CS 4240: Compilers

Lecture 23: Finite State Automata

Instructor: Vivek Sarkar
(vsarkar@gatech.edu)

April 15, 2019

ANNOUNCEMENTS & REMINDERS

- **Project 3 assigned on April 8th**
  - **Due by 11:59pm on Tuesday, April 23rd**
  - **Automatic penalty-free extension until 11:59pm on Tuesday, April 30th**
  - 10% of course grade
- **Homework 3 assigned today**
  - **Due by 11:59pm on Tuesday, April 23rd**
  - **Automatic penalty-free extension until 11:59pm on Friday, April 26th**
  - 5% of course grade
- **FINAL EXAM: Wednesday, May 1, 2:40 pm - 5:30 pm**
  - 30% of course grade

Worksheet # 22 Solution

(From Lecture #22 given on 4/10/2019)

$$
\begin{aligned}
S &\rightarrow \text{if } C \text{ then } S \text{ else } S && (1)\\
&\mid \text{if } C \text{ then } S && (2)\\
&\mid X && (3)\\
C &\rightarrow c_1 && (4)\\
&\mid c_2 && (5)\\
X &\rightarrow x_1 && (6)\\
&\mid x_2 && (7)
\end{aligned}
$$

a. Give two distinct right-most derivations for the string **`if c1 then if c2 then x1 else x2'.**

b. Can you also find two distinct left-most derivations for the above string? If so, provide both derivations. If not, explain why not.

$$
\begin{aligned}
S &\to \text{if } C \text{ then } S \text{ else } S && (1)\\
&\mid \text{if } C \text{ then } S && (2)\\
&\mid X && (3)\\
C &\to c_1 && (4)\\
&\mid c_2 && (5)\\
X &\to x_1 && (6)\\
&\mid x_2 && (7)
\end{aligned}
$$

- Two distinct **right-most derivations** for the string **`if c1 then if c2 then x1 else x2'.**

$$
\begin{aligned}
\underline{S} &\overset{1}{\to} \text{if } C \text{ then } S \text{ else } \underline{S}\\
&\overset{3}{\to} \text{if } C \text{ then } S \text{ else } \underline{X}\\
&\overset{7}{\to} \text{if } C \text{ then } \underline{S} \text{ else } x_2\\
&\overset{2}{\to} \text{if } C \text{ then if } C \text{ then } \underline{S} \text{ else } x_2\\
&\overset{3}{\to} \text{if } C \text{ then if } C \text{ then } \underline{X} \text{ else } x_2\\
&\overset{6}{\to} \text{if } C \text{ then if } \underline{C} \text{ then } x_1 \text{ else } x_2\\
&\overset{5}{\to} \text{if } \underline{C} \text{ then if } c_2 \text{ then } x_1 \text{ else } x_2\\
&\overset{4}{\to} \text{if } c_1 \text{ then if } c_2 \text{ then } x_1 \text{ else } x_2
\end{aligned}
$$

$$
\begin{aligned}
\underline{S} &\overset{2}{\to} \text{if } C \text{ then } \underline{S}\\
&\overset{1}{\to} \text{if } C \text{ then if } C \text{ then } S \text{ else } \underline{S}\\
&\overset{3}{\to} \text{if } C \text{ then if } C \text{ then } S \text{ else } \underline{X}\\
&\overset{7}{\to} \text{if } C \text{ then if } C \text{ then } \underline{S} \text{ else } x_2\\
&\overset{3}{\to} \text{if } C \text{ then if } C \text{ then } \underline{X} \text{ else } x_2\\
&\overset{6}{\to} \text{if } C \text{ then if } \underline{C} \text{ then } x_1 \text{ else } x_2\\
&\overset{5}{\to} \text{if } \underline{C} \text{ then if } c_2 \text{ then } x_1 \text{ else } x_2\\
&\overset{4}{\to} \text{if } c_1 \text{ then if } c_2 \text{ then } x_1 \text{ else } x_2
\end{aligned}
$$

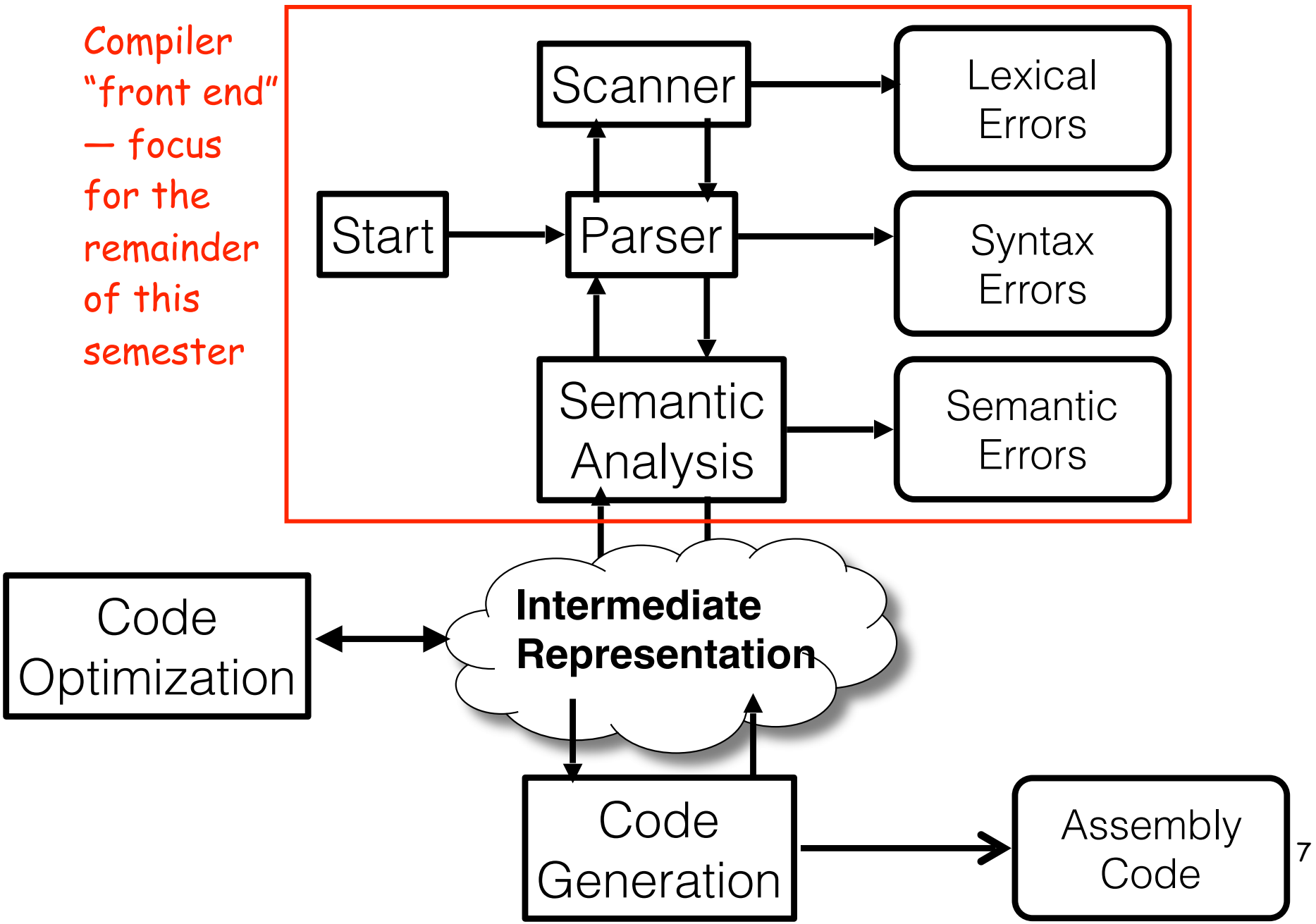
- Two distinct **left-most derivations** for the same string.

$$
\begin{aligned}
S &\to \text{if } C \text{ then } S \text{ else } S && (1)\\
&\mid \text{if } C \text{ then } S && (2)\\
&\mid X && (3)\\
C &\to c_1 && (4)\\
&\mid c_2 && (5)\\
X &\to x_1 && (6)\\
&\mid x_2 && (7)
\end{aligned}
$$

$$
\begin{aligned}
\underline{S} &\xrightarrow{1} \text{if } \underline{C} \text{ then } S \text{ else } S\\
&\xrightarrow{4} \text{if } c_1 \text{ then } \underline{S} \text{ else } S\\
&\xrightarrow{2} \text{if } c_1 \text{ then if } \underline{C} \text{ then } S \text{ else } S\\
&\xrightarrow{5} \text{if } c_1 \text{ then if } c_2 \text{ then } \underline{S} \text{ else } S\\
&\xrightarrow{3} \text{if } c_1 \text{ then if } c_2 \text{ then } \underline{X} \text{ else } S\\
&\xrightarrow{6} \text{if } c_1 \text{ then if } c_2 \text{ then } x_1 \text{ else } \underline{S}\\
&\xrightarrow{3} \text{if } c_1 \text{ then if } c_2 \text{ then } x_1 \text{ else } \underline{X}\\
&\xrightarrow{7} \text{if } c_1 \text{ then if } c_2 \text{ then } x_1 \text{ else } x_2
\end{aligned}
$$

$$
\begin{aligned}
\underline{S} &\xrightarrow{2} \text{if } \underline{C} \text{ then } S\\
&\xrightarrow{4} \text{if } c_1 \text{ then } \underline{S}\\
&\xrightarrow{1} \text{if } c_1 \text{ then if } \underline{C} \text{ then } S \text{ else } S\\
&\xrightarrow{5} \text{if } c_1 \text{ then if } c_2 \text{ then } \underline{S} \text{ else } S\\
&\xrightarrow{3} \text{if } c_1 \text{ then if } c_2 \text{ then } \underline{X} \text{ else } S\\
&\xrightarrow{6} \text{if } c_1 \text{ then if } c_2 \text{ then } x_1 \text{ else } \underline{S}\\
&\xrightarrow{3} \text{if } c_1 \text{ then if } c_2 \text{ then } x_1 \text{ else } \underline{X}\\
&\xrightarrow{7} \text{if } c_1 \text{ then if } c_2 \text{ then } x_1 \text{ else } x_2
\end{aligned}
$$

Structure of a Full Compiler (Recap from Lecture 1)



Front-end architecture

- Scanning: converting source code into stream of known chunks called tokens
  - Lexical rules of language dictate how legal token is formed as a sequence of alphabet symbols
    - Formalized using regular expressions
    - **TODAY'S LECTURE: how to automatically build a scanner using finite state automata**
- Parsing: building tree-based representation of code
  - Grammar dictates how legal tree is formed as a sequence of tokens
    - Formalized using context-free grammars

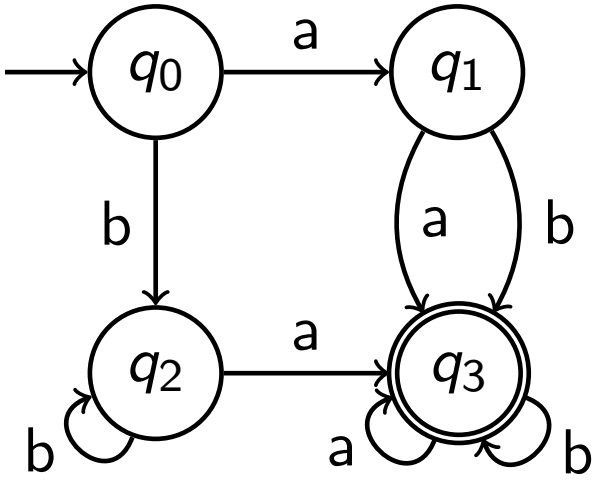
Deterministic/Non-deterministic Finite Automaton

What is one made out of?

- finite set of states, Q
- transition function (labeled, directed edges), δ
- some states are special, q_0, Q_f

How does it *behave*?

- string is fed in one char at a time
- move from state to state, according to δ
- at the end, decide to accept or reject string



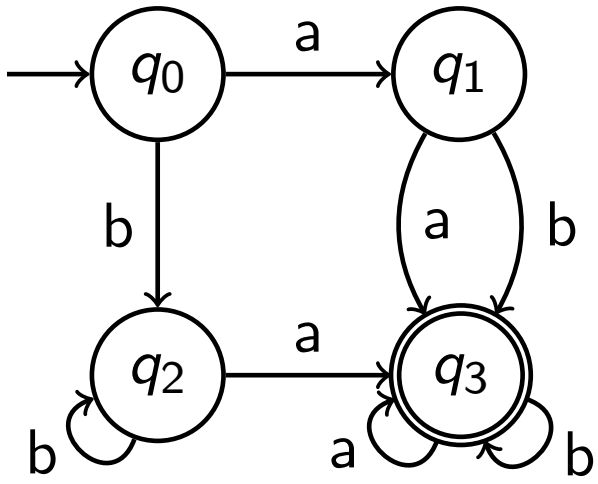
Deterministic/Non-deterministic Finite Automaton

What is one made out of?

- finite set of states, Q
- transition function (labeled, directed edges), δ
- some states are special, q_0, Q_f

How does it *behave*?

- string is fed in one char at a time
- move from state to state, according to δ
- at the end, decide to accept or reject string



$\langle q_0 \rangle$bbaba

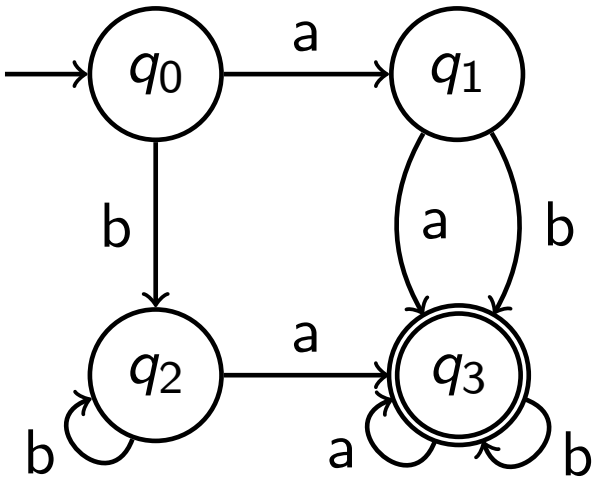
Deterministic/Non-deterministic Finite Automaton

What is one made out of?

- finite set of states, Q
- transition function (labeled, directed edges), δ
- some states are special, q_0, Q_f

How does it *behave*?

- string is fed in one char at a time
- move from state to state, according to δ
- at the end, decide to accept or reject string



$\texttt{b}\langle q_2 \rangle \texttt{baba}$

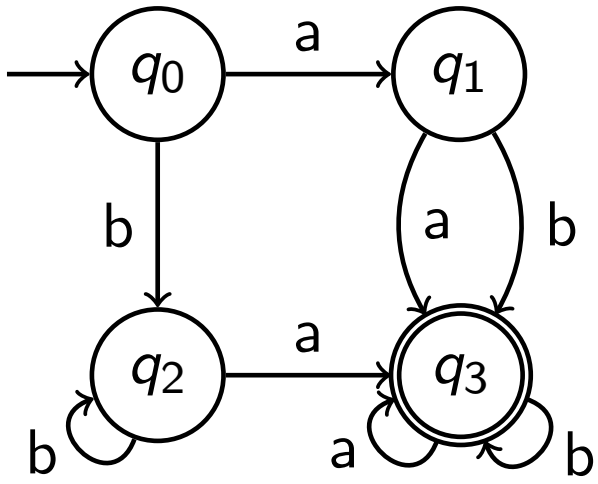
Deterministic/Non-deterministic Finite Automaton

What is one made out of?

- finite set of states, Q
- transition function (labeled, directed edges), δ
- some states are special, q_0, Q_f

How does it *behave*?

- string is fed in one char at a time
- move from state to state, according to δ
- at the end, decide to accept or reject string



$\texttt{bb}\langle q_2 \rangle\texttt{aba}$

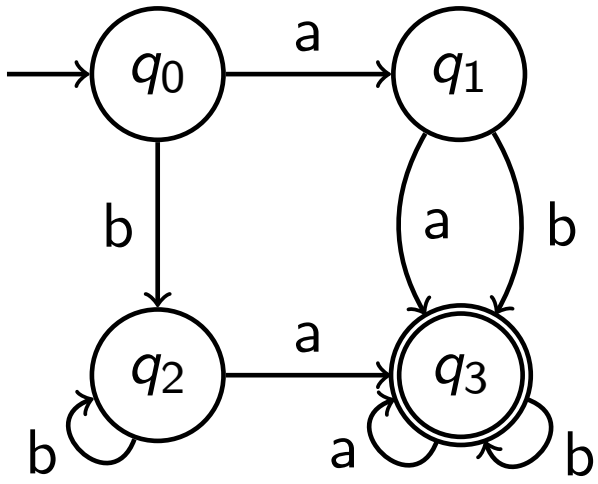
Deterministic/Non-deterministic Finite Automaton

What is one made out of?

- finite set of states, Q
- transition function (labeled, directed edges), δ
- some states are special, q_0, Q_f

How does it *behave*?

- string is fed in one char at a time
- move from state to state, according to δ
- at the end, decide to accept or reject string



$\texttt{bba}\langle q_3 \rangle \texttt{ba}$

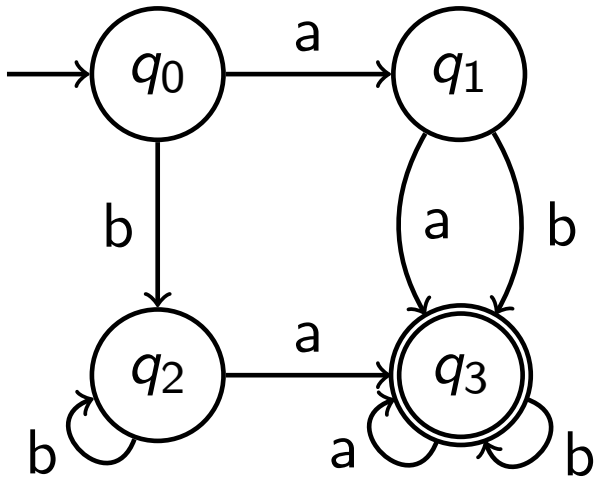
Deterministic/Non-deterministic Finite Automaton

What is one made out of?

- finite set of states, Q
- transition function (labeled, directed edges), δ
- some states are special, q_0, Q_f

How does it *behave*?

- string is fed in one char at a time
- move from state to state, according to δ
- at the end, decide to accept or reject string



bbab$\langle q_3 \rangle$a

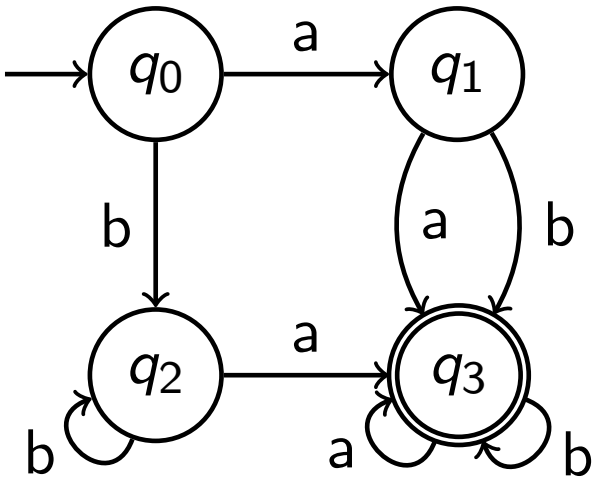
Deterministic/Non-deterministic Finite Automaton

What is one made out of?

- finite set of states, Q
- transition function (labeled, directed edges), δ
- some states are special, q_0, Q_f

How does it *behave*?

- string is fed in one char at a time
- move from state to state, according to δ
- at the end, decide to accept or reject string



bbaba$\langle q_3 \rangle$ ✓

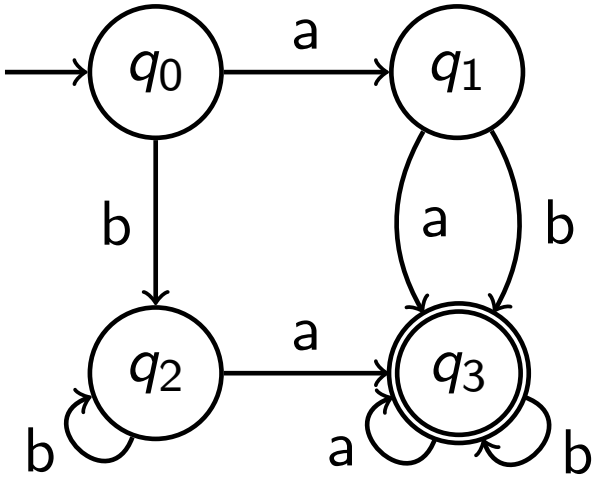
Deterministic/Non-deterministic Finite Automaton

What is one made out of?

- finite set of states, Q
- transition function (labeled, directed edges), δ
- some states are special, q_0, Q_f

How does it *behave*?

- string is fed in one char at a time
- move from state to state, according to δ
- at the end, decide to accept or reject string



bbaba$\langle q_3 \rangle$ ✓

$\langle q_0 \rangle$bbb

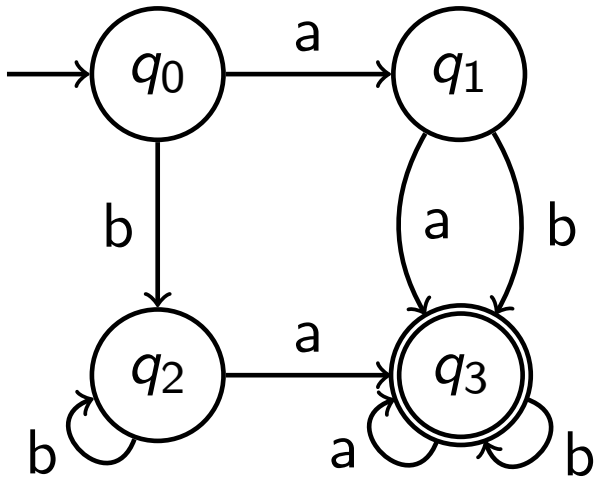
Deterministic/Non-deterministic Finite Automaton

What is one made out of?

- finite set of states, Q
- transition function (labeled, directed edges), δ
- some states are special, q_0, Q_f

How does it *behave*?

- string is fed in one char at a time
- move from state to state, according to δ
- at the end, decide to accept or reject string



bbaba$\langle q_3 \rangle$ ✓

b$\langle q_2 \rangle$bb

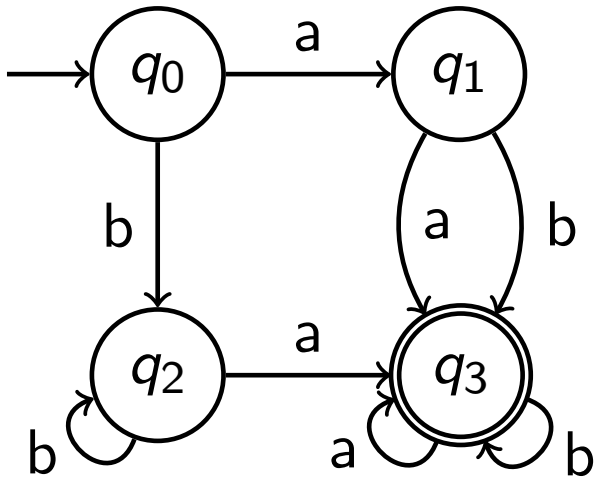
Deterministic/Non-deterministic Finite Automaton

What is one made out of?

- finite set of states, Q
- transition function (labeled, directed edges), δ
- some states are special, q_0, Q_f

How does it *behave*?

- string is fed in one char at a time
- move from state to state, according to δ
- at the end, decide to accept or reject string



bbaba$\langle q_3 \rangle$ ✓

bb$\langle q_2 \rangle$b

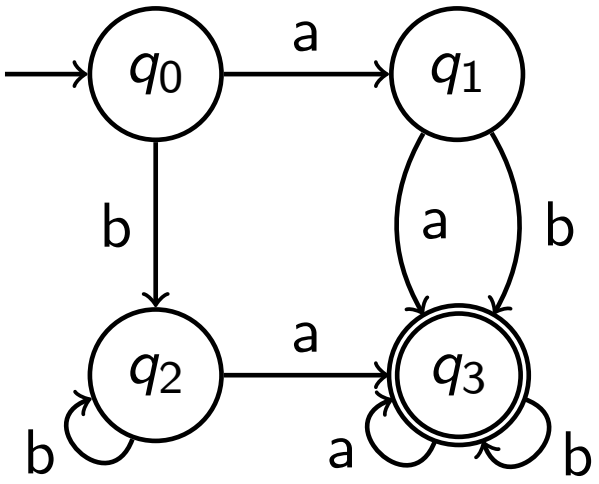
Deterministic/Non-deterministic Finite Automaton

What is one made out of?

- finite set of states, Q
- transition function (labeled, directed edges), δ
- some states are special, q_0, Q_f

How does it *behave*?

- string is fed in one char at a time
- move from state to state, according to δ
- at the end, decide to accept or reject string



bbaba$\langle q_3 \rangle$ ✓

bbb$\langle q_2 \rangle$ ✗

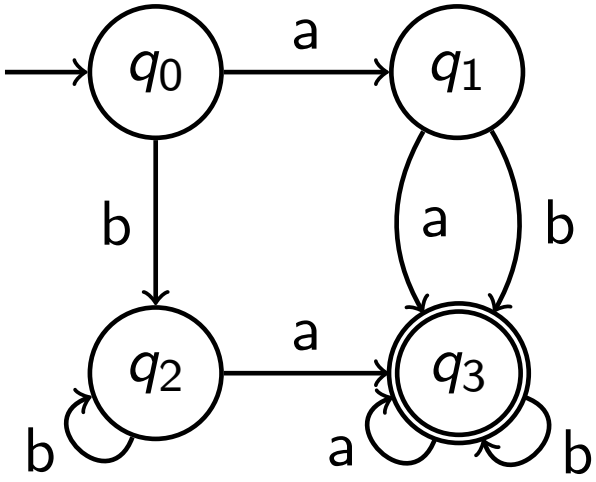
Deterministic/Non-deterministic Finite Automaton

What is one made out of?

- finite set of states, Q
- transition function (labeled, directed edges), δ
- some states are special, q_0, Q_f

How does it *behave*?

- string is fed in one char at a time
- move from state to state, according to δ
- at the end, decide to accept or reject string



bbaba$\langle q_3 \rangle$ ✓

bbb$\langle q_2 \rangle$ ✗

ε ?

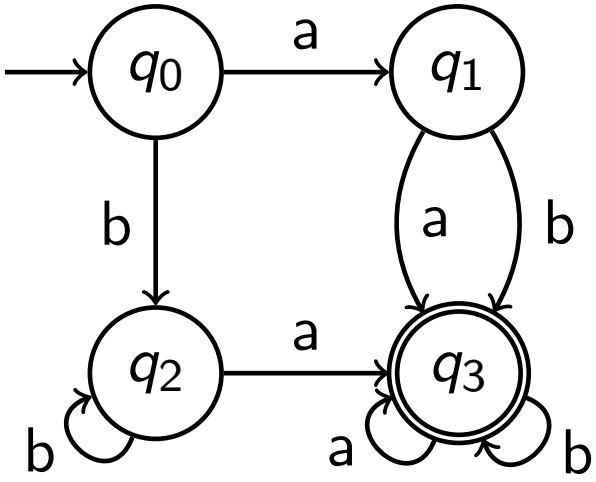
Deterministic/Non-deterministic Finite Automaton

What is one made out of?

- finite set of states, Q
- transition function (labeled, directed edges), δ
- some states are special, q_0, Q_f

How does it *behave*?

- string is fed in one char at a time
- move from state to state, according to δ
- at the end, decide to accept or reject string

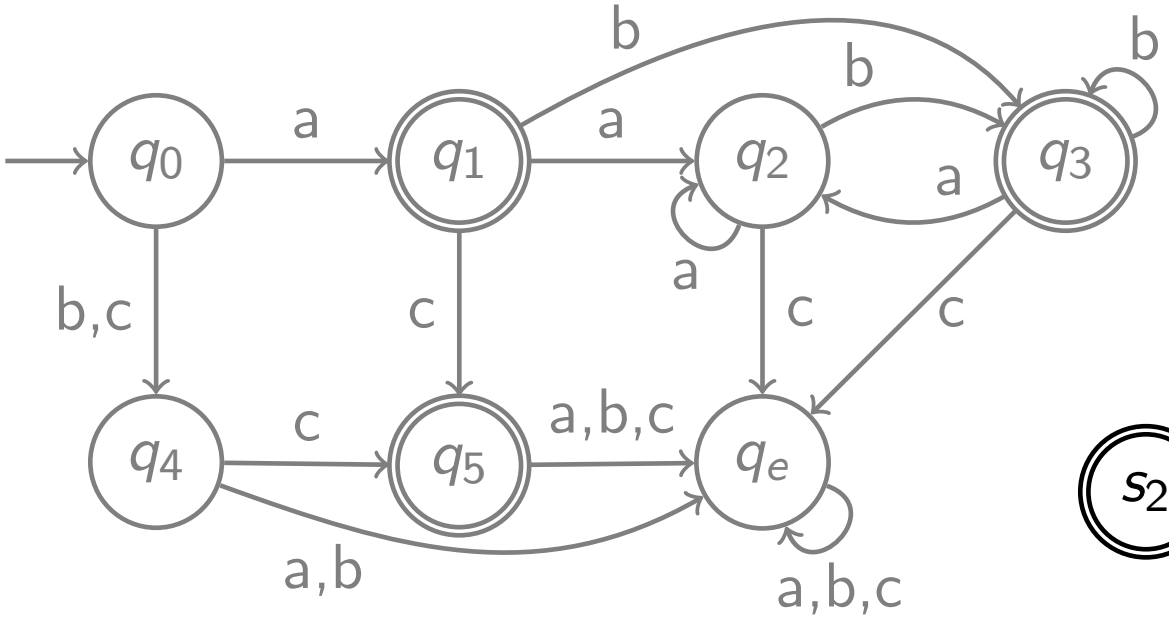


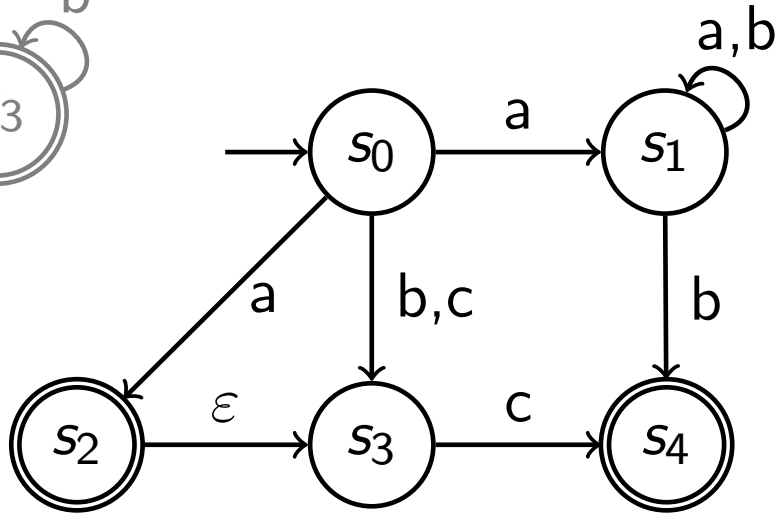
bbaba$\langle q_3 \rangle$ ✓
bbb$\langle q_2 \rangle$ ✗
$\langle q_0 \rangle$ ✗

NOTE: for convenience, a missing transition indicates an error if that symbol is encountered (in lieu of creating an explicit error state like q_e)

DFAs:

- *at most one* outgoing transition per $a \in \Sigma$
- decide acceptance directly



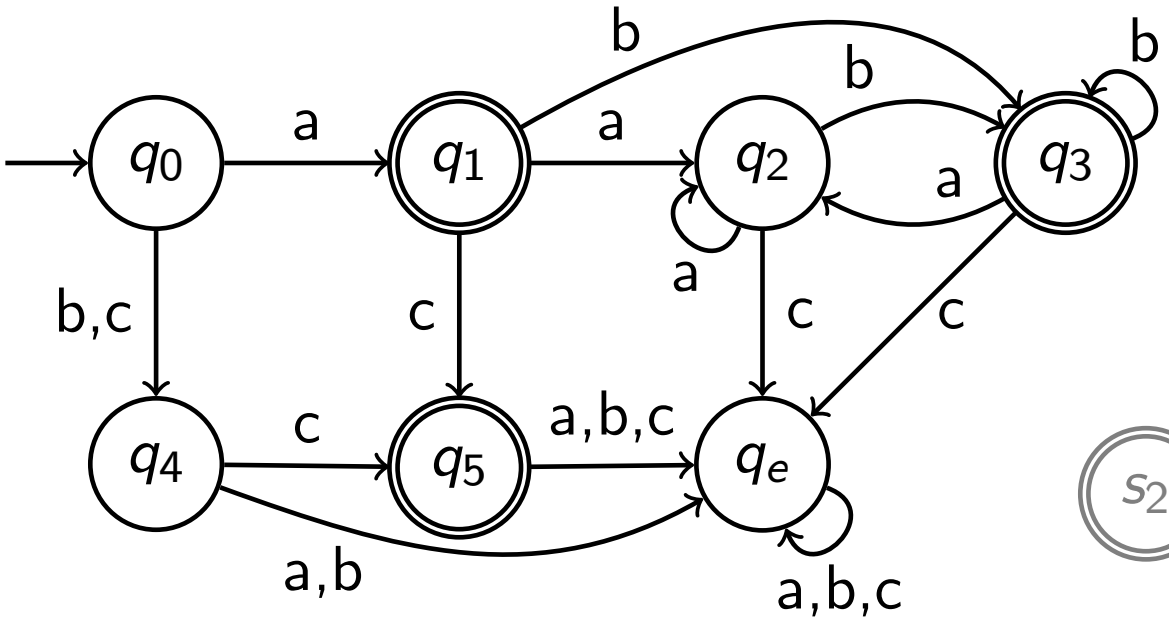


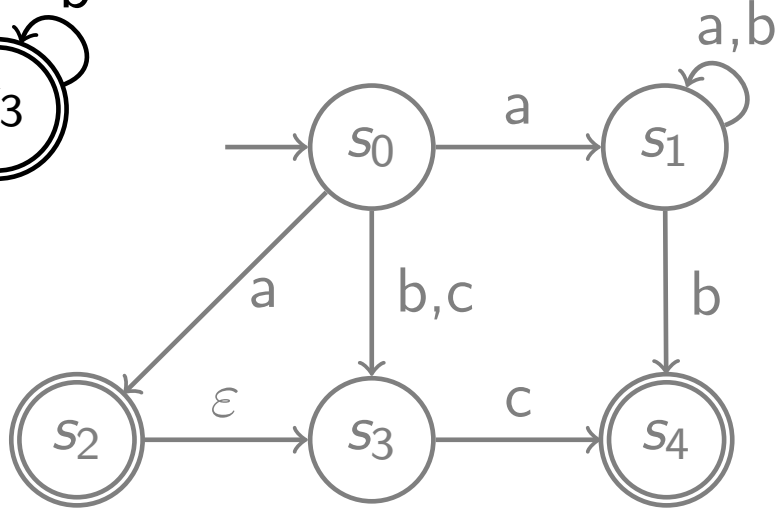
NFAs:

- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

DFAs:

- *at most one* outgoing transition per $a \in \Sigma$ $\qquad\qquad \langle q_0 \rangle \mathtt{abc}$
- decide acceptance directly



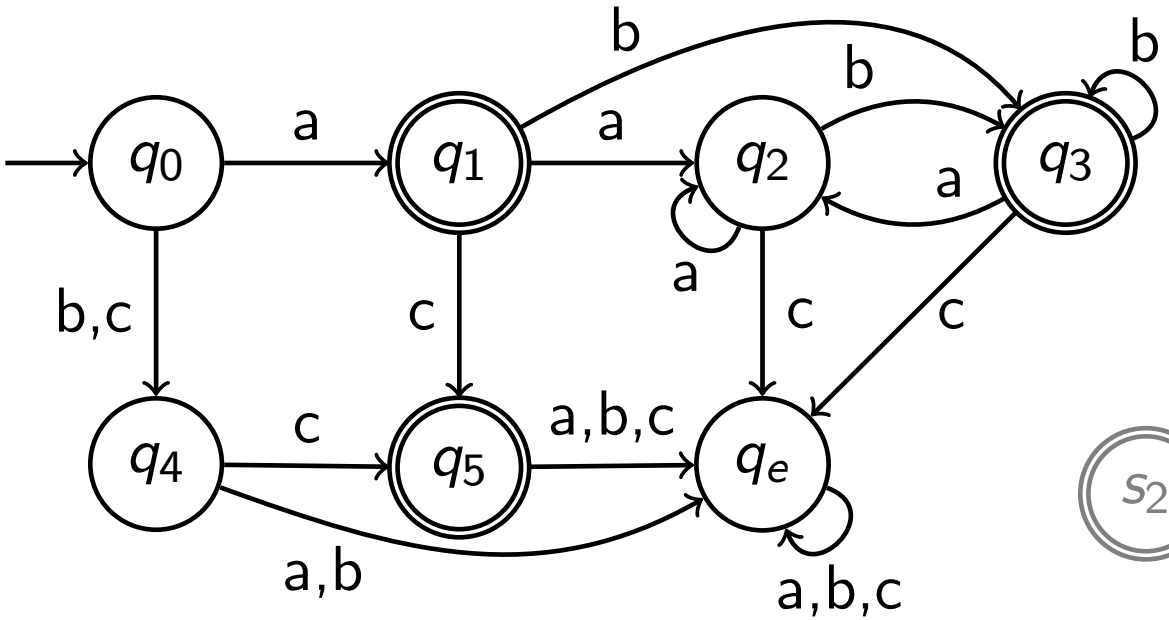


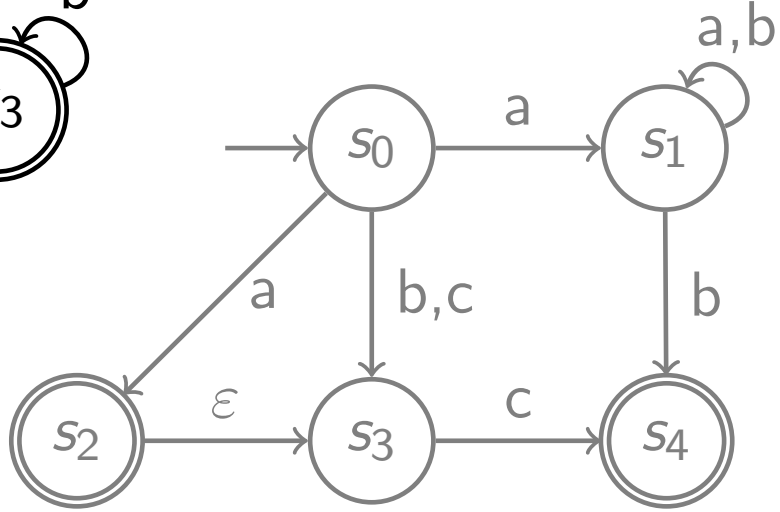
NFAs:

- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

DFAs:

- *at most one* outgoing transition per $a \in \Sigma$ $\qquad\qquad$ $\mathrm{a}\langle q_1 \rangle \mathrm{bc}$
- decide acceptance directly



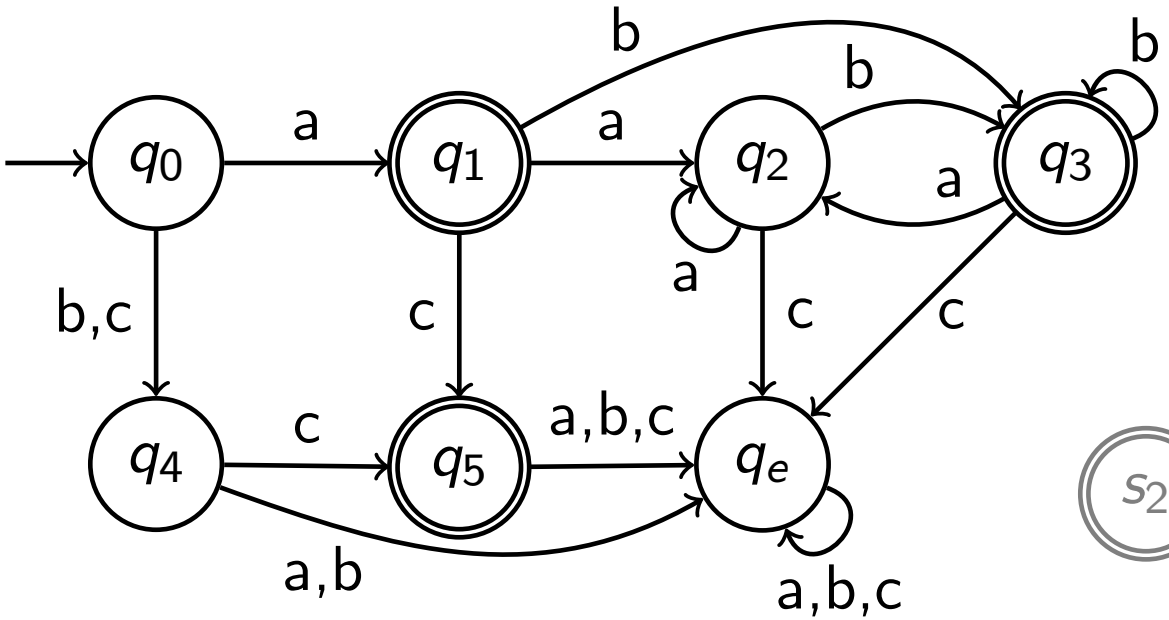


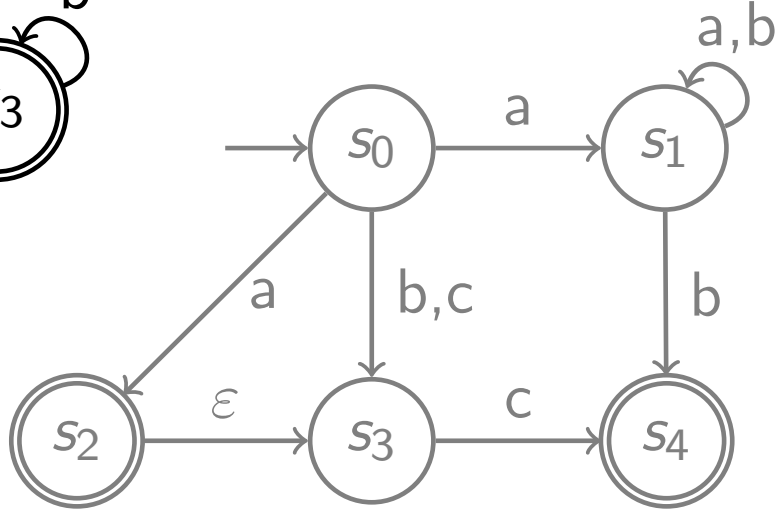
NFAs:

- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

DFAs:

- *at most one* outgoing transition per $a \in \Sigma$ $\qquad\qquad$ $\mathtt{ab}\langle q_3 \rangle \mathtt{c}$
- decide acceptance directly



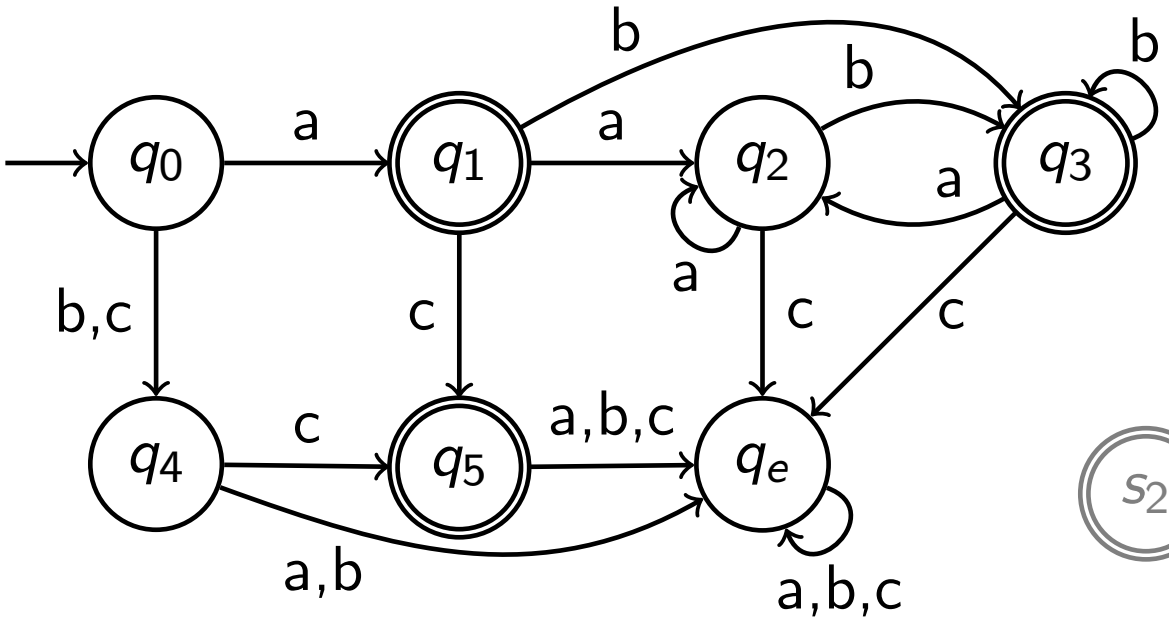


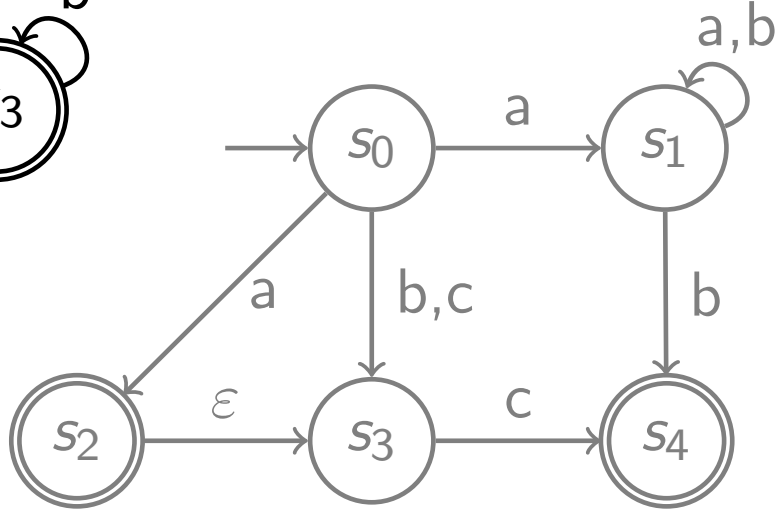
NFAs:

- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

DFAs:

- *at most one* outgoing transition per $a \in \Sigma$ $\quad\quad$ $\mathtt{abc}\langle q_e \rangle$ ✗
- decide acceptance directly





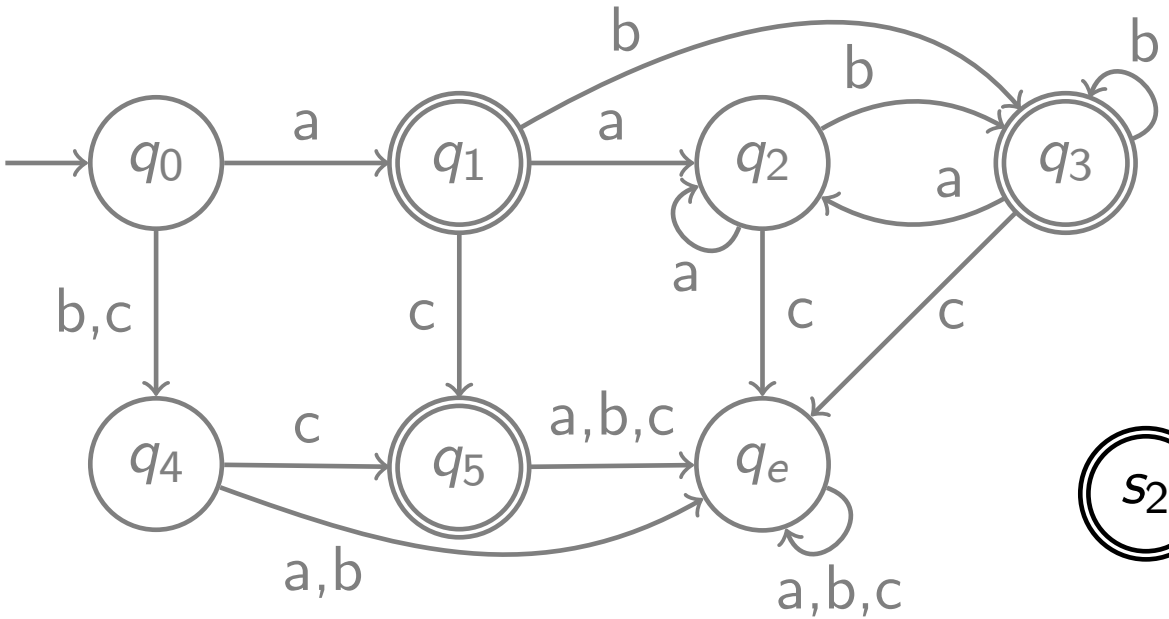
NFAs:

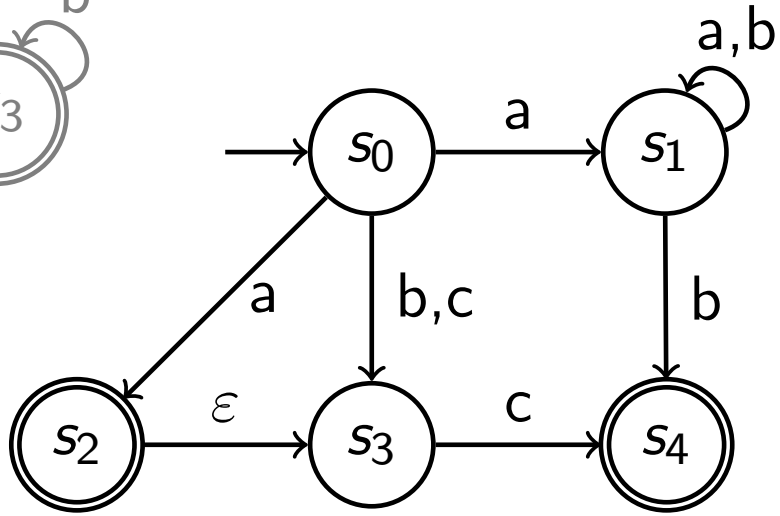
- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

DFAs:

- *at most* *one* outgoing transition per $a \in \Sigma$
- decide acceptance directly

$\mathtt{abc}\langle q_e \rangle$ ✗





NFAs:

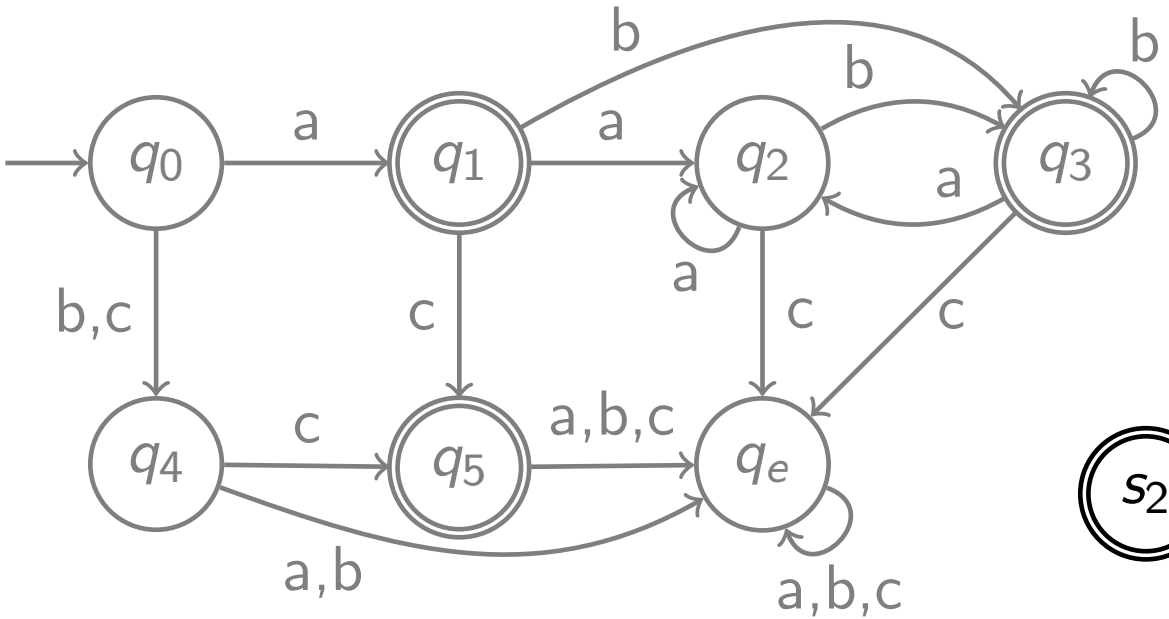
$\langle s_0 \rangle \mathtt{abc}$

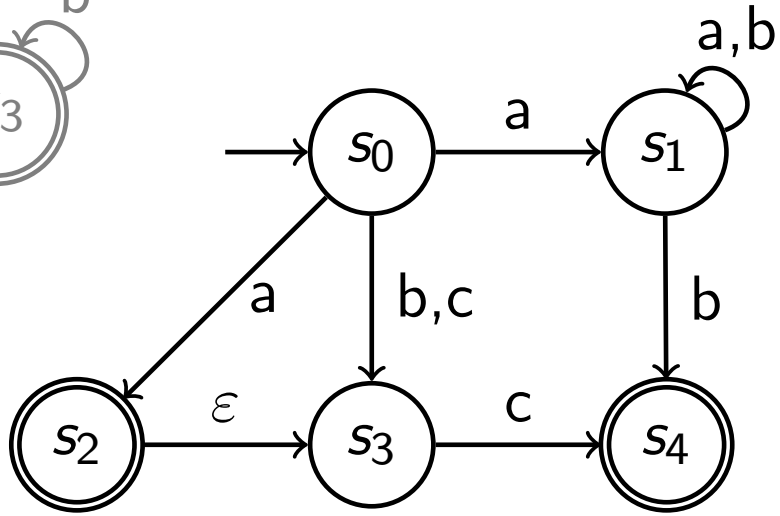
- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

DFAs:

- *at most one* outgoing transition per $a \in \Sigma$
- decide acceptance directly

$\mathtt{abc}\langle q_e \rangle$ ✗





NFAs:

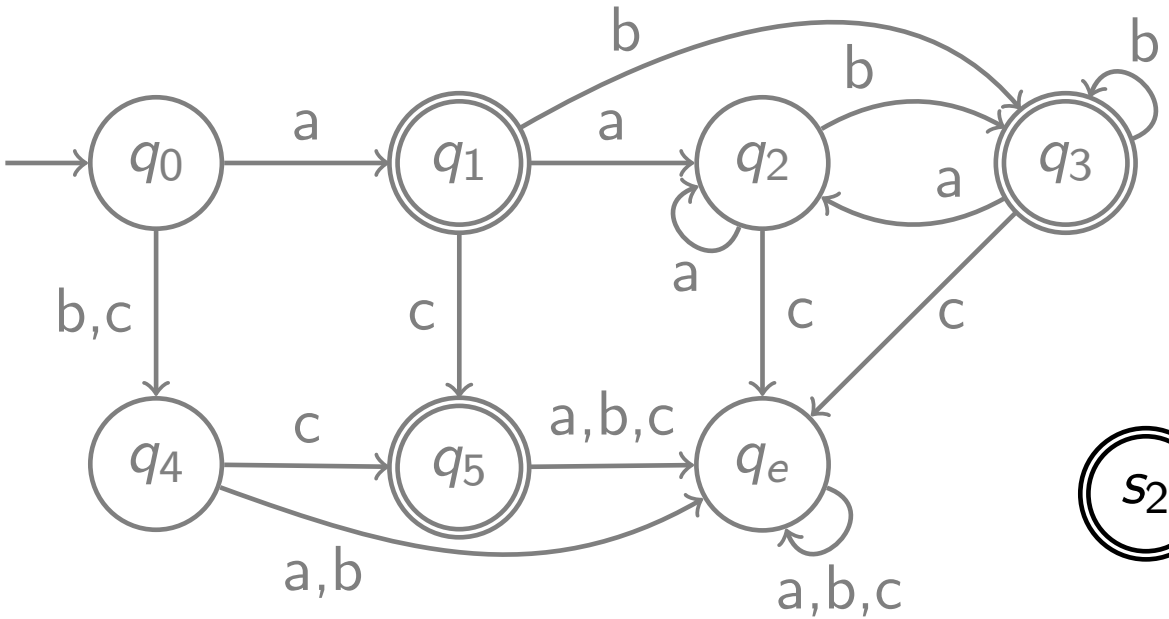
$\mathtt{a}\langle s_1 \rangle \mathtt{bc}$

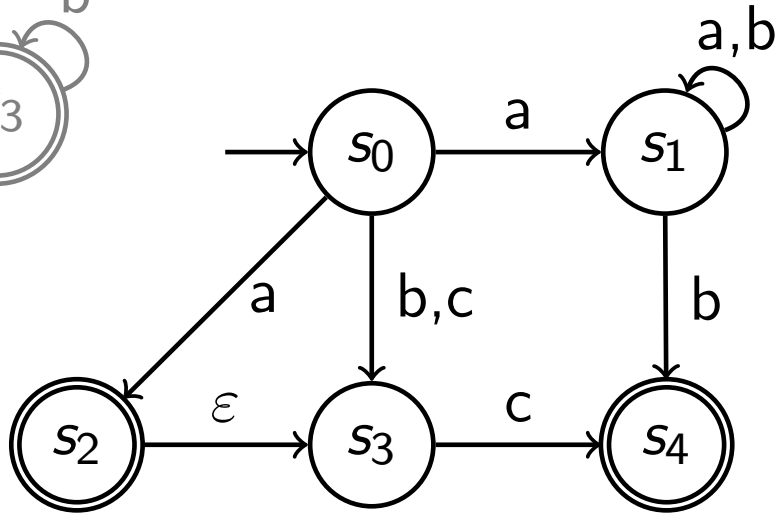
- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

DFAs:

- *at most* *one* outgoing transition per $a \in \Sigma$
- decide acceptance directly

$\mathtt{abc}\langle q_e \rangle$ ✗





NFAs:

$\mathtt{ab}\langle s_1 \rangle \mathtt{c}$

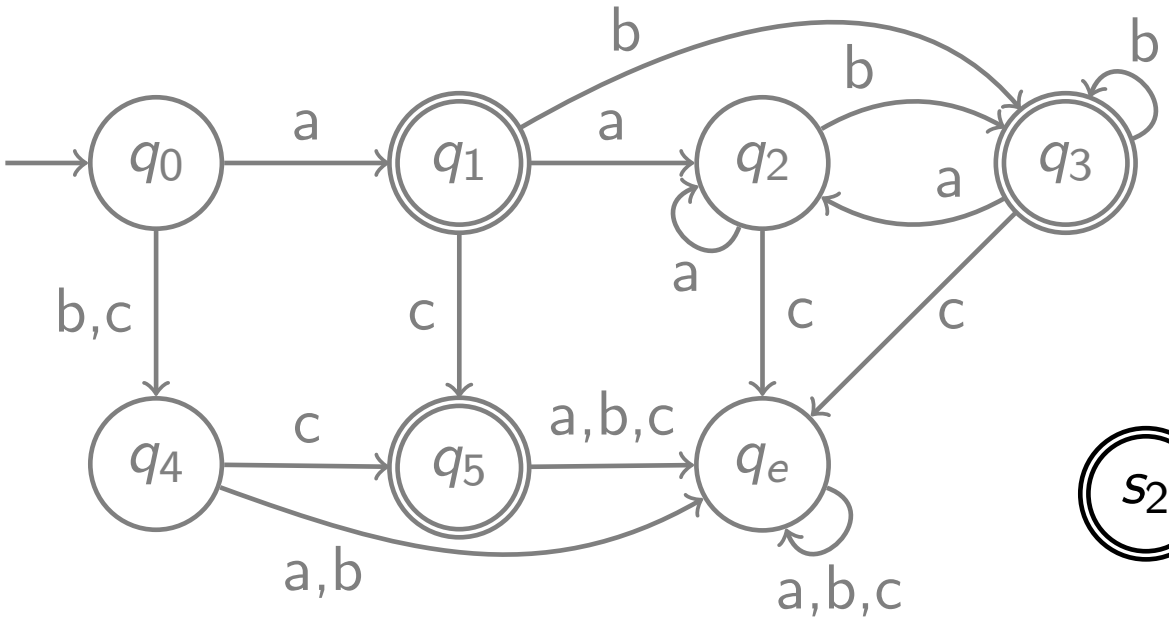
- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

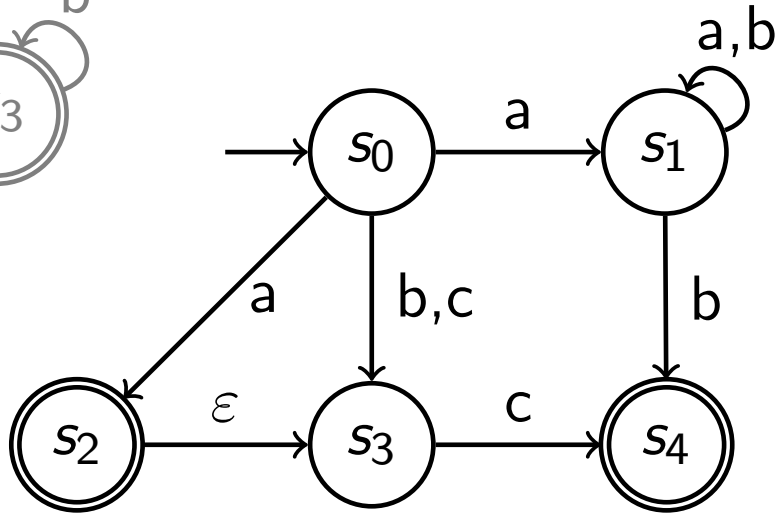
DFAs:

- *at most* *one* outgoing transition per $a \in \Sigma$

$\mathtt{abc}\langle q_e \rangle$ ✗

- decide acceptance directly





NFAs:

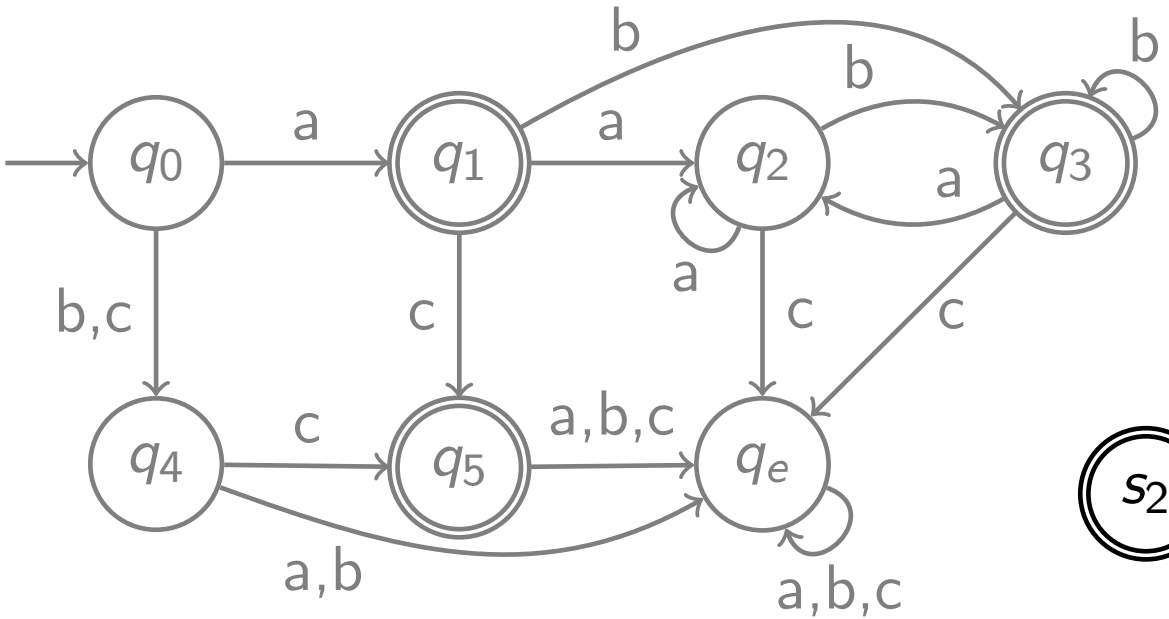
$\mathtt{ab}\langle s_1 \rangle \mathtt{c}$!

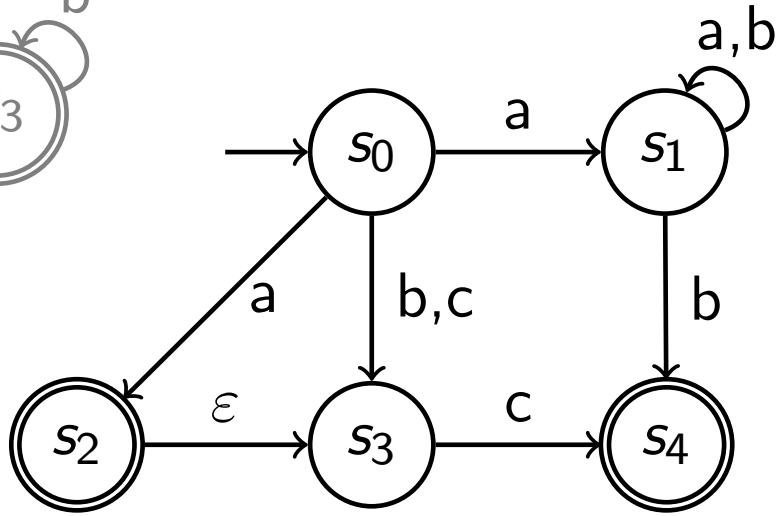
- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

DFAs:

- *at most one* outgoing transition per $a \in \Sigma$
- decide acceptance directly

$\mathtt{abc}\langle q_e \rangle$ ✗





NFAs:

- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

$\mathtt{ab}\langle s_1 \rangle \mathtt{c}$!

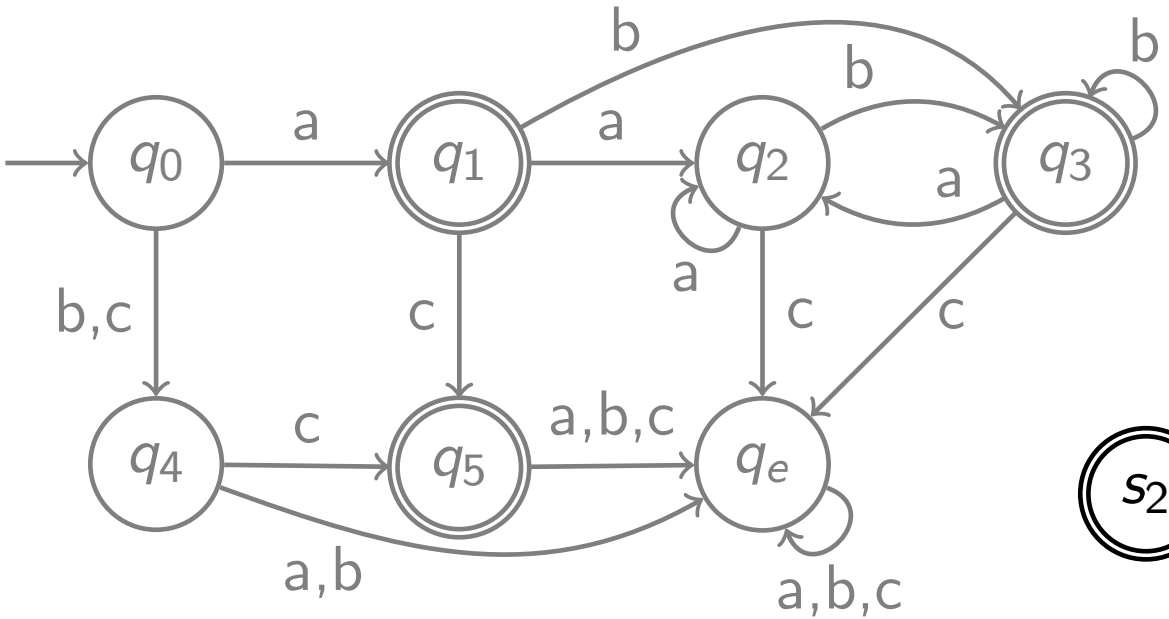
$\langle s_0 \rangle \mathtt{abc}$

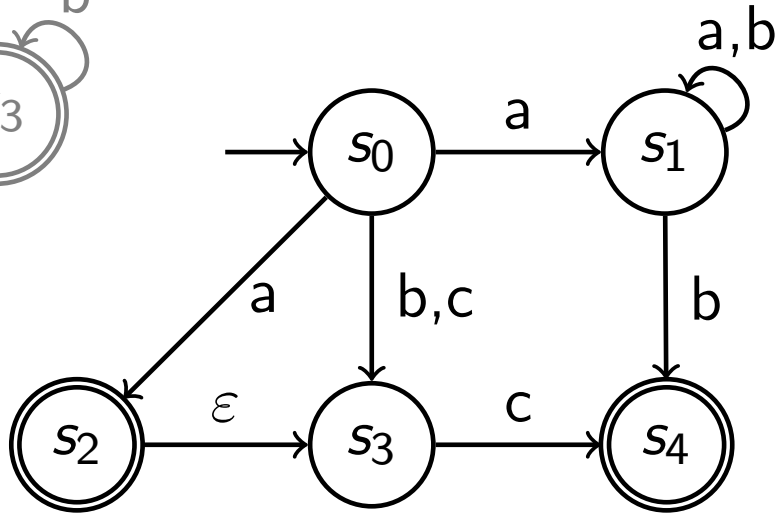
DFAs:

- *at most* *one* outgoing transition per $a \in \Sigma$

$\mathtt{abc}\langle q_e \rangle$ ✗

- decide acceptance directly





NFAs:

- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

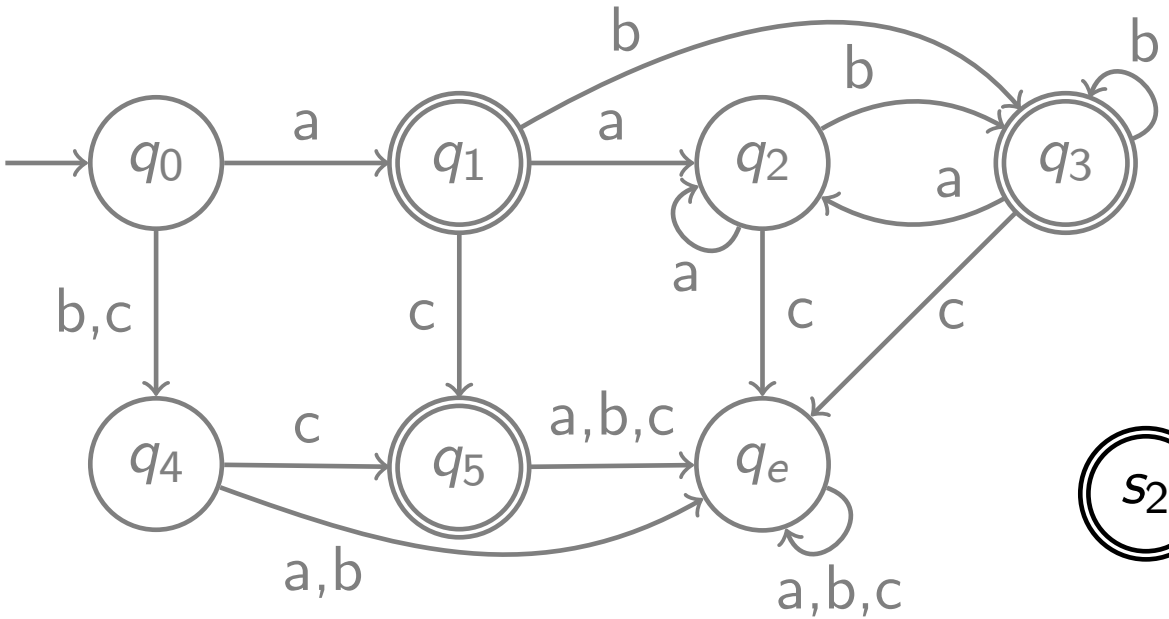
$\mathtt{ab}\langle s_1 \rangle\mathtt{c}$!

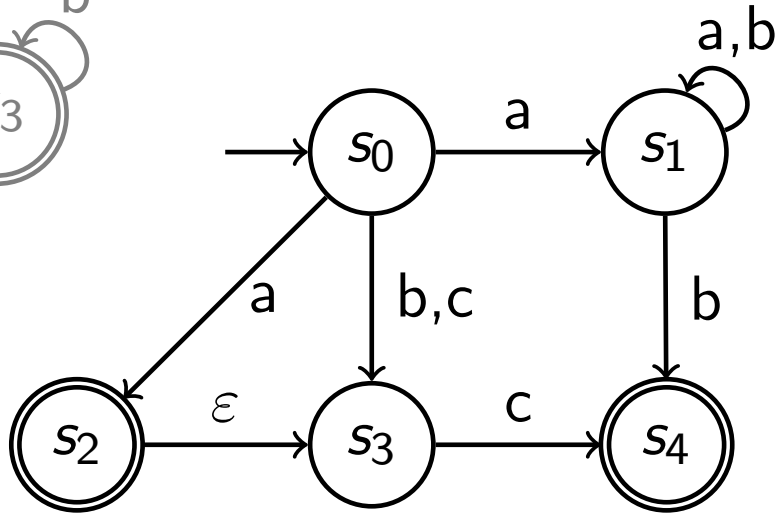
$\mathtt{a}\langle s_1 \rangle\mathtt{bc}$

DFAs:

- *at most* *one* outgoing transition per $a \in \Sigma$
- decide acceptance directly

$\mathtt{abc}\langle q_e \rangle$ ✗





NFAs:

- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

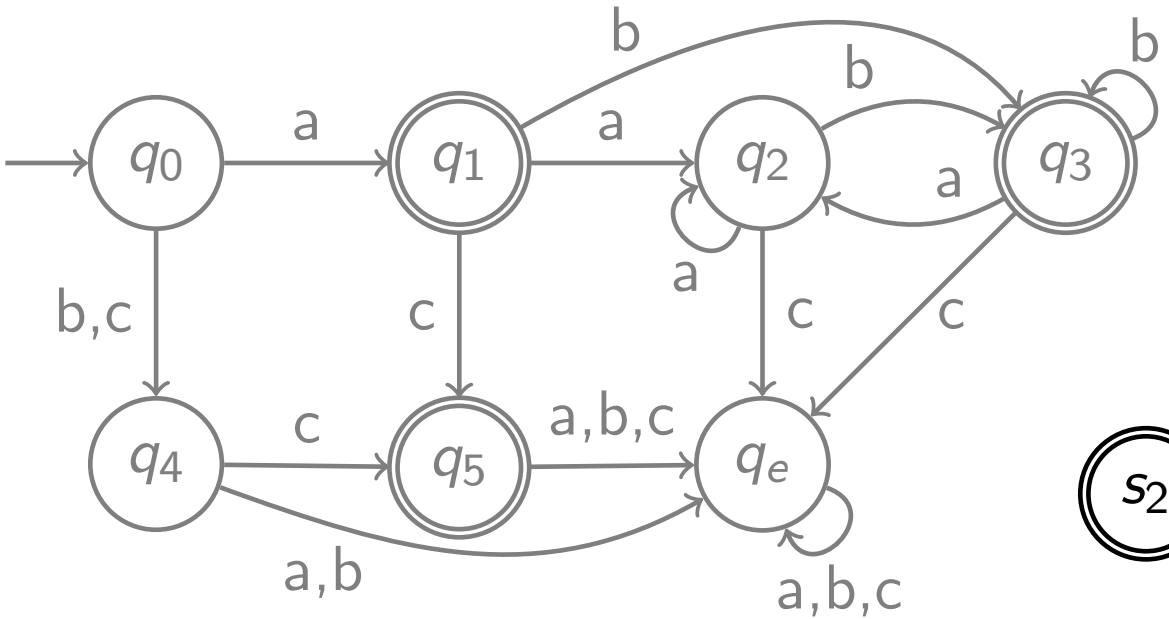
$\mathtt{ab}\langle s_1 \rangle \mathtt{c}$!

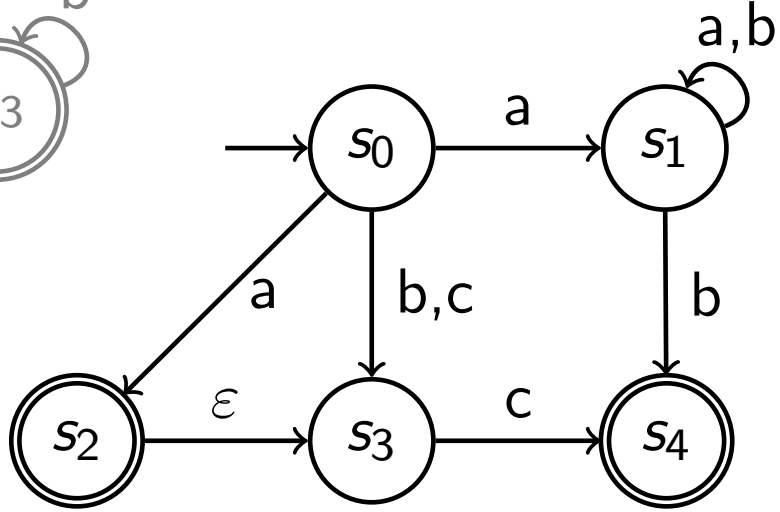
$\mathtt{ab}\langle s_4 \rangle \mathtt{c}$

DFAs:

- *at most one* outgoing transition per $a \in \Sigma$
- decide acceptance directly

$\mathtt{abc}\langle q_e \rangle$ ✗





NFAs:

- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

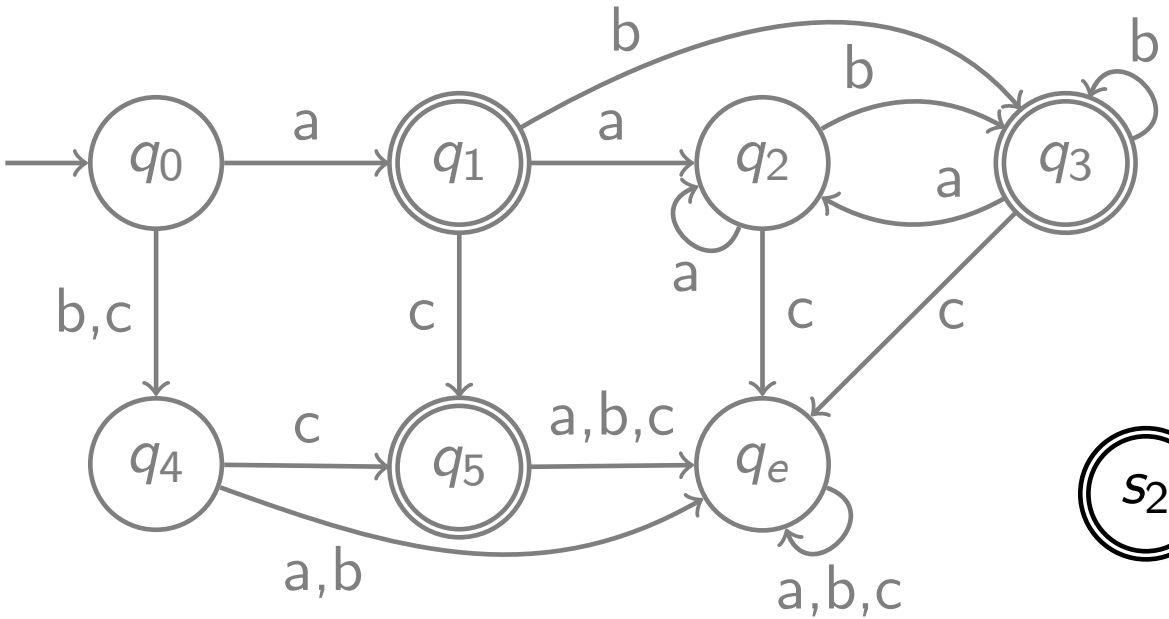
$\mathtt{ab}\langle s_1 \rangle \mathtt{c}$!

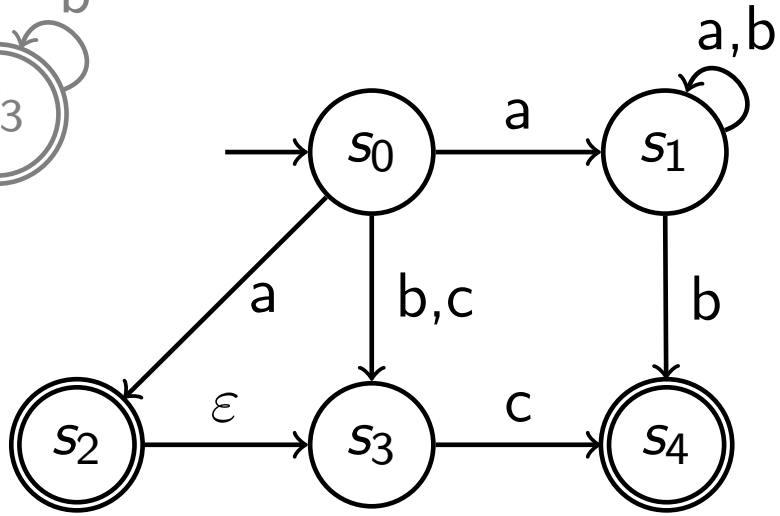
$\mathtt{ab}\langle s_4 \rangle \mathtt{c}$!

DFAs:

- *at most* *one* outgoing transition per $a \in \Sigma$
- decide acceptance directly

$\mathtt{abc}\langle q_e \rangle$ ✗





NFAs:

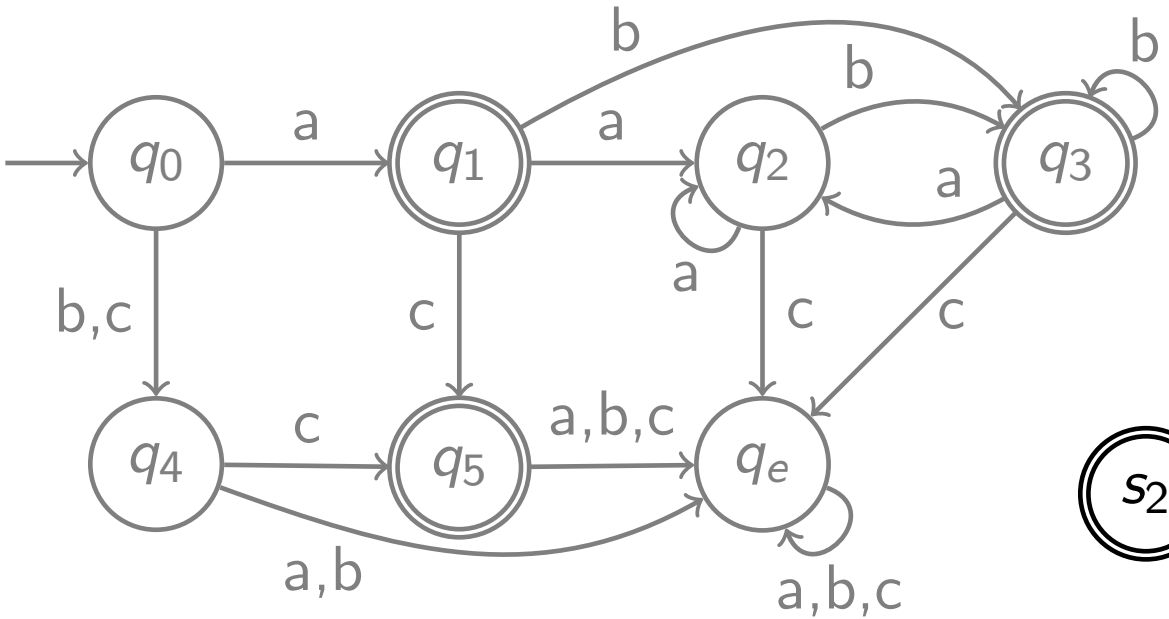
- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

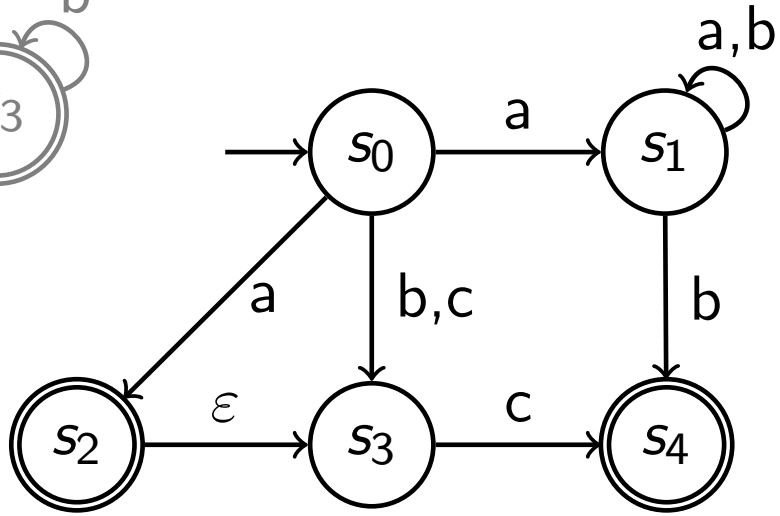
$\mathtt{ab}\langle s_1 \rangle \mathtt{c}$!
$\mathtt{ab}\langle s_4 \rangle \mathtt{c}$!
$\langle s_0 \rangle \mathtt{abc}$

DFAs:

- *at most* *one* outgoing transition per $a \in \Sigma$
- decide acceptance directly

$\mathtt{abc}\langle q_e \rangle$ ✗





NFAs:

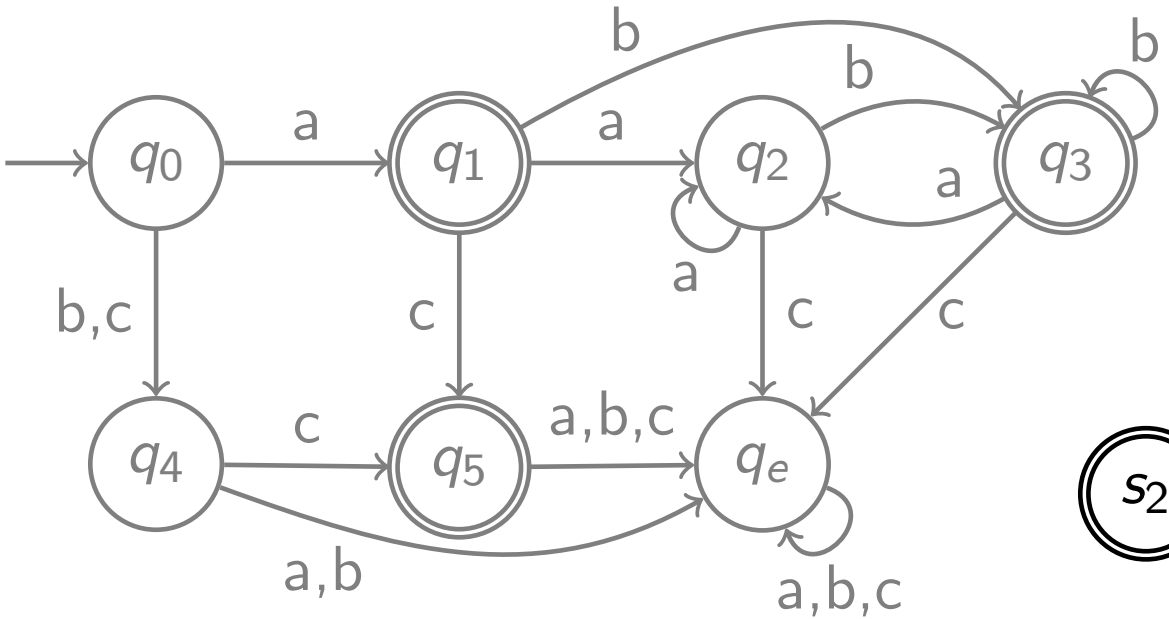
- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

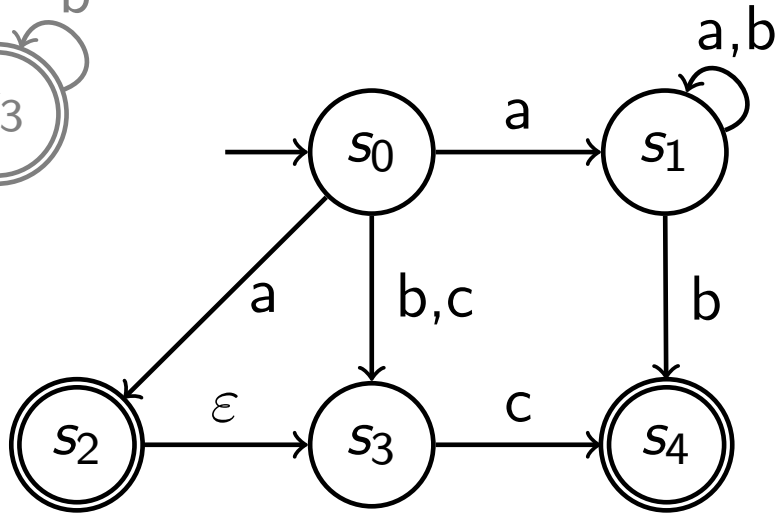
$\mathtt{ab}\langle s_1 \rangle \mathtt{c}$!
$\mathtt{ab}\langle s_4 \rangle \mathtt{c}$!
$\mathtt{a}\langle s_2 \rangle \mathtt{bc}$

DFAs:

- *at most* *one* outgoing transition per $a \in \Sigma$
- decide acceptance directly

$\mathtt{abc}\langle q_e \rangle$ ✗





NFAs:

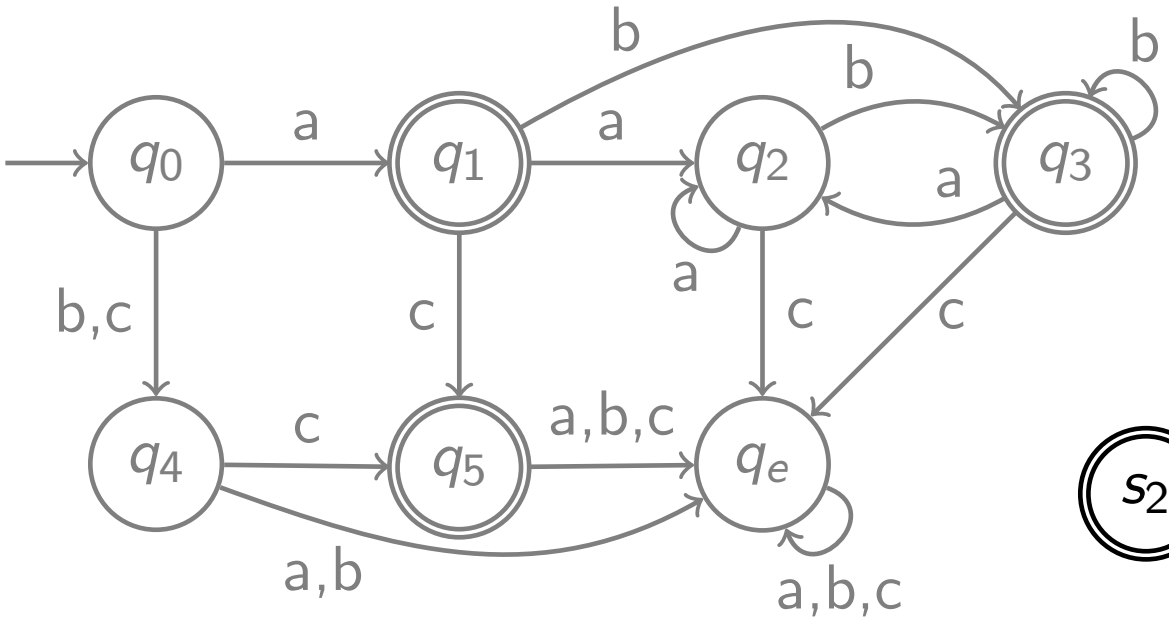
- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

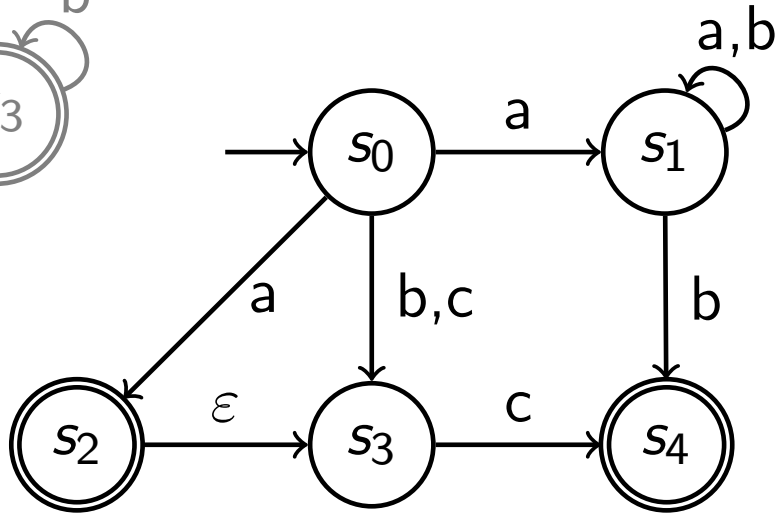
$\mathtt{ab}\langle s_1 \rangle \mathtt{c}$!
$\mathtt{ab}\langle s_4 \rangle \mathtt{c}$!
$\mathtt{a}\langle s_2 \rangle \mathtt{bc}$!

DFAs:

- *at most* *one* outgoing transition per $a \in \Sigma$
- decide acceptance directly

$\mathtt{abc}\langle q_e \rangle$ ✗





NFAs:

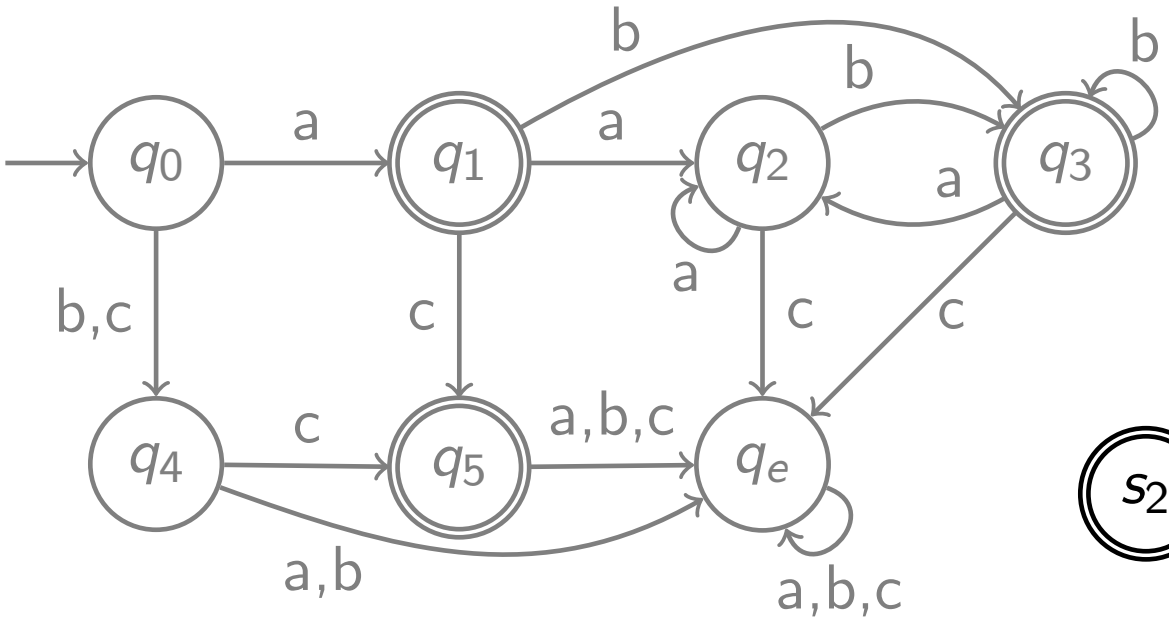
- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

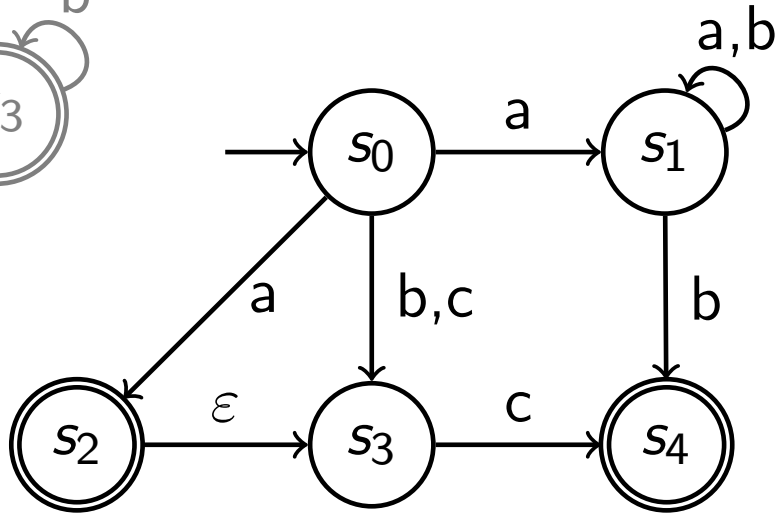
$\mathtt{ab}\langle s_1 \rangle \mathtt{c}$!
$\mathtt{ab}\langle s_4 \rangle \mathtt{c}$!
$\mathtt{a}\langle s_2 \rangle \mathtt{bc}$!
$\langle s_0 \rangle \mathtt{abc}$

DFAs:

- *at most* *one* outgoing transition per $a \in \Sigma$
- decide acceptance directly

$\mathsf{abc}\langle q_e \rangle$ ✗





NFAs:

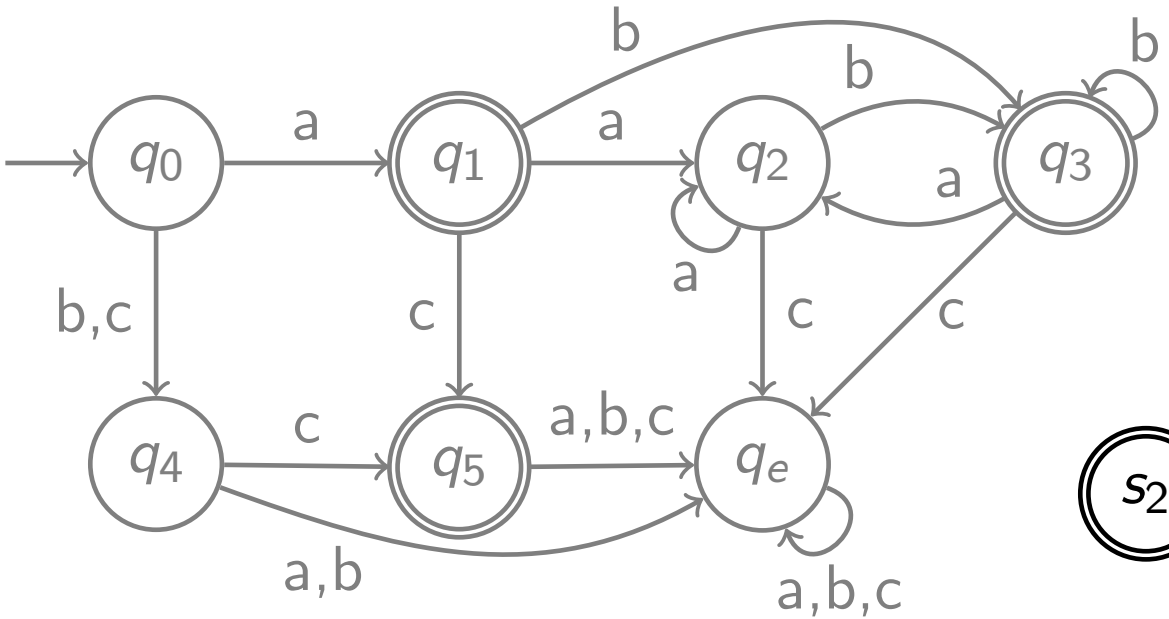
- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

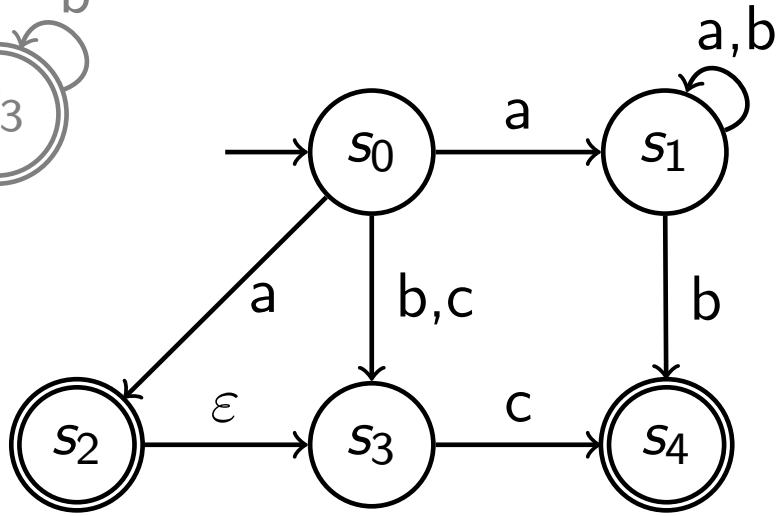
$\mathsf{ab}\langle s_1 \rangle \mathsf{c}$!
$\mathsf{ab}\langle s_4 \rangle \mathsf{c}$!
$\mathsf{a}\langle s_2 \rangle \mathsf{bc}$!
$\mathsf{a}\langle s_2 \rangle \mathsf{bc}$

DFAs:

- *at most* *one* outgoing transition per $a \in \Sigma$
- decide acceptance directly

$\mathtt{abc}\langle q_e \rangle$ ✗





NFAs:

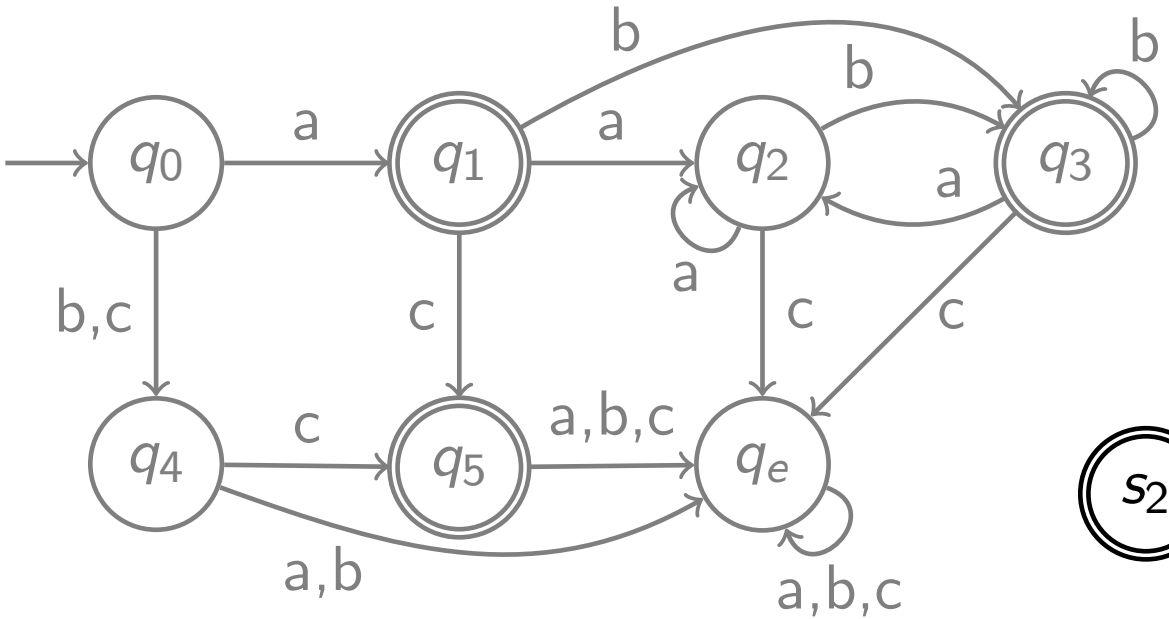
- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

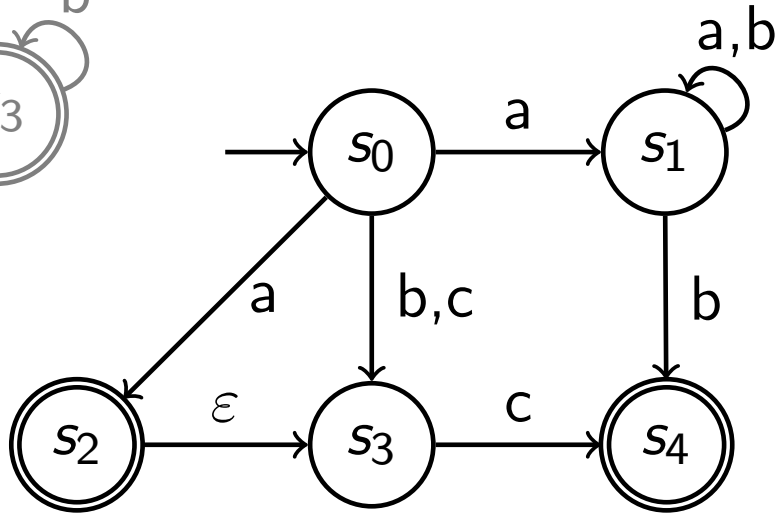
$\mathtt{ab}\langle s_1 \rangle \mathtt{c}$!
$\mathtt{ab}\langle s_4 \rangle \mathtt{c}$!
$\mathtt{a}\langle s_2 \rangle \mathtt{bc}$!
$\mathtt{a}\langle s_3 \rangle \mathtt{bc}$

DFAs:

- *at most* *one* outgoing transition per $a \in \Sigma$
- decide acceptance directly

$\mathsf{abc}\langle q_e \rangle$ ✗





NFAs:

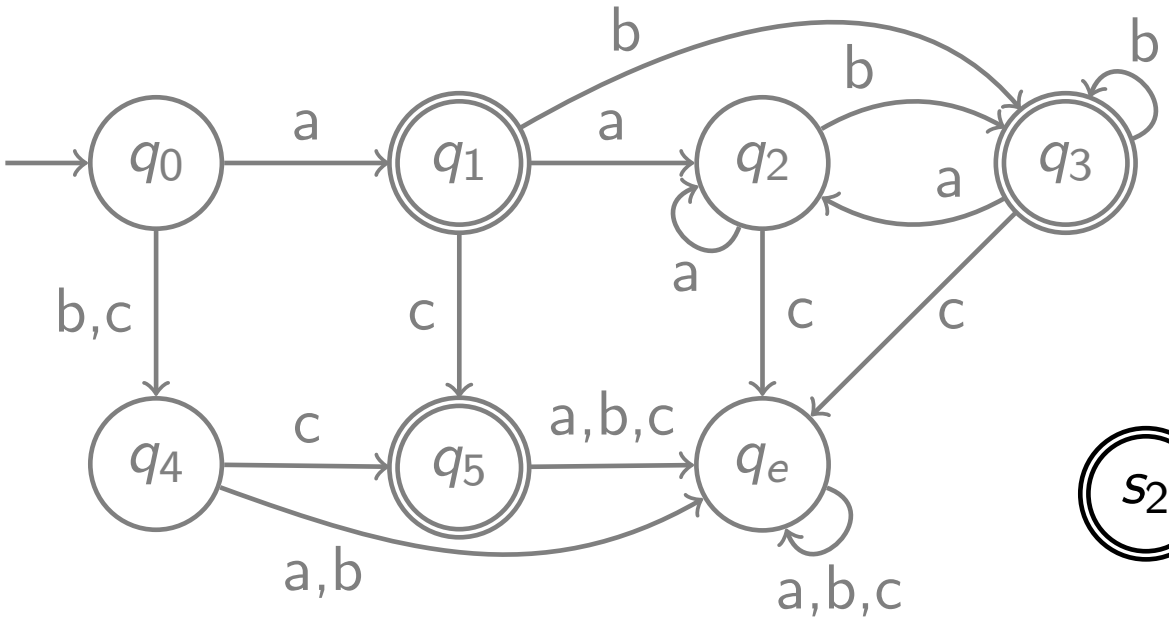
- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

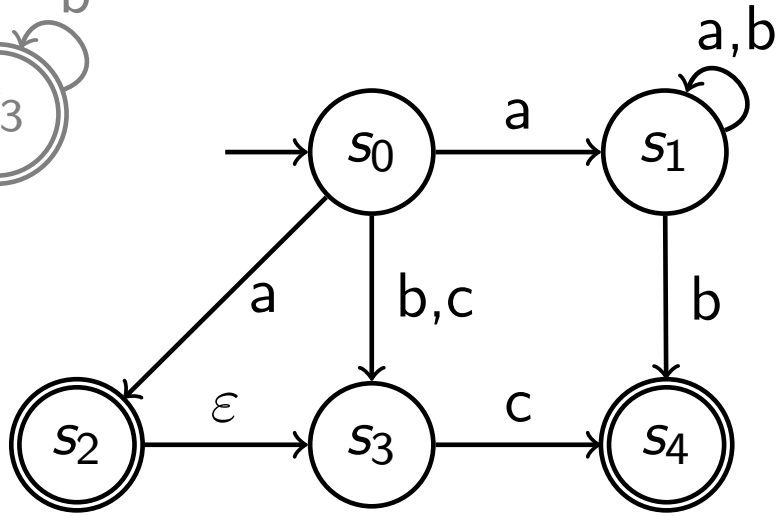
$\mathsf{ab}\langle s_1 \rangle \mathsf{c}$!
$\mathsf{ab}\langle s_4 \rangle \mathsf{c}$!
$\mathsf{a}\langle s_2 \rangle \mathsf{bc}$!
$\mathsf{a}\langle s_3 \rangle \mathsf{bc}$!

DFAs:

- *at most* *one* outgoing transition per $a \in \Sigma$
- decide acceptance directly

$\mathsf{abc}\langle q_e \rangle$ ✗





NFAs:

- *zero or more* transitions per $a \in \Sigma \cup \{\varepsilon\}$
- acceptance: exists an accepting run
- rejection: all possible runs reject

$\mathsf{ab}\langle s_1 \rangle \mathsf{c}$!
$\mathsf{ab}\langle s_4 \rangle \mathsf{c}$!
$\mathsf{a}\langle s_2 \rangle \mathsf{bc}$!
$\mathsf{a}\langle s_3 \rangle \mathsf{bc}$!

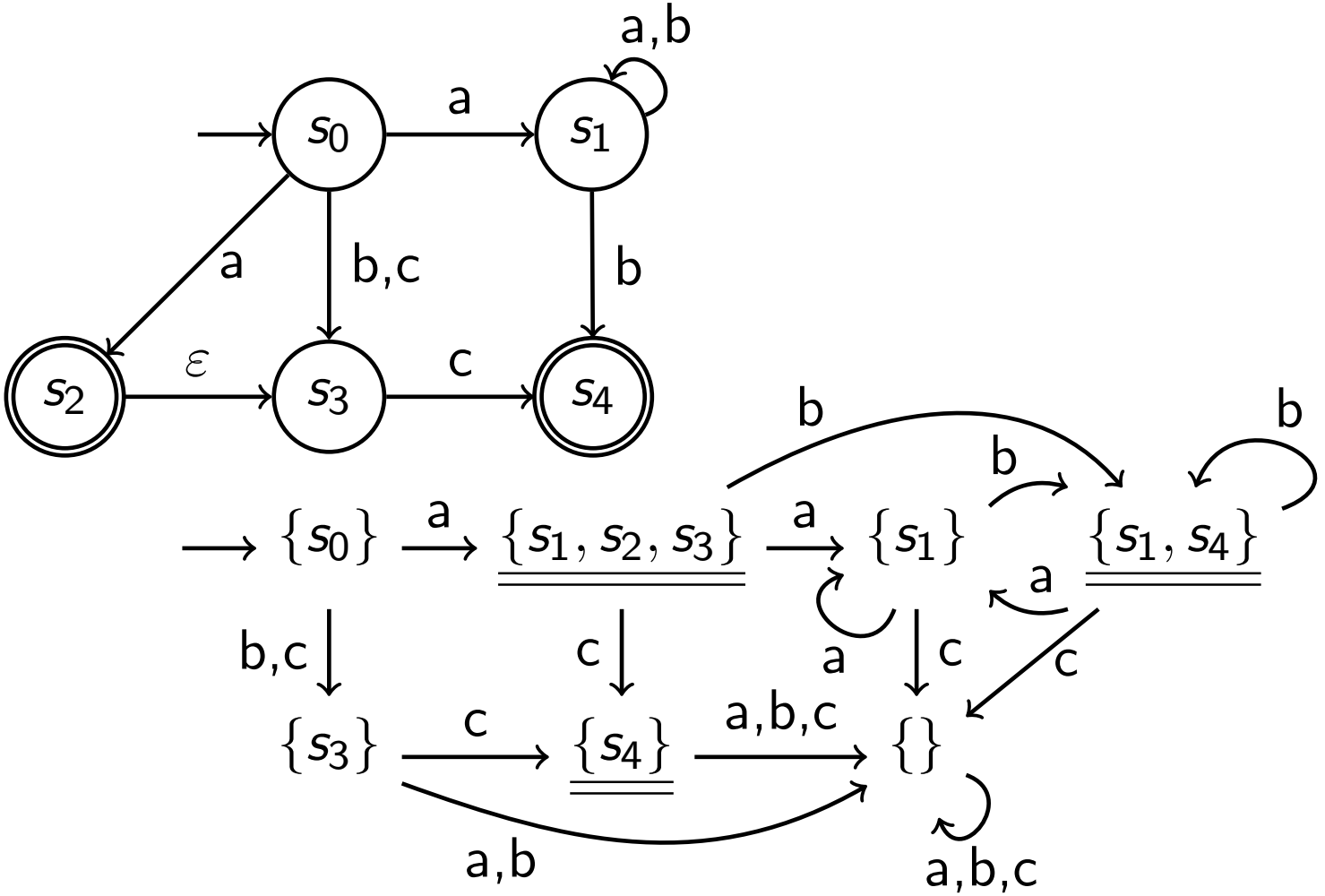
} ✗

Take-away: DFAs are easy to *use*; NFAs are easy to *make*

- **Q:** Is there a language that you can express with an RE but not with an NFA? Nope! **Thompson's construction** (2.4.2)
- **Q:** Is there a language that you can express with a DFA but not with an NFA? Nope! Every DFA is secretly an NFA.
- **Q:** Is there a language that you can express with an NFA but not with a DFA? ... Enter **Subset Construction**!

Subset Construction, NFA $\rightarrow$ DFA (2.4.3)

Idea: Build a DFA to simulate all NFA runs simultaneously!



Subset Construction, NFA $\rightarrow$ DFA (2.4.3)

Pseudocode:

- $Q = \underline{2^S}$ (though many may be unneeded)
- $\delta_D(q_1, a) = \varepsilon\text{-close}(\{s_j \mid s_i \in q_1, s_j \in \delta_N(s_i, a)\})$
- $q_0 = \varepsilon\text{-close}(\{s_0\})$
- q is accepting if any $s \in q$ is accepting

Take-away: NFAs are not more powerful than DFAs, but they can be *way* more concise!

Speaking of large DFAs... there is some hope!
Fact: Every DFA can be reduced (greedily!) to a unique (!) DFA having the *minimum possible number of states*.

First idea:

- what happens if you "fuse" two states in a DFA?
- under what condition does fusing two states have no effect?
- therefore, look for states that accept the same suffix language and fuse them
- problem: that's a *lot* of work!

Hopcroft's Algorithm, DFA $\rightarrow$ minimal DFA (2.4.4)

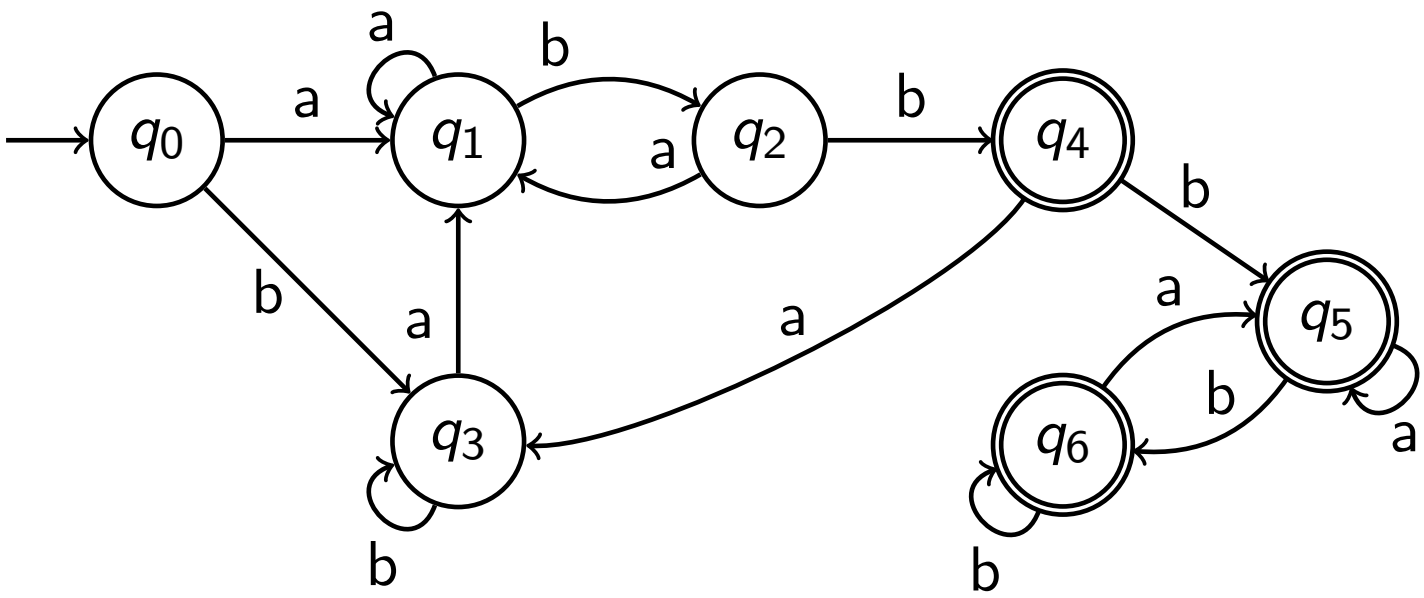
Gradually partition Q so that states in different partitions do not behave the same, but states in the same partition do

Pseudocode:

- initially assume all states are the same (one big partition)
- look for evidence to the contrary:
  - always start with ε, i.e., accepting states and non-accepting states are definitely not the same; now there are 2 partitions
  - then iterate through $a \in \Sigma$: if the transition on 'a' sends states in P to states in multiple different partitions, conclude that you must split P along this boundary
  - **Q:** is one pass through Σ enough? Generally no!

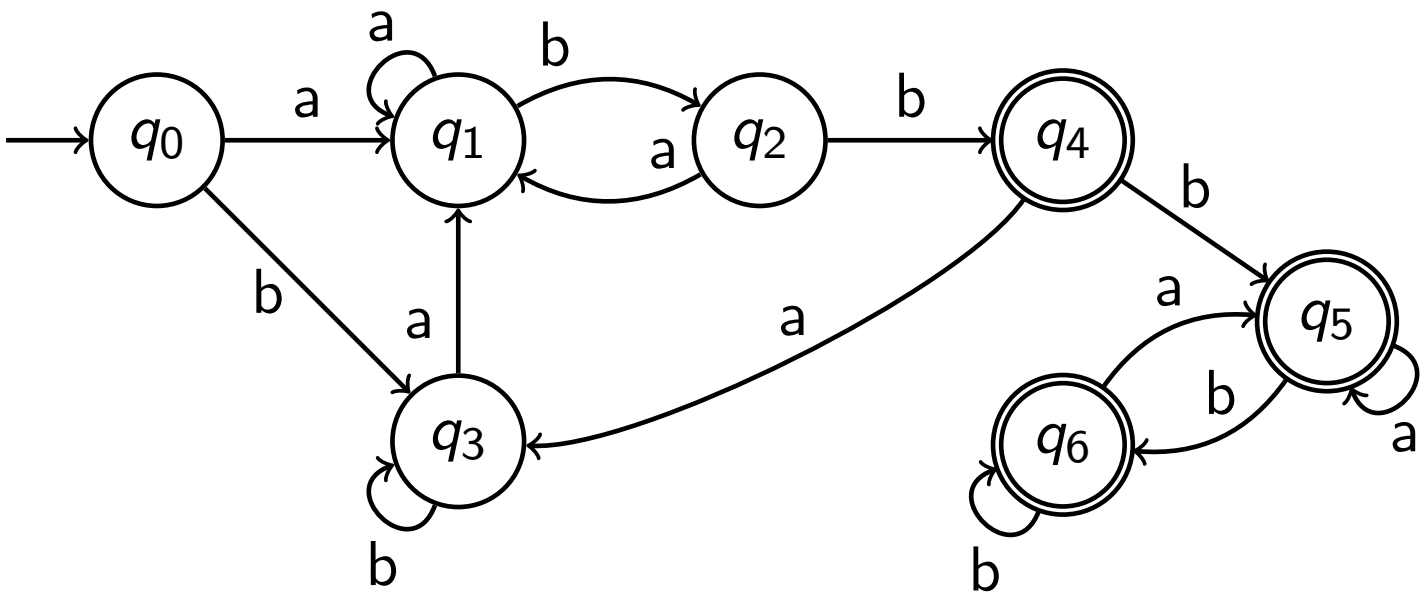
Hopcroft's Algorithm, DFA $\rightarrow$ minimal DFA (2.4.4)

P_0



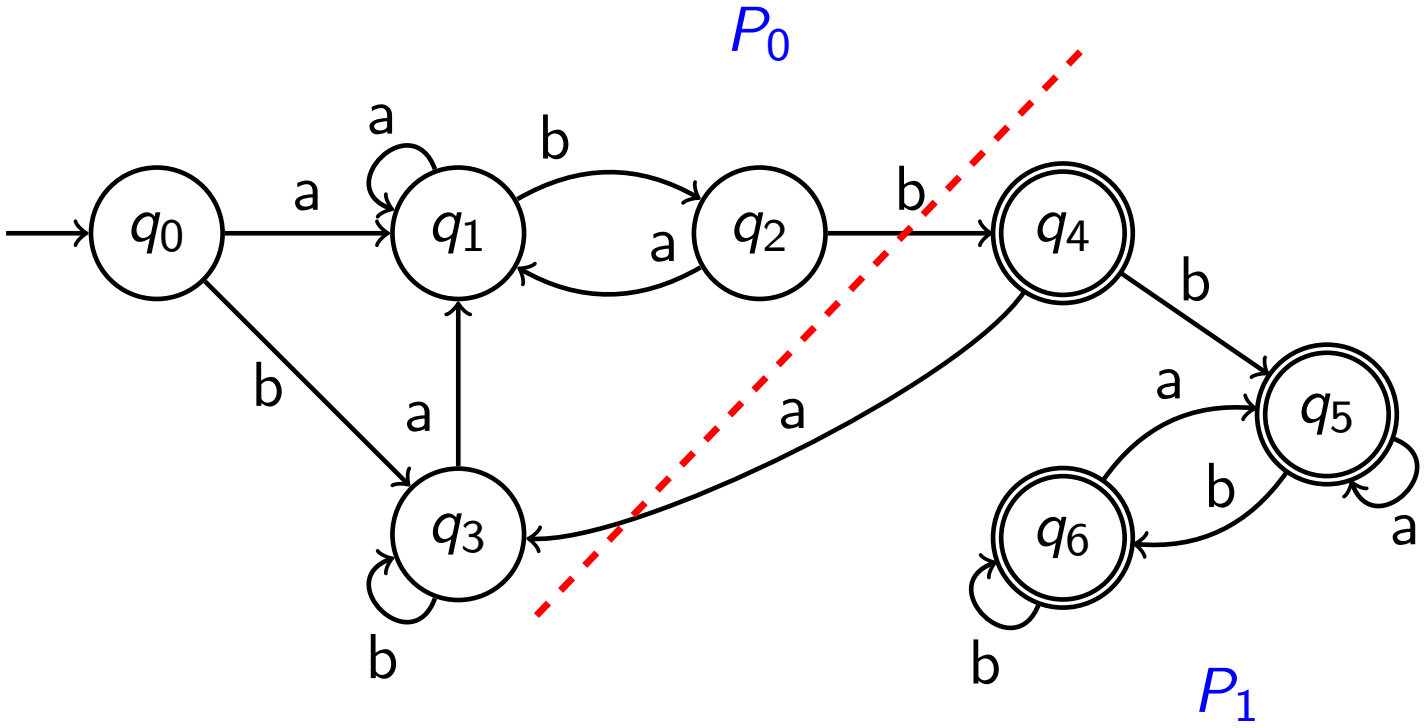
Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)

P_0



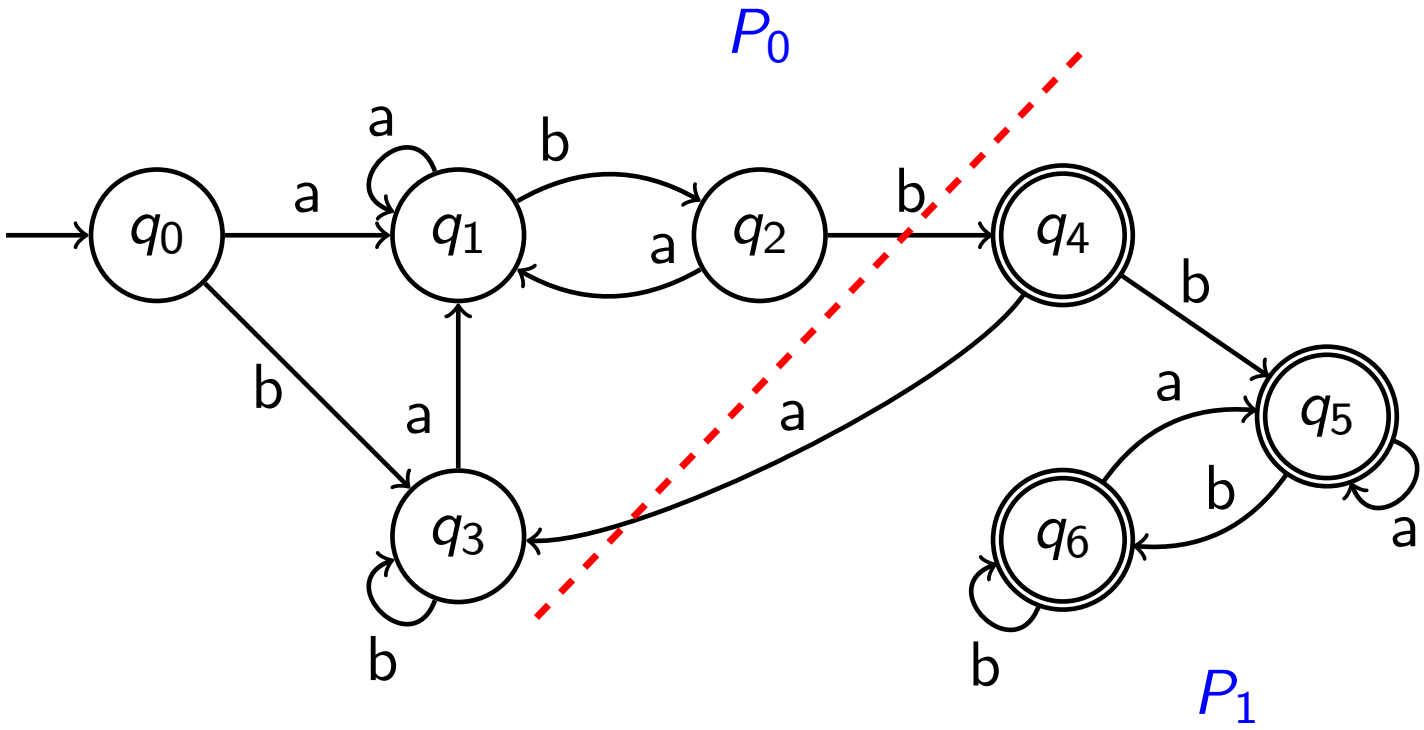
Are the partitions consistent w.r.t. ε?

Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



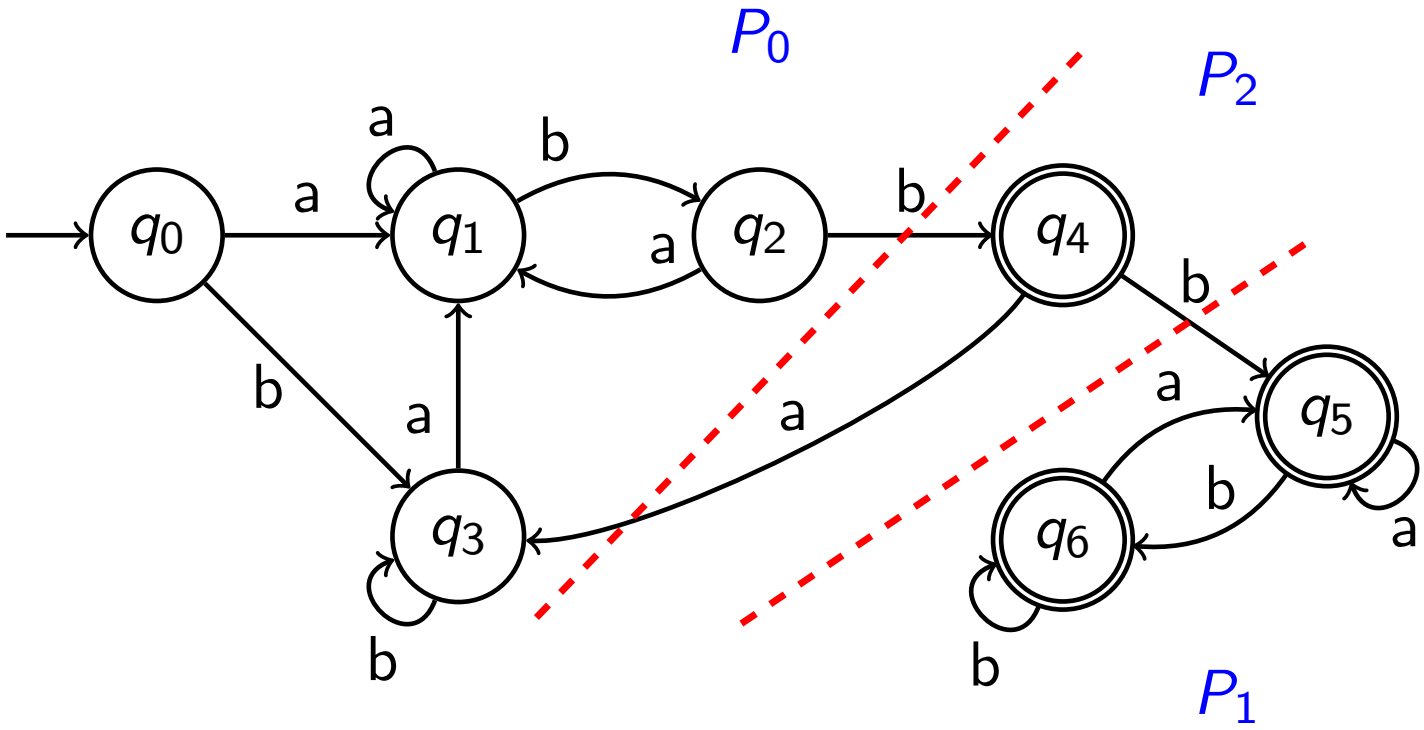
Are the partitions consistent w.r.t. ε? Split!

Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



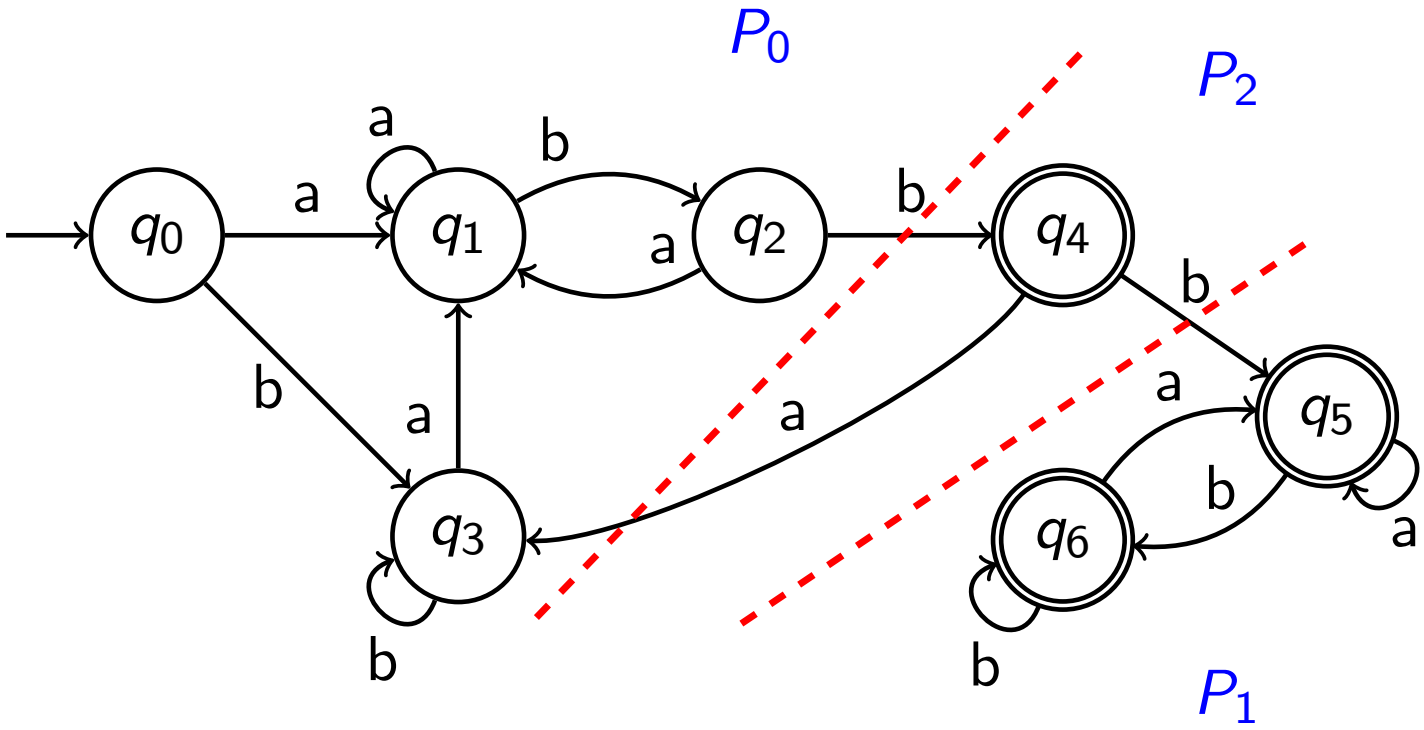
Are the partitions consistent w.r.t. a?

Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



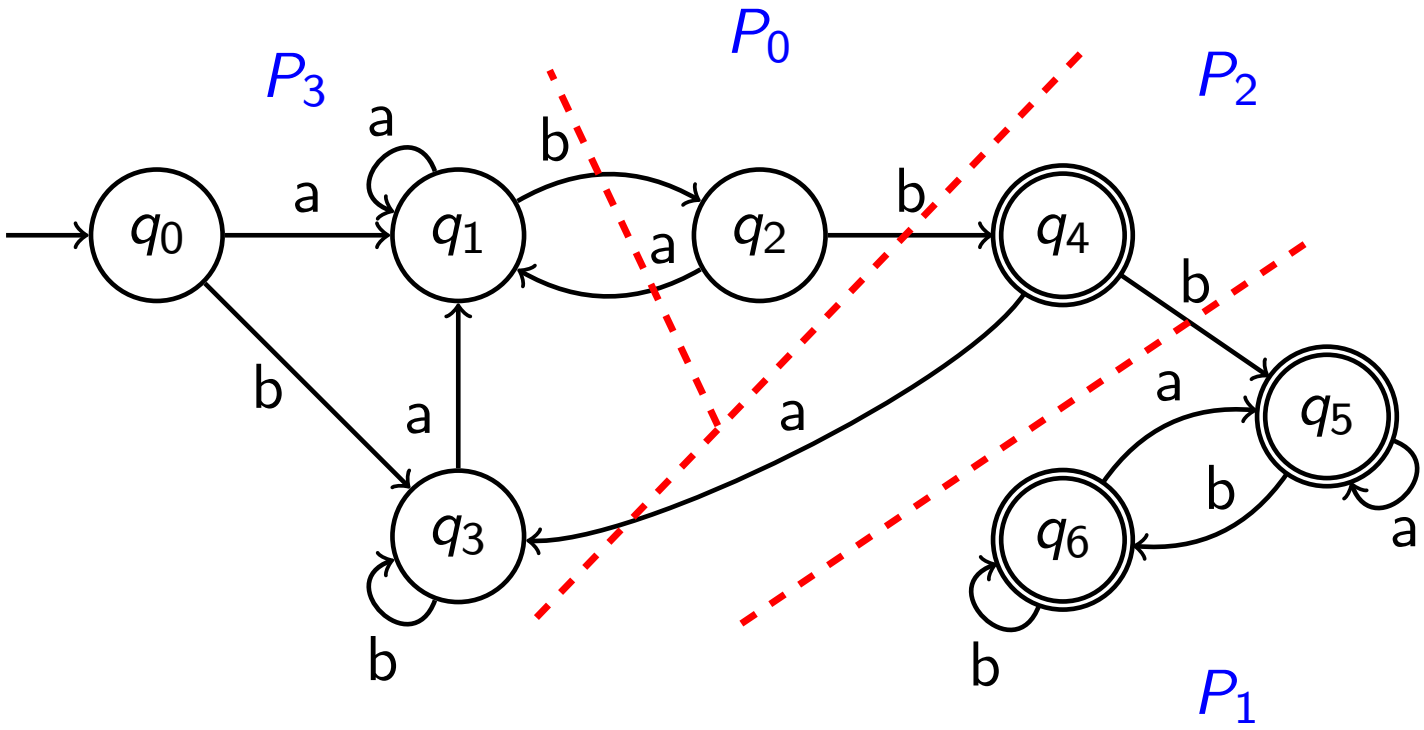
Are the partitions consistent w.r.t. a? Split!

Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



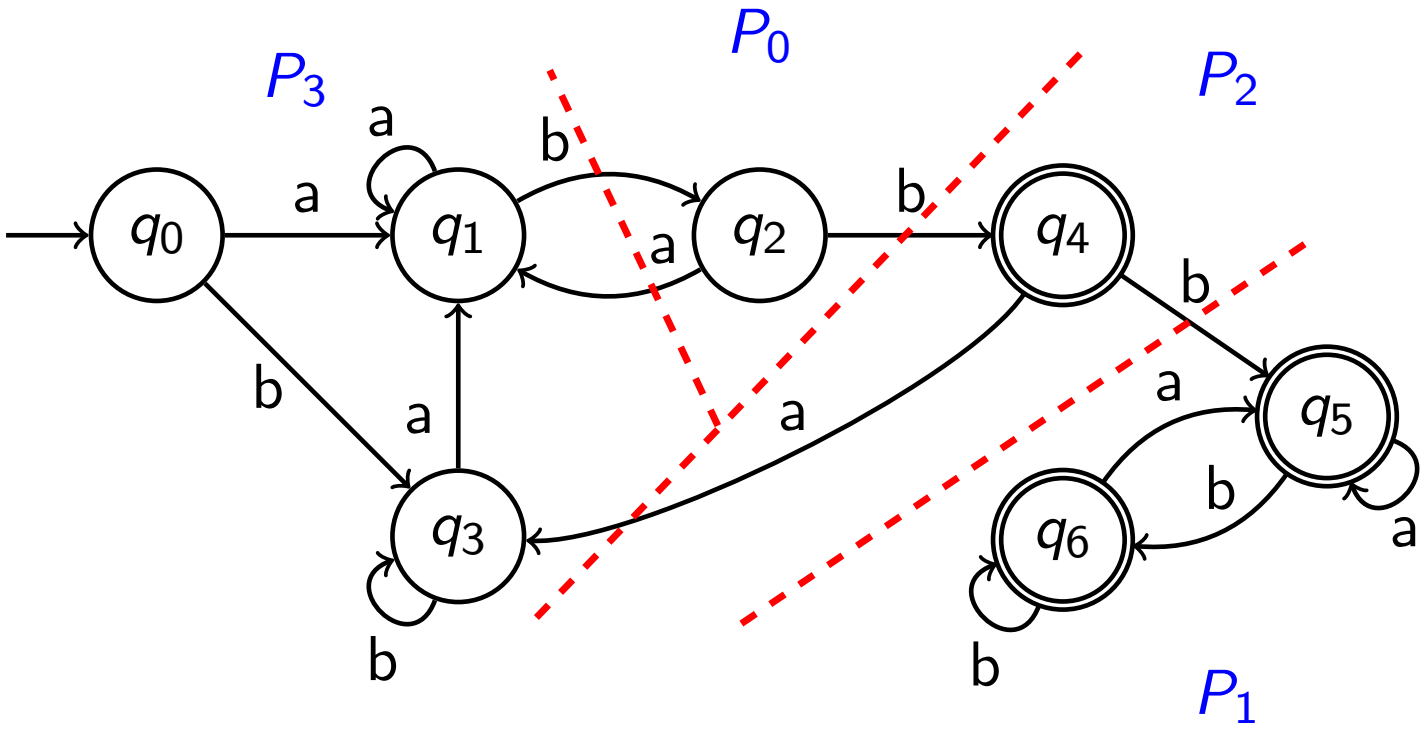
Are the partitions consistent w.r.t. b?

Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



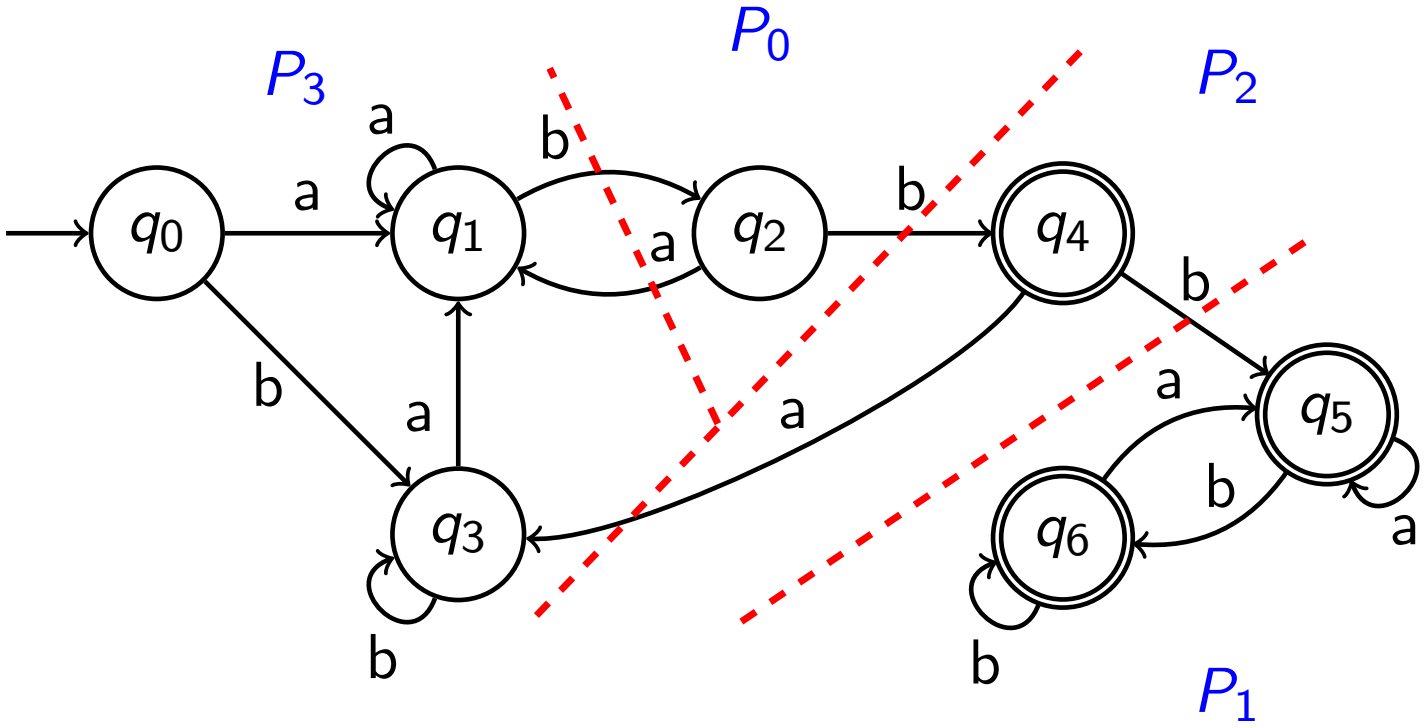
Are the partitions consistent w.r.t. b? Split!

Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



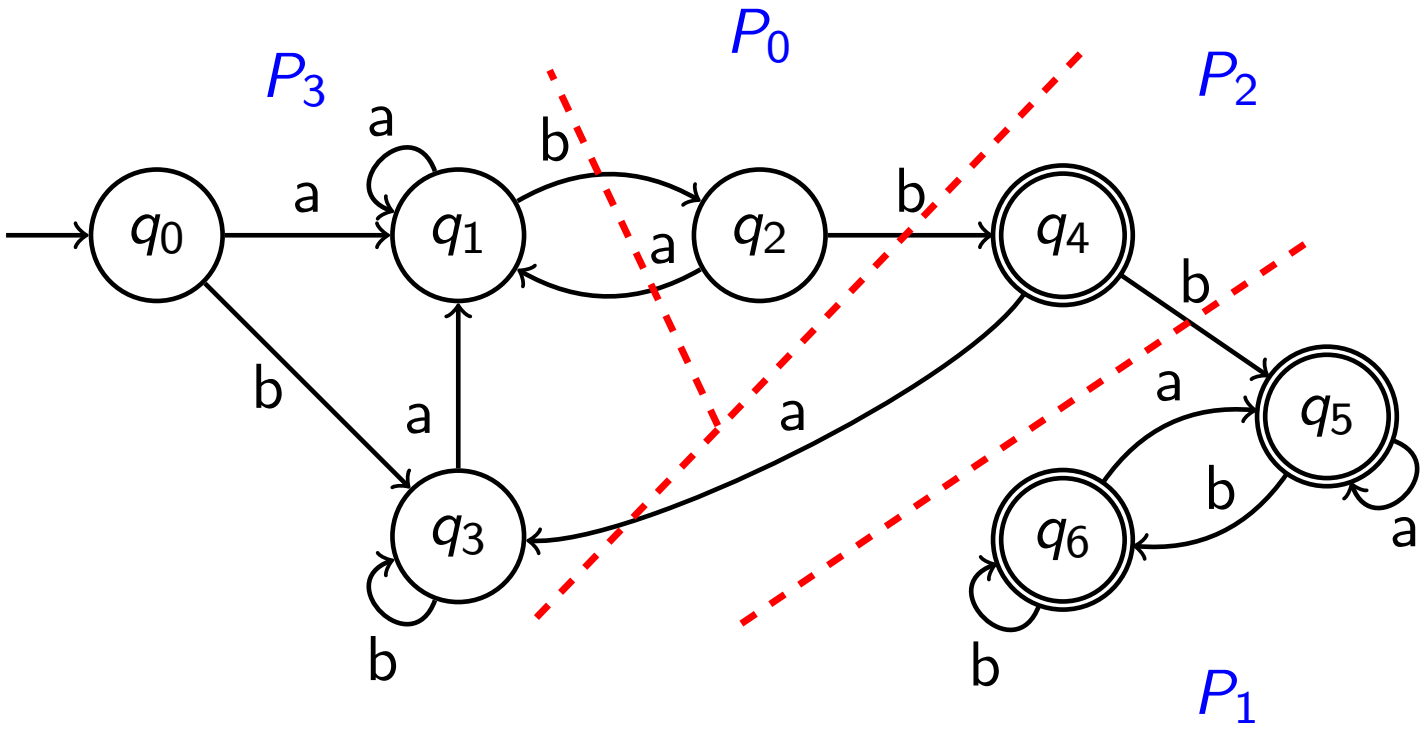
Are the partitions consistent w.r.t. a?

Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



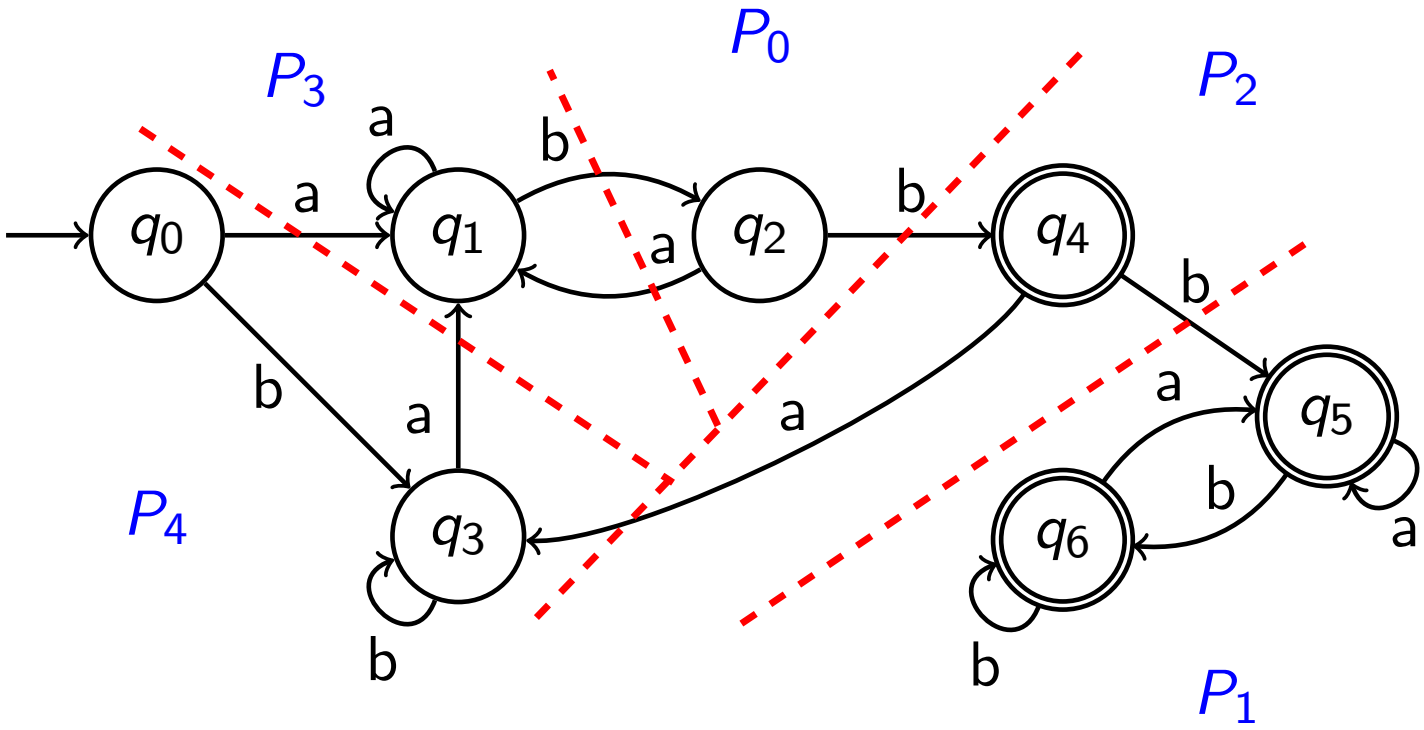
Are the partitions consistent w.r.t. a? Actually, yes.

Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



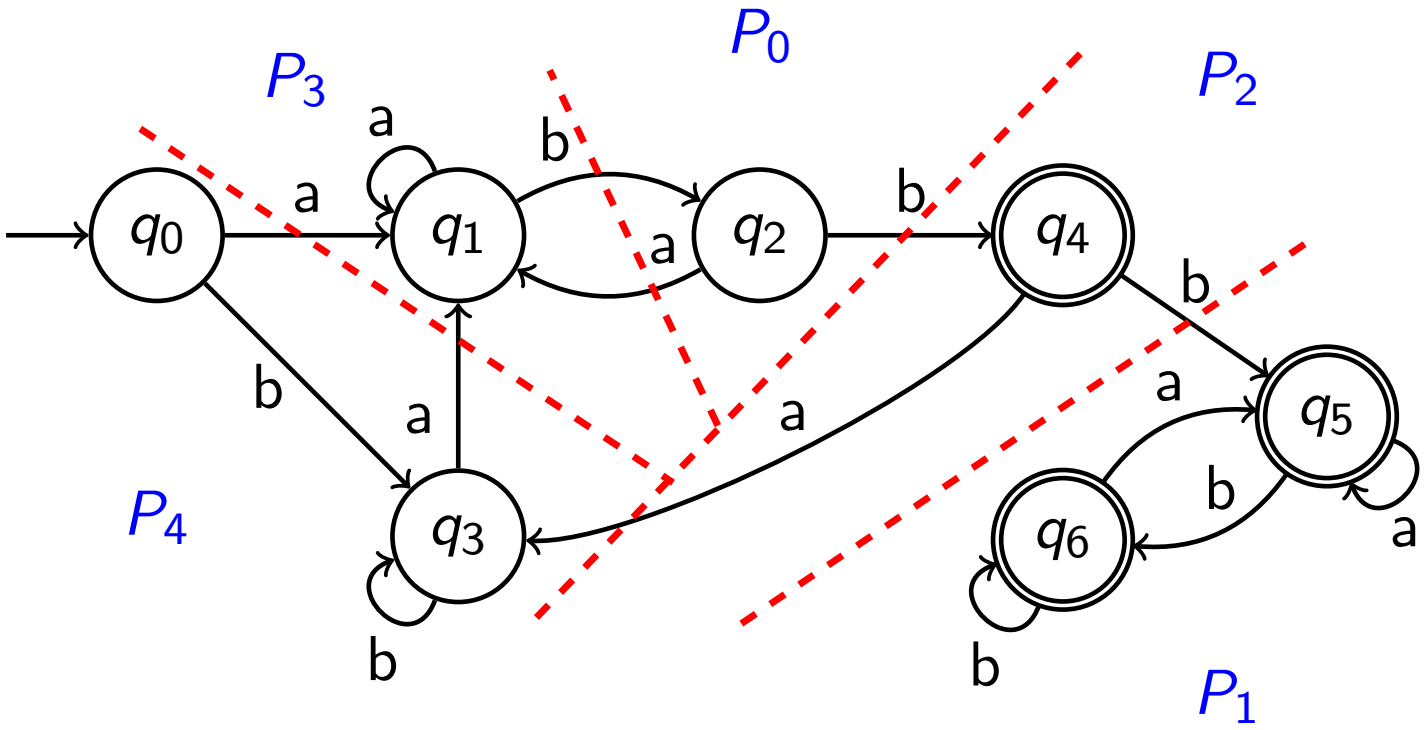
Are the partitions consistent w.r.t. b?

Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



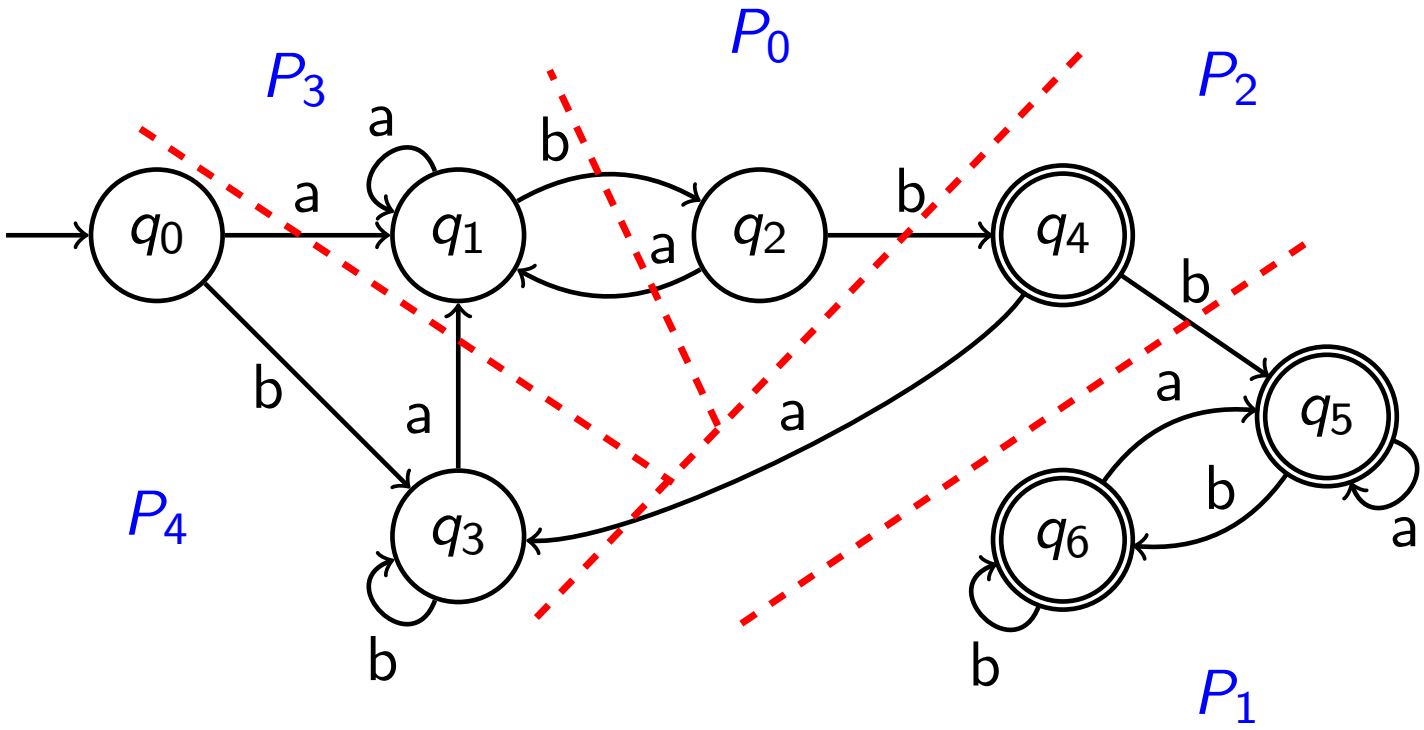
Are the partitions consistent w.r.t. b? Split!

Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



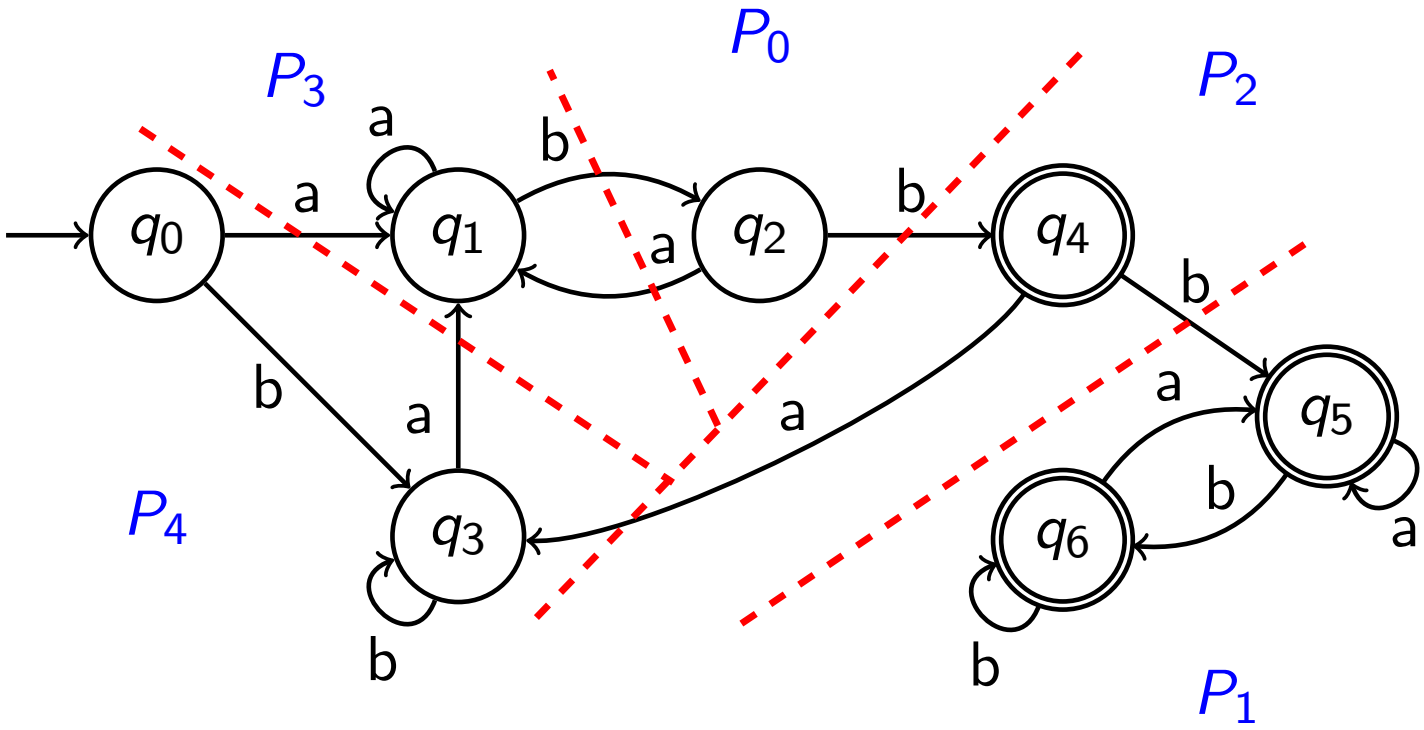
Are the partitions consistent w.r.t. a?

Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



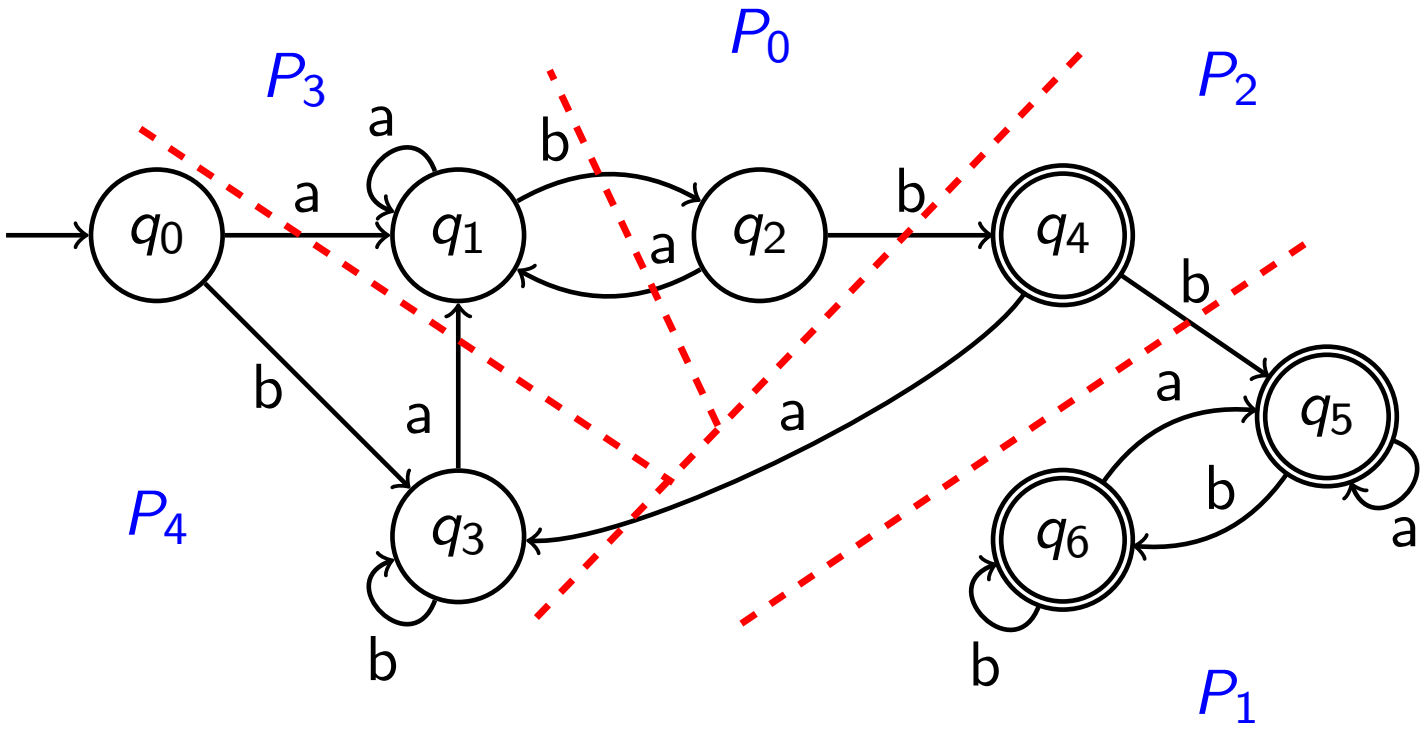
Are the partitions consistent w.r.t. a? Yes.

Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



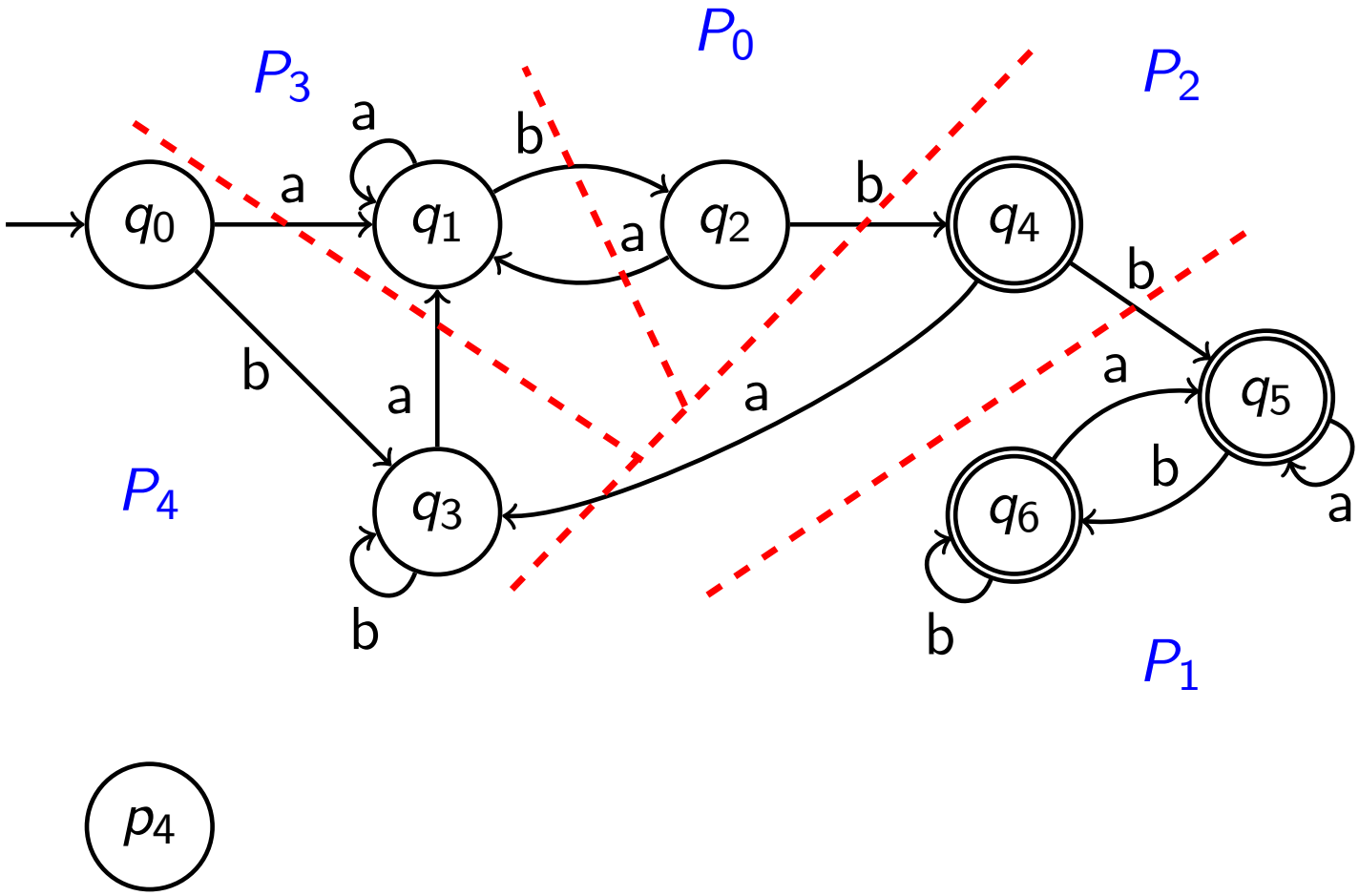
Are the partitions consistent w.r.t. b?

Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)

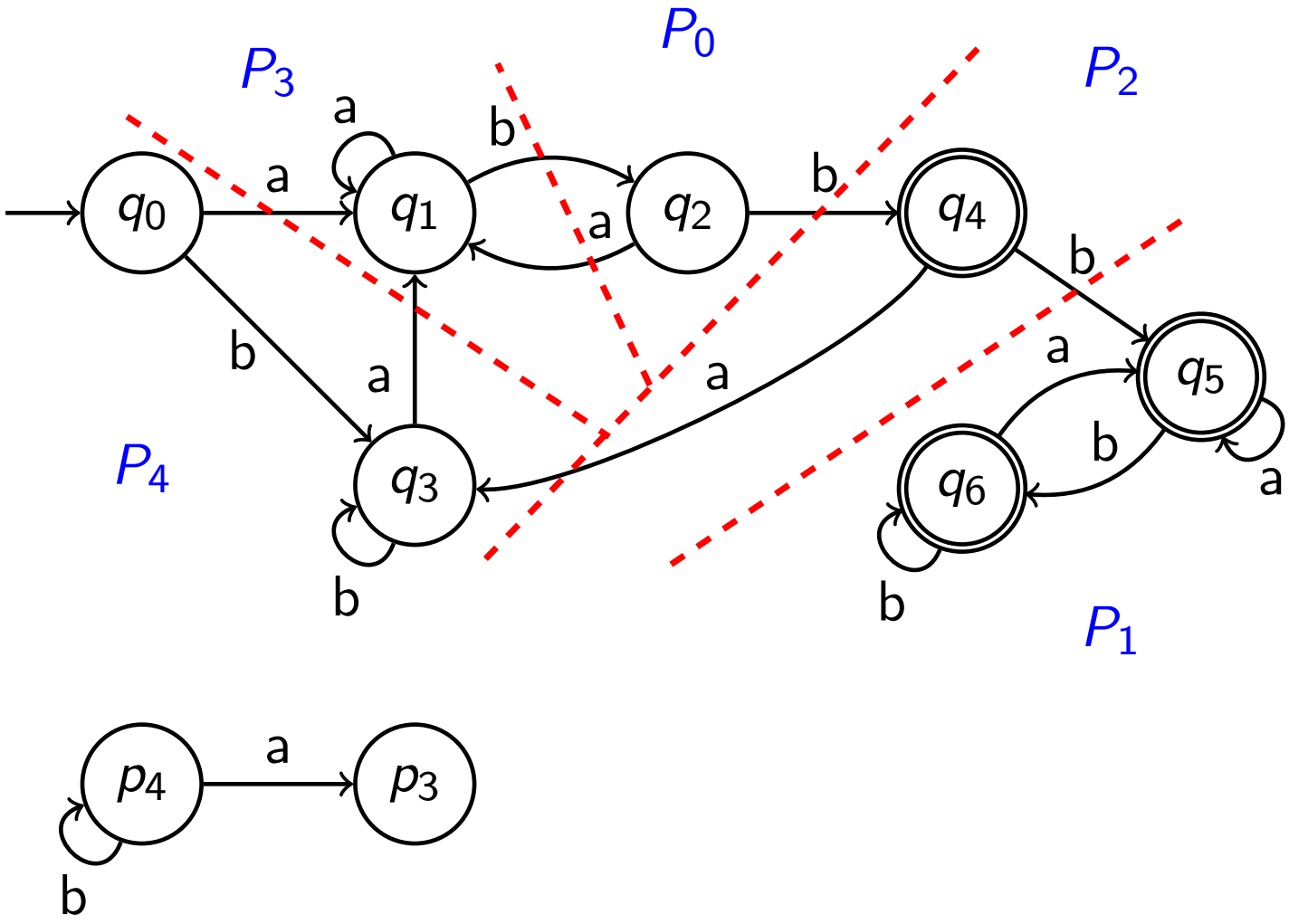


Are the partitions consistent w.r.t. b? Yes, and we're done.

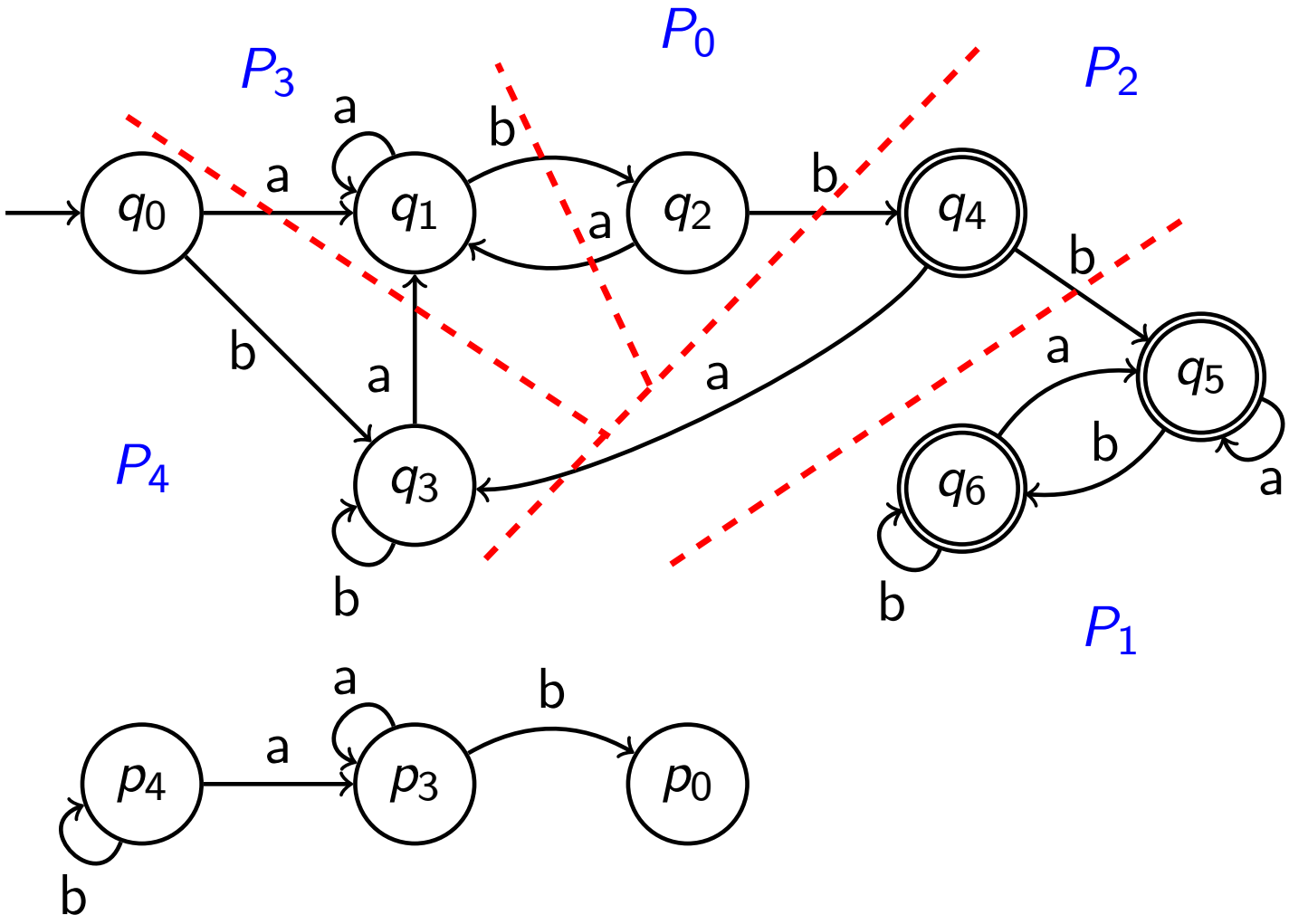
Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



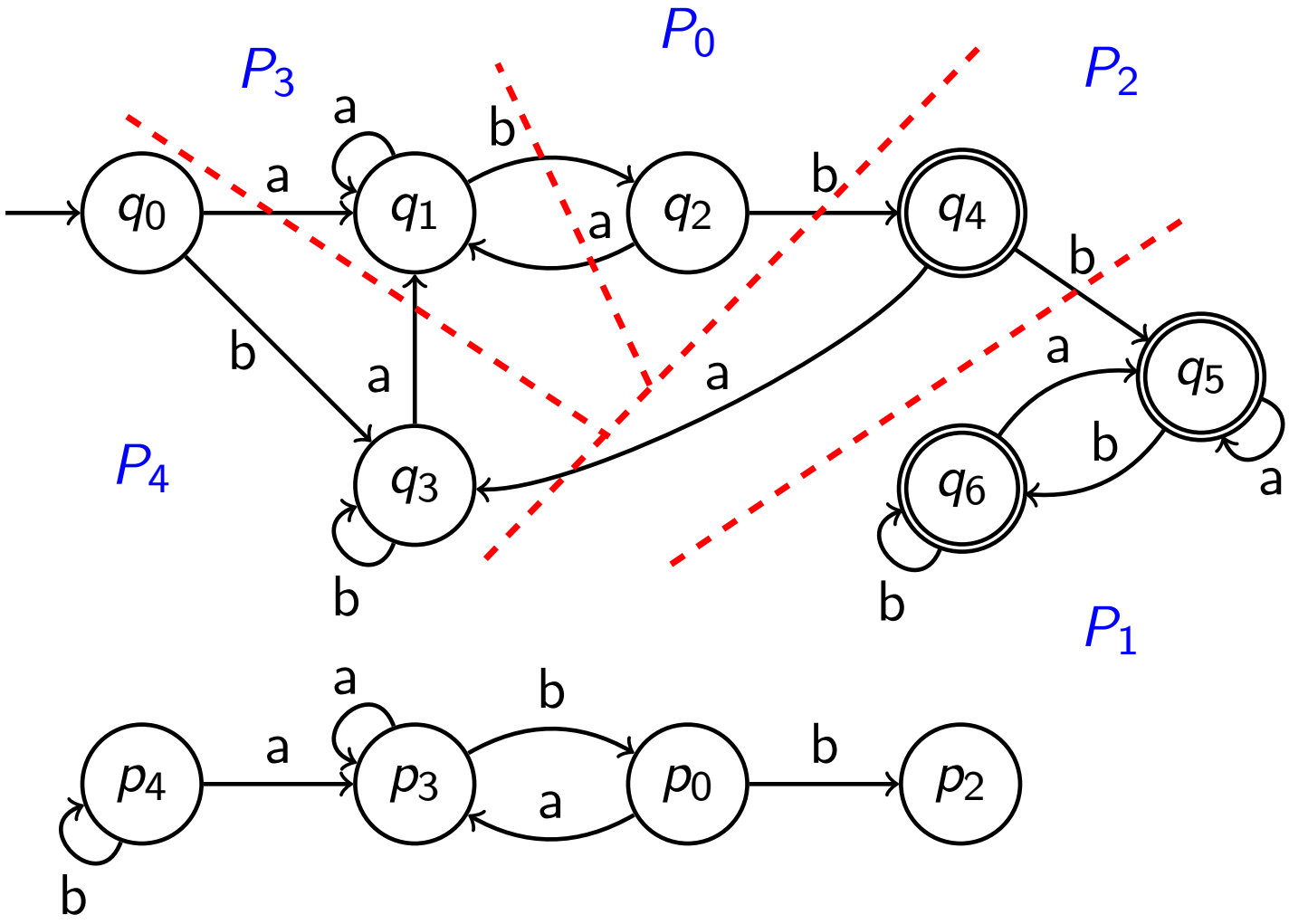
Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



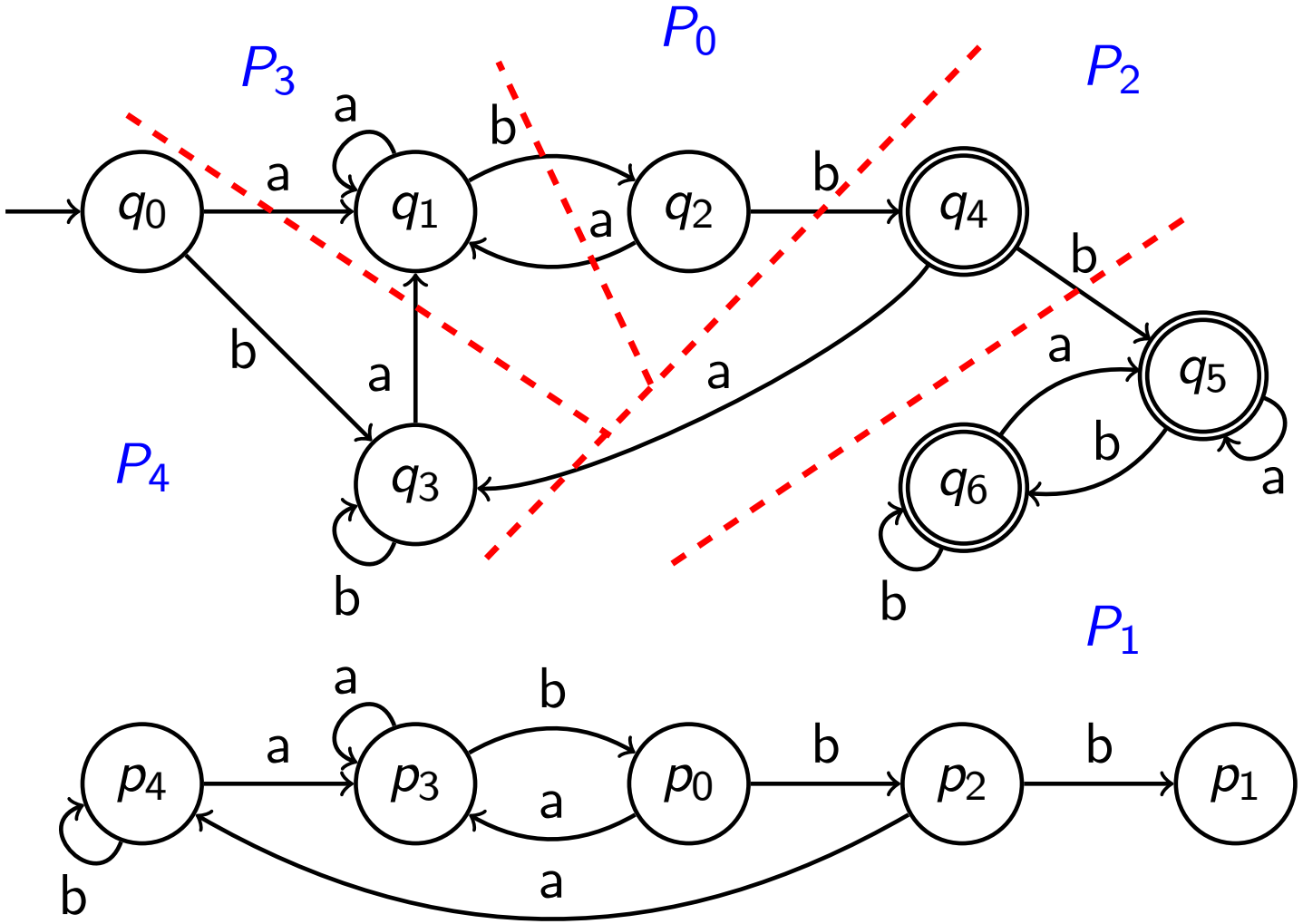
Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



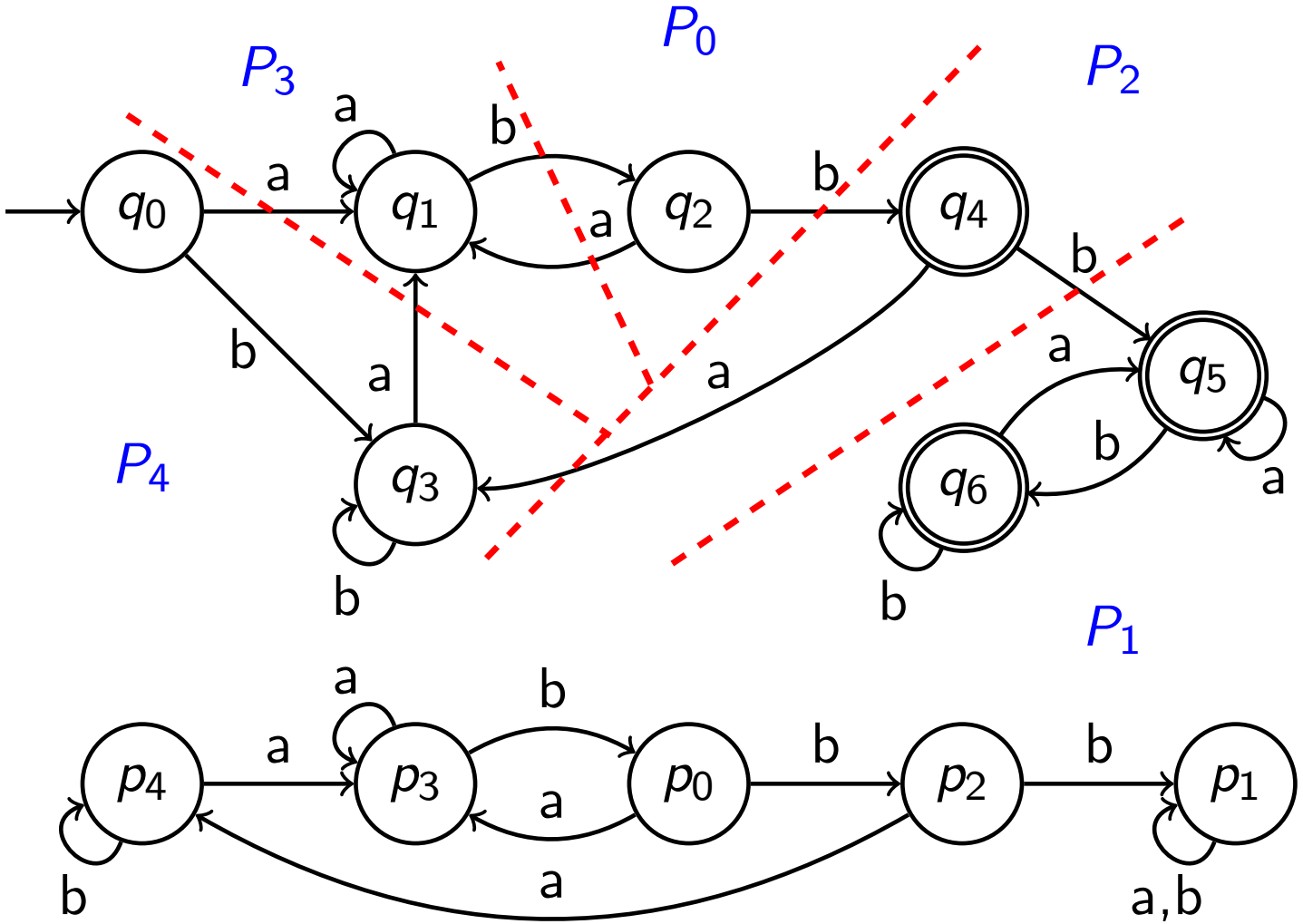
Hopcroft's Algorithm, DFA $\rightarrow$ minimal DFA (2.4.4)



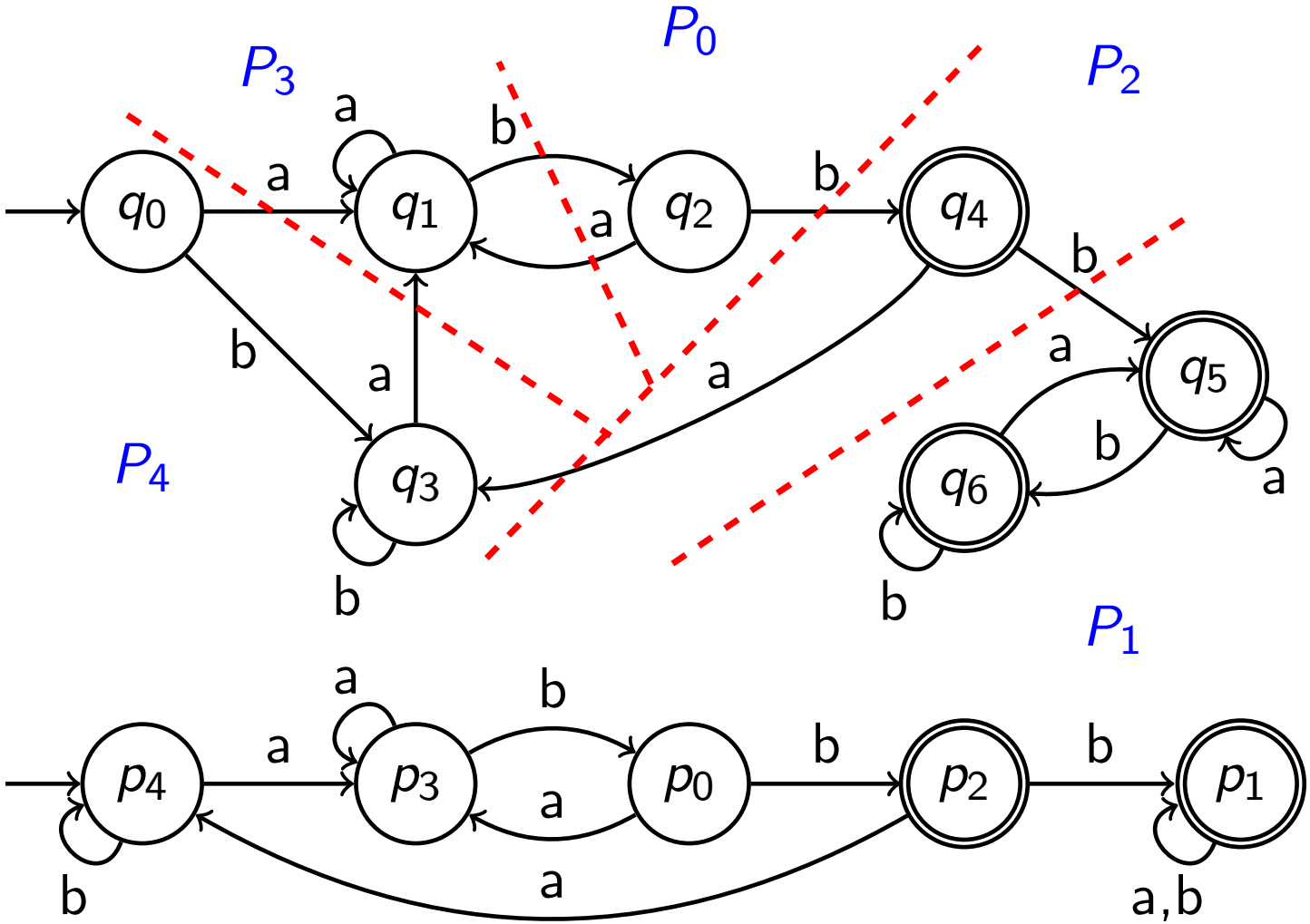
Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



Hopcroft's Algorithm, DFA → minimal DFA (2.4.4)



Bigger Picture

Final take-away: REs, DFAs, NFAs are 3 different mechanisms with **equal expressive power** to define/recognize regular languages.

Depending on the problem at hand, one of them may be the most convenient:

People reason with REs well

$\Uparrow$

NFAs bridge the gap!

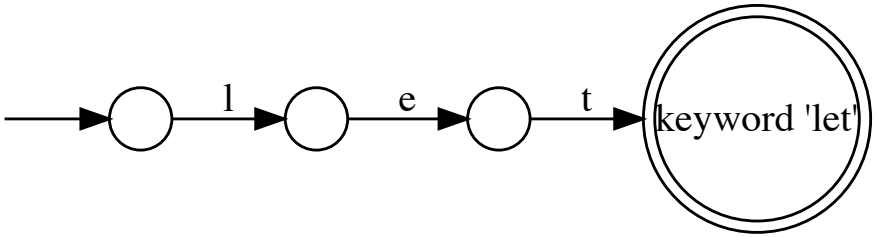
$\Downarrow$

Machines reason with DFAs well

A scanner is a DFA!

- Your scanner accepts as input a sequence of characters and generates tokens — it can be implemented in terms of a DFA

- Here's a DFA which recognizes the keyword "let"



- Your implementation will have to recognize multiple different types of tokens based on a specification

Possible Implementations

- The book describes 3 implementation strategies for scanners
    - Table-driven scanner
    - Direct coded scanner
    - Hand Coded Scanners
- All 3 strategies simulate the DFA which recognizes tokens in the target language

Table Driven Scanner

- Encode the scanner as a table.
  - Table encodes information of type
  - State x Char —> State

	A	B	C	...
s0	s1	S0	S0	
s1	S1	S2	S5	
s2	S5	S1	S2	
...				

Table Driven Scanner

Initialize state to start

While not in error state

    Read in character and transition based on table

While not in accept state and not in initial state

    Roll back to previous character in sequence

Is state accepted?

    If so, generate appropriate token

    Otherwise character string has a syntax error

Limits of Regular Languages

Advantages of Regular Expressions

- Simple & powerful notation for specifying patterns
- Automatic construction of fast recognizers
- Many kinds of syntax can be specified with REs

Example — an expression grammar

Term → [a-zA-Z] ([a-zA-Z] | [0-9])*

Op → + | - | * | /

Expr → (Term Op)* Term

Of course, this would generate a DFA …

If REs are so useful …

Why not use them for everything?

Limits of Regular Languages

Not all languages are regular

RL's $\subset$ CFL's $\subset$ CSL's

You cannot construct DFA's to recognize these languages

- $L = \{ p^k q^k \}$ (parenthesis languages)
- $L = \{ wcw^r \mid w \in \Sigma^* \}$

Neither of these is a regular language (nor an RE)

But, this is a little subtle. You <u>can</u> construct DFA's for

- Strings with alternating 0's and 1's

  $(\varepsilon \mid 1) (01)^* (\varepsilon \mid 0)$
- Strings with and even number of 0's and 1's

RE's can count bounded sets and bounded differences