Theoretical Computer Science (Bridging Course)

Regular Languages

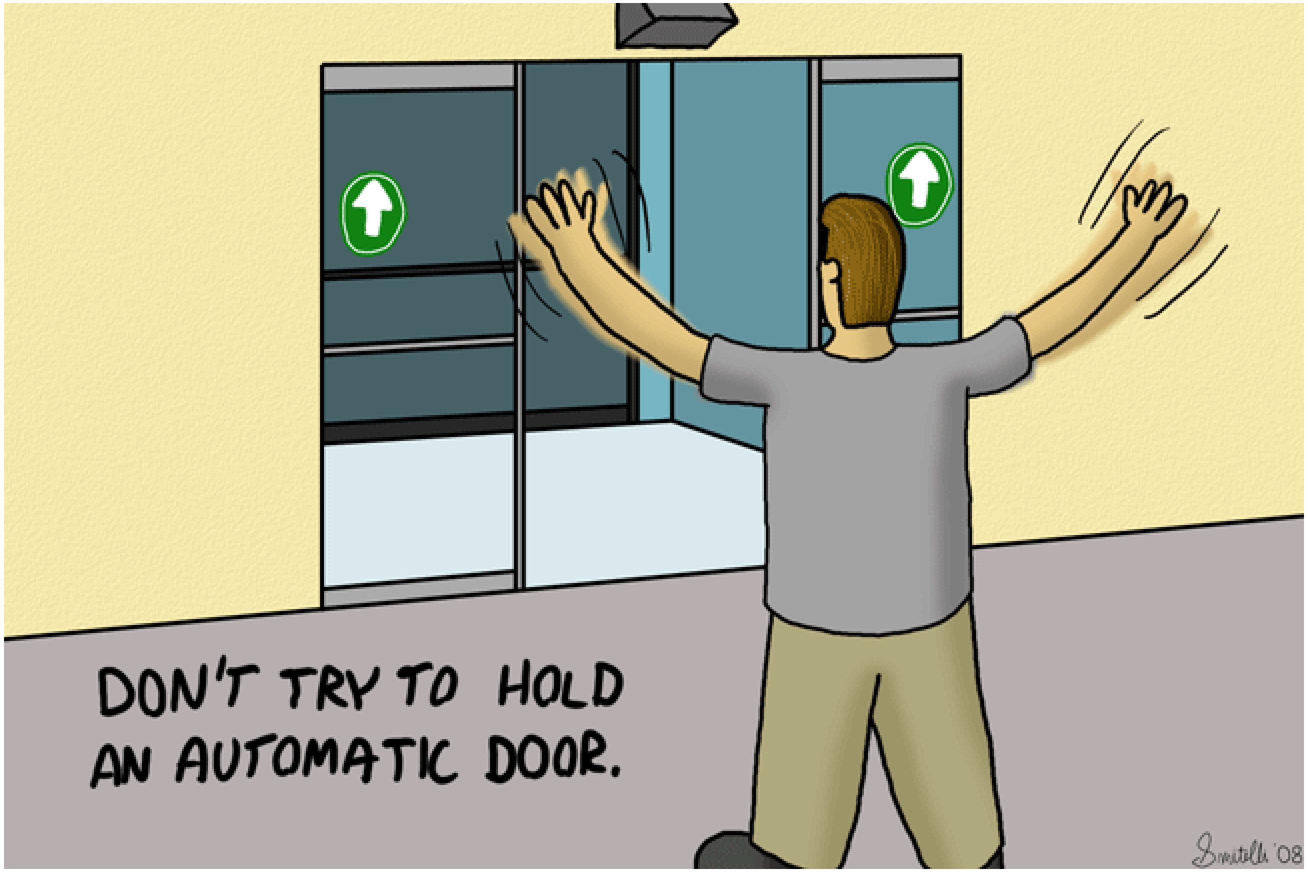
Gian Diego Tipaldi

Topics Covered

- Regular languages
- Deterministic finite automata
- Nondeterministic finite automata
- Closure
- Regular expressions
- Non-regular languages
- The pumping lemma

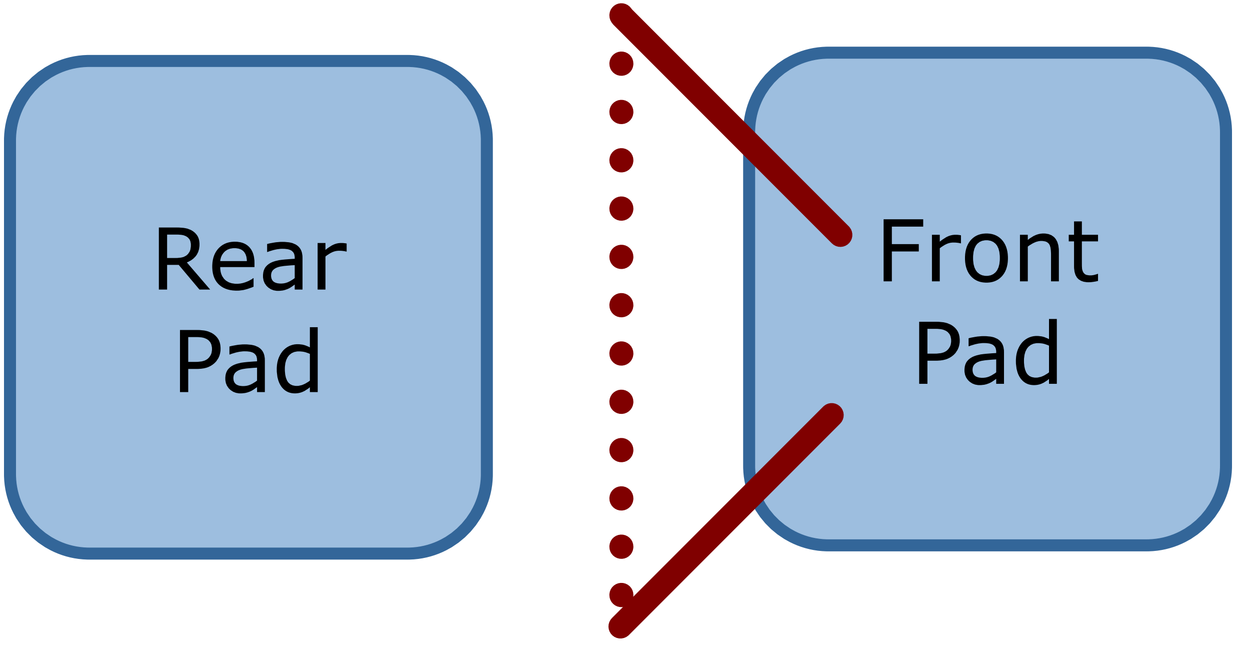
Finite Automata

- Supermarket door control



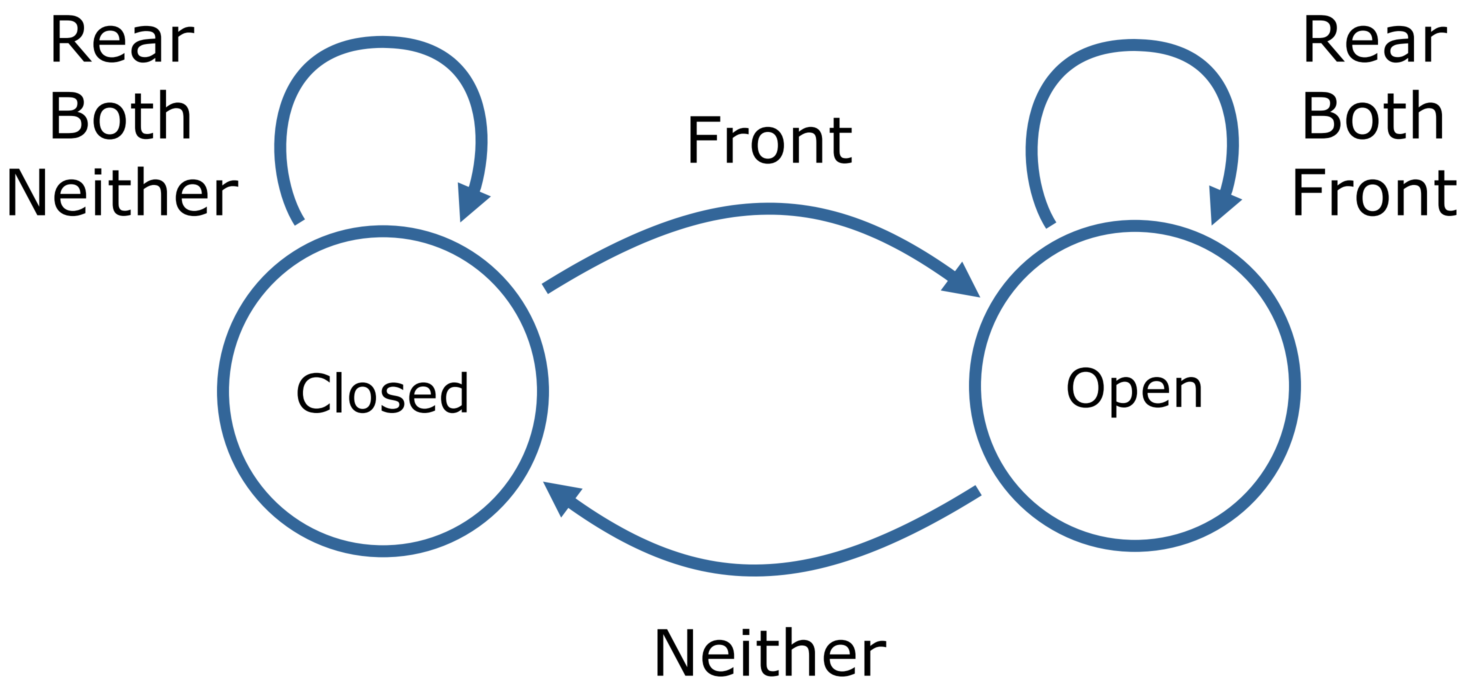
Finite Automata

- Supermarket door control



Finite Automata

- Supermarket door control



Finite Automata

- Supermarket door control

	Neither	Front	Rear	Both
Closed	Closed	Open	Closed	Closed
Open	Closed	Open	Open	Open

Finite Automata

- Supermarket door control

	Neither	Front	Rear	Both
Closed	Closed	Open	Closed	Closed
Open	Closed	Open	Open	Open

- Probabilistic counterparts exists
  - Markov chains
  - Bayesian networks

Finite Automata

- Supermarket door control

	Neither	Front	Rear	Both
Closed	Closed	Open	Closed	Closed
Open	Closed	Open	Open	Open

- Probabilistic counterparts exists
  - Markov chains
  - Bayesian networks

Finite Automata

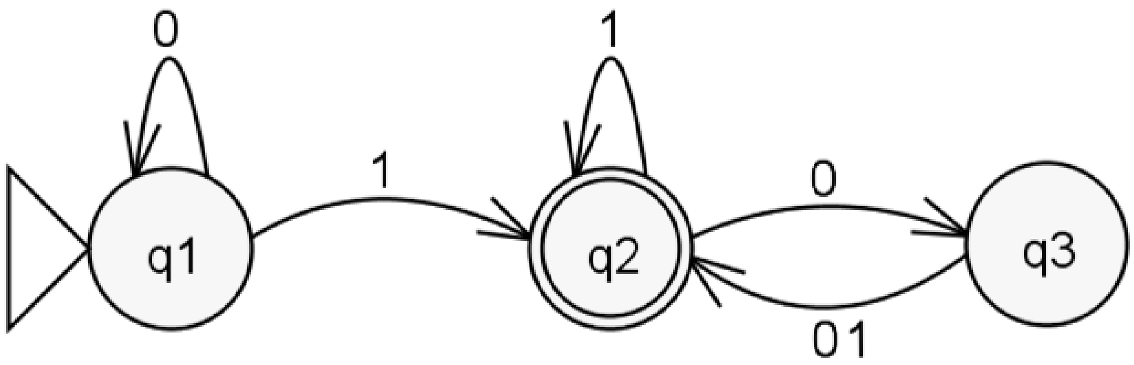
A **finite automaton** M is a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

where,

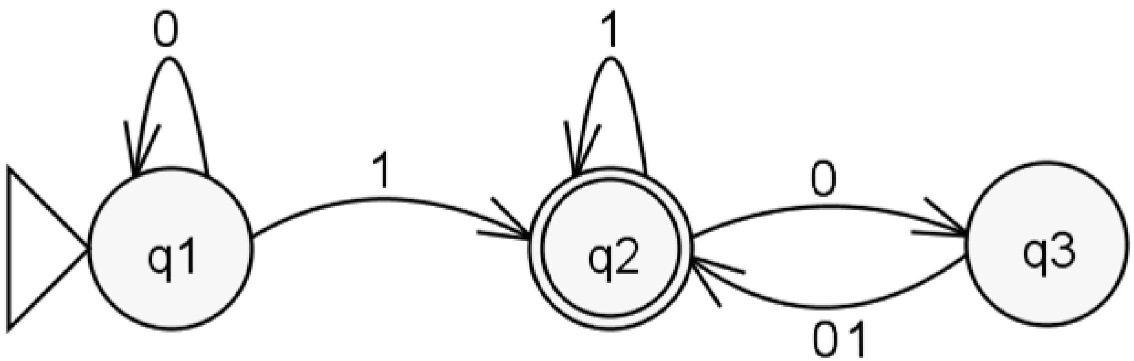
1. Q is a finite set called the **states**
2. Σ is a finite set called the **alphabet**
3. $\delta: Q \times \Sigma \rightarrow Q$ is the **transition function**
4. $q_0 \in Q$ is the **start state**
5. $F \subseteq Q$ is the set of **accept states** (also called **final states**)

Finite Automata



- States: $Q = \{q_1, q_2, q_3\}$
- Alphabet: $\Sigma = \{0, 1\}$
- Transition function: See edges
- Start state: q_1
- Final states: $F = \{q_2\}$

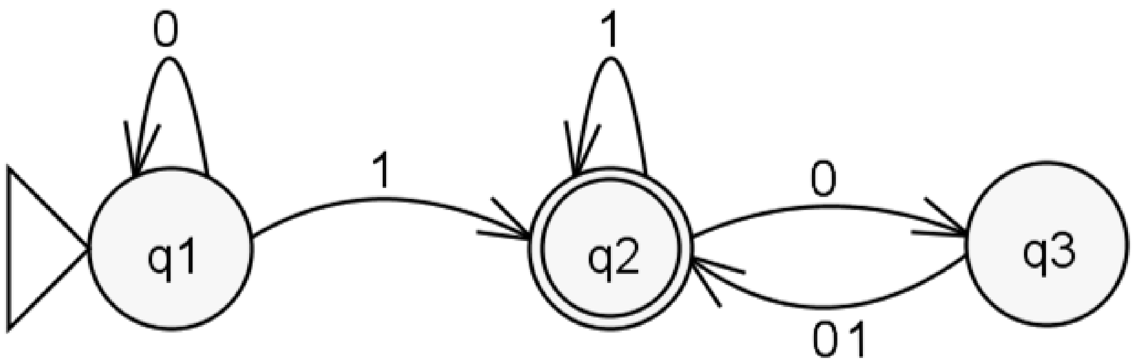
Finite Automata



Which kind of input is accepted?

- “aaaabbbbaaaa” ?
- “000000” ?
- An empty string?
- “1000111” ?

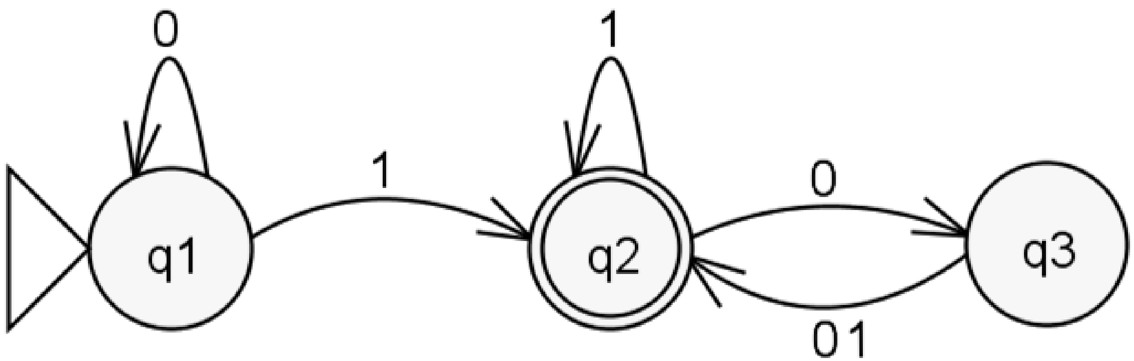
Finite Automata



Which language is accepted?

- “aaaabbbbaaaa” ?
- “000000” ?
- An empty string?
- “1000111” ?

Finite Automata



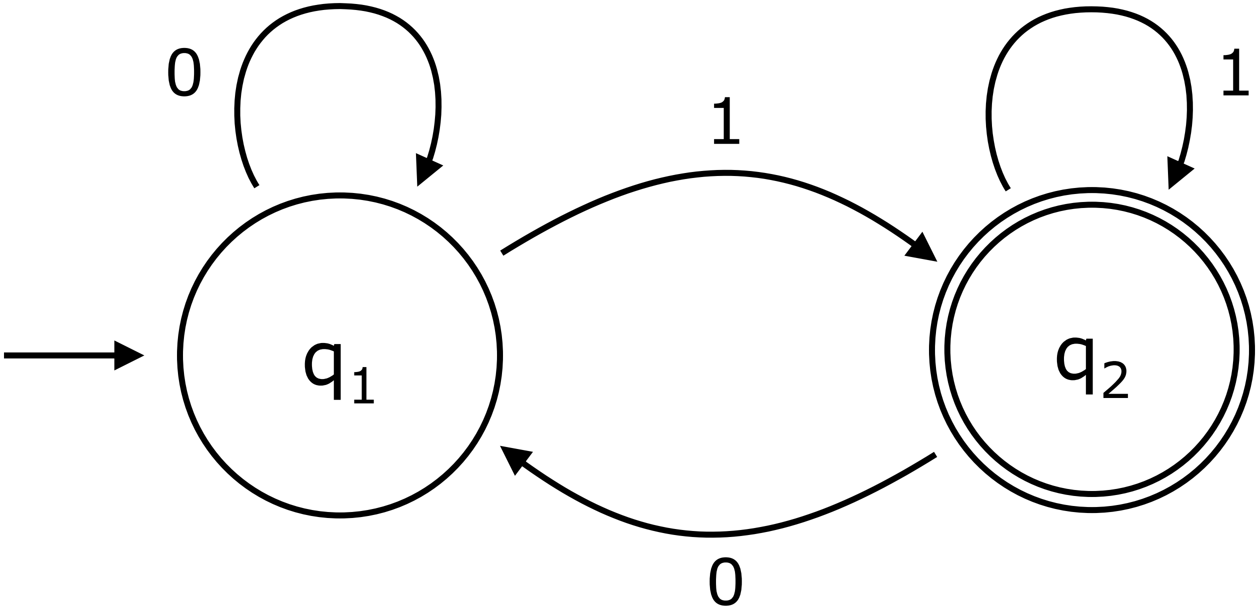
Which language is accepted?

$$A = \{w \mid w \text{ contains at least one 1 and an even number of 0s follows the last 1}\}$$

- "M recognizes A"
- "A is the language L(M)"

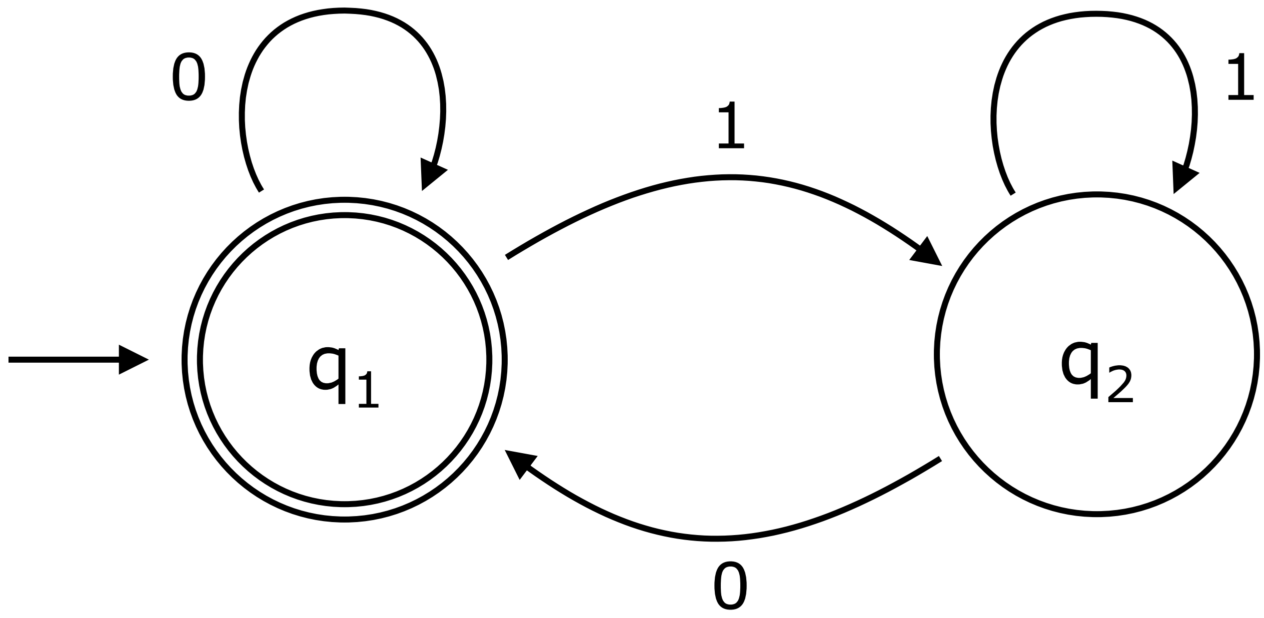
Finite Automata – Example

- Which language recognizes M?



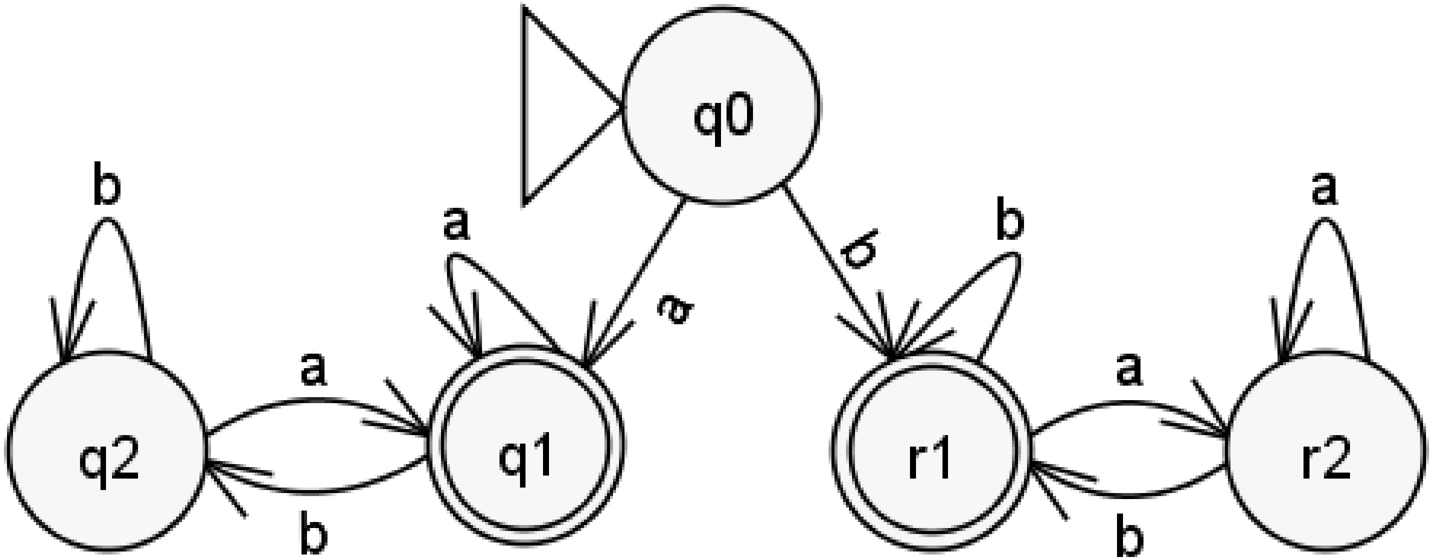
Finite Automata – Example

- And in this case?

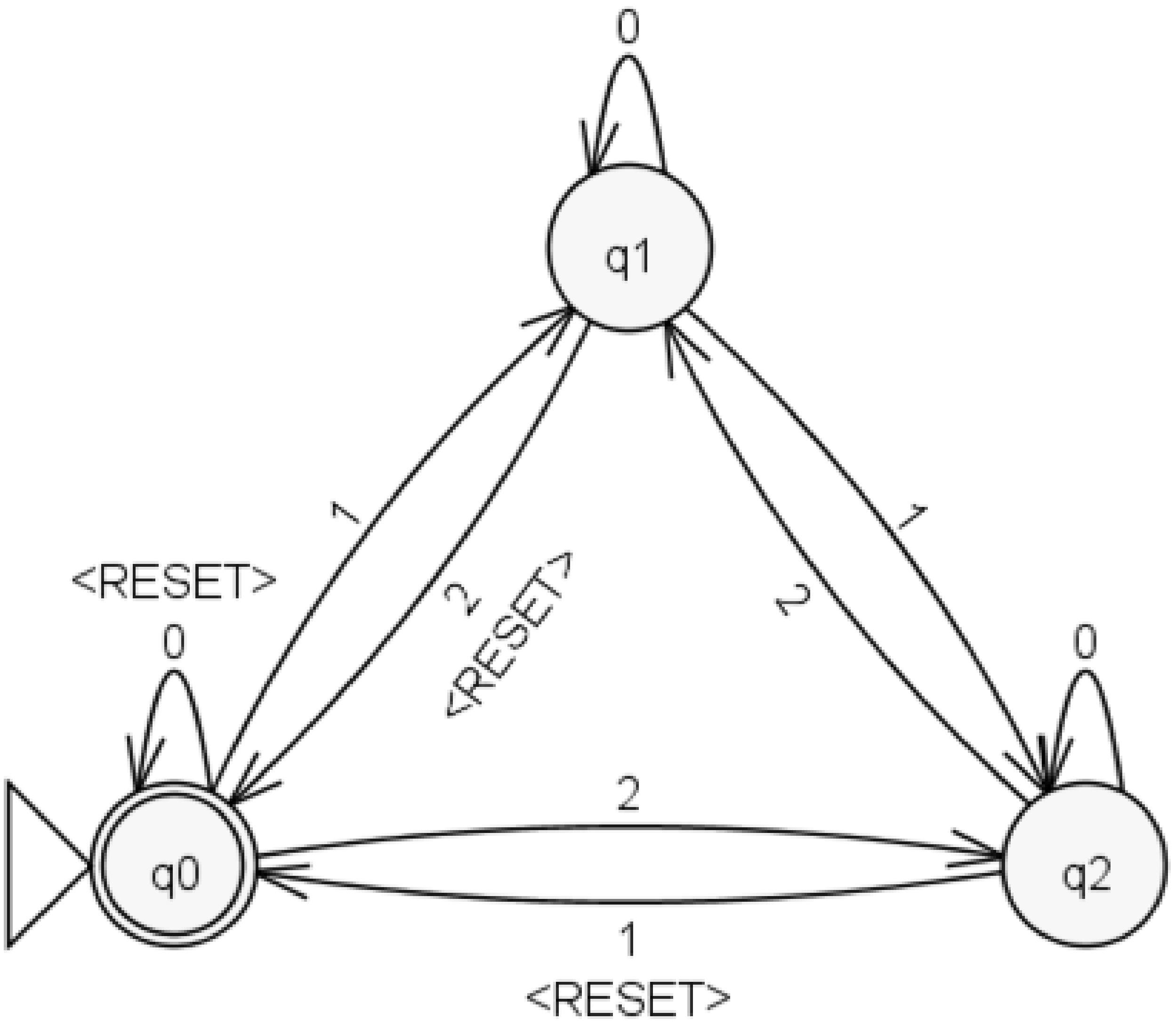


Finite Automata – Example

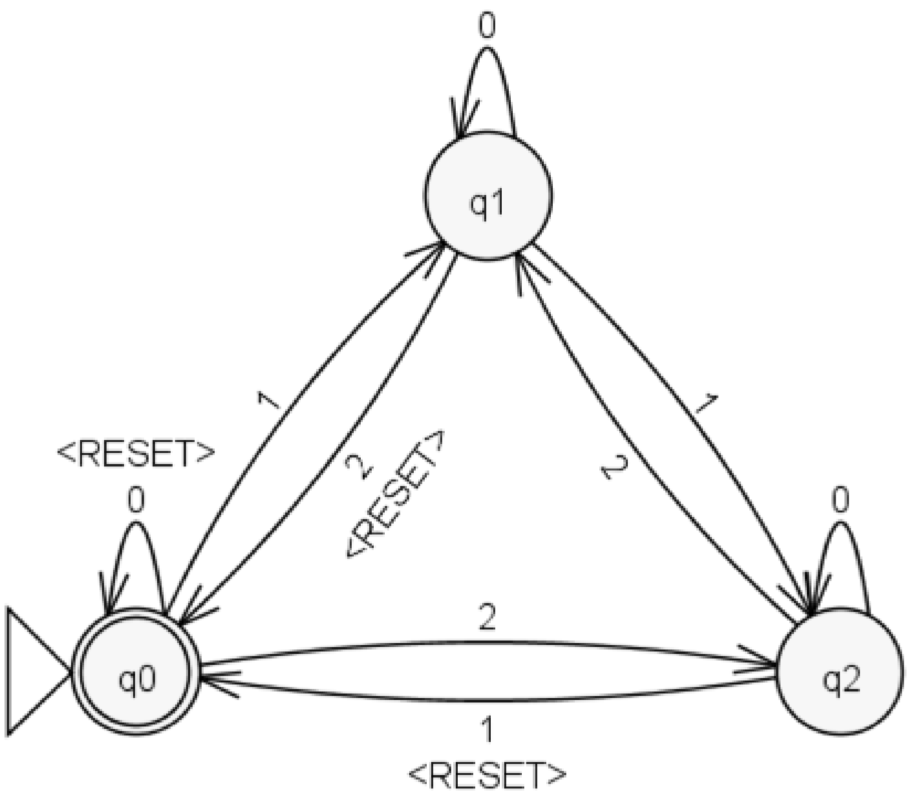
- What about this one?



Finite Automata – Example



Finite Automata – Example



- Sums all numerical symbols that reads, modulo 3.
- Resets the count, every time it receives <RESET>.
- Accepts, if the sum is a multiple of 3.

Definition of Computation

- Let M be a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$
- Let $w = w_1 \dots w_n$ be a string over Σ
  - M accepts w if a sequence of states $r_0, \dots r_n$ exists in Q such that
    1. $r_0 = q_0$
    2. $\delta(r_1, w_{i+1}) = r_{i+1}$ for all $i = 0, \dots, n-1$
    3. $r_n \in F$
  - M recognizes language A if $A = \{w \mid M\ accepts\ w\}$

<u>DEFINITION 1.16</u>:

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

Designing Finite Automata

We want to accept binary strings with an odd number of 1s

Designing Finite Automata

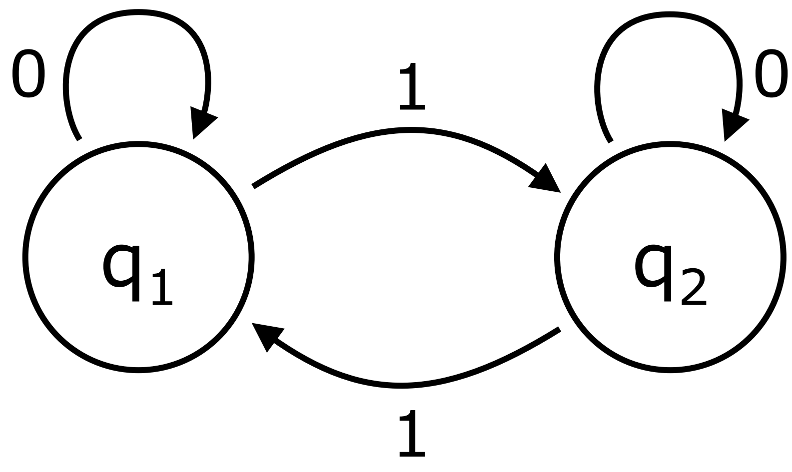
We want to accept binary strings with an odd number of 1s

1. Design states

Designing Finite Automata

We want to accept binary strings with an odd number of 1s

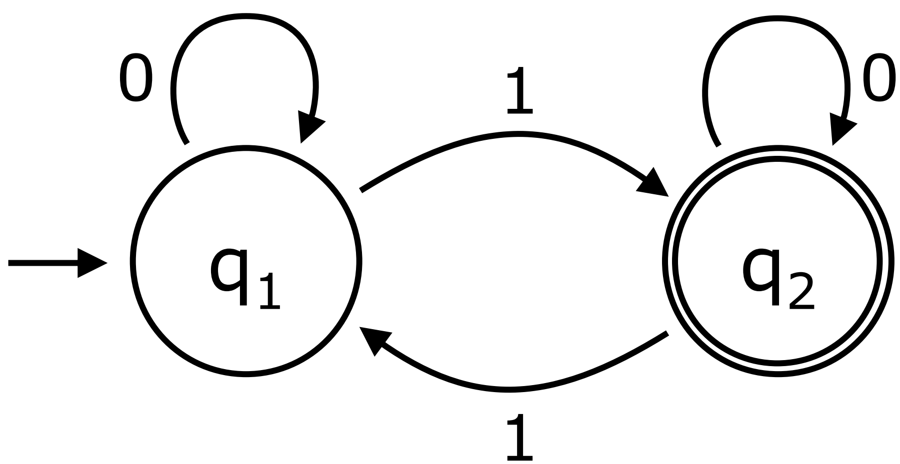
1. Design states
2. Design transitions



Designing Finite Automata

We want to accept binary strings with an odd number of 1s

1. Design states
2. Design transitions
3. Design start state and accept states



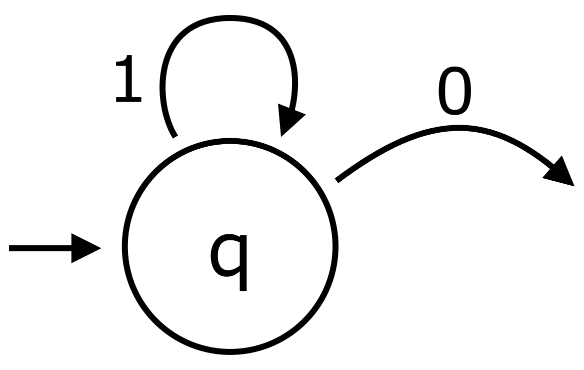
Designing Finite Automata

We want to accept binary strings containing 001 as substring

Designing Finite Automata

We want to accept binary strings containing 001 as substring

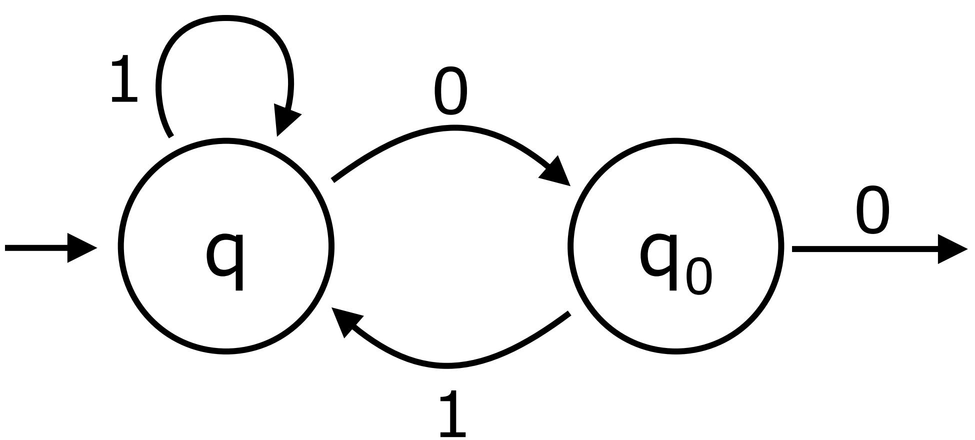
1. No symbols of the string



Designing Finite Automata

We want to accept binary strings containing 001 as substring

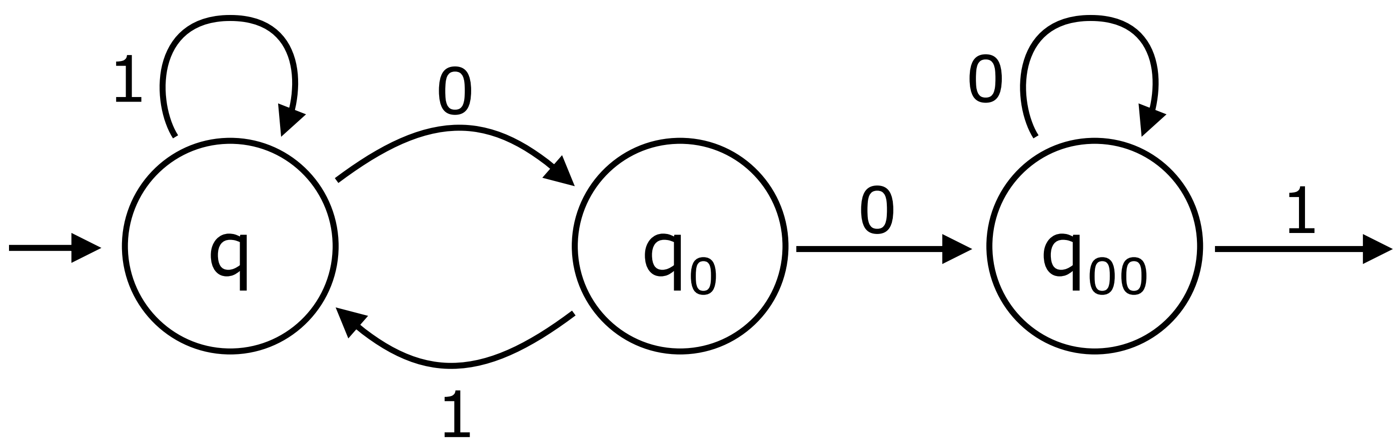
1. No symbols of the string
2. We have a 0



Designing Finite Automata

We want to accept binary strings containing 001 as substring

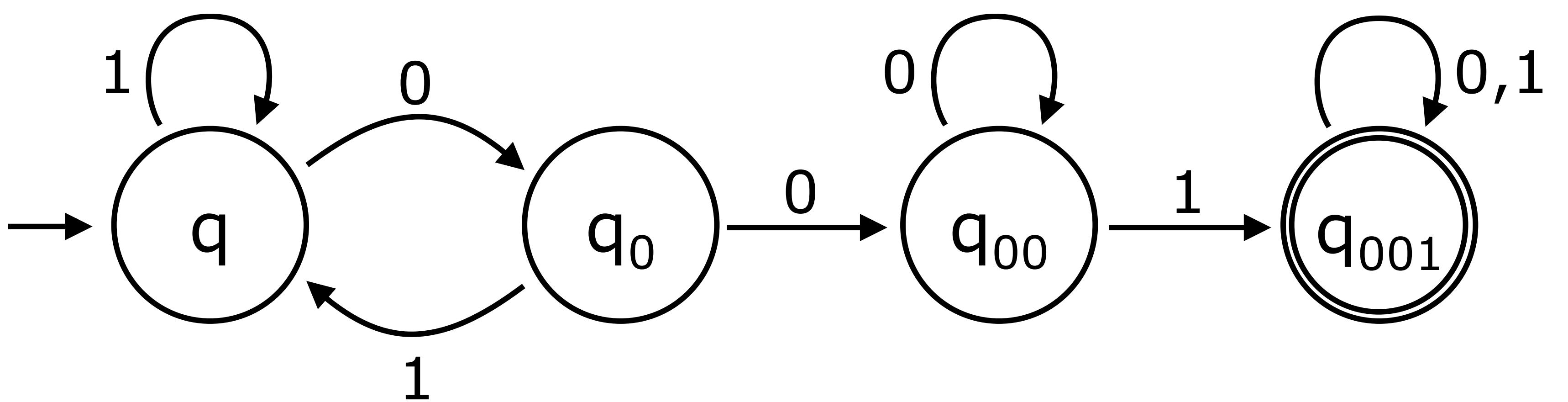
1. No symbols of the string
2. We have a 0
3. We have a 00



Designing Finite Automata

We want to accept binary strings containing 001 as substring

1. No symbols of the string
2. We have a 0
3. We have a 00
4. We have a 001



Regular Operations

Let A and B be languages, we have:

- Union: $A \cup B = \{x \mid x \in A \ or \ x \in B\}$
- Concatenation: $A \circ B = \{xy \mid x \in A \ and \ y \in B\}$
- Star: $A^* = \{x_1 x_2 \ldots x_n \mid n \geq 0 \ and \ x_i \in A\}$

- Example $A = \{empty, full\}; \ B = \{cup, glass\}$
  - $A \cup B?$
  - $A \circ B?$
  - $A^*?$

Closure of Regular Languages

A set S is **closed** under an operation o if applying o on elements of S yields elements of S.

- example: multiplication on natural numbers
- counterexample: division of natural numbers

Theorem 1.25:

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 by Construction

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 $M_1 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

Construct M to recognize $A_1 \cup A_2$, where $M = (Q, \Sigma, \delta, q_0, F)$.

1. $Q = \{(r_1, r_2) | \, r_1 \in Q_1 \, and \, r_2 \in Q_2\}$.
   This set is the **cartesian product** of the sets Q_1 and Q_2 (written $Q_1 \times Q_2$). It is the set of all pairs of states with the first from Q_1 and the second from Q_2.
2. Σ, the alphabet, is the same as in case of M_1 and M_2. The theorem remains true if they have different alphabets, Σ_1 and Σ_2. We would then modify the proof to let $\Sigma = \Sigma_1 \cup \Sigma_2$.

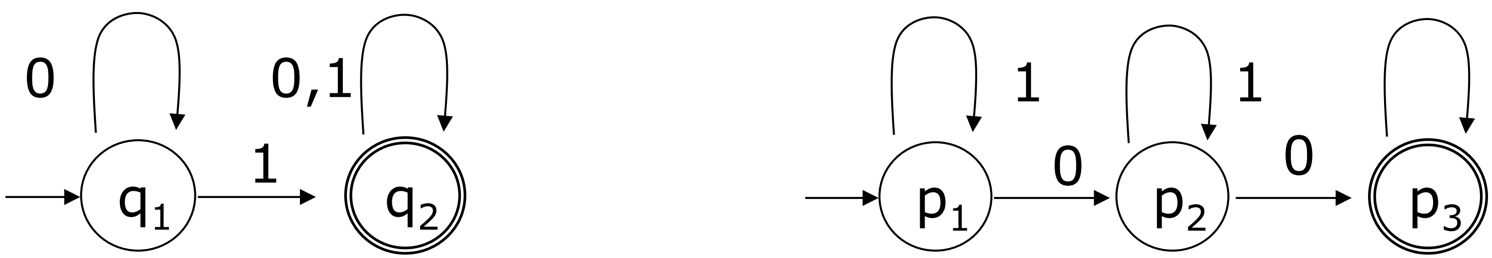
Proof by Construction

3. δ, the transistion function, is defined as follows.
   For each $(r_1, r_2) \in Q$ and each $a \in \Sigma$, let
   $$\delta\big((r_1, r_2), a\big) = (\delta_1(r_1, a), \delta_2(r_2, a)).$$
   Hence δ gets a state of M (which actually is a pair of states from M_1 and M_2), together with an input symbol, and returns M's next state.
4. q_0 is the pair (q_1, q_2).
5. F is the set of pairs, in which at leadt one member is an accept state of either M_1 or M_2. We can write this as
   $$F = \{(r_1, r_2) | \, r_1 \in F_1 \; or \; r_2 \in F_2\}.$$
   This expression is the same as $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$.
   (Note: it is not the same as $F = F_1 \times F_2$. What would that give us?)

■

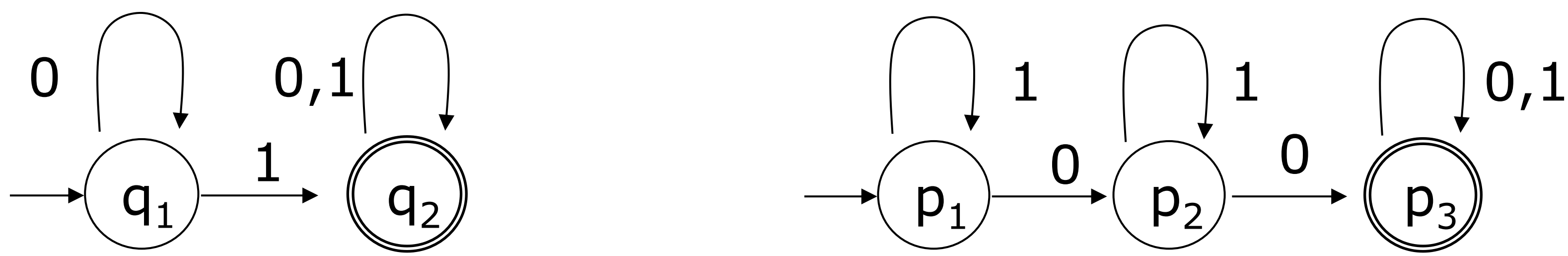
Example

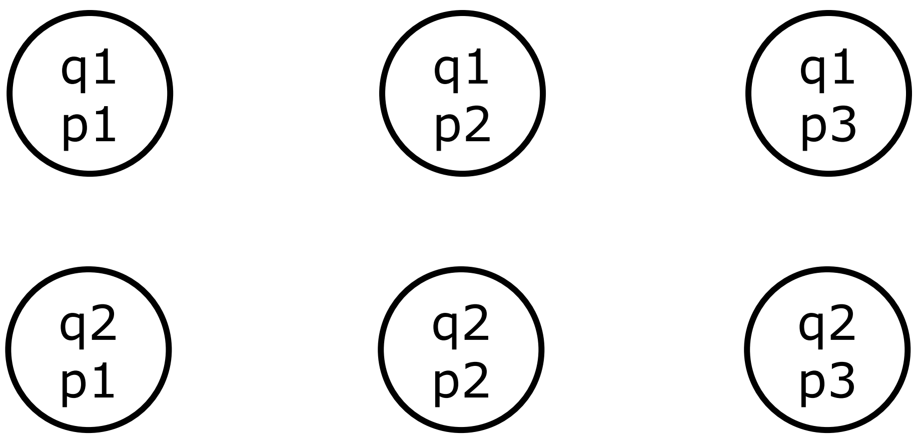
- L(M1) = {w|w contains a 1}
- L(M2) = {w|w contains at least two 0s}



Example

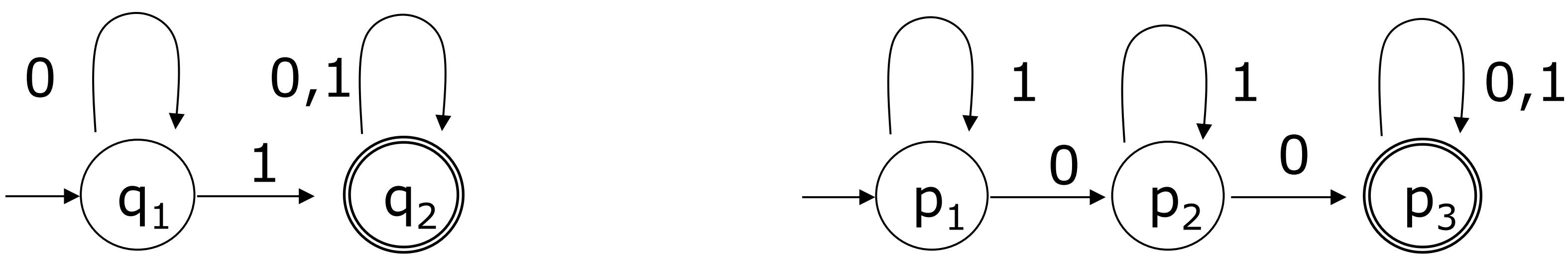
- L(M1) = {w|w contains a 1}
- L(M2) = {w|w contains at least two 0s}

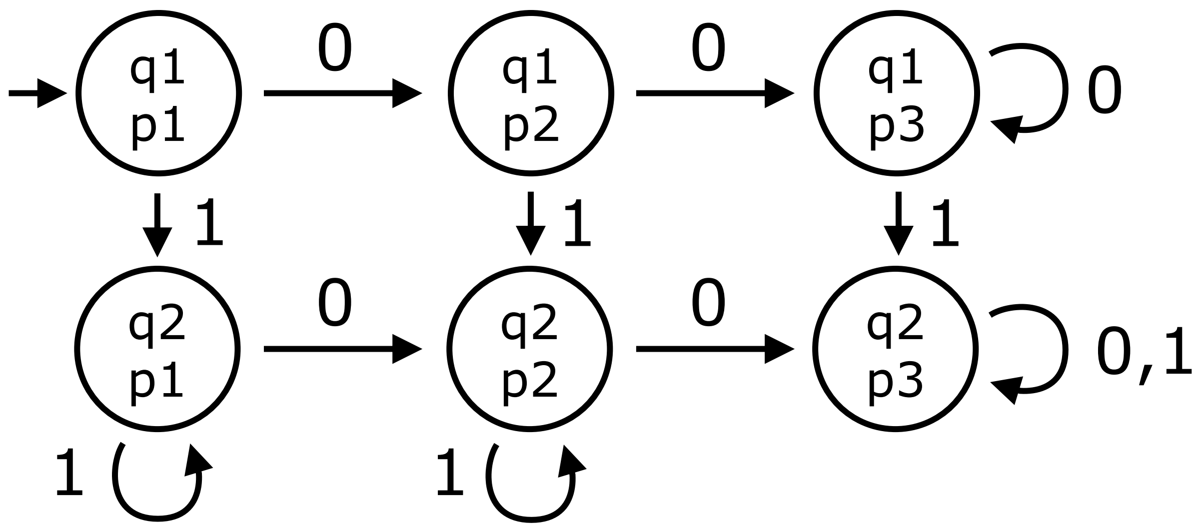




Example

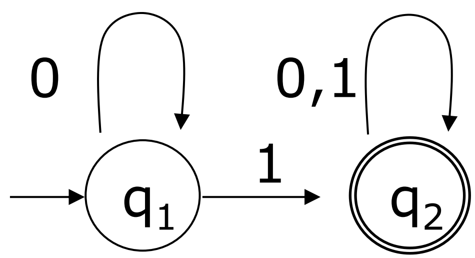
- $L(M1) = \{w|w \text{ contains a } 1\}$
- $L(M2) = \{w|w \text{ contains at least two 0s}\}$

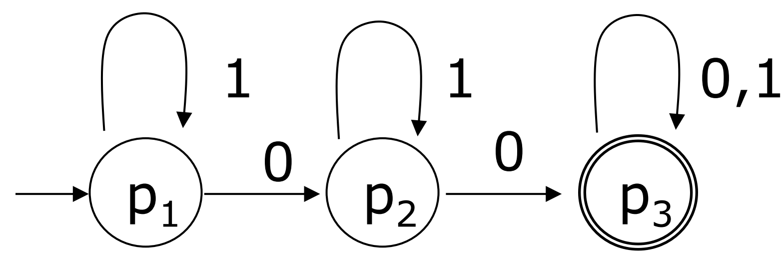


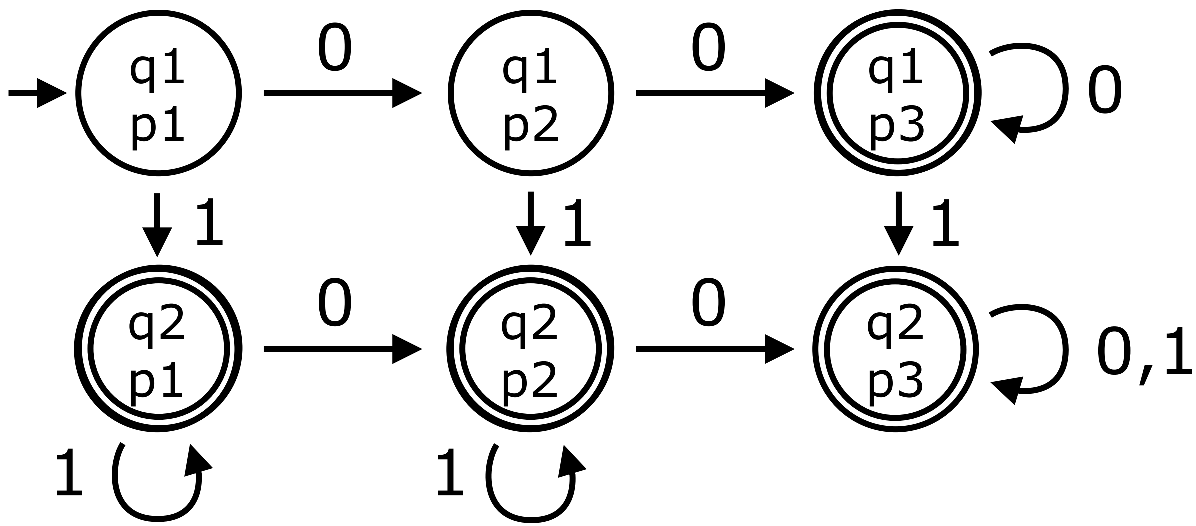


Example

- L(M1) = {w|w contains a 1}
- L(M2) = {w|w contains at least two 0s}







Closure of Regular Languages

Theorem 1.26:

The class of regular languages is closed under the concatenation operation.
(In other words: If A_1 and A_2 are regular languages, so is $A_1 \circ A_2$.)

Closure of Regular Languages

Theorem 1.26:

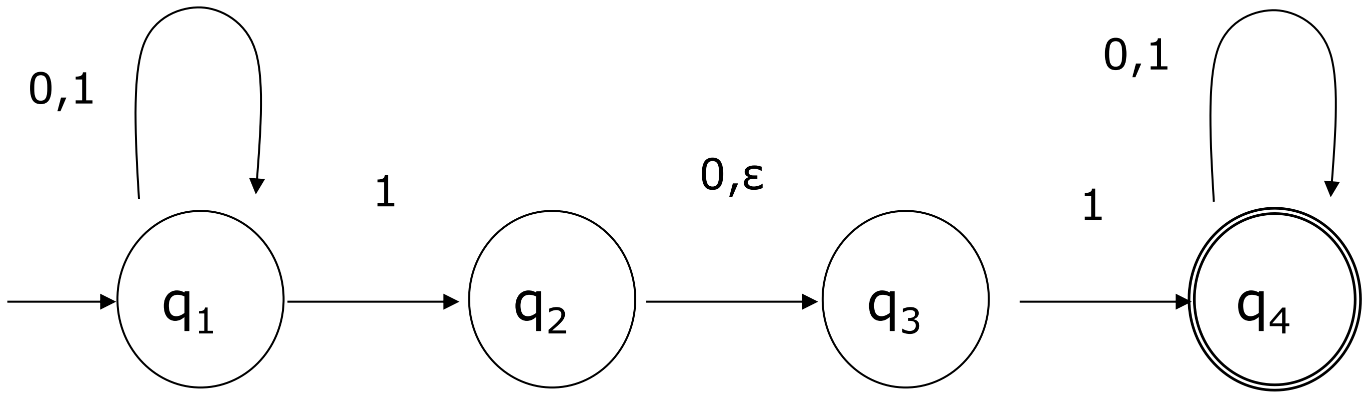
The class of regular languages is closed under the concatenation operation.

(In other words: If A_1 and A_2 are regular languages, so is $A_1 \circ A_2$.)

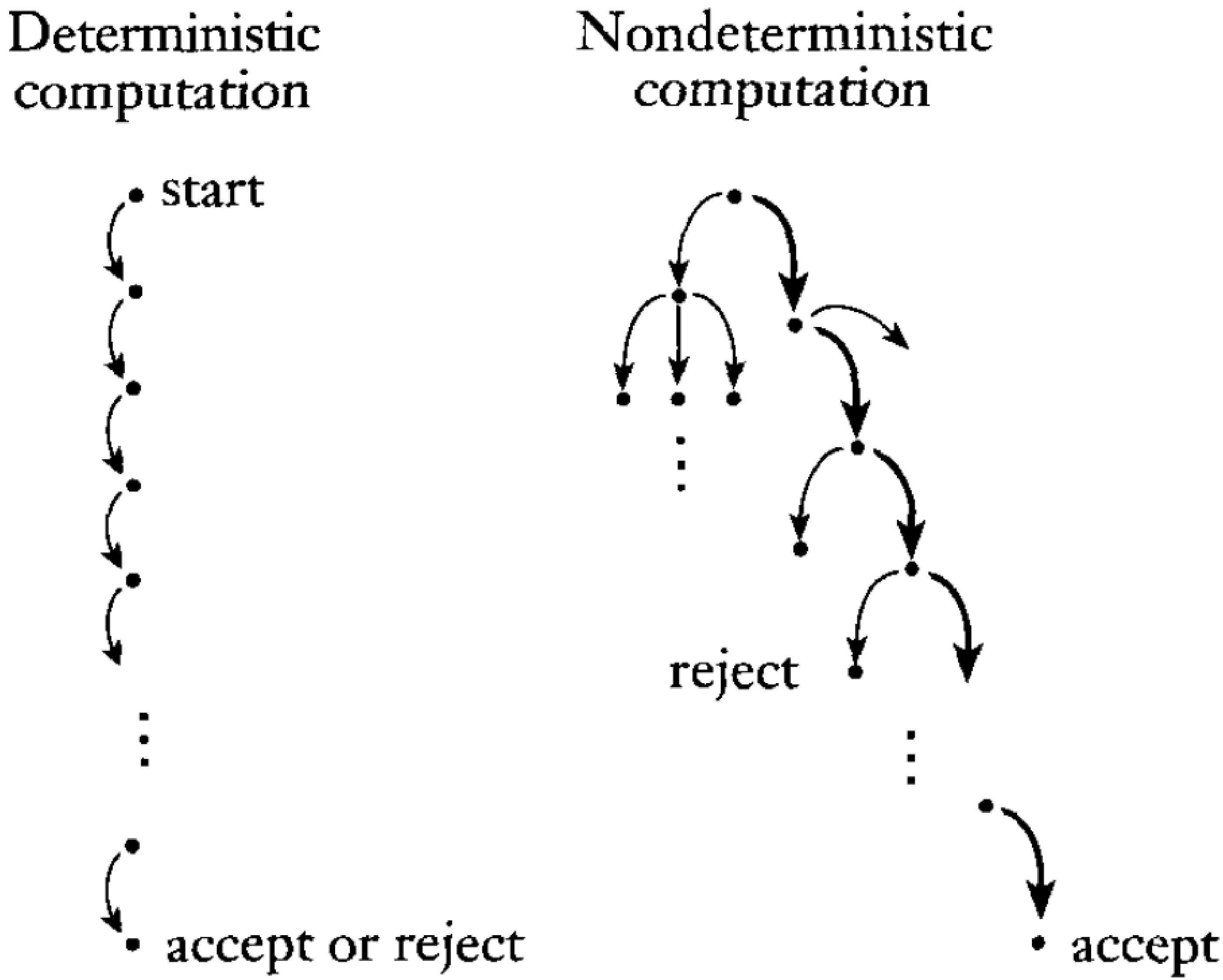
Non deterministic finite automata

Nondeterministic Automata

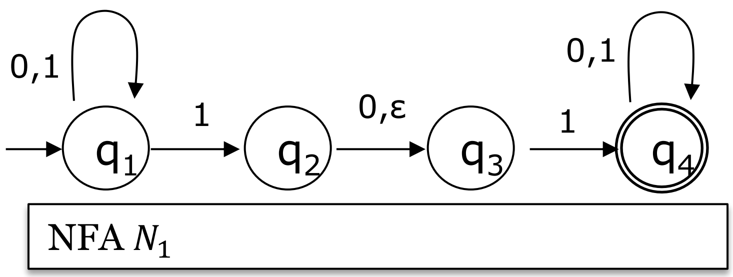
- Deterministic (DFA)
  - One successor state
  - ε transitions not allowed
- Nondeterministic (NFA)
  - Several successor states possible
  - ε transitions possible



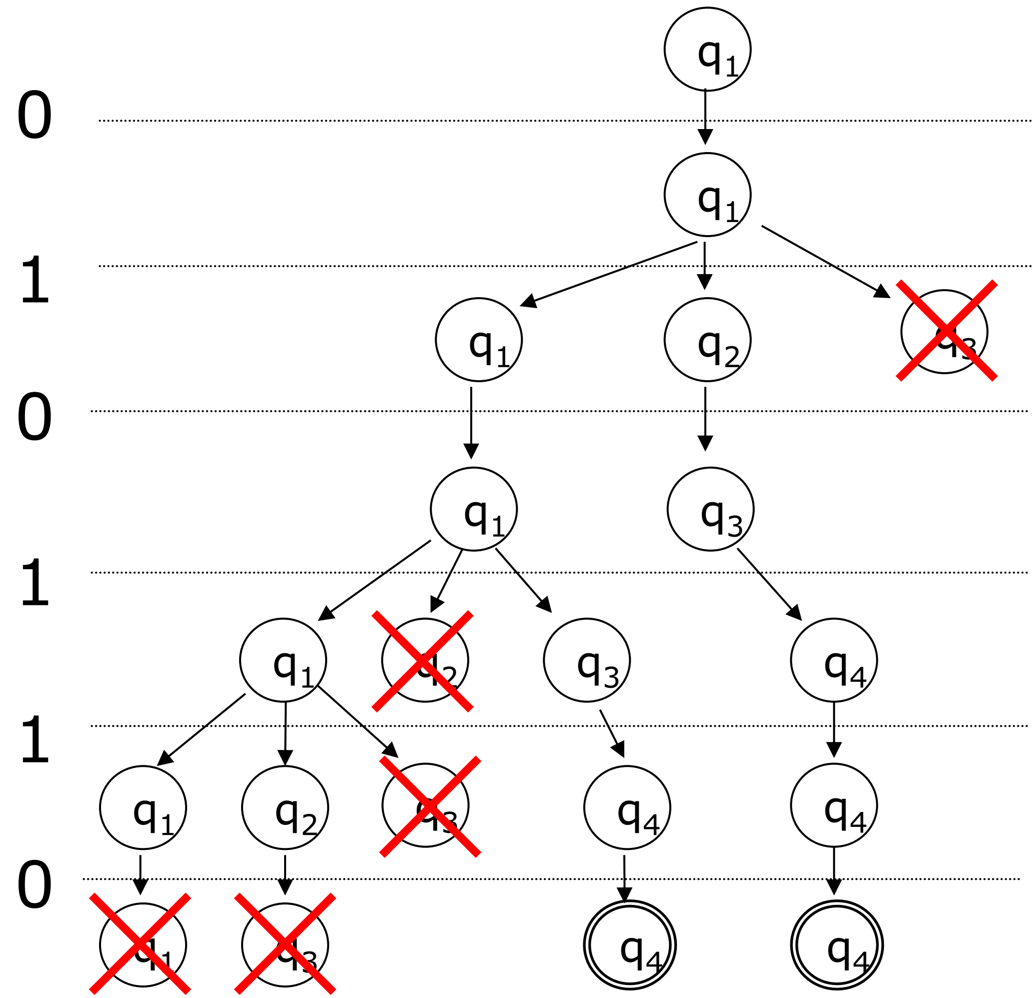
Nondeterministic Computation



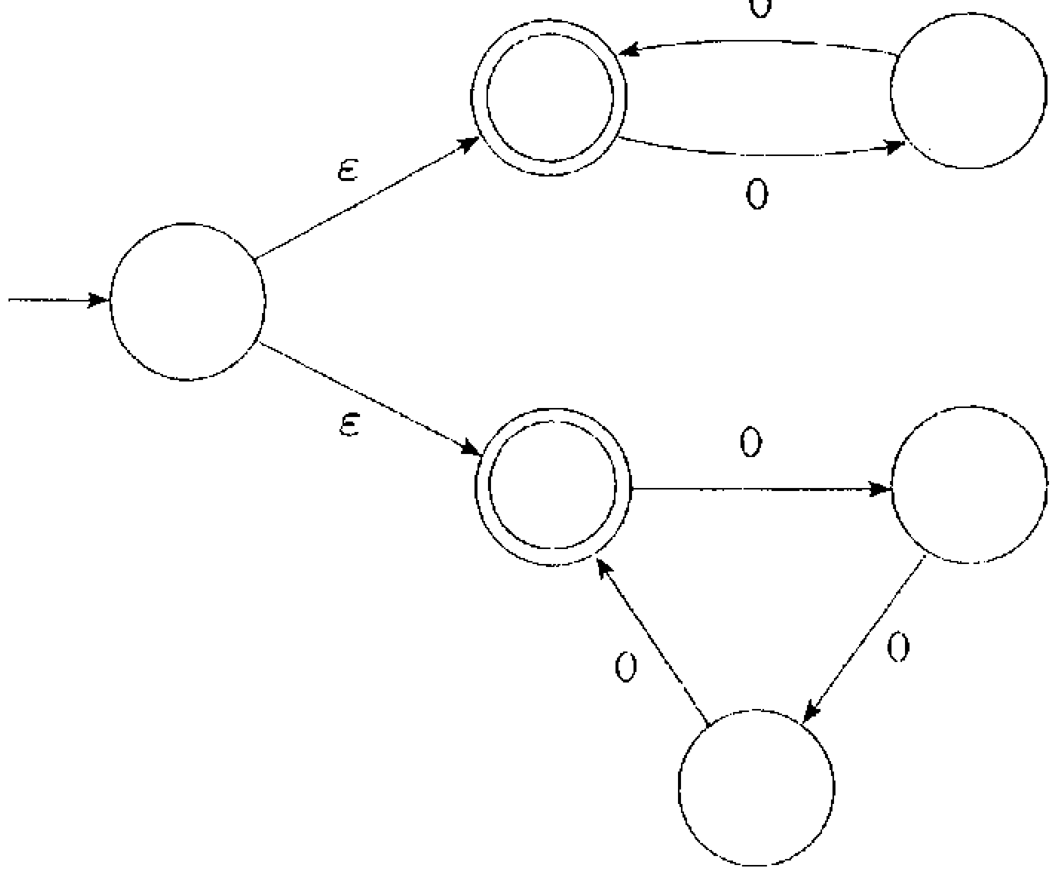
Example Run



Input: w = 010110



Which language is accepted?



Nondeterministic Automata

DEFINITION 1.37:

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

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

Σ_ε includes ε

$P(Q)$ the powerset of Q

Nondeterministic Automata

DEFINITION 1.37:

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

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

Σ_ε includes ε

$P(Q)$ the powerset of Q

Definition of computation

Let M be a finite automaton $(Q, \Sigma, \delta, q_0, F)$.
Let $w = w_1 \dots w_n$ be a string over Σ.

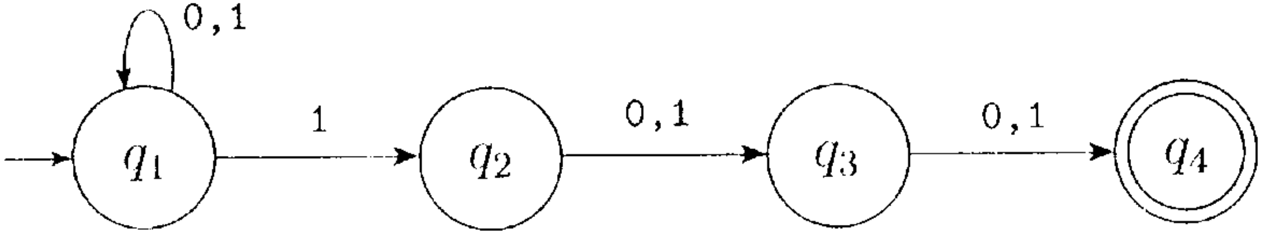
M **accepts** w if a sequence of states $r_0, \dots, r_n$ exists in Q such that

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

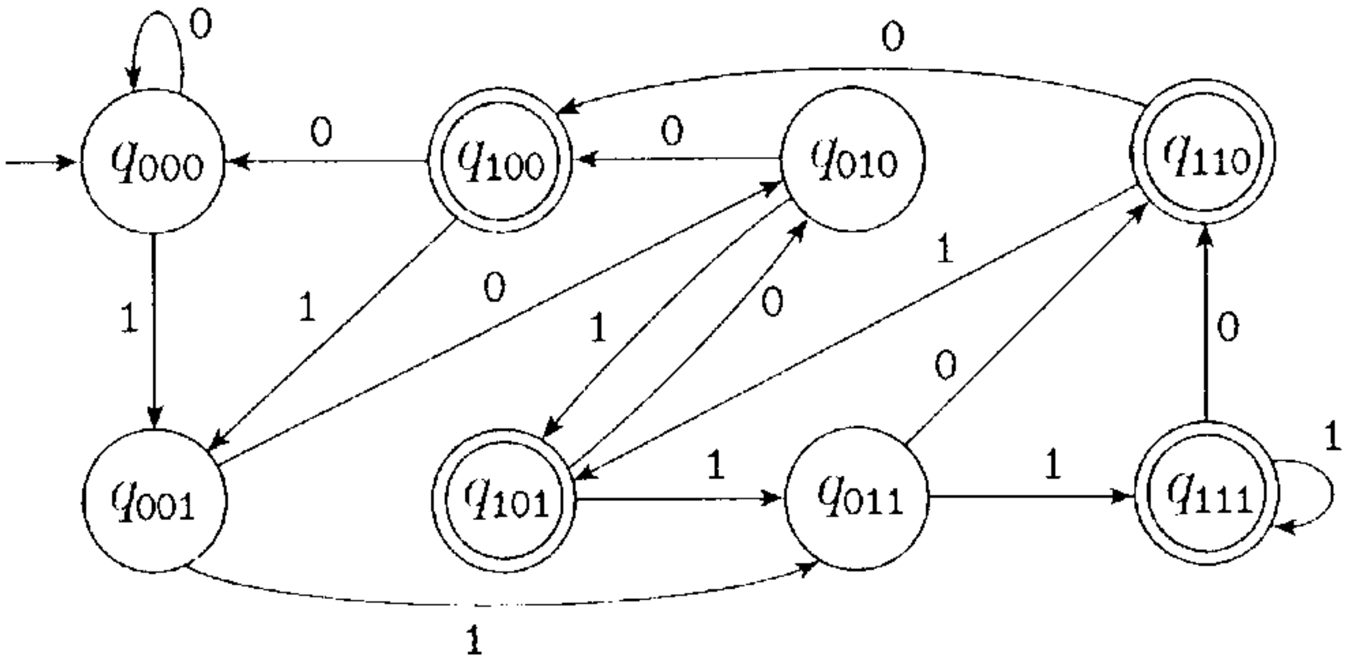
M **recognizes** language A if $A = \{w \mid M \textit{ accepts } w\}$.

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

A NFA has an equivalent DFA



NFA recognizing language *A*



DFA recognizing language *A*

Equivalence NFA and DFA

Theorem 1.39:

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

Corollary 1.40:

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

Proof: Theorem 1.39

Let $N = (Q, \Sigma, \delta_0, q_0, F)$ be the NFA recognizing some language A.

Idea: We show how to construct a DFA M recognizing A for any such NFA.

We start by only considering the easier case first, wherein N has no ε transitions. The ε transitions are taken into account later.

Proof: Theorem 1.39

Construct $M = (Q', \Sigma, \delta_0', q_0', F')$.

1. $Q' = P(Q)$.
   Every state of M is a set of states of N.
   (Recall that $P(Q)$ is the power set of Q).
2. For $R \in Q'$ and $a \in \Sigma$ let
   $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a)$ for some $r \in R\}$.
   If R is a state of M, it is also a set of states of N. When M reads a symbol a in state R, it tells us where a takes each state in R. Because each state leads to a set of states, we take the union of all these sets. Alternatively we can write:

   $$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$
3. $q_0' = \{q_0\}$. M starts in the state corresponding to the collection containing just the start state of N.

Proof: Theorem 1.39

4. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$.
   The machine M accepts if one of the possible states that N could be in at any given moment in an accept state.

The ε transitions need some extra notation:

a) For any state R of M we define $E(R)$ to be the collection of states that can be reached from R by means of any number of ε transitions alone, including the members of R themselves. Formally, for $R \subseteq Q$ let

$$E(R) = \{q \mid q \text{ can be reached from } R \text{ along } 0 \text{ or more } \varepsilon \text{ transitions}\}.$$

b) The transition function M is then modified to take into account all states that can be reached by going along ε transitions after every step. Replacing $\delta(r, a)$ by $E(\delta(r, a))$ achieves this. Thus,

$$\delta'(R, a) = \{q \in Q \mid q \in E\big(\delta(r, a)\big) \text{ for some } r \in R\}.$$

Proof: Theorem 1.39 (ctd.)

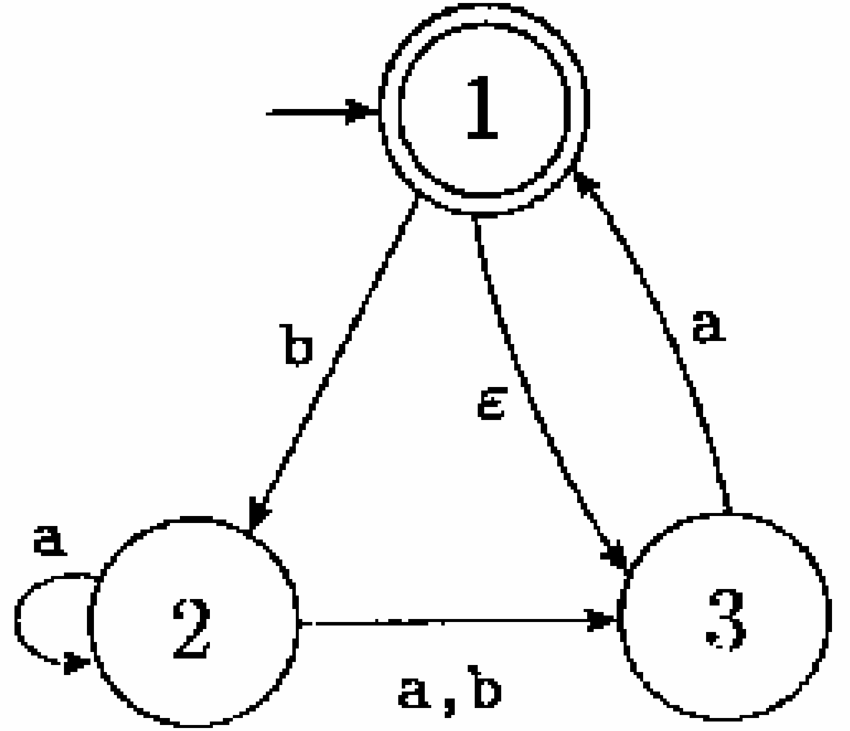
c) Finally, the start state of M has to cater for all possible states that can be reached from the start state of N along the ε transitions. Changing q_0 to be $E(\{q_0\})$ achieves this effect.

We have now completed the construction of the DFA M that simulates the NFA N.

■

Example

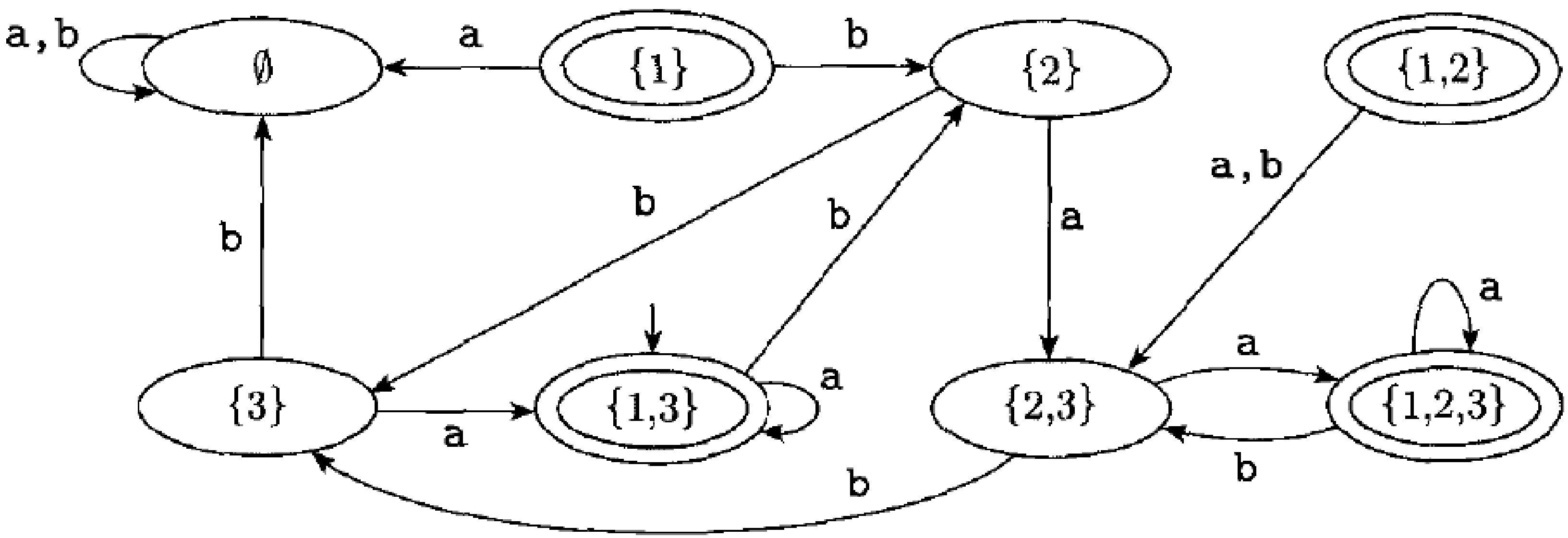
- Consider the following NFA



- What is the corresponding DFA?

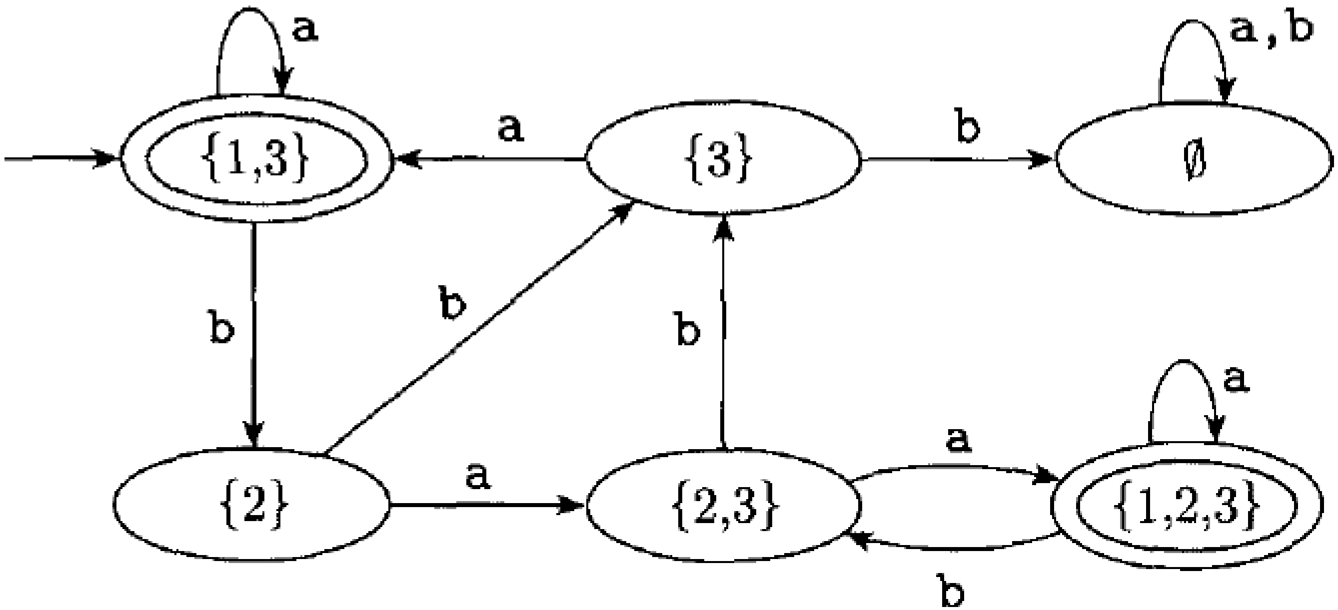
Example

- Resulting DFA for the example before



Example

- Simplified DFA for the example before



Closure of Regular Operations

Theorem 1.45:

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 $A_1 \cup A_2$.

Theorem 1.47:

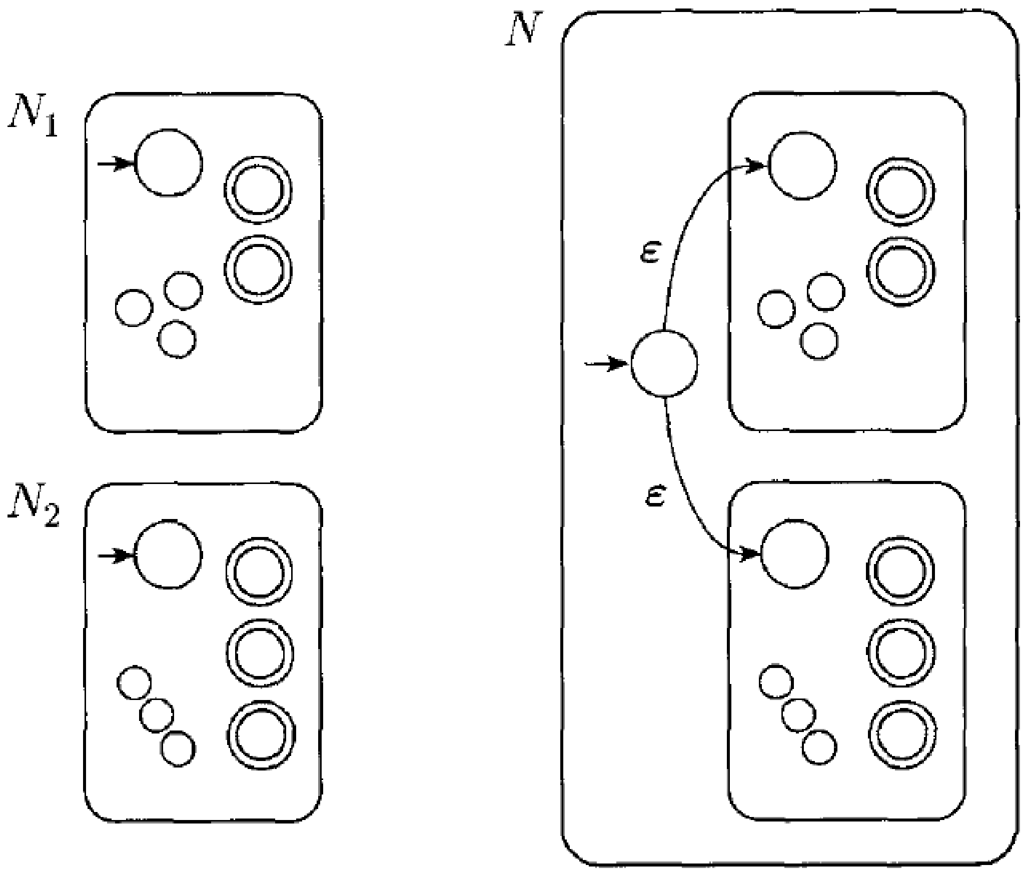
The class of regular languages is closed under the concatenation operation.

Theorem 1.49:

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

Closure of Regular Operations

- Regular languages are closed under the union operation



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 $A_1 \cup A_2$ as follows:

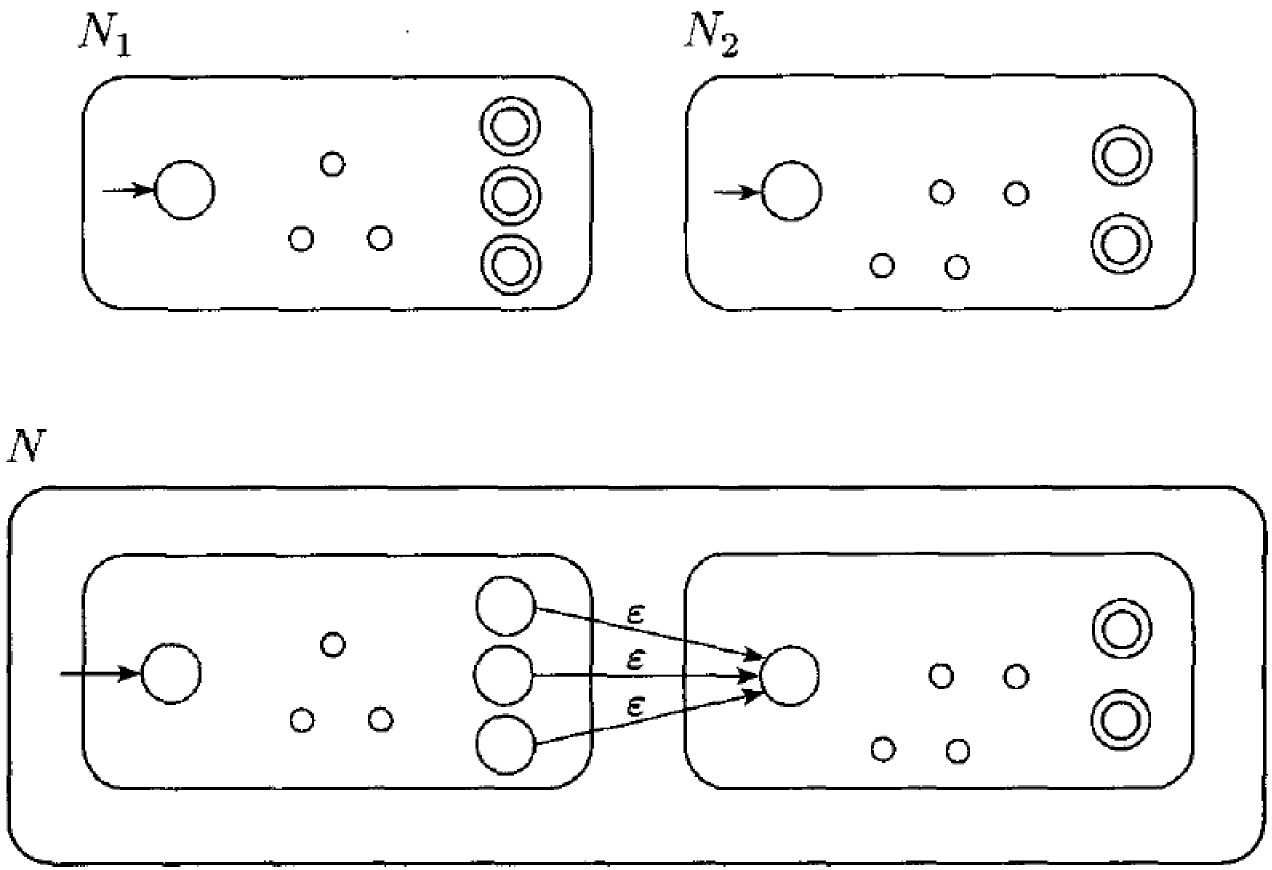
1. $Q = \{q_0\} \cup Q_1 \cup Q_2$. The **states** of N are all the states of N_1 and N_2, with the addition of the new start state q_0.
2. The state q_0 is the **start state** of N.
3. The **accept states** $F = F_1 \cup F_2$. The accept states are all the accept states of N_1 and N_2. That way N accepts if either N_1 or N_2 accepts.
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

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

∎

Closure of Regular Operations

- Regular languages are closed under the concatenation operation



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 $N = (Q, \Sigma, \delta, \boldsymbol{q_1}, \boldsymbol{F_2})$ to recognize $A_1 \circ A_2$ as follows:

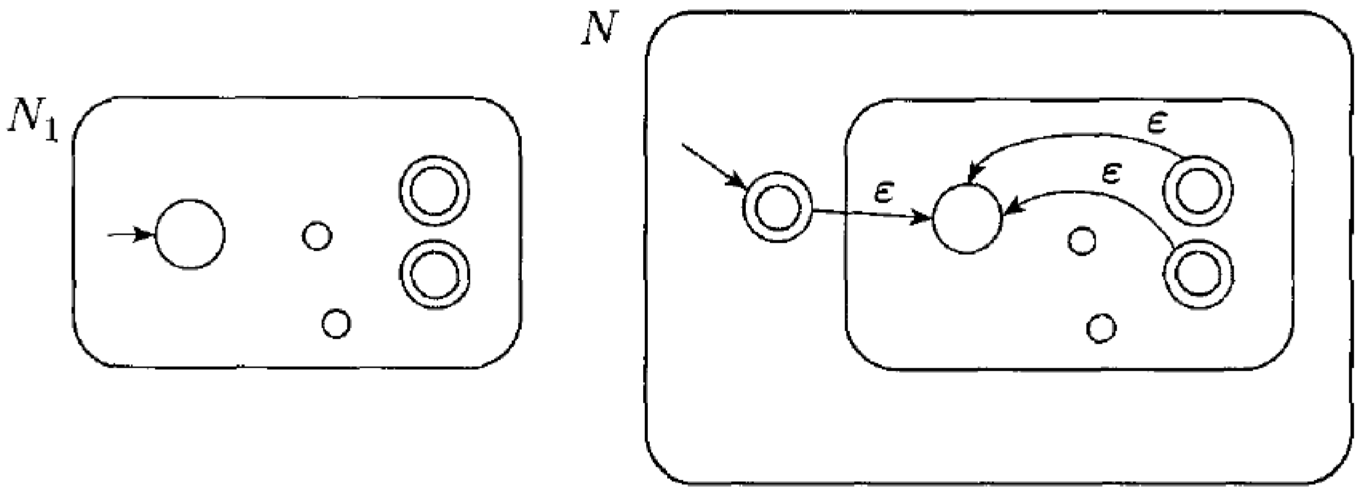
1. $Q = Q_1 \cup Q_2$. The **states** of N are all the states of N_1 and N_2.
2. The state q_1 is the **start state** of N, which is the same as the start state of N_1.
3. The **accept states** F_2 are the same as the accept states of N_2.
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

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

■

Closure of Regular Operations

- Regular languages are closed under the star operation



Proof

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1.

Construct $N = (Q, \Sigma, \delta, \boldsymbol{q_0}, \boldsymbol{F})$ to recognize A_1^* as follows:

1. $Q = \{q_0\} \cup Q_1$.
   The **states** of N are the states of N_1 plus a new start state q_0.
2. The state q_0 is the new **start state** of N.
3. $F = \{q_0\} \cup F_1$. The **accept states** are the old accept states plus the new start state.
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

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

■

Regular Expressions

DEFINITION 1.52:

Say that R is a **regular expression** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, 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 – Examples

Let $\Sigma = \{0,1\}$:

1. $0^*10^* = \{w \mid w \textit{ has exactly a single } 1\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w \textit{ has at least one } 1\}$.
3. $\Sigma^*001\Sigma^* = \{w \mid w \textit{ contains } 001 \textit{ as a substring}\}$.
4. $(01^+)^* = \{w \mid \textit{every } 0 \textit{ in } w \textit{ is followed by at least one } 1\}$.
5. $(\Sigma\Sigma)^* = \{w \mid w \textit{ is a string of even length}\}$.
6. $(\Sigma\Sigma\Sigma)^* = \{w \mid \textit{the length of } w \textit{ is a multiple of three}\}$.
7. $01 \cup 10 = \{01,10\}$.
8. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 =$
   $\{w \mid w \textit{ starts and with the same symbol as it ends}\}$.

Regular Expressions – Examples

Let $\Sigma = \{0,1\}$:

9. $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$.
The expression $0 \cup \varepsilon$ describes the language $\{0, \varepsilon\}$, so the concatenation operation adds either 0 or ε before every string in 1^*.

10. $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0,1,01\}$.

11. $1^*\emptyset = \emptyset$.
Concatenating the empty set to any set yields the empty set.

12. $\emptyset^* = \{\varepsilon\}$.
The star operation puts together any number of strings from the language to get a string in result. If the language is empty, the star operator can only put 0 strings together, giving only the empty string.

Applications of Regular Expressions

- Design of compilers
- Search for strings (awk, grep, …)
- Programming languages
- Bioinformatics (repetitive patterns)

Equivalence of RE and NFA

Theorem 1.54 (page 66):

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

Equivalence of RE and NFA

Theorem 1.54 (page 66):

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

Two directions to consider

RE <-> NFA

Equivalence of RE and NFA

Lemma 1.55 (page 67):

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

Lemma 1.60 (page 69):

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

Proof RE -> NFA

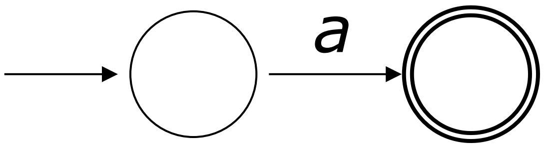
- Idea: Given a regular expression R describing a regular language A. We show how to convert R into an NFA recognizing A.
- Six cases have to be considered:
  1. $R = a$ for some $a \in \Sigma$, then $L(R) = \{a\}$.
  2. $R = \varepsilon$, then $L(R) = \{\varepsilon\}$.
  3. $R = \emptyset$, then $L(R) = \emptyset$.
  4. $R = R_1 \cup R_2$.
  5. $R = R_1 \circ R_2$.
  6. $R = R_1^*$.

Proof RE -> NFA: Case 1

Given: $R = a$ for some $a \in \Sigma$, then $L(R) = \{a\}$

The NFA $N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$
recognizes $L(R)$ with:

1. $\delta(q_1, a) = \{q_2\}$, and
2. $\delta(r, b) = \emptyset$, for $r \neq q_1$ or $b \neq a$.

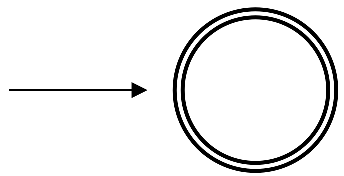


Note: this machine fits the definition of an NFA, but not that of a DFA, as not all input symbols have exiting arrows.

Proof RE -> NFA: Cases 2 & 3

Given: $R = \varepsilon$, then $L(R) = \{\varepsilon\}$.

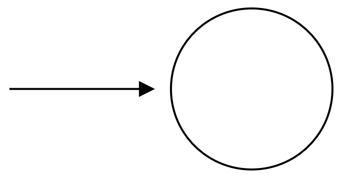
The NFA $N = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$
recognizes $L(R)$ with:



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

Given: $R = \emptyset$, then $L(R) = \emptyset$.

The NFA $N = (\{q\}, \Sigma, \delta, q, \emptyset)$
recognizes $L(R)$ with:



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

Proof RE -> NFA: Case 4, 5 & 6

Given:

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

The proofs for Theorems 1.45, 1.47, and 1.49 (slide 35, „closure of regular lanugages“) can be used to construct the NFA R from the NFAs for R_1 and R_2 (or just R_1 in case 6).

■

Example

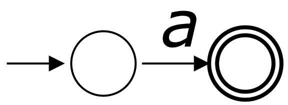
Let consider the expression (ab U a)*

- Convert the expression into a NFA
- Start from the smallest subexpression
- Include the other subexpressions

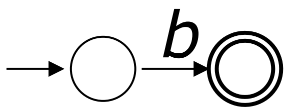
- Note: The states might be redundant!

Example: (ab U a)*

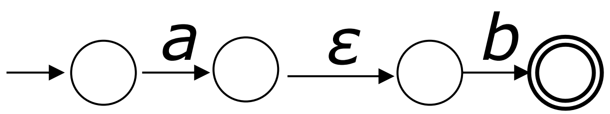
- a



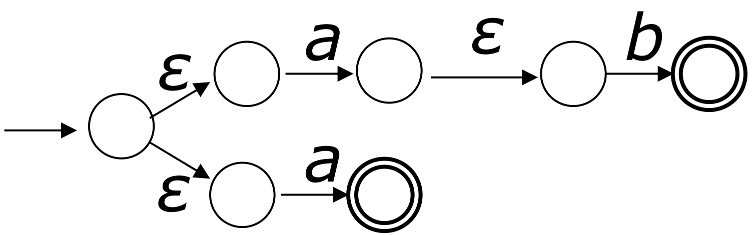
- b



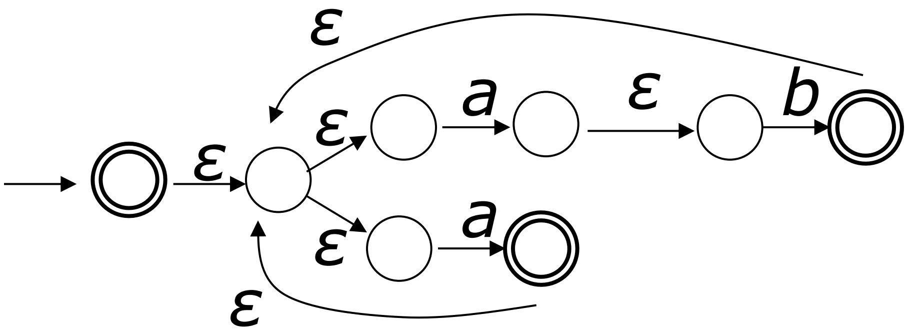
- ab



- ab U a



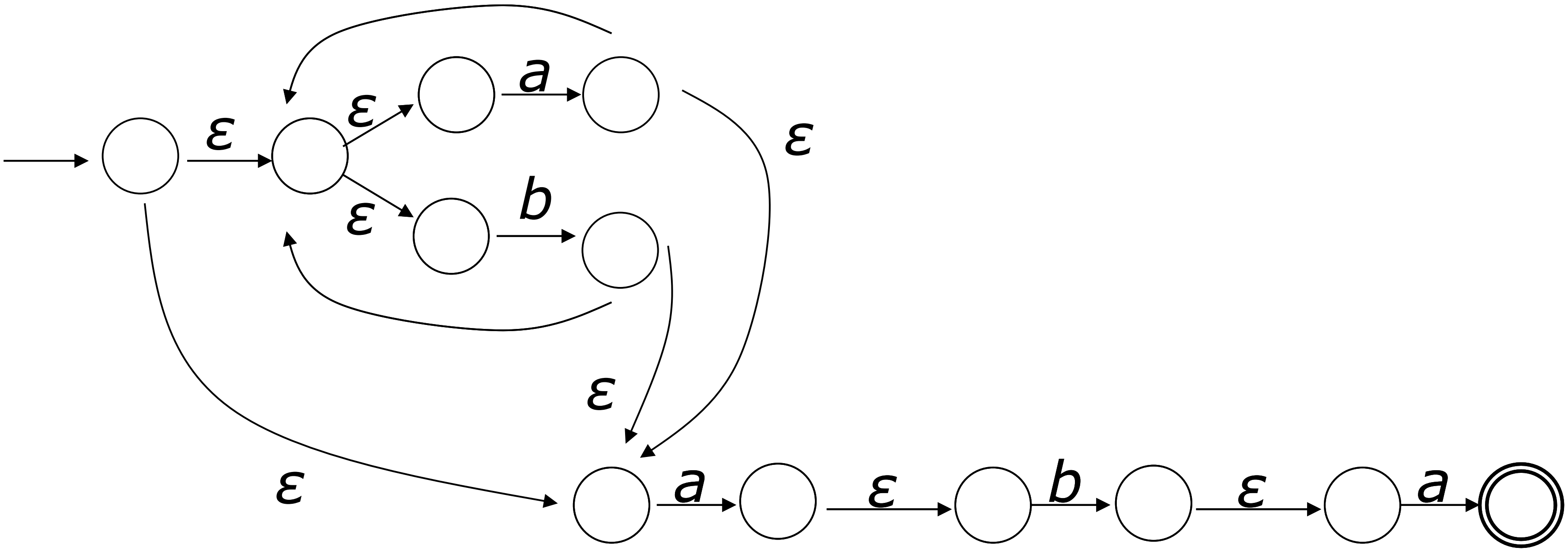
- (ab U a)*



Exercise: (ab U a)*

- Let’s do it together!

Exercise: (a U b)*aba



Proof NFA -> RE

Lemma 1.60 (page 69):

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

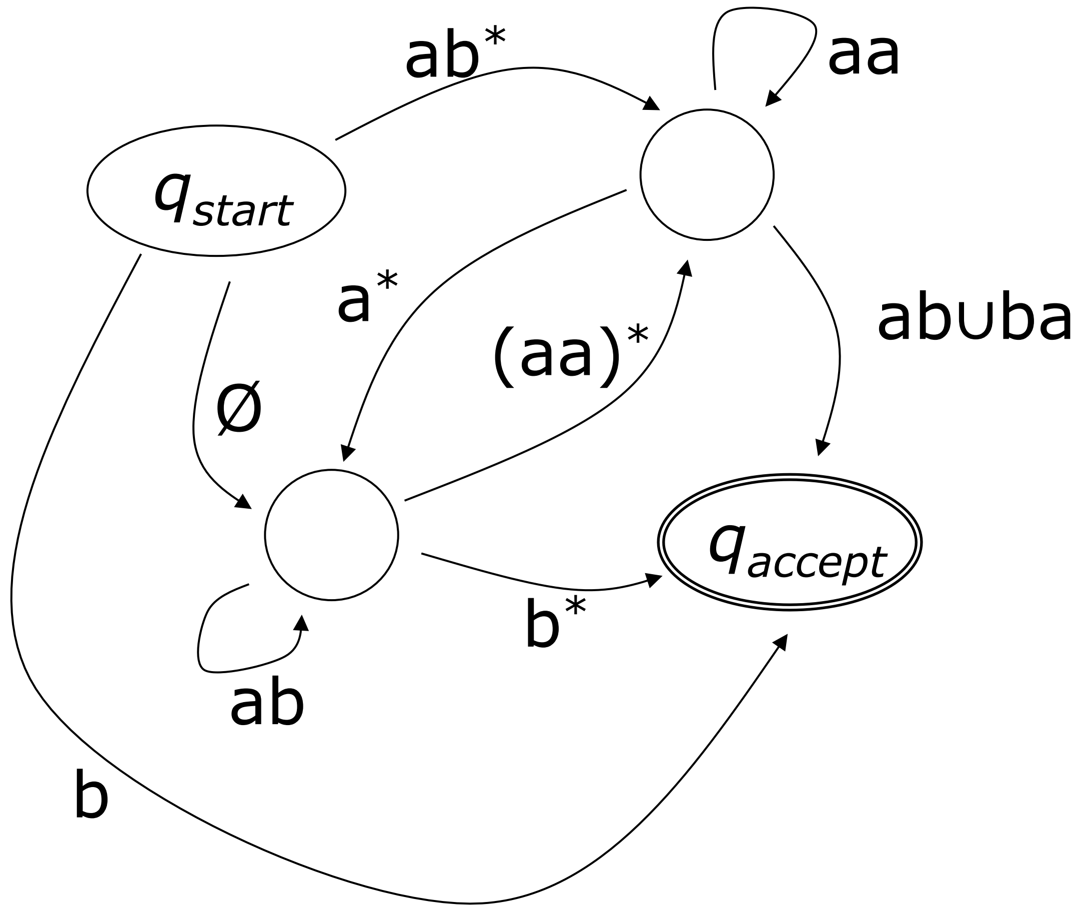
Two steps:

- Convert DFA into GNFA
- Convert GNFA into regular expression

Generalized NFA

- Labels are regular expressions
- States connected in both directions
- Start state only exit transitions
- Accept state only incoming transitions
- Only one accept state

Generalized NFA



Generalized NFA

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

1. Q a finite set of states
2. Σ a finite set, the alphabet
3. $\delta: (Q \setminus \{q_{accept}\}) \times (Q \setminus \{q_{start}\}) \rightarrow \mathcal{R}$ is the transition function
4. $q_{start} \in Q$ is the start state
5. $q_{accept} \in Q$ is the accept state

$\mathcal{R}$ represents the collection of all regular expressions over the alphabet Σ.

Generalized NFA

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

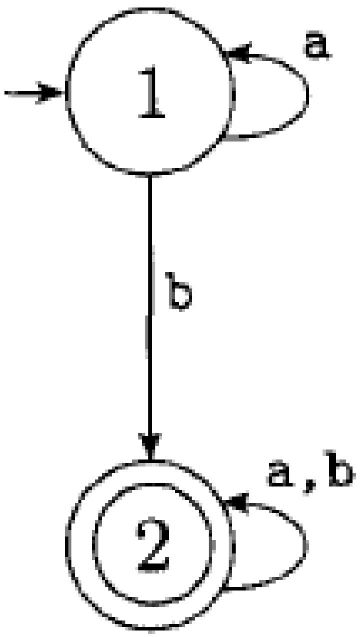
1. $q_0 = q_{start}$ is the start state,
2. $q_k = q_{accept}$ is the accept state, and
3. for each i, we have $w_i \in L(R_i)$, where $R_i = \delta(q_{i-1}, q_i)$; in other words, R_i is the expression on the arrow from q_{i-1} to q_i.

Proof DFA -> GNFA

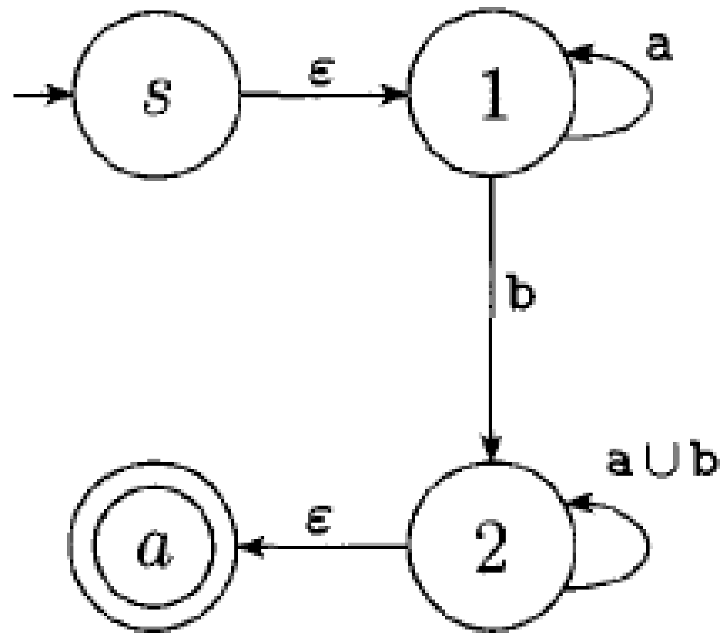
- Add a new start state
- Connect it with ε transitions
- Add a new accept state
- Connect it with ε transitions
- Replace multiple labels with unions
- Add transitions with $\emptyset$ when not present in the original DFA

Proof DFA -> GNFA

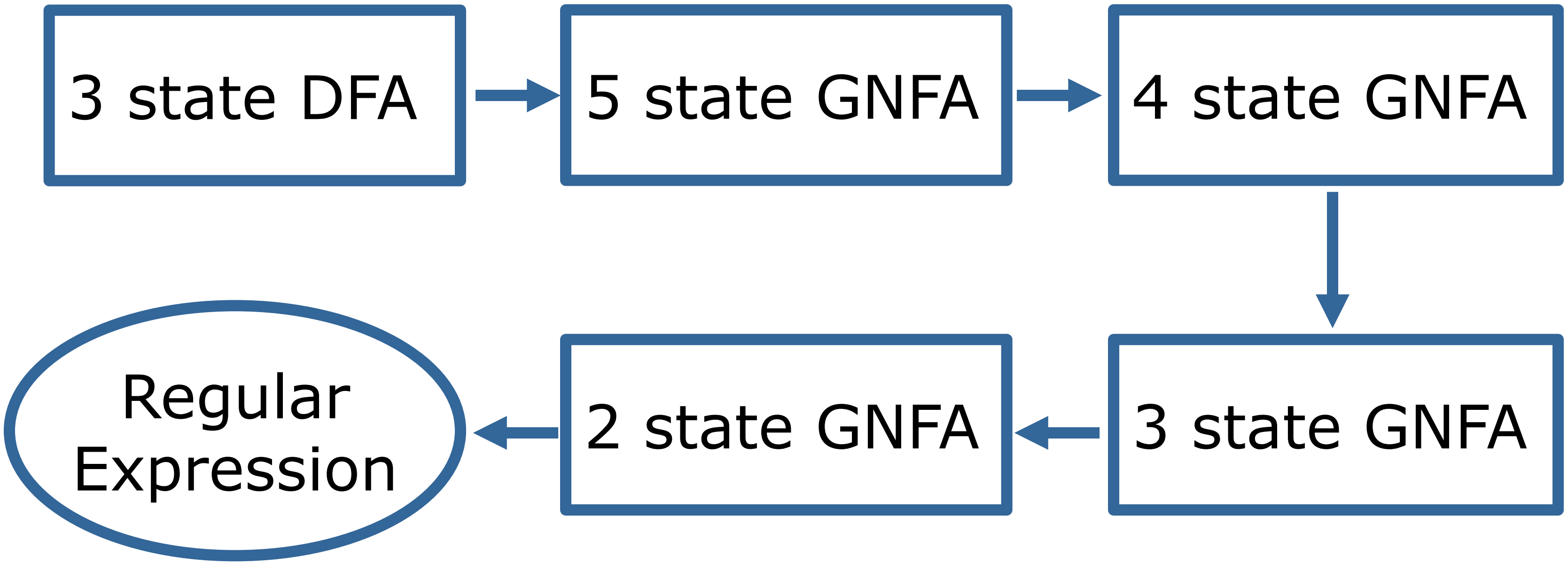
- DFA



- GNFA



Convert GNFA into RE

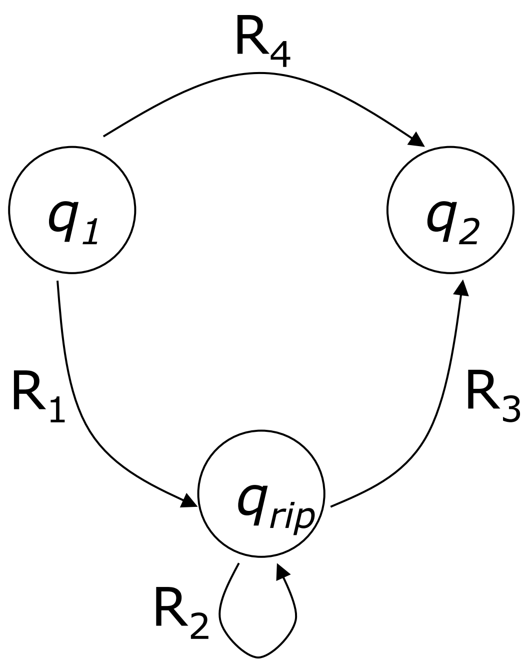


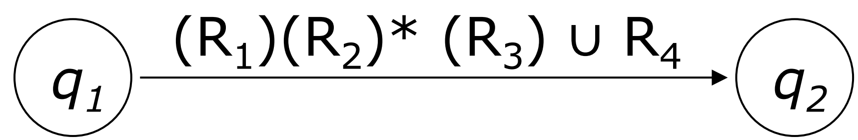
Convert GNFA into RE

1. Let k be the number of states of G.
2. If $k = 2$, then G must consists of a start state, an accept state, and a single transition connecting them, which is labeled with a regular expression R. Return the expression R and exit.
3. If $k > 2$, we select any state $q_{rip} \in Q$ different from q_{start} and q_{accept} and let G' ne the GNFA $(Q', \Sigma, \delta', q_{start}, q_{accept})$, where
$$Q' = Q \backslash \{q_{rip}\},$$
and for any $q_i \in Q' \backslash \{q_{accept}\}$ and any $q_j \in Q' \backslash \{q_{start}\}$ let
$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$
for $R_1 = \delta(q_i, q_{rip}), R_2 = \delta(q_{rip}, q_{rip}), R_3 = \delta(q_{rip}, q_j)$, and $R_4 = \delta(q_i, q_j)$.
4. Compute $Convert(G')$ and return this value.

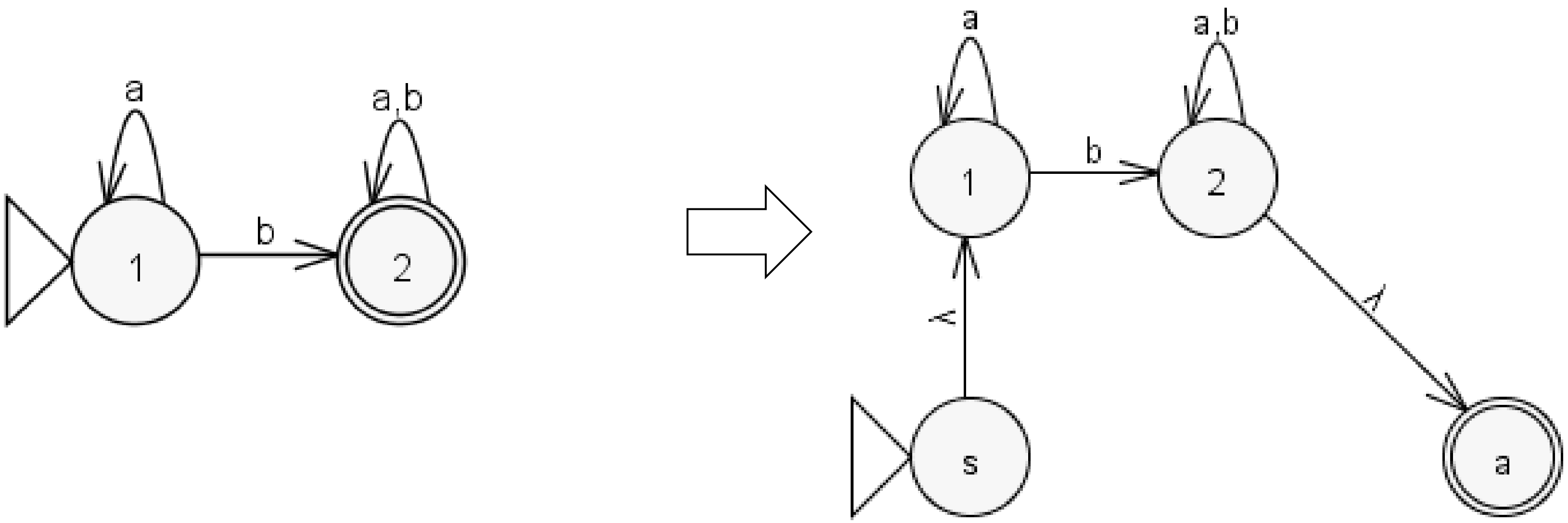
Ripping of States

Replace one state with the corresponding regular expression

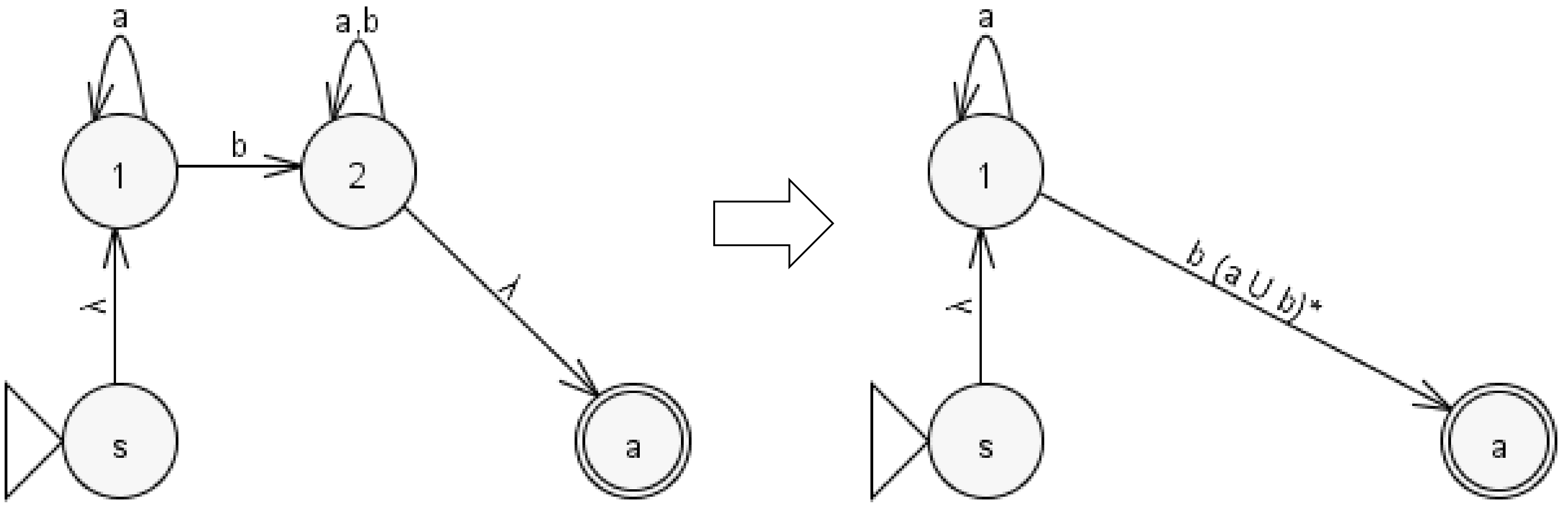




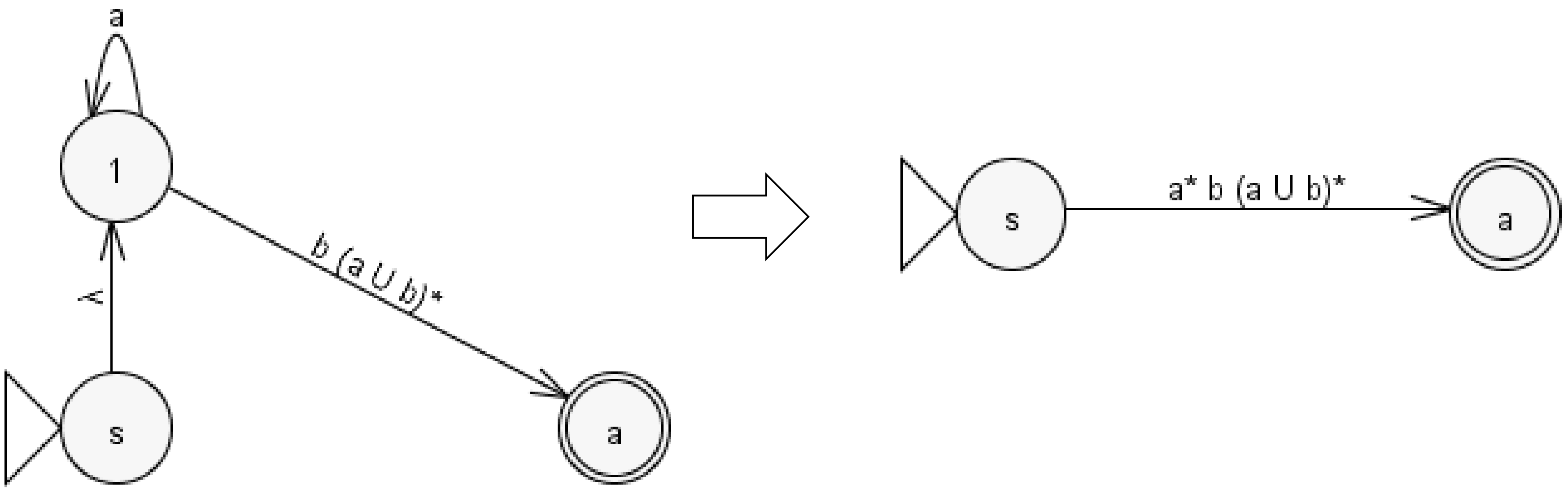
Example: From DFA to GNFA



Example: Rip State 2

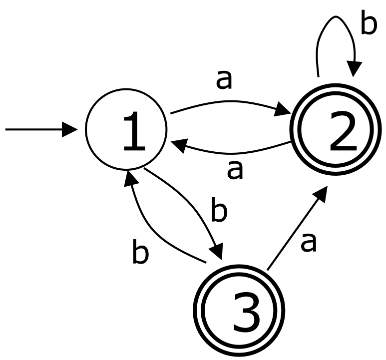


Example: Rip State 1

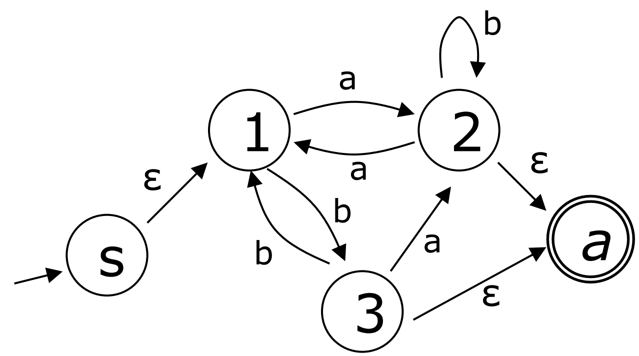


Another Example

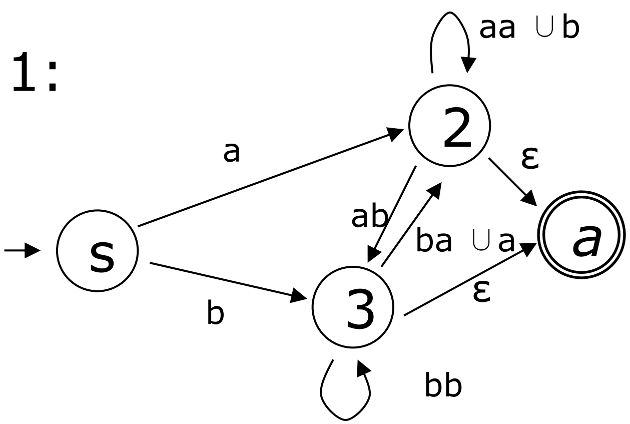
DFA:



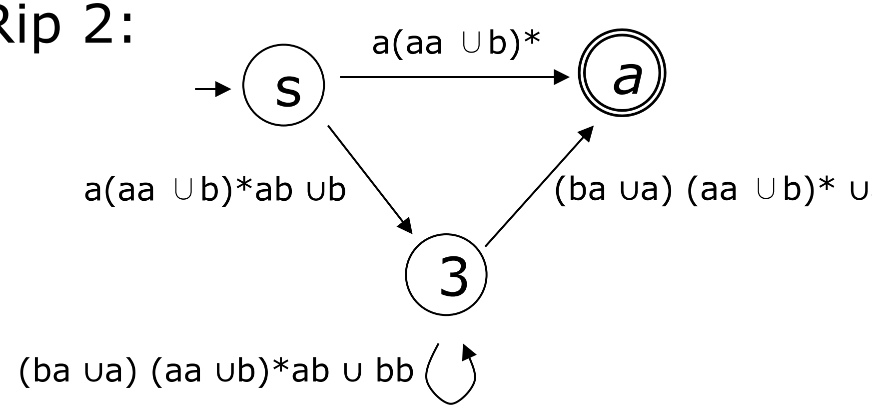
GNFA:



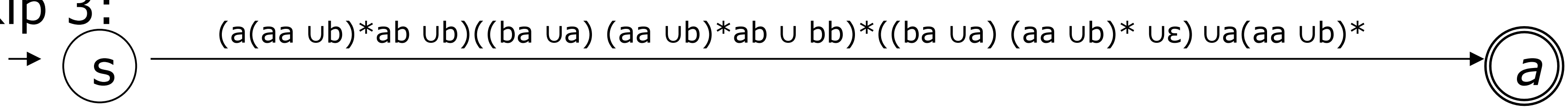
Rip 1:



Rip 2:



Rip 3:



Equivalence Proof

Claim 1.65: For any GNFA G, $Convert(G)$ is equivalent to G.

Procedure: We proof this claim by induction on k, the number of states of the GNFA.

Basis: Prove the claim true for $k = 2$ states. If G has only two states, it can have only a single transition, which goes from the start state to the accept state. The regular expression label on this transition describes all the strings that allow G to get to the accept state. Hence, this expression is equivalent to G.

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 we show that G and G' recognize the same language. Suppose that G accepts an input w. Then in an accepting branch of the computation G enters a sequence of states

$$q_{start}, q_1, q_2, q_3, \ldots, q_{accept}.$$

Equivalence Proof

$$q_{start}, q_1, q_2, q_3, \dots, q_{accept}.$$

If none of them is the removed state q_{rip}, clearly G' also accepts w, because each of the new regular expressions labeling the transitions of G' contains the old regular expression as part of a union.

If q_{rip} does appear, 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 transition between them that describes all strings taking q_i to q_j via q_{rip} on G. So G' accepts w.

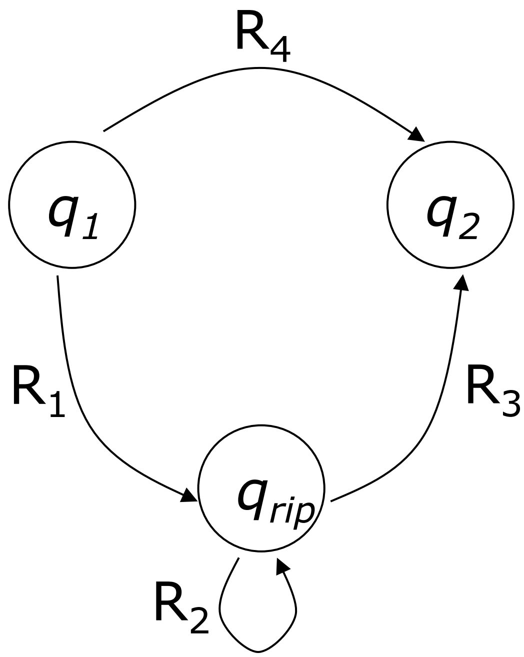
For the other direction, suppose that G' accepts an input w. As each transition between any two states q_i and q_j in G' describes the collection of strings taking q_i and q_j in G, either directly or via q_{rip}, G must also accept w. Thus, G and G' are equivalent.

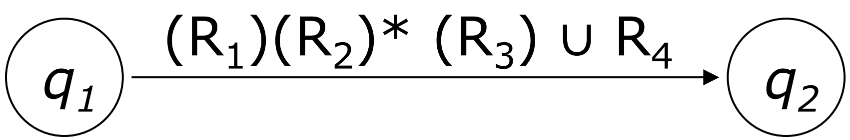
…

Equivalence Proof

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', because G' has $k-1$ states. Hence this regular expression also is equivalent to G, and the algorithm is proved correct.

■





Nonregular Languages

- Finite automata have finite memory
- Are the following language regular?

$$B = \{0^n 1^n \mid n \geq 0\}$$

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

$$D = \{w \mid w \text{ has an equal number of occurences of 01 and 10}\}$$

- How can we prove it mathematically?

The Pumping Lemma

If A is a regular language, then there is a number p (the pumping length), such that any string s of length at least p may be divided into three pieces, $s = xyz$, such that

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

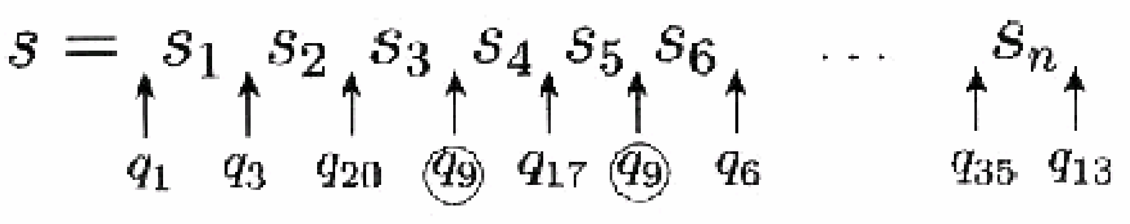
Note: from 2 follows that $y \neq \varepsilon$.

Proof Idea

- Let M be a DFA recognizing A
- Let p be the numbers of states in M
- Show that s can be broken into xyz
- Prove the conditions holds

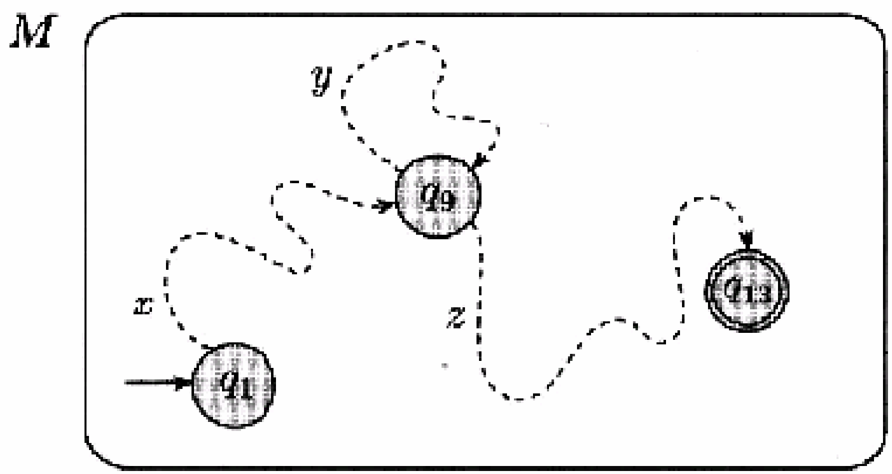
Proof Idea

- Let M be a DFA recognizing A
- Let p be the numbers of states in M
- Show that s can be broken into xyz
- Prove the conditions holds



Proof Idea

- Let M be a DFA recognizing A
- Let p be the numbers of states in M
- Show that s can be broken into xyz
- Prove the conditions holds



Proof of the Pumping Lemma

- Let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA recognizing A and $|Q| = p$.
- Let $s = s_1 s_2 \ldots s_n$ be a string in A, with $|s| = n$, and $n \geq p$
- Let $r = r_1, \ldots, r_{n+1}$ be the sequence of states that M enters for s,

  so $r_{i+1} = \delta(r_i, s_i)$ with $1 \leq i \leq n$. $|r_1, \ldots, r_{n+1}| = n + 1, n + 1 \geq p + 1$.
- Among the first $p + 1$ elements in r, there must be a r_j and a r_l being the same state q_m, with $j \neq l$.

  As r_l occurs in the first $p + 1$ states: $l \leq p + 1$.
- Let $\boldsymbol{x} = s_1 \ldots s_{j-1}$, $\boldsymbol{y} = s_j \ldots s_{l-1}$ and $\mathbf{z} = s_l \ldots s_n$:
  - as $\boldsymbol{x}$ takes M from r_1 to r_j, **y** from r_j to r_l, and **z** from r_l to r_{n+1}, being an accept state, M must accept xy^iz for $i \geq 0$.
  - with $j \neq l, |y| > 0$
  - with $l \leq p + 1, |xy| \leq p$

Use of the Pumping Lemma

Use pumping lemma to prove that a language *A* is not regular:

1. Assume that *A* is regular (Proof by contradiction)
2. use the lemma to guarantee the existence of *p*, such that strings of length *p* or greater can be pumped
3. find string ***s*** of *A*, with $|s| \geq p$ that cannot be pumped
4. demonstrate that s cannot be pumped using ***all different ways of dividing s into x,y, and z*** (using condition 3 is here very useful)
5. the existence of *s* contradicts the assumption, therefore *A* is not a regular language

Nonregular Languages

$$B = \{0^n 1^n \mid n \geq 0\}$$

- Choose string $s = 0^p 1^p$ for $p \in \mathbb{N}^+$ being the pumping length
- If we were to consider condition 2, then we would have that:
  1. string y consists only of 0s → $xyyz$ has more 0s than 1s → not a member of B → violates condition 1 → contradiction!
  2. string y consists only of 1s → similar argument as in case 1 → contradiction!
  3. string y consists of both 0s and 1s → $xyyz$ may have same number of 0s and 1s, but out of order with some 1s before 0s → contradiction!

Intuitive argument: A DFA M would need to be able to remember how many 0s have been seen so far as it reads the input. As the number of 0s isn't limited and all DFAs only have a finite number of states, B cannot be recognized by a DFA. Thus, the language B is not regular.

Nonregular Languages

$$C = \{w \mid w \; has \; an \; equal \; number \; of \; 0s \; and \; 1s\}$$

- Choose string $s = 0^p 1^p$ for $p \in \mathbb{N}^+$ being the pumping length
- Pumping s seems possible, but only if we ignored condition 3!
  - Condition3: $|xy| \leq p$
  - Thus, y consists of 0s only
  - Then $xyyz \notin C$ → Contradiction!

Alternative proof:

- We know that $B = \{0^n 1^n \mid n \geq 0\}$ is not regular.
- If C were regular, then $C \cap 0^* 1^* = B$ also regular, because regular languages are closed under intersection (cp. slide 14)!
  → Contradiction!

Nonregular Languages

$$F = \{ww \mid w \in \{0,1\}^*\}$$

- Choose string $s = 0^p0^p$ for $p \in \mathbb{N}^+$ being the pumping length
  - Does NOT WORK, because it CAN be pumped! Try again..
- Choose string $s = 0^p10^p1$ for $p \in \mathbb{N}^+$ being the pumping length
- We use condition 3 again:
  - Condition3: $|xy| \leq p$
  - Thus, y consists of 0s only
  - Then $xyyz \notin F$ → Contradiction!
- Choice of s is crucial
  - If some s does not work, try another one!

Nonregular Languages

$$E = \{0^i 1^j \mid i > j\}$$

- Choose string $s = 0^{p+1}1^p$ for $p \in \mathbb{N}^+$ being the pumping length
- We use condition 3 again:
  - Condition3: $|xy| \leq p$
  - Thus, y consists of 0s only
  - Then $xy^0z = xz \notin E$ → Contradiction!
- Here we use xy^0z instead of $xyyz$ as argument. This is commonly called „pumping down“.

Example Exam Question

Q: Use the pumping lemma to prove that
$L = \{0^k 1^j \mid k, j \geq 0 \; and \; k \; \geq 2j\}$ is not regular.

A: Assume that $L = \{0^k 1^j \mid k, j \geq 0 \; and \; k \; \geq 2j\}$ is regular. Let p be the pumping length of L. The pumping lemma states that for any string $s \in L$ of at least length p, there exist strings x, y, and z such that $s = xyz$, $|xy| \leq p$, $|y| \; > \; 0$, and for all $i \; \geq 0$: $xy^i z \; \in \; L$.

Choose $s = 0^{2p} 1^p$. Because $s \in L$ and $|s| = 3p \geq p$, we obtain from the pumping lemma the strings x, y, and z with the above properties. As $s = xyz$, $|xy| \leq p$, and s begins with $2p$ zeros, one can see that xy can only consist of zeros. If we pump s down, i.e. select $i = 0$, the string $xy^0 z = xz = 0^{2p-|y|} 1^p$.

As xz has p ones, and $|y| \; > \; 0$, xz has fewer than $2p$ zeros.

Hence $xz \notin L \Rightarrow$ CONTRADICTION.

Therefore L is not regular!

Summary

- Deterministic finite automata
- Regular languages
- Nondeterministic finite automata
- Closure operations
- Regular expressions
- Nonregular languages
- The pumping lemma