Finite Automata

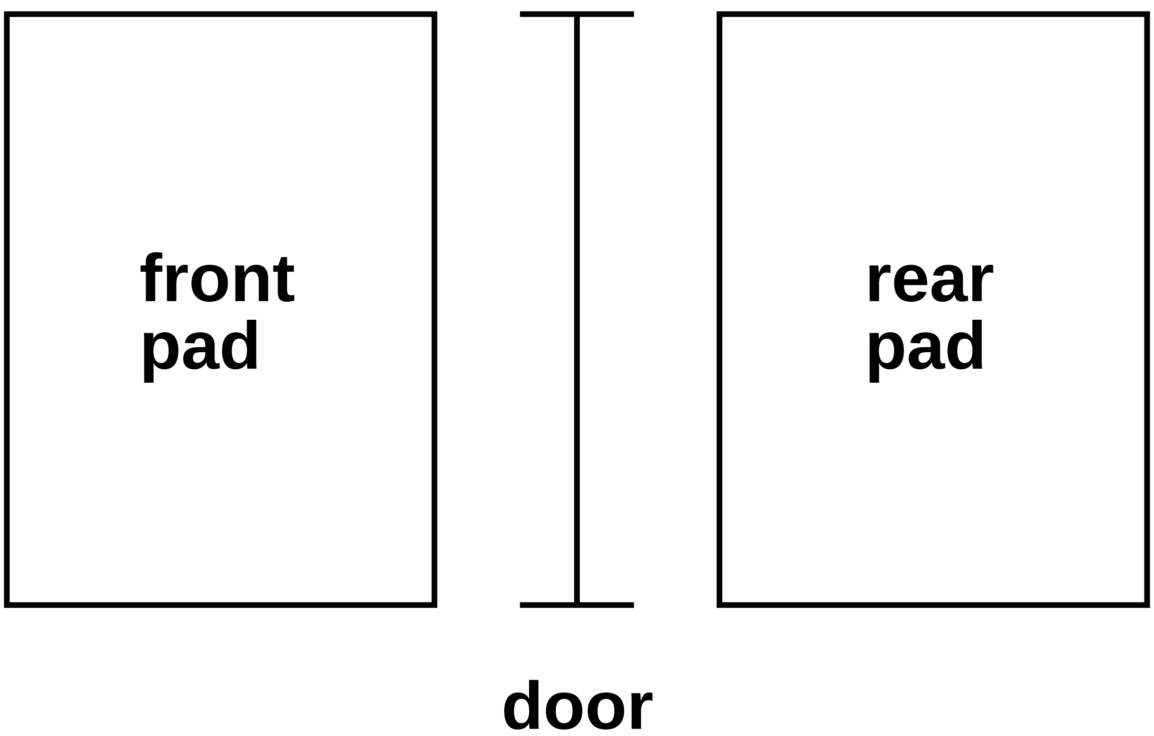
Question: What is a computer?

- real computers too complex for any theory
- need manageable mathematical abstraction
- idealized models: accurate in some ways, not others

important ideas

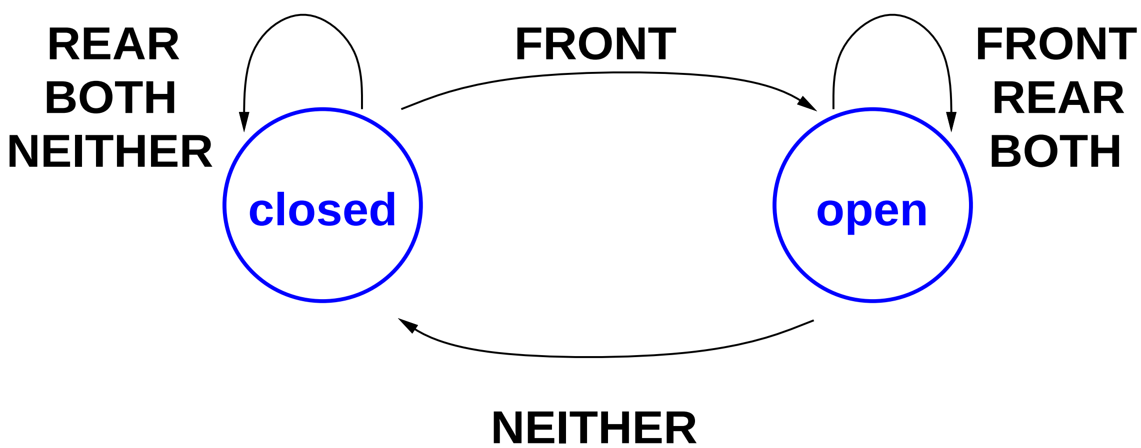
- formal definition of finite automata
- deterministic vs. non-deterministic finite automata
- regular languages
- operations on regular languages
- regular expressions
- pumping lemma

Example: An Automatic Door



- open when person approaches
- hold open until person clears
- don't open when someone standing behind the door

Example: An Automatic Door



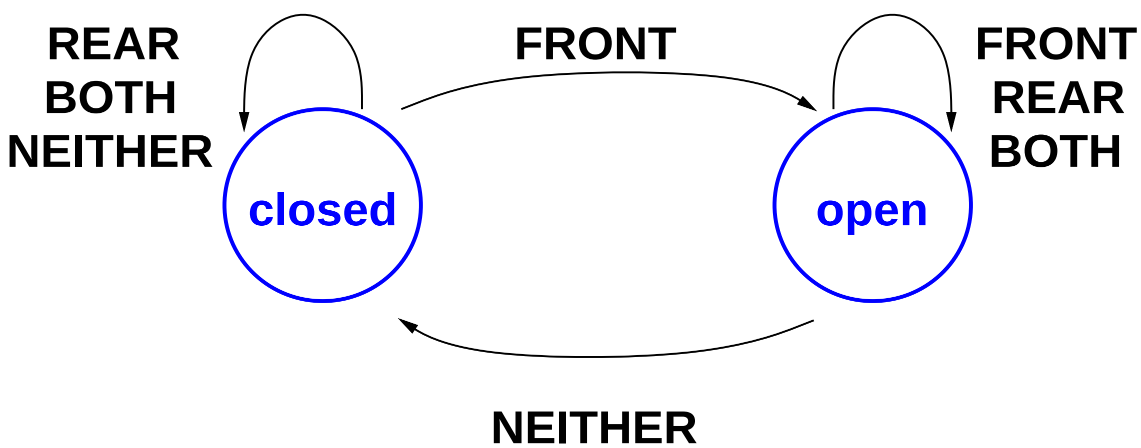
States:

- open
- closed

Sensor:

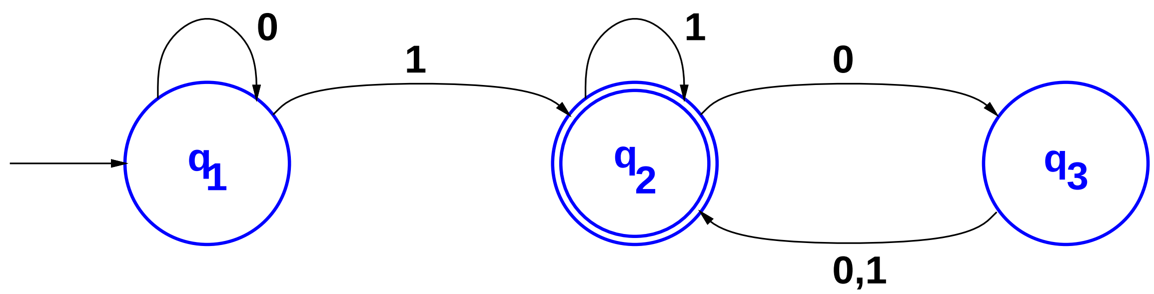
- FRONT: someone on rear pad
- REAR: someone on rear pad
- BOTH: someone on both pads
- NEITHER no one on either pad.

Example: An Automatic Door



	NEITHER	FRONT	REAR	BOTH
closed	closed	open	closed	closed
open	closed	open	open	open

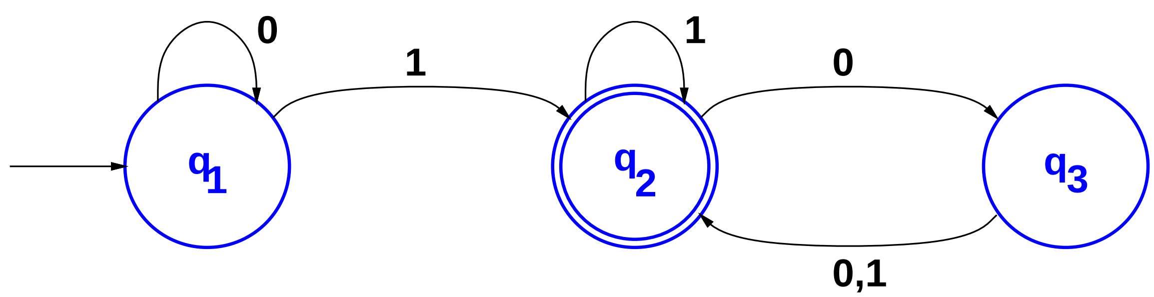
Informal Definition



The machine M_1:

- *states*: q_1, q_2, and q_3.
- *start state*: q_1 (arrow from nowhere).
- *accept state*: q_2 (double circle).
- *state transitions*: arrows.

Informal Definition



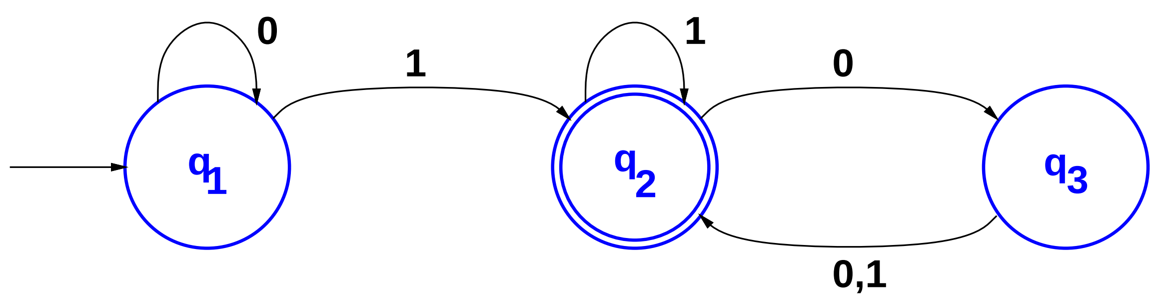
On an input string

- begins in start state q_1
- after reading each symbol, M_1 takes transition with matching label.

After reading last symbol, M_1 produces output:

- *accept* if M_1 is an accepting state.
- *reject* otherwise.

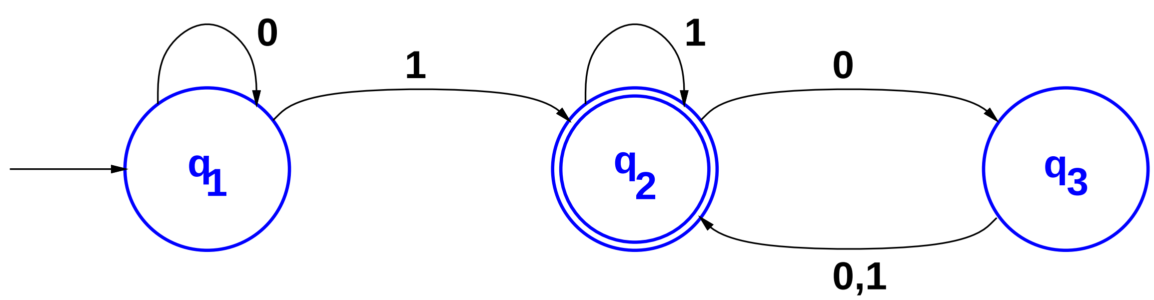
Informal Definition



What happens on input strings

- 1101
- 0010
- 01100

Informal Definition



M_1 accepts

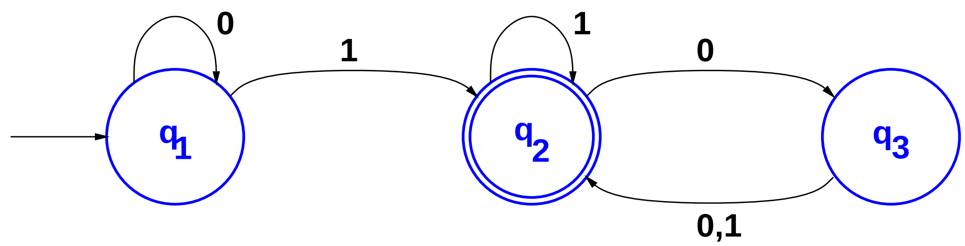
- all input strings that end with a 1
- all input strings that end with an even number of 0's
- no other strings

Formal Definitions

A *deterministic finite automaton* (DFA) 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 *alphabet*,
- $\delta : Q \times \Sigma \to Q$ is the *transition function*,
- $q_0 \in Q$ is the *start state*, and
- $F \subseteq Q$ is the set of *accept states*.

Back to M_1



$M_1 = (Q, \Sigma, \delta, q_1, F)$ where

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

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

- q_1 is the start state, and
- $F = \{q_2\}$.

Languages

Definition The *language of machine* M is the set of strings A that M accepts.

$$L(M) = A$$

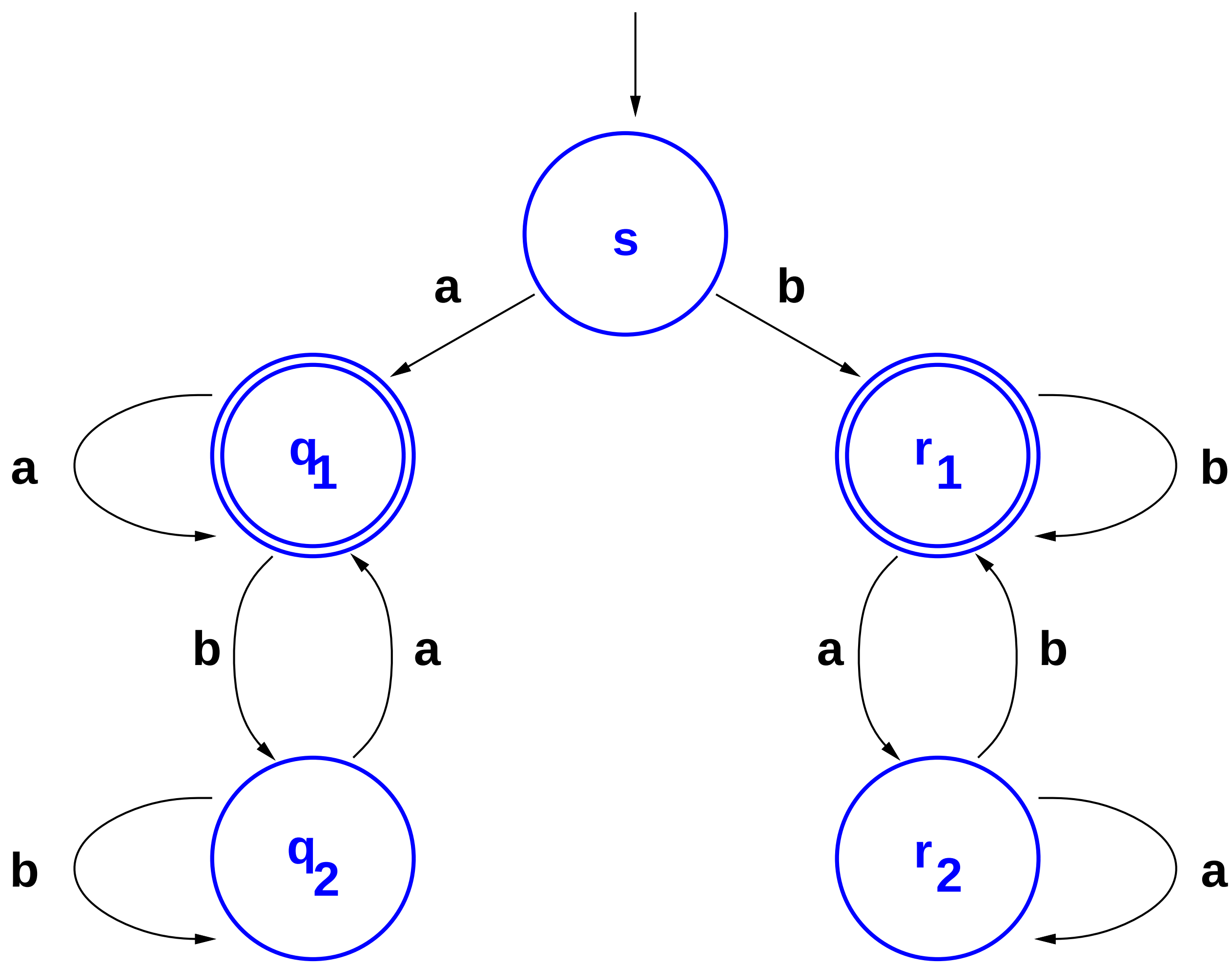
Note that

- M may accept many strings, but
- M accepts only one language

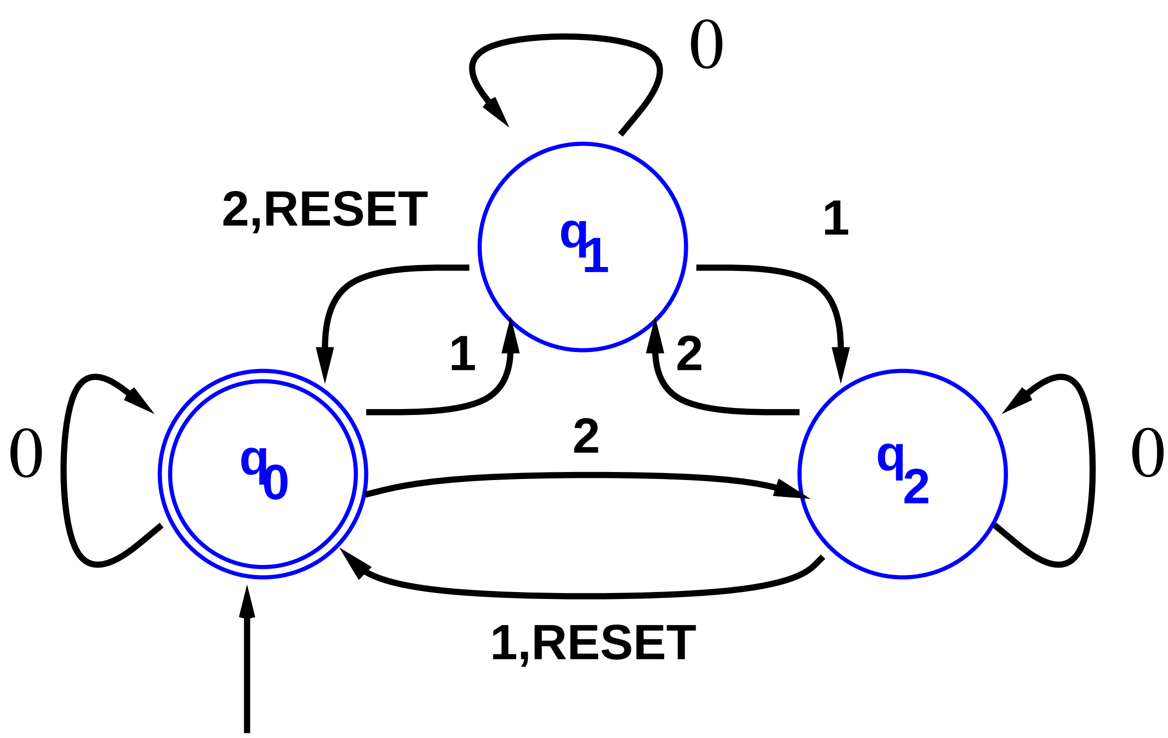
Question: What language does M accept if it accepts no strings?

Definition: A language is *regular* if some deterministic finite automaton accepts it.

Example



Example



Formal Model of Computation

Let

- $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA, and
- $w = w_1 w_2 \cdots w_n$ be a string over Σ.

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

- $r_0 = q_0$
- $\delta(r_i, w_{i+1}) = r_{i+1}$, $0 \leq i < n$
- $r_n \in F$

The Regular Operations

Let A and B be languages.

The *union* operation:

$$A \cup B = \{x | x \in A \text{ or } x \in B\}$$

The *concatenation* operation:

$$A \circ B = \{xy | x \in A \text{ and } y \in B\}$$

The *star* operation:

$$A^* = \{x_1 x_2 \ldots x_k | k \geq 0 \text{ and each } x_i \in A\}$$

The Regular Operations

Let $A = \{\text{good}, \text{bad}\}$ and $B = \{\text{boy}, \text{girl}\}$.

Union

$$A \cup B = \{\text{good}, \text{bad}, \text{boy}, \text{girl}\}$$

Concatenation

$$A \circ B = \{\text{goodboy}, \text{goodgirl}, \text{badboy}, \text{badgirl}\}$$

Star

$$A^* = \{\varepsilon, \text{good}, \text{bad}, \text{goodgood}, \text{goodbad}, \text{badbad}, \text{badgood}, \ldots\}$$

Regular Languages Closed Under Union

If A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Approach:

- some M_1 accepts A_1
- some M_2 accepts A_2
- construct M that accepts $A_1 \cup A_2$.

Idea:

- first simulate M_1, and
- if M_1 doesn't accept simulate M_2.

Question: What's wrong with this?

Instead, simulate both machines simultaneously.

Regular Languages Closed Under Union

Let

- $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accept A_1, and
- $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accept A_2.

Define M as follows:

- $Q = Q_1 \times Q_2$.
- Σ is the same.
- For each $(r_1, r_2) \in Q$ and $a \in \Sigma$,

$$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$$

- $q_0 = (q_1, q_2)$
- $F = (r_1, r_2) | r_1 \in F_1$ or $r_2 \in F_2$

Question: Why not choose $F = F_1 \times F_2$?

What About Concatenation?

Theorem: If A_1 and A_2 are regular languages, so is $A_1 \circ A_2$.

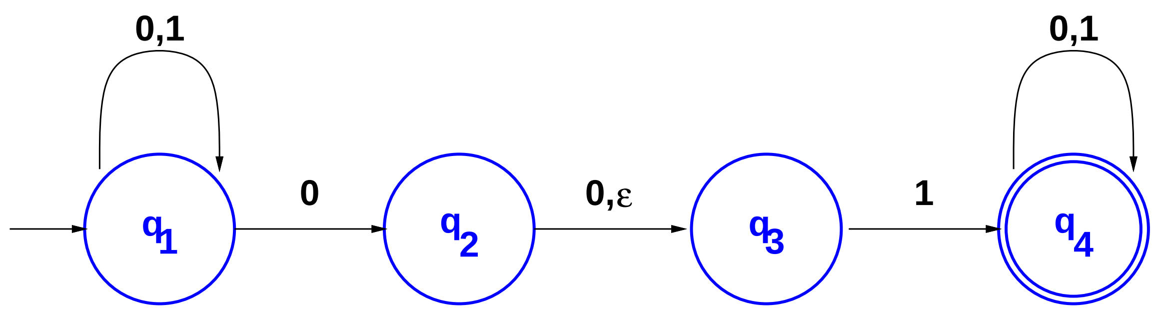
This is much harder to prove.

Idea: simulate A_1 for a while, then switch to A_2.

Question: But when do you switch?

Leads us into *non-determinism*.

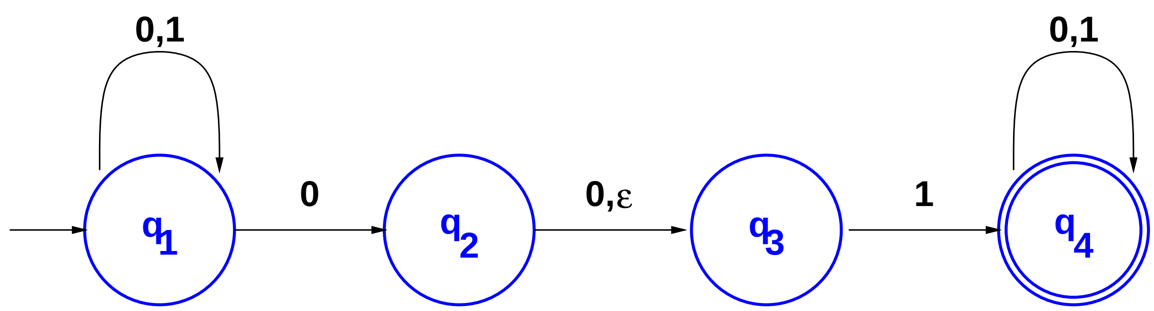
Non-Deterministic Finite Automata



Every DFA is also a non-deterministic finite automaton (NFA).

- a NFA may have more than one transition labeled with a symbol,
- a NFA may have no transitions labeled with a symbol, and
- transitions may be labeled with ε, the empty string.

Non-Deterministic Computation

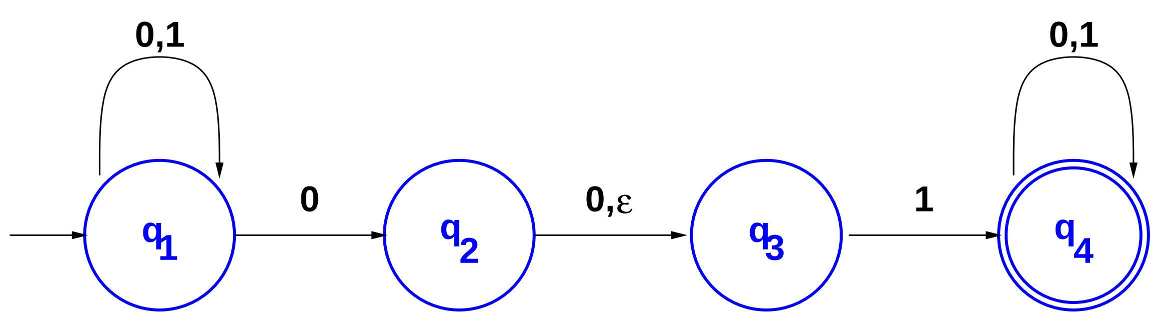


Question: What happens when more than one transition is possible?

- The machine "splits" into multiple copies
- each branch follows one possibility
- together, branches follow *all* possibilities.
- If the input doesn't appear, that branch "dies".
- Automaton accepts if *any* branch accepts.

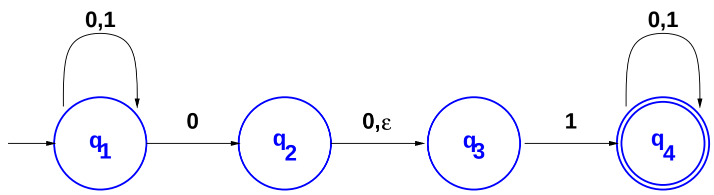
Question: What does an ε transition do?

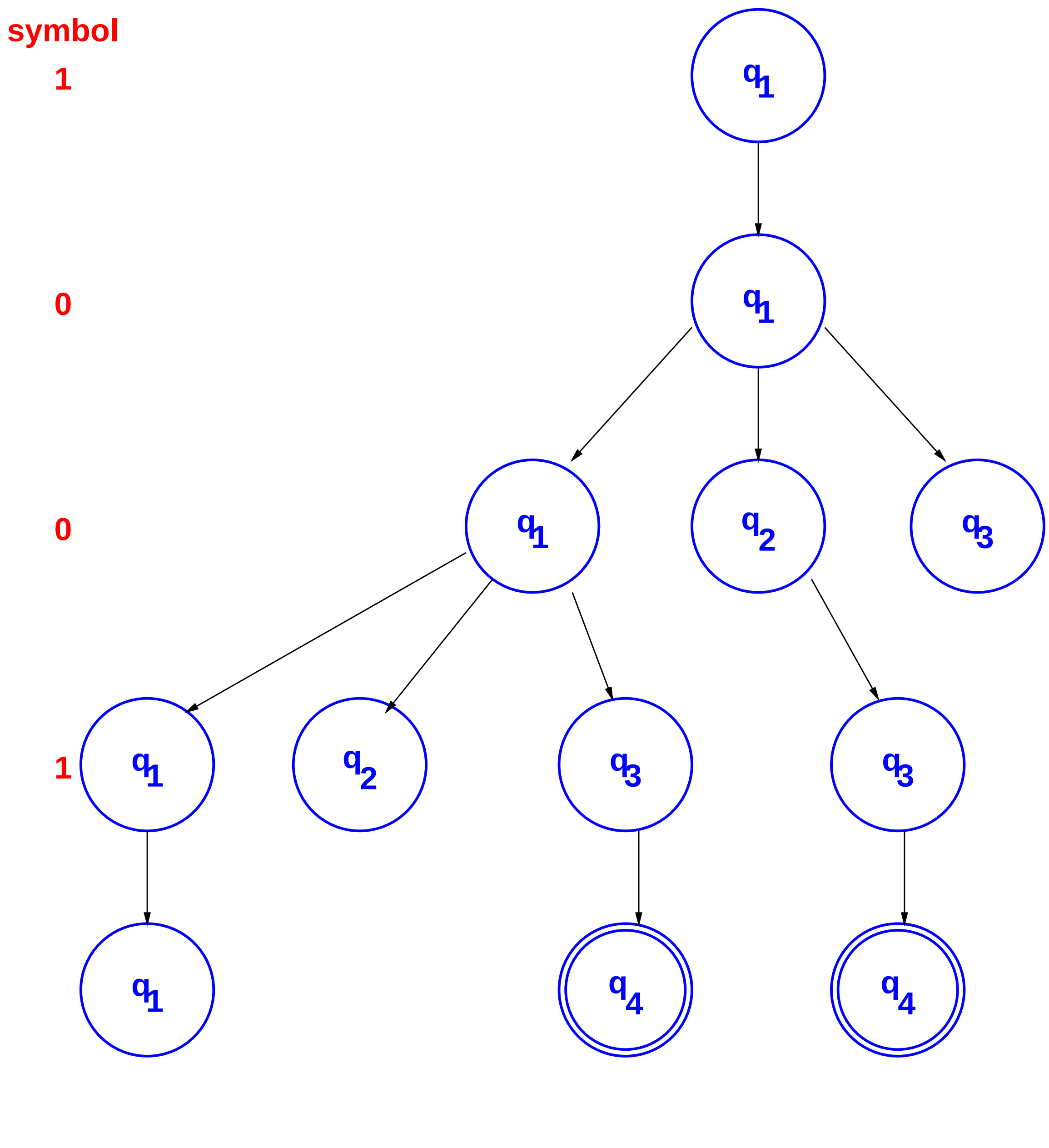
Non-Deterministic Computation



Question: What happens on string 1001?

String 1001


0,1
0,1
0
0,ε
1
q1
q2
q3
q4


symbol
1
0
0
1
q1
q1
q1
q2
q3
q1
q2
q3
q3
q1
q4
q4

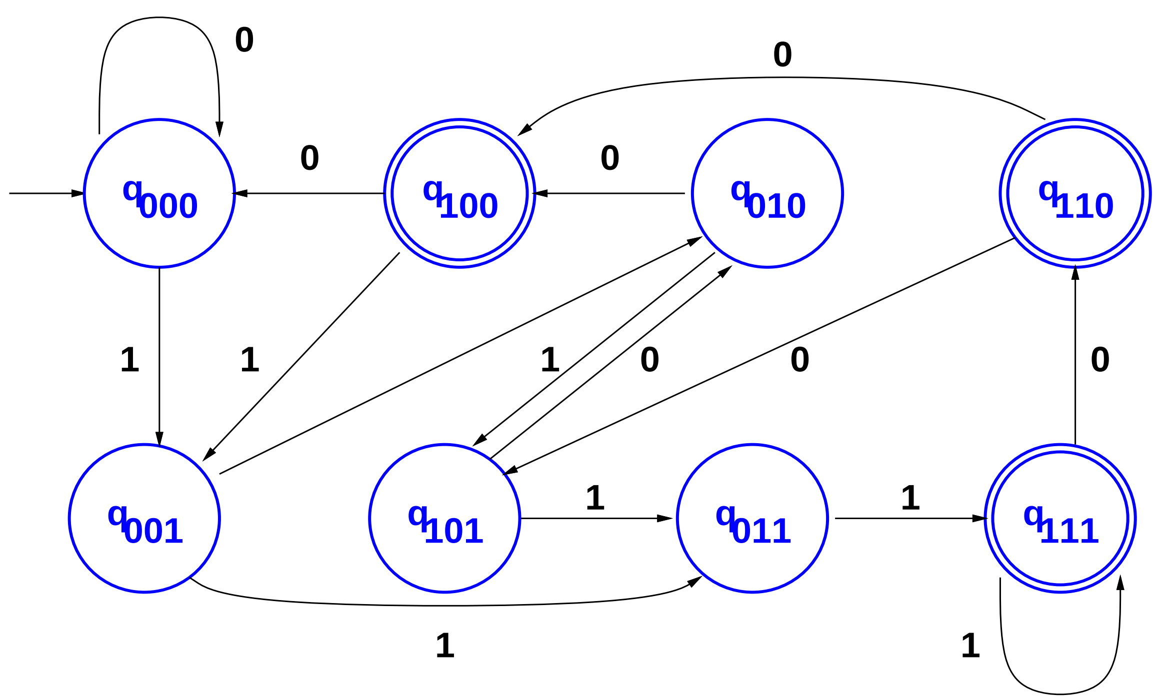
Why Non-Determinism?

Theorem: Deterministic and non-deterministic finite automata accept exactly the same set of languages.

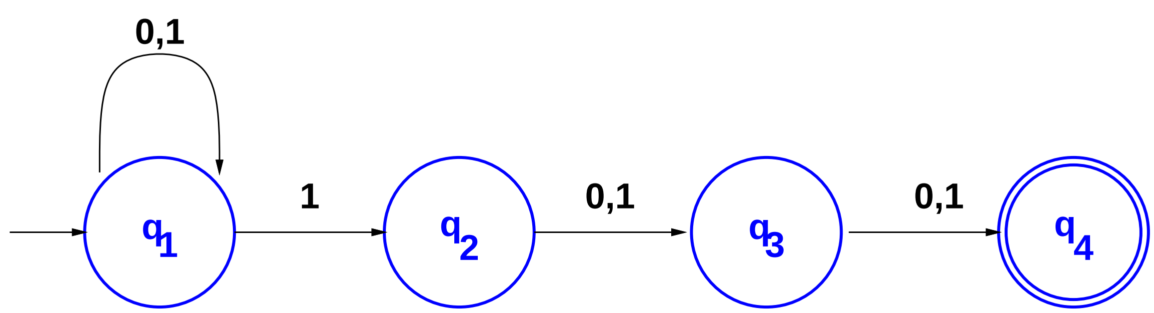
Question: So why do we need them?

Question: Can we design a finite automaton that accepts all strings with a 1 in the third-to-the-last position?

A Deterministic Automaton



A Non-Deterministic Automaton



- "Guesses" which symbol is third from the last, and
- checks that it's a 1.

Formal Definitions

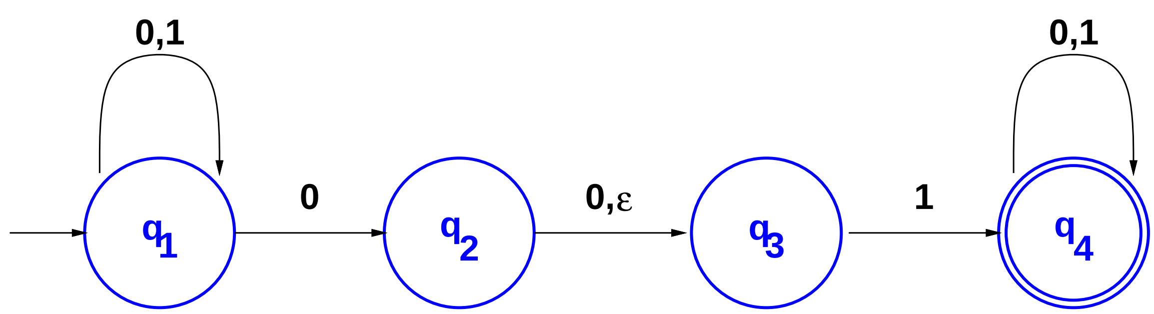
Transition function δ is going to be different.

- $\mathcal{P}(Q)$ is the powerset of Q.
- $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.

A *non-deterministic 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 *alphabet*,
- $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the *transition function*,
- $q_0 \in Q$ is the *start state*, and
- $F \subseteq Q$ is the set of *accept states*.

Example



$N_1 = (Q, \Sigma, \delta, q_1, F)$ where

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

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

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

Formal Model of Computation

Let

- $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA automaton, and
- w is a string over Σ that can be written $y_1 y_2 \cdots y_m$ where $y_i \in \Sigma_\varepsilon$.

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

- $r_0 = q_0$
- $\delta(r_i, y_{i+1}) \in r_{i+1}$, $0 \leq i < n$
- $r_n \in F$

Equivalence of NFA's and DFA's

If a language is accepted by a NFA, then we must construct a DFA that accepts the same language.

Make things easier by ignoring ε transitions (for now).

Have DFA simulate all possible NFA states.

If the NFA has k states, the DFA has 2^k states.

Equivalence of NFA's and DFA's

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA accepting A.

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

- $Q' = \mathcal{P}(Q)$.
- For $R \in Q'$ and $a \in \Sigma$, let

$$\delta'(R, a) = \{q \in Q | q \in \delta(r, a) \text{ for some } r \in R\}$$

- $q_0' = \{q_0\}$.
- $F' = \{R \in Q' | R \text{ contains an accept state of } N\}$

ε-Transitions

For any state R of M, define $E(R)$ to be the collection of states reachable from R by ε transitions only.

$E(R) = \{q | q$ can be reached from R by 0 or more ε transitions$\}$

Now define transition:

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

Change start state to

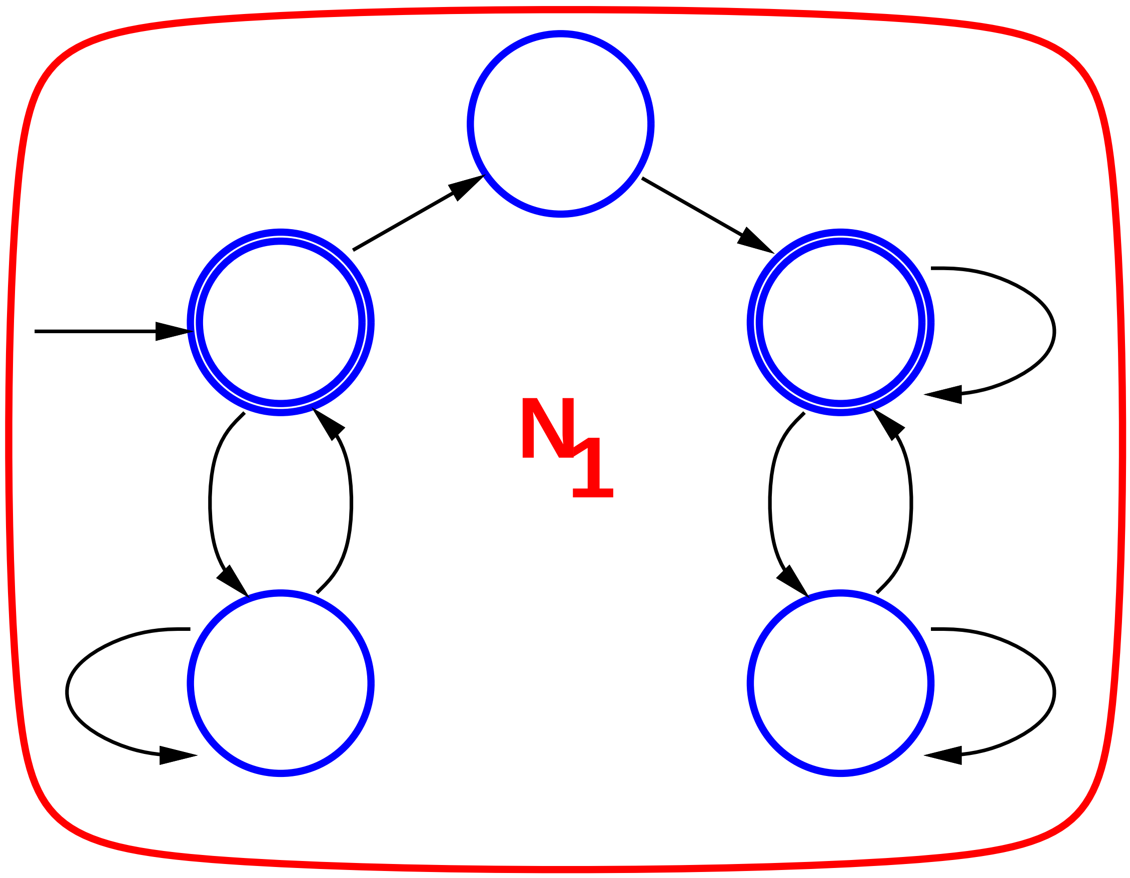
$$q_0' = E(\{q_0\})$$

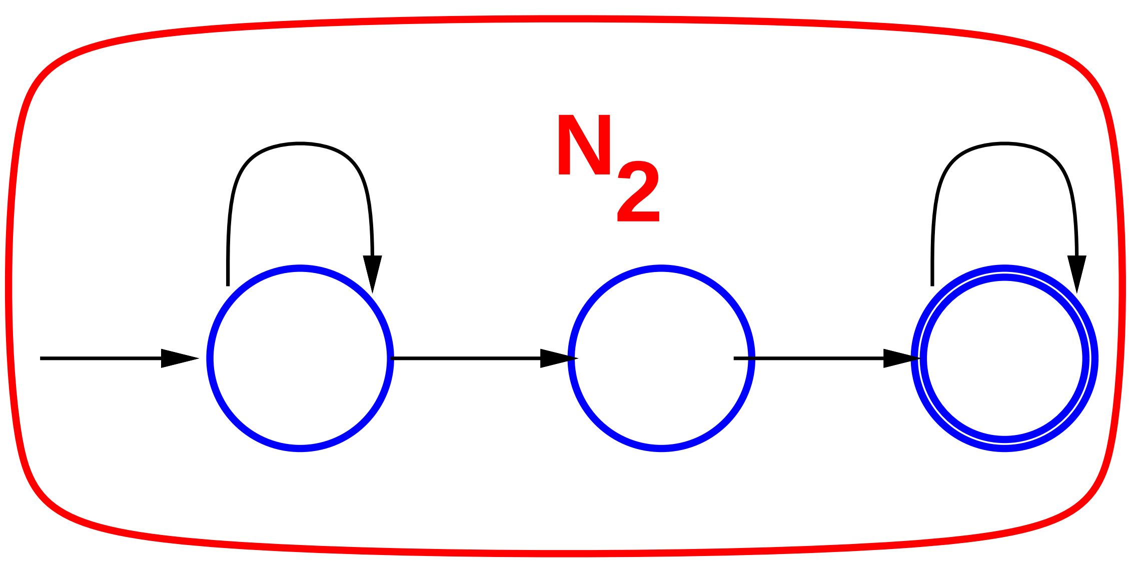
Corollary

A language is regular if and only if some NFA accepts it.

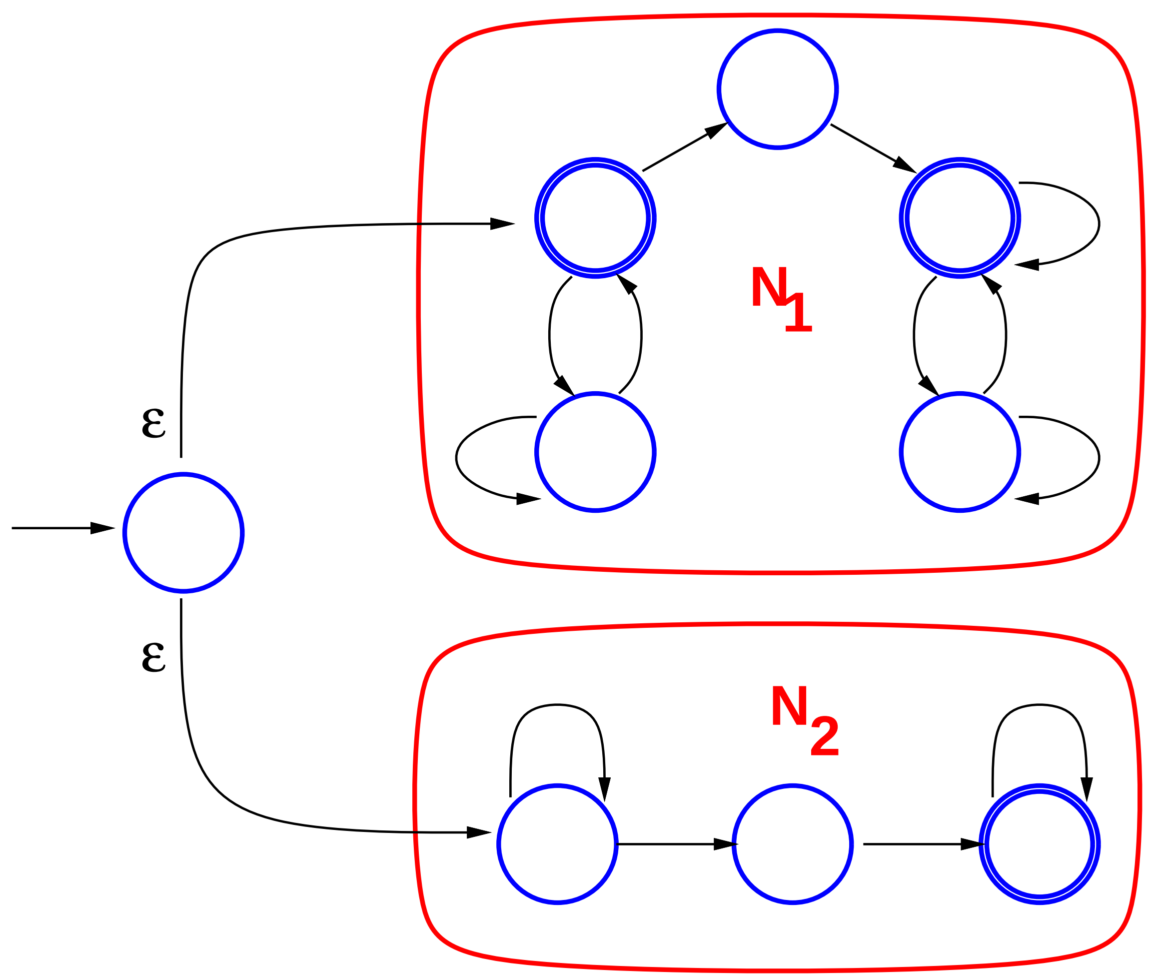
This is an alternative way of characterizing regular languages.

Regular Languages Closed Under Union





Regular Languages Closed Under Union



Regular Languages Closed Under Union

Let

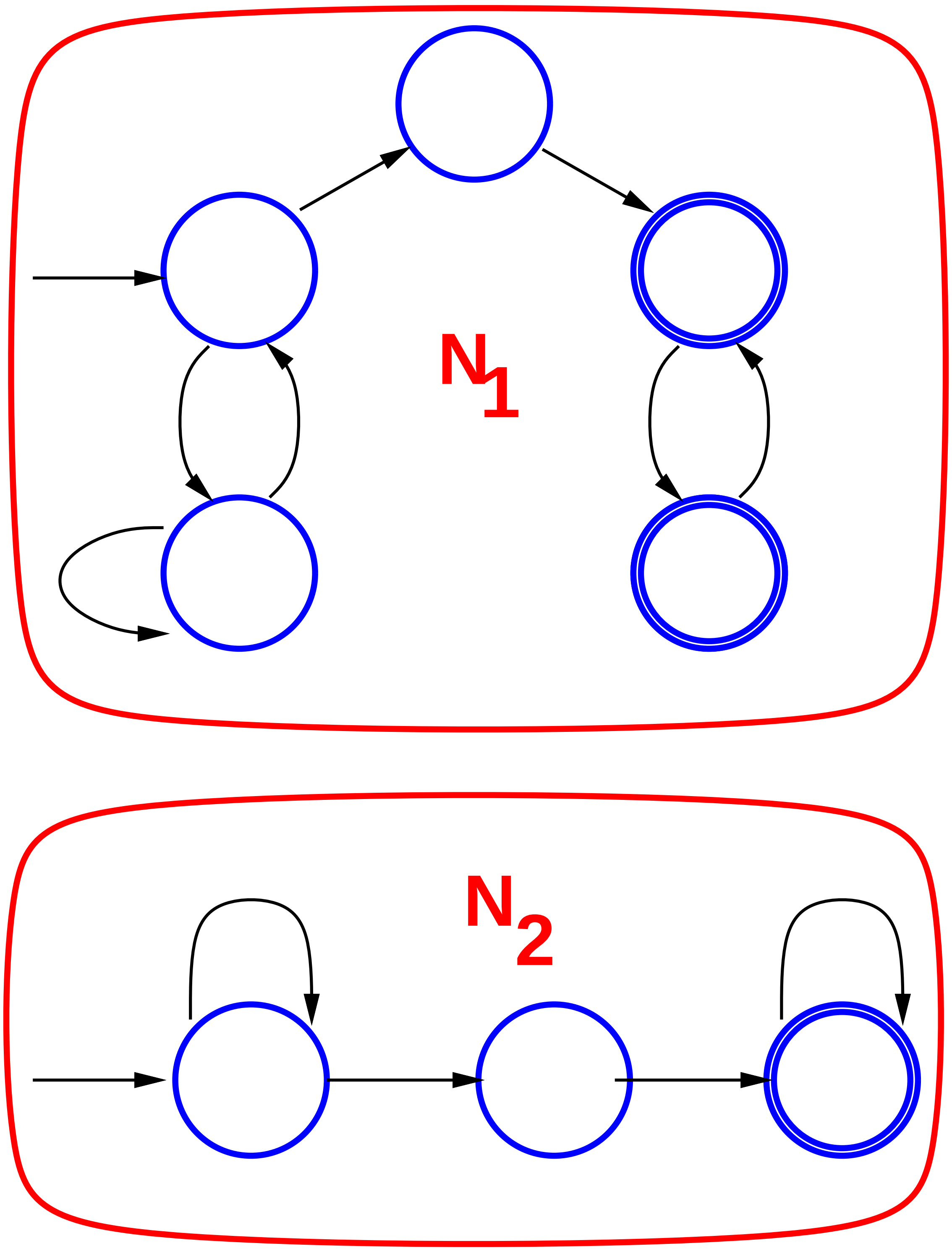
- $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accept A_1, and
- $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accept A_2.

Define $N = (Q, \Sigma, \delta, q_0, F)$:

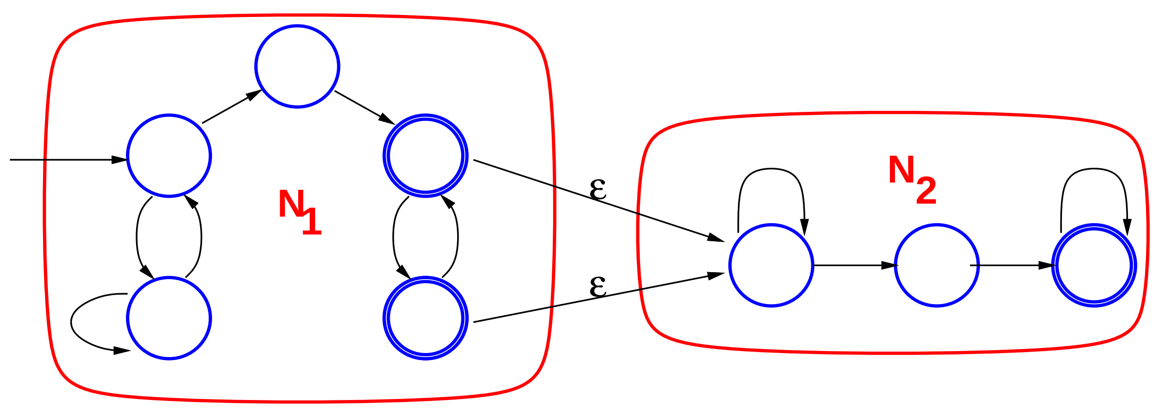
- $Q = q_0 \cup Q_1 \cup Q_2$
- Σ is the same.
- q_0 is the start state
- $F = F_1 \cup F_2$
-

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

Regular Languages Closed Under Concatenation



Regular Languages Closed Under Concatenation



Regular Languages Closed Under Concatenation

Let

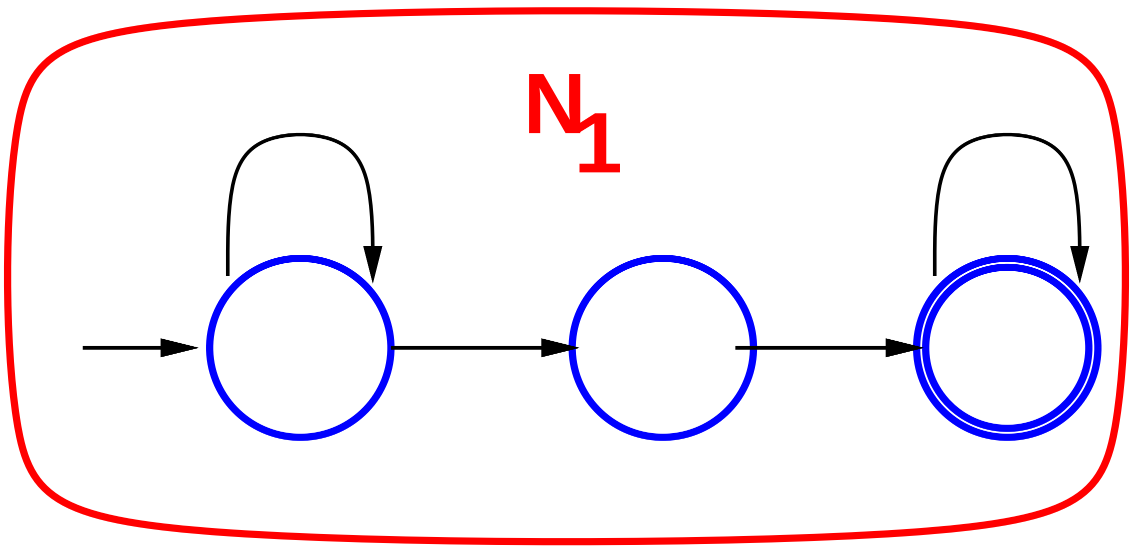
- $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accept A_1, and
- $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accept A_2.

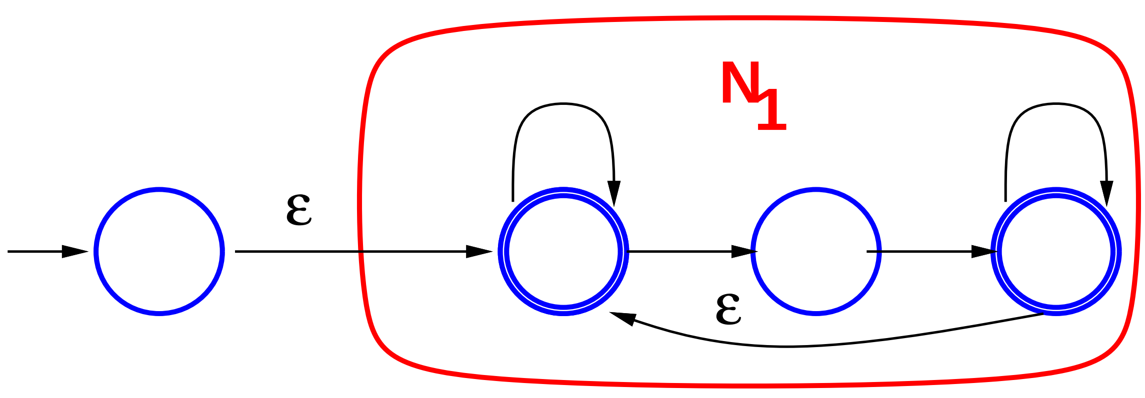
Define $N = (Q, \Sigma, \delta, q_1, F_2)$:

- $Q = Q_1 \cup Q_2$
- q_1 is the start state of N
- F_2 is the set of accept states of N

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

Regular Languages Closed Under Star





Regular Languages Closed Under Star

Let

- $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accept A_1, and

Define $N = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_0\} \cup Q_1$
- q_0 is the new start state.
- $F = \{q_0 \cup F_1\}$

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

Summary

- regular languages are closed under
  - union
  - concatenation
  - star
- non-deterministic finite automata
  - are equivalent to deterministic finite automata
  - but much easier to use in some proofs.

Regular Expressions

A notation for building up languages.

$$(0 \cup 1)0^*$$

- 0 and 1 shorthand for $\{0\}$ and $\{1\}$
- so $(0 \cup 1) = \{0, 1\}$.
- 0^* is $\{0\}^*$.
- concatenation, like multiplication, is implicit.

So $(0 \cup 1)0^*$ is the language of all strings starting with 0 or 1 followed by any number of 0's.

Often used in text editors or shell scripts.

More Examples

Let Σ be an alphabet.

- The regular expression Σ is language of one-symbol strings.
- Σ^* is all strings.
- $\Sigma^* 1$ all strings ending in 1.
- $0\Sigma^* \cup \Sigma^* 1$ strings either starting in 0 or ending in 1.

Operations have precedence:

- star first
- concatenation next
- union last
- parentheses used to change usual order

Formal Definition of Regular Expressions

Syntax: R is a *regular expression* if R is

- a for some $a \in \Sigma$
- ε
- $\emptyset$
- $(R_1 \cup R_2)$ for regular expressions R_1 and R_2
- $(R_1 \circ R_2)$ for regular expressions R_1 and R_2
- (R_1^*) for regular expression R_1

Formal Definition of Regular Expressions

Let $L(R)$ be the language denoted by regular expression R.

R	$L(R)$
a	$\{a\}$
ε	$\{\varepsilon\}$
$(R_1 \cup R_2)$	$L(R_1) \cup L(R_2)$
$(R_1 \circ R_2)$	$L(R_1) \circ L(R_2)$
$(R_1)^*$	$L(R_1)^*$

Question: What's the difference between $\emptyset$ and ε?

Question: Why isn't this definition circular?

Remarkable Fact

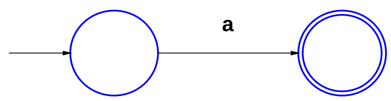
Theorem: If a language is described by a regular expression, then it is regular.

Step One: construct a NFA accepting R.

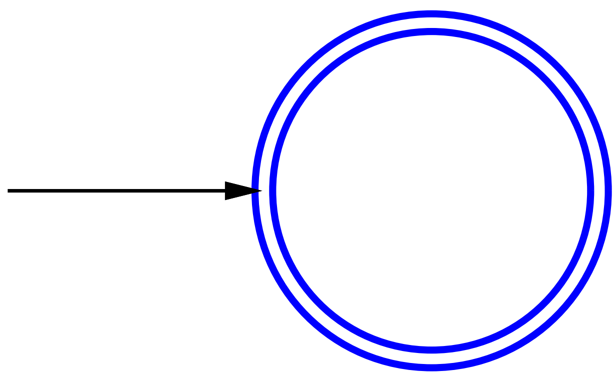
Step Two: Given a regular language, construct an equivalent regular expression.

NFA Accepting Regular Expression

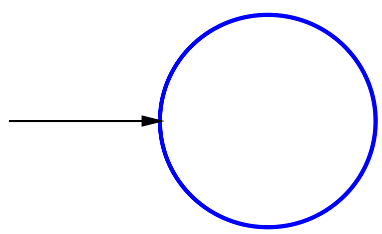
$R = a \in \Sigma$



$R = \varepsilon$

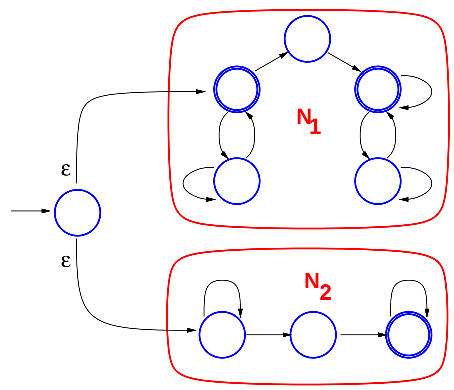


$R = \emptyset$

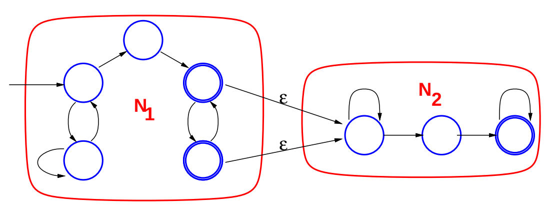


NFA Accepting Regular Expression

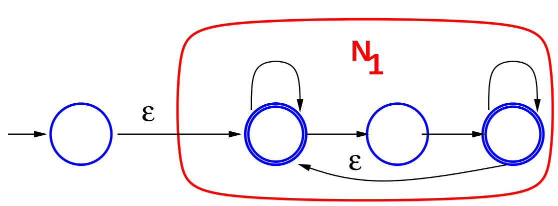
$(R_1 \cup R_2)$:



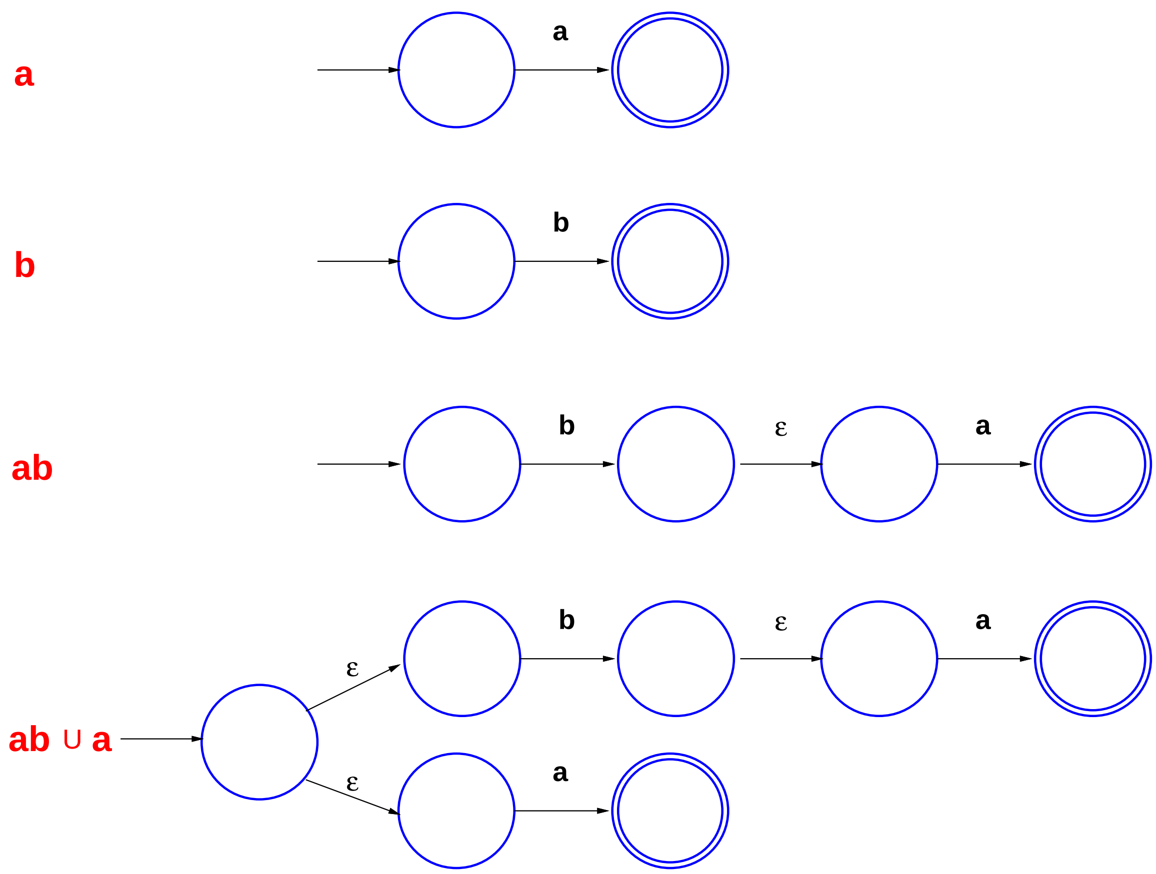
$(R_1 \circ R_2)$:



$(R_1)^*$:



Example



Regular Expression from a NFA

Define *generalized non-deterministic finite automata* (GNFA).

An NFA:

- Each transition labeled with a symbol or ε,
- reads zero or one symbols,
- takes matching transition, if any.

A GNFA:

- Each transition labeled with a *regular expression*,
- reads *zero or more* symbols,
- takes transition whose regular expression matches string, if any.

Natural generalization of NFA's.

GNFA Special Form

- Start state has outgoing arrows to every other state, but no incoming arrows.
- Unique accept state has incoming arrows from every other state, but no outgoing arrows
- Except for start and accept states, arrows goes from every state to every other state, including itself.

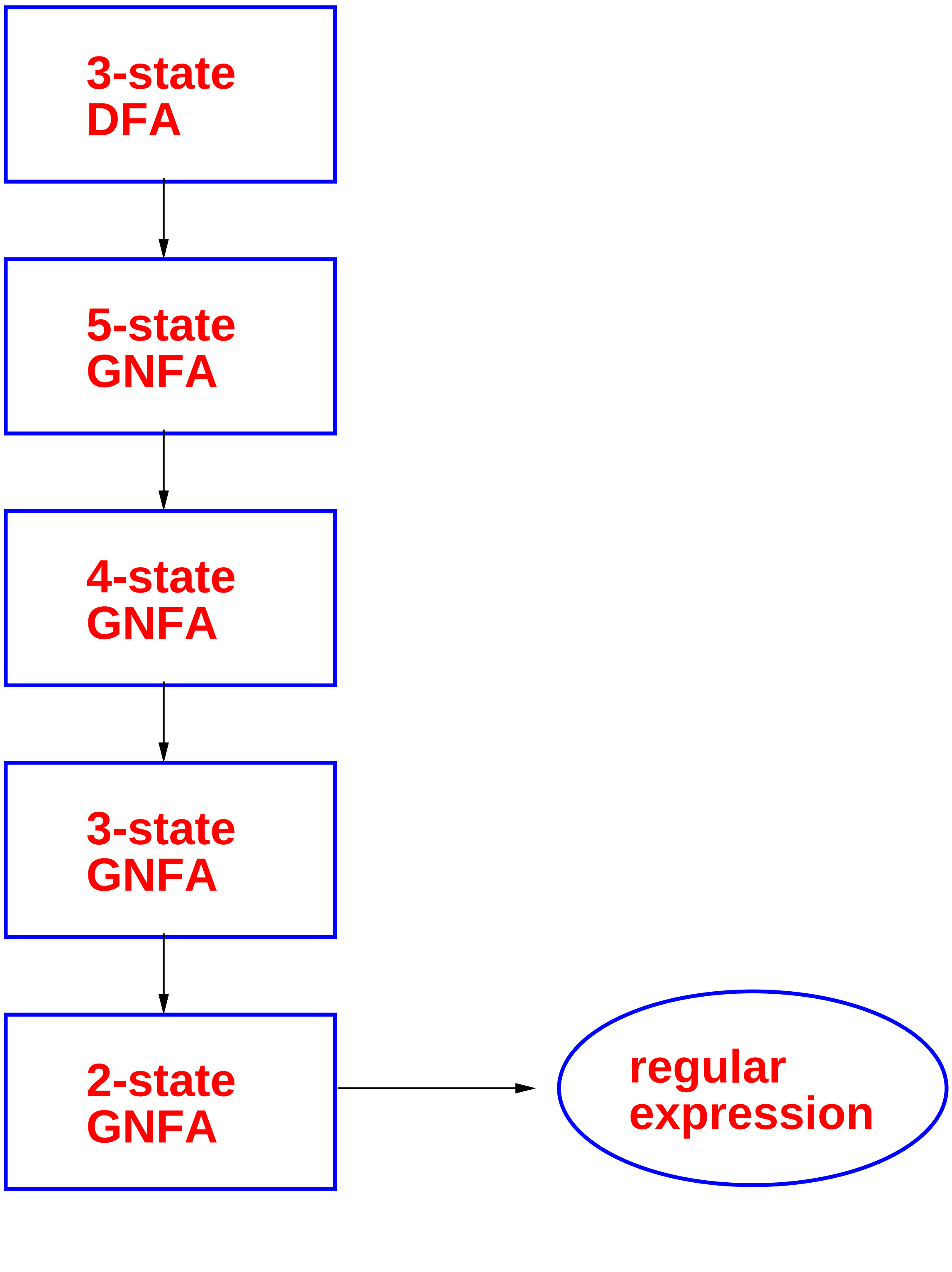
Easy to transform any GNFA into special form.

Question: How?

Strategy

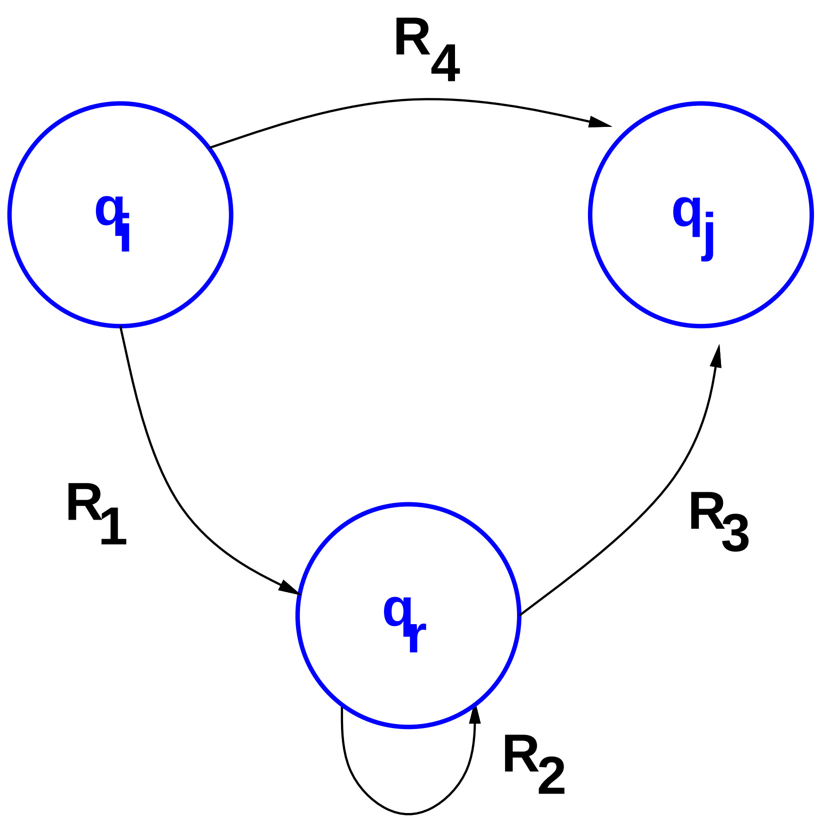
- given a k-state DFA
- transform into $(k+2)$-state GNFA
- transform GNFA into equivalent GNFA with one fewer state
- eventually reach 2-state GNFA
- label on single transition is desired regular expression.

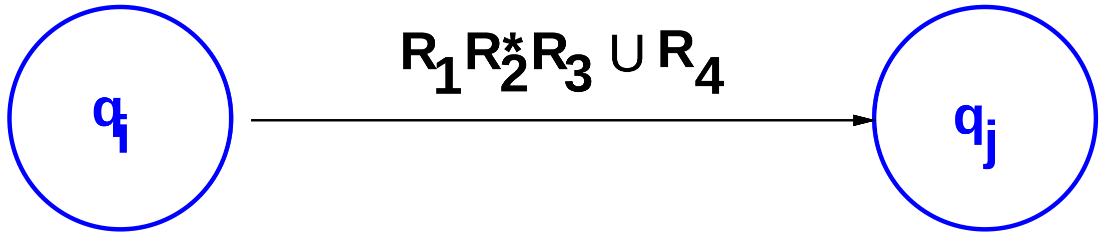
Strategy


3-state DFA
5-state GNFA
4-state GNFA
3-state GNFA
2-state GNFA
regular expression

More Strategy

We remove a state q_r, and then repair the machine by altering regular expression of other transitions.





Formal Treatment

- q_s is start state.
- q_a is accept state.
- $\mathcal{R}$ is collection of regular expressions over Σ.

Transition function is:

$$\delta : (Q - \{q_a\}) \times (Q - setq_s) \to \mathcal{R}$$

Arrows connects every state to every other state except:

- no arrow from q_a
- no arrow to q_s

If $\delta(q_i, q_j) = R$, then the arrow from q_i to q_j has label R.

Formal Definitions

A generalized deterministic finite automaton (GDFA) is $(Q, \Sigma, \delta, q_s, q_a)$, where

- Q is a finite set of *states*,
- Σ is the *alphabet*,
- $\delta : (Q - \{q_a\}) \times (Q - setq_s) \to \mathcal{R}$ is the *transition function*.
- $q_s \in Q$ is the *start state*, and
- $q_a \in Q$ is the unique *accept state*.

Formal Model of Computation

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

- $q_0 = q_s$, the start state,
- $q_k = q_a$, the accept state, and
- for each i, $w_i \in L(R_i)$, where $R_i = \delta(q_{i-1}, q_i)$.

The CONVERT Algorithm

Given a DFA M for language A, convert it to GNFA G.

Define procedure CONVERT(G).

1. let k be the number of states of G.
2. If $k = 2$, return the regular expression labeling the only arrow.
3. If $k > 2$, select any q_r distinct from q_s and q_a. Let

$$Q' = Q - \{q_r\}.$$

   For any $q_i \in Q' - \{q_a\}$ and $q_j \in Q' - \{q_s\}$, let
   - $R_1 = \delta(q_i, q_r)$,
   - $R_2 = \delta(q_r, q_r)$,
   - $R_3 = \delta(q_r, q_j)$, and
   - $R_4 = \delta(q_i, q_j)$.

   Define $\delta'(q_i, q_j) = (R_1)(R_2) * (R_3) \cup (R_4)$.
4. compute CONVERT(G') and return this value.

Proof

By induction, of course.

Induct on number of states

Basis: When there are only 2 states, there is a single label, which characterizes the strings accepted.

Induction Step: Assume claim for $k-1$ states. Three steps:

1. If G accepts w, then so does G'.
2. If G' accepts w, then so does G.
3. Therefore CONVERT(G') is correct result

Theorem: If G accepts w, then so does G'.

If G accepts w, then it traverses states

$$q_s, q_1, q_2, \ldots, q_a$$

If q_r does not appear, then G' also accepts w, because new regular expression contains old regular expression in union.

If q_r appears, then consider the subsequence:

$$\ldots q_i, q_r, \ldots, q_r, q_j \ldots$$

The new regular expression linking q_i and q_j encompasses any such string.

Either way, the claim holds.

Theorem: If G' accepts w, then so does G.

Each transition from q_i to q_j in G' encompasses a transition in G, either directly or through q_r.

G' has $k-1$ states, so by the induction hypothesis, CONVERT(G') returns a regular expression equivalent to $L(G')$.

Since $L(G) = L(G')$, the algorithm is correct.

Negative Results

We have made a lot of progress understanding what finite automata can do.

What *can't* they do?

Is there a DFA that accepts

- $B = \{0^n 1^n | n \geq 0\}$
- $C = \{w | w \text{has an equal number of 0's and 1's}\}$
- $D =$
  $\{w | w \text{has an equal number of occurrences of 01 and 10 substrings}\}$

Consider B:

- machine must "remember" how many 0's it has seen
- impossible with finite state.

The others are exactly the same.

Question: Is this a proof?

Answer: No. D is regular!

Pumping Lemma

We will show that all regular languages have a special property.

If a string is longer than a certain critical length ℓ, then it can be "pumped" to a larger length by repeating an internal substring.

This is a powerful technique for showing that a language is *not* regular.

Pumping Lemma

Theorem: If A is a regular language, then there is an $\ell > 0$ where if $s \in A$ and $|s| > \ell$, then $s = xyz$ such that

- for $i > 0$, $xy^iz \in A$,
- $|y| > 0$, and
- $|xy| \leq \ell$.

Pumping Lemma Proof

Let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA that accepts A.

Let ℓ be $|Q|$, the number of states of M.

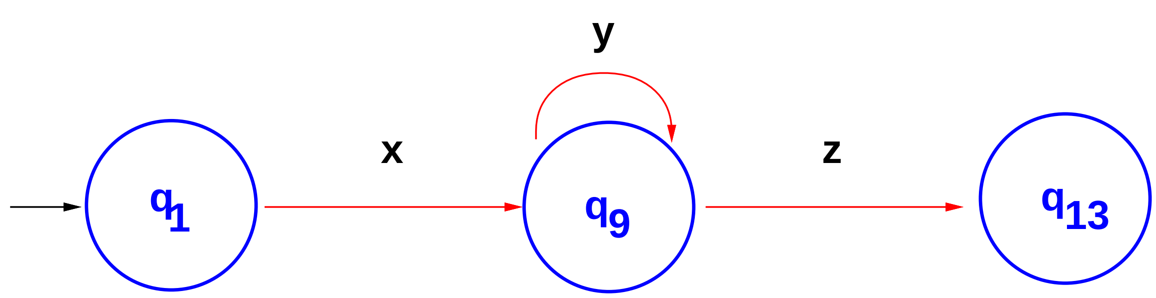
If $s \in A$ has length at least ℓ, consider the sequence of states:

$$s = \begin{array}{cccccccccccccc} s_1 & & s_2 & & s_3 & & s_4 & & s_5 & & s_6 & \cdots & & s_n \\ & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & & \uparrow & \\ & q_1 & & q_{20} & & q_9 & & q_{17} & & q_9 & & & q_{13} & \end{array}$$

Since there are more then ℓ states, at least one is repeated. *pigeonhole principle*.

Pumping Lemma Proof

We can divide $s = xyz$, where



- from inspection, M accepts $xy^k z$.
- $|y| > 0$ because state is repeated.
- pick first state repetition to ensure that $|xy| < \ell$.

An Application

Theorem: $B = \{0^n 1^n | n > 0\}$ is not regular.

Proof by contradiction.

Otherwise, let ℓ be the critical value.

Consider $s = 0^\ell 1^\ell$.

Theorem says $s = xyz$. where $xy^i z \in B$.

- If y is all 0, then too many 0's.
- If y is all 1, then too many 0's.
- If y is mixed, then string is out of order.

Another Application

Theorem:

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

is not regular.

Proof by contradiction.

Otherwise, let ℓ be the critical value.

Consider $s = 0^\ell 1^\ell$.

Theorem says $s = xyz$. where $xy^i z \in B$.

- If y is all 0, then too many 0's.
- If y is all 1, then too many 0's.
- If y is mixed, then $|xy| \leq \ell$, so y consists entirely of 0's, and contradiction follows.

Algorithms for NDA's

Question: Given NDA N and a string s, is $s \in L(N)$?

Answer: Construct the DFA equivalent to N and run it on w.

Question: Is $L(N) = \emptyset$?

Answer: By Pumping Lemma, suffices to check all strings of length less than the number of states of N.

Algorithms for NDA's

Question: Is $L(N) = \Sigma^*$?

Answer: Check if $\overline{L(M)} = \emptyset$.

Question: Given N_1 and N_2, is $L(N_1) \subseteq L(N_2)$?

Answer: Check if $\overline{L(N_2)} \cap L(M_1) = \emptyset$.

Question: Given N_1 and N_2, is $L(N_1) = L(N_2)$?

Answer: Check if $L(N_1) \subseteq L(N_2)$ and $L(N_2) \subseteq L(N_1)$.