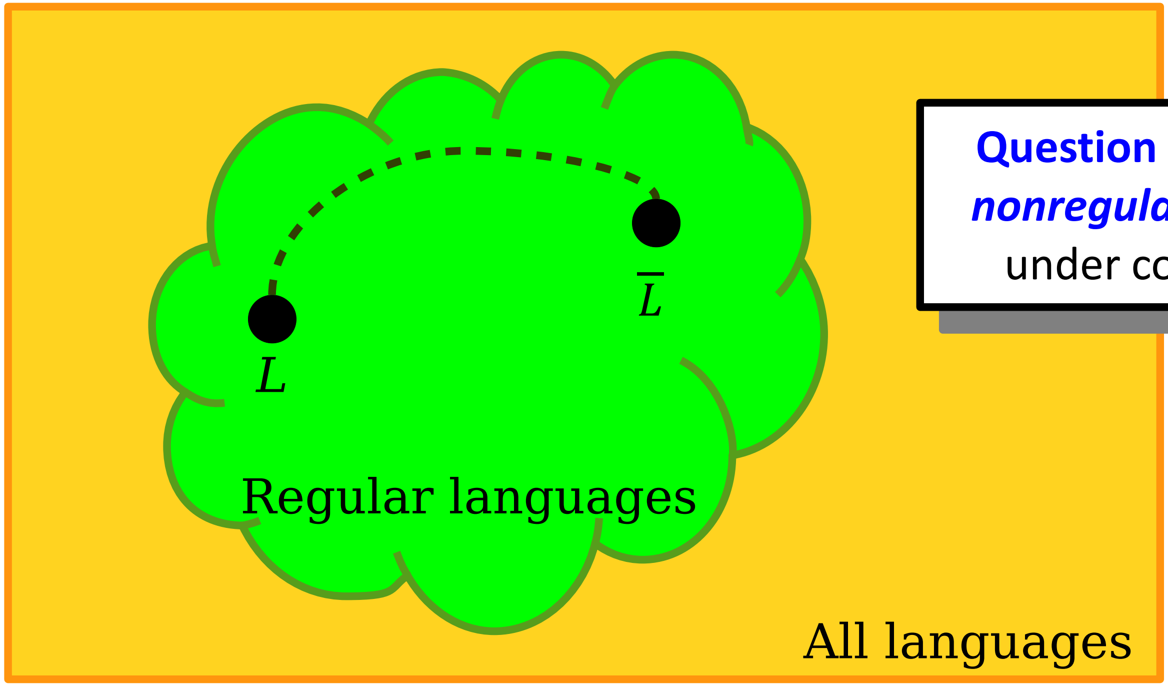
$L = \{\ w \in \{$**a**, *****, **/**$\}^* \mid w$ ***doesn't*** represent a C-style comment $\}$



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

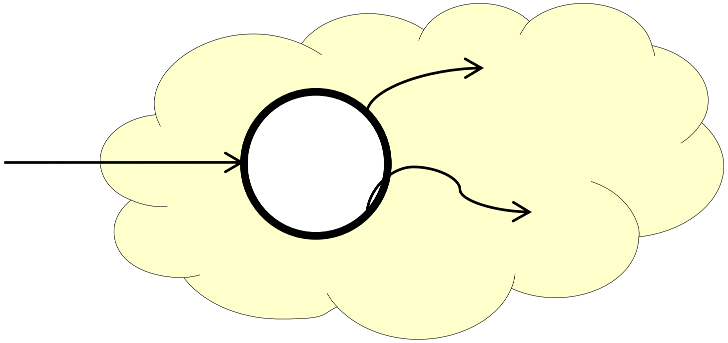
If L_1 and L_2 are languages over the alphabet Σ, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.

If L_1 and L_2 are regular languages, is $L_1 \cup L_2$?

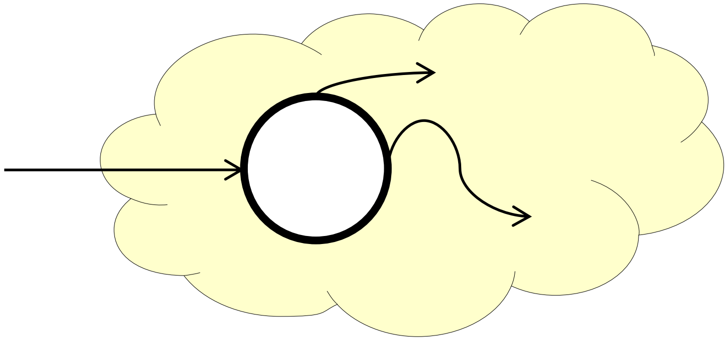
The Union of Two Languages

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 $L_1 \cup L_2$?



Machine for L_1

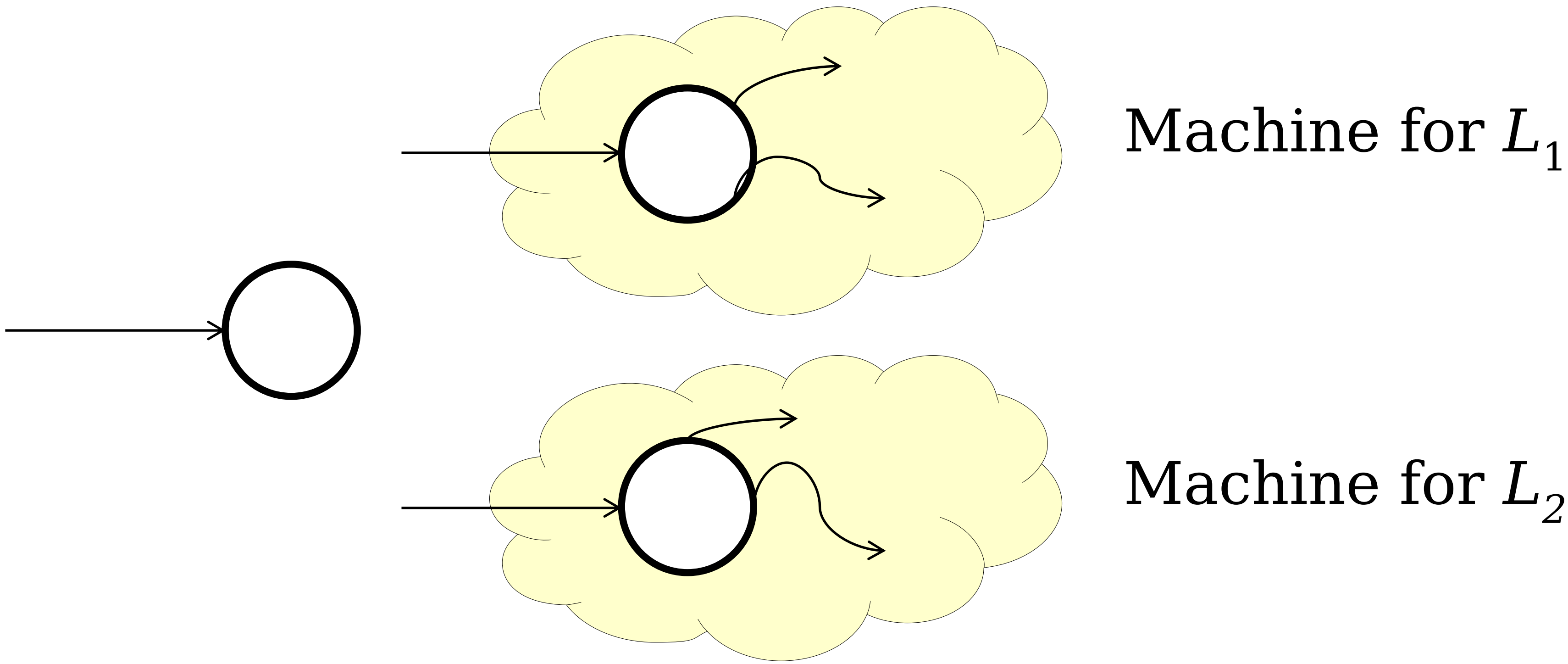


Machine for L_2

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.

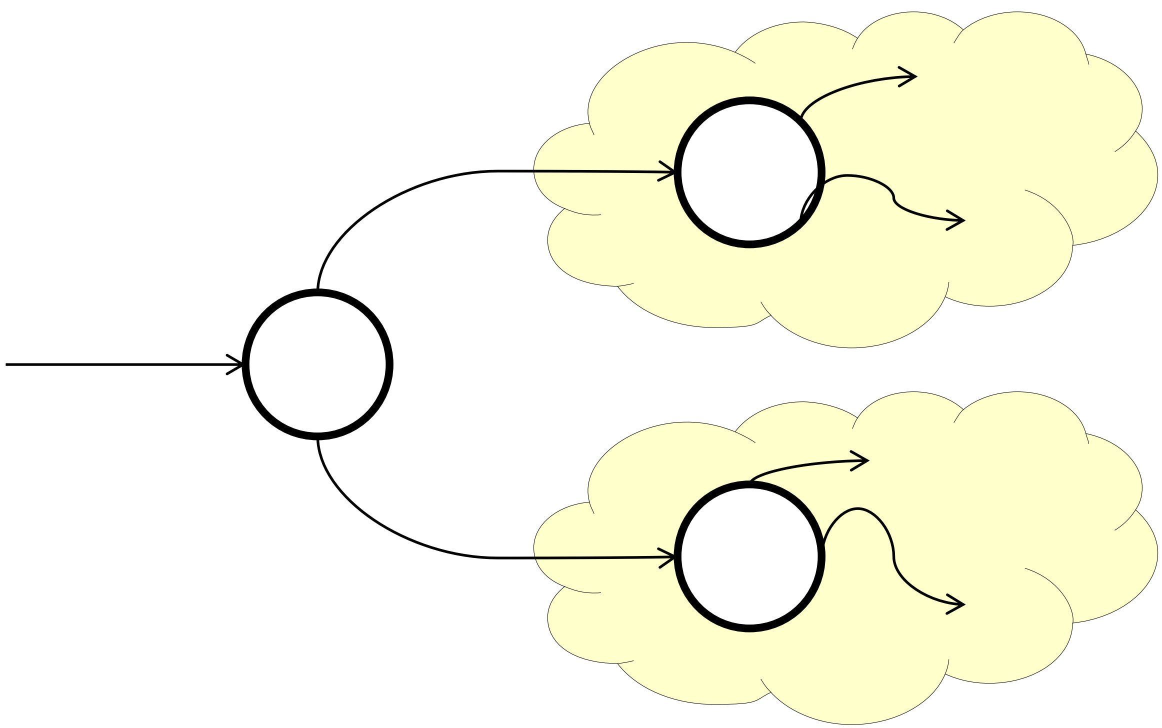
If L_1 and L_2 are regular languages, is $L_1 \cup L_2$?



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

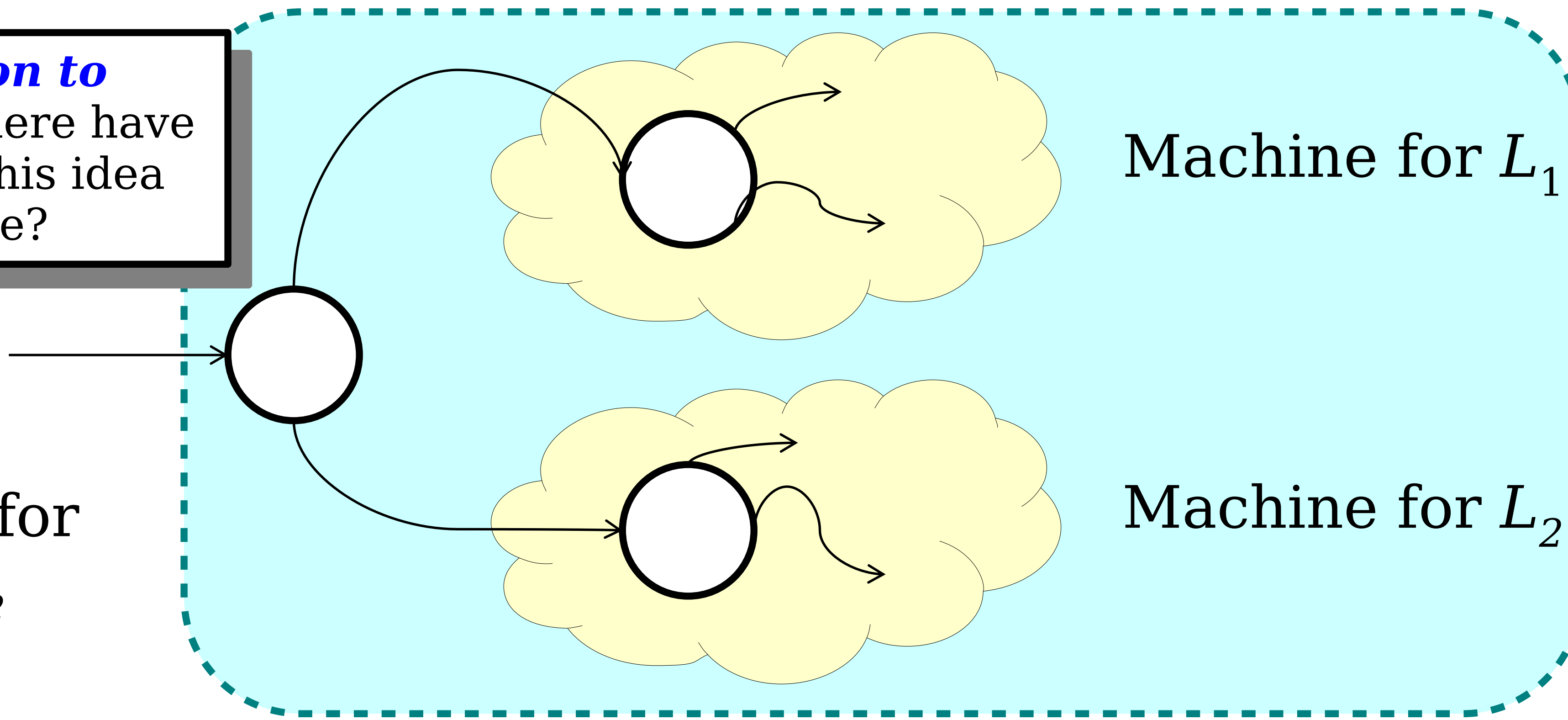
Machine for L_2

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 $L_1 \cup L_2$?

Question to ponder: where have you seen this idea before?



Machine for $L_1 \cup L_2$

The Intersection of Two Languages

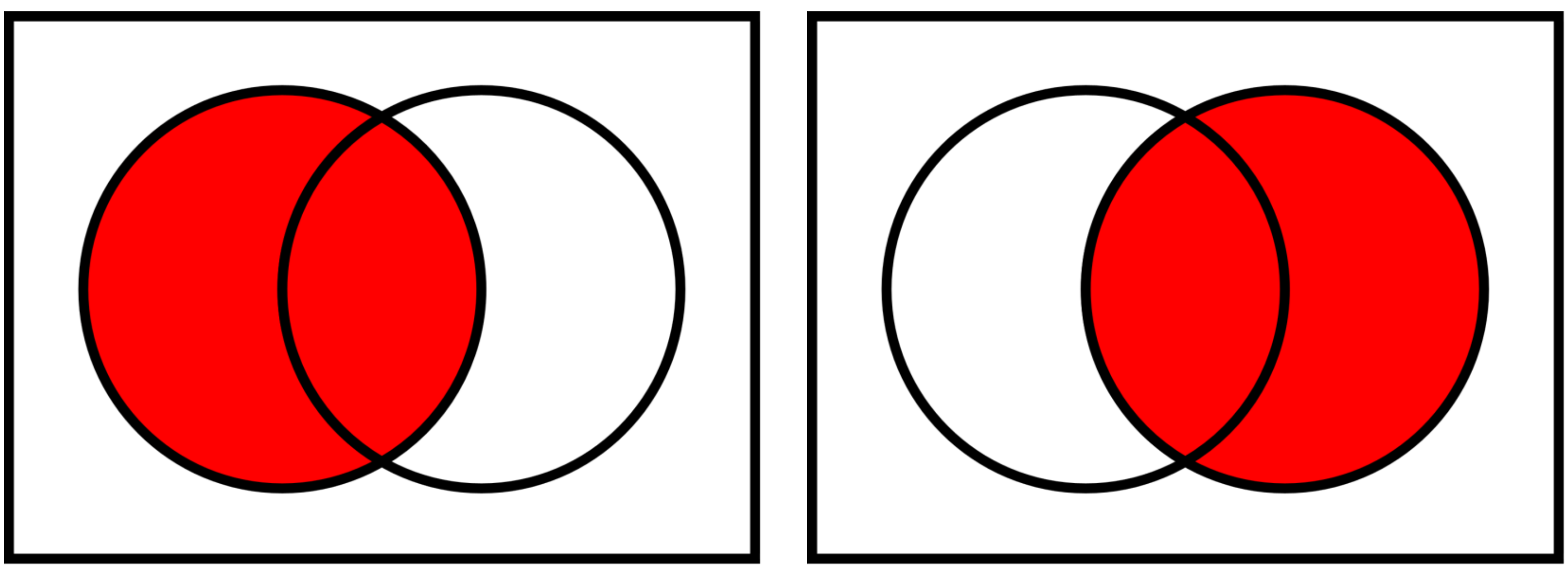
If L_1 and L_2 are languages over Σ, then $L_1 \cap L_2$ is the language of strings in both L_1 and L_2.

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?

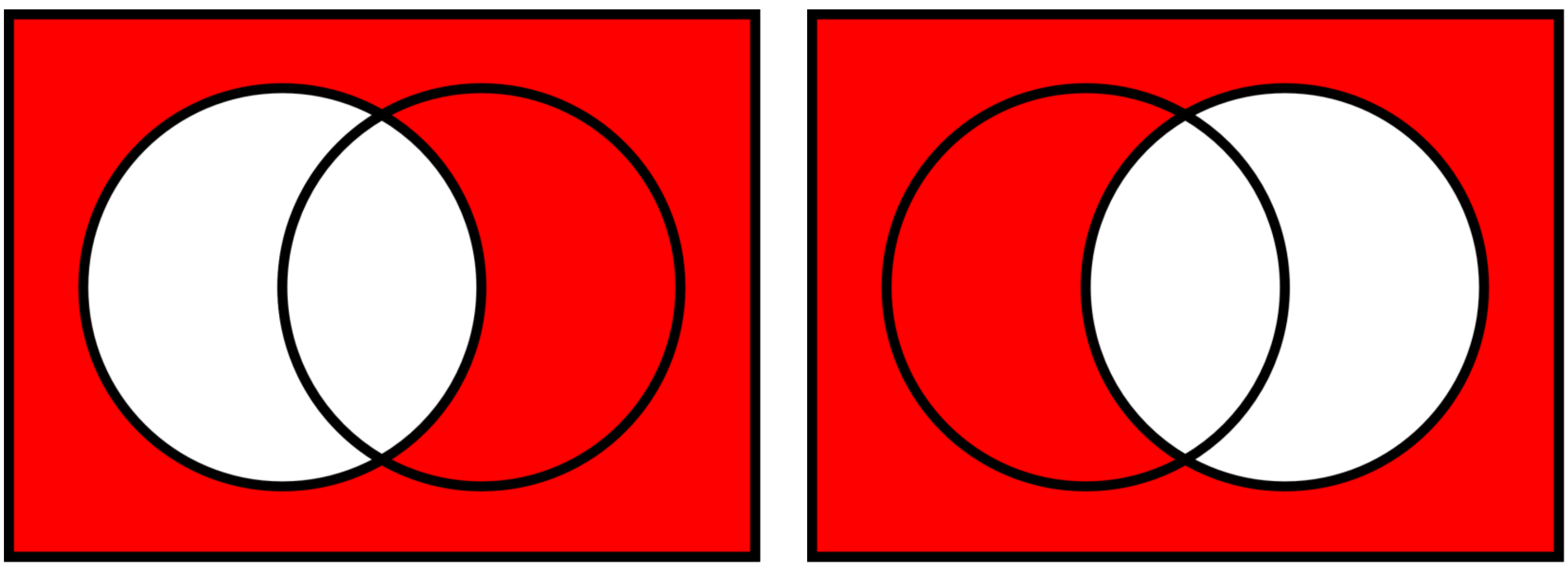


L_1 L_2

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?

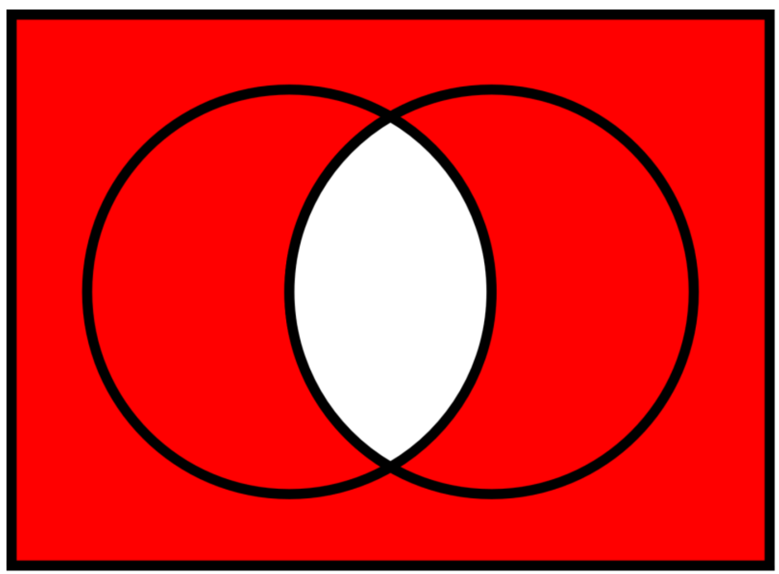


L_1 L_2

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?

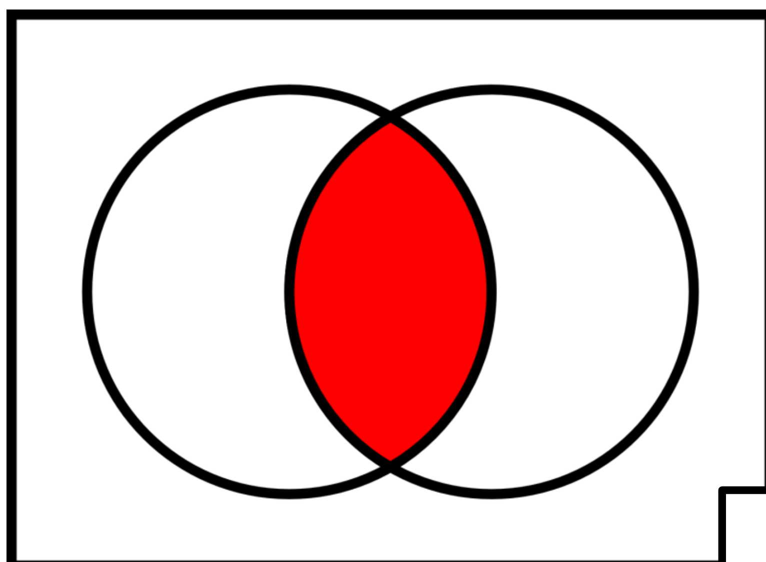


$$L_1 \cup L_2$$

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?



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

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**.

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

$$\boldsymbol{L_1 L_2 = \{ wx \in \Sigma^* \mid w \in L_1 \land x \in L_2 \}}$$

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

$$\boldsymbol{L_1 L_2 = \{ wx \in \Sigma^* \mid w \in L_1 \land x \in L_2 \}}$$

Two views of $L_1 L_2$:

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 $y \in L_2$?

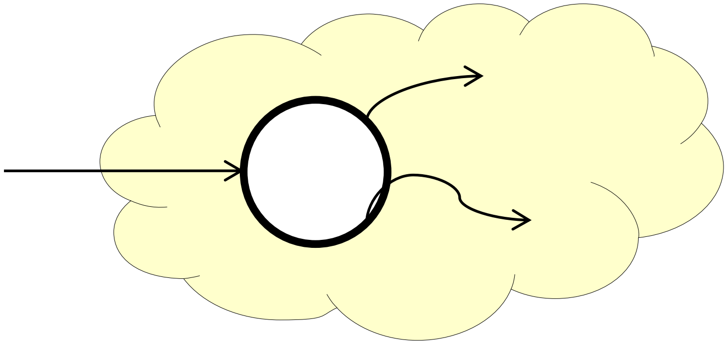
Idea:

Concatenating Regular Languages

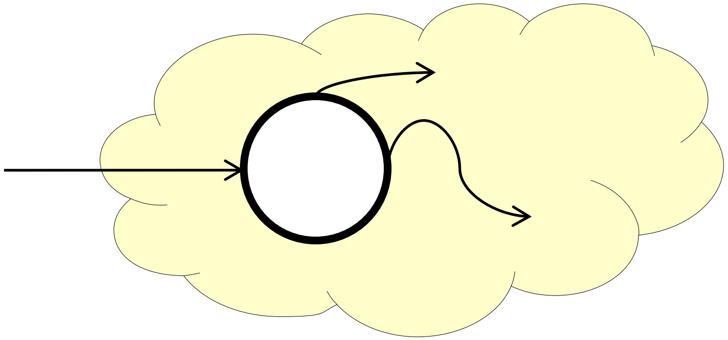
If L_1 and L_2 are regular languages, is L_1L_2?

Intuition – can we split a string w into two strings xy such that $x \in L_1$ and $y \in L_2$?

Idea:



Machine for L_1



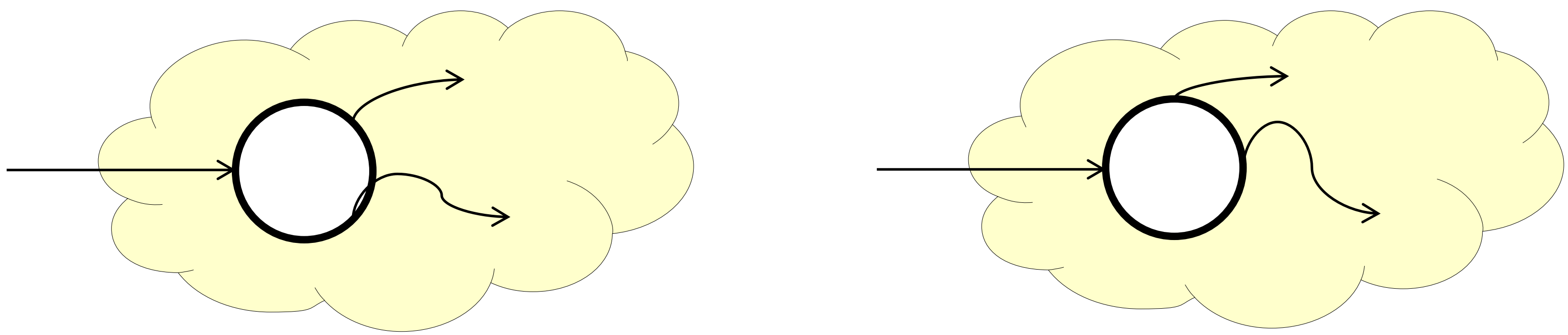
Machine for L_2

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 $y \in L_2$?

Idea:



Machine for L_1

Machine for L_2

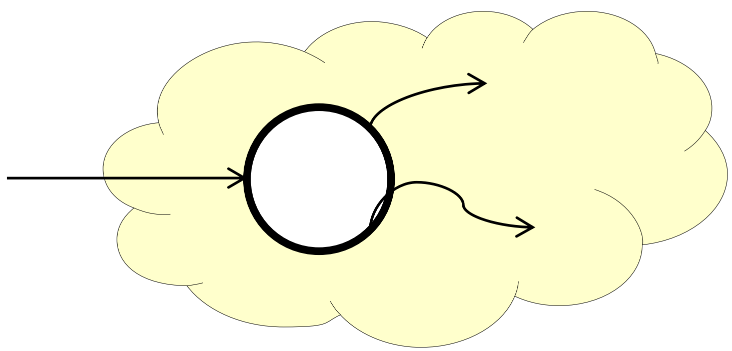
b	**o**	**o**	**k**	**k**	**e**	**e**	**p**	**e**	**r**

Concatenating Regular Languages

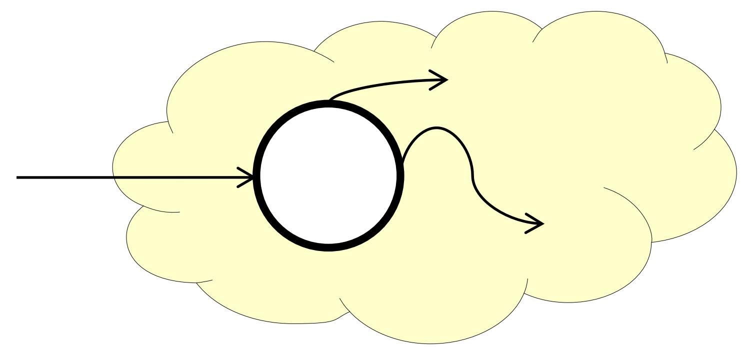
If L_1 and L_2 are regular languages, is L_1L_2?

Intuition – can we split a string w into two strings xy such that $x \in L_1$ and $y \in L_2$?

Idea:



Machine for L_1

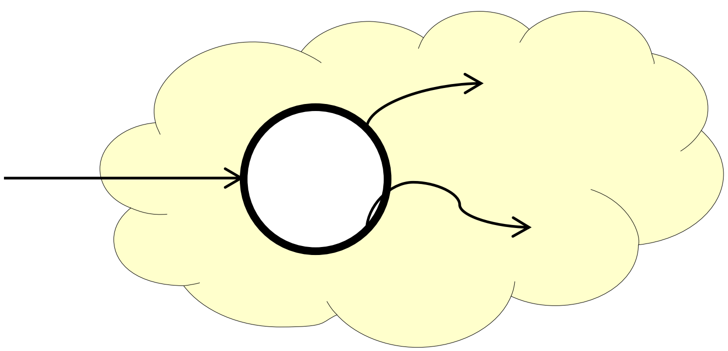


Machine for L_2

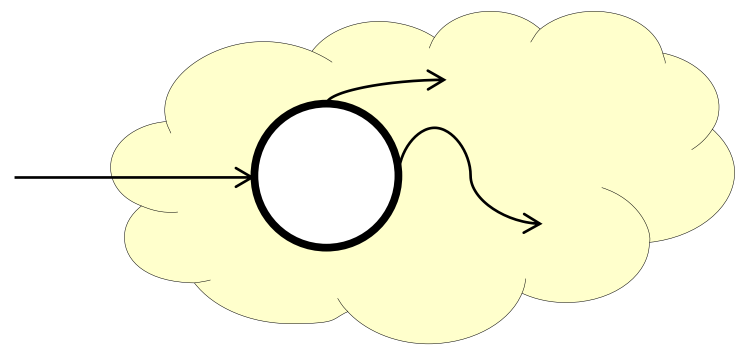
b	**o**	**o**	**k**	**k**	**e**	**e**	**p**	**e**	**r**

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***:



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.

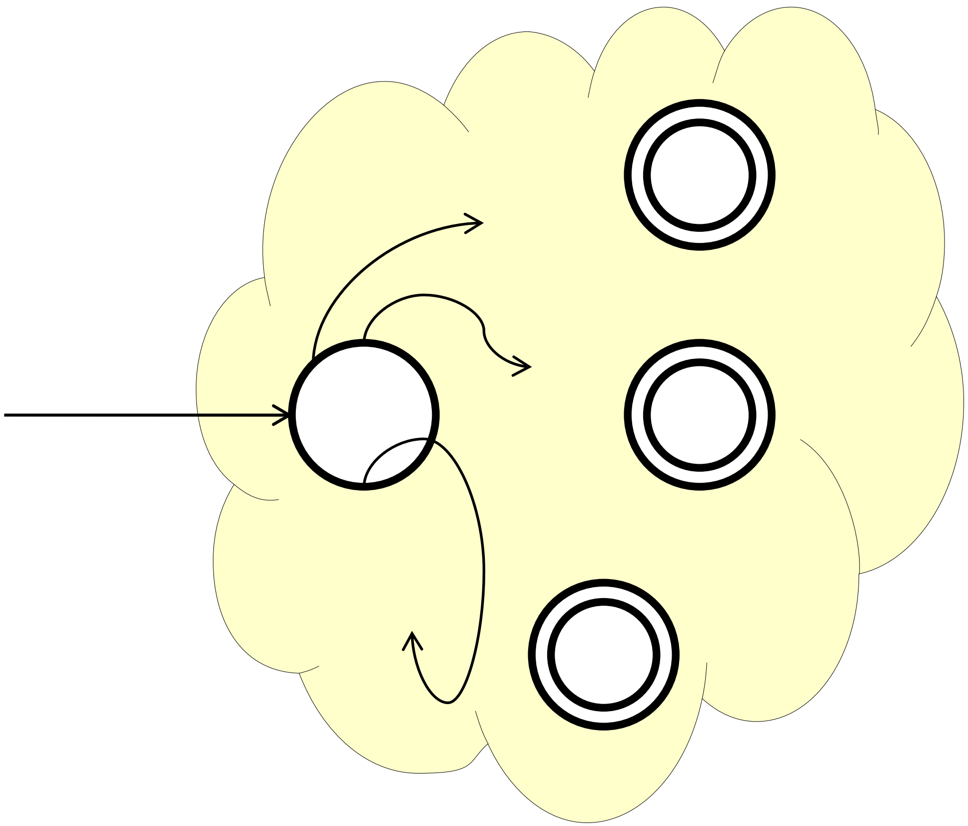
Whenever it reaches an accepting state, optionally hand the rest of w to a DFA/NFA for L_2.

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.

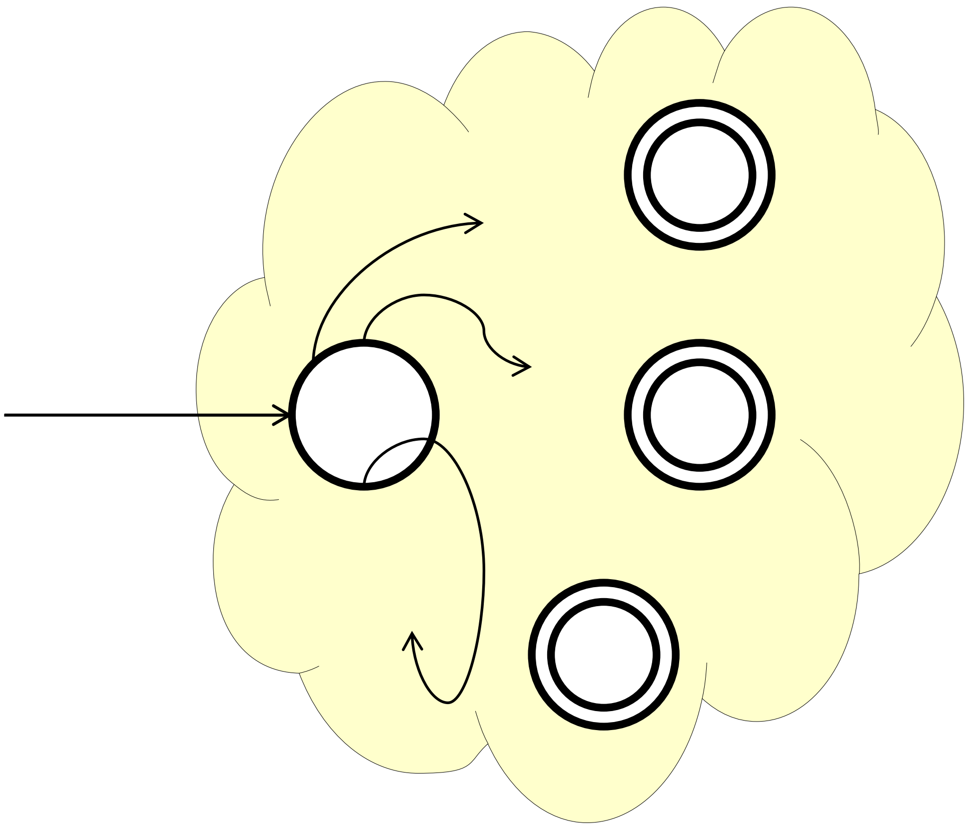
Concatenating Regular Languages

Concatenating Regular Languages

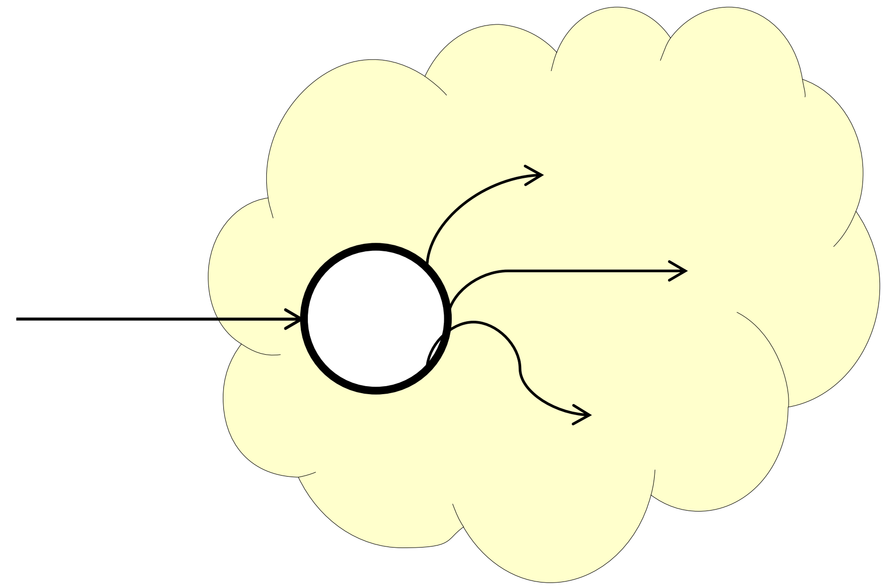


Machine for L_1

Concatenating Regular Languages

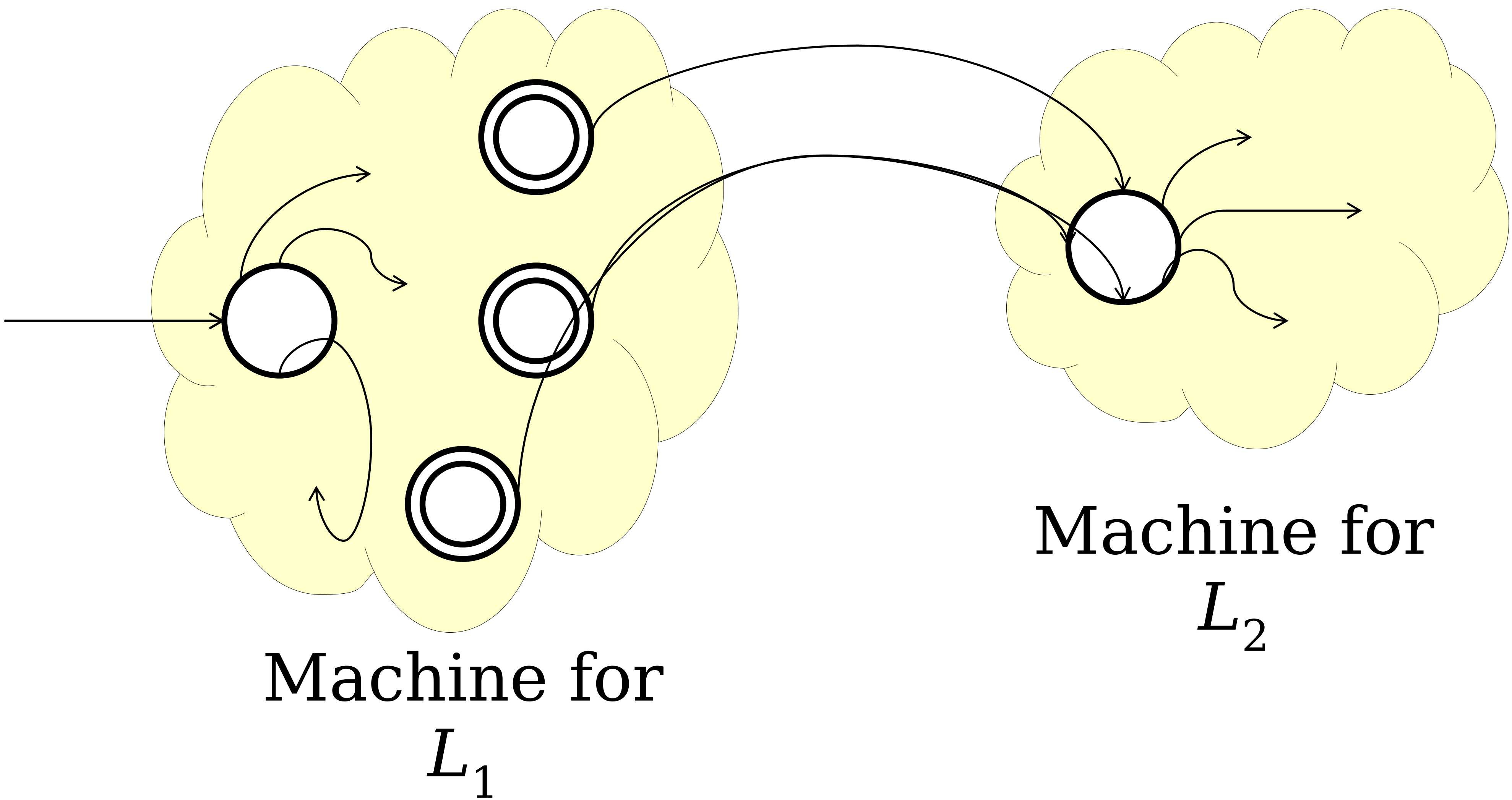


Machine for L_1

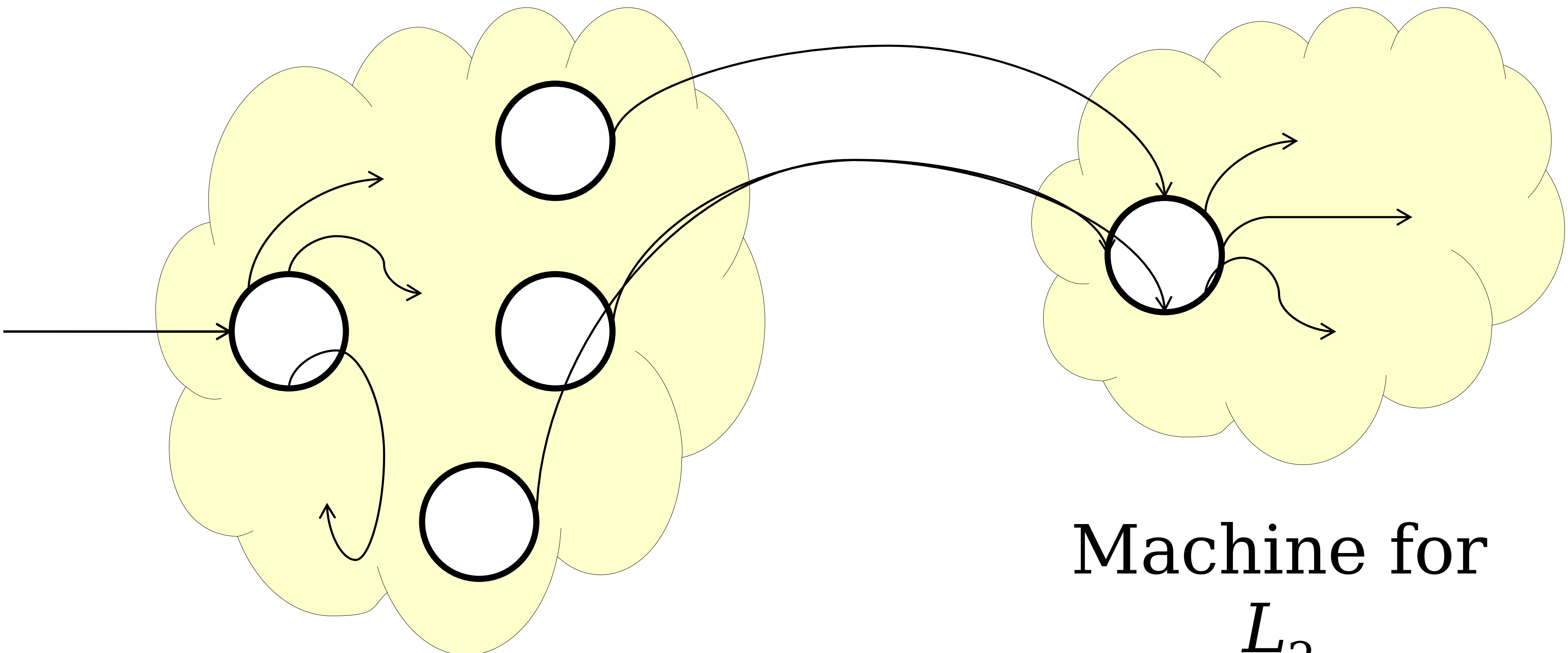


Machine for L_2

Concatenating Regular Languages



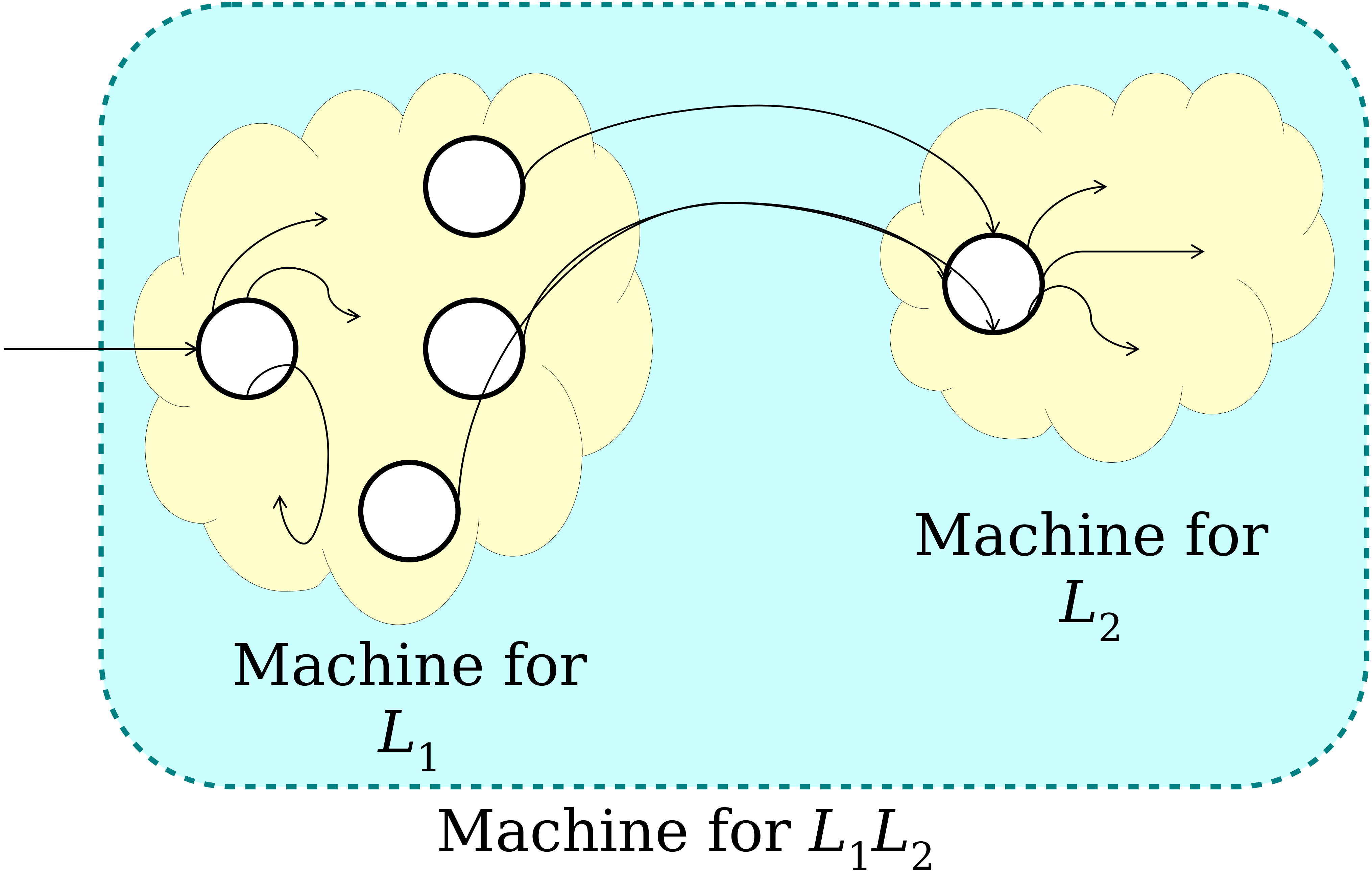
Concatenating Regular Languages



Machine for L_1

Machine for L_2

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:

$L^0 = \{\varepsilon\}$

The set containing just the empty string.

Idea: Any string formed by concatenating zero strings together is the empty string.

$L^{n+1} = LL^n$

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 $\emptyset^0$?

The Kleene Closure

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

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

Mathematically:

$$\boldsymbol{w \in L^* \quad \textbf{iff} \quad \exists n \in \mathbb{N}.\ w \in L^n}$$

Intuitively, all possible ways of concatenating zero or more strings in L together, possibly with repetition.

Question to ponder: What is $\emptyset^*$?

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.

Reasoning about Infinity

If L is regular, is L^* necessarily regular?

⚠ A Bad Line of Reasoning: ⚠

$L^0 = \{ \varepsilon \}$ is regular.

$L^1 = L$ is regular.

$L^2 = LL$ is regular

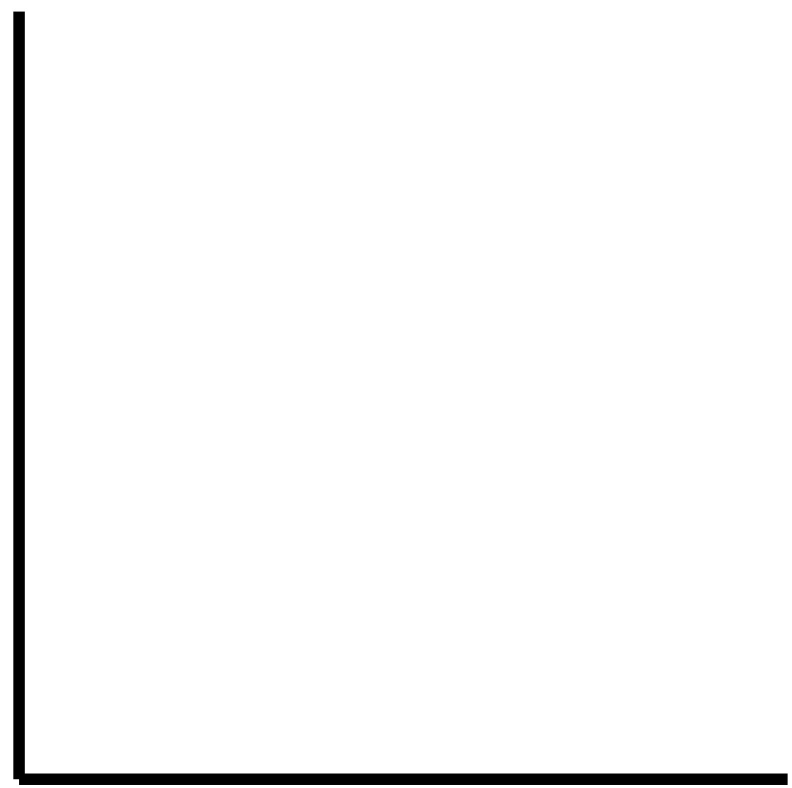
$L^3 = L(LL)$ is regular

…

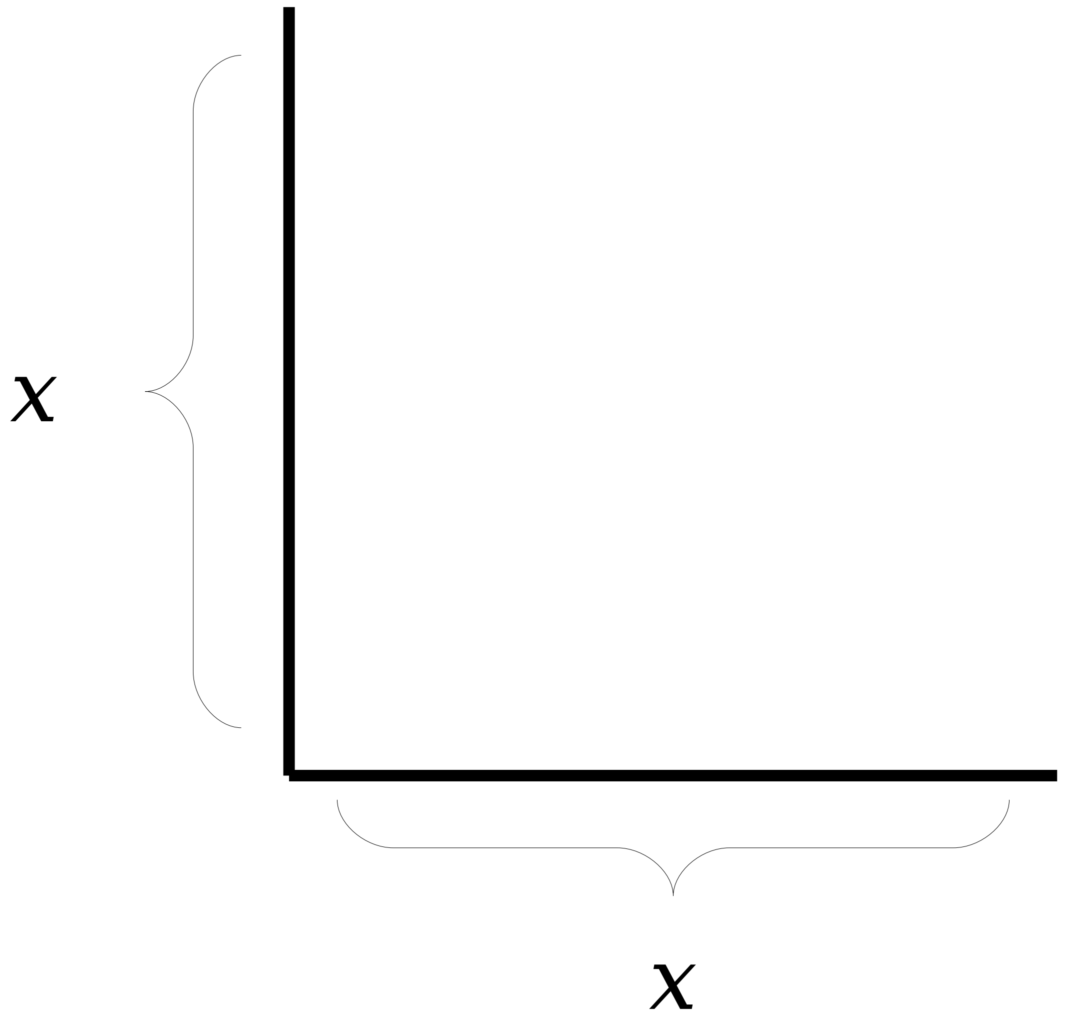
Regular languages are closed under union.

So the union of all these languages is regular.

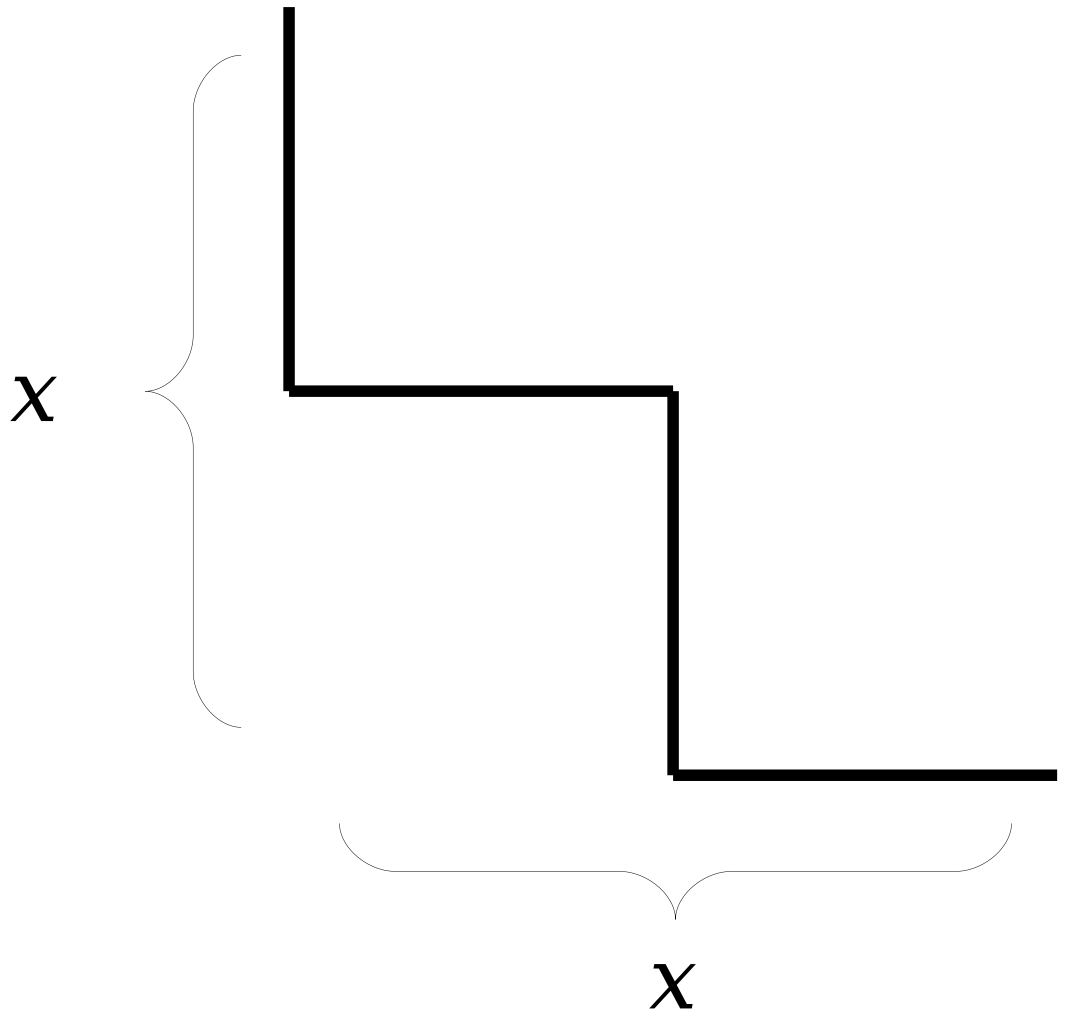
Reasoning about Infinity



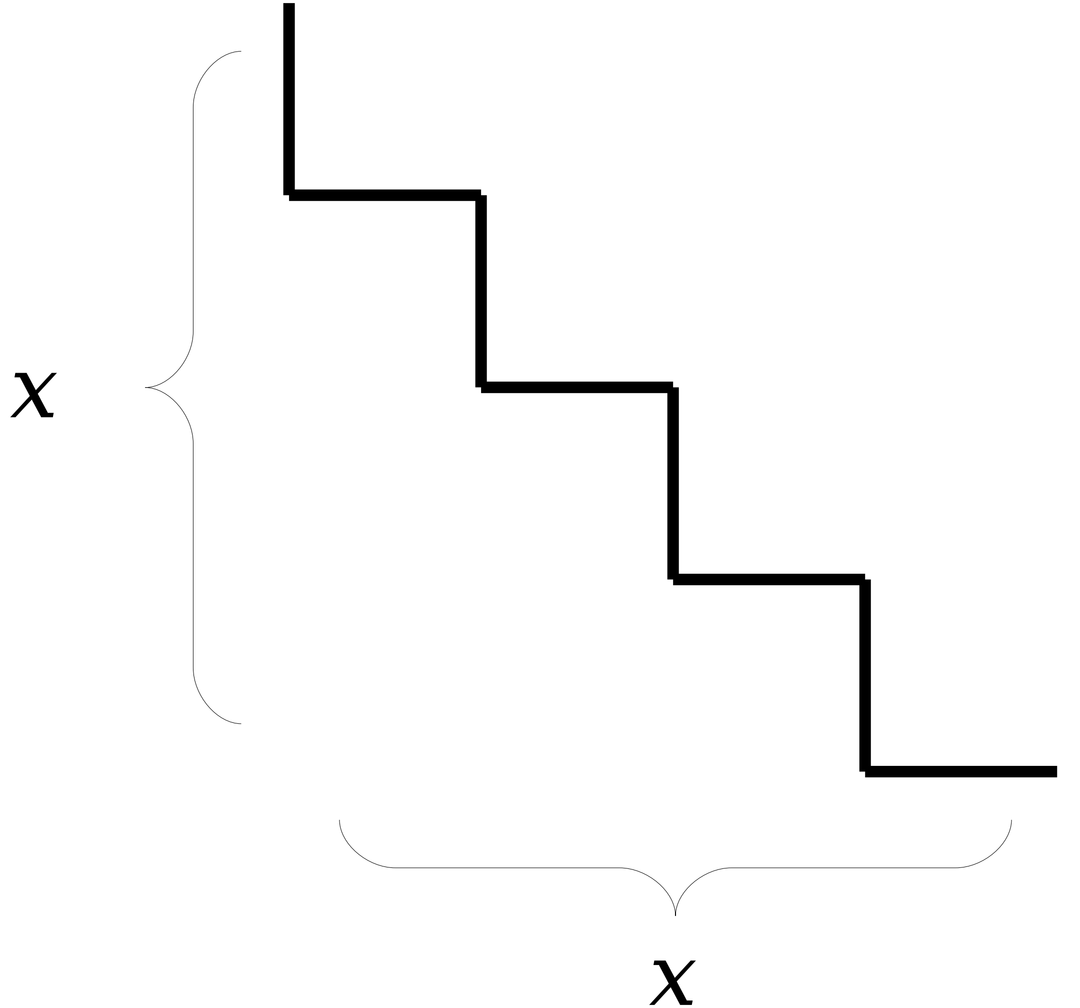
Reasoning about Infinity



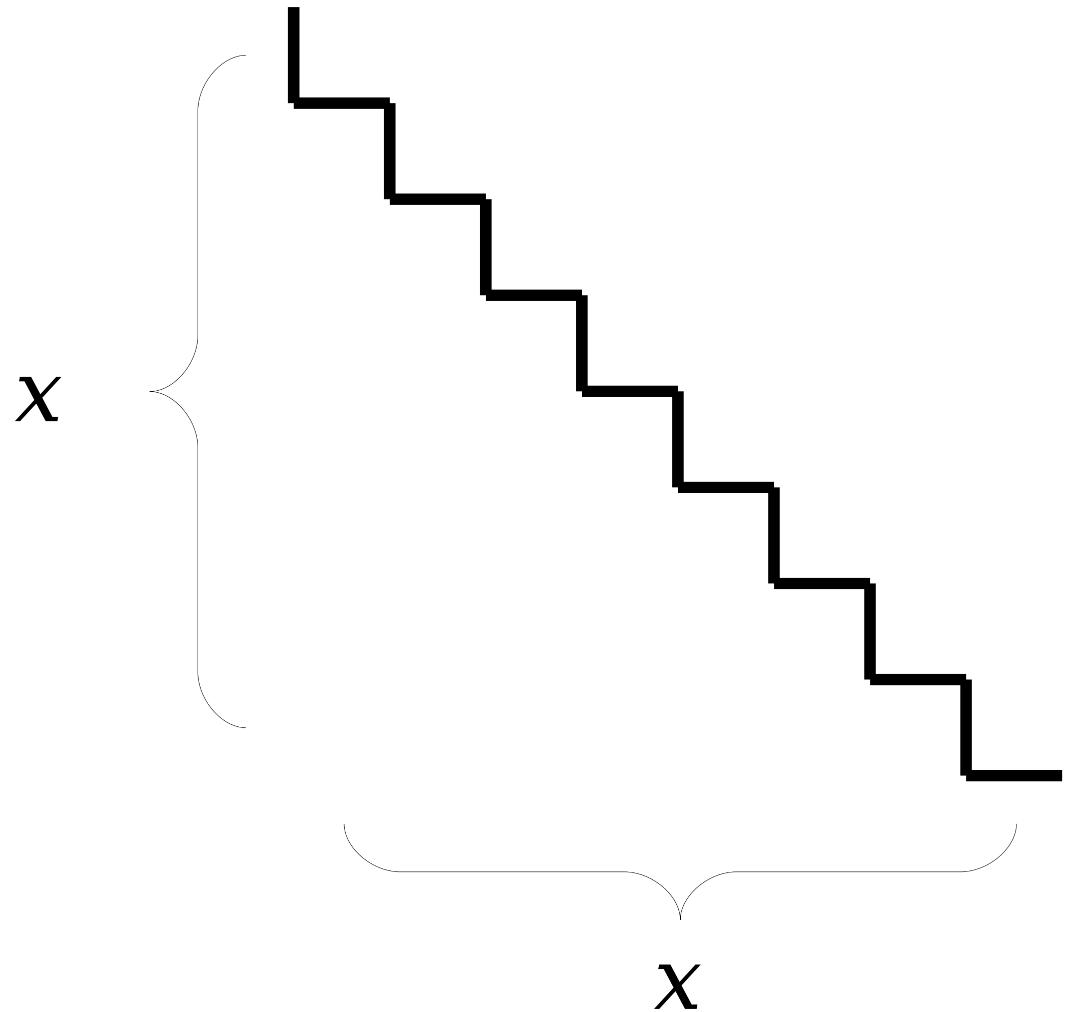
Reasoning about Infinity



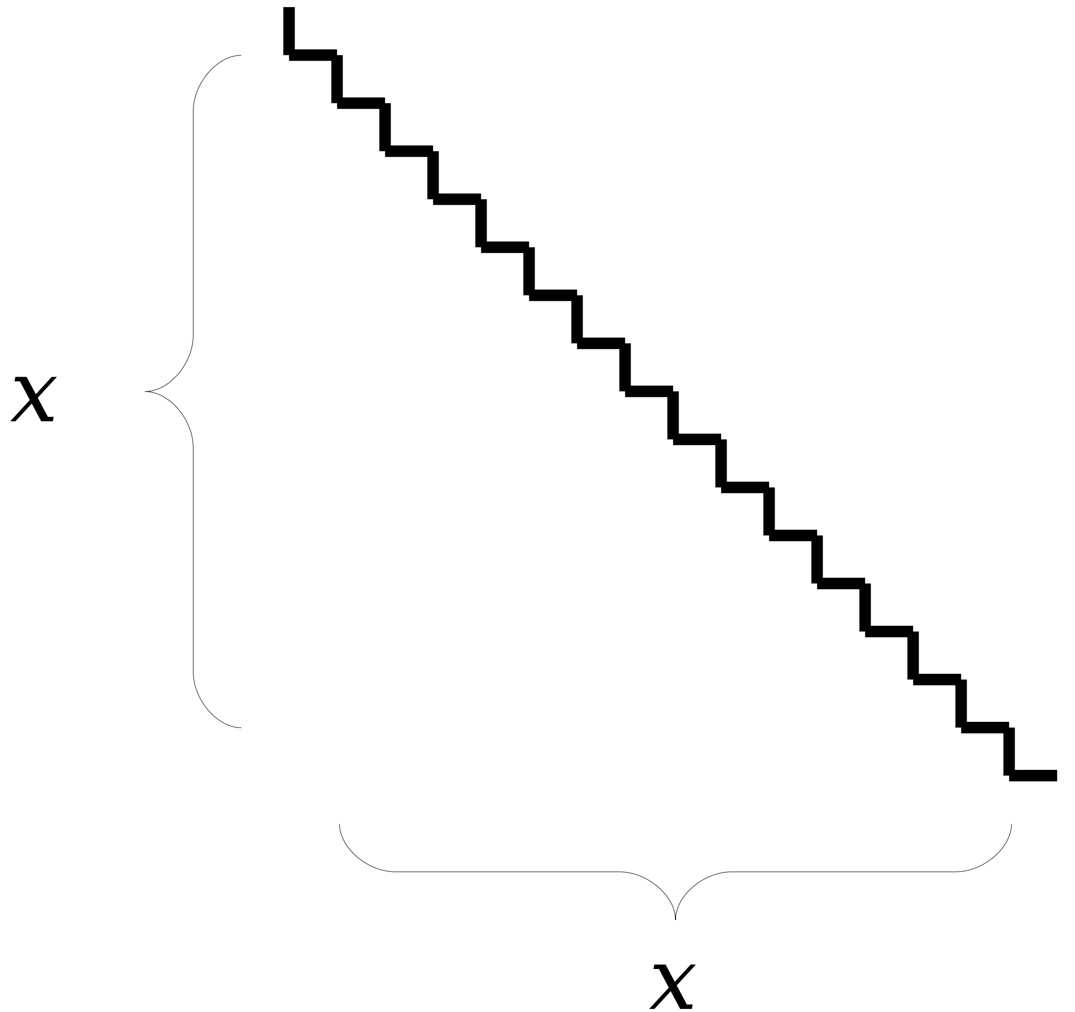
Reasoning about Infinity



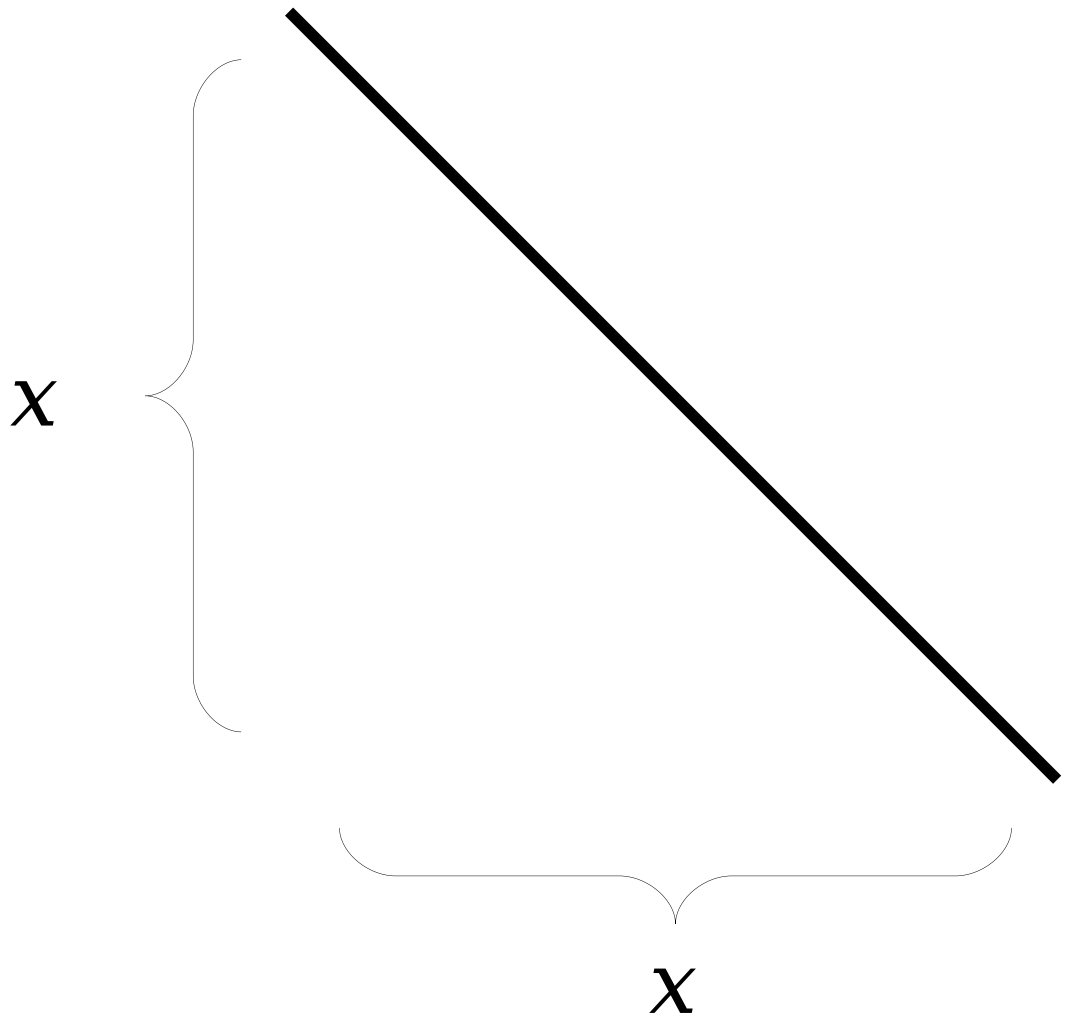
Reasoning about Infinity



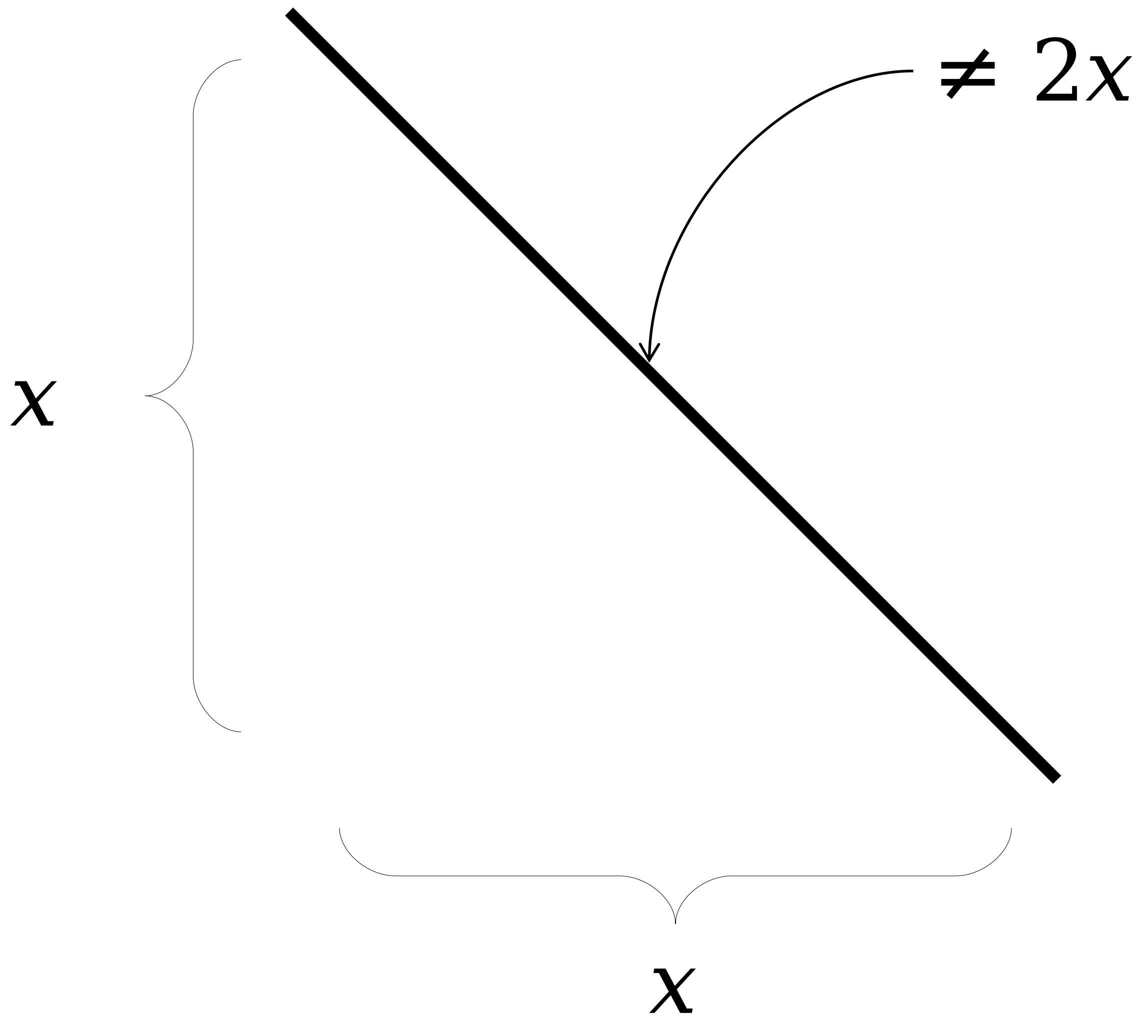
Reasoning about Infinity



Reasoning about Infinity



Reasoning about Infinity



Reasoning about Infinity

$$0.9 < 1$$

Reasoning about Infinity

$$0.99 < 1$$

Reasoning about Infinity

$$0.999 < 1$$

Reasoning about Infinity

$$0.9999 < 1$$

Reasoning about Infinity

$$0.99999 < 1$$

Reasoning about Infinity

$$\overline{0.99999} \not< 1$$

Reasoning about Infinity

0 is finite

Reasoning about Infinity

1 is finite

Reasoning about Infinity

2 is finite

Reasoning about Infinity

3 is finite

Reasoning about Infinity

4 is finite

Reasoning about Infinity

∞ is finite

Reasoning about Infinity

∞ is finite

^ not

Reasoning About the Infinite

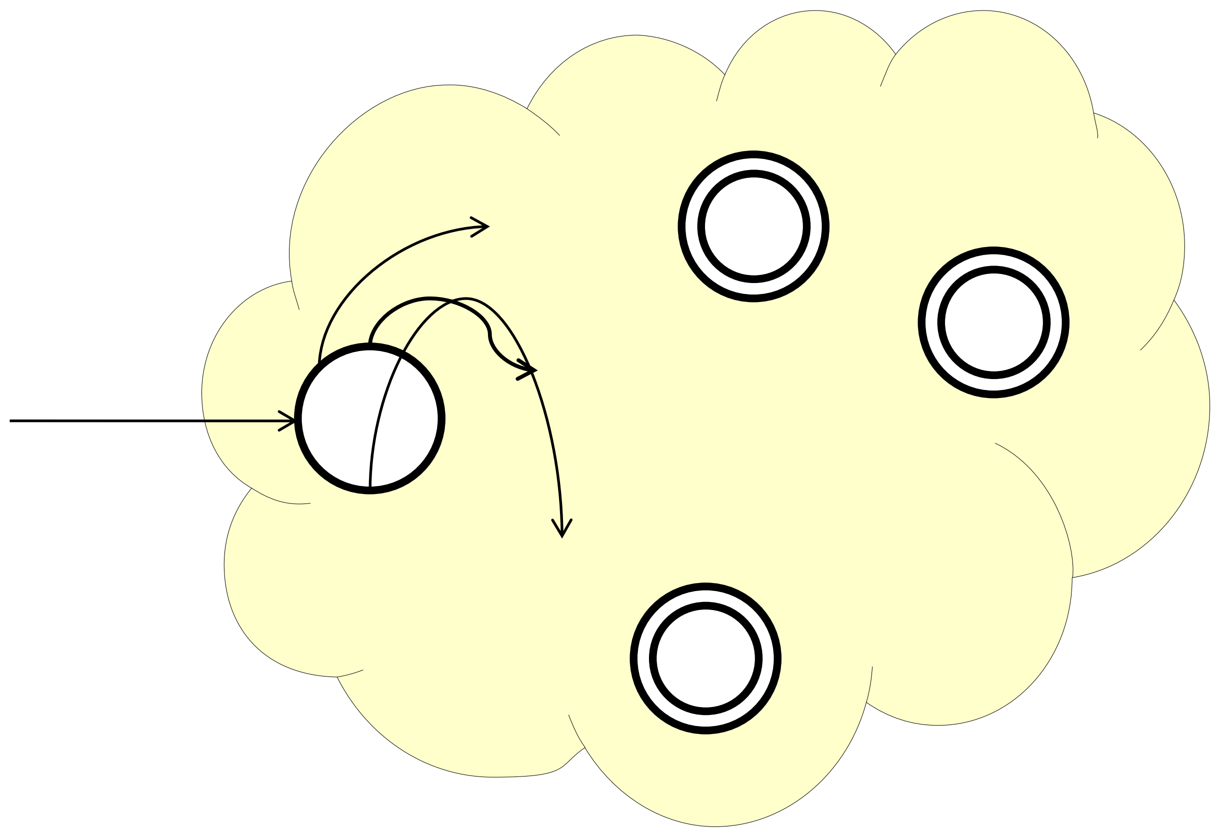
If a series of finite objects all have some property, the "limit" of that process *does not* necessarily have that property.

In general, it is not safe to conclude that some property that always holds in the finite case must hold in the infinite case.

(This is why calculus is interesting).

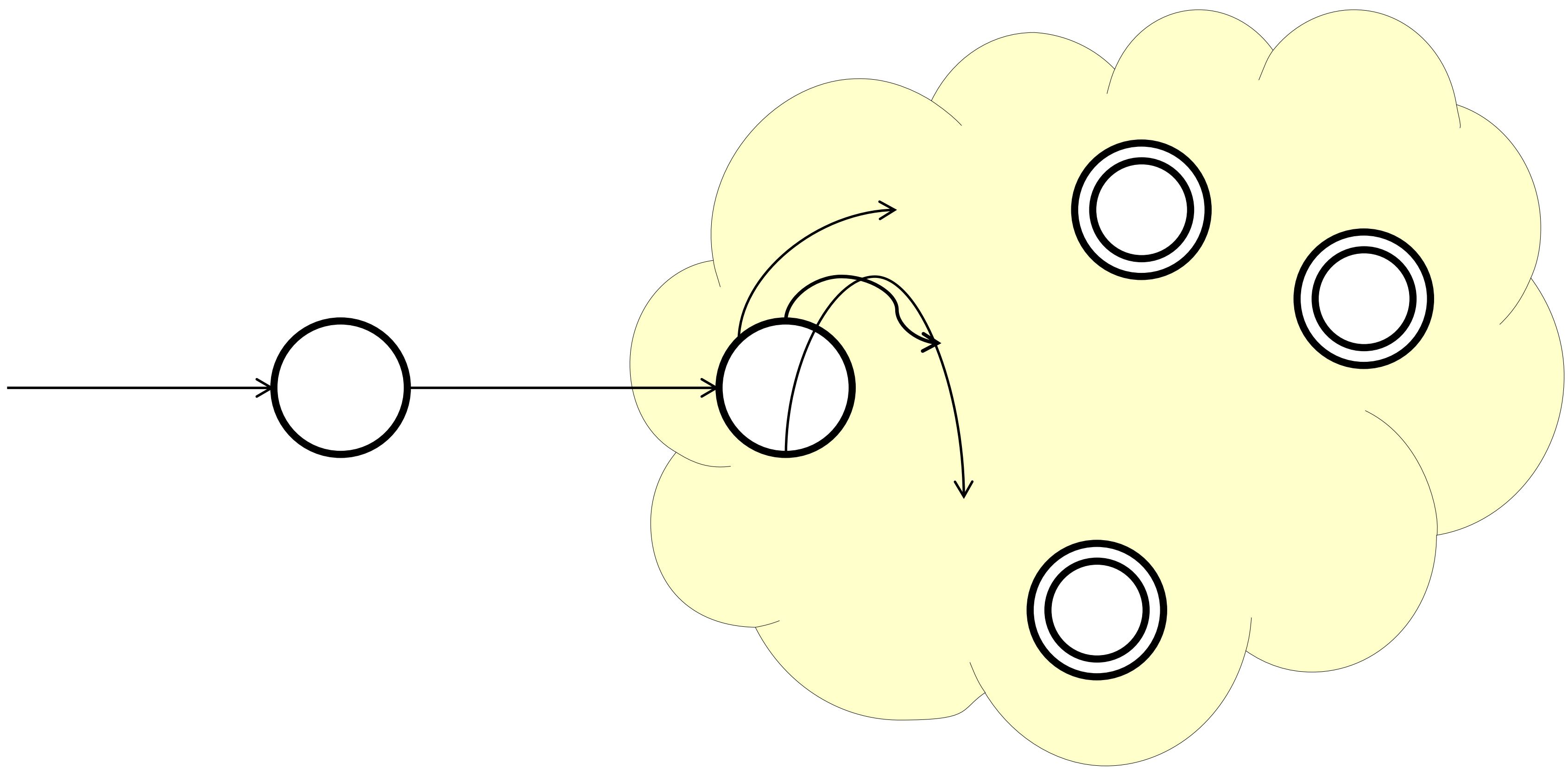
Idea: Can we directly convert an NFA for language L to an NFA for language L*?

The Kleene Star

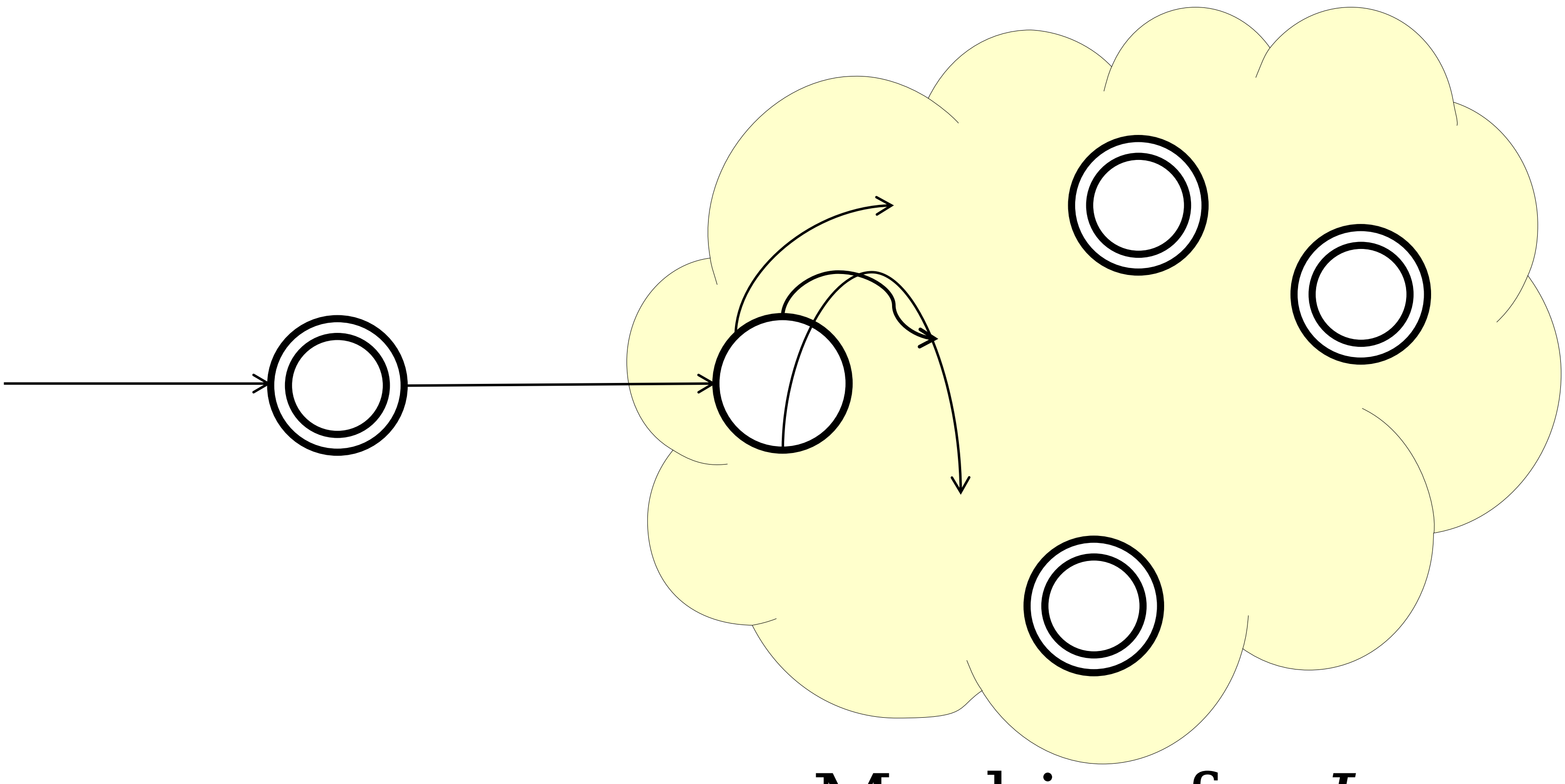


Machine for L

The Kleene Star

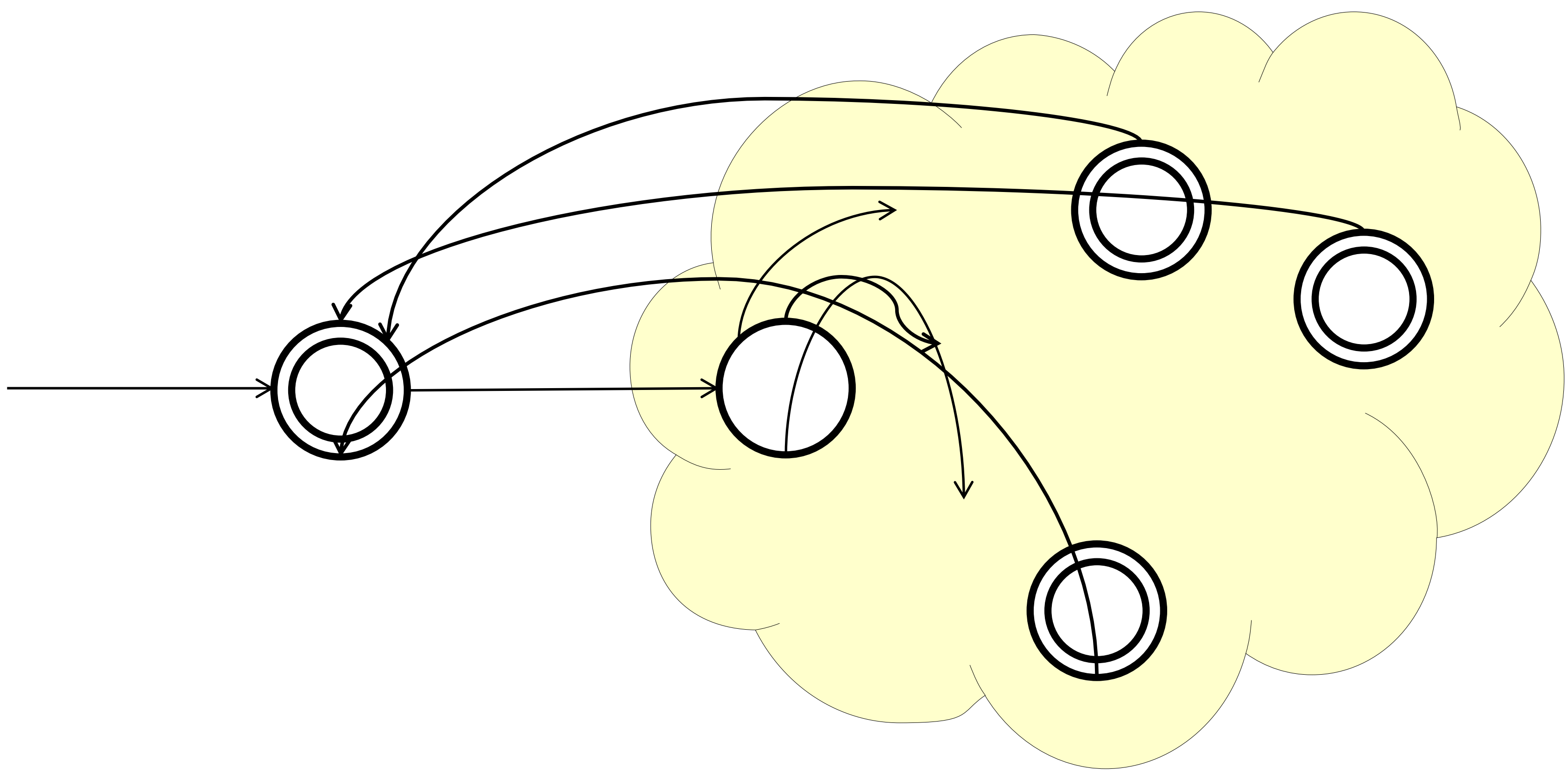


The Kleene Star



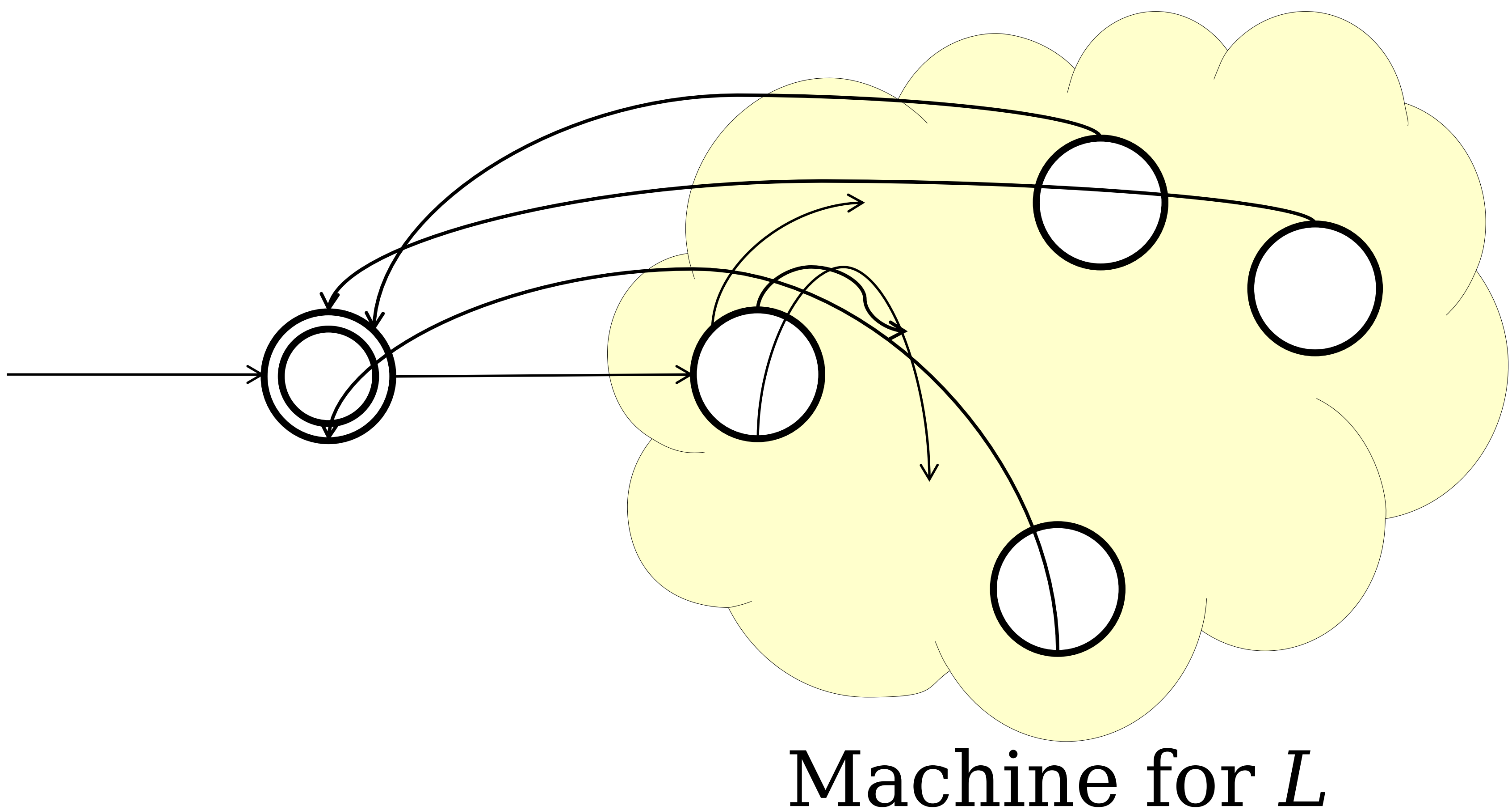
Machine for L

The Kleene Star

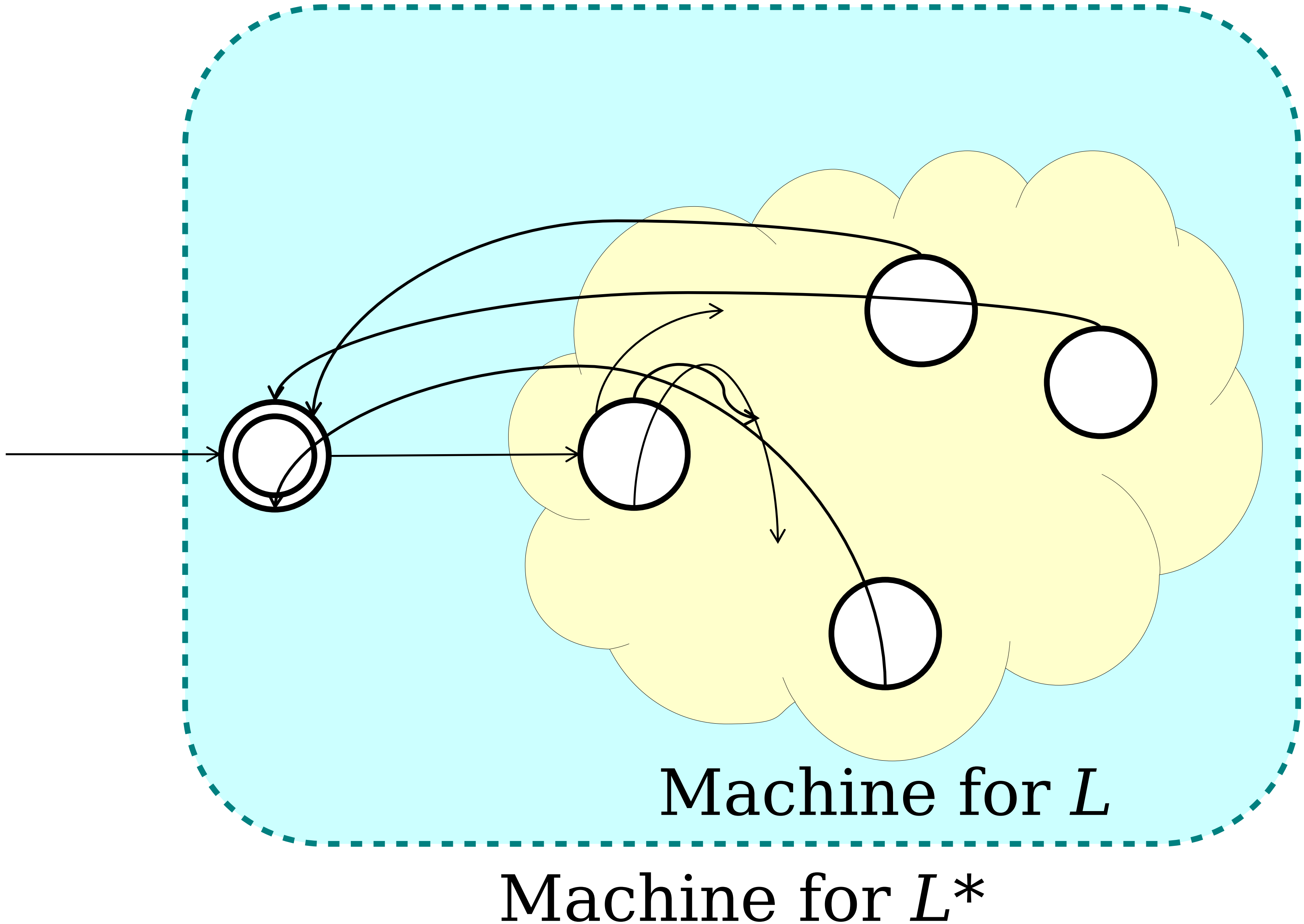


Machine for L

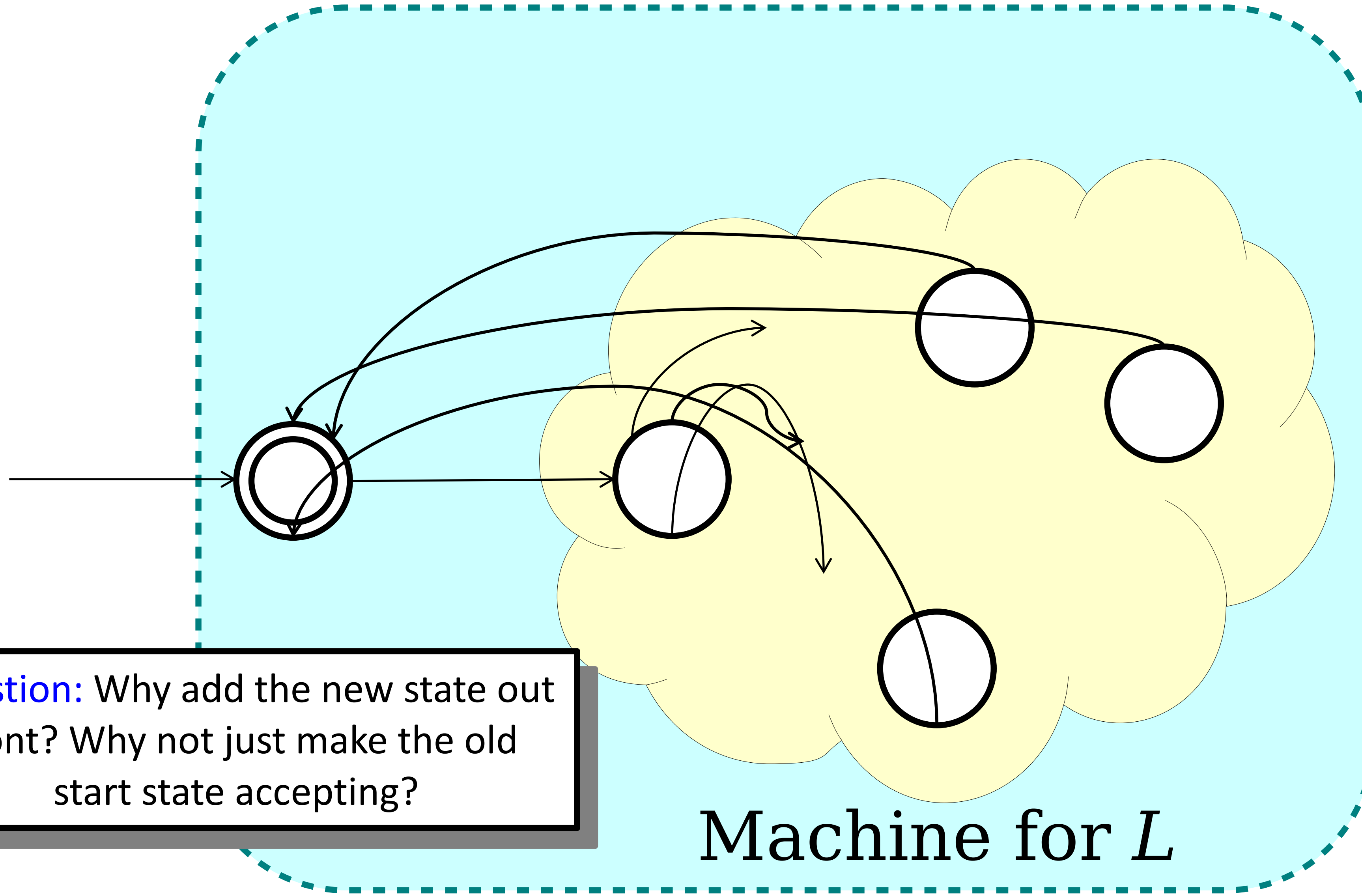
The Kleene Star



The Kleene Star



The Kleene Star



Machine for L^*

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

Closure Properties

Theorem: If L_1 and L_2 are regular languages over an alphabet Σ, then so are the following languages:

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!

Thought for the Weekend

Learning How to Learn