Theory of Computation
Regular Languages

Dimitris Diochnos
School of Computer Science
University of Oklahoma

Outline

Table of Contents

Finite Automata

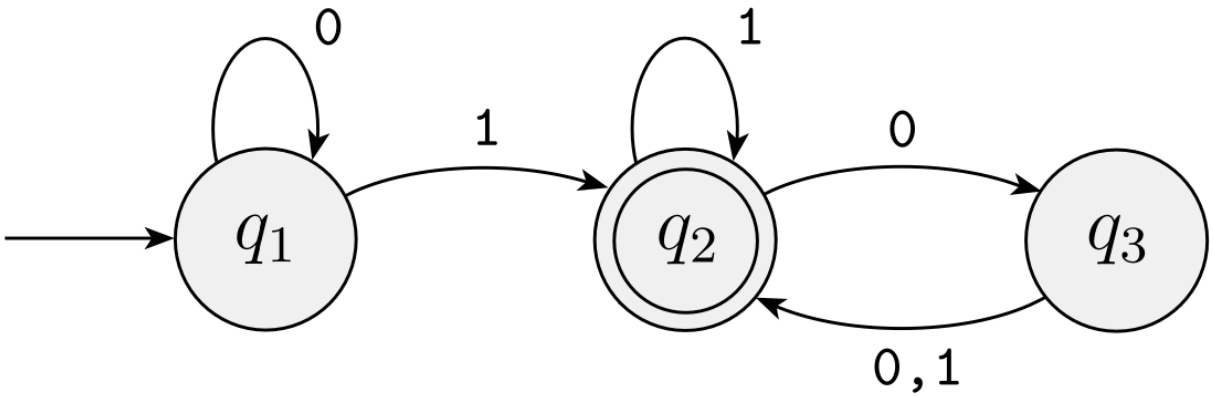
Simplest computational model:

Finite state machine or **finite automaton**

- Good model for computers with extremely limited amount of memory (e.g., automatic door).
- Markov chains are probabilistic counterparts of finite automata.

Finite Automata – example

Example 1 (A simple finite automaton M_1)



M_1 recognizes strings that:

- have at least one 1,
- end in a 1,
- end in an even number of zeros, following the last 1 (i.e., there is at least one 1)

Formal definition of a finite automaton

Finite automaton

A **finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where:

- Q is a finite set called the **states**,
- Σ is a finite set called the (input) **alphabet**,
- $\delta : Q \times \Sigma \to Q$ is the **transition function** (it includes the rules for moving),
- $q_0 \in Q$ is the **start state**, and
- $F \subseteq Q$ is the **set of accept states** (final states).

Language

Language

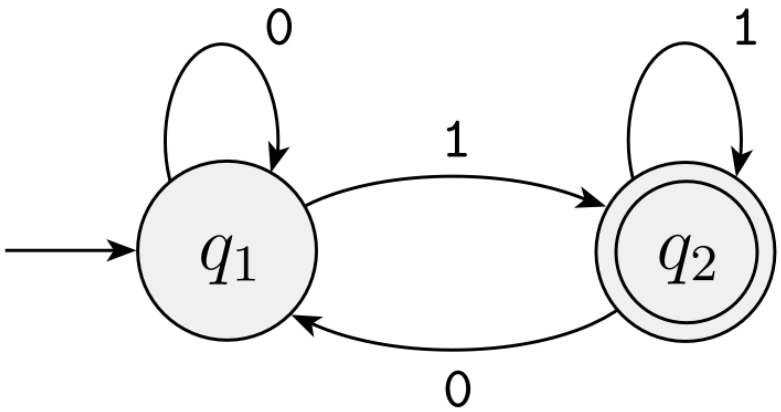
A Language of machine M is the set of strings that M accepts.
Notation: $L(M) = A$

We say that M recognizes A or that M accepts A.

In the example 1, with M_1, above, $L(M_1) = A$ where $A = \{w | w \text{ contains at least one 1 and an even number of zeros follow the last 1}\}$.

Examples of finite automata (a)

Example 2 (Automaton M_2)



$$M_2 = (\{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_2\})$$

δ	0	1
q_1	q_1	q_2
q_2	q_1	q_2

State diagram and formal description contain the same information.

Remark

A good way to *understand* a machine:

Try it on some sample input strings.

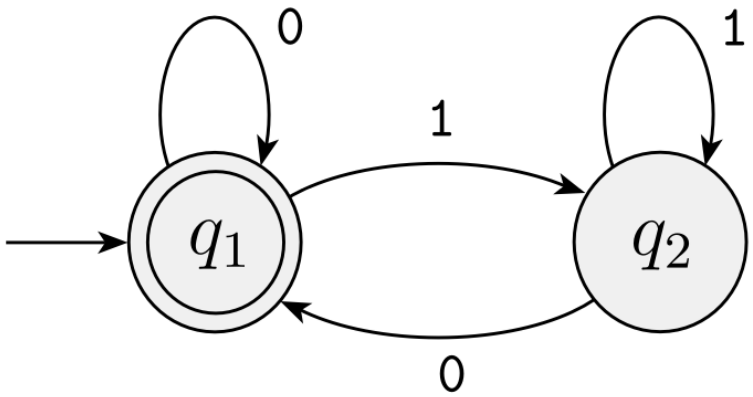
Example 3

String 1101 is accepted,
String 110 is rejected.

Then:
M_2 accepts all strings that end with a 1.
Thus, $L(M_2) = \{w | w \text{ ends in a } 1\}$.

Examples of finite automata (b)

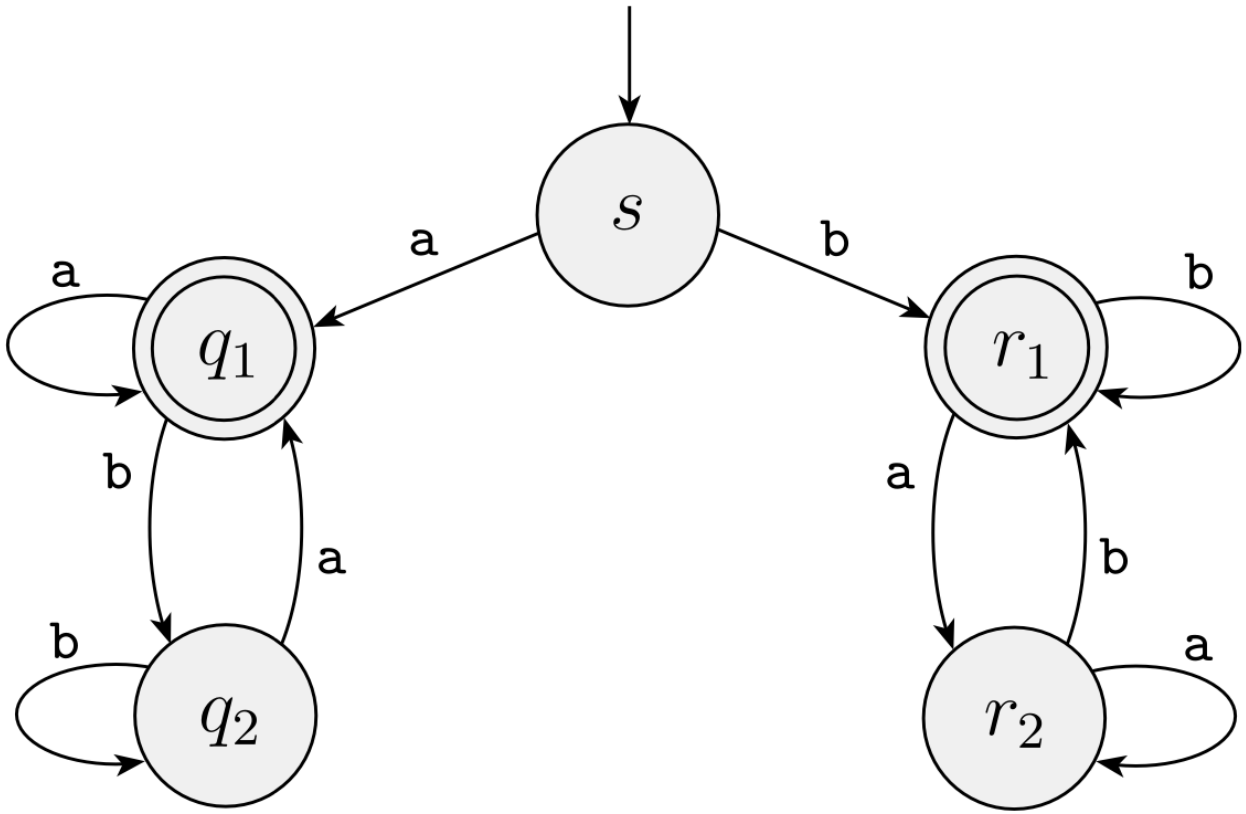
Example 4 (Automaton M_3)



$$L(M_3) = \{w | w \text{ is the empty string } \varepsilon \text{ or ends in a } 0\}$$

Examples of finite automata (c)

Example 5 (Automaton M_4)



$$L(M_4) = \{w | w \text{ starts and ends with the same symbol}\}$$

Formal definition of computation

Computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \cdots w_n$ be a string where each w_i is a member of the alphabet Σ. Then M **accepts** w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three conditions:

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots, n-1$, and
3. $r_n \in F$.

Condition 1 says that the machine starts in the start state.
Condition 2 says that the machine goes from state to state according to the transition function.
Condition 3 says that the machine accepts its input if it ends up in an accept state.
We say that M **recognizes language** A if $A = \{w | M \text{ accepts } w\}$.

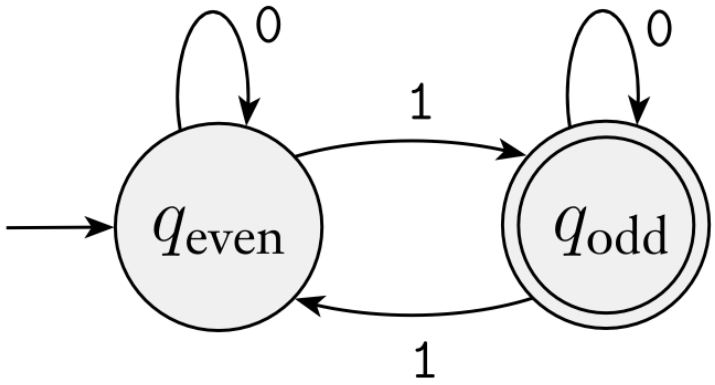
Definition of regular language

Regular language

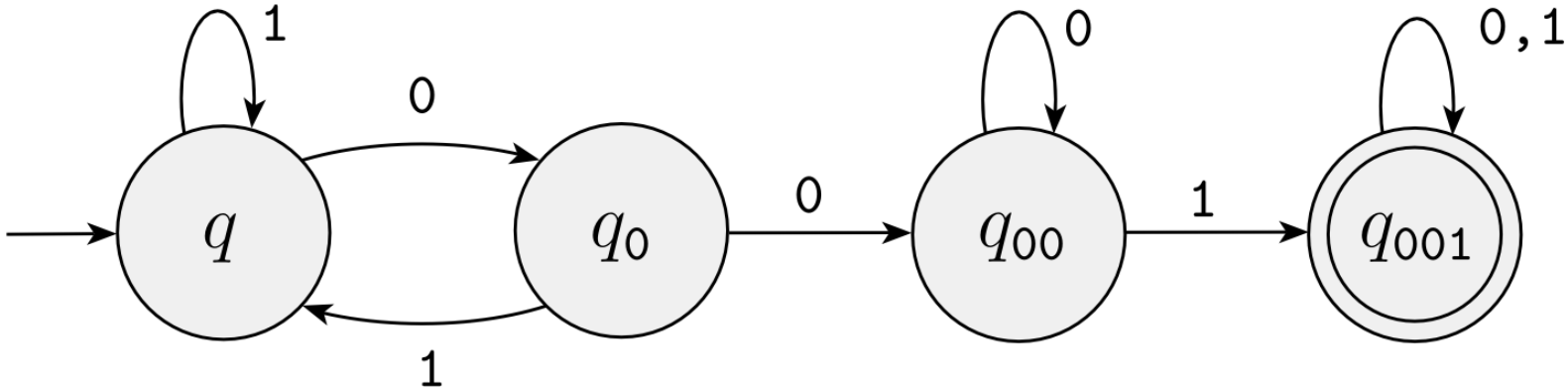
A language is called a **regular language** if some finite automaton recognizes it.

Designing Finite Automata

Example recognizing odd number of 1s:



Example recognizing 001 as a substring:



Regular Operations

Definition

Let A and B be languages. We define the regular operations **union**, **concatenation**, and **star** as follows:

- **Union**: $A \cup B = \{x | x \in A \text{ or } x \in B\}$.
- **Concatenation**: $A \circ B = \{xy | x \in A \text{ and } y \in B\}$.
- **Star**: $A^* = \{x_1 x_2 \dots x_k | k \geq 0 \text{ and each } x_i \in A\}$.

Remark: The empty string ε is always a member of A^*, no matter what A is.

Closeness under Operations

- A collection of objects is **closed** under some operation if applying that operation to members of the collection returns an object still in the collection.
- The collection of regular languages is closed under all three of the regular operations.

Example 6 (Closeness)

Let the alphabet Σ be the standard 26 letters $\{a, b, \ldots, z\}$.
If $A = \{\text{good, bad}\}$ and $B = \{\text{boy, girl}\}$, then
$A \cup B = \{\text{good, bad, boy, girl}\}$,
$A \circ B = \{\text{goodboy, goodgirl, badboy, badgirl}\}$, and
$A^* = \{\varepsilon, \text{good, bad, goodgood, goodbad, badgood, badbad,}$
$\text{goodgoodgood, goodgoodbad, goodbadgood, goodbadbad, } \ldots\}$.

Closeness under Union Operation

Theorem

The class of regular languages is closed under the union operation.
In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Proof Idea

Construct M from M_1 (that recognizes A_1) and M_2 (that recognizes A_2).
Simulate both M_1 and M_2 *simultaneously*, and accept if either of the simulations accept.

Remark: We can **not** "rewind the input tape" to try the simulation on M_2.

Closeness under Union Operation (cont.)

Proof

Let M_1 recognize A_1, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$,
and M_2 recognize A_2, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.
Construct M to recognize $A_1 \cup A_2$, where $M = (Q, \Sigma, \delta, q_0, F)$.

1. $Q = \{(r_1, r_2) | r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$.
   This set is the **Cartesian product** of sets Q_1 and Q_2 and is written $Q_1 \times Q_2$.
2. Σ is the same as in M_1 and M_2.
3. $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$.
4. $q_0 = (q_1, q_2)$.
5. $F = \{(r_1, r_2) | r_1 \in F_1 \text{ or } r_2 \in F_2\}$.

Correctness is straightforward.

Closeness under Union Operation (cont.)

Remarks on the proof:

- The last expression, $F = \{(r_1, r_2) | r_1 \in F_1 \text{ or } r_2 \in F_2\}$, is the same as $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$, which is not the same as $F = F_1 \times F_2$ obviously.
- $F = F_1 \times F_2$ proves closure for intersection.

Table of Contents

Motivation for Nondeterminism

If we wanted to prove closure under the concatenation operator $\circ$ for two regular languages A_1 and A_2 (i.e., $A_1 \circ A_2$), then a main problem is that for a given $w \in A_1 \circ A_2$ we do not know where to break w in w_1, w_2 words such that $w_1 \in A_1$, $w_2 \in A_2$ and $w = w_1 \circ w_2$.
To solve this problem, we introduce nondeterminism.

Thus we have:

- DFA: Deterministic Finite Automaton
- NFA: Nondeterministic Finite Automaton

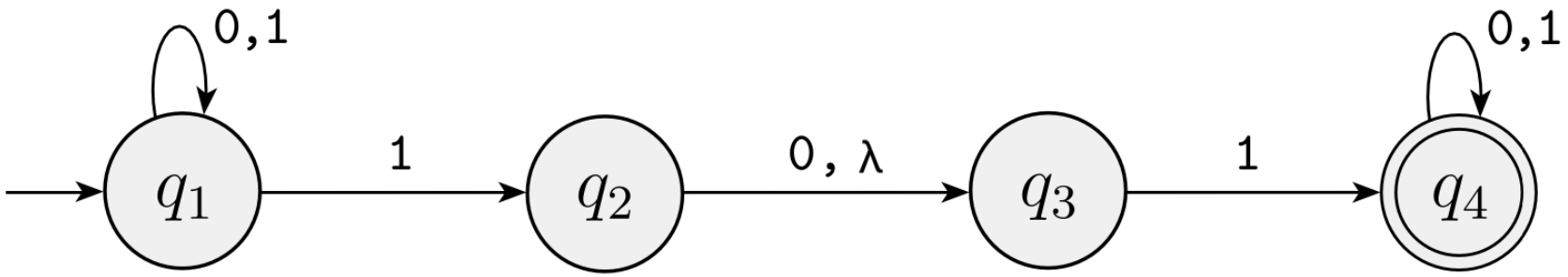
Note: The plural for Automaton is *Automata* or Automatons.

Difference between DFAs and NFAs

NFAs:

- may have zero, one, or many arrows exiting from each state for each alphabet symbol,
- may have zero, one, or many arrows exiting from each state with the label ε.

Example:



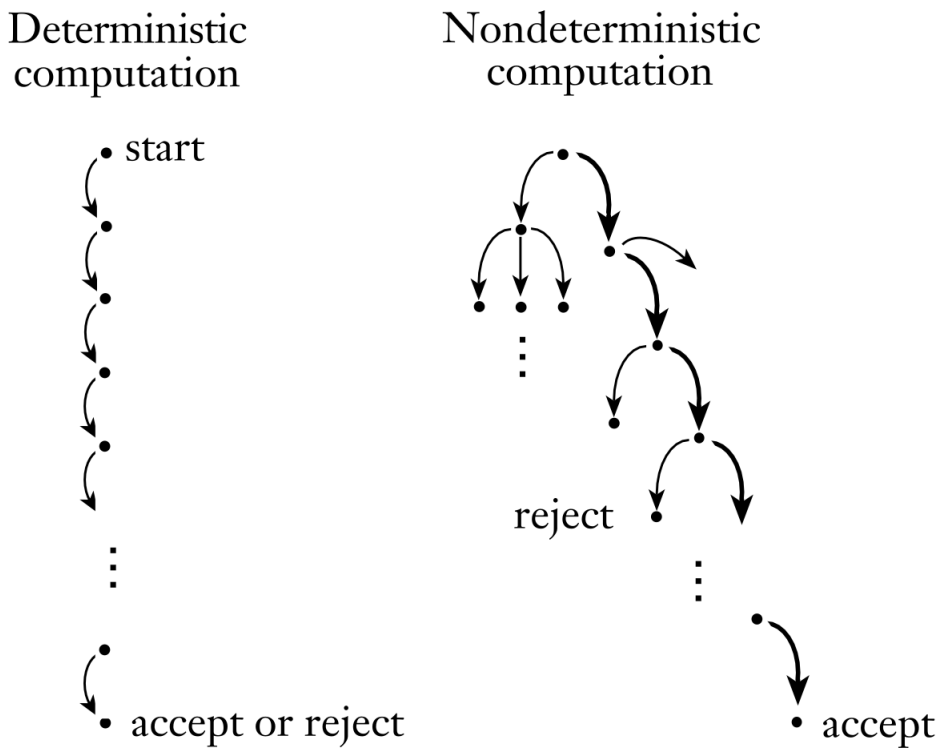
How do NFAs compute?

- We reach a state with multiple ways to proceed.
  $\Rightarrow$ the machine (automaton) splits into multiple copies of itself and **follows all the possibilities in parallel** (each copy takes one of the possible ways to proceed).
- We reach a state where no branch of the computation exiting from it can compute the current symbol
  $\Rightarrow$ the particular copy of the machine dies (along with the branch of the computation associated with it.
- If any of the copies of the machine is in an accept state at the end of the input, the NFA accepts the input string.
- States with an ε symbol on an exiting arrow is encountered
  $\Rightarrow$ splits into multiple copies.

Summing up

Nondeterminism may be seen:

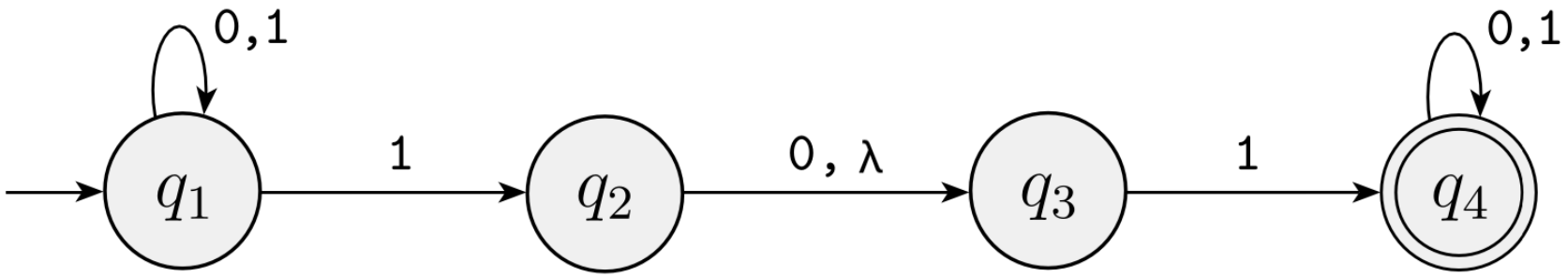
– as a parallel computation with multiple (independent) "processes" running in parallel,
– as a tree of possibilities.



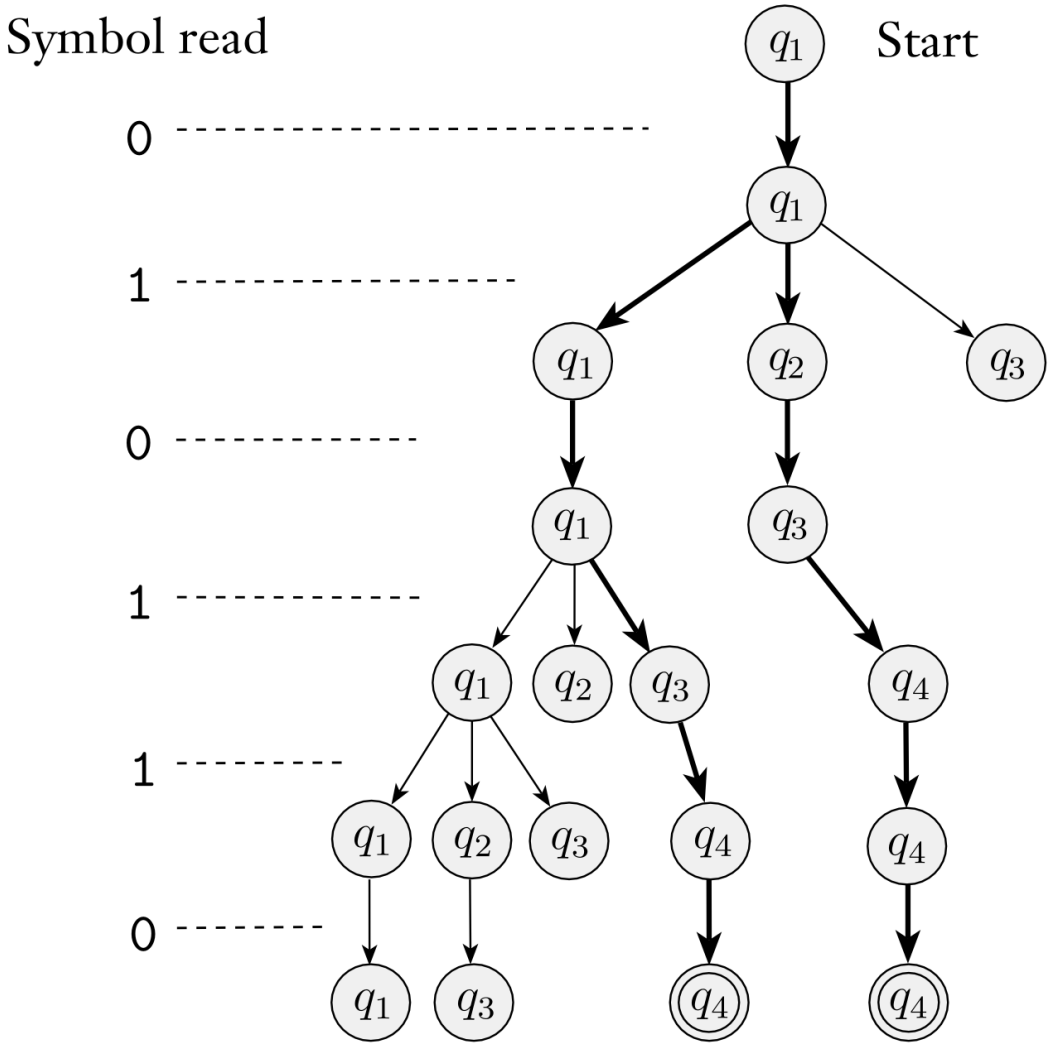
Example

Example 7 (Computation by automaton)

Input: 010110
is given to the following automaton.
What is the computation that takes place?



Example (cont.)



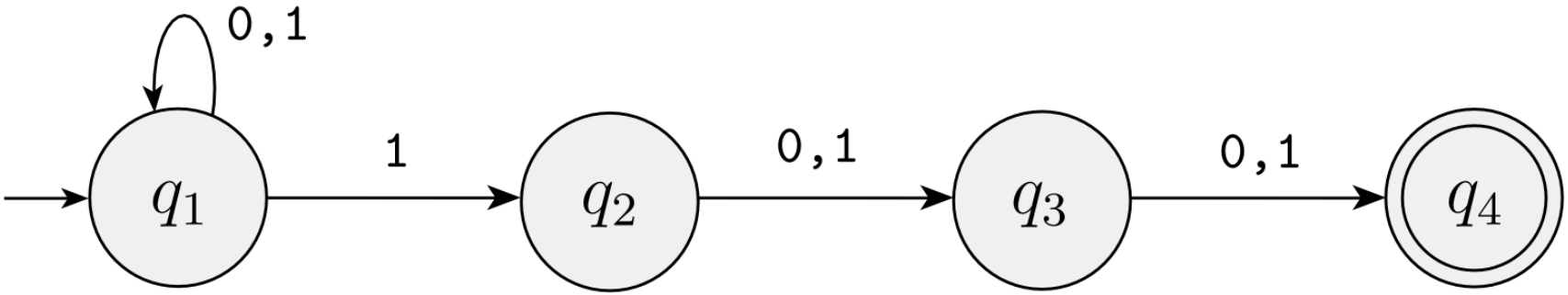
The computation of the automaton above on input 010110

Example

Example 8 (Automaton design)

Let A be the language consisting of all strings over $\{0, 1\}$ containing a 1 in the third position from the end (e.g., $000100 \in A$ but $0011 \notin A$).
Design an automaton that recognizes A and describe it.

The following NFA recognizes A.



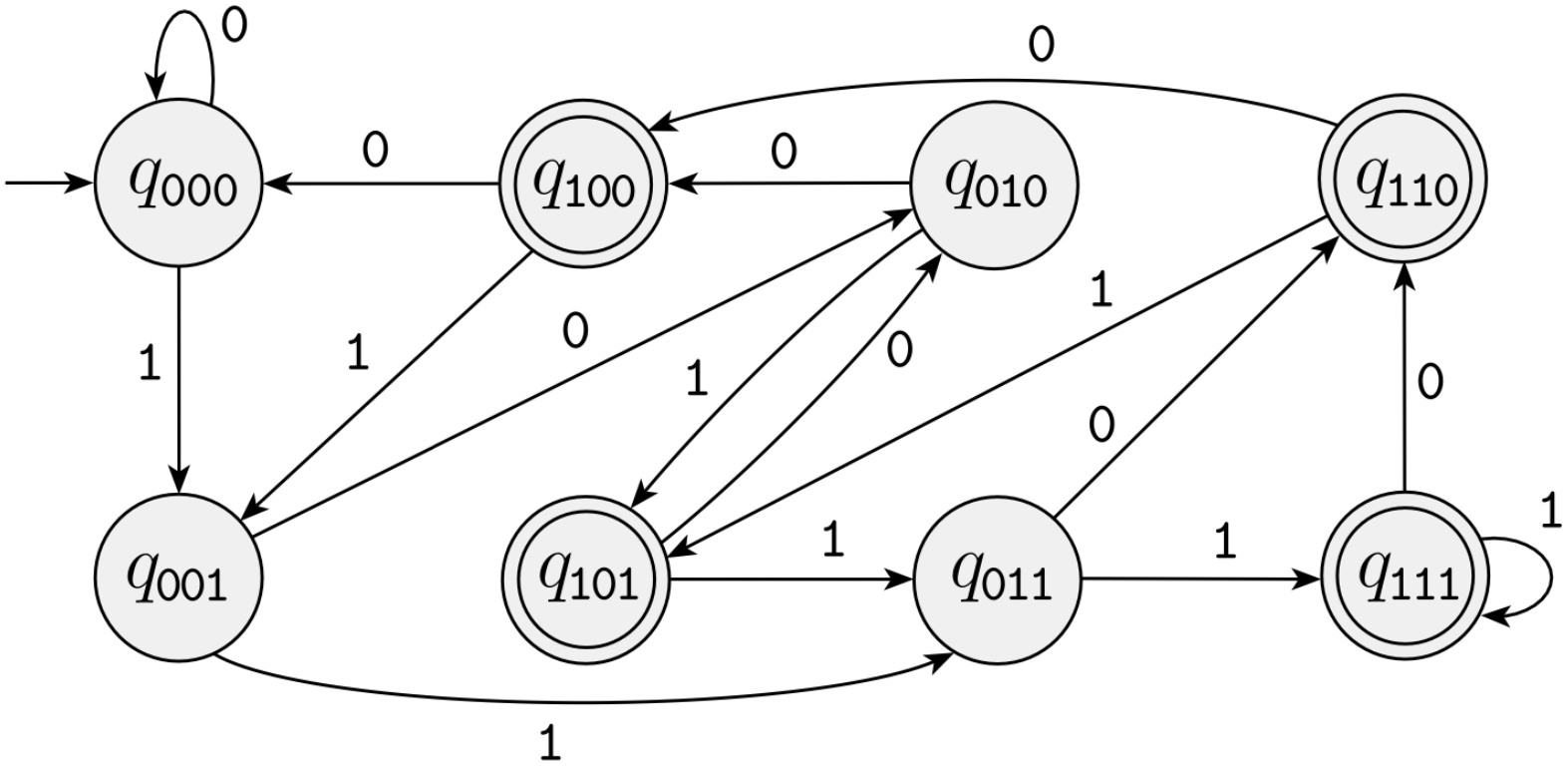
Idea

It stays in q_1 until it "guesses" that it is three places from the end. If the input symbol is 1, it branches to q_2 and uses q_3 and q_4 to "check" on whether its guess was correct.

Example (cont.)

Example 9 (Conversion of NFA to DFA)

Convert the above NFA into an equivalent DFA.



This is the smallest DFA for language A, containing 8 states.

Nondeterministic Finite Automaton

Definition

A **nondeterministic finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

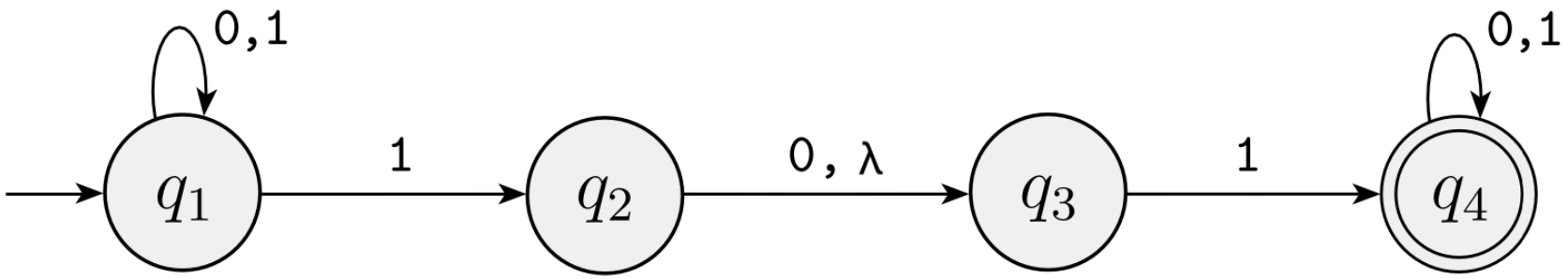
1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta : Q \times \Sigma_\varepsilon \to P(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Note:
$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$
$P(Q) =$ powerset of Q

Example

Example 10

Recall the NFA below that accepts all strings that contain either 101 or 11 as a substring. Provide its formal description.



Example (cont.)

The formal description of the above NFA is $(Q, \Sigma, \delta, q_1, F)$, where

- $Q = \{q_1, q_2, q_3, q_4\}$,
- $\Sigma = \{0, 1\}$,
- δ is given as

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

- q_1 is the start state,
- $F = \{q_4\}$.

NFA accepts w

We say that an NFA N accepts w if we can write w as $w = y_1 y_2 \cdots y_m$, where each y_i is a member of Σ_ε and a sequence of states $r_0, r_1, \ldots, r_m$ exists in Q with three conditions:

1. $r_0 = q_0$,
2. $r_{i+1} \in \delta(r_i, y_{i+1})$, for $i = 0, \ldots, m-1$, and
3. $r_m \in F$.

Equivalence of NFAs and DFAs

The equivalence between NFAs and DFAs is:

- **Surprising** because NFAs appear to have more power than DFAs,
- **Useful** because NFAs are usually much easier to describe than DFAs for a given language.

We say that two machines are **equivalent** if they recognize the same language.

Equivalence of NFAs and DFAs

Theorem 11

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

Proof idea

Construct a DFA M that simulates the NFA N that recognizes some language A.
We need to keep track of the set of states that are active during the computation performed by N.
$\Rightarrow$ If N has k states $\Rightarrow$ the DFA will have 2^k states (one for each subset of the k states).

Equivalence of NFAs and DFAs

Proof

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A.
Construct DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A.
(Assume for now that N has no ε arrows.)

1. $Q' = P(Q)$.
2. For $R \in Q'$ and $a \in \Sigma$, let
   $\delta'(R, a) = \{q \in Q | q \in \delta(r, a) \text{ for some } r \in R\}$.
   Another way of writing it down:
   $\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$.
3. $q_0' = \{q_0\}$.
4. $F' = \{R \in Q' | R \text{ contains an accept state of } N\}$.

Equivalence of NFAs and DFAs

Proof (cont.)

For $R \subseteq Q$ let
$\Lambda(R) = \{q \in Q \mid$
q can be reached from R by traveling along 0 or more ε arrows$\}$.
Thus, $\delta'(R, a) = \{q \in Q | q \in \Lambda(\delta(r, a))$ for some $r \in R\}$.
Additionally, $q_0' = \Lambda(\{q_0\})$ are all the possible states that can be reached from the start state of N (i.e., q_0) after following ε arrows (and not reading yet the input).
The construction of M is correct and thus we can simulate the NFA N.

Equivalence of NFAs and DFAs

Corollary 12

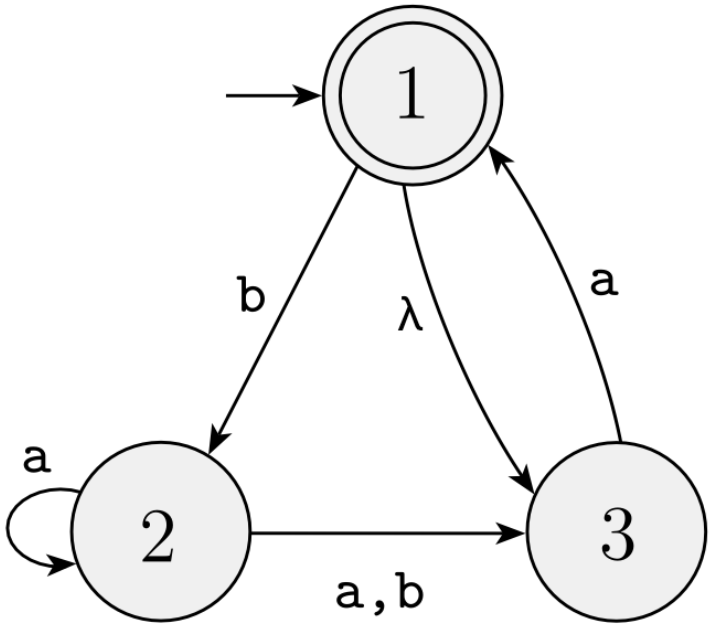
A language is regular if and only if some nondeterministic finite automaton recognizes it.

(Discuss both directions of "if and only if" by recalling the definition and the previous theorem.)

Example

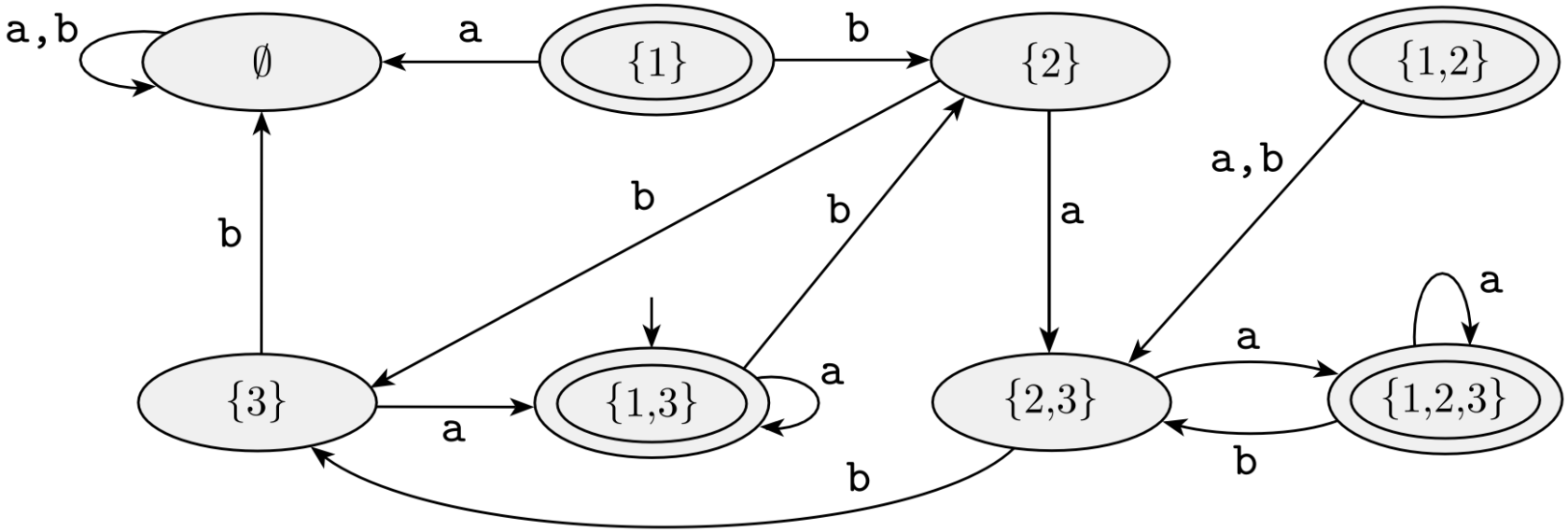
Example 13 (Converting an NFA to a DFA)

Convert the NFA below to a DFA:



Example (cont.)

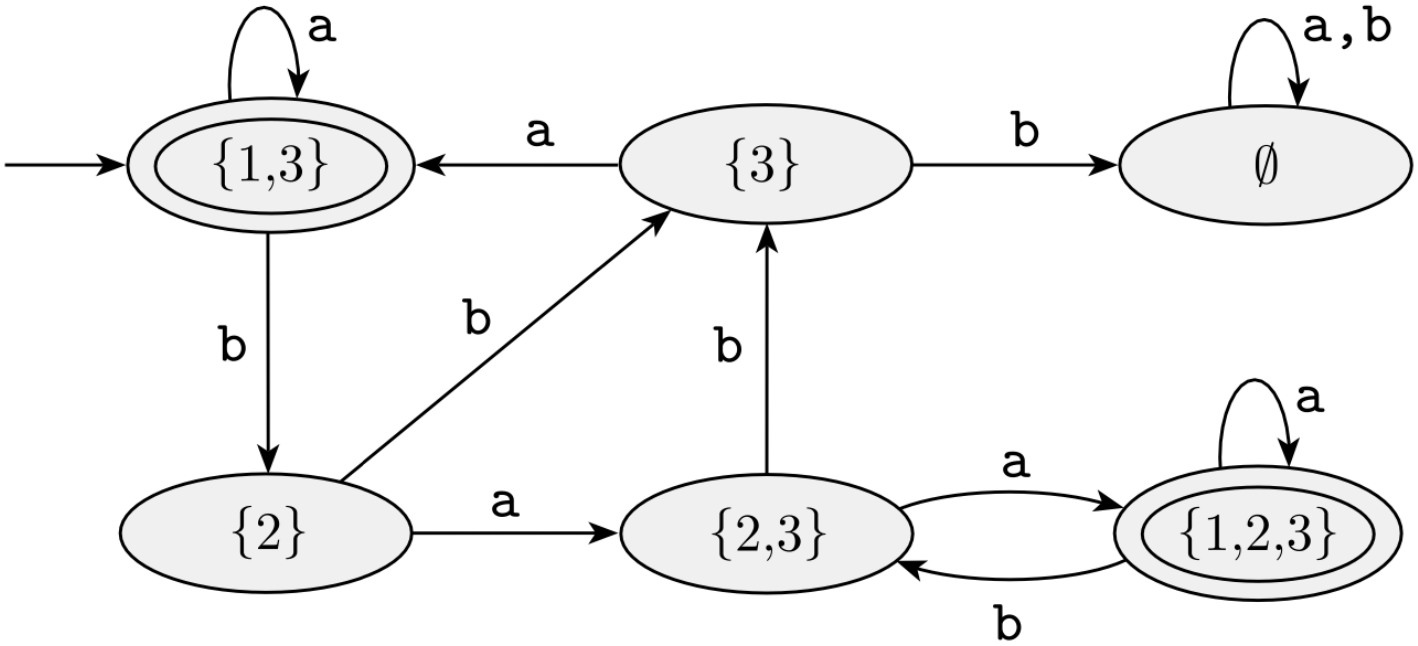
Solution:



The $\emptyset$ state performs the garbage collection.

Example (cont.)

We can simplify the DFA by eliminating states $\{1\}$ and $\{1,2\}$:



Closure under the Regular Operations: Union

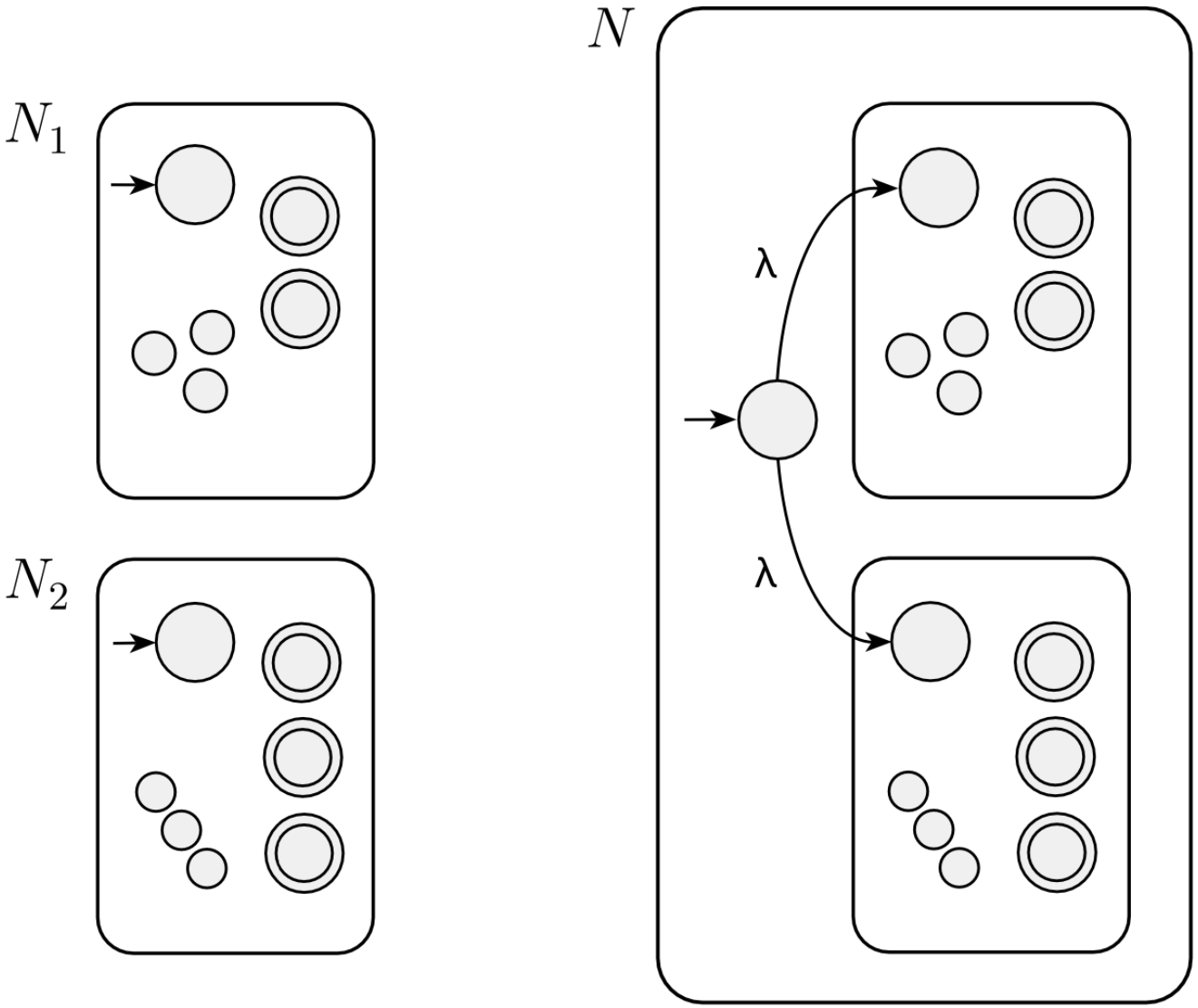
Theorem 14

The class of regular languages is closed under the union operation.

Proof idea.

Create an NFA N from N_1 and N_2 that recognize the languages A_1 and A_2 respectively. □

Closure under the Regular Operations: Union (cont.)



Closure under the Regular Operations: Union (cont.)

Proof.

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1,
and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.
Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$.

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.
2. q_0 is the start state of N.
3. $F = F_1 \cup F_2$.
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$:

$$\delta(q,a) = \begin{cases} \delta_1(q,a), & q \in Q_1 \\ \delta_2(q,a), & q \in Q_2 \\ \{q_1, q_2\}, & q = q_0 \text{ and } a = \varepsilon \\ \emptyset, & q = q_0 \text{ and } a \neq \varepsilon. \end{cases}$$

Closure under the Regular Operations: Concatenation

Theorem 15

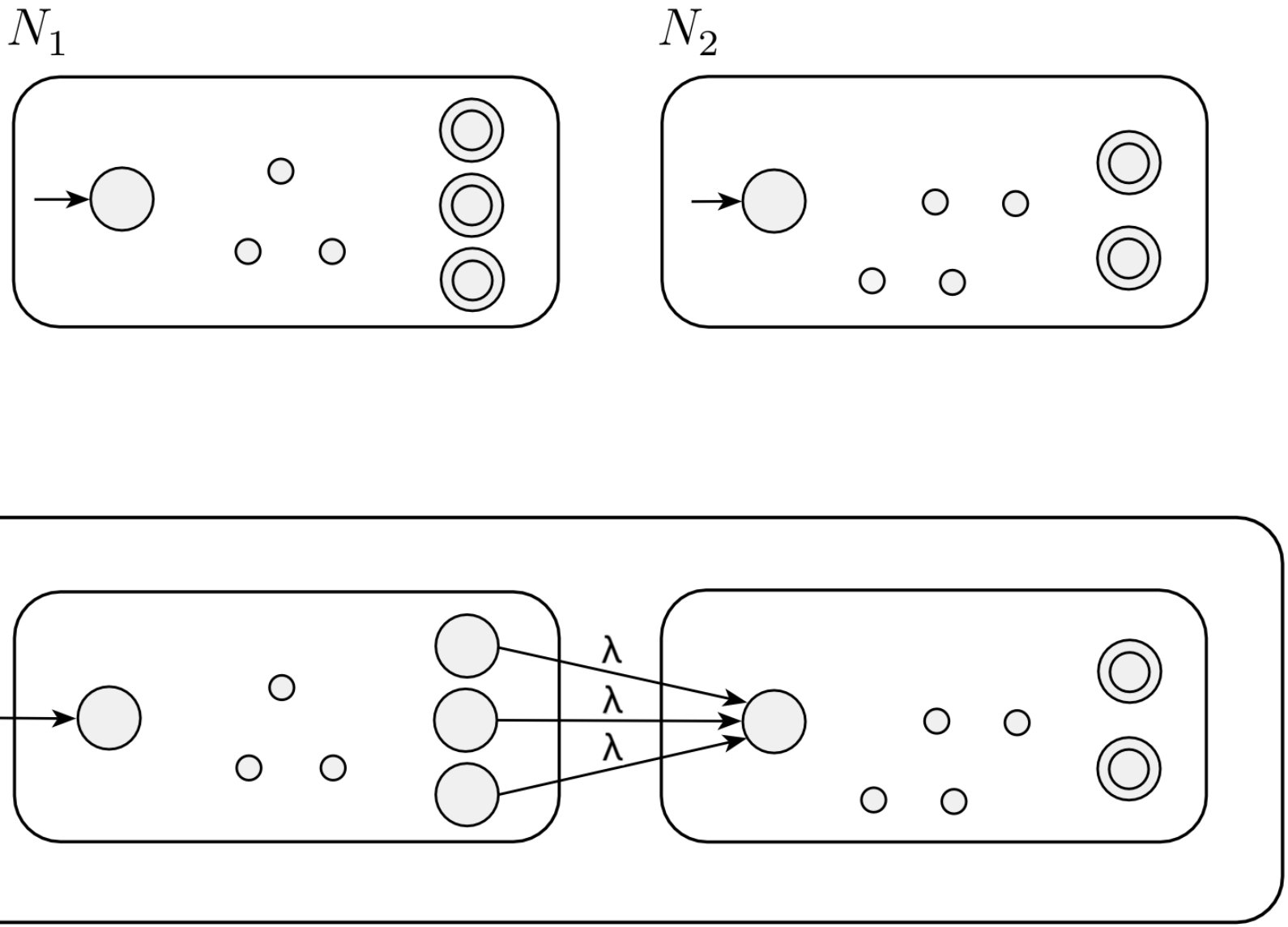
The class of regular languages is closed under concatenation.

Proof idea.

Nondeterministically guess where to make the split. □

(In the figure below, the originally terminal states in N_1 get connected to the start state of N_2.)

Closure under the Regular Operations: Concatenation



Closure under the Regular Operations: Concatenation

Proof.

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1,
and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.
Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$.

1. $Q = Q_1 \cup Q_2$.
2. q_1 is the start state of N as well.
3. Accept states for N are those of N_2 so F_2.
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$:

$$\delta(q,a) = \begin{cases} \delta_1(q,a), & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a), & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\}, & q \in F_1 \text{ and } a = \varepsilon \\ \delta_2(q,a), & q \in Q_2. \end{cases}$$

Closure under the Regular Operations: Star

Theorem 16

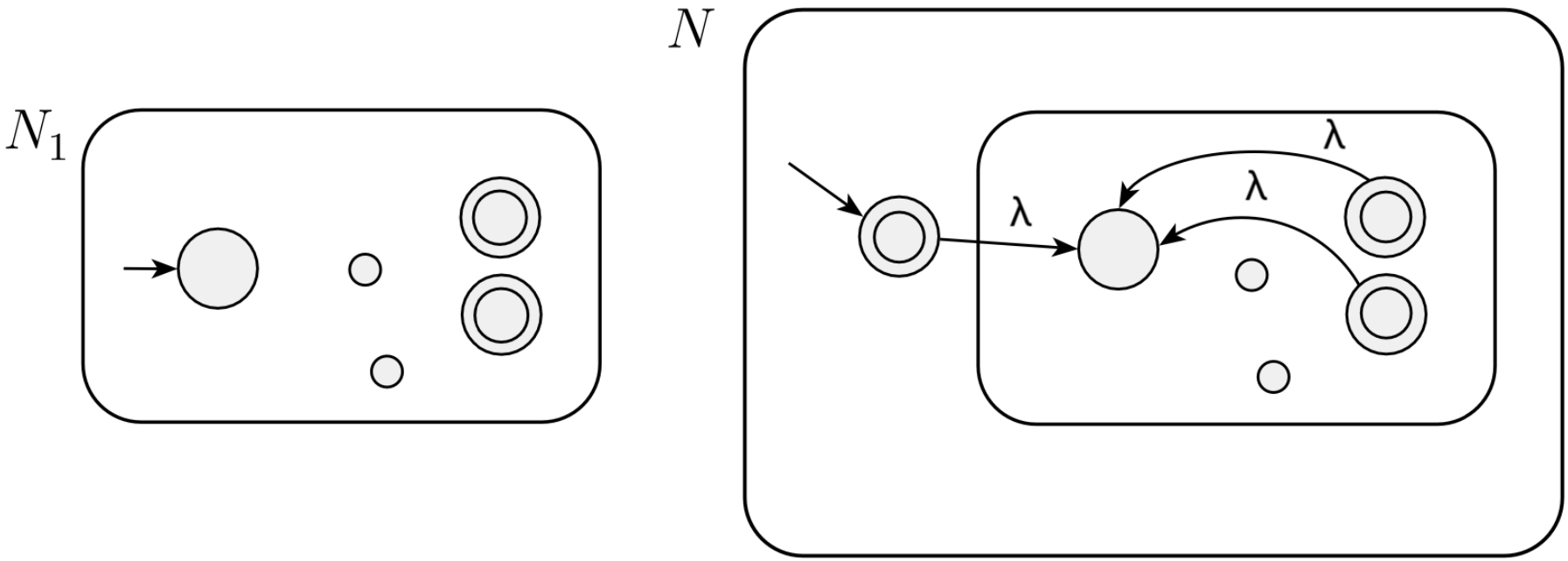
The class of regular languages is closed under the star operation.

Proof idea.

Construct an NFA N that accepts its input whenever it can be broken into several pieces and N_1 accepts each such piece. □

Note: It is a bad idea to make accepting the original starting state of N_1.

Closure under the Regular Operations: Star



Closure under the Regular Operations: Star

Proof.

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1.
Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^*.

1. $Q = q_0 \cup Q_1$.
2. q_0 is the new start state.
3. $F = \{q_0\} \cup F_1$.
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$:

$$\delta(q,a) = \begin{cases} \delta_1(q,a), & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a), & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\}, & q \in F_1 \text{ and } a = \varepsilon \\ \{q_1\}, & q = q_0 \text{ and } a = \varepsilon \\ \emptyset, & q = q_0 \text{ and } a \neq \varepsilon. \end{cases}$$

Table of Contents

Introduction to Regular Expressions

In arithmetic, we can use the operations $+$ and $\times$ to build up expressions such as $(5 + 3) \times 4$.

Similarly, we can use the regular operations to build up expressions describing languages, which are called **regular expressions**.
An example is: $(0 + 1)0^*$.

The value of this expression is the language.

- $(0 + 1)$ means $(\{0\} \cup \{1\})$ which is $\{0, 1\}$
- 0^* means $\{0\}^*$
- $(0 + 1)0^*$ is shorthand for $(0 + 1) \circ 0^*$ (i.e., usually drop/imply the concatenation symbol)

Examples of Regular Expressions

Example 17 (Simple Regular Expressions)

- $(0 + 1)^*$: all possible binary strings (i.e., strings of 0s and 1s).
- Let $\Sigma = \{0, 1\}$. Then Σ is a shorthand for the regular expression $(0 + 1)$, i.e., Σ is all the possible strings of one character (length 1).
- $\Sigma^* 1$: all strings that end with a 1.
- $(0\Sigma^*) + (\Sigma^* 1)$: strings that start with a 0 or end with a 1.

Regular Expressions

Definition 18 (Regular expression)

We say that R is a **regular expression** if R is

1. a for some $a \in \Sigma$,
2. ε,
3. $\emptyset$,
4. $(R_1 + R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

Regular Expressions

Some remarks on the definition:

- $L(R)$ is the language of a regular expression R.
- In (1) and (2), a and ε represent the languages $\{a\}$ and $\{\varepsilon\}$ respectively.
- (2) is different from (3):
  - (2) is a language with one string: $\{\varepsilon\}$
  - (3) is the empty language: $\{\}$
- When there are no parentheses, evaluate in the precedence order: star, concatenation, union.
- $R^+ = RR^*$. In other words, $R^+ + \varepsilon = R^*$.

Identities for Regular Expressions

- $R + \emptyset = R$.
- $R \circ \varepsilon = R$ (attaching the empty string to any string, will not change the string).
- In principle
  - $R + \varepsilon \neq R$.
    For example, if $R = 0$, then $L(R) = \{0\}$ but $L(R + \varepsilon) = \{\varepsilon, 0\}$.
  - $R \circ \emptyset \neq R$.
    For example, if $R = 0$, then $L(R) = \{0\}$ but $L(R \circ \emptyset) = L(\emptyset) = \emptyset$.
    (Concatenating the empty set to any set yields the empty set.)

Equivalence with Finite Automata

Theorem 19

A language is regular if and only if some regular expression describes it.

Lemma 20 ('$\Leftarrow$' direction of Thm 19)

If a language is described by a regular expression, then it is regular.

Proof idea.

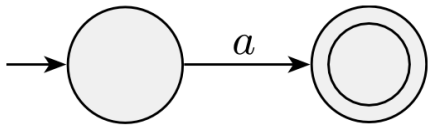
Show how to convert R into an NFA recognizing A. □

Equivalence with Finite Automata (cont.)

Proof.

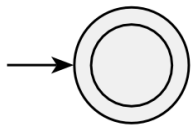
Let R be a regular expression. Let's convert R into an NFA N. We consider the six cases in the formal definition of regular expressions.

1. $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$, and the following NFA recognizes $L(R)$.



   Formally, $N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$, where $\delta(q_1, a) = \{q_2\}$ and $\delta(r, b) = \emptyset$ for $r \neq q_1$ or $b \neq a$.

2. $R = \varepsilon$. Then $L(R) = \{\varepsilon\}$, and the following NFA recognizes $L(R)$.

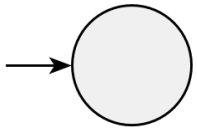


   ($\delta(r, b) = \emptyset$ for any r and b)

Equivalence with Finite Automata (cont.)

Proof (cont.)

3. $R = \emptyset$. Then $L(R) = \{\emptyset\}$, and the following NFA recognizes $L(R)$.



   ($\delta(r, b) = \emptyset$ for any r and b)

4. $R = R_1 + R_2$.
5. $R = R_1 \circ R_2$.
6. $R = R_1^*$.

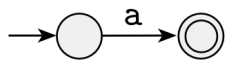
For the last three cases, we use the fact that regular languages are closed under the regular operations (and we use the constructions given in the corresponding proofs).
In other words, we construct the NFA for R from the NFAs for R_1 and R_2. □

Example

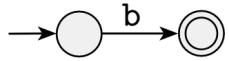
Example 21

Convert the regular expression $(ab + a)^*$ to an NFA in a sequence of stages.

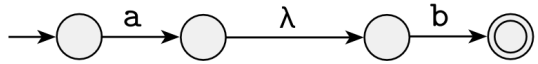
a



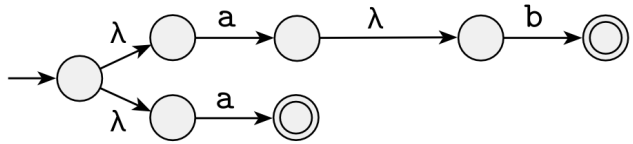
b



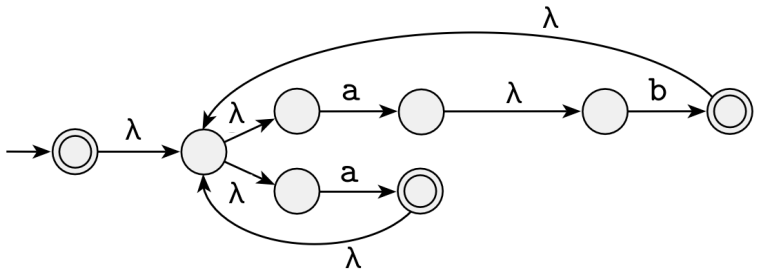
ab



ab + a



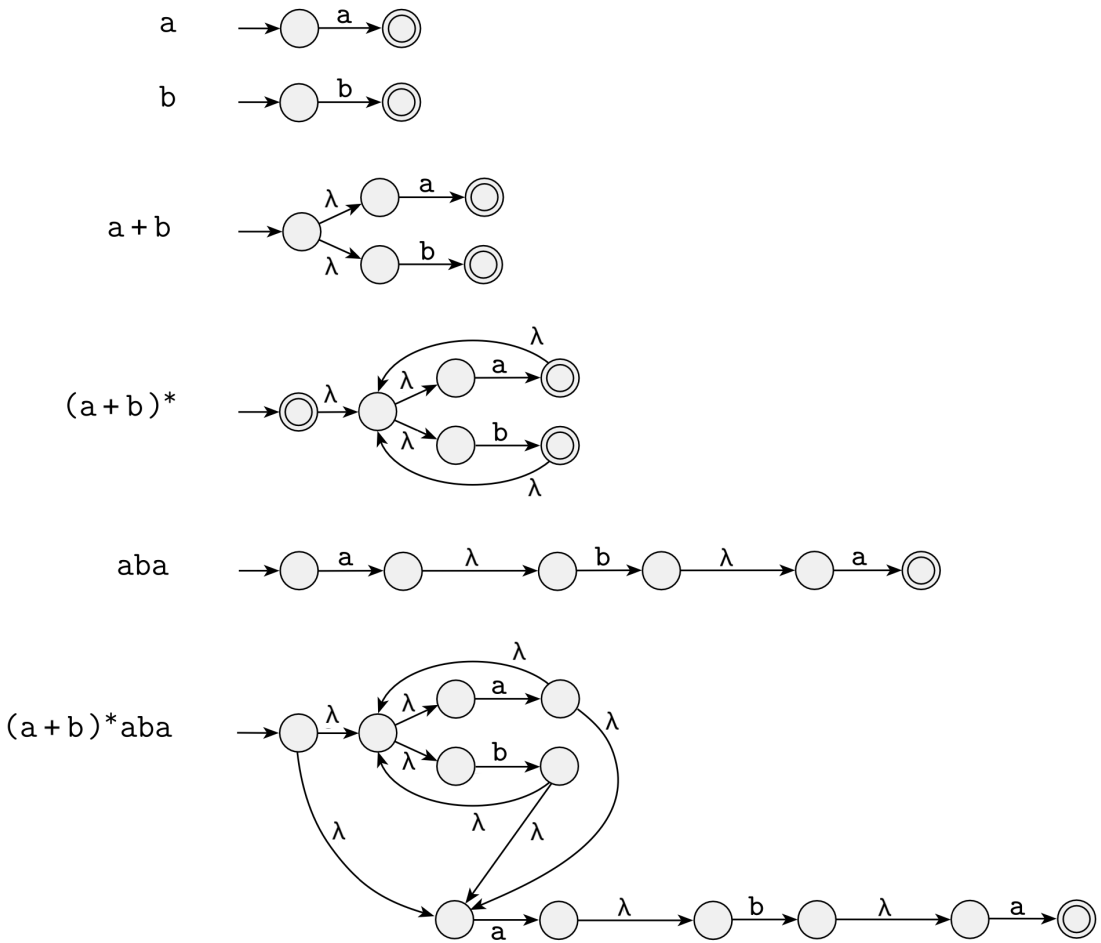
(ab + a)*



Example

Example 22

Convert the regular expression $(a + b)^* aba$ to an NFA.



Generalized Nondeterministic Finite Automata

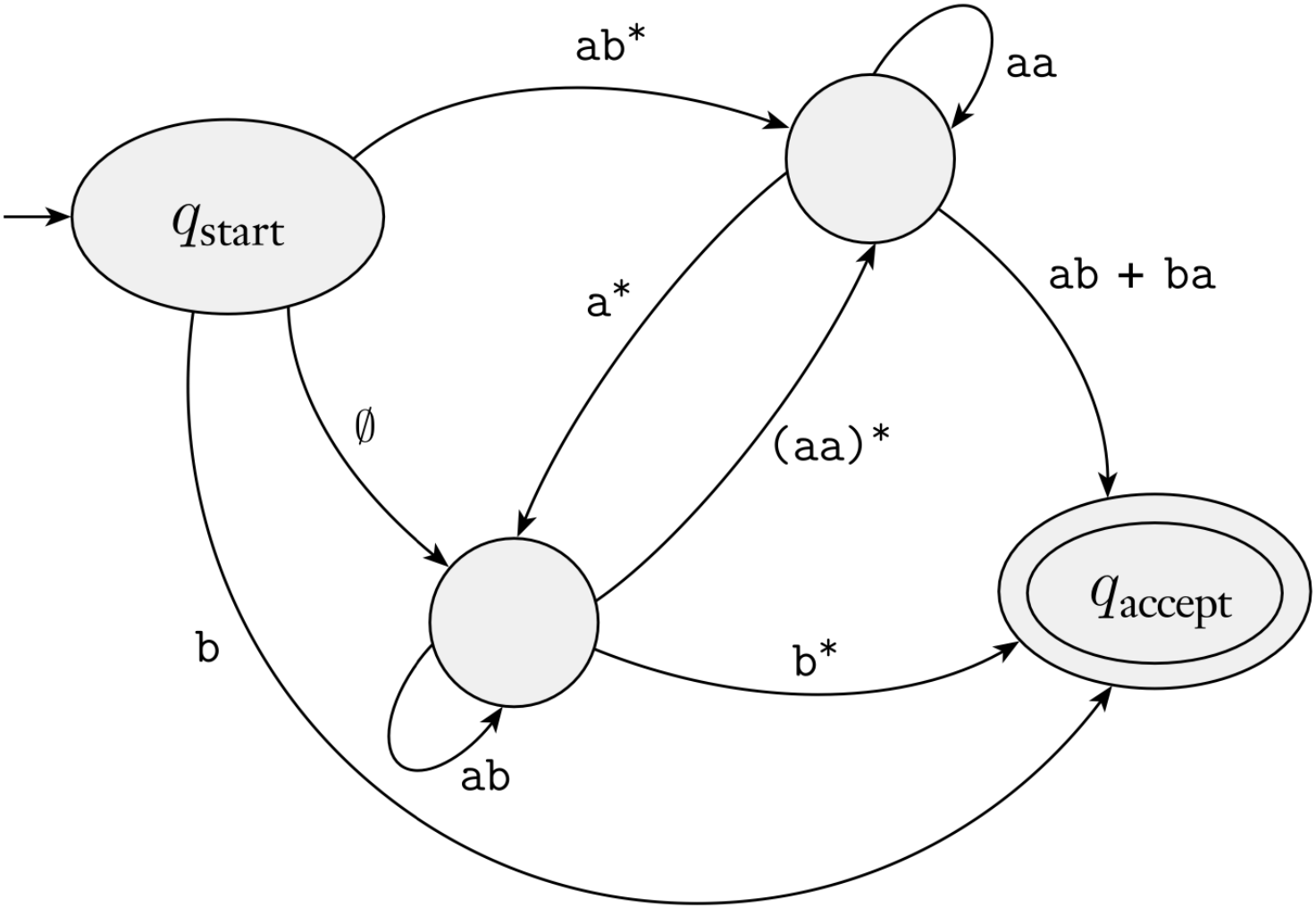
Definition

A Generalized Nondeterministic Finite Automaton (GNFA) is an NFA wherein the transition arrows may have any regular expression as labels (instead of only members of the alphabet or ε).

- The GNFA reads blocks of symbols from the input, not necessarily just one symbol at a time (as in an ordinary NFA).
- The GNFA moves along a transition arrow connecting two states by reading a block of symbols from the input; that string is a string that corresponds to the regular expression on that arrow.
- A GNFA, being nondeterministic, may have several different ways to process the same input string.

Example

Example 23 (A generalized nondeterministic finite automaton)



Conditions for GNFAs

For convenience, we require that GNFAs always have a special form that meets the following conditions:

(i) The start state has transition arrows going to every other state, but no arrows coming in from any other state.

(ii) There is only a **single accept state**, and it has:

- arrows coming in from every other state
- no arrows going to any other state
- $q_{\text{accept}} \neq q_{\text{start}}$

(iii) Except for q_{start} and q_{accept}, for every state:

- one arrow goes from it to every other state,
- one arrow goes from it to itself.

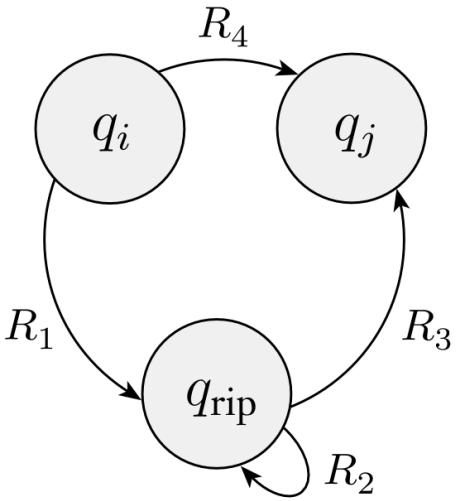
Conversion of a DFA into a GNFA

- Add a new start state with an ε arrow to the old start state.
- Add a new accept state with ε arrows coming in from every old accept state.
- Arrows with multiple labels are replaced each with a single arrow whose label is the union of the previous labels.
- Add arrows labeled $\emptyset$ between states that had no arrows.
  Note: $\emptyset$ does not recognize any string; not even ε. $\Rightarrow$ Such arrows can never be used.

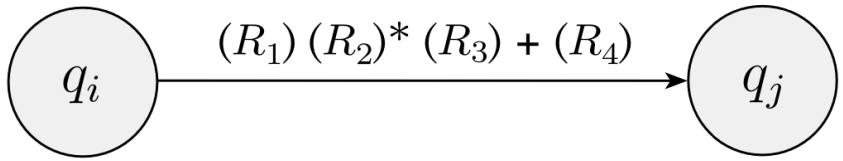
Constructing an equivalent GNFA with one fewer state starting with $k > 2$ states

Select a state, other than start or accept, rip it out of the machine, repair the remainder so that the same language is still recognized. Call the removed state q_{rip}. (This state exists because $k > 2$.)

New labels on arrows should compensate for the absence of q_{rip} by adding back the lost computations.



before



after

Constructing an equivalent GNFA with one fewer state starting with $k > 2$ states (cont.)

So, ultimately, we can construct a GNFA with just two states: starting and accept.
The label on the arrow connecting these two states is the regular expression that the original GNFA was accepting.

Generalized nondeterministic finite automaton (GNFA)

Definition

A **generalized nondeterministic finite automaton** (GNFA) is a 5-tuple, $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$, where:

1. Q is the finite set of states,
2. Σ is the input alphabet,
3. $\delta : (Q \setminus \{q_{\text{accept}}\}) \times (Q \setminus \{q_{\text{start}}\}) \to \mathcal{R}$ is the transition function, ($\mathcal{R}$ is the collection of all regular expressions over the alphabet Σ)
4. q_{start} is the start state, and
5. q_{accept} is the accept state.

Computation of a GNFA

Definition

A GNFA **accepts** a string w in Σ^* if $w = w_1 w_2 \cdots w_k$, where each w_i is in Σ^* and a sequence of states $q_0, q_1, \ldots, q_k$ exists such that

1. $q_0 = q_{\text{start}}$ is the start state,
2. $q_k = q_{\text{accept}}$ is the accept state, and
3. for each i, we have $w_i \in L(R_i)$, where $R_i = \delta(q_{i-1}, q_i)$ (in other words, R_i is the expression on the arrow from q_{i-1} to q_i).

Regular Languages

Now let's turn to the other direction of the proof of Theorem 19.

Lemma 24

If a language is regular, then it is described by a regular expression.

Proof idea

Let A be a regular language. Because A is regular, a DFA accepts it.
We convert DFAs into equivalent regular expressions.
First we convert DFAs into GNFAs, and then GNFAs into regular expressions.

Regular Languages

Proof.

Let M be the DFA for language A.
Create a GNFA G starting from M (as discussed earlier).
Use the procedure CONVERT(G), which

- takes as input a GNFA G, and
- outputs the equivalent regular expression.

Regular Languages

Proof (cont.)

CONVERT(G):

1. Let k be the number of states of G.
2. If $k = 2$, then G has a start state and an accept state, and a single arrow connecting them, labeled with a regular expression R.
   Return the expression R.
3. If $k > 2$, select any state $q_{\text{rip}} \in Q \setminus \{q_{\text{start}}, q_{\text{accept}}\}$.
   Let G' be the GNFA $(Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$, where $Q' = Q \setminus \{q_{\text{rip}}\}$, and for any $q_i \in Q' \setminus \{q_{\text{accept}}\}$ and any $q_j \in Q' \setminus \{q_{\text{start}}\}$, let $\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) + (R_4)$, for $R_1 = \delta(q_i, q_{\text{rip}})$, $R_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$, $R_3 = \delta(q_{\text{rip}}, q_j)$, and $R_4 = \delta(q_i, q_j)$.
4. Return CONVERT(G').

Regular Languages

Claim 72

For any GNFA G, CONVERT(G) is equivalent to G.

Proof

By induction on k, the number of states of the GNFA.
Basis ($k = 2$): G can only have a single arrow connecting the start state with the accept state.
The regular expression label $\mathcal{R}$ on this arrow describes all strings that allow G to get to the accept state.
Hence this expression is equivalent to G.

Regular Languages

Proof (cont.)

Induction step: Assume that the claim is true for $k-1$ states and use this assumption to prove that the claim is true for k states.
First show that G (k states) and G' ($k-1$ states) recognize the same language.
Suppose that G accepts an input w.
In an accepting computation, G enters the sequence of states:
$\sigma = (q_{\text{start}}, q_1, q_2, q_3, \ldots, q_{\text{accept}})$.

- If q_{rip} **does not** appear in σ, then G' also accepts w.
  The reason is that each of the new regular expressions labeling the arrows of G' contains the old regular expression as part of a union.
- If q_{rip} **does** appear in σ, removing each run of consecutive q_{rip} states forms an accepting computation for G'. The states q_i and q_j bracketing a run have a new regular expression on the arrow between them that describes all strings taking q_i to q_j via q_{rip} on G. So G' accepts w.

Regular Languages

Proof (cont.)

Conversely, suppose that G' accepts an input w.
An arrow between any two states q_i and q_j in G' describes the strings taking q_i to q_j in G,

- either directly
- or via q_{rip}.

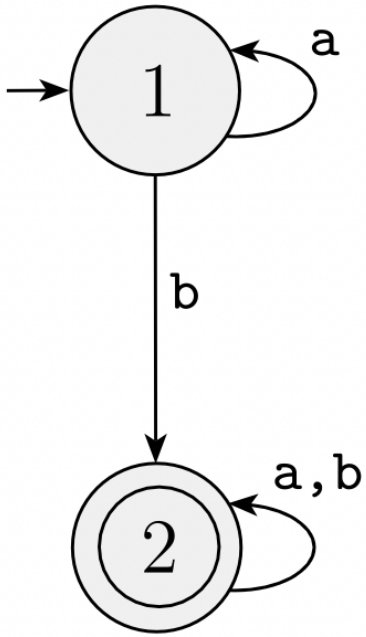
So G must also accept the input w. $\Rightarrow$ Thus G and G' are equivalent.

The induction hypothesis states that when the algorithm calls itself recursively on input G', the result is a regular expression that is equivalent to G' (since G' has $k - 1$ states).
Hence this regular expression also is equivalent to G, and the algorithm is proved correct.
This concludes the proof of Claim 72, Lemma 24, and Theorem 19. □

Example

Example 25

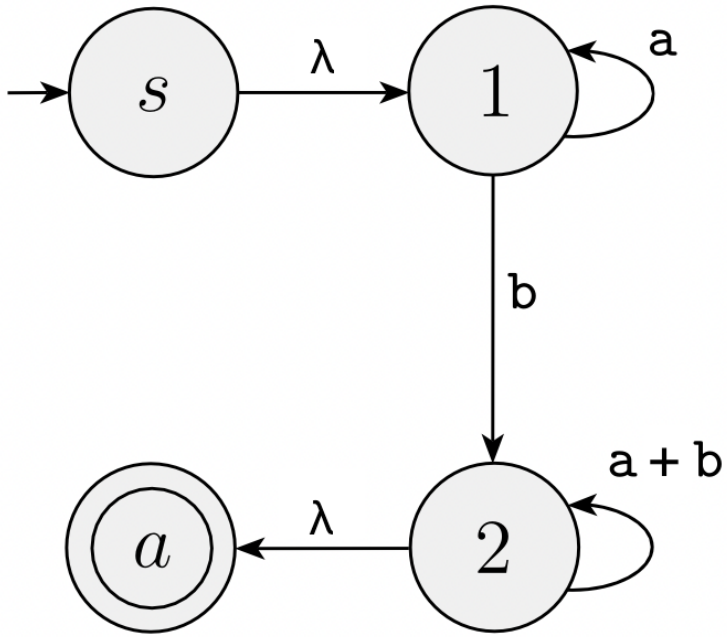
Convert the DFA below into a regular expression, using the preceding algorithm.



(a)

Example (cont.)

Add a new start state (s) and a new accept state (a):

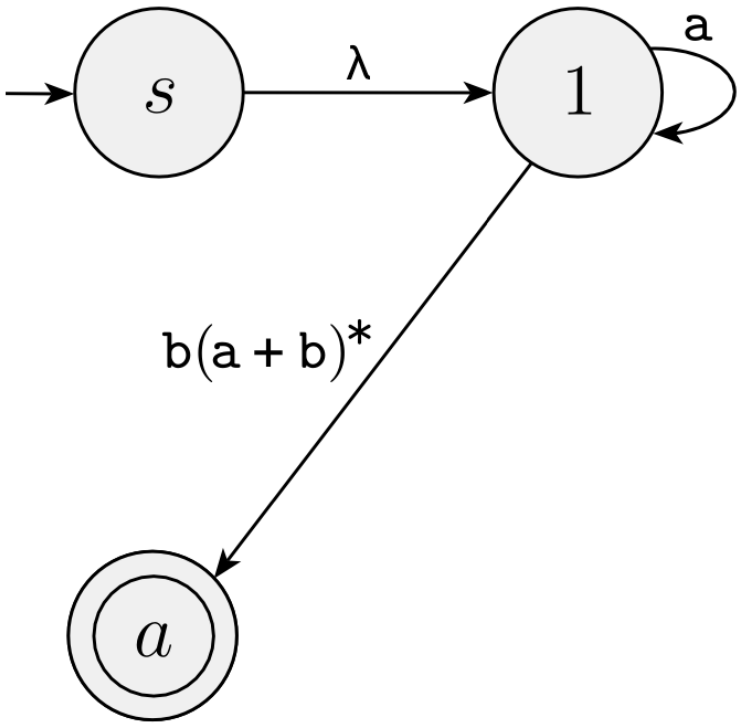


(b)

We avoid drawing arrows with labels $\emptyset$, even though the arrows are there.

Example (cont.)

Remove state 2:



(c)

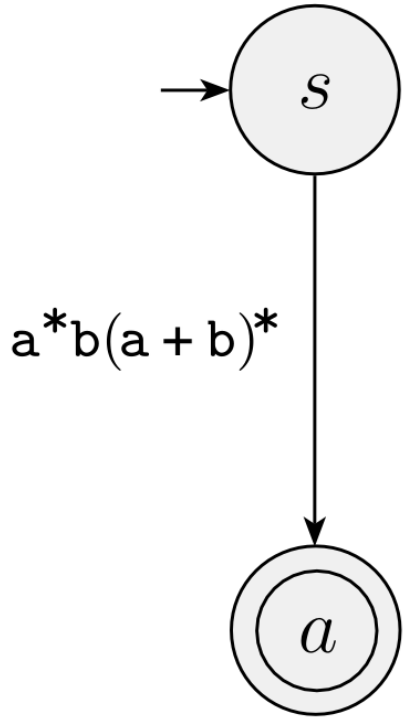
Note that the label $b(a+b)^*$ from 1 to a is consistent with CONVERT(G):

- $R_1 = b$
- $R_2 = a + b$
- $R_3 = \varepsilon$
- $R_4 = \emptyset$

so we get $(b)(a+b)^*(\varepsilon) = b(a+b)^*$

Example (cont.)

Remove state 1:



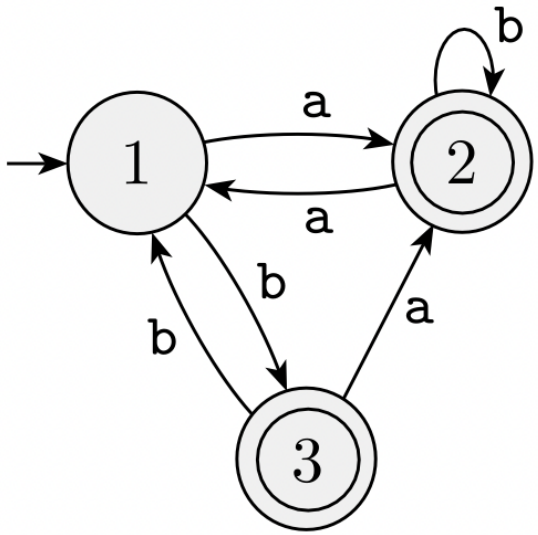
(d)

Follow the same procedure.
$\Rightarrow$ The label on this last arrow joining them is the regular expression that is equivalent to the original DFA.

Example

Example 26

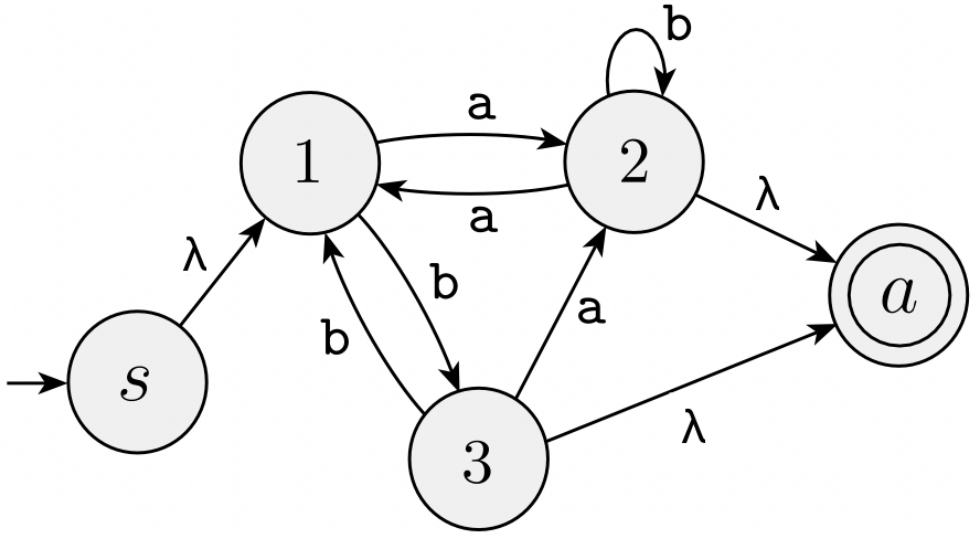
Convert the DFA below into a regular expression, using the preceding algorithm.



(a)

Example

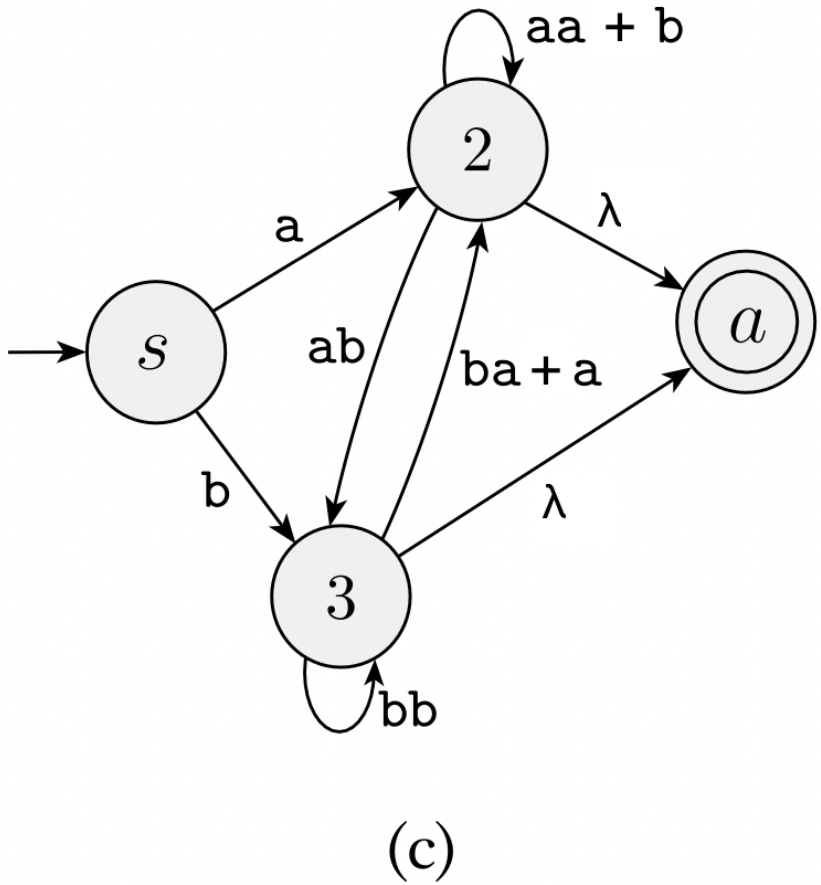
Introduce states s and a:



(b)

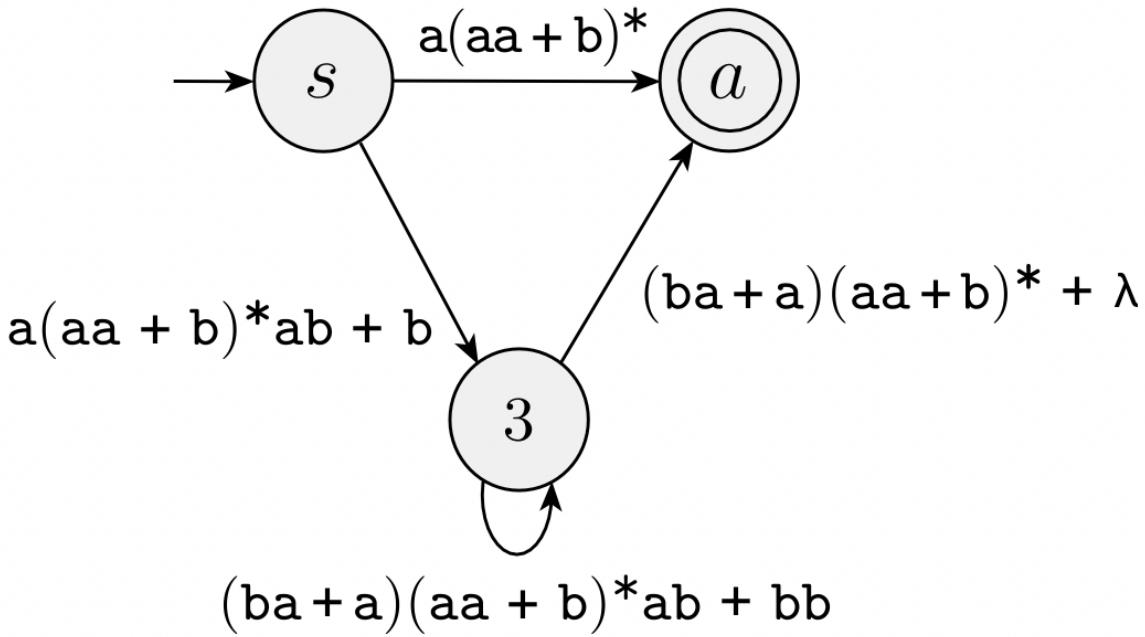
Example (cont.)

Remove state 1:



Example (cont.)

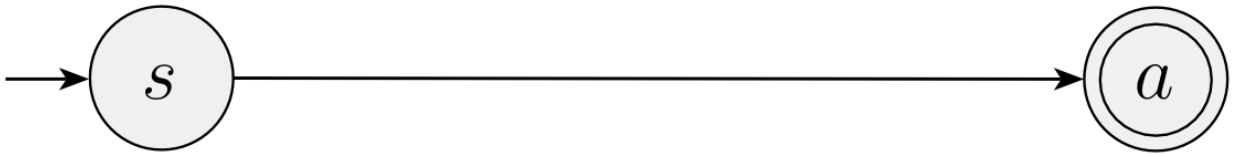
Remove state 2:



(d)

Example (cont.)

Remove state 3:



$$(a(aa+b)^*ab+b)((ba+a)(aa+b)^*ab+bb)^*((ba+a)(aa+b)^*+\lambda)+a(aa+b)^*$$

(e)

Table of Contents

Non-Regular Languages

Certain languages **cannot** be recognized by **any** finite automaton.

Consider the language $B = \{0^n 1^n | n \geq 0\}$.

Issue: The machine seems to need to remember how many 0s have been seen so far as it reads the input.
$\Rightarrow$ Because the number of 0s isn't limited
$\Rightarrow$ the machine will need to keep track of an unlimited number of possibilities.
$\Rightarrow$ Not possible with a finite number of states.

(Not a valid argument in general; see below.)

Non-Regular Languages

Consider

$$C = \{w | w \text{ has an equal number of 0s and 1s}\},$$

and

$$D = \{w | w \text{ has an equal number of occurrences of 01 and 10 as substrings}\}.$$

At first glance, a recognizing machine appears to need to count in each case, and therefore neither language appears to be regular.
As expected, C is not regular, but surprisingly D is regular!

We need a formal technique to be able to prove non-regularity.

Pumping Lemma for Regular Languages

All regular languages have a special property.

$\Rightarrow$ If a language does not have this property, then it is guaranteed that it is not regular.

Property:

All strings in the language can be "pumped" if they are at least as long as a certain special value, called the **pumping length**.

(That means each such string contains a section that can be repeated any number of times with the resulting string remaining in the language.)

Pumping Lemma

Theorem 27 (Pumping Lemma)

If A is a regular language, then there is a number p (the pumping length) where, if s is any string in A of length at least p, then s may be divided into three pieces,

$$s = xyz,$$

satisfying the following conditions:

1. *for each $i \geq 0$, $xy^iz \in A$,*
2. *$|y| > 0$, and*
3. *$|xy| \leq p$.*

Recall:

- $|s|$: the length of string s,
- y^i: i copies of y are concatenated together (hence $y^0 = \varepsilon$).

Pumping Lemma

Remarks on the Pumping Lemma:

- Either x or z may be ε, but $y \neq \varepsilon$ (due to (2)).
- Theorem trivially true without condition (2).
- Condition (3) is an extra technical condition that might be useful when proving certain languages to be nonregular.

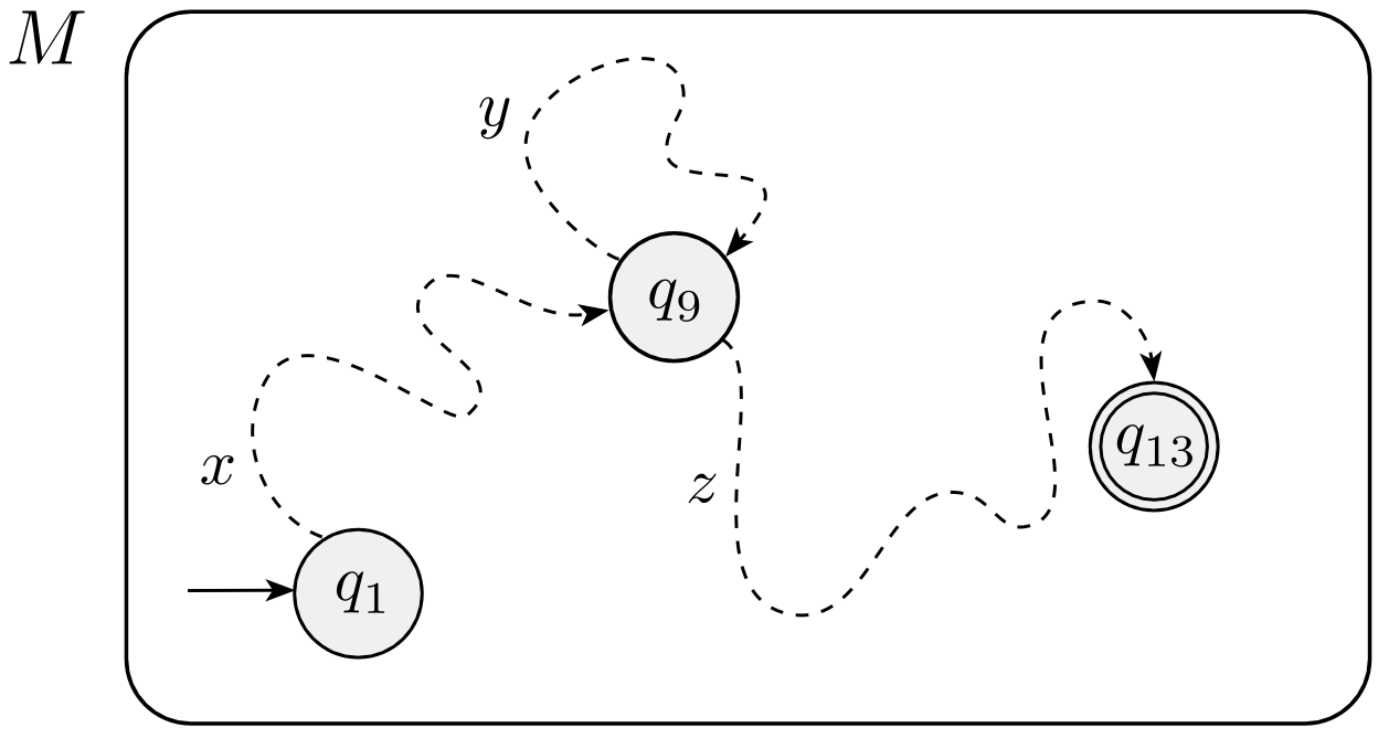
Pumping Lemma

Proof idea of the Pumping Lemma.

Let the pumping length p be the number of states of M.
Let n be the length of input s, such that $n \geq p$.
This means that s has n symbols $\Rightarrow$ dictates n transitions
$\Rightarrow$ If we write down the states visited by s, we will have a repetition. Here is why (pigeonhole principle):

$$s = \underset{q_1}{\uparrow} s_1 \underset{q_3}{\uparrow} s_2 \underset{q_{20}}{\uparrow} s_3 \underset{\textcircled{q_9}}{\uparrow} s_4 \underset{q_{17}}{\uparrow} s_5 \underset{\textcircled{q_9}}{\uparrow} s_6 \underset{q_6}{\uparrow} \quad \cdots \quad \underset{q_{35}}{\uparrow} s_n \underset{q_{13}}{\uparrow}$$

Pumping Lemma



It is $s = xyz$, so:

- M also accepts xy^iz, for any $i > 0$, as well as $xy^0z = xz$,
- $|y| > 0$, satisfying condition (2),
- The first $p + 1$ states in the sequence must contain a repetition (by pigeonhole principle). Therefore, $|xy| \leq p$.

Pumping Lemma

Proof.

Let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA recognizing A
and p be the number of states of M.
Let $s = s_1 s_2 \cdots s_n$ be a string in A of length n, where $n \geq p$.
Let $r_1, \ldots, r_{n+1}$ be the sequence of states that M enters while processing s, so

$$r_{i+1} = \delta(r_i, s_i), \text{ for } 1 \leq i \leq n.$$

This sequence has length $n + 1 \geq p + 1 \Rightarrow$ Among the first $p + 1$ states in the sequence, at least two must be the same (by pigeonhole principle). Call r_j the first of these ($j \neq l$) and r_l the second ($l \leq p + 1$).
Now let $x = s_1 \cdots s_{j-1}$, $y = s_j \cdots s_{l-1}$, and $z = s_l \cdots s_n$.
Since x takes M from r_1 to r_j, y takes M from r_j to r_j, and z takes M from r_j to r_{n+1}, which is an accept state, M must accept $xy^i z$ for $i \geq 0$.
We know that $j \neq l$, so $|y| > 0$; and $l \leq p + 1$, so $|xy| \leq p$.
Thus we have satisfied all conditions of the pumping lemma. □

Use of Pumping Lemma (to show non-regularity)

Steps:

1. Assume a language A is regular, in order to obtain a contradiction, with pumping length p such that all strings of length p or greater in A can be pumped.
2. Find a string $s \in A$ such that $|s| \geq p$ but s cannot be pumped, no matter how we split s into xyz.

Example

Example 28

Let $B = \{0^n 1^n | n \geq 0\}$.
Use the pumping lemma to prove that B is not regular.

Proof.

Assume to the contrary that B is regular. Let p be the pumping length given by the pumping lemma. Consider $s = 0^p 1^p$. (It is $s = xyz$, with $|y| > 0$.)
We consider 3 cases to show that this result is impossible.

- If string y consists only of 0s, then $xyyz$ has more 0s than 1s and thus $xyyz \notin B$, violating condition (1) of pumping lemma $\Rightarrow$ contradiction.
- If string y consists only of 1s $\Rightarrow$ contradiction as above.
- If string y consists of both 0s and 1s, then $xyyz$ may have the same number of 0s and 1s, but some 1s will appear before the 0s, so $xyyz \notin B$ again $\Rightarrow$ contradiction.

Example (cont.)

Note:
We could have avoided cases 2 and 3 by applying condition (3) of the pumping lemma that requires that $|xy| \leq p$, thus simplifying the above argument.

Example

Example 29

Let $C = \{w | w \text{ has an equal number of 0s and 1s}\}$.
Use the pumping lemma to prove that C is not regular.

Proof.

The proof is by contradiction.
Consider $s = 0^p 1^p$. By condition (3) of the pumping lemma $|xy| \leq p \Rightarrow$
So y contains only 0s $\Rightarrow$ So $xyyz \notin C \Rightarrow$ So s cannot be pumped.

The string selection is important. For example, $s' = (01)^p$ can be pumped, even taking condition (3) into account. Can you see how to pump it?
One way to do so is by setting: $x = \varepsilon$, $y = 01$, and $z = (01)^{p-1}$.
Then $xy^i z \in C$ for every i. □

Example (cont.)

Alternative proof.

Assume C is regular.
The language 0^*1^* is also regular.
Then $C \cap 0^*1^*$ also has to be regular (due to closure under intersection).
But $C \cap (0^*1^*) = B$, and we know that B is nonregular from Example 28. □

Example

Example 30

Let $F = \{ww | w \in \{0,1\}^*\}$.
Show that F is nonregular, using the pumping lemma.

Proof.

Assume to the contrary that F is regular.
Consider $s = 0^p 10^p 1$.
Because $s \in F$ and $|s| > p$, the pumping lemma guarantees that s can be split into three pieces, $s = xyz$, satisfying the three conditions of the lemma, hence $|xy| \leq p$. We show that this outcome is impossible.

With condition (3) the proof follows because y must consist only of 0s, and then $xyyz \in F$. □

Example

Example 31

Let $D = \{1^{n^2} | n \geq 0\}$. Use the pumping lemma to prove that D is not regular.

Proof.

The proof is by contradiction. Assume to the contrary that D is regular.
Let s be the string 1^{p^2}.
Now consider the two strings xyz and xy^2z.
The lengths of these strings differ by at most p (i.e., the length of y).
By condition (3) of the pumping lemma, $|xy| \leq p$ and thus $|y| \leq p$.
We have $|xyz| = p^2$ and so $|xyyz| \leq p^2 + p < p^2 + 2p + 1 = (p+1)^2$.
Moreover, condition (2) implies that $|y| > 0$ and so $|xy^2z| > p^2$.
Therefore, the length of $p^2 < |xyyz| < (p+1)^2$.
So we arrive at the contradiction $xyyz \notin D$ and conclude that D is not regular. □

Example

Example 32

Let $E = \{0^i 1^j | i > j\}$. Use the pumping lemma to prove that E is not regular.

Proof.

The proof is by contradiction. Assume that E is regular.
Let $s = 0^{p+1}1^p$. We reach a contradiction by **pumping down**.
The pumping lemma states that $xy^iz \in E$ even when $i = 0$, so the string $xy^0z = xz$ should also be in E.
But $|xy| \leq p$ so y only has 0s. Thus removing string y decreases the number of 0s in s, and xz cannot have more 0s than 1s. (Recall that s has just one more 0 than 1.) □