Regular Languages

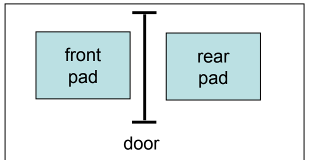
Andreas Karwath & Malte Helmert

Overview

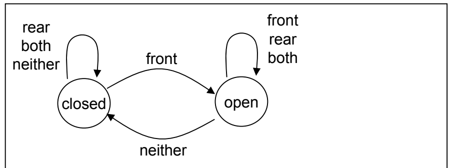
- Deterministic finite automata
- Regular languages
- Nondeterministic finite automata
- Closure operations
- Regular expressions
- Nonregular languages
- The pumping lemma

Finite Automata

- An intuitive example : supermarket door controller



Top view of an automatic door



State diagram for the automatic door controler

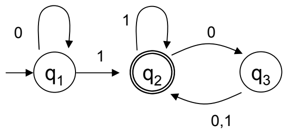
- Probabilistic counterparts exist
  - Markov chains, Bayesian nets, etc.
  - Not in this course

Transition table for the automatic door controler:

	neither	front	rear	both
closed	closed	open	closed	closed
open	closed	open	open	open

A finite automaton

- Figure 1.4



States : q_1, q_2, q_3

Startstate : q_1

Acceptstate : q_2

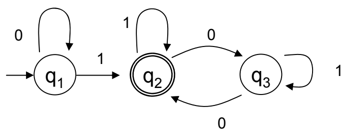
Transitions

Output : *accept* or *reject*

- Formally

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

1. Q is a finite set of states
2. Σ is a finite set, the alphabet
3. $\delta : Q \times \Sigma \to Q$ is the transition function
4. $q_o \in Q$ is the start state
5. $F \subseteq Q$ is the set of accept states



Describe M_1

$Q = \{q_1, q_2, q_3\}$

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

δ defined by

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

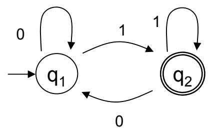
q_1 start state

$F = \{q_2\}$

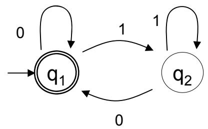
A is the language of machine M

we write $L(M) = A$

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



State diagram of the two-state finite automaton M_2



State diagram of the two-state finite automaton M_3

Other examples

- 7,8,9

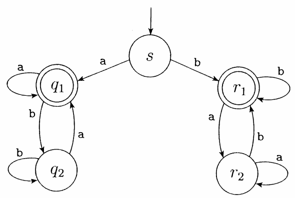


FIGURE 1.8
Finite automaton M_4

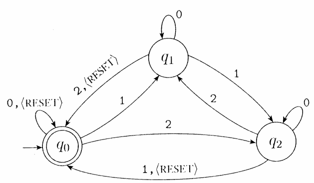


FIGURE 1.9
Finite automaton M_5

Another example

A generalisation : A_i is the language of all strings where the sum of the numbers is a multiple of i except that the sum is reset to 0 whenever the symbol $\langle reset \rangle$ appears

Automaton B_i =

1. $Q_i = \{q_0, ..., q_{i-1}\}$
2. $\Sigma = \{0,1,2,\langle reset \rangle\}$
3. $\delta(q_j, 0) = q_j$

   $\delta(q_j, 1) = q_k$ where $k = (j+1) \bmod i$

   $\delta(q_j, 2) = q_k$ where $k = (j+2) \bmod i$

   $\delta(q_j, \langle reset \rangle) = q_0$
4. $q_o \in Q$ is start and accept state

Formal definition of computation

Let M be a finite automaton $(Q, \Sigma, \delta, q_0, F)$

Let $w = w_1 \ldots w_n$ be a string over Σ

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

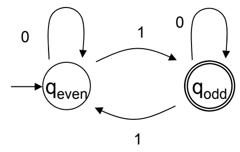
1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for all $i = 0, \ldots, n-1$
3. $r_n \in F$

M recognizes language A if $A = \{w \mid M \text{ accepts } w\}$

A language is regular if some finite automaton recognizes it.

Designing finite automata

- Design automaton for language consisting of binary strings with an odd number of 1s
- Design first states
- Then transitions
- Start state and accept states

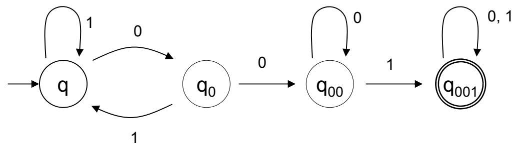


Another example

- Design an automaton to recognize the language of binary strings containing the string 001 as substring
- We have four possibilities:
  1. we haven't seen any symbol of the pattern yet, or
  2. we have seen a 0, or
  3. we have seen a 00, or
  4. we have seen the pattern 001

Another example

- Design an automaton to recognize the language of binary strings containing the string 001 as substring
- We have four possibilities:
  1. we haven't seen any symbol of the pattern yet, or
  2. we have seen a 0, or
  3. we have seen a 00, or
  4. we have seen the pattern 001



The Regular Operations

Let A and B be languages

We define :

Union : $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$

Concatenation : $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$

Star : $A^* = \{x_1 x_2 \ldots x_n \mid n \geq 0 \text{ and each } x_i \in A\}$

note: always $\varepsilon \in A^*$

Example

$A = \{good, bad\}$

$B = \{boy, girl\}$

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

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

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

Regular languages are closed under …

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 1.12

The class of the 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 1.12 (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 $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

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

1. $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$.
   This set is the **Cartesian product** of sets Q_1 and Q_2 (written $Q_1 \times Q_2$).
   It is the set of all pairs of states, the first from Q_1 and the second from Q_2.
2. Σ, the alphabet, is the same as in 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 $\Sigma = \Sigma_1 \cup \Sigma_2$.

3. δ, the transition function, is defined as follows. For each $(r_1, r_2) \in Q$ and each $a \in \Sigma$, let
   $$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a)).$$
   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 either member is an accept state of M_1 and M_2.
   We can write it as
   $$F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}.$$
   This expression is the same as $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$.

Note that it is not the same as $F = F_1 \times F_2$. What would that give us?

Example

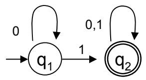
$M = (Q, \Sigma, \delta, q, F)$

constructed from $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$

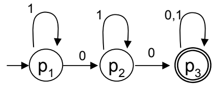
Define

1. $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$
2. $\Sigma = \Sigma_1 \cup \Sigma_2$
3. $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
4. $q = (q_1, q_2)$
5. $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$

M_1 with $L(M_1) = \{w \mid w$ contains a 1$\}$



M_2 with $L(M_2) = \{w \mid w$ contains at least two 0s$\}$



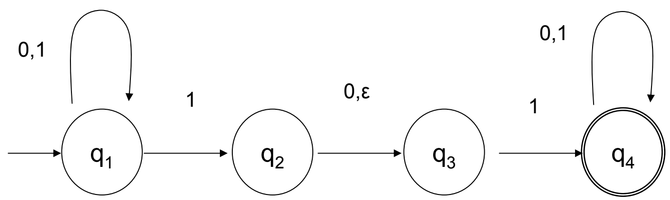
Theorem 1.13

The class of the 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

- Deterministic
  - One successor state
  - ε transitions not allowed
- Non deterministic
  - *Several* successor states possible
  - ε transitions possible



Deterministic versus non deterministic computation

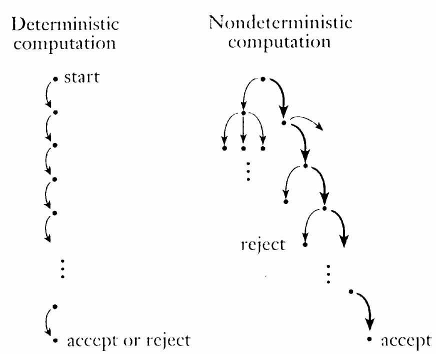
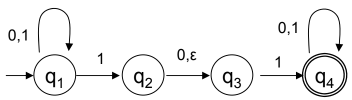
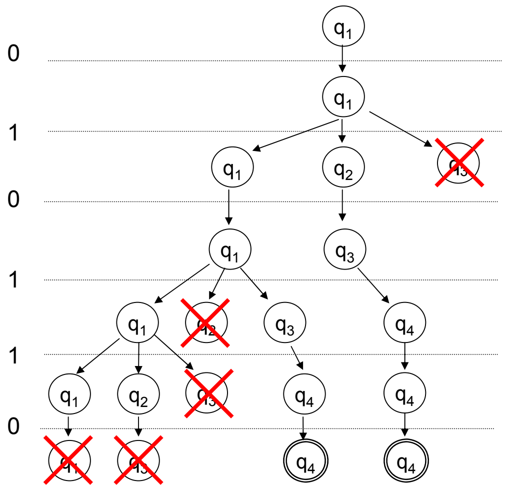


FIGURE 1.15
Deterministic and nondeterministic computations with an accepting branch



Input: w = 010110



Another NFA

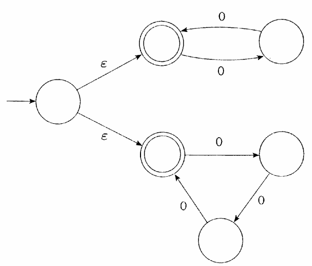


FIGURE 1.19
The NFA N_3

Nondeterministic finite automaton

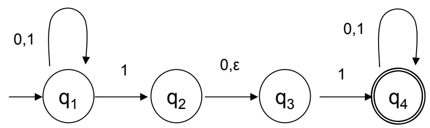
A nondeterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$

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

Σ_ε includes ε

$P(Q)$ the powerset of Q

Example



1. $Q = \{q_1, q_2, q_3, q_4\}$
2. $\Sigma = \{0,1\}$
3. δ is given as:

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

4. q_1 is the start state
5. $F = \{q_4\}$

Formal definition of computation

Let M be a nondeterministic finite automaton (Q,Σ,δ,q_0,F)

Let $w = w_1....w_n$ be a string over Σ

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

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

Every NFA has an equivalent DFA

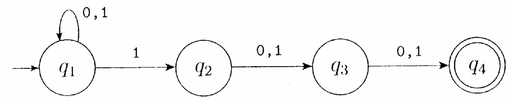


FIGURE 1.17
The NFA N_2 recognizing A

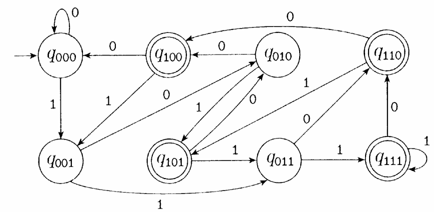


FIGURE 1.18
A DFA recognizing A

Equivalence NFA and DFA

Two machines are *equivalent* if they recognize the same language

Theorem 1.19
Every nondeterministic finite automaton has an equivalent finite automaton

Corollary 1.20
A language is regular if and only if some nondeterministic finite automaton recognizes it.

Proof: Theorem 1.19

Let $N = (Q,\Sigma,\delta_0,q_0,F)$ be the NFA recognizing some language A.
Construct a DFA M recognizing A.
First we consider the easier case wherein N has no ε arrows. The ε arrows are taken into account later.

Proof: Theorem 1.19 (cont.)

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 shows where a takes each state in R. Because each state leads to to a set of states, we take the union of all these sets. Alternativly we 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.
4. $F' = \{ R \in Q' \mid R$ contains an accept state of $N \}$.
   The machine M accepts if one of the possible states that N could be in at this point is an accept state.

Proof: Theorem 1.19 (cont.)

Now for the ε arrows one needs to set up an extra bit of notation.
For any state R of M we define $E(R)$ to be the collection of states that can be reached from R by going only along ε arrows, including the members of R themselves. Formally, for $R \subseteq Q$ let

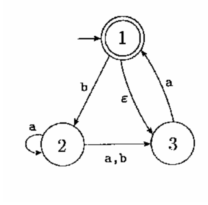
$E(R) = \{ q \mid q$ can be reached from R by traveling along 0 or more ε arrows $\}$.

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

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

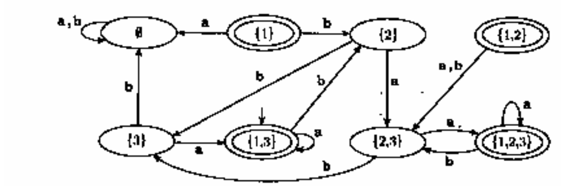
Additionally the start state of M has to be modified to cater for all possible states that can be reached from the start state of N along the ε arrows.
Changing q_0' to be $E(\{ q_0 \})$ achieves this effect.
We have now completed the construction of the DFA M that simulates the NFA N.

An example

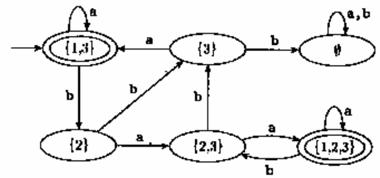


An example

The resulting DFA



The resulting DFA after removing redundant states



Closure under the regular operations

Theorem 1.12/1.22

The class of the 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 1.23

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

Theorem 1.24

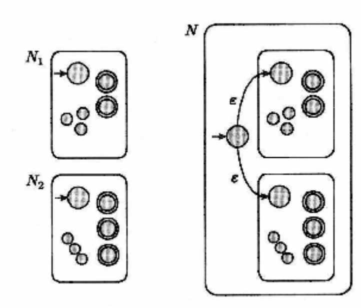
The class of the regular languages is closed under the star operation.

Proof idea

Theorem 1.12/1.22

The class of the 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 1.12/1.22

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 $A_1 \cup A_2$.

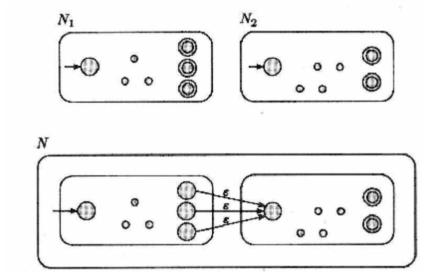
1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.
   The states of N are all the states of N_1 and N_2, with the addition of a new start state q_0.
2. The state q_0 is the start state of N.
3. The accept states $F = F_1 \cup F_2$.
   The accept states of N are all the accept states of N_1 and N_2. That way N accepts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

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

Proof idea

Theorem 1.23

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



Proof 1.23

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.

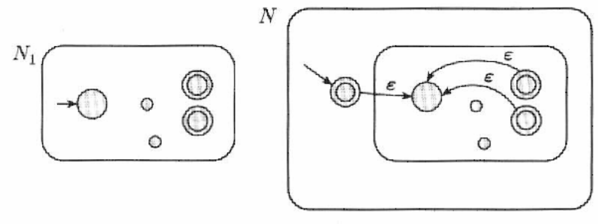
Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$.

1. $Q = Q_1 \cup Q_2$.
   The states of N are all the states of N_1 and N_2.
2. The state q_1 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 $a \in \Sigma_\varepsilon$,

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

Proof idea

Theorem 1.24
The class of the regular languages is closed under the star operation.



Proof 1.24

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

1. $Q = \{q_0\} \cup Q_1$.
   The states of N are the states of N_1 plus a new start state.
2. The state q_0 is the new start state.
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 $a \in \Sigma_\varepsilon$,

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

Regular expressions

Definition
Say that R is a regular expression if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\varnothing$,
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.

RE Examples

In the following examples we assume that the alphabet Σ is $\{0,1\}$.

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

RE Examples (cont.)

8. $(0 \cup \varepsilon)(1 \cup \varepsilon) = 01^* \cup 1^*$.
   The expression $0 \cup \varepsilon$ describes the language $\{0, \varepsilon\}$, so the concatenation operation adds either 0 or ε before every string in 1^*.
9. $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$.
10. $1^*\varnothing = \varnothing$.
    Concatenating the empty set to any set yields the empty set.
11. $\varnothing^* = \{\varepsilon\}$.
    The star operation puts together any number of strings from the language to get a string in the result. If the language is empty, the star operation can put together 0 string, giving only the empty string.

$R \cup \varnothing$

$R \circ \varepsilon$

$R \cup \varepsilon$

$R \circ \varnothing$

Applications

- Design of compilers

$$\{+,-,\varepsilon\}(DD^* \cup DD^*.D \cup D^*.DD^*)$$
$$\text{where } D = \{0,\ldots,9\}$$

- awk, grep, vi … in unix (search for strings)
- Perl, Python, or Java programming languages
- Bioinformatics
  - So called motifs (patterns occurring in sequences, e.g. proteins)

Equivalence RE and NFA

Theorem 1.28

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

Proof through :

Lemma 1.29

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

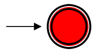
Lemma 1.32

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

Proof for Lemma 1.29 (cont.)

2. $R = \varepsilon$.

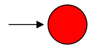
Then $L(R) = \{\varepsilon\}$, and the following NFA recognizes $L(R)$.



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

Proof for Lemma 1.29 (cont.)

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



Formally, $N = (\{q\}, \Sigma, \delta, q, \emptyset)$, where $\delta(r, b) = \emptyset$ for any r and b.

Proof for Lemma 1.29 (cont.)

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

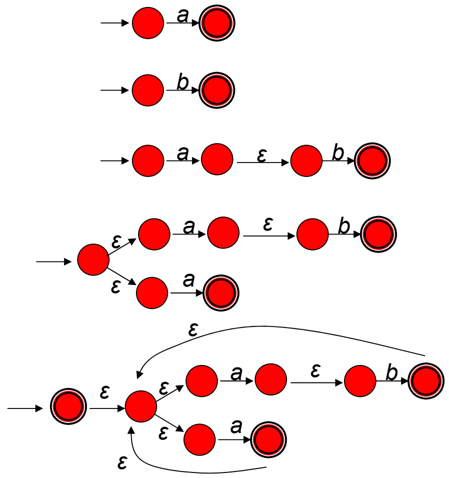
For the last three cases we use the constructions given in the proofs that the class of regular languages is closed under the regular operations. In other words, we construct the NFA for R from the NFAs for R_1 and R_2 (or just R_1 in case 6) and the appropriate closure construction.

Example 1.30

We convert the regular expression $(ab \cup a)^*$ to an NFA in a sequence of stages. We build up from the smallest subexpressions to larger subexpressions until we have an NFA for the original expression, as shown in the following diagram. Note that this procedure generally doesn't give the NFA with the fewest states!

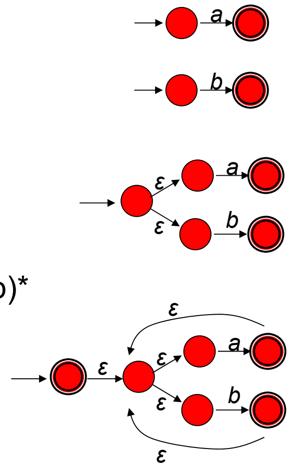
Example: NFA for: (ab ∪ a)*

- a:
- b:
- ab:
- ab ∪ a
- (ab ∪ a)*



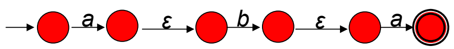
Exercise: NFA for: (a ∪ b)*aba

- a:
- b:
- a ∪ b
- (a ∪ b)*

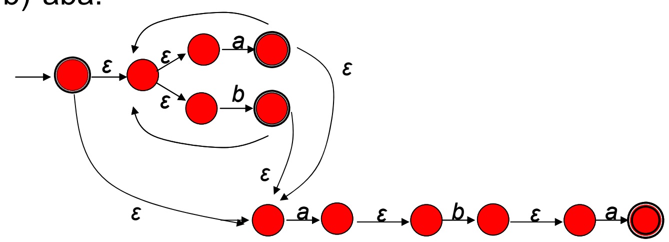


Example: NFA for: (a ∪ b)*aba (cont.)

- aba:



- (a ∪ b)*aba:



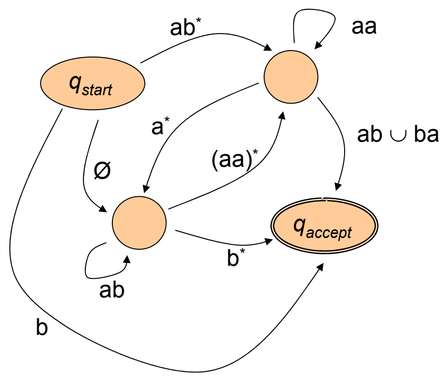
Lemma 1.32

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

- Two steps
  - DFA into GNFA (generalized nondeterministic finite automaton)
  - Convert GNFA into regular expression

GNFAs

- Labels are regular expressions
- Two states q and r are connected in both directions (fully connected)
- Exception :
  - *One direction only*
  - Start state (exiting transition arrows)
  - Accept state (only one!) (only incoming transition arrows)



Formally

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

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

A GNFA accepts $w = w_1 \ldots w_k$ where each $w_i \in \Sigma^*$ if a sequence of states $r_0, \ldots, r_n$ exists in Q such that

1. $r_0 = q_{start}$
2. $r_k = q_{accept}$
3. for all $i = 0, \ldots, n-1$, we have that $w_i \in L(R_i)$ where $R_i = \delta(r_{i-1}, r_i)$

Convert DFA into GNFA

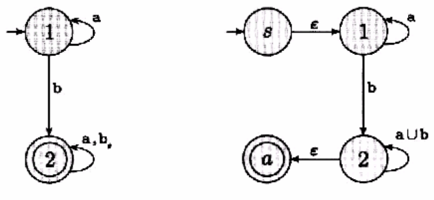
Add new start state, with ε arrow to old start state

Add new accept state, with ε arrows from old accept states

If any arrows have multiple labels a and b, replace by $a \cup b$

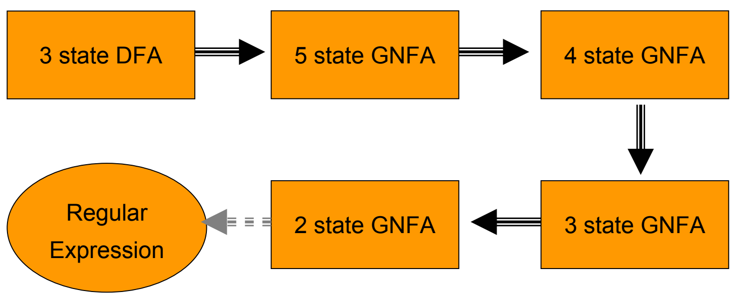
Add arrows with label $\varnothing$ between states where necessary*

(*:between states that had no arrows before)



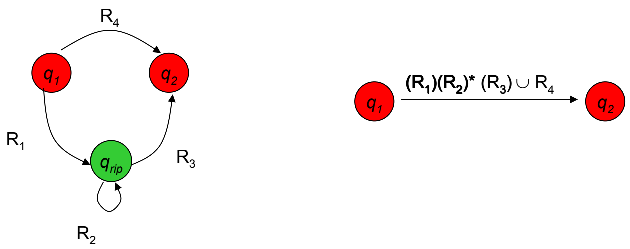
(a) (b)

Convert GNFA into regular expression



Ripping of states

Replace one state by the corresponding RE

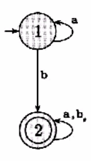


Convert(*G*)

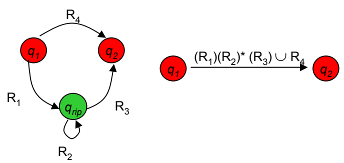
Convert(G):

1. Let k be the number of states of G.
2. If $k = 2$, then G must consist of a start state, an accept state, and a single arrow connectiong them and labeled with a regular expression R. Return the expression R.
3. If $k > 2$, we select any state $q_{rip} \in Q$ different from q_{start} and q_{accept} and let G' be the GNFA $(Q', \Sigma, \delta', q_{start}, q_{accept})$, where
$$Q' = Q - \{q_{rip}\},$$
and for any $q_i \in Q' - \{q_{accept}\}$ and any $q_j \in Q' - \{q_{start}\}$ let
$$\delta'(q_i, q_j) = (R_1)(R_1)^*(R_3) \cup (R_4),$$
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.

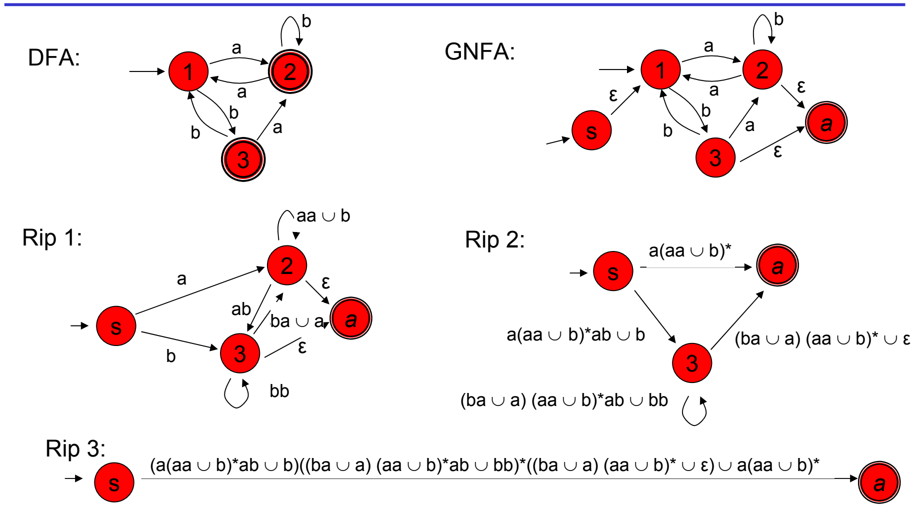
Example



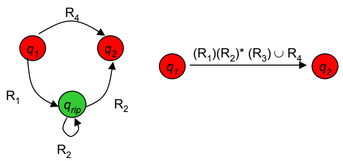
(a)



Another Example



Induction Proof



Claim
For any GNFA G, *Convert*(G) is equivalent to G.

We prove this claim by induction on k, the number of states of the GNFA.

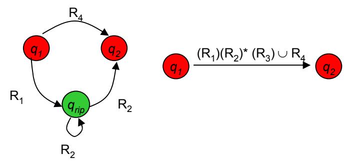
Basis: Prove the claim true for $k = 2$ states. If G has only two states, it can have only a single arrow, which goes from the start state to the accept state. The regular expression label on this arrow describes all the strings that allow G to get to the accept state. Hence this expression is equivalent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

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

If none of them is the removed state q_{rip}, clearly 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.

Induction Proof (cont.)



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 arrow 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 arrow between any two states q_i and q_j in G' describes the collection of strings taking q_i to q_j in G, either directly or via q_{rip}, G must also accept w thus G and G' are equivalent.

The induction hypothesis states that when the alorithm calls itself recursively on input G', the result is a regular expression that is equivalent to G' because G' has $k - 1$ states. Hence the regular expression also is equivalent to G, and the algorithm is proved correct.

This concludes the proof of Claim 1.34, Lemma 1.32, and theorem 1.28.

Nonregular Languages

- Finite Automata have a finite memory
- Are the following languages regular ?

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

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

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

- Mathematical proof necessary

The pumping lemma

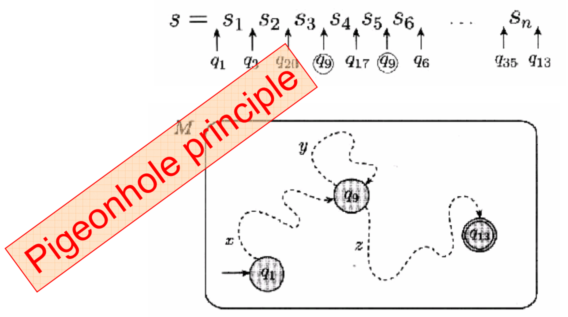
If A is regular language, then there is a number p (the pumping length), where, if s is any string in A of length at least p then s may be divided into three pieces $s = xyz$ such that

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

Note from 2: $y \neq \varepsilon$

Proof Idea

Let M be a DFA recognizing A. Assign **p to be the number of states in M.**
Show that string **s**, with length **at least p**, can be broken into ***xyz***.



Now prove that all three conditions are met

Proof: Pumping Lemma

- Let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA recognizing A and $|Q| = p$.
- Let $s = s_1 s_2 \dots s_n$ be a string in A, with $|s| = n$, and $n \geq p$
- Let $r = r_1, \dots, r_{n+1}$ be the sequence of states that M enters for s, so $r_{i+1} = \delta(r_i, s_i)$ with $1 \leq i \leq n$. $|r_1, \dots, r_{n+1}| = n+1$, $n+1 \geq p+1$.
  Amoung 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 **x** $= s_1 \dots s_{j-1}$, **y** $= s_j \dots s_{l-1}$ and **z** $= s_l \dots s_n$:
  - as **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 $l \leq p+1, |xy| \leq p$

Pumping Lemma (cont.)

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 examples

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

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

Choose $s = 0^p 1^p$

If would now only consider condition2, then we would have that:

1. string y consists only of 0s
2. string y consists only of 1s
3. string y consists of both 0s and 1s

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

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

Choose $s = 0^p1^p$

Would seem possible without condition 3!

However, condition 3 of lemma states $|xy| \leq p$

Thus y consists of 0s only

Then $xyyz \notin C$

Choice of s crucial. Consider $s = (01)^p$

Alternative proof :

B is nonregular

If C were regular, then $C \cap 0^*1^* = B$ regular

Regular languages closed under intersection

Example language *B* again

$$B = \{0^n1^n \mid n \geq 0\}$$

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

Choose $s = 0^p1^p$

condition 3 of lemma states $|xy| \leq p$

Thus y consists of 0s only

Then $xyyz \notin B$

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

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

Choose $s = 0^p10^p1$

Condition 3 of lemma states $|xy| \leq p$

Thus y consists of 0s only

Then $xyyz \notin F$

0^p0^p would not work, as it can be pumped !

$D = \{1^{n^2} \mid n > 0\}$

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

Choose $s = 1^{p^2}$

Consider xy^iz and $xy^{i+1}z$

Prove that for large i: xy^iz and $xy^{i+1}z$ cannot both be perfect squares,

which should be true according to pumping lemma. Therefore, D is not a regular language

Proof:

Let $m = n^2 = |xy^iz|$

Then: $(n+1)^2 - n^2 = 2n + 1 = 2\sqrt{m} + 1$

Choose $|y| < 2\sqrt{m} + 1 = 2\sqrt{|xy^iz|} + 1$

Indeed, observe

$|y| \leq |s| = p^2$; let $i = p^4$ then:

$$|y| \leq p^2 = \sqrt{p^4} < 2\sqrt{p^4} + 1 \leq 2\sqrt{|xy^iz|} + 1$$

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

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

Choose $s = 0^{p+1}1^p$

Condition 3 of lemma states $|xy| \leq p$

Thus y consists of 0s only

Then $xy^0 z \notin F$ □

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

Example Exam Question

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

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

Choose $s = 0^{2p}1^p$. Because s □ 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 $xy^0z = xz = 0^{2p-|y|}1^p$.

As xz has p ones, and $|y| > 0$, xz has fewer than $2p$ zeros.
Hence $xz \notin L$ CONTRADICTION.
Therfore L is not regular!

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

Example Exam Question

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

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

Choose $s = 0^{2p}1^p$. Because s □ 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 $xy^0z = xz = 0^{2p-|y|}1^p$.

As xz has p ones, and $|y| > 0$, xz has fewer than $2p$ zeros.
Hence $xz \notin L$ CONTRADICTION.
Therfore L is not regular!

Summary

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