

# Finite Automata

## Part Three

# Recap from Last Time: Some NFA Design Examples

# 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.

# Guess-and-Check

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

# Guess-and-Check

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

Ask yourself these design questions:

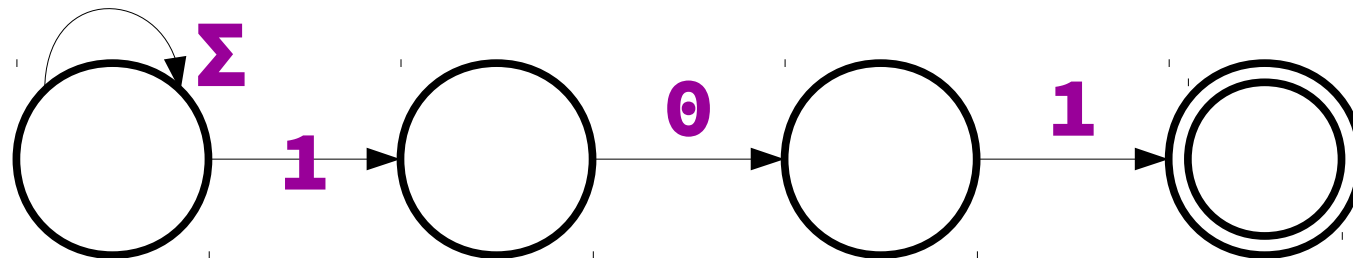
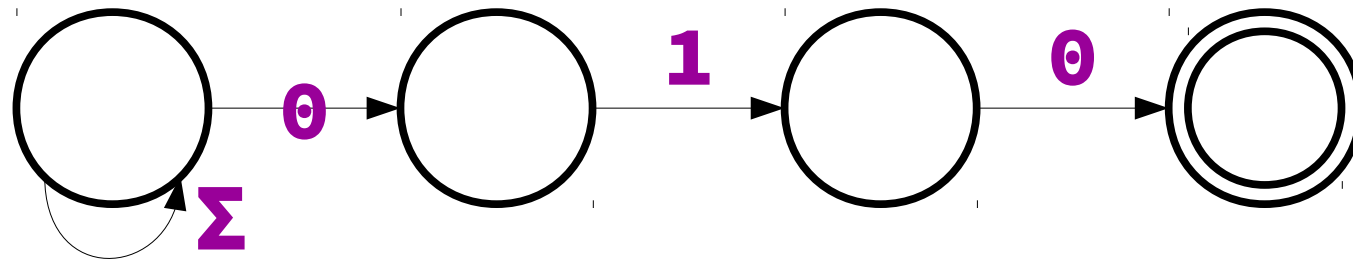
Would it be really easy to design an NFA to detect the substring 010 at the end, if you knew that's what you were looking for, and when you'd reached the near-end?

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

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

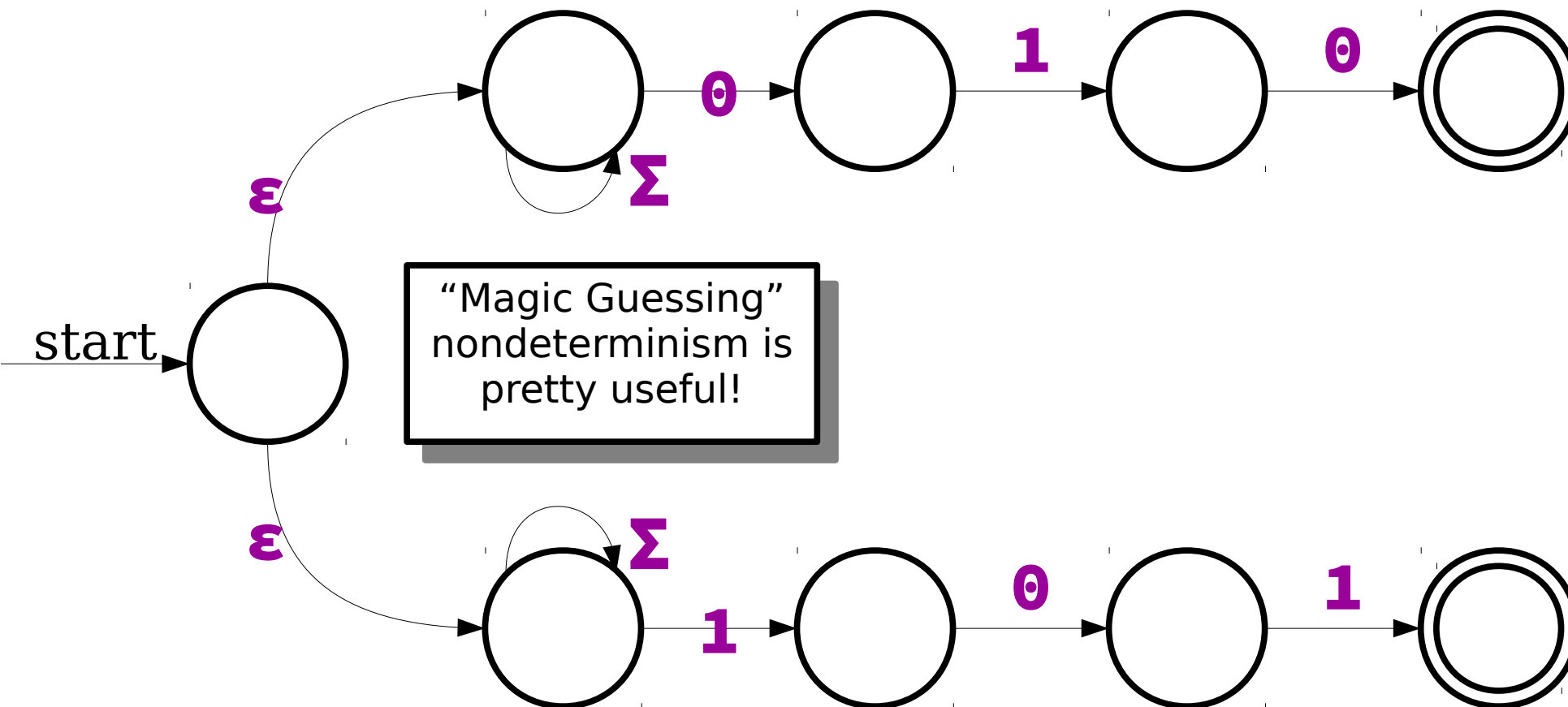
# Guess-and-Check

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



# Guess-and-Check

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



# Guess-and-Check

$$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$$
$$= L_1 \cup L_2 \text{ where:}$$

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

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

## **NFA Design Hack!**

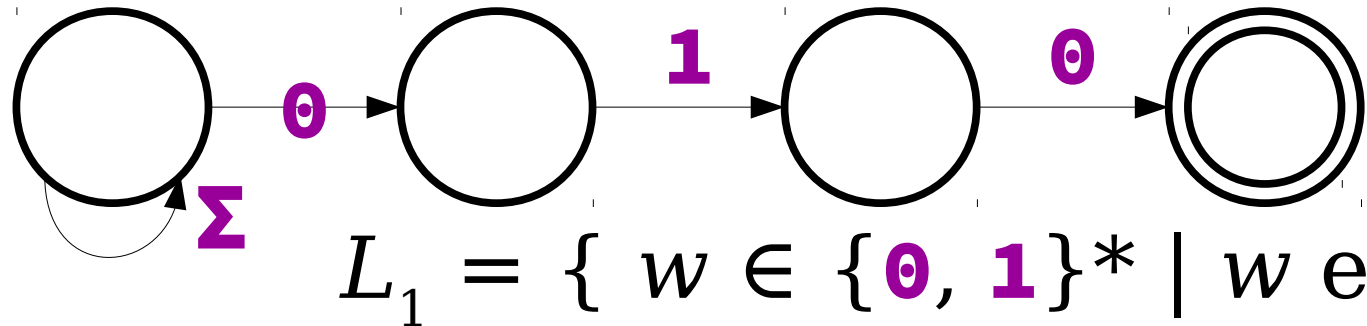
If you can write the language as the union of two or more very simple languages:

- (1) make simple DFA/NFAs for those simple languages
- (2) a single start state dispatches to the simple DFA/NFAs using epsilon transitions

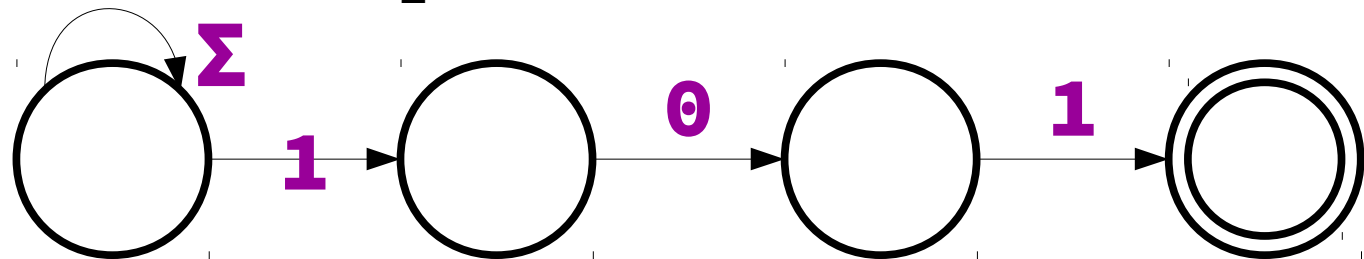


# Guess-and-Check

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

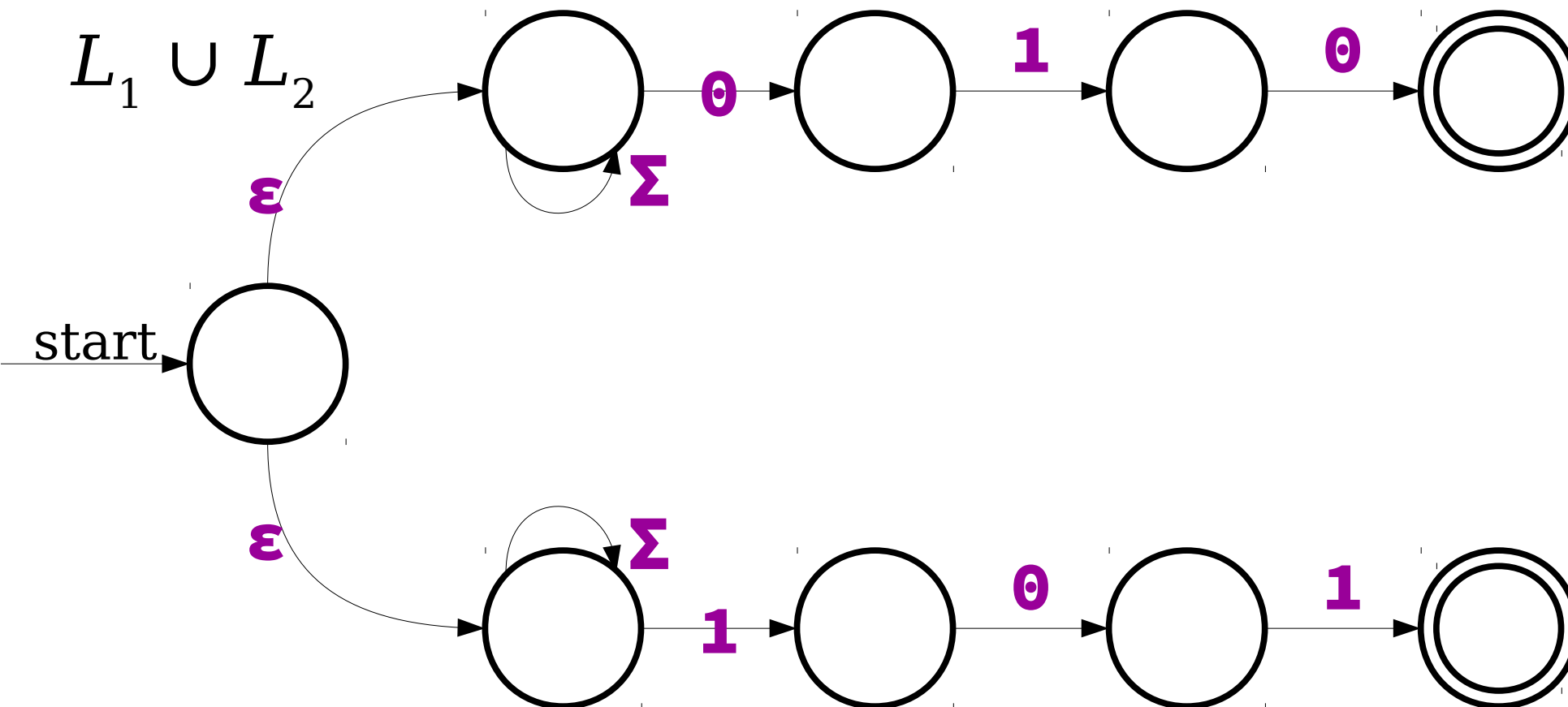


$$L_2 = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 101 \}$$



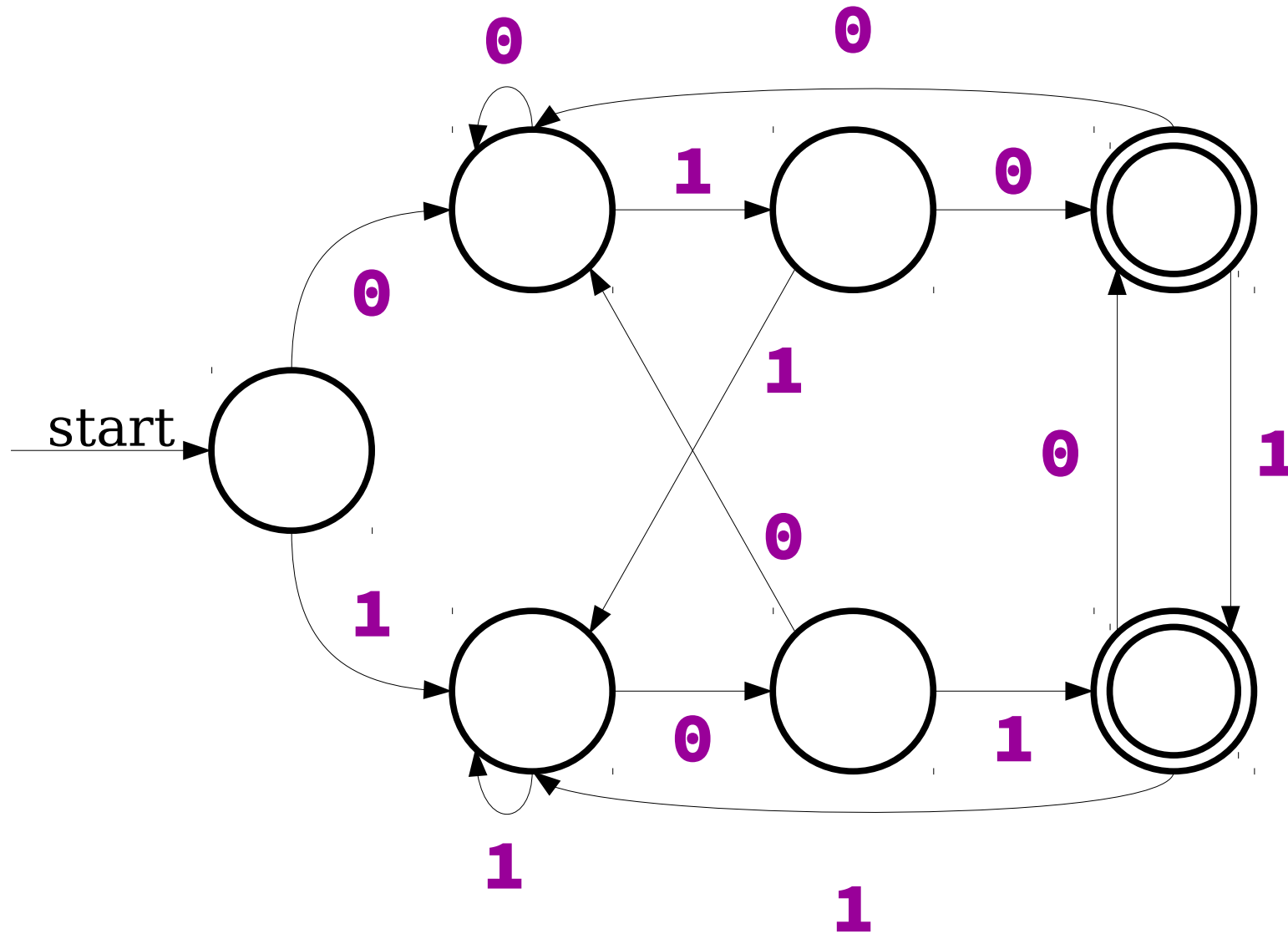
# Guess-and-Check

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



# Guess-and-Check

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



# Guess-and-Check

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

# Guess-and-Check

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

Ask yourself these design questions:

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

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

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

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

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

# Guess-and-Check

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

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

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

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

$L = L_1 \cup L_2 \cup L_3$

Ask yourself these design questions:

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

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

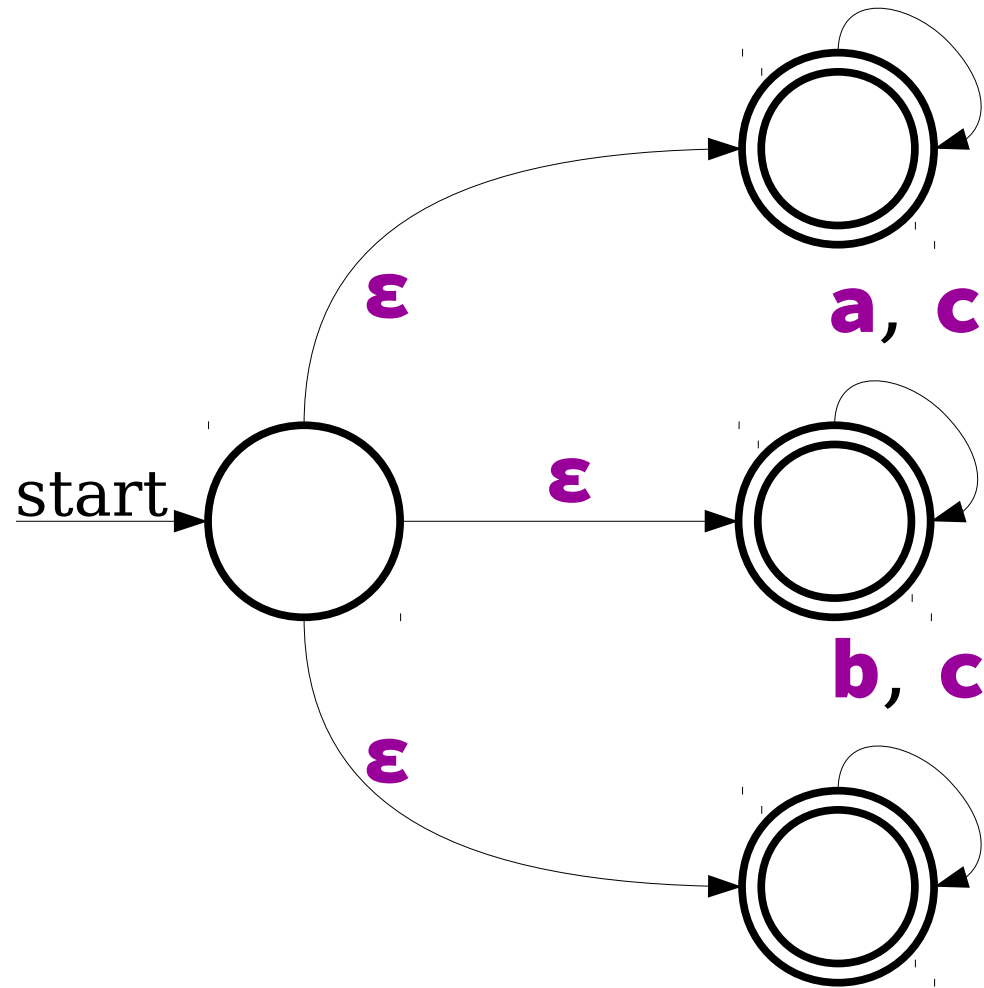
...has no b's in it, if you knew that's what you were looking for?

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

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

# Guess-and-Check

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



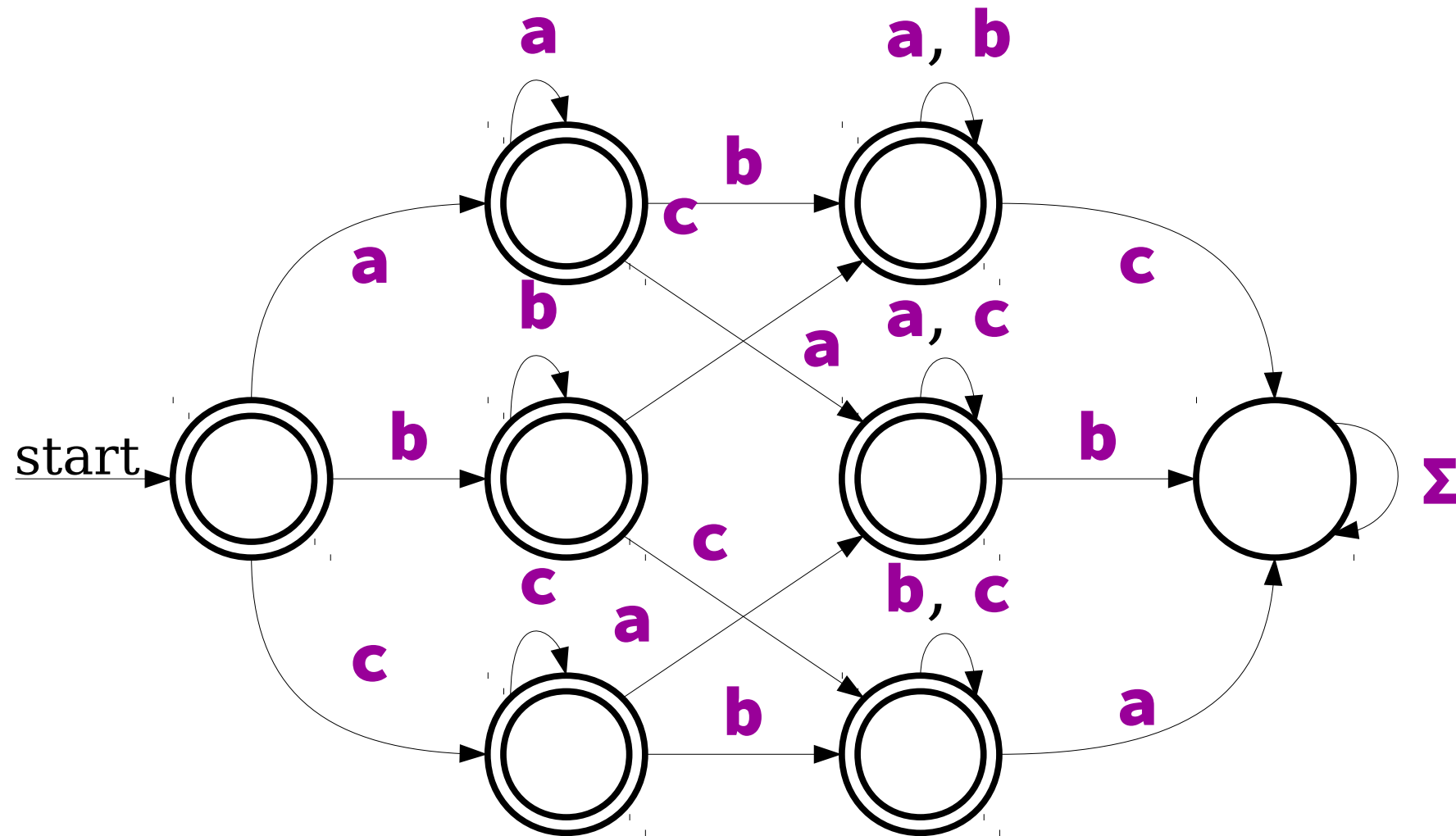
$$L_3 = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid \mathbf{c} \text{ is not in } w \}$$

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

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

# Guess-and-Check

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





New Stuff!

# 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!

# 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?

# NFAs and DFAs

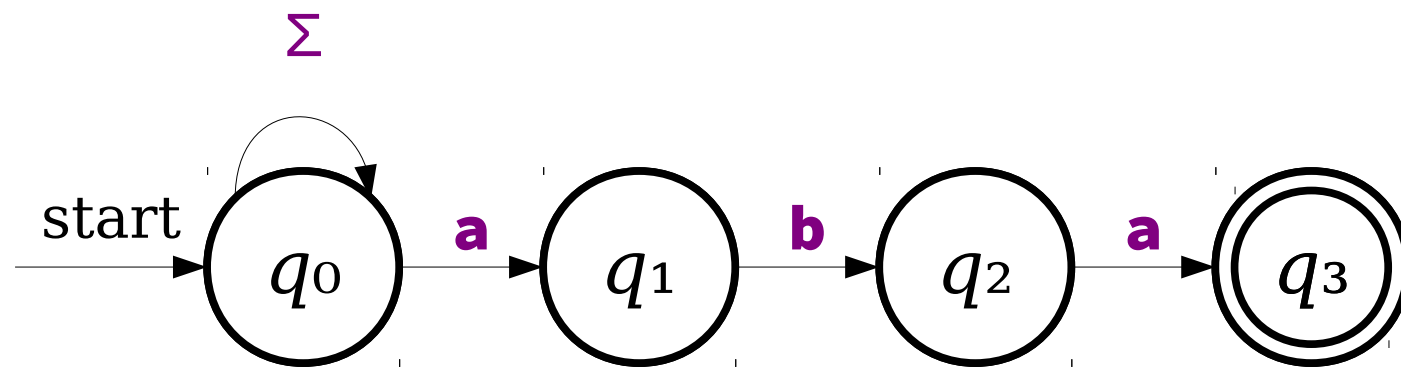
- Any language that can be accepted by a DFA can be accepted by an NFA.
- Why?
  - Every DFA essentially already *is* an NFA!
- **Question:** Can any language accepted by an NFA also be accepted by a DFA?
- Surprisingly, the answer is **yes**!

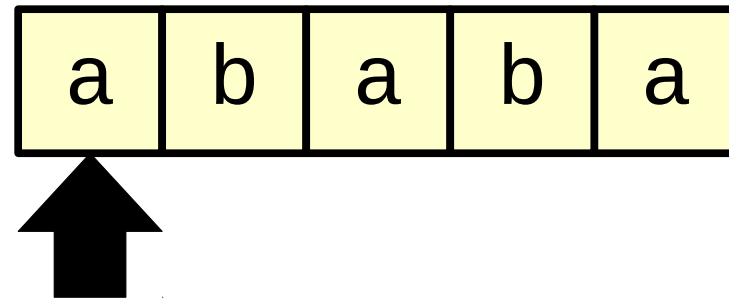
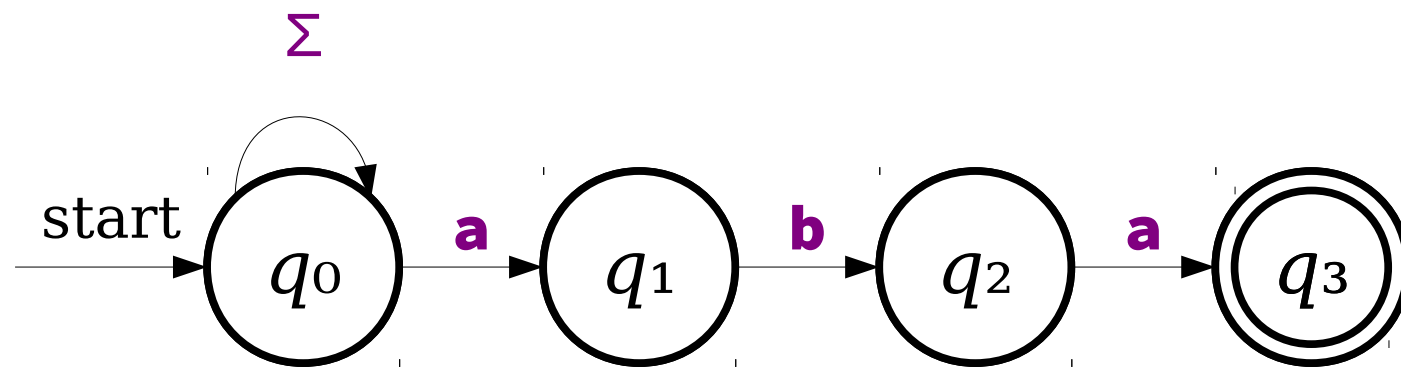
# NFAs and DFAs

- **Question:** Can any language accepted by an NFA also be accepted by a DFA?
- Surprisingly, the answer is **yes!**
  - To prove this, we need to:
    - Pick an arbitrary NFA
    - Describe how we would construct a DFA with the same language (in a generalizable way)
    - For the next few slides, we'll ponder how to approach that...

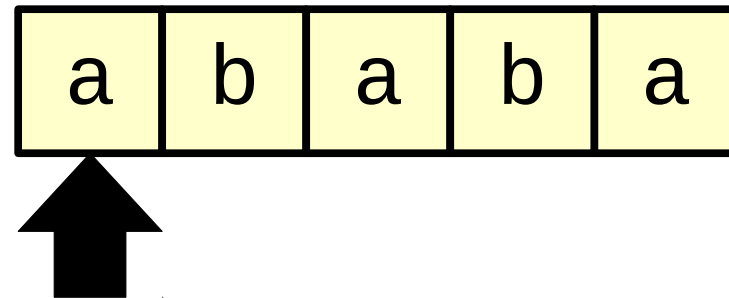
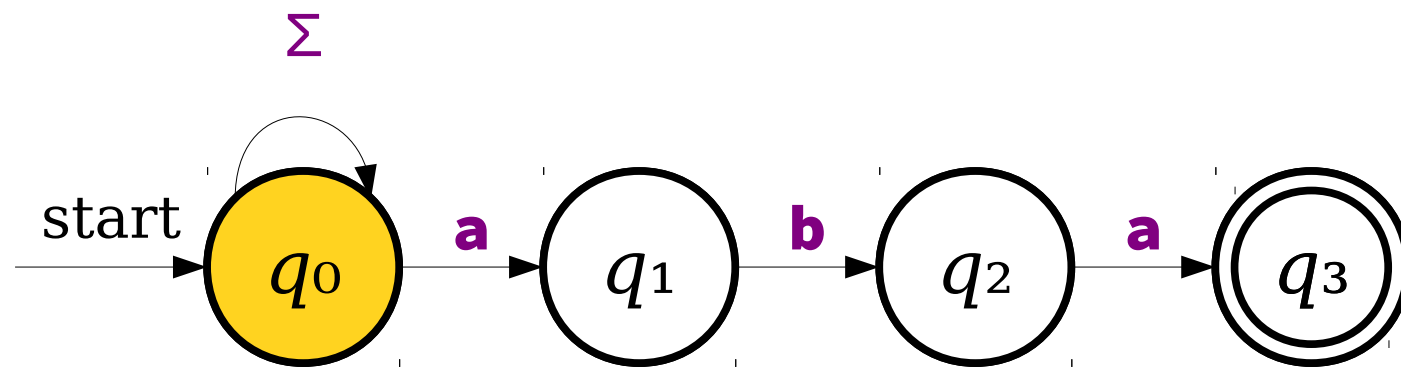
***Thought Experiment:***

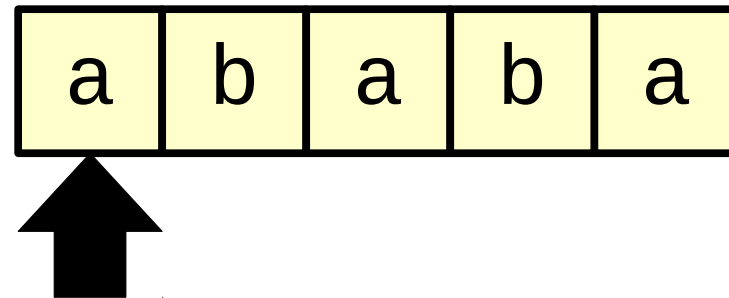
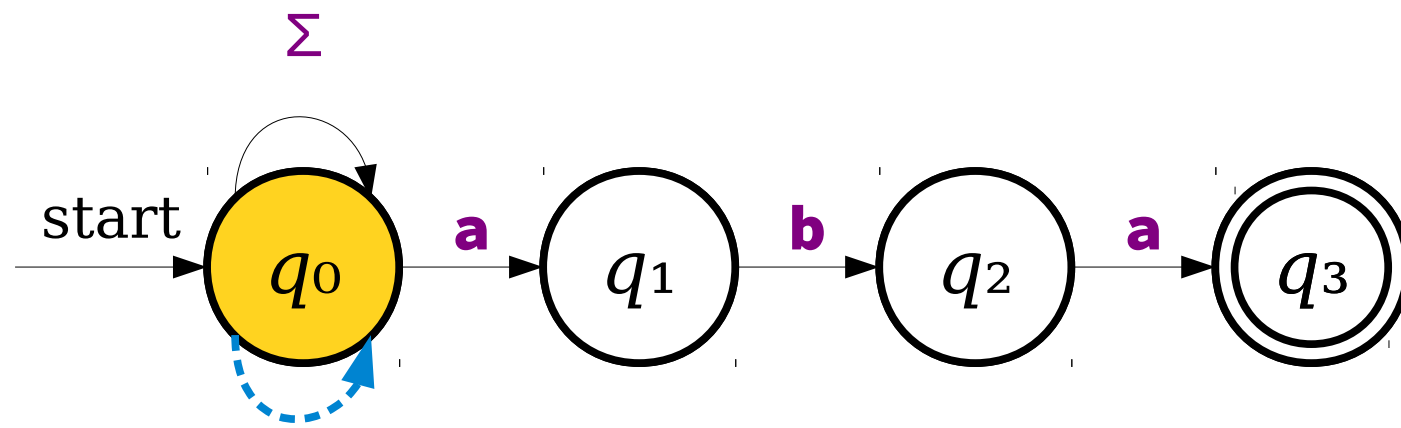
How would you simulate an NFA in software?

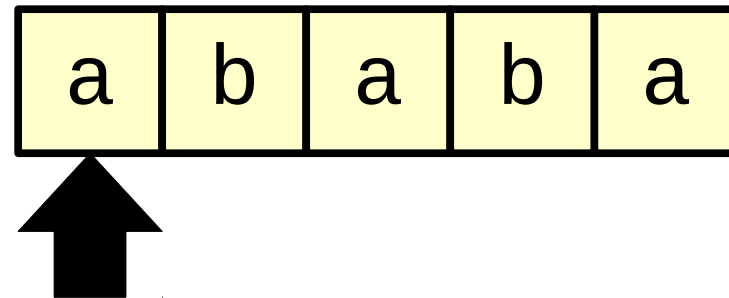
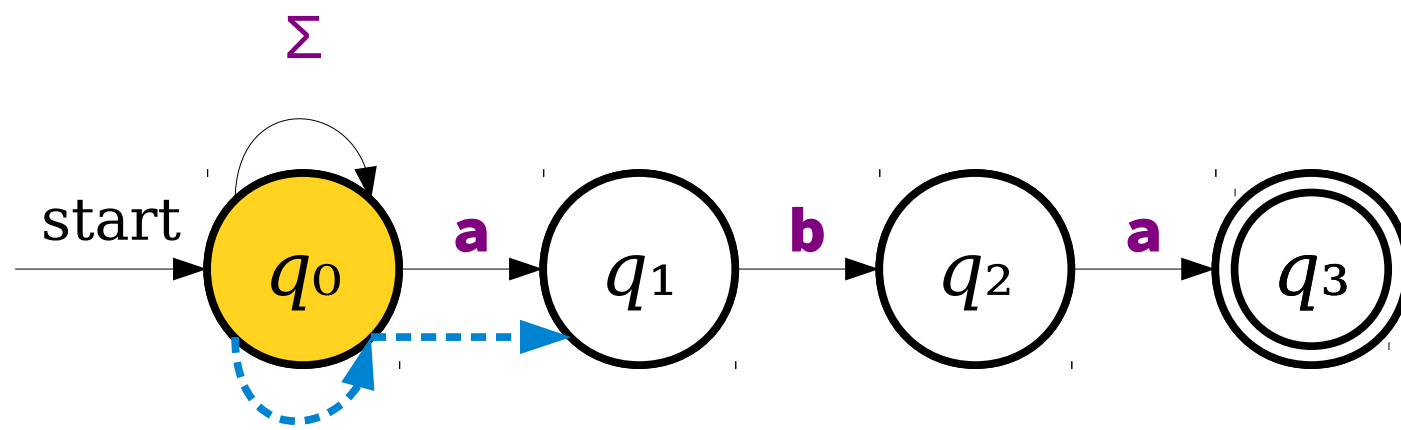


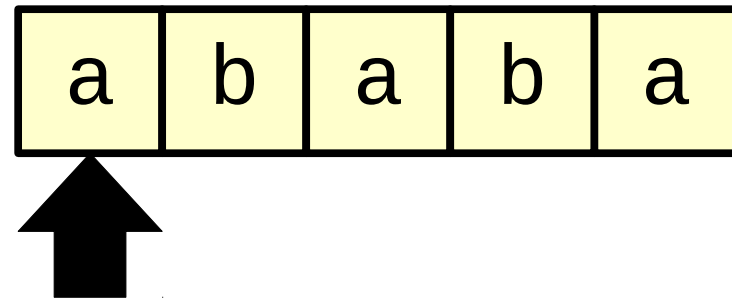
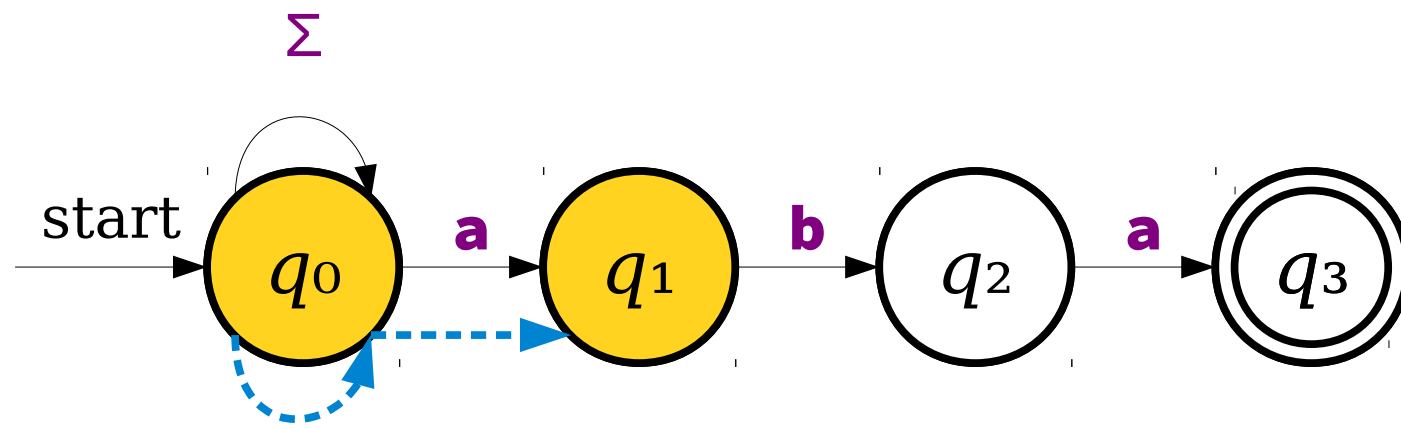


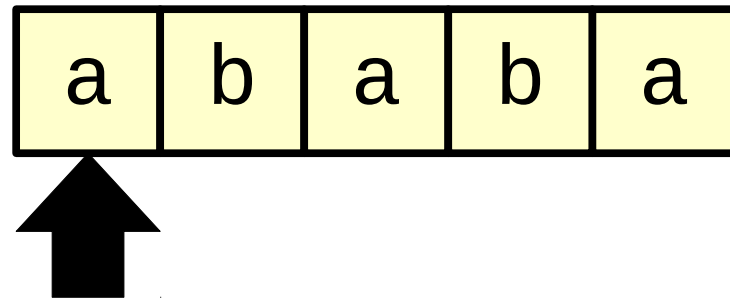
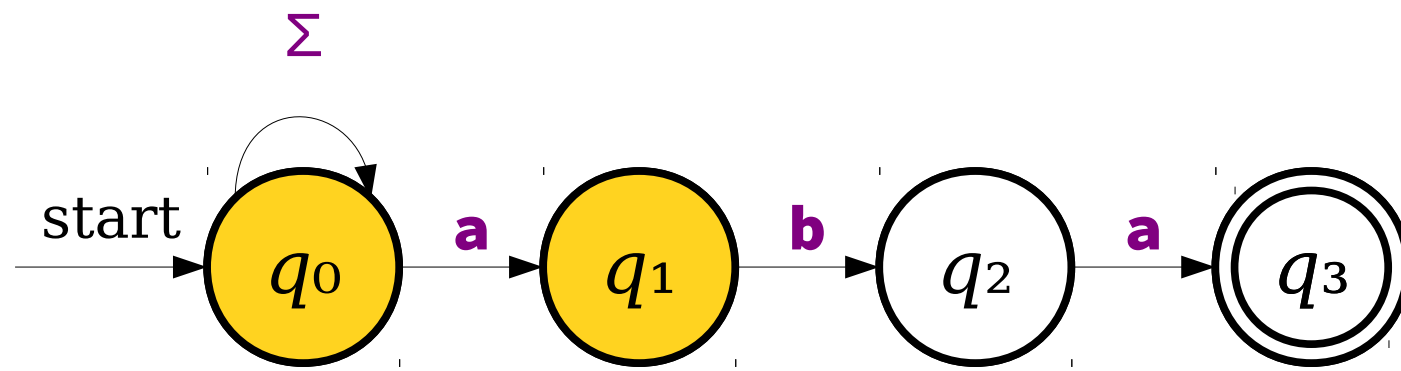


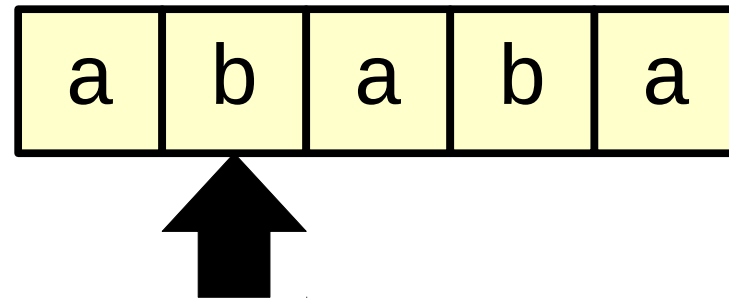
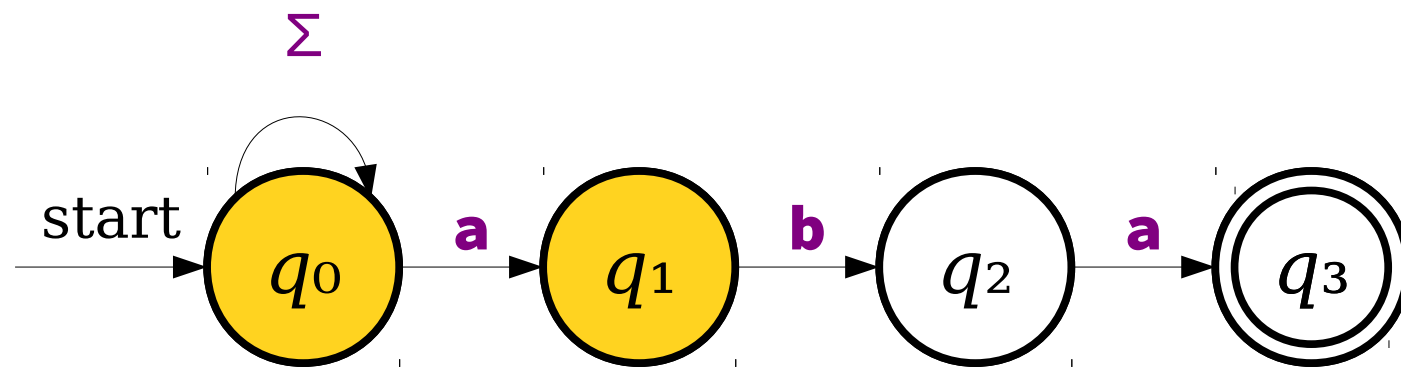


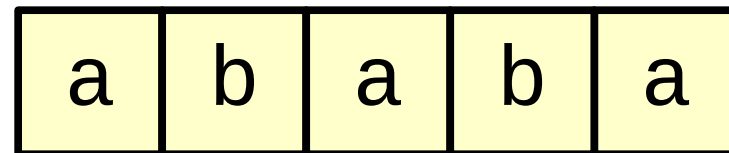
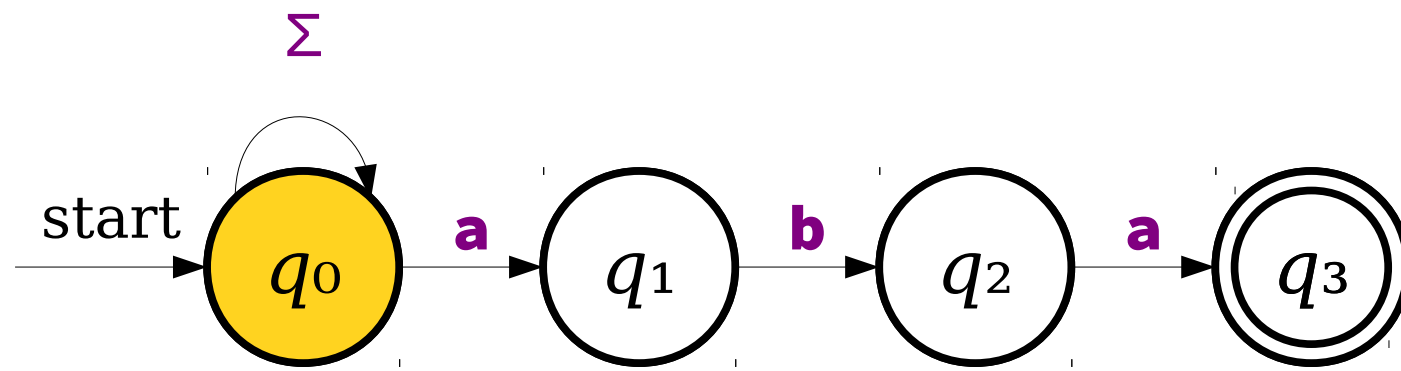


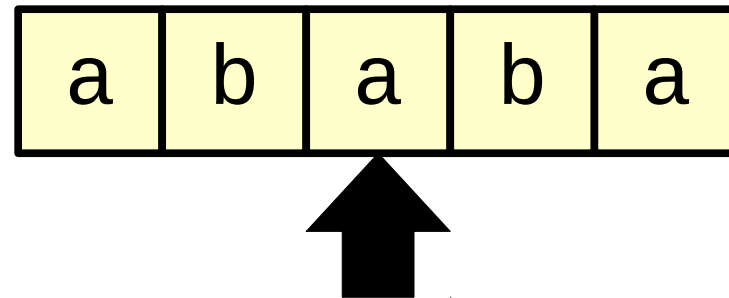
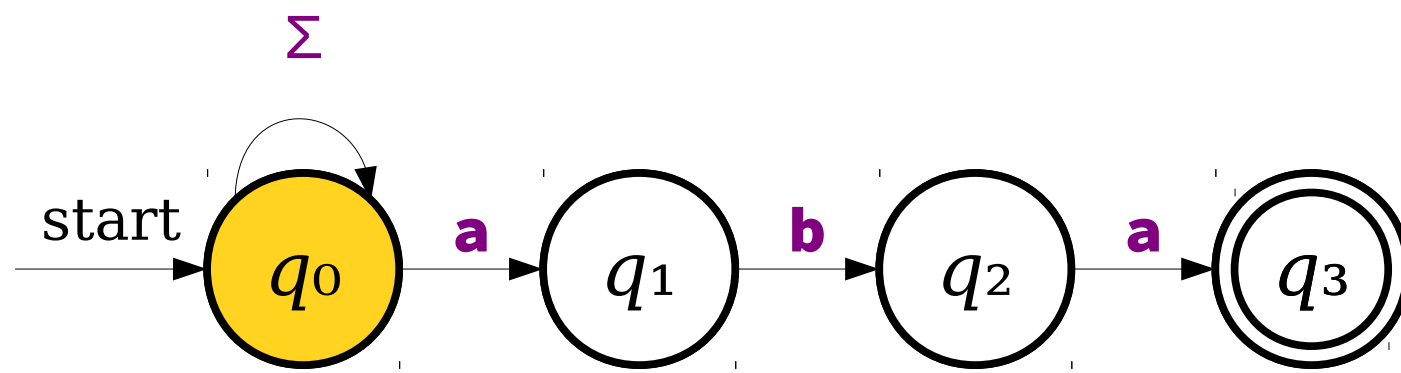




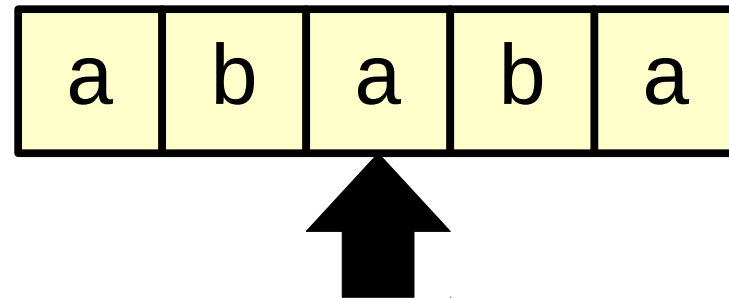
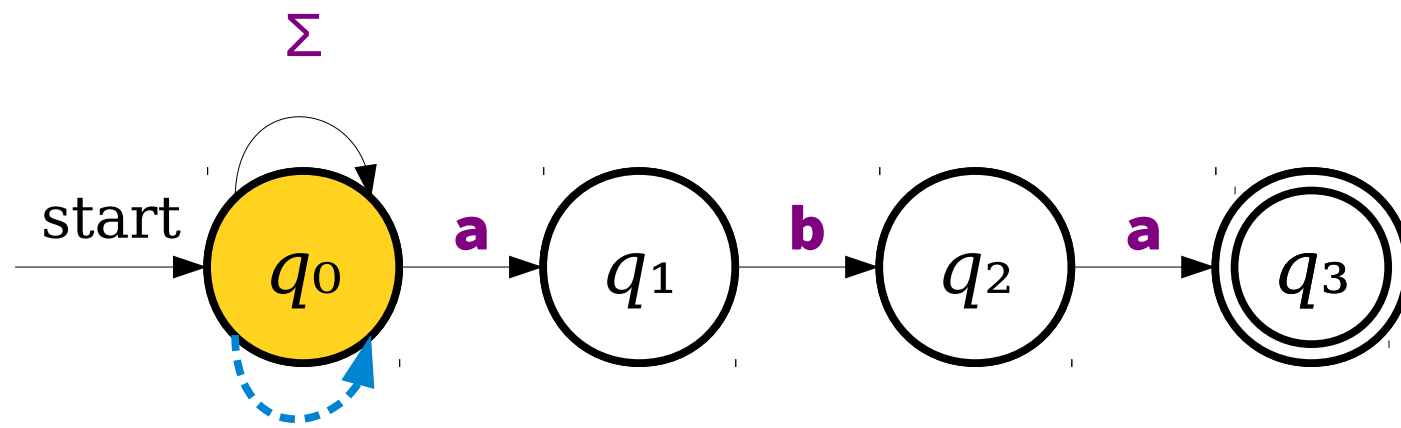


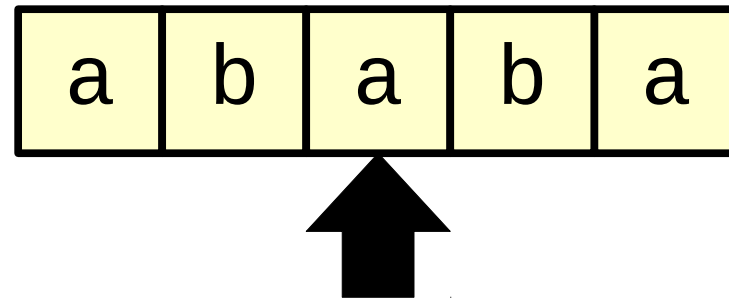
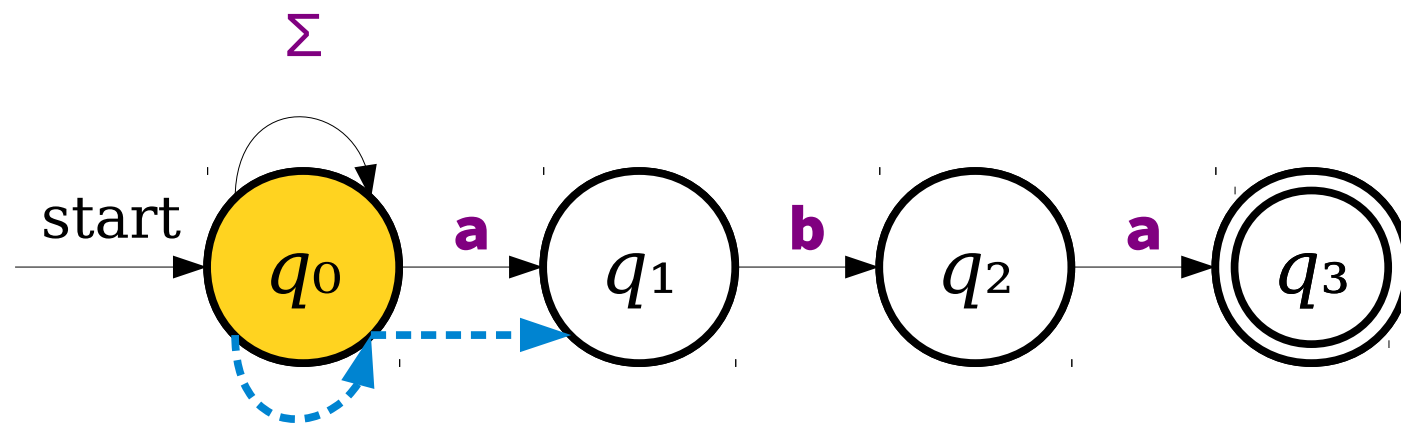


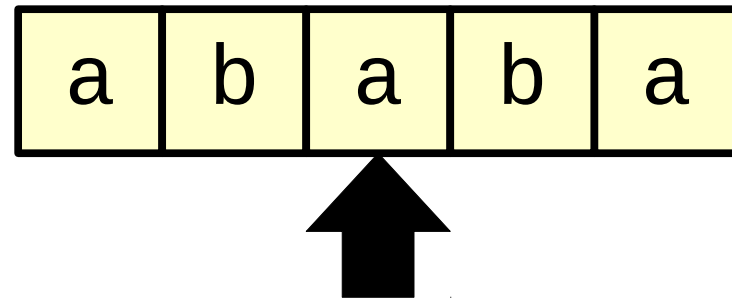
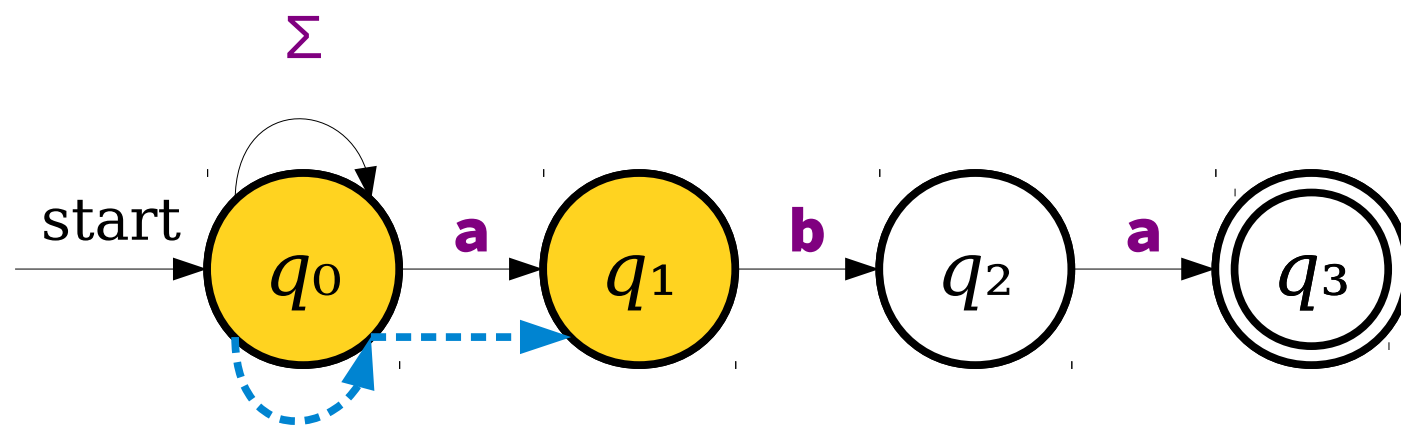


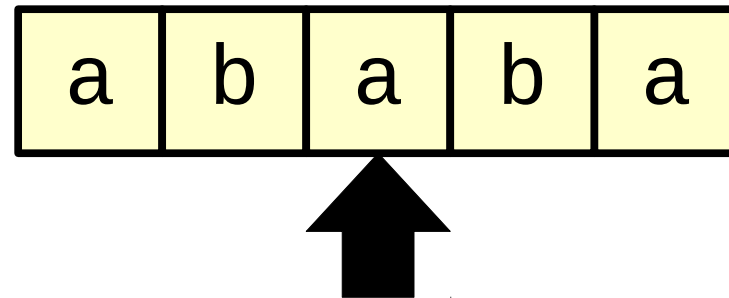
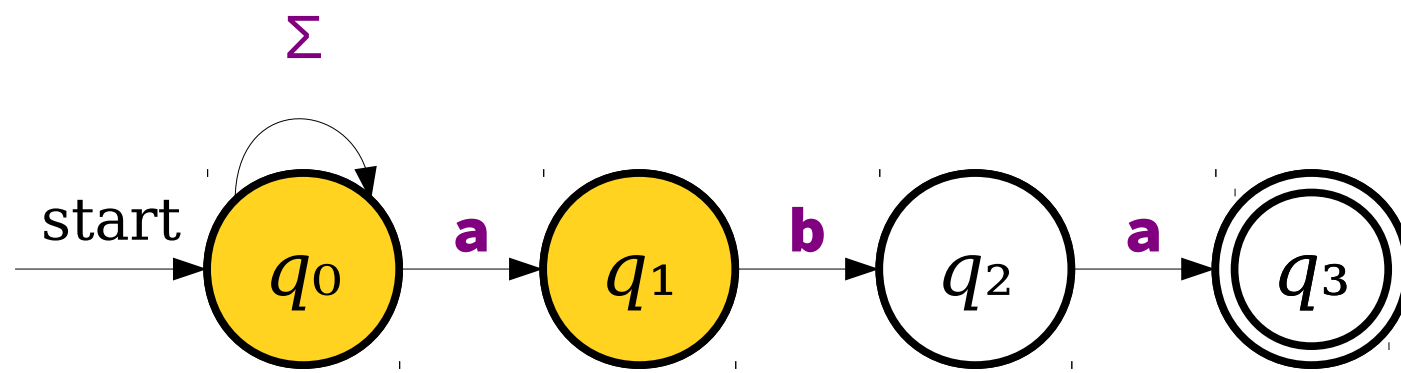


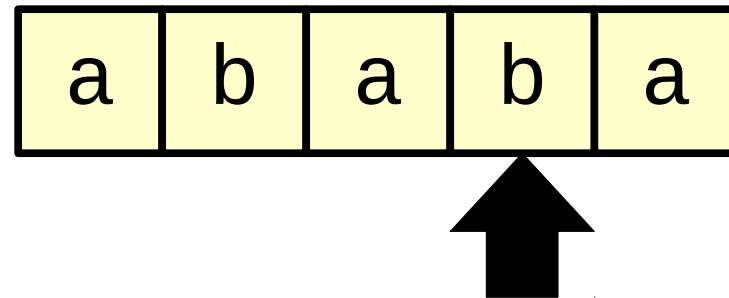
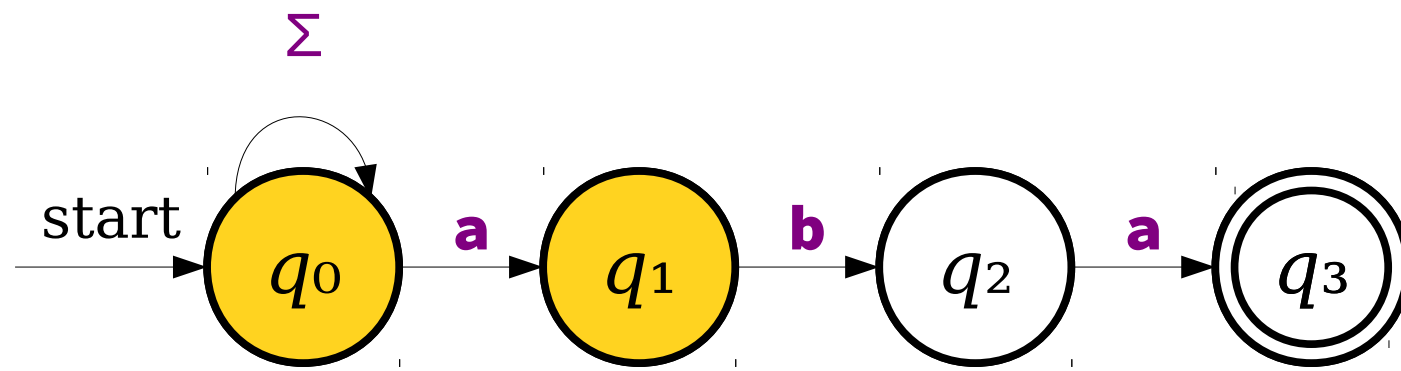


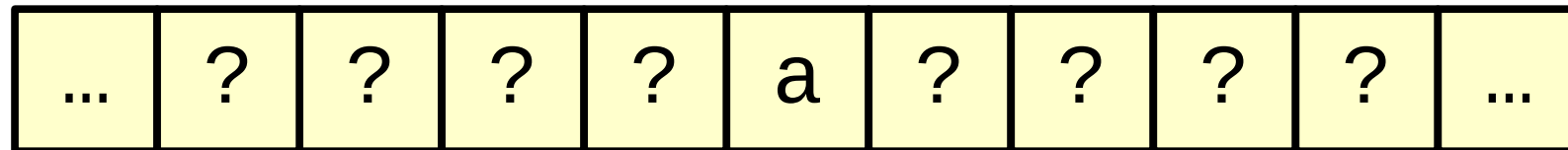
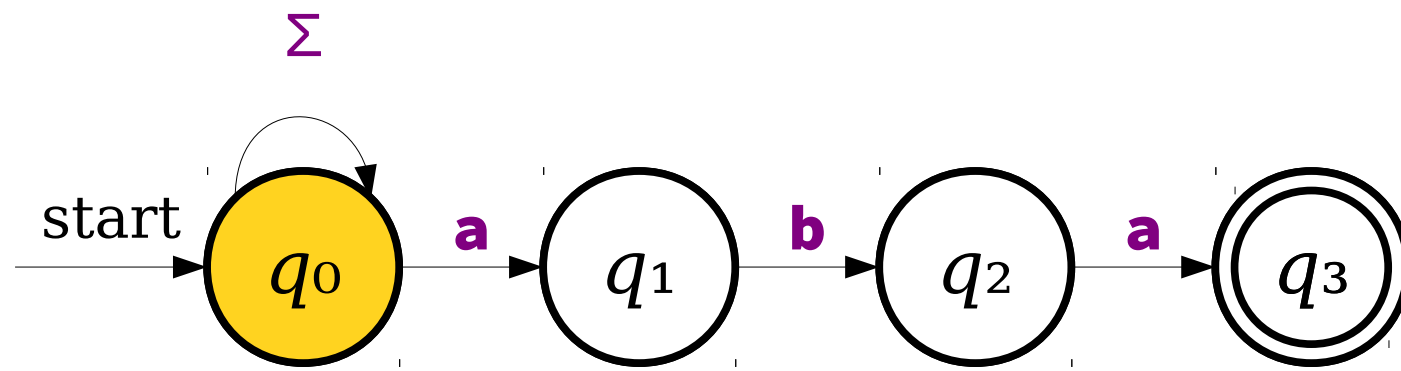


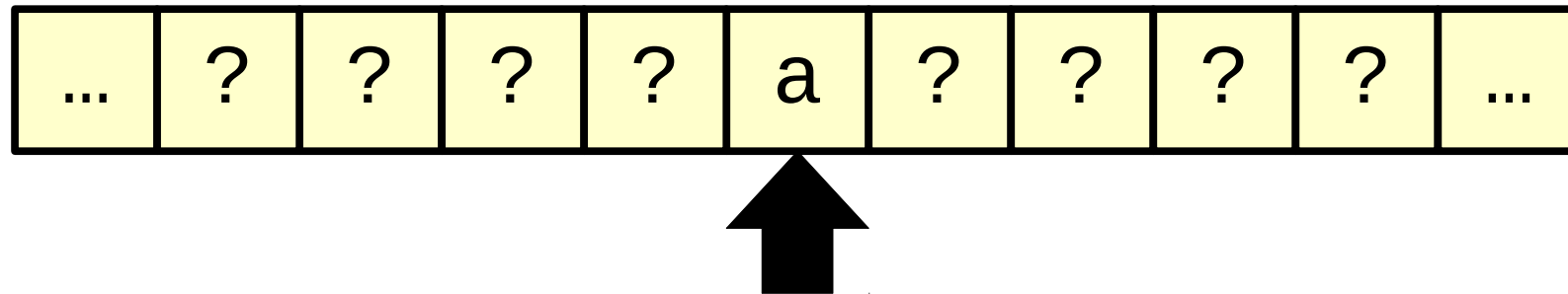
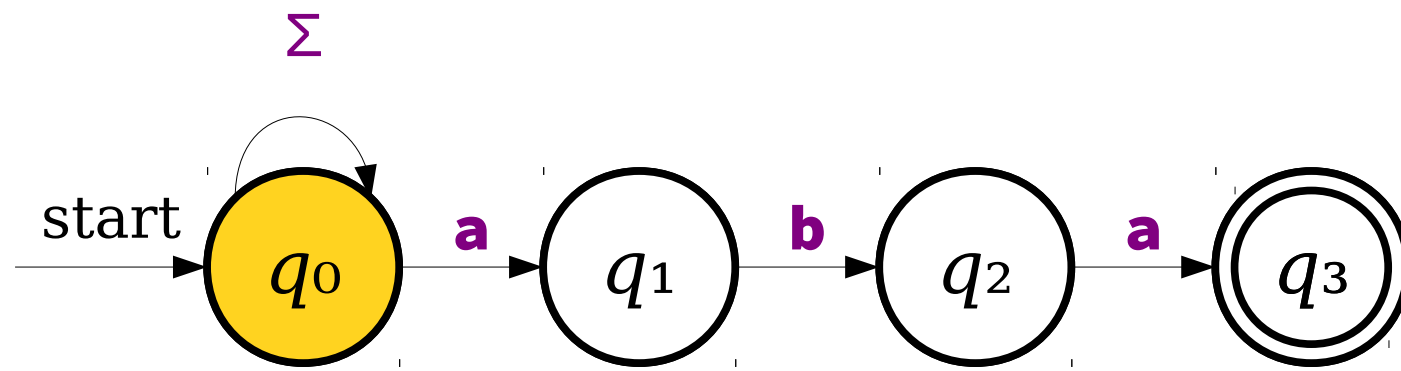


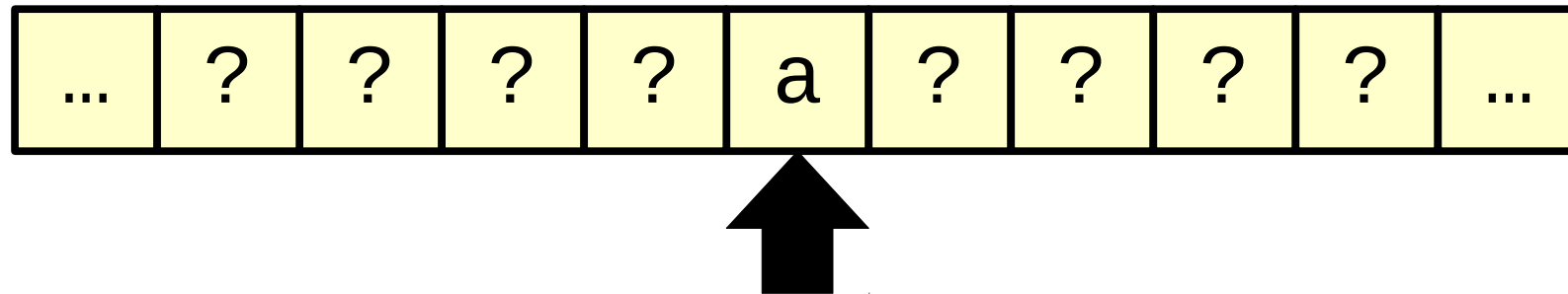
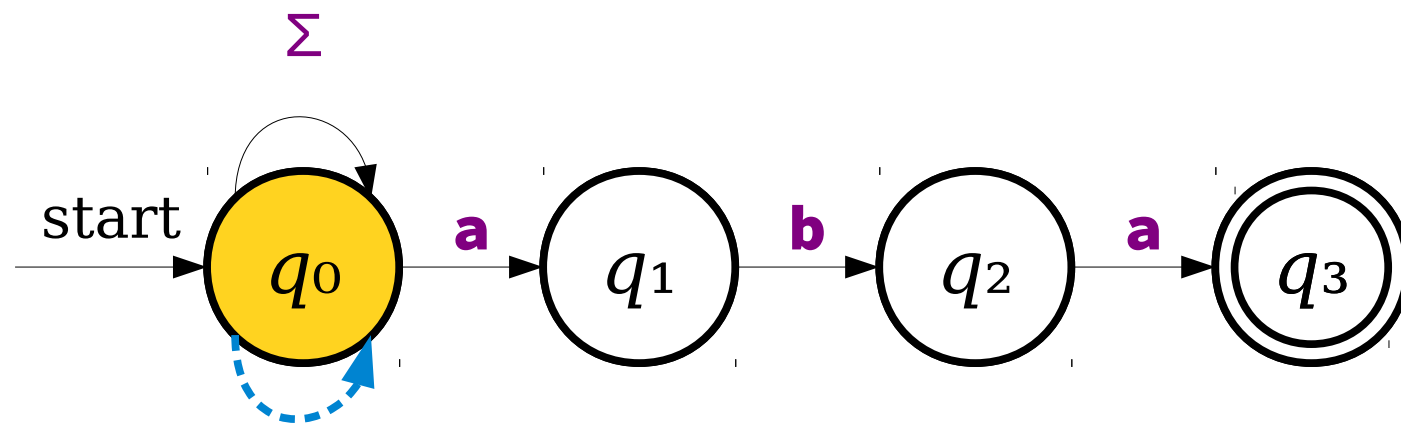




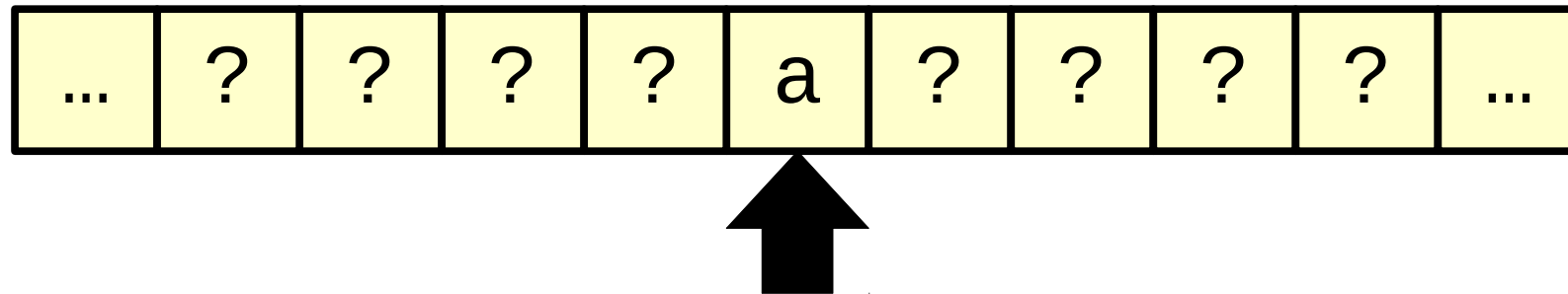
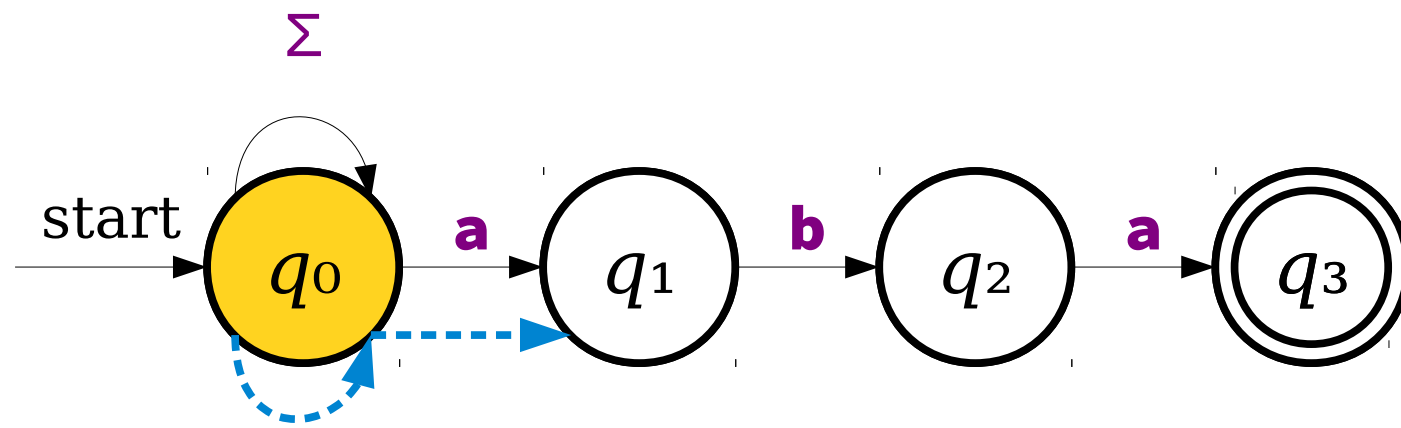


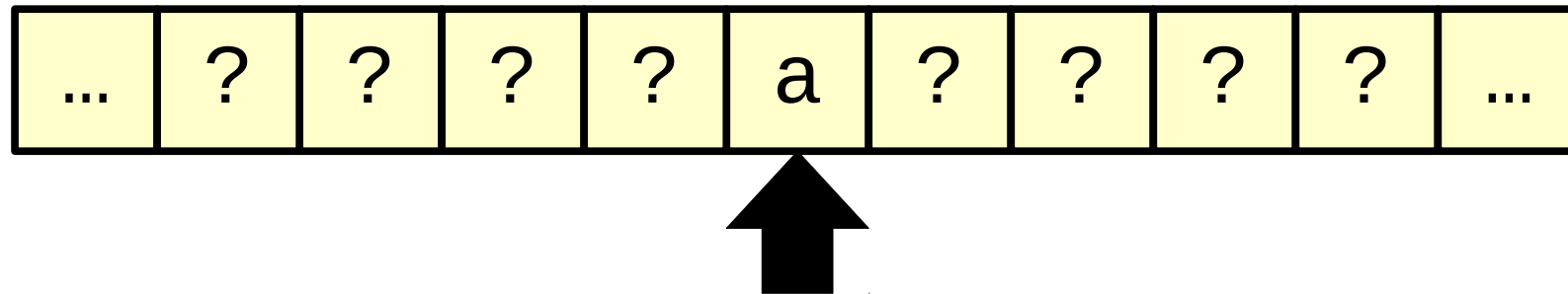
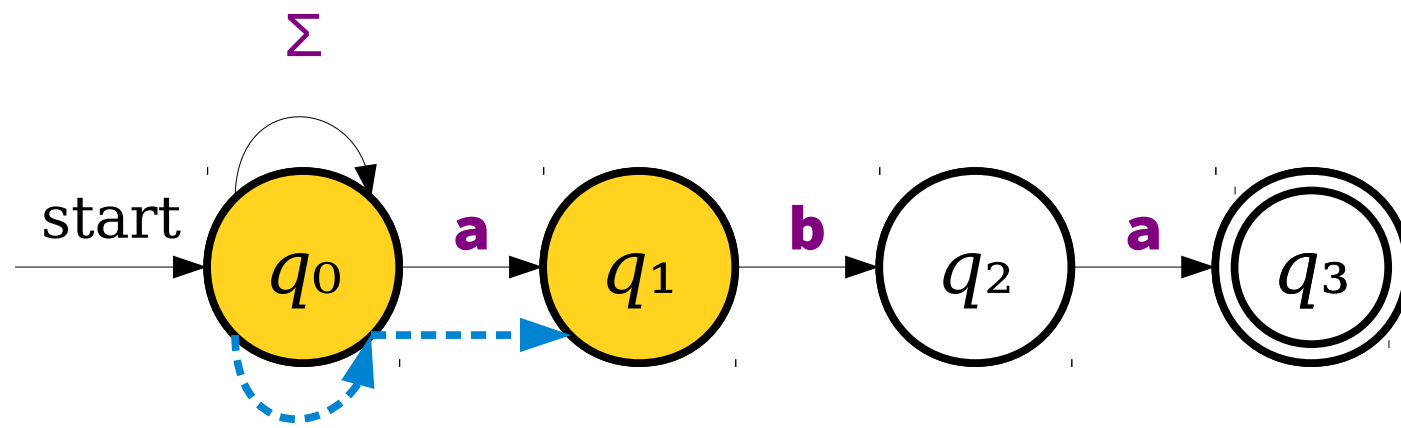


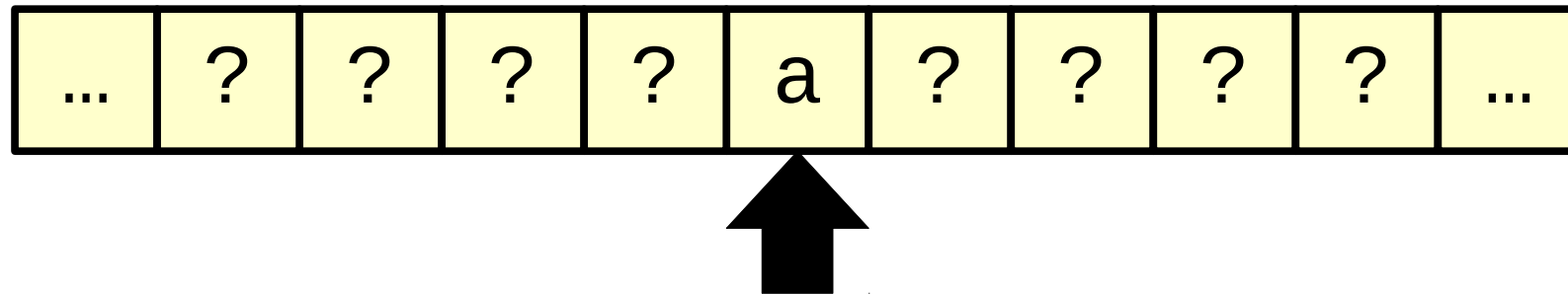
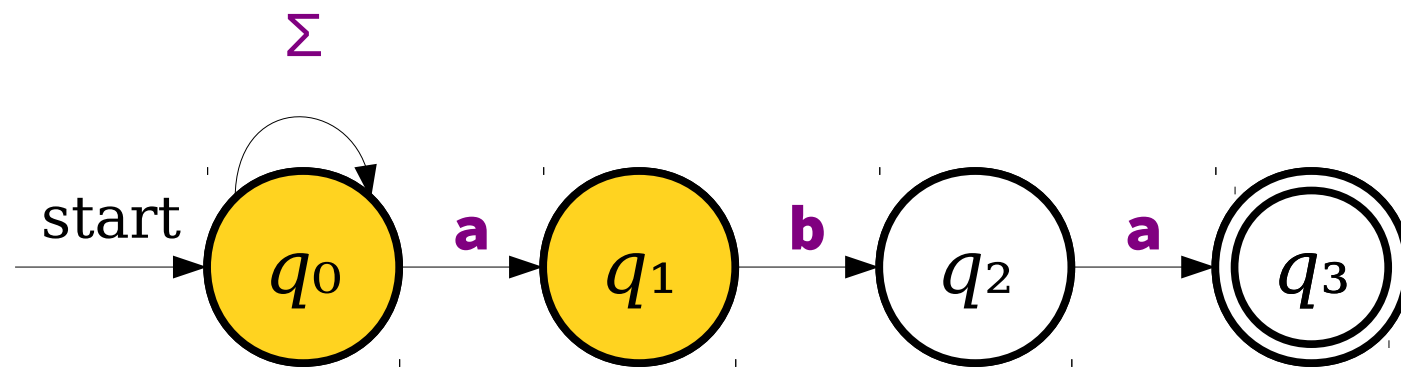


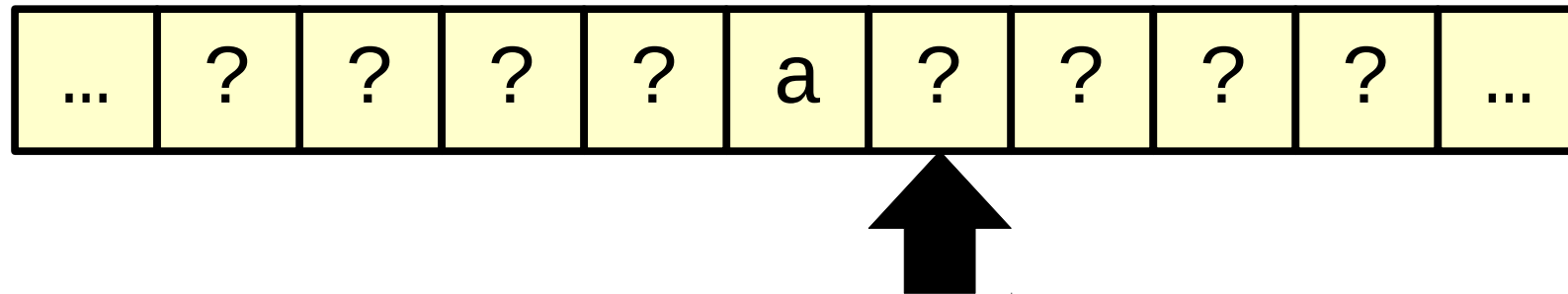
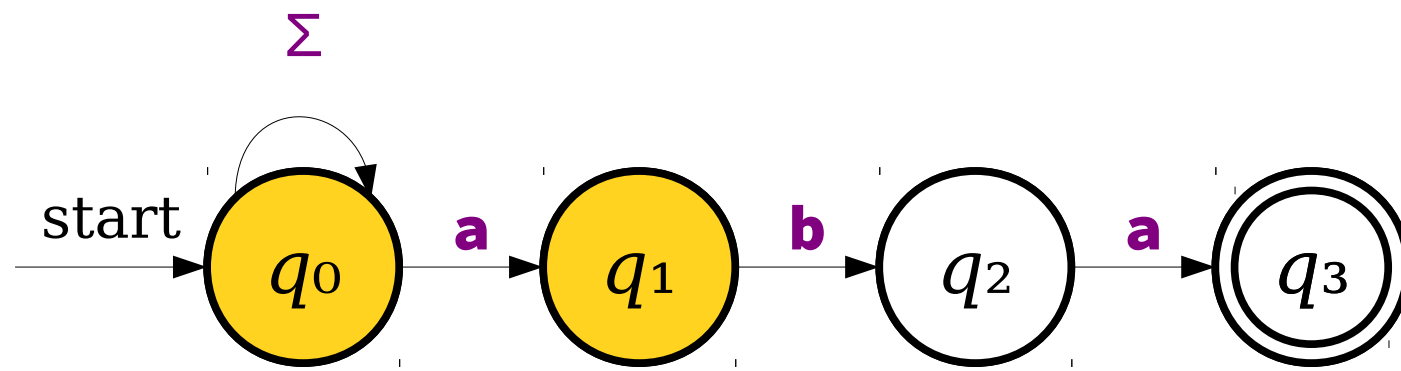


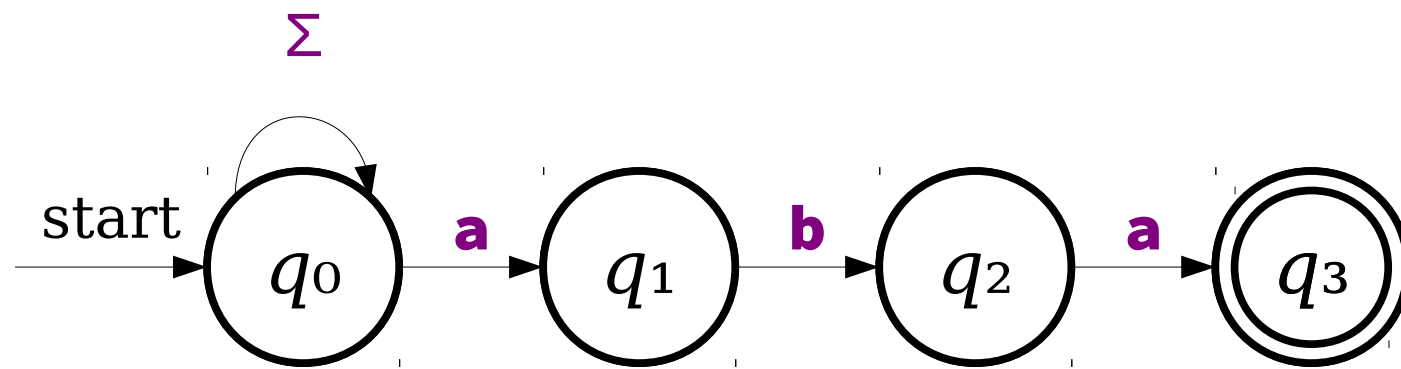




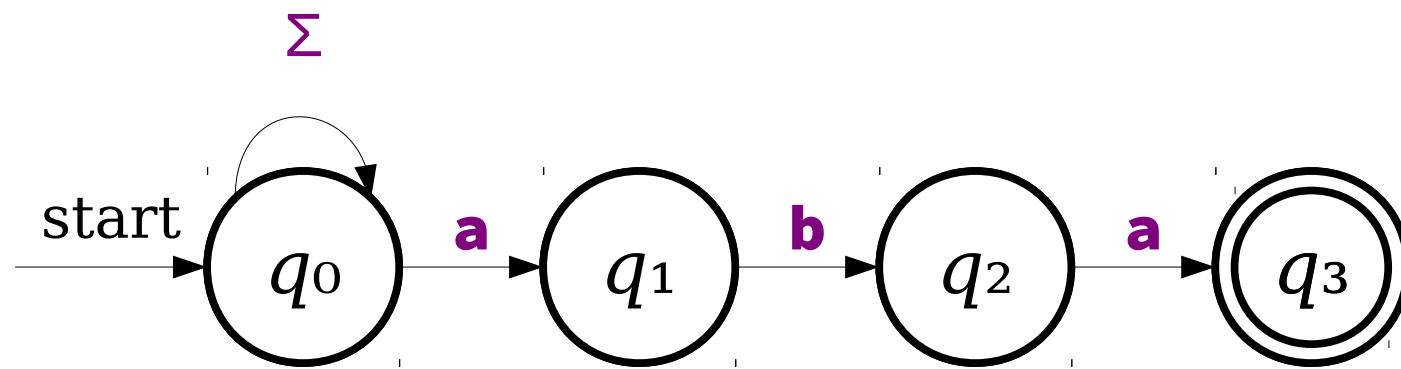




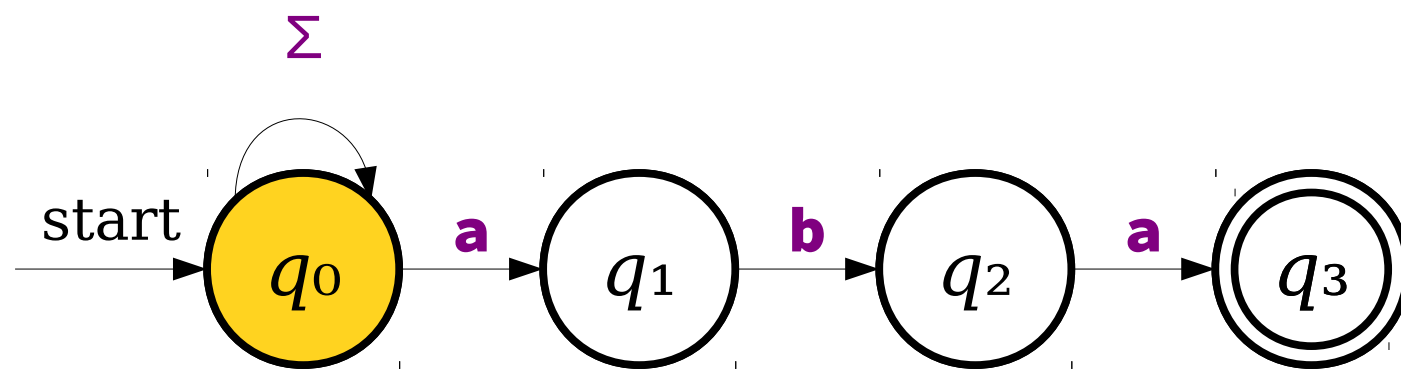




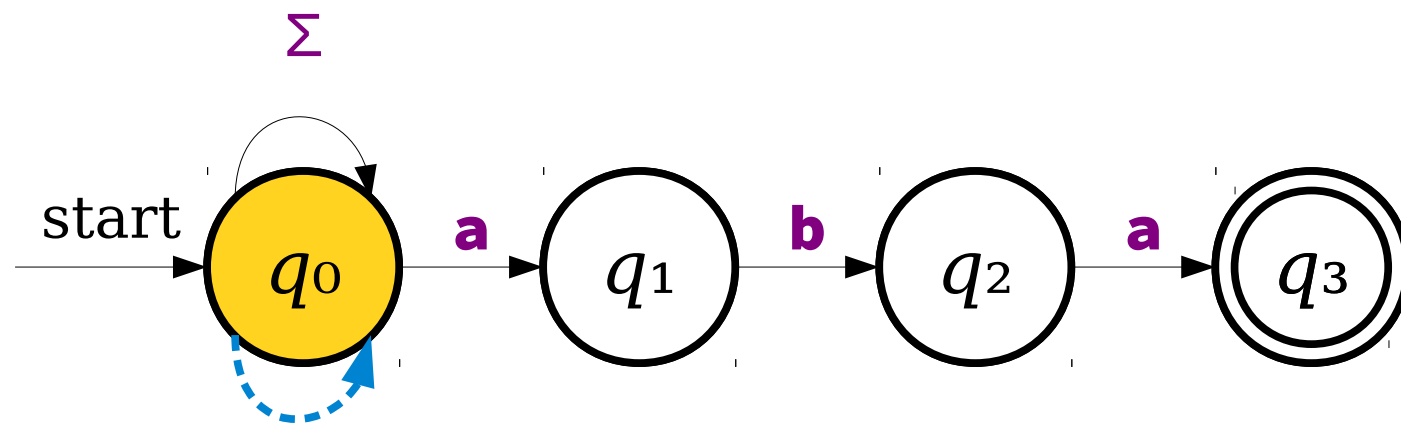
	$a$
$\{q_0\}$	$\{q_0, q_1\}$



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

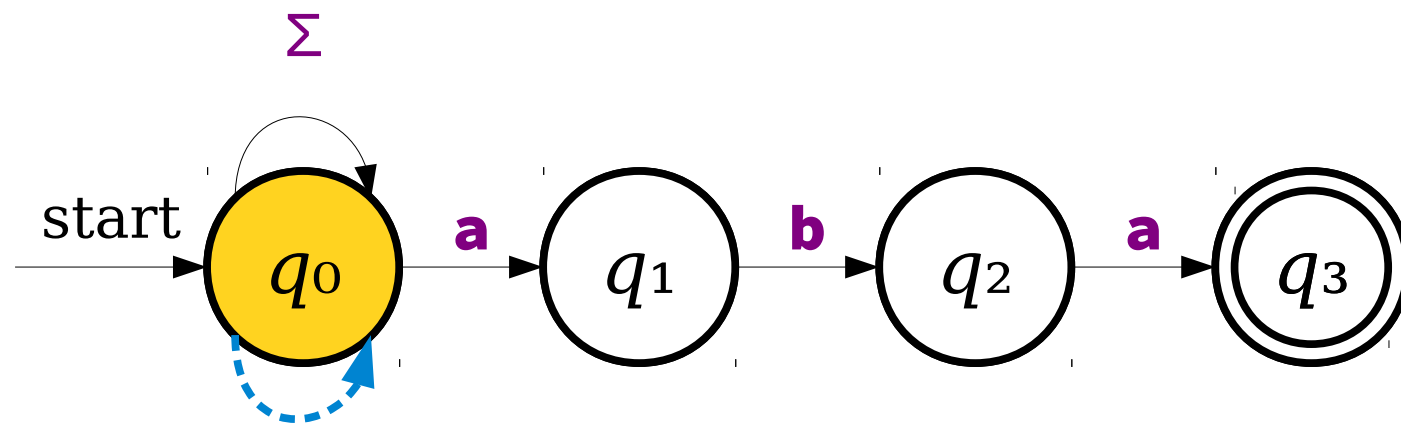


	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	

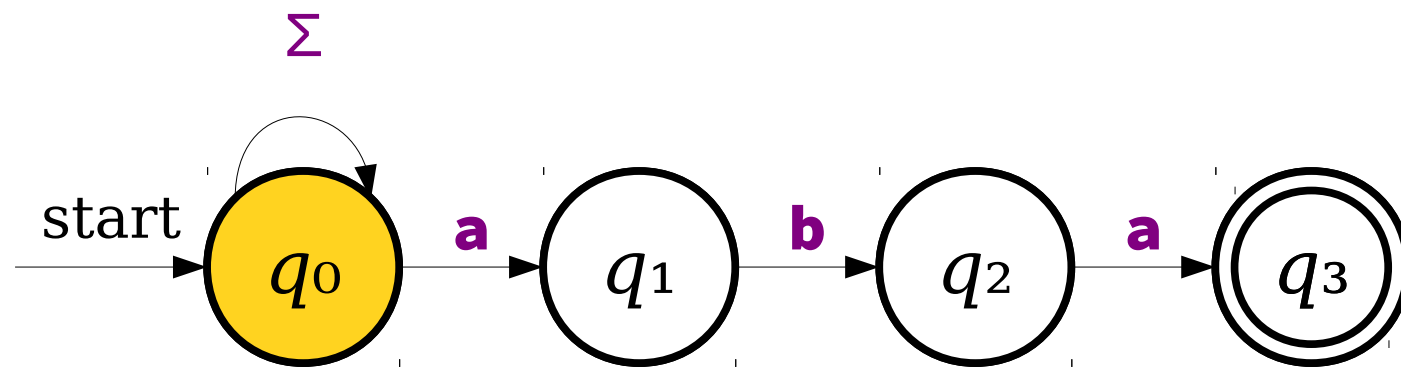


	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	

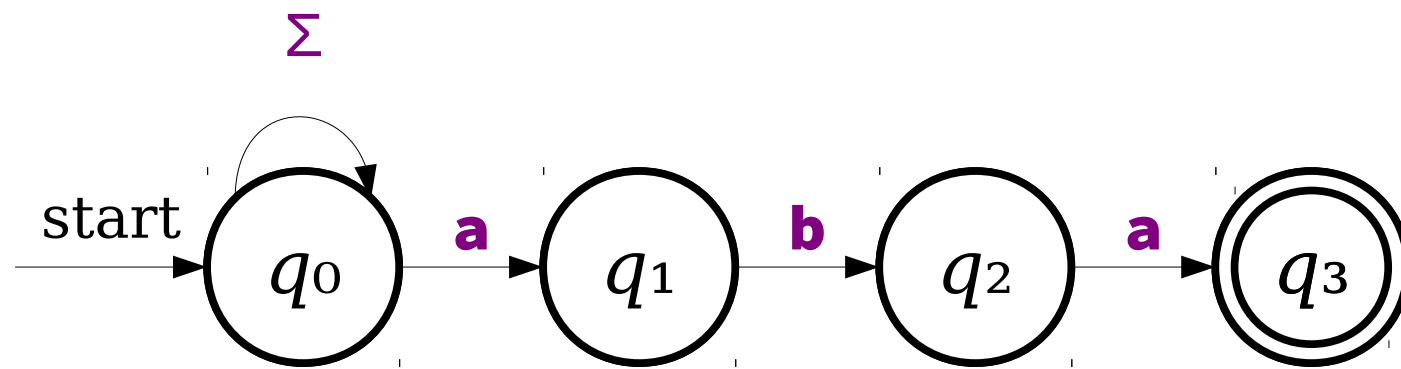




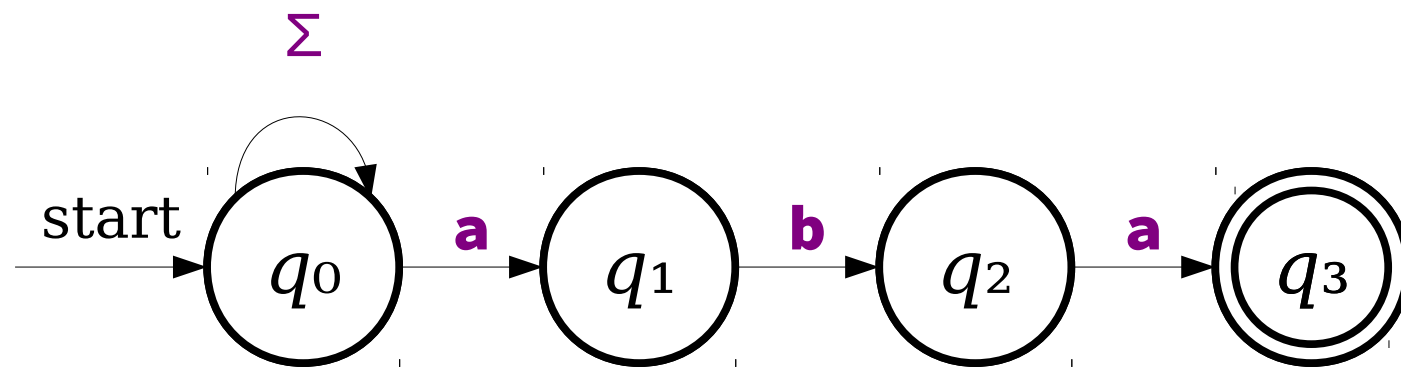
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$



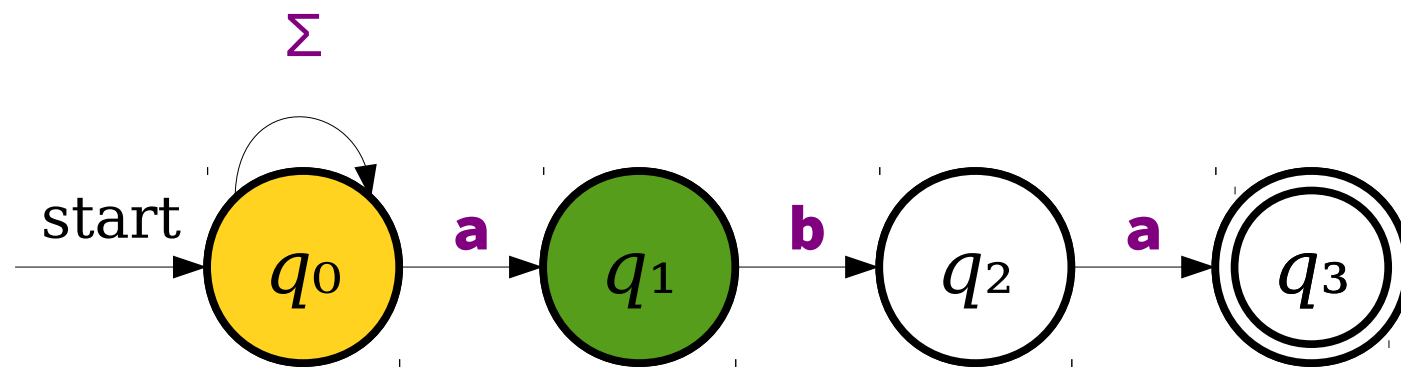
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$



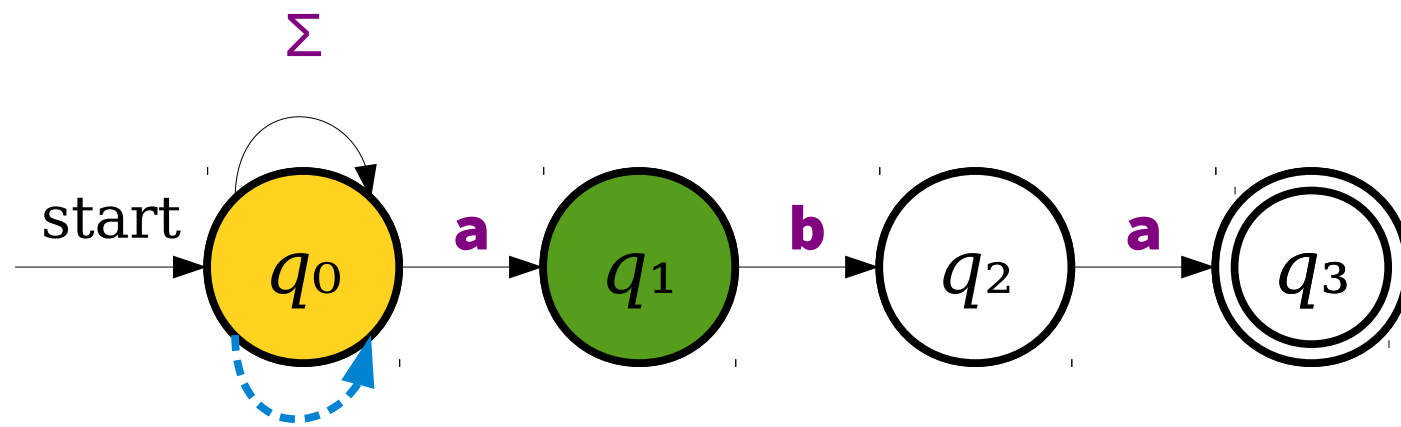
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$



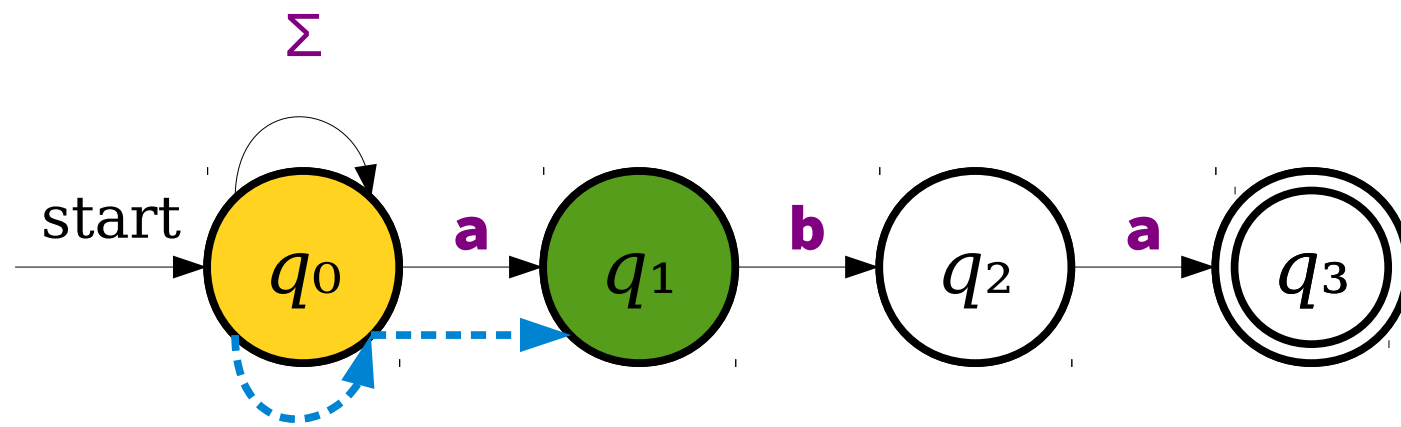
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



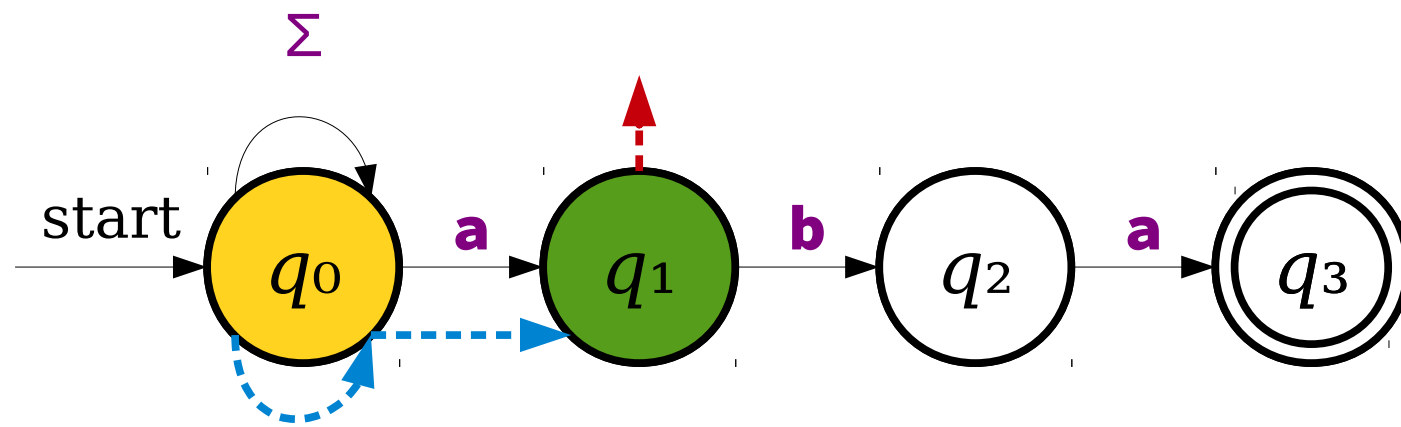
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



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

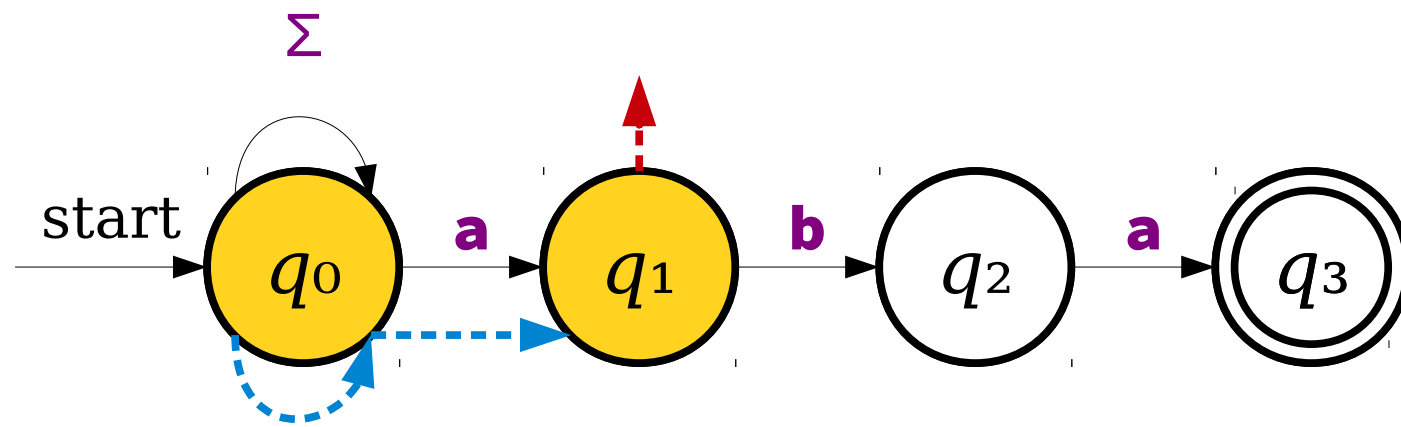


	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		

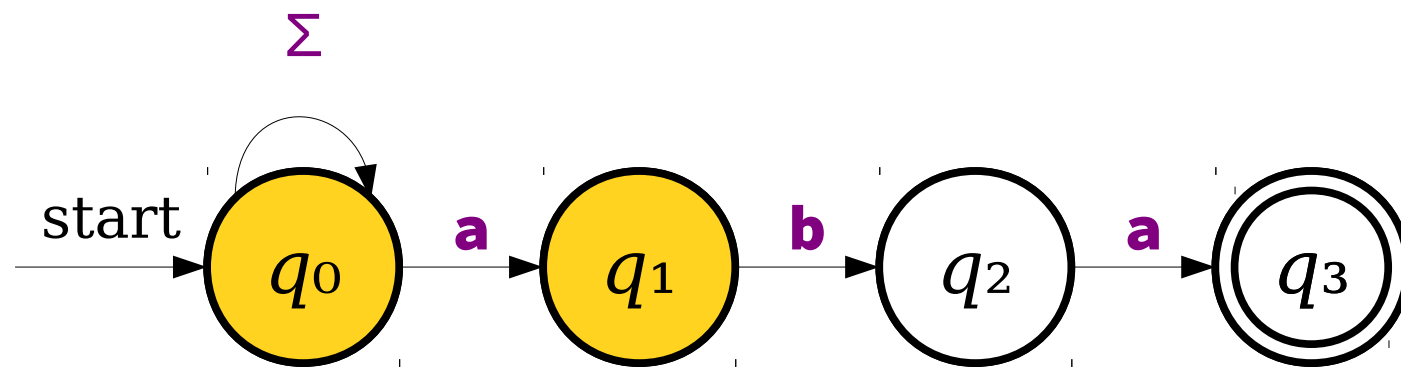


	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		

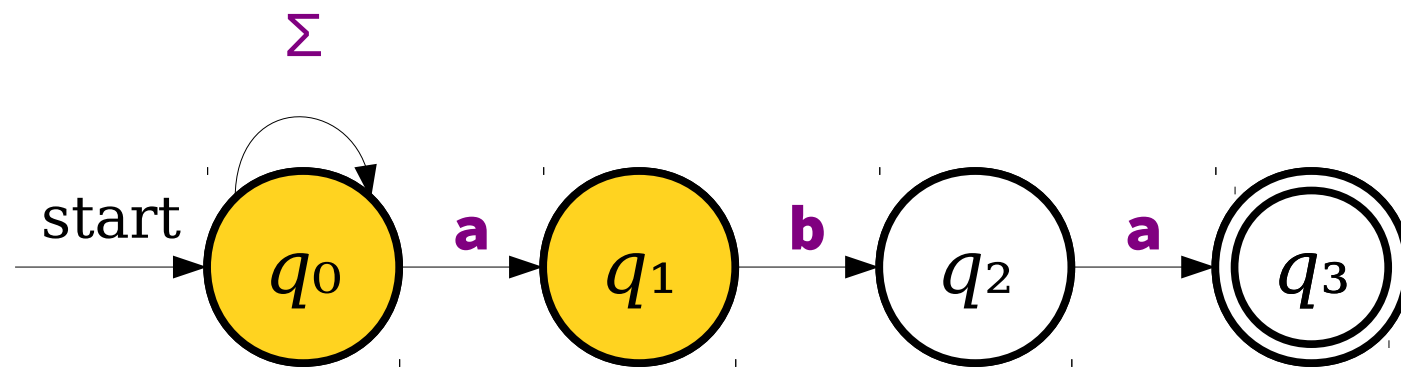




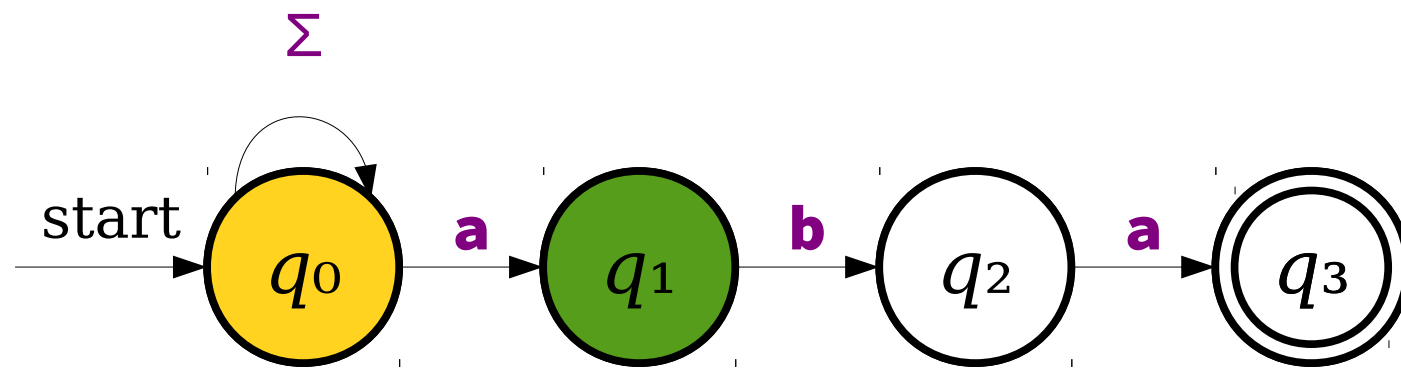
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



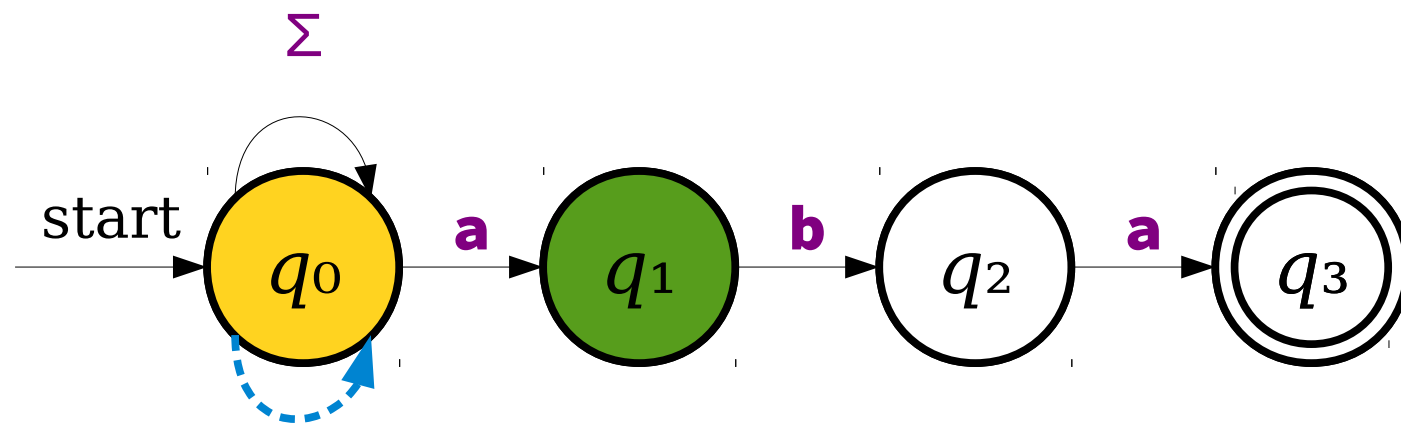
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$		



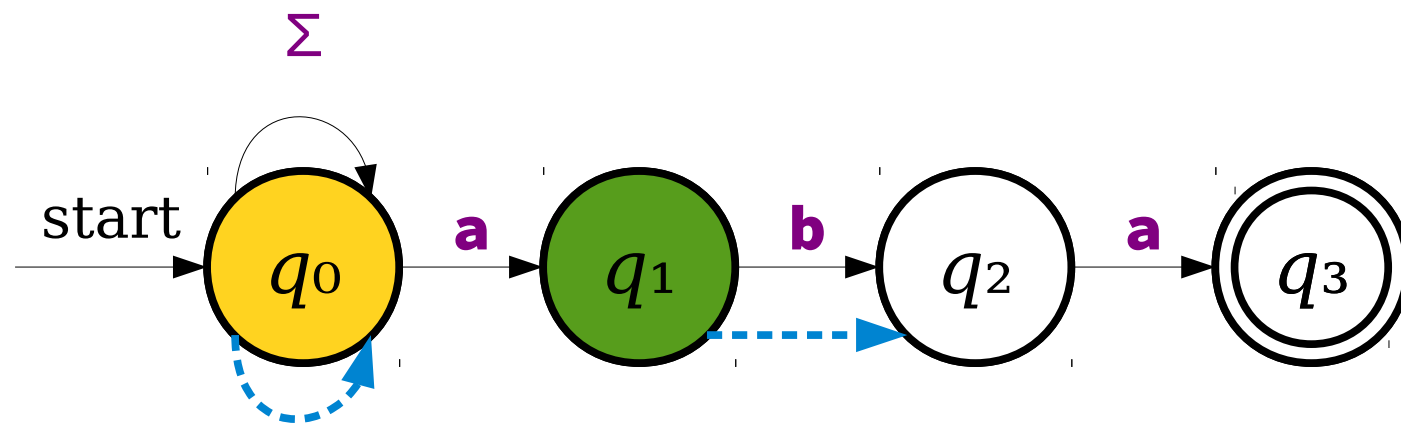
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	



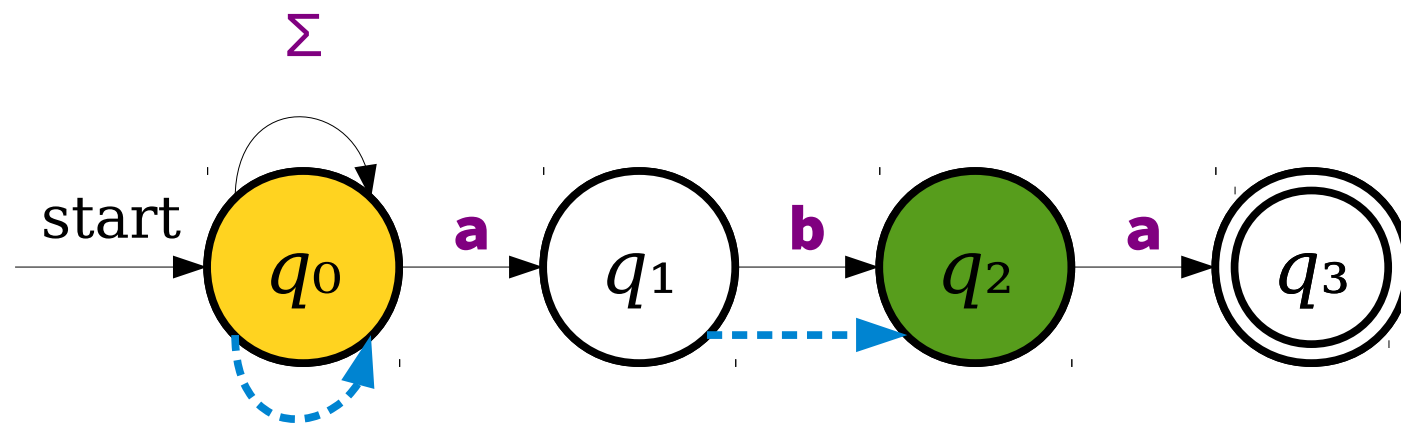
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	



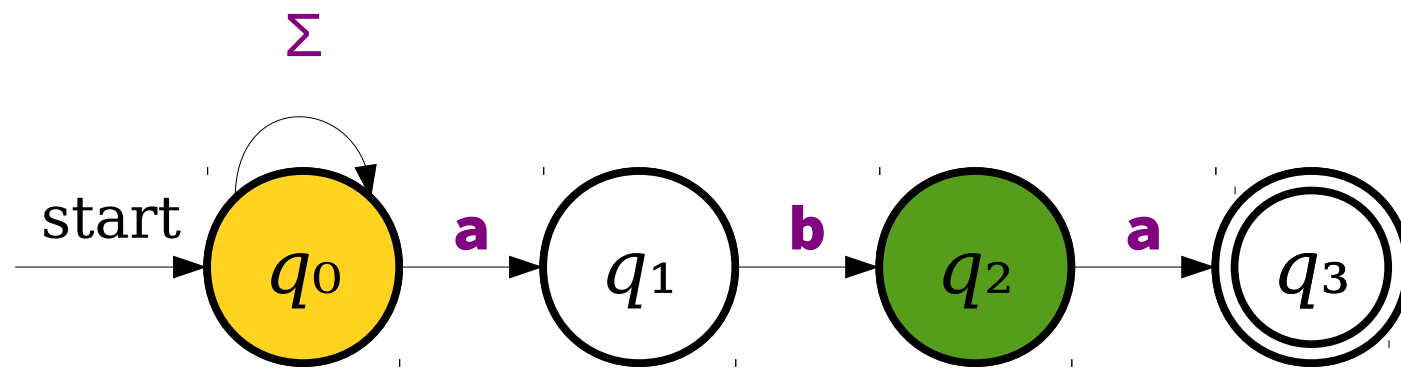
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	



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

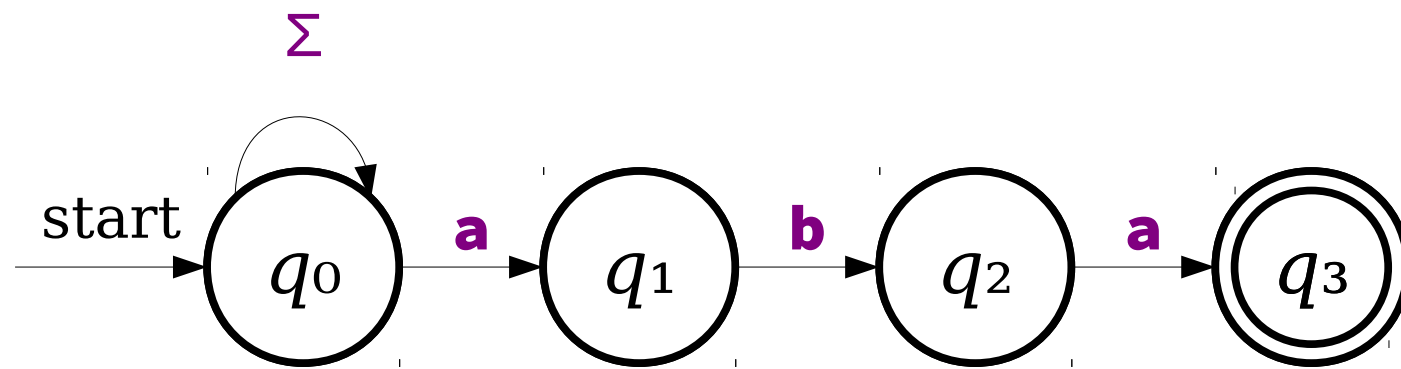


	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	

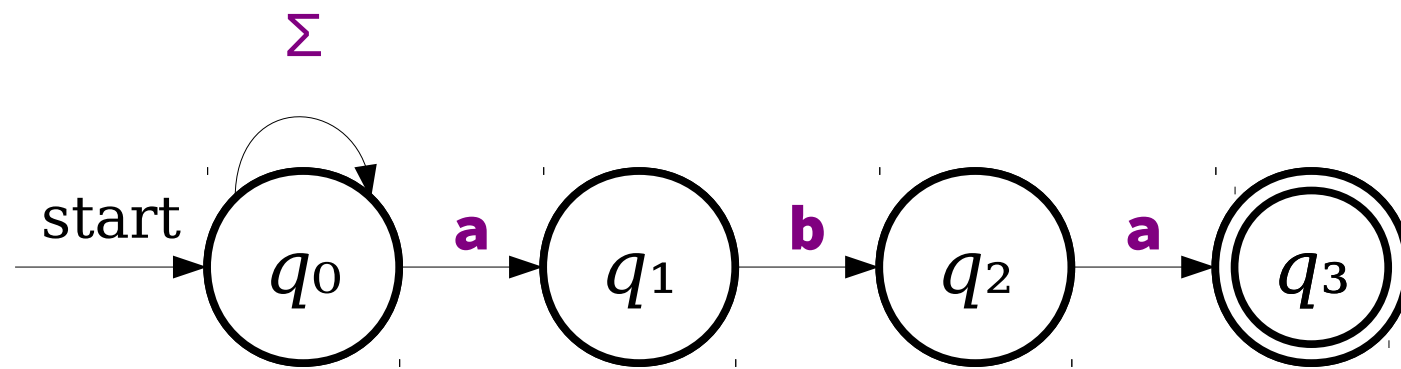


	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

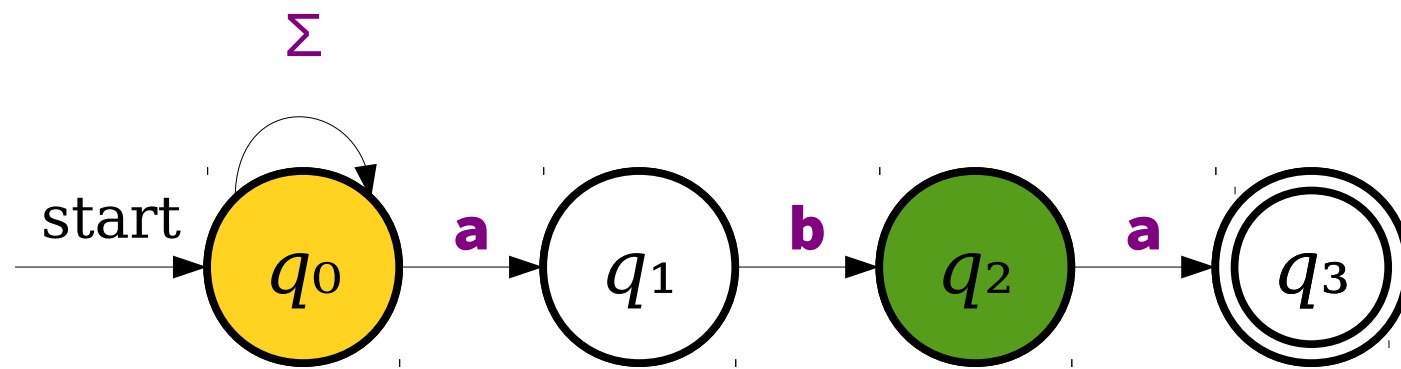




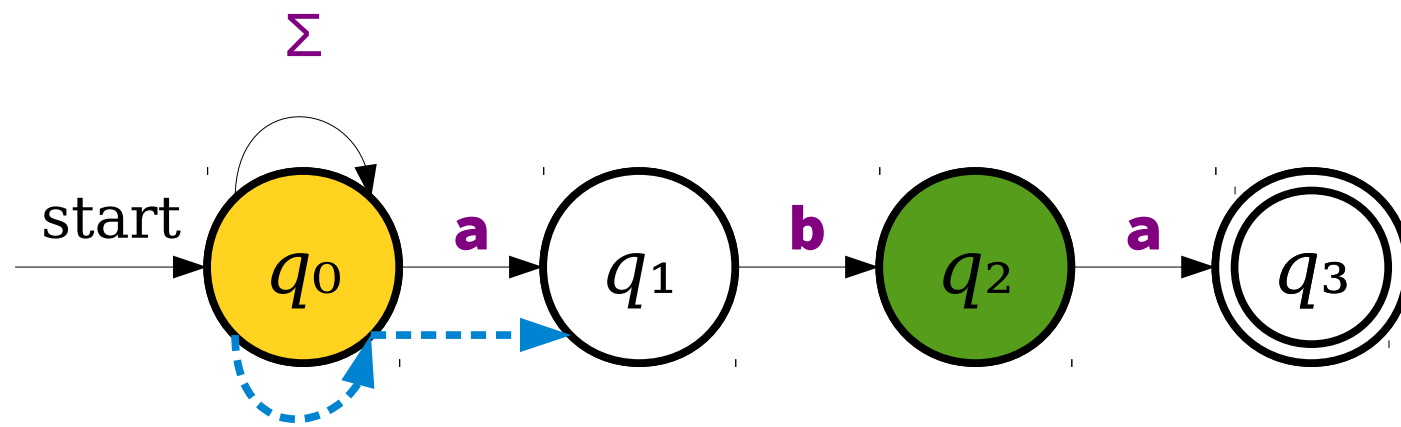
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



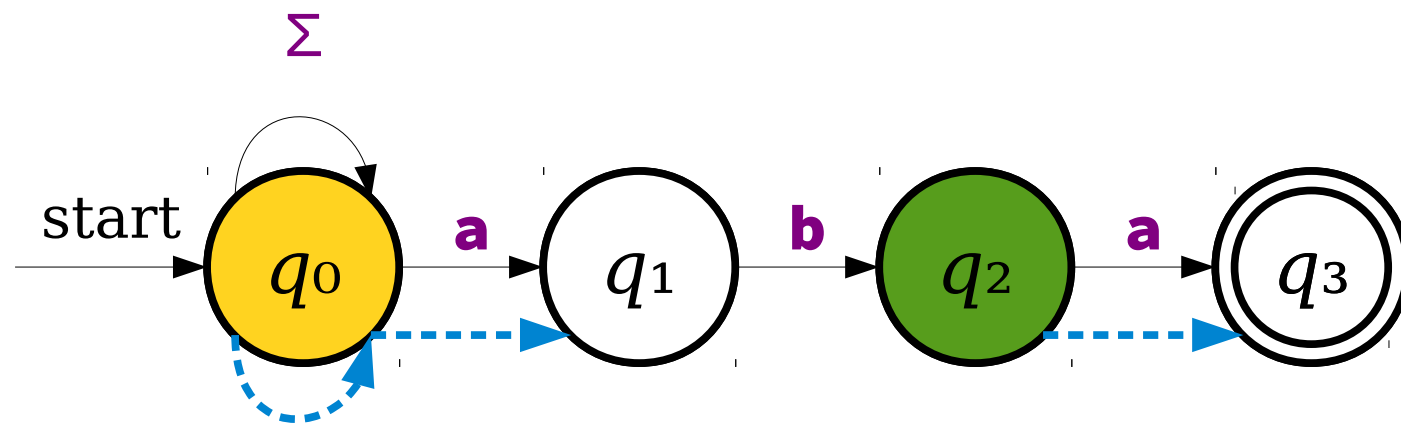
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		



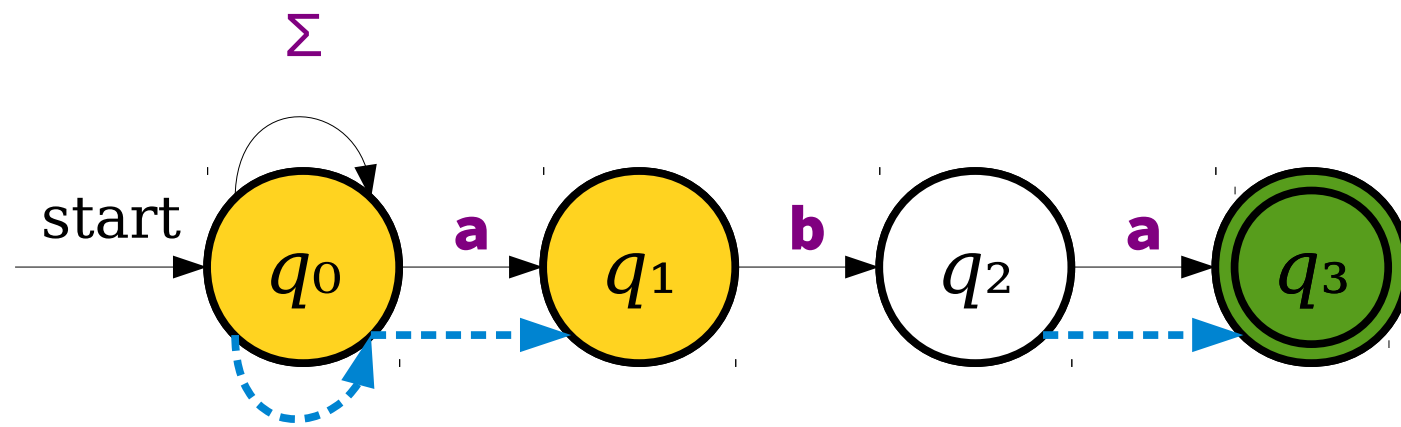
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		



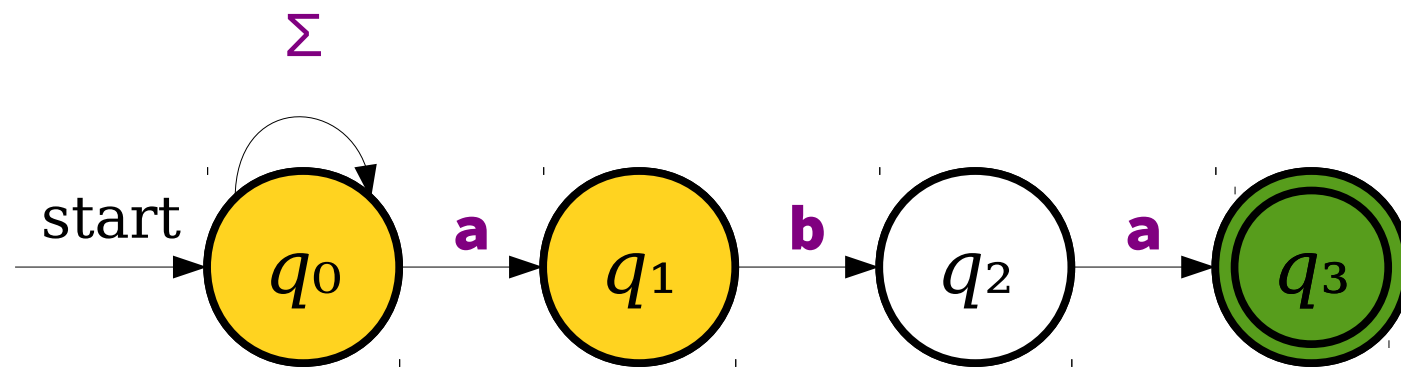
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		



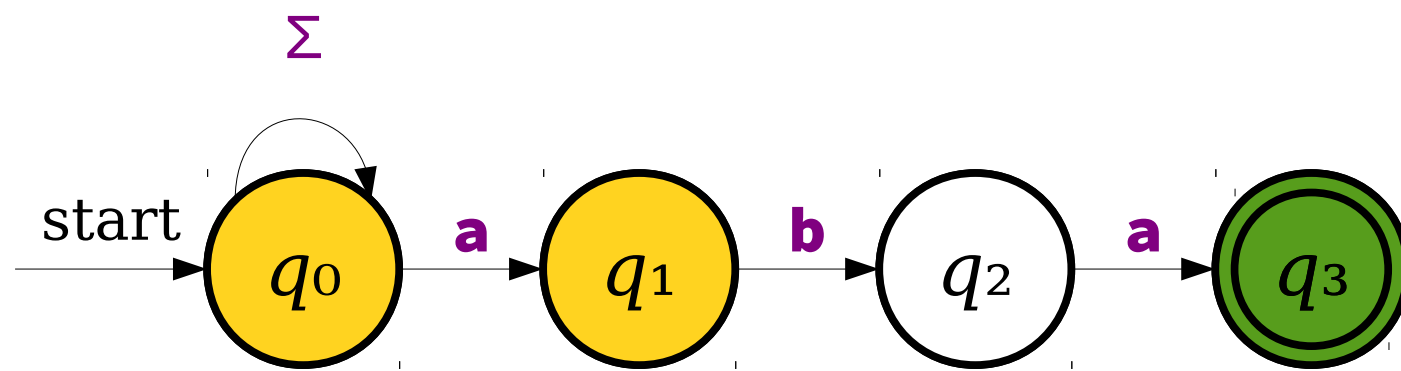
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		



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

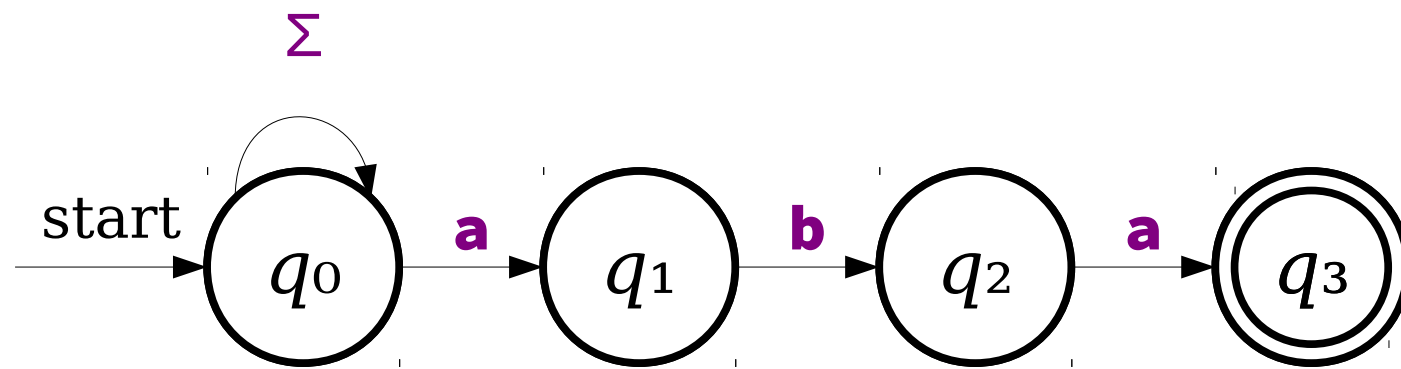


	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$		

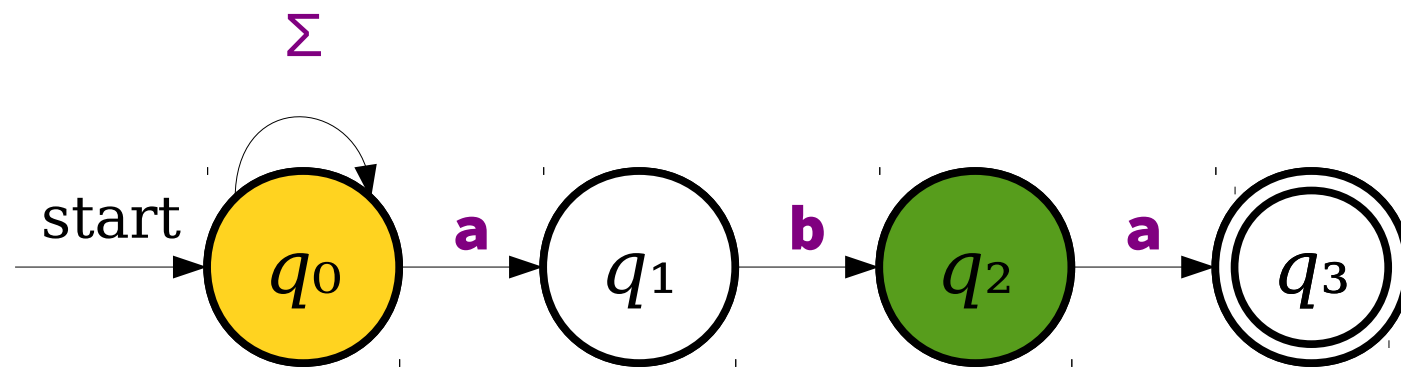


	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	

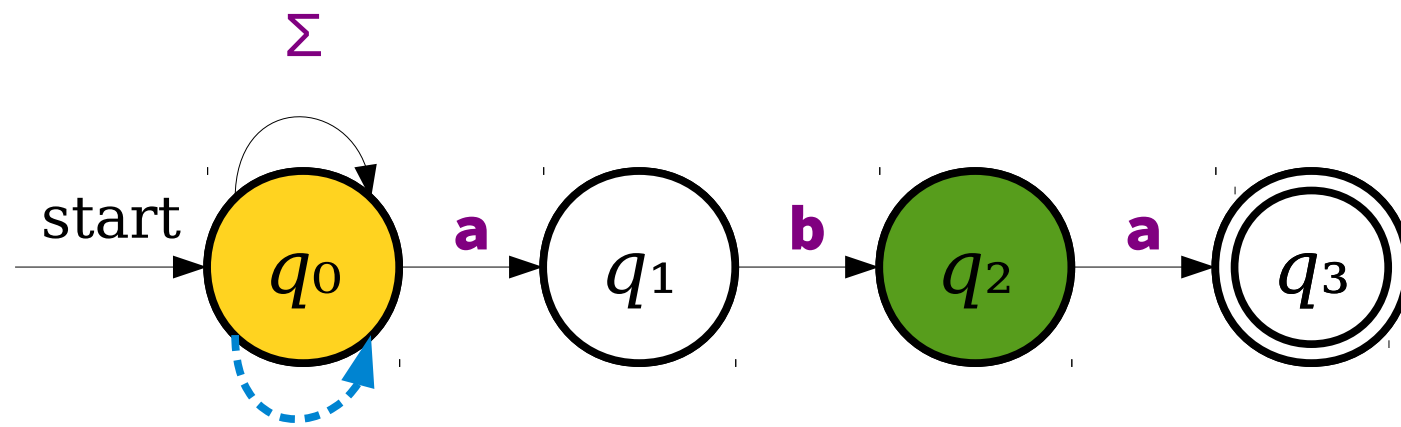




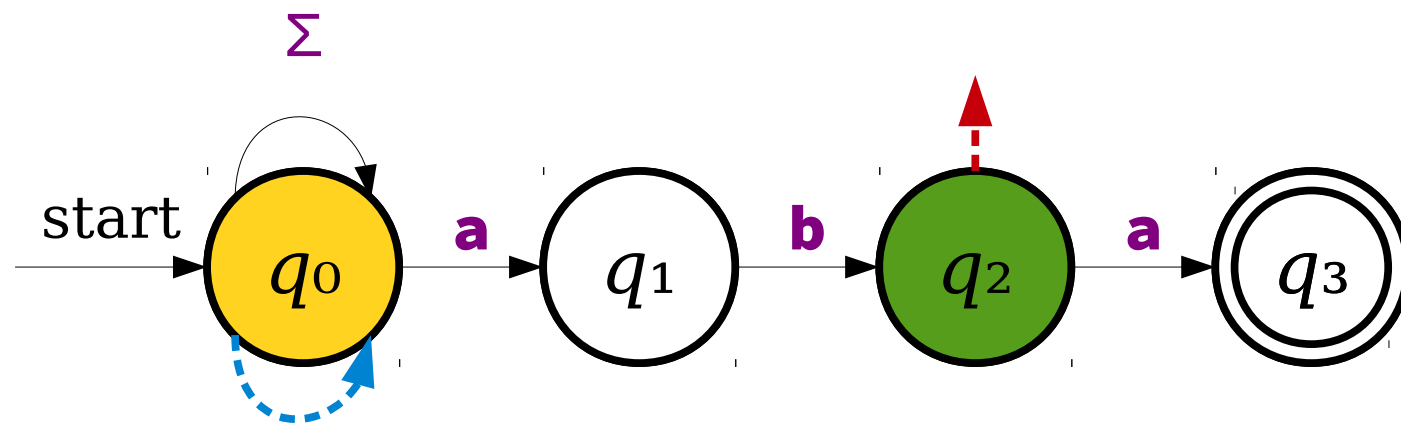
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



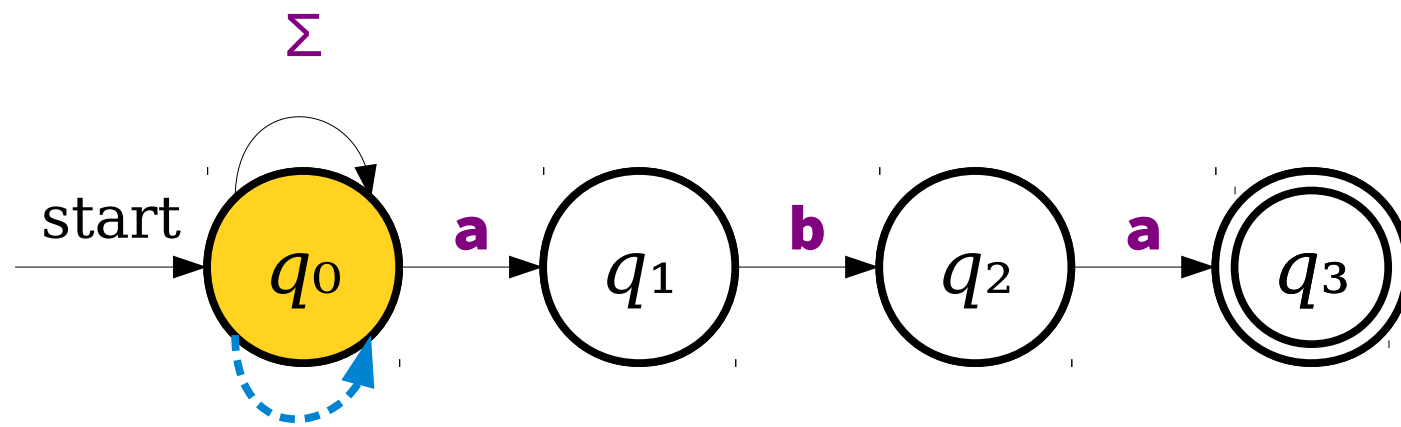
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



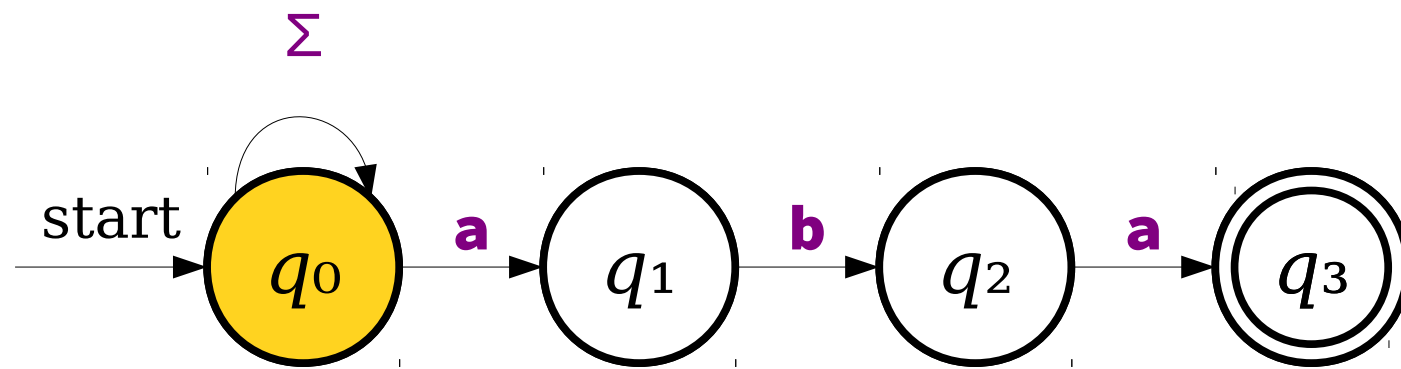
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



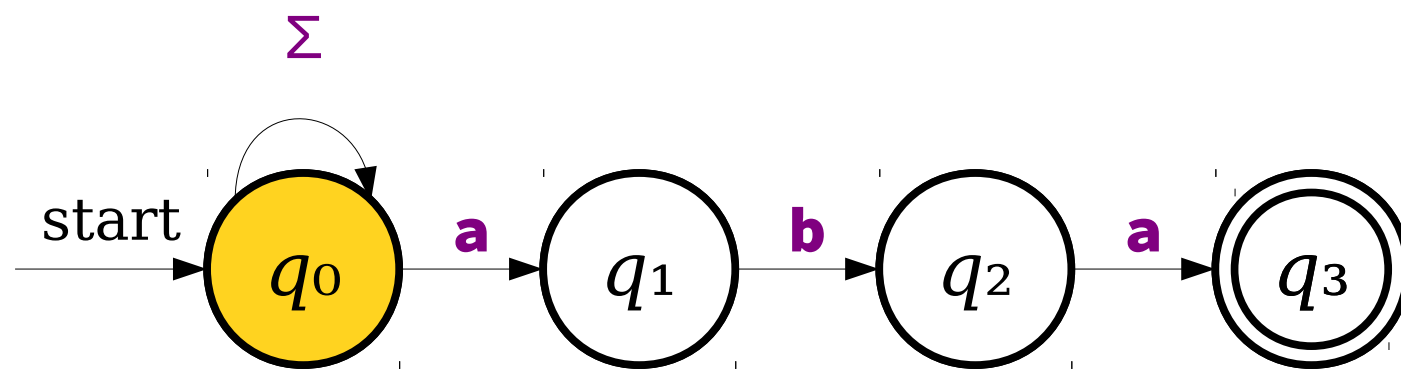
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



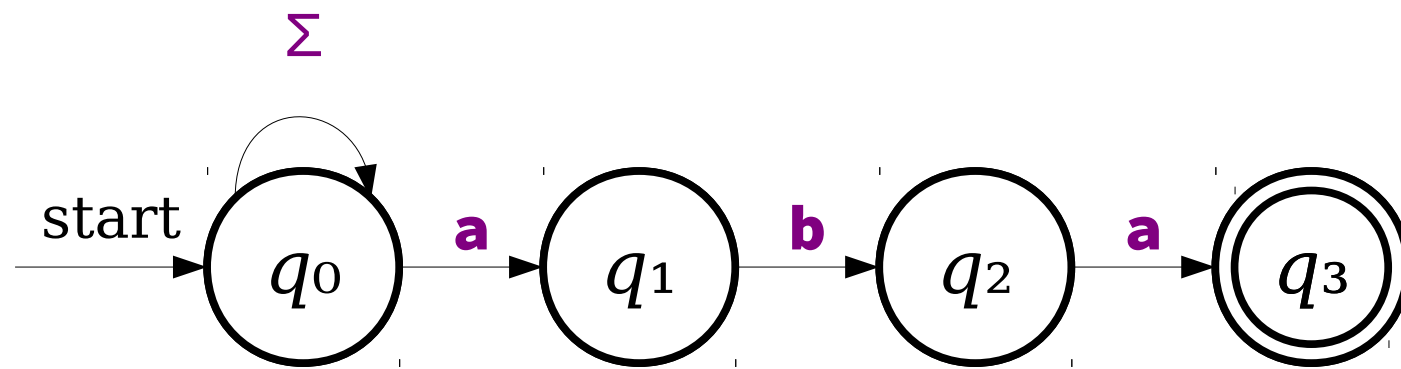
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	



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

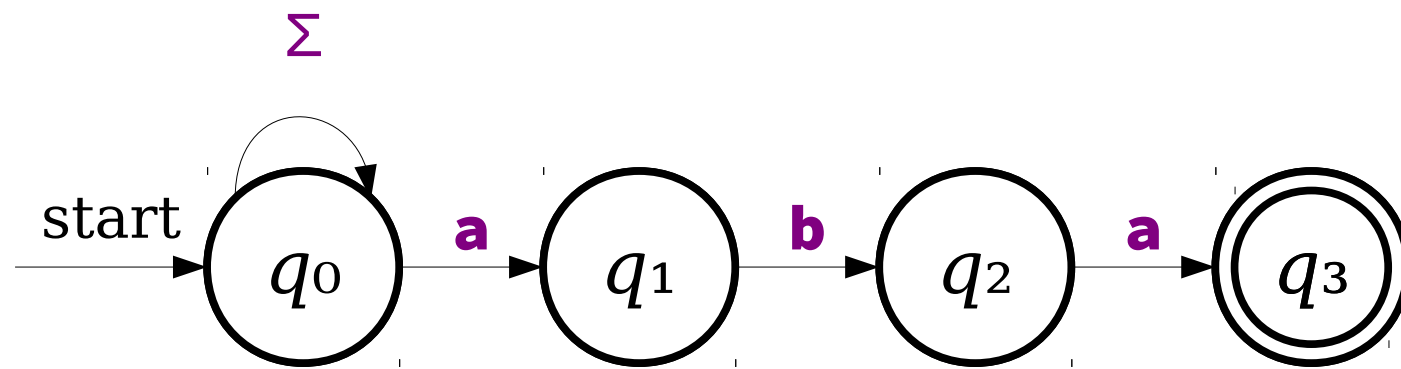


	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$

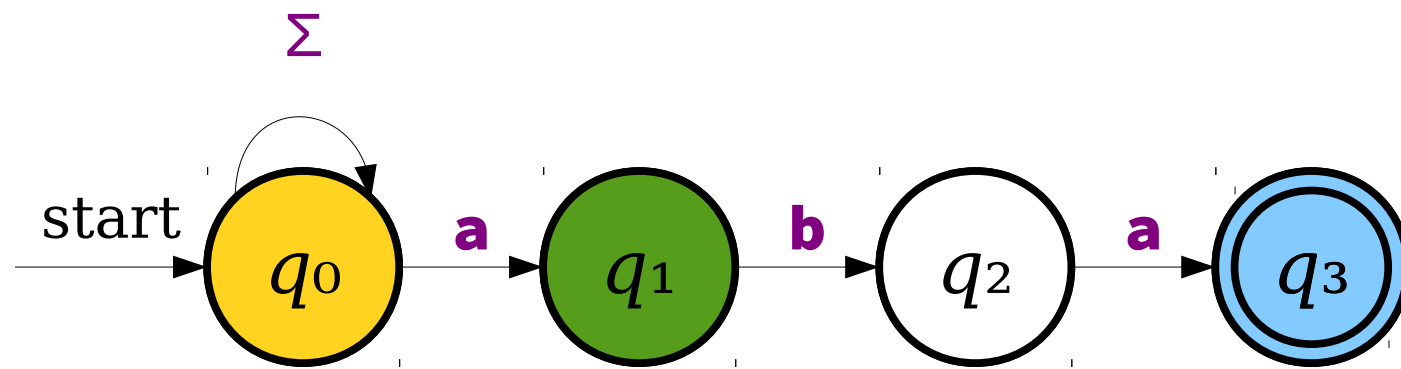


	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$

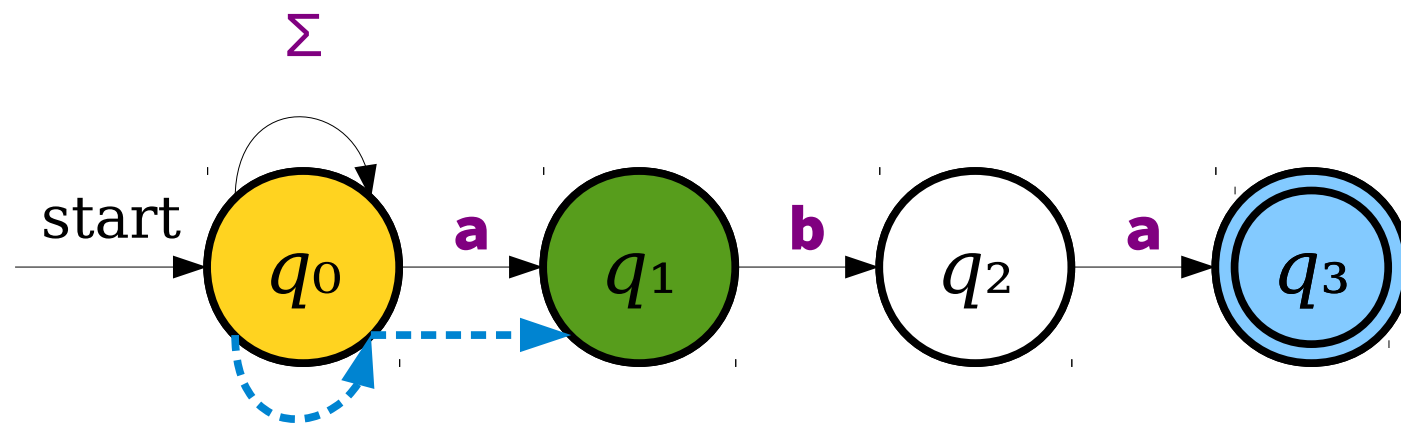




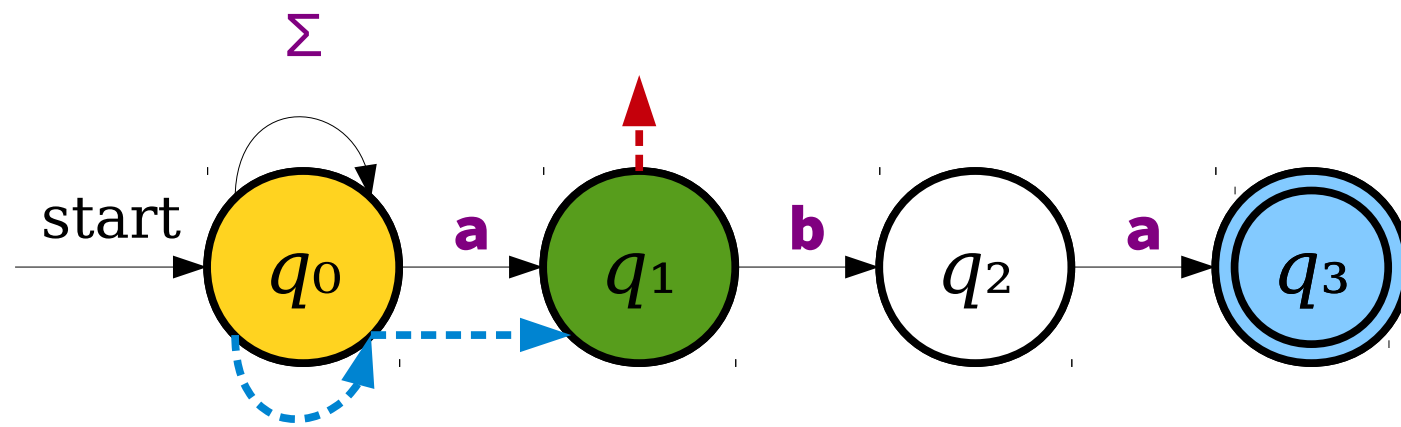
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



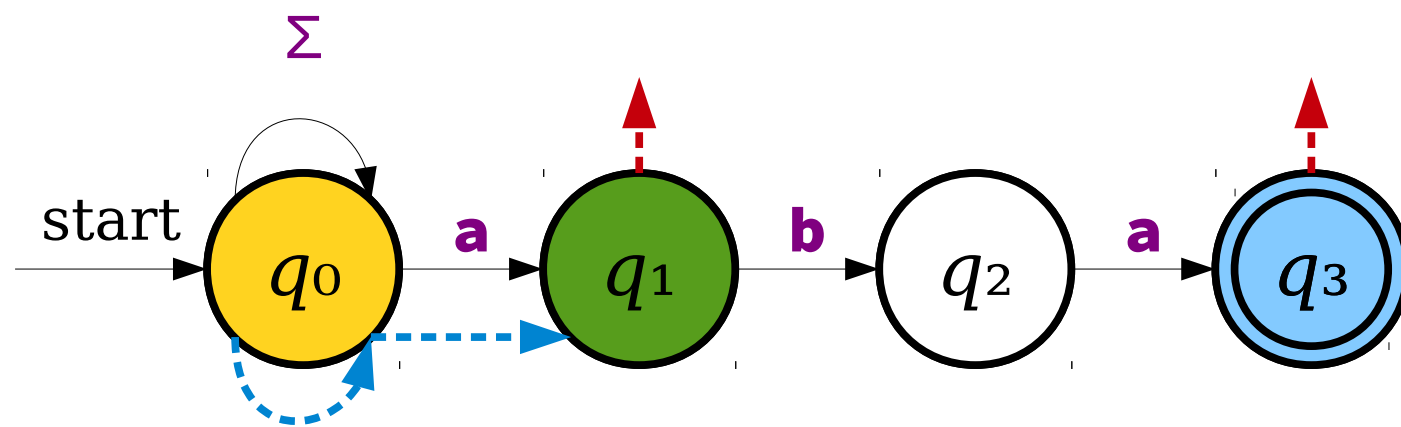
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



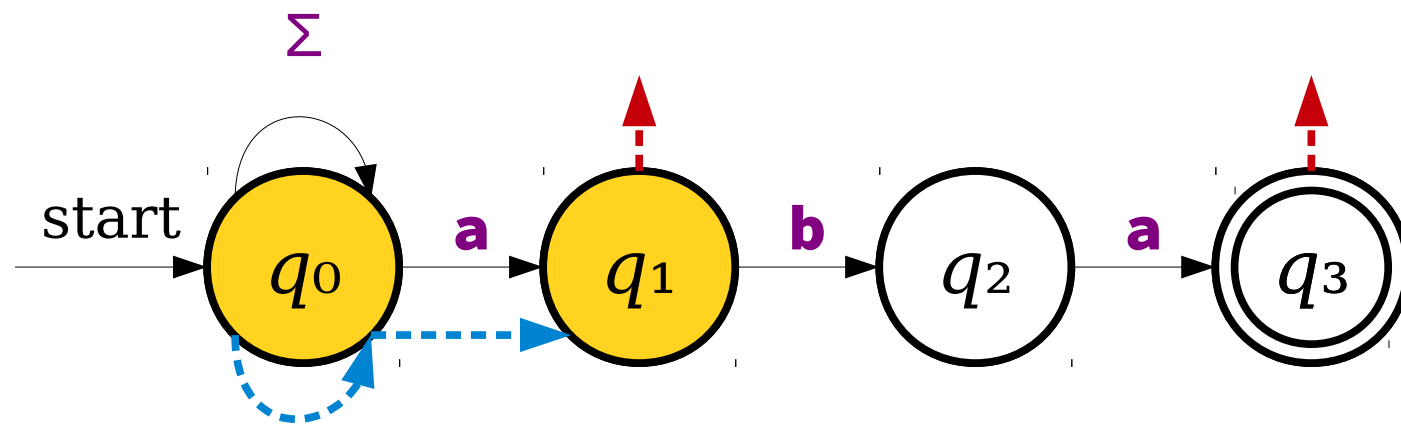
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



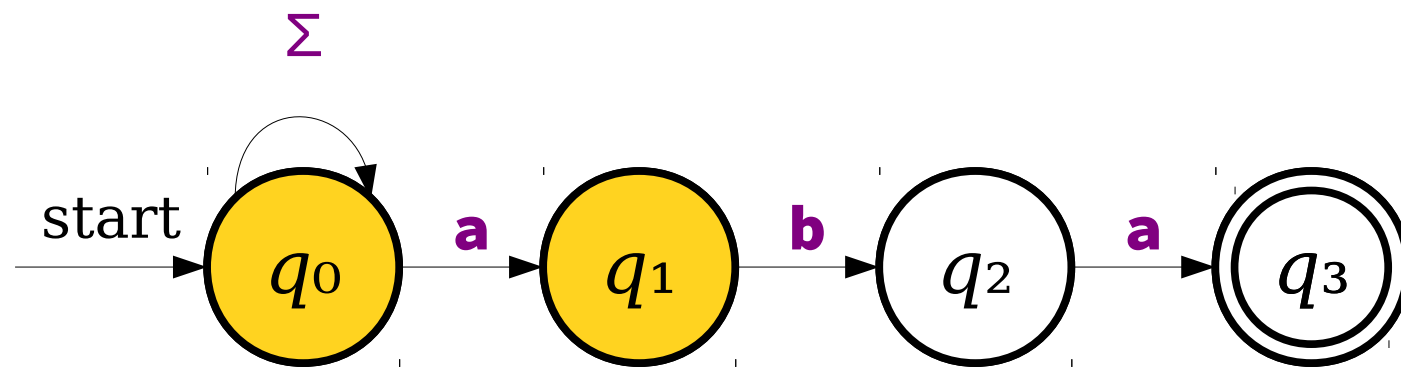
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



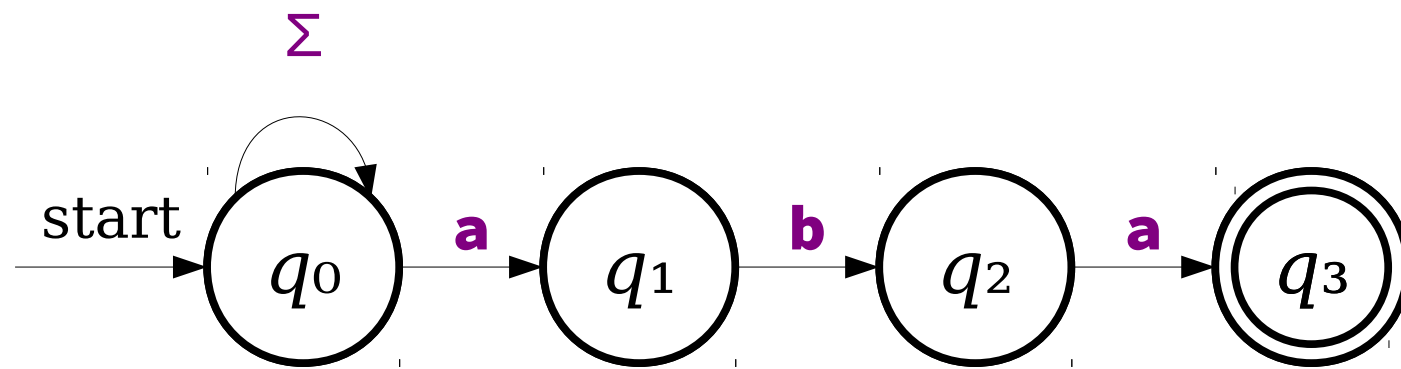
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$		



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

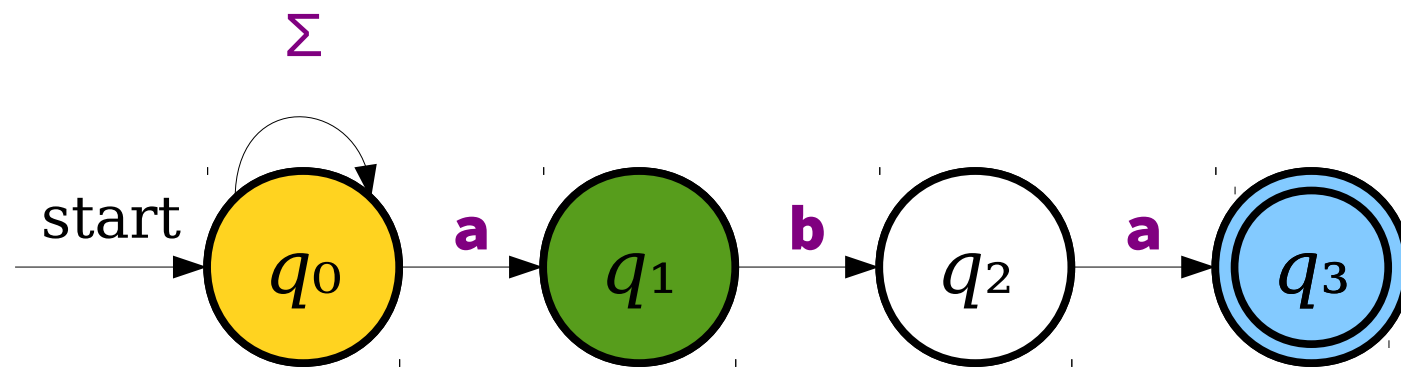


	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	

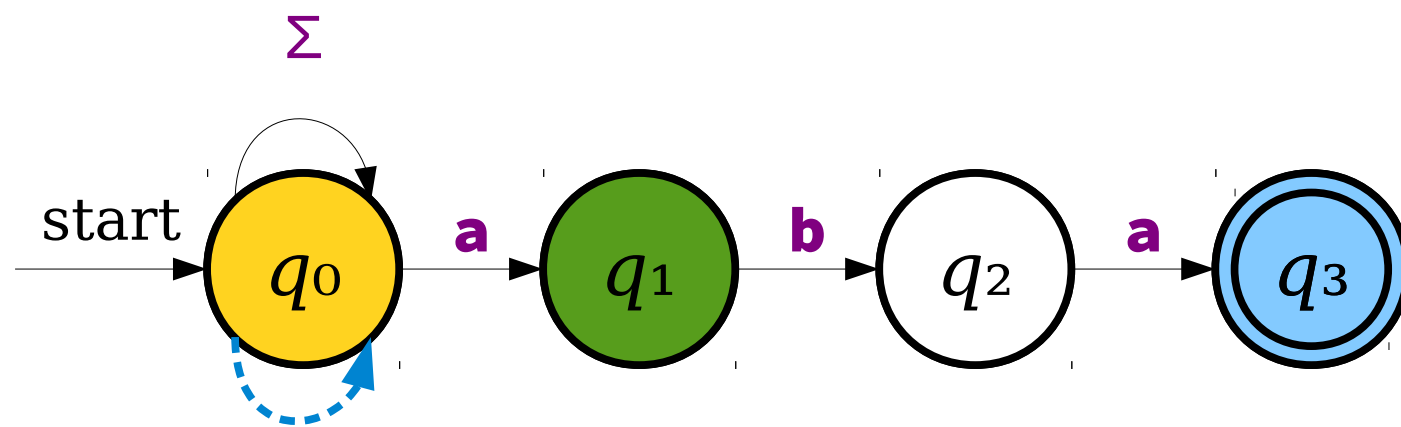


	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	

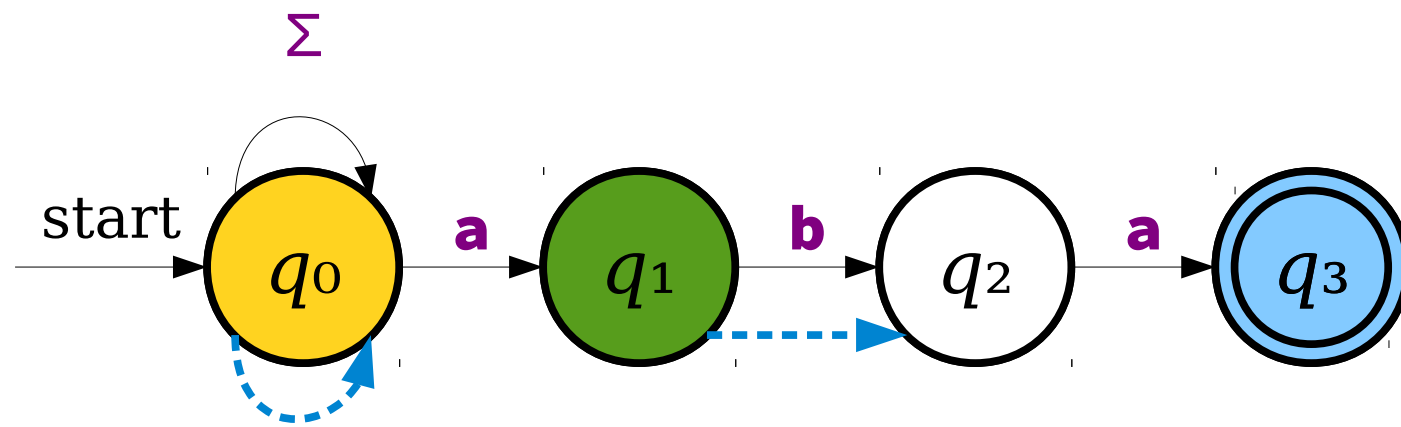




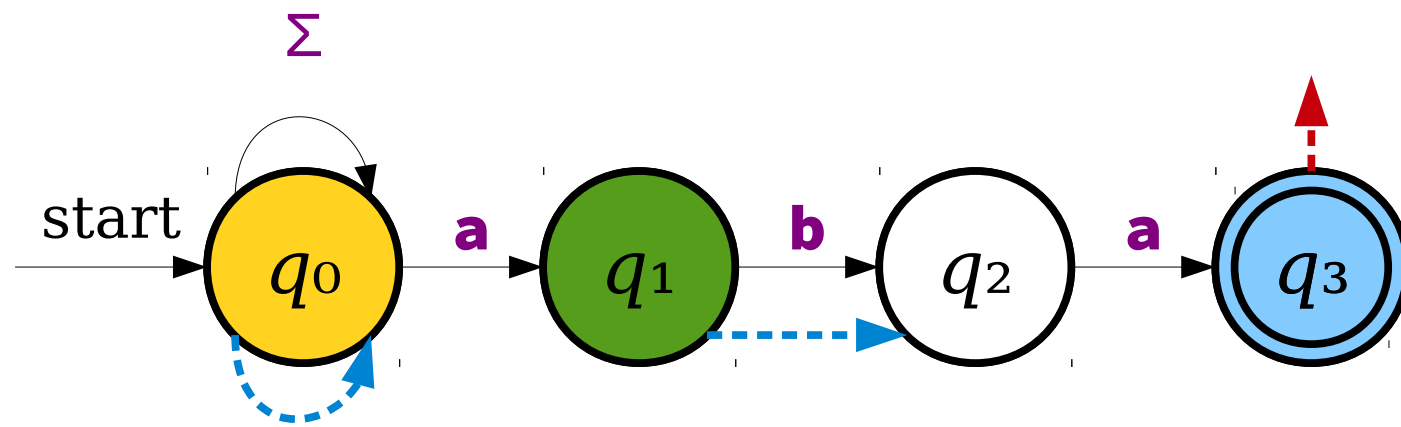
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



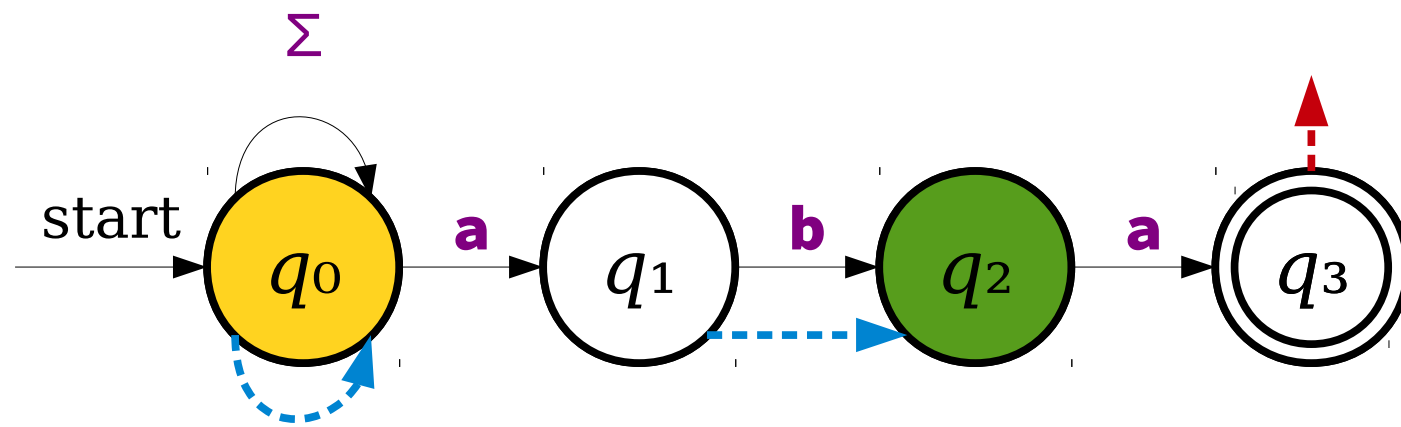
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



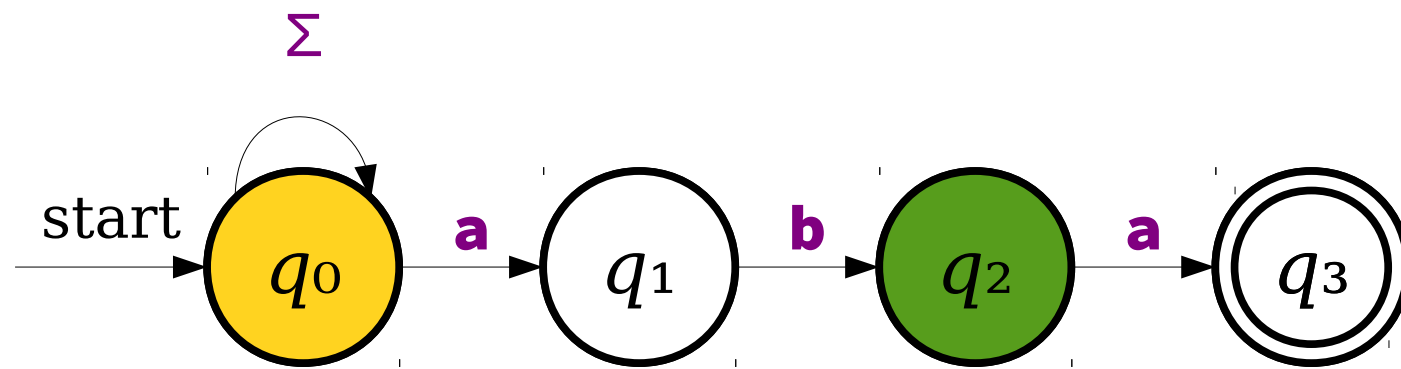
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



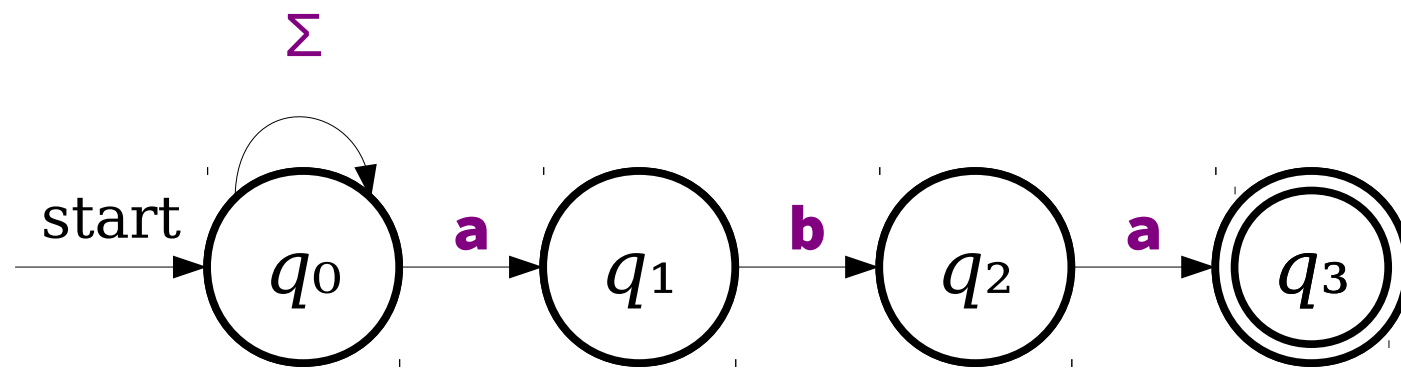
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



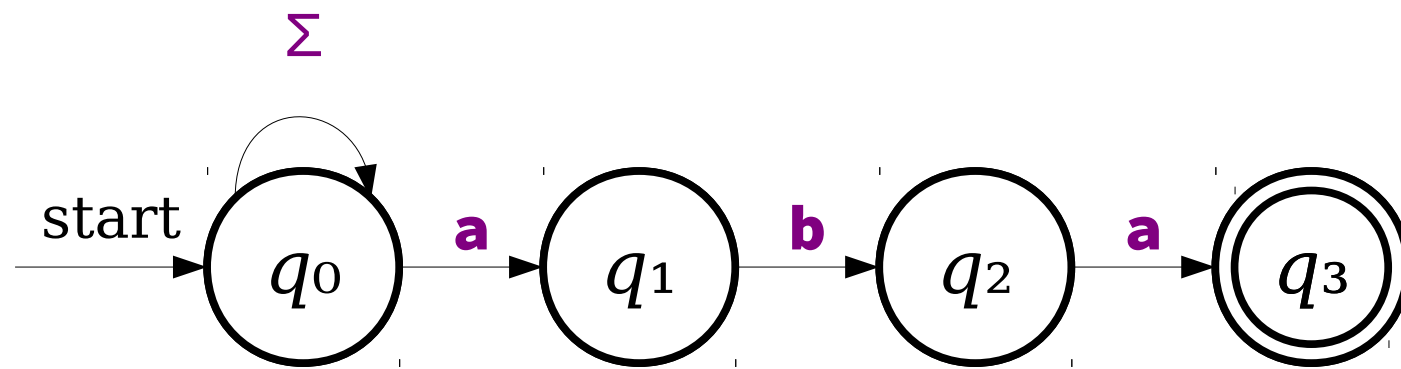
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	



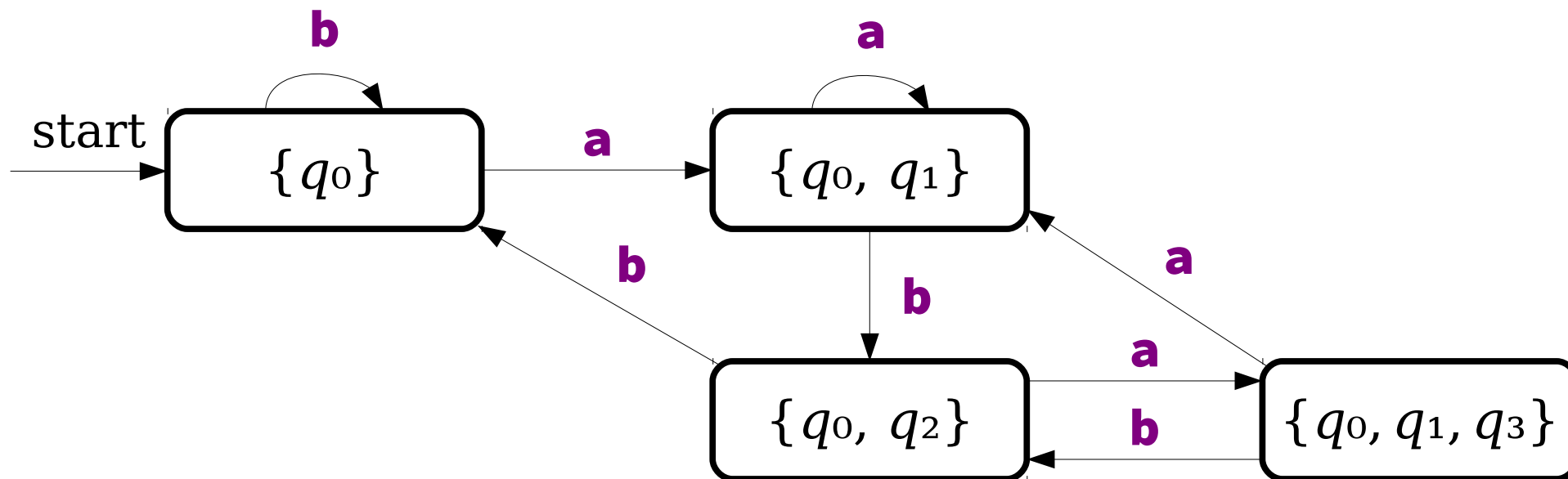
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



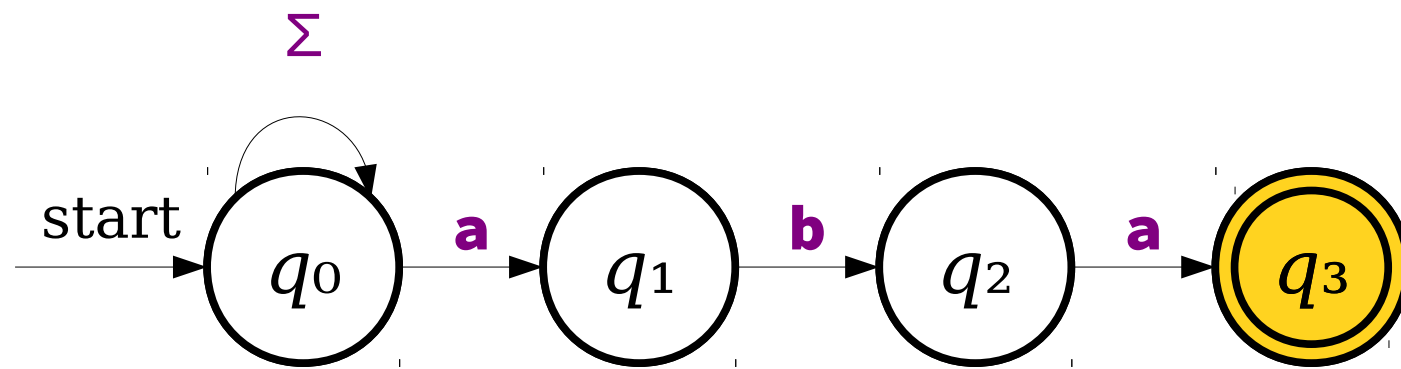
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



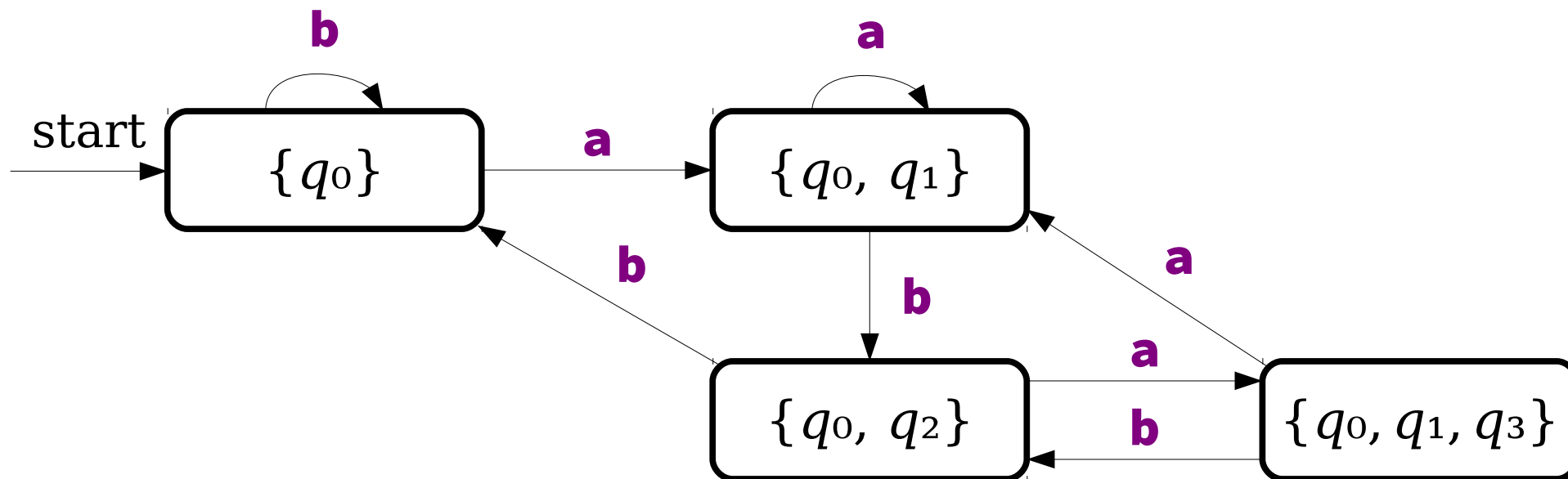
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

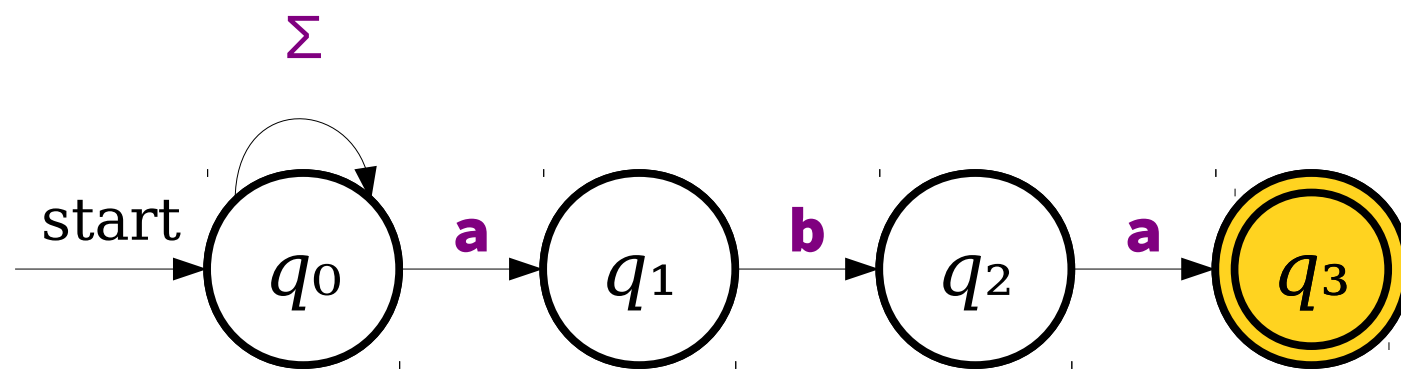




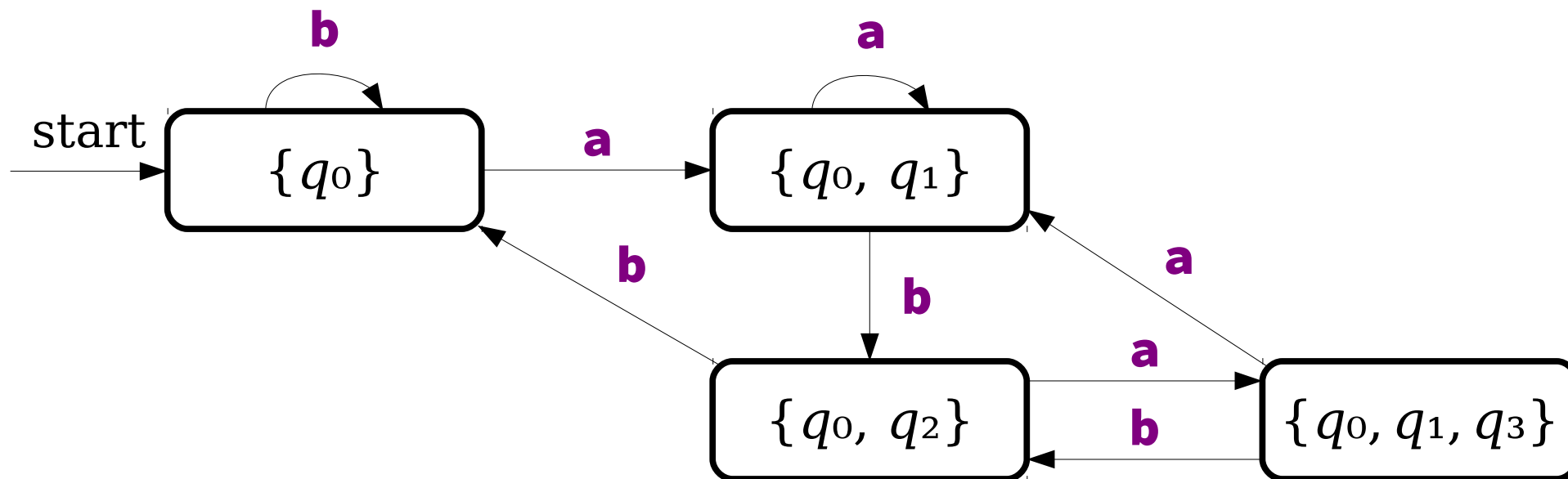


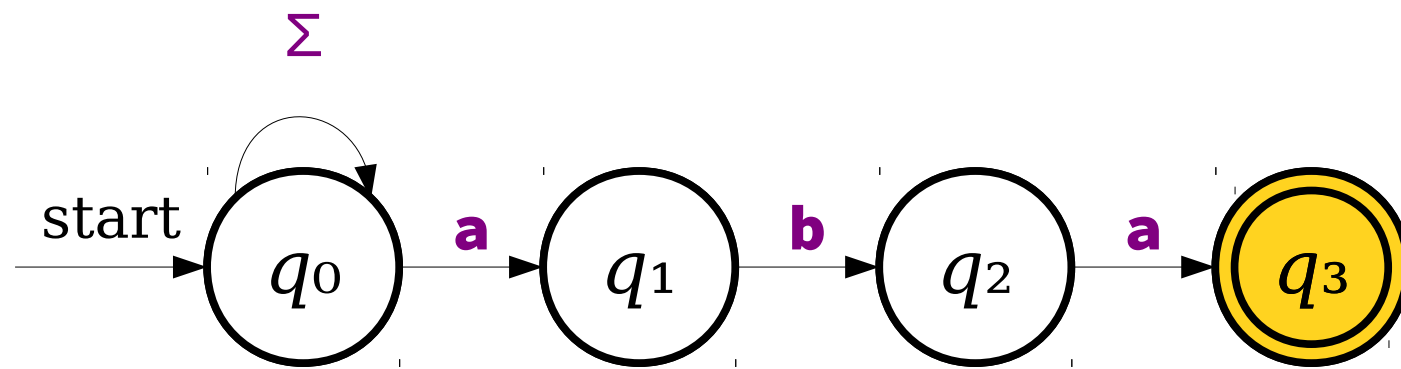
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



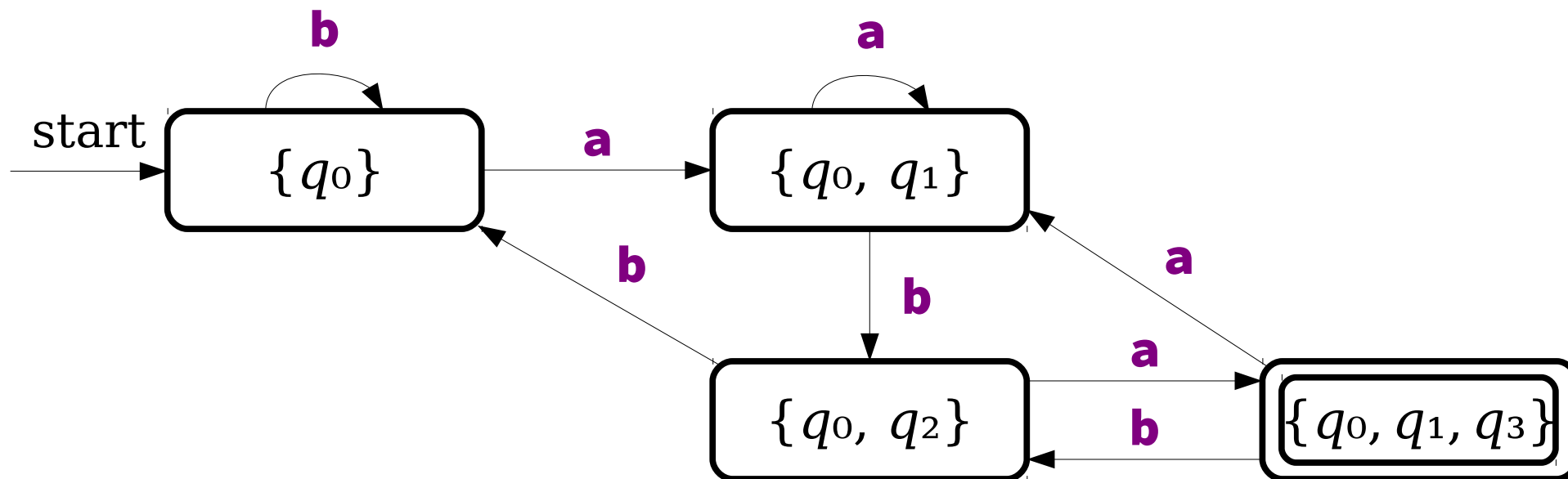


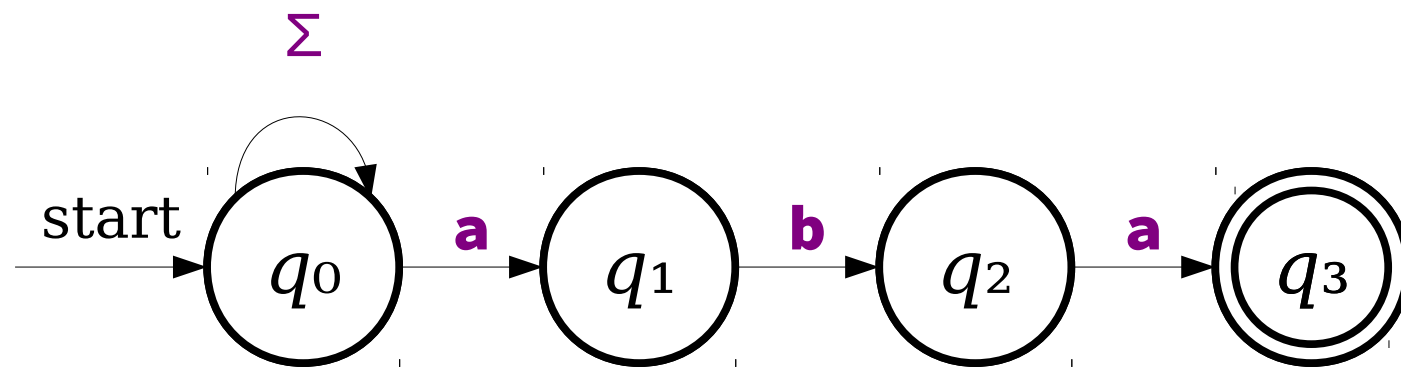
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$*\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



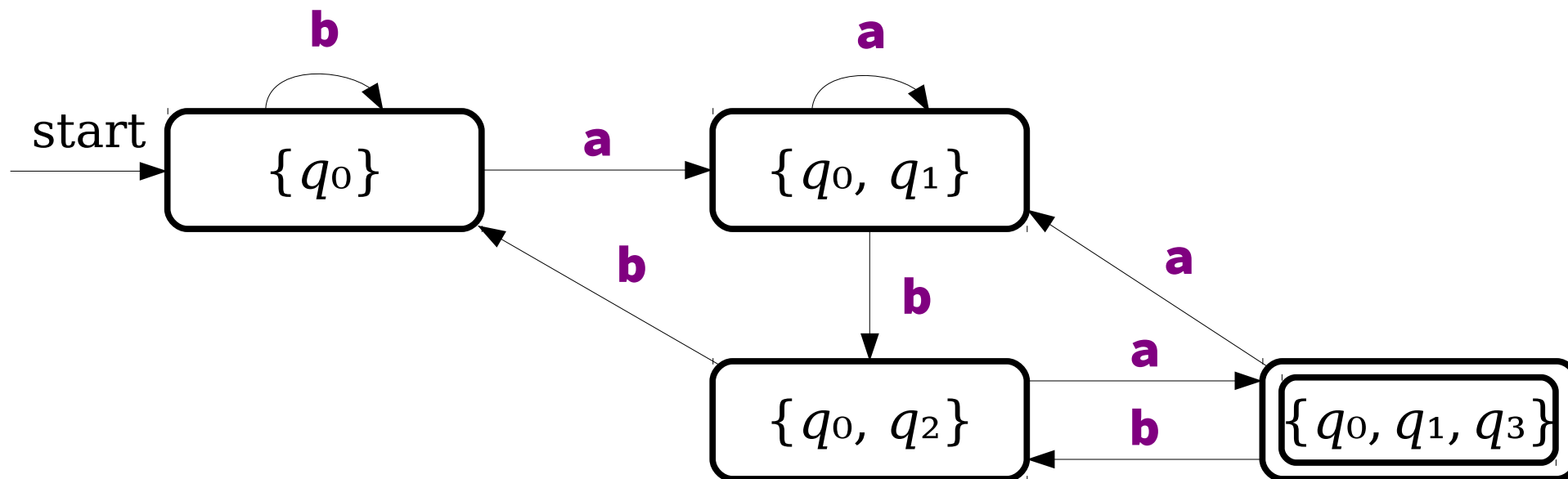


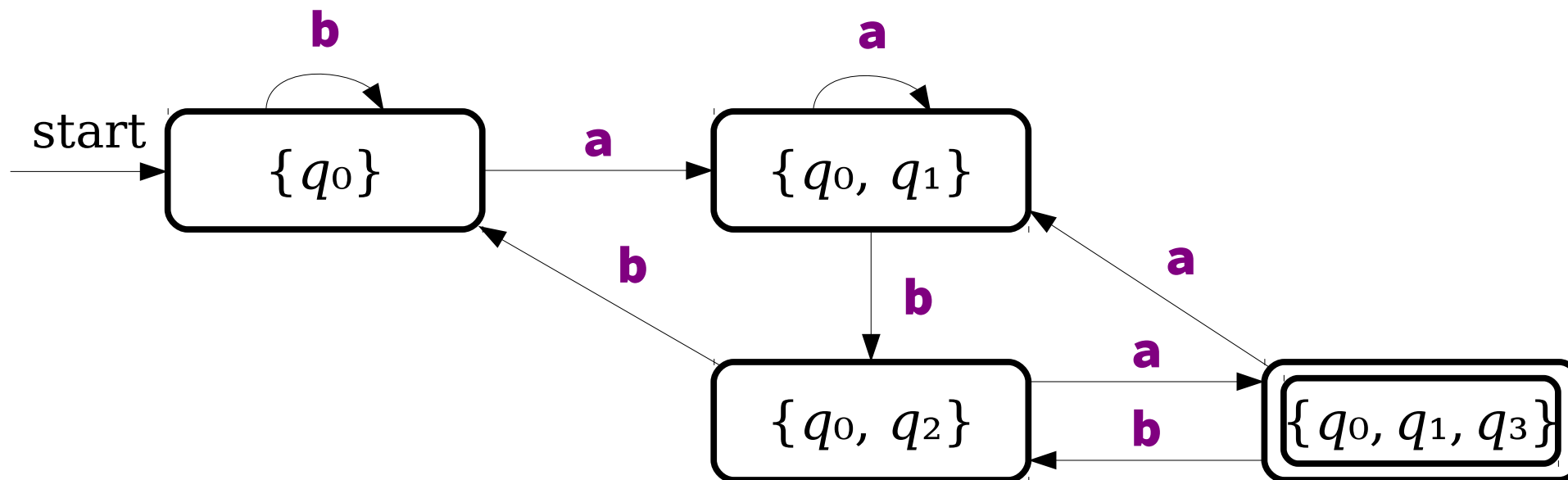
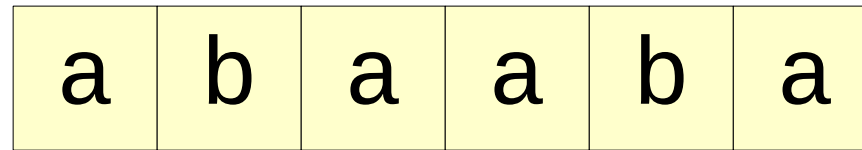
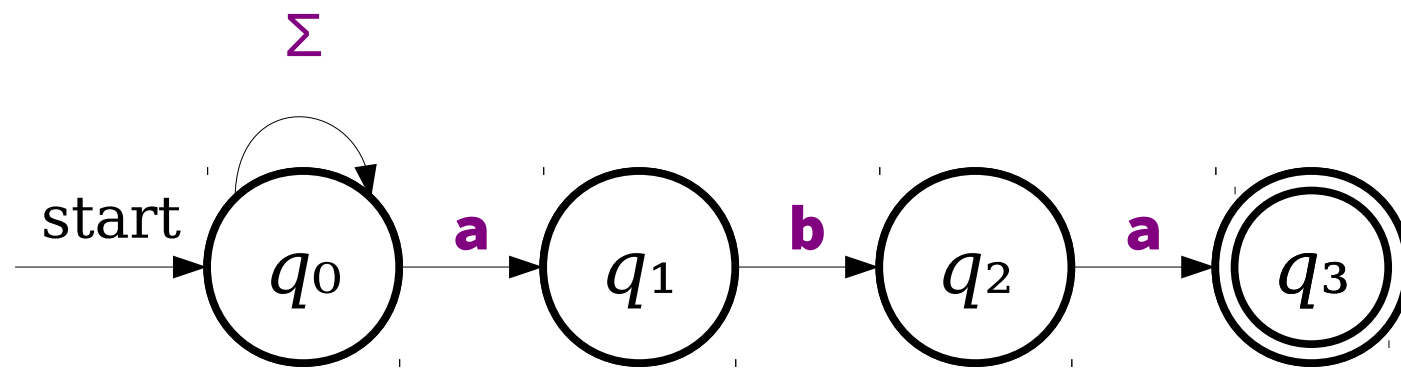
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$*\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

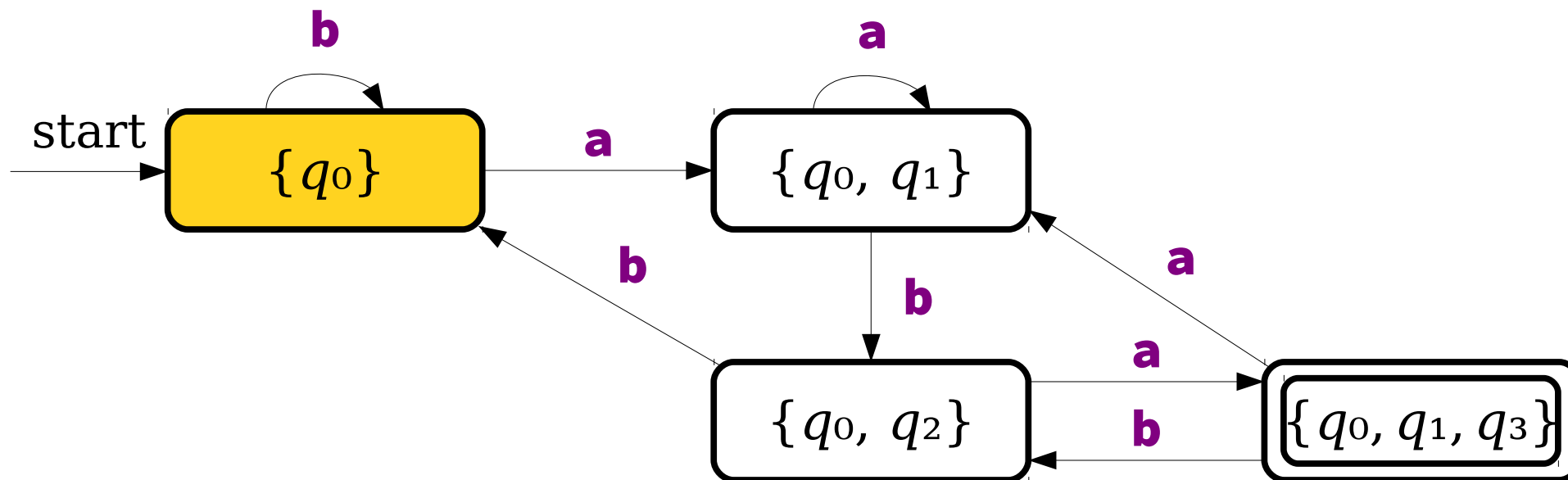
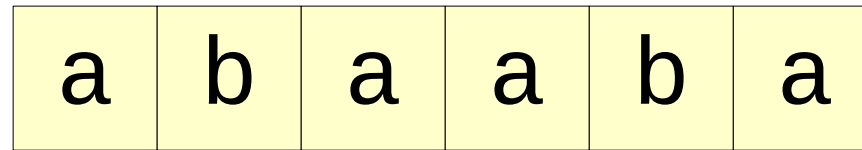
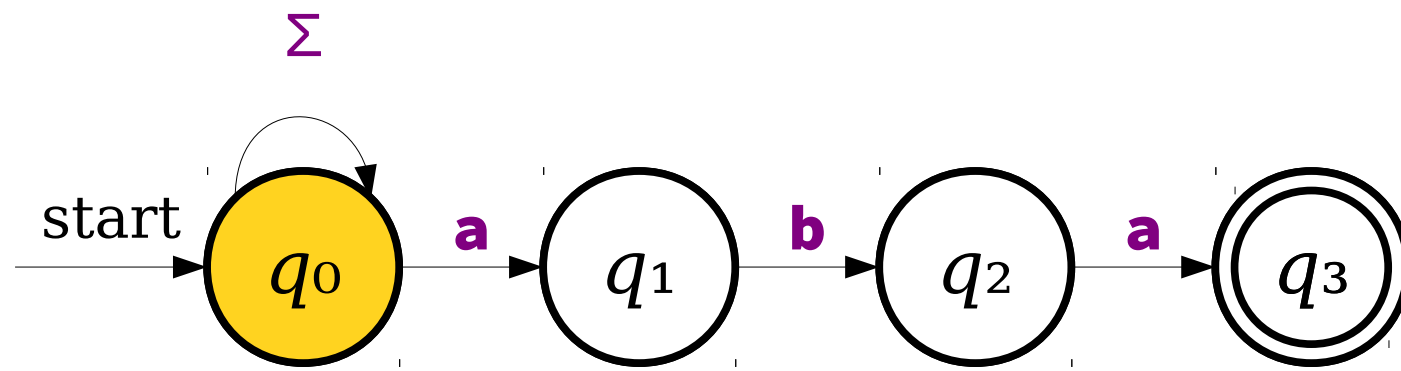


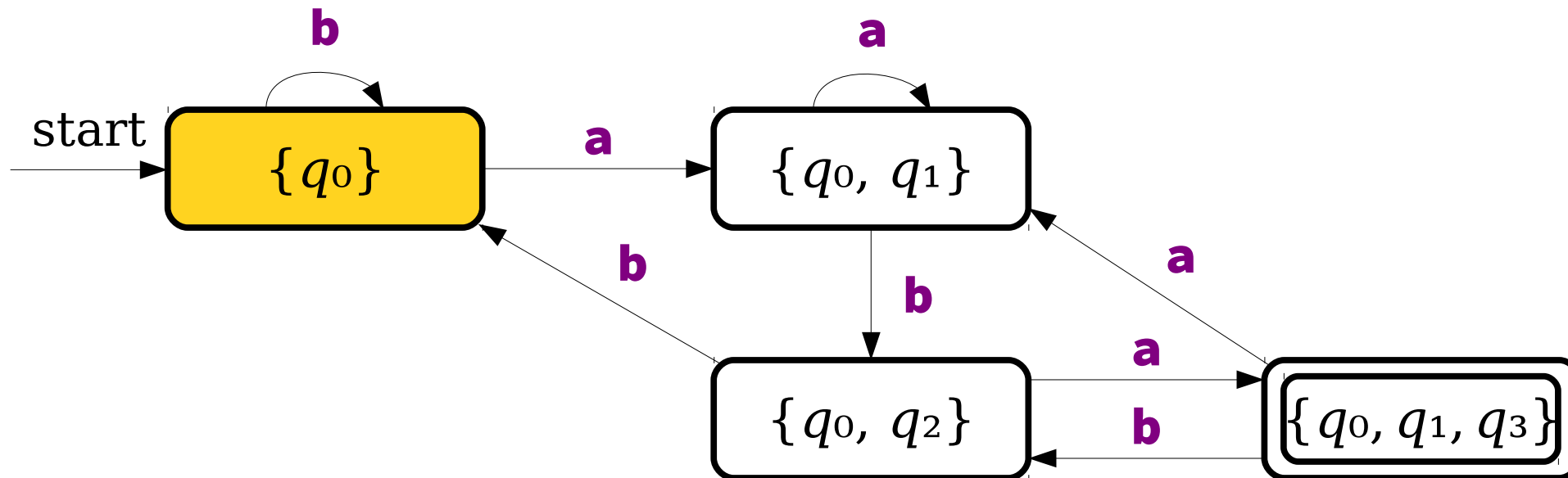
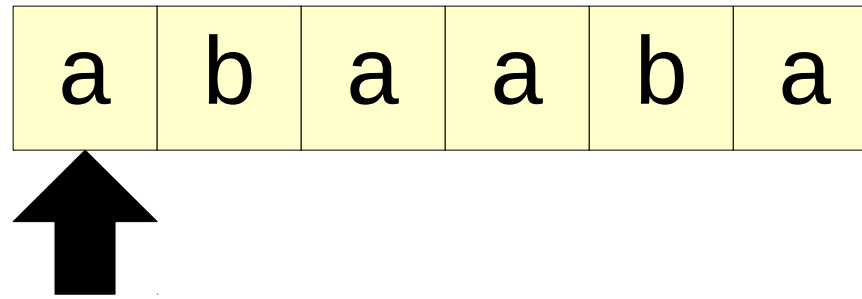
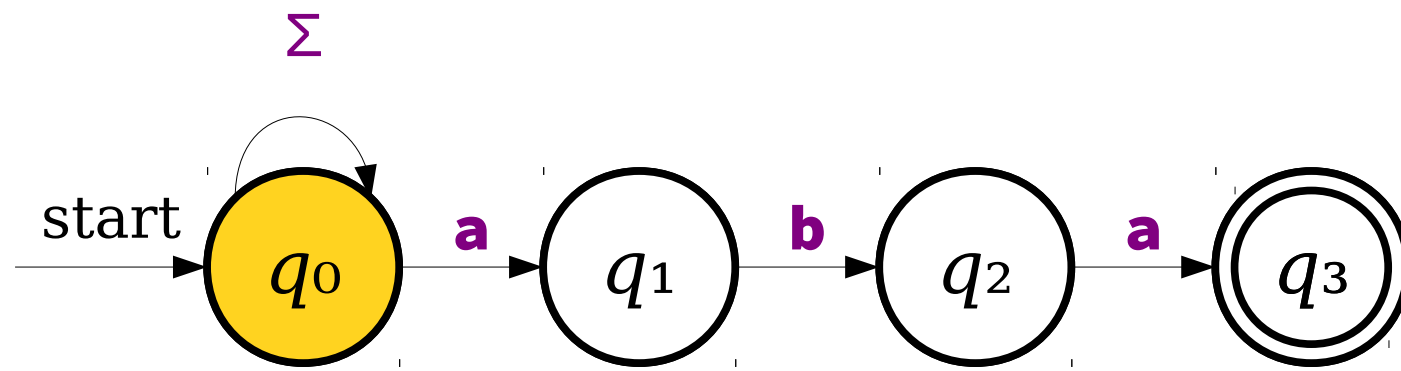


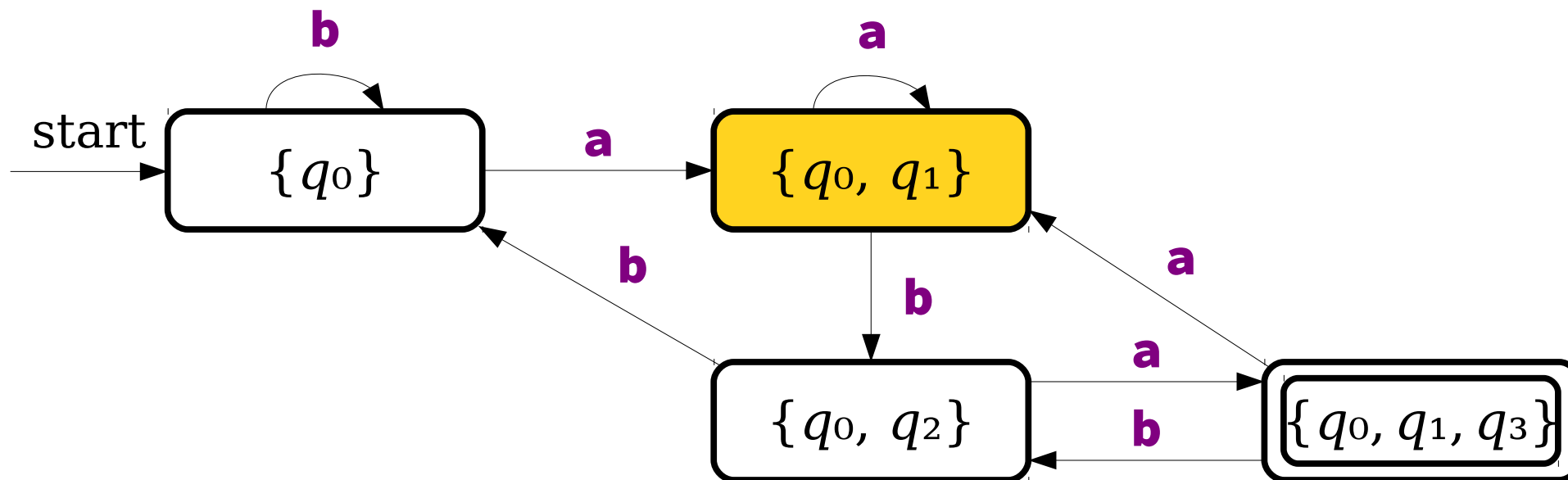
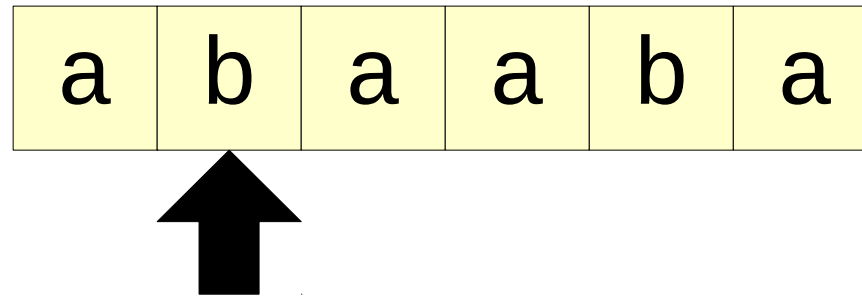
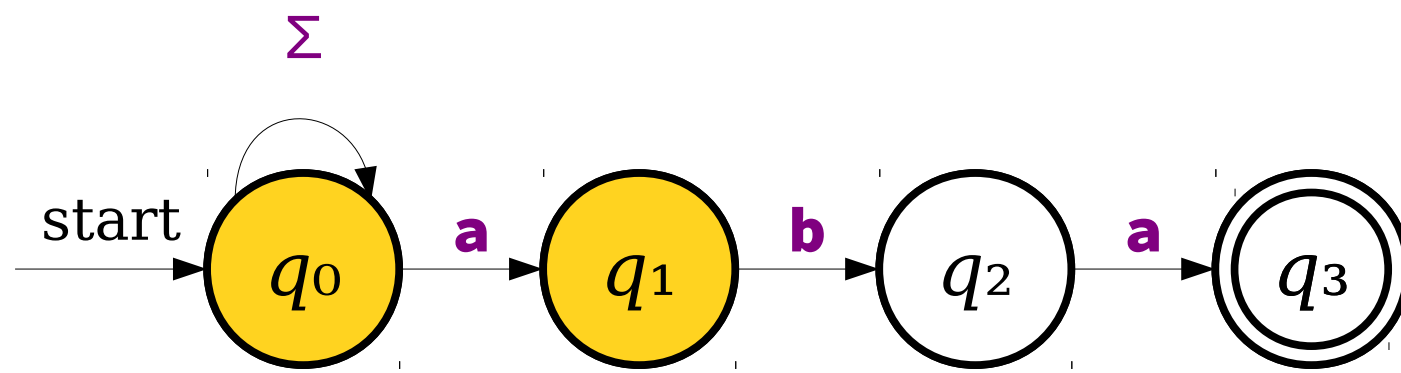
	$a$	$b$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$*\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



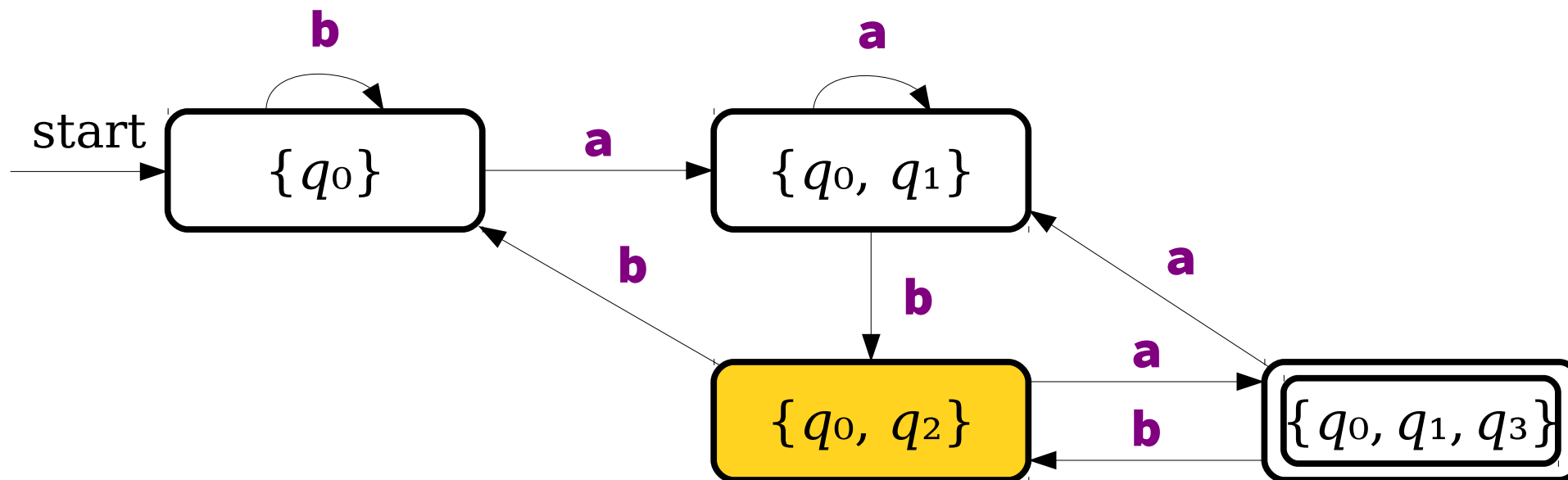
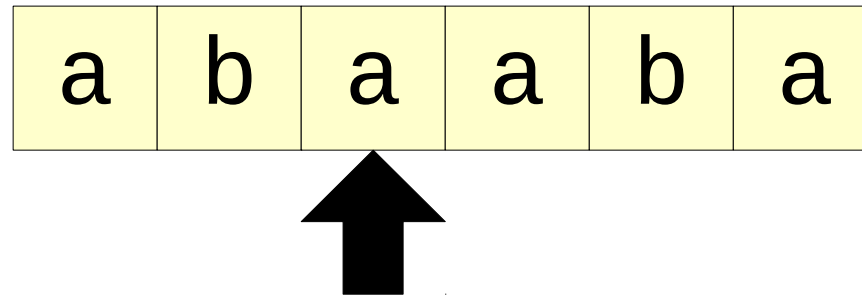
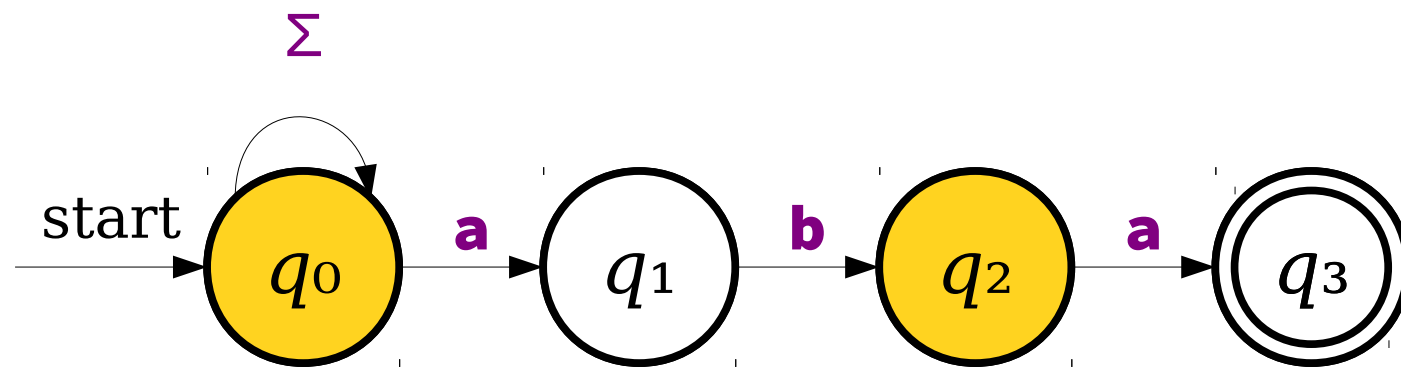


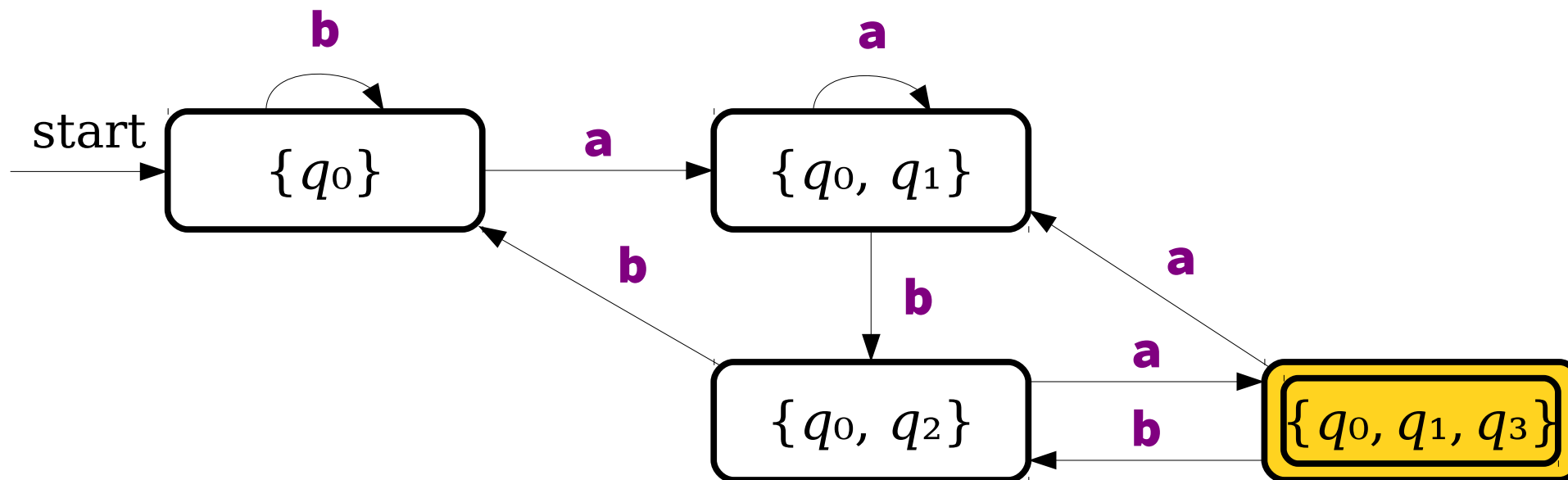
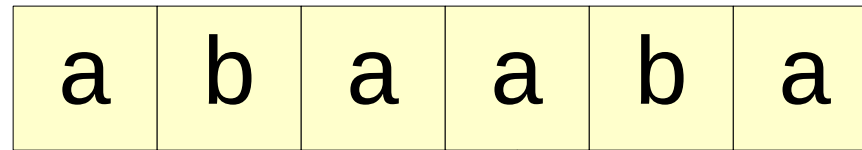
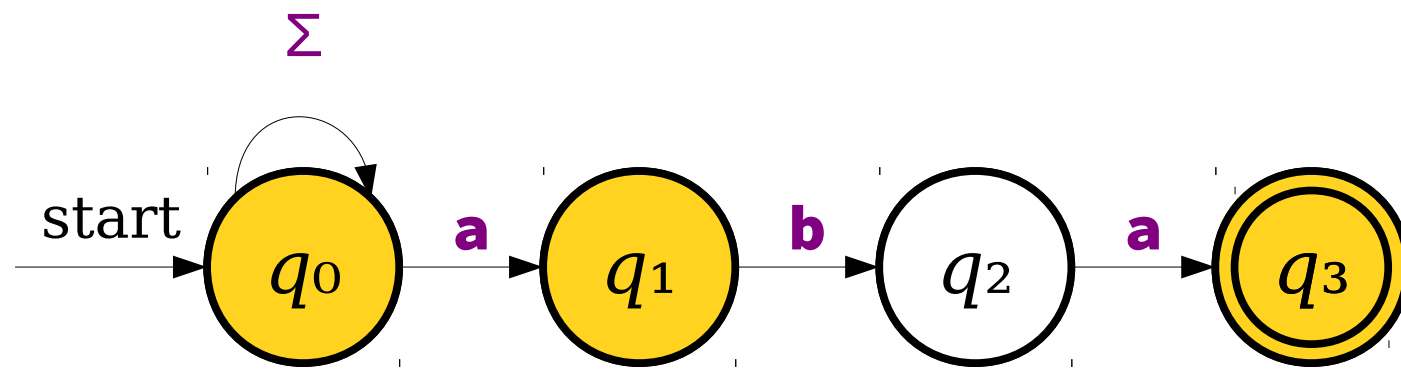


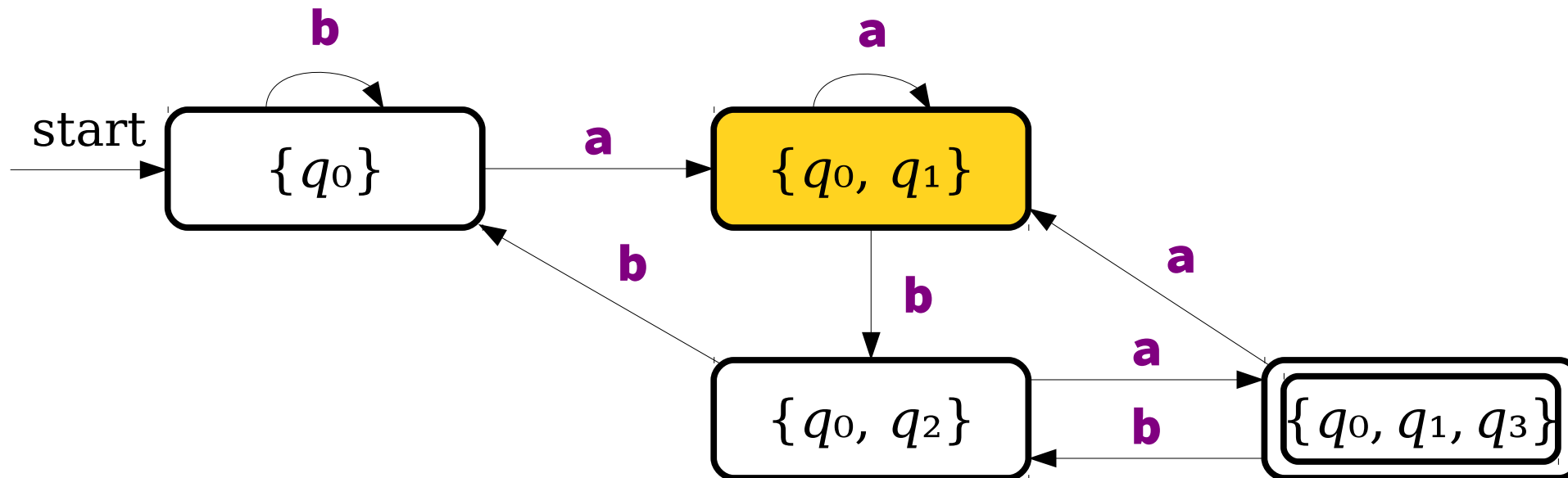
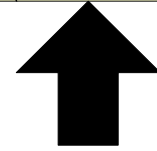
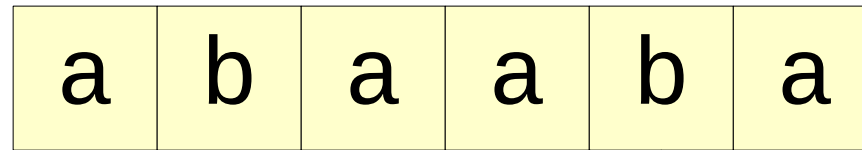
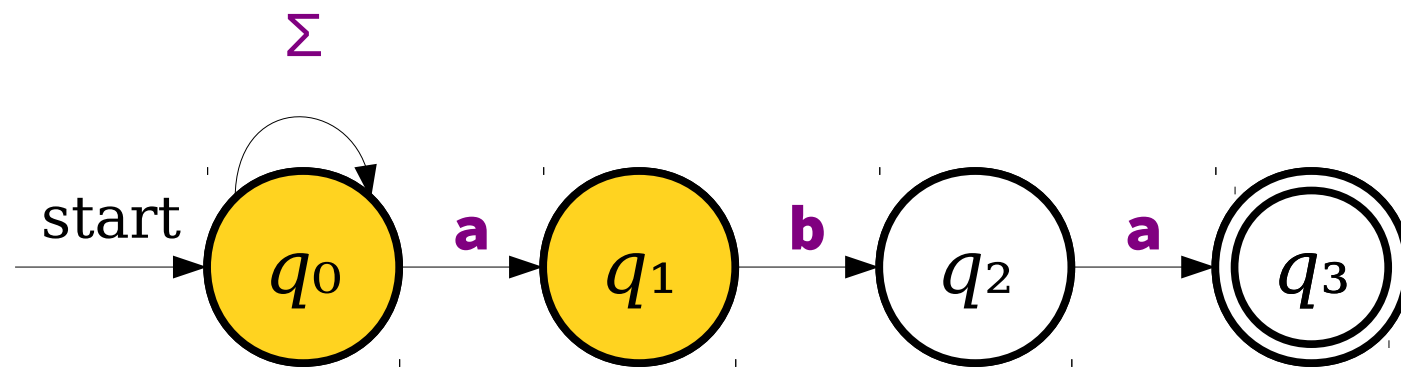


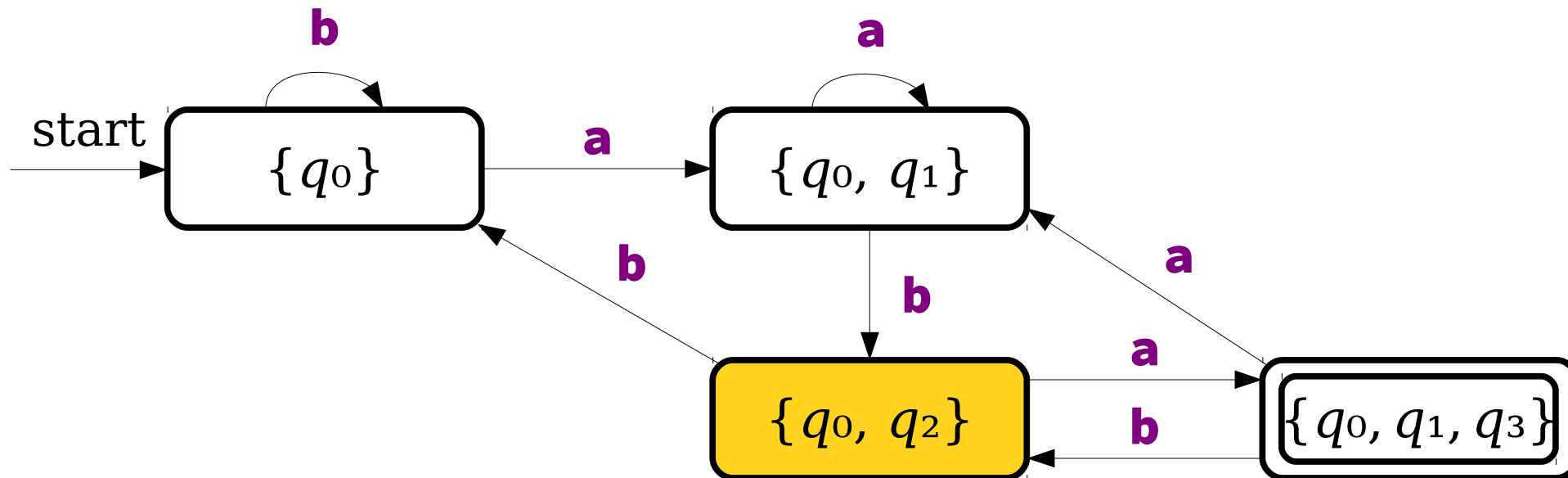
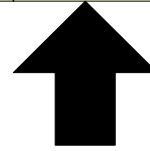
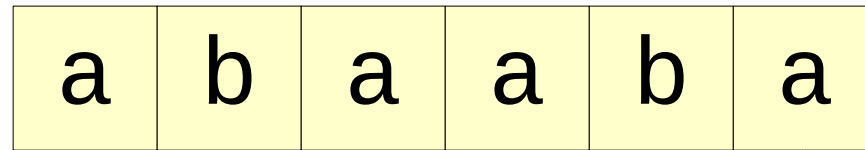
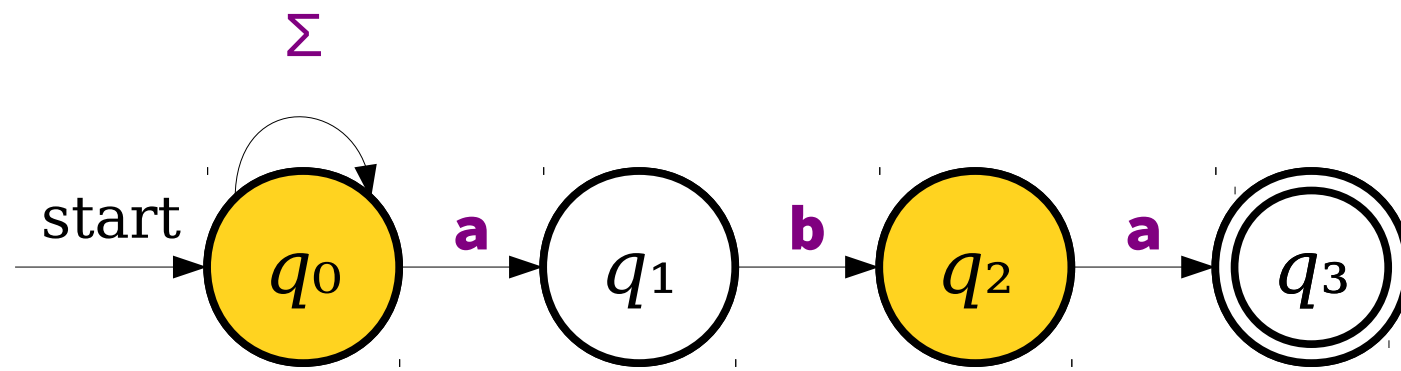


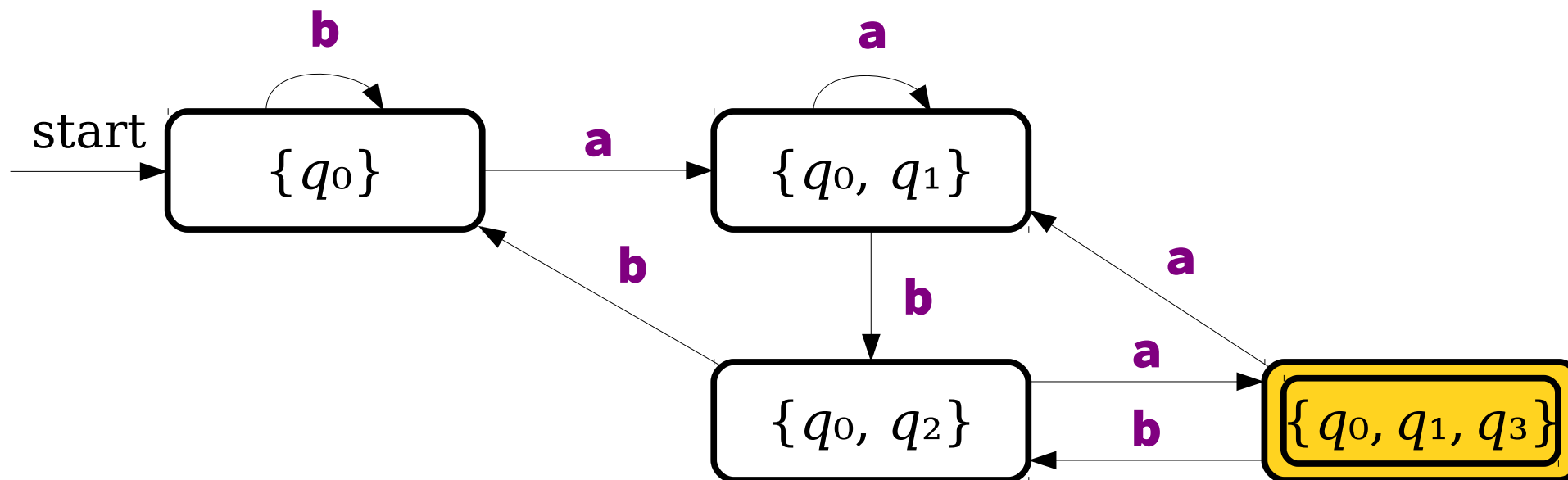
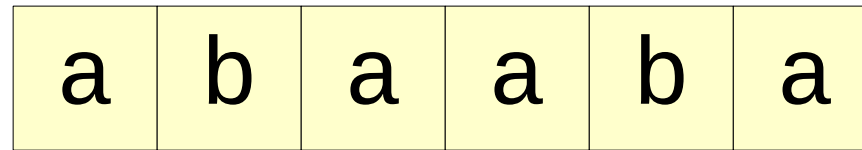
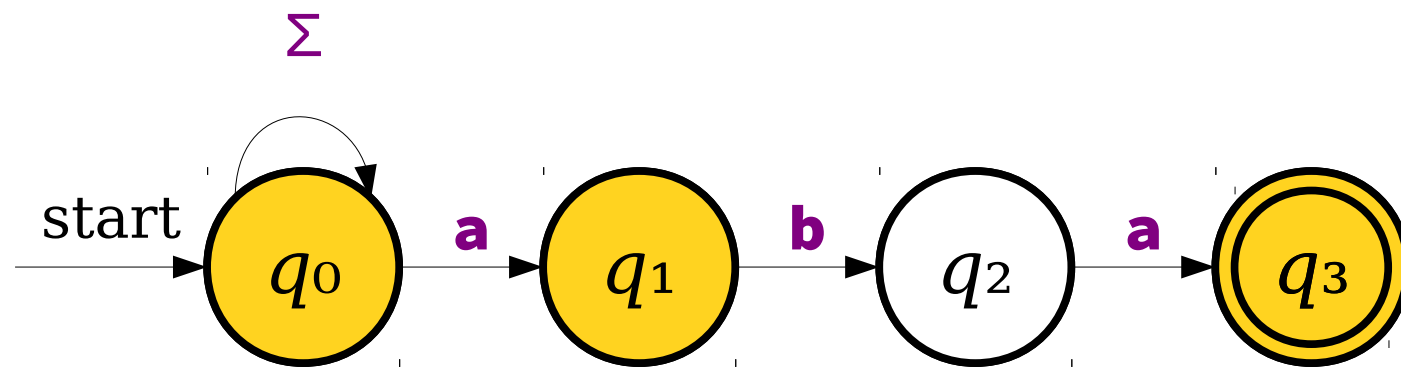












# The Subset Construction

- This procedure for turning an NFA for a language  $L$  into a DFA for a language  $L$  is called the **subset construction**.
  - It's sometimes called the **powerset construction**; it's different names for the same thing!
- Intuitively:
  - Each state in the DFA corresponds to a set of states from the NFA.
  - Each transition in the DFA corresponds to what transitions would be taken in the NFA when using the massive parallel intuition.
  - The accepting states in the DFA correspond to which sets of states would be considered accepting in the NFA when using the massive parallel intuition.
- There's an online **Guide to the Subset Construction** with a more elaborate example involving  $\epsilon$ -transitions and cases where the NFA dies; check that for more details.

# 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  $\mathcal{L}(D) = L$ .



# An Important Result

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

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

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

# Why This Matters

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

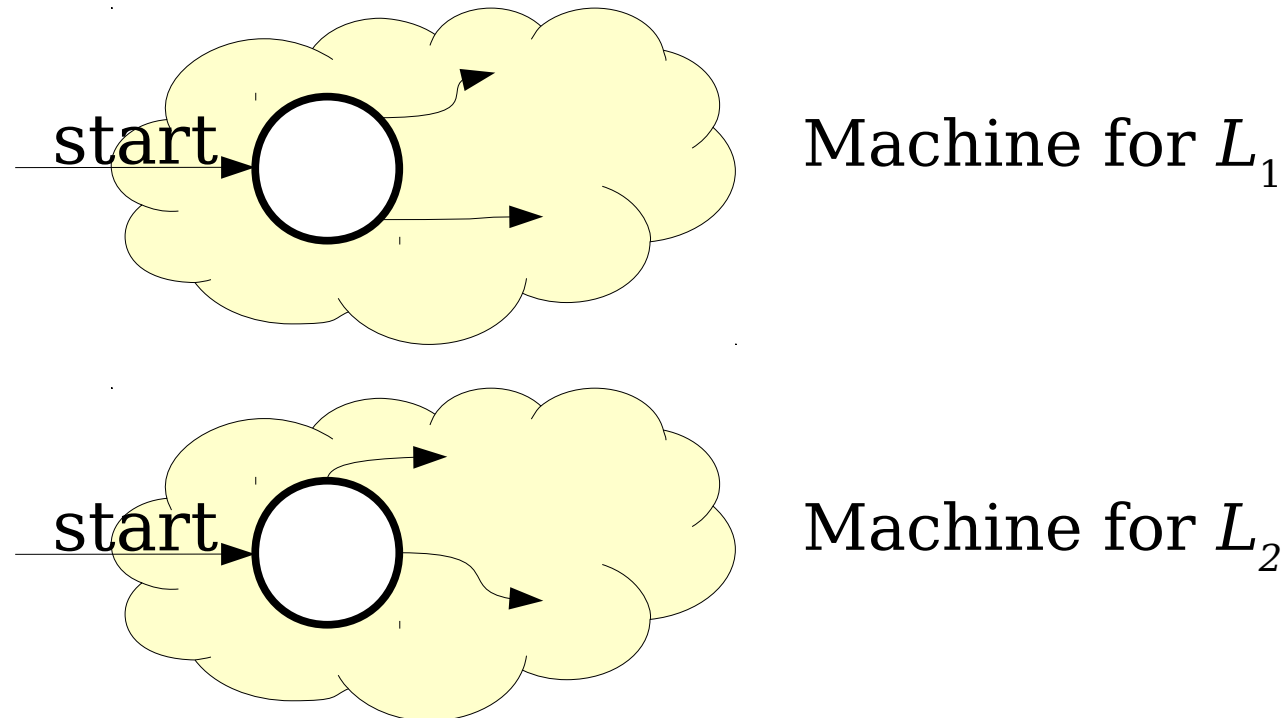
# Properties of Regular Languages

# The Union of Two Languages

- If  $L_1$  and  $L_2$  are languages over the alphabet  $\Sigma$ , 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  $L_1 \cup L_2$ ?

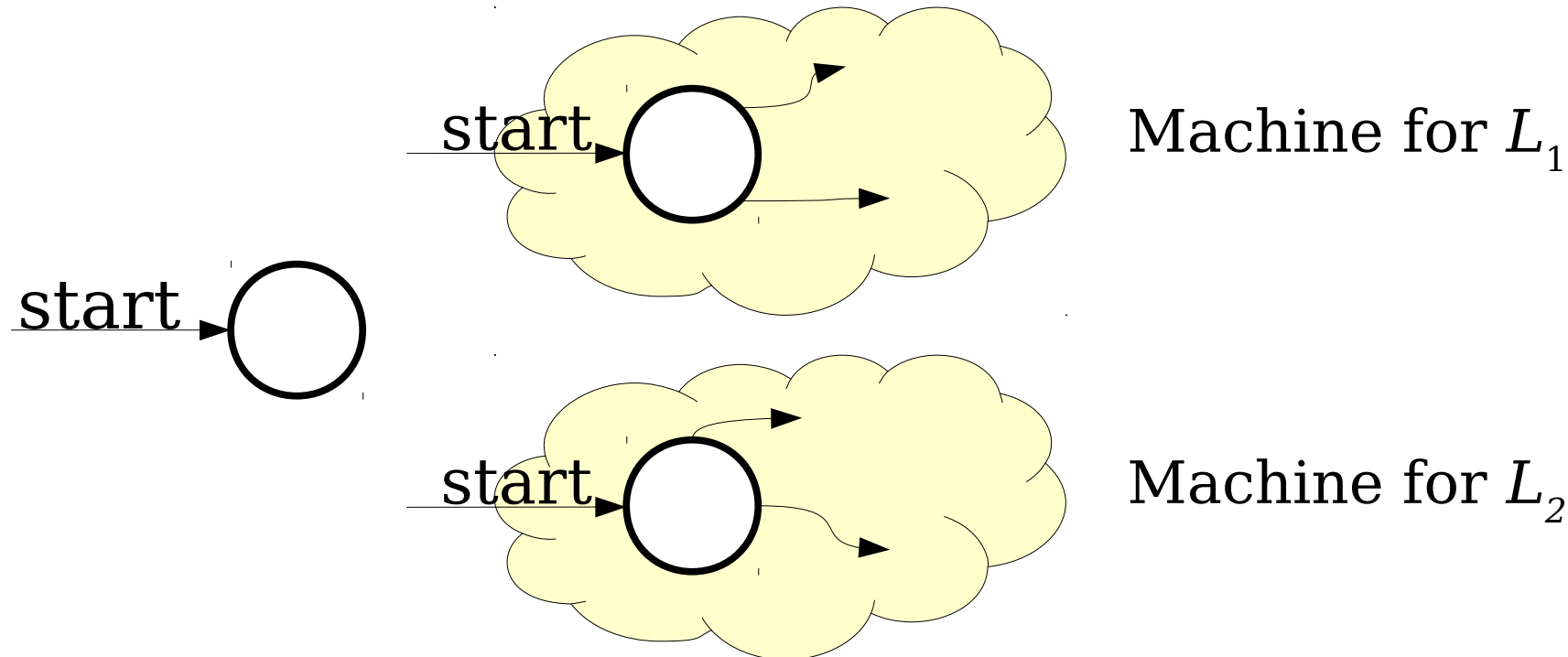
# The Union of Two Languages

- If  $L_1$  and  $L_2$  are languages over the alphabet  $\Sigma$ , 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  $L_1 \cup L_2$ ?



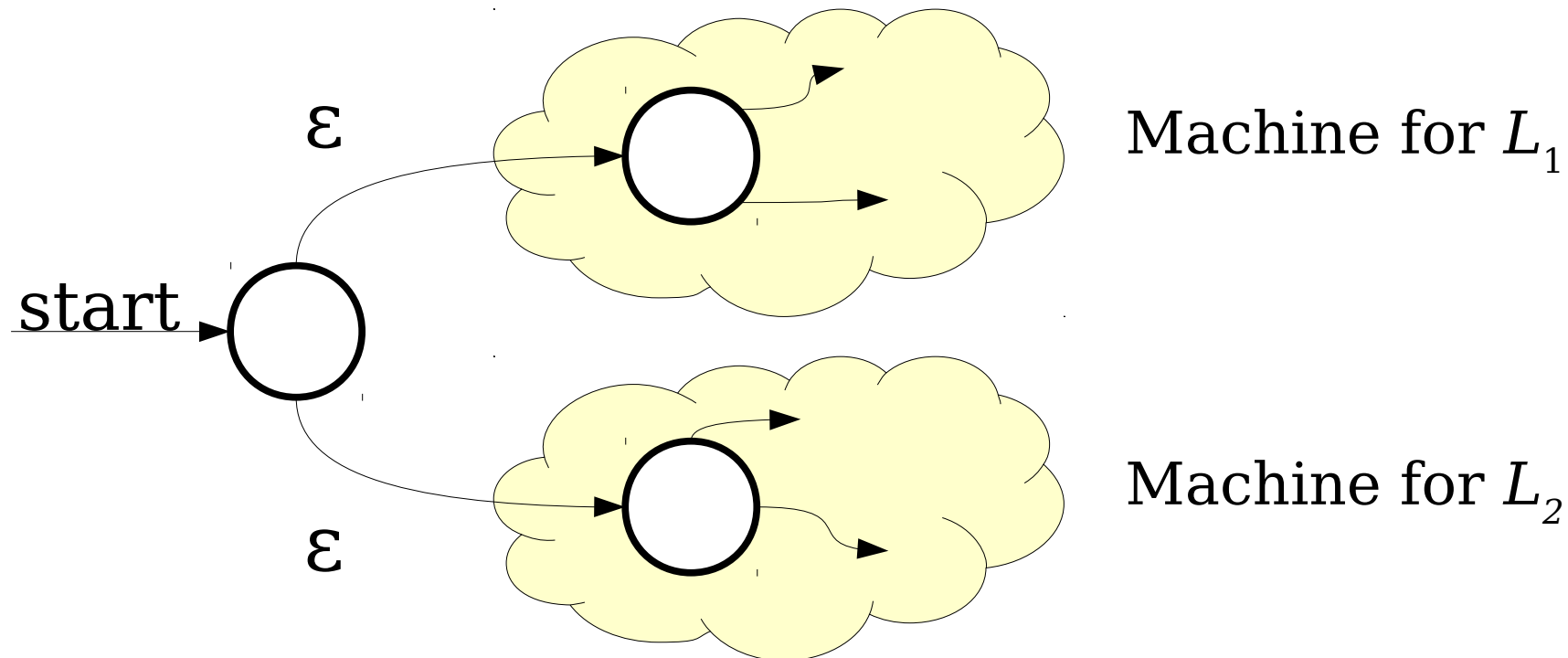
# The Union of Two Languages

- If  $L_1$  and  $L_2$  are languages over the alphabet  $\Sigma$ , 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

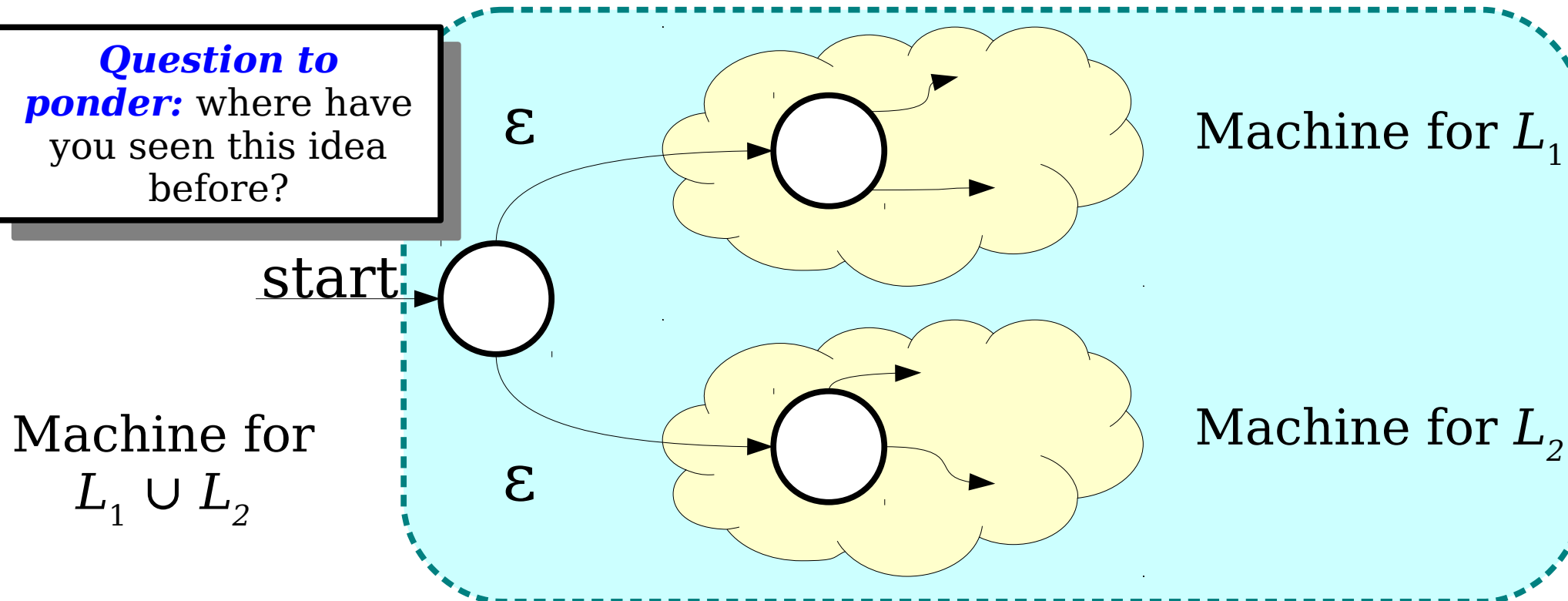
- If  $L_1$  and  $L_2$  are languages over the alphabet  $\Sigma$ , the language  $L_1 \cup L_2$  is the language of all strings in at least one of the two languages.
- If  $L_1$  and  $L_2$  are regular languages, is  $L_1 \cup L_2$ ?



# The Union of Two Languages

- If  $L_1$  and  $L_2$  are languages over the alphabet  $\Sigma$ , 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?



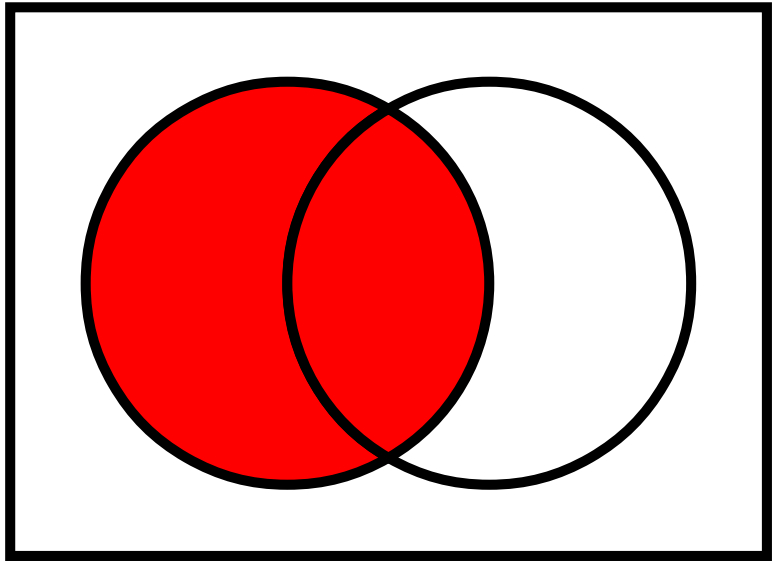


# The Intersection of Two Languages

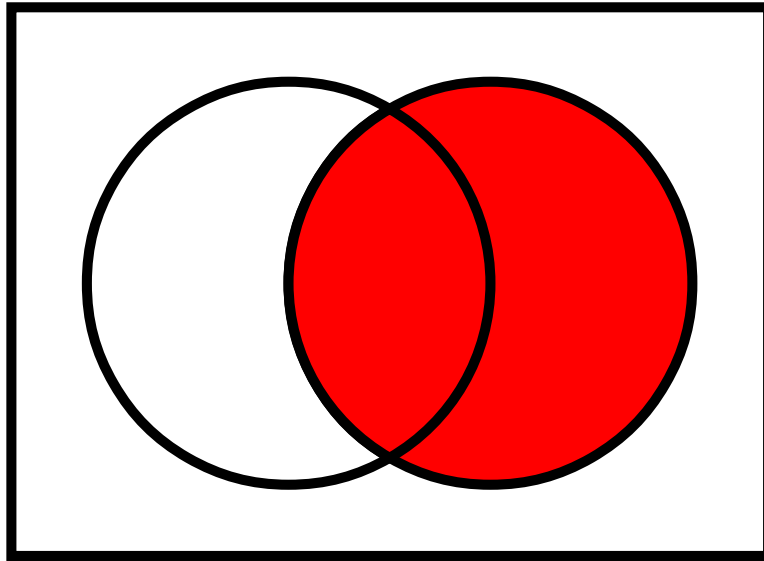
- If  $L_1$  and  $L_2$  are languages over  $\Sigma$ , then  $L_1 \cap L_2$  is the language of strings in both  $L_1$  and  $L_2$ .
- Question: If  $L_1$  and  $L_2$  are regular, is  $L_1 \cap L_2$  regular as well?

# The Intersection of Two Languages

- If  $L_1$  and  $L_2$  are languages over  $\Sigma$ , 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?



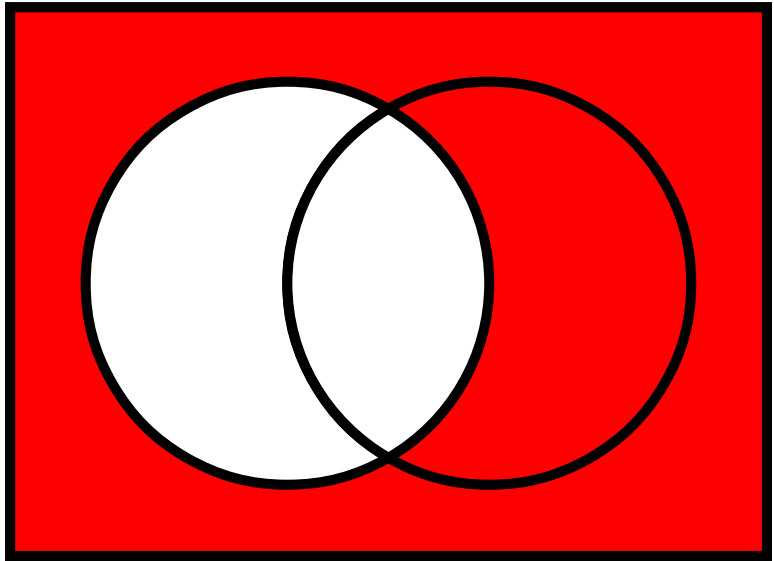
$L_1$



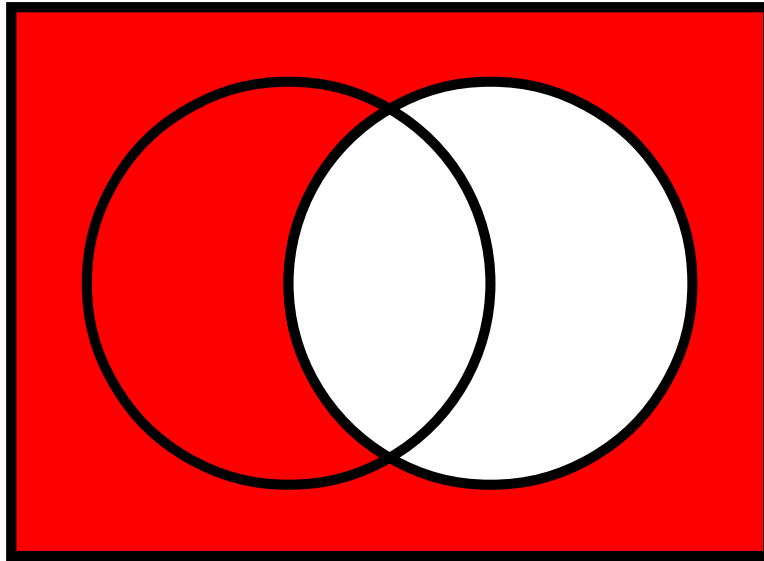
$L_2$

# The Intersection of Two Languages

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



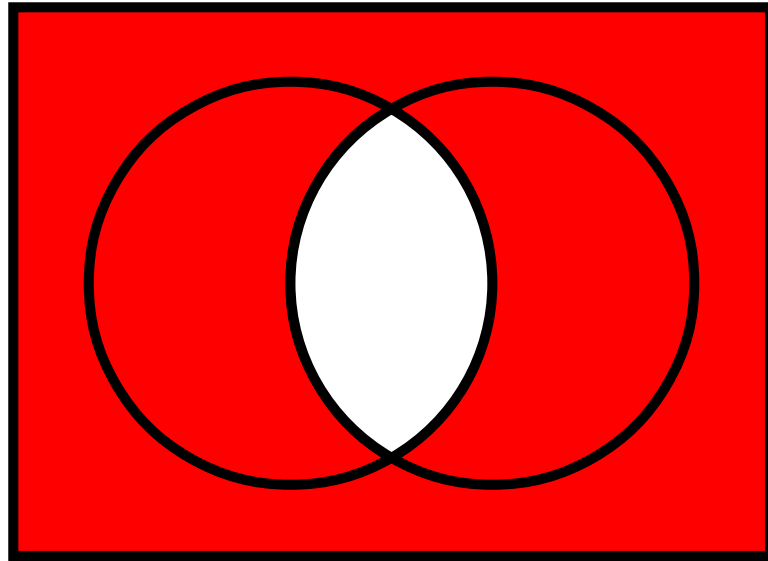
$\overline{L_1}$



$\overline{L_2}$

# The Intersection of Two Languages

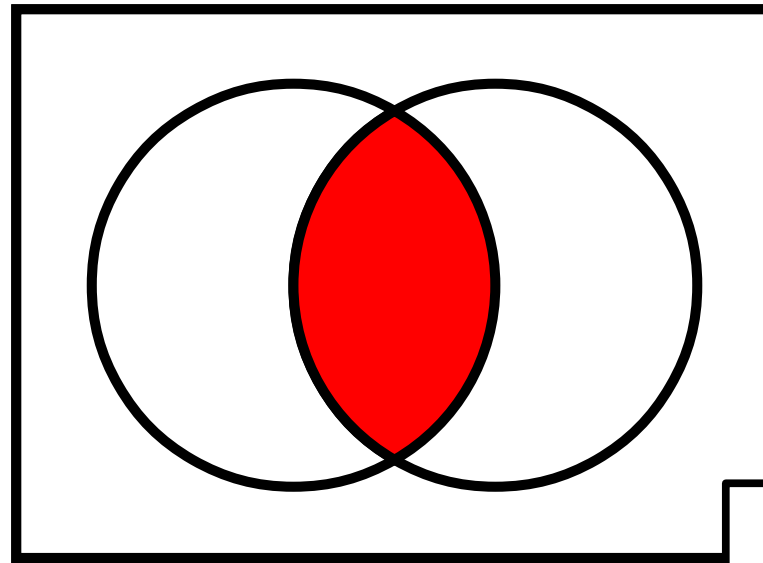
- If  $L_1$  and  $L_2$  are languages over  $\Sigma$ , then  $L_1 \cap L_2$  is the language of strings in both  $L_1$  and  $L_2$ .
- Question: If  $L_1$  and  $L_2$  are regular, is  $L_1 \cap L_2$  regular as well?



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

# The Intersection of Two Languages

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



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

Hey, it's De Morgan's laws!

Concatenation

# String Concatenation

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

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



# Concatenation Example

- Let  $\Sigma = \{ \mathbf{a}, \mathbf{b}, \dots, \mathbf{z}, \mathbf{A}, \mathbf{B}, \dots, \mathbf{Z} \}$  and consider these languages over  $\Sigma$ :
  - ***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  $\Sigma$  is the language

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

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

This is closely related to, but different than, the Cartesian product.

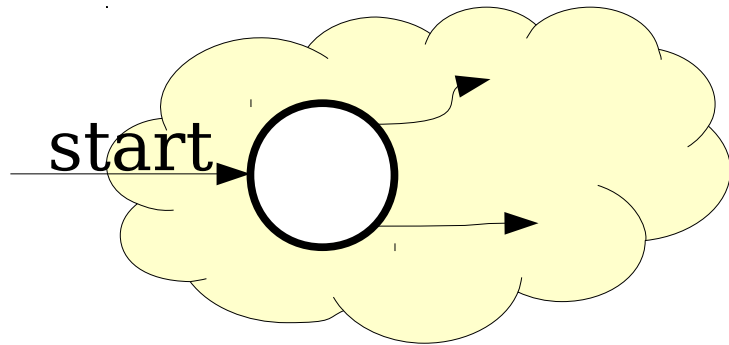
**Question to ponder:** In what ways are concatenations similar to Cartesian products? In what ways are they different?

# Concatenating Regular Languages

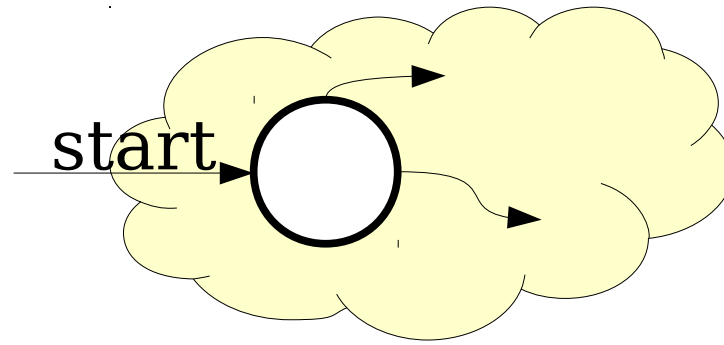
- If  $L_1$  and  $L_2$  are regular languages, is  $L_1L_2$ ?
- Intuition – can we split a string  $w$  into two strings  $xy$  such that  $x \in L_1$  and  $y \in L_2$ ?
- *Idon*.

# 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  $y \in L_2$ ?
- *Idon*.



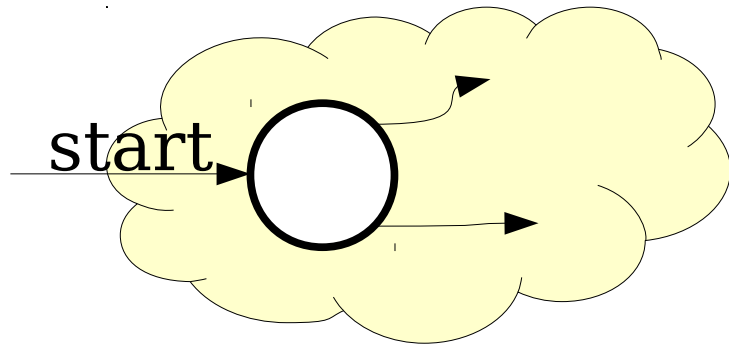
Machine for  $L_1$



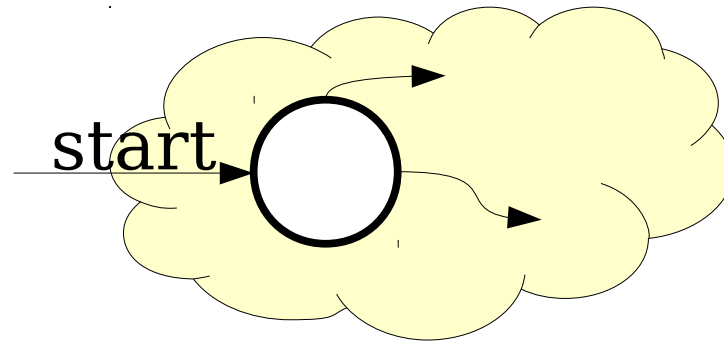
Machine for  $L_2$

# Concatenating Regular Languages

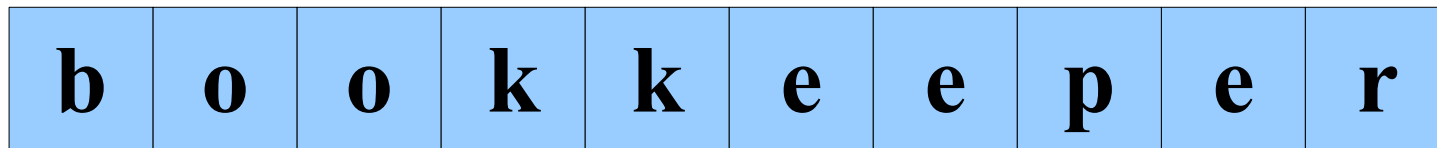
- If  $L_1$  and  $L_2$  are regular languages, is  $L_1L_2$ ?
- Intuition – can we split a string  $w$  into two strings  $xy$  such that  $x \in L_1$  and  $y \in L_2$ ?
- *Idon.*



Machine for  $L_1$

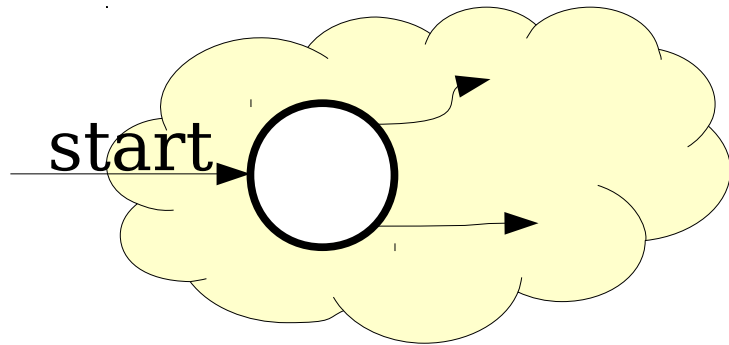


Machine for  $L_2$

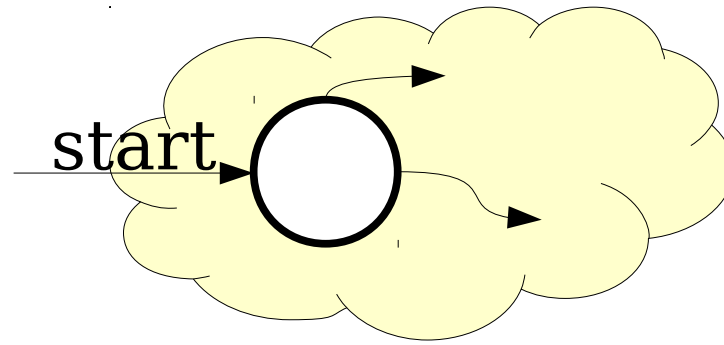


# Concatenating Regular Languages

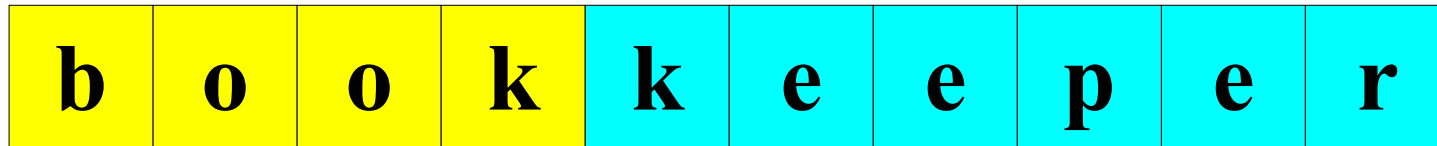
- If  $L_1$  and  $L_2$  are regular languages, is  $L_1L_2$ ?
- Intuition – can we split a string  $w$  into two strings  $xy$  such that  $x \in L_1$  and  $y \in L_2$ ?
- *Idon.*



Machine for  $L_1$

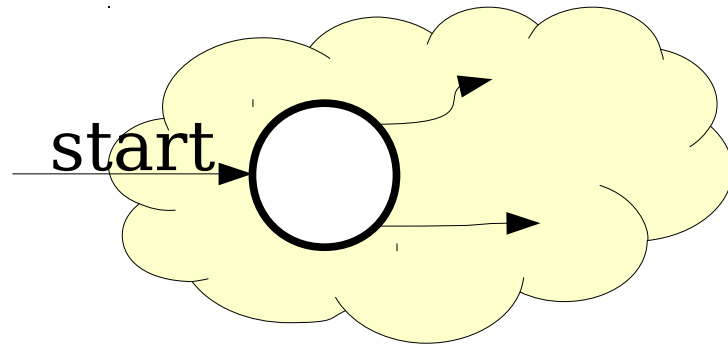


Machine for  $L_2$



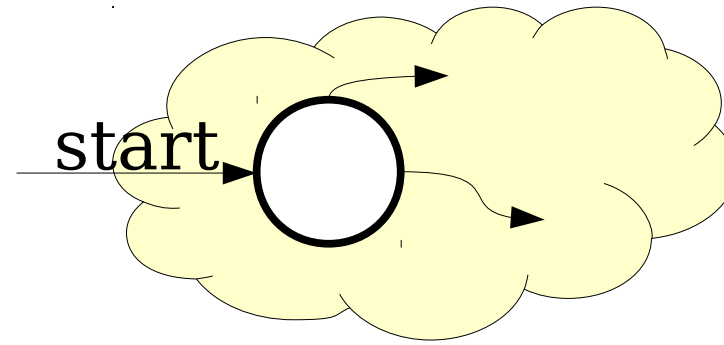
# Concatenating Regular Languages

- 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  $y \in L_2$ ?
- *Idon.*



Machine for  $L_1$

b	o	o	k
---	---	---	---



Machine for  $L_2$

k	e	e	p	e	r
---	---	---	---	---	---

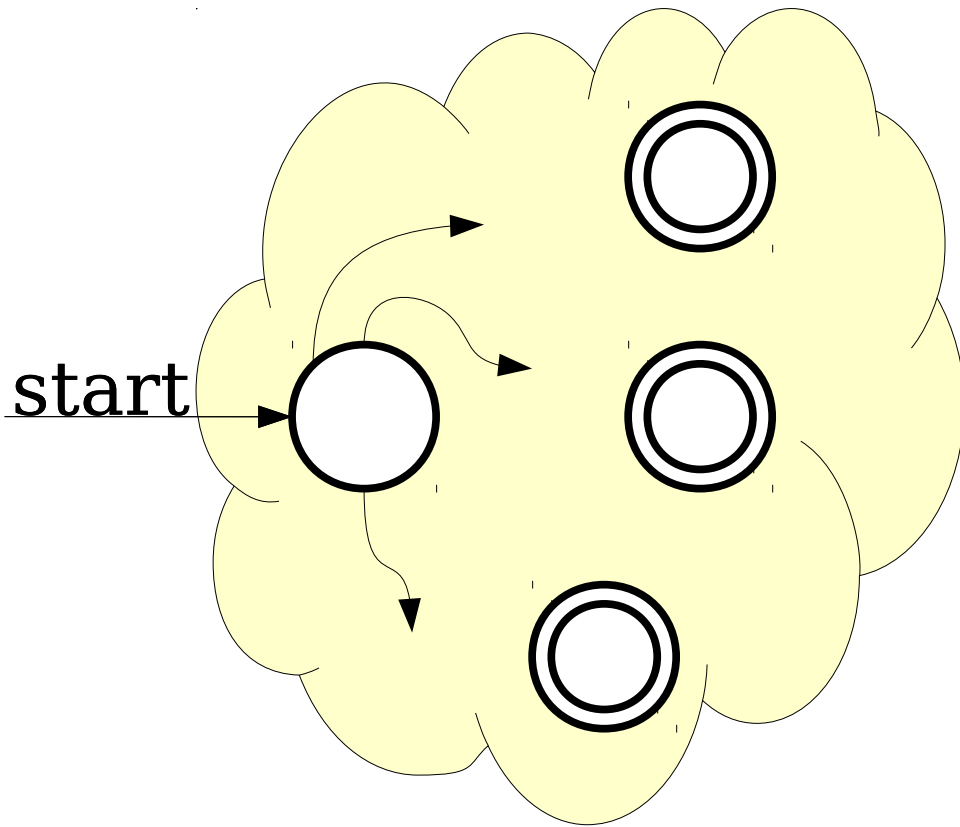
# Concatenating Regular Languages

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



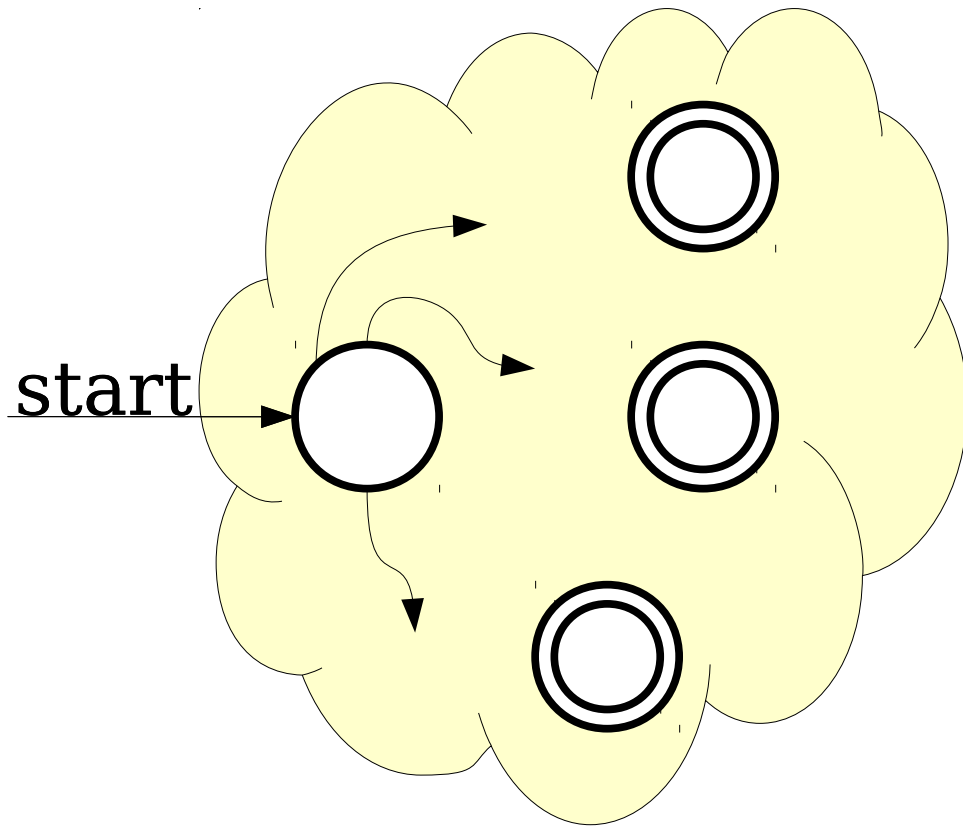
# Concatenating Regular Languages

# Concatenating Regular Languages

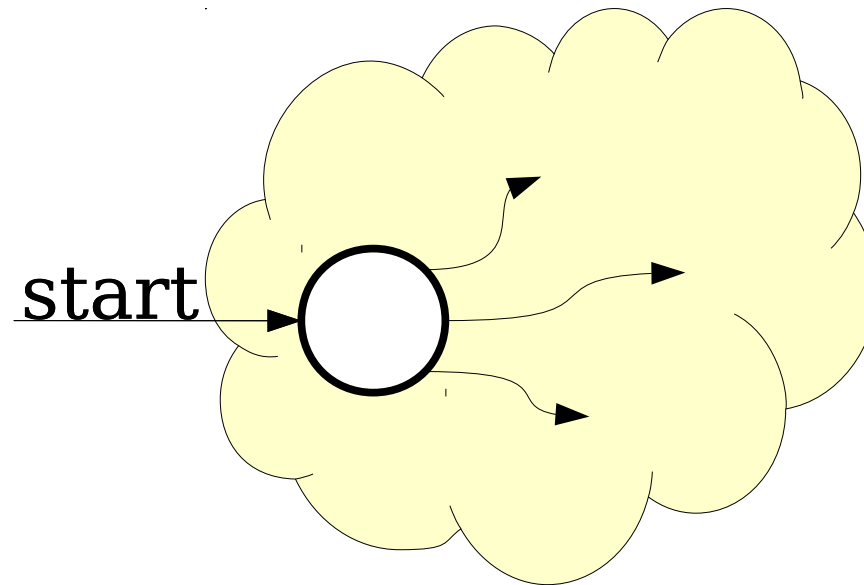


Machine for  
 $L_1$

# Concatenating Regular Languages

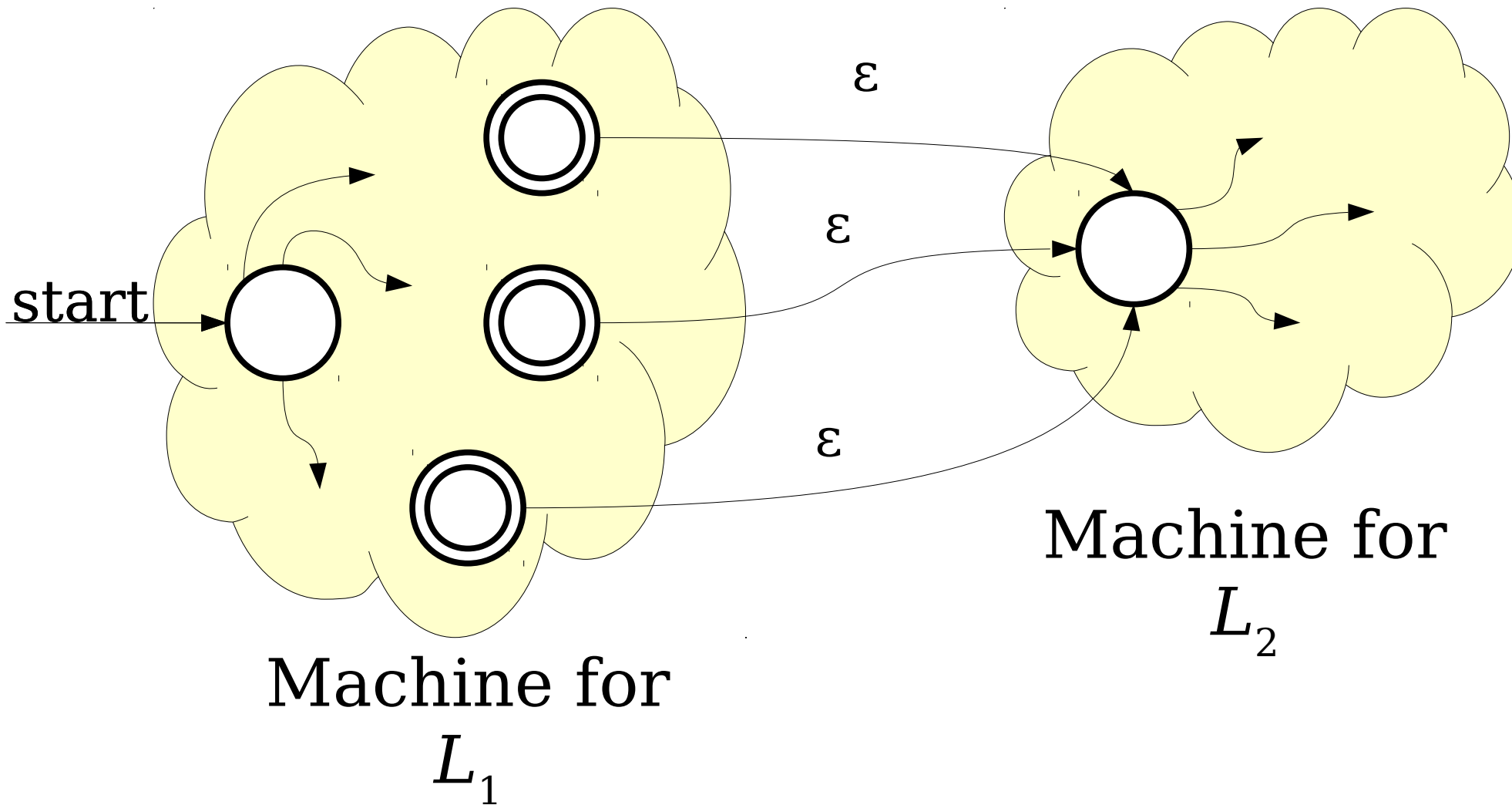


Machine for  
 $L_1$

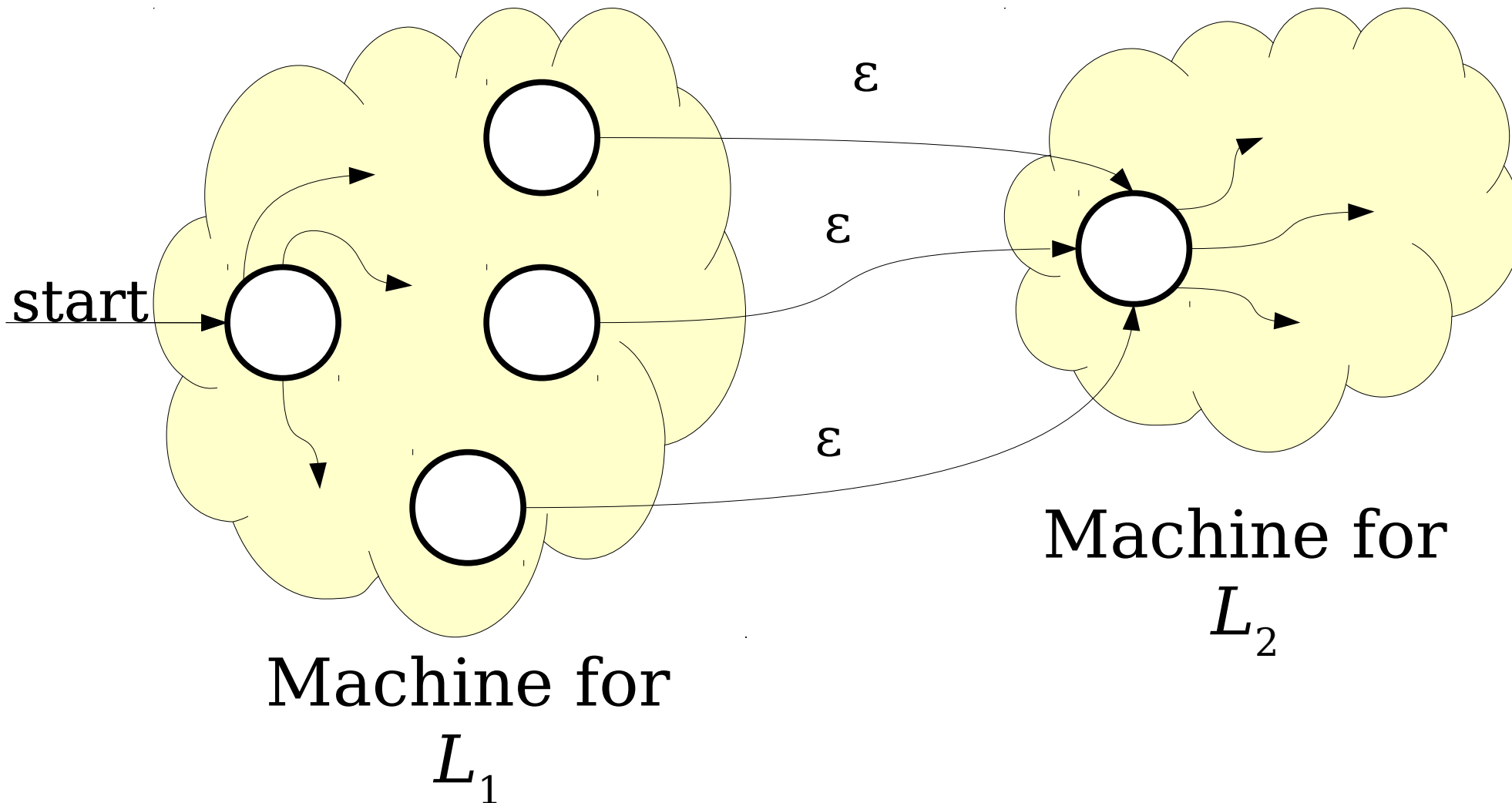


Machine for  
 $L_2$

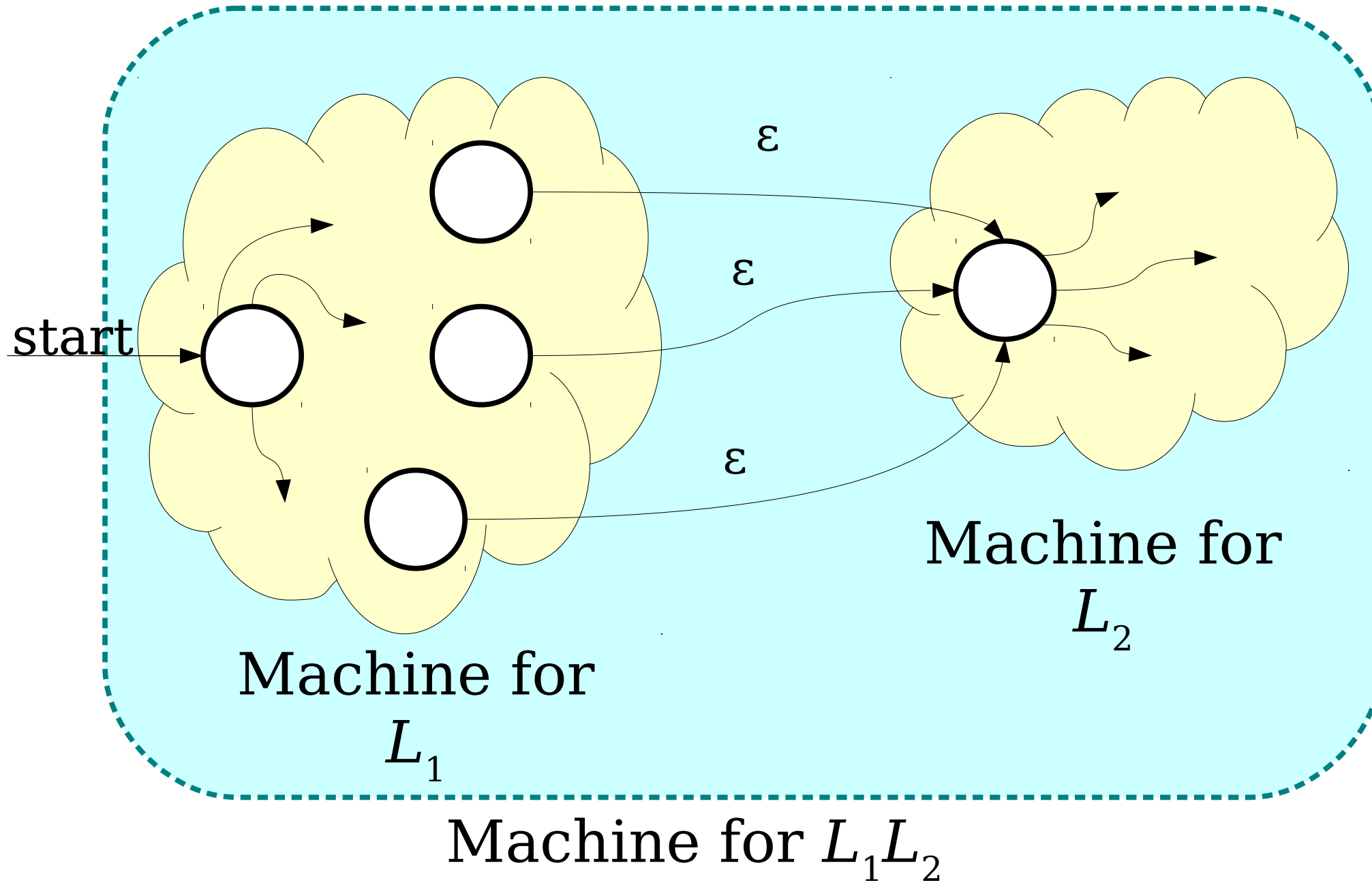
# Concatenating Regular Languages



# Concatenating Regular Languages



# Concatenating Regular Languages



# Lots and Lots of Concatenation

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

$\{ \text{aaaa}, \text{aab}, \text{baa}, \text{bb} \}$

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

$\{ \text{aaaaaa}, \text{aaaab}, \text{aabaa}, \text{aabb}, \text{baaaa}, \text{baab}, \text{bbaa}, \text{bbb} \}$

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

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

# Language Exponentiation

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



The Kleene Star

# The Kleene Closure

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

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

- Mathematically:

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



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

# The Kleene Closure

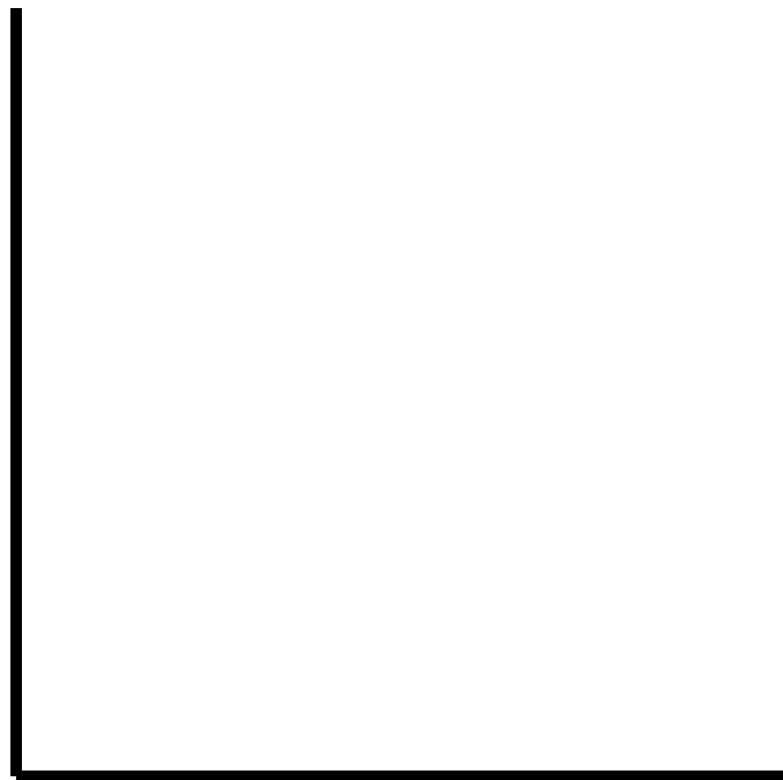
If  $L = \{ \text{a}, \text{bb} \}$ , then  $L^* = \{$   
 $\epsilon,$   
 $\text{a}, \text{bb},$   
 $\text{aa}, \text{abb}, \text{bba}, \text{bbbb},$   
 $\text{aaa}, \text{aabb}, \text{abba}, \text{abbbb}, \text{bbaa}, \text{bbabb}, \text{bbbba}, \text{bbbbbb},$   
 $\dots$   
 $\}$

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.

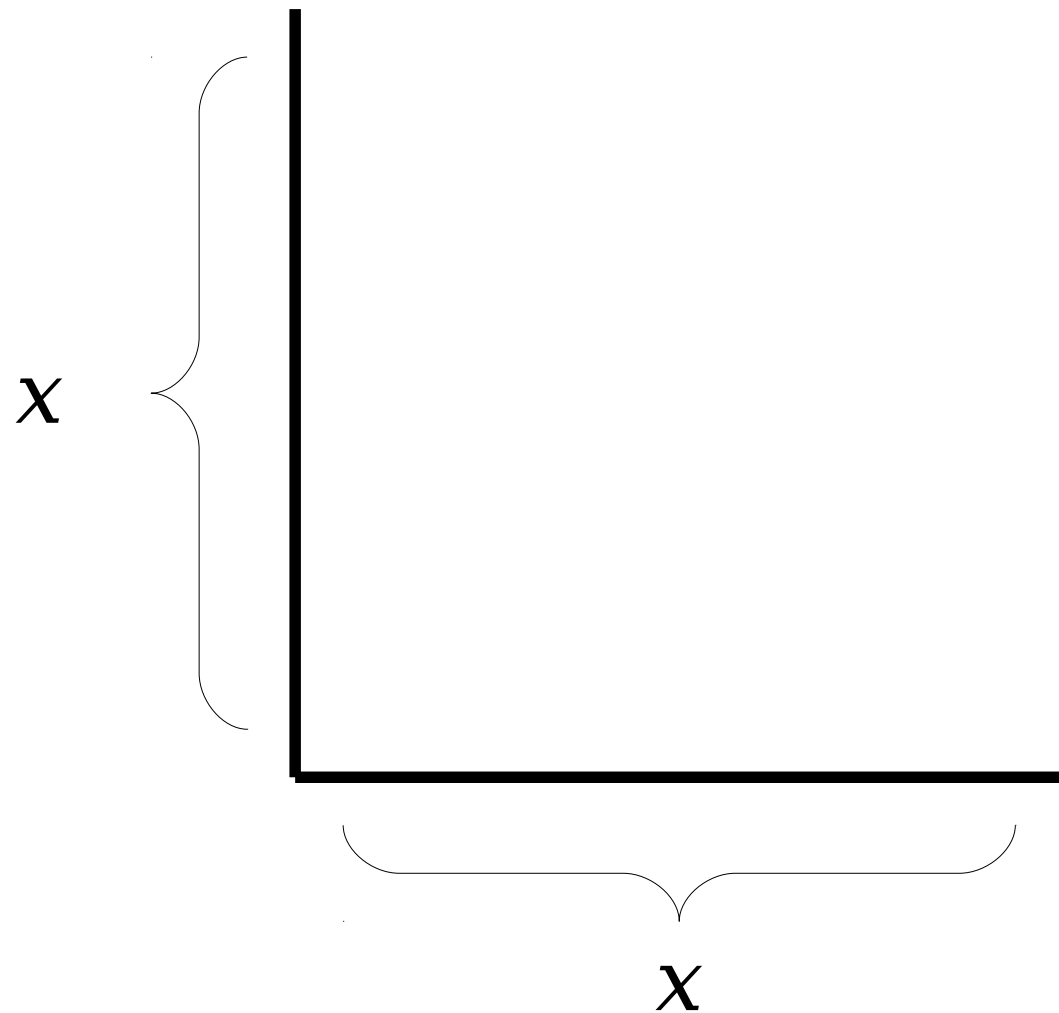
# Reasoning about Infinity

- 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.

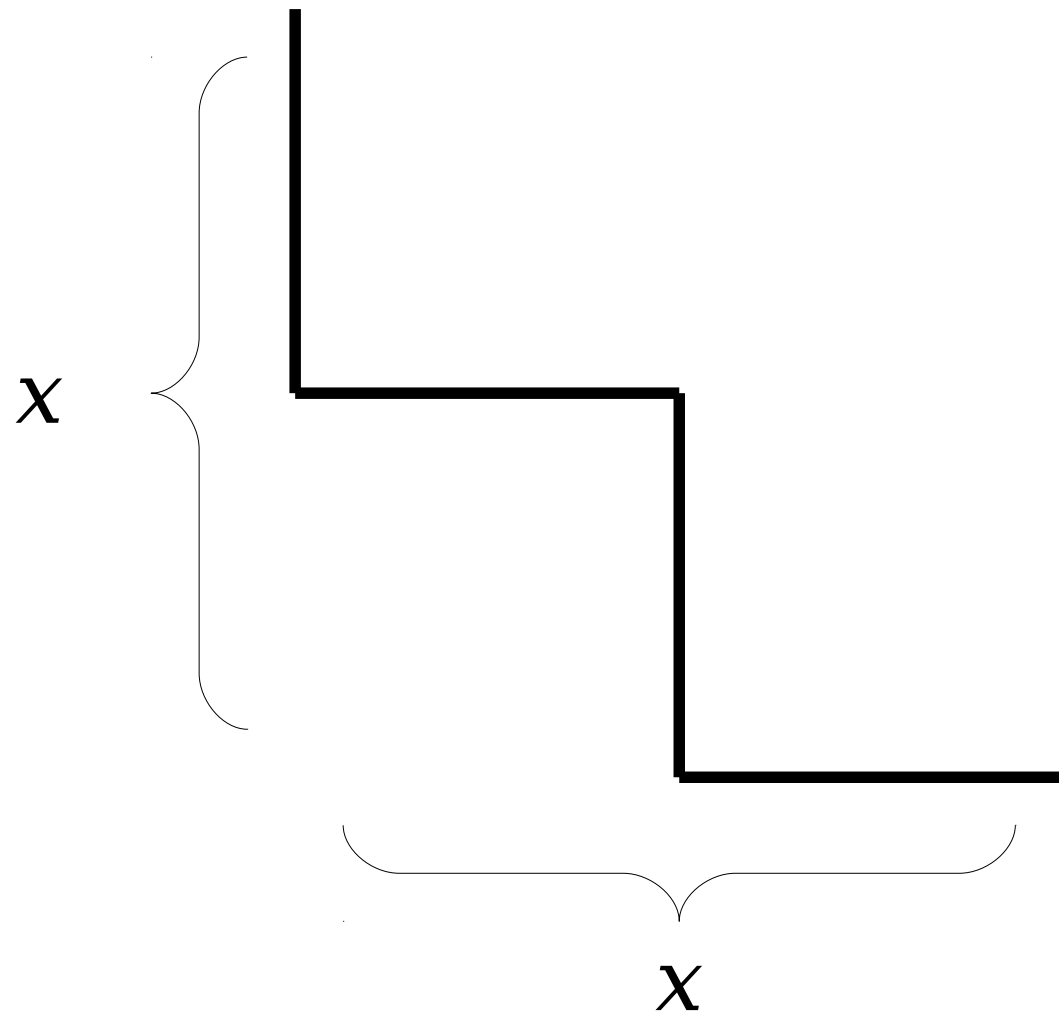
# Reasoning about Infinity



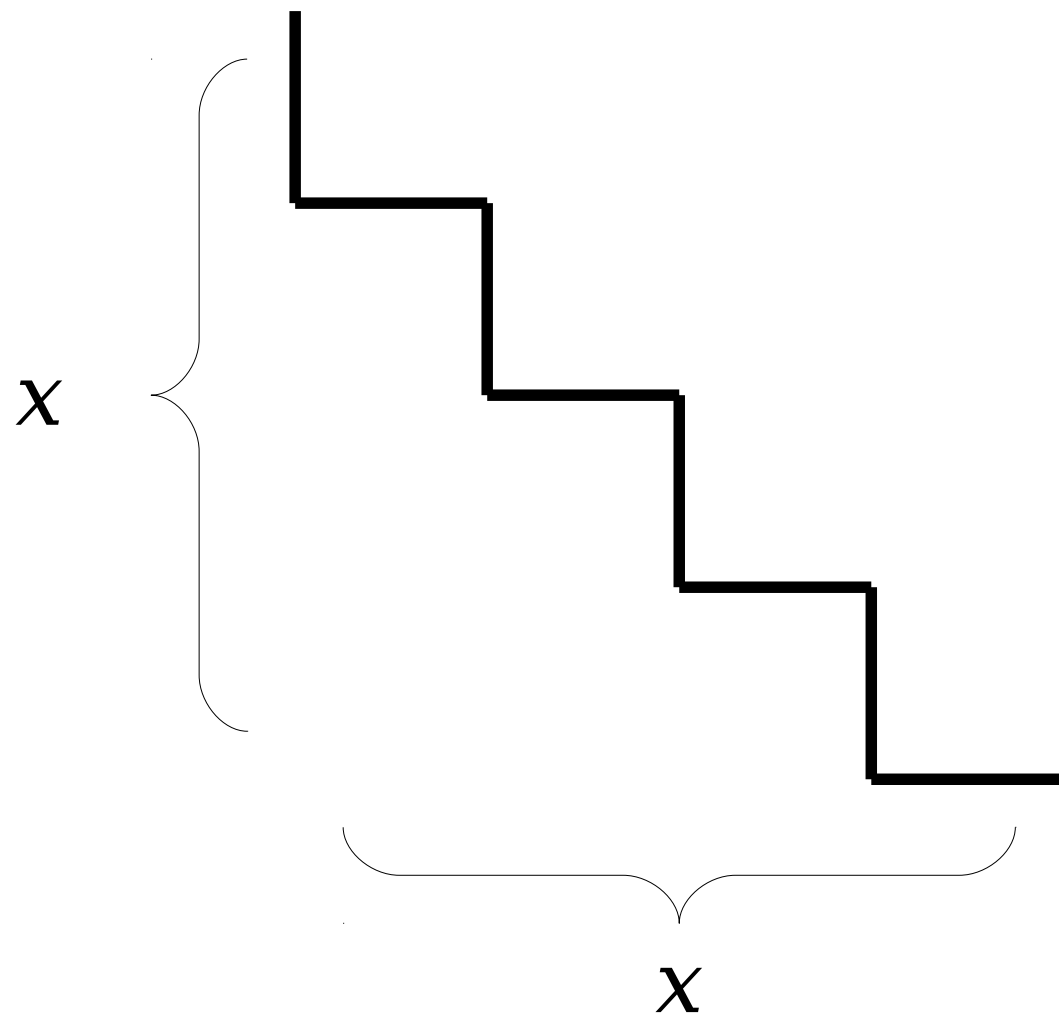
# Reasoning about Infinity



# Reasoning about Infinity

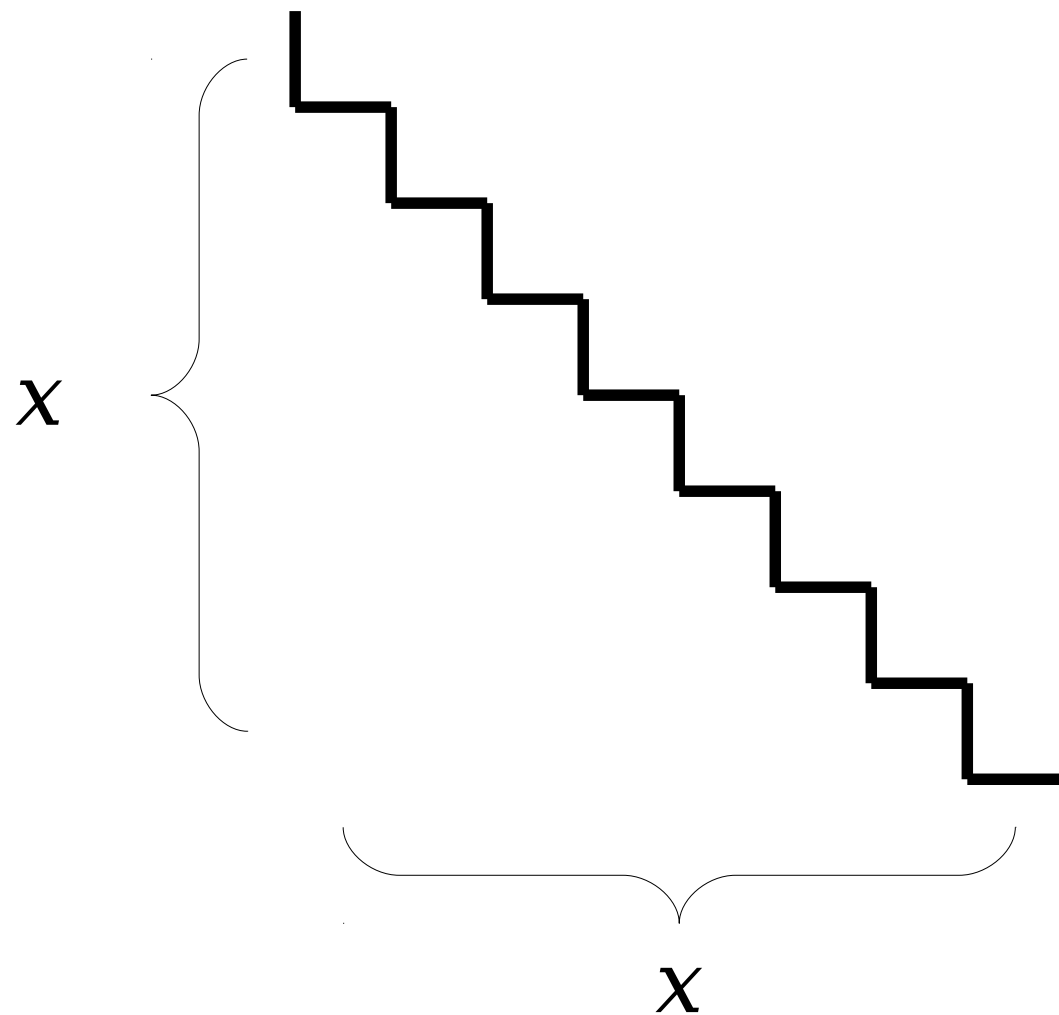


# Reasoning about Infinity

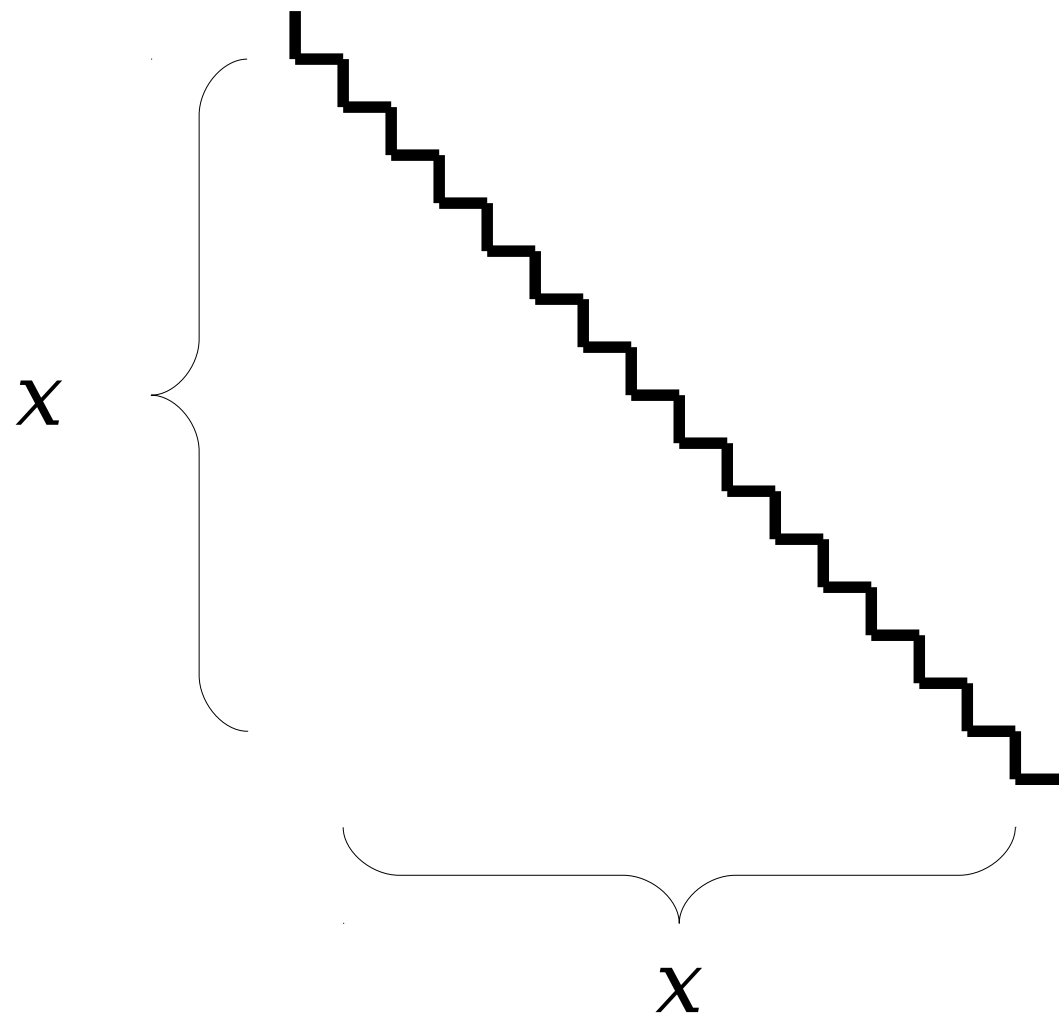




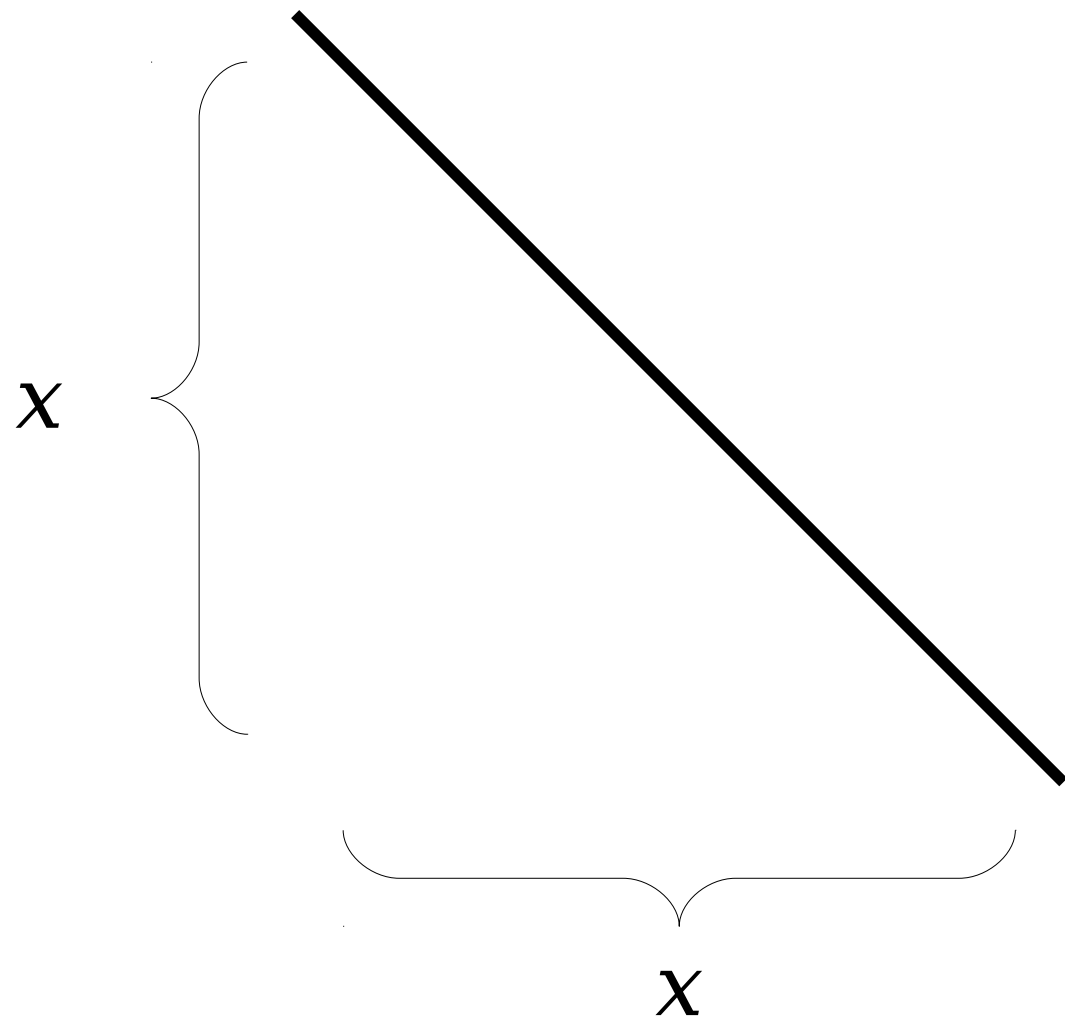
# Reasoning about Infinity



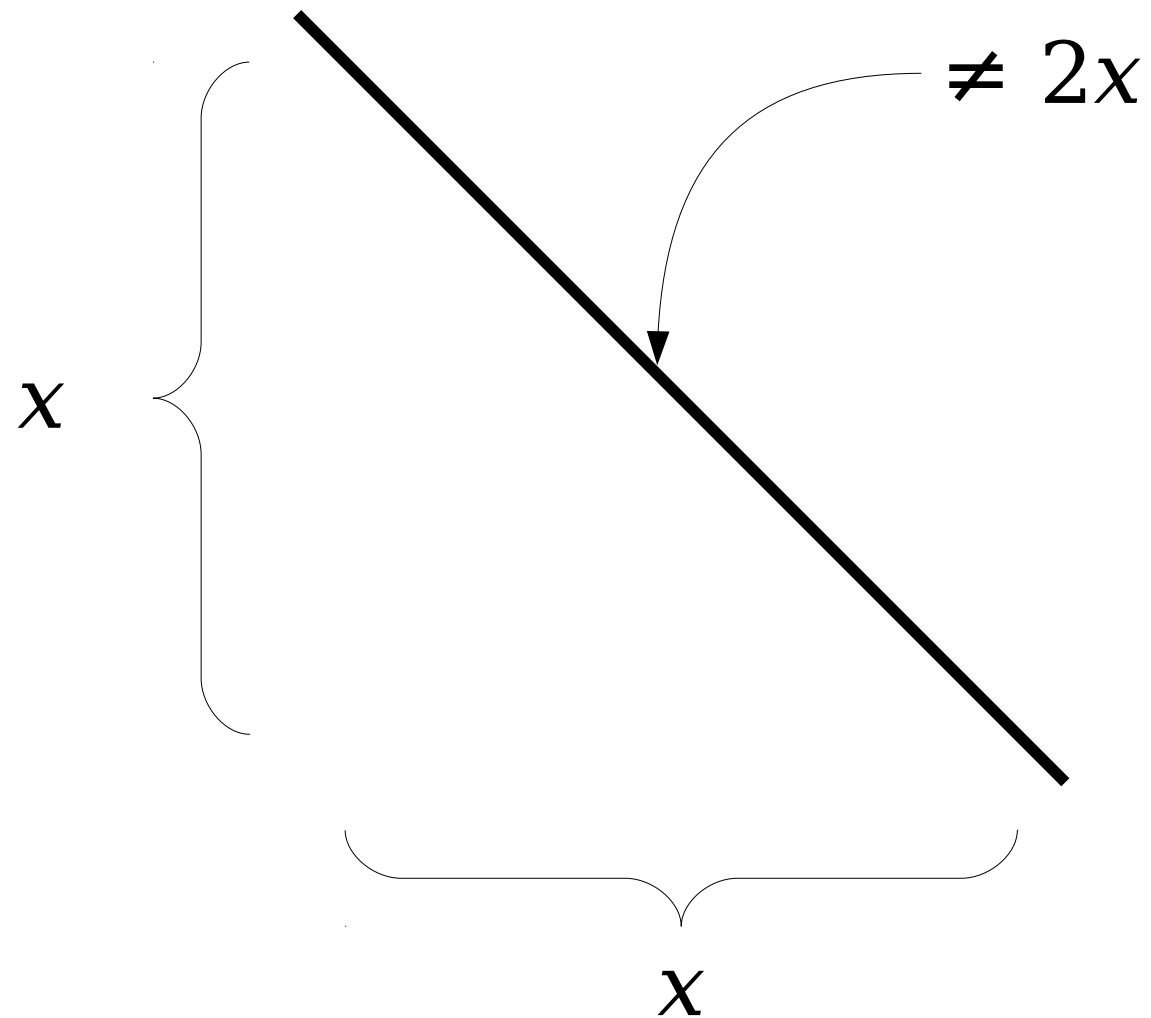
# Reasoning about Infinity



# 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

$$0.9999\overline{9} \not\leq 1$$

# 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

$\infty$  is finite

<sup>^</sup> not

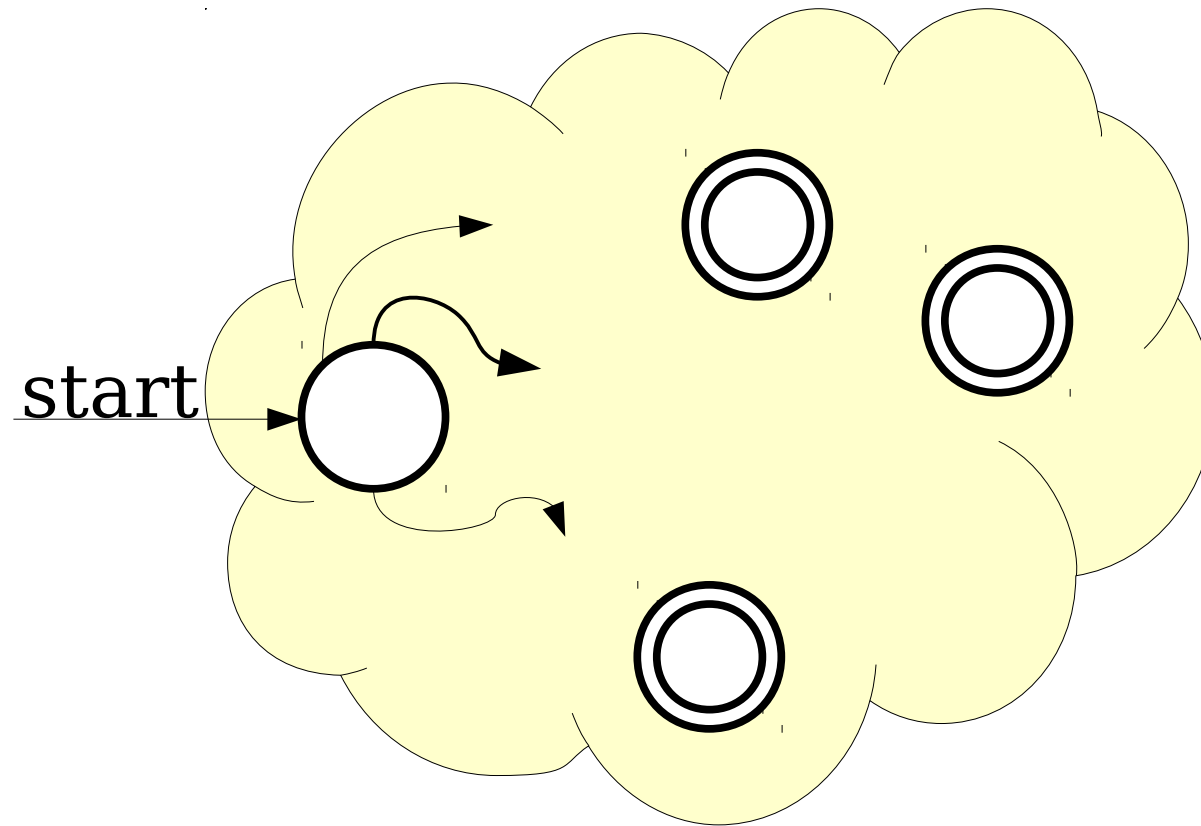


# Reasoning About the Infinite

- 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).
- So our earlier argument ( $L^* = L^0 \cup L^1 \cup \dots$ ) isn't going to work.
- We need a different line of reasoning.

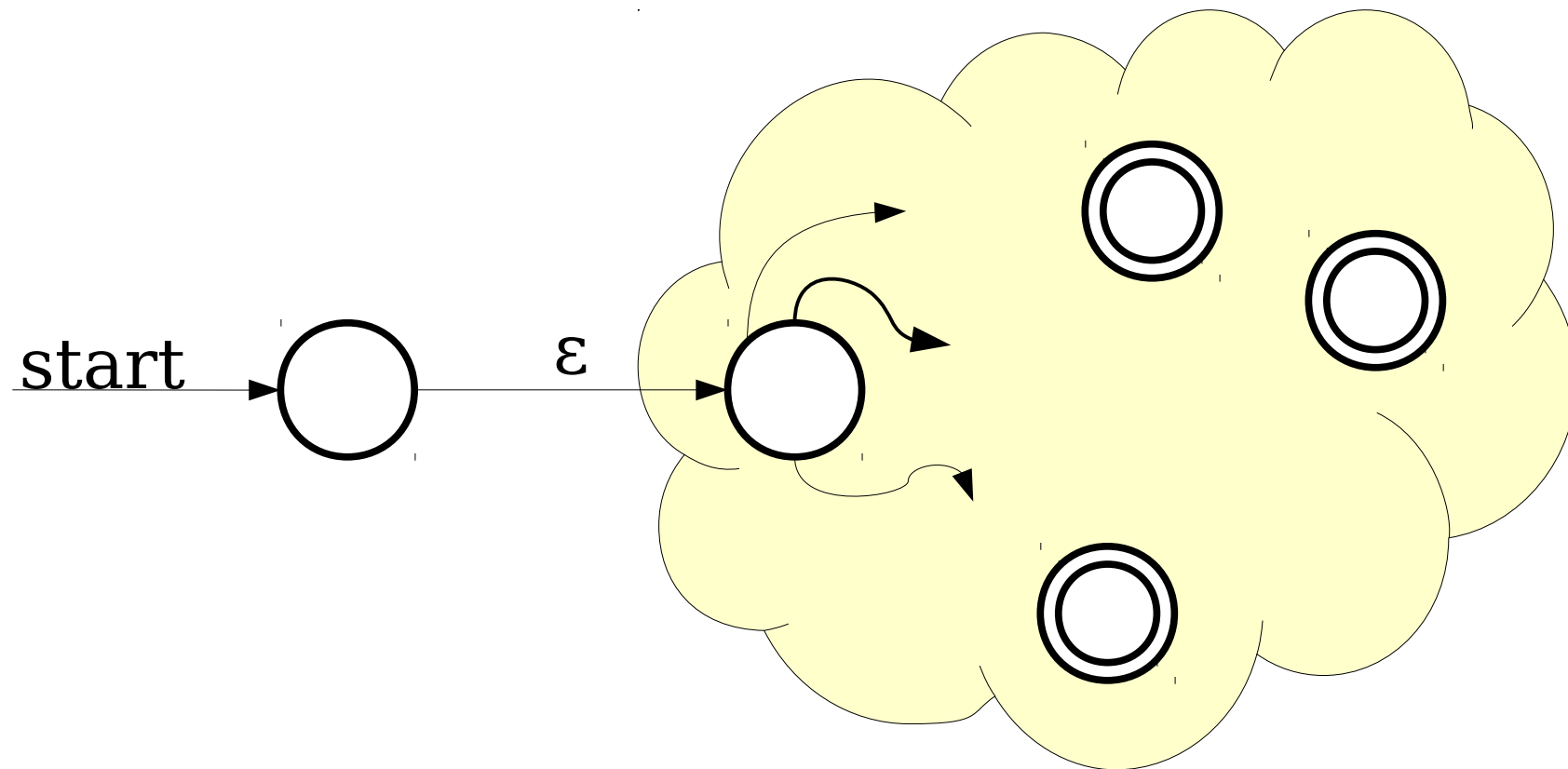
***Idea:*** Can we directly convert an NFA for language  $L$  to an NFA for language  $L^*$ ?

# The Kleene Star



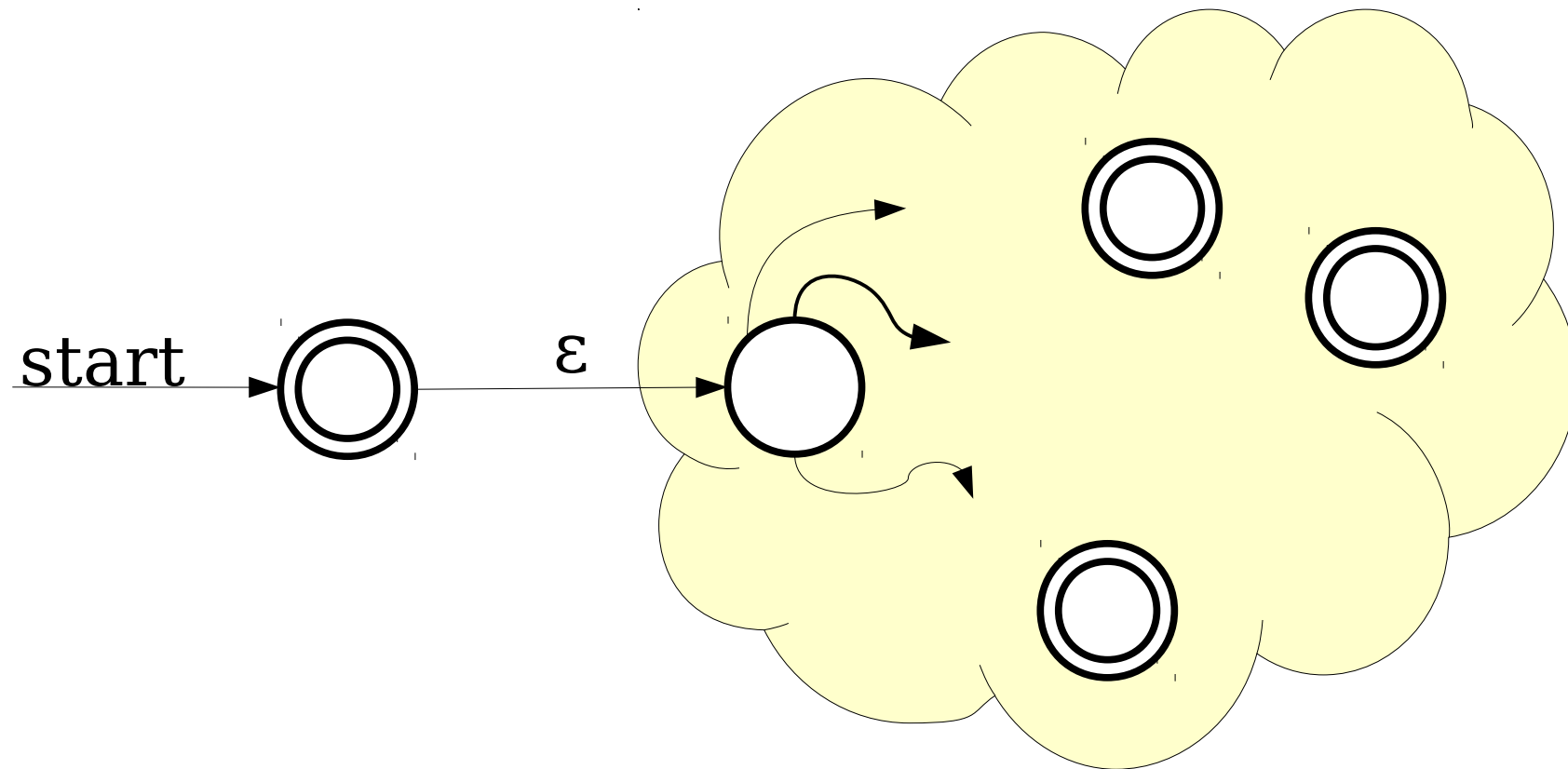
Machine for  $L$

# The Kleene Star



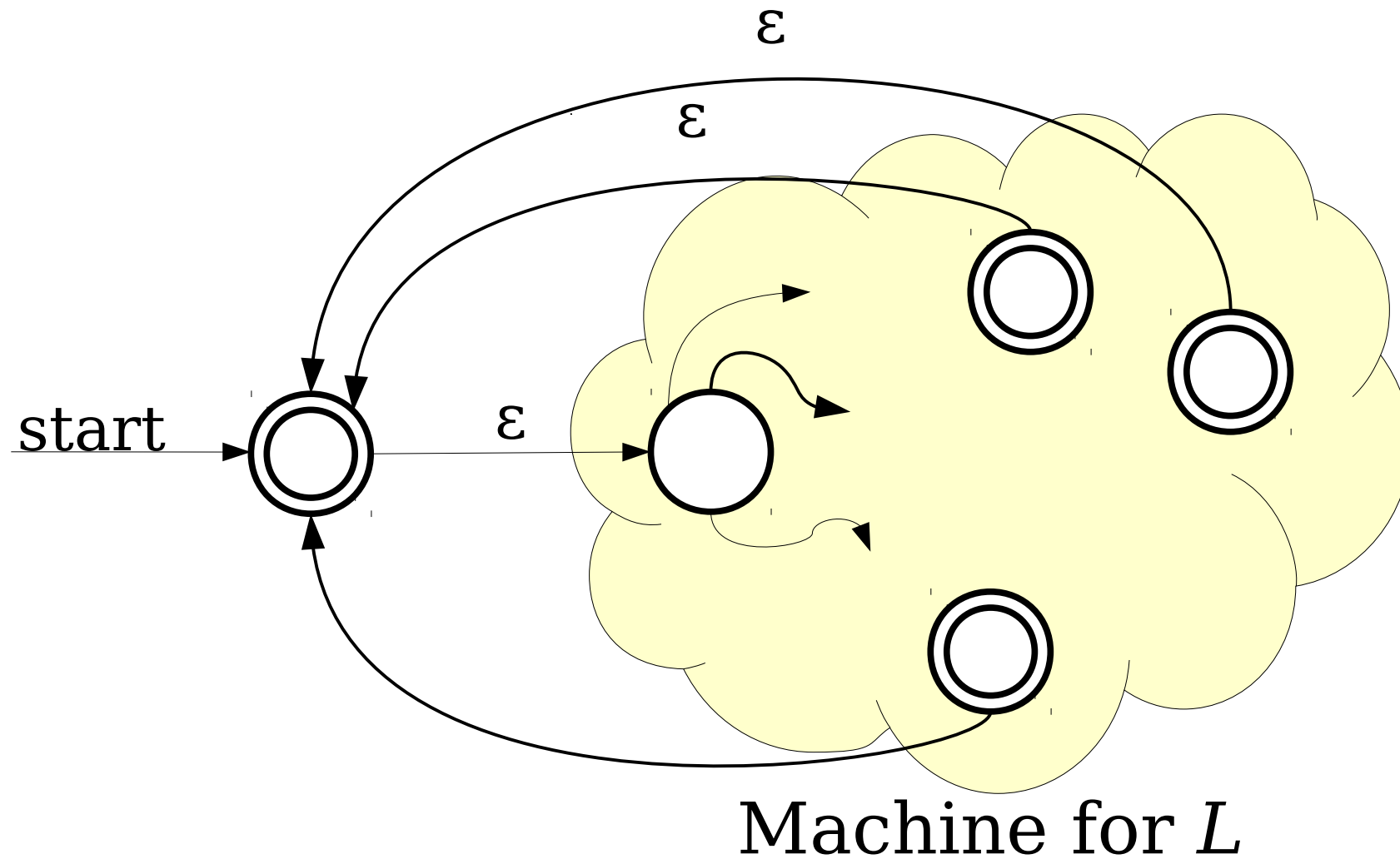
Machine for  $L$

# The Kleene Star

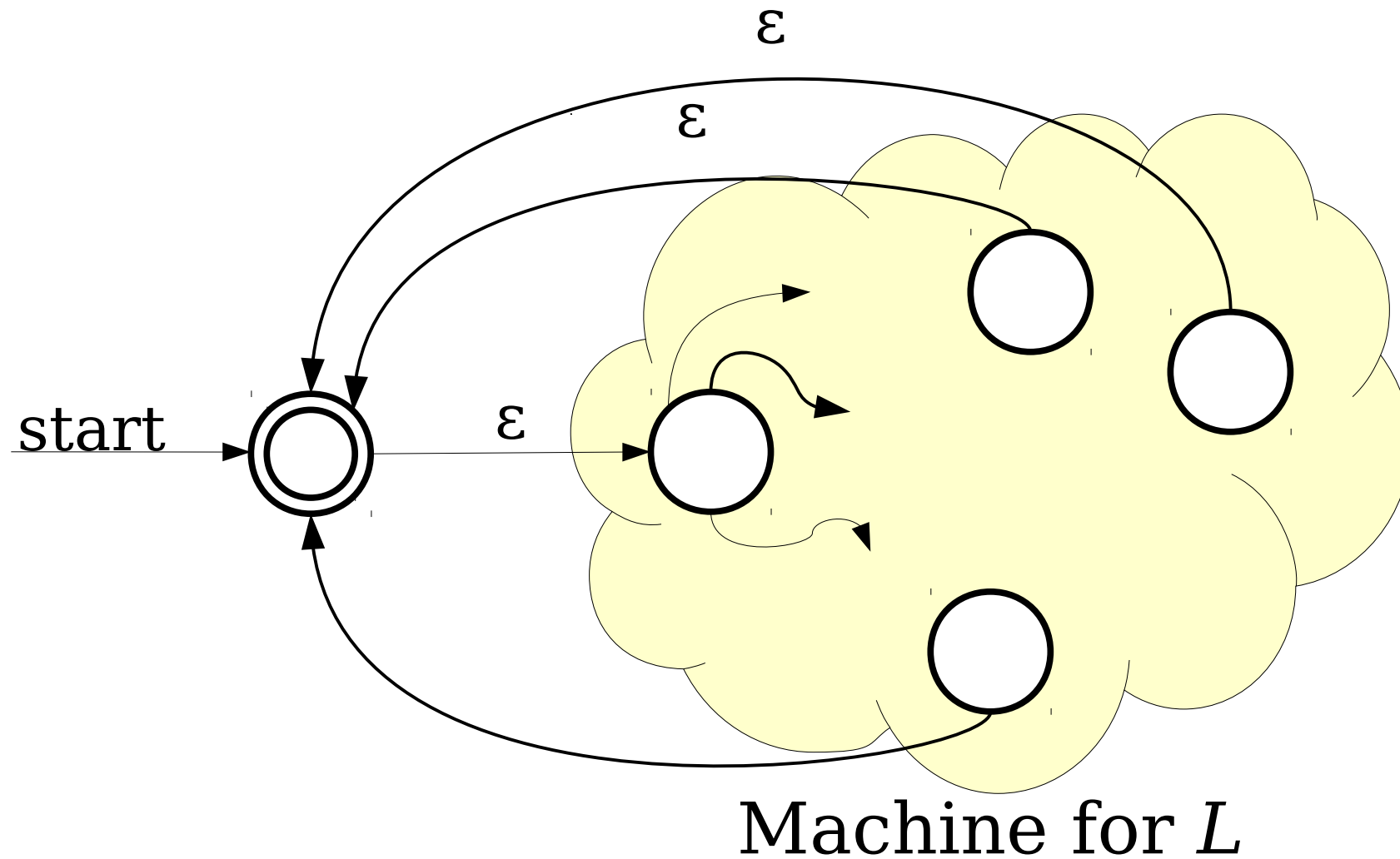


Machine for  $L$

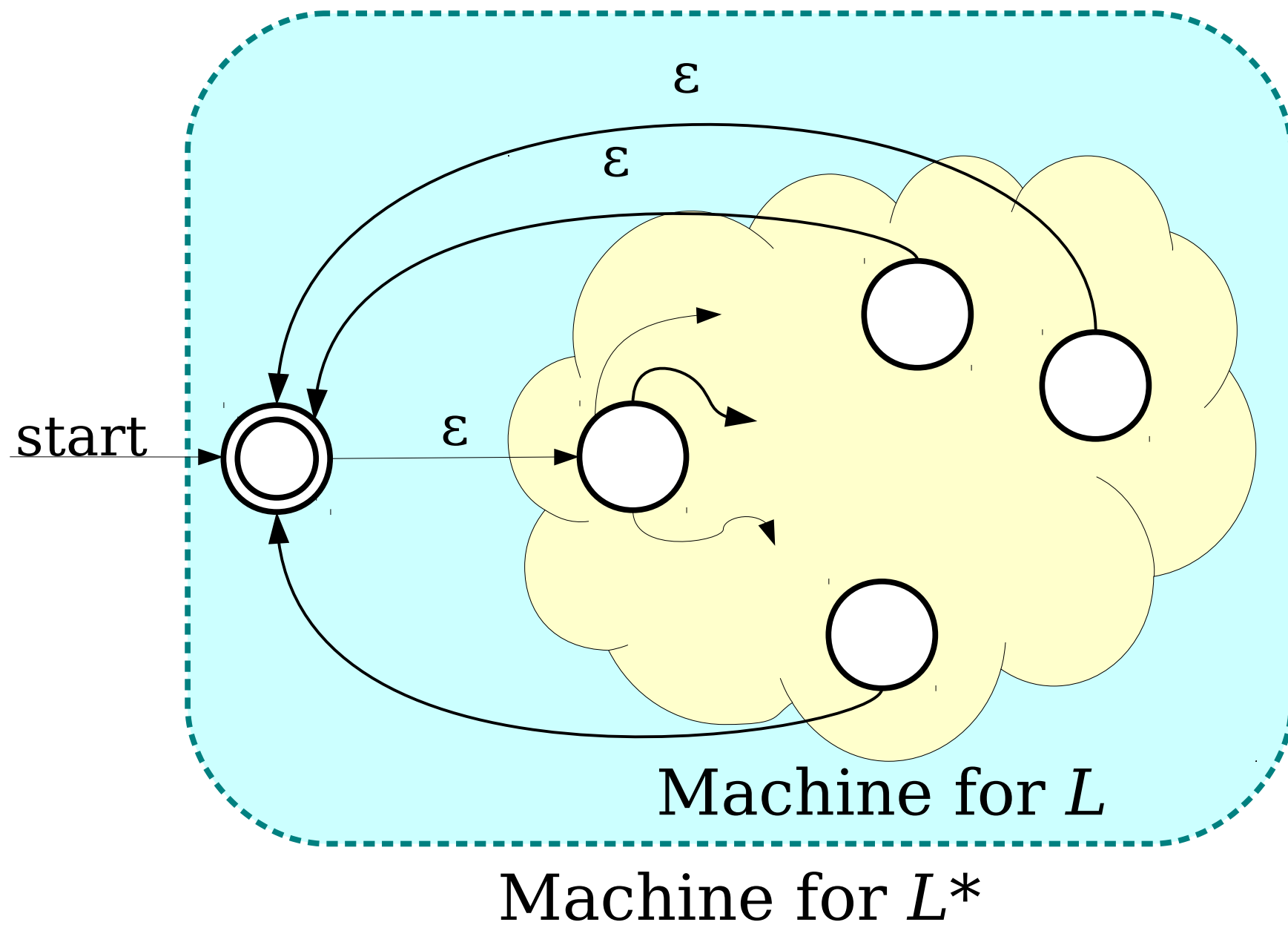
# The Kleene Star



# The Kleene Star

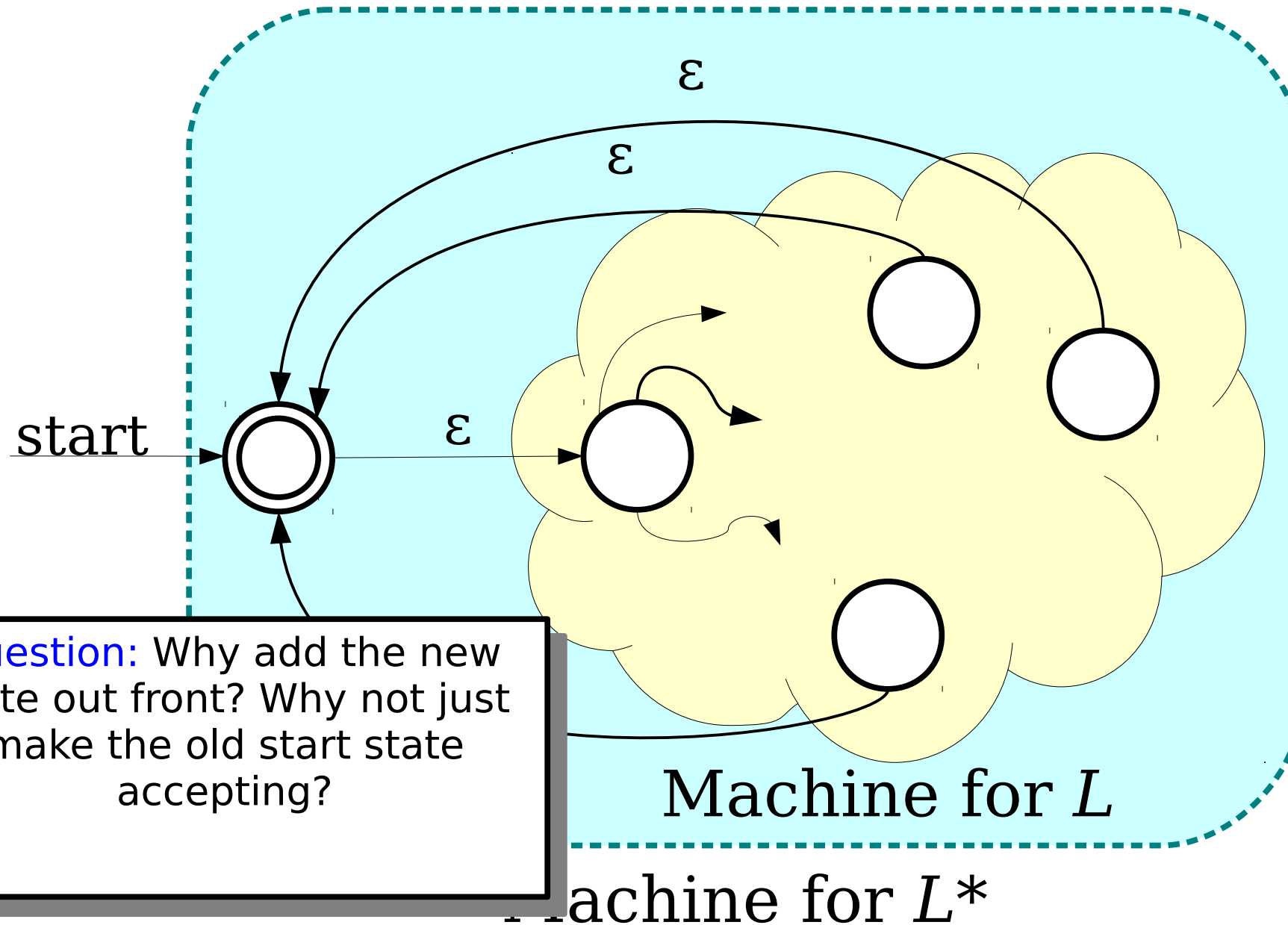


# The Kleene Star





# The Kleene Star



**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  $\Sigma$ , then so are the following languages:
  - $\overline{L_1}$
  - $L_1 \cup L_2$
  - $L_1 \cap L_2$
  - $L_1 L_2$
  - $L_1^*$
- These properties are called ***closure properties of the regular languages.***

# Next Time

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