Théorie des Langages Rationnels

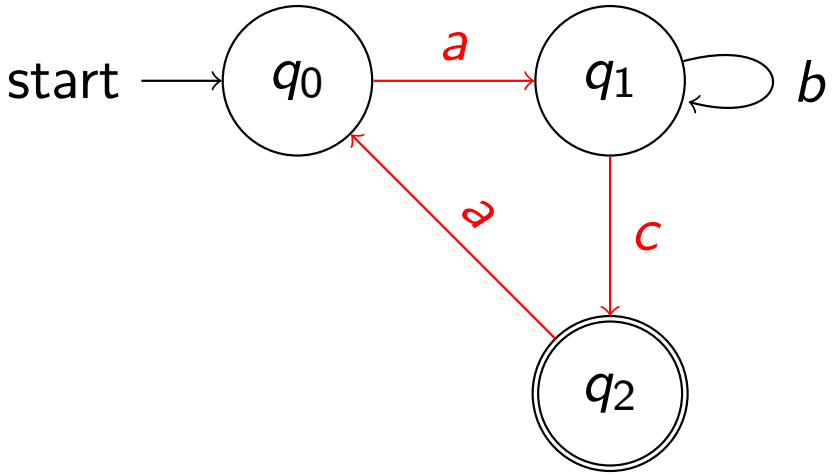
Automata and Regular Expressions

Adrien Pommellet, LRE

February 21, 2023

Completeness

A quick reminder



There are two ways to refuse a word:

- There is a path but **it does not lead to an accepting state**. As an example, consider the word *aca*.
- There is **no path** labelled by said word. As an example, consider the word *acb*.

Completeness

A definition

Completeness

An automaton $\mathcal{A}$ is said to be complete if for every letter $a \in \Sigma$ and every state q of $\mathcal{A}$, there exists from q **at least one** outgoing edge labelled by a.

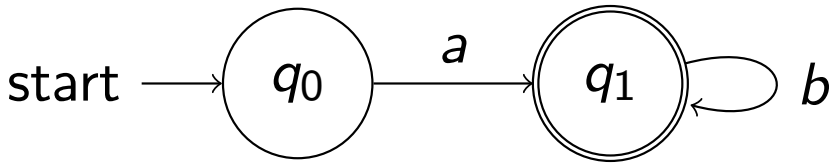
Lemma

*If an automaton $\mathcal{A}$ is complete, then for each word w in Σ^* there is **at least one path** labelled by w.*

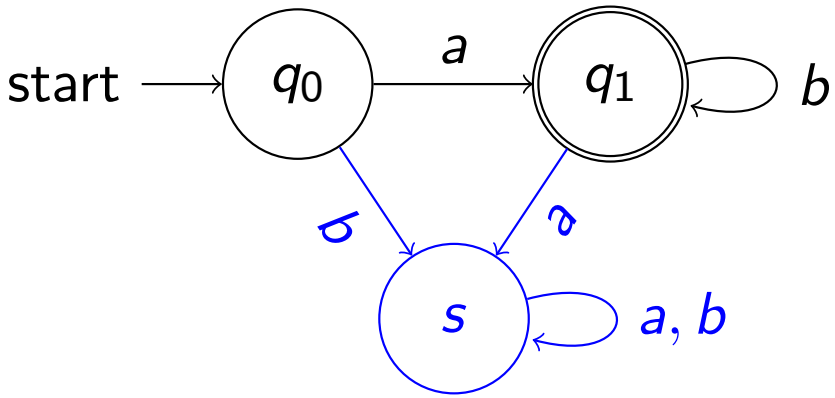
It is not to be mistaken with determinism, for which there exists **at most one** path and outgoing edge.

Practical application

Exercise 1. Find a **complete** automaton equivalent to the automaton on the alphabet $\Sigma = \{a, b\}$ below.



Answer



We create a new state s such that s is not accepting and for each letter $a \in \Sigma$, s features a self-loop labelled by a. We call such a state a **sink state**: once it has been entered, it can no longer be left.

We then redirect all the missing edges towards state s. The resulting automaton is **complete** and **equivalent to the original automaton**.

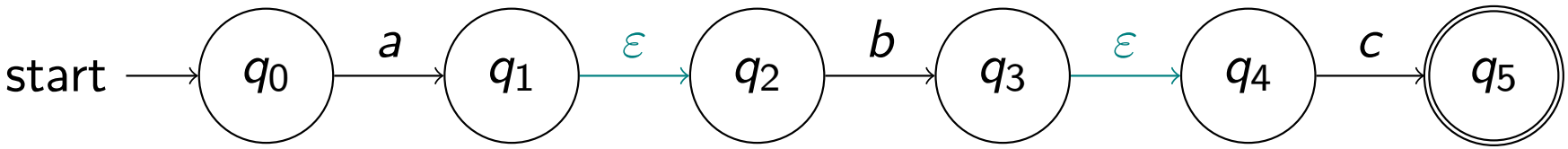
This **completion** algorithm can be applied to **any** finite automaton.

Spontaneous Transitions

Changing the model

We initially designed finite automata so that **one letter** of the input word must be read **each time an edge is taken**.

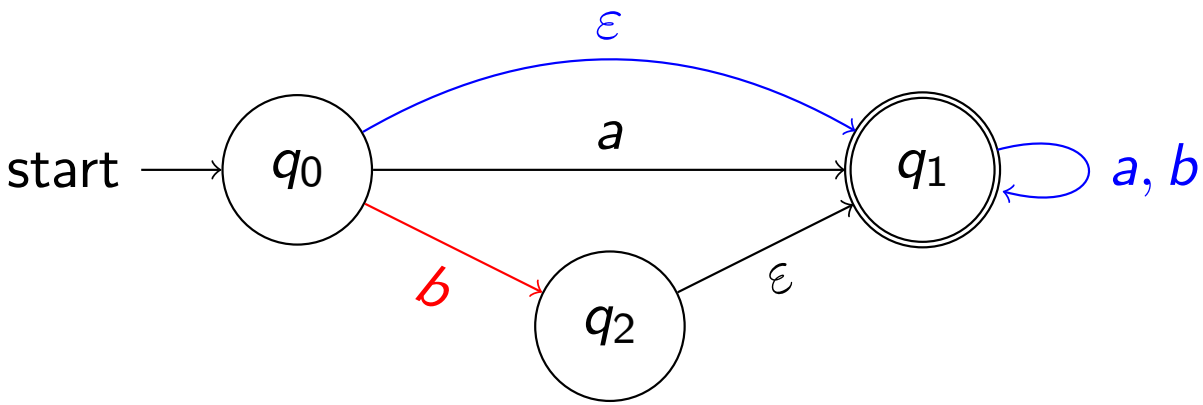
But we can relax this constraint and allow transitions labelled by the empty symbol ε, also known as **spontaneaous transitions**. It is possible to take such a transition **without reading a letter** at all.



Here, there is a path $q_0 \xrightarrow{abc}_{\mathcal{A}}^{*} q_5$ labelled by abc, thus $\mathcal{A}$ accepts abc.

Spontaneous Transitions

What about determinism?



While the automaton above complies with the definition of determinism we introduced previously (one initial state, at most one outgoing transition per letter and state), there are **multiple paths** labelled by the letter b.

Speaking of determinism does not make sense here. These automata are called **non-deterministic finite automata with ε-transitions**, or **ε-NFA**.

Practical Application

Exercise 2. Find a ε-NFA $\mathcal{A}$ recognizing the regular expression $(a^* + bc^*)b$ on the alphabet $\Sigma = \{a, b, c\}$. If possible, try to use a **single initial state**.

Answer

The use of ε-transitions allows us to branch early, using a form of **non-determinism**.

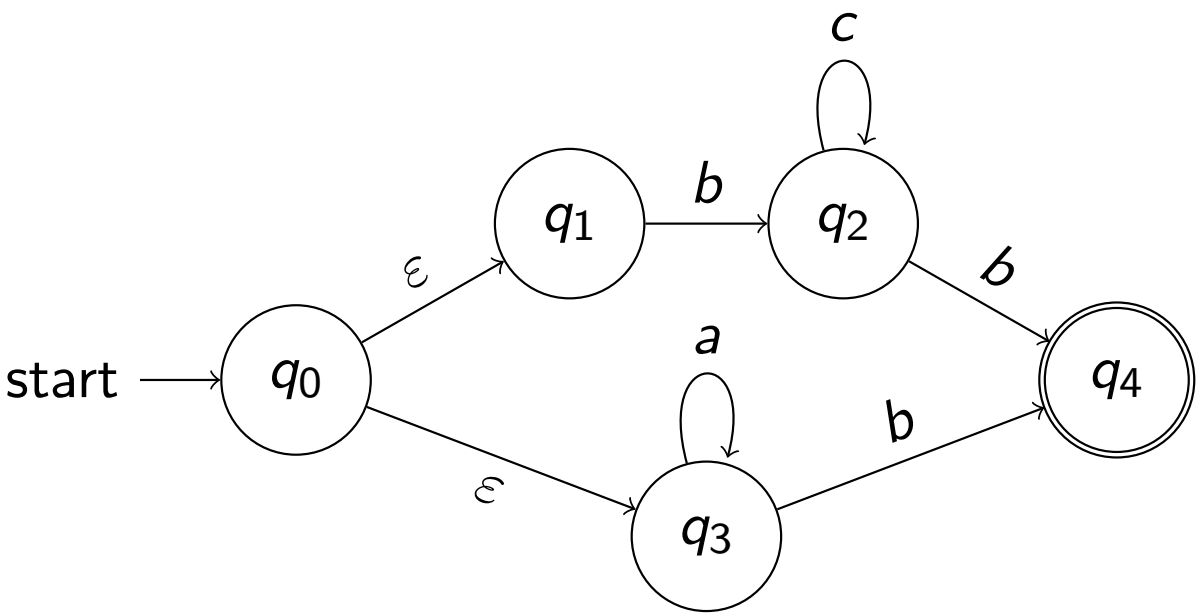


Figure 1: Automaton $\mathcal{A}$.

Thompson's Algorithm

From regular expressions to automata

We can generalize this result to any regular expression:

Theorem (Kleene)

Given a regular expression $e \in \mathrm{Reg}_\Sigma$, there exists a ε-NFA $\mathcal{A}$ on the alphabet Σ such that $\mathcal{L}(e) = \mathcal{L}(\mathcal{A})$.

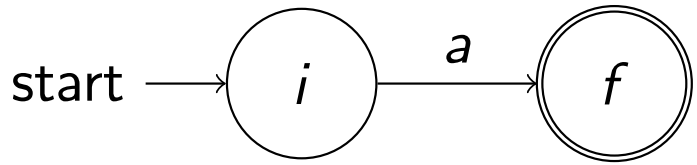
Regular expressions were defined inductively: we will therefore use an **inductive** method known as **Thompson's algorithm** that computes $\mathcal{A}$ knowing e.

Thompson's Algorithm

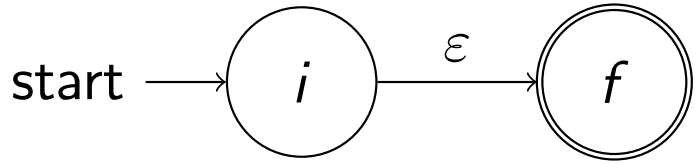
Handling the base cases

We will build $\mathcal{A}$ in such a manner it has a **single initial state** i and a **single, distinct accepting state** f.

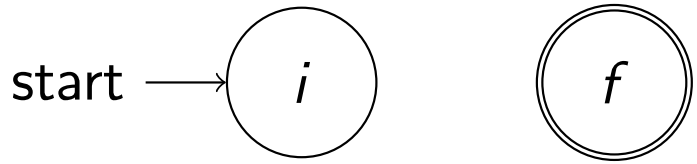
$a \in \Sigma$



ε



$\emptyset$



Thompson's Algorithm

Concatenation

Let $e_1, e_2 \in \mathrm{Reg}_\Sigma$ and $\mathcal{A}_1$, $\mathcal{A}_2$ such that $\mathcal{L}(e_1) = \mathcal{L}(\mathcal{A}_1)$, $\mathcal{L}(e_2) = \mathcal{L}(\mathcal{A}_2)$.

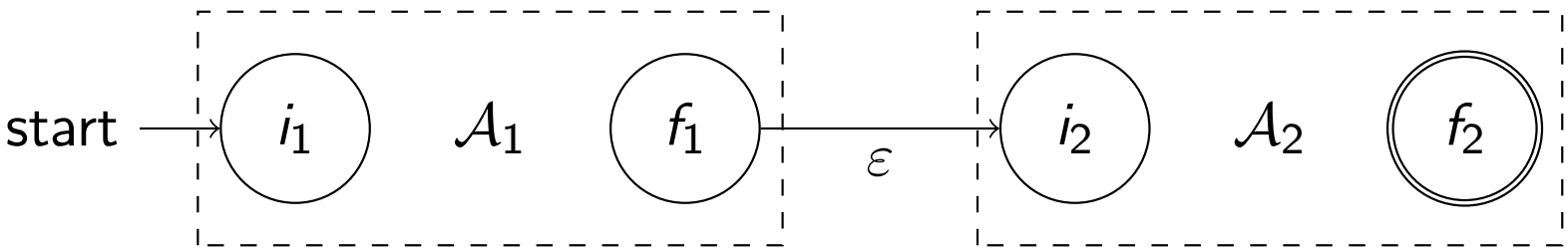
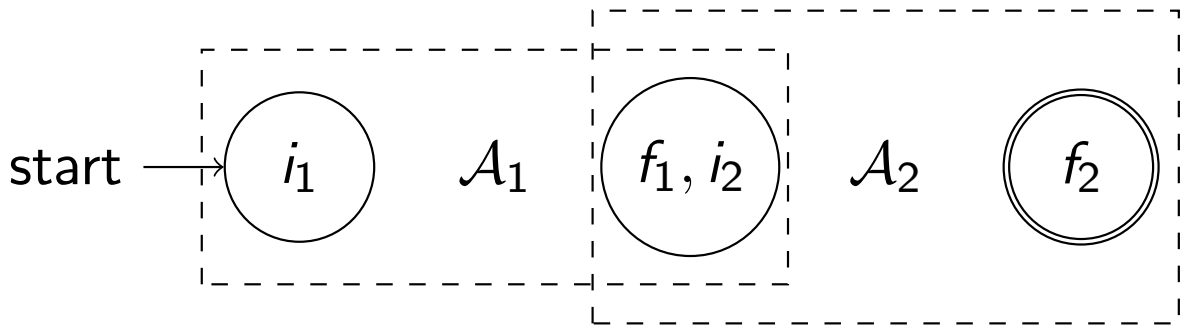


Figure 2: An automaton recognizing $\mathcal{L}(e_1 \cdot e_2)$.

Thompson's Algorithm

It's not that simple.

The following pattern is intuitive but **wrong** in some cases:

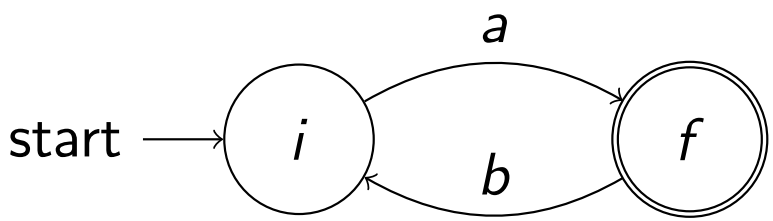


Indeed, let us design a **counter-example**.

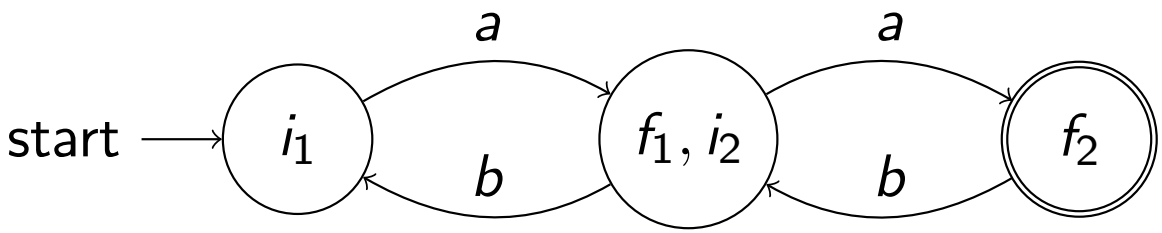
Thompson's Algorithm

A subtle counter-example

The following automaton recognizes the regular expression $a(ba)^*$.



But the automaton below **does not** recognize the regular expression $a(ba)^*a(ba)^*$. Consider indeed the counter-example $aabbaa$.



Thompson's Algorithm

Union

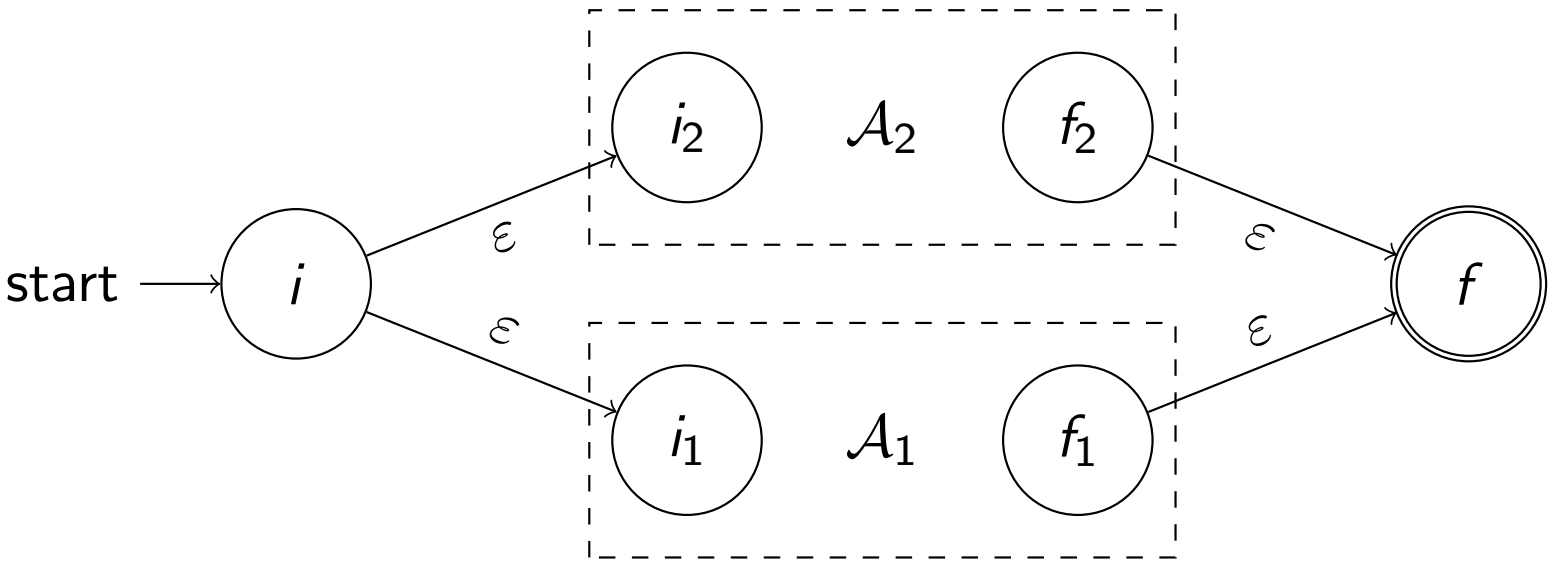


Figure 3: An automaton recognizing $\mathcal{L}(e_1 + e_2)$.

Thompson's Algorithm

Kleene's star

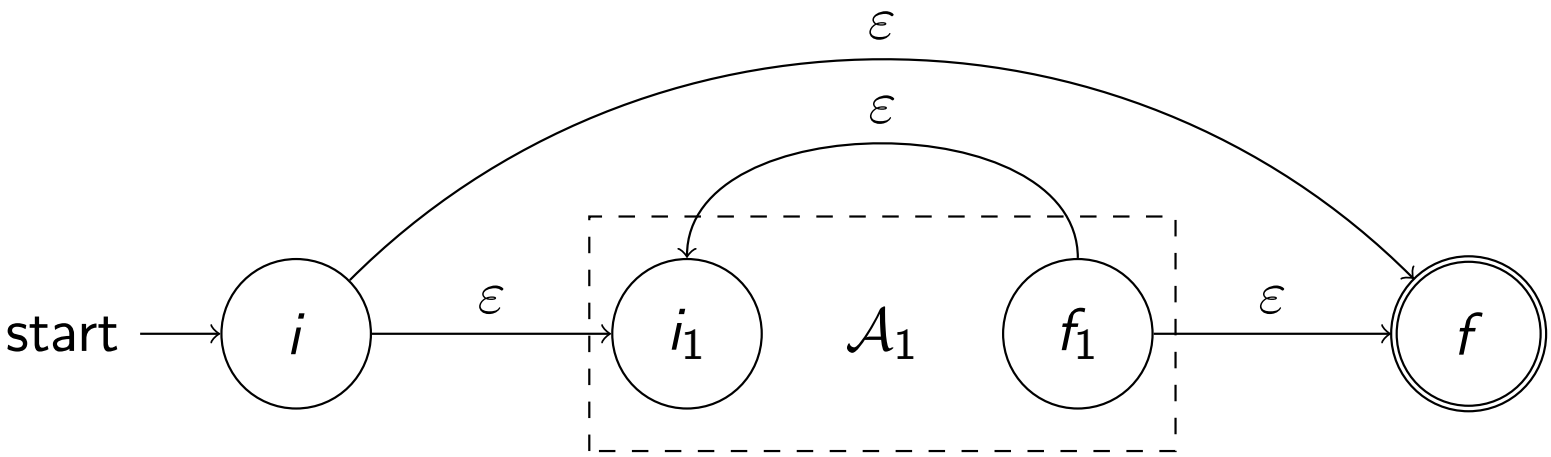


Figure 4: An automaton recognizing $\mathcal{L}(e_1^*)$.

Don't forget that $\varepsilon \in \mathcal{L}(e_1^*)$, always, even if $\varepsilon \notin \mathcal{L}(e_1)$.

Practical application

Exercise 3. Compute **according to the Thompson algorithm** a ε-NFA $\mathcal{A}$ recognizing the regular expression $a^* + bc$ on the alphabet $\Sigma = \{a, b, c\}$.

Answer

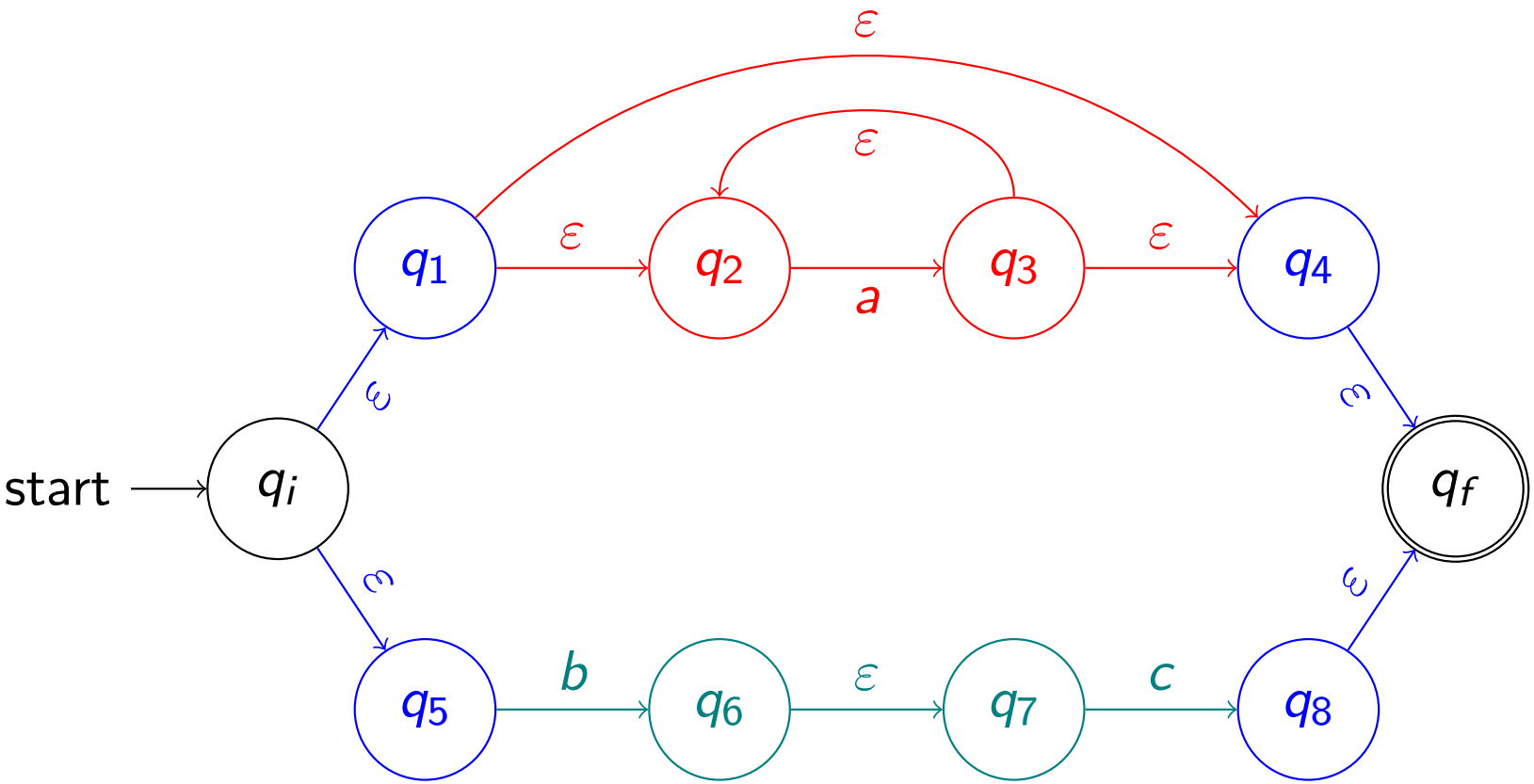


Figure 5: An automaton $\mathcal{A}$ recognizing $\mathcal{L}(a^* + bc)$.

Thompson's Algorithm

A common pattern

It is possible to predict the **size** (in terms of states) of the resulting automaton before even computing it:

Lemma

*Let $e \in \mathrm{Reg}_\Sigma$ be a regular expression, and let n be the number of occurrences in e of ε, $\emptyset$, $+$, $*$, and letters in Σ (**excluding parentheses**). Then the automaton computed by the Thompson algorithm has exactly $2n$ states.*

Proving this lemma is rather straightforward: every single pattern described previously adds **exactly two new states** to the automaton, with the exception of the concatenation rule.